UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

20 21

# Master thesis

Regularización Laplaciana en el espacio dual para SVMs

Laplacian regularization in the dual space for SVMs

David López Ramos

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Master as Research and Innovation on Computational Intelligence and Interactive Systems

# MASTER THESIS

# Regularización Laplaciana en el espacio dual para SVMs
# Laplacian regularization in the dual space for SVMs

Author: David López Ramos
Advisor: Carlos María Alaíz Gudín
Ponente: José Ramón Dorronsoro Ibero

September 2020

**David López Ramos**
***Regularización Laplaciana en el espacio dual para SVMs***

***Laplacian regularization in the dual space for SVMs***

**David López Ramos**
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

*If people do not believe that mathematics is simple,*
*it is only because they do not realize how complicated life is.*

*John von Neumann*

# AGRADECIMIENTOS

A Carlos María, por animarme a cursar el máster, así como por la ayuda prestada durante el mismo.

A la Cátedra UAM-IIC en Ciencia de Datos y Aprendizaje Automático y a la Fundación de la UAM por la ayuda económica para el estudio.

Al Centro de Computación Científica (CCC) de la UAM, por el uso de las instalaciones.

# Resumen

El aprendizaje automático es un campo con un gran impacto en la actualidad por la utilidad que tiene a la hora de resolver muchos tipos de problemas. Sin embargo, hoy en día se manejan grandes cantidades de datos y por ello los métodos tradiciones de aprendizaje pueden estar muy limitados en rendimiento. Para abordar este problema se utiliza el aprendizaje regularizado, donde el objetivo es hacer el modelo lo más flexible posible pero conservando las propiedades de generalización, de forma que se evite el sobreajuste.

Hay muchos modelos que utilizan regularización en sus formulaciones, como Lasso, o modelos que utilizan una regularización intrínseca, como es el caso de la Support Vector Machine (SVM). En este último modelo, se maximiza el margen de un hiperplano separador, dando como resultado una solución que depende únicamente de un subconjunto de las muestras, los llamados vectores soporte.

Este Trabajo de Fin de Master tiene como objetivo desarrollar un modelo de SVM con regularización Laplaciana en el espacio dual, bajo la idea intuitiva de que patrones cercanos deberían tener coeficientes similares. Para construir el término Laplaciano nos basamos en el Fused Lasso, que penaliza las diferencias de los coeficientes consecutivos, pero en nuestro caso buscaremos penalizar las diferencias de todos contra todos, utilizando como pesos los elementos de la matriz de *kernel*.

Este documento presenta las diferentes fases llevadas a cabo en la implementación de la nueva propuesta a partir de la SVM estándard, además de experimentos comparativos entre el modelo novedoso y el método original. Como consecuencia, vemos que la regularización Laplaciana es de gran utilidad, ya que la nueva propuesta introducida vence en puntuación de test al modelo SVM estándard en la mayoría de los datasets utilizados, tanto en clasificación como en regresión. Además, observamos que si únicamente consideramos el término Laplaciano y fijamos el parámetro $C$ (cota superior para los coeficientes) como si fuera infinito, obtenemos también un mejor rendimiento que el método SVM estándard.

# Palabras clave

Aprendizaje Automático, Máquinas de Vectores Soporte, Regularización Laplaciana

# Abstract

Nowadays, Machine Learning (ML) is a field with a great impact because of its usefulness in solving many types of problems. However, today large amounts of data are handled and therefore traditional learning methods can be severely limited in performance. To address this problem, Regularized Learning (RL) is used, where the objective is to make the model as flexible as possible but preserving the generalization properties, so that overfitting is avoided.

There are many models that use regularization in their formulations, such as Lasso, or models that use intrinsic regularization, such as the Support Vector Machine (SVM). In this model, the margin of a separating hyperplane is maximized, resulting in a solution that depends only on a subset of the samples called support vectors.

This Master Thesis aims to develop an SVM model with Laplacian regularization in the dual space, under the intuitive idea that close patterns should have similar coefficients. To construct the Laplacian term we will use as basis the Fused Lasso model which penalizes the differences of the consecutive coefficients, but in our case we seek to penalize the differences between every pair of samples, using the elements of the kernel matrix as weights.

This thesis presents the different phases carried out in the implementation of the new proposal, starting from the standard SVM, followed by the comparative experiments between the new model and the original method. As a result, we see that Laplacian regularization is very useful, since the new proposal outperforms the standard SVM in most of the datasets used, both in classification and regression. Furthermore, we observe that if we only consider the Laplacian term and we set the parameter $C$ (upper bound for the coefficients) as if it were infinite, we also obtain better performance than the standard SVM method.

# Keywords

Machine Learning, Support Vector Machines, Laplacian Regularization

# Table of Contents

# Lists

## List of algorithms

## List of codes

## List of equations

## List of figures

# List of tables

1

# Introduction

In this chapter we are going to motivate the work done, detailing the objectives and structure of the Master Thesis.

## 1.1 Motivation

Machine Learning (ML) is a field with a big impact on today's society, with great advances that have allowed modifying the way of addressing many problems. Nevertheless, the large amount of data generated today has led to the big data paradigm, problems that have a large amount of information, which can affect the performance of traditional ML techniques. A first approach to this type of situation is Regularized Learning (RL), where the bias of the model is intentionally increased to reduce its variance, in order to prevent the overfitting, take advantage of some prior knowledge, impose structure on the models, etc. Support Vector Machine (SVM) is one of the most important regularization models, but there are other linear models like Lasso and Fused Lasso.

In this work we study the regularization techniques used in ML and then we propose a novel combination of the standard SVM and Fused Lasso to create a new model. Later, we see its performance in different problems, comparing with the original techniques.

## 1.2 Objectives

This Master Thesis aims to analyze in depth the most common regularization techniques in ML, and as a result propose a combination based on the ideas behind some of them. The objectives are:

**Review in detail the regularization of the linear models** The Lasso model [1], its extension to Group Lasso [2], the total variation regularization and the Fused Lasso model [3] and the Generalized Lasso model [4].

**Study of the models with intrinsic regularization** Standard SVM [5], the Least Squares Support Vector Machine (LS-SVM) [6] and the Laplacian SVM [7].

**Proposal and implementation of the new model** Include an additional regularization in an SVM, so that it penalizes the differences of the dual coefficients of the model, based on the coefficient penalty that the Fused Lasso method uses. For this regularization, the Laplacian is used, since it encodes the information of the weights associated to each link of the adjacency graph of the data. As a result, we have two SVM regularizers: the original, which tries to maximize the margin, and the new Laplacian regularizer, which controls the difference of the coefficients of nearby patterns.

**Experiments with the new proposal** Compare our new model against the standard SVM model to see if the Laplacian regularizer is useful.

## 1.3 Document structure

This document consists of the following chapters:

**Chapter 1** The present chapter, that provides a brief introduction to the project, explaining its objective.

**Chapter 2** Here we do a little review on the fundamentals of ML and later we explain RL models, such as SVM and Fused Lasso, approaches used as the basis for this work.

**Chapter 3** In this part, we review the procedure to pass from the primal problem to the dual problem. Then, we introduce the new Laplacian regularizer in the dual space to build the formulations of the new proposed method.

**Chapter 4** In the experiments we compare the new method based on Laplacian regularization against the SVM standard model. We also check the performance of a model with only the Laplacian regularization term, a particular case of our proposal.

**Chapter 5** Here the conclusions obtained from the previous experiments are presented. Subsequently, a list of tasks that could be carried out in the future are detailed.

# 2 Regularized learning and Support Vector Machines

In this chapter we describe the state of the art of this area. We start from a Machine Learning (ML) review in which the basis of regression linear models and neural networks are explained. Then we describe the methods with regularization, as Lasso and its modified approaches, which are important, specially Fused Lasso, for the novel combination that we propose in the next chapter. Later, we illustrate the standard Support Vector Machine (SVM) and its derived methods, such as Least Squares Support Vector Machine (LS-SVM) or Laplacian SVM, because this Laplacian operator is the key term of our new regularized SVM approach.

## 2.1 Machine learning review

ML is the study of computer algorithms that improve automatically through experience [8]. It is a field of artificial intelligence where the algorithms build a mathematical model based on training data in order to make predictions with the test data.

As notation, we indicate as $\mathbf{x}^T = (x_1, ..., x_p)$ the input vector, $y$ the output, $N$ the number of patterns and $p$ the number of dimensions. The goal is to predict the output based on the input, fitting a set of parameters that define the model.

### 2.1.1 Linear models

We are going to review linear models for classification and regression. Although these models were developed in the statistical age of precomputing they are still widely used because of their simplicity and because they offer an interpretation of how inputs affect output [9]. Furthermore, in some cases they perform better than nonlinear methods.

#### Linear models for classification

A linear binary classifier defines a hyperplane in the space which separates positive from negative examples. This problem uses the function

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} x_j\beta_j, \tag{2.1}$$

and then the prediction is

$$y = \text{sign}\,(f(\mathbf{x})) = \begin{cases} +1 & \text{if} f(\mathbf{x}) \geqslant 0, \\ -1 & \text{otherwise.} \end{cases} \tag{2.2}$$

The slope of the hyperplane is determined by $\beta_1, ..., \beta_p$ and the intercept is determined by the bias $\beta_0$. Classification can be approached as a probability estimation problem, so we can maximize the likelihood by

$$\max_{\boldsymbol{\theta}} \quad P_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{X}), \tag{2.3}$$

with $\boldsymbol{\theta}$ an unknown set of parameters. Because each sample is assumed to be independent of the others and also we can include a negative logarithm (since the minimum of the negative logarithm function is the maximum of that positive function) we can formulate the problem as

$$\min_{\boldsymbol{\theta}} \left( - \sum_{i=1}^{N} \log P_{\theta}(y_i|\mathbf{x}_i) \right). \tag{2.4}$$

If we interpret $\hat{y}_i$ as the probability that the $i$-th example belongs to the positive class and $1 - \hat{y}_i$ the probabiliy that it belongs to the negative class, then the optimization problem can be written as

$$\min \quad \ell(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i). \tag{2.5}$$

This loss function in commonly called **log loss** and is also referred to as binary **cross entropy**.

## Linear models for regression

In linear regression problems, the target variable is a numerical value and the model assumes that the function $\mathrm{E}(y|\mathbf{x})$ is linear at the inputs $x_1, ..., x_p$. Formally, we start from the input $\mathbf{x}^T = (x_1, ..., x_p)$ and we want to predict the real output $y$, by

$$f(\mathbf{x}) = \beta_0 + \sum_{j=1}^{p} x_j\beta_j, \tag{2.6}$$

with $\beta_j$ unknown parameters. One method to estimate this parameters is **least squares**, which minimizes the Residual Sum of Squares (RSS),

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{N} (y_i - f(x_i))^2 = \sum_{i=1}^{N} (y_i - \sum_{i=1}^{N} x_{ij}\beta_j)^2, \tag{2.7}$$

that we can express in matrix format

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{2.8}$$

Minimizing with respect to $\boldsymbol{\beta}$

$$\nabla_{\boldsymbol{\beta}} \mathrm{RSS} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}, \tag{2.9}$$

we obtain the estimation

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.10}$$

Notice that, in general, RSS can be viewed as an estimation of the Mean Squared Error (MSE) of an estimator $\hat{\boldsymbol{\theta}}$. The MSE is related with the prediction accuracy,

$$\mathrm{MSE}(\hat{\boldsymbol{\theta}}) = \mathrm{E}[(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta})]^2 = \mathrm{Var}(\hat{\boldsymbol{\theta}}) + [\mathrm{E}(\hat{\boldsymbol{\theta}}) - \boldsymbol{\theta}]^2. \tag{2.11}$$

As we can see, it is formed by a first term, known as the variance, and a second term, the bias, so we must find a balance between both quantities. Regarding this, the least squares estimation usually has a very small bias but a great variance, especially if there is a large dimension compared to the number of patterns, so sometimes it is convenient to remove some coefficients sacrificing a little bias to reduce the variance and the global prediction error.

Another problem of least squares is the interpretability. The resulting models are dense, they depend on all the variables, and not all of them may be informative, so it is more difficult to interpret the results. If we retain only a representative subset of the variables, the interpretability can be improved.

A first way of solving this problem is to use the **subset selection methods**, among which Best Subset Selection [10] and Forward/Backward Stepwise Regression [11] stand out. These methods analyze the effect of the different variables and choose the ones that provide the most information to the model.

Another way to make simpler models is to control the variance, using **regularization methods**, such as Least Angle Regression (LARS) [12], which can be seen as a stepwise selection, or the well-known Ridge Regression [13] and Lasso [1], which we will see in detail in the regularization section.

### 2.1.2 Neural Networks

A natural extension of the linear models are the **Neural Networks (NN)** [14,15]. [1] To apply the models on some complex problems, it is necessary to adapt the data of the datasets. The SVM, which we will see later, approaches this problem using functions that focus on training data and selecting a subset during the procedure, by a convex quadratic optimization. Moreover, the SVM uses the so called kernel trick to obtain a non-linear transformation. On the other hand, NN are based on learning a transforma-

[1] Notice that the NN are not used explicitly in ths work, although there are explained given their importance and for the sake of completeness.

tion directly from the data, using a bio-inspired architecture. A paradigm specially important of NN are Deep Neural Networks (DNN) [16], whose techniques have drawn ever-increasing research interests because of their inherent capability of overcoming the drawback of traditional algorithms dependent on hand-designed features. Deep learning approaches have been found to be suitable for big data analysis with successful applications to computer vision, pattern recognition, speech recognition, natural language processing and recommendation systems [17].

Mathematically, and assuming the classical Multilayer Perceptron (MLP) architecture of only one hidden layer, the output is computed as

$$y(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} \omega_{kj}^{(2)} h\left(\sum_{i=1}^{p} \omega_{ji}^{(1)} x_i + \omega_{j0}^{(1)}\right) + \omega_{k0}^{(2)}\right), \tag{2.12}$$

where $w^{(i)}$ are the weights (and bias) of the respective layers $i$, $h$ is the activation function (sigmoid, hyperbolic tangent, etc.) and $\sigma$ is the sigmoidal function for classification (for regression we would use the identity). To train the network, we can determine the parameters by minimizing the RSS for regression and cross-entropy (maximum likelihood) for classification.

When it comes to minimizing the error $\mathcal{E}(\mathbf{w})$ in this case is not possible to find an analytical solution of $\nabla\mathcal{E}(\mathbf{w}) = 0$, therefore, iterative methods are usually used. One of them is the **gradient descent**, which consists in updating the weights in the opposite direction to the gradient, looking for a minimum,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla\mathcal{E}(\mathbf{w}^{(\tau)}), \tag{2.13}$$

where $\eta$ is the learning rate, and the initial weights are (usually) random. To evaluate the gradient efficiently, the **backpropagation** [18] technique is used, where an iterative process is applied, calculating the derivates by propagating the errors of the last layer in the opposite direction to that of the network.

Also in neural networks, regularization is used to control the complexity of the model, for example by **weight decay** with the control parameter $\lambda$

$$\tilde{\mathcal{E}}(\mathbf{w}) = \mathcal{E}(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{2.14}$$

Other common methods are, for example, Gaussian prioris, tangent propagation, or early stopping (where training is stopped when the validation error begins to increase). Concerning regularization, it is also worth mentioning Convolutional Neural Network (CNN) [19], in which invariant models are created using invariance properties in the network structure itself, hence controlling its complexity.

## 2.2 Regularized linear models

As explained above, regularization in ML models is important to have some control over the complexity of the model. Moreover, when there are many correlated variables in a linear regression model, the coefficients may be poorly determined and there may be a great variance in the resulting model, a problem that is reduced by penalizing the size of the weights. Next we are going to explain several regularized linear models.

### 2.2.1 Ridge Regression

One of these methods is **Ridge Regression** [13], which penalizes the coefficients with the $\ell_2$ norm:

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \left( \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right), \tag{2.15}$$

with $\lambda \geqslant 0$ the complexity parameter, that controls which term is the most important. If this parameter is too large the coefficients are reduced towards $0$ because the penalization is strong. On the other hand, if $\lambda$ is too small the regularization has no effect and the coefficients tend to the solution of the linear model. Figure 2.1 shows the Ridge coefficient estimates for a prostate cancer example, plotted as functions of df($\lambda$), the effective degrees of freedom implied by the penalty $\lambda$. Here it is clear to see that the coefficients are reduced towards $0$ when df($\lambda$) decrease.

As in the case of the linear regression model seen before, we can solve this problem in matrix form minimizing the RSS

$$\mathrm{RSS}(\boldsymbol{\beta}, \lambda) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}, \tag{2.16}$$

and deriving respect to $\boldsymbol{\beta}$

$$\nabla_{\boldsymbol{\beta}}\,\mathrm{RSS} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + 2\lambda\mathbf{I}\boldsymbol{\beta} = 0 \Rightarrow (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} - \mathbf{X}^T\mathbf{y} = 0. \tag{2.17}$$

Finally, we obtain the solution

$$\hat{\boldsymbol{\beta}}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.18}$$

### 2.2.2 Lasso

Another well-known regularization method is **Lasso** [1], similar to Ridge but with important differences, since it uses the $\ell_1$ norm of the weights as a regularizer,

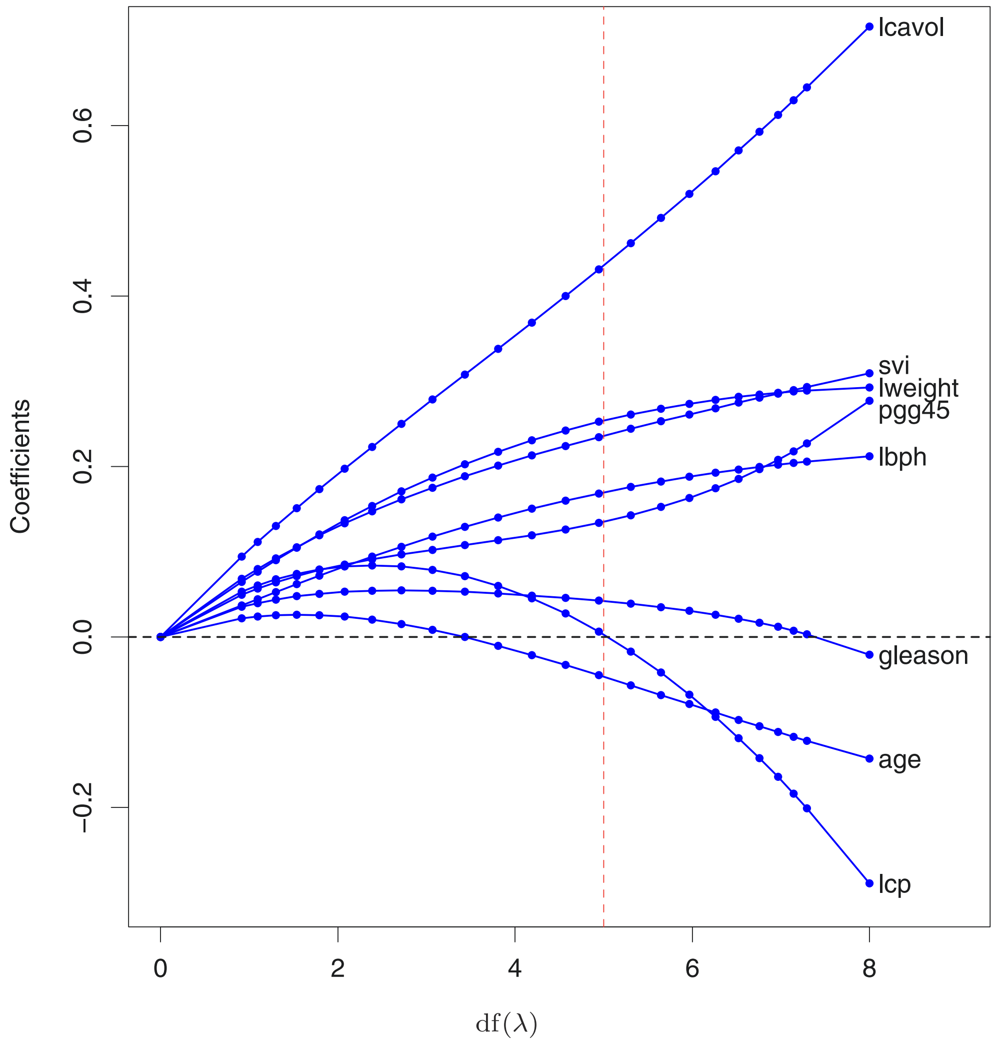


**Figure 2.1:** Figure 3.8 of [20]. Profiles of ridge coefficients for a prostate cancer example, as the tuning parameter $\lambda$ is varied. Coefficients are plotted versus df($\lambda$). The vertical line drawn at df $= 5.0$ is the value chosen by cross-validation.

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \left( \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right). \tag{2.19}$$

We can write it in the form with restrictions

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \left( \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right) \quad \text{subject to } \sum_{j=1}^{p} |\beta_j| \leqslant t, \tag{2.20}$$

with both formulations being equivalent for certain $\lambda$ and $t$. The constraint $\sum_{j=1}^{p} |\beta_j| \leqslant t$ causes the solution to be nonlinear in $y_i$. With $t$ small enough, some coefficients will be zero (it is a type of variable selection). The tuning parameter $t \geqslant 0$ controls the amount of shrinkage that is applied to the estimates. Let $\hat{\boldsymbol{\beta}}_j^{ls}$ be the full least squares estimates and let $t_0 = \sum |\hat{\boldsymbol{\beta}}_j^{ls}|$. Values of $t < t_0$ will cause reduction of coefficients towards $0$ and some of them may be exactly equal to $0$. Figure 2.2 shows the Lasso estimates as a function of bound $s = \frac{t}{\sum |\hat{\boldsymbol{\beta}}_j^{ls}|}$. Here, the curves decrease in a monotone fashion to $0$, but this does not always happen in general.

In Figure 2.3 we see the geometric comparison between Lasso and Ridge. Both methods find the first point where the elliptical contour intersects the constraint region. Unlike the disc, the diamond has corners; if the solution occurs in the corner, then some parameters $\beta_j = 0$. Another method similar to Lasso but earlier is Breiman's **non-negative garrote** [21].

Since the Lasso problem is not differentiable, iterative methods, for example, are used. Computation of the solution to equation (2.20) is a quadratic programming problem with linear inequality constraints. A wide variety of techniques from convex anaylisis and optimization theory have been developed to compute the solutions path of Lasso. These include coordinate descent [22], subgradient methods, LARS and proximal gradient methods [12].

### 2.2.3 Group Lasso

Lasso focuses on the selection of individual variables, rather than groups of them, so it sometimes chooses more factors than necessary. Moreover, there are cases where variables have a certain natural group structure, in which case it is interesting to obtain sparsity at that level, rather than at the level of separate coefficients. Therefore, the extension called **Group Lasso** [2] was proposed, which considers the problem of selecting groups of variables (factors) to improve prediction. If we use the $\ell_2$ norm for each of the $J$ groups, the estimation of Group Lasso is the solution of:

$$\hat{\boldsymbol{\beta}}^{\text{group-lasso}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^{p+1}}{\operatorname{argmin}} \left( \frac{1}{2} \|Y - \sum_{j=1}^{J} X_j \boldsymbol{\beta}_j\|^2 + \lambda \sum_{j=1}^{J} \|\boldsymbol{\beta}_j\|_2 \right), \tag{2.21}$$

with $\lambda \geqslant 0$ the regularization parameter. With this expression, sparsity is induced at the factor level, instead of the individual coefficients as in the case of Lasso.

Since the penalty reduces to an $\ell_2$ norm of the subspaces defined by each group, it cannot select out

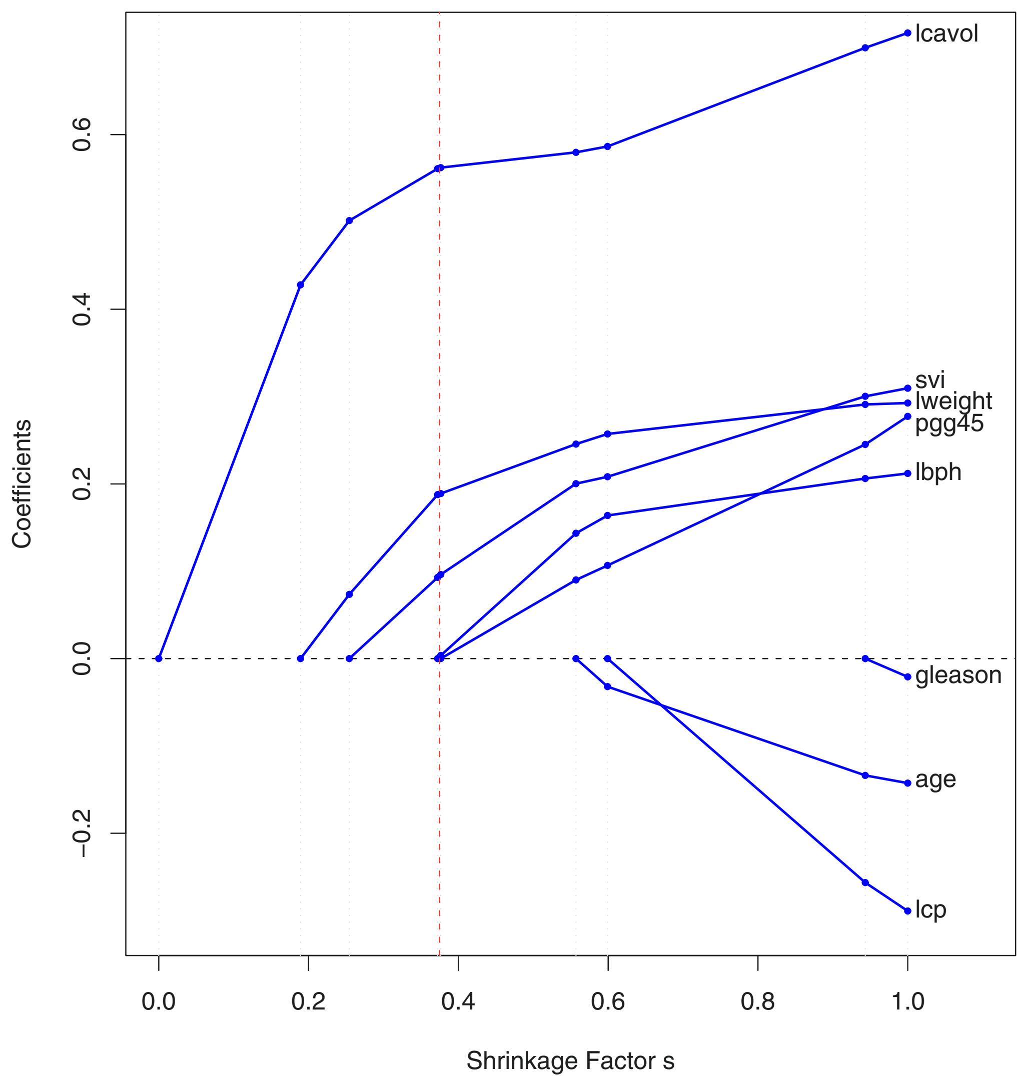


**Figure 2.2:** Figure 3.10 of [20]. Profiles of Lasso coefficientes, as the tuning parameter $t$ is varied, plotted versus $s = \frac{t}{\sum|\hat{\boldsymbol{\beta}}_j^{ls}|}$. The vertical line drawn is the value chosen by cross-validation. Compare Figure 2.1; the Lasso profiles hit zero, while those for Ridge do not. The solutions are piece-wise linear.

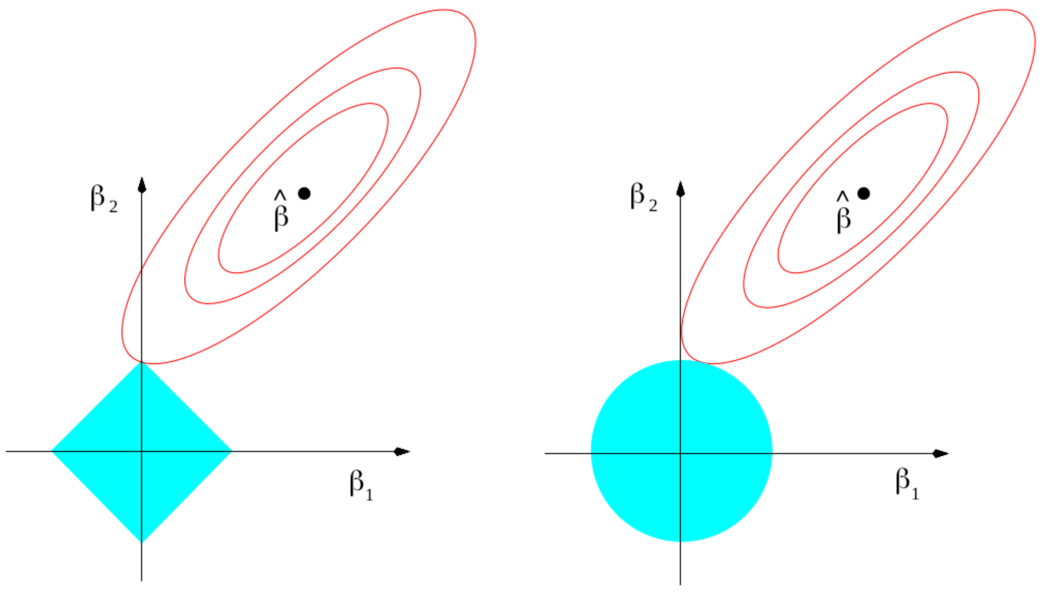


**Figure 2.3:** Figure 3.11 of [20]. Estimation picture for the Lasso (left) and Ridge Regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $\|\beta_1\| + \|\beta_2\| \leqslant t$ and $\beta_1^2 + \beta_2^2 \leqslant t_2$, respectively, while the red ellipses are the contours of the least squares error function.

only some of the covariates from a group (some $\ell_2$ norms are $0$ so all the coefficients are $0$). However, since the penalty is the sum over the different subspaces, as in the Lasso, the constraint has some non-differential points, corresponding to subspaces that are identically zero. Different extensions of Group Lasso are its sparse version [23] which can select individual covariates within a group by $\ell_1$ penalty; and Group Lasso with Overlap [24], which allows covariates to be shared between different groups.

In order to solve the Group Lasso, the algorithm cycles through the $J$ groups, and is a blockwise coordinate descent procedure [25].

### 2.2.4 Fused Lasso

Another extension of Lasso is **Fused Lasso** [3], which is a generalization for problems with characteristics that can be ordered and very useful when $p \gg N$, that is, when there are many more predictors than examples. Fused Lasso penalizes with the $\ell_1$ norm the coefficients and their successive differences, so its coefficients are given by:

$$\begin{aligned} \hat{\boldsymbol{\beta}}^{\text{fused-lasso}} &= \underset{\boldsymbol{\beta} \in \mathbb{R}^{p+1}}{\operatorname{argmin}} \left( \sum_i (y_i - \sum_j x_{ij}\beta_j)^2 \right) \\ \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leqslant s_1 \quad &\text{and} \quad \sum_{j=2}^{p} |\beta_j - \beta_{j-1}| \leqslant s_2. \end{aligned} \tag{2.22}$$

In the expression (2.22) the first constraint encourages sparsity in the coefficients and the second encourages sparsity in their differences, i.e. flatness of the coefficient $\beta_j$ as a function of $j$, producing a piecewise constant solution. This second term is based on the expression $\sum_j \|\beta_j - \beta_{j-1}\|^{\alpha} \leqslant s_2$ called Fusion term [26], proposed for various values of $\alpha$, especially $\alpha = 0, 1, 2$. The values $s_1$ and $s_2$ are the constraint limits, which can be set or restricted to a search for values in a grid, but taking into account that for $p$ very large the problem is computationally very expensive. In Figure 2.4 we see the geometry of Fused Lasso, where the solution is given by the intersection of the constraint areas.

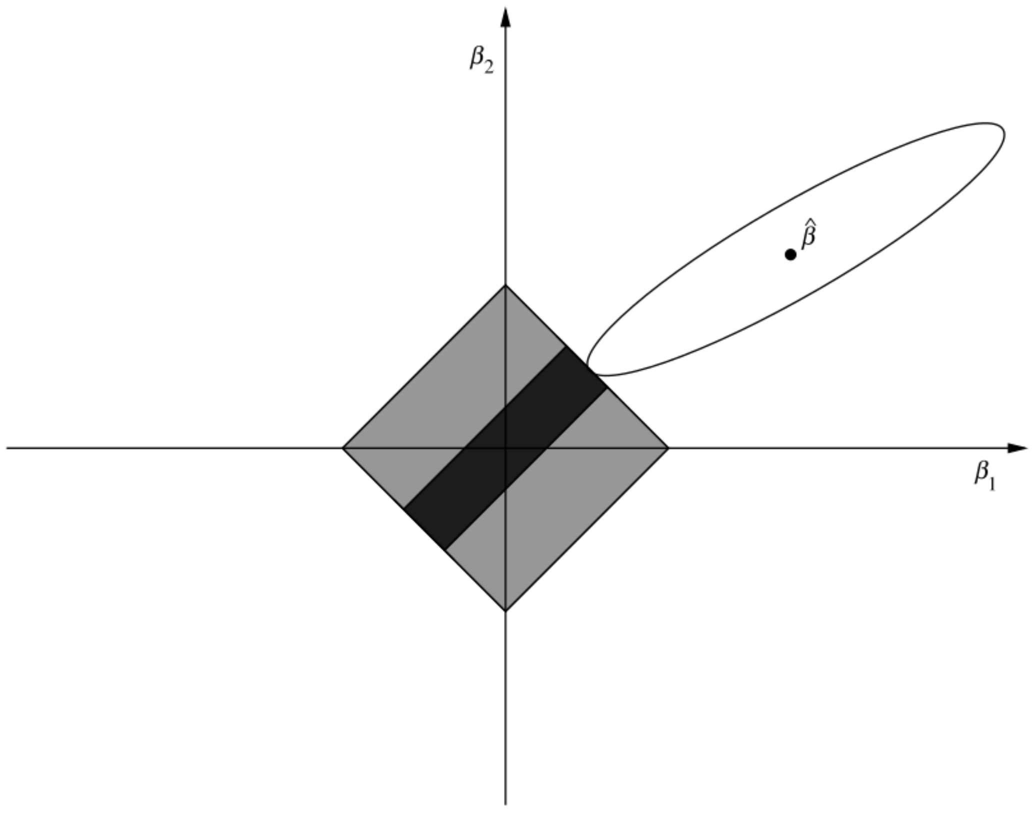


**Figure 2.4:** Figure 2 of Fused Lasso paper [3]. Shown are the contours of the error and constraint areas, $s_1$ (grey) and $s_2$ (black).

Figure 2.5 illustrates a simulated example with $100$ predictors and $10$ samples generated with Gaussian distribution and we can see the comparison in performance of Lasso, Fusion and Fused Lasso estimating the true underlying coefficients. Here, Lasso performs poorly, Fusion reasonably captures the plateau and Fused Lasso does a good job overall.

Fused Lasso has a similar sparsity property to Lasso. Instead of applying to the number of non-zero coefficients, however, the sparsity property applies to the number of sequences of non-zero coefficients. In Figure 2.6 we can see a prostate cancer example where sparsity of Lasso implies certain maximum number of non-zero dots and sparsity of Fused Lasso implies certain maximum number of non-zero sequences of consecutive feature values with the same coefficient.

There are many algorithms to optimize Fused Lasso problem, and some of them can solve it exactly

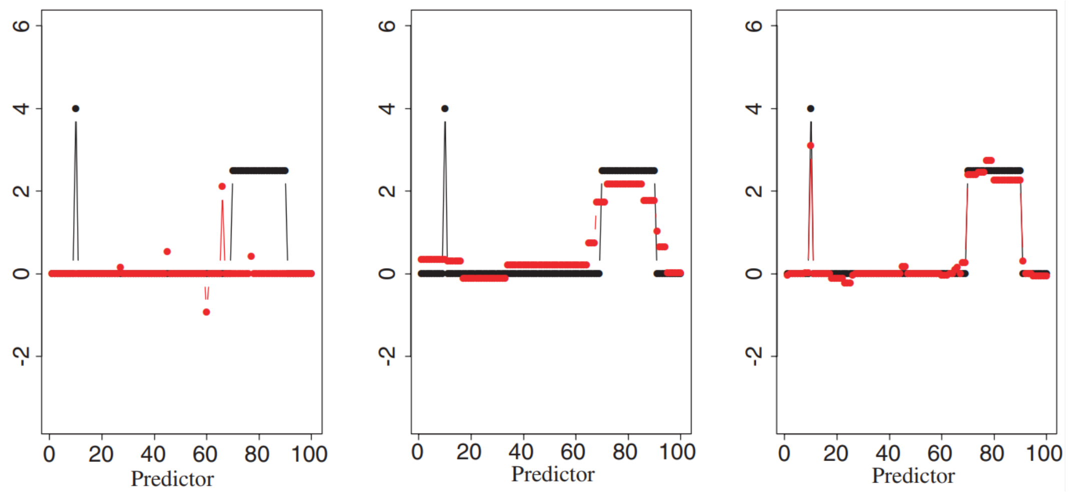


**Figure 2.5:** Figure 4 of Fused Lasso paper [3]. Simulated example with only two areas of non-zero coefficients (black points and lines; red points estimated coefficients from each method): Lasso (left), Fusion (center) and Fused Lasso (right).

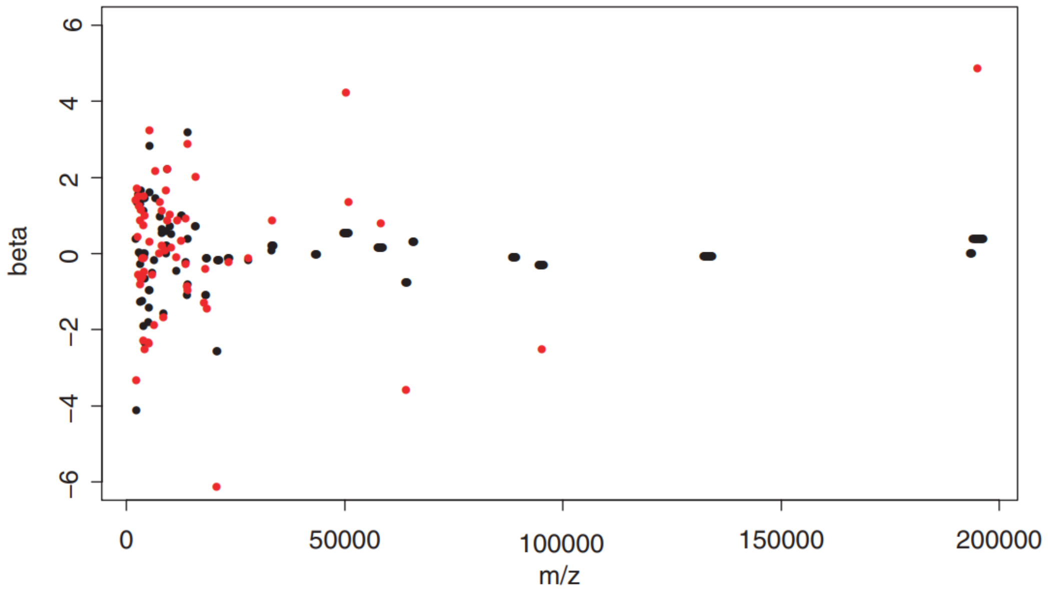


**Figure 2.6:** Figure 8 of Fused Lasso paper [3]. Results for the prostate cancer example: Fused Lasso non-zero coefficients (black) and Lasso non-zero coefficients (red).

in a finite number of operations [27].

### 2.2.5 Generalized Lasso

In the Fused Lasso model, differences of consecutive coefficients are penalized. However, this can be generalized to penalize differences given by a graph as done in the Graph Fused Lasso (GFL) [28]. This generalization is the idea under the **Generalized Lasso** method [4], which penalizes the $\ell_1$ norm of the coefficients transformed using a matrix $\mathbf{D}$. The goal behind this method is to force certain structural constraints instead of a pure sparsity of the coefficients. The problem is

$$\hat{\boldsymbol{\beta}}^{\text{gen-lasso}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left( \frac{1}{2} \|y - X\boldsymbol{\beta}\|_2^2 + \lambda \|\mathbf{D}\boldsymbol{\beta}\|_1 \right), \tag{2.23}$$

where $\mathbf{D} \in \mathbb{R}^{m \times p}$ is a specific penalty matrix. Several choices of $\mathbf{D}$ give known problems, as is the case with Fused Lasso. Here, $\mathbf{D}$ is a $(p-1) \times p$ matrix with $\mathbf{D}_{ii} = 1$, $\mathbf{D}_{i+1,i} = -1$ and $\mathbf{D}_{ij} = 0$ otherwise so that it penalizes the differences of consecutive coefficients. Note that if $\mathbf{D} \in \mathbb{R}^{p \times p}$ is invertible, we can go from Generalized Lasso to Lasso with $\boldsymbol{\theta} = \mathbf{D}\boldsymbol{\beta}$.

The Generalized Lasso problem (2.23) is difficult to analize directly because the non-differentiable $\ell_1$ penalty. So usually the corresponding Lagrange dual problem is used instead, making the calculation easier.

The structural constraint idea of Generalized Lasso and GFL is what we will use to define the new Laplacian regularizer in Section 3.2, in which we will take the Laplacian operator as regularizer, where we will define the weights using the kernel.

## 2.3 Support Vector Machines

In this section we will review the SVM model, on which our proposal is based. We will see the procedure to convert the primal problem into the dual problem and we detail the new Laplacian regularization.

### 2.3.1 Standard SVM

In the late 20th century, **Support Vector Machine** [5] was introduced to solve pattern recognition problems, using a transformation of the data to a larger dimension space to construct a separating hyperplane, maximizing its margin. The transformations and parameters are chosen to minimize the Vapnik-Chervonenkis (VC) dimension, which measures the complexity of the models. Given a training set $(y_k, \mathbf{x}_k)_k^N$ where $\mathbf{x}_k \in \mathbb{R}^p$, $y_k \in \mathbb{R}$, SVM seeks to construct a classifier of the form

$$y(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{N} \alpha_k y_k K(\mathbf{x}, \mathbf{x}_k) + b\right), \tag{2.24}$$

where $\alpha_k$ are positive real constants, $b$ a real constant and $K$ a kernel function. The model is trained by maximizing the margin of the separating hyperplane (as described in detail in Section 3.1), which lead to the following **classification dual problem**

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & Q(\boldsymbol{\alpha}; K) = -\frac{1}{2}\sum_{k,l=1}^{N} y_k y_l K(\mathbf{x}_k, \mathbf{x}_l)\alpha_k \alpha_l + \sum_{k=1}^{N} \alpha_k, \\ \text{subject to} \quad & \sum_{k=1}^{N} \alpha_k y_k = 0, \quad 0 \leqslant \alpha_k \leqslant C, \quad k = 1, ..., N, \end{aligned} \tag{2.25}$$

where $C$ is a control parameter and $K(\mathbf{x}_k, \mathbf{x}_l) = \varphi(\mathbf{x}_k)^T \varphi(\mathbf{x}_l)$. Thanks to the kernel trick it is not necessary to explicitly calculate the transformation to the larger dimension space. The solution obtained by the standard SVM is sparse in terms of $\mathbf{x}_i$, because only some of $\alpha_i$ are distinct to $0$ and are called **support vectors**.

In regression problems, we can use the $\varepsilon$-SVM that ignores errors that are smaller than $\varepsilon$. In Figure 2.7 we can see a linear SVM for regression, with the loss function that depends on $\varepsilon$. Only the points outside the shaded region contribute to the cost insofar, as the deviations are penalized in a linear fashion. The prediction of the model is

$$y(\mathbf{x}) = \sum_{k=1}^{N} (\alpha_k - \alpha_k^*) K(\mathbf{x}, \mathbf{x}_k) + b. \tag{2.26}$$

Like in classification, this problem is more easily solved in the dual space, obtaining the **regression dual problem**

$$\begin{aligned} \max_{\boldsymbol{\alpha}^{(*)}} \quad & Q(\boldsymbol{\alpha}^{(*)}; K) = -\frac{1}{2}\sum_{k,l=1}^{N} (\alpha_k - \alpha_k^*)(\alpha_l - \alpha_l^*) K(\mathbf{x}_k, \mathbf{x}_l) - \varepsilon \sum_{k=1}^{N} (\alpha_k + \alpha_k^*) + \sum_{k=1}^{N} y_k (\alpha_k - \alpha_k^*), \\ \text{subject to} \quad & \sum_{k=1}^{N} (\alpha_k - \alpha_k^*) = 0, \quad 0 \leqslant \alpha_k^{(*)} \leqslant C, \quad k = 1, ..., N. \end{aligned} \tag{2.27}$$

Training an SVM requires the solution of a very large Quadratic Programming (QP) optimizaton problem. However, to solve this issue, Sequential Minimal Optimization (SMO) algorithm was developed [29], which breaks this problem into a series of smallest possible QP problems, that are solved analytically. Nowadays, it is widely used for training an SVM and is implemented by the popular LIBSVM library [30].

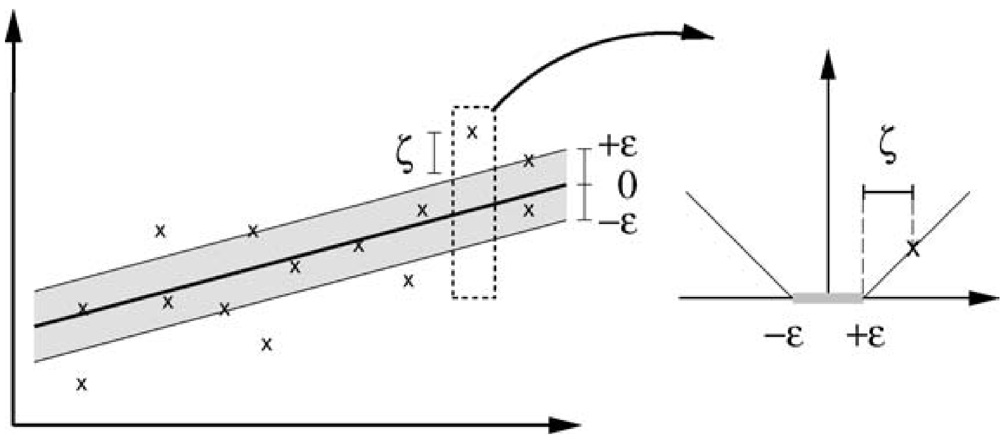


**Figure 2.7:** Figure 1 of SVM tutorial [5]. The image shows the $\varepsilon$-insensitive loss function.

## 2.3.2 Least Squares Support Vector Machine

One variant of the standard SVM is the **LS-SVM** [6]. In this case, the errors are penalized using the RSS instead of the hinge loss or the $\epsilon$-insensitive loss. As a result, the solution is obtained by solving a linear system of equations instead of a quadratic programming. Furthermore, while in classical SVM many support values (other than support vectors) were $0$, in LS-SVM the support values are proportional to the errors.

Mathematically, we formulate the problem as

$$\begin{aligned}&\min_{\mathbf{w},b,\mathbf{e}} \quad J(\mathbf{w},b,\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{k=1}^{N} e_k^2, \\ &\text{subject to} \quad y_k(\mathbf{w}^T\varphi(\mathbf{x}_k)+b) = 1 - e_k, \quad k = 1,...,N.\end{aligned} \tag{2.28}$$

We define the Lagrangian

$$L(\mathbf{w},b,e;\boldsymbol{\alpha}) = J(\mathbf{w},b,e) - \sum_{k=1}^{N}\alpha_k[y_k(\mathbf{w}^T\varphi(\mathbf{x}_k)+b) - 1 + e_k], \tag{2.29}$$

where $\alpha_k$ are Lagrange multipliers (which can be positive or negative due to the equality constraint and KKT conditions). The optimality conditions are

$$\begin{aligned}&\nabla_{\mathbf{w}} L = 0 \rightarrow \mathbf{w} = \sum_{k=1}^{N}\alpha_k y_k \varphi(\mathbf{x}_k), \\ &\partial_b L = 0 \rightarrow \sum_{k=1}^{N}\alpha_k y_k = 0, \\ &\partial_{e_k} L = 0 \rightarrow \alpha_k = \gamma e_k, \quad k = 1,...,N, \\ &\partial_{\alpha_k} L = 0 \rightarrow y_k(\mathbf{w}^T\varphi(\mathbf{x}_k)+b) - 1 + e_k = 0, \quad k = 1,...,N,\end{aligned} \tag{2.30}$$

which can be written as the solution of the linear system of equations

$$\left(\begin{array}{ccc|c} \mathbf{I} & 0 & 0 & -\mathbf{Z}^T \\ 0 & 0 & 0 & -\mathbf{Y}^T \\ 0 & 0 & \gamma\mathbf{I} & -\mathbf{I} \\ \hline \mathbf{Z} & \mathbf{Y} & \mathbf{I} & -0 \end{array}\right)\left(\begin{array}{c} \mathbf{w} \\ b \\ \mathbf{e} \\ \hline \boldsymbol{\alpha} \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline \bar{\mathbf{1}} \end{array}\right), \tag{2.31}$$

where $\mathbf{Z} = (\varphi(\mathbf{x}_1)^T y_1, ..., \varphi(\mathbf{x}_N)^T y_N)^T$, $\mathbf{Y} = (y_1, ..., y_N)^T$, $\mathbf{e} = (e_1, ..., e_N)^T$, $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_N)^T$ and $\bar{\mathbf{1}}$ is a vector of ones. The solution is given by

$$\left(\begin{array}{c|c} 0 & -\mathbf{Y}^T \\ \hline \mathbf{Y} & \mathbf{Z}\mathbf{Z}^T + \gamma^{-1} I \end{array}\right)\left(\begin{array}{c} b \\ \hline \boldsymbol{\alpha} \end{array}\right) = \left(\begin{array}{c} 0 \\ \hline \bar{\mathbf{1}} \end{array}\right). \tag{2.32}$$

The kernel trick can be applied to compute the matrix $\boldsymbol{\Omega} = \mathbf{Z}\mathbf{Z}^T$, where

$$\Omega_{kl} = y_k y_l \varphi(\mathbf{x}_k)^T \varphi(\mathbf{x}_l) = y_k y_l K(\mathbf{x}_k, \mathbf{x}_l). \tag{2.33}$$

### 2.3.3 Laplacian SVM

Within the SVM framework, a new family of learning algorithms based on a regularization that allows exploiting the geometry of the marginal distribution was proposed, **Laplacian SVM** [7]. Therefore, we will have two regularizers, one that controls the complexity of the classifier and the other that controls the complexity as measured by the geometry of the distribution.

Laplacian SVM is framed in the context of semi-supervised learning [31], problems without labels for all the data, where the Laplacian SVM tries to take advantage of the information in the patterns without labels by introducing a new regularizer. It is clear to see that the collection of labeled data is more complicated than that of unlabeled data. As a result, a pattern recognition approach that is able to make better use of unlabeled data to improve performance is of great practical importance. Most natural learning occurs in the semi-supervised setting. Figure 2.8 shows how unlabeled examples may force us to restructure our hypotheses during learning. On the left we have two labeled examples (one negative and one positive). A natural choice for classification is to use a linear separator, as shown. However, we could have more examples, in this case, not labeled, as we see on the right. Now we can visualize a particular geometry in which the most suitable classifier would be circular. This is the intuition of the method, in which success depends on extracting a certain structure from the marginal distribution.

Let us state the standard sample learning framework. There is a probability distribution in $X \times \mathbb{R}$ with respect to which the examples $(\mathbf{x}, y)$ have been generated. The unlabeled data is $\mathbf{x} \in X$ generated

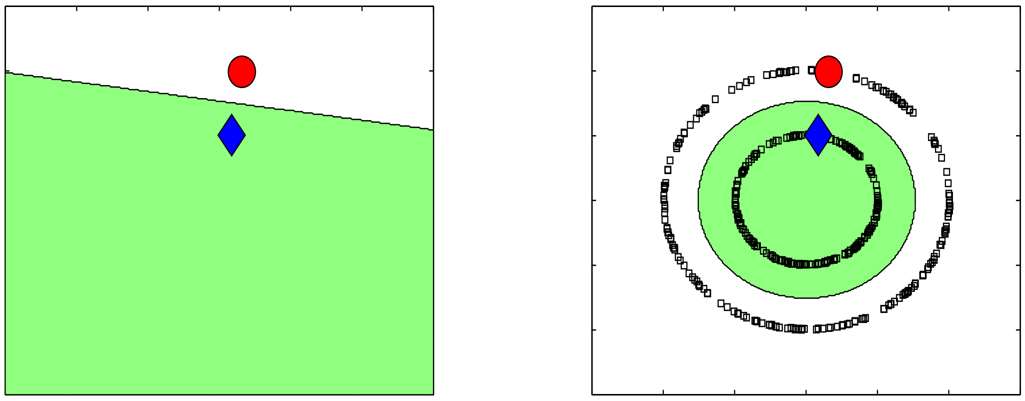


**Figure 2.8:** Figure 1 of Laplacian SVM original paper [7]. The picture shows a binary classification problem, with one case with only labeled data (left) and other case with labeled and unlabeled data (right).

according to a marginal distribution. We will assume that if two points are close in the intrinsic geometry given by the marginal, then the conditional distributions are similar. For a Mercer kernel $K : X \times X \to \mathbb{R}$, a reproducing kernel Hilbert space (RKHS) $H_k$ of functions $X \to \mathbb{R}$ is associated with the corresponding inner product $\| \cdot \|_K$. Given labeled data $(\mathbf{x}_i, y_i)$, $i = 1, ..., l$, the standard framework estimates an unknown function by minimizing

$$f^* = \underset{f \in H_k}{\operatorname{argmin}} \left( \frac{1}{l} \sum_{i=1}^{l} V(\mathbf{x}_i, y_i, f) + \gamma \|f\|_K^2 \right), \tag{2.34}$$

where $V$ is some function loss. By penalizing the RKHS norm, we achieve smooth conditions in possible solutions. The classic Representater Theorem says that the solution of this minimization exists in $H_k$ and is

$$f^* = \sum_{i=1}^{l} \alpha_i K(\mathbf{x}_i, \mathbf{x}). \tag{2.35}$$

Therefore the problem is reduced to optimizing over a finite space of coefficients $\alpha_i$, which is the basis of SVM, regularized least squares, and other regression and classification schemes.

The marginal distribution can be known or unknown. In most applications, it is unknown. Therefore, we must make empirical estimates of the marginal, for which we only need unlabeled data.

Thus, given a set of $l$ labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{l}$ and a set of $u$ unlabeled examples $\{\mathbf{x}_j\}_{j=l+1}^{j=l+u}$, by including an intrinsic smoothness penalty term, we can use the Representer Theorem and extend the SVM by solving the **primal problem for Laplacian SVM**:

$$\begin{aligned}
&\min_{\boldsymbol{\alpha}\in\mathbb{R}^{l+u},\boldsymbol{\xi}\in\mathbb{R}^{l}} \quad \frac{1}{l}\sum_{i=1}^{l}\xi_i + \gamma_A \boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} + \frac{\gamma_I}{(u+l)^2}\boldsymbol{\alpha}^T\mathbf{KLK}\boldsymbol{\alpha}, \\
&\text{subject to} \begin{cases} y_i(\sum_{j=1}^{l+u}\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)+b) \geqslant 1-\xi_i \quad i=1,...,l \\ \xi_i \geqslant 0 \quad i=1,...,l, \end{cases}
\end{aligned} \tag{2.36}$$

where $\gamma_A$ controls the complexity of the function in the ambient space, $\gamma_I$ controls the complexity of the function in the intrinsic geometry of the marginal, $\mathbf{L}$ is the Laplacian operator and $\xi_i$ are the slack variables. The Laplacian matrix, $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix and $\mathbf{A}$ is the adjacency matrix, is a representation of a graph, which can be used to find many useful properties of its structure.

Now, we can introduce the Lagrange multipliers $\beta_i, \zeta_i$ obtaining

$$\begin{aligned}
L(\boldsymbol{\alpha},\boldsymbol{\xi},b,\boldsymbol{\beta},\boldsymbol{\zeta}) = &\frac{1}{l}\sum_{i=1}^{l}\xi_i + \frac{1}{2}\boldsymbol{\alpha}^T(2\gamma_A\mathbf{K} + 2\frac{\gamma_A}{(l+u)^2}\mathbf{KLK})\boldsymbol{\alpha} \\
&- \sum_{i=1}^{l}\beta_i(y_i(\sum_{j=1}^{l+u}\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)+b) - 1 + \xi_i) - \sum_{i=1}^{l}\zeta_i\xi_i.
\end{aligned} \tag{2.37}$$

The optimality conditions are

$$\begin{aligned}
\partial_b L = 0 &\Rightarrow \sum_{i=1}^{l}\beta_i y_i = 0, \\
\partial_{\xi_i} L = 0 &\Rightarrow \frac{1}{l} - \beta_i - \zeta_i = 0, \\
&\Rightarrow 0 \leqslant \beta_i \leqslant \frac{1}{l} \quad (\xi_i, \zeta_i \geqslant 0).
\end{aligned} \tag{2.38}$$

After substituting in the Lagrangian, we obtain the **dual problem for Laplacian SVM**:

$$\begin{aligned}
&\boldsymbol{\beta}^* = \max_{\boldsymbol{\beta}\in\mathbb{R}^{l}} \quad \sum_{i=1}^{l}\beta_i - \frac{1}{2}\boldsymbol{\beta}^T\mathbf{Q}\boldsymbol{\beta} \\
&\text{subject to} \begin{cases} \sum_{i=1}^{l}\beta_i y_i = 0 \\ 0 \leqslant \beta_i \leqslant \frac{1}{l} \quad i=1,...,l, \end{cases}
\end{aligned} \tag{2.39}$$

where $\mathbf{Q} = \mathbf{YJK}(2\gamma_A\mathbf{I} + 2\frac{\gamma_I}{(l+u)^2}\mathbf{LK})^{-1}\mathbf{J}^T\mathbf{Y}$, with $\mathbf{J} = [\mathbf{I} \quad 0]$ an $l \times (l+u)$ matrix, $\mathbf{I}$ an $l \times l$ identity matrix and $\mathbf{Y} = \text{diag}(y_1, y_2, ..., y_l)$. Note that when $\gamma_I = 0$, we get coefficients of expansion to $0$ on the unlabeled data. The expansion of coefficients on the labeled data and the matrix $\mathbf{Q}$ are those of the standard SVM in this case, as it was to be expected.

The Laplacian used here penalizes the $f$ of the RKHS. In Chapter 3 we will use another approach closer to that of Fused Lasso (in fact, GFL), where we will penalize the differences between each pair of coefficients, using the kernel matrix as weights.

3

# Fused SVM

In this chapter we introduce the new SVM model with Laplacian regularization. First, we review the procedure to convert the primal problem to the dual problem in order to understand the formula in the dual space. Later, we start from this formula and we can add a new regularization Laplacian term to penalize the differences of the dual coefficients, following the idea of Fused Lasso. As a result, we finally obtain the formulas of the new methods proposed to improve the standard classification and regression SVM formulations.

## 3.1 Preliminaries: duality of the SVM problems

We are going to review the procedure to convert the primal problem into the dual problem, both in Support Vector Classifier (SVC) and in Support Vector Regressor (SVR). We use the notation defined in Section 2.1.

### 3.1.1 Support Vector Classifier

In the case of a binary classification problem, SVC tries to find a vector $\mathbf{w} \in \mathbb{R}^p$ and a bias $b \in \mathbb{R}$ such that the prediction given by sign $\left(\mathbf{w}^T\varphi(\mathbf{x}) + b\right)$ is correct for most samples. The classifier has the following constraints:

$$\begin{aligned} \mathbf{w}^T\varphi(\mathbf{x}_k) + b \geqslant +1 \text{ if } y_k = +1, \\ \mathbf{w}^T\varphi(\mathbf{x}_k) + b \leqslant -1 \text{ if } y_k = -1, \end{aligned} \tag{3.1}$$

and in an equivalent form:

$$y_k(\mathbf{w}^T\varphi(\mathbf{x}_k) + b) \geqslant 1 \quad k = 1, ..., N, \tag{3.2}$$

where $\varphi$ is a nonlinear function that transforms the original space into a higher dimension space. However, this function is not explicitly constructed, thanks to the kernel trick, $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$. To allow certain errors, the slack variables $\xi_k$ are introducted.

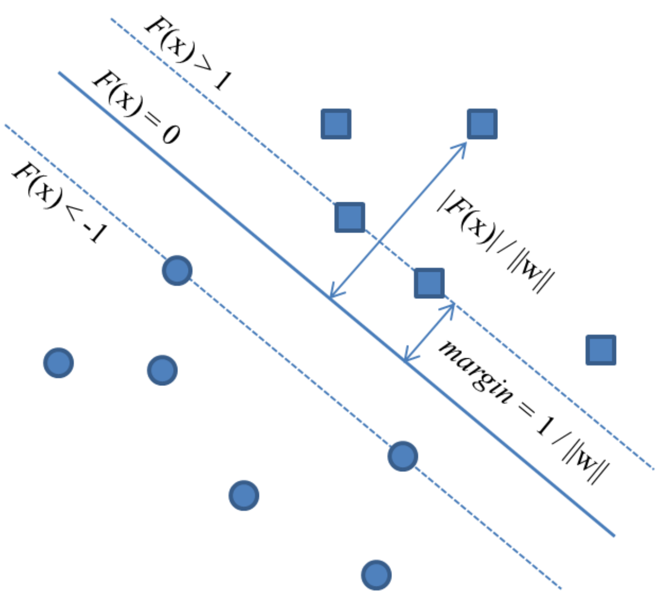


**Figure 3.1:** Figure 2 of [32]. SVC problem: the hyperplane maximizing the margin in a two-dimensional space.

The margin is defined to be the distance between the hyperplane $\mathbf{w}^T\varphi(\mathbf{x}) + b = 0$ and the hyperplane $\mathbf{w}^T\varphi(\mathbf{x}) + b = 1$. Reviewing the formula for the distance $d$ between two parallel hyperplanes with $b_1$ and $b_2$ constant terms, we get

$$d = \frac{|b_2 - b_1|}{\|\mathbf{w}\|} = \frac{|1|}{\|\mathbf{w}\|}. \tag{3.3}$$

Therefore, we want to maximize the margin $\frac{1}{\|\mathbf{w}\|}$ that is equivalent to minimizing $\|\mathbf{w}\|$. Figure 3.1 illustrates the geometry of a binary SVC problem. In addition, we want to penalize the errors with a constant $C$, obtaining the **classification primal problem**:

$$\boxed{\begin{aligned} &\min_{\mathbf{w},\xi_k} \quad J_1(\mathbf{w},\xi_k) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^{N}\xi_k, \\ &\text{subject to} \begin{cases} y_k(\mathbf{w}^T\varphi(\mathbf{x}_k) + b) \geqslant 1 - \xi_k, & k = 1, ..., N, \\ \xi_k \geqslant 0, \quad k = 1, ..., N. \end{cases} \end{aligned}} \tag{3.4}$$

With the objective function and constraints, we can construct the Lagrangian

$$L_1(\mathbf{w}, b, \xi_k; \alpha_k, \nu_k) = J_1(\mathbf{w},\xi_k) - \sum_{k=1}^{N}\alpha_k[y_k(\mathbf{w}^T\varphi(\mathbf{x}_k) + b)] - 1 + \xi_k] - \sum_{k=1}^{N}\nu_k\xi_k, \tag{3.5}$$

where $\alpha_k, \nu_k \geqslant 0$. The solution is given by the saddle point,

$$\max_{\alpha_k,\nu_k} \min_{\mathbf{w},b,\xi_k} L_1(\mathbf{w}, b, \xi_k; \alpha_k, \nu_k). \tag{3.6}$$

We construct the critical points by equating the derivates to $0$ with respect to the primal variables, obtaining the optimality conditions

$$\begin{aligned}
&\nabla_{\mathbf{w}} L_1 = 0 \rightarrow \partial_{\mathbf{w}} J_1(\mathbf{w},\xi_k) - \sum_{k=1}^{N} \alpha_k y_k = 0 \rightarrow \mathbf{w} = \sum_{k=1}^{N} \alpha_k y_k \varphi(\mathbf{x}_k),\\
&\partial_b L_1 = 0 \rightarrow \sum_{k=1}^{N} \alpha_k y_k = 0,\\
&\partial_{\xi_k} L_1 = 0 \rightarrow \partial_{\xi_k} J_1(\mathbf{w},\xi_k) - \alpha_k - \nu_k = 0 \rightarrow \quad \alpha_k = C - \nu_k, \quad 0 \leqslant \alpha_k \leqslant C, \quad k = 1,...,N.
\end{aligned} \tag{3.7}$$

Substituting the previous results in the Lagrangian,

$$\begin{aligned}
L_1(\mathbf{w},b,\xi_k;\alpha_k,\nu_k) &= \frac{1}{2}(\sum_{k=1}^{N} \alpha_k y_k \varphi(\mathbf{x}_k))^T(\sum_{k=1}^{N} \alpha_k y_k \varphi(\mathbf{x}_k)) + C\sum_{k=1}^{N}\xi_k -\\
&- \sum_{k=1}^{N} \alpha_k [y_k((\sum_{k=1}^{N} \alpha_k y_k \varphi(\mathbf{x}_k))^T\varphi(\mathbf{x}_k) + b)] + \sum_{k=1}^{N}\alpha_k - \sum_{k=1}^{N}\alpha_k\xi_k - \sum_{k=1}^{N}\nu_k\xi_k =\\
&= \frac{1}{2}\sum_{k,l=1}^{N} y_k y_l \varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l)\alpha_k\alpha_l - \sum_{k,l=1}^{N} y_k y_l \varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l)\alpha_k\alpha_l + \sum_{k=1}^{N}\alpha_k -\\
&- b\sum_{k=1}^{N}\alpha_k y_k + C\sum_{k=1}^{N}\xi_k - \sum_{k=1}^{N}(\alpha_k + \nu_k)\xi_k =\\
&= -\frac{1}{2}\sum_{k,l=1}^{N} y_k y_l \varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l)\alpha_k\alpha_l + \sum_{k=1}^{N}\alpha_k.
\end{aligned} \tag{3.8}$$

Now, we can apply the kernel trick and we obtain the **classification dual problem**:

$$\boxed{\begin{aligned}
&\min_{\boldsymbol{\alpha}} \quad Q_1(\boldsymbol{\alpha};K) = \frac{1}{2}\sum_{k,l=1}^{N} y_k y_l K(\mathbf{x}_k,\mathbf{x}_l)\alpha_k\alpha_l - \sum_{k=1}^{N}\alpha_k,\\
&\text{subject to} \begin{cases} \sum_{k=1}^{N}\alpha_k y_k = 0, \quad k = 1,...,N,\\ 0 \leqslant \alpha_k \leqslant C, \quad k = 1,...,N. \end{cases}
\end{aligned}} \tag{3.9}$$

Finally, the prediction of the model is:

$$y(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{N}\alpha_k y_k K(\mathbf{x},\mathbf{x}_k) + b\right), \tag{3.10}$$

because $\mathbf{w} = \sum_{k=1}^{N}\alpha_k y_k \varphi(\mathbf{x}_k)$ and $K(\mathbf{x}_i,\mathbf{x}_j) = \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$ as we have seen before.

### 3.1.2 Support Vector Regressor

In the $\varepsilon$-SVR the goal is to find a function $f(\mathbf{x})$ that has at most $\varepsilon$ deviation of the targets $y_k$, and if it is greater than $\varepsilon$ it will be penalized, so we allow certain errors by the slack variables $\xi,\xi^*$, depending on whether their predictions lie above or below the $\varepsilon$-tube. We start with the linear functions $f$ of the form $f(\mathbf{x}) = \mathbf{w}^T\varphi(\mathbf{x}) + b$, where $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$. Again, we want to maximice the margin or equivalently, minimize $\|\mathbf{w}\|$, obtaining, as a result, the **regression primal problem**:

$$\boxed{\begin{aligned} &\min_{\mathbf{w},\xi_k,\xi_k^*} \quad J_2(\mathbf{w},\xi_k,\xi_k^*) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^N(\xi_k+\xi_k^*),\\ &\text{subject to} \begin{cases} y_k - \mathbf{w}^T\varphi(\mathbf{x}_k) - b \leqslant \varepsilon + \xi_k,\\ \mathbf{w}^T\varphi(\mathbf{x}_k) + b - y_k \leqslant \varepsilon + \xi_k^*,\\ \xi_k, \xi_k^* \geqslant 0,\end{cases}\end{aligned}} \tag{3.11}$$

where the constant $C > 0$ determines the balance between the margin and the errors. Notice that the error term corresponds to the $\varepsilon$-insensitive loss function $|\xi|_\varepsilon$ defined as

$$|\xi|_\varepsilon = \begin{cases} 0 & \text{if } |\xi| \leqslant \varepsilon,\\ |\xi| - \varepsilon & \text{otherwise.}\end{cases} \tag{3.12}$$

As in classification, the problem can be more easily solved in the dual space, by the Lagrangian

$$\begin{aligned} L_2(\mathbf{w},b,\xi_k,\xi_k^*;\alpha_k,\alpha_k^*,\nu_k,\nu_k^*) = J_2(\mathbf{w},\xi_k) - \sum_{k=1}^N(\nu_k\xi_k + \nu_k^*\xi_k^*) - \sum_{k=1}^N \alpha_k(\varepsilon + \xi_k - y_k + \mathbf{w}^T\varphi(\mathbf{x}_k) + b)\\ - \sum_{k=1}^N \alpha_k^*(\varepsilon + \xi_k^* - y_k + \mathbf{w}^T\varphi(\mathbf{x}_k) + b),\end{aligned} \tag{3.13}$$

where $\alpha_k, \alpha_k^*, \nu_k, \nu_k^* \geqslant 0$. The solution is given by the saddle point problem

$$\max_{\alpha_k,\alpha_k^*,\nu_k,\nu_k^*} \min_{\mathbf{w},b,\xi_k,\xi_k^*} L_2(\mathbf{w},b,\xi_k,\xi_k^*;\alpha_k,\alpha_k^*,\nu_k,\nu_k^*). \tag{3.14}$$

Again, we calculate the critical points with respect to the primal variables and we define $\xi_k^{(*)}$ to refer to $\xi_k$ and $\xi_k^*$,

$$\begin{aligned} &\nabla_{\mathbf{w}} L_2 = 0 \to \partial_{\mathbf{w}} J_2(\mathbf{w},\xi_k,\xi_k^*) - \sum_{k=1}^N \alpha_k\varphi(\mathbf{x}_k) - \sum_{k=1}^N \alpha_k^*\varphi(\mathbf{x}_k) = 0 \to \mathbf{w} = \sum_{k=1}^N(\alpha_k - \alpha_k^*)\varphi(\mathbf{x}_k),\\ &\partial_b L_2 = 0 \to -\sum_{k=1}^N \alpha_k + \sum_{k=1}^N \alpha_k^* = 0 \to \sum_{k=1}^N(\alpha_k^* - \alpha_k) = 0,\\ &\partial_{\xi_k^{(*)}} L_2 = 0 \to \partial_{\xi_k^{(*)}} J_2 - \nu_k^{(*)} - \alpha_k^{(*)} = 0 \to \quad \alpha_k^{(*)} = C - \nu_k^{(*)}, \quad 0 \leqslant \alpha_k^{(*)} \leqslant C, \quad k = 1, ..., N.\end{aligned} \tag{3.15}$$

Now, we substitute in the Lagrangian:

$$
\begin{aligned}
L_2 &= \frac{1}{2}(\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)\varphi(\mathbf{x}_k))^T(\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)\varphi(\mathbf{x}_k)) + C\sum_{k=1}^{N}(\xi_k+\xi_k^*)- \\
&-\sum_{k=1}^{N}((C-\alpha_k)\xi_k+(C-\alpha_k^*)\xi_k^*)- \\
&-\sum_{k=1}^{N}\alpha_k(\varepsilon+\xi_k-y_k+(\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)\varphi(\mathbf{x}_k))^T\varphi(\mathbf{x}_k)+b)- \\
&-\sum_{k=1}^{N}\alpha_k^*(\varepsilon+\xi_k^*-y_k+(\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)\varphi(\mathbf{x}_k))^T\varphi(\mathbf{x}_k)+b) = \\
&= \frac{1}{2}\sum_{k,l=1}^{N}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)\varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l) + C\sum_{k=1}^{N}(\xi_k+\xi_k^*) - C\sum_{k=1}^{N}(\xi_k+\xi_k^*)+ \\
&+\sum_{k=1}^{N}\alpha_k\xi_k+\sum_{k=1}^{N}\alpha_k^*\xi_k^*-\sum_{k=1}^{N}\alpha_k\xi_k-\sum_{k=1}^{N}\alpha_k^*\xi_k^*- \\
&-\varepsilon\sum_{k=1}^{N}(\alpha_k+\alpha_k^*)+\sum_{k=1}^{N}y_k(\alpha_k-\alpha_k^*)-\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)b- \\
&-\sum_{k,l=1}^{N}(\alpha_k+\alpha_k^*)(\alpha_l+\alpha_l^*)\varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l) = \\
&= -\frac{1}{2}\sum_{k,l=1}^{N}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)\varphi(\mathbf{x}_k)^T\varphi(\mathbf{x}_l) - \varepsilon\sum_{k=1}^{N}(\alpha_k+\alpha_k^*)+\sum_{k=1}^{N}y_k(\alpha_k-\alpha_k^*).
\end{aligned} \tag{3.16}
$$

Finally, using the kernel trick we obtain the **regression dual problem**:

$$
\begin{aligned}
&\min_{\boldsymbol{\alpha}^{(*)}} \quad Q_2(\boldsymbol{\alpha}^{(*)};K) = \frac{1}{2}\sum_{k,l=1}^{N}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)K(\mathbf{x}_k,\mathbf{x}_l)+\varepsilon\sum_{k=1}^{N}(\alpha_k+\alpha_k^*)-\sum_{k=1}^{N}y_k(\alpha_k-\alpha_k^*), \\
&\text{subject to} \begin{cases} \sum_{k=1}^{N}(\alpha_k-\alpha_k^*)=0, & k=1,\dots,N, \\ 0\leqslant\alpha_k^{(*)}\leqslant C, & k=1,\dots,N. \end{cases}
\end{aligned} \tag{3.17}
$$

We know that the function of the regressor is $f(\mathbf{x}) = \mathbf{w}^T\varphi(\mathbf{x})+b$, so we can substitute the result obtained before, $\mathbf{w}=\sum_{k=1}^{N}(\alpha_k-\alpha_k^*)\varphi(\mathbf{x}_k)$ and use the kernel definition $K(\mathbf{x}_i,\mathbf{x}_j)=\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$. As a result, the prediction of the regressor is:

$$
y(\mathbf{x}) = \sum_{k=1}^{N}(\alpha_k-\alpha_k^*)K(\mathbf{x},\mathbf{x}_k)+b. \tag{3.18}
$$

## 3.2 Laplacian regularization

We have seen the standard SVM and now we are going to introduce a new proposal based on a penalty similar to the Fused Lasso in the dual space.

Fused Lasso uses a difference matrix of adjacent coefficients. In our case, for the Laplacian regularization in the dual space, we want to penalize the differences of all the coefficients (pairwise). We take the elements of the kernel matrix into account for the weights, because nearby samples will have a larger kernel, so the difference between them should be small, that is, more penalized. In this way, we build the difference matrix:

$$\mathbf{A} = \begin{pmatrix} \sqrt{k_{1,2}} & -\sqrt{k_{1,2}} & 0 & \cdots & 0 & 0 \\ \sqrt{k_{1,3}} & 0 & -\sqrt{k_{1,3}} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{k_{N-1,N}} & -\sqrt{k_{N-1,N}} \end{pmatrix}, \tag{3.19}$$

with size $\binom{N}{2} \times N = \frac{N(N-1)}{2} \times N$, where $N$ is the number of patterns.

Notice that, unlike Fused Lasso, the norm used in this case is the $\ell_2$ norm,

$$\|\mathbf{A}\boldsymbol{\alpha}\|_2^2 = (\mathbf{A}\boldsymbol{\alpha})^T(\mathbf{A}\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T\mathbf{A}^T\mathbf{A}\boldsymbol{\alpha}. \tag{3.20}$$

One reason to use the $\ell_2$ norm is that the problem is much simpler, because using the $\ell_1$ norm the optimization would be much more computationally complex. In addition, we use this norm because if we calculate the product $\mathbf{A}^T\mathbf{A}$ we can obtain:

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} \sqrt{k_{1,2}} & \sqrt{k_{1,3}} & \cdots & 0 \\ -\sqrt{k_{1,2}} & 0 & \cdots & 0 \\ 0 & -\sqrt{k_{1,3}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{k_{N-1,N}} \\ 0 & 0 & \cdots & -\sqrt{k_{N-1,N}} \end{pmatrix} \begin{pmatrix} \sqrt{k_{1,2}} & -\sqrt{k_{1,2}} & 0 & \cdots & 0 & 0 \\ \sqrt{k_{1,3}} & 0 & -\sqrt{k_{1,3}} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{k_{N-1,N}} & -\sqrt{k_{N-1,N}} \end{pmatrix} =$$

$$= \begin{pmatrix} \sum_{j\neq 1} k_{1,j} & -k_{1,2} & -k_{1,3} & \cdots & -k_{1,N} \\ -k_{2,1} & \sum_{j\neq 2} k_{2,j} & -k_{2,3} & \cdots & -k_{2,N} \\ -k_{3,1} & -k_{3,2} & \sum_{j\neq 3} k_{3,j} & \cdots & -k_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -k_{N,1} & -k_{N,2} & -k_{N,3} & \cdots & \sum_{j\neq N} k_{N,j} \end{pmatrix} = \mathbf{D} - \mathbf{K}, \tag{3.21}$$

where $\mathbf{K}$ is the kernel matrix and $\mathbf{D}$ the diagonal matrix such that

$$\mathbf{D} = \begin{pmatrix} \sum_j^N k_{1,j} & 0 & 0 & \cdots & 0 \\ 0 & \sum_j^N k_{2,j} & 0 & \cdots & 0 \\ 0 & 0 & \sum_j^N k_{3,j} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sum_j^N k_{N,j} \end{pmatrix}. \tag{3.22}$$

We see that expression (3.21) is exactly the **Laplacian matrix $\mathbf{L}$** [33], which is a matrix representation of a graph, in this case, the one induced by the kernel matrix computed over the problem samples. The Laplacian of a graph is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix and $\mathbf{A}$ is the adjacency matrix. Each graph vertex $e_{ij}$ has a weight $w_{ij}$, which in our case is the element $k_{ij}$ of the kernel matrix $\mathbf{K}$, which now is the adjacency matrix. If we apply this operator $\mathbf{L}$ to a vector $\boldsymbol{\alpha}$ of coefficients as a quadratic form, the operation is defined by

$$\boldsymbol{\alpha}^T \mathbf{L} \boldsymbol{\alpha} = \sum_{e_{ij}} k_{ij} (\alpha_i - \alpha_j)^2. \tag{3.23}$$

So we can see that expression (3.23) is equivalent to $\boldsymbol{\alpha}^T \mathbf{A}^T \mathbf{A} \boldsymbol{\alpha}$, the one obtained in (3.20). Therefore, the Laplacian can be expressed as $\mathbf{L} = \mathbf{A}^T \mathbf{A}$ and the dual regularization takes the form $\boldsymbol{\alpha}^T \mathbf{L} \boldsymbol{\alpha}$, where $\boldsymbol{\alpha}$ are the dual coefficients.

Now, we can build the function to be maximized in the dual space, starting from the well-known formulas for classification and regression SVM, to which we add the Laplacian term.

### 3.2.1 Fused Support Vector Classifier

To obtain the formulation of the new proposal for classification, we include the dual regularizer adapting the kernel matrix, using

$$\bar{\mathbf{K}} = \mathbf{K} + \beta \mathbf{L}, \tag{3.24}$$

where $\mathbf{L}$ is the Laplacian matrix and $\beta$ is its regularization parameter. As a result, we define the new training kernel as

$$\bar{K}(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j) + \beta L(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} k_{ij} + \beta d(\mathbf{x}_i) - \beta k_{ij} & \text{if } \mathbf{x}_i = \mathbf{x}_j, \\ k_{ij} - \beta k_{ij} & \text{if } \mathbf{x}_i \neq \mathbf{x}_j, \end{cases} \tag{3.25}$$

where $d(\mathbf{x}_i)$ is the degree of sample (node) $\mathbf{x}_i$.

We know that the dual problem for SVC is:

$$Q_1(\boldsymbol{\alpha}) = \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{y}^T \mathbf{K} \mathbf{y} \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}. \tag{3.26}$$

Therefore, we can substitute the original kernel in the formula with the new term $\bar{K}$:

$$\bar{Q}_1(\boldsymbol{\alpha}) = \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{y}^T \bar{\mathbf{K}} \mathbf{y} \boldsymbol{\alpha} - \boldsymbol{\alpha} \mathbf{1} = \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{y}^T \mathbf{K} \mathbf{y} \boldsymbol{\alpha} + \frac{\beta}{2} \boldsymbol{\alpha}^T \mathbf{y}^T \mathbf{L} \mathbf{y} \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}. \tag{3.27}$$

Finally, we have the **classification dual problem with Laplacian regularization**, from now on called **Fused Support Vector Classifier (FSVC)**:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \quad \bar{Q}_1(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{k,l=1}^{N} y_k y_l K(\mathbf{x}_k,\mathbf{x}_l)\alpha_k\alpha_l + \frac{\beta}{2}\sum_{k,l=1}^{N} y_k y_l L(\mathbf{x}_k,\mathbf{x}_l)\alpha_k\alpha_l - \sum_{k=1}^{N}\alpha_k, \\ &\text{subject to} \begin{cases} \sum_{k=1}^{N}\alpha_k y_k = 0, & k = 1,...,N, \\ 0 \leqslant \alpha_k \leqslant C, & k = 1,...,N. \end{cases} \end{aligned} \tag{3.28}$$

It is clear to see in (3.28) that we have the same original problem but with the addition of the new Laplacian regularization $\boldsymbol{\alpha}^T\mathbf{L}\boldsymbol{\alpha}$ (and its corresponding $\beta$ weight), as we mentioned before. In this way, the coefficients will change smoothly in the dual space, and there will not be very different coefficients between similar patterns.

### 3.2.2 Fused Support Vector Regressor

In the case of a regression problem, the kernel extension takes the same form as in classification:

$$\bar{\mathbf{K}} = \mathbf{K} + \beta\mathbf{L}, \tag{3.29}$$

with $\mathbf{L}$ the Laplacian matrix and $\beta$ its regularization parameter. So, the training kernel is the same as in expression (3.25).

Again, we start from the formula for SVR formulation

$$Q_2(\boldsymbol{\alpha}) = \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^T\mathbf{K}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \varepsilon\mathbf{1}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) - \mathbf{y}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*). \tag{3.30}$$

With the addition of the new term $\bar{K}$:

$$\begin{aligned} \bar{Q}_2(\boldsymbol{\alpha}) &= \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^T\bar{\mathbf{K}}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \varepsilon\mathbf{1}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) - \mathbf{y}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) = \\ &= \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^T\mathbf{K}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \frac{\beta}{2}(\boldsymbol{\alpha}-\alpha^*)^T\mathbf{L}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \varepsilon\mathbf{1}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) - \mathbf{y}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*). \end{aligned} \tag{3.31}$$

Therefore, we obtain the **regression dual problem with Laplacian regularization**, from now on called **Fused Support Vector Regressor (FSVR)**:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}^{(*)}} \quad \bar{Q}_2(\boldsymbol{\alpha}^{(*)}) = \frac{1}{2}\sum_{k,l=1}^{N}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)K(\mathbf{x}_k,\mathbf{x}_l) + \frac{\beta}{2}\sum_{k,l=1}^{N}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)L(\mathbf{x}_k,\mathbf{x}_l) \\ &\qquad\qquad + \varepsilon\sum_{k=1}^{N}(\alpha_k+\alpha_k^*) - \sum_{k=1}^{N} y_k(\alpha_k-\alpha_k^*), \\ &\text{subject to} \begin{cases} \sum_{k=1}^{N}(\alpha_k-\alpha_k^*) = 0, & k = 1,...,N, \\ 0 \leqslant \alpha_k^{(*)} \leqslant C, & k = 1,...,N. \end{cases} \end{aligned} \tag{3.32}$$

Again, we can see in (3.32) that the problem is the same as the original but including the dual regularization. As a result, the coefficients will vary smoothly in the dual space and will tend to be similar in close samples.

## 3.3 Implementation details

We use Python as a development language, because of its flexibility, and also because it has libraries adapted to ML, such as Scikit-learn [34]. Specifically, we will use the SVC [1] and SVR [2] classes to create classification and regression support vector machines, respectively.

The first idea to implement the new FSVC and FSVR proposals is to create classes that inherit from the Scikit-learn SVC and SVR classes. In this way, we can add the Laplacian regularizer to the kernel matrix used in the training, overriding the `fit` function. After training the model, we can reset the original kernel matrix to use it for prediction, since we want the Laplacian term to be taken into account only during fitting. We can see this procedure in Algorithm 3.1. The implementation of the new classes can be found in Appendix A.

```
input  : params of SVM and the Laplacian parameter beta
output: trained FSVM
compute the original kernel;
K ← computeKernel( params );
compute the degree matrix with the kernel as weights;
D ← diagonal( sumRows( K ) );
compute the Laplacian matrix;
L ← D − K;
compute the train kernel;
FSVM Kernel ← K + beta ∗ L;
call SVM fit;
super( fit( ) )
restore original kernel for predict;
FSVM Kernel ← K
```

**Algorithm 3.1:** Training Algorithm of FSVM.

The implemented method has some limitations. For example, it forces us to calculate the complete kernel matrix (LIBSVM calculates only the columns that it needs), but this way is valid for a first approximation, which can be extended in the future by modifying the C code of LIBSVM.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# 4 Experiments

In this chapter we test the performance of the new Laplacian-regularized SVMs: the FSVC and the FSVR. First, we create some synthetic datasets in which we are going to see graphically and numerically the effect of this regularizer. Finally, using real datasets, we perform comparative tests between the standard SVM model and the new Fused Support Vector Machine (FSVM) proposals, including a hard-margin FSVM with $C$ tending to infinity to take into account only the Laplacian term.

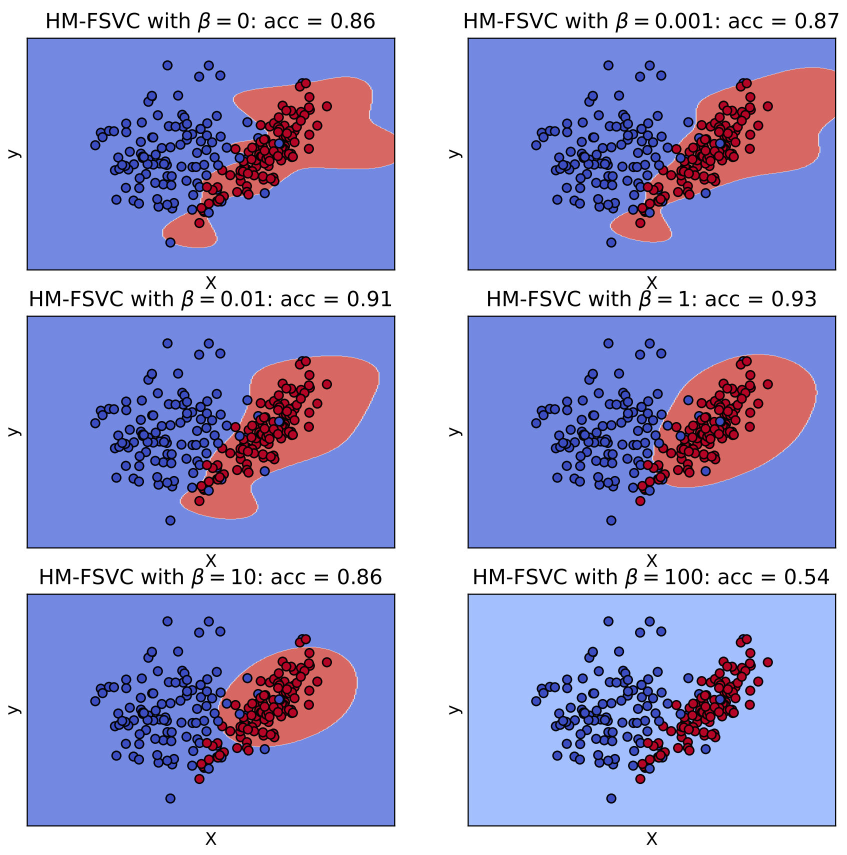
## 4.1 Synthetic datasets

To show the effect of Laplacian regularization, we create synthetic datasets and see how predictions and scores behave based on the parameter $\beta$. We focus on the FSVM model but we set the parameter $C$ large enough so that the solution depends only on the Laplacian penalty. We will call this model the Hard Margin Fused Support Vector Machine (HM-FSVM). The parameter $\gamma$ of the `rbf` kernel is fixed to $1$ in this set of experiments.

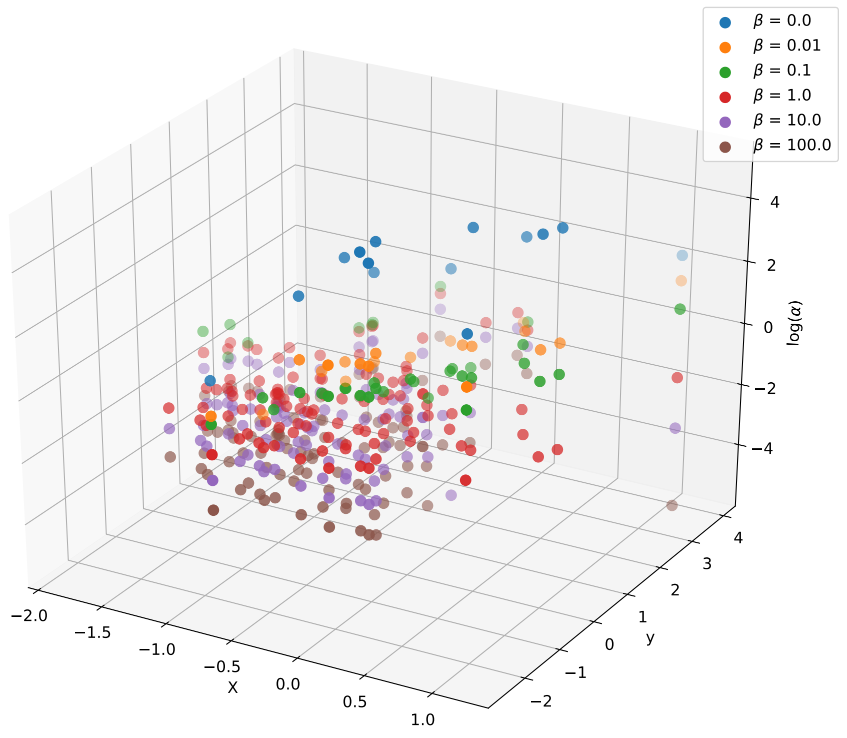
### 4.1.1 Classification

First, we build a random classification problem with different classes to test the performance of Hard Margin Fused Support Vector Classifier (HM-FSVC), with `make_classification` [1] function of Scikit-learn with $325$ samples, $2$ classes, $2$ features and $0.7$ of separation between classes, obtaining the predictions of Figure 4.1. In particular, in Figure 4.1(a) we can see how we start from $\beta = 0$ (top left) and we are increasing the regularization, in such a way that we reduce the overfitting and we manage to improve the level of prediction in the test, since the model does not learn the training noise by heart. If the parameter $\beta$ is too small we have overfitting (top row) and if $\beta$ is too large we have underfitting (bottom row). In Figure 4.1(b) we can see the corresponding dual coefficients for each value of $\beta$. Here, we can observe clearly how the Laplacian regularization reduces the dual coefficients towards a constant value while reducing the differences between those corresponding to the closest points.

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html`



(a) Decision boundary for a binary classification problem with different values of $\beta$.

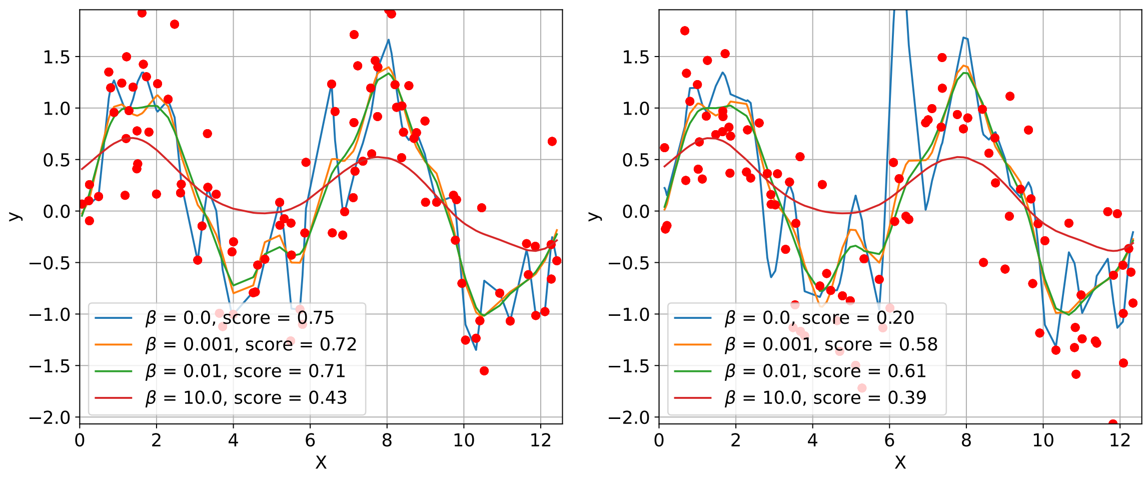


(b) Dual coefficients in classification changing the parameter $\beta$.

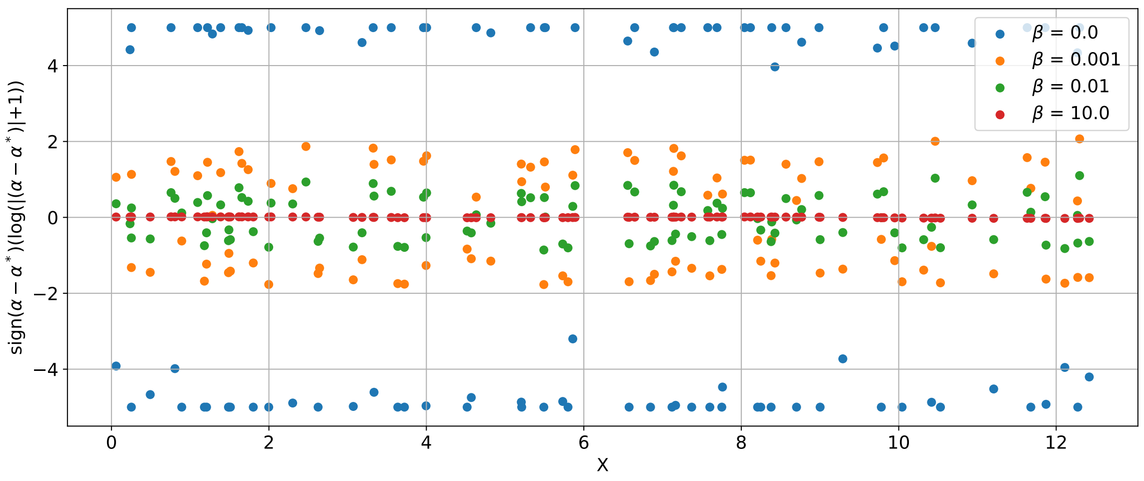
**Figure 4.1:** Synthetic classification problem for differents values of $\beta$, the Laplacian regularization parameter.

### 4.1.2 Regression

Next, we test the performance of Laplacian regularization in Hard Margin Fused Support Vector Regressor (HM-FSVR), creating a noisy sinusoidal signal, as shown in Figure 4.2. As in the classification case, in Figure 4.2(a) we see that we can sacrifice a bit of score in train (left), to obtain a better performance when we evaluate the test points (right), that is, we are avoiding overfitting by increasing the $\beta$ regularization parameter. In Figure 4.2(b) we see again how the dual coefficients (in this case $\alpha - \alpha^*$) tend to a constant value as we increase the Laplacian parameter.



(a) Train (left) and test (right) data for a sinusoidal signal fit with different $\beta$.



(b) Dual coefficients in regression changing the parameter $\beta$.

**Figure 4.2:** Synthetic problem of sinusoidal signal with noise for differents values of $\beta$, the Laplacian regularization term.

## 4.2 Real datasets

In this part, we carry out experiments in which we compare the standard SVM with the new FSVM proposal. In addition, in the comparison we also add the HM-FSVM model. It is clear to see that FSVM includes the case of SVM (with $\beta = 0$) and the case of HM-FSVM (model with sufficiently high $C$ fixed). The datasets we use are from the LIBSVM repository [30], where we obtain a total of $20$ classification and $7$ regression datasets.

For greater reliability in the comparison, we use cross-validation with Sklearn's `gridsearchCV` [2] function, which uses the hyper-parameters grid that we pass to optimize and returns the model with the best cross-validation score and the best parameters. In addition, we perform an external cross-validation with $5$ folds, using different splits of train and test, with the aim of increasing the differences in the scores and trying to make them depend as little as possible on the patterns intended for testing (since in classification very diffetent models may have the same hits and misses). Therefore, at the end of the training, for each of the three models we have $5$ cross-validation scores and $5$ test scores, which we can average to make comparisons between them.

We use the gaussian (`rbf`) kernel. We have a total of $4$ hyper-parameters: two of the SVM (although the parameter $\varepsilon$ is only used in regression problems), one of the kernel and the other one of the Laplacian. Names, range of values used in the grid search and definitons of each of them are the following:

- `C` ($C$) $\in \{10^i : i \in [-3, 5]\}$: Penalty parameter for the error term.
- `gamma` ($\gamma$) $\in \{10^i : i \in [-2, 2]\}$: Kernel coefficient.
- `epsilon` ($\varepsilon$) $\in \{10^i : i \in [-3, -2]\}$: Width of insensitivity.
- `beta` ($\beta$) $\in \{10^i : i \in [-6, -1]\} \cup \{0\}$: Laplacian regularization parameter.

The results of the (double) cross-validation for classification are found in Table 4.1. Here, we can see that the average score obtained by FSVC is, in general, better than the standard SVC, except in some ties, obtaining a better average performance of all datasets. The validation errors do not give an idea of how the model will work in general, but it allows us to check how the FSVM contains the others as a particular case. Looking at Table 4.2, we can see that the test results are similar to those of the validation, but in this case the one that obtains the best average performance among all the datasets is the HM-FSVC model, followed by FSVC. It seems that when optimizing a hyperparameter less the difference between validation and test is smaller than in the case of the complete FSVM. Again, both proposals improve the test score on average and on most of the datasets to the standard SVC.

In the case of regression, the cross-validation results are collected in Table 4.3, where we can see a similar behavior to classification: on average, FSVR again achieves the best performance (because it

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

| Dataset | CV mean FSVC (std) | CV mean SVC (std) | CV mean HM-FSVC (std) |
|---|---|---|---|
| **a1a** | **0.833116** (0.0040) | *0.828837* (0.0056) | **0.833116** (0.0040) |
| **a2a** | **0.822147** (0.0040) | 0.817933 (0.0053) | *0.820831* (0.0056) |
| **a3a** | **0.838063** (0.0068) | 0.836093 (0.0081) | *0.836655* (0.0074) |
| **australian** | **0.867078** (0.0088) | *0.864044* (0.0079) | 0.863619 (0.0090) |
| **breast-cancer** | **0.978987** (0.0047) | 0.977668 (0.0047) | *0.978113* (0.0050) |
| **diabetes** | **0.776250** (0.0082) | 0.773520 (0.0108) | *0.773547* (0.0073) |
| **dna** | **0.938591** (0.0027) | *0.937522* (0.0026) | 0.937315 (0.0045) |
| **german-numer** | **0.762388** (0.0125) | *0.752836* (0.0092) | **0.762388** (0.0125) |
| **ionosphere** | **0.947234** (0.0079) | 0.939574 (0.0083) | *0.946383* (0.0064) |
| **letter** | **0.917194** (0.0027) | *0.915045* (0.0035) | **0.917194** (0.0027) |
| **leukemia** | **0.768000** (0.0588) | **0.768000** (0.0588) | **0.768000** (0.0588) |
| **satimage** | **0.910871** (0.0028) | 0.908717 (0.0040) | *0.910669* (0.0026) |
| **segment** | **0.969879** (0.0033) | 0.968197 (0.0027) | *0.969750* (0.0033) |
| **sonar** | **0.858466** (0.0165) | **0.858466** (0.0165) | *0.854339* (0.0138) |
| **splice** | **0.846567** (0.0109) | *0.844478* (0.0108) | 0.844179 (0.0095) |
| **svmguide2** | **0.839071** (0.0146) | 0.829942 (0.0162) | *0.834470* (0.0206) |
| **svmguide3** | **0.849059** (0.0088) | *0.846668* (0.0080) | 0.844965 (0.0087) |
| **usps** | **0.931105** (0.0033) | *0.928127* (0.0024) | **0.931105** (0.0033) |
| **vehicle** | **0.839910** (0.0118) | 0.833579 (0.0134) | *0.835308* (0.0091) |
| **vowel** | **0.960370** (0.0078) | **0.960370** (0.0078) | *0.959227* (0.0063) |
| **avg datasets** | **0.872717** | 0.869481 | *0.871059* |

**Table 4.1:** Table of cross-validation scores for classification datasets. In bold best score, in italics second best.

| Dataset | Test mean FSVC (std) | Test mean SVC (std) | Test mean HM-FSVC (std) |
|---|---|---|---|
| **a1a** | **0.830943** (0.0080) | *0.828679* (0.0079) | **0.830943** (0.0080) |
| **a2a** | *0.816578* (0.0114) | 0.814171 (0.0152) | **0.818182** (0.0123) |
| **a3a** | *0.834030* (0.0172) | 0.832510 (0.0136) | **0.834221** (0.0161) |
| **australian** | *0.848246* (0.0066) | 0.845614 (0.0102) | **0.861404** (0.0116) |
| **breast-cancer** | *0.951327* (0.0074) | **0.952212** (0.0090) | 0.948673 (0.0127) |
| **diabetes** | *0.773228* (0.0144) | 0.766929 (0.0107) | **0.774016** (0.0224) |
| **dna** | *0.940693* (0.0035) | 0.933333 (0.0073) | **0.941126** (0.0037) |
| **german-numer** | **0.761212** (0.0173) | 0.758182 (0.0166) | *0.760606* (0.0174) |
| **ionosphere** | **0.936207** (0.0103) | *0.927586* (0.0229) | **0.936207** (0.0169) |
| **letter** | **0.930667** (0.0051) | *0.928606* (0.0042) | **0.930667** (0.0051) |
| **leukemia** | **0.600000** (0.1131) | **0.600000** (0.1131) | **0.600000** (0.1131) |
| **satimage** | *0.917486* (0.0066) | 0.911749 (0.0038) | **0.918169** (0.0062) |
| **segment** | **0.969069** (0.0062) | *0.968021* (0.0051) | **0.969069** (0.0062) |
| **sonar** | *0.843478* (0.0462) | *0.843478* (0.0462) | **0.866667** (0.0108) |
| **splice** | **0.870909** (0.0099) | 0.867273 (0.0120) | *0.868485* (0.0118) |
| **svmguide2** | **0.841538** (0.0210) | 0.821538 (0.0335) | *0.835385* (0.0310) |
| **svmguide3** | **0.834063** (0.0182) | **0.834063** (0.0142) | *0.831630* (0.0171) |
| **usps** | *0.941780* (0.0031) | **0.942081** (0.0034) | *0.941780* (0.0031) |
| **vehicle** | **0.830714** (0.0107) | *0.830000* (0.0173) | 0.822857 (0.0180) |
| **vowel** | **0.984000** (0.0098) | **0.984000** (0.0098) | **0.984000** (0.0067) |
| **avg datasets** | *0.862809* | 0.859501 | **0.863704** |

**Table 4.2:** Table of test scores for classification datasets. In bold best score, in italics second best.

includes other ones), followed by HM-FSVR, both methods outperforming the standard SVR. Regarding the test results in Table 4.4, we see that on average the HM-FSVR model obtain the best score, ahead of FSVR and SVR, as was already the case in classification, since this regularization does not optimize the $C$ hyper-parameter. Both methods beat SVR on most of the cases.

| Dataset | CV mean FSVR (std) | CV mean SVR (std) | CV mean HM-FSVR (std) |
|---|---|---|---|
| **abalone** | **0.574635** (0.0056) | 0.561273 (0.0057) | *0.574266* (0.0057) |
| **bodyfat** | **0.972212** (0.0124) | **0.972212** (0.0124) | *0.959162* (0.0182) |
| **eunite2001** | **0.837152** (0.0122) | *0.829336* (0.0091) | **0.837152** (0.0122) |
| **housing** | **0.846917** (0.0163) | 0.828358 (0.0178) | *0.846595* (0.0168) |
| **mg** | **0.711994** (0.0161) | *0.697568* (0.0160) | **0.711994** (0.0161) |
| **mpg** | **0.884043** (0.0078) | 0.877841 (0.0099) | *0.883855* (0.0078) |
| **pyrim** | **0.613715** (0.0986) | *0.611296* (0.0995) | 0.581545 (0.1099) |
| **avg datasets** | **0.777238** | 0.768269 | *0.770653* |

**Table 4.3:** Table of cross-validation scores for regression datasets. In bold best score, in italics second best.

| Dataset | Test mean FSVR (std) | Test mean SVR (std) | Test mean HM-FSVR (std) |
|---|---|---|---|
| **abalone** | **0.560079** (0.0099) | 0.541139 (0.0131) | *0.559599* (0.0102) |
| **bodyfat** | **0.978997** (0.0199) | **0.978997** (0.0199) | *0.969016* (0.0155) |
| **eunite2001** | **0.826597** (0.0280) | *0.817828* (0.0201) | **0.826597** (0.0280) |
| **housing** | *0.882388* (0.0225) | 0.854382 (0.0323) | **0.882406** (0.0225) |
| **mg** | **0.716632** (0.0199) | *0.694740* (0.0276) | **0.716632** (0.0199) |
| **mpg** | **0.872871** (0.0128) | 0.872423 (0.0152) | *0.872657* (0.0126) |
| **pyrim** | 0.506383 (0.1577) | *0.508999* (0.1581) | **0.540581** (0.2298) |
| **avg datasets** | *0.763421* | 0.752644 | **0.766784** |

**Table 4.4:** Table of test scores for regression datasets. In bold best score, in italics second best.

Regarding the obtained results, the datasets in which the standard SVM obtains a similar or better test performance to HM-FSVM are where the $\beta$ hyper-parameter chosen in cross-validation (see Appendix B) by FSVM is $0$ (the particular case of standard SVM) or is the lower limit set in the gridsearch, a value very close to $0$. Therefore, when the problem data does not tend to produce overfitting and the Laplacian regularization is not useful, it is the only case where the new FSVM proposal does not produce an advantage.

In summary, we can see the comparisons between the methods in Table 4.5. Here, we can see that FSVM and HM-FSVM have better test performance than SVM in $19$ of $27$ (Subtable 4.5(a)) and $20$ of $27$ (Subtable 4.5(b)) datasets, respectively. Regarding the comparison between FSVM and its particular case HM-FSVM (Subtable 4.5(c)), we see a total tie between wins, draws and losses, recalling the slightly better HM-FSVM test average mentioned above.

| Problem | Wins FSVM | Draws | Wins SVM |
|---|---|---|---|
| Classification | 14 | 4 | 2 |
| Regression | 5 | 1 | 1 |
| Total | 19 | 5 | 3 |

(a) FSVM vs SVM.

| Problem | Wins HM-FSVM | Draws | Wins SVM |
|---|---|---|---|
| Classification | 14 | 2 | 4 |
| Regression | 6 | 0 | 1 |
| Total | 20 | 2 | 5 |

(b) HM-FSVM vs SVM.

| Problem | Wins FSVM | Draws | Wins HM-FSVM |
|---|---|---|---|
| Classification | 6 | 7 | 7 |
| Regression | 3 | 2 | 2 |
| Total | 9 | 9 | 9 |

(c) FSVM vs HM-FSVM.

**Table 4.5:** Comparison of wins, draws and losses between models: FSVM, HM-FSVM and SVM.

Finally, Table 4.6 shows the ranking means of the models across the $27$ datasets. We can see that the new FSVM proposals have a similar performance, while the standard SVM clearly performs worse in ranking. Furthermore, in Figure 4.3 we can see the ranking graph of the models, where it is illustrated that the Critical Difference (CD) (using the Nemenyi test [35]) between two means is CD $= 0.6378$. So, if the difference between two means is greater than CD then it is significant. Therefore, we can conclude that the ranking difference is significant between the SVM and the FSVM models and also between the SVM and the HM-FSVM, but not between FSVM and HM-FSVM.

| | FSVM | SVM | HM-FSVM |
|---|---|---|---|
| avg ranking | 1.6851 | 2.5740 | 1.7407 |

**Table 4.6:** Mean rankings obtained by the different models.

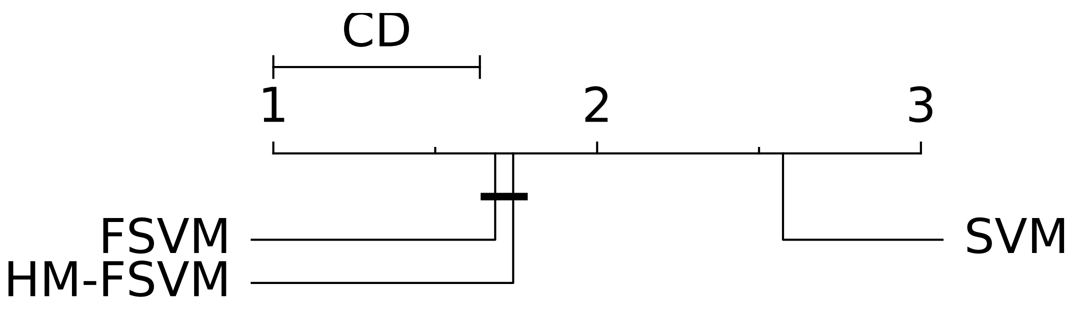


**Figure 4.3:** Critical difference graph of models ranking.

# 5 Conclusions and future work

In this chapter we explain the conclusions obtained from the comparative experiments between the standard SVM and the new proposal with Laplacian regularization. Finally, a series of tasks that could be carried out in the future are presented.

## 5.1 Conclusions

The objective of this Master Thesis was to introduce a Laplacian regularizer in the dual space for a Support Vector Machine (SVM). The idea was to penalize the differences of the dual coefficients based on their proximity, taking into account the kernel function as the weight for the Laplacian, because close patterns should also have close coefficients, as this would give the model a feeling of stability. To include this new regularizer, we have added a kernel extension in the dual formulation of the standard SVM, so that we obtain a dual expression that depends on the original kernel and the Laplacian term.

We have checked the performance of this approach in synthetic datasets. First, we have created some synthetic datasets for classification and regression. With them we could observe the effect of the new regularizer to avoid overfitting and improve test prediction, using the Hard Margin Fused Support Vector Machine (HM-FSVM) model, where $C$ is set high enough to only take into account the Laplacian term. We have also observed how the differences of the dual coefficients decrease to $0$ (that is, the coefficients tend to be constant) as the regularized parameter $\beta$ increases, as it was to be expected.

We have also worked with real datasets from the LIBSVM repository, which we have used to compare the new proposal with dual regularization, the Fused Support Vector Machine (FSVM), against the standard SVM. In this comparison we have also included the particular case HM-FSVM to see if it was enough just to use the Laplacian term. After performing a double cross-validation (internal and external) to adjust the hyperparameters, we have observed that the new FSVM and HM-FSVM approaches beat the standard SVM in test in more than $70\%$ of the cases, while SVM only beat the new proposals in less than $15\%$ of the datasets. Regarding the averages, the HM-FSVM model is the one that obtains the best average test score in both classification and regression problems, closely followed by the FSVM. This seems to be due to the fact that when optimizing one hyperparameter less, the difference between validation and test is smaller than in the case of the complete FSVM.

Closely examining the cross-validation results, we have been able to observe that in the datasets where the standard SVM equals or improves on our proposals, it is because the beta parameter chosen is $0$ or a value very close to $0$, corresponding to the lowest value set in the gridsearch. Therefore, in these cases the Laplacian regularizer was not useful, as the problem did not tend to cause overfitting, so adding the additional regularizer was not helpful. In all other cases, it can be concluded that the Laplacian regularizer in the dual space is quite useful to avoid overfitting and to improve the prediction in test. An important detail is that FSVM has one more hyperparameter than the standard SVM, so it is more computationally expensive, but HM-FSVM has the same number of parameters as SVM, therefore it gets better performance without having a larger computational cost.

## 5.2 Future work

We have used a total of $27$ classification and regression datasets and with a moderate size of both characteristics and samples. Experiments could continue to be carried out by increasing the collection of datasets, testing for example with different types of problems: with a considerably greater number of predictors than patterns, with several thousand samples to train, etc.

A more exhaustive comparison could also be made between FSVM and HM-FSVM, as they have performed very similarly in experiments. In this way, it could be analyzed if it compensates the effort of optimizing the extra parameter of FSVM, or if, on the contrary, the performance in general is usually comparable or even worse than HM-FSVM.

Another option that could be carried out in the future is the implementation directly in the LIBSVM library, to avoid having to pre-calculate the training kernel with the Laplacian term and later redefine it as the original kernel to predict.

Finally, it may be interesting to study the relationship between the Laplacian regularizer and the number of support vectors. For example, if $\beta$ is very large, the dual coefficients will tend to be constant but they will all be support vectors if $C$ is large enough or perhaps there will be no support vectors (every $\alpha$ will be $0$) if $C$ is too small. In general, with $\beta$ tending to infinity it should give a weighted average of the coefficients.

# Bibliography

[1] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[2] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[3] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[4] Ryan J Tibshirani, Jonathan Taylor, et al. The solution path of the generalized lasso. *The Annals of Statistics*, 39(3):1335–1371, 2011.

[5] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[6] J. A. K. Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[7] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

[8] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[9] Robyn Mason Dawes and Bernard Corrigan. Linear models in decision making. *Psychological Bulletin*, 81(2):95–106, 1974.

[10] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers*, (9):917–922, 1977.

[11] Shelley Derksen and Harvey J Keselman. Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables. *British Journal of Mathematical and Statistical Psychology*, 45(2):265–282, 1992.

[12] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[13] J. R. Alldredge and N. S. Gilb. Ridge regression: An annotated bibliography. *International Statistical Review / Revue Internationale de Statistique*, 44(3):355–360, 1976.

[14] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[17] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[18] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[20] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[21] Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995.

[22] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[23] Arnau Tibau Puig, Ami Wiesel, and Alfred O Hero. A multidimensional shrinkage-thresholding operator. In *2009 IEEE/SP 15th Workshop on Statistical Signal Processing*, pages 113–116. IEEE, 2009.

[24] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. Group lasso with overlap and graph lasso. In *Proceedings of the 26th annual international conference on machine learning*, pages 433–440, 2009.

[25] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[26] Stephanie Land and Jerome Friedman. Variable fusion: a new method of adaptive signal regression. *Tehnical Report*, 1996.

[27] Jose Bento, Ralph Furmaniak, and Surjyendu Ray. On the complexity of the weighted fused lasso. *IEEE Signal Processing Letters*, 25(10):1595–1599, 2018.

[28] Wesley Tansey, Alex Athey, Alex Reinhart, and James G Scott. Multiscale Spatial Density Smoothing: An Application to Large-Scale Radiological Survey and Anomaly Detection. *Journal of the American Statistical Association*, 112(519):1047–1063, 2017.

[29] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[30] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[31] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

[32] Hwanjo Yu and Sungchul Kim. Svm tutorial-classification, regression and ranking. *Handbook of Natural computing*, 1:479–506, 2012.

[33] Russell Merris. Laplacian matrices of graphs: a survey. *Linear algebra and its applications*,

197:143–176, 1994.

[34] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, M. Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[35] Thorsten Pohlert. The pairwise multiple comparison of mean ranks package (pmcmr). *R package*, 27(2020):10, 2014.

# Acronyms

| | |
|---|---|
| **CD** | Critical Difference. |
| **CNN** | Convolutional Neural Network. |
| **DNN** | Deep Neural Networks. |
| **FSVC** | Fused Support Vector Classifier. |
| **FSVM** | Fused Support Vector Machine. |
| **FSVR** | Fused Support Vector Regressor. |
| **GFL** | Graph Fused Lasso. |
| **HM-FSVC** | Hard Margin Fused Support Vector Classifier. |
| **HM-FSVM** | Hard Margin Fused Support Vector Machine. |
| **HM-FSVR** | Hard Margin Fused Support Vector Regressor. |
| **LARS** | Least Angle Regression. |
| **LS-SVM** | Least Squares Support Vector Machine. |
| **ML** | Machine Learning. |
| **MLP** | Multilayer Perceptron. |
| **MSE** | Mean Squared Error. |
| **NN** | Neural Networks. |
| **QP** | Quadratic Programming. |
| **RKHS** | reproducing kernel Hilbert space. |
| **RL** | Regularized Learning. |
| **RSS** | Residual Sum of Squares. |
| **SMO** | Sequential Minimal Optimization. |
| **SVC** | Support Vector Classifier. |
| **SVM** | Support Vector Machine. |
| **SVR** | Support Vector Regressor. |
| **VC** | Vapnik-Chervonenkis. |

# APPENDICES

# A

# Code

**Code A.1:** Code of Fused SVC class that inhterits from standard SVC.

```python
class FusedSVC(SVC):
    """Fused-Support Vector Classification.
    Similar to SVC but uses a parameter beta to control the differences of dual coefficients.
    Parameters
    ----------
    beta : float, default=0.
        Control the penalty term of differences of dual coefficients.
    """
    def __init__(self, *, beta=0., C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0,
                 shrinking=True, probability=False, tol=1e-3, cache_size=200,
                 class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',
                 break_ties=False, random_state=None):

        #add new parameter
        self.beta = beta
        #copy of original kernel
        self.svc_kernel = kernel

        super().__init__(
            C=C, kernel=kernel, degree=degree, gamma=gamma, coef0=coef0, shrinking=shrinking,
            probability=probability, tol=tol, cache_size=cache_size, class_weight=class_weight,
            verbose=verbose, max_iter=max_iter, decision_function_shape=decision_function_shape,
            break_ties=break_ties, random_state=random_state)

    def fit(self, X, y=None, sample_weight=None, **params):
        """ Override fit function with laplacian train kernel
        """

        def fused_kernel(X, Y=None):

            if(self.svc_kernel == 'linear'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svc_kernel)
            elif(self.svc_kernel == 'rbf'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svc_kernel, gamma = self.gamma)
            elif(self.svc_kernel == 'poly'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svc_kernel, gamma = self.gamma,
                                     degree = self.degree, coef0 = self.coef0)
            D = np.diag(K.sum(axis=0))
            L = D -K
            kernel_train = K + self.beta*L

            return kernel_train

        #save and calculate the original kernel
        self.kernel = fused_kernel
        #fit parent SVC
        super().fit(X=X, y=y)
        #restore the original kernel for predict
        self.kernel = self.svc_kernel
        #set support_vectors attribute
        self.support_vectors_ = np.float64(X[self.support_])
        self._gamma = self.gamma

        return self
```

**Code A.2:** Code of Fused SVR class that inhterits from standard SVR.

```
class FusedSVR(SVR):
    """Fused-Support Vector Regression.
    Similar to SVR but uses a parameter beta to control the differences of dual coefficients.
    Parameters
    ----------
    beta : float, default=0.
        Control the penalty term of differences of dual coefficients.
    """
    def __init__(self, *, beta=0., C=1.0, kernel='rbf', degree=3, gamma='scale',
                 coef0=0.0, shrinking=True, tol=1e-3, cache_size=200, epsilon=0.1,
                 verbose=False, max_iter=-1):

        #add new parameter
        self.beta = beta
        #copy of original kernel
        self.svr_kernel = kernel

        super().__init__(
            C=C, kernel=kernel, degree=degree, gamma=gamma, epsilon=epsilon,
            coef0=coef0, shrinking=shrinking, tol=tol, cache_size=cache_size,
            verbose=verbose, max_iter=max_iter)


    def fit(self, X, y=None, sample_weight=None):
        """ Override fit function with laplacian train kernel
        """

        def fused_kernel(X, Y=None):

            if(self.svr_kernel == 'linear'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svr_kernel)
            elif(self.svr_kernel == 'rbf'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svr_kernel, gamma = self.gamma)
            elif(self.svr_kernel == 'poly'):
                K = pairwise_kernels(X=X, Y=Y, metric=self.svr_kernel, gamma = self.gamma,
                                     degree = self.degree, coef0 = self.coef0)
            D = np.diag(K.sum(axis=0))
            L = D -K
            kernel_train = K + self.beta*L

            return kernel_train

        #save and calculate the original kernel
        self.kernel = fused_kernel
        #fit parent SVR
        super().fit(X=X, y=y)
        #restore the original kernel for predict
        self.kernel = self.svr_kernel
        #set support_vectors attribute
        self.support_vectors_ = np.float64(X[self.support_])
        self._gamma = self.gamma

        return self
```

B

# External cross-validation parameters

| Dataset | $\beta$ CV1 | $\beta$ CV2 | $\beta$ CV3 | $\beta$ CV4 | $\beta$ CV5 |
|---|---|---|---|---|---|
| **a1a** | -3 | -4 | -2 | -4 | -3 |
| **a2a** | -4 | -4 | -3 | -4 | -3 |
| **a3a** | -4 | -5 | -4 | -4 | -6 |
| **australian** | -6 | -6 | -5 | -6 | -2 |
| **breast-cancer** | -2 | -6 | -6 | -3 | -2 |
| **diabetes** | -6 | -3 | -6 | -6 | -3 |
| **dna** | -3 | -4 | -4 | -6 | -6 |
| **german**-numer | -3 | -3 | -3 | -4 | -6 |
| **ionosphere** | -3 | -2 | -2 | -6 | -3 |
| **letter** | -4 | -4 | -4 | <NA> | -4 |
| **leukemia** | -6 | -6 | -6 | -6 | -6 |
| **satimage** | -3 | -6 | -3 | -3 | -3 |
| **segment** | -4 | -5 | -6 | -4 | -5 |
| **sonar** | -6 | -6 | -6 | -6 | -6 |
| **splice** | -4 | -4 | -4 | -6 | -5 |
| **svmguide2** | -2 | -3 | -2 | -4 | -2 |
| **svmguide3** | -4 | -5 | <NA> | <NA> | -4 |
| **usps** | -3 | -3 | -3 | -4 | -3 |
| **vehicle** | <NA> | -5 | -5 | -4 | -5 |
| **vowel** | -6 | -6 | -6 | -6 | -6 |

**Table B.1:** Table of best Laplacian parameter $\beta$ chosen by cross-validations for Fused Support Vector Classifier (FSVC). The results are in $\log$ scale, so the cells with <NA> correspond to $\beta = 0$.

| Dataset | $\beta$ CV1 | $\beta$ CV2 | $\beta$ CV3 | $\beta$ CV4 | $\beta$ CV5 |
|---|---|---|---|---|---|
| **a1a** | -3 | -4 | -2 | -4 | -3 |
| **a2a** | -4 | -4 | -3 | -4 | -3 |
| **a3a** | -4 | -4 | -4 | -4 | -4 |
| **australian** | -4 | -3 | -4 | -4 | -4 |
| **breast**-cancer | -1 | -2 | -4 | -3 | -2 |
| **diabetes** | -5 | -3 | -4 | -1 | -5 |
| **dna** | -3 | -4 | -4 | -6 | -3 |
| **german-numer** | -3 | -3 | -3 | -4 | -3 |
| **ionosphere** | -3 | -4 | -2 | -2 | -3 |
| **letter** | -4 | -4 | -4 | <NA> | -4 |
| **leukemia** | -6 | -6 | -6 | -6 | -6 |
| **satimage** | -3 | -3 | -3 | -3 | -3 |
| **segment** | -4 | -5 | -4 | -4 | -5 |
| **sonar** | -6 | -6 | -6 | -6 | -6 |
| **splice** | -4 | -4 | -4 | -6 | -4 |
| **svmguide2** | -4 | -3 | -2 | -4 | -3 |
| **svmguide3** | -4 | -5 | -5 | -5 | -4 |
| **usps** | -3 | -3 | -3 | -4 | -3 |
| **vehicle** | -4 | -4 | -5 | -4 | -5 |
| **vowel** | -6 | -6 | -6 | -6 | -3 |

**Table B.2:** Table of best Laplacian parameter $\beta$ chosen by cross-validations for Hard Margin Fused Support Vector Classifier (HM-FSVC). The results are in $\log$ scale, so the cells with <NA> correspond to $\beta = 0$.

| Dataset | $\beta$ CV1 | $\beta$ CV2 | $\beta$ CV3 | $\beta$ CV4 | $\beta$ CV5 |
|---|---|---|---|---|---|
| **abalone** | -4 | -5 | -6 | -3 | -3 |
| **bodyfat** | <NA> | <NA> | <NA> | <NA> | <NA> |
| **eunite2001** | -5 | -5 | -5 | -5 | -5 |
| **housing** | -4 | -4 | -4 | -4 | -4 |
| **mg** | -2 | -3 | -3 | -3 | -3 |
| **mpg** | -2 | -2 | -4 | -4 | -5 |
| **pyrim** | <NA> | -3 | -5 | <NA> | <NA> |

**Table B.3:** Table of best Laplacian parameter $\beta$ chosen by cross-validations for Fused Support Vector Regressor (FSVR). The results are in $\log$ scale, so the cells with <NA> correspond to $\beta = 0$.

| **Dataset** | $\beta$ **CV1** | $\beta$ **CV2** | $\beta$ **CV3** | $\beta$ **CV4** | $\beta$ **CV5** |
|---|---|---|---|---|---|
| **abalone** | -4 | -5 | -6 | -3 | -3 |
| **bodyfat** | -4 | -5 | -4 | -5 | -4 |
| **eunite2001** | -5 | -5 | -5 | -5 | -5 |
| **housing** | -4 | -4 | -4 | -4 | -4 |
| **mg** | -2 | -3 | -3 | -3 | -3 |
| **mpg** | -2 | -2 | -4 | -4 | -5 |
| **pyrim** | -3 | -3 | -4 | -2 | -6 |

**Table B.4:** Table of best Laplacian parameter $\beta$ chosen by cross-validations for Hard Margin Fused Support Vector Regressor (HM-FSVR). The results are in $\log$ scale, so the cells with <NA> correspond to $\beta = 0$.