Universidad Autónoma de Madrid
Escuela Politécnica Superior / Facultad de Ciencias
Departamento de Ingeniería Informática / Departamento de Matemáticas

SVR, General Noise Functions and Deep Learning. General Noise Deep Models.

PhD's thesis presented to apply for the
Phd in Informatics and Telecommunications Engineering

By
Jesús Prada Alonso

under the direction of
José R. Dorronsoro Ibero

Madrid, November 11, 2022

Contents

Abstract

Machine learning, ML, is a branch of artificial intelligence that allows to build systems that learn to solve a task automatically from data, in the sense that they do not need to be explicitly programmed with the rules or method to do it. ML encompasses different types of problems; one of them, regression, involves predicting a numerical output and will be the focus of this thesis.

Among ML models used for regression, Support Vector Machines, SVM, is one of the main algorithms of choice and is usually called Support Vector Regression, SVR, when applied to regression tasks. This type of models usually employs the ϵ-insensitive loss function, which implies a particular assumption of noise distribution in the data, but general noise cost functions have been recently proposed for SVR. These cost functions should be more effective when applied to regression problems whose underlying noise distribution follows the one assumed for that particular cost function. However, the use of these general functions, with the disparity in mathematical properties like differentiability that it implies, makes the standard optimization method used in SVR, Sequential minimal optimization or SMO, no longer a possibility.

Additionally, when working with large sample sizes, a common situation in the big data era, Deep Learning or DL models are able to extract more complex and meaningful relationships from the data than other ML families of models, being this one of the fundamental reasons to explain DL recent popularity.

Finally, although SVR models have been thoroughly studied, construction of error intervals for them seems to have received less attention and remains an unsolved problem. This is a significant handicap, as in many applications that involve solving a regression problem not only an accurate prediction is useful but also a confidence interval can be extremely valuable.

Taking all these factors into account, this thesis has four main goals: First, propose a framework to train General Noise SVR Models using Naive Online R Minimization Algorithm, NORMA, optimization. Second, give a method to build Deep General Noise Models that combine the highly non-linear feature processing of DL models with the predictive potential of using general noise loss functions, from which the ϵ-insensitive loss function used in SVR is just a particular example. Third, describe a direct approach to build error intervals for SVR or other regression models, based on the assumption of residuals following some probability distribution. And finally, unify the previous three goals in a single and final model framework to train Deep General Noise Models for regression prediction with confidence or error intervals.

For each one of these goals we will perform experiments, using both artificial and real datasets corresponding to the task of wind and solar energy prediction, to test the effectiveness of our proposals compared to standard SVM and DL models. Furthermore, in accordance with the principle of reproducible research, we make the implementations developed and the datasets employed in the experiments publicly and easily available.

Resumen

El aprendizaje automático, ML por sus siglas en inglés, es una rama de la inteligencia artificial que permite construir sistemas que aprendan a resolver una tarea automáticamente a partir de los datos, en el sentido de que no necesitan ser programados explícitamente con las reglas o el método para hacerlo. ML abarca diferentes tipos de problemas; Uno de ellos, la regresión, implica predecir un resultado numérico y será el foco de atención de esta tesis.

Entre los modelos ML utilizados para la regresión, las máquinas de vectores soporte o Support Vector Machines, SVM, son uno de los principales algoritmos de elección, habitualmente llamado Support Vector Regression, SVR, cuando se aplica a tareas de regresión. Este tipo de modelos generalmente emplea la función de pérdida $\epsilon-$insensitive, lo que implica asumir una distribución concreta en el ruido presente en los datos, pero recientemente se han propuesto funciones de coste de ruido general para SVR. Estas funciones de coste deberían ser más efectivas cuando se aplican a problemas de regresión cuya distribución de ruido subyacente sigue la asumida para esa función de coste particular. Sin embargo, el uso de estas funciones generales, con la disparidad en las propiedades matemáticas como la diferenciabilidad que implica, hace que el método de optimización estándar utilizado en SVR, optimización mínima secuencial o SMO, ya no sea una posibilidad.

Además, posiblemente el principal inconveniente de los modelos SVR es que pueden sufrir problemas de escalabilidad al trabajar con datos de gran tamaño, una situación común en la era de los grandes datos. Por otro lado, los modelos de Aprendizaje Profundo o Deep Learning, DL, pueden manejar grandes conjuntos de datos con mayor facilidad, siendo esta una de las razones fundamentales para explicar su reciente popularidad.

Finalmente, aunque los modelos SVR se han estudiado a fondo, la construcción de intervalos de error para ellos parece haber recibido menos atención y sigue siendo un problema sin resolver. Esta es una desventaja significativa, ya que en muchas aplicaciones que implican resolver un problema de regresión no solo es útil una predicción precisa, sino que también un intervalo de confianza asociado a esta predicción puede ser extremadamente valioso.

Teniendo en cuenta todos estos factores, esta tesis tiene cuatro objetivos principales: Primero, proponer un marco para entrenar Modelos SVR de ruido general utilizando como método de optimización Naive Online R Minimization Algorithm, NORMA. En segundo lugar, proporcionar un método para construir modelos DL de ruido general que combinen el procesamiento de características altamente no lineales de los modelos DL con el potencial predictivo de usar funciones de pérdida de ruido general, de las cuales la función de pérdida $\epsilon-$insensitive utilizada en SVR es solo un ejemplo particular. Tercero, describir un enfoque directo para construir intervalos de error para SVR u otros modelos de regresión, basado en asumir la hipótesis de que los residuos siguen una función de distribución concreta. Y finalmente, unificar los tres objetivos anteriores en un marco de modelos único que permita construir modelos profundos de ruido general para la predicción en problemas de regresión con la posibilidad de obtener intervalos de confianza o intervalos de error asociados.

Para cada uno de estos objetivos realizaremos experimentos utilizando conjuntos de datos artificiales y reales correspondientes a problemas de predicción de energía eólica y solar, para probar la efectividad de nuestras propuestas en comparación con los modelos estándar SVM y DL. Además, de acuerdo con el principio de investigación reproducible, las implementaciones desarrolladas y los conjuntos de datos empleados en los experimentos están pública y fácilmente disponibles.

Acknowledgements

With partial support from Spain's grants TIN2016-76406-P and S2013/ICE-2845 CASI-CAM-CM. Work partially supported also by project FACIL-Ayudas Fundación BBVA a Equipos de Investigación Científica 2016, and the UAM-ADIC Chair for Data Science and Machine Learning. We also gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

The research presented here would not have been possible without the support and guidance of the members of the Machine Learning Group from UAM, with a special mention to my tutor José, who helped me to find my path in the long, complex and often hard, but extremely rewarding, life of a researcher during these almost 8 years.

I want to also thank my professors and, even more, my colleagues during my time at the university, the Instituto de Ingeniería del Conocimiento, IIC, and the Machine Learning Group from UAM, who help to ignite the spark of curiosity and thirst for knowledge in me, specially in this fascinating world of machine learning and big data. Professor Juan Luis Vázquez and my colleagues Sara, Lucia and Luis are some of the most important names here, but surely others have also played his part in what I am today. Special thanks to David Diaz Vico, with whom I have shared research, papers, suffering, and really interesting discussions.

Thanks to my family and to my mother in particular for their contribution to what I am today, good and bad. I have enough self-esteem to congratulate myself for the effort and skills required to get to this point, but not as much as to not realize that I am also here because I was fortunate enough to be born in a place and time that made this possible, and with a family by my side that made sacrifices so I could get to the university in the first place. I have promised myself not to forget that.

I must also express my gratitude to Miguel Bravo and Ana Sierra. I have always felt like a mentor towards you but, at the end of the day, I am sure that I have learnt much more from you than what I could teach you. Your motivation and desire to learn and improve have been refreshing, and your curiosity challenging. They say that a good professor is the one that ends surpassed by their students and, if that is true, I think I did not do it that badly.

Marta Lopez Cortijo, *the dancer girl*, my personal psychologist. You have always been there for me in the lowest moments, ready to cheer me up, make me smile and guide me with your wise advices. I am not sure if you know how grateful I will always be for that.

Gabriel Maicas, *Guevara*, you were an inspiration for me during all these years. To know first-hand that finishing the PhD. was possible and seeing that people as valuable as you, not only in the academic regard but even more important in the personal and social aspect, were part of that group made me aim to get there one day.

María González, *Amelie*, I do not know if you will get to read this thank-you message one day, but I felt you deserved it anyway. Perhaps you do not realize it, but you have been one of the main factors to keep my motivation to finish this PhD. alive. Your eternal support, no matter the issue, gave me assurance, and the appreciation and admiration towards me that I could see in your eyes gave me energy and made feel a better person, better than what I probably am. I hope I was able to give you at least a tenth of the light and magic you gave to me. Thanks for everything.

I want to also dedicate some special words to Carolina Espejo, *Tequeña*. In the past, when I have discussed with other people my love for the Machine Learning world, often I have felt that I am a little bit crazy, but your passion and dedication to your work have made feel not alone regarding this feeling. Besides, and probably more important, your constant support and trust has given me strength to endurance the difficulties of living life the way I do. Having a partner that shares this non-conformist style of life is priceless. Please keep being the way you are.

Last but not least, extra special thanks to Yvonne, *Pheebs*, my closest companion during all these years. We have lived so many moments together, good and bad, that I just cannot imagine what my life would have been or would be without you. Nevertheless, of one thing I am certain, life would have been duller and greyer. I am a completely different person from where this journey began and many people have had an impact on molding what I am today, but surely you have been the main shaper of my personality, the one from whom I have learnt more things and the one that made me grow more as a person. Hopefully it would remain the same in the future. You have made me a better person, and I cannot think of something more valuable than that.

Chapter 1

Introduction

> Begin at the beginning, and go on till you come to the end: then stop.
>
> *Lewis Carroll, Alice in Wonderland*

Building computer systems to solve, by themselves or by assisting an expert, human problems has been a vital task in many studies and applications, both academic and corporate, for many years.

Initially this type of tasks were tackled by expert systems, computer systems that emulate the decision-making ability of a human expert [1]. The first ones appeared in the 1970s and clearly dominated the field of artificial intelligence during the 1980s. In these systems, the decision-making algorithm is explicitly coded, primarily as a sequence of if-then rules. The Lisp family of programming languages were very popular to build this kind of systems.

In contrast to expert systems, Machine Learning, ML [2], techniques try to infer from the data the best algorithm to model the problem and give a solution to it, whether it is a division into clusters, a classification into previously specified groups, the prediction of a real number, or creating an artificial player for games like chess or Go. In ML the dependence on and need of expert knowledge is lessened, specially when using Deep Learning, DL, frameworks, although not completely removed. Furthermore, more general and less ad hoc applications can be built using ML models.

These and others advantages have made Machine Learning a very popular tool nowadays and one that has been widely studied and used in a variety of problems in recent years. Its popularity has been strengthened by the coming of the so-called Big Data era, as well as by the great increase in computational power and recent research in the Deep Learning field.

We will start this chapter with a brief description of the goal of this thesis in Section 1.1. Then, we will describe the basic concepts of the three technologies mentioned earlier and used here to achieve this goal: Machine Learning in general in Section 1.2, Deep Learning in Section 1.3, and Big Data in Section 1.4. Section 1.5 discusses how well Machine Learning, specially Deep Learning models, combine with Big Data technologies. Finally, Section 1.6 presents the structure for the rest of the thesis.

1.1 Goal

This thesis has five main goals:

1. Design a framework to train kernel-based General Noise Models using the Naive Online R Minimization Algorithm, NORMA, to solve supervised regression problems.

2. Propose a method to build Deep General Noise Models that combine the highly non-linear feature processing of Deep Learning models with the predictive potential achieved in the previous step due to the use of general noise loss functions.

3. Design a direct approach to build error intervals for SVR or other regression models, based on the assumption of the residuals following some probability distribution. This way, we will be able to not only give a prediction for our supervised regression problem, but to also provide a confidence interval.

4. Finally, we want to unify the previous three goals in a single and final model framework to train Deep General Noise Models for regression prediction with confidence or error intervals.

5. Furthermore, in accordance with the principle of reproducible research, we want to make the implementations developed and the datasets employed in the experiments publicly and easily available.

For each one of these goals we will perform experiments using both artificial and real datasets corresponding to the task of wind and solar energy prediction, to test the effectiveness of our proposals compared to standard SVM and DL models. These problems have been selected from our previous experience on the topic and because they are tasks were previous research has shown to follow specific noise distributions. However, we strongly believe that our proposed framework can adapt to any type of regression problems.

1.2 Machine Learning

Machine learning is a branch of Artifical Intelligence that aims to build computer systems that automatically learn from data how to solve a task. From this basic and brief definition, it is important to notice two significant aspects:

- *ML is a branch of Artificial Intelligence.* Frequently these two terms are confused or treated as independent fields. The most accepted definition is that ML tools are just a subset of a bigger toolbox called Artificial Intelligence, which comprises other fields like expert systems and part of robotics.

- *ML automatically learns from data.* Automatically not in a strict sense, as someone still has to code an implementation of the ML method to apply in order to train the model, but in the sense of ML models not needing to be explicitly programmed with the rules or methods to solve the task at hand. The algorithm used to tackle the problem will be built automatically from the data used to train the ML model.

In [2], the following formal definition of learning is given:

Definition 1. *A machine learns with respect to a particular task* $\boldsymbol{T}$*, performance metric* $\boldsymbol{P}$*, and type of experience* $\boldsymbol{E}$*, if the system reliably improves its performance P at task T, following experience E.*

For instance, using wind energy prediction as an example, we would have:

- **T:** Wind energy production forecast using weather information.
- **E:** Past data of weather and energy production.
- **P:** A particular metric, such as the mean absolute error, MAE, or the mean squared error, MSE, defined as

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{f}(X_i) - y_i| \tag{1.2.1}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{f}(X_i) - y_i)^2 , \tag{1.2.2}$$

where $\{X_i\}_{i=1}^N = \{(x_{i1}, x_{i2}, ..., x_{id})^T\}_{i=1}^N$, is the weather information used as input data, $\hat{f}(X)$ is the forecast outputted by the Machine Learning model given input X, $\{y_i\}_{i=1}^N$ is the real wind energy production or *target*, N is the number of *instances* or samples and d is the number of variables, or *dimensions*, in the data.

The field of Machine Learning encompasses a wide variety of problems. Some examples are:

- Forecasting of wind, solar or other type of energy production in different geographic areas learning from past productions and numerical weather predictions.
- Use of historical medical records to learn which people could suffer from some particular disease or which treatments are optimal for a particular patient.
- Design of autonomous vehicles that learn to navigate and interact with other vehicles from their own experience.
- Recommender systems to automatically customize marketing actions of a company to their users' interests using their information and past interactions.
- Prediction of sporting events outcome using past results, statistics and information from other sources such as social networks like Twitter.

In [3] and [4], ML tasks are divided into three main groups:

1. **Supervised Learning**: In supervised learning problems, you have available a labeled training dataset. These labels are the goal, or *target*, you want your ML model to be able to predict. Depending on the nature of these labels supervised learning can be divided, in turn, into two subgroups:

   (a) **Classification:** Labels are categorical, representing the belonging of a particular instance to a specific class. 0 and 1 are standard labels for a 2-class or binary classification problem, but classification over more than 2 classes is also perfectly doable.

(b) **Regression:** Here the target is a numeric label in the form of a real number indicating a particular property of each instance, like the price of an item in a shop.

In both cases, the principles of the Machine Learning cycle are the same, but algorithms, evaluation metrics and *objective functions* used are different for each type of problem. In supervised learning, the data available is normally split into three different datasets, each one with a specific purpose:

(a) **Train:** Dataset used to build, or *train*, the model. Parameters of the model are chosed to minimize a particular objective function for this training set.

(b) **Validation:** Each family of ML models encompasses infinite different models in itself. This is due to the fact that each family of models is linked to a set of configuration parameters, usually called *hyperparameters*. The validation set is used to select the best hyperparameters for the ML model in hand. Hyperparameters selected as optimal are the ones that minimize a chosen *evaluation metric* over the entire validation data.

(c) **Test:** The model built from this training and validation process is then used to predict the class or value of new instances belonging to a labeled test dataset. Prediction errors resulting from this process, obtained by the same evaluation metric used in the validation step, are used as a measure of the expected model accuracy when put into production in real life to give predictions for new unlabeled data.

There is no golden rule to decide the ratio of data set aside for training, validation and test purposes, as the optimal value is strongly problem-dependent as it is often the case in the ML field. Nevertheless, a standard recommendation to select these splitting percentages is 70/15/15.

It is important to remark that frequently a fixed validation dataset is not employed and, instead, a technique called **cross-validation**, CV, [5] is applied to find the optimal hyperparameters of a ML family of models, following the schema shown in Figure 1.2.1. When this method is used, the training dataset is divided into k subsets, then the model is trained using $k-1$ of these subsets and the remaining one is used as validation set. This process is repeated k times until all subsets have performed the role of validation set. The errors coming out of these k iterations, after applying the selected evaluation metric, are then averaged and used as validation error. The hyperparameters chosen are the ones that minimize this validation error, as was the case when using a fixed validation set.

One of the key factors in supervised learning models is the *bias-variance* tradeoff:

(a) **Bias:** When a model has high bias it remains mainly unaffected by changes in the input data, leading to what is called *underfitting*. This phenomenon can be detected when a high training error occurs, as this is pointing to a ML model that is not adapting or learning from the train dataset.

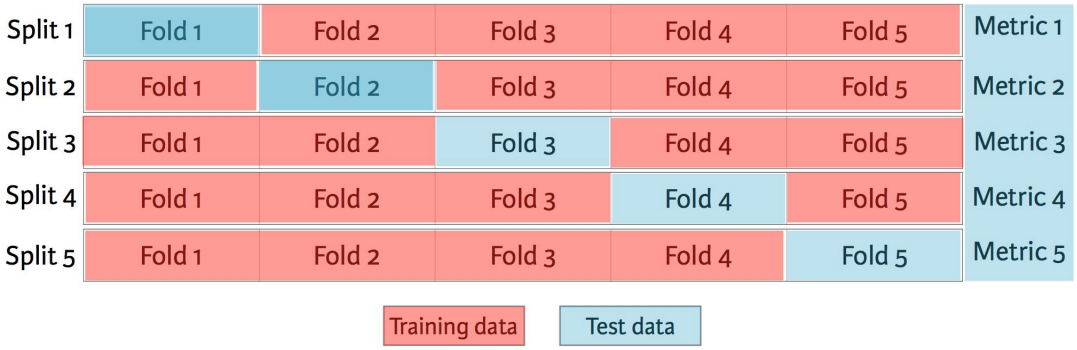


Figure 1.2.1: Cross-validation schema.

(b) **Variance:** High variance leads to a model that adjusts too much to the variations in the training set and thus is not able to generalize well to new data. This situation is often called *overfitting* and its existence can be assumed when the test error is much bigger than the train error and this behavior is not logical taking into account the nature of both datasets.

To deal with these phenomena, normally a *regularization or penalty term* is added to the usual error measure, or *loss function*, to form the final objective function that will be minimized to build the model. One example is *ridge regression* [6], a model that has as objective function

$$\frac{1}{N}\sum_{i=1}^{N}\Big(X_i^T\beta+\beta_0-y_i)^2+\frac{\lambda}{2}||\beta||^2\ , \tag{1.2.3}$$

where $\hat{f}(X_i)=X_i^T\beta+\beta_0$ is the output of the model, β_0 is a constant called the *intercept* or *bias* of the model and $\beta=(\beta_1,\beta_2,...,\beta_d)^T$ are the parameters or coefficients of the model.

This ridge regression objective function is a combination of two terms:

(a) **Loss function:** The error measure we want to minimize while training the model. In this case the mean squared prediction error: $\frac{1}{N}\sum_{i=1}^{N}(X_i^T\beta+\beta_0-y_i)^2$

(b) **Regularization term:** Controls the bias-variance tradeoff and therefore avoids underfitting and overfitting phenomena. In this case there is a quadratic penalty over the model parameters: $\frac{\lambda}{2}||\beta||^2$

2. **Unsupervised Learning**: In these problems there are no labels and thus the purpose is not to train a model to be able to predict these labels for future examples, as it was the case in the supervised learning approach. Here the goal is to find hidden structure in this unlabeled data, existing three main types of problems:

(a) **Clustering:** Divide data into different groups, with instances in the same cluster being more similar among themselves than to instances in other clusters.

(b) **Recommender Systems:** Give content recommendations to users of a particular application taking into account their past interaction: purchases, reviews, etc.

(c) **Dimensionality Reduction:** Reduce the number of variables or columns in a dataset. This can be done to decrease computational costs, get a new dataset with more relevant variables, carry on visualizations, etc.

In this type of tasks there is no error metric to evaluate a potential solution. Sometimes, unsupervised learning techniques are employed prior to applying supervised learning models as an additional data pre-processing step. In this case, goodness of unsupervised learning methods can be measured by their impact on the accuracy of the subsequent supervised learning model.

3. **Reinforcement Learning**: In this type of tasks the concern is the problem of finding suitable actions to take in a given situation in order to maximize a *reward*. Here the learning model is not given labels of optimal outputs, in contrast to supervised learning tasks, but must instead discover them using an iterative process of trial and error. Usually, there is a sequence of states and actions in which the model is interacting with its environment, and frequently the current action not only has an impact on the immediate reward but also affects the reward at all subsequent time steps. Only at the end of this process the reward signal, positive or negative, is received.

   The exploration-exploitation trade-off between *exploration*, in which the system tries out new actions to see how effective they are, and *exploitation*, in which actions that are known to yield a high reward are applied, is vital in these learning algorithms.

This thesis focuses its attention on supervised learning problems, although some unsupervised learning techniques like clustering are applied as a preprocessing step. Usually, the design of a supervised learning system entails an iterative cycle composed of different steps, where several of them are carried out again in each iteration. Figure 1.2.2 summarizes the general workflow in a Machine Learning project. Although different divisions of this process into steps have been described, normally there are four main stages:

1. **Data Collection**: Consists in gathering the data needed to train, validate and test the model. It is important to remark that this step should start as soon as possible, because normally several months of data collection are needed before we can advance to the next step. Sometimes it is a very costly stage, so a tradeoff between the volume of data collected and the cost of this gathering process must be made. It is also important to keep in mind the complexity of the problem and of the model chosen to decide when we have an adequately large amount of data for a particular problem.

2. **Data Preprocessing**: This stage involves several steps that allow to transform our original raw dataset into a final dataset with more analytical potential, so our ML model can extract the maximum information possible from the available data. Even if not as popular and headline-catching as other stages like modelization itself, this

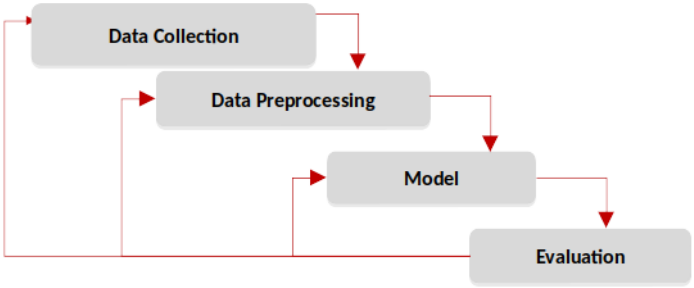


Figure 1.2.2: Design cycle for supervised learning systems.

is a critical step and frequently consumes most of the time spent by a team on this type of projects.

Some examples of pre-processing steps, although there a lot more, are the following ones:

- **Feature choice:** There are two different methods to carry out feature choice. *Feature selection* picks a subset of the original features and discards the rest, while *feature extraction* generates derived variables from the original ones.

  Dimensionality reduction techniques, like Principal Component Analysis or *PCA*, where the features extracted are intended to be informative and non redundant, can be applied to improve the accuracy of the ML model, reducing the risk of overfitting, although for complex models with a regularization term this is not really necessary and feature choice is done implicitly by the model. In addition, dimensionality reduction can also provide a significant decrease in computational cost, which is vital to many real-world ML systems, particularly for those that need to be able to give a real time response such as fraud detection in banking systems.
- **Scaling:** Most ML models are sensitive to the magnitude of the input variables, so it is common that variables with bigger values have a stronger impact in the model learning than other variables. To avoid this problem, some method of scaling should be applied before passing the data to the model. One standard choice of scaling method is to scale all input features to 0 mean and standard deviation equal to 1.
- **Fill missing values:** Although there are some exceptions, like Random Forests [7], generally ML models do not accept missing values in the input data. In order to solve this drawback, a method to fill missing values must be chosen. There is a comprehensive list of methods to perform this task, with bootstaping combined with Expected Maximization [8] or the use of simpler versions of ML models to predict the missing data being popular choices.

3. **Modelization**: This stage implies in first place the selection of a set of families of ML models and hyperparameters for each of these families. There is a wide variety of ML models for supervised learning problems, ranging from a simple linear regression to deep learning techniques.

   The optimal model for a particular problem depends on factors as the nature of the problem and its complexity, the data available and its dimensions and the presence of noise in the data. These factors must be taken into account when choosing the model to use and the common mistake of choosing a particular model for some personal preference and not for being the most suitable for the task at hand should be avoided when opting for a model to solve a real-world problem.

   Once it has been decided which models and configurations to try, all these combinations must go through the train-validation framework described earlier, to choose the optimal parameters and hyperparameters, respectively, for each family of models selected in the previous step.

4. **Evaluation**: Once training and validation of the models is done, their performance over the test dataset is evaluated through some particular error measure, obtaining an expected error. Typical choices for regression problems are MAE (1.2.1) and MSE (1.2.2), and for binary classification problems the following ones are common measures:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{1.2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{1.2.6}$$

$$FPR = \frac{FP}{FP + TN} \tag{1.2.7}$$

$$F1 = \frac{2TP}{2TP + FP + FN}\,, \tag{1.2.8}$$

   where TP are true positives, i.e. instances classified as positive by the model that are in fact positive, FP are false positives, instances classified as positive that are in fact negative, TN are true negatives, instances classified as negative that are in fact negative, and FN are false negatives, instances classified as negative that are in fact positive.

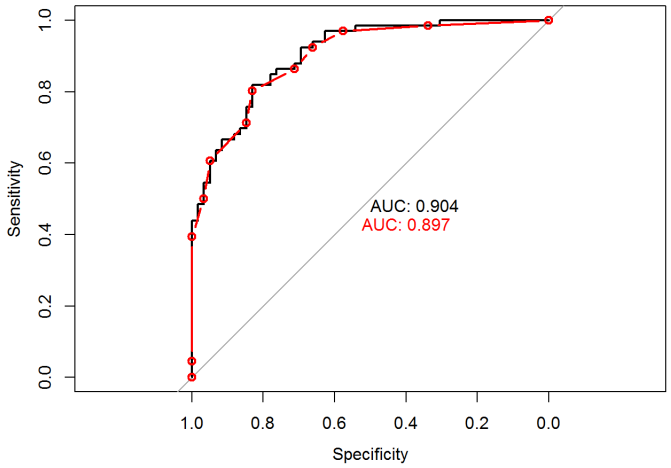


Figure 1.2.3: Area Under the ROC Curve, AUROC.

Precision vs recall and ROC or FPR vs recall curves are also frequently used as evaluation measures for two class classification problems. In particular, the Area under the ROC Curve, AUC or AUROC, is often the standard choice for binary classification. The ROC, which stands for Receiver Operating Characteristic, curve is created by plotting the *recall* or true positive rate, also known as *sensitivity*, against the false positive rate, also known as probability of false alarm and which is the opposite of *specificity*, where *sensitivity* and *specificity* are defined as follows:

$$Sensitivity = \frac{TP}{TP + FN} \tag{1.2.9}$$

$$Specificity = \frac{TN}{TN + FP} . \tag{1.2.10}$$

Each point of the ROC curve corresponds to a particular choice of decision threshold. Predictions greater or equal than the selected threshold will be predicted as positive class, and the remaining ones as negative class. Therefore, lower values of the threshold lead to more positive class predictions, or equivalently, a point of the curve that moves upwards in the y-axis but to the right on the x-axis. Figure 1.2.3 shows two examples of ROC curves compared, where each value for the decision threshold corresponds to a particular point in each of the two curves.

These binary classification measures can be extended to a multiclass framework by means of averaging their result over pairs of classes. Two main methods can be followed here:

- **one-vs-all:** The selected metric is applied considering as class A one of the existing classes in the problem to tackle, and class B the union of all the other classes. This is done for all the classes, i.e. in a way that each existing class performs the role of class A one time. Results are then averaged to get a multiclass error.

- **one-vs-one:** The selected metric is applied considering as class A one of the existing classes and class B a different one of these existing classes. This is done so all possible combinations of (class A, class B) are used once. Results are then averaged to get a multiclass error.

1.3 Deep Learning

Although often considered an independent field, Deep Learning, DL [9], is no less and no more than just another family of Machine Learning models. However, it is a family of models with some extremely relevant properties, such as

- **Predictive Potential:** Nowadays larger and larger datasets are becoming available for their use to train ML models. In order to take as much information and predictive potential from these big datasets as possible, it is necessary to use complex enough ML methods, able to extract the most information possible from this data. Support Vector Machines, SVMs, which we describe in Section 2.1, are one of the most complex models among all the standard families of ML models, and that is the main reason for its dominance during a long span of time. However, they present important scalibility problems when dealing with large volumes of data.

  With the rise of Deep Learning frameworks it has been shown that these DL models are able to achieve an even better performance when trained with datasets that are sufficiently big. This fact is probably the main factor why this family of models is becoming the preferred choice when solving a high variety of large supervised learning tasks.

- **End-to-End learning:** In a normal ML project, as described earlier, one of the stages in the pipeline is data pre-processing, which encompasses several steps including what is usually called feature engineering, i.e. creation and selection of variables. However, due to the specific nature of Deep Learning frameworks, consisting of several layers which carry out intermediate tasks necessary to solve ML problems, these steps of the pipeline can be no longer needed when applying DL models.

  This property is often called End-to-End learning [10] and allows researchers and data scientists to avoid complex and time-consuming steps that were required previously and that commonly demanded or were easily and better done with the help of human experts in the field corresponding with the task at hand.

1.4 Big Data

The concept of Big Data is a rather new one but has grown in importance very quickly in recent years in the field of computer science, swiftly becoming a key concept in many studies and applications. Despite what its name could suggest, Big Data is not only related to the volume of raw information, involving other data qualities as well. There are three main properties that are required to consider a data environment as Big Data, and they are called the **3Vs** of Big Data. These properties, shown in Figure 1.4.1, are the following ones:

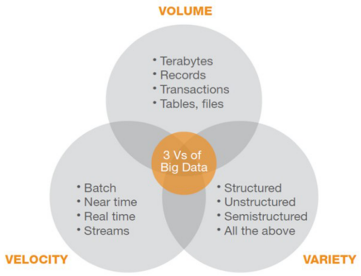


Figure 1.4.1: 3Vs of Big Data: Volume, Variety, and Velocity. From [14].

1. **Volume**: Refers to the volume of data available. It is very important regarding Machine Learning projects and the name Big Data itself contains a term which is related to size.

2. **Variety**: It makes reference to the diversity of the data and variance among the sources from which this data is gathered. Usually, raw data is unstructured or has different structures depending on its source, so an appropiate pre-processing step is key.

3. **Velocity**: The last of the 3 Vs of Big Data refers to the speed of generation of data and how fast the data must be processed to meet the demands and challenges of a particular task. This is particularly relevant in problems that require real-time or pseudo real-time answers [11].

Taking into account these 3Vs, several formal definitions of Big Data have been proposed in recent years [12] [13], such as

Definition 2. *Big Data are high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization.*

Definition 3. *Big Data represents the information assets characterized by such a high Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value.*

Other additional Vs have been proposed recently. An article from 2013 by Mark van Rijmenam adds four more, reaching a total of **7Vs**. Apart from the three mentioned above, these are:

4. **Variability**: Refers to data whose meaning is changeable. This is particularly the case when data collecting relies on natural language processing, NLP. Words do not have static definitions, and their meaning can vary wildly depending on context. Thus, programmes which can process context and decode the precise meaning of words through it, like recent Deep Learning application do, need to be used and applied.

5. **Veracity**: Data loses its usefulness if it is not accurate, and in very rare cases data available is noise free. Hence, it is again vital to apply a good pre-processing pipeline that takes into account the usually noisy nature of data and produces a sufficiently accurate dataset before proper analysis can start.

6. **Visualization**: Once it has been processed and analyzed, you need a way of presenting the data in a manner that is readable and accessible, and this is what visualization refers to. Visualizations can contain thousands of samples and variables, and finding a way to present this information that makes the findings clear is one of the challenges in the Big Data era.

7. **Value**: The potential value of Big Data is huge. However, the cost generated by the use of poor data is also really significant. In essence, data on its own is virtually worthless, the value of it lying in rigorous analysis of accurate data, and the information and insights this provides both in the academic and corporate worlds.

1.5 Deep Learning & Big Data. The Perfect Couple

Machine learning, particularly Deep Learning models, and Big Data are two concepts that have extremely close ties, both benefiting from one another. On the one hand, the potential of ML models to give accurate outputs is boosted with bigger and more diverse, i.e. with more Volume and Variety, datasets available to train, validate and test the performance of the model predictions. This is of course only true when the data is accurate and meaningful, i.e. we have Veracity and Value. Of course, to be able to extract information from these large datasets, complex and scalable ML models are needed, and it is in this regard where Deep Learning has shown to be most valuable.

On the other hand, Big Data problems have changed the entire way of thinking about knowledge extraction and interpretation. Traditionally, data science has always been dominated by trial-and-error analysis, an approach that becomes impossible when datasets are large and heterogeneous as occurs when Big Data comes into play. Ironically, availability of more data usually leads to fewer options in constructing predictive models, because very few tools allow for processing large datasets in a reasonable amount of time, scalability being more important now than ever. In addition, traditional statistical solutions typically focus on static analytics that are limited to the analysis of samples that are frozen in time, which often results in surpassed and outdated conclusions. Machine Learning techniques allow researchers to overcome those problems and to build systems that can provide models updated after the arrival of new datapoints. ML models can also be used to build real-time systems, programs that must guarantee response within specified time constraints, which is related to the Velocity property.

To help researchers to combine the use of ML and Big Data, several software tools have been developed in recent years. In addition to supercomputers and Remote Procedure Call, RPC, communication, there has been a recent appearance of open frameworks that make computations for Big Data ML problems easier, such as *Apache Hadoop*, *Cloudera*,

and *Apache Mahout*. Particularly relevant here is *Apache Spark* [15], a framework for parallelized ML computing that has boosted investigation in this line of research.

1.6 Outline

The rest of this thesis is structured as follows:

- In Chapter 2 we present the main theoretical contributions from past research that have been used as building blocks for this thesis, with special focus on SVM and DL models. In particular, we describe theoretical details and mathematical formulations for SVM models, both for classification and regression problems, NORMA optimization, DL structures, and clustering algorithms like K-means and K-prototypes.

- Then we describe our proposed framework in Chapter 3, which has as its main goal to combine the virtues of both SVMs and DL for regression problems and lessen their drawbacks, as well as to add confidence intervals to the predictions given. We first describe how to build General Noise Models trained using NORMA optimization. Then, we propose a deep version of SVR models. After this, we propose to plug general noise cost functions into DL structures, creating our proposed Deep General Noise Models, D-GNM. Finally, we propose different methods to compute uncertainty intervals for the predictions of any regression model, including a method based on the use of clustering algorithms, in order to be able to build D-GNM models that are able to give not only a prediction value, but a corresponding uncertainty interval for that prediction.

- Experiments carried out to test the usefulness of the proposed model over different problems and datasets, including synthetic data, popular datasets, and real-world problems related to renewable energies, are described in Chapter 4, as well as their corresponding results. Details about the implementation of the models tested, selection of model hyperparameters, and the datasets employed in the experiments are described. Then, each of the experiments is described together with the results obtained. Finally, conclusions drawn from these results are discussed.

- Chapter 5 contains the main conclusions one can extract from this thesis. In addition, potential lines for further work on the research topic are also highlighted.

- Finally, appendices regarding author's publications and an extended table with information regarding one of the datasets employed in the experiments are included in Appendix A and Appendix B, respectively.

Chapter 2

Theoretical Background

> Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.
>
> *Albert Einstein*

In this chapter we present the main theoretical contributions from past research that have been used as building blocks for this thesis. Although the focus of this thesis is regression problems, this chapter starts with a thorough explanation of SVMs for classification in Section 2.1. This section continues with a description of their regression counterpart, SVR in Section 2.2. In addition, a general noise version of the SVM for regression, where a particular noise distribution for the data is assumed and plugged into the model is described in Section 2.3. Finally, in Section 2.4 a Bayesian framework, which allows the calculation of confidence intervals for SVR predictions is detailed.

Next, in Section 2.5 NORMA or Naive Online R Minimization Algorithm, an optimization method usually applied to train SVM models in an online manner, is detailed. Deep Learning models are described in Section 2.6, focusing on the reasons behind the spectacular growth in popularity of this family of ML models in recent years. The technical and mathematical details of this type of models will also be analyzed in detail. Finally, Section 2.7 describes the standard partition clustering methods K-means and K-prototypes.

2.1 SVM for Classification

Support vector machines, SVM, have been widely used in real-world problems such as fraud detection [16] or cancer prediction [17], and have remained as one of the most used models in ML around the world. Despite the fact that this work focuses on the use of SVM models for regression problems, this section follows for clarity a classical approach to the explanation of this family of ML algorithms, beginning with the description of its classification version, which can be useful to understand the intuition and mechanisms behind SVM models for regression.

This section focuses on the use of SVM for 2-class or binary classification problems. For tasks with more classes, a 2-class SVM for each pair of classes can be built. If k is the number of classes, then $\frac{k(k-1)}{2}$ classifiers are constructed and each one trains with data from

two classes. Then a voting strategy is used, where each binary classification is considered to be a vote. In the end, a point is designated to be in the class with the maximum number of votes. This is called one-vs-one approach for multiclassification and is the one followed by the most popular SVM implementation, LIBSVM [18].

Our training data consists of pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in R^d$ the input data, d the dimension of our dataset, N the number of examples, and $y_i \in \{-1, 1\}$ the classes or target to predict. Define a hyperplane, H, by

$$H = \{x : f(x) = x^T\beta + \beta_0 = 0\} , \tag{2.1.1}$$

where $\beta = (\beta_1, \beta_2, ..., \beta_p)^T$ is a unit vector, $||\beta|| = 1$, representing the parameters of the model, and β_0 is the bias of the model. Using this formulation, $f(x)$ induces a classification rule given by

$$G(x) = \text{sgn}(x^T\beta + \beta_0) , \tag{2.1.2}$$

where sgn is the sign or signum function.

The aim of Support Vector Classification is to obtain the best separating hyperplane possible. This standard formulation of the SVM for classification problems can be divided in three separate cases, from simpler to more complex and general: the linear separable case, the linear non-separable case, and finally the non-linear non-separable case, where it is necessary to deal with the problem of non-linearity and the key concept of *kernel trick* is vital.

2.1.1 Linear Separable Case. Hard Margin Classification

This is the most basic situation. Although it seldom appears in real-world problems, its explanation is interesting to introduce the SVM principles and formulations. Since the classes are separable, we can find a function $f(x) = x^T\beta + \beta_0$ where it is true that

$$y_i f(x_i) > 0, \ \forall i. \tag{2.1.3}$$

Therefore, we may be able to find the hyperplane that creates the biggest margin between the training points from classes 1 and -1. This is computed solving the following optimization problem

$$\begin{aligned} \max_{\beta,\beta_0} \quad & M \\ \text{subject to} \quad & y_i(x_i^T\beta + \beta_0) \geq M, \ i = 1, ...N , \\ & ||\beta|| = 1 . \end{aligned} \tag{2.1.4}$$

As stated in [19] we can get rid of the $||\beta|| = 1$ constraint by replacing the conditions in (2.1.4) with

$$\frac{1}{||\beta||} y_i(x_i^T\beta + \beta_0) \geq M, \ i = 1, ...N , \tag{2.1.5}$$

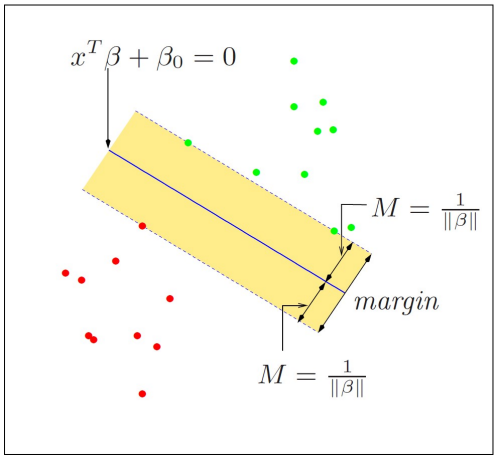


Figure 2.1.1: Support vector classifier for the separable case. The band in the figure is M units away from the hyperplane on either side, and hence $2M$ units wide. M is called the *margin*. Image from [5].

which leads to a redifinition of β_0. This can be equivalently expressed as

$$y_i(x_i^T\beta + \beta_0) \geq M||\beta||, \ i = 1, ...N \ . \tag{2.1.6}$$

For any β and β_0 satisfying these inequalities, any positively scaled multiple fulfills them too, so we can arbitrarily set $||\beta|| = \frac{1}{M}$ and get

$$\begin{aligned} &\max_{\beta,\beta_0} && M \\ &\text{subject to} && y_i(x_i^T\beta + \beta_0) \geq 1, \ i = 1, ...N \ . \end{aligned} \tag{2.1.7}$$

As we have defined that $M = \frac{1}{||\beta||}$. A visualization of this SVM formulation can be found in Figure 2.1.1. This problem is equivalent to

$$\begin{aligned} &\max_{\beta,\beta_0} && \frac{1}{||\beta||} \\ &\text{subject to} && y_i(x_i^T\beta + \beta_0) \geq 1, \ i = 1, ...N \ . \end{aligned} \tag{2.1.8}$$

Maximizing $\frac{1}{||\beta||}$ is equivalent to minimizing $||\beta||$, so we have

$$\begin{aligned} &\min_{\beta,\beta_0} && ||\beta|| \\ &\text{subject to} && y_i(x_i^T\beta + \beta_0) \geq 1, \ i = 1, ...N \ . \end{aligned} \tag{2.1.9}$$

For convenience to compute derivatives, usually the following equivalent form of (2.1.9), known as the **primal problem**, is used

$$\begin{aligned} &\min_{\beta,\beta_0} && \frac{1}{2}||\beta||^2 \\ &\text{subject to} && y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, ...N \ . \end{aligned} \tag{2.1.10}$$

In practice, the problem solved is the dual formulation derived using standard **Lagrangian techniques** [20]. First, in order to solve the constrained optimization problem, we add to the objective function the Lagrange multipliers as negative terms, one for each constraint in the primal problem, obtaining the **Lagrangian function**

$$L = \frac{1}{2}||\beta||^2 - \sum_{i=1}^{N} \alpha_i [y_i(x_i^T \beta + \beta_0) - 1] \ , \ \alpha_i \geq 0 \ . \tag{2.1.11}$$

Now, we want to obtain the dual optimization problem corresponding to maximizing this Lagrangian function. For this purpose, we first compute the derivatives in (2.1.11), obtaining

$$\frac{\partial L}{\partial \beta_0} = -\sum_{i=1}^{N} \alpha_i y_i \ , \tag{2.1.12}$$

$$\frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^{N} \alpha_i y_i x_i \ . \tag{2.1.13}$$

Setting the derivative (2.1.12) to zero we get

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \ , \tag{2.1.14}$$

and setting (2.1.13) equal to zero we arrive at

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \ , \tag{2.1.15}$$

Plugging (2.1.15) and (2.1.14) into (2.1.11) we finally obtain the **dual problem**

$$\begin{aligned} \max_{\alpha_i} \quad & D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{subject to} \quad & \alpha_i \geq 0, \ i = 1, ...N \ , \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \ , \end{aligned} \tag{2.1.16}$$

where the following conditions, called **Karush-Kuhn-Tucker, KKT, conditions**, are fulfilled at the optimal point, which we will represent using the $\widehat{\ }$ symbol for its parameters, namely $\widehat{\alpha}$, $\widehat{\beta}$, and $\widehat{\beta}_0$:

$$\widehat{\beta} = \sum_{i=1}^{N} \widehat{\alpha}_i y_i x_i \ , \tag{2.1.17}$$

$$\widehat{\alpha}_i \geq 0 \ , \tag{2.1.18}$$

$$\widehat{\alpha}_i[y_i(x_i^T\widehat{\beta} + \widehat{\beta}_0) - 1] = \ 0 \ . \tag{2.1.19}$$

From (2.1.17) we obtain the solution for $\widehat{\beta}$, and from the KKT condition $\widehat{\alpha}_i[y_i(x_i^T\widehat{\beta} + \widehat{\beta}_0) - 1] = 0$ we get

$$\widehat{\alpha}_i > 0 \Rightarrow y_i(x_i^T\widehat{\beta} + \widehat{\beta}_0) - 1 = 0 \ . \tag{2.1.20}$$

The observations where it is true that $\widehat{\alpha}_i > 0$ are called the **support vectors** and give name to the model, since the solution for $\widehat{\beta}$ in (2.1.17) is only influenced by these points. This is why it is said that SVMs are sparse models. Using the right part of (2.1.20) we can solve for $\widehat{\beta}_0$ as

$$\widehat{\beta}_0 = \ \frac{1 - y_i x_i^T\widehat{\beta}}{y_i} \ . \tag{2.1.21}$$

Usually an average of the solutions for each support vector point is used for numerical stability reasons. Finally, plugging (2.1.15) and (2.1.21) into (2.1.3) we obtain the final solution function, $\widehat{f}(x)$

$$\widehat{f}(x) = \ \sum_{i=1}^{N} \widehat{\alpha}_i y_i x^T x_i + \widehat{\beta}_0 \ . \tag{2.1.22}$$

2.1.2 Linear Non-Separable Case. Soft Margin Classification

In real-world problems, usually finding a hyperplane which separates perfectly the data is not possible, which leads to the non-separable case. Furthermore, even if it is possible to find this hyperplane, it might not be desirable because the probability of the model overfitting the data, due to the outliers present in the dataset, is rather high and normally a decision boundary that ignores some points of the data which do not represent the general behavior of the problem is preferred. An example of this problem, where a hard margin hyperplane presents a severe overfitting problem due to one outlier point, is shown in Figure 2.1.2.

To deal with the overlap, the idea is to still maximize the margin, M, but allowing some points of the dataset to be on the wrong side of the margin. Defining the slack variables $\xi = (\xi_1, \xi_2, ..., \xi_N)$, one natural way to modify the constraint shown in (2.1.4) will be

$$y_i(x_i^T\beta + \beta_0) \geq \frac{1}{||\beta||} - \xi_i, \ i = 1, ...N \ , \xi_i > 0 \ , \tag{2.1.23}$$

where the value ξ_i is the value of the amount by which the point x_i is on the wrong side of its margin, as represented in Figure 2.1.3 .

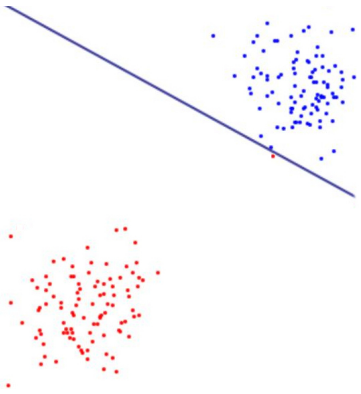


Figure 2.1.2: Hard Margin Classification overfitting. The hyperplane adapts in excess to a single outlier red point.

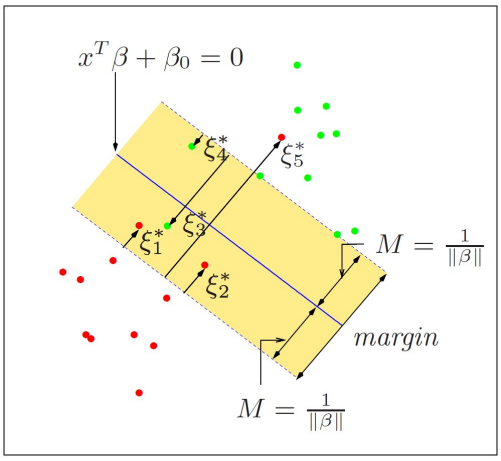


Figure 2.1.3: Linearly Non-Separable case. Hyperplane allows some points of the dataset to be on the wrong side of the margin. Image from [5].

Although this choice seems very natural, since it measures actual distance from the margin $M = \frac{1}{||\beta||}$, unfortunately it results in a nonconvex optimization problem, which leads to uniqueness of the solution not being assured. To define a convex optimization problem, where any local minimum of the unconstrained optimization problem is a global minimum and, hence, the solution is unique, the following modification is carried out

$$y_i(x_i^T\beta + \beta_0) \geq \frac{1}{||\beta||}(1 - \xi_i), \ i = 1, ...N \ , \tag{2.1.24}$$

where now ξ_i is again the distance by which the point x_i is on the wrong side of its margin, but expressed in relative value with respect to M .

With this formulation of the SVM problem misclassifications occur when $\xi_i > 1$, whereas other points on the wrong side of the margin but where $0 < \xi_i < 1$ are still predicted by the model as the correct class. Therefore, bounding $\sum_{i=1}^{N} \xi_i$ at a value λ sets the upper bound of the total number of training misclassifications to be λ.

As we did in the separable case, we can write the equation in the equivalent form

$$\begin{aligned} \min_{\beta,\beta_0} \quad & \frac{1}{2}||\beta||^2 \\ \text{subject to} \quad & y_i(x_i^T\beta+\beta_0) \geq 1-\xi_i,\ i=1,...N\ , \\ & \xi_i \geq 0,\ i=1,...N\ , \\ & \sum_{i=1}^{N} \xi_i \leq \lambda\ . \end{aligned} \tag{2.1.25}$$

As described in [5], computationally it is convenient to re-express (2.1.25) as

$$\begin{aligned} \min_{\beta,\beta_0} \quad & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i(x_i^T\beta+\beta_0) \geq 1-\xi_i,\ i=1,...N\ , \\ & \xi_i \geq 0,\ i=1,...N\ , \end{aligned} \tag{2.1.26}$$

where the parameter C, commonly called the *cost*, replaces the role of λ in (2.1.25).

From (2.1.26), it is straightforward to see that the hard margin case corresponds to $C=\infty$ that leads to $\sum_{i=1}^{N}\xi=0$, i.e. not a single point can be on the wrong side of the hyperplane margins.

The primal problem in (2.1.26) is quadratic with a positive semi-definite matrix and with linear inequality constraints, hence it is a convex optimization problem. Existence of the solution is guaranteed by the quadratic nature of the objective function and uniqueness of the global optima is ensured due to its convex nature. As in the separable case, the problem solved in practice is the dual formulation derived using Lagrangian techniques. Once more, first we get the Lagrangian, that in this case has the form

$$L = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta+\beta_0)-(1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i\ , \alpha_i,\mu_i \geq 0. \tag{2.1.27}$$

To obtain the formulation for the primal values that minimize this function, we compute the derivatives in (2.1.27) , obtaining

$$\begin{aligned} \frac{\partial L}{\partial \beta_0} &= -\sum_{i=1}^{N}\alpha_i y_i\ , \\ \frac{\partial L}{\partial \beta} &= \beta - \sum_{i=1}^{N}\alpha_i y_i x_i\ , \\ \frac{\partial L}{\partial \xi_i} &= C-\alpha_i-\mu_i,\ i=1,...N\ , \end{aligned} \tag{2.1.28}$$

and setting the derivatives (2.1.28) to zero we get

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \ , \tag{2.1.29}$$

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \ , \tag{2.1.30}$$

$$\alpha_i = C - \mu_i, \ i = 1, ...N \ . \tag{2.1.31}$$

Plugging these equations into (2.1.26) we finally obtain the **dual problem**

$$\begin{array}{ll} \max\limits_{\alpha_i} & D = \sum\limits_{i=1}^{N} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{subject to} & \alpha_i \geq 0, \ i = 1, ...N \ , \\ & \alpha_i \leq C, \ i = 1, ...N \ , \\ & \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \ , \end{array} \tag{2.1.32}$$

with the following **KKT conditions** at the optimal point

$$\begin{array}{l} \widehat{\beta} = \sum\limits_{i=1}^{N} \widehat{\alpha}_i y_i x_i \ , \\ \widehat{\alpha}_i [y_i (x_i^T \widehat{\beta} + \widehat{\beta}_0) - (1 - \widehat{\xi}_i)] = 0 \ , \\ \widehat{\mu}_i \widehat{\xi}_i = 0, \ i = 1, ...N \Rightarrow \ (C - \widehat{\alpha}_i) \widehat{\xi}_i = 0 \ . \end{array} \tag{2.1.33}$$

Thus, from (2.1.32) we see that, as in the case of the hard margin classificator in the previous section, the solution for $\widehat{\beta}$ has the form (2.1.17). Using the KKT condition $\widehat{\alpha}_i [y_i (x_i^T \widehat{\beta} + \widehat{\beta}_0) - (1 - \widehat{\xi}_i)] = 0$ we get

$$\widehat{\alpha}_i > 0 \Rightarrow y_i (x_i^T \widehat{\beta} + \widehat{\beta}_0) - (1 - \widehat{\xi}_i) = 0 \ , \tag{2.1.34}$$

These points are the support vectors for the non-separable case, and the solution for $\widehat{\beta}$ in (2.1.17) is only influenced by them. For the support vectors where it holds that $0 < \widehat{\alpha}_i < C$, (2.1.31) gives us that $\widehat{\mu}_i = C - \widehat{\alpha}_i > 0$. Plugging this into the KKT condition $\widehat{\mu}_i \widehat{\xi}_i = 0$, we get that $\widehat{\xi}_i = 0$, i.e., these are the points that will lie on the edge of the margin. Taking into account (2.1.34), for these points the following will hold

$$y_i (x_i^T \widehat{\beta} + \widehat{\beta}_0) - 1 = 0 \ . \tag{2.1.35}$$

From (2.1.35) we see that the solution for $\widehat{\beta}_0$ can be obtained using any of these points lying in the margin and is given again by (2.1.21). The remaining support vectors are characterized by

$$\widehat{\xi}_i > 0 \Rightarrow \widehat{\alpha}_i = C \ , \tag{2.1.36}$$

due to the KKT condition $(C - \widehat{\alpha}_i) \widehat{\xi}_i = 0$.

2.1.3 Non-Linear Non-Separable Case

The two previous sections focus on describing how to find linear boundaries in the input feature space. We can enlarge the feature space using basis expansions such as polynomials or splines, for example. Generally linear boundaries in the enlarged space achieve better class separation, and translate to nonlinear boundaries in the original feature space.

Once the basis functions $\{h_m(x)\}_{m=1}^{M}$ are selected, the procedure is the same as before. We use this time as input features $\{h(x_i)\}_{i=1}^{N} = \{(h_1(x_i), h_2(x_i), ..., h_M(x_i))\}_{i=1}^{N}$ instead of the original $\{x_i\}_{i=1}^{N}$, and produce the function

$$f(x) = \ < h(x), \beta > + \beta_0 \ , \tag{2.1.37}$$

where this time $f(x)$ is a non-linear function.

Replacing $\{x_i\}_{i=1}^{N}$ for the new input features $\{h(x_i)\}_{i=1}^{N}$ in (2.1.32) we get, instead of the formulation in (2.1.32), the following dual function

$$D = \ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j < h(x_i), h(x_j) > \ . \tag{2.1.38}$$

Doing the same replacement in (2.1.30) we have

$$\widehat{\beta} = \ \sum_{i=1}^{N} \widehat{\alpha}_i y_i h(x_i) \ . \tag{2.1.39}$$

Furthermore, plugging (2.1.39) into (2.1.37) we obtain

$$\widehat{f}(x) = \ \sum_{i=1}^{N} \widehat{\alpha}_i y_i < h(x), h(x_i) > + \widehat{\beta}_0 \ . \tag{2.1.40}$$

Looking at (2.1.38) and (2.1.40) we can see that $h(x)$ is involved only through inner products, i.e. $< h(x), h(x_i) >$. Thus, we do not need to specify explicitly the transformation $h(x)$, needing only to know the **kernel function** that defines this product

$$k(x, x') = \ < h(x), h(x') > \ . \tag{2.1.41}$$

Thus, we can reformulate (2.1.38) and (2.1.40) as

$$D = \ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \ , \tag{2.1.42}$$

$$\widehat{f}(x) = \ \sum_{i=1}^{N} \widehat{\alpha}_i y_i k(x, x_i) + \widehat{\beta}_0 \ , \tag{2.1.43}$$

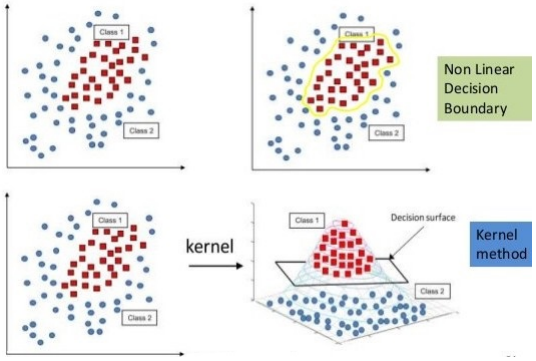


Figure 2.1.4: Example of non-linear SVM. Kernel trick makes the two classes separable. Image from [21].

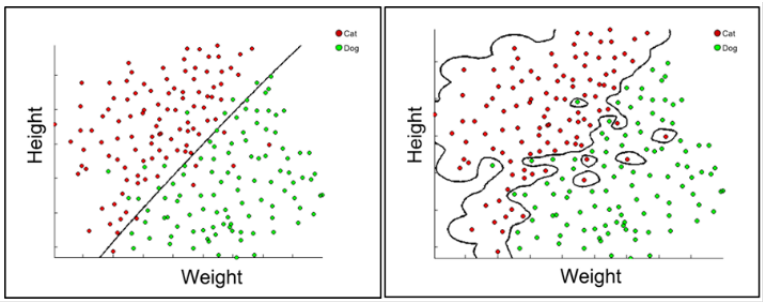


Figure 2.1.5: Low C values, on the left, tend to underfitting. High C values, on the right, tend to overfitting.

where the basis functions do not appear explicitly, but only through their inner products defined by the kernel k. An example of the effect of applying these kernel functions can be seen in Figure 2.1.4. This important property of support vector machines is often called the **kernel trick** as is one of the reasons of the usefulness and popularity of SVM models during all these years.

This kernel trick allows us to make the dimension of the enlarged space very large, infinite in some cases, only defining a suitable kernel function which satisfies some particular properties that we will define later. It might seem that, since perfect separation is often achievable in these enlarged spaces, overfitting would occur. Here is when the role of the cost parameter C becomes clearer. A large value of C will discourage any positive ξ_i, and lead to high variance and an overfit wiggly boundary in the original feature space, while a value of C too small will encourage a small value of $||\beta||^2$, which in turn causes $f(x)$ and hence the hyperplane to have high bias and low variance, tending to underfit the model. Therefore, the bias-variance tradeoff mentioned in Section 1.2 is controlled through this cost parameter. Figure 2.1.5 shows this phenomenon.

One of the most popular functions used as kernel for SVM models is the Radial Basis or **Gaussian Kernel**:

$$k(x, x') = \quad e^{-\gamma||x-x'||^2} \ . \tag{2.1.44}$$

The Gaussian kernel has been shown in the past to be the best choice for SVM models for several tasks [22]. However, there are other possibilities to be used as kernel functions. In fact, a function needs only to verify Mercer's condition, i.e., to be positive semi-definite, to be a valid kernel function. The condition stated in Mercer's theorem is the following:

Theorem 1. *Mercer's Theorem. If a scalar function $k(x_i, x_j)$ is positive semi-definite,i.e.*

$$\iint k(x_i, x_j)g(x_i)g(x_j)dx_i dx_j \geq 0 \ \forall \ g \in L_2 \ ,$$

then there is a mapping function $\phi : \mathbb{R}^d \to F$, with F a Hilbert space, such that k can be decomposed as an inner product

$$k(x_i, x_j) =< \phi(x_i), \phi(x_j) > \ .$$

2.2 SVM for Regression, SVR

The support vector method can also be applied to regression. When SVM models are applied to regression problems, they are usually called Support Vector Regression, SVR. We present here the ϵ-SVR formulation as well as the quadratic L2-SVR one. Although the first one is usually the standard choice, L2-SVR, where outliers are greatly penalized, is also commonly used and thus we describe both versions in detail here.

2.2.1 ϵ-SVR

As the case with its classification counterpart, we can divide the standard SVR formulation into two different cases, the linear one and the non-linear case. In SVR, the linear regression model is considered to be

$$f(x) = \quad x^T\beta + \beta_0 \ . \tag{2.2.1}$$

To obtain the optimal $f(x)$, an objective function analogous to the one described in (1.2.3) is minimized, but this time with other loss function different to the hinge error. The loss function used for standard ϵ-SVR is called the ϵ–**insensitive loss function, or** ϵ–**ILF**, and is defined as

$$l_\epsilon(\delta) = \begin{cases} -\delta - \epsilon, & \delta < -\epsilon \ , \\ 0, & \delta \in [-\epsilon, \epsilon] \ , \\ \delta - \epsilon, & \delta > \epsilon \ . \end{cases} \ . \tag{2.2.2}$$

A visualization of the ϵ-ILF function can be seen in Figure 2.2.1. This loss function, as is the case in the linear MAE error, provides robustness against outliers. However, it is not only a robust cost function because of its linear behavior outside the interval $[-\epsilon, \epsilon]$, but it is also sparse in the sense that it ignores the errors within a certain margin, ϵ, to the target value, y_i, assigning zero cost to errors smaller than ϵ.

The quadratic loss function, generally used in regression, is well justified under the assumption of Gaussian additive noise in the data. However, the noise model underlying the choice of the ϵ-ILF is not so clear. In [23], the use of the ϵ-ILF is partially justified under the assumption that the noise is additive and Gaussian, where the variance and mean of the Gaussian are random variables.

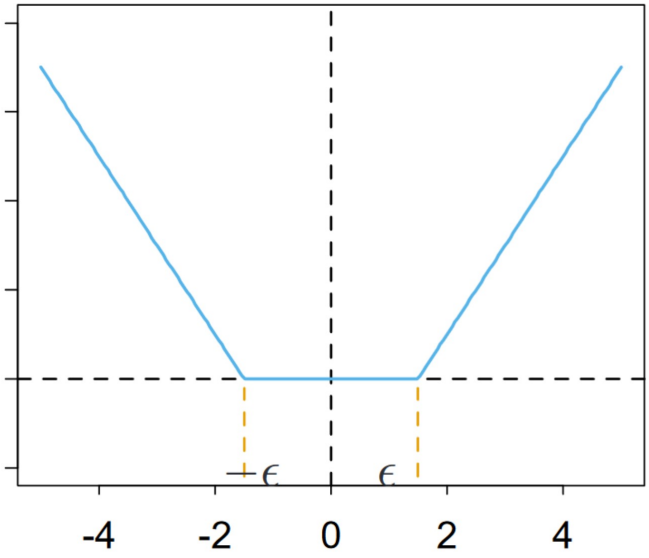


Figure 2.2.1: The ϵ–insensitive loss function. Image from [5].

This l_ϵ loss fuction is employed in combination with the ridge regression regularization to get the objective function and the optimization problem finally used in standard SVR, which is the following one

$$\min_{\beta,\beta_0} \quad H(\beta,\beta_0) = \sum_{i=1}^{N} l_\epsilon(y_i - f(x)) + \frac{\lambda}{2}||\beta||^2 \ . \tag{2.2.3}$$

As shown in [5] and [19] this formulation is equivalent to the following problem

$$\begin{aligned} \min_{\beta,\beta_0,\xi_i,\xi_i^*} \quad & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & \xi_i, \xi_i^* \geq 0, \ i = 1, ...N \ , \\ & f(x_i) - y_i \leq \epsilon + \xi_i, \ i = 1, ...N \ , \\ & y_i - f(x_i) \leq \epsilon + \xi_i^*, \ i = 1, ...N \ . \end{aligned} \tag{2.2.4}$$

where the ξ_i values quantify the errors above the ϵ-band, and ξ_i^* the ones below the ϵ-band, as can be seen in Figure 2.2.2.

The Lagrange function corresponding to 2.2.4 is

$$\begin{aligned} L = & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}\alpha_i(y_i - x_i^T\beta + \beta_0 + \epsilon + \xi_i) - \\ & \sum_{i=1}^{N}\alpha_i^*(x_i^T\beta + \beta_0 - y_i + \epsilon + \xi_i^*) - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\mu_i^*\xi_i^* \\ & \text{with } \alpha_i, \alpha_i^*, \mu_i, \mu_i^* \geq 0 \ . \end{aligned} \tag{2.2.5}$$

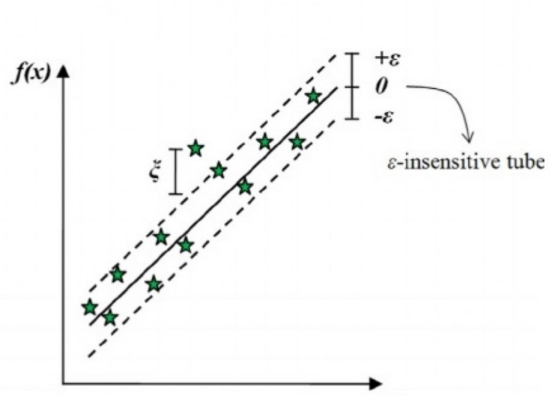


Figure 2.2.2: Linear SVR. Errors inside the ϵ-band are not penalized. Image from [19].

Computing the derivatives in (2.2.5) with respect to the primal variables we obtain

$$\begin{aligned}
\frac{\partial L}{\partial \beta_0} &= \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \ , \\
\frac{\partial L}{\partial \beta} &= \beta + \sum_{i=1}^{N} \alpha_i x_i - \sum_{i=1}^{N} \alpha_i^* x_i = \beta + \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) x_i \ , \\
\frac{\partial L}{\partial \xi_i} &= C - \alpha_i - \mu_i, \ i = 1, ...N \ , \\
\frac{\partial L}{\partial \xi_i^*} &= C - \alpha_i^* - \mu_i^*, \ i = 1, ...N \ ,
\end{aligned} \tag{2.2.6}$$

and setting all these derivatives to zero we get

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = \ 0 \ , \tag{2.2.7}$$

$$\beta = \ \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) x_i \ , \tag{2.2.8}$$

$$\alpha_i = \ C - \mu_i, \ i = 1, ...N \ , \tag{2.2.9}$$

$$\alpha_i^* = \ C - \mu_i^*, \ i = 1, ...N \ . \tag{2.2.10}$$

Plugging (2.2.8), (2.2.7), (2.2.9) and (2.2.10) into (2.2.5) we get the dual problem for the standard SVR formulation

$$\begin{aligned} \max_{\alpha_i,\alpha_i^*} \quad & D = \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)x_i^T x_j - \epsilon\sum_{i=1}^{N}(\alpha_i^* + \alpha_i) \\ \text{subject to} \quad & \alpha_i, \alpha_i^* \geq 0,\ i = 1, ...N\ , \\ & \alpha_i, \alpha_i^* \leq C,\ i = 1, ...N\ , \\ & \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0\ . \end{aligned} \tag{2.2.11}$$

The KKT conditions at the optimal point are now

$$\begin{aligned} & \widehat{\beta} = \sum_{i=1}^{N}(\widehat{\alpha}_i^* - \widehat{\alpha}_i)x_i\ , \\ & \widehat{\alpha}_i(y_i - \widehat{f}(x_i) + \epsilon + \xi_i) = 0\ , \\ & \widehat{\alpha}_i^*(\widehat{f}(x_i) - y_i + \epsilon + \xi_i^*) = 0\ , \\ & (C - \widehat{\alpha}_i)\xi_i = 0\ , \\ & (C - \widehat{\alpha}_i^*)\xi_i^* = 0\ . \end{aligned} \tag{2.2.12}$$

The solution for $\widehat{\beta}$ is given by (2.2.8). In this case we have that support vectors are the points where $(\widehat{\alpha}_i^* - \widehat{\alpha}_i) \neq 0$, as these are the ones affecting the value of $\widehat{\beta}$. This is equivalent to stating that the support vector points are the ones where either $\widehat{\alpha}_i^* > 0$ or $\widehat{\alpha}_i > 0$, as both values cannot be different from zero for the same point. In order to prove that the latter statement is true, let us imagine a point where $\widehat{\alpha}_i, \widehat{\alpha}_i^* > 0$. In this situation and due to conditions $\widehat{\alpha}_i(y_i - \widehat{f}(x_i) + \epsilon + \widehat{\xi}_i) = 0$ and $\widehat{\alpha}_i^*(\widehat{f}(x_i) - y_i + \epsilon + \widehat{\xi}_i^*) = 0$ in (2.2.11) we get that

$$y_i - \widehat{f}(x_i) + \epsilon + \widehat{\xi}_i = 0 \Rightarrow \widehat{\xi}_i = \widehat{f}(x_i) - y_i - \epsilon\ , \tag{2.2.13}$$

$$\widehat{f}(x_i) - y_i + \epsilon + \widehat{\xi}_i^* = 0 \Rightarrow \widehat{\xi}_i^* = y_i - \widehat{f}(x_i) - \epsilon\ . \tag{2.2.14}$$

Now, as per definition $\widehat{\xi}_i$ and $\widehat{\xi}_i^*$ cannot be greater than 0 for the same point, three cases are possible:

1. $\widehat{\xi}_i = 0$: Due to (2.2.13) we have that $\epsilon = \widehat{f}(x_i) - y_i$. If we plug this into (2.2.14) we get $\widehat{\xi}_i^* = 2(y_i - \widehat{f}(x_i))$ which is incompatible with the definition of $\widehat{\xi}_i^*$ as the prediction errors below the ϵ-band.
2. $\widehat{\xi}_i^* = 0$: Due to (2.2.14) we have that $\epsilon = y_i - \widehat{f}(x_i)$. If we plug this into (2.2.13) we get $\widehat{\xi}_i = 2(\widehat{f}(x_i) - y_i)$ which is incompatible with the definition of $\widehat{\xi}_i$ as the prediction errors above the ϵ-band.
3. $\widehat{\xi}_i = \widehat{\xi}_i^* = 0$: Due to (2.2.13) and (2.2.14) we have that $\epsilon = \widehat{f}(x_i) - y_i = y_i - \widehat{f}(x_i)$. This can only be true if $\widehat{f}(x_i) = y_i$ and $\epsilon = 0$, the latter expression being incompatible with the use of the ϵ-ILF cost function with $\epsilon > 0$.

On the other hand, the solution for $\widehat{\beta}_0$ can be obtained using any of the support vector points in the $\epsilon-$border, i.e, those where $\widehat{\xi}_i = 0$ and $\widehat{\xi}_i^* = 0$, and the KKT conditions $\widehat{\alpha}_i(y_i - \widehat{f}(x_i) + \epsilon + \widehat{\xi}_i) = 0$ and $\widehat{\alpha}_i^*(\widehat{f}(x_i) - y_i + \epsilon + \widehat{\xi}_i^*) = 0$, respectively, and has the form

$$\widehat{\beta}_0 = \begin{cases} y_i + \epsilon - x_i^T\widehat{\beta} \ , \ 0 < \widehat{\alpha}_i < C \ , \\ y_i - \epsilon - x_i^T\widehat{\beta} \ , \ 0 < \widehat{\alpha}_i^* < C \ . \end{cases} \quad . \tag{2.2.15}$$

Normally an average of the values obtained in (2.2.15) for all these support vector points in the $\epsilon-$border is used. Therefore, the final expression of $\widehat{\beta}_0$ is

$$\widehat{\beta}_0 = \frac{1}{2}\left[\frac{1}{|S|}\sum_{i\in S}(y_i + \epsilon - x_i^T\widehat{\beta}) + \frac{1}{|S^*|}\sum_{i\in S^*}(y_i - \epsilon - x_i^T\widehat{\beta})\right] \ , \tag{2.2.16}$$

where S is the set of all the support vector points in the $\epsilon-$border where $\widehat{\alpha}_i \in (0, C)$ and S^* the ones where $\widehat{\alpha}_i^* \in (0, C)$.

Therefore, the final solution function can be shown to have the form

$$\widehat{f}(x) = \sum_{i=1}^{N}(\widehat{\alpha^*}_i - \widehat{\alpha}_i)x_i^T x + \widehat{\beta}_0 \ . \tag{2.2.17}$$

As occurred in the classification case, we can enlarge the feature space to large dimensions, even infinite, through basis expansions. Replacing the original input features, $\{x_i\}_{i=1}^N$, with their corresponding features in the enlarged space, $\{h(x_i)\}_{i=1}^N$, in (2.2.11) we get the corresponding dual function for the SVR formulation

$$\begin{aligned} D = &\sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) < h(x_i), h(x_j) > - \\ &\epsilon\sum_{i=1}^{N}(\alpha_i^* + \alpha_i) \ . \end{aligned} \tag{2.2.18}$$

Moreover, applying the same replacement approach leading to (2.2.17) we obtain

$$\widehat{f}(x) = \sum_{i=1}^{N}(\widehat{\alpha^*}_i - \widehat{\alpha}_i) < h(x_i), h(x) > + \widehat{\beta}_0 \ . \tag{2.2.19}$$

Once again, the values $h(x)$ only appear through their inner products, so using a kernel function $k(x_i, x_j) = < h(x_i), h(x_j) >$ satisfying the Mercer's condition defined previously in Theorem 1, we can get the following equations equivalent to (2.2.18) and (2.2.19)

$$\begin{aligned} D = &\sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) - \\ &\epsilon\sum_{i=1}^{N}(\alpha_i^* + \alpha_i) \ , \end{aligned} \tag{2.2.20}$$

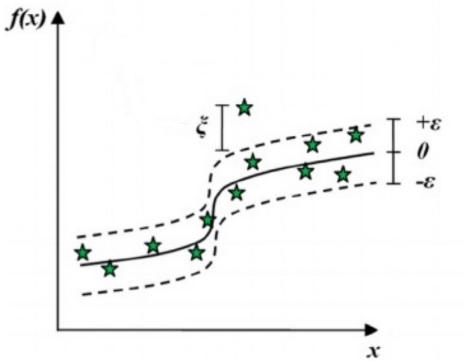


Figure 2.2.3: Non-linear SVR. In this case, the SVR estimator follows a non-linear shape due to the application of the kernel trick. Image from [19].

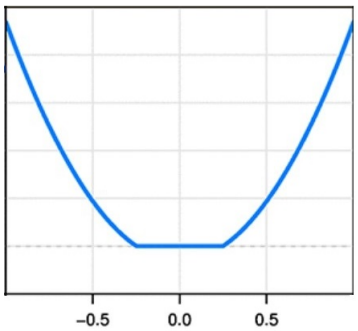


Figure 2.2.4: Quadratic ϵ-insensitive loss function with $\epsilon = 0.25$.

$$\widehat{f}(x) = \sum_{i=1}^{N} (\widehat{\alpha^*}_i - \widehat{\alpha}_i) k(x, x_i) + \widehat{\beta}_0 \ . \tag{2.2.21}$$

Proceeding like this there is no need of an explicit definition of the basis functions $\{h_m(x)\}_{m=1}^{M}$. Figure 2.2.3 shows an example of a one dimensional non-linear SVR function with an ϵ-insensitive band.

2.2.2 L2-SVR

Although the ϵ-ILF cost function defined in (2.2.2) is clearly the standard choice when building SVR models, sometimes a quadratic ϵ-insensitive loss function, or L2-ϵ-ILF, is preferred. This L2-ϵ-ILF is defined as follows

$$l2_\epsilon(\delta) = \begin{cases} (-\delta - \epsilon)^2, & \delta < -\epsilon \ , \\ 0, & \delta \in [-\epsilon, \epsilon] \ , \\ (\delta - \epsilon)^2, & \delta > \epsilon \ . \end{cases} \tag{2.2.22}$$

Figure 2.2.4 shows how this loss function looks. It has the same sparseness property as the standard ϵ-ILF, or L1-ILF, since values in the interval $[-\epsilon, \epsilon]$ all are given zero value.

Nevertheless, in contrast to the L1-ILF, this loss function is not robust to outliers, since a quadratic penalty is given to errors with absolute value greater than ϵ, as would be the case when using a MSE penalty instead of MAE. This lack of robustness may suppose an important drawback for some problems, so appropriateness of the election of L2-SVR must be carefully considered. The choice, again, is problem-dependent. Therefore, both versions can be useful depending on the problem at hand, and thus are discussed here.

Using this L2-ϵ-ILF instead of the L1-ϵ-ILF, and working directly with the basis expansions $\{h(x_i)\}_{i=1}^N$, we get the following Lagrange function

$$\begin{aligned} L =& \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}(\xi_i^2 + \xi_i^{*2}) - \sum_{i=1}^{N}\alpha_i(y_i - f(h(x_i)) + \epsilon + \xi_i) - \\ & \sum_{i=1}^{N}\alpha_i^*(f(h(x_i)) - y_i + \epsilon + \xi_i^*) - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\mu_i^*\xi_i^* \\ & \text{with } \alpha_i, \alpha_i^*, \mu_i, \mu_i^* \geq 0 \ . \end{aligned} \tag{2.2.23}$$

Computing the derivatives in (2.2.23) with respect to the primal variables we obtain

$$\begin{aligned} \frac{\partial L}{\partial \beta_0} &= \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \ , \\ \frac{\partial L}{\partial \beta} &= \beta + \sum_{i=1}^{N}\alpha_i h(x_i) - \sum_{i=1}^{N}\alpha_i^* h(x_i) = \beta + \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)h(x_i) \ , \\ \frac{\partial L}{\partial \xi_i} &= 2C\xi_i - \alpha_i - \mu_i, \ i = 1, ...N \ , \\ \frac{\partial L}{\partial \xi_i^*} &= 2C\xi_i^* - \alpha_i^* - \mu_i^*, \ i = 1, ...N \ , \end{aligned} \tag{2.2.24}$$

and setting the derivatives (2.2.24) to zero we get:

$$\begin{aligned} &\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \ , \\ &\beta = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)h(x_i) \ , \\ &\alpha_i = 2C\xi_i - \mu_i, \ i = 1, ...N \ , \\ &\alpha_i^* = 2C\xi_i^* - \mu_i^*, \ i = 1, ...N \ . \end{aligned} \tag{2.2.25}$$

Finally, plugging (2.2.25) into (2.2.23) we get the dual problem

$$\begin{aligned}
\max_{\alpha_i,\alpha_i^*} \quad & D = \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(k(x_i,x_j) + \frac{1}{C}\delta_{ij}) - \\
& \epsilon\sum_{i=1}^{N}(\alpha_i^* + \alpha_i) \\
\text{subject to} \quad & \alpha_i, \alpha_i^* \geq 0,\ i = 1,...N\ , \\
& \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0\ ,
\end{aligned} \tag{2.2.26}$$

with the following KKT conditions as constraints at the optimal point

$$\begin{aligned}
& \widehat{\alpha}_i = 2C\widehat{\xi}_i,\ i = 1,...N\ , \\
& \widehat{\alpha}_i^* = 2C\widehat{\xi}_i^*,\ i = 1,...N\ , \\
& \widehat{\beta} = \sum_{i=1}^{N}(\widehat{\alpha}_i^* - \widehat{\alpha}_i)h(x_i)\ , \\
& \widehat{\alpha}_i(y_i - \widehat{f}(h(x_i)) + \epsilon + \widehat{\xi}_i) = 0\ , \\
& \widehat{\alpha}_i^*(\widehat{f}(h(x_i)) - y_i + \epsilon + \widehat{\xi}_i^*) = 0\ , \\
& (2C\widehat{\xi}_i - \widehat{\alpha}_i)\widehat{\xi}_i = 0\ , \\
& (2C\widehat{\xi}_i^* - \widehat{\alpha}_i^*)\widehat{\xi}_i^* = 0\ .
\end{aligned} \tag{2.2.27}$$

The support vectors are again the points with $(\widehat{\alpha}_i^* - \widehat{\alpha}_i) \neq 0$, or equivalently the ones where $\widehat{\alpha}_i^* > 0$ or $\widehat{\alpha}_i > 0$. The solutions for $\widehat{\beta}$, $\widehat{\beta}_0$ and $\widehat{f}(x)$ have the same form described for SVR using the standard ϵ-ILF in (2.2.8), (2.2.16) and (2.2.21), respectively.

2.3 General Noise SVR

As explained before, the use of the ϵ–insensitive loss function in the standard SVR implies the assumption of a particular error distribution in the data [23]. However, it has been observed that the noise in some real-world applications, such as wind or solar power forecasting, satisfies other distributions, including the Beta distribution [24], [25], the Weibull distribution [26] or the Laplacian distribution [27], to name a few. Therefore, it could be interesting to use SVR formulations using loss functions other than the ϵ–ILF where the assumption of the error distribution resembles the one in the data corresponding to the task at hand.

2.3.1 Primal and Dual Formulations

In 2002, a general noise version of SVR was proposed in [28]. This variation of SVR can be used with any particular loss function $c(y_i, f(x_i))$. For instance, if Gaussian noise is assumed to be present in the data, a particular loss function will be inserted, which will be different to the one used if Laplace is assumed to be the underlying distribution. Following a similar approach to the one already described in Section 2.2.1 for classical SVR models, let's define $c(\xi_i) = c(y_i, f(x_i))$ when $\xi_i = [f(x_i) - y_i]_+$, and $c(\xi_i^*) = c(y_i, f(x_i))$ when $\xi_i^* = [y_i - f(x_i)]_+$.

The primal optimization problem corresponding to this general noise SVR is then described as

$$\begin{array}{ll} \min\limits_{\beta,\beta_0,\xi_i,\xi_i^*} & \frac{1}{2}||\beta||^2 + C\sum\limits_{i=1}^{N}(c(\xi_i)+c(\xi_i^*)) \\ \text{subject to} & \xi_i,\xi_i^* \geq 0,\ i=1,...N\ , \\ & f(x_i)-y_i \leq \epsilon+\xi_i,\ i=1,...N\ , \\ & y_i-f(x_i) \leq \epsilon^*+\xi_i^*,\ i=1,...N\ , \end{array} \tag{2.3.1}$$

where ϵ and ϵ^* are chosen such that

$$c(\xi)=0\ ,\ \forall\ \xi \in [-\epsilon^*,\epsilon]\ . \tag{2.3.2}$$

Notice that the formulation allows for the use of different values for ϵ and ϵ^*, i.e., different widths for the band above and below the prediction, respectively, although in the formulation of the classical SVR $\epsilon=\epsilon^*$. Let us define now

$$T(\xi_i)=c(\xi_i)-\xi_i\frac{\partial c}{\partial \xi}(\xi_i)\ ,\quad T^*(\xi_i^*)=c(\xi_i^*)-\xi_i^*\frac{\partial c}{\partial \xi^*}(\xi_i^*)\ . \tag{2.3.3}$$

Using (2.3.3) and denoting again with k the kernel function selected, the following dual formulation derived from (2.3.1) can be found in [29]

$$\begin{array}{ll} \max\limits_{\alpha_i,\alpha_i^*} & D=\sum\limits_{i=1}^{N}\Big(y_i(\alpha_i^*-\alpha_i)-\frac{1}{2}\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{N}(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)K(x_i,x_j)- \\ & \sum\limits_{i=1}^{N}\epsilon_i\alpha_i-\sum\limits_{i=1}^{N}\Big(\epsilon_i^*\alpha_i^*+C\sum\limits_{i=1}^{N}\Big(T(\xi_i)+T^*(\xi_i^*)) \\ \text{subject to} & \alpha_i,\alpha_i^* \geq 0,\ i=1,...N\ , \\ & \sum\limits_{i=1}^{N}\Big(\alpha_i^*-\alpha_i)=0\ , \\ & \alpha_i,\alpha_i^* \leq C\frac{\partial c}{\partial \xi^*}(\xi^*),\ i=1,...N\ , \end{array} \tag{2.3.4}$$

with the KKT conditions at the optimal point being

$$\begin{array}{l} \widehat{\alpha}_i(y_i-\widehat{f}(x_i)+\epsilon_i+\xi_i)=0\ , \\ \widehat{\alpha}_i^*(\widehat{f}(x_i)-y_i+\epsilon_i^*+\xi_i^*)=0\ , \\ (C\frac{\partial c}{\partial \xi}(\xi_i)-\widehat{\alpha}_i)\xi_i=0\ , \\ (C\frac{\partial c}{\partial \xi^*}(\xi_i^*)-\widehat{\alpha}_i^*)\xi_i^*=0\ , \end{array} \tag{2.3.5}$$

The formulation in (2.3.4) allows us to build a general noise SVR model based on any particular choice of cost function. The process encompasses the following steps:

1. Determine the cost function, $c(\xi)$, to be used. Next section describes how to obtain the optimal cost function for a particular error distribution assumption.
2. Plug $c(\xi)$ into the dual formulation in (2.3.4).
3. Solve for (2.3.4).

However, plugging general cost functions into the general dual formulation problem defined in (2.3.4) often leads to Sequential Minimal Optimization [30], SMO, the algorithm usually employed to solve the quadratic programming problem that arises during the training of support vector machines, both for classification and regression problems, no longer being a feasible choice as optimization method. Therefore, other optimization methods should be used to avoid this problem. This topic is discussed in more detail in later sections.

Furthermore, the concept of support vectors may not longer be present when using generic cost functions. Therefore, these models are referred as general noise kernel-based models from now on throughout this work.

2.3.2 Optimal Cost Function

The next step will be to obtain the optimal cost function in a maximum likelihood sense for a particular choice of error distribution in the data. We assume the noise in the data is additive and

$$f(x_i) = \quad y_i + \xi_i, \ i = 1, ...N \ , \tag{2.3.6}$$

where ξ_i are independent and identically distributed, i.i.d., random variables. Following [25], the general approach to obtain the optimal cost function is to minimize

$$H[f] = \quad \sum_{i=1}^{N} c(\xi_i) + \lambda \Phi[f] \ , \tag{2.3.7}$$

where λ is a positive number and $\Phi[f]$ is a smoothness functional that acts as a regularizer.

A probabilistic approach is now taken, and the function f is regarded as the realization of a random field with a known prior probability distribution, denoted $P[f]$. The goal is to maximize the posterior probability of f given the data D, i.e., $P[f|D]$. Using the Bayes Theorem one can get to the following formulation for this probability

$$P[f|D] = \quad \frac{P[D|f]P[f]}{P[D]} \propto P[D|f]P[f] \ , \tag{2.3.8}$$

$P[D|f]$ in (2.3.8) represents the conditional probability of the data D given the function f. Therefore, $P[D|f]$ is essentially a model of the noise, and if this noise is assumed to be additive, as in (2.3.6), and i.i.d. with probability distribution $p(\xi_i)$, this conditional probability can be written as

$$P[D|f] = \quad \prod_{i=1}^{N} p(\xi_i) \ . \tag{2.3.9}$$

As explained in [25] the prior is often written as

$$P[f] \propto \quad e^{-\lambda\Phi[f]} \ . \tag{2.3.10}$$

Now, replacing (2.3.10) and (2.3.9) into equation (2.3.8) we have

$$P[f|D] \propto \quad e^{-\lambda\Phi[f]} \prod_{i=1}^{N} p(\xi_i) \ . \tag{2.3.11}$$

We want to maximize $P[f|D]$ or, equivalently, to minimize $-\log\left(P[f|D]\right)$, for which we have that

$$\begin{aligned} -\log\left(P[f|D]\right) &\propto -\log \quad e^{-\lambda\Phi[f]} \prod_{i=1}^{N} p(\xi_i) \\ &= -\log\left(e^{-\lambda\Phi[f]}\right) - \log \quad \prod_{i=1}^{N} p(\xi_i) \\ &= \lambda\Phi[f] - \sum_{i=1}^{N} \log p(\xi_i) \ . \end{aligned} \tag{2.3.12}$$

Combining equations (2.3.7) and (2.3.12) we can write the following

$$H[f] = \quad \sum_{i=1}^{N} c(\xi_i) + \lambda\Phi[f] = \sum_{i=1}^{N} -\log p(\xi_i) + \lambda\Phi[f] \ , \tag{2.3.13}$$

which leads to the conclusion that the optimal loss function in a maximum likelihood sense for a given error distribution, $p(\xi_i)$, is

$$c(\xi_i) = \quad -\log p(\xi_i) = -\log p(f(x_i) - y_i) \ . \tag{2.3.14}$$

Using (2.3.14) we can obtain now the optimal loss function for a given choice of noise distributions. However, the cost function resulting from this reasoning might be nonconvex. In this case, one may have to find a convex proxy in order to deal with the optimization problem or use a non-convex optimization method, such as the one proposed in [31] for SVMs.

Due to this and other reasons, related to the loss of some mathematical properties that existed when using the ϵ-ILF function, plugging general cost functions into the general dual formulation problem defined in (2.3.4) often leads to SMO no longer being a feasible choice as optimization method, as we mentioned before. Therefore, choosing a new optimization algorithm to solve general noise SVR formulations avoiding this problem will be one of the steps we aim to solve in our proposed approach.

2.3.3 Loss Function and Dual Problem for Different Noise Distributions

In order to get the formulation of a general noise SVR for a particular choice of noise distribution it is necessary to follow these steps:

- Decide the noise distribution to be used.
- Compute the corresponding optimal loss function for that noise distribution.
- Insert that loss function into the general noise SVR formulation shown in (2.3.1).
- Get the Lagrange formulation corresponding to the result of the previous step.
- Compute derivatives to obtain the dual problem.

We will show in this section these steps for the Laplace and Gaussian distributions, both for the zero and non-zero mean cases. We will also describe these steps for the Soft Insensitive Loss Function, SILF, an alternative loss function to the standard ϵ-ILF [32]. Although we will not be making use of this loss function for the proposed methods or experiments described in this work, we considered that it was interesting to include it in this section for the sake of completeness.

2.3.3.1 Laplace

Let us start with the simpler zero-mean case. The error distribution is then assumed to be

$$p(\xi_i) = \frac{1}{2\sigma} e^{-\frac{|\xi_i|}{\sigma}} , \tag{2.3.15}$$

where $\sigma > 0$ is a parameter to be estimated and in Section 2.4.2.1 we will describe how to obtain the optimal estimate for it. Replacing (2.3.15) into (2.3.14) we obtain

$$c(\xi_i) = -\log\left(\frac{1}{2\sigma} e^{-\frac{|\xi_i|}{\sigma}}\right) = -\log\left(\frac{1}{2\sigma}\right) \left(-\log\left(e^{-\frac{|\xi_i|}{\sigma}}\right)\right) = -\log\left(\frac{1}{2\sigma}\right) \left(+\frac{|\xi_i|}{\sigma}\right) . \tag{2.3.16}$$

The term $-\log\left(\frac{1}{2\sigma}\right)$ is independent of ξ_i and therefore a constant for any error value, so it is valid to ignore it from now on and work with the following expression instead

$$c(\xi_i) = \frac{|\xi_i|}{\sigma} = \frac{|f(x_i) - y_i|}{\sigma} . \tag{2.3.17}$$

As described in Section 2.3.1, in particular equation (2.3.2), for the general noise SVR formulation given in [28] we have that

$$c(\xi) = 0 \ , \ \forall \ \xi \in [-\epsilon^*, \epsilon] \ . \tag{2.3.18}$$

Taking into the account the definion of $c(\xi)$ in (2.3.17), this implies necessarily that $\epsilon_i^* = \epsilon_i = 0$. This is an important property, as it means that in this SVR formulation we will not have a hyperparameter ϵ.

Now, inserting (2.3.17) into (2.3.1) we arrive to the following general formulation of the primal problem for Laplace noise

$$\begin{array}{ll} \min\limits_{\beta,\beta_0,\xi_i,\xi_i^*} & \frac{1}{2}||\beta||^2 + \frac{C}{\sigma}\sum\limits_{i=1}^{N}(\xi_i + \xi_i^*) \\ \text{subject to} & \xi_i, \xi_i^* \geq 0,\ i = 1, ...N\ , \\ & f(x_i) - y_i \leq \xi_i,\ i = 1, ...N\ , \\ & y_i - f(x_i) \leq \xi_i^*,\ i = 1, ...N\ . \end{array} \tag{2.3.19}$$

where $f(x_i) =< h(x_i), \beta > +\beta_0$ and h the corresponding basis expansions.

The Lagrange formulation corresponding to (2.3.19) is

$$\begin{array}{l} L = \frac{1}{2}||\beta||^2 + \frac{C}{\sigma}\sum\limits_{i=1}^{N}(\xi_i + \xi_i^*) - \sum\limits_{i=1}^{N}\alpha_i(y_i - f(x_i) + \xi_i) - \\ \sum\limits_{i=1}^{N}\alpha_i^*(f(x_i) - y_i + \xi_i^*) - \sum\limits_{i=1}^{N}\mu_i\xi_i - \sum\limits_{i=1}^{N}\mu_i^*\xi_i^*\ , \\ \text{with } \alpha_i, \alpha_i^*, \mu_i, \mu_i^* \geq 0\ . \end{array} \tag{2.3.20}$$

Computing the derivatives in (2.3.20) after replacing $f(x_i)$ by $< h(x_i), \beta > +\beta_0$ we obtain

$$\begin{array}{l} \frac{\partial L}{\partial \beta_0} = \sum\limits_{i=1}^{N}(\alpha_i - \alpha_i^*)\ , \\ \frac{\partial L}{\partial \beta} = \beta + \sum\limits_{i=1}^{N}\alpha_i h(x_i) - \sum\limits_{i=1}^{N}\alpha_i^* h(x_i) = \beta + \sum\limits_{i=1}^{N}(\alpha_i - \alpha_i^*)h(x_i)\ , \\ \frac{\partial L}{\partial \xi_i} = \frac{C}{\sigma} - \alpha_i - \mu_i,\ i = 1, ...N\ , \\ \frac{\partial L}{\partial \xi_i^*} = \frac{C}{\sigma} - \alpha_i^* - \mu_i^*,\ i = 1, ...N\ , \end{array} \tag{2.3.21}$$

and setting the derivatives (2.3.21) to zero we get

$$\begin{array}{l} \sum\limits_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0\ , \\ \beta = \sum\limits_{i=1}^{N}(\alpha_i^* - \alpha_i)h(x_i)\ , \\ \alpha_i = \frac{C}{\sigma} - \mu_i,\ i = 1, ...N\ , \\ \alpha_i^* = \frac{C}{\sigma} - \mu_i^*,\ i = 1, ...N\ . \end{array} \tag{2.3.22}$$

Plugging (2.3.22) into (2.3.20) and denoting by k the kernel function we get the dual problem for Laplace noise

$$\begin{aligned} \max_{\alpha_i,\alpha_i^*} \quad & D = \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) \\ \text{subject to} \quad & \alpha_i, \alpha_i^* \geq 0, \ i = 1, ...N \ , \\ & \alpha_i, \alpha_i^* \leq \frac{C}{\sigma}, \ i = 1, ...N \ , \\ & \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \ , \end{aligned} \tag{2.3.23}$$

with the following KKT conditions as constraints at the optimal point $\widehat{\alpha}_i, \widehat{\alpha}_i^*$

$$\begin{aligned} & \widehat{\beta} = \sum_{i=1}^{N}(\widehat{\alpha}_i^* - \widehat{\alpha}_i)h(x_i) \ , \\ & \widehat{\alpha}_i(y_i - \widehat{f}(x_i) + \widehat{\xi}_i) = 0 \ , \\ & \widehat{\alpha}_i^*(\widehat{f}(x_i) - y_i + \widehat{\xi}_i^*) = 0 \ , \\ & \left(\frac{C}{\sigma} - \widehat{\alpha}_i\right)\widehat{\xi}_i = 0 \ , \\ & \left(\frac{C}{\sigma} - \widehat{\alpha}_i^*\right)\xi_i^* = 0 \ . \end{aligned} \tag{2.3.24}$$

It can be seen that (2.3.23) is very similar to the classical SVR formulation given in (2.2.11) setting $\epsilon = 0$ and adding the presence of the parameter σ

Following an analogous procedure, we can get the dual problem formulation for the **non-zero-mean Laplace** loss function choice. The only difference in that case is that the error distribution is assumed to be

$$P(\xi_i) = \frac{1}{2\sigma}e^{-\frac{|\xi_i - m|}{\sigma}} \ , \tag{2.3.25}$$

where $\sigma > 0$ and $m \in (-\infty, \infty)$ are parameters to be optimized and we will give their optimal values in Section 3.3.1. Replacing (2.3.25) into (2.3.14) we get

$$c(\xi_i) = -\log\left(\frac{1}{2\sigma}e^{-\frac{|\xi_i - m|}{\sigma}}\right) = -\log\left(\frac{1}{2\sigma}\right) + \frac{|\xi_i - m|}{\sigma} \ . \tag{2.3.26}$$

The term $-\log(\frac{1}{2\sigma})$ is again independent of ξ_i so can be ignored to work with the following expression

$$c(\xi_i) = \frac{|\xi_i - m|}{\sigma} \ . \tag{2.3.27}$$

Replacing (2.3.17) for (2.3.27) in our previous steps, the dual problem for non-zero Laplace noise can be obtained. First, we have that

$$c(\xi) = 0 \ , \ \forall \ \xi \in [-\epsilon^*, \epsilon] \ . \tag{2.3.28}$$

Taking into account the definion of $c(\xi)$ in (2.3.27), this implies necessarily that $\epsilon_i^* = \epsilon_i = m$. Now, inserting (2.3.27) into (2.3.1) we arrive to the following general formulation of the primal problem for Laplace noise

$$\begin{aligned} \min_{\beta,\beta_0,\xi_i,\xi_i^*} \quad & \frac{1}{2}||\beta||^2 + \frac{C}{\sigma}\sum_{i=1}^{N}(|\xi_i - m| + |\xi_i^* - m|) \\ \text{subject to} \quad & \xi_i, \xi_i^* \geq 0, \ i = 1, ...N \ , \\ & f(x_i) - y_i \leq m + \xi_i, \ i = 1, ...N \ , \\ & y_i - f(x_i) \leq m + \xi_i^*, \ i = 1, ...N \ . \end{aligned} \tag{2.3.29}$$

where again $f(x_i) =< h(x_i), \beta > +\beta_0$ and h represents the corresponding basis expansions. The Lagrange formulation corresponding to (2.3.29) is

$$\begin{aligned} L =& \frac{1}{2}||\beta||^2 + \frac{C}{\sigma}\sum_{i=1}^{N}(|\xi_i - m| + |\xi_i^* - m|) - \sum_{i=1}^{N}\alpha_i(y_i - f(x_i) + m + \xi_i) - \\ & \sum_{i=1}^{N}\alpha_i^*(f(x_i) - y_i + m + \xi_i^*) - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\mu_i^*\xi_i^* \ , \\ & \text{with } \alpha_i, \alpha_i^*, \mu_i, \mu_i^* \geq 0 \ . \end{aligned} \tag{2.3.30}$$

However, when computing the derivatives in (2.3.30) problems arise due to the absolute values in the term $\frac{C}{\sigma}\sum_{i=1}^{N}(|\xi_i - m| + |\xi_i^* - m|)$ which is involved in the derivatives for ξ_i and ξ_i^*. Therefore, the use of SMO over the dual problem for this distribution, and others where analogous problems exist, is not feasible. This is one of the main reasons that justify our proposed approaches, by means of NORMA optimization or Deep Learning frameworks, to build general noise models, which will be described in Chapter 3.

2.3.3.2 Gaussian

Let us start again with the simpler zero-mean case. The error distribution is assumed to be

$$p(\xi_i) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\xi_i^2}{2\sigma^2}} \ , \tag{2.3.31}$$

where $\sigma^2 > 0$. Replacing (2.3.31) into (2.3.14) we obtain

$$\begin{aligned} c(\xi_i) &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{\xi_i^2}{2\sigma^2}}\right) \\ &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \log\left(e^{-\frac{\xi_i^2}{2\sigma^2}}\right) \\ &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \frac{\xi_i^2}{2\sigma^2} \ . \end{aligned} \tag{2.3.32}$$

As in the Laplace case, $-\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)$ is again independent of ξ_i so it can be ignored to work with the following simpler expression

$$c(\xi_i) = \frac{\xi_i^2}{2\sigma^2} = \frac{(f(x_i) - y_i)^2}{2\sigma^2} \ . \tag{2.3.33}$$

Following the approach proposed in [33] for the Gaussian distribution assumption case, we use a slightly different formulation of (2.3.1), changing the slack variables, ξ_i, ξ_i^* to

$$\xi_i = y_i - f(x_i) \ . \tag{2.3.34}$$

This formulation allows for negative slack values, so it is not necessary to add a second set of variables ξ_i^*, in contrast to previous SVR formulations given. Thus, the problem in (2.3.1) is reformulated as

$$\begin{aligned} \min_{\beta,\beta_0,\xi_i} \quad & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\frac{\xi_i^2}{2\sigma^2} \\ \text{subject to} \quad & y_i - f(x_i) = \xi_i, \ i = 1, ...N \ . \end{aligned} \tag{2.3.35}$$

We have that $c(\xi) = 0 \Rightarrow \xi = 0$. Using this result and the conditions detailed in Section 2.3.1, in particular equation (2.3.2), we get

$$c(\xi) = 0 \ , \ \forall \ \xi \in [-\epsilon^*, \epsilon] \Rightarrow \epsilon^* = \epsilon = 0 \ , \tag{2.3.36}$$

Now, inserting (2.3.36) into (2.3.35) we arrive to the following formulation of the primal problem for Gaussian noise

$$\begin{aligned} \min_{\beta,\beta_0,\xi_i} \quad & \frac{1}{2}||\beta||^2 + \frac{C}{2\sigma^2}\sum_{i=1}^{N}\xi_i^2 \\ \text{subject to} \quad & y_i - f(x_i) = \xi_i, \ i = 1, ...N \ . \end{aligned} \tag{2.3.37}$$

It can be seen that this formulation is analogous to the proposed Least Squares SVR, LS-SVR, by Suykens [34], which is equivalent to kernel ridge regression.

Using now $f(x_i) =< h(x_i), \beta > +\beta_0$, the Lagrangian for the primal problem corresponding to (2.3.37) is

$$L = \frac{1}{2}||\beta||^2 + \frac{C}{2\sigma^2}\sum_{i=1}^{N}\xi_i^2 - \sum_{i=1}^{N}\alpha_i(< h(x_i), \beta > +\beta_0 - y_i + \xi_i) \ . \tag{2.3.38}$$

Computing the derivatives in (2.3.38) we obtain

$$\begin{aligned}
\frac{\partial L}{\partial \beta_0} &= -\sum_{i=1}^{N} \alpha_i \ , \\
\frac{\partial L}{\partial \beta} &= \beta - \sum_{i=1}^{N} \alpha_i h(x_i) \ , \\
\frac{\partial L}{\partial \xi_i} &= \frac{C}{\sigma^2}\xi_i - \alpha_i, \ i = 1, ...N \ ,
\end{aligned} \tag{2.3.39}$$

and setting the derivatives (2.3.39) to zero we get

$$\begin{aligned}
&\sum_{i=1}^{N} \alpha_i = 0 \ , \\
&\beta = \sum_{i=1}^{N} \alpha_i h(x_i) \ , \\
&\alpha_i = \frac{C}{\sigma^2}\xi_i, \ i = 1, ...N \ .
\end{aligned} \tag{2.3.40}$$

Plugging (2.3.40) into (2.3.38) we get the dual problem for the Gaussian case, which as can be seen is analogous to the one used in kernel ridge regression [35] and has the following formulation

$$\begin{aligned}
\max_{\alpha_i} \quad & D = \sum_{i=1}^{N} y_i \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \left(k(x_i, x_j) + \frac{\delta_{ij}\sigma^2}{C} \right) \\
\text{subject to} \quad & \alpha_i \geq 0, \ i = 1, ...N \\
& \sum_{i=1}^{N} \alpha_i = 0 \ ,
\end{aligned} \tag{2.3.41}$$

with the KKT conditions at the optimal point

$$\begin{aligned}
\widehat{\beta} &= \sum_{i=1}^{N} \widehat{\alpha}_i h(x_i) \ , \\
\widehat{\alpha}_i &= \frac{C}{\sigma^2}\widehat{\xi}_i, \ i = 1, ...N \ , \\
\widehat{\alpha}_i &(\widehat{f}(x_i) - y_i + \widehat{\xi}_i) = 0 \ .
\end{aligned} \tag{2.3.42}$$

Following an analogous procedure, we can get the dual problem formulation for the **non-zero-mean Gaussian** loss function choice. The only thing left in order to do this is to get the optimal loss function for that case. The error distribution now is

$$p(\xi_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\xi_i - m)^2}{2\sigma^2}} \ , \tag{2.3.43}$$

where $\sigma^2 > 0$ and $m \in (-\infty, \infty)$. Replacing (2.3.43) into (2.3.14) we obtain

$$\begin{aligned} c(\xi_i) &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\xi_i - m)^2}{2\sigma^2}}\right) = \\ &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \log\left(e^{-\frac{(\xi_i - m)^2}{2\sigma^2}}\right) = \\ &= -\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \frac{(\xi_i - m)^2}{2\sigma^2} \ . \end{aligned} \tag{2.3.44}$$

The term $-\log(\frac{1}{\sqrt{2\pi}\sigma})$ is independent of ξ_i and thus can be ignored to work with the following expression

$$c(\xi_i) = \frac{(\xi_i - m)^2}{2\sigma^2} \ . \tag{2.3.45}$$

Replacing (2.3.33) for (2.3.45) in our previous steps, the dual problem for non-zero mean Gaussian noise can be obtained. First, we have that $c(\xi) = 0 \ , \ \forall \, \xi \in [-\epsilon^*, \epsilon] \Rightarrow \epsilon^* = \epsilon = m$. Now, inserting this result into (2.3.1) and reformulating as previously did for the zero-mean case, we arrive to the following formulation of the primal problem for non-zero mean Gaussian noise

$$\begin{aligned} \min_{\beta, \beta_0, \xi_i} \quad & \frac{1}{2}||\beta||^2 + \frac{C}{2\sigma^2}\sum_{i=1}^{N}(\xi_i - m)^2 \\ \text{subject to} \quad & y_i - f(x_i) = \xi_i, \ i = 1, ...N \ . \end{aligned} \tag{2.3.46}$$

Using now $f(x_i) =< h(x_i), \beta > +\beta_0$, the Lagrangian for the primal problem corresponding to (2.3.46) is

$$L = \frac{1}{2}||\beta||^2 + \frac{C}{2\sigma^2}\sum_{i=1}^{N}(\xi_i - m)^2 - \sum_{i=1}^{N}\alpha_i(< h(x_i), \beta > +\beta_0 - y_i + \xi_i) \ . \tag{2.3.47}$$

Next, computing the derivatives in (2.3.47) we get the same results as in (2.3.39), except for the term corresponding to $\frac{\partial L}{\partial \xi_i}$, which is now

$$\frac{\partial L}{\partial \xi_i} = \frac{C}{\sigma^2}(\xi_i - m) - \alpha_i, \ i = 1, ...N \ . \tag{2.3.48}$$

Setting the derivatives to zero and plugging them into (2.3.47) we get the dual problem for the non-zero mean Gaussian case. It can be seen that this dual problem has the same formulation as the zero-mean case shown in (2.3.41) but replacing the KKT condition $\widehat{\alpha}_i = \frac{C}{\sigma^2}\widehat{\xi}_i$ for $\widehat{\alpha}_i = \frac{C}{\sigma^2}(\widehat{\xi}_i - m)$.

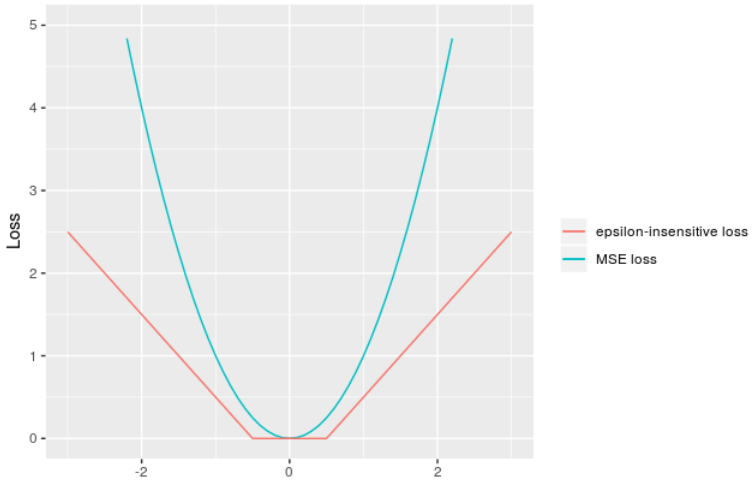


Figure 2.3.1: SILF vs ϵ-ILF. SILF presents a smoother shape.

2.3.3.3 SILF

As stated before, a loss function called SILF has been proposed as an alternative to the standard ϵ-ILF [32]. We will not be making use of this loss function for the purpose of this work, but for the sake of completeness we include here the correspoding general noise SVR formulation adapted to this loss function. The formulation of SILF is the following one

$$c(x_i, y_i, f(x_i)) = \begin{cases} -(f(x_i) - y_i) - \epsilon, & (f(x_i) - y_i) \in \Delta_{C^*} \\ \frac{((f(x_i) - y_i) + (1-\rho)\epsilon)^2}{4\rho\epsilon}, & (f(x_i) - y_i) \in \Delta_{M^*} \\ 0, & (f(x_i) - y_i) \in \Delta_0 \\ \frac{((f(x_i) - y_i) - (1-\rho)\epsilon)^2}{4\rho\epsilon}, & (f(x_i) - y_i) \in \Delta_M \\ (f(x_i) - y_i) - \epsilon, & (f(x_i) - y_i) \in \Delta_C \ , \end{cases} \tag{2.3.49}$$

where

$$\begin{array}{l} 0 < \rho \leq 1 \ , \\ \epsilon > 0 \ , \\ \Delta_{C^*} = (-\infty, -(1+\rho)\epsilon) \ , \\ \Delta_{M^*} = [-(1+\rho)\epsilon, -(1-\rho)\epsilon] \quad , \quad \Delta_0 = (-(1-\rho)\epsilon, (1-\rho)\epsilon) \ , \\ \Delta_M = [(1-\rho)\epsilon, (1+\rho)\epsilon] \ , \\ \Delta_C = ((1+\rho)\epsilon, +\infty) \ . \end{array} \tag{2.3.50}$$

Figure imsvmsilf shows a visual comparison of SILF and ϵ-insensitive loss functions. The purpose of using SILF as loss function in SVR models is to combine in a single loss function two properties:

- The **sparseness** of the ϵ-insensitive loss function, which means that training samples with small noise that fall in the flat zero region are not involved in the representation of regression functions and therefore the computational cost is reduced.

- The **smoothness** similar to that of the quadratic and Huber's loss functions, that can grant favorable mathematical properties.

Using the conditions detailed in Section 2.3.1 we get

$$c(\xi) = 0 \ , \ \forall \ \xi \in [-\epsilon^*, \epsilon] \Rightarrow \epsilon^* = \epsilon = (1-\rho)\epsilon \ , \tag{2.3.51}$$

and by (2.3.51) and the conditions in Section 2.3.1 we get

$$c(\xi_i) = \begin{cases} \frac{\xi_i^2}{4\rho\epsilon} \ , \ \xi_i \in [0, 2\rho\epsilon) \ , \\ \xi_i - \rho\epsilon \ , \ \xi_i \in [2\rho\epsilon, \infty) \ , \end{cases} \tag{2.3.52}$$

$$c(\xi_i^*) = \begin{cases} \frac{\xi_i^{*2}}{4\rho\epsilon} \ , \ \xi_i^* \in [0, 2\rho\epsilon) \ , \\ \xi_i^* - \rho\epsilon \ , \ \xi_i^* \in [2\rho\epsilon, \infty) \ . \end{cases} \tag{2.3.53}$$

Thus, inserting (2.3.51), (2.3.52), and (2.3.53) into (2.3.1) we arrive to the following formulation of the general SILF SVR problem

$$\begin{aligned} \min_{\beta, \beta_0, \xi_i, \xi_i^*} \quad & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}(c(\xi_i) + c(\xi_i^*)) \\ \text{subject to} \quad & \xi_i, \xi_i^* \geq 0, \ i = 1, ...N \\ & f(x_i) - y_i \leq (1-\rho)\epsilon + \xi_i, \ i = 1, ...N \\ & y_i - f(x_i) \leq (1-\rho)\epsilon + \xi_i^*, \ i = 1, ...N \ . \end{aligned} \tag{2.3.54}$$

The Lagrange function corresponding to the previous primal problem (2.3.54) is

$$\begin{aligned} L = & \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}(c(\xi_i) + c(\xi_i^*)) - \sum_{i=1}^{N}\alpha_i(y_i - f(x_i) + (1-\rho)\epsilon + \xi_i) - \\ & - \sum_{i=1}^{N}\alpha_i^*(f(x_i) - y_i + (1-\rho)\epsilon + \xi_i^*) - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\mu_i^*\xi_i^* \ . \end{aligned} \tag{2.3.55}$$

Writting $f(x)$ in the form of basis expansions, i.e., $f(x_i) =< h(x_i), \beta > +\beta_0$ and computing the derivatives in (2.3.55) we obtain

$$\begin{aligned} & \frac{\partial L}{\partial \beta_0} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \ , \\ & \frac{\partial L}{\partial \beta} = \beta + \sum_{i=1}^{N}\alpha_i h(x_i) - \sum_{i=1}^{N}\alpha_i^* h(x_i) = \beta + \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)h(x_i) \ , \\ & \frac{\partial L}{\partial \xi_i} = C\frac{\partial c}{\partial \xi}(\xi) - \alpha_i - \mu_i, \ i = 1, ...N \ , \\ & \frac{\partial L}{\partial \xi_i^*} = C\frac{\partial c}{\partial \xi^*}(\xi^*) - \alpha_i^* - \mu_i^*, \ i = 1, ...N \ , \end{aligned} \tag{2.3.56}$$

and setting all these derivatives to zero we get

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = \ 0 \ , \tag{2.3.57}$$

$$\beta = \ \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)h(x_i) \ , \tag{2.3.58}$$

$$\alpha_i = \ C\frac{\partial c}{\partial \xi}(\xi) - \mu_i, \ i = 1, ...N \ , \tag{2.3.59}$$

$$\alpha_i^* = \ C\frac{\partial c}{\partial \xi^*}(\xi^*) - \mu_i^*, \ i = 1, ...N \ . \tag{2.3.60}$$

Plugging (2.3.57), (2.3.58), (2.3.59) and (2.3.60) into (2.3.55) we get the dual problem for the SILF

$$\begin{array}{ll}
\max\limits_{\alpha_i, \alpha_i^*} & D = \sum\limits_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j) - \\
& (1-\rho)\epsilon\sum\limits_{i=1}^{N}(\alpha_i + \alpha_i^*) + C\sum\limits_{i=1}^{N}[c(\xi_i) + c(\xi_i^*) - \xi_i\frac{\partial c}{\partial \xi}(\xi) - \xi_i^*\frac{\partial c}{\partial \xi^*}(\xi^*)] \\
\text{subject to} & \xi_i, \xi_i^* \geq 0 \ , \\
& \alpha_i, \alpha_i^* \geq 0, \ i = 1, ...N \ , \\
& \alpha_{\leq} C\frac{\partial c(\xi)}{\partial \xi}(\xi), \ i = 1, ...N \ , \\
& \alpha_i^* \leq C\frac{\partial c}{\partial \xi^*}(\xi^*), \ i = 1, ...N \ , \\
& \sum\limits_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \ ,
\end{array} \tag{2.3.61}$$

with the following KKT conditions at the optimal solution

$$\begin{array}{l}
\beta = \sum\limits_{i=1}^{N}(\alpha_i^* - \alpha_i)h(x_i) \ , \\
\alpha_i(y_i - < h(x_i), \beta > + \beta_0 + (1-\rho)\epsilon + \xi_i) = 0 \ , \\
\alpha_i^*(< h(x_i), \beta > + \beta_0 - y_i + (1-\rho)\epsilon + \xi_i^*) = 0 \ , \\
(C\frac{\partial c}{\partial \xi}(\xi) - \alpha_i)\xi_i = 0 \ , \\
(C\frac{\partial c}{\partial \xi^*}(\xi^*) - \alpha_i^*)\xi_i^* = 0 \ .
\end{array} \tag{2.3.62}$$

As shown in [32], terms involving ξ_i and ξ_i^* can be simplified through easy steps to arrive at the following expression

$$\begin{aligned}\max_{\alpha_i,\alpha_i^*} \quad D = & \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(x_i, x_j) \\ & - (1-\rho)\epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) \\ & - \frac{\rho\epsilon}{C}\sum_{i=1}^{N}(\alpha_i^2 + \alpha_i^{*2}) \ ,\end{aligned} \tag{2.3.63}$$

where the constraints are the same as in (2.3.61) with the exception of the Lagrangian coefficients α_i and α_i^* now being upper bounded by the hyperparameter C, i.e.,

$$\alpha_i, \alpha_i^* \leq C \ \ i = 1, ...N \ . \tag{2.3.64}$$

Although formulations for the SILF loss function have been described here, this cost function will not be used in the experiments carried out as part of this thesis, due to the extra hyperparameter ρ implying difficulties for the implementation of this cost function in our proposed framework, which is described in Section 3.

Table 2.3.1 shows a summary of all the loss functions we have discussed up to this point. Mathematical computations required to obtain all these formulations can be found in the previous sections of this work. Parameter m for the non-zero mean Laplace and Gaussian distributions will be denoted μ from now on, as there is no risk of confusion with the Lagrange parameter μ_i in the following chapters.

2.4 Constant Width Error Intervals for SVR

SVR models have been widely used in a variety of regression problems with excellent results, being considered one of the state-of-the-art choices in different areas. However, the previously described classical ϵ-SVR only gives a prediction, with no possibility of obtaining probability intervals to address the uncertainty in these predictions. It is important to notice that, for these type of models, approaches such as the well known ones for linear regression under Gaussian models are not feasible. First, because the familiar analytic estimates of the linear coefficients are impossible in SVR due to its formulation, and less so any asymptotic analysis. Besides, it should also be worth noting the difficulty of ensuring the assumption of normal random variables in most scenarios.

To deal with these issues, C.J. Lin proposed a direct approach to build error intervals for SVR [18] [27]. This method assumes prediction errors to follow a specific probability distribution and uses this density function hypothesis to define probability intervals for the model errors. The main idea behind this proposed method is that if the distribution assumption is true and hence the underlying noise distribution is accurately estimated, the resulting computed error intervals will adapt well to the real prediction errors of the regression model. We proceed in the next section with the technical definition of this method.

Table 2.3.1: Loss functions corresponding to several error distributions.

Error Distribution	Loss Function
ϵ-ILF	$c(\xi_i) = \begin{cases} -\xi_i - \epsilon, & \xi_i < -\epsilon \\ 0, & \xi_i \in [-\epsilon, \epsilon] \\ \xi_i - \epsilon, & \xi_i > \epsilon \ . \end{cases}$
SILF	$c(\xi_i) = \begin{cases} -\xi_i - \epsilon, & \xi_i \in \Delta_{C^*} \\ \frac{(\xi_i + (1-\beta)\epsilon)^2}{4\beta\epsilon}, & \xi_i \in \Delta_{M^*} \\ 0, & \xi_i \in \Delta_0 \\ \frac{(\xi_i - (1-\beta)\epsilon)^2}{4\beta\epsilon}, & \xi_i \in \Delta_M \\ \xi_i - \epsilon, & \xi_i \in \Delta_C \ , \end{cases}$
Zero-mean Laplace	$c(\xi_i) = \frac{\lvert\xi_i\rvert}{\sigma}$
Laplace	$c(\xi_i) = \frac{\lvert\xi_i - \mu\rvert}{\sigma}$
Zero-mean Gaussian	$c(\xi_i) = \frac{\xi_i^2}{2\sigma^2}$
Gaussian	$c(\xi_i) = \frac{(\xi_i - \mu)^2}{2\sigma^2}$

2.4.1 Method

In the method proposed in [27] zero mean Gaussian and Laplace families are considered. The idea is to model the distribution of errors, Ψ, assuming one of these distribution, based on a set of out-of-sample residuals $\{\psi_i\}_{i=1}^N$. The error distribution assumed is fitted by maximum likelihood estimation, MLE, [36] using the previously computed out-of-sample residuals of an SVR model used to predict a regression target.

The residuals are the result of conducting a k-fold cross-validation over the training data to get the estimated functions $\widehat{f}^j, j = 1, ..., k$, and then setting

$$\psi_i^j \equiv \widehat{f}^j(x_i) - y_i \ , \tag{2.4.1}$$

for (x_i, y_i) in fold j of the training data. Although this cross-validation scheme is proposed in [27], the same idea could be applied but using a fixed validation set to obtain the residuals.

Assuming that the ψ_i are independent, we can estimate the distributions parameters θ by maximizing the likelihood L. If ψ_i are independent we have

$$L(\theta; \psi_1, ..., \psi_n) = \prod_{i=1}^{N} p(\psi_i|\theta) \ , \tag{2.4.2}$$

where p represents the density function of the distribution of ψ_i.

Now, denoting l the logarithm of the likelihood we get

$$l(\theta; \psi_1, ..., \psi_n) = \sum_{i=1}^{n} \log p(\psi_i|\theta) \ . \tag{2.4.3}$$

Maximizing l is equivalent to maximize L so both formulations can be applied, although using l leads to a simpler problem.

A difficulty with the method proposed in [27] is that it assumes the residual distribution to be independent of x and, therefore, probability intervals have exactly the same width for all input instances. In theory, the density distribution may depend on the input x, and therefore the length of the predictive interval with a pre-specified coverage probability may vary from one example to another, reflecting the fact that the prediction variances vary with different input values. In fact, it is easy to think of real-world problems where this behaviour has a strong impact.

Nevertheless, the authors who proposed this method claim that despite the fact that their error interval is not influenced by x, and hence it does not reflect this property, it can be justified if we consider the probability to be taken over all possible input values. In [27] it is proposed to model ψ_i by zero-mean Gaussian and Laplace distributions because residuals of data studied in previous work seem to be symmetric about zero and both Gaussian and Laplace captured their shape reasonably well. We describe how to model ψ_i using these distributions following the method proposed in [27].

2.4.2 Parameters and Error Intervals for Different Distributions

2.4.2.1 Zero mean Laplace

Assuming a zero mean Laplace distribution we get the following equation for l

$$\begin{aligned} l(\theta; \psi_1, ..., \psi_n) = \sum_{i=1}^{n} \log \frac{1}{2\sigma} e^{-\frac{|\psi_i|}{\sigma}} = \sum_{i=1}^{n} \log \frac{1}{2\sigma} - \sum_{i=1}^{n} \frac{|\psi_i|}{\sigma} = \\ = -n \log 2 - n \log \sigma - \frac{1}{\sigma} \sum_{i=1}^{n} |\psi_i| \ . \end{aligned} \tag{2.4.4}$$

We can compute now the corresponding derivative

$$\frac{\partial l}{\partial \sigma} = -n \frac{1}{\sigma} + \frac{1}{\sigma^2} \sum_{i=1}^{n} |\psi_i| \ . \tag{2.4.5}$$

In the maximum point, where we denote by $\widehat{\sigma}$ the corresponding density parameter value, the first derivative must be equal to zero, so

$$-n \frac{1}{\widehat{\sigma}} + \frac{1}{\widehat{\sigma}^2} \sum_{i=1}^{n} |\psi_i| = 0 \ . \tag{2.4.6}$$

Solving (2.4.6) we obtain

$$\widehat{\sigma} = \frac{\sum_{i=1}^{n} |\psi_i|}{n} , \tag{2.4.7}$$

The equation in (2.4.7) is just the mean absolute error, MAE.

Finally, we need to compute the upper sth percentile, p_s, of the corresponding probability distribution of $\psi = \widehat{f}(x) - y$. For a zero-mean symmetric variable with density $p(z)$, we can obtain p_s just by solving

$$1 - s = \int_{-\infty}^{p_s} p(z)dz . \tag{2.4.8}$$

The prediction error interval is then $(-p_s, p_s)$. For the zero mean Laplace distribution, we can replace $p(z)$ in 2.4.8 for

$$p(z) = \frac{1}{2\sigma} e^{-\frac{|z|}{\sigma}}$$

and we get

$$\begin{aligned} 1 - s = \int_{-\infty}^{p_s} p(z)dz &= \\ \int_{-\infty}^{p_s} \frac{1}{2\sigma} e^{-\frac{|z|}{\sigma}} dz &= \\ \left[\frac{z(1 - e^{-\frac{|z|}{\sigma}})}{2|z|} \right]_{-\infty}^{p_s} &= \\ \frac{1 - e^{-\frac{p_s}{\sigma}}}{2} + \frac{1}{2} &= \\ 1 - \frac{e^{-\frac{p_s}{\sigma}}}{2} \qquad \Rightarrow \quad p_s = -\sigma \log 2s . \end{aligned} \tag{2.4.9}$$

Therefore, as described in [27], the error interval for the case of the zero mean Laplace distribution is

$$(\sigma \log 2s, -\sigma \log 2s) . \tag{2.4.10}$$

2.4.2.2 Zero mean Gaussian

Assuming a zero mean Gaussian distribution we get the following equation for l

$$\begin{aligned} l(\theta; \psi_1 ... \psi_n) &= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\psi_i^2}{2\sigma^2}} = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{n} \frac{\psi_i^2}{2\sigma^2} \\ &= -n \log \sqrt{2\pi} - n \log \sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \psi_i^2 . \end{aligned} \tag{2.4.11}$$

In the maximum point, where we denote $\widehat{\sigma}$ the corresponding density parameter value, the first derivative must be equal to zero, so

$$\frac{\partial l}{\partial \widehat{\sigma}} = -n\frac{1}{\widehat{\sigma}} + \frac{1}{\widehat{\sigma}^3}\sum_{i=1}^{n}\psi_i^2 = 0 \ . \tag{2.4.12}$$

Solving (2.4.12) we obtain

$$\widehat{\sigma}^2 = \frac{\sum_{i=1}^{n}\psi_i^2}{n} \ . \tag{2.4.13}$$

The equation in (2.4.13) is just the mean squared error, MSE.

The formulation for the prediction error interval for the zero mean Gaussian distribution assumption is again the same as its previously defined Laplace counterpart, i.e. $(-p_s, p_s)$, with p_s as defined in (2.4.8). For the zero mean Gaussian distribution, we can replace $p(z)$ in 2.4.8 for

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{z^2}{2\sigma^2}}$$

and it can be seen [27] that this leads to

$$p_s = \frac{2}{\sigma[erf(\frac{1-s}{\sqrt{2}})+1]} \ , \tag{2.4.14}$$

where erf is the Gauss error function defined as

$$erf(z) = \frac{2}{\pi}\int_0^z e^{-t^2}dt \ . \tag{2.4.15}$$

Therefore, the error interval for the case of the zero mean Gaussian distribution is

$$\left(-\frac{2}{\sigma[erf(\frac{1-s}{\sqrt{2}})+1]}, \frac{2}{\sigma[erf(\frac{1-s}{\sqrt{2}})+1]}\right) . \tag{2.4.16}$$

2.5 NORMA Optimization

Support Vector Machines are a type of ML models belonging to what is usually called kernel-based algorithms. This type of methods has achieved considerable success in various problems when working in a batch setting, i.e. where all of the training data is available in advance and you train your model in a single step in which you provide all available data as input to your model, even if it is split into train/validation/test subsets. However, there has been little use of this family of methods in an online setting, in which you train your model in a multistep process where at each iteration you provide one sample, or a subset of samples, as input to your model, which automatically is updated correspondingly. Besides, the cost order of the algorithm usually employed in batch training of SVM models, SMO, does not go below quadratic, which does not mix well with the large volumes of data available in the big data era.

Optimization methods suitable to be used to train SVM models in an online setting have been proposed, like Pegasos [37] or Naive Online Reg Minimization Algorithm, NORMA [38]. We will focus here in the latter. This optimization algorithm is developed by considering classical stochastic gradient descent within a feature space and the use of some straightforward tricks, and it is shown to be a computationally efficient algorithm for a wide range of problems such as classification, regression, and novelty detection.

The goal of this method is, given a particular supervised learning problem to solve, to obtain an optimal predictive function, f, following an online setting. In particular, this method employs an iterative process, where at the end of a particular iteration $t+1$, we will obtain an approximation f_{t+1}, dependant on the previous iteration result f_t, to the optimal predictive function. NORMA uses as update rule the following one

$$f_{t+1} = \quad f_t - \eta_t \partial_{f_t} R_{inst,\lambda}[f_t, x_t, y_t] \ , \tag{2.5.1}$$

where $\eta_t > 0$ is the learning rate, which usually is chosen to be constant, i.e., $\eta_t = \eta$, and $\partial_{f_t} R_{inst,\lambda}[f_t, x_t, y_t]$ is the gradient of

$$R_{inst,\lambda}[f_t, x_t, y_t] := \quad l(f_t(x_t), y_t) + \frac{\lambda}{2}||f_t||^2 \ , \tag{2.5.2}$$

with respect to the predictive function f, where $\lambda \geq 0$ is the regularization parameter and l is a given loss function. In particular, $l(f_t(x_t), y_t)$ is the loss the learning algorithm makes when it tries to predict the target y_t based on the features x_t and the current estimate f_t based on the previous samples $\{x_i, y_i\}_{i=1}^{t-1}$.

Therefore, NORMA performs gradient descent with respect to what the authors called the *instantaneous regularized risk*, $R_{inst,\lambda}$. Due to the definition in (2.5.2), the derivative $\partial_{f_t} R_{inst,\lambda}[f_t, x_t, y_t]$ can be divided into two factors:

$$\partial_{f_t} l(f_t(x_t), y_t) \ , \tag{2.5.3}$$

$$\partial_{f_t} \left(\frac{\lambda}{2}||f_t||^2 \right) \Big(. \tag{2.5.4}$$

Regarding (2.5.3), as stated in [38], when working in a *reproducing kernel Hilbert space*, $\mathbb{H}$, [28] and due to the reproducing property, it is satisfied that

$$< f, k(x, \cdot\,) >_{\mathbb{H}} = \quad f(x), \forall x \in \mathbb{R} \ , \tag{2.5.5}$$

where k is a particular kernel function $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, the following equality holds

$$\partial_{f_t} l(f_t(x_t), y_t) = \quad l'(f_t(x_t), y_t) k(x_t, \cdot\,) \ . \tag{2.5.6}$$

As for (2.5.4), it holds that

$$\partial_{f_t} \left(\frac{\lambda}{2}||f_t||^2 \right) \Big(= \quad \frac{\lambda}{2} 2 f_t = \lambda f_t \ . \tag{2.5.7}$$

Therefore, combining (2.5.6) and (2.5.7) we get

$$\partial_{f_t} R_{inst,\lambda}[f_t, x_t, y_t] = \quad l'(f_t(x_t), y_t) k(x_t, \cdot\,) + \lambda f_t \ . \tag{2.5.8}$$

Now, plugging (2.5.8) into (2.5.1) we obtain

$$\begin{aligned} f_{t+1}(x) = &\quad f_t(x) - \eta_t(l'(f_t(x_t), y_t)k(x_t, x) + \lambda f_t(x)) \\ = &\quad (1 - \eta_t\lambda)f_t(x) - \eta_t l'(f_t(x_t), y_t)k(x_t, x) \ , \end{aligned} \tag{2.5.9}$$

where the condition $\eta_t < \frac{1}{\lambda}$ must hold for the algorithm to work properly.

Writing f_t in (2.5.9) in the form of kernel expansions, as proposed in [39], we have

$$f_{t+1}(x) = \sum_{i=1}^{t} \alpha_{t+1}^i k(x_i, x) \ . \tag{2.5.10}$$

where the coefficients α_{t+1}^i are updated at iteration $t+1$ based on the values α_t^i in the previous iteration via

$$\alpha_{t+1}^i = (1 - \eta_t\lambda)\alpha_t^i \text{ for } i < t \ , \tag{2.5.11}$$

and a new coefficient α_{t+1}^t is added, with the following value

$$\alpha_{t+1}^t = -\eta_t l'(f_t(x_t), y_t) \ . \tag{2.5.12}$$

Plugging (2.5.11) and (2.5.12) into (2.5.10) we obtain the following equivalent formulation for the updating step in (2.5.9)

$$\widehat{f}_{t+1}(x) = (1 - \eta_t\lambda) \sum_{i=1}^{t-1} \alpha_t^i k(x_i, x) - \eta_t l'(\hat{f}_t(x_t), y_t)k(x_t, x) \ , \tag{2.5.13}$$

The combination of (2.5.10), (2.5.11), and (2.5.12) constitute the update rules for the NORMA optimization at iteration $t+1$ used in practice. f_0 is called the initial hypothesis and commonly takes the value $f_0 = 0$. As shown in [38] an extra update rule to include the possibility of the existence of an offset b term for the function f, i.e. $f_t^b(x_t) = f_t(x_t) + b_t$, can be added.

There are several ways of speeding up the algorithm implementation. For instance, as proposed in [38], instead of updating all old coefficients α^i for $i < t$ for each iteration, one may simply choose η_t to be constant, i.e $\eta_t = \eta$, and cache the power series

$$\begin{aligned} \alpha_2^1 &= (1 - \eta\lambda)\alpha_1^1 \\ \alpha_3^1 &= (1 - \eta\lambda)\alpha_2^1 = (1 - \eta\lambda)^2\alpha_1^1 \\ &\cdots \\ \alpha_t^1 &= (1 - \eta\lambda)^{n-1}\alpha_1^1 \end{aligned} \tag{2.5.14}$$

and then pick the suitable terms as needed. Stopping criteria for these updating iterative process are given in Section 3.1.3.

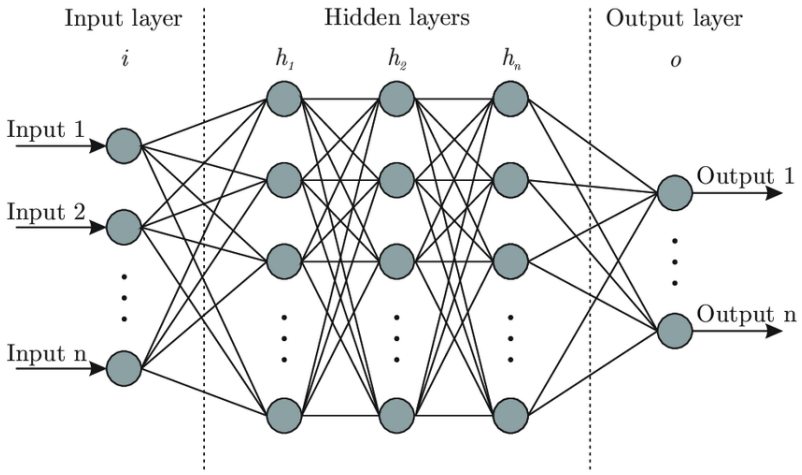


Figure 2.6.1: Artificial Neural Network following a Fully Connected schema, where each neuron from one layer is connected to all neurons in the following layer. Taken from [41].

2.6 Deep Learning

The concept of Deep Learning, DL, has had different interpretations in recent years. Often, DL is employed simply to refer to a specific subset of Artificial Neural Networks or ANNs [40], another family of Machine Learning models that can be used both for classification and regression. In particular, it is used to name ANNs with a large number of what are called *hidden layers*. An ANN model is made up of a collection of connected units called neurons, where the output of each neuron is computed by some non-linear function, called *activation function*, of the weighted sum of its inputs. Neuron connections have weights, so activations of different neurons can have bigger impact than others. Neurons of one layer can connect to neurons of the preceding and following layers. The layer that receives external data is the input layer and the last layer, the one that produces the ultimate result, is the output layer. In between them are zero or more hidden layers. When the number of these hidden layers is large, we talk about Deep Artifical Neural Networks, often referred simply as DL models.One example of this type of models is shown in Figure 2.6.1.

However, the DL denomination has also been used to refer to any type of Machine Learning model framework which consists of a training schema containing several optimization layers, each one affecting the result of the preceding and following layers. An example of this are Deep Belief Networks or DBNs [42], a type of ML models used for unsupervised learning and based on multiple layers, but with significant differences to the standard schema of an ANN.

Nevertheless, it is true that clearly the link between DL and Deep ANNs is strong and almost ever-present nowadays. Several factors have probably had an impact on this, including the fact that the ANNs schema adapts almost perfectly to the concept of DL framework and that some of the first groundbreaking advances in DL correspond to deep ANNs.

We will focus here on the fully connected version of Deep ANNs as we consider it adapts better to our purposes than other frameworks like Convolutional Neural Networks, popu-

lar in image recognition problems, or Recurrent Neural Networks, often used in Natural Language Processing tasks.

2.6.1 DL Special Properties

One of the deciding factors for the recent prevalence of DL in the Machine Learning world is that this family of models present some extremely relevant properties. We will focus here on two of them: Complexity and End-to-End learning.

Complexity: When trained with large enough datasets, DL models commonly achieve better results than other families of ML models. The reason for this behaviour is the special nature of DL frameworks, consisting of several layers, where usually at each level the complexity of relationships and patterns detected by the network increases. This behaviour has been specially studied for image recognition tasks, where it can be shown that the first layers learn to recognize basic patterns, such as lines, squares, etc., while deeper layers get trained to find much more complex relationships, for instance, the presence of a person or a specific animal in the picture. This property of DL frameworks allows for the construction of models with bigger complexity potential than their classic ML counterparts, and thus to a higher predictive potential.

End-to-End Learning: One of the most exciting recent developments in DL has been the rise of end-to-end learning. Traditionally, there existed Machine Learning systems that required multiple stages of pre-processing and model training. What end-to-end DL allows is to take all those multiple stages, remove the pre-processing stage and replace this schema with just a single step, the training and computation of the DL models.

One illustrative example can be found in speech recognition, where your goal is to take an input such an audio clip, and map it to an output which is a transcript of the audio clip. Traditionally, speech recognition required more than one stage of processing. First, you had to extract some features of the audio using pre-processing methods like the mel-frequency cepstral coefficients or MFCC [43]. Then, having extracted some low level features, one could apply a machine learning algorithm to find, for instance, the phonemes, the basic units of sound, in the audio clip.

When using DL frameworks, this pipeline with multiple stages can be replaced by the training of a deep neural network, allowing to just input the audio clip and obtain directly the transcript as an output. However, it is important to remark that one of the challenges of end-to-end DL is that usually large volumes of data are needed before it works in a comparable way to classical multistep ML frameworks, and even larger to be able to surpass the performance of its counterparts.

2.6.2 Backpropagation

Given a training sample and a target to predict, an ANN will compute all the activation functions, described more in detail later in the section, from the input layer to the output layer, obtaining a final prediction as a result. We call this a *forward pass*.

Once this forward pass has been performed, we can calculate an error between its output and the real target using the selected error function. Using gradient descent theory [44] over this error, it could be possible to obtain new weight values for the output layer units to

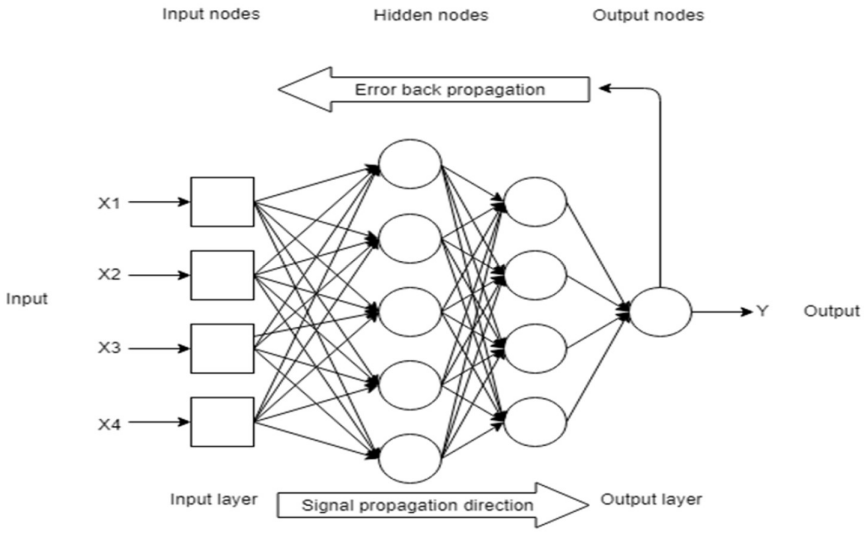


Figure 2.6.2: Backpropagation schema. The goal is to extend gradient descent to all the layers in the network after the forward pass. Taken from [41].

try to improve its output. Nevertheless, this would only modify the weights of the output layer and not of all the preceding layers, which also have an impact on the resulting output, and thus will have a far from optimal effect.

Therefore, we need an algorithm to propagate backwards the error from the units in the output layer to the units in the preceding layers. This algorithm is called backpropagation and is used to optimize ANNs. The backpropagation schema is illustrated in Figure 2.6.2.

The goal of backpropagation is to be able to extend gradient descent to all the layers in the network. Backpropagation defines the generalized error associated to a hidden unit as a weighted average of the errors of the units in the adjacent layer. The gradient value for a unit j in layer J, will have the following formulation

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} = \delta_j \frac{\partial s_j}{\partial w_{ji}} , \tag{2.6.1}$$

where E represents the error function, w_{ji} is the weight of the connection from unit i to unit j, $s_j = \sum_i w_{ji} z_i$ the sum of the weighted inputs of unit j in layer J, z_i the output of unit i in layer $J-1$, and $\delta_j = \frac{\partial E}{\partial s_j}$ the generalized error at unit j.

It holds that

$$\frac{\partial s_j}{\partial w_{ji}} = z_i , \tag{2.6.2}$$

and expanding the term δ_j we get

$$\begin{aligned}\delta_j = \sum_k \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial s_j} = \sum_k \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial z_j}\frac{\partial z_j}{\partial s_j} = \left(\sum_k \frac{\partial E}{\partial s_k} w_{kj}\right) F_j'(s_j) = \\ = \left(\sum_k \delta_k w_{kj}\right) F_j'(s_j) \ ,\end{aligned} \tag{2.6.3}$$

where F_j is the activation function used in unit j. Finally, plugging (2.6.2) and (2.6.3) into (2.6.1) we get the final expression

$$\frac{\partial E}{\partial w_{ji}} = \left(\sum_k \delta_k w_{kj}\right) F_j'(s_j) z_i \ . \tag{2.6.4}$$

Therefore, using backpropagation we can compute the error of the output units, then the generalized error of the units of the last hidden layer and successively all the previous hidden layers, and finally the gradient with respect to the weights. This is called the *backward pass* and allow the possibility of ANNs optimization.

2.6.3 Activation Functions

For a given neuron in layer J of a Deep ANN, the inputs are multiplied by the weights and summed together, i.e., $s_j = \sum_i w_{ji} z_i$, where z_i is the output of unit i in layer $J-1$. This value is referred to as the activation of the neuron. This summed activation is then transformed via an activation function which defines the specific output of that neuron. The simplest activation function is referred to as the linear activation.

$$F(x) = x \ . \tag{2.6.5}$$

A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. Linear activation functions are still used in the output layer for networks that predict a quantity, as is the case in regression problems.

Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. Traditionally, two were the most widely used nonlinear activation functions in ANNs or Deep ANNs:

1. **Sigmoid function:** The sigmoid activation function, also called the logistic function, is traditionally a very popular activation function for neural networks. It has the following formulation

$$F(x) = \frac{1}{1+e^{-x}} \ . \tag{2.6.6}$$

The input to the function is transformed into a value between 0 and 1. Inputs that are much larger than 1 are transformed to essentially the value 1; similarly, negative values far below 0 are snapped to 0. The shape of the function for all possible inputs is an S-shape from zero up to 1, having value 0.5 at $x = 0$, which is the middle point of the S-shape. For a long time, through the early 1990s, it was the default activation used on neural networks.

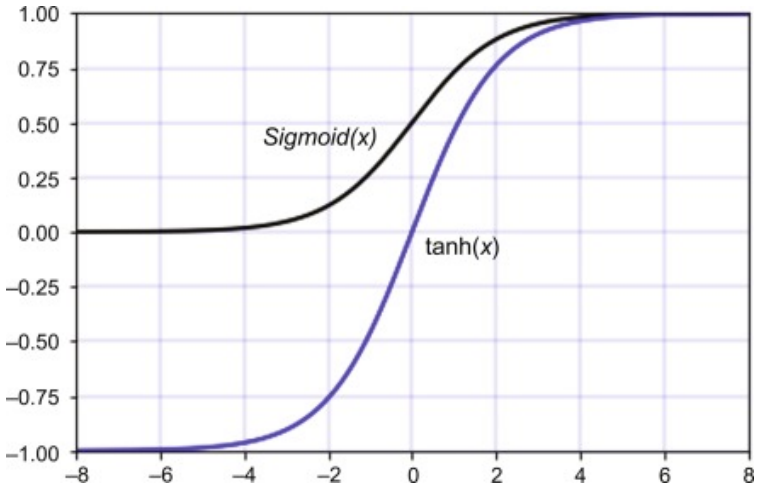


Figure 2.6.3: Sigmoid and hyperbolic tangent functions.

2. **Hyperbolic tangent function:** The hyperbolic tangent function, or tanh for short, is a similar shaped nonlinear activation function that outputs values between -1 and 1. It is represented by this formulation

$$F(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \ . \tag{2.6.7}$$

In the late 1990s and through the 2000s, the tanh function was preferred over the sigmoid activation function as models that used it were easier to train and often had better predictive performance. A visual comparison between these two activation functions can be seen in Figure 2.6.3.

2.6.4 DL Recent Advances

The basic theory corresponding to the multi layer perceptron or MLP [45], was already well established in the 80s, as well as the backpropagation algorithm for gradient computation during ANN training described in the previous section. In fact, they can be considered as the first example of modern machine learning algorithms that could be used in both regression and classification problems with minimal conceptual variations. However, some technical problems, essentially due to knowledge gaps about the training of these models plus the lack of computing power and large volumes of data at the time, led to their relative decline in the late 90s and the rise of alternative methods, particularly SVMs, for classification and regression.

Nevertheless, in recent years the popularity of DL models has increased in a spectacular manner, due to the wide availability of powerful computing facilities, advances on the theoretical underpinnings of MLPs, several improvements on their training procedures and a better understanding of the difficulties related to many layered architectures. To all these factors we can add the appearance of multiple development frameworks such as TensorFlow [46] and Keras [47], that have allowed the practitioners to experiment with different architectures, non-differentiable activations and, even, non-differentiable loss functions. Besides, training of these models has been shown to be linear time computations, i.e., $O(n)$, and

calculation of predictions is $O(1)$. Last, but not least, DL models have been shown to be able to extract more predictive power when trained with sufficiently large datasets than other ML frameworks.

In the following sections we describe some of the most important of these theoretical and technological advances that have contributed to dissipate problems related to DL frameworks and allowed their recent popularity. First, we will describe Adam Optimization in Section 2.6.4.1, a quite recent optimization method frequently used in DL models. Next, we discuss new methods to initialize the layer weights in Section 2.6.4.2, with special focus on Xavier initialization. In Section 2.6.4.3 we define the Rectified Linear Unit activation function, and describe some of the properties that make it specially suitable for DL schemas. Finally, the exponential increase in computational and memory resources available to train ML models is discussed in Section 2.6.4.4.

2.6.4.1 Adam Optimization

In [48] a novel optimization method, called Adaptive Moment Estimation or Adam, was proposed. Its goal was to combine the advantages of two other extensions of stochastic gradient descent, Adagrad [49] and RMSprop [50]. Adam is an adaptive learning rate method, which means it computes individual learning rates for different weights, using for that purpose estimations of first and second moments of the gradient to adapt the learning rate for each weight of the neural network. The $n-th$ moment, m^n, of a random variable, X, is defined as the expected value of that variable to the power of n, i.e.

$$m^n = \quad E[X^n] \ . \tag{2.6.8}$$

Adam uses the first and second moments, i.e m^1 and m^2. In order to estimate their values, Adam utilizes exponential moving averages, computed on the gradient evaluated on a current mini-batch in the following way

$$\widehat{m}_t^1 = \quad \beta_1 m_{t-1}^1 + (1-\beta_1) g_t \ , \tag{2.6.9}$$

$$\widehat{m}_t^2 = \quad \beta_2 m_{t-1}^2 + (1-\beta_2) g_t^2 \ . \tag{2.6.10}$$

where g_t is the gradient on current mini-batch, and β_1, β_2 are new introduced hyperparameters of the algorithm used to control Adam optimization.

The vectors of moving averages are initialized as

$$\widehat{m}_0^1 = 0 \qquad \widehat{m}_0^2 = 0 \tag{2.6.11}$$

Since $\widehat{m}$ is an estimate of the momentums, m, for the gradient g, we would like the following equations to hold

$$E[\widehat{m}_t^1] \approx E[g_t] \qquad E[\widehat{m}_t^2] \approx E[g_t^2] \tag{2.6.12}$$

To check if these properties are true let us expand the values of $\widehat{m}_t^1$ using (2.6.9)

$$\begin{aligned}
\widehat{m}_0^1 &= 0 \\
\widehat{m}_1^1 &= \beta_1 \widehat{m}_0^1 + (1-\beta_1) g_1 = (1-\beta_1) g_1 \\
\widehat{m}_2^1 &= \beta_1 \widehat{m}_1^1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2 \\
\widehat{m}_3^1 &= \beta_1 \widehat{m}_2^1 + (1-\beta_1) g_3 = \beta_1^2 (1-\beta_1) g_1 + \beta_1 (1-\beta_1) g_2 + (1-\beta_1) g_3 \\
&\dots \\
\widehat{m}_t^1 &= (1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i
\end{aligned} \tag{2.6.13}$$

Next, we evaluate the expected value of m using (2.6.13)

$$E[\widehat{m}_t^1] = \quad E[(1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i] \ . \tag{2.6.14}$$

Now, let us approximate g_i by g_t, assuming an approximation error ψ

$$E[\widehat{m}_t^1] = \quad E[g_t](1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} + \psi \ . \tag{2.6.15}$$

Equation (2.6.15) is equivalent to

$$E[\widehat{m}_t^1] = \quad E[g_t](1-\beta_1^t) + \psi \ . \tag{2.6.16}$$

As stated in (2.6.12) we want $E[\widehat{m}_t^1]$ to be as close as possible to $E[g_t]$. Therefore, assuming error ψ as unavoidable when $E[g_t]$ is not stationary, we can still correct the bias term $(1-\beta_1^t)$ replacing equation (2.6.9) by the following expression

$$\widehat{m}_t^1 = \quad \frac{\beta_1 m_{t-1}^1 + (1-\beta_1) g_t}{(1-\beta_1^t)} \ . \tag{2.6.17}$$

Applying the same logic for the second moment estimate in (2.6.10) we get

$$\widehat{m}_t^2 = \quad \frac{\beta_2 m_{t-1}^2 + (1-\beta_2) g_t^2}{(1-\beta_2^t)} \ . \tag{2.6.18}$$

Finally, once we have our moment estimators, $\widehat{m}_t^1$ and $\widehat{m}_t^2$, the only thing left to do is to use those moving averages to scale the learning rate individually for each parameter. The way it is done in Adam is the following one

$$w_t = \quad w_{t-1} - \eta_t \frac{\widehat{m}_t^1}{\sqrt{\widehat{m}_t^2} + \epsilon} \ , \tag{2.6.19}$$

where w_t are the model weights at iteration t, η_t is a hyperparameter called step size or learning rate which can be constant, i.e $\eta_t = \eta$, and ϵ is a dummy constant, usually a very small value, used to prevent any division by zero.

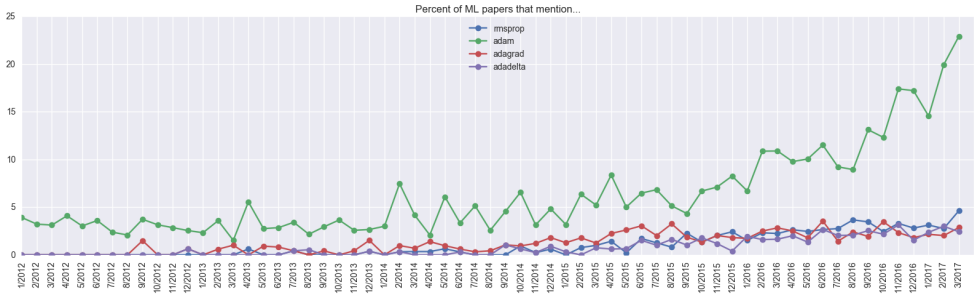


Figure 2.6.4: % of ML papers using different optimization methods. Adam is the most popular option among the standard optimization techniques for DL structures, and this difference is clearer in recent years. Taken from [48].

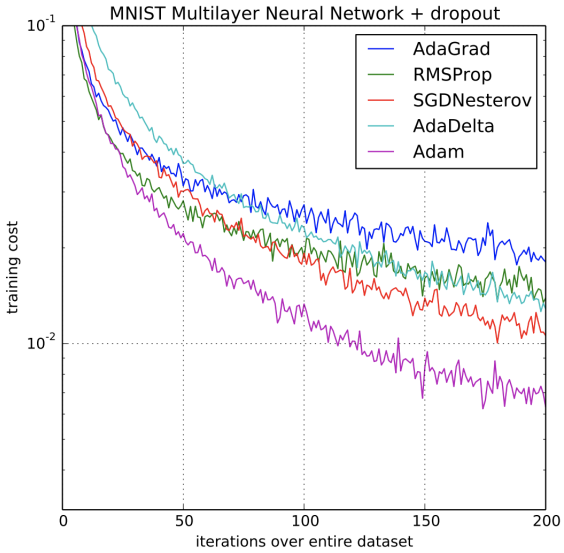


Figure 2.6.5: Performance of different optimization methods while training a multilayer perceptron for MNIST dataset. The computational performance of Adam is better than the standard optimization alternatives. Taken from [48].

Figure 2.6.4 shows that Adam has been the most selected optimization algorithm in DL research. Moreover, Adam optimization has been proved to be more effective in some classical problems, as can be seen in Figure 2.6.5.

2.6.4.2 Weight Initialization

When working with Deep ANNs, initializing the network with the right weights can be the difference between the network converging in a reasonable amount of time and the network loss function not going anywhere even after millions of iterations. This is usually due to a phenomenon called **vanishing gradient**. When using activation functions like the ones described in Section 2.6.3, which squash a large input space into a small output space, a large change in the input of the activation function will cause a small change in the output. In other words, the derivative becomes small. Figure 2.6.6 illustrates this behaviour for the sigmoid activation case. For shallow networks with only a few layers

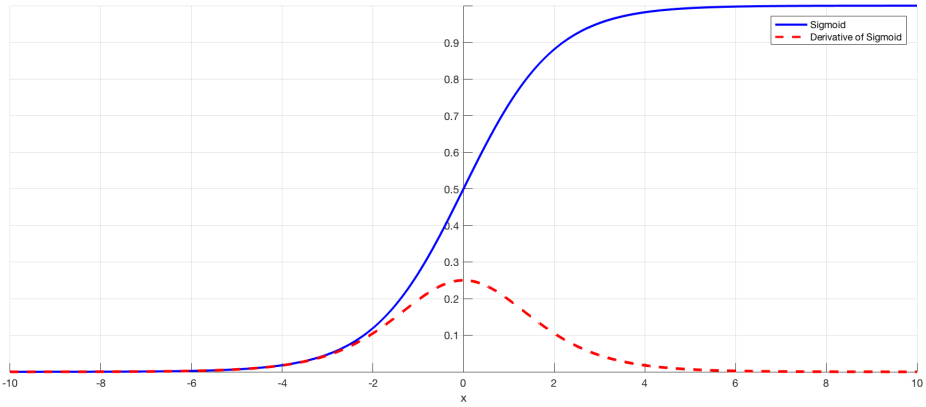


Figure 2.6.6: Vanishing derivative for the sigmoid function. The derivative takes small values which, after being multiplied with other small derivatives among different layers, tends to zero and leads to the vanishing gradient problem.

that use these activations, this is not a significant problem. However, when more layers are used, it can cause the gradient to be too small for the training of DL models to work effectively.

As described in Section 2.6.2 gradients of neural networks are found using backpropagation. By the chain rule, the derivatives of each layer are multiplied down the network from the final layer to the initial one to compute the partial derivatives of all the layers in the network. However, when L hidden layers use an activation like the sigmoid function, L small derivatives are multiplied together. Thus, the gradient may decrease exponentially as we propagate down to the initial layers. A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training iteration. Since these initial layers are often crucial to recognize the core elements of the input data, this can lead to overall inaccuracy of the whole network. As we said, this is called the vanishing gradient problem.

Therefore, initializing the network with the right weights is very important in order to avoid this vanishing effect and allow a Deep Neural Network to function properly. It is necessary to that the weights are in a reasonable range before training of the network starts. This is where **Xavier initialization** [51], one of the main advances that relaunched DL frameworks, comes into the picture. Xavier initialization method initializes the weights from a Uniform or Gaussian distribution with zero mean and some finite variance. In order to find which value to assign to this variance, let us consider a linear neuron like this

$$s = \quad w_1 z_1 + w_2 z_2 + ... + w_{N_{in}} z_{N_{in}} + b \ , \tag{2.6.20}$$

where s is the activation of the neuron, N_{in} the number of neurons in the previous layer, z_i the outputs of neurons from that previous layer, and w their corresponding weights. Without loss of generality, we will assume that the data is centered on the mean and the bias term b is therefore set to zero.

With each passing layer, the idea is that the variance remains the same. This helps keeping the signal from exploding to a high value or vanishing to zero. In other words, it is necessary to initialise the weights in such a way that the variance remains the same for z and s. The variance of s can be formulated as

$$\text{Var}(s) = \text{Var}(w_1 z_1 + w_2 z_2 + ... + w_{N_{in}} z_{N_{in}}) \ . \tag{2.6.21}$$

Assuming that the terms in the right side of the equation are independent, the following equivalence holds

$$\text{Var}(s) = \text{Var}(w_1 z_1) + \text{Var}(w_2 z_2) + ... + \text{Var}(w_{N_{in}} z_{N_{in}}) \ . \tag{2.6.22}$$

It is also true that

$$\text{Var}(w_i x_i) = E[z_i]^2 \text{Var}(w_i) + E[w_i]^2 \text{Var}(z_i) + \text{Var}(w_i)\text{Var}(z_i) \ . \tag{2.6.23}$$

where $E[]$ stands for expectation of a given variable. As the assumption was that the inputs and weights are coming from independent distributions of zero mean, the E[] terms vanish and we get

$$\text{Var}(w_i z_i) = \text{Var}(w_i)\text{Var}(z_i) \ . \tag{2.6.24}$$

Plugging (2.6.24) into (2.6.22) we get

$$\text{Var}(s) = \text{Var}(w_1)\text{Var}(z_1) + ... + \text{Var}(w_{N_{in}})\text{Var}(z_{N_{in}}) \ . \tag{2.6.25}$$

Assuming that the terms in the right side of the equation are not only independent but also identically distributed, we can write

$$\text{Var}(s) = N_{in}\text{Var}(w)\text{Var}(z) \ . \tag{2.6.26}$$

Therefore, if we want the variance of s to be the same as that of z, i.e., $\text{Var}(s) = \text{Var}(z)$, then it is necessary that $N_{in}\text{Var}(w) = 1$ or equivalently

$$\text{Var}(w) = \frac{1}{N_{in}} \ . \tag{2.6.27}$$

A similar analysis over the backward pass [51], leads us to the following analogous formula

$$\text{Var}(w) = \frac{1}{N_{out}} \ , \tag{2.6.28}$$

where N_{out} is the number of neurons in the next layer. Taking this into account, in the original paper the authors take the average of the number of input neurons, N, and the number of output neurons, N_{out}, in order to find a compromise between these two constraints. Therefore, the final proposed formula becomes

$$\text{Var}(w) = \frac{2}{N_{in} + N_{out}} \ . \tag{2.6.29}$$

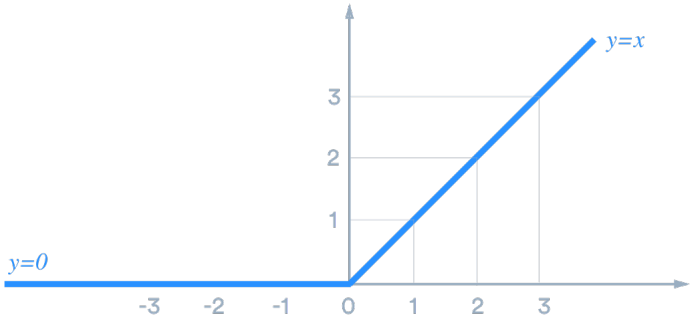


Figure 2.6.7: Rectified Linear Unit, ReLU.

2.6.4.3 Rectified Linear Unit, ReLU

Apart from using an appropiate weight initialization such as the Xavier initialization, another method to avoid the vanishing gradient effect is to employ an activation function other than the classical sigmoid and hyperbolic tangent functions. To achieve this goal, the Rectified Linear Unit, ReLU, was proposed [52].

The idea here is to find an activation function that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. The function must also provide more sensitivity to the activation sum input and avoid easy saturation. The rectified linear activation function is a simple piecewise linear function that returns the value provided as input directly if this input is positive, or the value 0 if the input is 0 or less, i.e

$$f(x) = \begin{cases} x, & x > 0 \ , \\ 0, & x \leq 0 \ . \end{cases} \tag{2.6.30}$$

A visualization of the ReLU activation function can be seen in Figure 2.6.7. The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. It also preserves many of the properties that make linear models generalize well. In particular, this linearity makes this activation function resistant to saturation, as positive values are not compressed into a narrow range of values, as was the case for the sigmoid or tanh functions, but over the entire range of possible positive values. However, it is still a nonlinear function as negative values lead always to a zero output, allowing the model to learn more complex relationships taking advantage of the power of using a DL framework with several hidden layers.

Looking at (2.6.30) it can be seen that ReLU is a non-differentiable function at $x = 0$. However, it is important to notice that having exactly $x = 0$ during ReLU computations is very rare, and hence this is not a significant issue. Subgradients can be used in the backpropagation algorithm, which in this case are 1 on the right side of $x = 0$ and 0 on the left side.

The mathematical calculations described in Section 2.6.4.2 do not fit the case of an activation function like ReLU, which is a non-differentiable function at $x = 0$. A similar

method, called He initialization, is proposed in [53] to initialize the Rectified Linear Unit function, which gives as final result

$$\mathrm{Var}(w) = \frac{2}{N_{in}}\ , \tag{2.6.31}$$

for the forward pass, i.e, twice the variance value obtained employing Xavier initialization. This would transform into $\mathrm{Var}(w) = 4/(N_{in} + N_{out})$ if we try to again find a compromise between forward and backward passes, although the authors of the paper do not propose this in [53], stating instead that it is sufficient to use (2.6.31) because if the initialization properly scales the forward signal, then this is also the case for the backward signal; and vice versa.

2.6.4.4 Computational Power and Data Volume

When asked to estimate the growth of computer technology, Gordon Moore stated the following, in a claim which would later become known as *Moore's Law*:

> "The number of transistors on an affordable CPU would double every two years"

This has been commonly, and mistakenly, rephrased as "the processing power of computers will double every two years". Although this is not a technically correct interpretation, it is true that Moore's law is directly related to the increase of computational power. The processing power of a computer processor, or CPU, can be measured in Floating Operations Per Second, FLOPS. Recent research [54] has drawn comparisons between the most powerful computer processors from 1956 to 2015. Over that time period, the authors claim that there has been a one-trillion-fold increase in FLOPS of computer processing power.

On the other hand, a graphics processing unit, GPU, is very efficient, clearly more than a CPU, in matrix multiplication and convolution, two types of computations extremely relevant and frequent in DL training. The use of GPUs for scientific computing started some time back in 2001 with implementations of matrix multiplication. One of the first common algorithms to be implemented on GPU in a faster manner was LU factorization in 2005. But, at this time researchers had to code every algorithm on a GPU and had to understand low level graphic processing. In 2006, Nvidia came out with a high level language, CUDA [55], which helps you write programs for graphic processors. This was probably one of the most significant changes in they way researchers interacted with GPUs. All these advances in computational power have allowed researchers to train and validate more complex and varied DL frameworks, a task that can be computationally expensive.

Apart from the rise in computational power, it is also important to bear in mind the increase in the data available to train the models. DL models are more strongly influenced by the volume of data available than other families of ML models. This property has two sides. On the one hand, usually DL frameworks can only outperform other Machine Learning methods when the data available is sufficiently large. If this is not the case, standard algorithms can be a better option either for better performance of for lower computational costs. On the other hand, when significantly large datasets are available, DL models, due to their special complexity and its feature learning prowess, are able to extract more relevant information for the task at hand, often leading to clearly better

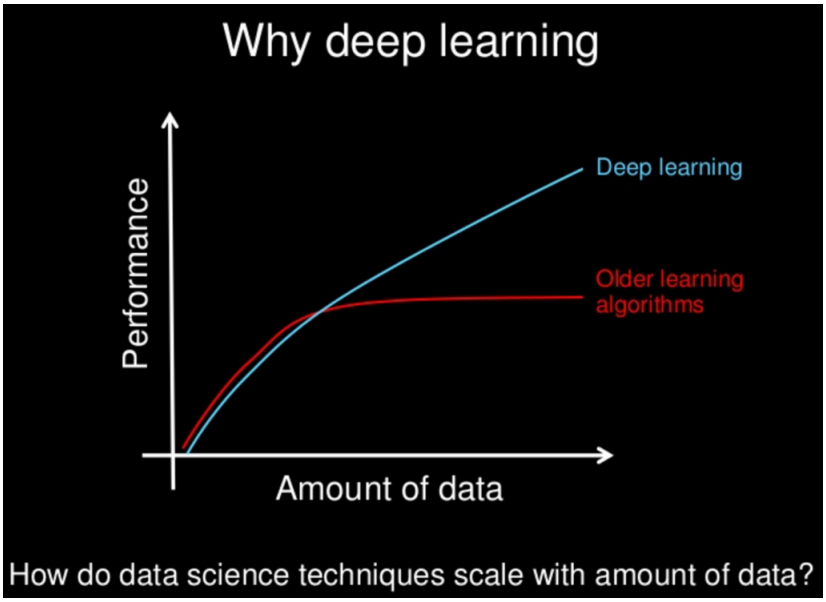


Figure 2.6.8: DL vs standard ML depending on data volume. DL outperforms classical ML when the data available is large enough. Taken from [56].

results. Figure 2.6.8 illustrates how this behaviour is present in the performance of DL models. As stated in Section 1, datasets available are becoming exponentially bigger in the big data era, being this fact an important factor in the recent grow in popularity of DL frameworks.

2.7 Clustering

Clustering methods are unsupervised Machine Learning techniques in which we do not have a target and in which we want to group the data to find patterns. The idea is to automatically find groupings or clusters of elements according to a measure of similarity between them. The fundamental objective of clustering techniques is to identify groups or clusters of elements such that the following properties are achieved:

- High intra-cluster similarity: The average similarity between elements of the same cluster is high.
- Low inter-cluster similarity: The average similarity between elements of different clusters is low.

There are several classes of clustering according to the technique used to separate the groups. The two main groups of clustering methods are hierarchical clustering and partition clustering. We will focus here on partition clustering. The partition clustering technique distributes the elements among a predetermined number of clusters or groups. We define the centroid of a cluster s_i, which we will denote C_i, as the point y that minimizes the sum of the similarities to the rest of the elements of the cluster, where similarity is measured by a particular choice of distance metric d.

$$C_i = \arg\min_{y} \sum_{x \in s_i} d(x, y) \ . \tag{2.7.1}$$

Algorithm 1: K-means algorithm.

1 Initialization of centroids.
2 $t = 0$.
3 **do**
4 | $t = t + 1$
5 | Assignment step: Each point is assigned to the cluster s_i^t with the closest centroid C_i^{t-1}.
6 | Update step: Centroids C^t of new clusters S^t are computed.
while $C^t \; != \; C^{t-1}$;
7 **return** $S^t = \{s_1^t, s_2^t, ..., s_K^t\}$

This technique receives as input the number of clusters to be formed in addition to the elements to be classified and the matrix of similarities. The most popular partition clustering technique is K-means.

2.7.1 K-means

In [57] and [58], the K-means algorithm is proposed. Its aim is to divide N points in d dimensions into K clusters so that the within-cluster sum of squares is minimized. In other words, its objective is to find

$$\min_{S} \quad \sum_{i=1}^{K} \left(\sum_{x \in s_i} ||x - C_i||^2 \right) , \tag{2.7.2}$$

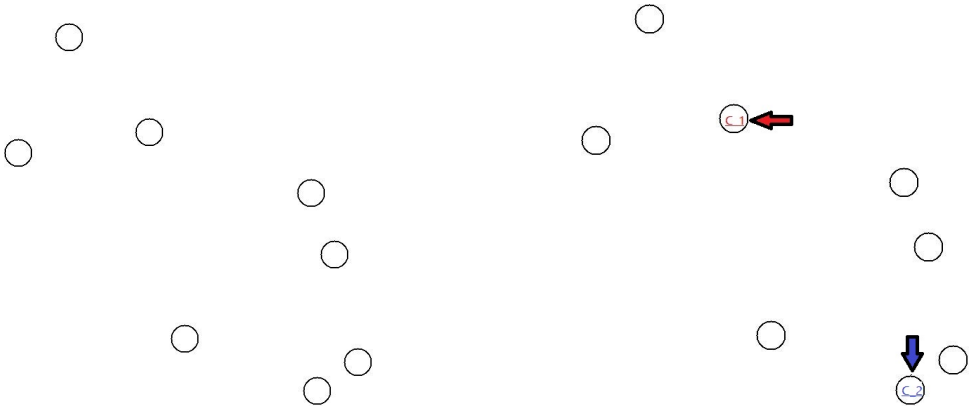
where $S = \{s_1, s_2, ..., s_K\}$ are the different clusters created and C_i is the centroid of s_i.

The K-means algorithm can be summarized as Algorithm 1. To solve this problem it is required as input a matrix of N points in d dimensions and a matrix of K initial cluster centroids in d dimensions, $C^0 = \{C_1, C_2, ..., C_K\}$. The solution found by K-means depends on the choice of C^0, i.e., this is not a problem with a unique solution.

There are several initialization methods to choose C^0. We propose here to employ the Forgy method because according to [59], for the standard K-means algorithms, which is the one we use in this thesis, the Forgy method of initialization is preferable. The Forgy method is an initialization algorithm that randomly chooses K observations from the data set and uses them as the initial cluster centroids C^0. An example of this initialization method can be seen in Figure 2.7.1.

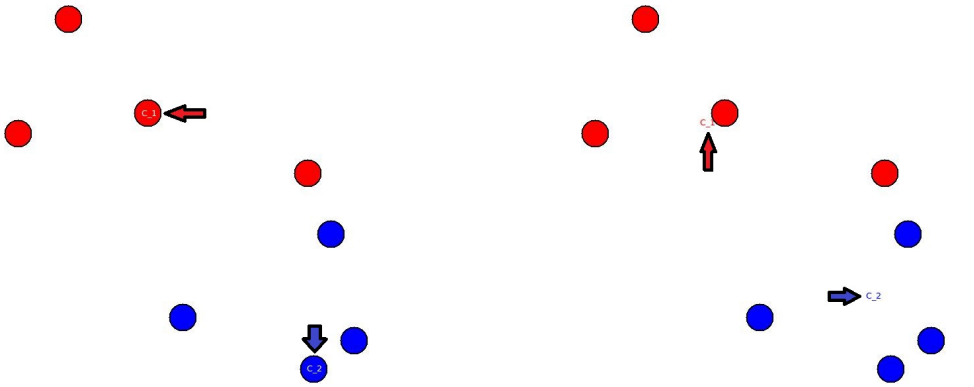
As there is not an unique solution and the K-means algorithm can get stuck at a bad local minimum, it is recommendable to run the algorithm for different initial centroid values and choose the solution that gives a smaller within-cluster sum of squares among all executions.

Once we have an initial set of cluster centroids C^0, the K-means algorithm proceeds by iterating two steps:

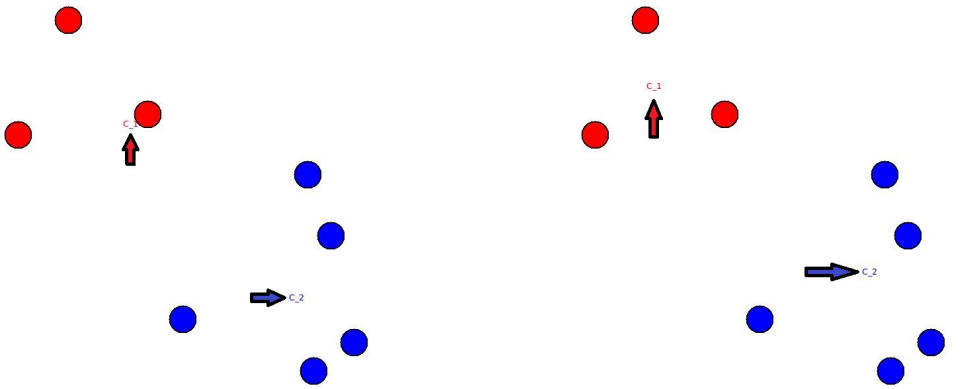


(a) Initial data. (b) Initialization step.

Figure 2.7.1: Initialization of centroids using Forgy method. For each cluster, an observation is selected randomly as its centroid.



(a) Assignment step of iteration 1. (b) Update step of iteration 1.



(a) Assignment step of iteration 2. (b) Update step of iteration 2.

Figure 2.7.3: K-means iterating steps. Assignment and update steps are repeated in each iteration of the algorithm, until the stopping criteria is fulfilled.

- **Assignment step:** Assign each observation, x^j, to the cluster s_i with the minimum euclidean distance between its centroid, C_i, and the observation, i.e. x^j is assigned to cluster s_i where

$$i = \arg\min_k \quad ||x^j - C_k|| \ . \tag{2.7.3}$$

- **Update step:** Compute the mean of all points in each cluster obtained in the previous step and set it to be the new cluster centroid, i.e., we have

$$C_i = \frac{1}{|s_i|} \sum_{x \in s_i} x \ , \ i = 1, ...K \ , \tag{2.7.4}$$

where $|s_i|$ is the total number of points in cluster s_i.

A visualization of the first two iterations of these steps is shown in Figure 2.7.3. As a result of the previous loop, the centroids may change their position in a step by step manner. These two steps are iterated until a situation is reached where none of the centroids changes anymore, i.e. $C^t = C^{t-1}$. This indicates the convergence criterion for clustering and hence at this point the iterations stop and the resulting clusters are the final solution given by the K-means algorithm. Thus, at least one iteration of the algorithm is needed to reach convergence, as it is necessary to observe no change in the centroids between the beginning of an iteration and its end for the convergence criterion to be reached. In this case, convergence is reached after the first two iterations shown in Figure 2.7.3, so the algorithm will stop there and retrieve the clusters at that iteration, S^2, as the final output.

Finally, the only thing left to do will be to select the value of K, i.e. how many clusters we want to create. In our particular case, the K-means algorithm is employed as a previous step to a posterior application of other algorithm or model, which in this thesis is our proposed approach to build prediction error intervals, aiming to improve their accuracy. In these cases, it seems preferable to select the value of K according to its positive or negative impact in this posterior goal. Therefore, in our experiments K is chosen as the value that produces the greatest improvement in the error intervals accuracy. In other words, we consider K as an extra hyperparameter to optimize in our model framework.

2.7.2 K-prototypes

K-means can only be applied to numerical values. In [60] an algorithm called K-prototypes which extends the K-means method to datasets with mixed numeric and categorical values is presented. In this method, the metric used to measure the dissimilarity between two mixed-type objects, x^a and x^b, is not the squared Euclidean distance used in K-means but the following one

$$d_P(x^a, x^b) = \sum_{j=1}^{p} (x_j^a - x_j^b)^2 + \gamma \sum_{j=p+1}^{m} \delta(x_j^a, x_j^b) \ , \ i = 1, ...K \ , \tag{2.7.5}$$

where $\{x_1, ..., x_p\}$ are numerical variables, $\{x_{p+1}, ..., x_m\}$ are categorical variables, γ is a weight factor to balance the relevance of each type of attribute, and δ is the simple matching similarity measure, whose formulation is the following

$$\delta(x_j^a, x_j^b) = \begin{cases} 0, & x_j^a = x_j^b \\ 1, & x_j^a \neq x_j^b \end{cases} \ . \tag{2.7.6}$$

It is easy to see that the first term is the squared Euclidean distance measure on the numeric attributes and the second term is the simple matching dissimilarity measure on the categorical attributes. The influence of the balancing parameter γ in the clustering process is discussed in [61].

The rest of the logic behind the K-prototypes method, namely inicialization of centroids, assignment and update steps, is analogous to the one described in Algorithm 1 for K-means, but replacing the sum of squares metric in (2.7.2) and (2.7.3) by (2.7.5). Therefore, in the update step of K-prototypes x^j is assigned to cluster s_i where

$$i = \arg\min_k \quad d_P(x^j, C_k) \ . \tag{2.7.7}$$

In the experiments carried out during this work we have not made use of K-prototypes because our datasets contained only numerical variables, as is explained in Chapter 4. Nevertheless, if we, for instance, had decided to include temporal information like the month or week corresponding to each observation, which should be treated as categorical variables, as input variables in the solar or wind datasets we employed to test our models, the use of K-protoypes will allow to consider these additional predictor variables in a straightforward manner.

Chapter 3

General Noise Models

Research is creating new knowledge.

Neil Armstrong

As stated earlier, general noise versions of SVR models should be more effective than standard versions like $\epsilon-$SVR when applied to regression problems whose underlying noise distribution follows the one assumed for that particular cost function. However, the use of these general loss functions implies that SMO [30] is no longer a suitable optimization method to solve the corresponding formulations of the problem.

In addition, when working with sufficiently large sample sizes, Deep Learning, DL, frameworks are able to extract more complex and meaningful relationships from the data than other ML families of models. Therefore, it is to be expected that a DL version of general noise models could also show more predictive prowess than their standard counterparts.

Finally, construction of error intervals for SVR models has not received too much attention in the academic world and remains mainly an unsolved problem. Being able to compute this type of uncertainty intervals would mean a significant advance, as in many applications that involve solving a regression problem, not only an accurate prediction is useful but also an error interval can be extremely valuable.

For these reasons, and using all the theoretical building blocks described in Chapter 2, the goal here is to propose a novel framework to build General Noise Models, deep and kernel-based, for regression, with the possibility of also having uncertainty intervals for their predictions. In order to do this we follow the next steps:

- First, Section 3.1 proposes a framework to train General Noise SVR Models using Naive Online R Minimization Algorithm, NORMA, detailed in Section 2.5.
- Next, Section 3.2 gives a method to build Deep General Noise Models that combine the highly non-linear feature processing of DL models with the predictive potential of using general noise loss functions, among which the ϵ-insensitive loss function used in SVR is just a particular example.
- Formulations to obtain estimations of the parameters for all the probability distributions considered in this work are given in Section 3.3.

- Section 3.4 describes a direct approach to build error intervals for SVR or other type of regression models. An enhanced version of the previous uncertainty intervals, where clustering techniques are used to create different intervals for each cluster and thus improve their performance, is also explained in this section.

- Finally, Section 3.5 unifies the previous sections in a single and final model framework to train Deep General Noise Models for regression prediction with uncertainty intervals.

In this chapter we cover the theoretical aspects of all these proposed steps. Experiments and results for the proposed frameworks are included in Chapter 4.

3.1 General Noise Models Trained Using NORMA

The use of the ϵ-insensitive loss function in the classical SVR formulation described in (2.2.2) implies the assumption of a particular error distribution, related to the Gaussian family, in the data [23]. However, it has been observed that the noise in some real-world applications may satisfy other distributions [62] [63].

In Section 2.3 we detailed how a general noise version of SVR was proposed in [28], allowing to formulate a version of this model adapted to be used with any particular loss function $l(y, f(x))$. The problem with this proposal was that for several reasons, related mainly to the loss of some mathematical properties that existed when using the ILF function, plugging general cost functions into the general dual formulation problem defined in (2.3.4) often leads to SMO no longer being a feasible choice as the optimization method [28]. Therefore, we propose here to use NORMA optimization [38] to avoid solving the dual formulation and tackle instead the corresponding primal optimization problem in an online setting. We will call these models General Noise Models.

Additionally, in Section 2.3 we also detailed the specific formulation of the optimal loss function for Laplace and Gaussian distributions, as proposed in [27]. We add here to these formulations the ones corresponding to the Beta and Weibull ones. These two distributions were selected while doing this thesis because they have been shown to be relevant in wind and solar energy forecasting [64] [65], which are the fields to which the real datasets used in our experiments belong.

3.1.1 The Beta Loss

When assuming that Beta is the underlying distribution for the error, the error density formula is assumed to be [66]:

$$P(\xi_i) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\xi_i^{\alpha-1}(1-\xi_i)^{\beta-1} , \tag{3.1.1}$$

where $\alpha, \beta > 0$ and $\Gamma(z)$ is the gamma function defined by

$$\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}dx . \tag{3.1.2}$$

Taking the negative log of (3.1.1) as in equation (2.3.14) and denoting it by l we get

$$\begin{aligned} l(\xi_i) &= -\log\Gamma(\alpha+\beta) + \log\Gamma(\alpha)\Gamma(\beta) - \log\xi_i^{\alpha-1} - \log(1-\xi_i)^{\beta-1} \\ &= \log\Gamma(\alpha) + \log\Gamma(\beta) - \log\Gamma(\alpha+\beta) - (\alpha-1)\log\xi_i - (\beta-1)\log(1-\xi_i) \\ &= K + (1-\alpha)\log\xi_i + (1-\beta)\log(1-\xi_i)\ , \end{aligned} \tag{3.1.3}$$

where $K = \log\Gamma(\alpha) + \log\Gamma(\beta) - \log\Gamma(\alpha+\beta)$. K is independent of ξ_i, so we ignore it and work with the following expression

$$l(\xi_i) = \quad (1-\alpha)\log\xi_i + (1-\beta)\log(1-\xi_i)\ , \tag{3.1.4}$$

where, due to the definition of the Beta distribution, $0 < \xi_i < 1$.

3.1.2 The Weibull Loss

The error distribution is assumed now to be [67]

$$P(\xi_i) = \begin{cases} \frac{\kappa}{\lambda}\left(\frac{\xi_i}{\lambda}\right)^{\kappa-1} e^{-\left(\frac{\xi_i}{\lambda}\right)^{\kappa}}\ ,\ \xi_i > 0 \\ 0\ ,\ \xi_i \leq 0 \end{cases}\ , \tag{3.1.5}$$

where $\lambda, \kappa > 0$. We will focus here in the case where $\xi_i > 0$, which can be achieved simply by computing the absolute error. Taking the negative log of the equation for $\xi_i > 0$ in (3.1.5) as in equation (2.3.14) we get

$$\begin{aligned} l(\xi_i) &= -\log\frac{\kappa}{\lambda} - \log\left(\frac{\xi_i}{\lambda}\right)^{\kappa-1} - \log e^{-(\frac{\xi_i}{\lambda})^{\kappa}} \\ &= -\log\kappa + \log\lambda - (\kappa-1)\log\xi_i + (\kappa-1)\log\lambda + \left(\frac{\xi_i}{\lambda}\right)^{\kappa} \\ &= K + (1-\kappa)\log\xi_i + \left(\frac{\xi_i}{\lambda}\right)^{\kappa}\ , \end{aligned} \tag{3.1.6}$$

where $K = -\log\kappa + \log\lambda + (\kappa-1)\log\lambda$. K is again independent of ξ_i so we ignore it and work with the following expression

$$l(\xi_i) = (1-\kappa)\log\xi_i + \left(\frac{\xi_i}{\lambda}\right)^{\kappa}\ . \tag{3.1.7}$$

Merging the loss functions described in Table 2.3.1 with the formulations for Beta and Weibull just obtained, we get Table 3.1.1, which contains all the loss functions we will consider for our general noise models.

Table 3.1.1: Loss functions corresponding to several error distributions.

Error Distribution	Loss Function
ILF	$l(\xi_i) = \begin{cases} -\xi_i - \epsilon, & \xi_i < -\epsilon \\ 0, & \xi_i \in [-\epsilon, \epsilon] \\ \xi_i - \epsilon, & \xi_i > \epsilon \ . \end{cases}$
Laplace	$l(\xi_i) = \frac{\lvert\xi_i - \mu\rvert}{\sigma}$
Gaussian	$l(\xi_i) = \frac{(\xi_i - \mu)^2}{2\sigma^2}$
Beta	$l(\xi_i) = (1-\alpha)\log\xi_i + (1-\beta)\log(1-\xi_i)$
Weibull	$l(\xi_i) = (1-\kappa)\log\xi_i + \left(\frac{\xi_i}{\lambda}\right)^{\kappa}$

3.1.3 General Noise Models Formulation

As described in Section 2.3.1, a general noise formulation for SVR has been proposed in [28], providing us with an expression of the dual problem that allows to insert different loss functions into it. As previously discussed, the difficulty with this formulation is that it aims to solve the dual problem, which for some choices of noise distributions results in a very complex optimization problem, one that is not possible to tackle using standard optimization techniques such as SMO [30], or cannot even be obtained due to the impossibility of computing derivatives over the Lagrangian. Therefore, we need to find a different optimization method for our proposed model, and we will follow the approach we first presented in [68] to tackle this issue.

We recall that NORMA optimization [38] can be used in a straightforward manner, both for classification and regression problems. Furthermore, its extension from linear models to non-linear ones is also largely direct via the use of the kernel trick. Finally, its formulation and implementation is fairly simple and its generalization to any loss function does not suppose great difficulties and is perfectly suited to avoid the extra complexity derived of inserting general noise functions into the dual problem. For all these reasons, NORMA is the optimization method we use to implement the proposed general noise kernel-based models, which we will call Kernel-GNM.

We study now the optimization problem resulting of using NORMA with the distributions considered for this work. These distributions have been chosen for either being standard alternatives, as Laplace and Gaussian distributions [18] [27], or being related to radiation or wind forecasting, as is the case for the Beta and Weibull distributions [64]. First, we needed to compute their associated loss functions, and we did it in Sections 2.3.3, 3.1.1, and 3.1.2, with the formulations for these loss functions available in Table 3.1.1.

Now, we want to insert these loss functions into the update rules in (2.5.11) and (2.5.10). However, the loss function l only appears in these expressions through its derivative, so this must be first computed. In Table 3.1.2 we show the derivatives for all the loss functions

Table 3.1.2: Derivatives of the loss functions corresponding to the considered distributions.

Distribution	Loss Function Derivative
ILF	$l'(\xi_i) = \begin{cases} -1, & \xi_i < -\epsilon \\ 0, & \xi_i \in [-\epsilon, \epsilon] \\ 1, & \xi_i > \epsilon \ . \end{cases}$
Laplace	$l'(\xi_i) = \begin{cases} \frac{1}{\sigma} \ , \xi_i - \mu > 0 \\ 0 \ , \xi_i - \mu = 0 \\ -\frac{1}{\sigma} \ , \xi_i - \mu < 0 \end{cases}$
Gaussian	$l'(\xi_i) = \frac{\xi_i - \mu}{\sigma^2}$
Beta	$l'(\xi_i) = \frac{1-\alpha}{\xi_i} - \frac{1-\beta}{1-\xi_i}$
Weibull	$l'(\xi_i) = \frac{1-\kappa}{\xi_i} + \frac{\kappa}{\lambda}(\frac{\xi_i}{\lambda})^{(\kappa-1)}$

we will consider in the experiments of this thesis. Following the same approach as in the case of the ReLU activation function, for the Laplace distribution when the values of l are 0 we set the derivative to 0, which would correspond to one of the subderivatives of l at that point.

As explained in Section 2.5, the solution function formulation when using NORMA optimization can be expressed at the step t as

$$\widehat{f}_t(x) = \sum_{i=1}^{t-1} \widehat{\alpha}_i^t k(x_i, x) \ , \tag{3.1.8}$$

where $\widehat{\alpha}_i$ are the NORMA coefficients and k a particular kernel function. An offset term could be added to this formula, but we will not consider it here for the sake of simplicity. Our goal now is to obtain explicit formulations for $\widehat{\alpha}$ for each one of the noise distributions considered, which will yield an adjusted NORMA formulation.

We can now plug the derivatives in Table 3.1.2 into the update rules for NORMA described in Section 2.5. In particular, we will use the following equation

$$\widehat{\alpha}_t^t = -\eta_t l'(f_t(x_t), y_t) \ . \tag{3.1.9}$$

Inserting the l' formulation corresponding to our choice of noise distribution we can get this $\widehat{\alpha}_t$ formulation adapted to this particular distribution. We describe here the explicit formulations for all the distributions considered in the experiments except for the ILF, where the SVR model will be computed using the standard dual formulation solved by SMO.

Let's denote by ψ_t the prediction error, $\psi_t = \widehat{f}(x_t) - y_t$. For the Laplace and Gaussian distributions we will use in their formulations $\xi_t = \psi_t$. For the Beta distribution, it is required that $0 < \xi_t < 1$ so we will be using as error $\xi_t = min(\frac{|\psi_t|}{max(\psi^l)}, 1)$, where ψ^l are the errors obtained by a basic model like linear regression. Finally, for the Weibull distribution it is enough that $\xi_t > 0$, so we will use $\xi_t = |\psi_t|$. Using this notation, we end up with the following formulations.

1. **Laplace:**

$$\widehat{\alpha_t^t} = \begin{cases} -\frac{\eta_t}{\sigma} \ , \ \xi_t - \mu > 0 \ , \\ 0 \ , \xi_t - \mu = 0 \ , \\ \frac{\eta_t}{\sigma} \ , \ \xi_t - \mu < 0 \ . \end{cases} \tag{3.1.10}$$

2. **Gaussian:**

$$\widehat{\alpha_t^t} = -\eta_t \frac{\xi_t - \mu}{\sigma^2} \ .$$

3. **Beta:**

$$\widehat{\alpha_t^t} = -\eta_t \left(\frac{1-\alpha}{\xi_t} - \frac{1-\beta}{1-\xi_t} \right) \ .$$

4. **Weibull:**

$$\widehat{\alpha_t^t} = -\eta_t \left[\frac{1-\kappa}{\xi_t} + \frac{\kappa}{\lambda} \left(\frac{\xi_t}{\lambda} \right)^{(\kappa-1)} \right] \ .$$

Regarding the stopping criteria for these updating iterative process we use two rules. First, we define a maximum number of NORMA update iterations. Second, we define a minimum tolerance threshold, so if the relative difference between the $\widehat{\alpha_t^t}$ obtained in two consecutive iterations is lower than this threshold, we stop the iterative process and return the last $\widehat{\alpha_t^t}$ computed.

As stated before, NORMA is based on stochastic gradient descent. Asymptotic convergence to a stationary point for these optimization methods is proved in [44] in the non-convex case, but this point is not guaranteed to be a global minima as opposed to the convex situation. To avoid possible issues related to this, we opt to constrain the parameters of the chosen distribution to be outside the set of parameters which cause the function to be non-convex. This means that in the Beta distribution for instance we need to use the constraints $\alpha > 1$, $\beta > 1$. Regarding the Weibull distribution, convexity depends on the value κ. When this parameter, usually called *shape*, is greater or equal than 1, the distribution curve is convex over the entire range.

We consider it is important to point out that the extension of the approach presented here to other choices of distribution assumption outside the ones considered here, like the Poisson distribution, is also possible, with only simple computations of Maximum Likelihood Estimation to get the optimal loss functions and the required calculation of the derivatives of these functions. This will be one of the lines for further work we will discuss in Section 5.1

3.2 Deep SVR and Deep General Noise Models, D-GNM

Deep Learning, DL, frameworks have been shown to achieve better performance than the more classical Machine Learning families of models when trained with datasets that are sufficiently large. Besides, SVM models are not computationally feasible in most setups when dealing with large datasets of hundreds of thousands or millions of samples and hundreds or thousands of variables. These factors have been analyzed and explained in more detail in some of our previous research papers [69] [70]. This fact is probably one of the main factors why DL models are becoming the preferred choice over SVMs when solving a high variety of supervised learning tasks regarding large tabular data. For this reason, in this thesis we propose to plug our General Noise Models, GNM, into a DL framework, adding to the potential of adapting to any noise distribution inherent of GNM the predictive potential that DL models have when trained with large datasets.

Although recently the concept of DL has been used almost interchangeably with Deep Neural Networks, we could consider as DL any Machine Learning structure that uses several layers, adding complexity with each one. In particular, the use of the common structure in a fully connected DL model, i.e., several layers of neurons or units, combined with applying a loss function different to the ones normally considered for ANNs would also be a DL appoach. This is the methodology we propose here to integrate our GNM models into a DL structure.

First, we will describe in Section 3.2.1 how to build a Deep version of the $\epsilon-$SVR model, where the ILF loss function replaces the classical loss function used in DL applied to regression, which is the squared error. Next, we will follow the same approach to insert any given loss function l into the DL framework to obtain Deep General Noise Models adapted to a particular choice of noise distribution in the data.

3.2.1 Deep SVR

Our goal now is to build a Deep version of an SVR, in the sense that we want to combine a DL structure with the use of the $\epsilon-$insensitive loss function. Let us consider a standard Deep ANN architecture where an input layer is followed by a number of hidden layers and, finally, one or several linear output activations leading to our final prediction. The transformation of such a network can be written as

$$f(x, w, b, W_h) = \quad w \cdot F(x, W_h) + b \ , \tag{3.2.1}$$

where w and b denote the linear weights and bias acting on the last hidden layer, respectively, W_h denotes the weights and biases up to the last hidden layer, and $F(x, W_h)$ represents the last hidden layer outputs.

The optimal weights of such a model are to be obtained minimizing a regularized cost or objective function, which applies weight decay as an additional penalty term to the selected loss function [70]. This objective function has the following formulation

$$c(w, b, W_h) = \quad \frac{1}{N} \sum_{i=1}^{N} l(y_i, w \cdot F(x_i, W_h) + b) + \lambda_1 ||w||^2 + \lambda_2 ||W_h^*||^2 \ , \tag{3.2.2}$$

where l is the given choice of loss function selected for the task at hand, y_i the targets to predict, W_h^* the weights W_h except from the bias, and λ_1, λ_2 regularization parameters.

Notice that in (3.2.1) we keep the option of using different weight penalties λ_1 and λ_2 for the linear output weights and for the hidden layer ones, respectively, although this is optional and one could simply choose $\lambda_1 = \lambda_2 = \lambda$, which leads us to the following simplified formulation

$$\min_{w,b,W_h} \quad J(w,b,W_h) = \frac{1}{N}\sum_{i=1}^{N} l(y_i, w \cdot F(x_i, W_h) + b) + \lambda(||w||^2 + ||W_h^*||^2) \ . \tag{3.2.3}$$

In any case, these regularization hyperparameters, as is common in ML models, are to be selected by some form of search over cross validation or a fixed validation set, as explained in detail in Section 1.2.

Is is important to remark here that there exists another popular regulatization technique in DL structures, which is called dropout. When dropout is applied, during the training of the DL model, some number of layer outputs are randomly ignored. This has the effect of making the layer look like it has a different number of nodes and connectivity. In effect, each update to a layer during training is performed with a different view of the configured layer. Dropout can be used together with or instead of weight decay, the latter being the option we selected in the experiments described in Chapter 4. Nevertheless, we have decided to include weight decay in this section to present the most generic possible formulations of the problem.

We focus here in the case of DL for regression, as it is the branch of problems we aim to solve with our proposed models. In regression, probably the most standard choice for the loss function in DL structures is the Mean Squared Error, MSE, defined by the following loss function

$$l(y_i, w \cdot F(x_i, W_h) + b) = \quad (w \cdot F(x_i, W_h) + b - y_i)^2 \ . \tag{3.2.4}$$

Inserting (3.2.4) into the formulation of J expressed in (3.2.1) we get

$$J(w,b,W_h) = \quad \frac{1}{N}\sum_{i=1}^{N} (w \cdot F(x_i, W_h) + b - y_i)^2 + \lambda(||w||^2 + ||W_h^*||^2) \ . \tag{3.2.5}$$

If we want to transform this classical DL framework into what we called a Deep SVR, it is enough to replace l in (3.2.4) for the formulation of the ILF function, i.e.

$$l_\epsilon(y_i, w \cdot F(x_i, W_h) + b) = \begin{cases} y_i - w \cdot F(x_i, W_h) - b - \epsilon, & if \ \ w \cdot F(x_i, W_h) + b - y_i < -\epsilon \ , \\ 0, \quad if \ \ w \cdot F(x_i, W_h) + b - y_i \in [-\epsilon, \epsilon] \ , \\ w \cdot F(x_i, W_h) + b - y_i - \epsilon, & if \ \ w \cdot F(x_i, W_h) + b - y_i > \epsilon \ , \end{cases} \tag{3.2.6}$$

which will give us the following formulation of the problem

$$\min_{w,b,W_h} \quad J(w,b,W_h,\epsilon) = \frac{1}{N}\sum_{i=1}^{N} [|w \cdot F(x_i, W_h) + b - y_i| - \epsilon]_+ + \lambda(||w||^2 + ||W_h^*||^2) \ , \tag{3.2.7}$$

where we recall that $[z]_+ = \max(0, z)$.

The minimization of (3.2.1) can be achieved by standard ANN solvers such as the combination of backpropagation and gradient descent described in Section 2.6.2. However, in real implementations of Deep ANN solvers, newer and more efficient optimization methods, such as stochastic gradient descent [71], Adagrad [49], RMSProp [50] or Adam [48], are preferred. We choose here to use Adam because of its advantage in non-convex optimization, which can arise in a DL framework when working with general noise loss functions, and because it has been clearly the most popular choice as optimization algorithm for Deep ANN during these last years, as shown in Section 2.6.4.1.

3.2.2 Deep General Noise Models, D-GNM

Now we propose to combine the use of general noise loss functions, described in Section 3.1, with the DL version of SVR, described in Section 3.2.1, into a single framework. The idea is to replace the loss function l defined in (3.2.2), not necessarily by the ILF loss function, l_ϵ, defined in (3.2.6), but by any choice of loss function corresponding to a relevant distribution assumed to be present in the data noise. In our experiments we will focus again on the same distributions described in Table 3.1.1. The Deep SVR defined before will be a special case of these proposed D-GNM models where the loss function used is the ILF.

We will call the models resulting of this combination Deep General Noise Models, or D-GNMs. The main purpose of these D-GNM models is to bring together the predictive potential of DL frameworks when applied to large volumes of data with the flexibility of General Noise SVR models.

Plugging the loss functions considered in Table 3.1.1 into (3.2.2), apart from ILF whose formulation has already been described in (3.2.7), we will get several problem formulations for our proposed D-GNMs, one for each distribution assumed to be present in the data. Notice here that, although ideally it would be logical to expect that the prediction error, $\psi_i = w \cdot F(x_i, W_h) + b - y_i$, has mean zero and thus we are working with an unbiased predictive model, this is often not the case when applying ML in real-world problems. Therefore, we allow the use of non-zero mean versions of Laplace and Gaussian distributions.

Recall here that $0 < \xi_i < 1$ for the Beta distribution, and $\xi_i > 0$ for the Weibull one, so we will actually be using again $\xi_i = min(\frac{|\psi_i|}{max(\psi^l)}, 1)$, where ψ^l are the errors obtained by a basic model like linear regression as prediction errors for the Beta distribution, and $\xi_i = |\psi_i|$ for the Weibull distribution. Below are the resulting loss and cost functions for each of the distributions considered, using a common regularization parameter $\lambda = \lambda_1 = \lambda_2$.

1. **Laplace:**

$$l(\xi_i) = \frac{|\xi_i - \mu|}{\sigma} , \tag{3.2.8}$$

$$J(w, b, W_h) = \frac{1}{N} \sum_{i=1}^{N} \frac{|\xi_i - \mu|}{\sigma} + \lambda(||w||^2 + ||W_h^*||^2) . \tag{3.2.9}$$

2. **Gaussian:**

$$l(\xi_i) = \frac{(\xi_i - \mu)^2}{2\sigma^2} , \tag{3.2.10}$$

$$J(w, b, W_h) = \frac{1}{N} \sum_{i=1}^{N} \frac{(\xi_i - \mu)^2}{2\sigma^2} + \lambda(||w||^2 + ||W_h^*||^2). \tag{3.2.11}$$

3. **Beta:**

$$\begin{aligned} l(\xi_i) =& (1 - \alpha) \log(\xi_i) + \\ & (1 - \beta) \log(1 - \xi_i) , \end{aligned} \tag{3.2.12}$$

$$\begin{aligned} J(w, b, W_h) =& \frac{1}{N} \sum_{i=1}^{N} [(1 - \alpha) \log(\xi_i) + (1 - \beta) \log(1 - \xi_i)] + \\ & \lambda(||w||^2 + ||W_h^*||^2) . \end{aligned} \tag{3.2.13}$$

4. **Weibull:**

$$l(\xi_i) = (1 - \kappa) \log(\xi_i) + \left(\frac{\xi_i}{\lambda}\right)^{\kappa} , \tag{3.2.14}$$

$$J(w, b, W_h) = \frac{1}{N} \sum_{i=1}^{N} \left[(1 - \kappa) \log(\xi_i) + \left(\frac{\xi_i}{\lambda}\right)^{\kappa} \right] + \lambda(||w||^2 + ||W_h^*||^2) . \tag{3.2.15}$$

In Table 3.2.1 we give a summary of all the D-GNM formulations described here. It is important to recall that in our experiments we have decided to employ dropout for regularization purposes as a replacement for weight decay.

From Table 3.2.1 one can notice that some of the expressions for J are quite complex, which could lead to issues when carrying out the optimization problem corresponding to these DL models, as it was the case for SMO in the non-deep versions of GNM models we described in Section 3.1. However, new DL programming frameworks, like *Tensorflow* or *Keras*, ease this process, applying automatic differentiation to the cost functions chosen and using it in the corresponding backpropagation steps. Furthermore, *Tensorflow* also allows for the use of loss functions that are non-differentiable at individual isolated points, which would be a problem in GNM even when using NORMA and not SMO as optimization method.

Table 3.2.1: D-GNM formulations corresponding to several error distributions.

Error Distribution	D-GNM Formulation
ILF	$\frac{1}{N}\sum_{i=1}^{N}\left[\lvert\xi_i\rvert-\epsilon\right]_+ + \lambda(\lvert\lvert w\rvert\rvert^2 + \lvert\lvert W_h^*\rvert\rvert^2)$
Laplace	$\frac{1}{N}\sum_{i=1}^{N}\frac{\lvert\xi_i-\mu\rvert}{\sigma} + \lambda(\lvert\lvert w\rvert\rvert^2 + \lvert\lvert W_h^*\rvert\rvert^2)$
Gaussian	$\frac{1}{N}\sum_{i=1}^{N}\frac{(\xi_i-\mu)^2}{2\sigma^2} + \lambda(\lvert\lvert w\rvert\rvert^2 + \lvert\lvert W_h^*\rvert\rvert^2)$
Beta	$\frac{1}{N}\sum_{i=1}^{N}\left[(1-\alpha)\log\xi_i + (1-\beta)\log(1-\xi_i)\right] + \lambda(\lvert\lvert w\rvert\rvert^2 + \lvert\lvert W_h^*\rvert\rvert^2)$
Weibull	$\frac{1}{N}\sum_{i=1}^{N}\left[(1-\kappa)\log\xi_i + (\frac{\xi_i}{\lambda})^{\kappa}\right] + \lambda(\lvert\lvert w\rvert\rvert^2 + \lvert\lvert W_h^*\rvert\rvert^2)$

3.3 Estimation of Loss Functions Parameters

We have described in this Chapter how to build general noise models, both employing kernel-based, GNM, as describen in Section 3.1 or DL frameworks, D-GNM, explained in Section 3.2. Both these models require to estimate some distribution parameters to be used in the corresponding loss functions. For instance, if we assume a Beta distribution as underlying noise of model errors, it is needed to estimate the α and β parameters of the distribution.

We propose here to follow a similar approach to the one described in [27] and discussed in Section 2.4, where the error distribution assumed is fitted by maximum likelihood estimation, MLE, using the previously computed out-of-sample residuals, ξ_i, of a classical ϵ-SVR model used to predict a regression target. In [27] zero mean Gaussian and Laplace families are considered as possible error distribution assumptions. We will extend here this approach to all the distributions considered in this work, which include non-zero mean Laplace and Gaussian distributions, as well as the Beta and Weibull ones. In any case, we are aware that other methods to select these distribution parameters could be considered and we have identified the study of other alternative approaches as a possible line of further work.

3.3.1 Parameters for the Laplace Distribution

MLE parameters for the zero mean Laplace distribution are discussed in Section 2.4.2.1. Following the same steps here for the non-zero mean case we get

$$
\begin{aligned}
l(\theta;\psi_1,...,\psi_n) &= \sum_{i=1}^{n} \log \frac{1}{2\sigma} e^{-\frac{|\xi_i-\mu|}{\sigma}} = \sum_{i=1}^{n} \log \frac{1}{2\sigma} - \sum_{i=1}^{n} \frac{|\xi_i-\mu|}{\sigma} \\
&= -n\log 2 - n\log\sigma - \frac{1}{\sigma}\sum_{i=1}^{n} |\xi_i-\mu| \ .
\end{aligned}
\tag{3.3.1}
$$

We can compute now the corresponding derivative

$$
\frac{\partial l}{\partial \sigma} = -n\frac{1}{\sigma} + \frac{1}{\sigma^2}\sum_{i=1}^{n} |\xi_i-\mu| \ . \tag{3.3.2}
$$

In the maximum point, where we denote $\widehat{\sigma}$ the corresponding density parameter value, the first derivative must be equal to zero, so

$$
-n\frac{1}{\widehat{\sigma}} + \frac{1}{\widehat{\sigma}^2}\sum_{i=1}^{n} |\xi_i-\mu| = 0 \ . \tag{3.3.3}
$$

Solving (3.3.3) we obtain

$$
\widehat{\sigma} = \frac{\sum_{i=1}^{n} |\xi_i-\mu|}{n} \ . \tag{3.3.4}
$$

and we set $\hat{\mu}$ to be the median of the ξ_i residuals.

3.3.2 Parameters for the Gaussian Distribution

MLE parameters for the zero mean Gaussian distribution are discussed in Section 2.4.2.2. Following the same steps here for the non-zero mean Gaussian we get

$$
\begin{aligned}
l(\theta;\psi_1...\psi_n) &= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\xi_i-\mu)^2}{2\sigma^2}} = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{n} \frac{(\xi_i-\mu)^2}{2\sigma^2} \\
&= -n\log\sqrt{2\pi} - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n} (\xi_i-\mu)^2 \ .
\end{aligned}
\tag{3.3.5}
$$

Setting the first derivative to zero to find the maximum point we obtain

$$
\frac{\partial l}{\partial \sigma} = -n\frac{1}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n} (\xi_i-\mu)^2 = 0 \ . \tag{3.3.6}
$$

Solving (3.3.6) we obtain

$$
\widehat{\sigma} = \frac{\sum_{i=1}^{n} (\xi_i-\mu)^2}{n} \ , \tag{3.3.7}
$$

and we set $\hat{\mu}$ to be

$$
\hat{\mu} = \sum_{i=1}^{n} \frac{\xi_i}{n} \ . \tag{3.3.8}
$$

3.3.3 Parameters for the Beta Distribution

The density of the Beta distribution is defined in (3.1.1). The likelihood function of a distribution, assuming i.i.d., is defined as

$$L(\theta;\xi_1,...,\xi_n) = \prod_{i=1}^{n} P(\theta;\xi_1,...,\xi_n) \ , \tag{3.3.9}$$

where θ are the parameters that define the distribution and we should recall that, in order to avoid possible issues due to the use of non-convex loss functions, we opt in this work to constrain the parameters of the Beta distribution to $\alpha > 1,\ \beta > 1$. Denoting $l = \log(L)$ the log-likelihood of the distribution, we get

$$l(\alpha,\beta;\xi_1,...,\xi_n) = \sum_{i=1}^{n} \log\left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\xi_i^{\alpha-1}(1-\xi_i)^{\beta-1}\right) \ . \tag{3.3.10}$$

If we expand the equation (3.3.10), we obtain

$$\begin{aligned} l(\theta;\xi_1,...,\xi_n) &= \sum_{i=1}^{n} \log\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} + \sum_{i=1}^{n}\log\xi_i^{\alpha-1} + \sum_{i=1}^{n}\log(1-\xi_i)^{\beta-1} \\ &= n(\log\Gamma(\alpha+\beta) - \log\Gamma(\alpha) - \log\Gamma(\beta)) + (\alpha-1)\sum_{i=1}^{n}\log\xi_i + \\ &\quad (\beta-1)\sum_{i=1}^{n}\log(1-\xi_i) \ . \end{aligned} \tag{3.3.11}$$

Computing and setting the derivatives of l to zero we obtain

$$\begin{aligned} \frac{\partial l}{\partial\alpha} &= n\left(\frac{\Gamma'(\alpha+\beta)}{\Gamma(\alpha+\beta)} - \frac{\Gamma'(\alpha)}{\Gamma(\alpha)}\right) + \sum_{i=1}^{n}\log\xi_i = 0 \ , \\ \frac{\partial l}{\partial\beta} &= n\left(\frac{\Gamma'(\alpha+\beta)}{\Gamma(\alpha+\beta)} - \frac{\Gamma'(\beta)}{\Gamma(\beta)}\right) + \sum_{i=1}^{n}\log(1-\xi_i) = 0 \ . \end{aligned} \tag{3.3.12}$$

where with $\Gamma(x)$ defined as in (3.1.2), we have that $\Gamma'(x)$ is

$$\Gamma'(x) = \int_0^\infty t^{x-1}e^{-t}\log t\ dt \ , \tag{3.3.13}$$

and $\frac{\Gamma'(x)}{\Gamma(x)}$ is the digamma function, which we will denote $\phi(x)$ from now on. Plugging this equation into (3.3.12) we end with the following system of two equations

$$\begin{aligned} F_1(\widehat{\alpha},\widehat{\beta},\xi_i) &= \phi(\widehat{\alpha}+\widehat{\beta}) - \phi(\widehat{\alpha}) + \frac{1}{n}\sum_{i=1}^{n}\log\xi_i = 0 \ , \\ F_2(\widehat{\alpha},\widehat{\beta},\xi_i) &= \phi(\widehat{\alpha}+\widehat{\beta}) - \phi(\widehat{\beta}) + \frac{1}{n}\sum_{i=1}^{n}\log(1-\xi_i) = 0 \ . \end{aligned} \tag{3.3.14}$$

Iterative methods may be employed for the numerical solution of the equations in (3.3.14). For instance, the Newton-Raphson's method proposes as iterative update the following equation

$$X_{n+1} = X_n - \lambda[J_f(X_n)]^{-1} f(X_n) \ , \tag{3.3.15}$$

where λ will be a learning rate that, for simplificity's sake, we will set to 1, and $J_f(X)$ is the Jacobian matrix of $f(X)$, defined by $[J_f(X)]_{ij} = \frac{\partial f_i(X)}{\partial x^j}$, which in our case is

$$J_f(\alpha, \beta) = \begin{matrix} \frac{\partial F_1}{\partial \alpha} & \frac{\partial F_1}{\partial \beta} \\ \frac{\partial F_2}{\partial \alpha} & \frac{\partial F_2}{\partial \beta} \end{matrix} \Bigg) \Bigg(\tag{3.3.16}$$

For a single variable function, i.e. $X = x$, this formulation is reduced to the following well known expression

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \ . \tag{3.3.17}$$

It can be shown [72] that under sufficient assumptions and a sufficiently accurate initial guess x_0, the updated value x_{n+1} is a better approximation of the root than x_n.

Applying Newton-Raphson's method to (3.3.14) leads to the following iterative scheme

$$\begin{bmatrix} \alpha_{n+1} \\ \beta_{n+1} \end{bmatrix} = \begin{bmatrix} \alpha_n \\ \beta_n \end{bmatrix} \Bigg(- [J_f(\alpha_n, \beta_n)]^{-1} \begin{bmatrix} F_1(\alpha_n, \beta_n, \xi_i) \\ F_2(\alpha_n, \beta_n, \xi_i) \end{bmatrix} \Bigg(\tag{3.3.18}$$

The initial values (α_0, β_0) are pivotal for a good convergence of Newton-Raphson's method. We propose here to use the following initial estimations obtained using the method of moments [73]

$$\alpha_0 = \frac{m_1(m_1 - m_2)}{m_2 - (m_1)^2} \ , \tag{3.3.19}$$

$$\beta_0 = \frac{\alpha_0(1 - m_1)}{m_1} \ , \tag{3.3.20}$$

where the $n - th$ moment, m_n, of a random variable, X, is defined as the expected value of that variable to the power of n, i.e.

$$m_n = E[X^n] \ . \tag{3.3.21}$$

3.3.4 Parameters for the Weibull Distribution

The density function of the Weibull distribution is defined in (3.1.5). We will consider here $\xi_i = |\psi_i|$, where again $\psi_i = \widehat{f}(x_i) - y_i$. Taking this into account, the log-likelihood, l, corresponding to a Weibull distribution is the following

$$l(\theta; \xi_1, ..., \xi_n) = \sum_{i=1}^{n} \Bigg(\log \left[\frac{\kappa}{\lambda} \left(\frac{\xi_i}{\lambda} \right)^{\kappa - 1} e^{-\left(\frac{\xi_i}{\lambda}\right)^{\kappa}} \right] \Bigg(, \tag{3.3.22}$$

where we should recall that, in order to avoid possible issues due to the use of non-convex loss functions, we opt in this work to use the constrain $\kappa > 1$.

If we expand the equation (3.3.22) we obtain

$$\begin{aligned} l(\theta;\xi_1,...,\xi_n) &= \sum_{i=1}^{n}\log\frac{\kappa}{\lambda}+\sum_{i=1}^{n}\log\left(\frac{\xi_i}{\lambda}\right)^{\kappa-1}-\sum_{i=1}^{n}\left(\frac{\xi_i}{\lambda}\right)^{\kappa} \\ &= n\log\kappa-n\log\lambda+(\kappa-1)\sum_{i=1}^{n}\log\xi_i-(\kappa-1)n\log\lambda-\frac{1}{\lambda^{\kappa}}\sum_{i=1}^{n}\xi_i^{\kappa} \quad (3.3.23) \\ &= n\log\kappa-\kappa n\log\lambda+(\kappa-1)\sum_{i=1}^{n}\log\xi_i-\frac{1}{\lambda^{\kappa}}\sum_{i=1}^{n}\xi_i^{\kappa}\ . \end{aligned}$$

Computing the partial derivatives of l in (3.3.23) and setting them to zero we get

$$\frac{\partial l}{\partial\lambda} = -\frac{\kappa n}{\lambda}+\frac{\kappa}{\lambda^{\kappa+1}}\sum_{i=1}^{n}\xi_i^{\kappa}=0\ , \quad (3.3.24)$$

$$\frac{\partial l}{\partial\kappa} = \frac{n}{\kappa}-n\log\lambda+\sum_{i=1}^{n}\log\xi_i-\sum_{i=1}^{n}\left(\frac{\xi_i}{\lambda}\right)^{\kappa}\log\frac{\xi_i}{\lambda}=0\ . \quad (3.3.25)$$

Solving first (3.3.24) we obtain

$$\frac{\widehat{\kappa}}{\widehat{\lambda}}\left[-n+\frac{1}{\widehat{\lambda}^{\widehat{\kappa}}}\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\right]= 0\ . \quad (3.3.26)$$

Since $\widehat{\kappa}/\widehat{\lambda}$ cannot ever be zero as $\kappa,\lambda>0$, so we have that $-n+\frac{1}{\widehat{\lambda}^{\widehat{\kappa}}}\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}=0$. Therefore it holds that

$$n = \frac{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}}{\widehat{\lambda}^{\widehat{\kappa}}}\ , \quad (3.3.27)$$

and finally

$$\widehat{\lambda} = \frac{1}{n}\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\right)^{\frac{1}{\widehat{\kappa}}}\ . \quad (3.3.28)$$

Now, plugging (3.3.28) into (3.3.25) we get

$$\begin{aligned} &\frac{n}{\widehat{\kappa}}-n\log\left(\frac{1}{n}\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\right)^{\frac{1}{\widehat{\kappa}}}+\sum_{i=1}^{n}\log\xi_i-\frac{n}{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}}\left[\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\xi_i-\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\left(\frac{1}{n}\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\right)^{\frac{1}{\widehat{\kappa}}}\right]= \\ &\frac{n}{\widehat{\kappa}}+\sum_{i=1}^{n}\log\xi_i-n\frac{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\xi_i}{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}}=0\ , \end{aligned} \quad (3.3.29)$$

which leads us to the following equation

$$\frac{\sum_{i=1}^{n}\log\xi_i}{n} = \frac{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\xi_i}{\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}} - \frac{1}{\widehat{\kappa}} \ . \tag{3.3.30}$$

Denoting the expression on the right-hand side of (3.3.30) as $G(\widehat{\kappa})$, the equation can be rewritten as

$$\frac{\sum_{i=1}^{n}\log\xi_i}{n} = G(\widehat{\kappa}) \ . \tag{3.3.31}$$

A simple proof of the existence and uniqueness of the solution of (3.3.31) is the following one. Let us denote

$$H(\kappa) = G(\widehat{\kappa}) - \frac{\sum_{i=1}^{n}\log\xi_i}{n};$$

we want to prove the existence and uniqueness of the root of $H(\widehat{\kappa}) = 0$. For $\widehat{\kappa} > 0$ and any $\xi_i \in \mathbb{R}^+$, we have that

$$\begin{aligned} &\lim_{\widehat{\kappa}\to 0} G(\widehat{\kappa}) = \sum_{i=1}^{n}\log\xi_i - \infty = -\infty \Rightarrow H(\widehat{\kappa}) < 0 \ , \\ &\lim_{\widehat{\kappa}\to\infty} G(\widehat{\kappa}) = \log\max(\xi_i) \Rightarrow H(\widehat{\kappa}) > 0 \ , \end{aligned} \tag{3.3.32}$$

and, therefore, the image of $H(\widehat{\kappa})$ contains both positive and negative values. Besides, the function is continuous, so combining these two factors the existence of a root can be assured.

Now, to prove the uniqueness of this root we only need to show the global monotonicity of $H(\widehat{\kappa})$ and this is equivalent to demonstrate $H'(\widehat{\kappa}) > 0$. The derivative takes the form:

$$H'(\widehat{\kappa}) = \frac{1}{\widehat{\kappa}^2} + \left(\sum_{i=1}^{n}\frac{1}{\xi_i^{\widehat{\kappa}}}\right)^2 \left[\left(\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log^2\xi_i\right)\left(\sum_{i=1}^{n}\xi_i^{k}\right) - \left(\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\xi_i\right)^2\right] \tag{3.3.33}$$

In equation (3.3.33), $\frac{1}{\widehat{\kappa}^2}$ and $\left(\sum_{i=1}^{n}\frac{1}{\xi_i^{\widehat{\kappa}}}\right)^2$ will always take positive values, so we focus on

$$I(\widehat{\kappa}, n, \xi_i) = \left(\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log^2\xi_i\right)\left(\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\right) - \left(\sum_{i=1}^{n}\xi_i^{\widehat{\kappa}}\log\xi_i\right)^2 \ . \tag{3.3.34}$$

If $n = 1$ we have

$$I(\widehat{\kappa}, 1, \xi_i) = \xi_1^{\widehat{\kappa}}\xi_1^{\widehat{\kappa}}\log^2\xi_1 - \xi_1^{2\widehat{\kappa}}\log^2\xi_1 = 0 \ . \tag{3.3.35}$$

If $n = 2$ we have

$$I(\widehat{\kappa}, 2, \xi_i) = \xi_1^{\widehat{\kappa}}\xi_2^{\widehat{\kappa}}\left(\log\xi_2 - \log\xi_1\right)^2 \geq 0 \ . \tag{3.3.36}$$

For $n \geq 3$, as shown in [74], if we suppose $I(\widehat{\kappa}, n-1, \xi_i) \geq 0$, then

$$I(\widehat{\kappa}, n, \xi_i) = I(\widehat{\kappa}, n-1, \xi_i) + \xi_n^{\widehat{\kappa}} \sum_{i=1}^{n-1} \xi_i^{\widehat{\kappa}} \left(\log \xi_n - \log \xi_i\right)^2 \geq 0 \ . \tag{3.3.37}$$

This proves the global monotonicity of $H(\widehat{\kappa})$ and consequently the uniqueness of the root of $H(\widehat{\kappa}) = 0$.

As in the Beta case, we use Newton-Raphson's method, defined in (3.3.15), to solve (3.3.31), obtaining the following iterative scheme

$$\widehat{\kappa}_{n+1} = \widehat{\kappa}_n - \frac{H(\widehat{\kappa})}{H'(\widehat{\kappa})} \ . \tag{3.3.38}$$

This time the initial value $\widehat{\kappa}_0$ is chosen empirically through experimentation. In our case $\widehat{\kappa}_0 = 1$ seemed to ensure a fast convergence for the datasets used in the experiments. Finally, once a final estimation of $\widehat{\kappa}$ is obtained the only thing left to do is to plug this value into equation (3.3.28) to solve for $\widehat{\lambda}$.

3.4 Uncertainty Intervals

In Section 2.4 we described a direct approach to build error intervals for SVR that was originally proposed in [27]. This method assumes prediction errors to follow a specific probability distribution that is used to define probability intervals for them. If the assumption is true and the underlying noise distribution in the data is accurately estimated, one should expect an increase in the accuracy of the uncertainty intervals.

Here we make a proposal based in the previous method with two enhancements:

1. We adapt the method and formulations proposed in [27], which covered the zero-mean Laplace and Gaussian cases, to the non-zero mean Laplace and Gaussian, as well as to the Beta and Weibull distributions, specially interesting for the problem of wind and solar energy prediction. These new formulations are detailed in Section 3.4.1.

2. A drawback when applying this method is that it assumes the residual distribution to be independent of x and, therefore, probability intervals have exactly the same width for all input samples. However, it is easy to see that in several problems the distribution of the prediction errors may depend on the input x, and therefore the length of the predictive interval with a pre-specified coverage probability may vary from one example to another, reflecting the fact that the prediction variances vary with different input values.

   To lessen the impact of this drawback, in Section 3.4.2 we propose to use clustering methods to split the data into several groups and build different intervals for each one of them.

3.4.1 Error Intervals for Different Distributions

We present now the formulations for the error intervals corresponding to each of the distributions considered in this work, namely Laplace, Gaussian, Beta, and Weibull.

Laplace

In Section 2.4.2.1 we described the formulations of parameters and error intervals for the zero-mean Laplace distribution. Now, if we assume a non zero-mean Laplace distribution, and denote $\psi_i = \hat{f}(x_i) - y_i$, the formulation for l changes into

$$l(\theta; \psi_1, ..., \psi_n) = \sum_{i=1}^{n} \log \frac{1}{2\sigma} e^{-\frac{|\psi_i - \mu|}{\sigma}} \ .$$

For a non-zero mean Laplace distribution the percentile p_s is determined as in the zero mean case by

$$1 - s \ = \ \int_{-\infty}^{p_s} p(z)dz \ . \tag{3.4.1}$$

However, as in this case the distribution is centered at μ and not at zero, the prediction error interval is $(\mu - p_s, \mu + p_s)$.

During the experiments carried out as part of this work, error intervals formulas for Laplace and the other distributions considered have been solved by means of numerical integration.

Gaussian

We described the formulations of parameters and error intervals for the zero-mean Gaussian distribution in Section 2.4.2.2. Now, if we assume a **non zero-mean Gaussian** distribution the formulation for l changes into

$$l(\theta; \psi_1, ..., \psi_n) \ = \ \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(\psi_i - \mu)^2}{2\sigma^2}} \ .$$

The formulation for the prediction error interval for the Gaussian distribution assumption is again the same as the previously defined Laplace counterpart, i.e. $(\mu - p_s, \mu + p_s)$, with p_s as defined in (3.4.1).

Beta

We need to compute the upper sth percentile, p_s as described in 2.4.8. The only difference is that, by definition of the Beta distribution, $z \geq 0$, so in this case we obtain p_s by solving

$$1 - s \ = \ \int_{0}^{p_s} p(z)dz = \frac{\Gamma(\widehat{\alpha} + \widehat{\beta})}{\Gamma(\widehat{\alpha})\Gamma(\widehat{\beta})} \int_{0}^{p_s} z^{\widehat{\alpha}-1}(1-z)^{\widehat{\beta}-1}dz \ . \tag{3.4.2}$$

The prediction error interval is then $(0, p_s)$.

Weibull

As stated before, for the Weibull distribution we only consider the case $z \geq 0$, so we determine the prediction error interval the same way as for the Beta distribution

$$1 - s = \int_0^{p_s} p(z)dz = \int_0^{p_s} \frac{\kappa}{\lambda} \left(\frac{\psi_i}{\lambda}\right)^{\kappa-1} e^{-\left(\frac{\psi_i}{\lambda}\right)^{\kappa}} dz \ . \tag{3.4.3}$$

The prediction error interval is then $(0, p_s)$.

3.4.2 Uncertainty Intervals by Clusters

As explained before, the proposed approach to build uncertainty intervals takes the assumption that the prediction error interval is not directly influenced by the input values, x_i, so the calculated interval is constant for all instances in the dataset. To try to limit the loss of accuracy that this problematic assumption can cause, we propose in this thesis to cluster available data into different groups and apply the proposed technique on each group. This way, different uncertainty intervals will be obtained for each cluster of instances.

We consider that this addition, that we first proposed in [75], is a highly relevant one, as intervals with the same width for each test instance could suppose a critical drawback for data whose distribution strongly depends on variables and entails a strong limitation to the application of these methods to general regression tasks. Figure 3.4.1 and Figure 3.4.2 depict real uncertainty intervals computed for the problem of wind energy production forecasting, showing how this interval computation after clustering will work and the advantages it presents. In particular, it can be observed how the constant uncertainty intervals, presented in Figure 3.4.1, fail to capture the real behaviour of wind energy production in a real-world problen, specially for high production values. In contrast, intervals after clustering, shown in Figure 3.4.2, adapt much better, creating intervals with bigger width for higher energy production values which result in better accuracy.

In particular, two different clustering approaches will be tested in our experiments:

1. The use of *standard clustering* techniques, like K-means or K-prototypes, described in Section 2.7, to group data points based on the input variables.

2. The use of clusters based on the magnitude or scale of the values to predict, i.e. the target. We will call this clustering method, explained later in this section, *magnitude clustering*.

Regarding standard clustering, as mentioned before, we propose to use the popular K-means and its counterpart for datasets mixing numerical and categorical variables, K-prototypes, although we will only made use of the former in our experiments as the datasets employed do not contain categorical variables. Details of these models have already been described in-depth in Section 2.7 as part of the chapter about theoretical background so they will not be discussed here to avoid redundancy.

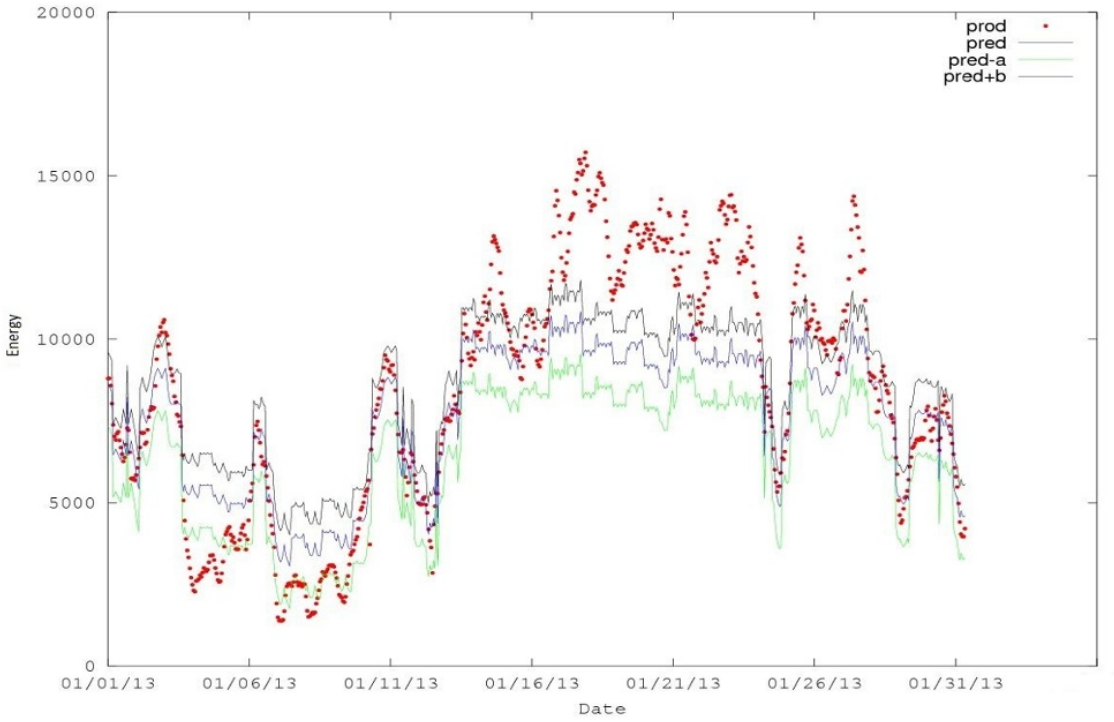


Figure 3.4.1: Constant uncertainty intervals. Figure shows real production, prod, prediction given by the model, pred, lower bound of prediction interval, pred-a, and upper bound, pred+b, where (a,b) is the corresponding uncertainy interval.

With respect to magnitude clustering, we propose a significantly different method to divide our data into groups or clusters. The idea is that the error obtained in a particular instance or observation is often correlated with the magnitude of the target. If this correlation would always be positive and constant, it would be enough to use a scaled version of our target and then apply a constant interval to it to solve this problem. However, this is often not the case.

Solar energy is a good example of this behaviour. When trying to predict solar energy production, in most cases smaller errors are found in the summer days where the target presents its largest values. This is due to the fact that in these summer days the weather behaviour regarding solar radiation is much more stable, and thus more predictable, than in more volatile seasons like autumn. By means of illustration of this phenomenon, Figure 3.4.3 shows the average solar radiation by month for the city of London from 2009 to 2019, where clearly months from May to August have the highest and also most stable irradiation levels.

Furthermore, in the case of solar energy prediction there is also a clear daily pattern depending on the hour, with usually more energy production on noon hours, which could also be taken into account using these uncertainty intervals over clusters built based on magnitude clustering.

Taking this into account, in the case of solar energy production forecast it seems logical to cluster our data based on this magnitude before applying different uncertainty intervals

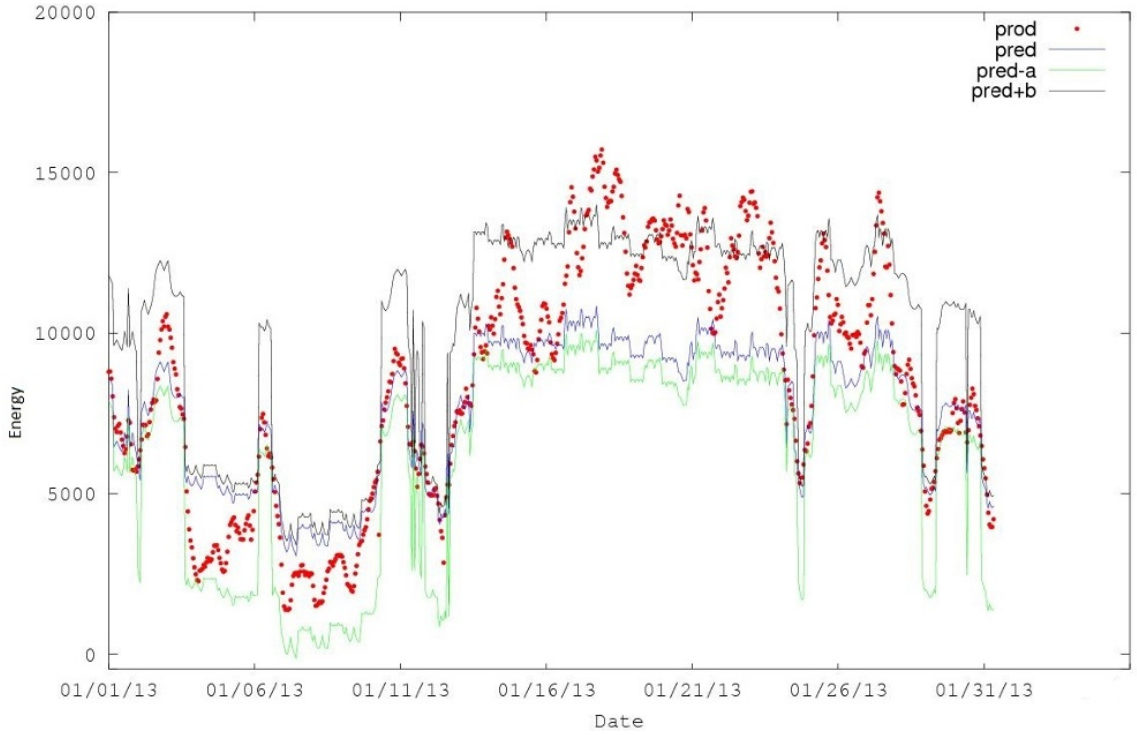


Figure 3.4.2: Error intervals after clustering. Figure shows real production, prod, prediction given by the model, pred, lower bound of prediction interval, pred-a, and upper bound, pred+b, where (a,b) is the corresponding uncertainy interval.

to each scale group. Using this approach and considering by means of illustration the case where the number of clusters, K, is set to 2, one cluster will be built containing the training observations where the target to predict has the lowest values, and the other one including the instances with the highest target values. This way, we could create an interval band for summer days and another one for the rest of the year, being the latter probably wider and hence taking into account the higher probability of bigger solar energy prediction errors on these days. More generally, training observations are sorted based on their target value and then K equal-sized groups of observations are created taking into account this order, so lowest target values are in cluster 1 and highest target values are assigned to cluster K. Then, validation and test observations will be assigned to the cluster, and corresponding error interval, with the closest centroid based on the squared Euclidean distance.

Again, as was the case when using standard clustering, k will be treated as a hyperparameter and its optimal value will be chosen by validation as the value that leads to the computation of the most accurate error intervals. Details of how the train, validation, and test sets are selected are given in Section 4.

3.5 D-GNM with Uncertainty Intervals

In this section we combine all of our previous proposals into one single final framework to compute D-GNM models with uncertainty intervals. The method to compute uncertainty intervals described in [27], which we use as building block for our proposed method in Section 3.4, was originally suggested for the purpose of computing error intervals for

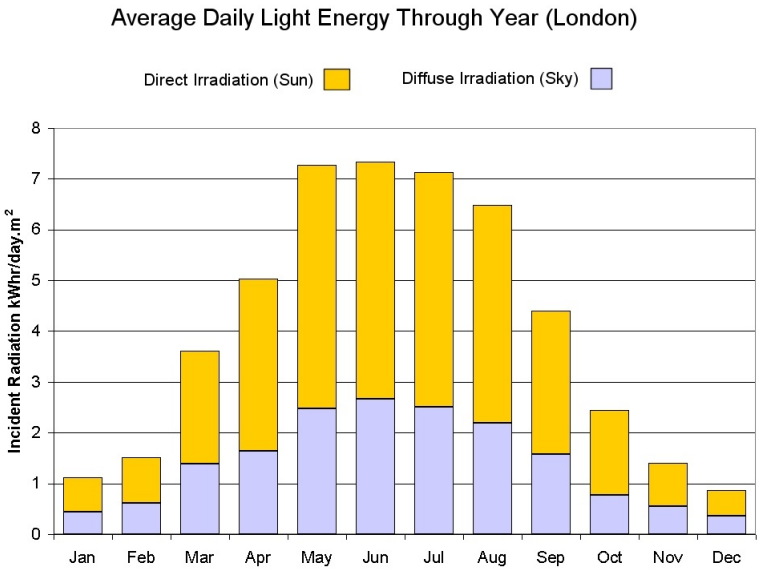


Figure 3.4.3: Seasonal Variation in Solar Energy, London, 2009-2019. From May to August solar radiation levels are higher and more stable. Source: World Irradiation Database.

SVR prediction. However, its nature and mathematical formulations do not hinder the possibility of using it for other regression models. In particular, it seems clear that it fits well with our proposed D-GNMs, as both are based on making assumptions regarding the distribution present in the data noise, and the errors of both uncertainty intervals and D-GNM predictions rely on the correctness of this assumption. For this reason, we propose to combine both methods. These uncertainty intervals could also be combined with our Kernel-GNM models, but we will focus here in their use over D-GNM estimators as they show greatest predictive prowess, as will be seen in Section 4.

In more detail, the goal of this proposed combination is to create a single framework that gathers all the advantages of its individual components, namely:

- The adaptative potential and flexibility of using general cost functions assuming a particular noise distribution in the data. This allows to build ML models with lower prediction errors when the error distribution hypothesis is accurate enough.

- The prediction prowess of DL frameworks, that enable to train regression models whose accuracy has been shown to often be significantly higher than for other ML models when large volumes of data are available for training.

- The advantage of being able to give not only predictions for each instance, but also an associated uncertainty interval for that prediction, which is extremely valuable in several regression applications.

Putting all these pieces together we aim to build ML models for regression able to give more precise predictions and add to this the possibility of also computing accurate error

intervals for each one of these predictions. A good performance of these proposed models would only be possible, of course, when the noise distribution assumed to be present in the data correctly resembles the true underlying noise distribution in the data at hand.

The workflow of the proposed framework to build D-GNM models with uncertainty intervals encompasses the following steps:

Model framework definition

- **Step 1:** Select a Deep Neural Network schema. As in this thesis we focus on standard regression tasks, we will choose to use fully connected versions of Deep ANNs. However, extension of D-GNM to other DL structures like Convolutional Neural Networks is straightforward, as only the loss function applied needs to be modified.

- **Step 2:** Select a noise distribution hypothesis. This step is critical as it will impact in a strong way the accuracy of both predictions and uncertainty intervals of our models. In some cases, this decision could be made based on expert knowledge on the topic or investigation on past research, as is the case for the solar [64] and wind [24] energy problems. In case there is no prior knowledge about which could be a correct noise distribution assumption, this choice could be made defining the noise distribution to be used as an additional hyperparameter to select by grid search over validation.

  In our experiments we will test several distributions. First, the standard ILF cost function used in classical SVR models. Second, the popular and widely applied Laplace and Gaussian distributions, with or without zero mean. And finally, Beta and Weibull distributions, which have been shown [24] [26] to be particularly relevant for our real-world problems at hand, wind energy and solar radiation prediction.

Compute and plug in the optimal cost function

- **Step 3:** Compute the parameters of the optimal cost function corresponding to the distribution selected in Step 2. Details of these formulations are presented in Section 2.3.3 for standard distributions, and in Section 3.1.1 we added to these ones the corresponding formulations for the Beta and Weibull distributions.

- **Step 4:** Plug this cost function into the Deep ANN schema as the loss function, l, in (3.2.1).

Hyperparameter optimization and model training

- **Step 5:** Train the regression model resulting of Step 4, after performing a grid search to obtain optimal hyperparameters [1]. This will give a ML model able to make predictions for a given task that is adapted to a particular choice of noise distribution assumption.

Clustering and uncertainty intervals computation

[1] Details of how the train, validation, and test sets are selected are given in Section 4 as part of the experiments description.

- **Step 6:** Divide the data into clusters following the approaches described in Section 2.7 and 3.4.2, namely standard techniques like K-means and K-prototypes on one hand, and magnitude clustering on the other. Several values for K, the number of clusters, will be used and the corresponding divisions into clusters stored.

- **Step 7:** Compute uncertainty intervals, one for each cluster created in Step 6. This step allows for the computations of error intervals corresponding to the predictions obtained in Step 5. The selection of clusters with the lowest average uncertainy interval error over a validation set [2] will be chosen as the final cluster division.

  These uncertainty intervals will, as the prediction of the model, be fitted to a specific noise distribution hypothesis. Furthermore, they will have non-constant width, due to the previous clustering of the data proposed in this thesis which allows the intervals to adapt to the specific nature of different inputs.

The main contributions of this thesis regarding D-GNM involve the four steps corresponding to the areas of *Compute and plug in the optimal cost function* and *Clustering and uncertainty intervals computation*. In particular, and concerning the previous steps:

- **Step 3:** We have carried out computations in order to give explicit formulations for the optimal loss functions corresponding to the Beta and Weibull distributions.

- **Step 4:** We propose to replace the loss function l used in classical formulations of Deep Neural Network models, usually the MSE for regression, for any choice of general noise loss function to adapt our model to the noise distribution assumed to be present on the problem at hand.

- **Step 6:** We combine clustering methods, both standard ones and others based on target scale, with the method proposed in [27] to compute uncertainty intervals. This allows to solve the problem of some mathematical assumptions leading to constant error intervals not dependant on the input value.

- **Step 7:** We apply the computation of these uncertainty intervals not only to SVR models but also to our proposed D-GNM models. The combination of both methods is feasible in a straightforward manner.

Notice that here we describe the main novelties regarding the final proposed model, D-GNM with uncertainty intervals, due to its special complexity. Nevertheless, intermediate steps and other proposed models like the creation of Kernel-GNMs using NORMA have their own developments that should also be considered as part of this thesis's contributions. A full summary of all these contributions can be found on Section 5.1.

[2] Details of how the train, validation, and test sets are selected are given in Section 4.

Chapter 4

Experiments

> An experiment is a question which science poses to Nature, and a measurement is the recording of Nature's answer.
>
> *Max Planck*

Several experiments have been carried out during the development of this thesis to test the hypotheses formulated and the suitability of the proposed models. We describe in this Chapter all the information necessary to understand these experiments and analyze their results. Furthermore, and in accordance with the principle of reproducible research, we make public all the details regarding the implementation of our proposed models and the datasets that we used to perform our experiments. In particular, we point to several R and Python libraries we created to implement our frameworks and that are accessible from public repositories. Datasets employed have also been made publicly available. Links and references are given in Section 4.1.

The rest of this Chapter is structured as follows: Section 4.1 describes how we implemented the proposed models in this thesis to be able to test them in our experiments. Next, details of how hyperparameter selection was carried out for each type of ML models are presented in Section 4.2. Section 4.3 gives an in-depth description of all the datasets used in the experiments, which comprehend artificial, classical, and real-world datasets, the latter corresponding to solar and wind energy contests. Evaluation metrics used for measurement of the goodness of the proposed methods are explained in Section 4.4. Finally, each one of the final four Sections corresponds to the description of one of the experiments carried out during this thesis, as well as the analysis of the pertinent results. Each experiment tests the usefulness of one of the proposals described in Chapter 3 over all the datasets considered in this work. The summary of these experiments is the following:

1. **Experiment I:** Compare the performance in terms of predictive performance of the proposed Kernel-based General Noise Models, i.e., Kernel-GNM, trained using NORMA with respect to the standard $\epsilon-$SVR model.

2. **Experiment II:** Test the prediction error of the proposed Deep General Noise Models, i.e., Deep-GNM, versus the non-deep Kernel-GNM models, and also against

classical $\epsilon-$SVR model, both deep and kernel versions.

3. **Experiment III:** This last experiment tests the performance of our final proposed framework, Deep-GNM models with uncertainty intervals, in terms of accuracy of error intervals, as experiment II already measures this in terms of prediction error.

4.1 Implementation Details

Several Machine Learning families of models and methods are tested during our experiments. All these methods have been implemented and tested using the two most popular programming languages with regards to Machine Learning or Data Science tasks: R and Python.

Regarding the methods corresponding to Chapter 2, already existing libraries have been selected and applied. As for the proposed methods suggested in Chapter 3, new libraries have been developed in order to be able to test these models in our experiments.

4.1.1 Pre-existing Libraries

The following list comprehends all the already existing libraries used during our experiments:

- **LIBSVM:** Used to train and apply classical $\epsilon-$SVR. Details of this library are described in [18] and its content is publicly available through official repositories for several programming languages.

  For R you can access LIBSVM through the library *e1071* available on the Comprehensive R Archive Network, CRAN [1], and for Python it is integrated in the popular scikit-learn [2] library used for Machine Learning.

- **stats:** $K-$means algorithm is implemented in widely used libraries both for R and Python. We decided to use the R version for this thesis, implemented in the stats library through the kmeans function. This library, pre-included in the basic packages for R, also contains other functions for statistical calculations and random number generation.

- **ncdf4:** Provides a high-level R interface to files written using Unidata's network common data form version 4, netCDF4, as is the case for one of the datasets used in our experiments. Again, publicly available through CRAN.

- **Tensorflow:** Is a free and open-source software library for dataflow and differentiable programming across a range of tasks. In this thesis we use its Python version [3] for Deep Neural Networks implementation. In particular we used Tensorflow 1 in our implementation.

[1] `https://cran.r-project.org/web/packages/available_packages_by_name.html`
[2] `https://scikit-learn.org/stable/index.html`
[3] `https://www.tensorflow.org/install`

- **Keras:** is an open-source neural-network library written in Python. The Keras library [4] works as a high-level API of TensorFlow, providing developers with a more user-friendly layer for the Tensorflow functionality.

4.1.2 Developed Libraries and Functions

Apart from the pre-existing libraries listed in the previous section, we also developed our own R libraries and Python functions and variable classes in order to implement all the methods proposed in Chapter 3. In particular, the following two R libraries and one Python code repository were developed:

- **errint:** Employed to compute and analyze error intervals for a particular model predictions assuming different distributions for noise in the data. It is the corresponding implementation for the uncertainty interval computation method described in Section 3.4. Available on CRAN [5].

- **NORMA:** Used to build general noise kernel-based SVR models by applying NORMA optimization and for the implementation of the Kernel-GNM models detailed in Section 3.1. Also available on the official repository CRAN [6].

- **Deep-GNM:** We created a freely accesible GitHub repository [7] with all the Python code needed to implement our Deep General Noise Models. It uses Tensorflow, Keras, and scikit-learn as base libraries.

4.2 Hyperparameter Selection

All the tested methods have their own set of hyperparameters to be selected. We describe here the algorithms used to perform this hyperparameter optimization for each family of models.

4.2.1 Classical $\epsilon-$SVR

As described in Section 2.1, the standard formulation for kernel-based $\epsilon-$ SVR encompasses three hyperparameters:

1. **Cost, C:** Controls the magnitude of the regularization term or, in other words, the bias-variance tradeoff.

2. **epsilon, ϵ:** Selects the range of the insensitive band for the ILF cost functions. Errors with absolute magnitude below this ϵ value will not be penalized.

3. **gamma, γ:** Hyperparameter used to compute the Gaussian kernel defined by equation (2.1.44), which is the kernel used in our experiments, both for classical and general noise versions of kernel SVR formulations.

[4] `https://keras.io/`
[5] `https://cran.rstudio.com/web/packages/errint/index.html`
[6] `https://cran.r-project.org/web/packages/NORMA/index.html`
[7] `https://github.com/jesuspradaalonso/phd`

Algorithm 2: Zoom in Exhaustive Grid Search.

Set $P_0 = (C_0, \epsilon_0, \gamma_0)$ as the parameters obtained after using the *Cherkassky's approach* [77].
Define maximum number of zooms, n_{zooms}.
for $i = 1$ *to* n_{zooms} **do**
  Set number of points as $N = 2(i-1)+3$.
  Select as grid range $[P_0 \frac{10^i}{10^{n_{zooms}+1}}, P_0 \frac{10^{n_{zooms}+1}}{10^i}]$.
  Select N equidistant points from the previous grid range.
  Train model with each one of these points and apply models over validation.
  Update P_0 to be the grid point with the lowest validation error.
end
Set optimal hyperparameters as P_0

In our experiments these three hyperparameters are optimized by a zoom in version of the exhaustive grid search over a fixed validation set, as defined in Algorithm 2, where we decided to select as number of zooms $n_{zooms} = 3$. We decided to use a fixed validation set, instead of a cross-validation setting, because it had yielded better results in some of our previous experiments regarding wind and solar energy [76] [68], probably because it is most suited when working with datasets with strong seasonality and temporal structure. Although we could have employed cross-validation for artificial and classical datasets, we decided to unify the framework and apply a fixed validation set for all experiments. Train, validation, and test splits are defined for each dataset in Section 4.3.

In [77], Cherkassky recommends to use a set of equations to compute the optimal values for these three SVR hyperparameters. From our experience, this should be considered only a recommendation and applying a grid is strongly advisable. Therefore, we have opted to use these recommended Cherkassky's values as the initial center of our grid for the first iteration in Algorithm 2. The formulations suggested by Cherkassky are the following ones

$$\begin{aligned} C &= \max\left(|\bar{y} - 3\sigma_y|, |\bar{y} + 3\sigma_y|\right) \ , \\ \epsilon &= 3\sigma\sqrt{\frac{\log n}{n}} \ , \\ \gamma^d &\in (0.2, 0.5) \ , \end{aligned} \tag{4.2.1}$$

where d is the number of variables in the dataset, also referred to as its *dimensionality*. The Cherkassky's proposal for γ is subject to data being scaled to [0,1].

4.2.2 Kernel Gaussian Noise Models, Kernel-GNM

As stated in Section 3.3, for general noise models using loss functions other than ILF, the density parameters are computed applying the Maximum Likelihood Estimation, MLE, formulas shown in Table 4.2.1, which in some cases involve solving numerically the equations over a set of residuals.

The residuals employed for the calculation of these MLE formulas, e.g. α and β estimations for the Beta distribution, are the corresponding prediction errors obtained over cross-validation by a previously computed $\epsilon-$insensitive SVR model, which is optimized

Table 4.2.1: Estimated parameters via MLE corresponding to several distributions.

Distribution	**MLE parameters**
Zero-mean Laplace	$\hat{\sigma} = \frac{\sum_{i=1}^{n} \lvert\psi_i\rvert}{n}$
Laplace	$\hat{\sigma} = \frac{\sum_{i=1}^{n} \lvert\psi_i - \mu\rvert}{n}$, $\mu = m_{\psi_i}$. m_{ψ_i} is the median of $\{\psi_i\}_{i=1}^{n}$
Zero-mean Gaussian	$\hat{\sigma} = \frac{\sum_{i=1}^{n} \psi_i^2}{n}$
Gaussian	$\hat{\sigma} = \frac{\sum_{i=1}^{n} (\psi_i - \mu)^2}{n}$, $\mu = \sum_{i=1}^{n} \frac{\psi_i}{n}$
Beta	$\begin{bmatrix} \widehat{\alpha}_{n+1} \\ \widehat{\beta}_{n+1} \end{bmatrix} = \begin{bmatrix} \widehat{\alpha}_{n} \\ \widehat{\beta}_{n} \end{bmatrix} - [J_f(\widehat{\alpha}_n, \widehat{\beta}_n)]^{-1} \begin{bmatrix} F_1(\widehat{\alpha}_n, \widehat{\beta}_n, \xi_i) \\ F_2(\widehat{\alpha}_n, \widehat{\beta}_n, \xi_i) \end{bmatrix}$
Weibull	$\lambda = \left(\frac{1}{n} \sum_{i=1}^{n} \psi_i^{\kappa}\right)^{\frac{1}{\kappa}}$, $\widehat{\kappa}_{n+1} = \widehat{\kappa}_n - \frac{H(\widehat{\kappa})}{H'(\widehat{\kappa})}$

using 5-fold cross-validation following the hyperparameter selection process described in Section 4.2.1. Although theoretically it could be argued that it would be more correct to apply this methodology employing a nonparametric method that does not make asumptions regarding noise distribution on the data, like Random Forest, we opted here to use the most generic or neutral of the models we consider in our analysis, which is the classical $\epsilon-$insensitive SVR model.

Regarding Kernel-GNMs, as in the case of the classical $\epsilon-$SVR model, we select the Gaussian kernel as the one to employ and we estimate its optimal γ value using the same approach as in Algorithm 2 over a fixed validation set.

As for the stopping criteria parameters, we select 1000 as the maximum number of NORMA update iterations and 10^{-3} as the minimum tolerance threshold, i.e., the minimum relative difference between the $\widehat{\alpha}_t^t$ obtained in two consecutive iterations required to continue with the NORMA iterative updates.

4.2.3 Deep ANN

Deep Neural Networks are Machine Learning models with quite a long list of possible hyperparameters to optimize. From all the parameters we decided to optimize the ones that seemed more important judging by the literature and our previous experiments. In particular, these are the hyperparameters we selected:

1. **Number of hidden layers:** This hyperparameter selects the number of hidden layers, i.e., the ones between the input and output layer, that the Deep ANN setting will have.

2. **Number of neurons per layer:** We set the number of neurons or units to be constant among all hidden layers for computational reasons, but the specific value set for this hyperparameter is optimized in our grid search.

Algorithm 3: Zoom in Random Grid Search.

1 For each hyperparameter h to optimize in the grid search, set the initial minimum value, min_h, and the initial maximum value, max_h, according to Table 4.2.2.
2 **for** *i = 1 to* n_{zooms} **do**
3 Set number of points as $N = 2^i$.
4 For each hyperparameter h, select randomly N valid points in the interval $[min_h, max_h]$.
5 Train model with each one of these points and apply models over validation.
6 Select P as the grid point with the lowest validation error.
7 For each hyperparameter, update minimum and maximum values as $min_h = P - \frac{max_h - min_h}{2^{i+1}}$ and $max_h = P + \frac{max_h - min_h}{2^{i+1}}$, respectively.
end
8 Set optimal hyperparameters as P.

3. **Dropout probability:** Dropout refers to ignoring neurons chosen at random during the training phase of our model, i.e., these units are not considered during a particular forward pass. Individual nodes are either dropped out of the net with probability $1 - p$ or kept with probability p. This p value is the one we optimize in our grid search. This dropout mechanism allows us to do regularization of our Deep ANN models. Although dropout and weight decay are regularization techniques that are not mutually exclusive, we have decided to not apply weight decay in this work and carry out regularization by means of dropout, in order to reduce computational complexity of the hyperparameter tuning of these models.

4. **Batch size:** This one is a hyperparameter of Adam optimization that controls the number of training samples to work through before the model's weights are updated.

5. **Learning Rate:** This is the η_t value appearing in equation 2.6.19. It controls the step size of each updating in the Adam optimization. For computation reasons, we decide again to make this hyperparameter constant, i.e $\eta_t = \eta$.

Due to the computational expense of training an exhaustive grid search of Deep ANN models with combinations of all these hyperparameters, we opted here to apply a zoom-in random grid search [78] over a fixed validation set, as we did in [79] with good results, instead of the zoom in exhaustive version shown in Algorithm 2. A visual comparison between these two grid search methods can be seen in Figure 4.2.1. The initial random grid search space for each hyperparameter is specified in Table 4.2.2 and the zoom-in random grid search algorithm is described in Algorithm 3. In our experiments we used $n_{zooms} = 6$.

4.2.4 Deep General Noise Models, Deep-GNM

The proposed Deep General Noise Models are a combination of a Deep ANN with the use of a general cost function. Therefore, it has two different set of hyperparameters to optimize:

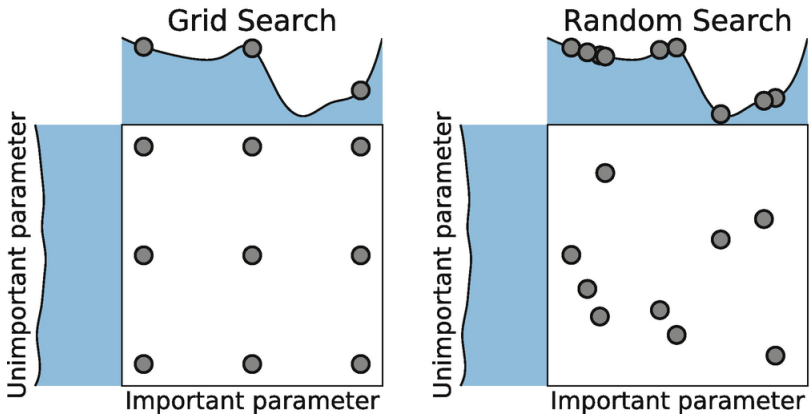


Figure 4.2.1: Exhaustive Grid Search vs Random Grid Search. Exhaustive method explicitly defines the values to test for a particular hyperparameter, while Random Grid Search test a set of N points randomly selected among a specified range. Image from [80].

1. **Deep ANN hyperparameters:** Optimized using the zoom-in random grid search shown in Algorithm 3 over the hyperparameters described in Section 4.2.3.

2. **General cost function density parameters:** For Deep-GNM models using loss functions other than ILF, the density parameters are computed applying the MLE formulas shown in Table 4.2.1, as described in Section 4.2.2 for Kernel-GNM models. When using ILF, hyperparameter ϵ is optimized by a zoom in version of the exhaustive grid search, as defined in Algorithm 2.

4.2.5 Uncertainty Intervals

To compute error intervals following the method proposed in Section 3.4, it is necessary to obtain the density parameters corresponding to the noise distribution assumed to be present in the data.

In order to get these density parameters, we apply the MLE formulas shown in Table 4.2.1 in the same vein we do to compute the proposed Kernel-GNM models. However, instead of using the residuals obtained in the hyperparameter grid search of classical $\epsilon-$SVR described in Algorithm 2, as we proposed for Kernel-GNM models, in this case the validation errors of the corresponding optimum general noise model calculated previously, i.e., the one yielding the best results, are used as residuals for MLE computation.

Finally, when clustering is carried out prior to computation of the uncertainty intervals, the K value is selected by means of an exhaustive grid search, where values between 2 and 10 are evaluated. The K value that results in the error intervals with better performance in terms of per_err over 5-fold cross-validation is selected as optimal value.

Table 4.2.2: Initial random grid search space for Deep ANN models.

Hyperparameter	Min	Max
Number of hidden layers	1	10
Number of neurons per layer	10	100
Dropout probability	0	0.5
Batch size	8	512
Learning Rate	10^{-5}	10^{-1}

4.3 Datasets

4.3.1 Artificial Datasets

We created several artificial datasets consisting of $500,000$ instances following this expression

$$y_i \quad = \quad x_i^a \cdot b + \delta_i, \;\; i = 1, 2, \ldots, 500,000 \;, \tag{4.3.1}$$

with x_i, a and b 1024-dimensional vectors where each element is randomly chosen from the uniform distributions over the intervals $[0.1, 2]$, $[1, 5]$, and $[1, 10]$ respectively, and δ_i random noise following different distributions; 70% of each dataset is used for training, 15% for validation, and 15% for testing.

We consider here five different types of datasets, each one with a different noise distribution. Distribution parameters such as σ in the Laplace noise are randomly computed in the selected interval only once for each dataset, i.e., they remain constant for all δ_i extractions of a particular dataset and hence the noise for each instance of the dataset follows exactly the same distribution, although the specific noise value will differ between instances. Following a similar approach to the one we used in [76], the concrete distributions applied to each dataset are the following:

1. **Zero-noise:** $\delta_i = 0$. This is a dataset without noise following a specific distribution, so we may expect that no clear winner between the proposed Laplace, Gaussian, Beta, and Weibull noise models is found.

2. **Laplace:** δ_i extracted from a Laplace with $|\mu| \in [\overline{y}, \max(y)]$, $\sigma \in [\text{std}(y), \max(y)]$.

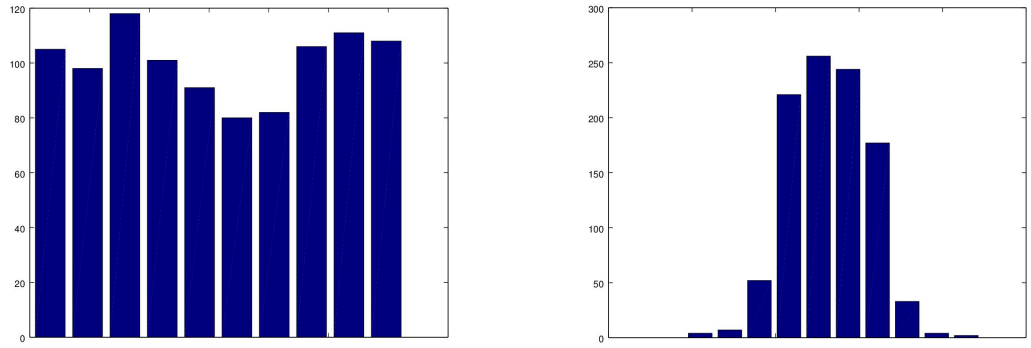
3. **Gauss:** δ_i extracted from a Gaussian with $|\mu| \in [\overline{y}, \max(y)]$, $\sigma^2 \in [\text{std}(y)^2, \max(y)^2]$.

4. **Beta:** δ_i extracted from a Beta distribution with $\alpha, \beta \in [2, 10]$ [8].

5. **Weibull:** δ_i extracted from a Weibull distribution with $\kappa, \lambda \in [1, 10]$[9].

[8] Notice here that Beta loss function is convex when $\alpha, \beta > 1$

[9] Notice here that Weibull loss function is convex when $\kappa > 1$



(a) Test errors for data without noise. (b) Test errors for data with Laplace noise.

Figure 4.3.1: Histograms of test errors for data without noise vs. test errors for data corrupted with Laplace noise.

Three datasets are built for each of these types using the same distribution parameters for y_i in (4.3.1) but different distribution parameters for the noise δ_i inserted, and the mean of the experiment results over them is computed to contemplate possible deviations on results obtained.

As an example, histograms of errors for a random forest model [7] fitted to the zero-noise artificial test set and the one corrupted with Laplace noise are shown in Figure 4.3.1. Random forest models are used here due to the fact that it is a nonparametric method that does not make asumptions regarding noise distribution on the data. It is clear that a Laplace distribution fits better the latter histogram and, therefore, it is to be expected that the intervals corresponding to this distribution are the ones that achieve greatest accuracy, and probably a Kernel-GNM or Deep-GNM using this distribution hypothesis would also yield better predictions.

4.3.2 Classical Datasets

We also use in our experiments a set of widely used regression datasets belonging to the LIBSVM repository. In particular, the list of datasets evaluated is the following: *abalone, bodyfat, cpusmall, housing, mg, mpg, pyrim*, and *space_ga*. All these datasets are publicly accesible through the LIBSVM repository [10]. Their sample sizes and number of features are given in Table 4.3.1.

All the classical datasets considered are split into train, validation, and test sets using 50% for train, 15% for validation, and 35% for test. We decided to use a bigger ratio for the test set than usual, due to the reduced volume of some of the datasets and the need of further splitting the test set in a second set of train and test divisions in order to build the clustering methods we employ to compute error intervals in Experiment III, as will be described in Section 4.7.

4.3.3 Solar Dataset

The dataset analyzed regarding solar tasks corresponds to the Kaggle AMS 2013-2014 solar radiation prediction contest. The goal of this contest is to discover which statistical and machine learning models provide the best predictions of daily–aggregated solar

[10] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

Table 4.3.1: Classical datasets dimensions.

Dataset	Patterns	Features
abalone	4177	8
bodyfat	252	14
cpusmall	8192	12
housing	506	13
mg	1385	6
mpg	392	7
pyrim	74	27
space_ga	3107	6

radiation. In particular, models must predict the total daily incoming solar radiation at 98 Oklahoma mesonet sites [81], which will serve as solar farms proxies for the contest. Mesonet sites are a network of collectively owned and operated automated weather stations that are installed close enough to each other to observe, measure, and track mesoscale meteorological phenomena.

Real values of total daily incoming solar radiation in Jm^{-2} at these 98 points are provided in the AMS dataset, although we will work with KJm^{-2} values instead for clarity. Location coordinates and elevation for each station are also given.

Input numerical weather prediction data for the contest comes from the NOAA/ESRL Global Ensemble Forecast System [82], GEFS, Reforecast Version 2 [11]. The data are in netCDF4 files; each one contains the total data for one of the model variables and is stored in a multidimensional array. The first dimension is the date of the model run. The second dimension is the ensemble member which the forecast comes from. The GEFS has 11 ensemble members for which the GFS model is applied with perturbed initial conditions. We use only ensemble 1 in our experiments for simplicity. The third dimension is the forecast hour, which runs from 12 noon to 12 midnight UTC in 3 hour increments, so rows for different days will always correspond to the same universal time although local solar time will vary over each year. The fourth and fifth dimensions are the latitude and longitude on a uniform spatial grid. The longitudes in the file are in positive degrees from the Prime Meridian, so subtracting 360 from them will translate them to a similar range of values as the ones given for the stations, which are provided in a separate file together with their corresponding elevation. The comprehensive list of all variables includen in these files is shown in Table B.0.1 in Appendix B. Elevation of each GEFS point is also provided in an additional file.

Data of the contest covers the years from 1994 to 2007. For the purpose of our experiments, we split this dataset into train, validation and test as follows:

- **train:** 1994-2005.

[11] `https://psl.noaa.gov/forecasts/reforecast2/`

- **validation:** 2006.

- **test:** 2007.

The complete dataset is freely accesible at `https://www.kaggle.com/c/ams-2014-solar-energy-prediction-contest/data`

4.3.4 Wind Dataset

The wind energy related dataset employed in the experiments is the one used in the GEFCom2014 contest [83], where the probabilistic wind power forecasting track aims to estimate the probabilistic distribution, in quantiles, of wind power generation for 10 wind farms. The wind farms are all located in Australia, and there could be potential dependencies between the sites, both in space and in time.

The target variable is the power generation, normalized here by the respective nominal capacities of each wind farm. The predictors included are wind forecasts at two heights, 10 and 100 meters above ground level, obtained from the European Centre for Medium-range Weather Forecasts, ECMWF [12]. These forecasts were for the zonal and meridional wind components, denoted U and V respectively. The meteorological convention for winds is that U component is positive for a west to east flow, i.e., eastward wind, and the V component is positive for south to north flow, i.e., northward wind. Therefore, the complete list of 4 variables available is the following.

1. **VAR 1:** 10 metre U wind component in m s^{-1}.

2. **VAR 2:** 10 metre V wind component in m s^{-1}.

3. **VAR 3:** 100 metre U wind component in m s^{-1}.

4. **VAR 4:** 100 metre V wind components in m s^{-1}.

Vector modules for $W = (U, V)$, i.e., $\sqrt{U^2 + V^2}$, both for 10 and 100 metres, were also computed and added to the input dataset.

Data is provided in comma separated values with each row corresponding to one hour of a particular day. The dataset includes 15 different tracks, but we will focus only in track 15 for the purpose of these experiments. Data available goes from 2012-04-01 to 2014-07-01. We split the data using the following approach:

- **train:** From 2012-06-01 to 2013-05-31.

- **validation:** From 2013-06-01 to 2014-05-31.

[12] `https://www.ecmwf.int/`

- **test:** From 2014-06-01 to 2014-07-01.

The complete dataset is accesible via [83].

4.4 Evaluation Metrics

In order to measure the suitability of the methods proposed in Section 3 during our experiments, we make use of several evaluation metrics, with different intuitions and purposes.

4.4.1 Prediction Evaluation

For purposes of comparing the results obtained with the ones present in previous research and in the Kaggle Leaderboard, the primary choice of evaluation metric for our experiments is the Mean Absolute Error, MAE. This is the main metric we will use throughout this chapter to analyze prediction performance. As we have seen earlier, the MAE is defined as

$$MAE = \frac{1}{N}\sum_{i=1}^{N} \mid \hat{f}(x_i) - y_i \mid \tag{4.4.1}$$

However, based on our experience in solar and wind energy tasks we consider that the Mean Absolute Percentage Error, *MAPE*, may be a better choice to evaluate performance of a model in this particular task. For this reason, we will also include in this chapter results of experiments applying the more demanding MAPE metric, which is defined as follows

$$MAPE = 100 * \frac{\sum_{i=1}^{N} \frac{|\hat{f}(x_i) - y_i|}{max(|y_i|, |\delta_\epsilon|)}}{N} \ . \tag{4.4.2}$$

where δ_ϵ is set to be equal to the minimum value of $|y_i|$ that is not zero over the evaluation set.

Finally, in the GEFCom2014 competition the goal is to find the best quantile predictions for wind power generation. Therefore, an evaluation metric suited to this purpose must be used. They opt to use the pinball loss function to evaluate the accuracy of these probabilistic forecasts. For comparison with the GEFCom leaderboard for this contest, prediction accuracy using this metric is also included. This specific metric is defined as follows.

$$L_\tau(y, z) = \begin{cases} (y - z)\tau \ , \ y \geq z \ , \\ (z - y)(1 - \tau) \ , \ y < z \ , \end{cases} \tag{4.4.3}$$

where τ is the target quantile, z the predicted quantile value and y the exact numerical value of wind power. More details regarding the definition of this metric and why it was chosen for this contest can be found in [83]. A visualization of the pinball loss function can be found in Figure 4.4.1, where $\xi = y - z$.

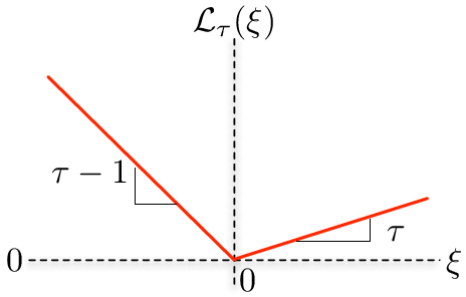


Figure 4.4.1: Pinball Loss Function.

4.4.2 Uncertainty Intervals Evaluation

Regarding evaluation of the uncertainty interval accuracy, given a pre-specified probability $1-2s$, where s is the s-th percentile for which we have computed our error intervals, we compare the percentage of test prediction errors, ψ_i^{test}, lying inside the corresponding uncertainty intervals, $[a_s, b_s]$, with the expected number, $(1-2s)$, i.e.,

$$per_{err}(s) = \frac{\{\#\ of\ \psi_i^{test} \in [a,b]\}}{N} - (1-2s) \quad . \tag{4.4.4}$$

We choose an absolute error as accuracy measure over one with weights for positive or negative errors because preference towards a positive or negative error, i.e., which one is considered less detrimental of the two, is problem-dependent. In some tasks it is preferable to take a more conservative approach, penalizing more negative errors, but in others a more risky approach could be a better option, tending to punish positive errors more. Here we opt to use the most neutral possible choice as our measure.

4.5 Experiment I. Kernel-GNM Models

The purpose of this experiment is to test the performance of classical ϵ-SVR versus our proposed general noise SVR models, Kernel-GNM. In particular, we build general models following the approach proposed in Section 3.1 using Laplace, Gaussian, Beta, and Weibull distributions as noise assumptions. We make this comparison over all the datasets considered, namely artificial, classical and the two competition datasets.

Results are presented in Tables 4.5.3, 4.5.5, 4.5.6, 4.5.8, 4.5.9, 4.5.11, using MAE, MAPE, and Pinball loss as evaluation metrics. For each of these metrics, evaluation scores and relative rank for each dataset are shown. For MAE and MAPE average metrics and ranks are also provided, although it is important to note that average MAE over datasets can be misleading if not analyzed carefully, due to the possibility of significant differences in target magnitudes among datasets. These tables show the following columns:

1. **Dataset:** Reference to the dataset to which metric results of that row of the table corresponds.

Table 4.5.1: Structure of matrix residuals employed in Friedman and Wilcoxon tests.

Observation	Model 1	Model 2	Model 3	Model 4	Model 5
1	ψ_1^1	ψ_1^2	ψ_1^3	ψ_1^4	ψ_1^5
2	ψ_2^1	ψ_2^2	ψ_2^3	ψ_2^4	ψ_2^5
3	ψ_3^1	ψ_3^2	ψ_3^3	ψ_3^4	ψ_3^5
4	ψ_4^1	ψ_4^2	ψ_4^3	ψ_4^4	ψ_4^5
5	ψ_5^1	ψ_5^2	ψ_5^3	ψ_5^4	ψ_5^5
...	...	...	...	...	...
N	ψ_N^1	ψ_N^2	ψ_N^3	ψ_N^4	ψ_N^5

2. **ϵ-ILF:** Classical ϵ-SVR model, as described in Section 2.2.1.

3. **Lap:** Kernel-GNM model proposed in Section 3.1 assuming Laplace distribution allowing for non-zero mean values.

4. **Gau:** Kernel-GNM model proposed in Section 3.1 assuming Gaussian distribution allowing for non-zero mean values.

5. **Beta:** Kernel-GNM model proposed in Section 3.1 assuming Beta distribution.

6. **Weib:** Kernel-GNM model proposed in Section 3.1 assuming Weibull distribution.

For each dataset, we compute Kernel-GNM models following each one of the four distributions considered in this thesis, and show the corresponding results. In a real case scenario, this process could be considered as an aditional step of the hyperparameter selection, where the particular distribution to be employed as noise distribution hypothesis could be considered as an additional hyperparameter and selected by grid search over validation. Additionally, for some tasks like the solar and wind datasets, expert knowledge on the topic or investigation on past research [64] [24] could point us to potential good choices of noise distribution assumption.

Hyperparameters for *ϵ-ILF* as well as the distribution parameters for *Lap*, *Gau*, *Beta*, and *Weib* are selected following the approaches described in Section 4.2.

Furthermore, we carried out for each experiment a Friedman test over the residuals, $\psi_i^m = \widehat{f^m}(x_i) - y_i$, obtained in the test set by each model m to check for significant differences between the performance obtained between the different models analyzed. When a significant p-value was found after applying this test, we also carried out a pairwise comparison with Wilcoxon signed-rank test over the residuals ψ_i^m to analyze if there exists significant differences between the results obtained by each tested model, where p-values were adjusted by a step-down method using Sidak adjustments [84]. Therefore, we apply the aforementioned statistical tests over a matrix with the structure shown in Table 4.5.1, where in this experiment the five models considered are *ϵ-ILF*, *Lap*, *Gau*, *Beta*, and *Weib*.

4.5.1 Artificial Datasets

Table 4.5.3 shows the MAE for Kernel-GNM models using different noise distribution assumptions over our artificial datasets. For all the artificial datasets used in this experiment, when the target presents noise corresponding to a particular distribution, the best general noise model is the one that uses the loss function matching the same noise distribution and, in particular, its results are clearly better than the ones obtained using the classical ϵ-ILF. These results seem to confirm our initial hypothesis and the usefulness of the approach proposed in this paper to build general noise models, and supports the validity of our proposed method to select the density parameters, which is described in Section 3.3.

It is also interesting to note that for the artificial dataset with Gaussian noise, the second best performing one corresponds to the model using ϵ-ILF as loss function, a result that fits in well with the statement made in [23] that the use of the ϵ-ILF is also justified under the assumption that the noise is additive and Gaussian, where the variance and mean of the Gaussian are random variables.

We carried out a Friedman test over the ψ_i^m matrix to analyze if there exists significant differences between the residuals obtained by each tested general noise model. The p-value obtained is 0.20, clearly above the threshold for a significance level of 0.05. This is a result to be expected, as each general noise model is the best solution when the underlying noise in the dataset corresponds to the distribution assumption employed to build the model, but no overall best or worst model exist.

Taking this into account, we decided to carry out statistical tests for each artificial dataset independently. Thus, for each one of these datasets a different Friedman test was conducted over the matrix of residuals obtained only for that particular artificial dataset. When this Friedman test for a particular dataset showed significant differences, a pairwise Wilcoxon test was also applied to this matrix. This means that a matrix of residuals following the structure shown in Table 4.5.1 was built using only residuals obtained by each model for observations in the Zero-noise dataset, i.e., only a subset of rows from the n observations shown in Table 4.5.1 are selected, and then the statistical tests were applied over this reduced matrix. Next, a new reduced matrix with the same structure but containing only residuals for observations in the Laplace dataset, i.e., a different subset of rows from Table 4.5.1, was created and statistical tests carried out. Finally, this process was repeated in the same manner for all the remaining artificial datasets.

Following this methodology, the Friedman tests showed a p-value below 0.05 for all datasets except for the *Zero-noise* one, where it is to be expected that differences between models are not necessarily significant. For all the other datasets, we decided to carry out a pairwise Wilcoxon test comparing the reference model, i.e., the one using as noise assumption the same distribution employed to build the dataset, with all the other models. This means that from the results obtained from the pairwise Wilcoxon test over, for instance, the *Laplace* dataset, we focus only on the p-values obtained for the row corresponding to the *Lap* model, as shown in Table 4.5.2.

Results of the Friedman and pairwise Wilcoxon tests described above are shown in Table 4.5.4. It can be observed that there are significant differences between the performance of

Table 4.5.2: Pairwise Wilcoxon test results corresponding to the reference model.

	ϵ-ILF	**Lap**	**Gau**	**Beta**	**Weib**
ϵ-ILF	...	...	...	...	...
Lap	**0.03**	1.00	**0.03**	**0.00**	**0.01**
Gau	...	...	...	...	...
Beta	...	...	...	...	...
Weib	...	...	...	...	...

Table 4.5.3: MAE obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over artificial datasets.

Dataset	**ϵ-ILF**	**Lap**	**Gau**	**Beta**	**Weib**
Zero-noise	0.99 (2)	**0.98 (1)**	0.99 (2)	1.04 (5)	1.02 (4)
Laplace	2.07 (2)	**1.86 (1)**	2.11 (3)	2.57 (5)	2.34 (4)
Gaussian	1.97 (2)	2.23 (3)	**1.91 (1)**	2.61 (5)	2.25 (4)
Beta	1.16 (2)	1.23 (4)	1.20 (3)	**1.06 (1)**	1.27 (5)
Weibull	1.58 (3)	1.66 (4)	1.53 (2)	1.88 (5)	**1.40 (1)**
mean	1.55	1.59	**1.52**	1.83	1.66
mean rank	**2.2**	2.6	**2.2**	4.2	3.6

the reference model and all the others, with the exception of the Gaussian and standard ϵ-ILF for the dataset build using Gaussian noise, which is again a conclusion that fits in well with the statement made in [23]. For the zero-noise dataset, the Friedman test did not show significant differences between the residuals obtained by each tested general noise model, as was to be expected.

Results are similar for the MAPE metric, which are presented in Table 4.5.5. The only significant difference is that for MAPE the winner in terms of average error is ϵ-ILF while for MAE it was the Gaussian loss, with the two of them tied regarding mean rank for both MAE and MAPE metrics. This is probably due to the effect of having targets with different scales affecting the MAE results. Therefore, it seems logical to conclude that ϵ-ILF loss is the most robust option when the underlying noise in the data is not known or cannot be deduced from the data itself, which fits well with the fact that it is the standard choice when computing a SVR model.

Table 4.5.4: Friedman and Wilcoxon tests for Experiment I results over artificial datasets.

Dataset	Friedman p-value	Reference	ϵ-ILF	Lap	Gau	Beta	Weib
Zero-noise	0.37	-	-	-	-	-	-
Laplace	**0.01**	Lap	**0.03**	1.00	**0.03**	**0.00**	**0.01**
Gaussian	**0.04**	Gau	0.27	**0.03**	1.00	**0.00**	**0.03**
Beta	$1.85e^{-29}$	Beta	**0.02**	**0.01**	**0.02**	1.00	**0.00**
Weibull	$2.73e^{-7}$	Weib	**0.03**	**0.01**	**0.03**	**0.00**	1.00

Table 4.5.5: MAPE obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over artificial datasets.

Dataset	ϵ-ILF	Lap	Gau	Beta	Weib
Zero-noise	**7.7 (1)**	**7.7 (1)**	7.8 (3)	8.2 (5)	8.0 (4)
Laplace	16.3 (2)	**14.6 (1)**	16.6 (3)	20.2 (5)	18.4 (4)
Gaussian	15.1 (2)	17.5 (3)	**14.6 (1)**	20.5 (5)	17.7 (4)
Beta	8.2 (2)	9.7 (4)	9.4 (3)	**7.8 (1)**	10.0 (5)
Weibull	12.1 (3)	13.0 (4)	12.0 (2)	14.8 (5)	**11.0 (1)**
mean	**11.9**	12.5	12.1	14.3	13.0
mean rank	**2.0**	2.6	**2.0**	4.2	3.6

4.5.2 Classical Datasets

In Table 4.5.6 MAE results are shown for each of the LIBSVM datasets evaluated and five types of Kernel-GNM models used, namely ϵ-*ILF*, *Laplace*, *Gaussian*, *Beta*, and *Weibull*, where hyperparameters and density parameters are selected as described in Section 4.2. Several relevant conclusions can be obtained from these results. First, Kernel-GNM models are competitive with classical SVR in general, and for some datasets even better. Although standard SVR gets the best average MAE among all the datasets, this cannot be considered too relevant due to big differences in target scale among the datasets. As a matter of fact, although classical ϵ-SVR is the model that achieves the best performance for four out of eight datasets, Gaussian Kernel-GNM, is the winner in terms of average rank.

Secondly, there is not a Kernel-GNM model that outperforms ϵ–insensitive SVR over all datasets evaluated, but some of these models are clearly better for problems like *cpusmall*, *space_mpg*, and *housing*, where *Lap*, *Gau*, and *Weib* models appear to be the best option, respectively. This fact seems to point to a particular distribution of noise for these problems outside the one corresponding to the ϵ-ILF function. On the other hand, the model assuming *Beta* noise obtains the worst overall results and does not manage to be the best option for any of the classical datasets analyzed in this experiment.

Finally, when Kernel-GNM models are the best option, the particular distribution assumption that yields the best performance is not always the same, with Laplace, Gaussian and Weibull being the best suited option for at least one dataset. This indicates the importance of being accurate when formulating the hypothesis of the noise distribution assumed to be present in the data and thus inserted in the formulation of the Kernel-GNM model to be used for a particular task and, therefore, points out that using ϵ-ILF regardless of the specific nature of the task at hand is not always advisable, although it remains to be a competitive option.

Applying a Friedman test over the residuals obtained for all the datasets following the structure described in Table 4.5.1 yields a p-value of 0.01, showing significant differences. Therefore, in this case we did not carry out a separate Friedman and Wilcoxon tests over the residuals obtained for each dataset, as we did for the artificial datasets, but only an overall test taking into account the residuals obtained for observations belonging to all classical datasets, i.e., we built a single matrix with the structure shown in Table 4.5.1, where the residuals obtained for *bodyfat* observations are placed below the residuals corresponding to *abalone* observations, and so on and so forth. Then we applied a Friedman test, resulting in the aforementioned p-value of 0.01, and a pairwise Wilcoxon test over that matrix of residuals.

Regarding pairwise Wilcoxon tests for residuals obtained over all classical datasets, which results are shown in Table 4.5.7, significant differences using a significance level of 0.05 are detected for *Beta* with respect to classical *ϵ-ILF* and *Gaussian* models, and also between *Beta* and *Weibull* if a significance level of 0.1 is applied. Looking at the results obtained in Table 4.5.6 this seems reasonable, as here *Beta* appear to have a worse overall performance in general, wirh a mean rank of 4.5 out of 5.

Analogous results for the MAPE metric are shown in Table 4.5.8. The main conclussions described before can also be drawn here looking at the MAPE scores, although some differences exist when analyzing particular datasets. For instance, Laplace noise distribution assumption, and not Weibull like in the MAE case, achieves the best performance for the *housing* dataset. Taking into account MAPE, Gaussian Kernel-GNM is not only the winner in terms of average rank, although by a small margin, but also achieves the same mean score over all datasets than *ILF*.

4.5.3 Solar and Wind Contest Datasets

The global results for Experiment I over the contest datasets are shown in Table 4.5.9 for AMS and Table 4.5.11 for GEFCOM. We show MAE and MAPE metrics for each version of Kernel-GNM. Regarding the wind contest, we also compute the results in terms of the Pinball loss. For both contests, the ranking that these models would have got in the official leaderboard, based on MAE for AMS and Pinball loss for GEFCOM, is also shown. Finally, relative rank for each model and metric is described between parentheses. We will focus here in the results for the MAE metric, as conclusions are similar when analyzing the other metrics.

First, the choice of noise distribution assumption is highly relevant for model accuracy, as the worst results are 2.3% and 6.7% higher than the lowest MAE obtained for the AMS and GEFCom2014 datasets, respectively. For instance, this decline in performance would

Table 4.5.6: MAE obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over classical datasets.

Dataset	ϵ-ILF	Lap	Gau	Beta	Weib
abalone	**1.48 (1)**	1.58 (3)	1.53 (2)	1.67 (4)	1.58 (3)
bodyfat	**0.00 (1)**	0.10 (2)	0.10 (2)	0.17 (5)	0.12 (4)
cpusmall	2.13 (3)	**2.07 (1)**	2.12 (2)	2.21 (5)	2.13 (3)
housing	2.28 (3)	2.39 (5)	2.27 (2)	2.36 (4)	**2.24** (1)
mg	**0.09 (1)**	0.17 (5)	0.13 (2)	0.15 (4)	0.13 (2)
mpg	1.91 (2)	1.95 (3)	**1.89 (1)**	2.49 (5)	2.03 (4)
pyrim	**0.06 (1)**	0.10 (3)	**0.06 (1)**	0.19 (5)	0.10 (3)
space_ga	0.14 (2)	0.14 (2)	**0.11 (1)**	0.16 (4)	0.17 (5)
mean	**1.01**	1.06	1.03	1.18	1.06
mean rank	1.75	3.00	**1.63**	4.50	3.13

mean to drop 11 positions in the AMS Kaggle leaderboard. Second, provided that the distribution assumption is properly chosen, general noise SVR models achieve significantly higher precision than classical ϵ-SVR for both contests.

Regarding results corresponding to the solar competition, the Weibull and especially the Beta distributions seem to capture better the underlying noise distribution for the task of solar prediction. Although further testing of our models with different datasets would be needed to confirm these results, they seem to be in line with previous works, such as [64], [62] or [68], that suggest the Beta distribution as a good choice to model solar irradiation.

As for the significance of results obtained for the AMS competition, again we carried out a Friedman test, this time over the global matrix containing the residuals obtained for each of the 98 solar stations by each model tested. This means that the rows of this matrix are the observations for each of the 98 stations concatenated vertically, i.e., one below the other, and the columns are each of the five models evaluated in this experiment. This Friedman test yielded a p-value of $1.85\ 10^{-16}$. Thus, we proceed to compute pairwise Wilcoxon tests, whose results are shown in Table 4.5.10. In this case, adjusted p-values clearly below the 0.05 significance threshold are found. In particular, only models ϵ-*ILF* and *Gau* seem to have a similar performance, with significant differences found for all the other comparisons. This is again reasonable, as ϵ-*ILF* and *Gau* obtained the same position in the leaderboard and similar average metrics. Combining these results with the previously disccused, it seems logical to conclude that the model using the Beta noise distribution assumption, the winner in terms of MAE, MAPE and leaderboard position, achieves a significantly better performance for the problem of solar forecasting in the AMS competition.

Table 4.5.7: Wilcoxon test for Experiment I MAE results over classical datasets.

	ϵ-ILF	**Lap**	**Gau**	**Beta**	**Weib**
ϵ-ILF	1.00	0.28	0.71	**0.04**	0.22
Lap	0.28	1.00	0.30	0.28	0.57
Gau	0.71	0.30	1.00	**0.04**	0.28
Beta	**0.04**	0.28	**0.04**	1.00	**0.08**
Weib	0.22	0.57	0.28	**0.08**	1.00

Similar conclusions can be extracted when analyzing the GEFCom2014 results, as can be seen in Table 4.5.11. This time the Weibull and Beta distributions, in this order, seem to be the best choices for noise assumption, which also seems in line with previous research such as [65] or [68] where Weibull is pointed to be a good fit for wind behaviour. Results for Wilcoxon pairwise test over residuals are shown in Table 4.5.12. Significant differences are found among all models, except for *Beta* and *Weib*, which also obtained the same pinball loss and leaderboard position. Following a similar reasoning to the one we applied for the AMS contest, we could infer that these two are the overall best solutions for GEFCom2014 problem.

Finally, it is interesting to note the results regarding the official leaderboard. The goal of this work is not to find the best possible model in terms of accuracy, as we follow a simple and straightforward pipeline to tackle the problem with almost no data processing, feature engineering or expertise integration, and we also use a relatively small grid for the hyperparameter search; our aim was instead to compare the perfomance of the different noise distributions among themselves and to compare the proposed models with classical $\epsilon-$SVR using ϵ-ILF. However, results obtained are quite positive, with the model using Beta noise assumption getting a score of 2207.12 KJm^{-2}, good enough for eight place among all the 160 participants visible on the Kaggle private leaderboard for the AMS Kaggle contest. As for GEFCom2014, Beta and Weibull models obtain a respectable sixth position.

4.6 Experiment II. Deep-GNM Models

This experiment is analogous to Experiment I but now the goal is to test the performance of the proposed Deep-GNM models. Taking this into account, the experiment consists in comparing for each dataset analyzed the performance of Deep SVM and Deep-GNM models with the results obtained both by classical ϵ-insensitive SVM and the best Kernel-GNM model from the previous experiment. To build these deep models we follow the proposed method described in Section 3.2. We again carry out this experiment using all the datasets detailed in Section 4.3.

Results are presented in Tables 4.6.1, 4.6.2, 4.6.5, 4.6.6, 4.6.9, and 4.6.11, using MAE, MAPE, and Pinball loss as evaluation metrics. These tables consist of the following columns:

Table 4.5.8: MAPE obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over classical datasets.

Dataset	ϵ-ILF	Lap	Gau	Beta	Weib
abalone	**13.2 (1)**	13.6 (3)	13.5 (2)	14.9 (5)	13.3 (2)
bodyfat	**0.3 (1)**	0.4 (2)	**0.3 (1)**	0.5 (3)	1.3 (5)
cpusmall	**2.5 (1)**	**2.5 (1)**	**2.5 (1)**	2.6 (2)	2.6 (2)
housing	18.5 (3)	**17.5 (1)**	18.3 (2)	23.1 (5)	22.3 (4)
mg	9.7 (3)	10.9 (4)	9.1 (2)	13.1 (5)	**9.0 (1)**
mpg	8.3 (2)	**8.1 (1)**	8.3 (2)	13.1 (5)	9.1 (4)
pyrim	**14.1 (1)**	15.0 (4)	14.3 (2)	14.6 (3)	17.4 (5)
space_ga	18.3 (4)	18.3 (4)	18.2 (2)	18.2 (2)	**17.5 (1)**
mean	**10.6**	10.8	**10.6**	12.5	11.6
mean rank	2.0	2.5	**1.75**	4.5	3.1

Table 4.5.9: Metrics obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over the AMS contest dataset.

Metric	ϵ-ILF	Lap	Gau	Beta	Weib
MAE	2234.40 (3)	2260.86 (5)	2234.63 (4)	**2207.12 (1)**	2223.72 (2)
MAPE	12.35 (3)	13.38 (5)	12.47 (4)	**9.76 (1)**	10.81 (2)
leaderboard position	15 (3)	19 (5)	15 (3)	**8 (1)**	12 (2)
mean rank	3	5	3.67	**1**	2

1. **Dataset:** Reference to the dataset to which metric results of that row of the table corresponds.

2. **ϵ-SVR:** Classical ϵ-SVR model, as described in Section 2.2.1.

3. **Kernel-GNM:** Kernel-GNM model proposed in Section 3.1 assuming the noise distribution that yielded the best results in terms of MAE in Experiment I for the corresponding dataset. For instance, Beta distribution would be the choice for the AMS solar dataset, as it was the winner in the previous experiment, as shown in Table 4.5.9. In case of a tie, we will use MAPE as tiebreaker and, if needed, also the Pinball loss.

4. **Deep SVR:** Deep version of the SVR model, as proposed in Section 3.2.1.

5. **Deep-GNM:** Deep-GNM model proposed in Section 3.2 assuming the noise distribution that yielded the best results in Experiment I for the corresponding dataset,

Table 4.5.10: Wilcoxon test for Experiment I results over the AMS contest dataset.

	ϵ-ILF	Lap	Gau	Beta	Weib
ϵ-ILF	1.00	**0.00**	0.87	**0.00**	0.08
Lap	**0.00**	1.00	**0.00**	**0.00**	**0.00**
Gau	0.87	**0.00**	1.00	**0.00**	**0.03**
Beta	**0.00**	**0.00**	**0.00**	1.00	**0.00**
Weib	0.08	**0.00**	**0.03**	**0.00**	1.00

Table 4.5.11: Metrics obtained in Experiment I for each choice of distribution assumption in a Kernel-GNM model over the GEFCOM contest dataset.

Metric	ϵ-ILF	Lap	Gau	Beta	Weib
MAE	3711 (2)	3821 (5)	3727 (4)	3715 (3)	**3581 (1)**
MAPE	12.35 (3)	13.38 (5)	12.47 (4)	**9.76 (1)**	10.81 (2)
pinball loss	0.045 (4)	0.053 (5)	0.044 (3)	**0.039 (1)**	**0.039 (1)**
leaderboard position	10 (3)	14 (5)	10 (3)	**6 (1)**	**6 (1)**
mean rank	3	5	3.5	1.5	**1.25**

i.e., the same that was used in column *Kernel-GNM*.

As in Experiment I, we also carried out a Friedman test using observation residuals obtained over the test set of all datasets, as shown in Table 4.5.1, to check for significant differences between the performance obtained by each model. When a significant p-value was found after applying this test, we also carried out a pairwise comparison with Wilcoxon signed-rank test, where p-values were adjusted by a step-down method using Sidak adjustments.

4.6.1 Artificial Datasets

Table 4.6.1 shows the MAE for each of the five artificial datasets and four types of models used in this experiment. Table 4.6.2 shows analogous results for MAPE. It can be seen that the proposed Deep-GNM models are consistently the best model for all datasets tested, and classical ϵ-SVR provides the worst results. Both Kernel-GNM and deep versions of SVR are able to improve the performance of classical SVR models, but are still below the performance of Deep-GNM.

Two main conclusions can be drawn from these results. First, the deep versions, both standard and general noise, improve or at least equal the performance of the non-deep versions. Second, general noise cost functions give better results than ϵ-ILF when the

Table 4.5.12: Wilcoxon test for Experiment I results over the GEFCOM contest dataset.

	ϵ-ILF	Lap	Gau	Beta	Weib
ϵ-ILF	1.00	**0.00**	0.34	**0.00**	**0.00**
Lap	**0.00**	1.00	**0.00**	**0.00**	**0.00**
Gau	0.34	**0.00**	1.00	**0.00**	**0.00**
Beta	**0.00**	**0.00**	**0.01**	1.00	0.82
Weib	**0.00**	**0.00**	**0.01**	0.82	1.00

Table 4.6.1: MAE obtained in experiment II for each type of model over artificial datasets.

Dataset	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
Zero-noise	0.99 (4)	0.98 (3)	**0.95 (1)**	**0.95 (1)**
Laplace	2.07 (4)	1.86 (2)	1.89 (3)	**1.76 (1)**
Gaussian	1.97 (3)	1.91 (2)	1.97 (3)	**1.83 (1)**
Beta	1.16 (4)	**1.06 (1)**	1.14 (3)	**1.06 (1)**
Weibull	1.58 (4)	1.40 (2)	1.55 (3)	**1.27 (1)**
mean	1.55	1.44	1.5	**1.37**
mean rank	3.8	2.0	2.6	**1.0**

distribution chosen resembles the underlying noise in the data, which is the case here as Kernel-GNM and Deep-GNM columns show results of models assuming the noise distribution that worked best in Experiment I, which for artificial datasets always corresponded to the real noise in the data, as could be expected.

Regarding significance of results, Friedman test over residuals for all observations gives a p-value of 0.01. Therefore, in this case we did not carry out independent Friedman and Wilcoxon tests over the residuals obtained for each artificial dataset, as we did in Experiment I, but only an overall test taking into account the residuals obtained for observations belonging to all the datasets. Results from this pairwise Wilcoxon test analysis show that only ϵ-SVR and Deep SVR models do not present significant differences. Therefore, we could conclude that ϵ-ILF and Deep-GNM are consistently the worst and best performers, respectively. The corresponding Wilcoxon adjusted p-values are shown in Table 4.6.3.

4.6.2 Classical Datasets

In Table 4.6.5 MAE results corresponding to each of the LIBSVM classical datasets evaluated and four types of models used, namely classical ϵ–insensitive SVR, General Noise Models or Kernel-GNM, Deep SVR and Deep General Noise Models or Deep-GNM. The

Table 4.6.2: MAPE obtained in experiment II for each type of model over artificial datasets.

Dataset	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
Zero-noise	7.7 (3)	7.7 (3)	**7.5 (1)**	**7.5 (1)**
Laplace	16.3 (4)	14.6 (2)	14.6 (2)	**14.1 (1)**
Gaussian	15.1 (4)	14.6 (2)	14.8 (3)	**14.3 (1)**
Beta	8.2 (4)	7.9 (2)	8.0 (3)	**7.8 (1)**
Weibull	12.1 (4)	11.0 (2)	11.7 (3)	**10.7 (1)**
mean	11.9	11.2	11.3	**10.9**
mean rank	3.8	2.2	2.4	**1.0**

Table 4.6.3: Wilcoxon test for Experiment II results over artificial datasets.

	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
ϵ-SVR	1.00	**0.02**	0.47	**0.00**
Kernel-GNM	**0.02**	1.00	**0.02**	**0.04**
Deep SVR	0.47	**0.02**	1.00	**0.00**
Deep-GNM	**0.00**	**0.04**	**0.00**	1.00

Kernel-GNM and Deep-GNM columns show the results of choosing the distribution among Laplace, Gaussian, Beta, and Weibull which gave the best results over Experiment I. In particular, we select ϵ-ILF for *abalone*, *bodyfat*, *mg*, and *pyrim*, Laplace for *cpusmall*, Gaussian for *mpg* and *space_ga*, and Weibull for *housing*, as summarized in Table 4.6.4.

Analyzing these results, it can be seen that Deep SVR models are competitive with classical SVR in general, and for some datasets like *cpusmall* or *space_ga* even better. As a matter of fact, standard SVR gets the best average MAE among all the datasets, but this value can be misleading due to different target magnitudes among the evaluated datasets, while Deep SVR is the winner in terms of the more informative average rank metric.

In addition, as seen in Experiment I, General Noise SVR Models do not outperform ϵ–insensitive SVR over all datasets evaluated, but are clearly better for some problems like *mpg* or *cpusmall*, where these models appear to be the best option, which seems to point to a particular distribution of noise for these problems different from the one corresponding to the ϵ-ILF function. Finally, Deep-GNM achieve the best results for *mpg* and *housing*, and consistently give similar if not better results than their kernel-based, i.e., non-deep, General Noise Model counterparts.

Some of the previous conclusions can also be extracted from Table 4.6.6, which presents the results in terms of MAPE for this experiment. However, one relevant difference here

Table 4.6.4: Distribution selected for each classical dataset to build Kernel-GNM and Deep-GNM models.

	abalone	bodyfat	cpusmall	housing	mg	mpg	pyrim	space_ga
Winner Distribution	ϵ-ILF	ϵ-ILF	Lap	Weib	ϵ-ILF	Gau	ϵ-ILF	Gau

Table 4.6.5: MAE obtained in experiment II for each type of model over classical datasets.

Dataset	**ϵ-SVR**	**Kernel-GNM**	**Deep SVR**	**Deep-GNM**
abalone	**1.48 (1)**	1.53 (3)	**1.48 (1)**	1.53 (3)
bodyfat	**0.00 (1)**	0.10 (3)	**0.00 (1)**	0.10 (3)
cpusmall	2.13 (4)	**2.07 (1)**	2.12 (2)	2.12 (2)
housing	2.28 (3)	2.24 (2)	2.30 (4)	**2.19 (1)**
mg	**0.09 (1)**	0.13 (3)	**0.09 (1)**	0.15 (4)
mpg	1.91 (3)	**1.89 (1)**	2.28 (4)	**1.89 (1)**
pyrim	**0.06 (1)**	**0.06 (1)**	**0.06 (1)**	0.09 (4)
space_ga	0.14 (4)	0.11 (3)	**0.09 (1)**	0.10 (2)
mean	**1.01**	1.02	1.05	1.02
mean rank	2.13	2.50	**1.88**	2.38

is that now our proposed Deep-GNM models have a better performance than classical ϵ–insensitive SVR both in terms of mean MAPE value among all the datasets and mean rank. MAPE results are probably more significant here than the ones obtained using MAE, as scale can greatly affect conclusions using that metric. Deep-GNM is in fact the overall winner for both measures, MAPE and mean rank MAPE, although Kernel-GNM achieves very similar performance. This is supported by the results obtained from parwise Wilcoxon tests over the residuals of each type of models, shown in Table 4.6.7, where Kernel-GNM and Deep-GNM show significant differences with respect the other models, but not between themselves, with same behaviour being observed for ϵ-SVR and Deep-SVR. However, it is important to notice here that Deep-GNM present computational advantages over Kernel-GNM, as discussed in Section 3, that will make them the preferred choice in the case of a statitiscally equivalent performance. These Wilcoxon tests were conducted despite the Friedman test yielding a p-value of 0.27 pointing to not overall significant differences because we nevertheless considered them worthy of analysis.

4.6.3 Solar and Wind Contest Datasets

Results of experiment II over the AMS solar and GEFCOM wind contest datasets are shown in Table 4.6.9 and Table 4.6.11. Table 4.6.9 presents MAE and MAPE results for each type of model for the AMS contest and the ranking that this error would have

Table 4.6.6: MAPE obtained in experiment II for each type of model over classical datasets.

Dataset	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
abalone	**13.2 (1)**	13.3 (3)	13.3 (3)	**13.2 (1)**
bodyfat	0.3 (2)	0.3 (2)	**0.2 (1)**	0.3 (2)
cpusmall	2.5 (2)	2.5 (2)	2.8 (3)	**2.3 (1)**
housing	18.5 (4)	17.5 (2)	17.9 (3)	**16.7 (1)**
mg	9.7 (3)	9.0 (2)	**8.4 (1)**	9.9 (4)
mpg	8.3 (3)	**8.1 (1)**	8.8 (4)	**8.1 (1)**
pyrim	**14.1 (1)**	14.3 (3)	14.2 (2)	14.3 (3)
space_ga	18.3 (3)	**17.5 (1)**	18.5 (4)	17.6 (2)
mean	10.6	**10.3**	10.5	**10.3**
mean rank	2.4	2.0	2.6	**1.9**

Table 4.6.7: Wilcoxon test for Experiment II results over classical datasets.

	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
ϵ-SVR	1.00	**0.03**	0.43	**0.01**
Kernel-GNM	**0.03**	1.00	**0.04**	0.27
Deep SVR	0.43	**0.04**	1.00	**0.04**
Deep-GNM	**0.01**	0.27	**0.04**	1.00

got in the official leaderboard of the contest. As usual, relative rank is shown between parentheses and mean rank between the different scores is given as overall evaluation metric. In addition, the pinball loss function is also computed as part of the GEFCOM results shown in Table 4.6.11, as it was used to compute the leaderboard position in this contest. The Kernel-GNM and Deep-GNM columns show the results of choosing the distribution among Laplace, Gaussian, Beta, and Weibull which gave the best results over Experiment I for these datasets. In particular, we select Beta distribution for the AMS contest and Weibull for GEFCOM, as summarized in Table 4.6.8.

Results show that Deep-GNM is the winner for all the considered metrics in both contests. Furthermore, classical ϵ-SVR shows the worst performance both for AMS and GEFCOM datasets. Finally, Kernel-GNM shows better results than Deep-SVR for the AMS contest, but in GEFCOM both approaches yield similar performance, with no clear winner and the best model between the two decided by which metric is considered.

Table 4.6.8: Distribution selected for each contest dataset to build Kernel-GNM and Deep-GNM models.

	AMS	GEFCOM
Winner Distribution	Beta	Weib

Table 4.6.9: Metrics obtained in Experiment II for each type of model over the AMS contest dataset.

Metric	**ϵ-SVR**	**Kernel-GNM**	**Deep SVR**	**Deep-GNM**
MAE	2234.40 (4)	2207.13 (2)	2221.73 (3)	**2199.16 (1)**
MAPE	12.35 (4)	9.76 (2)	11.73 (3)	**9.21 (1)**
leaderboard position	15 (4)	**8 (1)**	10 (3)	**8 (1)**
mean rank	4	2.67	3	**1**

A Friedman test over AMS residuals for all observations showed a p-value of 7.26 10^{-22} and a p-value of 6.63 10^{-27} over GEFCOM, pointing in both cases to significant differences on model performance between the different types of models analyzed. Pairwise Wilcoxon test results for AMS and GEFCOM residuals are shown in Table 4.6.10 and Table 4.6.12, respectively. In both cases significant differences between Deep-GNM and all the other models are found, which combined to the previously remarked fact that these models give the best results for all metrics considered, points to the selection of Deep-GNM as best overall model. Furthermore, significant performance differences between classical ϵ-SVR and Deep SVR are also found, with better metrics being achieved in the case of Deep SVR models. Finally, Kernel-GNM show significant differences with respect to both ϵ-SVR and Deep SVR for the AMS contest, but similar performance with respect to Deep SVR in GEFCOM.

These results stongly support our hypothesis of the usefulness of applying Deep frameworks, as they show that combining both standard and general noise models with deep learning structures significantly increases model performances. This is the reason why we will focus in this type of models in our third and last experiment, described next in Section 4.7.

4.7 Experiment III. Deep-GNM Models with Uncertainty Intervals

In this experiment we test the error of uncertainty intervals built following the method proposed in Section 3.5 under different assumptions of noise distribution and distinct choices of clustering methods.

Table 4.6.10: Wilcoxon test for Experiment II results over the AMS dataset.

	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
ϵ-SVR	1.00	**0.00**	**0.03**	**0.00**
Kernel-GNM	**0.00**	1.00	**0.02**	**0.04**
Deep SVR	**0.03**	**0.02**	1.00	**0.00**
Deep-GNM	**0.00**	**0.04**	**0.00**	1.00

Table 4.6.11: Metrics obtained in Experiment II results over the GEFCOM dataset.

Metric	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
MAE	3711 (4)	3581 (2)	3597 (3)	**3523 (1)**
MAPE	18.05 (4)	15.83 (2)	17.63 (3)	**15.57 (1)**
pinball loss	0.045 (4)	0.039 (3)	0.038 (2)	**0.037 (1)**
leaderboard position	10 (4)	6 (3)	5 (2)	**2 (1)**
mean rank	4	2.5	2.5	**1**

We build these error intervals for the best Deep-GNM model for each dataset observed in Experiment II, and the noise distribution used to compute the intervals is the same applied to train the corresponding Deep-GNM model. In this experiment we only take into account prediction residuals over the test datasets employed in Experiment II, where a new split into train, validation, and test is carried out, using 50% to train and validate the clustering methods and build the associated error intervals, and 50% to test them.

The experiment is carried out two times, the first one computing intervals that should contain 80% of the test predictions, and the second with 90% intervals, i.e., choosing $s = 0.1$ and $s = 0.05$ respectively. Performance is measured using the per_{err} metric, defined in Section 4.4.2, as evaluation metric, which basically measures the difference in absolute value between the percentage of points that should fall in the confidence interval, 0.8 and 0.9 for the first and the second iterations of the experiment, respectively, and the ratio of points that is actually captured by the interval. We recall that the formula corresponding to this metric is the following one

$$per_{err}(s) = \frac{\{\#\ of\ \psi_i^{test} \in [a,b]\}}{N} - (1-2s) \quad . \tag{4.7.1}$$

The mean of the results obtained using $s = 0.1$ and $s = 0.05$ is then computed to obtain the final error per_{err} metric shown in out results, in order to get a better overall idea of the performance obtained and avoid adjusting the conclusions too much to a particular choice of confidence threshold.

Table 4.6.12: Wilcoxon test for Experiment II MAE results over the GEFCOM dataset.

	ϵ-SVR	Kernel-GNM	Deep SVR	Deep-GNM
ϵ-SVR	1.00	**0.00**	**0.00**	**0.00**
Kernel-GNM	**0.00**	1.00	0.26	**0.02**
Deep SVR	**0.00**	0.26	1.00	**0.00**
Deep-GNM	**0.00**	**0.02**	**0.00**	1.00

Results are presented in tables 4.7.2, 4.7.3, and 4.7.5. These tables have the following columns:

1. M_{unique}**:** Method where we build a unique interval for all instances in the test set.

2. M_k**:** Method where we cluster the data using standard and general methods as described in Section 2.7. In particular, we use k-means here as all features are numerical. We try values for k ranging from 2 to 10 and keep the intervals with the best performance over cross-validation.

3. $M_{magnitude}$**:** Analogous to M_k but this time we use techiques based on magnitude scaling to cluster data, as explained in Section 3.4.2. Again, we try between 2 and 10 clusters and select the value that yields the best results after cross-validation.

Furthermore, we also carried out a Friedman test to check for significant differences between the performance obtained by each of these three methods to build intervals. When a significant p-value was found after applying this test, we also carried out a pairwise comparison with Wilcoxon signed-rank test, where p-values were modified by a step-down method using Sidak adjustments. For this experiment we carried out the statistical tests over the overall per_{err} metric for each dataset, or each station in the case of the solar and wind contests, as individual residuals for each sample of the test set could not be computed for this metric and did not make sense in the case of confidence interval analysis. Therefore, in this case we use the matrix structure shown in Table 4.7.1 as analogous of the structure described in Table 4.5.1 that was employed in the previous experiments. We are aware that the number of rows m of this matrix, 5 for the artificial datasets, 8 for the classical datasets, 98 for AMS, and 10 for GEFCOM, could be not enough in order to obtain statistically significant results in our Friedman and Wilcoxon tests, and that this could be a limitation of the conclusions drawn in this experiment.

4.7.1 Artificial Datasets

Table 4.7.2 contains the results obtained in Experiment III for the artificial datasets. Although M_k is the method that achieves slightly lower errors, results show an almost equivalent performance for the three types of error interval computations, which is to be expected taking into account that the noise inserted in this case is totally independent from the input features x, so the hypothesis that leads to the existence of a constant interval in the method proposed in [27] is correct here. This is confirmed by the results obtained after applying a Friedman test over the per_{err} matrix shown in Table 4.7.1, a p-value of

Table 4.7.1: Structure of matrix residuals employed in Friedman and Wilcoxon tests for Experiment III.

Dataset/station	M_{unique}	M_k	$M_{magnitude}$
Dataset/station 1	$per_{err_1}^{M_{unique}}$	$per_{err_1}^{M_k}$	$per_{err_1}^{M_{magnitude}}$
Dataset/station 2	$per_{err_2}^{M_{unique}}$	$per_{err_2}^{M_k}$	$per_{err_2}^{M_{magnitude}}$
Dataset/station 3	$per_{err_3}^{M_{unique}}$	$per_{err_3}^{M_k}$	$per_{err_3}^{M_{magnitude}}$
Dataset/station 4	$per_{err_4}^{M_{unique}}$	$per_{err_4}^{M_k}$	$per_{err_4}^{M_{magnitude}}$
Dataset/station 5	$per_{err_5}^{M_{unique}}$	$per_{err_5}^{M_k}$	$per_{err_5}^{M_{magnitude}}$
...	...	...	...
Dataset/station M	$per_{err_m}^{M_{unique}}$	$per_{err_m}^{M_k}$	$per_{err_m}^{M_{magnitude}}$

0.44, showing no significant differences are found among the three interval computation approaches.

Nevertheless, M_k is still able to equal or improve the performance obtained by the standard M_{unique} constant interval, except for the particular case of the zero-noise dataset. This is an interesting result, as it is reasonable to think that M_k is almost never going to significantly decrease the performance results obtained by M_{unique}. Unless the k value is selected in a really wrong manner, which can be avoided doing hyperparameter tuning of k as proposed here, in the worst case scenario M_k should provide results similar to M_{unique}.

It is also interesting to note that the per_{err} is higher in the zero-noise dataset, i.e., the one without noise and that is defined by the following formulation

$$y_i \quad = \quad x_i^a \cdot b, \; i = 1, 2, \ldots, 500,000 \; , \tag{4.7.2}$$

with x_i, a and b 1024-dimensional vectors where each element is randomly chosen from the uniform distributions over the intervals $[0.1, 2]$, $[1, 5]$, and $[1, 10]$ respectively. The higher per_{err} here is probably due to the fact that for that dataset the errors correspond entirely to the model itself and not to the existence of an underlying noise in the data and, therefore, these errors probably do not follow a particular distribution and the proposed uncertainty intervals are not able to capture them correctly.

4.7.2 Classical Datasets

In Table 4.7.3 per_{err} results over the classical datasets considered in this thesis are shown. Is is clear when analyzing the results of this experiment that the drawback of having a constant interval for all the samples, M_{unique}, as is the case for the proposed method in [27], has a significant impact in the interval's accuracy, an impact that is clearly lessened when our proposed clustering methods are applied before the construction of these intervals. In particular, it is easy to see how both M_k and $M_{magnitude}$ methods for uncertainty intervals construction consistently yield better results than the standard M_{unique}, the one corresponding to the proposal in [27].

Table 4.7.2: Uncertainty intervals per_{err} over artificial datasets.

Dataset	M_{unique}	M_k	$M_{magnitude}$
Zero-noise	**1.2 (1)**	1.4 (3)	**1.2 (1)**
Laplace	0.7 (2)	**0.6 (1)**	0.8 (3)
Gaussian	**0.4 (1)**	**0.4 (1)**	0.6 (3)
Beta	0.4 (3)	**0.2 (1)**	**0.2 (1)**
Weibull	0.6 (2)	**0.2 (1)**	0.6 (2)
mean	0.7	**0.6**	0.7
mean rank	1.8	**1.4**	2.0

$M_{magnitude}$ appears to be the best option for most datasets, although M_k is competitive and achieves the best performance for several datasets. Probably which one is the winner depends directly on the correlation between the model error and the magnitude of the target, which favours the $M_{magnitude}$ type of clustering. When there is a strong correlation between real target values and residuals, which can arise for instance when high values correspond to situations with more uncertainty, a constant error interval is no longer well defined and building wider intervals for high target values is recommended. This is precisely the output expect when applying $M_{magnitude}$ clustering.

These conclusions are backed up by the output of carrying out a Friedman test, yielding a p-value of 0.002 and then a pairwise Wilcoxon test over the per_{err} matrix presented in Table 4.7.1, which is shown in Table 4.7.4. Significant difference is found between M_{unique} and the two other methods, so we could conclude it yields significant worse results overall. However, no significant difference is found between M_k and $M_{magnitude}$ intervals.

4.7.3 Solar and Wind Contest Datasets

Table 4.7.5 contains the results obtained in Experiment III for the real-world datsets corresponding to the AMS solar and GEFCOM wind energy contests. The negative impact of computing error intervals with constant width in method M_{unique} is clear when looking at these results, as the best per_{err} obtained when using this approach is more than twice the ones accomplished when applying some sort of clustering techniques, as is the case for M_k and $M_{magnitude}$.

In particular, the error corresponding to the uncertainty intervals is decreased to less than half, both for AMS and GEFCOM datasets, when a suitable clustering technique is selected, which allows to avoid the drawback of having one unique constant interval. $M_{magnitude}$ is clearly the winner for the AMS dataset and M_k for the GEFCOM one.

Regarding the AMS contest, one reason that may explain this is the fact that solar prediction errors are strongly related to the hour of the day, which is intrinsically connected to the magnitude of the solar radiation, the target in this case, as this normally follows a pattern similar to the curve known as *clear sky* curve, as shown in Figure 4.7.1. This

Table 4.7.3: Uncertainty intervals per_{err} over classical datasets.

Dataset	M_{unique}	M_k	$M_{magnitude}$
abalone	1.8 (3)	1.6 (2)	**1.2 (1)**
bodyfat	9.1 (3)	**4.5 (1)**	**4.5 (1)**
cpusmall	2.0 (3)	**1.4 (1)**	**1.4 (1)**
housing	3.4 (3)	2.3 (2)	**1.1 (1)**
mg	1.2 (3)	**0.4 (1)**	0.8 (2)
mpg	2.9 (2)	2.9 (2)	**1.5 (1)**
pyrim	15.4 (3)	**7.7 (1)**	**7.7 (1)**
space_ga	0.9 (3)	**0.4 (1)**	**0.4 (1)**
mean	4.6	2.7	**2.3**
mean rank	2.9	1.4	**1.3**

Table 4.7.4: Wilcoxon test for Experiment III errint results over classical datasets.

	M_{unique}	M_k	$M_{magnitude}$
M_{unique}	1.00	**0.03**	**0.02**
M_k	**0.03**	1.00	0.25
$M_{magnitude}$	**0.02**	0.25	1.00

figure shows the comparison between the clear sky curve expected radiation value at each hour for the 13th of June at a location in Medan city of Indonesia vs the actual radiation measurements obtained that day in that location. It can be seen that uncertainty in the central hours is low and follows different patterns than, for instance, the evening or sunset hours.

In the GEFCOM wind contest, the error patterns seem to not be as much correlated with a phenomenon like the clear sky curve as in AMS, which may be the reason why a more generic clustering method like M_k yields better results. However, $M_{magnitude}$ is still able to obtain an error of less than 60% the one yielded by the standard M_{unique} constant error interval.

For both contest datasets, a Friedman test over a matrix of per_{err} values following the structure in Table 4.7.1, where each station per_{err} is presented in a different row, shows significant differences, with p-values of $7.47\ 10^{-43}$ and $5.85\ 10^{-13}$, respectively.

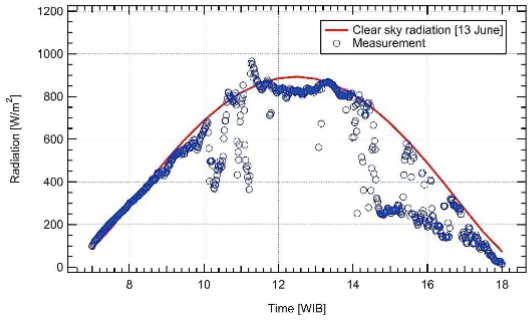


Figure 4.7.1: Clear Sky Curve. Red line shows the clear sky curve expected radiation value and blue points represent the actual radiation measurements. Taken from [85].

Table 4.7.5: Uncertainty intervals per_{err} over AMS and GEFCOM contests datasets.

Dataset	M_{unique}	M_k	$M_{magnitude}$
AMS	1.78	0.82	**0.60**
GEF	2.72	**1.34**	1.62

Table 4.7.6: Wilcoxon test for Experiment III per_{err} results over AMS.

	M_{unique}	M_k	$M_{magnitude}$
M_{unique}	1.00	**0.00**	**0.00**
M_k	**0.00**	1.00	**0.02**
$M_{magnitude}$	**0.00**	**0.02**	1.00

Table 4.7.7: Wilcoxon test for Experiment III per_{err} results over GEFCOM.

	M_{unique}	M_k	$M_{magnitude}$
M_{unique}	1.00	**0.00**	**0.00**
M_k	**0.00**	1.00	**0.01**
$M_{magnitude}$	**0.00**	**0.01**	1.00

Wilcoxon test also shows significant results when comparing the performance of the three methods. Results for pairwise Wilcoxon tests are shown in Table 4.7.6 and Table 4.7.7 for AMS solar contest and GEFCOM2014 wind dataset, respectively. In this case, the input used in both these statistical tests has been the overall per_{err} obtained for each station applying each one of the interval computation methods analyzed in this work which, specially for the GEFCOM contest, could pose a limitation regarding statistical significance of the conclusions drawn, as we only have available information for 10 stations and this means that we are aplying our statistical tests over a matrix of only 10 rows.

These results seem to confirm our hypothesis regarding the usefulness of applying clustering to build different intervals for each group of points, solving or at least lessening the negative impact of having error intervals with constant width for all input points.

Chapter 5

Conclusions and Further Work

> Perhaps one did not want to be loved so much as to be understood.
>
> *George Orwell, 1984*

5.1 Conclusions

The main goals of this thesis were five:

1. To propose a framework to train General Noise SVR Models using a suitable optimization method.

2. To give a method to build Deep General Noise Models that combine the highly nonlinear feature processing of DL models with the predictive potential of using general noise loss functions, from which the ϵ-insensitive loss function used in SVR is just a particular example.

3. To describe a direct approach to build error intervals for SVR or other regression models, based on the assumption of model residuals following a particular probability distribution.

4. To unify the previous three goals in a single and final model framework to train Deep General Noise Models for regression with uncertainty intervals associated to each prediction.

5. To follow the principles of reproducible research, with all implementations and datasets used being publicly accesible. In particular, the algorithms necessary to apply these techniques have been implemented using R and Python as programming languages and made publicly available via CRAN or GitHub [1] repositories. Moreover, the datasets employed in the experiments are available online [2] [3] [4] or, in the case of

[1] `https://github.com/jesuspradaalonso/phd`.

[2] Classical datasets available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[3] AMS solar dataset available at `https://www.kaggle.com/competitions/ams-2014-solar-energy-prediction-contest/overview/description`.

[4] GEFCOM wind dataset available at [83].

the artificial datasets created as part of this thesis, easy to replicate following the formulas and code given in this work.

In order to achieve these goals we have presented the necessary theoretical background in Chapter 2, described our proposed methods with all the formulations and implementation details required in Chapter 3, and conducted experiments in Chapter 4 to test their usefulness.

Regarding the methods proposed in this work, these are the main contributions of this thesis:

- The use of NORMA optimization method, to solve the problem of SMO no longer being a suitable option, in combination with general noise versions of the SVR formulation, to create Kernel-GNM models.

- Explicit formulations for the optimal loss functions corresponding to the Beta and Weibull distributions. In addition, a method is proposed to avoid the problems that could arise due to the fact of possible non-convexity in the K-GNM optimization problem, consisting on restraining the search of density parameters to the space of values, previously computed, that make the formulations convex.

- A proposal to build a deep version of the SVR model. The method is based on replacing the loss function used in classical formulations of Deep Neural Network models, usually the MSE for regression, for the ILF loss function usually employed in classical SVR formulations, this way combining the positive aspects of Deep Learning frameworks and the ϵ-ILF.

- Going a step further with respect to the previous point, a proposal to use a chosen general noise loss function, with ILF as just a specific case, in a Deep Fully Connected Neural Network to adapt our model to the particularities of the problem at hand. We have called these models Deep-GNM.

- Formulations to extend the method suggested in [27] to build uncertainty intervals for SVR for the Beta and Weibull distributions, giving all the explicit formulations needed.

- The combination of clustering methods, both standard ones and others based on target scale, with the method proposed in [27] to compute uncertainty intervals. This allows to solve the drawback of some mathematical assumptions done in [27] which led to constant error intervals, i.e., not dependant on the input value.

- The computation of these uncertainty intervals not only for SVR models but also for our proposed K-GNM and D-GNM models. The combination of both intervals and general noise models is feasible in a straightforward manner.

- The formula employed to create artificial datasets, where noise following different distributions is inserted, and scripts run to build the actual artificial datasets employed in our experiments, which are provided via GitHub [1].

- The code necessary to compute all the models used in the experiments, which is implemented in R libraries created during this research and uploaded to CRAN, or by means of Python scripts made available through GitHub repositories. Links to all this available code are given in Section 4.1.

Experiments show the usefulness of the proposed methods. First, our tests show that the suggested general noise models can achieve more accurate predictions than classical ϵ-SVR models if the noise distribution is properly chosen. In case there is no prior knowledge about which could be a correct noise distribution assumption, the particular distribution to be employed could be considered as an additional hyperparameter and selected by grid search over validation.

Second, results show that deep versions, of both classical ϵ-SVR and the proposed GNM models, clearly outperform their non-deep kernel-based counterparts when the volume of the datasets used is large enough, which is the case for the artificial and contest datasets.

Furthermore, the noise distributions that seem to capture best the underlying noise distribution in the solar task are the Weibull and, even more so, the Beta distributions. Regarding the wind energy problem, the Weibull distribution is the one that seems to be most suited to the problem. Both results agree with the conclusions drawn from previous research regarding the nature of solar radiation and wind behaviour.

Finally, the proposed clustering methods seem to largely solve the critical drawback of a constant width in the uncertainty estimates that could arise in our framework, surely the main difficulty present in the original formulation of this method proposed in [27] to build error intervals.

5.2 Further Work

Regarding possible lines of further research, one of them could be to add more distributions to the ones studied in this thesis, such as Logistic or Poisson, and then test the performance of our proposed framework for problems where these distributions may be of relevance, like healthcare tasks. The *Keras* library also includes several regression losses that could be plugged into our D-GNM models to measure their predictive performance.

Also, although we have focused during this work on fully connected models as DL frameworks, it should be possible to extend our proposed model in a relatively straightforward manner to other deep structures like Convolutional Neural Networks. It could be interesting to check if conclusions extracted from experiments with this type of DL models are analogous to the ones obtained in this research. In particular, this could prove to be specially relevant for the wind and solar prediction problems, as numerical weather predictions are given over a grid of latitude and longitude coordinates and, therefore, there is a spatial structure in the information used as input of the models.

Another reasonable extension of the research carried out here will be to compare the accuracy of the uncertainty intervals built following the approach suggested here versus error intervals computed using ensemble weather prediction as the one from the Global Ensemble Forecast System [82], which provides 11 separate forecasts, or ensemble members, and therefore allows to build 11 different predictions and compute error intervals by counting how many of these 11 predictions fall within a specific range, in similar fashion to the methodology proposed in [86].

Finally, regarding selection of distribution parameters, like κ and λ parameters in the Weibull distribution, two alternatives to the approach followed in this work could be tested. First, instead of applying Maximum Likelihood Estimation, MLE, using cross-validation residuals of the most generic model analyzed in our experiments, which is the classical ϵ-SVR model, a nonparametric model like Random Forest, which does not make asumptions regarding noise distribution on the data, could have been applied and its corresponding cross-validation residuals employed in the MLE formulations. A second alternative would have been to just consider these parameters as additional hyperparameters and select their optimal values by the zoom in grid search algorithms described in Section 4.2.

We plan to continue our research in the future studying these and other possible lines of further research.

Appendices

Appendix A

Appendix: Author's Publications

The research carried out during the elaboration of this thesis has lead to the following publications:

A.1 Journals

1. Prada, J., & Dorronsoro, J. R. (2018). General noise support vector regression with non-constant uncertainty intervals for solar radiation prediction. Journal of Modern Power Systems and Clean Energy, 6(2), 268-280. IF: 2,85. Q2 (48/127 in Renewable Energy, Sustainability and the Environment).

2. Díaz-Vico, D., Prada, J., Omari, A., & Dorronsoro, J. R. (2020). Deep support vector neural network, Integrated Computer-Aided Engineering, 27(4): 389-402. IF: 4,87, Q1 (28/112 in Computer science, interdisciplinary applications).

A.2 Conference Papers

1. Díaz-Vico, D., Prada, J., Omari, A., & Dorronsoro, J. R. (2019, June). Deep Support Vector Classification and Regression. In International Work-Conference on the Interplay Between Natural and Artificial Computation (pp. 33-43). Lecture Notes in Computer Science 11487, Springer. Core C.

2. Prada, J., & Dorronsoro, J. R. (2017, June). General noise SVRs and uncertainty intervals. In International Work-Conference on Artificial Neural Networks (pp. 734-746). Lecture Notes in Computer Science 10306, Springer. Core B.

3. Prada, J., & Dorronsoro, J. R. (2015, June). SVRs and uncertainty estimates in wind energy prediction. In International Work-Conference on Artificial Neural Networks (pp. 564-577). Lecture Notes in Computer Science 9095S. Springer. Core B.

4. Torres, A., Prada, J., & Dorronsoro, J. R. (2014). Nowcasting Meteorological Readings for Wind Energy Prediction. Proceedings of the 2014 Conference of the European Wind Energy Association, Barcelona March 11-13, 596-605.

A.3 Other Publications with no Connection to Thesis

1. Prada, J., Gala, Y., & Sierra, A. L. (2021). COVID-19 Mortality Risk Prediction Using X-Ray Images. International Journal of Interactive Multimedia & Artificial Intelligence, 6(6). IF: 4,94. Q2 (48/145 in Computer Science, Artificial Intelligence)

2. Prada, J. (2015, July). Predicting with Twitter. In 2nd European Conference on Social Media ECSM (pp. 734-746). ACPI.

Appendix B

Appendix: AMS solar contest dataset

Table B.0.1: Kaggle AMS solar contest dataset variables and their corresponding units.

Variable	**Description**	**Units**
apcp_sfc	3-Hour accumulated precipitation at the surface	kg m^{-2}
dlwrf_sfc	Downward long-wave radiative flux average at the surface	W m^{-2}
dswrf_sfc	Downward short-wave radiative flux average at the surface	W m^{-2}
pres_msl	Air pressure at mean sea level flux average at the surface	Pa
pwat_eatm	Precipitable Water over the entire depth of the atmosphere	kg m^{-2}
spfh_2m	Specific Humidity at 2 m above ground	kg kg^{-1}
tcdc_eatm	Total cloud cover over the entire depth of the atmosphere	%
tcolc_eatm	Total column-integrated condensate over the entire atmos	kg m^{-2}
tmax_2m	Max. Temperature over the past 3 hours at 2 m above the ground	K
tmin_2m	Min. Temperature over the past 3 hours at 2 m above the ground	K
tmp_2m	Current temperature at 2 m above the ground	K
tmp_sfc	Temperature of the surface flux average at the surface	K
ulwrf_sfc	Upward long-wave radiation at the surface	W m^{-2}
ulwrf_tatm	Upward long-wave radiation at the top of the atmosphere	W m^{-2}
uswrf_sfc	Upward short-wave radiation at the surface	W m^{-2}

Bibliography

[1] P. Jackson and P. Jackson, *Introduction to expert systems*, vol. 2. Addison-Wesley Reading, MA, 1990.

[2] T. M. Mitchell, *The discipline of machine learning*, vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning, 2006.

[3] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[5] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.

[6] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[7] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] J. Honaker, G. King, M. Blackwell, *et al.*, "Amelia ii: A program for missing data," *Journal of Statistical Software*, vol. 45, no. 7, pp. 1–47, 2011.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[10] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[11] C. M. Kirsch, "Principles of real-time programming," in *International Workshop on Embedded Software*, pp. 61–75, Springer, 2002.

[12] M. A. Beyer and D. Laney, "The importance of big data: a definition," *Stamford, CT: Gartner*, pp. 2014–2018, 2012.

[13] A. De Mauro, M. Greco, and M. Grimaldi, "What is big data? a consensual definition and a review of key research topics," in *AIP conference proceedings*, vol. 1644, pp. 97–104, AIP, 2015.

[14] R. Mello, L. R. Leite, and R. A. Martins, "Is big data the next big thing in performance measurement systems?," in *IIE Annual Conference. Proceedings*, p. 1837, Institute of Industrial and Systems Engineers (IISE), 2014.

[15] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[16] Y. Sahin, S. Bulkan, and E. Duman, "A cost-sensitive decision tree approach for fraud detection," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5916–5923, 2013.

[17] M.-W. Huang, C.-W. Chen, W.-C. Lin, S.-W. Ke, and C.-F. Tsai, "Svm and svm ensembles in breast cancer prediction," *PloS One*, vol. 12, no. 1, p. e0161501, 2017.

[18] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[19] N. Cristianini, J. Shawe-Taylor, *et al.*, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[20] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[21] A. Statnikov, *A gentle introduction to support vector machines in biomedicine: Theory and methods*, vol. 1. World Scientific, 2011.

[22] T. Van Gestel, J. A. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle, "Benchmarking least squares support vector machine classifiers," *Machine Learning*, vol. 54, no. 1, pp. 5–32, 2004.

[23] M. Pontil, S. Mukherjee, and F. Girosi, "On the noise model of support vector machines regression," in *International Conference on Algorithmic Learning Theory*, vol. 1968 of *Lecture Notes in Computer Science*, pp. 316–324, Springer, 2000.

[24] H. Bludszuweit, J. A. Domínguez-Navarro, and A. Llombart, "Statistical analysis of wind power forecast error," *IEEE Transactions on Power Systems*, vol. 23, no. 3, pp. 983–991, 2008.

[25] Q. Hu, S. Zhang, Z. Xie, J. Mi, and J. Wan, "Noise model based ν-support vector regression with its application to short-term wind speed forecasting," *Neural Networks*, vol. 57, pp. 1–11, 2014.

[26] A. N. Celik, "A statistical analysis of wind power density based on the weibull and rayleigh models at the southern region of turkey," *Renewable Energy*, vol. 29, no. 4, pp. 593–604, 2004.

[27] C.-J. Lin, R.-C. Weng, *et al.*, "Simple probabilistic predictions for support vector regression," tech. rep., Department of Computer Science, National Taiwan University, 2004.

[28] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2001.

[29] A. J. Smola and B. Schölkopf, "On a kernel-based method for pattern recognition, regression, approximation, and operator inversion," *Algorithmica*, vol. 22, no. 1-2, pp. 211–231, 1998.

[30] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," in *Advances in Kernel Methods-Support Vector Learning*, 1999.

[31] R. Collobert, F. Sinz, J. Weston, and L. Bottou, "Trading convexity for scalability," in *Proceedings of the 23rd international conference on Machine learning*, pp. 201–208, ACM, 2006.

[32] W. Chu, S. S. Keerthi, and C. J. Ong, "Bayesian support vector regression using a unified loss function," *IEEE Transactions on Neural Networks*, vol. 15, no. 1, pp. 29–44, 2004.

[33] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[34] J. A. Suykens, J. Vandewalle, and B. De Moor, "Optimal control by least squares support vector machines," *Neural Networks*, vol. 14, no. 1, pp. 23–35, 2001.

[35] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[36] R. A. Fisher, "Theory of statistical estimation," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 22, pp. 700–725, Cambridge University Press, 1925.

[37] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for svm," *Mathematical Programming*, vol. 127, no. 1, pp. 3–30, 2011.

[38] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.

[39] B. Schölkopf, R. Herbrich, and A. J. Smola, "A generalized representer theorem," in *International conference on computational learning theory*, pp. 416–426, Springer, 2001.

[40] W. SARLE, "Neural networks and statistical models," in *Proceedings Of The 19th Annual SAS Users Group International Conference*, pp. 1538–1550, SAS Institute, 1994.

[41] M. A. F. Azlah, L. S. Chua, F. R. Rahmad, F. I. Abdullah, and S. R. Wan Alwi, "Review on techniques for plant leaf classification and recognition," *Computers*, vol. 8, no. 4, p. 77, 2019.

[42] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, pp. 609–616, 2009.

[43] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of mfcc," *Journal of Computer science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[44] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[45] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron) a review of applications in the atmospheric sciences," *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[46] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation*, pp. 265–283, 2016.

[47] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2545–2553, JMLR. org, 2017.

[50] Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for nonconvex optimization," *corr abs/1502.04390*, 2015.

[51] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9 of *JMLR Proceedings*, pp. 249–256, JMLR.org, 2010.

[52] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[54] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–35, 2015.

[55] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010.

[56] P. R. Kumar and E. Manash, "Deep learning: A branch of machine learning," in *Journal of Physics: Conference Series*, vol. 1228, p. 012045, IOP Publishing, 2019.

[57] J. A. Hartigan, *Clustering Algorithms*. Wiley, 1975.

[58] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[59] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," in *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 600–607, ACM, 2002.

[60] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283–304, 1998.

[61] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining,(PAKDD)*, pp. 21–34, Citeseer, 1997.

[62] H. Assuncao, J. Escobedo, and A. Oliveira, "Modelling frequency distributions of 5 minute-averaged solar radiation indexes using beta probability functions," *Theoretical and Applied Climatology*, vol. 75, no. 3-4, pp. 213–224, 2003.

[63] J. P. Klein, N. Keiding, and C. Kamby, "Semiparametric Marshall-Olkin models applied to the occurrence of metastases at multiple sites after breast cancer," *Biometrics*, vol. 45, no. 4, pp. 1073–1086, 1989.

[64] F. Y. Ettoumi, A. Mefti, A. Adane, and M. Bouroubi, "Statistical analysis of solar measurements in algeria using beta distributions," *Renewable Energy*, vol. 26, no. 1, pp. 47–67, 2002.

[65] C. Carrillo, J. Cidrás, E. Díaz-Dorado, and A. F. Obando-Montaño, "An approach to determine the Weibull parameters for wind energy analysis: The case of Galicia (Spain)," *Energies*, vol. 7, no. 4, pp. 2676–2700, 2014.

[66] A. K. Gupta and S. Nadarajah, *Handbook of beta distribution and its applications*. CRC Press, 2004.

[67] H. Rinne, *The Weibull distribution: a handbook*. CRC Press, 2008.

[68] J. Prada and J. R. Dorronsoro, "General noise support vector regression with non-constant uncertainty intervals for solar radiation prediction," *Journal of Modern Power Systems and Clean Energy*, vol. 6, no. 2, pp. 268–280, 2018.

[69] D. Díaz-Vico, J. Prada, A. Omari, and J. R. Dorronsoro, "Deep support vector classification and regression," in *International Work-Conference on the Interplay Between Natural and Artificial Computation*, vol. 11487 of *Lecture Notes in Computer Science*, pp. 33–43, Springer, 2019.

[70] D. Diaz-Vico, J. Prada, A. Omari, and J. Dorronsoro, "Deep support vector neural networks," *Integrated Computer-Aided Engineering*, vol. 27, no. 4, pp. 389–402, 2020.

[71] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[72] S. Akram and Q. U. Ann, "Newton Raphson method," *International Journal of Scientific & Engineering Research*, vol. 6, no. 7, pp. 1748–1752, 2015.

[73] J. M. Wooldridge, "Applications of generalized method of moments estimation," *Journal of Economic Perspectives*, vol. 15, no. 4, pp. 87–100, 2001.

[74] D. Mao and W. Li, "A bounded derivative method for the maximum likelihood estimation on weibull parameters," *arXiv preprint arXiv:0906.4823*, 2009.

[75] J. Prada and J. R. Dorronsoro, "SVRs and uncertainty estimates in wind energy prediction," in *International Work-Conference on Artificial Neural Networks*, vol. 9095 of *Lecture Notes in Computer Science*, pp. 564–577, Springer, 2015.

[76] J. Prada and J. R. Dorronsoro, "General noise SVRs and uncertainty intervals," in *International Work-Conference on Artificial Neural Networks*, vol. 10306 of *Lecture Notes in Computer Science*, pp. 734–746, Springer, 2017.

[77] V. Cherkassky and Y. Ma, "Practical selection of svm parameters and noise estimation for svm regression," *Neural Networks*, vol. 17, no. 1, pp. 113–126, 2004.

[78] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[79] J. Prada, Y. Gala, and A. Sierra, "COVID-19 mortality risk prediction using X-ray images," *International Journal of Interactive Multimedia & Artificial Intelligence*, vol. 6, no. 6, pp. 7–14, 2021.

[80] K. E. S. Pilario, Y. Cao, and M. Shafiee, "A kernel design approach to improve kernel subspace identification," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 7, pp. 6171–6180, 2020.

[81] F. V. Brock, K. C. Crawford, R. L. Elliott, G. W. Cuperus, S. J. Stadler, H. L. Johnson, and M. D. Eilts, "The Oklahoma Mesonet: a technical overview," *Journal of Atmospheric and Oceanic Technology*, vol. 12, no. 1, pp. 5–19, 1995.

[82] X. Zhou, Y. Zhu, D. Hou, Y. Luo, J. Peng, and R. Wobus, "Performance of the new NCEP Global Ensemble Forecast System in a parallel experiment," *Weather and Forecasting*, vol. 32, no. 5, pp. 1989–2004, 2017.

[83] T. Hong, P. Pinson, S. Fan, H. Zareipour, A. Troccoli, and R. J. Hyndman, "Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond," *International Journal of forecasting*, vol. 32, no. 3, pp. 896–913, 2016.

[84] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[85] Y. P. Sibagariang, H. V. Sihombing, E. Y. Setyawan, K. Kishinami, and H. Ambarita, "The potency of solar energy on medan city of indonesia: Comparison of clear sky, satellite and field measurements," in *AIP Conference Proceedings*, vol. 2221, p. 070002, AIP Publishing LLC, 2020.

[86] A. Catalina and J. R. Dorronsoro, "NWP ensembles for wind energy uncertainty estimates," in *International Workshop on Data Analytics for Renewable Energy Integration*, vol. 10691 of *Lecture Notes in Computer Science*, pp. 121–132, Springer, 2017.