

**UNIVERSIDAD AUTÓNOMA DE MADRID  
ESCUELA POLITÉCNICA SUPERIOR**



**Grado en Ingeniería Informática**

**TRABAJO FIN DE GRADO**

**Many Objective Bayesian Optimization**

**Autor: Lucía Asencio Martín  
Tutor: Eduardo Garrido Merchán**

**junio 2021**

**Todos los derechos reservados.**

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

**DERECHOS RESERVADOS**

© 20 de junio de 2021 por UNIVERSIDAD AUTÓNOMA DE MADRID

Francisco Tomás y Valiente, n.º 1

Madrid, 28049

Spain

**Lucía Asencio Martín**

**Many Objective Bayesian Optimization**

**Lucía Asencio Martín**

C\ Francisco Tomás y Valiente N.º 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

# RESUMEN

---

Muchos escenarios reales requieren de la optimización de varios objetivos a la vez para la obtención de soluciones que representen el mejor equilibrio entre ellos y, según el escenario, estos objetivos pueden ser muy costosos de evaluar y el número de evaluaciones (posiblemente ruidosas) que podremos hacer será limitado. La optimización Bayesiana (BO) es una clase de métodos de optimización que, modelizando los objetivos con modelos probabilísticos subyacentes como los procesos Gaussianos (GPs), habilita la optimización de funciones cuya expresión analítica es desconocida. La BO multi-objetivo se encarga generalmente de la optimización de hasta tres objetivos pero, hasta donde alcanza nuestro conocimiento, no se ha desarrollado un algoritmo para la BO de muchos (más de tres) objetivos. Por ello, proponemos un algoritmo de BO de muchos objetivos basado en la detección y supresión de GPs que son redundantes por estar correlacionados. Mediante esta eliminación, evitamos la evaluación de objetivos de los que no se adquiere apenas información sobre la localización de los puntos óptimos. Al evitar esta evaluación, se puede ahorrar coste computacional o de recursos de diversa índole.

Para la detección de objetivos redundantes proponemos una medida de similitud entre las distribuciones predictivas de los GPs. La medida de similitud es un modelo de suma ponderada donde se tienen en cuenta la distancia entre las medias predictivas, su correlación y la incertidumbre de la predicción. Usando esta medida proponemos un algoritmo de reducción de objetivos que suprime aquéllos que considera redundantes según el criterio de la medida de similitud. Integramos el algoritmo y la medida en Spearmint, una herramienta de software libre de optimización Bayesiana.

Los experimentos de juguete, sintéticos y simulando un escenario real realizados sobre la medida de similitud y el algoritmo de optimización Bayesiana de muchos objetivos aportan evidencia empírica para apoyar la hipótesis de su utilidad en escenarios reales. En particular, probamos el algoritmo en el proceso de ajuste de hiperparámetros de una red neuronal en el que nos proponemos minimizar el error, el tiempo de ejecución y el tamaño de la misma. En este escenario, el tiempo y el tamaño son detectados como objetivos redundantes y uno de ellos puede ser reducido. En un escenario real, la no evaluación del tiempo o tamaño de una red neuronal profunda de gran tamaño puede implicar un ahorro de coste computacional, económico y medioambiental.

# PALABRAS CLAVE

---

Optimización Bayesiana, optimización Bayesiana multiobjetivo, optimización de muchos objetivos, medida de similitud, proceso Gaussiano.



# ABSTRACT

---

Many real life scenarios require the optimization of several objectives to obtain solutions representing a compromise between them and, depending on the scenario, these objectives can be expensive to evaluate and the number of (possibly noisy) evaluations will be limited. Bayesian optimization (BO) is a class of optimization methods which, modelling the objectives with surrogate probabilistic models such as Gaussian processes (GPs) allows the optimization of functions when their analytical expression is unknown. Multi-objective BO generally takes care of the optimization of at most three objectives but, as far as we are concerned, no algorithm exists for many (more than three) objective BO. Therefore we propose a many objective BO algorithm based on the detection and suppression of correlated GPs. Through this elimination, we avoid the evaluation of objectives from which we barely acquire information about the optimal point location. Computational or other resources cost is saved when avoiding these evaluations.

In order to detect redundant objectives we propose a similarity measure between GPs predictive distributions. The similarity measure consists of a weighted sum model where we take in account the distance between the predictive means, their correlation and uncertainty about the prediction. Using this measure, we propose an objective reduction algorithm that removes those objectives which are considered redundant according to the measure. We integrate the similarity measure and this algorithm into Spearmint, a free software tool for Bayesian optimization.

The toy, synthetic and real life simulating experiments performed on the similarity measure and the many objective Bayesian optimization algorithm provide empirical evidence to support the hypothesis of their usefulness in real scenarios. In particular, we use the algorithm for hyperparameter tuning in a neural network where we aim to minimize the error, running time and size of the network. In this scenario, running time and size are identified as redundant objectives and one of them can be removed. In a real life scenario, not evaluating the running time or size of a deep neural network can result in a computational, economic and environmental cost savings.

# KEYWORDS

---

Bayesian optimization, multiobjective Bayesian optimization, many objective optimization, similarity measure, Gaussian process.



# ÍNDICE

---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introducción</b>  | <b>1</b>  |
| <b>2</b> | <b>Estado del arte</b>                                       | <b>3</b>  |
| 2.1      | Optimización Bayesiana                                       | 3         |
| 2.2      | Optimización de más de un objetivo                           | 4         |
| 2.3      | Medidas de similitud entre distribuciones de probabilidad    | 5         |
| <b>3</b> | <b>Definición del proyecto</b>                               | <b>7</b>  |
| 3.1      | Objetivos  | 7         |
| 3.2      | Hipótesis  | 7         |
| 3.3      | Supuestos  | 8         |
| 3.4      | Restricciones  | 8         |
| <b>4</b> | <b>Diseño del proyecto</b>                                   | <b>9</b>  |
| 4.1      | Diseño funcional   | 9         |
| 4.1.1    | Procesos Gaussianos  | 9         |
| 4.1.2    | Medida de similitud entre distribuciones predictivas de GPs  | 10        |
| 4.1.3    | Optimización Bayesiana                                       | 13        |
| 4.1.4    | Optimización Bayesiana multiobjetivo                         | 14        |
| 4.1.5    | Optimización Bayesiana de muchos objetivos                   | 16        |
| 4.2      | Diseño técnico   | 17        |
| 4.3      | Planificación del proyecto                                   | 19        |
| <b>5</b> | <b>Implementación</b>  | <b>23</b> |
| <b>6</b> | <b>Experimentación</b>                                       | <b>25</b> |
| 6.1      | Experimentos de la medida de similitud                       | 25        |
| 6.1.1    | Primer experimento: GPs de una dimensión.                    | 26        |
| 6.1.2    | Segundo experimento: GPs bidimensionales con forma de cuenco | 26        |
| 6.1.3    | Tercer experimento: GPs bidimensionales muy distintos        | 27        |
| 6.1.4    | Cuarto experimento: GPs ajustando funciones <i>Ackley</i>    | 28        |
| 6.2      | Experimentos del algoritmo de reducción de objetivos         | 29        |
| 6.2.1    | Primer experimento   | 29        |
| 6.2.2    | Segundo experimento: funciones cuenco                        | 30        |
| 6.2.3    | Tercer experimento: funciones <i>Michalewicz</i>             | 32        |
| 6.2.4    | Cuarto experimento: ajuste de hiperparámetros red neuronal   | 33        |

|  |           |
|--|-----------|
| <b>7 Conclusiones y trabajo futuro</b>       | <b>35</b> |
| <b>Bibliografía</b>                          | <b>40</b> |
| <b>Apéndices</b>                             | <b>41</b> |
| <b>A Paquetes instalados</b>                 | <b>43</b> |
| A.1 Paquetes instalados con Python 2.7 ..... | 43        |
| A.2 Paquetes instalados con Python 3.8 ..... | 43        |



# LISTAS

---

## Lista de algoritmos

|     |                              |    |
|-----|------------------------------|----|
| 4.1 | Optimización Bayesiana ..... | 14 |
| 4.2 | Reducción objetivos .....    | 17 |

## Lista de ecuaciones

## Lista de figuras

|     |  |    |
|-----|--|----|
| 4.1 | Visualización algoritmo de BO .....              | 15 |
| 4.2 | Diagrama de flujo de datos de nivel 0 .....      | 17 |
| 4.3 | Diagrama de flujo de datos de nivel 1 .....      | 18 |
| 4.4 | Diagrama de Gantt 1/2 .....                      | 20 |
| 4.5 | Diagrama de Gantt 2/2 .....                      | 21 |
| 6.1 | GPs ajustando funciones unidimensionales .....   | 26 |
| 6.2 | GPs ajustando funciones en forma de cuenco ..... | 27 |
| 6.3 | GPs ajustando funciones muy diferentes .....     | 28 |
| 6.4 | GPs ajustando funciones <i>Ackley</i> .....      | 28 |
| 6.5 | Funciones <i>Branin</i> .....                    | 29 |
| 6.6 | Funciones paraboloideas .....                    | 31 |
| 6.7 | Funciones <i>Michalewicz</i> .....               | 32 |
| A.1 | Paquetes instalados en Python 2.7 .....          | 44 |
| A.2 | Paquetes instalados en Python 3.8 1/2 .....      | 45 |
| A.3 | Paquetes instalados en Python 3.8 2/2 .....      | 46 |

## Lista de tablas

|     |   |    |
|-----|---|----|
| 6.1 | Tabla experimento 1 reducción objetivos ..... | 30 |
|-----|---|----|



# INTRODUCCIÓN

---

Cuando tomamos decisiones importantes , intentamos hacer la mejor elección evaluando muchos factores a la vez. Si nos mudamos de casa intentamos que sea a un lugar grande, bien localizado, accesible, luminoso, económico, etc., pero encontrar un hogar que cumpla todo esto es mucho más difícil que si únicamente pudiéramos mudarnos a una casa luminosa.

Lo mismo ocurre en los procesos industriales: supongamos que queremos construir un coche que sea lo más seguro, cómodo, espacioso y barato posible. Para conseguir estos objetivos hay que tomar decisiones acerca de variables como los materiales y la geometría de la carrocería, la disposición de alerones, los neumáticos, el sistema de frenos, etc.

Como a priori no sabemos con qué decisiones de diseño se obtiene el mejor vehículo, podríamos diseñar coches con todas las combinaciones posibles de estas variables, medir lo seguro, cómodo, espacioso y costoso que es cada diseño y sacar al mercado aquél que diera mejores resultados. Pero no podemos medir la seguridad de un coche sin fabricarlo, así que necesitaríamos fabricar un coche por cada diseño que quisiéramos probar, y como el coste y el tiempo de fabricación de estos vehículos es muy elevado, esta solución es inviable. Necesitamos encontrar, fabricando el menor número de coches posibles, la combinación de variables que produce el mejor coche.

Si sólo buscáramos fabricar un coche espacioso sin importarnos su coste o comodidad nos encontraríamos ante un problema relativamente sencillo, pero intentar satisfacer a la vez tantos objetivos (comodidad, seguridad, bajo coste y espacio) es muy complejo si no disponemos de los recursos necesarios para fabricar y probar todos los coches que queramos. Ahora bien: puede que los coches más seguros y los más cómodos sean a su vez los más espaciosos. Si esto fuera así podríamos centrarnos en satisfacer únicamente dos objetivos, espacio y bajo coste, transformando nuestro problema en otro mucho más sencillo que sí podríamos solucionar con los recursos disponibles, ya que podemos saber lo espacioso y caro que es un coche sin fabricarlo sólo conociendo las medidas y los materiales del diseño.

En este trabajo de fin de grado se plantea una metodología para encontrar objetivos que estén relacionados entre sí (como la seguridad y la comodidad en el ejemplo anterior) en procesos de optimización Bayesiana, con el fin de reducir el coste de la optimización.

## Estructura del documento

Después de la introducción, el segundo capítulo expone el estado del arte de la optimización bayesiana multiobjetivo (*multi-objective bayesian optimization*), así como de la optimización de muchos objetivos (*many-objective optimization*) y de la comparación de distribuciones de probabilidad.

En el tercer capítulo de esta memoria definimos el proyecto en base a los objetivos que buscamos satisfacer, las hipótesis y supuestos que hemos tomado, así como las limitaciones que hemos encontrado durante el desarrollo del mismo.

El capítulo cuarto está dedicado al diseño del proyecto desde el punto de vista funcional, técnico y de planificación temporal, y el quinto capítulo a la implementación.

El sexto capítulo recopila los experimentos realizados, tanto su definición como los resultados obtenidos, y el último presenta las conclusiones del mismo y plantea el trabajo futuro a realizar sobre el proyecto.

## Notación

A lo largo de este trabajo, emplearemos la siguiente notación:

- Las letras  $x, y, z$  denotan escalares.
- Las letra minúsculas en negrita  $\mathbf{x}, \mathbf{y}$ , y otras denotan vectores.
- Las letras mayúsculas en negrita  $\mathbf{X}, \mathbf{Y}$ , etc. denotan matrices.
- Dado un vector o una matriz,  $\mathbf{x}^T, \mathbf{X}^T$  denotan vector o matriz traspuesta.
- Las letras mayúsculas caligráficas denotan conjuntos, por ejemplo  $\mathcal{D}$  es el conjunto de datos observados,  $\mathcal{X}$  el conjunto de datos de entrada de una función, etc.
- Todas las gráficas bidimensionales de este proyecto tienen como eje horizontal el eje  $X$  y al eje  $Y$  como eje vertical.
- Todas las gráficas tridimensionales de este proyecto tienen como eje vertical el eje  $Z$ , el eje  $Y$  es el eje horizontal izquierdo y el eje  $X$  el horizontal derecho.

## ESTADO DEL ARTE

---

A continuación introduciremos el concepto de optimización Bayesiana y expondremos su estado del arte. Nos centraremos en la optimización Bayesiana multiobjetivo (hasta tres objetivos) y también en la optimización de muchos objetivos (más de tres objetivos). Con el fin de establecer una noción de distancia entre los objetivos de la optimización, exploraremos el estado del arte en medidas de similitud de distribuciones de probabilidad.

### 2.1. Optimización Bayesiana

La optimización Bayesiana (BO) es una estrategia utilizada para la optimización de funciones cuya expresión analítica es desconocida y donde, por tanto, no podemos acceder a los gradientes de las mismas [10]. Las funciones a optimizar se denominan funciones objetivo y la BO es adecuada para funciones objetivo que tengan soporte continuo de menos de 20 dimensiones [19], posiblemente ruidosas y muy costosas de evaluar [59], ya que la BO es eficiente en cuanto a número de evaluaciones de la función objetivo que requiere [33, 48, 58, 63].

Usamos el transformador [67] BERT [15] como ejemplo para ilustrar el coste que puede tener la evaluación de una función objetivo en un proceso de optimización. Para minimizar el error de generalización (función objetivo) de BERT es necesario entrenarlo, y este entrenamiento incurre en un gasto económico, temporal y medioambiental estudiado en [64]. Un sólo entrenamiento tiene la misma huella de carbono que un vuelo de Nueva York a San Francisco, supone un coste de hasta 12000 dólares estadounidenses y toma aproximadamente 79 horas.

La optimización Bayesiana ha sido utilizada para el ajuste de hiperparámetros en algoritmos de aprendizaje automático [5, 59] así como para la elección de experimentos en el diseño de materiales [50], la mejora de sistemas de energía renovable [13] e incluso para la creación de una receta de galletas perfecta [61]. Encontramos otras aplicaciones en la aceleración de máquinas de vectores de soporte [21], la optimización de una red de sensores [22] o la observación y monitorización medioambiental [44]. La BO también ha sido utilizada en robótica para optimizar la manera de andar de un robot [41], y en la optimización de interfaces interactivas para facilitar el diseño de animaciones

gráficas [9].

La optimización Bayesiana es un proceso iterativo con dos componentes esenciales: un modelo probabilístico de la función objetivo a optimizar y una función de adquisición [23]. La función de adquisición indica a cada iteración en qué punto conviene evaluar las funciones objetivo en base a un criterio de explotación (evaluar puntos cercanos a los que sospechamos pueden ser óptimos) y exploración (evaluar en entornos no explorados).

Un ejemplo de función de adquisición es PI (probabilidad de mejora) que devuelve el punto con más probabilidad de superar al óptimo actual [65]. Otro es EI (esperanza de mejora), que escoge el punto que más mejoraría en media al óptimo actual [34].

Como modelos de los objetivos se han usado procesos de Wiener [38], redes neuronales [60] profundas o bosques aleatorios [31]. El modelo probabilístico más utilizado es el proceso Gaussiano (GP) [53], que es un proceso estocástico [57] donde cada subconjunto finito de variables aleatorias sigue una distribución normal. Los GPs equivalen a redes neuronales profundas completamente conexas con infinitas unidades ocultas [40]. De hecho, la BO con GPs supone una ventaja sobre las redes neuronales profundas [39] al proporcionar información acerca de la incertidumbre de sus predicciones [42, 53]. Como los procesos Gaussianos son el modelo estándar utilizado en BO [59] hemos decidido centrarnos en ellos para el desarrollo de este trabajo; no obstante, nuestro enfoque podría ser aplicado fuera del ámbito de los GPs.

## 2.2. Optimización de más de un objetivo

Cuando dentro de la BO intentamos optimizar hasta tres cajas negras (funciones objetivo) nos encontramos con la optimización Bayesiana multiobjetivo (MOBO) [35]. La optimización de varios objetivos a la vez supone un desafío ya que, en general, los puntos óptimos de una caja negra no coincidirán con los de otra. Una posible solución es centrar la optimización multiobjetivo en buscar el llamado conjunto de Pareto [75], formado por aquellas soluciones que no están dominadas por otras, es decir, aquellas soluciones tales que no hay ninguna otra solución que sea mejor en todos los objetivos [7]. Este criterio de dominancia entre soluciones se llama eficiencia de Pareto [69].

El propósito de la función de adquisición en MOBO es la identificación de puntos potenciales del conjunto de Pareto. Un ejemplo de ello es EHVI, una función de adquisición que utiliza el hipervolumen como indicador para escoger el próximo punto a evaluar. En los últimos años se han estudiado funciones de adquisición como GP-UCB, basado en un algoritmo que soluciona el problema del bandido multibrazo [62]. También se han diseñado funciones basadas en teoría de información como ES [28] o su reciente mejora PESMO [29, 30]. Se han desarrollado funciones de adquisición que permiten dar restricciones sobre el espacio de búsqueda (PESMOC, [24]) o incluso establecer preferencias entre las funciones objetivo [1].

La optimización de muchos objetivos (many objective optimization) trata el problema de la optimización de más de tres funciones objetivo [18]. Al aumentar el número de objetivos el tamaño del conjunto de Pareto crece de manera exponencial y con él el número de evaluaciones necesarias para explorarlo [36]. Ante tantas soluciones es el usuario quien escoge manualmente entre las propuestas por el algoritmo de optimización, pero el aumento de la dimensión imposibilita la visualización de estas soluciones y dificulta la elección [32].

Para hacer frente a la optimización de muchos objetivos se han propuesto distintos enfoques. Por ejemplo [73] y [4] transforman el problema en la optimización de un sólo objetivo, y en [25] se ofrecen varias alternativas que no se centran en la eficiencia de Pareto para comparar dos soluciones potenciales. Recientemente el comportamiento de palomas mensajeras ha servido de inspiración para el diseño de otro algoritmo de optimización de muchos objetivos [14]. También se ha propuesto PCSEA, un algoritmo que reduce la dimensión del problema detectando objetivos irrelevantes y eliminándolos.

La optimización de muchos objetivos se ha aplicado para optimizar el controlador de un coche híbrido [49]. También se ha utilizado en la refactorización en software [47], en la optimización del diseño de naves industriales prefabricadas [3] y en la planificación de horarios de enfermería [51]. En aeronáutica, se ha empleado para optimizar el diseño de un perfil alar [70] y de aeronaves [2].

A pesar de la variedad de soluciones existentes para la optimización de muchos objetivos, hasta donde sabemos la optimización Bayesiana de muchos objetivos no ha sido abordada todavía. En 2020 se propuso un enfoque que, en vez de centrarse en todas las soluciones del conjunto de Pareto, escogía una usando teoría de juegos [8]. En este trabajo sugerimos una solución basada en la reducción de la dimensión del espacio de objetivos, mediante la detección de funciones objetivo (procesos Gaussianos) similares.

## 2.3. Medidas de similitud entre distribuciones de probabilidad

Para poder comparar objetivos de un proceso de optimización Bayesiana sería interesante tener una noción de distancia en el espacio de procesos Gaussianos. Por ello estudiamos el estado del arte en la comparación de distribuciones de probabilidad.

En teoría de probabilidad, las distancias estadísticas son funciones que nos permiten comparar dos objetos estadísticos como variables aleatorias, distribuciones de probabilidad o muestras aleatorias entre sí [68]. En este contexto, las distancias no son estrictamente métricas ya que no se les exige simetría [52].

Un ejemplo de distancia estadística es la distancia de Mahalanobis, que permite cuantificar la distancia entre dos muestras de una misma variable aleatoria o de distintas variables con una misma

covarianza [43]. Recientemente, esta distancia ha sido extendida para abarcar el caso en el que las muestras pertenecen una distribución de funciones en  $L^2[0, 1]$  [6, 17].

Otras distancias estadísticas son las divergencias [55], que comparan funciones de distribución de variables aleatorias. Un ejemplo de la divergencia de Kullback-Leibler (KL) [37], que se calcula como una integral dependiente de las densidades de las distribuciones. La divergencia KL ha sido aplicada para predecir hipertensión en pacientes [12], para estimar la duración de llamadas telefónicas [27] y para la detección de las condiciones de iluminación de una imagen [56]. Divergencias conocidas son también la de Hellinger [45] o la de Jensen-Shannon [46]. Otra herramienta utilizada para la medir la similitud de dos distribuciones de probabilidad es la distancia Bhattacharyya [20]. Esta última ha sido utilizada en investigación para la extracción de características [11], el procesamiento de imágenes [26] y reconocimiento de voz [74].



## DEFINICIÓN DEL PROYECTO

---

En este capítulo definimos los objetivos del trabajo y las hipótesis a contrastar. También exponemos las suposiciones que se han hecho al diseñar el modelo, así como las restricciones que hemos encontrado desde el punto de vista técnico.

### 3.1. Objetivos

Estos son los objetivos a alcanzar en el proyecto:

- O<sub>1</sub>.** Implementación en Python de una distancia estadística para comparar procesos gaussianos que comparten el mismo espacio de entrada.
- O<sub>2</sub>.** Demostrar efectividad de la distancia desarrollada con funciones de referencia.
- O<sub>3</sub>.** Integrar la distancia en la herramienta de optimización bayesiana Spearmint.
- O<sub>4</sub>.** Implementar un algoritmo de reducción de objetivos en Spearmint.
- O<sub>5</sub>.** Reducir el tiempo de ejecución de Spearmint cuando dos objetivos sean considerados similares por la herramienta
- O<sub>6</sub>.** No reducir drásticamente el rendimiento de la optimización al reducir el número de objetivos.

### 3.2. Hipótesis

A continuación, las hipótesis a contrastar:

- H<sub>1</sub>.** Podemos desarrollar una herramienta capaz de medir la similitud entre dos procesos gaussianos.
- H<sub>2</sub>.** Podemos reducir el coste de la BO de muchos objetivos en Spearmint.
- H<sub>3</sub>.** El rendimiento de Spearmint no bajará significativamente al reducir el coste de la BO de muchos objetivos.

### 3.3. Supuestos

Esto es lo que hemos asumido para alcanzar los objetivos anteriores:

- S<sub>1</sub>.** Las funciones objetivo de la BO tienen igual dominio.
- S<sub>2</sub>.** Las funciones objetivo de la BO van de  $\mathbb{R}^n$  en  $\mathbb{R}^m$ .
- S<sub>3</sub>.** Los procesos gaussianos son capaces de ajustar las funciones objetivo de la optimización.
- S<sub>4</sub>.** En los casos de uso de BO, el coste de la BO es despreciable frente al de las funciones objetivo.
- S<sub>5</sub>.** El número de muestras Monte Carlo para los hiperparámetros de los GP en Spearmint es 1.
- S<sub>6</sub>.** Spearmint es una de la de las mejores herramientas de código abierto para MOBO, como demuestra su presencia en artículos de estado del arte como [16], [54] y [60].
- S<sub>7</sub>.** El hipervolumen es un indicador fiable de la calidad de la solución de la optimización en un problema multiobjetivo.

### 3.4. Restricciones

A nivel técnico, nos hemos encontrado con las siguientes restricciones:

- R<sub>1</sub>.** Capacidad computacional limitada por un ordenador portátil Toshiba Satellite P50-B-11L de 2015.
- R<sub>2</sub>.** Limite temporal para el proyecto de 266 días.
- R<sub>3</sub>.** Documentación casi inexistente para la herramienta Spearmint.
- R<sub>4</sub>.** Falta de mantenimiento de la herramienta Spearmint.
- R<sub>5</sub>.** Dificultad en la comunicación con el tutor por la imposibilidad de reuniones presenciales debido a la crisis COVID-19.

# DISEÑO DEL PROYECTO

---

Este capítulo está dedicado al diseño del trabajo desde tres enfoques distintos. En primer lugar mostramos el diseño funcional, en segundo el diseño técnico y por último veremos la planificación temporal del proyecto.

## 4.1. Diseño funcional

Vamos a dividir esta sección en cuatro partes. Primero introduciremos algunas nociones básicas acerca de procesos Gaussianos que motivarán el diseño de la medida de similitud para GPs en la segunda subsección. Dedicaremos la tercera subsección a entender la optimización Bayesiana de hasta tres objetivos, para finalmente en la cuarta subsección explicar nuestra propuesta de optimización Bayesiana para más de tres objetivos utilizando la medida de similitud desarrollada.

### 4.1.1. Procesos Gaussianos

Ya hemos mencionado que un proceso Gaussiano es un proceso estocástico (una colección posiblemente infinita de variables aleatorias indexadas) donde cada subconjunto finito de la colección sigue una distribución normal. En la optimización Bayesiana, los GPs se emplean para obtener una distribución predictiva para generar una función de adquisición que nos recomiende qué punto es considerado el mejor para ser evaluado en las siguientes iteraciones.

Sea  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)^T$  una matriz con los datos de entrenamiento y  $\mathbf{y} = (y_1, \dots, y_N)^T$  el vector de etiquetas a predecir. Definimos  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \dots, N\}$  como el conjunto de datos que ya han sido etiquetados junto a sus etiquetas. Un GP queda unívocamente determinado por un vector de medias (que suele inicializarse al vector nulo) y por una función de covarianza  $k(\mathbf{x}, \mathbf{x}')$ . Esta información se recoge bajo la notación  $f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$ .

Dado  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \dots, N\}$  el conjunto de datos ya observados donde  $y_i = f(\mathbf{x}_i) + \epsilon_i$  para cierto ruido Gaussiano  $\epsilon_i$ , un proceso Gaussiano nos permite conocer la distribución predictiva del valor  $f(\mathbf{x}^*)$  dado un punto  $\mathbf{x}^*$ . La distribución predictiva sigue una distribución normal, es decir

$p(f(\mathbf{x}^*)|\mathbf{y}) = \mathcal{N}(f(\mathbf{x}^*)|\mu(\mathbf{x}^*), v(\mathbf{x}^*))$ , donde la media  $\mu(\mathbf{x}^*)$  y la varianza  $v(\mathbf{x}^*)$  pueden escribirse como:

$$\mu(\mathbf{x}^*) = \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}, \quad (4.1)$$

$$v(\mathbf{x}^*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_*. \quad (4.2)$$

El vector  $\mathbf{y} = (y_1, \dots, y_N)^T$  contiene todas las observaciones hechas hasta el momento, el valor  $\sigma^2$  es la varianza del ruido Gaussiano  $\epsilon_i$ , el vector  $N$ -dimensional  $\mathbf{k}_* = \mathbf{k}(\mathbf{x}_*)$  contiene la covarianza de la distribución a priori entre el nuevo punto  $f(\mathbf{x}^*)$  y cada uno de los puntos del conjunto de entrenamiento  $f(\mathbf{x}_i)$ , y la matriz  $N \times N$  llamada  $\mathbf{K}$  contiene las covarianzas de la distribución a priori entre cada par de  $f(\mathbf{x}_i)$  para  $i = 1, \dots, N$  del conjunto de entrenamiento. Los elementos de la matriz  $\mathbf{K}$  vienen dados por el valor  $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$  que describe la covarianza entre dos puntos  $\mathbf{x}_i, \mathbf{x}_j$  para  $i, j = 1, \dots, N$  del conjunto de entrenamiento. Todo aquello que sabemos acerca de la forma y otras características de  $f(\mathbf{x})$  como por ejemplo su suavidad, su ruido, etc. se codifican a través de la elección de la función de covarianzas  $k(\mathbf{x}, \mathbf{x}')$  del proceso Gaussiano. Algunos ejemplos de funciones de covarianza son funciones constantes, lineales, la llamada función Matérn u ótras, una función de covarianza conocida y utilizada es la exponencial cuadrática, dada por la expresión:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp \left( - \left( \frac{r^2}{2\ell^2} \right) \right) \left( + \sigma_n^2 \delta_{pq} \right), \quad (4.3)$$

donde el valor  $r$  simboliza la distancia euclídea entre  $\mathbf{x}$  y  $\mathbf{x}'$ ,  $\ell$  es el hiperparámetro llamado escala de longitud que define la suavidad de las funciones generadas por el GP, el hiperparámetro de amplitud  $\sigma_f^2$  mide la variabilidad entre las muestras del GP y el valor  $\sigma_n^2 \delta_{pq}$  es la varianza del ruido Gaussiano que se aplica cuando la función de covarianza es calculada sobre un par de la forma  $k(\mathbf{x}, \mathbf{x})$ . Estos son los hiperparámetros del GP, que se recogen en un conjunto que llamaremos  $\theta$ . Podemos encontrar valores óptimos para cada hiperparámetro en  $\theta$  aplicando el método de máxima verosimilitud [53].

#### 4.1.2. Medida de similitud entre distribuciones predictivas de GPs

En este trabajo hemos desarrollado la medida de similitud entre las distribuciones predictivas de procesos Gaussianos que explicamos a continuación. Sean  $f(\mathbf{x})$ ,  $g(\mathbf{x})$  dos procesos Gaussianos. Trabajamos asumiendo que las funciones de covarianza de ambos procesos tienen expresiones analíticas parecidas, por ejemplo, ambas son del tipo exponencial cuadrática. Dado un conjunto de datos de entrada  $\mathbf{X}_*$ , podemos calcular la predicción de los vectores de media de cada proceso  $\mu_f(\mathbf{X}_*)$ ,  $\mu_g(\mathbf{X}_*)$  así como las matrices de covarianza  $v_f(\mathbf{X}_*)$  y  $v_g(\mathbf{X}_*)$ .

El propósito de esta subsección es desarrollar una noción de distancia entre estos dos procesos, que definimos con esta expresión matemática:

$$\begin{aligned}
d(f(\mathbf{x}), g(\mathbf{x})) = & \varepsilon_1 d_\mu (T(\mu_f(\mathbf{X}_\star)), \mu_g(\mathbf{X}_\star), \delta) + \\
& \varepsilon_2 (1 - \max(0, \rho(\mu_f(\mathbf{X}_\star), \mu_g(\mathbf{X}_\star))) + \\
& (1 - \varepsilon_1 - \varepsilon_2) d_v (v_f(\mathbf{X}_\star), v_g(\mathbf{X}_\star)) .
\end{aligned} \tag{4.4}$$

Observamos que la medida de similitud sigue un modelo de suma ponderada (WSM, “Weighted sum model” [66]), donde cada sumando está ponderado por un peso  $\varepsilon_1, \varepsilon_2, 1 - \varepsilon_1 - \varepsilon_2$  que oscila entre 0 y 1 y cuya suma debe ser exactamente 1. La WSM consta de tres sumandos a los cuales nos referiremos como

$$\begin{aligned}
s_1 &= \varepsilon_1 d_\mu (T(\mu_f(\mathbf{X}_\star)), \mu_g(\mathbf{X}_\star), \delta) , \\
s_2 &= \varepsilon_2 (1 - \rho(\mu_f(\mathbf{X}_\star), \mu_g(\mathbf{X}_\star))) , \\
s_3 &= (1 - \varepsilon_1 - \varepsilon_2) d_v (v_f(\mathbf{X}_\star), v_g(\mathbf{X}_\star)) .
\end{aligned} \tag{4.5}$$

La función que tienen los sumandos  $s_1$  y  $s_2$  es la de describir la distancia entre los dos vectores de media, mientras  $s_3$  define la distancia entre las matrices de covarianza. A continuación, analizaremos cada uno de estos sumandos y sus parámetros.

El primer sumando.  $s_1$ , está dado en función de una tolerancia  $\delta$ , una función de transformación  $T$  y una distancia  $d_\mu$  entre los vectores de media.

La elección de la función  $T$  describe cuándo dos vectores de media  $\mu_f(\mathbf{X}_\star) \neq \mu_g(\mathbf{X}_\star)$  deberían considerarse como iguales. Por ejemplo, como en un futuro queremos optimizar estos vectores, si se cumple que  $\mu_f(\mathbf{X}_\star) = 2\mu_g(\mathbf{X}_\star)$ , los puntos críticos de ambos vectores serán idénticos y podemos querer considerarlos como el mismo vector. Aunque la función  $T$  sea un parámetro escogido por el usuario, la implementación de medida de similitud que proponemos proporciona una función de transformación por defecto dada por  $T(\mu_f(\mathbf{X}_\star)) = |a|\mu_f(\mathbf{X}_\star) + b\mathbf{1}$  donde  $a$  y  $b$  son escalares calculados con el método de mínimos cuadrados [71] para ajustar lo mejor posible  $\mu_f(\mathbf{X}_\star)$  a  $\mu_g(\mathbf{X}_\star)$ . La transformación  $T$  propuesta refleja que dos vectores proporcionales y de diferencia constante se comportan idénticamente desde el punto de vista de la optimización.

La función de distancia  $d_\mu$  permite al usuario escoger cómo se desea medir la distancia entre  $T(\mu_f(\mathbf{X}_\star))$  y  $\mu_g(\mathbf{X}_\star)$ . Nuestra implementación ofrece varias opciones entre las que se debería escoger tomando en consideración la naturaleza del problema que el GP modela. Algunas de estas opciones son definir  $d_\mu$  como el número o porcentaje de puntos en los que  $T(\mu_f(\mathbf{X}_\star)) \neq \mu_g(\mathbf{X}_\star)$ , o como una norma  $p$  entre ambos vectores ( $\|\mu_g(\mathbf{X}_\star) - T(\mu_f(\mathbf{X}_\star))\|_p$ ) como la distancia euclídea, la norma infinito, etc.

Por último,  $d_\mu$  depende también del parámetro  $\delta$  que se utiliza para cambiar el nivel de tolerancia que el usuario da a  $d_\mu$ , es decir, hace que la distancia se calcule sólo utilizando los componentes de los vectores donde  $d_\mu$  es mayor que  $\delta$  en sus operaciones a nivel de componente. Por ejemplo, si la

distancia  $d_\mu$  es la dada por la norma  $p$  con  $p = 1$ ,  $s_1$  sería  $\sum_{|\mu_g - T(\mu_f)| > \delta} |\mu_g(\mathbf{X}_*) - T(\mu_f(\mathbf{X}_*))|$ .

El otro sumando de la WSM que compara los vectores de media es  $s_2$ . Éste es el único término que hemos fijado en la suma y representa la correlación de Pearson entre las medias de los GPs. El coeficiente de correlación de Pearson se define como

$$\rho(\mu_f(\mathbf{X}_*), \mu_g(\mathbf{X}_*)) = \frac{\mathbb{E}[(\mu_f(\mathbf{X}_*) - \overline{\mu_f(\mathbf{X}_*)})(\mu_g(\mathbf{X}_*) - \overline{\mu_g(\mathbf{X}_*)})]}{\sigma_{\mu_f} \sigma_{\mu_g}}, \quad (4.6)$$

donde  $\overline{\mu(\cdot)}$  denota la media de un vector  $\mu$  y  $\sigma_\mu$  su desviación típica.

Este operador nos interesa por su interpretación. El coeficiente de correlación  $\rho(\mu_f(\mathbf{X}_*), \mu_g(\mathbf{X}_*))$  devuelve un valor entre -1 y 1. Si es igual a uno, existe una función lineal positiva que describe el vector  $\mu_g(\mathbf{X}_*)$  a partir de  $\mu_f(\mathbf{X}_*)$ ; si es igual a -1 la función lineal tiene pendiente negativa y si es 0 no existe correlación lineal entre  $\mu_f(\mathbf{X}_*)$  y  $\mu_g(\mathbf{X}_*)$ .

De hecho, analizando la Ecuación (4.6) vemos que  $\rho$  es mayor si los dos vectores  $\mu_f(\mathbf{X}_*)$  y  $\mu_g(\mathbf{X}_*)$  crecen en los mismos intervalos y decrecen en los mismos intervalos. Por el contrario,  $\rho$  disminuye si su crecimiento es distinto (intervalos en los que un vector crece, el otro decrece). Esta interpretación del coeficiente es muy valiosa para nuestro problema ya que queremos identificar cuándo dos vectores crecen y decrecen de una manera parecida, es decir, los máximos y mínimos de un vector se encuentran cercanos a los del otro vector.

Por lo que hemos visto, este coeficiente es una manera precisa y eficiente de detectar procesos similares ya que es capaz de detectar la similitud entre vectores de muestras de funciones que, a priori y usando distancias convencionales diríamos que son diferentes, pero que su crecimiento es muy parecido.

Por último, el sumando  $s_3$  se encarga de medir la distancia entre los vectores de varianza a través de la función  $d_v$  escogida por el usuario. Al igual que con  $d_\mu$ , podemos usar cualquier medida de similitud entre vectores para este propósito. En nuestra metodología, proponemos por defecto la distancia euclídea. No hemos comprobado empíricamente que la distancia entre las varianzas sea significativa la hora de decidir si dos procesos Gaussianos deben optimizarse similarmente. Una hipótesis es que sólo se deba a la asunción que hicimos acerca de que las funciones de covarianza debían tener expresión analítica parecida. La razón de contemplar la varianza dentro de la herramienta a pesar de que por ahora no parezca significativa es que creemos que sería útil en trabajo futuro en el que contemplemos distintas funciones de covarianza, nos interese dar una incertidumbre a los cálculos de nuestra herramienta, etc.

### 4.1.3. Optimización Bayesiana

Ahora que conocemos los procesos Gaussianos, veremos cómo se integran dentro de la optimización Bayesiana. Ya hemos mencionado que la BO es una técnica de optimización global de funciones de expresión analítica desconocida, y nos interesa entender cómo se efectúa esta optimización.

Sea  $d$  la dimensión del espacio de entrada  $\mathcal{X}$  y  $\mathbf{x} = (x_1, \dots, x_d)$  un vector y  $f(\mathbf{x})$  la función objetivo que queremos optimizar. La BO con GPs asume que puede modelar la función objetivo con un proceso Gaussiano, es decir,  $f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{x}, k(\mathbf{0}, \mathbf{x}'))$  para cierta función de covarianza  $k$ . Tomamos  $\mathcal{D} = \{\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T, \mathbf{y} = (y_1, \dots, y_n)^T\}$  el conjunto de datos ya observados, con  $y_i = f(\mathbf{x}_i) + \epsilon_i$  para un ruido Gaussiano  $\epsilon_i$ .

La BO es un proceso iterativo donde, al comienzo de cada iteración y en función de  $\mathcal{D}$ , se optimiza una función de adquisición  $\alpha(\mathbf{x})$  en el dominio  $\mathcal{D}$  de la función objetivo  $f(\mathbf{x})$ . Tomando  $\mathbf{x}_{n+1} = \arg \max \alpha(\mathbf{x})$ , el algoritmo de BO evalúa la función objetivo en  $\mathbf{x}_{n+1}$  obteniendo así el valor  $y_{n+1} = f(\mathbf{x}_{n+1}) + \epsilon_{n+1}$  donde  $\epsilon_{n+1}$  es el ruido de la observación. Se añade  $\mathbf{x}_{n+1}$  a  $\mathbf{X}$ ,  $y_{n+1}$  a  $\mathbf{y}$ , habiendo actualizado el conjunto de observaciones  $\mathcal{D}$ , se inicia la siguiente iteración.

Tras un número finito de iteraciones (cuando el usuario está satisfecho con los valores óptimos recogidos en  $\mathcal{D}$ , cuando se consume el tiempo o presupuesto disponible para las evaluaciones de  $f(\mathbf{x})$ ...), algoritmo recomienda como valor óptimo la mejor observación recogida en  $\mathcal{D}$  ó el valor resultante de optimizar la media de la distribución predictiva del GP.

La función de adquisición es una función que representa un balance entre exploración y explotación del espacio  $\mathcal{D}$  tomando la información procedente de la distribución predictiva de un GP. Aplicada a un vector  $\mathbf{x}$  nos indica cuán conveniente o útil es evaluar la función objetivo en el punto  $\mathbf{x}$ . Un ejemplo de función de adquisición es la esperanza de mejora (EI). El toma el punto  $y' = \max_{y \in \mathbf{y}} y$  y dado un vector  $\mathbf{x}$  considera el valor  $i(\mathbf{x}) = \max(f(\mathbf{x}) - y', 0)$  como la mejora de  $\mathbf{x}$ . Teniendo en cuenta que podemos emplear la expresión 4.2 para predecir el valor  $f(\mathbf{x})$  para un vector  $\mathbf{x} \in \mathcal{D}$ , El calcula la esperanza de mejora como  $EI(\mathbf{x}) = \mathbb{E}[i(\mathbf{x})|\mathcal{D}]$ .

La Figura 4.1, cuyas imágenes han sido creadas utilizando el paquete `GPYOpt` de Python, es muy ilustrativa del funcionamiento de BO y del papel en ésta de la función de adquisición. En ella, se muestran seis iteraciones consecutivas de un proceso de optimización Bayesiana que busca minimizar la función dibujada en la Figura 4.1(a). En las imágenes sucesivas la curva azul muestra la media de la distribución predictiva del GP que modela la función objetivo, y las curvas grises y la región azul representan la incertidumbre acerca de la predicción en cada punto. Los puntos rojos sobre la curva de la media son los puntos en los que la función objetivo ha sido evaluada y por ello la incertidumbre en esos puntos es tan baja. La curva roja representada sobre el eje  $X$  es la función de adquisición, y la línea vertical roja señala el valor de  $x$  donde la función de adquisición alcanza su máximo. A cada iteración se explora un nuevo punto, el que hacía máxima la función de adquisición en la iteración

anterior.

Por último, en el Algoritmo 4.1 el lector puede ver reflejado el funcionamiento de la optimización Bayesiana explicado a lo largo de esta subsección.

```

entrada: El conjunto  $D$  de observaciones, la función objetivo  $f$ , un entero  $\text{maxiter} \geq 0$  que indica el número
           máximo de iteraciones.
salida : El vector recomendado como máximo
1  for  $i \leftarrow 0$  to  $\text{maxiter}$  do
2       $\mathbf{x}' \leftarrow \arg \max \text{acq}(\mathbf{x}, f, D);$ 
3       $D \leftarrow D \cup (\mathbf{x}', f(\mathbf{x}'));$ 
4       $i \leftarrow i + 1$ 
5  end
6  return  $\text{best}(D, f)$ 

```

**Algoritmo 4.1:** Algoritmo de optimización Bayesiana

#### 4.1.4. Optimización Bayesiana multiobjetivo

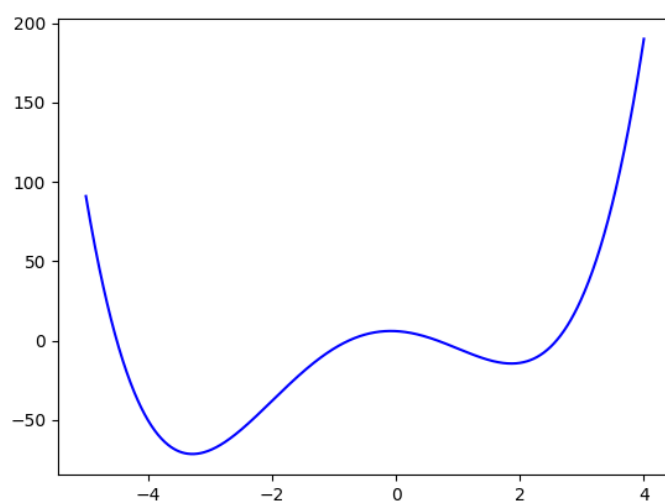
El algoritmo de BO se puede generalizar para optimizar varios objetivos. Los escenarios donde tenemos hasta tres funciones objetivo a optimizar se recogen en la optimización Bayesiana multiobjetivo (MOBO). En este caso, cada una de las funciones objetivo es modelada por un GP y la función de adquisición debe combinar las distribuciones predictivas de cada GP dados unos datos de entrada  $D$ .

Un ejemplo de función de adquisición en escenarios multiobjetivo es la esperanza de mejora del hipervolumen (EHVI) [72]. El hipervolumen es un indicador que, dado un conjunto de vectores  $\mathbf{X}' = \{\mathbf{x}_1, \dots, \mathbf{x}_l\}$ , un punto de referencia  $\mathbf{x}_r$  y una relación de orden  $\prec$ , devuelve la medida de Lebesgue del conjunto  $\{\mathbf{y} \mid \exists \mathbf{x} \in \mathbf{X}' : \mathbf{x} \prec \mathbf{y} \prec \mathbf{x}_r\}$ . Denotamos  $\mathcal{H}(\mathbf{X}')$  como el hipervolumen del conjunto  $\mathbf{X}'$

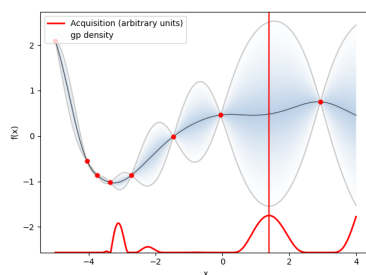
La relación de orden utilizada en MOBO es la dominancia ya introducida en la Sección 2.3, que considera que una solución es mayor que otra si es mejor que ella en todos los objetivos. El conjunto sobre el que calculamos el hipervolumen es el conjunto de Pareto, denotado  $\mathcal{X}^*$  y formado por las soluciones observadas que no están dominadas por ninguna otra del conjunto. Así, dado un punto  $\mathbf{x}$  calculamos su mejora como  $i(\mathbf{x}) = \mathcal{H}(\mathcal{X}^* \cup \{\mathbf{x}\}) - \mathcal{H}(\mathcal{X}^*)$ . De manera similar a EI, EHVI devuelve la esperanza de mejora de un vector  $\mathbf{x}$ .

Los escenarios de optimización de cuatro o más objetivos no se contemplan dentro de MOBO sino de la llamada optimización Bayesiana de muchos objetivos. En estos escenarios, la alta dimensionalidad provoca que el conjunto de Pareto crezca de manera exponencial y por tanto explorarlo en problemas con cuatro o más funciones objetivo requiere muchas evaluaciones de los objetivos. Como la optimización Bayesiana se ocupa de optimizar objetivos que son muy costosos de evaluar, explorar el conjunto de Pareto es impensable en problemas de optimización Bayesiana de muchos objetivos.

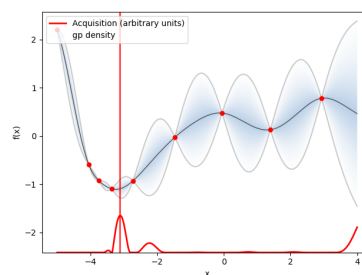




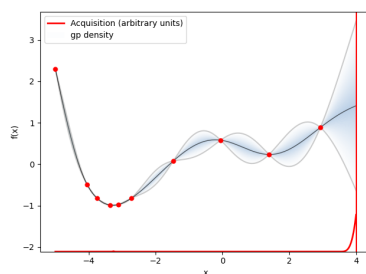
(a) Función objetivo



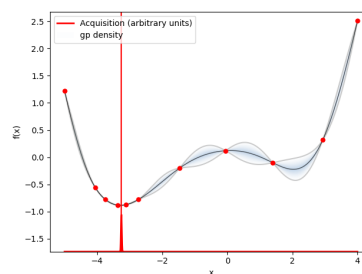
(b) Primera iteración



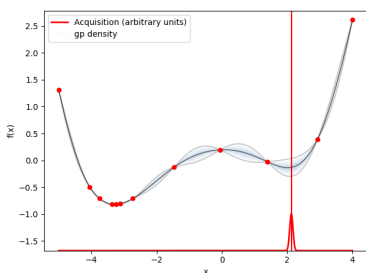
(c) Segunda iteración



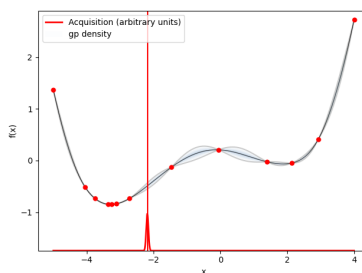
(d) Tercera iteración



(e) Cuarta iteración



(f) Quinta iteración



(g) Sexta iteración

**Figura 4.1:** Iteraciones sucesivas de BO

### 4.1.5. Optimización Bayesiana de muchos objetivos

Hasta donde nuestro conocimiento de la literatura alcanza, la optimización Bayesiana de muchos objetivos es un problema todavía sin solución, y en este trabajo proponemos una solución basada en la reducción de objetivos. Nuestro propósito es detectar cuándo un problema de más de tres objetivos puede ser reducido a otro con tres o menos objetivos, hacer la reducción y resolver el nuevo problema empleando MOBO.

En cada iteración de la optimización Bayesiana, el algoritmo dispone de un conjunto  $\mathcal{D}$  que recoge todos los datos observados. La distribución predictiva de un proceso Gaussiano dada en 4.2 nos permite conocer la distribución de  $f(\mathbf{x})|\mathcal{D}$  para cualquier otro punto  $\mathbf{x}$  del dominio. Nuestro enfoque utiliza esta distribución predictiva de GPs y la medida de similitud desarrollada en la Subsección 4.1.2 para detectar cuándo dos procesos Gaussianos son parecidos y podemos prescindir de uno de ellos. A continuación explicamos el algoritmo desarrollado para ello.

Sean  $f_1(\mathbf{x}), \dots, f_d(\mathbf{x})$  las funciones objetivo contempladas en el problema de optimización Bayesiana. En cierta iteración de la optimización, sea  $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)^T$  la matriz con  $\mathbf{x}_i$  los puntos que ya han sido observados. Para cada  $\mathbf{x}_i$  el vector  $\mathbf{y}_i = (y_{i1}, \dots, y_{id})$  representa las evaluaciones (ruidosas) de los objetivos en el punto  $\mathbf{x}_i$ , es decir,  $y_{ij} = f_j(\mathbf{x}_i) + \epsilon_{ij}$  para cierto ruido Gaussiano  $\epsilon_{ij}$ . Sea  $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_n)^T$  la matriz con todas las evaluaciones. Denotamos el conjunto de datos observados como  $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ .

Empleando  $\mathcal{D}$  para calcular la distribución predictiva de cada objetivo, computamos  $D = \{d_{ij} = d(f_i(\mathbf{x}), f_j(\mathbf{x})) \mid 1 < i < j < n\}$  donde  $D$  es un conjunto que recoge la similitud entre cada par de objetivos de la optimización. A cada iteración, nuestro algoritmo calcula el conjunto  $D$  y lo recorre. Si encuentra un valor  $d_{ij} < \epsilon$  para cierto  $\epsilon$  especificado por el usuario, nuestro algoritmo elimina el objetivo  $f_i(\mathbf{x})$  de la optimización por considerarlo redundante y así, a la siguiente iteración la optimización se ejecuta con un objetivo menos.

Para asegurarnos de no perder demasiada información acerca del problema de una vez, nuestro algoritmo limita a 1 el número de objetivos que pueden reducirse en una iteración dada. Además, desconocemos el tamaño de  $\mathcal{D}$  en la primera iteración y cuántas iteraciones tarda la optimización Bayesiana en aproximar acertadamente las funciones objetivo. En algunos problemas podríamos comenzar con  $\mathcal{D}$  de gran tamaño y que las distribuciones predictivas en las primeras iteraciones ya fueran acertadas, y en otros problemas podríamos tardar más iteraciones para poder aproximar las funciones objetivo. Así, nuestro algoritmo depende de un parámetro  $\delta$  escogido por el usuario en función de la naturaleza de la optimización y la reducción de objetivos no se pone en práctica hasta la  $\delta$ -ésima iteración.

El Algoritmo 4.2 resume los pasos descritos en los anteriores párrafos.

**entrada:** Un entero  $iter \geq 0$  que representa el número de iteración de la optimización, un entero  $\delta \geq 0$  que indica en qué iteración comenzar la reducción, una tolerancia real  $0 \leq \varepsilon \leq 1$ ,  $obj$  una lista de longitud  $dim$  conteniendo los modelos de las funciones objetivo.

**salida :** La lista de objetivos después de haber sido reducida

```

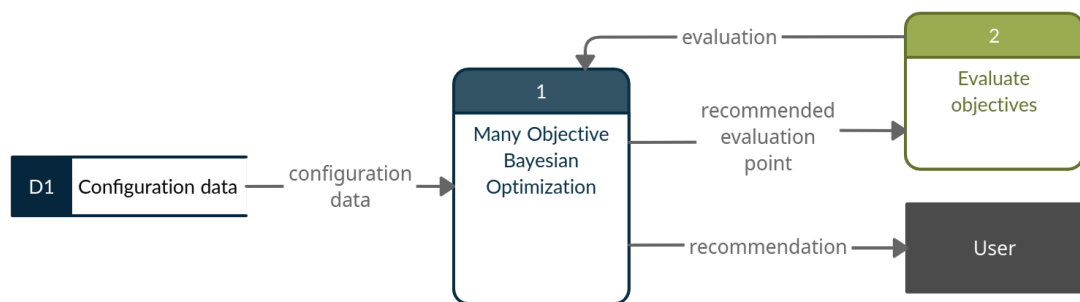
1  if  $iter \geq \delta$  then
2      ndim  $\leftarrow$  Length (  $obj$  );
3      for  $i \leftarrow 1$  to ndim do
4          for  $j \leftarrow i + 1$  to ndim do
5              distances [ $obj[i], obj[j]$ ]  $\leftarrow$  Distance (  $obj[i], obj[j]$  );
6          end
7      end
8      foreach Distance [ $i, j$ ]  $\in$  distances do
9          if distances [ $i, j$ ]  $< \varepsilon$  then
10             return Delete (  $obj[i]$  )
11          end
12      end
13  end

```

**Algoritmo 4.2:** Reducción del número de objetivos de un problema de optimización Bayesiana de muchos objetivos empleando la medida de similitud entre GPs

## 4.2. Diseño técnico

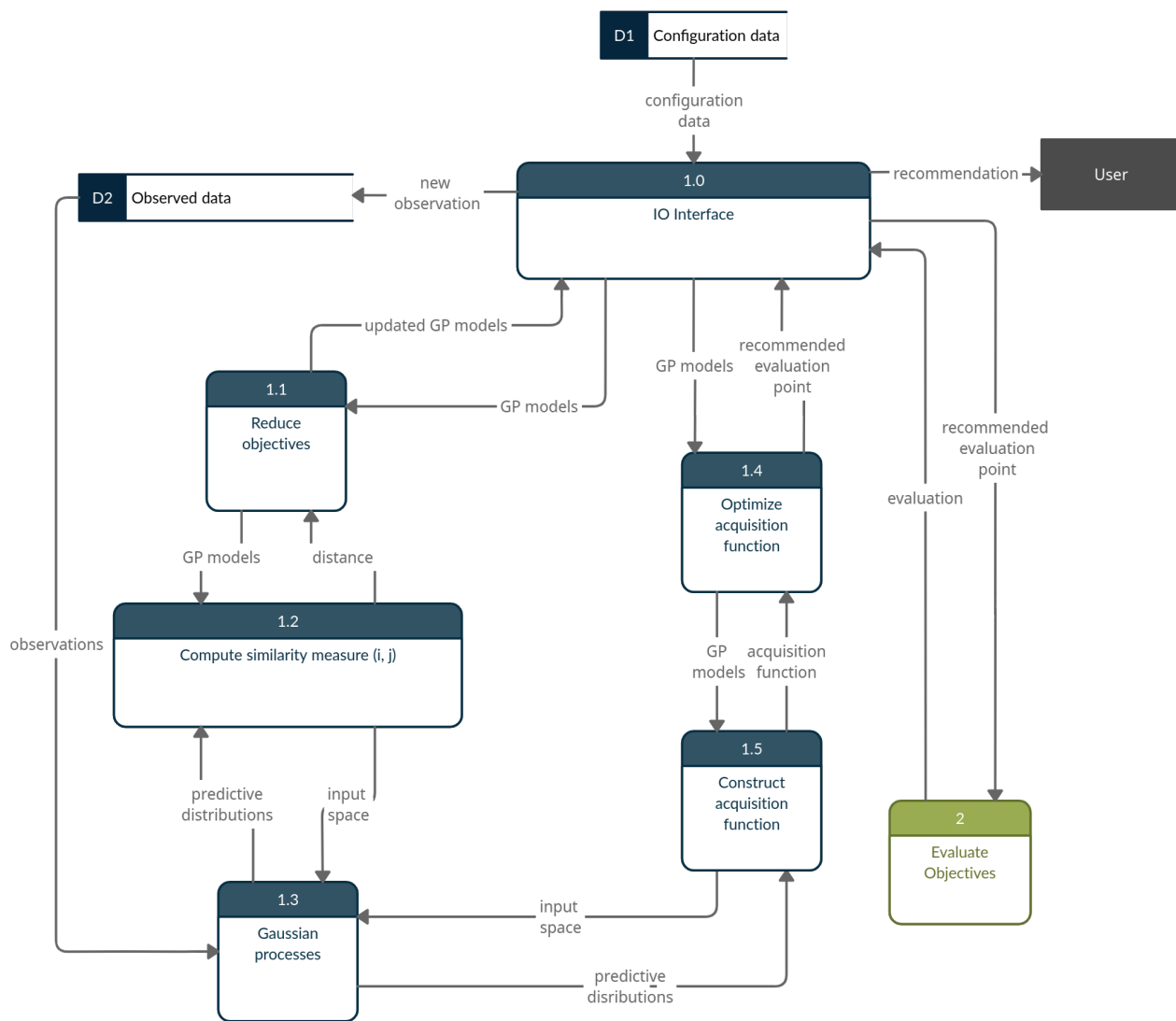
En esta sección describiremos el diseño técnico de esta tesis. Para ello, nos ayudaremos de los dos diagramas de flujo de datos ilustrados en las siguientes figuras: en la Figura 4.2 encontramos el diagrama de nivel cero o diagrama de contexto, que nos proporciona un panorama básico del sistema, y en la Figura 4.3 encontramos un diagrama de nivel uno en el que desglosamos el proceso principal, el proceso 1 (Many objective Bayesian optimization).



**Figura 4.2:** Diagramas de flujo de datos de nivel 0

Comenzamos desarrollando la información que contiene el diagrama de datos de nivel cero, la Figura 4.2. En él, D1 es un almacén de datos, User es una entidad externa y el proceso 2 es un proceso externo. El proceso 1 es el principal, encargado de llevar a cabo la optimización Bayesiana. Para ello, este proceso lee del almacén de datos D1 toda la configuración del algoritmo (número máximo de

iteraciones, objetivos, función de adquisición escogida, iteración a partir de la cual ejecutar la reducción de objetivos, entre otros). A lo largo de la optimización, el proceso 1 se comunica con el proceso 2, encargado de evaluar las funciones objetivo. El proceso 1 envía al proceso 2 el punto en el que se van a evaluar los objetivos y éste le devuelve las evaluaciones computadas. Cuando la optimización se da por terminada, el proceso 1 transmite al usuario, representado como una entidad externa, el punto sugerido como óptimo.



**Figura 4.3:** Diagramas de flujo de datos de nivel 1

Veamos ahora cómo se desglosa el proceso 1 a través del diagrama de flujo de datos de nivel uno de la Figura 4.3. Toda la información de configuración D1 llega a la interfaz de entrada/salida, el proceso 1.0. Este proceso será el encargado de comunicar entre sí los distintos aspectos de la optimización.

En primer lugar, el proceso 1.1 recibe del proceso 1.0 los GPs que modelan los objetivos. El proceso 1.1 es el encargado de ejecutar el algoritmo de reducción de objetivos, para lo que necesita

comunicarse con el proceso 1.2 que calcula la similitud entre dos GPs dados. Para ello, el proceso 1.2 recibe los GPs del proceso 1.1, transmite al proceso 1.3 el espacio de entrada de estos GPs y recibe del mismo las distribuciones predictivas de los GPs. El proceso 1.3 es por tanto el encargado de manejar los procesos Gaussianos, y calcula las distribuciones predictivas de estos gracias al conjunto de datos observados disponible en el almacén de datos D2. Así, el proceso 1.2 es capaz de devolver al proceso 1.1 la distancia entre dos objetos y el proceso 1.1 entrega al proceso 1.0 el conjunto de GPs actualizado y posiblemente reducido.

A continuación, se calcula cuál será el próximo punto en el que evaluaremos los objetivos. Para ello el proceso 1.5, recibiendo los GPs del proceso 1.4 y sus distribuciones predictivas del proceso 1.2 (distribuciones que obtiene enviándole el espacio de entrada de los GPs), se encarga de calcular la función de adquisición. Ésta es transmitida al proceso 1.4 que la optimiza y devuelve al proceso 1.0 su máximo, que es el punto donde se recomienda hacer la próxima evaluación de objetivos.

Así, el proceso 1.0 envía al proceso 2 este punto a evaluar y, como ya hemos explicado en el diagrama de nivel 0, recibe a cambio su evaluación. El proceso 1.0 es encargado de guardar en el almacén D2 esta nueva observación. Cuando el algoritmo termina, el valor óptimo se hace llegar al Usuario desde el proceso 1.0.

## 4.3. Planificación del proyecto

A continuación, detallaremos cuál ha sido la planificación del tiempo de este proyecto desde su asignación hasta su entrega. El proyecto ha sido desarrollado mediante la metodología en cascada, dividiéndolo en etapas de tal manera que las tareas de cada etapa comenzaban cuando se habían cerrado las de la etapa anterior. La naturaleza del proyecto exigía hacer un análisis de la BO y los GPs antes de diseñar la medida de similitud y probarla, diseñar la medida antes de integrarla en un algoritmo de reducción de objetivos, etc. y por ello esta metodología secuencial se adecuaba al proyecto. Ilustramos la planificación a través de un diagrama de Gantt que más adelante explicaremos en profundidad y que el lector puede consultar en la Figura 4.4 y en la Figura 4.5. El proyecto se ha desarrollado entre el 1 de septiembre de 2020 y el 20 de junio de 2021 a lo largo de 38 semanas de trabajo y 3 semanas de descanso. En él hemos invertido aproximadamente 298 horas, lo que supone una media de 7 horas y 51 minutos por semana.

Describimos y comentamos ahora cada una de las tareas y grupos de tareas especificados en el diagrama, junto con su grado de dificultad.

- Estudio. En esta primera fase del proyecto, dedicamos algo más de cinco semanas a entender los conceptos que íbamos a trabajar a lo largo de todo el curso. Se dividió en distintas subtarefas:
  - Estudio de procesos Gaussianos a través de clases grabadas y publicadas en Youtube por

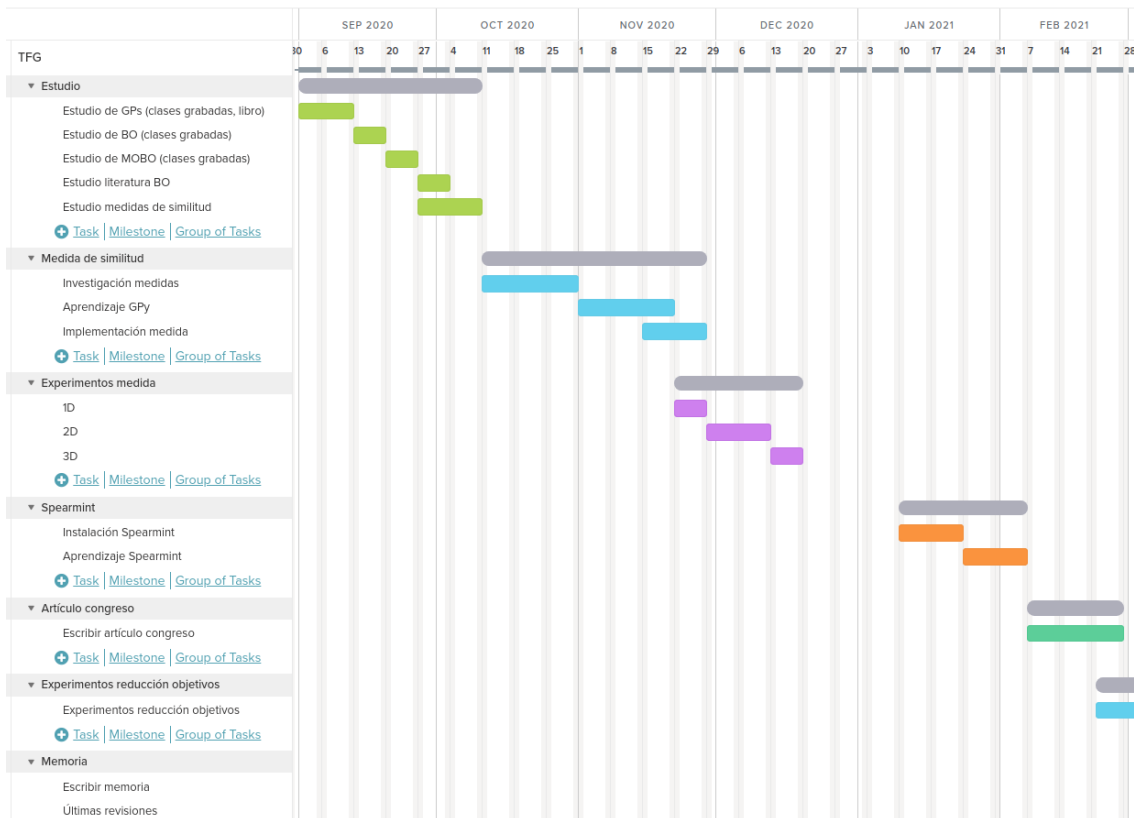


Figura 4.4: Diagrama de Gantt

el profesor e investigador Nando de Freitas. Nivel de dificultad: medio-alto.

- Estudio de la optimización Bayesiana mediante clases grabadas y publicadas en Youtube de nuevo por el profesor e investigador Nando de Freitas. Nivel de dificultad: medio.
  - Estudio de optimización Bayesiana multiobjetivo a través de diversas charlas de conferencias publicadas en Youtube. Entre ellas, la “Conference on Uncertainty in Artificial Intelligence”. Dificultad alta.
  - Estudio de literatura acerca de procesos Gaussianos, por ejemplo a través del libro de Rasmussen [53]. Dificultad media.
  - Estudio introductorio a las medidas de similitud de distribuciones de probabilidad, a través de la búsqueda de artículos. Nivel de dificultad: medio.
- Medida de similitud. Este segundo grupo de tareas, que tomó aproximadamente siete semanas, se centró en el desarrollo de la medida de similitud para procesos Gaussianos. Podemos dividirla en tres subtareas:
- Investigación de medidas. Ésta fue la subtarea más compleja de este grupo, consistente en probar distintas nociones de similitud e, intentando entender qué queríamos conseguir teniendo en cuenta la naturaleza de los procesos Gaussianos, desarrollar una medida acorde. Nivel de dificultad: medio.

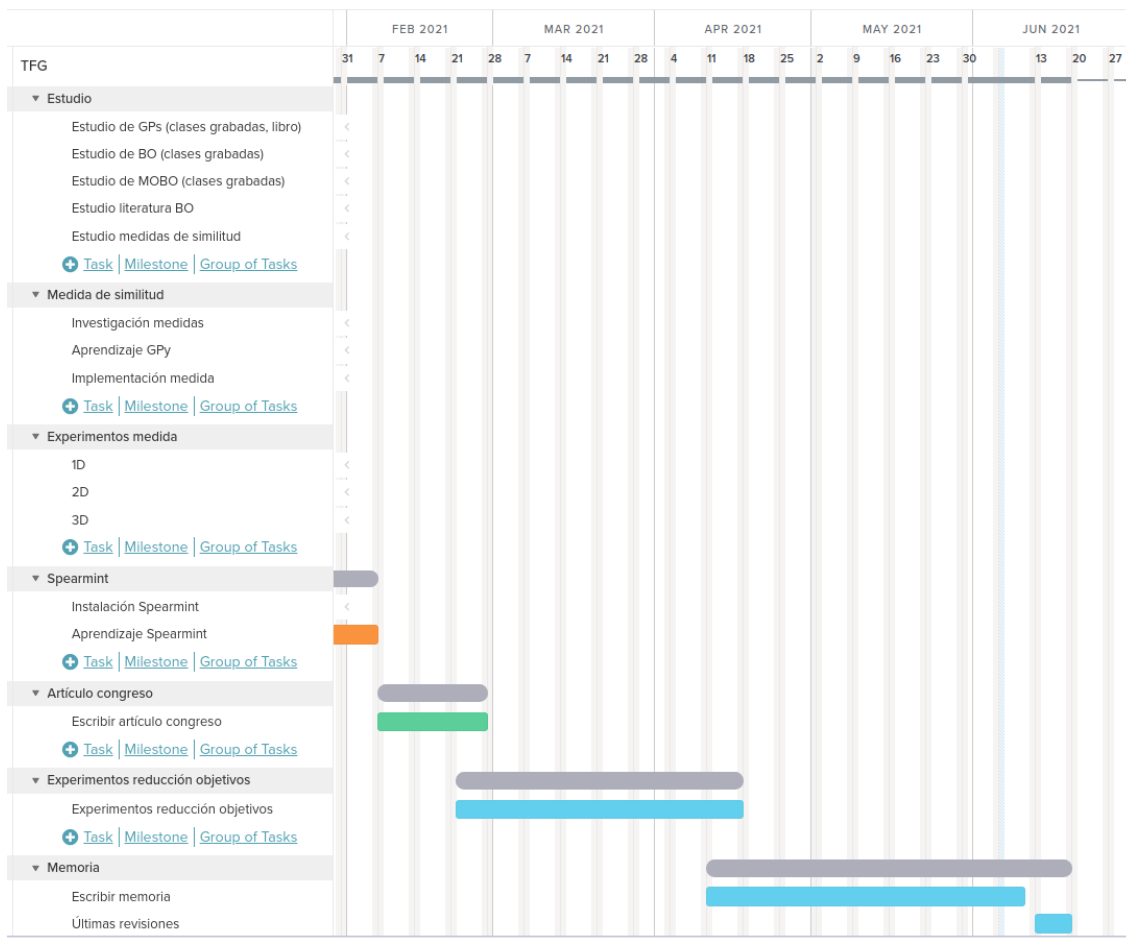


Figura 4.5: Diagrama de Gantt

- Aprendizaje del paquete GPy: familiarización y prueba de este paquete de Python que permite el trabajo con GPs. Nivel de dificultad: bajo.
- Implementación de medida de similitud: implementación en Python de la distancia desarrollada. Nivel de dificultad: bajo
- Experimentos sobre la medida: dentro de este grupo de tareas, que se llevó a cabo durante unas cuatro semanas, probamos la medida de similitud para optimización Bayesiana de objetivos cuyo espacio de entrada era de una, dos o tres dimensiones.
  - Experimentos objetivos 1D. Nivel de dificultad: bajo.
  - Experimentos objetivos 2D. Aquí experimentamos con muchas funciones de benchmark, ya que en dos dimensiones era más difícil entender la noción de similitud. Nivel de dificultad: alto.
  - Experimentos 3D: la imposibilidad de visualizar gráficas de funciones de tres dimensiones dificultó esta tarea, ya que no podíamos conocer la misma cantidad de funciones que en el caso bidimensional. Nivel de dificultad: medio.
- Spearmint: En este punto del trabajo introducimos la herramienta Spearmint para probar nuestro algoritmo de reducción de objetivos. Tomó unas cuatro semanas y se dividió en dos subtareas.
  - Instalación de la herramienta Spearmint. Nivel de dificultad: alto debido a la pobre compatibilidad de la herramienta con versiones semirecientes de los paquetes que requiere y a la casi nula documentación.
  - Aprendizaje Spearmint: durante estas semanas necesitamos entender el funcionamiento de la herramienta para así integrar en ella nuestro algoritmo de reducción de objetivos y la distancia implementada. Nivel de dificultad: alto debido a la falta de documentación de la herramienta y a la complejidad de su estructura.
- Escribir artículo congreso: Durante tres semanas escribimos y formalizamos la medida de similitud y los experimentos realizados para presentar el trabajo en el congreso organizado por la Asociación Española para la Inteligencia Artificial (CAEPIA). Nivel de dificultad: medio-alto debido a la inexperiencia en la escritura de textos científicos.
- Experimentos reducción de objetivos: durante estas semanas realizamos distintos experimentos de optimización Bayesiana de más de un objetivo para comprobar el funcionamiento de nuestro algoritmo de reducción de objetivos. Nivel de dificultad: medio-alto.
- Memoria: los últimos dos meses de trabajo son dedicados a la escritura del documento de memoria del trabajo.
  - Escribir memoria. El grueso de la memoria se realiza en esta subtarea. Nivel de dificultad: alto.
  - Últimas revisiones junto al tutor del documento. Nivel de dificultad: bajo.



# IMPLEMENTACIÓN

---

En esta sección detallaremos los aspectos del proyecto que conciernen a su implementación para que todos los experimentos puedan reproducirse en un futuro. Especificaremos el sistema operativo sobre el que se ha trabajado, los lenguajes de programación que hemos utilizado, los entornos de programación así como todos los paquetes necesarios para ejecutar el código del proyecto. Además, explicaremos los ficheros de código que hemos creado a lo largo del año.

Todo el trabajo ha sido realizado utilizando el sistema operativo Ubuntu en su versión 20.04.2 LTS (Ubuntu focal), y el software empleado es libre.

El lenguaje de programación utilizado ha sido Python. Para la primera implementación de la medida de similitud y para los experimentos que hicimos de la misma, la versión de Python empleada ha sido la 3.7. Para la integración en Spearmint del algoritmo de reducción de objetivos hemos necesitado la versión 2.7, que es la que la herramienta Spearmint requiere.

Con Python 2.7 hemos utilizado la distribución Anaconda en su versión conda 4.5.11.

El código de la medida de similitud y sus experimentos se ha desarrollado en cuadernos de Jupyter en la versión 5.6.0, y el código del algoritmo de reducción de objetivos en el editor de texto VIM en la versión 8.1 del mismo.

Para el desarrollo de la medida de similitud y sus experimentos hemos utilizado todos los paquetes que se muestran en el Apéndice A.2.

En la segunda parte del proyecto hemos probado el algoritmo de reducción de objetivos integrándolo en la herramienta Spearmint. Este trabajo se ha desarrollado en un entorno virtual de Python 2.7, con los paquetes listados en el Apéndice A.1. La herramienta Spearmint se instaló clonando el código disponible en este repositorio de GitHub. Para la instalación, fueron cruciales las versiones de las librerías `boost`, `boost-cpp`, `numpy`, `libgcc` y `pygmo` que se muestran en el Apéndice A.1.

A continuación, nos centraremos en los ficheros que han sido necesarios para la implementación del proyecto. Todos ellos están disponibles en este repositorio, que es un *fork* de la herramienta Spearmint ya mencionada.

Comenzaremos por los ficheros relacionados con la medida de similitud, y luego veremos los que involucran al algoritmo de BO de muchos objetivos. La medida de similitud se ha implementado dentro del fichero `distances.py` que se encuentra en el directorio `spearmint/utills`. En particular, nuestra distancia se ha implementado en la función `distance` de este fichero, que refleja la funcionalidad del proceso 1.2 del diagrama de flujo de datos de la Figura 4.3. Este fichero corresponde a la tarea “Medida de similitud/Implementacion medida” del diagrama de la Figura 4.4. En esta parte del trabajo, además del fichero `distances.py` hemos utilizado los ficheros del directorio `expt_distance`. Dentro de él, el fichero `correlation_study.ipynb` fue implementado para el estudio de distintas opciones de medida durante la tarea “Medida de similitud/Investigación medidas” que reflejamos en el diagrama de Gantt de la Figura 4.4. Los demás ficheros de ese directorio (`dimension1.ipynb`, `dimension2.ipynb`, `dimension2plus.ipynb`, `dimension3.ipynb` y `experimentos.ipynb`) implementan los experimentos realizados sobre la medida de similitud, reflejados en las tareas dentro de “Experimentos medida” del diagrama de Gantt ya referenciado.

Nos centramos ahora en los ficheros que conciernen al algoritmo de optimización Bayesiana de muchos objetivos. La optimización Bayesiana se lleva a cabo utilizando la herramienta *Spearmint*, cuyo código se encuentra bajo el directorio `spearmint`. Mencionamos ahora los ficheros implicados en algunas de las funcionalidades más importantes de la optimización. El proceso 1.0 del DFD de la Figura 4.3 está implementado en el fichero `spearmint/main.py`, y el algoritmo de reducción de objetivos (proceso 1.2 del mismo diagrama) en el fichero `spearmint/choosers/default_chooser.py`, en particular en la función `suggest_redundant_process`. También está implementada en este fichero, en las funciones `suggest` y `compute_acquisition_function`, la funcionalidad de los procesos 1.4 y 1.5 del DFD de nivel 1. La funcionalidad de los procesos Gaussianos (proceso 1.3 del diagrama de la Figura 4.3) se realiza en el fichero `spearmint/models/gp.py`.

Si analizamos el DFD de la Figura 4.3, sólo queda entender cómo se implementan los almacenes de datos y el proceso 2. Estos tres elementos dependen del experimento de optimización en cuestión. Cada experimento tiene su propio directorio, algunos ejemplos de esto se encuentran bajo el directorio `experiments`. Cada experimento contiene un fichero de configuración `config.json` que corresponde al almacén de datos D1. Los datos observados D2 son almacenados por *Spearmint* utilizando una base de datos de MongoDB y, por último, el proceso 2 evalúa los objetivos mediante la evaluación de funciones cuyos nombres y ficheros contenedores se especifican a la herramienta *Spearmint* a través del fichero `config.json`.

Por último, los ficheros de experimentos del algoritmo de reducción de objetivos pueden encontrarse en el directorio `expt_reduction`. En él, cada directorio corresponde a uno de los experimentos explicados en la Sección 6.2. Para facilitar los experimentos hemos utilizado además el script que se encuentra en el fichero `expt_reduction/test.py`. Así, queda explicada la implementación del proyecto y podemos centrarnos en estudiar los experimentos realizados durante el curso.

## EXPERIMENTACIÓN

---

En este capítulo explicaremos los experimentos que hemos llevado a cabo a lo largo de este proyecto, ya que éstos nos permitirán contrastar las hipótesis hechas en el Capítulo 3. Para cada experimento, describiremos las condiciones del mismo y luego daremos sus resultados. Dividimos el capítulo en dos secciones, la primera para exponer los experimentos realizados sobre la medida de similitud y la segunda para probar el algoritmo de reducción de objetivos.

### 6.1. Experimentos de la medida de similitud

Comencemos con los experimentos que hemos llevado a cabo para probar la medida de similitud para procesos Gaussianos desarrollada en la Sección 4.1.2. En cada uno de ellos, escogemos varias funciones de benchmark y tomamos procesos Gaussianos que ajustamos a cada una de ellas tomando vectores de muestra de las funciones para ilustrar el funcionamiento de la medida. Empleamos la distancia implementada para calcular la similitud entre estos procesos Gaussianos e interpretamos los resultados.

En todos los experimentos los parámetros de la medida de similitud han sido configurados como sigue:

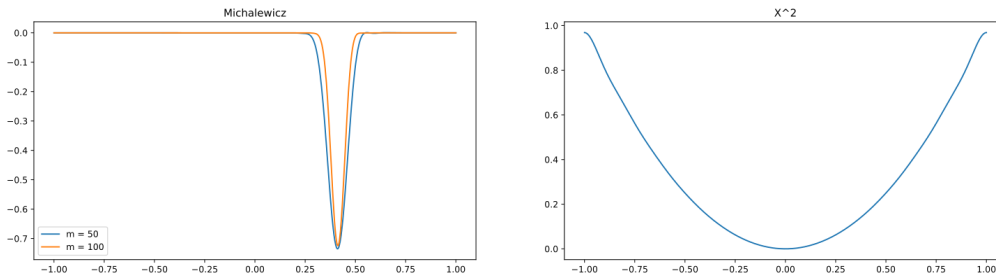
- La tolerancia  $\delta$  se ha tomado como 0.
- En la suma ponderada,  $\varepsilon_1 = 0,25$  y  $\varepsilon = 0,75$  (por tanto, el peso asignado a la distancia entre las varianzas es cero).
- La transformación  $T$  es la transformación lineal descrita en la Sección 4.1.2.
- La distancia  $d_1$  la implementamos como la distancia media relativa entre los puntos de los vectores de media, es decir, la media del vector dado por  $|T(\mu_f(\mathbf{X}_*)) - \mu_g(\mathbf{X}_*)|$  dividido por la resta del máximo menos el mínimo elemento del vector de media.

A continuación, describimos cada uno de los experimentos realizados.

### 6.1.1. Primer experimento: GPs de una dimensión.

Comenzamos comparando tres GPS que se ajustan a funciones de juguete cuyo dominio es unidimensional. Hemos escogido dos funciones que sabemos que son parecidas y una tercera que se comporta de manera diferente. En la Figura Fig.6.1 hemos representado los vectores de media de los tres GPs.

El primer GP modela una función de *Michalewicz*, que se define como  $f_1(x) = -\sin x \left(\sin \frac{x^2}{\pi}\right)^{2m}$  con el parámetro  $m = 50$ , el segundo GP modela otra función de *Michalewicz* con el parámetro  $m = 100$  y el tercer GP modela una parábola  $x^2$ .



**Figura 6.1:** Vectores de media de GPs que ajustan algunas funciones unidimensionales.

Los resultados del experimento han sido los siguientes. La correlación entre las medias predictivas de los dos GPs de *Michalewicz* es de 0.97, y su distancia media relativa es de 0.02. En total, distancia calculada es de 0.02 sobre 1, lo que quiere decir que los dos GPs son muy parecidos de acuerdo a nuestra medida de similitud.

Por otro lado, comparando el GP de *Michalewicz* con  $m = 100$  con la parábola, obtenemos una correlación de 0.12 que refleja que ambos procesos son muy diferentes en cuanto a su crecimiento. La distancia media relativa es de 0.27 y la distancia total es de 0.72 sobre 1, es decir, son procesos muy distintos.

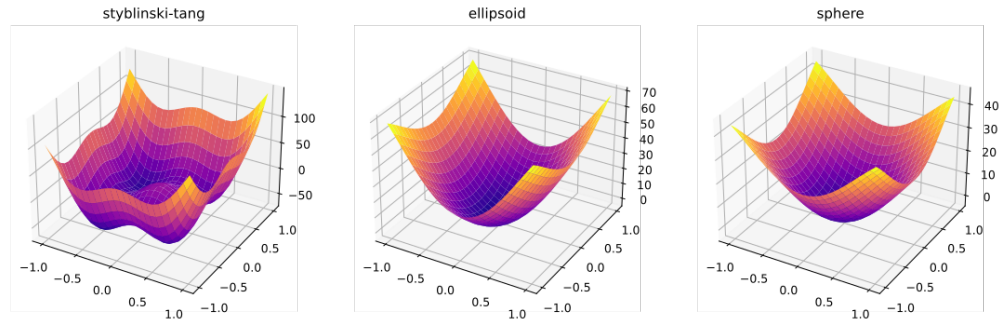
### 6.1.2. Segundo experimento: GPs bidimensionales con forma de cuenco

A continuación vamos a comparar tres GPs que ajustan funciones con forma de cuenco. Uno de ellos modela una función de *Styblinski-Tang*, el segundo un elipsoide y el tercero una esfera.

La función de *Styblinski-Tang* es la dada por  $f_2(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^2 (x_i^4 - 16x_i^2 + 5x_i)$ , el elipsoide es  $f_3(\mathbf{x}) = \sum_{i=1}^2 \sum_{j=1}^i x_j^2$  y la esfera  $f_4(\mathbf{x}) = x_1^2 + x_2^2$ .

Las medias de los tres procesos están ilustrados en la Figura 6.2. De esta imagen podemos deducir que, como los tres tienen forma de cuenco, serán procesos similares. Sin embargo, la forma del

proceso de *Styblinski-Tang* es ligeramente distinta a las de los otros dos.



**Figura 6.2:** Vectores de media de GPs que ajustan algunas funciones bidimensionales en forma de cuenco.

El resultado del experimento ha reflejado que la correlación entre esfera y elipsoide es de 0.94, y su distancia media relativa es 0.6. En total, la distancia entre estos dos procesos es 0.05 por lo que efectivamente estos procesos son muy parecidos.

En contraste, si comparamos el proceso de *Styblinski-Tang* con el elipsoide la correlación es de 0.74, un valor alto pero no tan cercano a 1 como en el caso anterior, reflejando el hecho de que estos dos procesos no son tan parecidos como la esfera y el elipsoide. La distancia media relativa entre los vectores de media es 0.13 (un valor bajo pero de nuevo mayor que en el caso anterior, el doble) y la distancia total es 0.22 sobre 1.

### 6.1.3. Tercer experimento: GPs bidimensionales muy distintos

Ahora compararemos dos procesos que ajustan funciones que sabemos de antemano que no son parecidas, las ilustradas en la Figura 6.3.

El primer GP modela una función de *Griewank*, cuya expresión matemática es

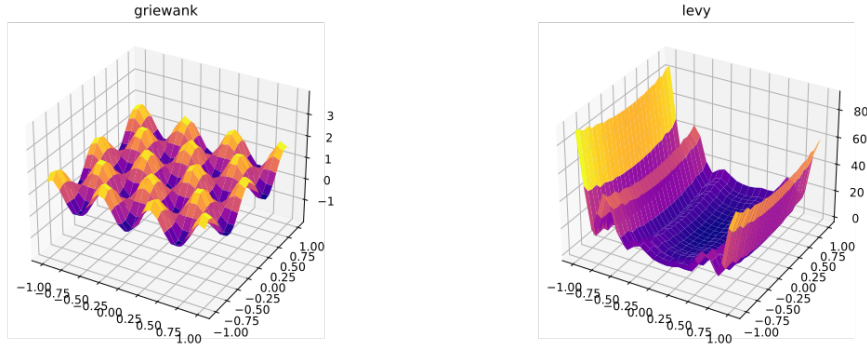
$$f_5(\mathbf{x}) = \frac{1}{4000} (x_1^2 + x_2^2) - \cos(x_1) \cos\left(\frac{x_2}{\sqrt{2}}\right) + 1,$$

y el segundo una función de Levy

$$f_6(\mathbf{x}) = \sin^2(\pi w_1) + (w_1 - 1)^2 \left(1 + 10 \sin^2(\pi w_1 + 1)\right) + (w_2 - 1)^2 (1 + \sin^2(2\pi w_2)) \left($$

donde  $w_i = 1 + (x_i - 1)^{\frac{1}{4}}$

Los resultados del experimento han sido los siguientes: como la forma de ambos procesos es muy distinta, la correlación entre las medias es muy baja, 0.04. Su distancia relativa es 0.16 y por tanto la



**Figura 6.3:** Vectores de media de GPs que ajustan algunas funciones bidimensionales muy diferentes.

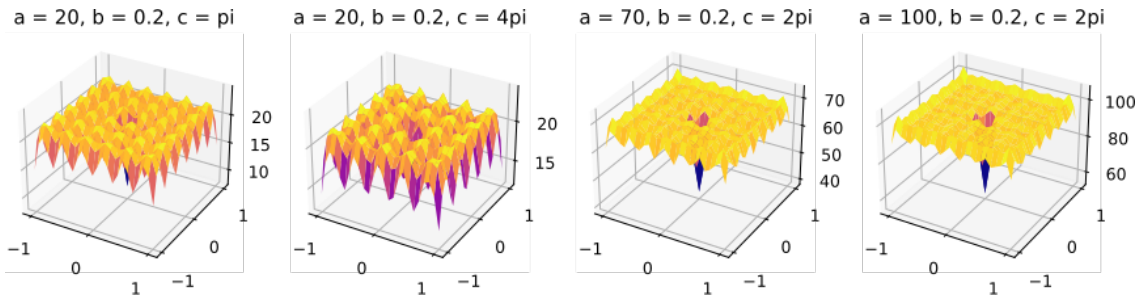
distancia relativa es de 0.75 sobre 1. Como esperábamos, una distancia grande.

#### 6.1.4. Cuarto experimento: GPs ajustando funciones Ackley

Por último, vamos a comparar distintos procesos que ajustan funciones de Ackley. Recordamos que la función de Ackley es aquella dada por la expresión

$$f_7(\mathbf{x}) = -a \exp \left( -b \sqrt{\frac{1}{d} \sum_{i=1}^d f_i^2} \right) \left( \exp -\frac{1}{2} \sum_{i=1}^2 \cos(cx_i) \right) + a + e$$

En primer lugar comparamos dos procesos con el parámetro  $a = 20$  y  $b = 0.2$ . En uno de ellos,



**Figura 6.4:** Vectores de media de GPs que ajustan algunas funciones Ackley.

fijaremos  $c = \pi$  y en el otro  $c = 6\pi$ , que resulta en dos procesos muy distintos como se ilustra en la Figura 6.4.

Para este caso, el resultado del experimento ha sido una correlación de 0.31 entre ambos procesos, y una distancia media relativa de 0.5, dando una distancia total de 0.55 sobre 1.

En segundo lugar hemos comparado dos procesos que modelaban funciones Ackley que, a pesar de tener parámetros distintos, tenían forma parecida. En ambos procesos hemos fijado los parámetros

$b = 0,2, c = 2\pi$ , en uno de ellos hemos escogido  $a = 70$  y en el otro  $a = 100$ .

El experimento ha resultado en una correlación 0.98, muy cercana a 1, y una distancia media relativa de 0.01. En total una distancia de 0.01, lo que confirma que los dos procesos son muy parecidos.

## 6.2. Experimentos del algoritmo de reducción de objetivos

A continuación, describiremos los experimentos realizados con optimización Bayesiana de muchos objetivos utilizando nuestro algoritmo de reducción de objetivos. Empezaremos con un experimento “de juguete” en el que la reducción de objetivos es obvia, y después trataremos dos experimentos en los que partiremos de un problema con cuatro objetivos. En los cuatro experimentos, veremos cómo se comporta el algoritmo según los hiperparámetros de reducción escogidos por el usuario. Para medir la calidad de la solución sugerida por cada experimento utilizaremos el indicador del hipervolumen.

En todos los experimentos, la configuración de la función de distancia ha sido la misma que en los experimentos de la sección anterior, y la función de adquisición empleada es PESH (predictive entropy search for multi-objective Bayesian optimization [29]).

### 6.2.1. Primer experimento

Para este experimento hemos utilizado la función de *Branin*, dada por la expresión

$$f_8(x, y) = (y - 5,1x^2 / (4\pi^2) + 5x/\pi - 6)^2 + 10(1 - 1/8\pi) \cos x + 10.$$

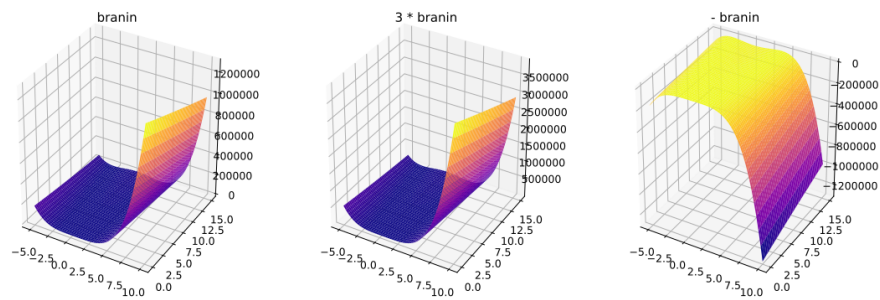


Figura 6.5: Objetivos basados en funciones *Branin*

Hemos optimizado los tres objetivos ilustrados en la Figura 6.5: el primero una función de *Branin*, el segundo una función de *Branin* triplicada y el tercero una función de *Branin* multiplicada por -1. Claramente, la función de *Branin* y su triplicada se comportan de manera muy similar. La optimización Bayesiana ha sido ejecutada con un máximo de 25 iteraciones y con distintos valores para sus

hiperparámetros. El parámetro  $\delta$  que permitía aplicar la reducción a partir de la  $\delta$ -ésima iteración de la optimización ha sido probado para los valores 10, 15 y 20. El parámetro  $\varepsilon$  que indicaba que dos objetivos eran considerados idénticos si su distancia era menor que  $\varepsilon$  ha sido probado para los valores 0.05, 0.1 y 0.2. Por tanto, este experimento ha constado de 9 pruebas.

Como en este caso el primer y segundo objetivo están relacionados por una función lineal, el algoritmo de optimización Bayesiana ha eliminado uno de estos dos objetivos en cada una de las 9 ejecuciones en la primera iteración en la que podía hacerlo (la décima, decimoquinta o vigésima según el valor de  $\delta$ ). Discutamos ahora la calidad de la solución obtenida después de la reducción basándonos en el hipervolumen. El hipervolumen de la solución de esta optimización era 1143516861, y el hipervolumen de las soluciones obtenidas con la reducción se muestran en la Tabla 6.1.

| Hipervolumen         | $\delta = 10$ | $\delta = 15$ | $\delta = 20$ |
|----------------------|---------------|---------------|---------------|
| $\varepsilon = 0,05$ | 1142795726    | 1142795726    | 1143013872    |
| $\varepsilon = 0,10$ | 1142682063    | 1142795726    | 1143013872    |
| $\varepsilon = 0,20$ | 1142682063    | 1142795726    | 1143013872    |

**Tabla 6.1:** Hipervolumen de las soluciones de la optimización del primer experimento

En la tabla puede observarse que en todos los casos la recomendación propuesta por el optimización tiene un hipervolumen cercano al de la solución (el error en el peor caso es de 0.07 %). Además, se observa que la solución recomendada es mejor cuánto más tarde se activa la reducción de objetivos (cuanto mayor era  $\delta$ ). Esto tiene sentido ya que cuantas más iteraciones se realizan sin la reducción, mejor es la predicción que los GPs hacen de los objetivos y por tanto más acertada es la distancia entre los objetivos.

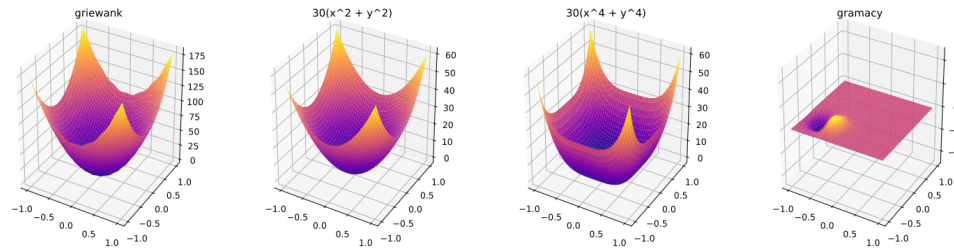
Otra observación de este experimento que no puede verse en la tabla pero sí en los ficheros del experimento es que, aunque eventualmente la solución propuesta era de buena calidad independientemente del valor de  $\varepsilon$ , la calidad de los puntos explorados por el algoritmo (provenientes de optimizar la función de adquisición) en las primeras iteraciones después de la reducción de un objetivo bajaban más cuanto mayor era  $\varepsilon$ . La razón de esto es que a mayor  $\varepsilon$ , más “laxa” es nuestra reducción: menos parecidos deben ser dos GPs para ser considerados iguales y por tanto, la exploración de nuevos puntos se ve perjudicada en las primeras iteraciones después de la eliminación de un objetivo. No obstante, después de estas primeras iteraciones los puntos evaluados son igual de buenos que en los demás casos.

### 6.2.2. Segundo experimento: funciones cuenco

En este experimento nos enfrentamos a un problema de optimización de cuatro objetivos. El primero es una función de *Griewank*, dada por la expresión  $1 - \cos(x) \cos(y/\sqrt{2}) + (x^2 + y^2)/4000$ , el segundo es



un paraboloide de grado dos ( $x^2 + y^2$ ), el tercero un paraboloide de grado cuatro ( $x^4 + y^4$ ) y el cuarto es una función de *Gramacy*  $xe^{-x^2-y^2}$ . Los cuatro objetivos se ilustran en la Figura 6.6. La función de *Griewank* tiene forma de paraboloide vista desde lejos, pero de cerca su superficie está llena de pequeñas montañas.



**Figura 6.6:** Objetivos basados en paraboloides y otras funciones

En este caso, hemos fijado el número máximo de iteraciones de la optimización a 30 y hemos realizado el experimento con dos combinaciones de hiperparámetros: la primera con una reducción más laxa ( $\delta = 15, \varepsilon = 0,10$ ) y la segunda más estricta ( $\delta = 20, \varepsilon = 0,05$ ). El hipervolumen de la solución del problema en este caso era 1000398078127.

En la primera ejecución ( $\delta = 15, \varepsilon = 0,10$ ) el GP correspondiente al paraboloide de grado dos fue eliminado en la décimo sexta iteración por ser muy similar al de *Griewank*, y el otro paraboloide fue eliminado en la vigésimo primera por considerarse también igual al de *Griewank*. En este caso, la solución recomendada tiene hipervolumen 1000397940485 (error  $1,37 \times 10^{-7}$ ) y se explora entre la primera y la segunda reducción. En la imagen se ve claramente por qué la función de *Griewank* y el paraboloide de grado dos se consideran iguales. Aunque el otro paraboloide no sea tan parecido a estas dos funciones, tomar un valor alto de  $\varepsilon$  (0.10) es la razón de que este objetivo también se elimine. La función que indiscutiblemente no se parece a las demás es la función de *Gramacy*, y por ello este objetivo no es reducido.

En la segunda ejecución ( $\delta = 20, \varepsilon = 0,05$ ) el GP correspondiente al paraboloide de grado dos se elimina tras la vigésima iteración (la primera en la que pueden reducirse objetivos). Ningún otro objetivo se elimina antes de la trigésima y última iteración. La solución recomendada en este caso tiene un hipervolumen 1000397851175 (error  $2,26 \times 10^{-7}$ ) y se explora después de la primera reducción. En este caso, haber retrasado la reducción de objetivos y reducir el valor de  $\varepsilon$  a 0.05 hace que sólo se reduzca un objetivo y no se considere al paraboloide de grado cuatro lo suficientemente similar a los otros paraboloides.

### 6.2.3. Tercer experimento: funciones *Michalewicz*

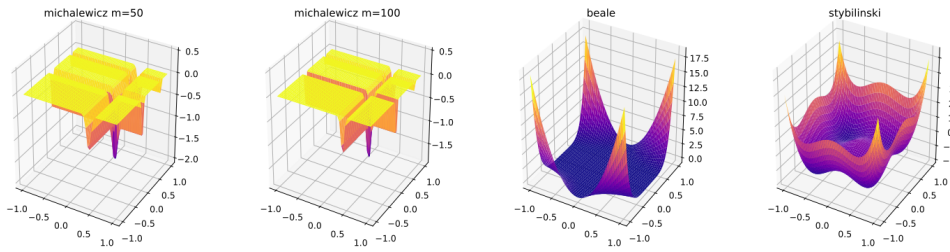
Para el tercer experimento hemos optimizado las cuatro funciones ilustradas en la Figura 6.7: dos funciones de *Michalewicz*, cuya expresión es

$$f_9(x, y) = -\sin x \sin^{2m}(x^2/\pi) \left( \sin y \sin^{2m}(2y^2/\pi) \right)$$

variando  $m$ , una función de *Beale*

$$f_{10}(x, y) = (1,5 - x - xy)^2 + (2,25 - x - xy^2)^2 + (2,625 - x - xy * 3)^2$$

y la función de *Styblinski-Tang* definida en la Sección 6.1.



**Figura 6.7:** Objetivos basados en funciones *Michalewicz*

En este experimento hemos fijado el el número máximo de iteraciones a 30 y hemos realizado dos pruebas. La primera ha sido realizada con dos funciones de *Michalewicz*, una con  $m = 75$  y otra con  $m = 100$ , y con los hiperparámetros  $\delta = 10, \varepsilon = 20$ . Para este caso, las dos funciones de *Michalewicz* eran muy similares y una de ellas fue eliminada en la vigésimo primera iteración. La solución del problema tenía hipervolumen 1004234499054 y la recomendación del algoritmo fue un punto con hipervolumen 1003947732936 (error del 0.02 %). En la segunda prueba hemos utilizado una función de *Michalewicz* con  $m = 50$  y otra con  $m = 100$ . Como estas funciones son más distintas entre sí que las de la primera prueba, hemos reducido  $\varepsilon$  a 0.05. Con esto, hemos exigido que el algoritmo esté “más seguro” de la similitud de dos objetivos antes de eliminar uno de ellos. En este caso, una de las dos funciones *Michalewicz* ha sido borrada en la vigésimo séptima iteración. El hipervolumen de la solución del problema era 1004348965464 y el de la recomendación del algoritmo 10043488477896 (error  $4,85 \times 10^{-7}$ ). Ésta es la única prueba donde la solución se explora antes de la reducción de objetivo.

#### 6.2.4. Cuarto experimento: ajuste de hiperparámetros red neuronal

En el último experimento, aplicamos nuestro algoritmo de reducción de objetivos al problema del ajuste de hiperparámetros de una red neuronal profunda que clasifica imágenes en dígitos entre el 0 y el 9. Para ello hemos utilizado Digit, un pequeño conjunto de datos de 1797 imágenes que contiene dígitos escritos a mano. El espacio de entrada del experimento está dado por los hiperparámetros número de capas ocultas y número de neuronas en cada capa oculta, y los objetivos de la optimización Bayesiana son minimizar el tiempo de ejecución, el tamaño de la red y el error de la predicción.

Debido a las restricciones  $\mathbf{R}_1$  y  $\mathbf{R}_2$ ., hemos hecho algunas simplificaciones sobre el experimento. Por ejemplo, exigimos que todas las capas tengan el mismo número de neuronas. Para poder evaluar redes más o menos grandes no entrenamos la red sino que inicializamos los pesos de manera aleatoria. Esto provoca que el error siempre sea muy alto, pero no nos preocupa ya que a nosotros nos interesa comprobar la relación que existe entre el tamaño de una red y el tiempo de evaluación. Además, no hemos podido calcular el hipervolumen aproximado de la solución ya que su cómputo requería demasiadas evaluaciones de los objetivos. Si no fuera por las restricciones de tiempo y de capacidad computacional, podríamos ejecutar la optimización teniendo en cuenta más hiperparámetros, considerando un conjunto de datos mayor, entrenando la red (pudiendo así minimizar el error de predicción) y conociendo el hipervolumen de la solución.

En este experimento hemos realizado dos pruebas. En ambas, el número de capas ocultas varía entre 3 y 100, el número de neuronas en cada capa entre 3 y 300 y el número máximo de iteraciones de la optimización es 40.

En la primera prueba fijamos los hiperparámetros  $\delta = 15$  y  $\varepsilon = 0,1$ . En este caso, en la décimo sexta iteración el algoritmo detectó que la distancia entre los objetivos tamaño y tiempo era 0.0006 y la distancia entre tiempo y error, y entre tamaño y error era 0.75. Por tanto, en esa iteración el algoritmo eliminó el objetivo del tamaño.

En la segunda prueba escogimos  $\delta = 20$ ,  $\varepsilon = 0,05$  para ver si, realizando la reducción unas iteraciones más tarde (cuando los GPs dieran una mejor predicción de los objetivos) y siendo más estrictos con  $\varepsilon$ , un objetivo también se eliminaría. El resultado aquí fue que en la vigésimo primera iteración, la distancia entre los objetivos tamaño y tiempo era 0.0004 (menor que en la prueba anterior) y las otras dos distancias eran 0.7502. Por tanto, también en este caso un objetivo fue eliminado.



## CONCLUSIONES Y TRABAJO FUTURO

---

Para concluir esta tesis, haremos un análisis del grado de consecución de los objetivos, veremos cómo hemos contrastado las hipótesis definidas en el Capítulo 3 y plantearemos algunas posibles líneas de trabajo futuro.

Comenzaremos detallando qué objetivos se han alcanzado en el trabajo y cómo se ha hecho:

- O<sub>1</sub>.** Implementación en Python de una distancia estadística para comparar procesos gaussianos que comparten el mismo espacio de entrada. Objetivo alcanzado con la función `distance` del fichero `spearmint/utils/distances.py`.
- O<sub>2</sub>.** Demostrar efectividad de la distancia desarrollada con funciones de referencia. Objetivo satisfecho con los resultados de los cuatro experimentos de la Sección 6.1.
- O<sub>3</sub>.** Integrar la distancia en la herramienta de optimización bayesiana Spearmint. Este objetivo se satisfizo al implementar la función `distance` del fichero `spearmint/utils/distances.py`.
- O<sub>4</sub>.** Implementar un algoritmo de reducción de objetivos en Spearmint. El objetivo se ha alcanzado en la función `suggest_redundant_process` del fichero `spearmint/choosers/default_chooser.py`.
- O<sub>5</sub>.** Reducir el tiempo de ejecución de Spearmint cuando dos objetivos sean considerados similares por la herramienta. Debido a que la optimización Bayesiana se utiliza para optimizar objetivos que son muy costosos de evaluar y dadas las restricciones **R<sub>1</sub>**. y **R<sub>2</sub>**., no hemos podido realizar los experimentos en escenarios reales en los que se reflejara la mejora de tiempos al eliminar un objetivo. Sin embargo, en los experimentos de la Sección 6.2 sí comprobamos que se eliminan objetivos. Siguiendo la suposición **S<sub>4</sub>**., concluimos que si tuviéramos los medios para ejecutar objetivos que fueran realmente costosos de evaluar la eliminación de un sólo objetivo ya incurriría en una reducción del tiempo.
- O<sub>6</sub>.** No reducir drásticamente el rendimiento de la optimización al reducir el número de objetivos. Los experimentos de la Sección 6.2 que analizan el hipervolumen de la solución del problema y el de la solución tras la reducción demuestran que este objetivo ha sido alcanzado bajo la suposición **S<sub>7</sub>**. de que el hipervolumen es un buen indicador de la calidad de las soluciones.

Como hemos visto, los objetivos han sido contrastados mediante la implementación de la medida de similitud, del algoritmo de reducción de objetivos y de la realización de experimentos. Analicemos ahora cómo hemos contrastado las hipótesis planteadas en el Capítulo 3.

**H<sub>1</sub>.** Podemos desarrollar una herramienta capaz de medir la similitud entre dos procesos gaussianos.

La hipótesis fue contrastada al implementar la función `distance.py` y ha resultado ser cierta.

**H<sub>2</sub>.** Podemos reducir el coste de la BO de muchos objetivos en Spearmint. Demostramos que la hipótesis es válida mediante el segundo y tercer experimento de la Sección 6.2 en los que el coste del algoritmo de optimización se ve reducido cada vez que se elimina un objetivo.

**H<sub>3</sub>.** El rendimiento de Spearmint no bajará significativamente al reducir el coste de la BO de muchos objetivos. De nuevo, esta hipótesis es válida y se contrasta al comparar el hipervolumen de la solución del problema con el hipervolumen de la solución sugerida por la optimización Bayesiana en los experimentos de la Sección 6.2.

El trabajo desarrollado nos abre diversas líneas de trabajo futuro, de las cuales comentamos a continuación las que nos parecen más interesantes.

Aunque nuestra distancia tiene en cuenta la varianza predictiva proporcionada por los procesos Gaussianos, en la práctica apenas es utilizada porque no detectamos que la distancia entre varianzas fuera útil para eliminar GPs redundantes. Sería interesante aprovechar la información que nos proporcionan estas varianzas predictivas para mejorar nuestra medida de similitud. Una manera de hacer esto sería penalizar en el cálculo de la distancia y correlación entre las medias predictivas aquellos puntos o intervalos donde la varianza, y por tanto la incertidumbre, fuera alta. Así, un intervalo en el que dos medias predictivas estuvieran muy correlacionadas y en el que la incertidumbre de la predicción fuera baja aportaría más información a la medida de similitud que si la incertidumbre fuera alta.

Otra mejora de la distancia consistiría en dar más importancia a la distancia y correlación entre medias predictivas calculadas en áreas en las que se encuentren potenciales soluciones de la optimización. Por ejemplo, podríamos hacer que, dada la distancia entre las medias predictivas de dos GPs evaluadas en un punto, su aportación a la distancia total fuera proporcional a la distancia de ese punto a una solución potencial o al hipervolumen del punto.

Por último, cuando la medida de similitud considera que dos GPs son redundantes, nuestro algoritmo de optimización Bayesiana de muchos objetivos elimina aleatoriamente cualquiera de los dos. Idealmente querríamos eliminar el más costoso. Una manera de implementar esto sería que el usuario detallara a través del fichero de configuración el coste de cada objetivo para así eliminar el objetivo más costoso. Si esta información no pudiera ser proporcionada, la propia herramienta podría guardar información acerca del tiempo de ejecución de cada objetivo y eliminar el objetivo que tomara más tiempo evaluar.

# BIBLIOGRAFÍA

---

- [1] M. ABDOLSHAH, A. SHILTON, S. RANA, S. GUPTA, AND S. VENKATESH, *Multi-objective bayesian optimisation with preferences over objectives*, 2019.
- [2] M. ASAFUDDOULA, T. RAY, AND R. SARKER, *A decomposition-based evolutionary algorithm for many objective optimization*, IEEE Transactions on Evolutionary Computation, 19 (2014), pp. 445–460.
- [3] S. BANDYOPADHYAY AND A. MUKHERJEE, *An algorithm for many-objective optimization with reduced objective computations: A study in differential evolution*, IEEE Transactions on Evolutionary Computation, 19 (2014), pp. 400–413.
- [4] P. J. BENTLEY AND J. P. WAKEFIELD, *Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms*, in *Soft Computing in Engineering Design and Manufacturing*, P. K. Chawdhry, R. Roy, and R. K. Pant, eds., London, 1998, Springer London, pp. 231–240.
- [5] J. BERGSTRA, R. BARDENET, Y. BENGIO, AND B. KÉGL, *Algorithms for hyper-parameter optimization*, in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, eds., vol. 24, Curran Associates, Inc., 2011.
- [6] J. R. BERRENDERO, B. BUENO-LARRAZ, AND A. CUEVAS, *On mahalanobis distance in functional settings*, Journal of Machine Learning Research, 21 (2020), pp. 1–33.
- [7] M. K. BESHARATI GIVI, P. ASADI, S. BAG, D. YADUWANSI, S. PAL, A. HEIDARZADEH, S. MUDANI, K. KAZEMI-CHOOBI, H. HANIFIAN, D. BRAGA, A. SILVA-MAGALHÃES, P. MOREIRA, V. INFANTE, C. VIDAL, E. AKINLABI, S. AKINLABI, H. ARORA, S. MUKHERJEE, H. GREWAL, AND P. ZOLGHADR, *Advances in Friction-Stir Welding and Processing*, 08 2014.
- [8] M. BINOIS, V. PICHENY, P. TAILLANDIER, AND A. HABBAL, *The kalai-smorodinsky solution for many-objective bayesian optimization*, Journal of Machine Learning Research, 21 (2020), pp. 1–42.
- [9] E. BROCHU, T. BROCHU, AND N. DE FREITAS, *A bayesian interactive optimization approach to procedural animation design*, in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010, pp. 103–112.
- [10] E. BROCHU, V. M. CORA, AND N. DE FREITAS, *A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning*, 2010.
- [11] E. CHOI AND C. LEE, *Feature extraction based on the bhattacharyya distance*, Pattern Recognition, 36 (2003), pp. 1703–1709.
- [12] A. CLIM, R. D. ZOTA, AND G. TINICĂ, *The kullback-leibler divergence used in machine learning algorithms for health care applications and hypertension prediction: A literature review*, Procedia Computer Science, 141 (2018), pp. 448–453. The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018) / The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018) / Affiliated Workshops.
- [13] L. CORNEJO-BUENO, E. GARRIDO-MERCHN, D. HERNÁNDEZ-LOBATO, AND S. SALCEDO-SANZ, *Bayesian optimization of a hybrid system for robust ocean wave features prediction*, 275 (2018), p. 818–828.
- [14] Z. CUI, J. ZHANG, Y. WANG, Y. CAO, X. CAI, W. ZHANG, AND J. CHEN, *A pigeon-inspired optimization algorithm for many-objective optimization problems*, Science China Information Sciences, 62 (2019).
- [15] J. DEVLIN, M.-W. CHANG, K. LEE, AND K. TOUTANOVA, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018.
- [16] I. DEWANCKER, M. MCCOURT, S. CLARK, P. HAYES, A. JOHNSON, AND G. KE, *A stratified analysis of bayesian optimization methods*, 2016.
- [17] R. M. DUDLEY, *Lp Spaces; Introduction to Functional Analysis*, Cambridge Studies in Advanced Mathematics, Cambridge University Press, 2 ed., 2002, p. 152–187.

- [18] P. J. FLEMING, R. C. PURSHOUSE, AND R. J. LYGOE, *Many-objective optimization: An engineering design perspective*, in *Evolutionary Multi-Criterion Optimization*, C. A. Coello Coello, A. Hernández Aguirre, and E. Zitzler, eds., Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 14–32.
- [19] P. I. FRAZIER, *A tutorial on bayesian optimization*, 2018.
- [20] K. FUKUNAGA, *In side front cover*, in *Introduction to Statistical Pattern Recognition (Second Edition)*, Academic Press, Boston, second edition ed., 1990, p. ii.
- [21] J. R. GARDNER, M. J. KUSNER, Z. E. XU, K. Q. WEINBERGER, AND J. P. CUNNINGHAM, *Bayesian optimization with inequality constraints.*, in *ICML*, vol. 2014, 2014, pp. 937–945.
- [22] R. GARNETT, M. A. OSBORNE, AND S. J. ROBERTS, *Bayesian optimization for sensor set selection*, IPSN '10, New York, NY, USA, 2010, Association for Computing Machinery.
- [23] E. C. GARRIDO-MERCHÁN AND A. ALBARCA-MOLINA, *Suggesting cooking recipes through simulation and bayesian optimization*, (2018).
- [24] E. C. GARRIDO-MERCHÁN AND D. HERNÁNDEZ-LOBATO, *Predictive entropy search for multi-objective bayesian optimization with constraints*, *Neurocomputing*, 361 (2019), pp. 50–68.
- [25] M. GARZA-FABRE, G. T. PULIDO, AND C. A. C. COELLO, *Ranking methods for many-objective optimization*, in *MICA 2009: Advances in Artificial Intelligence*, A. H. Aguirre, R. M. Borja, and C. A. R. Garcá, eds., Berlin, Heidelberg, 2009, Springer Berlin Heidelberg, pp. 633–645.
- [26] F. GOUDAIL, P. RÉFRÉGIER, AND G. DELYON, *Bhattacharyya distance as a contrast parameter for statistical processing of noisy optical images*, *J. Opt. Soc. Am. A*, 21 (2004), pp. 1231–1240.
- [27] J. GUO, F. LIU, AND Z. ZHU, *Estimate the call duration distribution parameters in gsm system based on k-l divergence method*, in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, 2007, pp. 2988–2991.
- [28] P. HENNIG AND C. J. SCHULER, *Entropy search for information-efficient global optimization*, 2011.
- [29] D. HERNÁNDEZ-LOBATO, J. M. HERNÁNDEZ-LOBATO, A. SHAH, AND R. P. ADAMS, *Predictive entropy search for multi-objective bayesian optimization*, 2015.
- [30] ———, *Predictive entropy search for multi-objective bayesian optimization*, 2015.
- [31] F. HUTTER, H. H. HOOS, AND K. LEYTON-BROWN, *Sequential model-based optimization for general algorithm configuration*, in *International conference on learning and intelligent optimization*, Springer, 2011, pp. 507–523.
- [32] H. ISHIBUCHI, N. TSUKAMOTO, AND Y. NOJIMA, *Evolutionary many-objective optimization: A short review*, in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 2419–2426.
- [33] D. JONES, *A taxonomy of global optimization methods based on response surfaces*, *J. of Global Optimization*, 21 (2001), pp. 345–383.
- [34] D. JONES, M. SCHONLAU, AND W. WELCH, *Efficient global optimization of expensive black-box functions*, *Journal of Global Optimization*, 13 (1998), pp. 455–492.
- [35] N. KHAN, D. E. GOLDBERG, AND M. PELIKAN, *Multi-objective bayesian optimization algorithm*, in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, Citeseer, 2002, pp. 684–684.
- [36] V. KHARE, P. YAO, AND K. DEB, *Performance scaling of multi-objective evolutionary algorithms*, (2003).
- [37] S. KULLBACK AND R. A. LEIBLER, *On information and sufficiency*, *The Annals of Mathematical Statistics*, 22 (1951), pp. 79–86.
- [38] H. J. KUSHNER, *A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise*, *Journal of Basic Engineering*, 86 (1964), pp. 97–106.
- [39] Y. LECUN, Y. BENGIO, AND G. HINTON, *Deep learning*, *nature*, 521 (2015), pp. 436–444.
- [40] J. LEE, Y. BAHRI, R. NOVAK, S. S. SCHOENHOLZ, J. PENNINGTON, AND J. SOHL-DICKSTEIN, *Deep neural networks as gaussian processes*, 2018.
- [41] D. J. LIZOTTE, T. WANG, M. H. BOWLING, AND D. SCHUURMANS, *Automatic gait optimization with gaussian process regression.*, in *IJCAI*, vol. 7, 2007, pp. 944–949.
- [42] D. J. MACKAY, *Introduction to gaussian processes*, *NATO ASI Series F Computer and Systems Sciences*, 168 (1998), pp. 133–166.



- [43] P. C. MAHALANOBIS, *On the generalized distance in statistics*, Proceedings of the National Institute of Sciences (Calcutta), 2 (1936), pp. 49–55.
- [44] R. MARCHANT AND F. RAMOS, *Bayesian optimisation for intelligent environmental monitoring*, in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 2242–2249.
- [45] M. MARKATOU, Y. CHEN, G. AFENDRAS, AND B. G. LINDSAY, *Statistical distances and their role in robustness*, (2016).
- [46] M. MENÉNDEZ, J. PARDO, L. PARDO, AND M. PARDO, *The jensen-shannon divergence*, Journal of the Franklin Institute, 334 (1997), pp. 307–318.
- [47] M. W. MKAOUER, M. KESSENTINI, S. BECHIKH, K. DEB, AND M. Ó CINNÉIDE, *High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using nsga-iii*, in Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 1263–1270.
- [48] J. MOCKUS, *Application of bayesian approach to numerical methods of global and stochastic optimization*, Journal of Global Optimization, 4 (1994), pp. 347–365.
- [49] K. NARUKAWA AND T. RODEMANN, *Examining the performance of evolutionary many-objective optimization algorithms on a real-world application*, in 2012 Sixth International Conference on Genetic and Evolutionary Computing, 2012, pp. 316–319.
- [50] D. M. NEGOSCU, P. I. FRAZIER, AND W. B. POWELL, *The knowledge-gradient algorithm for sequencing experiments in drug discovery*, 23 (2011).
- [51] S. Otake, T. YOSHIKAWA, AND T. FURUHASHI, *Basic study on aggregation of objective functions in many-objective optimization problems*, in 2010 World Automation Congress, IEEE, 2010, pp. 1–6.
- [52] A. PAPADOPOULOS, *Metric Spaces, Convexity and Nonpositive Curvature (Second edition)*, vol. 6 of IRMA Lectures in Mathematics and Theoretical Physics Vol. 6, European Mathematical Society, 2014.
- [53] C. E. RASMUSSEN AND C. K. I. WILLIAMS, *Gaussian Processes for Machine Learning*, MIT Press, 2006.
- [54] B. REAGEN, J. M. HERNÁNDEZ-LOBATO, R. ADOLF, M. GELBART, P. WHATMOUGH, G.-Y. WEI, AND D. BROOKS, *A case for efficient accelerator design space exploration via bayesian optimization*, in 2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2017, pp. 1–6.
- [55] A. RÉNYI ET AL., *On measures of entropy and information*, in Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, The Regents of the University of California, 1961.
- [56] C. ROSENBERG, M. HEBERT, AND S. THRUN, *Color constancy using kl-divergence*, in Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, vol. 1, 2001, pp. 239–246 vol.1.
- [57] S. M. ROSS, J. J. KELLY, R. J. SULLIVAN, W. J. PERRY, D. MERCER, R. M. DAVIS, T. D. WASHBURN, E. V. SAGER, J. B. BOYCE, AND V. L. BRISTOW, *Stochastic processes*, vol. 2, Wiley New York, 1996.
- [58] M. SASENA, *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations.*, 2002.
- [59] J. SNOEK, H. LAROCHELLE, AND R. P. ADAMS, *Practical bayesian optimization of machine learning algorithms*, 2012.
- [60] J. SNOEK, O. RIPPEL, K. SWERSKY, R. KIROS, N. SATISH, N. SUNDARAM, M. PATWARY, M. PRABHAT, AND R. ADAMS, *Scalable bayesian optimization using deep neural networks*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015, PMLR, pp. 2171–2180.
- [61] B. SOLNIK, D. GOLOVIN, G. KOCHANSKI, J. E. KARRO, S. MOITRA, AND D. SCULLEY, *Bayesian optimization for a better dessert*, in Proceedings of the 2017 NIPS Workshop on Bayesian Optimization, December 9, 2017, Long Beach, USA, 2017. The workshop is BayesOpt 2017 NIPS Workshop on Bayesian Optimization December 9, 2017, Long Beach, USA.
- [62] N. SRINIVAS, A. KRAUSE, S. M. KAKADE, AND M. SEEGER, *Gaussian process optimization in the bandit setting: No regret and experimental design*, (2009).
- [63] S. STRELTZOV AND P. VAKILI, *A non-myopic utility function for statistical global optimization algorithms*, J. of Global Optimization, 14 (1999), p. 283–298.
- [64] E. STRUBELL, A. GANESH, AND A. MCCALLUM, *Energy and policy considerations for deep learning in nlp*, 2019.

- [65] A. TORN AND A. ZILINSKAS, *Global Optimization*, Springer-Verlag, Berlin, Heidelberg, 1989.
- [66] E. TRIANTAPHYLLOU, *Multi-criteria decision making methods : a comparative study*, Applied optimization 44, Kluwer Academic Publishers, Dordrecht, 2000.
- [67] A. VASWANI, N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, L. KAISER, AND I. POLOSUKHIN, *Attention is all you need*, 2017.
- [68] M. VENTURINI AND G. ALEJANDRO, *Statistical distances and probability metrics for multivariate data, ensembles and probability distributions*, 2015.
- [69] M. VOORNEVELD, *Characterization of pareto dominance*, Operations Research Letters, 31 (2003), pp. 7–11.
- [70] U. K. WICKRAMASINGHE, R. CARRESE, AND X. LI, *Designing airfoils using a reference point based evolutionary many-objective particle swarm optimization algorithm*, in IEEE congress on evolutionary computation, IEEE, 2010, pp. 1–8.
- [71] J. WOLBERG, *Data analysis using the method of least squares: extracting the most information from experiments*, Springer Science & Business Media, 2006.
- [72] K. YANG, M. EMMERICH, A. DEUTZ, AND T. BÄCK, *Multi-objective bayesian global optimization using expected hypervolume improvement gradient*, Swarm and evolutionary computation, 44 (2019), pp. 945–956.
- [73] X.-S. YANG, *Nature-Inspired Optimization Algorithms*, Morgan Kaufmann, Amsterdam, 2014.
- [74] C. YOU, K. A. LEE, AND H. LI, *An svm kernel with gmm-supervector based on the bhattacharyya distance for speaker recognition*, Signal Processing Letters, IEEE, 16 (2009), pp. 49 – 52.
- [75] E. ZITZLER, J. KNOWLES, AND L. THIELE, *Quality assessment of pareto set approximations*, Multiobjective optimization, (2008), pp. 373–404.

# APÉNDICES



# PAQUETES INSTALADOS

---

## A.1. Paquetes instalados con Python 2.7

El lector puede encontrar en la Figura A.1 un listado con todos los paquetes instalados en el entorno de Python con la versión 2.7 del lenguaje.

## A.2. Paquetes instalados con Python 3.8

En la Figura A.2 y la Figura A.3 se listan los paquetes utilizados, así como sus versiones, en el entorno con Python 3.8.

|                                   |              |                |             |
|-----------------------------------|--------------|----------------|-------------|
| libgcc_mutex                      | 0.1          | main           |             |
| absl-py                           | 0.13.0       | <pip>          |             |
| astor                             | 0.8.1        | <pip>          |             |
| backports                         | 1.0          | pyhd3eb1b0_2   |             |
| backports.functiontools_lru_cache | 1.0.1        | pyhd3eb1b0_0   |             |
| backports.weakref                 | 1.0.post1    | <pip>          |             |
| backports_abc                     | 0.5          | py_1           |             |
| binutils_impl_linux-64            | 2.33.1       | he0710b0_7     |             |
| binutils_linux-64                 | 2.33.1       | h9595d00_15    |             |
| blas                              | 1.0          | mk1            |             |
| boost                             | 1.03.0       | py27_5         | conda-forge |
| boost-cpp                         | 1.03.0       | 2              | conda-forge |
| bzip2                             | 1.0.8        | h7b6447c_0     |             |
| ca-certificates                   | 2020.12.8    | h00a4308_0     |             |
| cachetools                        | 3.1.1        | <pip>          |             |
| certifi                           | 2020.6.20    | pyhd3eb1b0_3   |             |
| cycler                            | 0.10.0       | py27hc7354d3_0 |             |
| dbus                              | 1.13.18      | hbzf20db_0     |             |
| enum34                            | 1.1.10       | <pip>          |             |
| expat                             | 2.2.10       | he0710b0_2     |             |
| fontconfig                        | 2.13.0       | h9428a91_0     |             |
| freetype                          | 2.10.4       | h5ab3b9f_0     |             |
| funcsigs                          | 1.0.2        | <pip>          |             |
| functools32                       | 3.2.3.2      | py27_1         |             |
| futures                           | 3.3.0        | py27_0         |             |
| gast                              | 0.2.2        | <pip>          |             |
| gcc_impl_linux-64                 | 7.3.0        | habb00fd_1     |             |
| gcc_linux-64                      | 7.3.0        | h553295d_15    |             |
| glib                              | 2.60.1       | h92f7085_0     |             |
| google-auth                       | 1.31.0       | <pip>          |             |
| google-auth-oauthlib              | 0.4.1        | <pip>          |             |
| google-pasta                      | 0.2.0        | <pip>          |             |
| grpcio                            | 1.38.0       | <pip>          |             |
| gst-plugins-base                  | 1.14.0       | h8213a91_2     |             |
| gststreamer                       | 1.14.0       | h28cd5cc_2     |             |
| gxx_impl_linux-64                 | 7.3.0        | hdf03c00_1     |             |
| gxx_linux-64                      | 7.3.0        | h553295d_15    |             |
| h5py                              | 2.10.0       | <pip>          |             |
| icu                               | 58.2         | he0710b0_3     |             |
| intel-openmp                      | 2020.2       | 254            |             |
| jpeg                              | 9b           | h024ee3a_2     |             |
| Keras                             | 2.3.1        | <pip>          |             |
| Keras-Applications                | 1.0.8        | <pip>          |             |
| Keras-Preprocessing               | 1.1.2        | <pip>          |             |
| kiwisolver                        | 1.1.0        | py27he0710b0_0 |             |
| ld_impl_linux-64                  | 2.33.1       | h53a041e_7     |             |
| libedit                           | 3.1.20191231 | h14c3975_1     |             |
| libffi                            | 3.3          | he0710b0_2     |             |
| libgcc                            | 5.2.0        | 0              | nsarahan    |
| libgcc-ng                         | 9.1.0        | hdf03c00_0     |             |
| libgfortran-ng                    | 7.3.0        | hdf03c00_0     |             |
| libpng                            | 1.6.37       | hbc83047_0     |             |
| libstdcxx-ng                      | 9.1.0        | hdf03c00_0     |             |
| libuuid                           | 1.0.3        | h1bed415_2     |             |
| libxcb                            | 1.14         | h7b6447c_0     |             |
| libxml2                           | 2.9.10       | hb55308b_3     |             |
| matplotlib                        | 2.2.3        | py27hb09df0a_0 |             |
| mk1                               | 2020.2       | 250            |             |
| mk1-service                       | 2.3.0        | py27he904b0f_0 |             |
| mk1_fft                           | 1.0.15       | py27ha843d7b_0 |             |
| mk1_random                        | 1.1.0        | py27hd0b4f25_0 |             |
| nock                              | 3.0.5        | <pip>          |             |
| ncurses                           | 6.2          | he0710b0_1     |             |
| numpy                             | 1.16.0       | py27hbc911f0_0 |             |
| numpy-base                        | 1.16.0       | py27hde5b4d0_0 |             |
| oauthlib                          | 3.1.0        | <pip>          |             |
| openssl                           | 1.1.1i       | h27cfd23_0     |             |
| opt-einsum                        | 2.3.2        | <pip>          |             |
| pcrc                              | 8.44         | he0710b0_0     |             |
| pip                               | 19.3.1       | py27_0         |             |
| protobuf                          | 3.17.3       | <pip>          |             |
| pyasn1                            | 0.4.8        | <pip>          |             |
| pyasn1-modules                    | 0.2.8        | <pip>          |             |
| pygmo                             | 1.1.7        | py27_2         | conda-forge |
| pymongo                           | 3.9.0        | py27he0710b0_0 |             |
| pyarsing                          | 2.4.7        | py_0           |             |
| pyqt                              | 5.9.2        | py27h05f1152_2 |             |
| python                            | 2.7.18       | h15b4118_1     |             |
| python-dateutil                   | 2.8.1        | py_0           |             |
| pytz                              | 2020.5       | pyhd3eb1b0_0   |             |
| PyYAML                            | 5.4.1        | <pip>          |             |
| qt                                | 5.9.7        | h5807ecd_1     |             |
| readline                          | 8.0          | h7b6447c_0     |             |
| requests                          | 2.25.1       | <pip>          |             |
| requests-oauthlib                 | 1.3.0        | <pip>          |             |
| rsa                               | 4.5          | <pip>          |             |
| scikit-learn                      | 0.20.4       | <pip>          |             |
| scipy                             | 1.2.2        | <pip>          |             |
| scipy                             | 1.2.1        | py27h7c811a0_0 |             |
| setuptools                        | 44.0.0       | py27_0         |             |
| singledispatch                    | 3.4.0.3      | py_1001        |             |
| sip                               | 4.19.8       | py27hf484d3e_0 |             |
| six                               | 1.15.0       | py_0           |             |
| sklearn                           | 0.0          | <pip>          |             |
| spearmin                          | 0.1          | <pip>          |             |
| sqlite                            | 3.33.0       | h02c20be_0     |             |
| subprocess32                      | 3.5.4        | py27h7b6447c_0 |             |
| tensorboard                       | 2.1.0        | <pip>          |             |
| tensorflow                        | 2.1.0        | <pip>          |             |
| tensorflow-estimator              | 2.1.0        | <pip>          |             |

Figura A.1: Paquetes instalados en el entorno virtual con Python 2.7

```
# packages in environment at /home/lucia/anaconda2:
#
# Name Version Build Channel
_ipyw_jlab_nb_ext_conf 0.1.0 py27_0
alabaster 0.7.11 py27_0
anaconda 5.3.0 py27_0
anaconda-client 1.7.2 py27_0
anaconda-navigator 1.9.2 py27_0
anaconda-project 0.8.2 py27_0
appdirs 1.4.3 py27h28b3542_0
asn1crypto 0.24.0 py27_0
astroid 1.6.5 py27_0
astropy 2.0.8 py27h035aef0_0
atomicwrites 1.2.1 py27_0
attrs 18.2.0 py27h28b3542_0
automat 0.7.0 py27_0
babel 2.6.0 py27_0
backports 1.0 py27_1
backports.functools_lru_cache 1.5 py27_1
backports.shutil_get_terminal_size 1.0.0 py27_2
backports_abc 0.5 py27_0
beautifulsoup4 4.6.3 py27_0
bitarray 0.8.3 py27h14c3975_0
bkcharts 0.2 py27_0
blas 1.0 mkl
blaze 0.11.3 py27_0
bleach 2.1.4 py27_0
blosc 1.14.4 hdbcaa40_0
bokeh 0.13.0 py27_0
boto 2.49.0 py27_0
bottleneck 1.2.1 py27h035aef0_1
bzip2 1.0.6 h14c3975_5
ca-certificates 2018.03.07 0
cairo 1.14.12 h8948797_3
cdecimal 2.3 py27h14c3975_3
certifi 2018.8.24 py27_1
cffi 1.11.5 py27he75722e_1
chardet 3.0.4 py27_1
click 6.7 py27_0
cloudpickle 0.5.5 py27_0
clyent 1.2.2 py27_1
colorama 0.3.9 py27_0
conda 4.5.11 py27_0
conda-build 3.15.1 py27_0
conda-env 2.6.0 1
condafigparser 3.5.0 py27_0
constantly 15.1.0 py27h28b3542_0
contextlib2 0.5.5 py27_0
cryptography 2.3.1 py27hc365091_0
curl 7.61.0 h84994c4_0
cyclar 0.10.0 py27_0
cython 0.28.5 py27hf484d3e_0
cytoolz 0.9.0.1 py27h14c3975_1
dask 0.19.1 py27_0
dask-core 0.19.1 py27_0
datashape 0.5.4 py27_1
dbus 1.13.2 h714fa37_1
decorator 4.3.0 py27_0
defusedxml 0.5.0 py27_1
distributed 1.23.1 py27_0
docutils 0.14 py27_0
entrypoints 0.2.3 py27_2
enum34 1.1.6 py27_1
et_xmlfile 1.0.1 py27_0
expat 2.2.6 he6710b0_0
fastcache 1.0.2 py27h14c3975_2
filelock 3.0.8 py27_0
flask 1.0.2 py27_1
flask-cors 3.0.6 py27_0
fontconfig 2.13.0 h9420a91_0
freetype 2.9.1 h8a8886c_1
fribidi 1.0.5 h7b6447c_0
funcsign 1.0.2 py27_0
functools32 3.2.3.2 py27_1
futures 3.2.0 py27_0
get_terminal_size 1.0.0 haa9412d_0
gevent 1.3.6 py27h7b6447c_0
glib 2.56.2 hd408876_0
glob2 0.6 py27_0
gmp 6.1.2 h6c8ec71_1
gmpy2 2.0.8 py27h10f8cd9_2
graphite2 1.3.12 h23475e2_2
```

(a) Paquetes en Python 3.8

```
greenlet 0.4.15 py27h7b6447c_0
grin 1.2.1 py27_4
gst-plugins-base 1.14.0 hbbd80ab_1
gstreamer 1.14.0 hb453b48_1
h5py 2.8.0 py27h989c5e5_3
harfbuzz 1.8.8 hffaf4a1_0
hdf5 1.10.2 hba1933b_1
heapdict 1.0.0 py27_2
html5lib 1.0.1 py27_0
hyperlink 18.0.0 py27_0
icu 58.2 h9c2bf20_1
idna 2.7 py27_0
imageio 2.4.1 py27_0
imagesize 1.1.0 py27_0
incremental 17.5.0 py27_0
intel-openmp 2019.0 118
ipaddress 1.0.22 py27_0
ipykernel 4.9.0 py27_1
ipython 5.8.0 py27_0
ipython_genutils 0.2.0 py27_0
ipywidgets 7.4.1 py27_0
isort 4.3.4 py27_0
itsdangerous 0.24 py27_1
jbig 2.1 hdba287a_0
jdcals 1.4 py27_0
jedi 0.12.1 py27_0
jinja2 2.10 py27_0
jpeg 9b h024ee3a_2
jsonschema 2.6.0 py27_0
jupyter 1.0.0 py27_7
jupyter_client 5.2.3 py27_0
jupyter_console 5.2.0 py27_1
jupyter_core 4.4.0 py27_0
jupyterlab 0.33.11 py27_0
jupyterlab_launcher 0.11.2 py27h28b3542_0
kiwisolver 1.0.1 py27hf484d3e_0
lazy-object-proxy 1.3.1 py27h14c3975_2
libcurl 7.61.0 h1ad7b7a_0
libedit 3.1.20170329 h6b74fdf_2
libffi 3.2.1 hd88cf55_4
libgcc-ng 8.2.0 hdf63c60_1
libgfortran-ng 7.3.0 hdf63c60_0
libpng 1.6.34 hb9fc6fc_0
libsodium 1.0.16 h1bed415_0
libssh2 1.8.0 h9cfc8f7_4
libstdcxx-ng 8.2.0 hdf63c60_1
libtiff 4.0.9 he85c1e1_2
libtool 2.4.6 h544aabb_3
libuuid 1.0.3 h1bed415_2
libxcb 1.13 h1bed415_1
libxml2 2.9.8 h26e45fe_1
libxslt 1.1.32 h1312cb7_0
linecache2 1.0.0 py27_0
llvmlite 0.24.0 py27hdbcaa40_0
locket 0.2.0 py27_1
lxml 4.2.5 py27hefd8a0e_0
lzo 2.10 h49e0be7_2
markupsafe 1.0 py27h14c3975_1
matplotlib 2.2.3 py27hb69df0a_0
mccabe 0.6.1 py27_1
mistune 0.8.3 py27h14c3975_1
mkl 2019.0 118
mkl-service 1.1.2 py27h90e4bf4_5
mkl_fft 1.0.4 py27h4414c95_1
mkl_random 1.0.1 py27h4414c95_1
more-itertools 4.3.0 py27_0
mpc 1.1.0 h10f8cd9_1
mpfr 4.0.1 hdf1c602_3
mpmath 1.0.0 py27_2
msgpack-python 0.5.6 py27h6bb024c_1
multipledispatch 0.6.0 py27_0
navigator-updater 0.2.1 py27_0
nbconvert 5.4.0 py27_1
nbformat 4.4.0 py27_0
ncurses 6.1 hf484d3e_0
networkx 2.1 py27_0
nltk 3.3.0 py27_0
nose 1.3.7 py27_2
notebook 5.6.0 py27_0
numba 0.39.0 py27h04863e7_0
numexpr 2.6.8 py27hd89afb7_0
numpy 1.15.1 py27h1d66e8a_0
```

(b) Paquetes en Python 3.8

Figura A.2: Paquetes instalados en el entorno virtual con Python 3.8

```

numpy 1.15.1 py27h1d66e8a_0
numpy-base 1.15.1 py27h81de0dd_0
numpydoc 0.8.0 py27_0
odo 0.5.1 py27_0
olefile 0.46 py27_0
openpyxl 2.5.6 py27_0
openssl 1.0.2p h14c3975_0
packaging 17.1 py27_0
pandas 0.23.4 py27h04863e7_0
pandoc 1.19.2.1 hea2e7c5_1
pandocfilters 1.4.2 py27_1
pango 1.42.4 h049681c_0
parso 0.3.1 py27_0
partd 0.3.8 py27_0
patchelf 0.9 hf484d3e_2
path.py 11.1.0 py27_0
pathlib2 2.3.2 py27_0
patsy 0.5.0 py27_0
pcre 8.42 h439df22_0
pep8 1.7.1 py27_0
pexpect 4.6.0 py27_0
pickleshare 0.7.4 py27_0
pillow 5.2.0 py27heded4f4_0
pip 10.0.1 py27_0
pixman 0.34.0 hceecf20_3
pkginfo 1.4.2 py27_1
pluggy 0.7.1 py27h28b3542_0
ply 3.11 py27_0
prometheus_client 0.3.1 py27h28b3542_0
prompt_toolkit 1.0.15 py27_0
psutil 5.4.7 py27h14c3975_0
ptyprocess 0.6.0 py27_0
py 1.6.0 py27_0
pyasn1 0.4.4 py27h28b3542_0
pyasn1-modules 0.2.2 py27_0
pycairo 1.17.1 py27h2a1e443_0
pycodestyle 2.4.0 py27_0
pycosat 0.6.3 py27h14c3975_0
pyparser 2.18 py27_1
pycrypto 2.6.1 py27h14c3975_9
pycurl 7.43.0.2 py27hb7f436b_0
pyflakes 2.0.0 py27_0
pygments 2.2.0 py27_0
pylint 1.9.2 py27_0
pyodbc 4.0.24 py27he6710b0_0
pyopenssl 18.0.0 py27_0
pyparsing 2.2.0 py27_1
pyqt 5.9.2 py27h05f1152_2
pysocks 1.6.8 py27_0
pytables 3.4.4 py27ha205bf6_0
pytest 3.8.0 py27_0
python 2.7.15 h1571d57_0
python-dateutil 2.7.3 py27_0
pytz 2018.5 py27_0
pywavelets 1.0.0 py27hdd07704_0
pyyaml 3.13 py27h14c3975_0
pyzmq 17.1.2 py27h14c3975_0
qt 5.9.6 h8703b6f_2
qtawesome 0.4.4 py27_0
qtconsole 4.4.1 py27_0
qtpy 1.5.0 py27_0
readline 7.0 h7b6447c_5
requests 2.19.1 py27_0
rope 0.11.0 py27_0
ruamel_yaml 0.15.46 py27h14c3975_0
scandir 1.9.0 py27h14c3975_0
scikit-image 0.14.0 py27hf484d3e_1
scikit-learn 0.19.2 py27h4989274_0
scipy 1.1.0 py27hfa4b5c9_1
seaborn 0.9.0 py27_0
send2trash 1.5.0 py27_0
service_identity 17.0.0 py27h28b3542_0
setuptools 40.2.0 py27_0
simplegeneric 0.8.1 py27_2
singledispatch 3.4.0.3 py27_0
sip 4.19.8 py27hf484d3e_0
six 1.11.0 py27_1
snappy 1.1.7 hbae5bb6_3
snowballstemmer 1.2.1 py27_0
sortedcollections 1.0.1 py27_0
sortedcontainers 2.0.5 py27_0
sparsint 0.1 <pip>
sphinx 1.7.9 py27_0

```

(a) Paquetes en Python 3.8

```

sphinx 1.7.9 py27_0
sphinxcontrib 1.0 py27_1
sphinxcontrib-websupport 1.1.0 py27_1
spyder 3.3.1 py27_1
spyder-kernels 0.2.6 py27_0
sqlalchemy 1.2.11 py27h7b6447c_0
sqlite 3.24.0 h84994c4_0
ssl_match_hostname 3.5.0.1 py27_2
statsmodels 0.9.0 py27h035aef0_0
subprocess32 3.5.2 py27h14c3975_0
sympy 1.2 py27_0
tblib 1.3.2 py27_0
terminado 0.8.1 py27_1
testpath 0.3.1 py27_0
tk 8.6.8 hbc83047_0
toolz 0.9.0 py27_0
tornado 5.1 py27h14c3975_0
tqdm 4.26.0 py27h28b3542_0
traceback2 1.4.0 py27_0
traitlets 4.3.2 py27_0
twisted 18.7.0 py27h14c3975_1
typing 3.6.6 py27_0
unicodectsv 0.14.1 py27_0
unittest2 1.1.0 py27_0
unixodbc 2.3.7 h14c3975_0
urllib3 1.23 py27_0
wcwidth 0.1.7 py27_0
webencodings 0.5.1 py27_1
werkzeug 0.14.1 py27_0
wheel 0.31.1 py27_0
widgetsnextension 3.4.1 py27_0
wrapt 1.10.11 py27h14c3975_2
xlrd 1.1.0 py27_1
xlsxwriter 1.1.0 py27_0
xlwt 1.3.0 py27_0
xz 5.2.4 h14c3975_4
yaml 0.1.7 had09818_2
zeromq 4.2.5 hf484d3e_1
zict 0.1.3 py27_0
zlib 1.2.11 ha838bed_2
zope 1.0 py27_1
zope.interface 4.5.0 py27h14c3975_0

```

(b) Paquetes en Python 3.8

Figura A.3: Paquetes instalados en el entorno virtual con Python 3.8