



Universidad de Concepción
Dirección de Pregrado
Facultad de Ingeniería - Programa de Magíster en Ciencias de la Computación

SELECCIÓN AUTOMÁTICA DE ALGORITMOS PARA EL PROBLEMA DE CLASIFICACIÓN BINARIA

Tesis para optar al grado de
MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

POR

Diego Henríquez Valenzuela
CONCEPCIÓN, CHILE

Marzo, 2023

Profesor guía: Julio Erasmo Godoy del Campo
Co-guía externo: Roberto Javier Asín Achá
Departamento de Ingeniería Informática y Ciencias de la Computación
Facultad de Ingeniería
Universidad de Concepción

Resumen

El problema de la clasificación binaria surge en diversas situaciones diferentes, generando instancias del problema que pueden ser muy diferentes entre sí. Este problema ha sido ampliamente estudiado y, por lo tanto, existen múltiples algoritmos diferentes para resolverlo. Estos algoritmos tienen rendimientos variables dependiendo de las instancias del problema que enfrentan. En este documento estudiamos la creación de un selector automático de algoritmos que predice el mejor algoritmo dentro de un portafolio para una instancia dada del problema de clasificación binaria. Además, se explican los procesos involucrados en su creación, como la generación de instancias de clasificación binaria artificiales para entrenar y testear el modelo, y el proceso de construcción de un portafolio de algoritmos desde el cual el modelo seleccionará el algoritmo adecuado para cada instancia. Se propone una caracterización de las instancias de clasificación binaria, en base a approximatePCA y se utiliza la red neuronal convolucional AlexNet para recibir las caracterizaciones y seleccionar algoritmos. Después, se evalúan los resultados, comparándolos con un modelo del estado del arte y con el mejor algoritmo promedio, gracias a la métrica \hat{m} . Finalmente, se reflexiona acerca de los resultados obtenidos y los futuros avances que se pueden realizar.

Tabla de Contenido

Resumen	ii
Índice de Figuras	v
Capítulo 1 INTRODUCCIÓN	1
1.1 Contexto	1
1.2 Descripción del problema	2
1.3 Hipótesis	2
1.4 Preguntas de investigación	3
1.5 Objetivo Principal	3
1.5.1 Objetivos Específicos	3
1.6 Alcances y limitaciones	4
Capítulo 2 MARCO TEÓRICO	5
2.1 Clasificación binaria	5
2.1.1 Definición del problema	5
2.1.2 Algoritmos de clasificación	5
2.1.3 Métricas de evaluación	7
2.2 Selección automática de algoritmos	10
2.2.1 Definición del problema	10
2.2.2 Métrica \hat{m}	10
2.3 Principal component analysis (PCA)	11
2.4 Deep learning	12
2.4.1 Redes neuronales convolucionales	12
2.5 Optimización bayesiana	13
Capítulo 3 TRABAJOS RELACIONADOS	15
3.1 Selección automática de algoritmos para TSP	15
3.2 Matriz de similitud rala	16

3.3	Auto-SKlearn	17
3.3.1	Meta-learning	18
3.3.2	Pipelines	20
3.3.3	Construcción del conjunto de pipelines	20
3.4	Auto-SKlearn 2.0	21
3.4.1	Construcción del portafolio	21
Capítulo 4	SELECCIÓN DE ALGORITMOS PARA CLASIFICACIÓN BINARIA BASADA EN DEEP LEARNING	22
4.1	Generación de Datos	22
4.2	Construcción del Portafolio de Algoritmos	23
4.3	Caracterización de las instancias	24
4.4	Red Neuronal Convolutiva	25
Capítulo 5	RESULTADOS	28
Capítulo 6	CONCLUSIONES Y TRABAJO FUTURO	32
Bibliografía		34



Índice de Figuras

1.1	Problema de Clasificación Binaria	2
2.1	Matriz de confusión	7
2.2	Optimización bayesiana en un problema simple de 1 dimensión [1] .	14
3.1	Representación de grilla matricial para TSP [2]	16
3.2	Ranking de algoritmos según el tiempo [2]	16
3.3	Bloques de la grilla con $p=3$ dimensiones y $k_1, k_2, k_3=5$. Los bloques amarillos son adyacentes al bloque naranja [3]	17
3.4	Auto-SKlearn [4]	18
3.5	Meta-features utilizadas por auto-SKlearn [4]	19
3.6	Clasificadores (izquierda), preprocesadores (derecha) y sus hiperparámetros categóricos y continuos[4]	20
4.1	Imágenes correspondientes a 6 instancias de problemas de clasificación binaria	26
4.2	Accuracy del entrenamiento (naranja) y la validación (morado) de la red AlexNet	27
4.3	Pérdida durante el entrenamiento (naranja) y la validación (morado) de la red AlexNet	27
5.1	Clases de los datos de entrenamiento	29
5.2	Clases de los datos de prueba	29
5.3	Matriz de confusión	30

Capítulo 1

INTRODUCCIÓN

1.1 Contexto

El problema de clasificación binaria ha sido ampliamente estudiado en el campo del aprendizaje automático [5, 6, 7, 8, 9, 10, 11] y se han desarrollado múltiples algoritmos para solucionarlo. Muchos problemas de la realidad pueden ser vistos como problemas de clasificación binaria, lo que hace que este problema tenga muchas aplicaciones prácticas. Algunos ejemplos de esto son encontrar en un conjunto de exámenes médicos aquellos que pertenecen a pacientes que poseen una determinada condición, o verificar si una serie de productos cumplen o no con ciertas condiciones. La naturaleza y distribución de los datos de todos estos problemas reales es distinta, por lo que las instancias del problema de clasificación binaria pueden ser muy diferentes entre sí. Las múltiples soluciones propuestas muestran resultados de calidad variable dependiendo de la instancia específica a la que se aplican, lo que hace que no haya un solo algoritmo con el mejor rendimiento en todos los casos. Dado esto, sería interesante poder seleccionar, para cada instancia del problema de clasificación binaria, el algoritmo que logra los mejores resultados en esa instancia en particular. Esto nos lleva al problema de selección automática de algoritmos, el cual es estudiado en el campo del aprendizaje automático [12, 13, 14, 15, 16, 17]. Actualmente, la mejor solución para selección automática de algoritmos para el problema de clasificación binaria se obtiene con AutoSKlearn, un modelo propuesto en el trabajo de M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, y F. Hutter, “Efficient and robust automated machine learning” [4]. Este modelo realiza 50 iteraciones de su algoritmo para obtener la solución, lo que toma tiempo. Nos interesa encontrar una buena solución en un tiempo menor, entrenando solo un modelo.

1.2 Descripción del problema

El problema de clasificación binaria consiste en separar un conjunto de datos en 2 clases, a partir de los atributos de cada dato. El objetivo es seleccionar para cada dato, la clase que le corresponde, como se puede ver en la Figura 1.1.

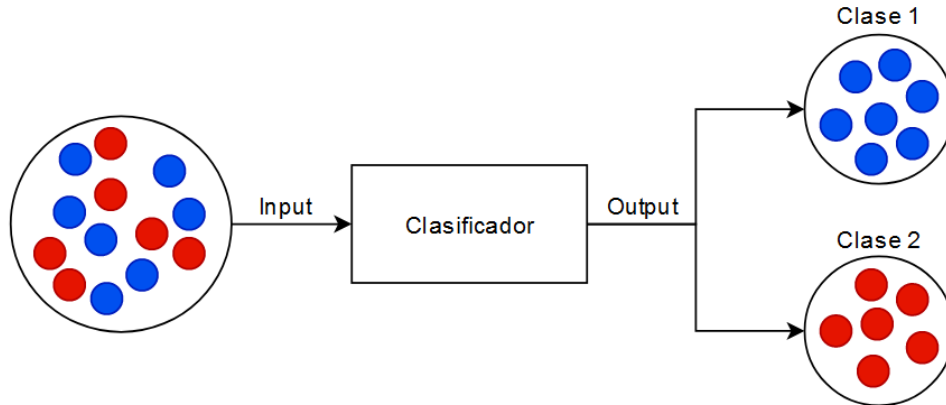


Figura 1.1: Problema de Clasificación Binaria

Las instancias de este problema son muy diversas y distintas entre sí, y las múltiples soluciones propuestas para el problema muestran resultados de calidad variable según a qué instancia del problema se enfrentan. Nos interesa tener una forma de seleccionar, dada una instancia del problema de selección binaria, la clase que le corresponde.

El problema de la selección automática de algoritmos se define como la elección, para una instancia específica de un problema dado, del algoritmo dentro de un portafolio de algoritmos, que logra el mejor resultado entre todos los algoritmos del portafolio para la instancia dada, según una métrica específica.

El problema de selección automática de algoritmos para clasificación binaria es entonces, dada una instancia del problema de clasificación binaria y una métrica de evaluación, seleccionar el algoritmo dentro de un portafolio de algoritmos para clasificación binaria que obtenga el mejor resultado para la instancia específica, según la métrica de evaluación.

1.3 Hipótesis

Una caracterización rápida y eficaz de las instancias de problemas de clasificación binaria que provea suficiente información a una red neuronal convolucional, permitirá a este

modelo mejorar el estado del arte en selección automática de algoritmos para clasificación binaria, obteniendo mejores resultados que la primera iteración de AutoSKlearn, según la métrica \hat{m} .

1.4 Preguntas de investigación

Queremos responder las siguientes preguntas.

- ¿La medición de la dispersión de la proyección en 3 dimensiones de los puntos que componen una instancia de clasificación binaria puede ser usada para caracterizar la instancia, para realizar tareas de aprendizaje de máquinas?
- ¿Se pueden utilizar los algoritmos de aprendizaje de máquina actuales para explotar la caracterización antes descrita para realizar selección automática de algoritmos en un tiempo corto?

1.5 Objetivo Principal

El objetivo principal de este trabajo es encontrar una forma efectiva y rápida de caracterizar instancias de clasificación binaria y proponer un selector de algoritmos que utilice esta caracterización que sea capaz de mejorar al mejor algoritmo individual promedio en el portafolio, así como superar en rendimiento a la primera iteración de AutoSKlearn.

1.5.1 Objetivos Específicos

- Crear un entorno de experimentación. Esto nos permitirá generar los datos necesarios para la realización del trabajo.
- Crear una forma eficiente de caracterizar instancias de problemas de clasificación binaria. La caracterización deberá dar suficiente información a un selector, para que este tenga mejores resultados que el estado del arte según la métrica \hat{m} .
- Crear un modelo de machine learning adecuado para la caracterización definida, que seleccione el mejor algoritmo de clasificación binaria para la instancia entregada. Se evaluará con la métrica \hat{m} , con el objetivo de superar la primera iteración de

AutoSKlearn. Además se medirá el tiempo de ejecución del modelo, sumado al tiempo de caracterización de la instancia. Este tiempo debe ser menor al que toma la primera iteración de AutoSKlearn.

- Evaluar el desempeño del modelo y compararlo con el mejor algoritmo individual promedio y el modelo resultado de la primera iteración de AutoSKlearn.

1.6 Alcances y limitaciones

Los datos de entrenamiento utilizados en este trabajo fueron generados artificialmente según parámetros determinados. Las limitaciones de estos datos serán reflejadas en el modelo. A pesar de que para el problema de clasificación binaria existen algoritmos supervisados y no supervisados, en este trabajo se aborda clasificación binaria como un problema de aprendizaje de máquina supervisado, por lo que sólo serán considerados los algoritmos supervisados. Otro límite es que los problemas de clasificación binaria tengan únicamente atributos continuos. Problemas con atributos categóricos no serán considerados en la solución. Para los experimentos se usó la partición savio3 del cluster Savio de la UC Berkeley. Esta cuenta con una CPU Intel Xeon Skylake 6130 @ 2.1 GHz y 96 GB de memoria por nodo.

Capítulo 2

MARCO TEÓRICO

En este capítulo se definen los problemas a tratar y las métricas de evaluación utilizadas en el resto del trabajo, explicando sus ventajas y desventajas. También se revisan otras herramientas que serán útiles en el resto del trabajo.

2.1 Clasificación binaria

2.1.1 Definición del problema

El problema de clasificación binaria involucra un conjunto de datos de entrenamiento X_{train} y un conjunto de datos de prueba X_{test} , junto con sus etiquetas Y_{train} y Y_{test} . La etiqueta para cada punto de datos es 0 o 1, indicando la clase a la que pertenece. El objetivo es entrenar una función $f(X_{train}, X_{test}, Y_{train}) = Y'$ de manera que Y' sea lo más cercano posible a Y_{test} , lo que es medido a través de una métrica de evaluación [18]. Existen múltiples métricas de evaluación para el problema de clasificación binaria. Detallamos algunas ellas en 2.1.3. En este trabajo utilizamos f1-score.

2.1.2 Algoritmos de clasificación

La tabla 2.1 contiene algunos algoritmos de clasificación que se utilizan para resolver problemas de clasificación binaria.

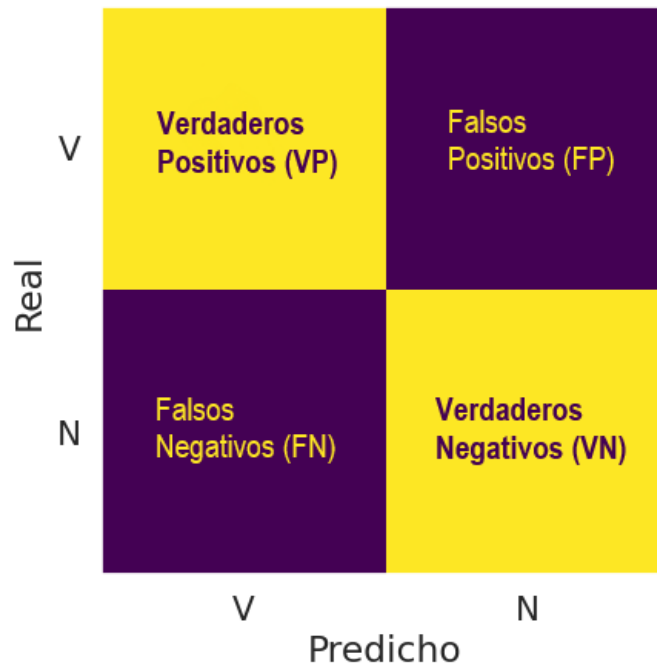
Algoritmo	Descripción	Debilidades
Discriminant Analysis [19]	Proyecta los datos en un espacio con menor número de dimensiones, intentando maximizar la distancia entre las medias de ambas clases y minimizar la variación dentro de cada clase.	Sensible a distribuciones no gaussianas.
Support Vector Machine [20]	Separa las clases encontrando el hiperplano en el espacio de atributos que se encuentre más lejos de los puntos más cercanos de cada clase.	Ineficiente en grandes conjuntos de datos.
Passive Aggressive [21]	Algoritmo de machine learning que recibe los datos de entrada secuencialmente. Se llama así porque se comporta de forma pasiva, sin realizar cambios al modelo, hasta que encuentra un error de clasificación. En cuanto encuentra un error, se comporta de manera agresiva, actualizando el modelo rápidamente.	Sensible a la elección del parámetro de penalización.
MLP [22]	Multilayer Perceptron es una red neuronal que consiste en múltiples capas, cada una conectada completamente a la siguiente. Esta red se entrena con los datos de entrenamiento, con el objetivo de que aprenda a clasificar los datos en sus respectivas clases.	Requiere un gran conjunto de datos para entrenar eficientemente.

Tabla 2.1: Algoritmos de clasificación binaria

2.1.3 Métricas de evaluación

- **Matriz de confusión**

Una matriz de confusión es una manera de visualizar los resultados obtenidos en un problema de clasificación. Es una matriz cuadrada con n filas y n columnas, donde n es el número de clases en el problema de clasificación. Las columnas de la matriz corresponden a las clases predichas para cada punto de datos, mientras que las filas representan sus clases reales. Así, un punto de datos predicho como perteneciente a la clase j y que realmente pertenece a la clase i , se encuentra en la fila i y columna j de la matriz de confusión. Es importante señalar que si la clase predicha de un punto de datos es correcta, el punto de datos estará en una celda en la diagonal de la matriz. En el caso de la clasificación binaria, el número de clases (n) es 2, por lo que la matriz de confusión es una matriz de 2×2 . Además, una clase se designa como positiva y la otra como negativa. Obtenemos entonces la matriz de la figura 2.1



Real	V	Verdaderos Positivos (VP)	Falsos Positivos (FP)
	N	Falsos Negativos (FN)	Verdaderos Negativos (VN)
		V	N
		Predicho	

Figura 2.1: Matriz de confusión

- **Accuracy**

La métrica *accuracy* mide cuántos aciertos logró el algoritmo de clasificación binaria [18]. En una matriz de confusión, esto es la proporción de los datos que se encuentran en la diagonal de la matriz.

$$Accuracy(Y_{pred}, Y_{test}) = \frac{VP + VN}{VP + FN + FP + VN}$$

Esta métrica funciona bien cuando los datos son balanceados, pero tiene problemas cuando no lo son, o sea, cuando hay muchos más datos pertenecientes a una clase que a la otra, puesto que evalúa cada acierto con el mismo puntaje. Esto es un problema puesto que muchos de los problemas de clasificación binaria del mundo real son desbalanceados, puesto que consisten en identificar los elementos que se salen de la norma y por lo tanto pertenecen a otra clase. Un ejemplo de esto es la identificación de una enfermedad en exámenes médicos, donde la mayoría de los exámenes pertenecen a personas sin la enfermedad, pero el interés del problema está en identificar los exámenes de las personas enfermas.

- **Precision**

La métrica de *precision* [18] [23] se concentra en una de las 2 clases, la cual llamaremos positiva. Queremos saber qué proporción de los datos que fueron etiquetados por el algoritmo de forma positiva, fueron etiquetados correctamente. En la matriz de confusión, esto es la proporción de los datos en la fila de la clase positiva que se encuentran también en la columna de la clase positiva.

$$Precision(Y_{pred}, Y_{test}) = \frac{TP}{TP + FP}$$

Esta métrica funciona mucho mejor en casos desbalanceados, siendo su principal objetivo evitar que se les de etiquetas positivas a datos que deben ser etiquetados negativamente. En el ejemplo de los exámenes médicos, esto sería evitar que la enfermedad sea identificada en los exámenes de pacientes sin la enfermedad.

- **Recall**

Recall [18] [23], es la métrica que evalúa la proporción de los datos que debían ser etiquetados de forma positiva, fueron etiquetados correctamente por el algoritmo.

$$\text{Recall}(Y_{pred}, Y_{test}) = \frac{TP}{TP + FN}$$

Al igual que la métrica anterior, esta métrica funciona bien para casos desbalanceados. Esta vez, el objetivo es etiquetar de forma positiva la mayor cantidad de casos que deben ser etiquetados positivamente. En el ejemplo esto quiere decir identificar la enfermedad en la mayor cantidad de exámenes de pacientes con la enfermedad posible. En la matriz de confusión, esto es la proporción de los datos en la columna de la clase positiva que se encuentran también en la fila de la clase positiva.

- **F1-score**

Como *precision* busca etiquetar como positivos, sólo los datos positivos, y *recall*, etiquetar la mayor cantidad de datos positivos correctamente, queremos, idealmente, maximizar ambas métricas. En el caso de los exámenes médicos, querríamos minimizar la cantidad de identificaciones falsas de la enfermedad en exámenes de pacientes que no la tienen, al mismo tiempo que maximizamos las identificaciones de la enfermedad en exámenes de pacientes que sí la tienen. Sin embargo, en la práctica, cuando se intenta maximizar una de estas 2 métricas, es a costa de obtener un menor valor en la otra.

F1-score es una métrica que intenta solucionar este conflicto, combinando las 2 métricas anteriores. Se define *F1-score* como la media armónica de *precision* y *recall* [18] [23].

$$F1\text{-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Notamos que todas las métricas descritas son deterministas y toman un valor entre 0 y 1.

2.2 Selección automática de algoritmos

2.2.1 Definición del problema

El problema de la selección automática de algoritmos [24] consiste en un conjunto de instancias de un problema, un portafolio P de n algoritmos $A_{i \in \{0 \dots n-1\}}$ para este problema y una métrica de evaluación que evalúa los resultados de estos algoritmos. En este trabajo, consideramos la selección automática de algoritmos como un problema de clasificación, donde la etiqueta l para cada punto de datos I es $l \in \{0 \dots n-1\} | \forall A_i \in P, m(A_i(I)) \geq m(A_l(I))$. Tenemos entonces, un conjunto de datos de entrenamiento y prueba X_{train} y X_{test} , sus etiquetas Y_{train} y Y_{test} . El objetivo es, una vez más, entrenar una función $f(X_{train}, X_{test}, Y_{train}) = Y'$ para que Y' sea lo más similar posible a Y_{test} .

2.2.2 Métrica \hat{m}

Una forma útil de evaluar los resultados de un modelo de selección automática de algoritmos es la métrica \hat{m} propuesta por Lindauer, M., van Rijn, J. N., & Kotthoff, L. [25]. Tenemos que cada instancia $I_i \in I$ es resuelta por cada algoritmo $A_j \in A$ con una evaluación de $f(A_j, I_i)$. La evaluación total m de un algoritmo A_j para el conjunto de instancias I de tamaño n , es, por lo tanto, el promedio de las evaluaciones con la métrica f de este algoritmo con cada instancia I_i .

$$m_{A_j} = \frac{\sum_{i=0}^{n-1} f(A_j, I_i)}{n}$$

Se define como *Single Best Solver (SBS)*, el algoritmo A_j que obtenga la mejor evaluación según m .

Se define como *Virtual Best Solver (VBS)*, el algoritmo ficticio que se comporta para cada instancia I_i como el algoritmo A_j con el mayor $f(A_j, I_i)$

Para evaluar nuestro selector de algoritmos S utilizaremos la métrica \hat{m} [25].

$$\hat{m}_S = \frac{m_S - m_{VBS}}{m_{SBS} - m_{VBS}}$$

Esta métrica es interesante porque compara el rendimiento del selector con el SBS y el VBS. Si el puntaje obtenido por el modelo es menor a 1, quiere decir que es mejor que el

SBS, y si es mayor a 1, que es peor.

2.3 Principal component analysis (PCA)

En problemas como clasificación binaria, los datos suelen tener muchos atributos. Esto hace que visualizar los datos sea muy difícil, puesto que tratamos cada uno de estos atributos como una dimensión. Para solucionar esto, se pueden emplear técnicas de reducción de dimensionalidad, como PCA [26].

En PCA queremos encontrar un número determinado p de dimensiones que sean combinaciones lineales de las dimensiones originales y que den la mayor cantidad de información, o, en otras palabras, tengan la mayor varianza posible. Para esto, PCA selecciona un conjunto de p componentes principales. La primera componente principal es la dimensión en la que los datos tengan mayor varianza, luego, la segunda componente principal es la dimensión perpendicular a la primer componente principal en que los datos tengan mayor varianza. La tercera componente principal es la dimensión perpendicular al plano de las dos primeras componentes, que tenga la mayor varianza. Así, se seleccionan componentes principales hasta llegar a p . Estas p componentes, son las p dimensiones en las que se representan los datos.

Obtener las p componentes descritas es equivalente a obtener los p vectores propios v con los mayores valores propios λ . Primero obtenemos los n valores propios λ , que serán los valores que resuelvan la siguiente ecuación, siendo C la matriz de covarianza de los datos centrados en las dimensiones originales, e I la matriz identidad.

$$\det(C - \lambda I) = 0$$

Luego, obtenemos los vectores propios, resolviendo la siguiente ecuación, para cada valor propio λ_i , $i = 1 \dots n$ y asegurándonos de que el módulo de cada vector propio v_i sea igual a 1.

$$Cv_i = \lambda_i v_i$$

Una vez que tenemos los vectores propios correspondientes a los p mayores valores propios, estos vectores propios son las nuevas dimensiones en las que representamos los datos.

2.4 Deep learning

Las redes neuronales son modelos computacionales compuestos de capas de procesamiento, cuyo objetivo es aprender representaciones de datos con múltiples niveles de abstracción. Cada capa está compuesta por neuronas. Cada neurona contiene un valor y está conectada con las neuronas de la capa siguiente a través de un canal, el cual tiene un peso. El valor de las neuronas en una capa depende del valor que reciben de la capa anterior, lo que depende de los valores de las neuronas en la capa anterior y de los pesos que conectan estas 2 capas. Además, cada neurona tiene un sesgo que determina el valor total que debe recibir de la capa anterior para que su valor sea mayor a 0. A medida que una red neuronal es entrenada para resolver un problema, sus pesos y sesgos van cambiando, intentando minimizar el error de sus resultados. Así, se espera que la red neuronal aprenda a resolver el problema. En concreto, el objetivo de una red de neuronal es aproximar alguna función f . Por ejemplo, para un clasificador, $y = f(x)$ asigna una entrada x a una clase y . Una red neuronal define $y = f(x; \theta)$ y aprende el valor de los parámetros θ que resultan en la mejor aproximación de la función [27].

2.4.1 Redes neuronales convolucionales

Las redes neuronales convolucionales están diseñadas para procesar datos en forma de arreglos, como por ejemplo una imagen a color, compuesta por 3 arreglos de 2 dimensiones, que representan los colores en RGB [?]. La primera parte de una red neuronal convolucional, está compuesta por capas de convolución y de max pooling. Las capas de convolución contienen filtros, con los que se realiza convolución a los datos, antes de pasar a la siguiente capa. La convolución es un proceso en que cada dato es transformado según los valores de los datos dentro del tamaño del filtro y según los pesos que el filtro asigne a estos valores. Una red neuronal convolucional (CNN) utiliza operaciones de convolución para procesar datos estructurados, como imágenes. Matemáticamente, esto se expresa como:

$$Z^{[l]} = f(W^{[l]} * A^{[l-1]} + b^{[l]})$$

Donde $A^{[l-1]}$ es la activación de la capa anterior, $W^{[l]}$ es el conjunto de pesos de los filtros de la capa actual, $b^{[l]}$ es el sesgo de la capa actual, $*$ representa la operación de

convolución y f es una función de activación.

Este proceso de convolución es capaz de reconocer patrones en los datos. Las capas de max pooling tienen un tamaño de filtro y un valor de stride. Su objetivo es reducir las dimensiones de los datos. Se construyen moviendo el filtro por los datos según el stride y seleccionando así grupos de datos. De cada grupo de datos, se selecciona sólo el mayor para continuar hacia la siguiente capa. Algunas redes neuronales convolucionales conocidas son:

- **Alexnet** [28] Alexnet es una red neuronal convolucional creada en 2012. Contiene 8 capas: 5 capas de convolución, 2 capas ocultas completamente conectadas y una capa completamente conectada de salida.
- **VGG16** [29] VGG16 fue publicada en 2015 y contiene 13 capas de convolución, seguidas por 3 capas completamente conectadas. Esta red requiere de una gran cantidad de tiempo y espacio en disco para ser entrenada, lo que puede volverla ineficiente.
- **ResNet50** [30] La red contiene conexiones de atajo que se saltan algunas capas, lo que la convierte en una red residual y le permite añadir un mayor número de capas de convolución. Esta red contiene 50 capas ponderadas. Su arquitectura está basada en la de las redes VGG, sin embargo es menos compleja. Esta red fue publicada en 2015.
- **GoogLeNet** [31] Esta red neuronal convolucional fue publicada en 2014 y contiene 20 capas de convolución divididas en bloques llamados módulos de incorporación.

2.5 Optimización bayesiana

La optimización bayesiana es un proceso iterativo que consta de una función objetivo y una función de adquisición. El proceso intenta encontrar el máximo de la función objetivo tomando muestras. Para tomar las muestras de manera eficiente, se ayuda de la función de adquisición. En cada iteración, se toma una muestra de la función objetivo en el punto en el que la función de adquisición sea más alta. La función de adquisición es mayor en los puntos en que hay más incertidumbre acerca de la función objetivo, pero además se

espera que la función objetivo sea mayor [1] [32]. Luego de obtener el resultado de la muestra, la información que tenemos acerca de la función objetivo cambia. Esto altera la función de adquisición para la siguiente iteración. El proceso puede verse en la figura 2.2.

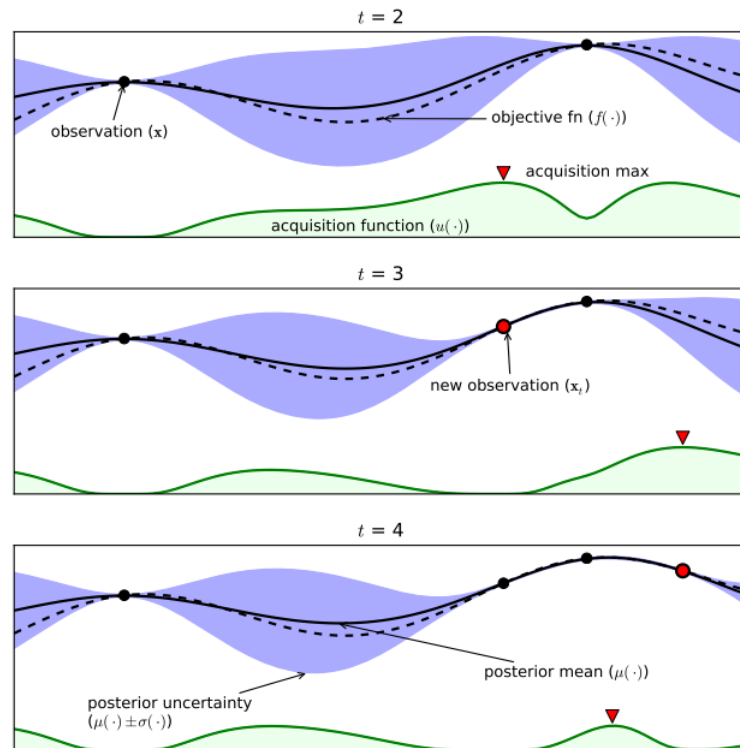


Figura 2.2: Optimización bayesiana en un problema simple de 1 dimensión [1]

Capítulo 3

TRABAJOS RELACIONADOS

Este capítulo contiene la revisión de trabajos similares que inspiraron la realización de éste y que aportan técnicas útiles para la caracterización de instancias. Luego, revisa el funcionamiento del modelo de auto-SKlearn, que será de importancia para este trabajo.

3.1 Selección automática de algoritmos para TSP

Este trabajo está inspirado en el trabajo realizado por I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy y R. Asín-Achá, “Improving the state-of-the-art in the Traveling Salesman Problem: An Anytime Automatic Algorithm Selection” [2]. Este trabajo presenta una nueva metaheurística para el Traveling Salesman Problem euclidiano (TSP) utilizando un modelo de selección automática de algoritmos para este problema. El modelo selecciona entre 5 algoritmos del estado del arte, el que obtenga el mejor resultado para una instancia del TSP y el tiempo entregado para obtener la solución. Para entregar las instancias del problema a una red neuronal convolucional, se usa una caracterización que simplifica la información y que evita cálculos costosos. La caracterización usada es una representación espacial de los nodos utilizando una grilla matricial. Se separa el espacio en el que se encuentran los nodos, separando cada dimensión en intervalos, creando así una grilla. Se cuenta cuántos nodos se encuentran dentro de cada celda de la grilla y este número es representado en cada celda. Así se obtiene una representación matricial de la instancia. El proceso se puede ver en la figura 3.1.

La caracterización de cada instancia de TSP consiste, entonces, en esta representación matricial. Esta caracterización es utilizada para entrenar una red neuronal convolucional. Esta red logra después predecir cuál algoritmo presentará un mejor resultado obteniendo información de la representación compacta de la instancia del problema. La red entrega un resultado en forma de un ranking sobre qué algoritmo es mejor para cada instante de tiempo, considerando intervalos de tiempo discretos, como en la figura 3.2.

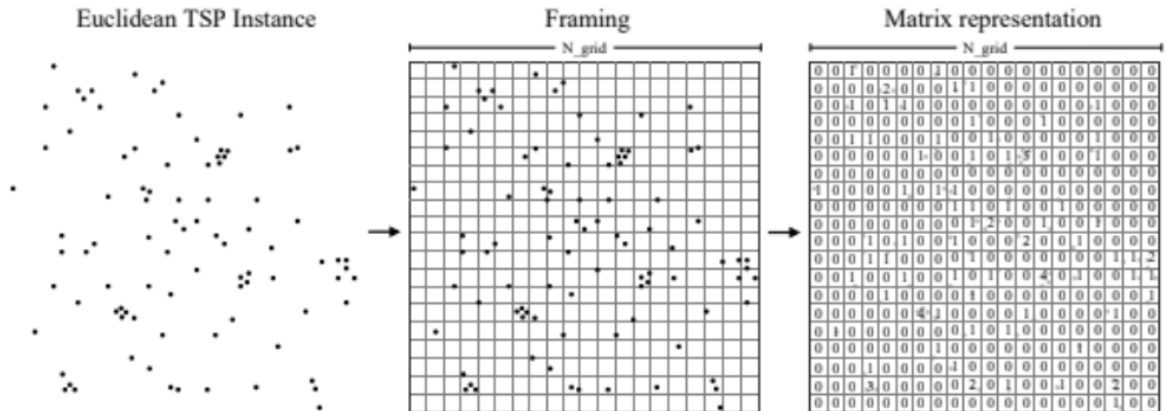


Figura 3.1: Representación de grilla matricial para TSP [2]

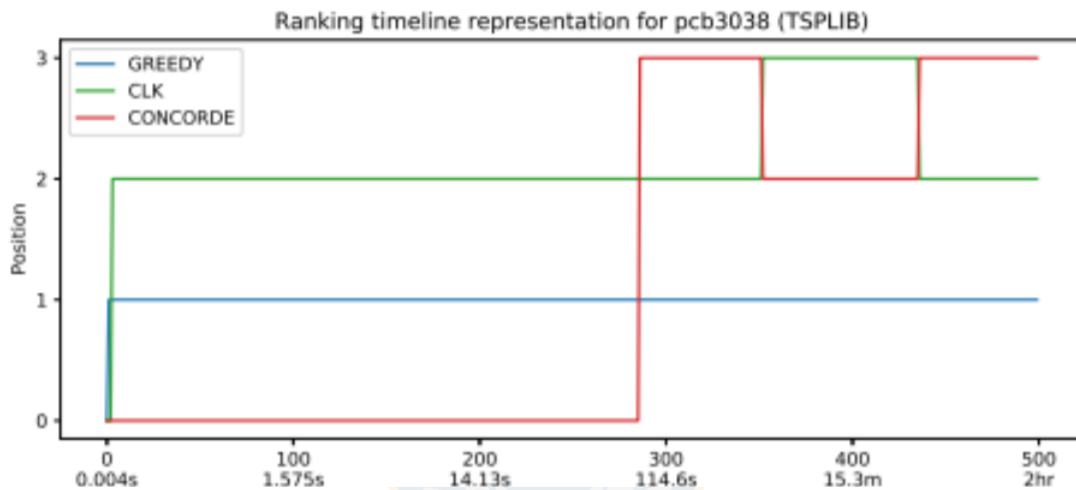


Figura 3.2: Ranking de algoritmos según el tiempo [2]

Esta red fue entrenada con instancias generadas, y testeada con instancias públicas, del mundo real. Los resultados obtenidos tienen una *accuracy* del 79.8%.

Queremos utilizar la idea de [2], pero esta vez para clasificación binaria. Para esto será necesario encontrar una caracterización rápida que le dé suficiente información a la red.

3.2 Matriz de similitud rala

Una idea para la caracterización se encuentra en el trabajo de Hochbaum, D. S., & Baumann, P. “Sparse Computation for Large-Scale Data Mining.” [3]. En este trabajo se

utiliza `approximatePCA` para reducir la dimensionalidad de los datos a p dimensiones (ellos recomiendan $p \leq 3$). `ApproximatePCA` es un método que toma una cantidad determinada de datos y de atributos al azar de la instancia y efectúa PCA en ellos, utilizando luego esta transformación en la totalidad de la instancia. Después de esto, se determina un valor k_i para cada dimensión $i = 1 \dots p$ y se divide cada dimensión i en k_i particiones. De esta forma obtenemos un espacio de $\prod_{i=1}^p k_i$ bloques en los que podemos dividir todos los datos, como se ve en la figura 3.3. Finalmente se construye una matriz de similitudes, pero en vez de comparar cada dato con todos los demás, solo se comparan los datos que se consideren similares. Los datos se consideran similares si pertenecen al mismo bloque o a bloques adyacentes. Este método permite construir una matriz de similitudes rara, que es muy útil para representar datos en grandes cantidades.

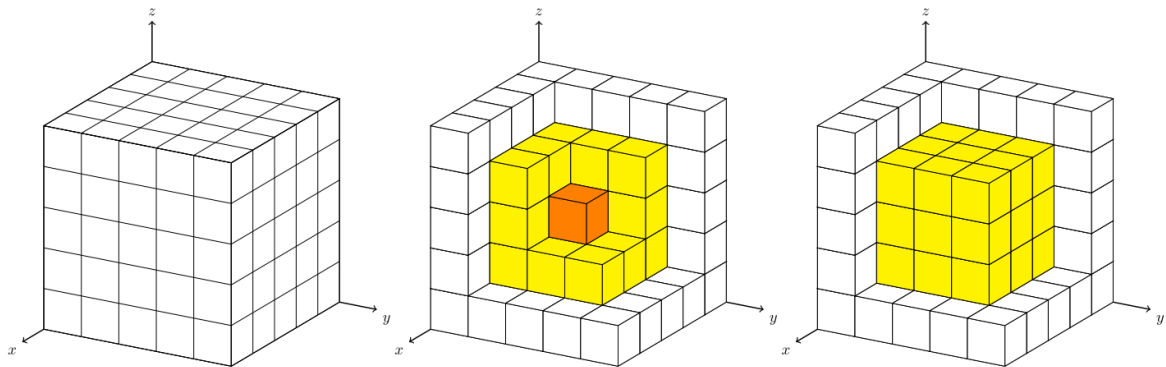


Figura 3.3: Bloques de la grilla con $p=3$ dimensiones y $k_1, k_2, k_3=5$. Los bloques amarillos son adyacentes al bloque naranja [3]

3.3 Auto-SKlearn

Auto-SKlearn es un modelo de *auto machine learning* que resultó ganador de las competencias de autoML de los años 2015 y 2018 [33]. Este modelo combina la librería `scikit-learn` con el método del estado del arte de optimización bayesiana, SMAC [32]. El modelo está implementado para los problemas de clasificación binaria, clasificación con múltiples clases y clasificación con múltiples etiquetas. En este trabajo, nos concentramos en su funcionamiento para clasificación binaria. De forma simple, auto-SKlearn recibe una

instancia de clasificación binaria, un tiempo límite y una función de evaluación. Luego, obtiene información acerca de la instancia, y con ayuda de optimización bayesiana, crea un conjunto de algoritmos. Utiliza este conjunto de algoritmos para predecir en la instancia, y devuelve los resultados. El proceso está ilustrado en la figura 3.4.

En concreto, el funcionamiento de auto-SKlearn es el siguiente:

- Recibe una entrada compuesta por un conjunto de datos de entrenamiento (X_{train}), sus etiquetas (Y_{train}), un conjunto de datos de test (X_{test}), un presupuesto en tiempo (b) y una función de evaluación (L)
- Realiza *meta-learning* sobre los datos, adquiriendo *meta-features* que le ayudarán a comenzar el proceso de optimización bayesiana [34]
- Genera pipelines iterativamente mediante el proceso de optimización bayesiana [1]. Cada pipeline está compuesta por un conjunto de preprocesadores de datos, un preprocesador de atributos y un clasificador.
- Construye un conjunto de 50 pipelines de entre las que fueron generadas.
- Entrega el problema de clasificación binaria (X_{train} , Y_{train} y X_{test}) a este conjunto y devuelve el resultado.

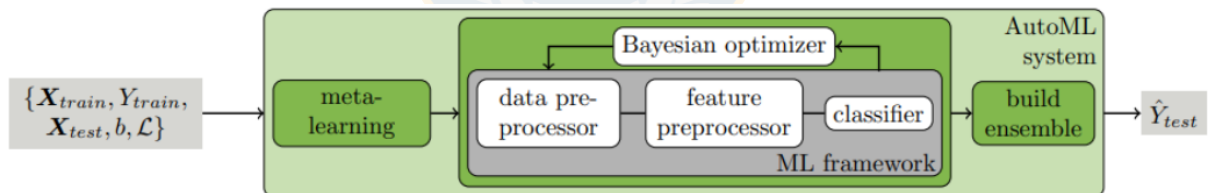


Figura 3.4: Auto-SKlearn [4]

Revisamos los pasos en mayor detalle a continuación.

3.3.1 Meta-learning

Para mejorar la configuración inicial del proceso de optimización bayesiana, auto-SKlearn utiliza *meta-learning* [34]. El proceso de *meta-learning* consiste en extraer *meta-features* de los datos. Estas *meta-features* aportarán información acerca de los datos y permitirán

seleccionar una mejor configuración inicial. Algunas *meta-features* extraídas son simples, como la cantidad de datos o la cantidad de atributos, pero otras son más complejas. En la figura 3.5 se ven en detalle las *meta-features* utilizadas por auto-SKlearn, además de los valores que pueden alcanzar y el tiempo que cuesta calcularlas.

Meta-feature	Value			Calculation time (s)		
	Minimum	Mean	Maximum	Minimum	Mean	Maximum
class-entropy	0.64	1.92	4.70	0.00	0.00	0.00
class-probability-max	0.04	0.43	0.90	0.00	0.00	0.00
class-probability-mean	0.04	0.28	0.50	0.00	0.00	0.00
class-probability-min	0.00	0.19	0.48	0.00	0.00	0.00
class-probability-std	0.00	0.10	0.35	0.00	0.00	0.00
dataset-ratio	0.00	0.06	0.62	0.00	0.00	0.00
inverse-dataset-ratio	1.62	141.90	1620.00	0.00	0.00	0.00
kurtosis-max	-1.30	193.43	4812.49	0.00	0.01	0.05
kurtosis-mean	-1.30	24.32	652.23	0.00	0.01	0.05
kurtosis-min	-3.00	-0.59	5.25	0.00	0.01	0.05
kurtosis-std	0.00	48.83	1402.86	0.00	0.01	0.05
landmark-1NN*	0.20	0.79	1.00	0.01	0.61	8.97
landmark-decision-node-learner*	0.07	0.55	0.96	0.00	0.13	1.34
landmark-decision-tree*	0.20	0.78	1.00	0.00	0.49	5.23
landmark-lda*	0.26	0.79	1.00	0.00	1.39	70.08
landmark-naive-bayes*	0.10	0.68	0.97	0.00	0.06	1.05
landmark-random-node-learner*	0.07	0.47	0.91	0.00	0.02	0.26
log-dataset-ratio	-7.39	-3.80	-0.48	0.00	0.00	0.00
log-inverse-dataset-ratio	0.48	3.80	7.39	0.00	0.00	0.00
log-number-of-features	1.10	2.92	5.63	0.00	0.00	0.00
log-number-of-instances	4.04	6.72	9.90	0.00	0.00	0.00
number-of-Instances-with-missing-values	0.00	96.00	2480.00	0.00	0.00	0.01
number-of-categorical-features	0.00	13.25	240.00	0.00	0.00	0.00
number-of-classes	2.00	6.58	28.00	0.00	0.00	0.00
number-of-features	3.00	33.91	279.00	0.00	0.00	0.00
number-of-features-with-missing-values	0.00	3.54	34.00	0.00	0.00	0.00
number-of-instances	57.00	2126.33	20000.00	0.00	0.00	0.00
number-of-missing-values	0.00	549.49	22175.00	0.00	0.00	0.00
number-of-numeric-features	0.00	20.67	216.00	0.00	0.00	0.00
pca-95percent*	0.02	0.52	1.00	0.00	0.00	0.00
pca-kurtosis-first-pc*	-2.00	13.38	730.92	0.00	0.00	0.01
pca-skewness-first-pc*	-27.07	-0.16	6.46	0.00	0.00	0.04
percentage-of-Instances-with-missing-values	0.00	0.14	1.00	0.00	0.00	0.00
percentage-of-features-with-missing-values	0.00	0.16	1.00	0.00	0.00	0.00
percentage-of-missing-values	0.00	0.03	0.65	0.00	0.00	0.00
ratio-categorical-to-numerical	0.00	1.35	33.00	0.00	0.00	0.00
ratio-numerical-to-categorical	0.00	0.49	7.00	0.00	0.00	0.00
skewness-max	0.00	5.34	67.41	0.00	0.00	0.04
skewness-mean	-0.56	1.27	14.71	0.00	0.00	0.04
skewness-min	-21.19	-0.62	1.59	0.00	0.00	0.04
skewness-std	0.00	1.60	18.89	0.00	0.01	0.05
symbols-max	0.00	13.09	429.00	0.00	0.00	0.00
symbols-mean	0.00	3.01	41.38	0.00	0.00	0.00
symbols-min	0.00	1.44	12.00	0.00	0.00	0.00
symbols-std	0.00	3.06	107.21	0.00	0.00	0.00
symbols-sum	0.00	71.04	1648.00	0.00	0.00	0.00

Figura 3.5: Meta-features utilizadas por auto-SKlearn [4]

3.3.2 Pipelines

En auto-SKlearn, cada iteración del proceso de optimización bayesiana genera una pipeline. Cada pipeline está compuesta por preprocesadores de datos, un preprocesador de atributos y un clasificador. Las distintas opciones que pueden ser seleccionadas y sus hiperparámetros, separados en hiperparámetros categóricos y continuos, se pueden ver en la figura 3.6. Todo esto genera un espacio de 110 hiperparámetros.

name	# λ	cat (cond)	cont (cond)	name	# λ	cat (cond)	cont (cond)
AdaBoost (AB)	4	1 (-)	3 (-)	densifier	-	-	-
Bernoulli naïve Bayes	2	1 (-)	1 (-)	extreml. rand. trees prepr.	5	2 (-)	3 (-)
decision tree (DT)	4	1 (-)	3 (-)	kernel PCA	5	1 (-)	4 (3)
extreml. rand. trees	5	2 (-)	3 (-)	rand. kitchen sinks	2	-	2 (-)
Gaussian naïve Bayes	-	-	-	linear SVM prepr.	3	1 (-)	2 (-)
gradient boosting (GB)	6	-	6 (-)	no preprocessing	-	-	-
kNN	3	2 (-)	1 (-)	nystroem sampler	5	1 (-)	4 (3)
LDA	4	1 (-)	3 (1)	random trees embed.	4	-	4 (-)
linear SVM	4	2 (-)	2 (-)	select percentile	2	1 (-)	1 (-)
kernel SVM	7	2 (-)	5 (2)	select rates	3	2 (-)	1 (-)
multinomial naïve Bayes	2	1 (-)	1 (-)	truncated SVD	1	-	1 (-)
passive aggressive	3	1 (-)	2 (-)	one-out-of-k encoding	2	1 (-)	1 (1)
QDA	2	-	2 (-)	imputation	1	1 (-)	-
random forest (RF)	5	2 (-)	3 (-)	balancing	1	1 (-)	-
SGD	10	4 (-)	6 (3)	rescaling	1	1 (-)	-

Figura 3.6: Clasificadores (izquierda), preprocesadores (derecha) y sus hiperparámetros categóricos y continuos[4]

3.3.3 Construcción del conjunto de pipelines

Para construir el conjunto final de pipelines, auto-SKlearn sigue el siguiente algoritmo.

- Primero, crea un conjunto vacío.
- Luego, añade al conjunto, la pipeline que maximice el desempeño del conjunto, que es el puntaje obtenido según la función de evaluación.
- Repite el paso anterior hasta que haya 50 pipelines en el conjunto, permitiendo repeticiones.

El conjunto de pipelines permite repeticiones, esto es para darle mayor peso o importancia a algunas pipelines sobre otras, dando más valor a los resultados obtenidos por las pipelines que se encuentran repetidas más veces dentro del conjunto.

3.4 Auto-SKlearn 2.0

Auto-SKlearn 2.0 propone algunas modificaciones a auto-SKlearn, con el propósito de volverlo un proceso totalmente automático [35]. Se descarta la idea de las *meta-features* reemplazándolas por un portafolio fijo de pipelines complementarias que cubren tantos datasets distintos como es posible. Además se incluye Successive Halving [36] como algoritmo de asignación de presupuesto para administrar el tiempo.

3.4.1 Construcción del portafolio

Teniendo un set de datasets D , se ejecuta optimización bayesiana sobre cada uno de ellos para obtener la mejor pipeline para cada dataset, lo que nos da las pipelines candidatas C . Luego se ejecuta cada dataset de D en cada pipeline de C para obtener una matriz de resultados. Esta matriz de resultados se usará para obtener la información necesaria para construir el portafolio. Se inicializa el portafolio P como un set vacío. Luego, iterativamente, se agrega al portafolio P , la pipeline de C que reduzca más el error de generalización estimado de P sobre todos los datasets de D y se quita esa pipeline de C . Se itera hasta que P llegue a un tamaño predeterminado. El error de generalización estimado de P sobre un dataset en D es el mejor desempeño que tenga una pipeline en P sobre ese dataset [35].

Auto-SKlearn requiere de muchas iteraciones de su algoritmo para encontrar la solución. Esto toma mucho tiempo. Quisiéramos encontrar una buena solución en menor tiempo. Nos interesa, por lo tanto, encontrar una forma de caracterizar las instancias de clasificación binaria de forma que una red neuronal convolucional pueda extraer información suficiente de ellas para optimizar el puntaje de sus resultados según la métrica de evaluación.

Capítulo 4

SELECCIÓN DE ALGORITMOS PARA CLASIFICACIÓN BINARIA BASADA EN DEEP LEARNING

En este capítulo, proponemos un modelo para la selección automática de algoritmos de instancias de problemas de clasificación binaria al que llamaremos Aproximate-PCA Based Selector (APBS). Primero, discutimos las instancias de problemas que utilizamos y el proceso para construir el portafolio de algoritmos de clasificación binaria. Luego, explicamos la caracterización de las instancias de problemas de clasificación binaria y, finalmente, revisamos la red neuronal convolucional utilizada.

4.1 Generación de Datos

Generamos instancias de clasificación binaria utilizando la herramienta `datasets.make_classification` de scikit-learn. Para crear diferentes tipos de instancias, los parámetros del generador se aleatorizaron para cada instancia de problema generado, como se muestra en la Tabla 4.1. Con este método, obtuvimos 5995 instancias de problemas de clasificación binaria, de las cuales 4800 se utilizaron para entrenamiento y 1195 para pruebas.

Parámetro	Valor
Número de muestras	$X \sim \mathcal{N}(\mu = 4000, \sigma = 12000)$, min = 100.
Número de características	$X \sim \mathcal{N}(\mu = 25, \sigma = 650)$, min = 4.
Porcentaje de características informativas	$X \sim \mathcal{N}(\mu = 8, \sigma = 30)$, min = 1, max = 100.
Número de características redundantes	$X \sim \mathcal{U}(\text{min} = 0)$
Número de clústeres por clase	$X \sim \mathcal{N}(\mu = 1, \sigma = 3)$, min = 1, max = 10.
Pesos	$X \sim \mathcal{N}(\mu = 50, \sigma = 20)$, min = 0, max = 100.
Separación de clases	$X \sim \mathcal{U}\{0.25, 0.5, 1, 2, 4\}$
Hipercubo	$X \sim \mathcal{U}\{\text{True}, \text{False}\}$

Tabla 4.1: Parámetros de Generación de Datos

4.2 Construcción del Portafolio de Algoritmos

Nuestro modelo selecciona, para cada instancia de clasificación binaria, un algoritmo de un portafolio de algoritmos que construimos. Para construir este portafolio, buscamos elegir algoritmos que no solo produzcan buenos resultados individualmente, sino que también se complementen entre sí. Para seleccionar los algoritmos considerados por nuestro modelo, utilizamos el siguiente proceso inspirado en la construcción del portafolio de auto-SKlearn 2.0 [35]:

- Primero, ejecutamos la primera versión de auto-SKlearn, creando conjuntos de pipelines de tamaño 1, con cada instancia de entrenamiento. Así, obtuvimos un pipeline de auto-SKlearn para cada instancia.
- A continuación, eliminamos los pipelines duplicados. Nos referimos a este conjunto de pipelines candidatos como C .
- Sea P el portafolio de algoritmos considerado por nuestro modelo. Agregamos a P el pipeline en C que más mejora el rendimiento general de P en las instancias de entrenamiento. El rendimiento general de P en las instancias de entrenamiento es la suma del mejor rendimiento alcanzado por cualquier algoritmo en P en cada instancia.
- Eliminamos el pipeline agregado de C .
- Finalmente, repetimos los últimos 2 pasos hasta que P alcance un tamaño predefinido. El tamaño de P determina el número de clases con las que trabaja el modelo selector. Trabajamos con un portafolio de 5 algoritmos de clasificación binaria.

4.3 Caracterización de las instancias

Para caracterizar las instancias decidimos utilizar el método de la matriz de similitudes rala [3]. Sin embargo, convertir esta matriz en imagen no es trivial y pensamos que el paso anterior a la construcción de la matriz rala, puede aportar también bastante información. Es por esto que queremos usar el espacio de `approximatePCA` dividido en intervalos como caracterización.

Antes que todo, preprocesamos la instancia de forma que sea más fácil obtener información de su caracterización. Entrenamos `random forest`, la versión implementada en `scikit-learn`, con los datos de entrenamiento, para obtener el atributo `feature_importances_`. Así, obtenemos un valor por cada atributo, al que llamaremos su importancia. Luego, normalizamos los datos, haciendo que cada atributo tenga media 0 y desviación estándar 1. Finalmente, multiplicamos cada atributo por la raíz cuadrada de su importancia. Esto ayuda a la futura caracterización a identificar los atributos más importantes. Después de esto procedemos a caracterizar la instancia.

Para esto, primero realizamos `approximatePCA` en la instancia para reducir sus dimensiones a 3. Esto es, seleccionar aleatoriamente un subconjunto de datos y atributos a considerar para la transformación de PCA y luego aplicar esta transformación a la totalidad de la instancia, obteniendo así cada dato de la instancia de clasificación binaria, representado en 3 dimensiones. Luego, dividimos sus dos primeras dimensiones en 256 intervalos y la tercera, en 3. Obtenemos entonces un espacio de $256 \times 256 \times 3$, en el cual inicializamos cada bloque en 0. Después, por cada dato de la instancia, aumentamos el valor del bloque al que pertenece en 1. Buscamos el mayor valor V entre todos los bloques y dividimos los valores de todos los bloques por V . Finalmente, tenemos 3 imágenes de 256×256 que forman parte de la caracterización de la instancia. Esto se representa como una imagen en RGB. En la figura 4.1 se ven las imágenes RGB correspondientes a 6 instancias diferentes.

Además de estas imágenes, hay 5 valores adicionales de una instancia que nos interesa incluir. Estos son: La cantidad de datos, la cantidad de atributos, la cantidad de datos etiquetados, la cantidad de datos etiquetados con una etiqueta positiva y el valor V por el cual todos los valores de las imágenes son divididos. Para incluirlos, generamos 5 matrices de 256×256 y las llenamos, cada una, con uno de los valores adicionales. La entrada de nuestro modelo es entonces de $256 \times 256 \times 8$.

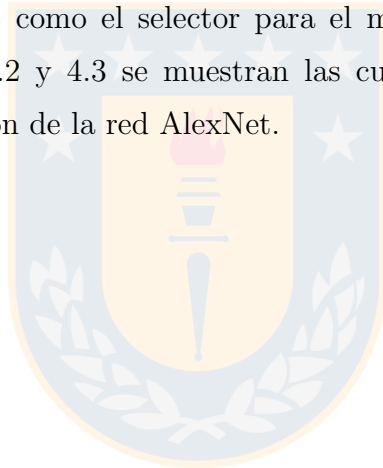
4.4 Red Neuronal Convolutacional

Probamos 4 diferentes redes neuronales convolucionales: AlexNet, Vgg16, ResNet50 y GoogLeNet. Las 4 fueron entrenadas con la función de pérdida categorical crossentropy, el optimizador Adam, un tamaño de batch de 16 y una división de validación del 10% de los datos de entrenamiento. Sus resultados se pueden ver en la tabla 4.2.

Red Neuronal Convolutacional	Accuracy
AlexNet [28]	0.389
Vgg16 [29]	0.104
ResNet50 [30]	0.356
GoogLeNet [31]	0.324

Tabla 4.2: Accuracy de las redes neuronales convolucionales probadas

Decidimos usar AlexNet como el selector para el modelo, ya que obtuvo la mayor accuracy. En las figuras 4.2 y 4.3 se muestran las curvas de accuracy y pérdida del entrenamiento y la validación de la red AlexNet.



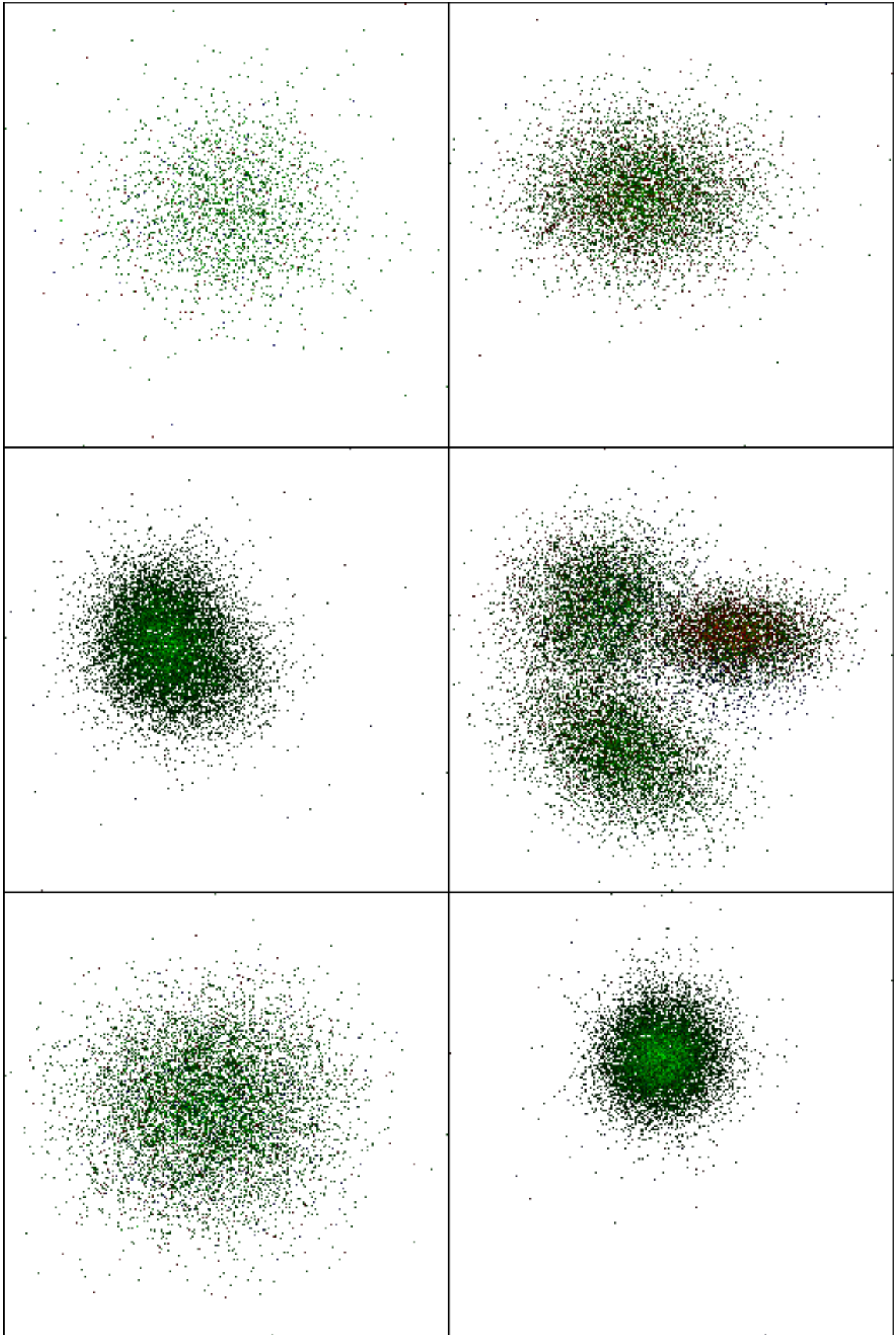


Figura 4.1: Imágenes correspondientes a 6 instancias de problemas de clasificación binaria

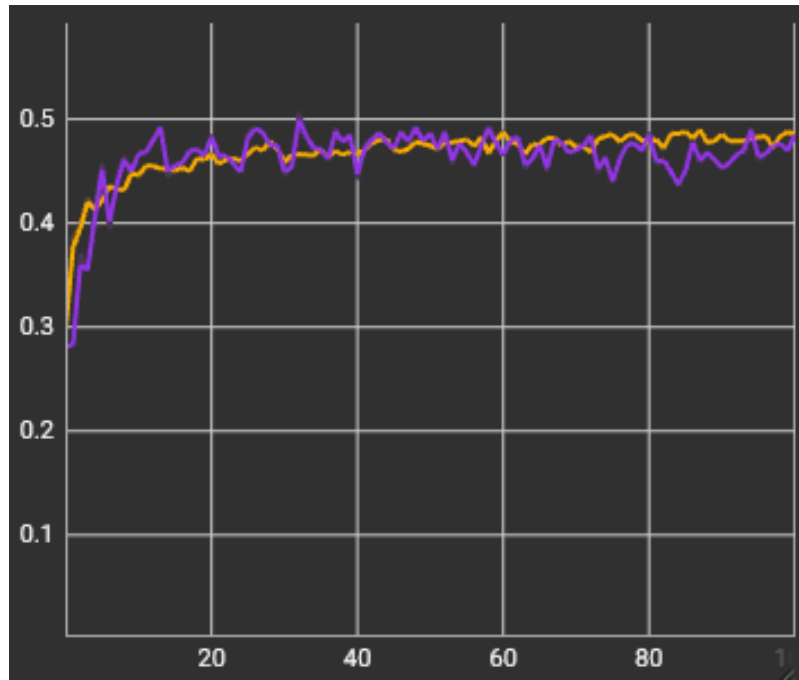


Figura 4.2: Accuracy del entrenamiento (naranja) y la validación (morado) de la red AlexNet

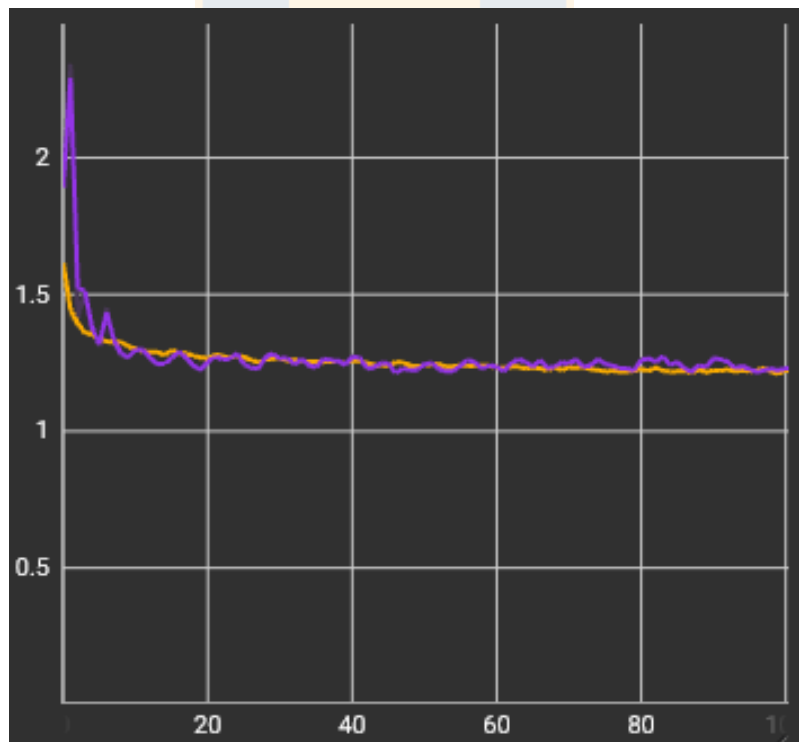


Figura 4.3: Pérdida durante el entrenamiento (naranja) y la validación (morado) de la red AlexNet

Capítulo 5

RESULTADOS

El modelo fue entrenado con 4800 instancias de problemas de clasificación binaria y luego probado con otras 1195. El portafolio de algoritmos contiene 5 pipelines, cuyos clasificadores son algoritmos de clasificación de la biblioteca scikit-learn. La tabla 5.1 contiene información de estas pipelines y sus resultados.

Pipeline	Clasificador	f1-score promedio entrenamiento	f1-score promedio prueba
0	SVC (C-Support Vector Classification)	0.839	0.819
1	QDA (Quadratic Discriminant Analysis)	0.794	0.703
2	MLP Classifier	0.683	0.701
3	Passive Aggressive Classifier	0.768	0.679
4	Gradient Boosting Classifier	0.767	0.683

Tabla 5.1: Portafolio de algoritmos

Las figuras 5.1 y 5.2 muestran cuantos datos pertenecen a cada clase en los conjuntos de entrenamiento y de prueba.

La matriz de confusión de los datos de prueba y el F1-score promedio de APBS se pueden ver en la figura 5.3 y en la tabla 5.2.

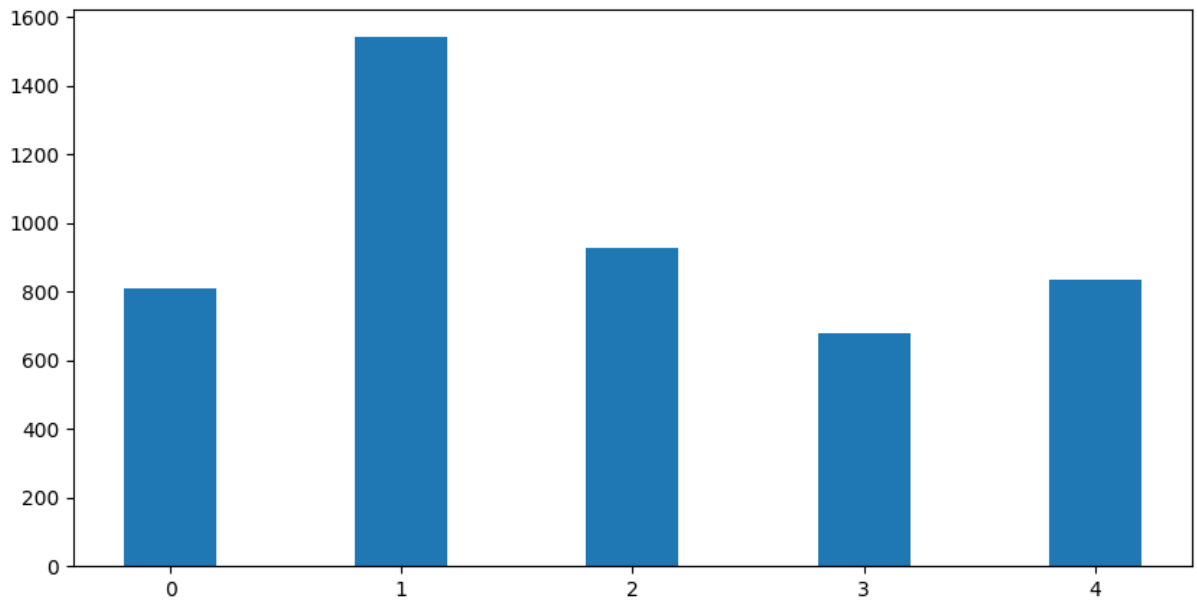


Figura 5.1: Clases de los datos de entrenamiento

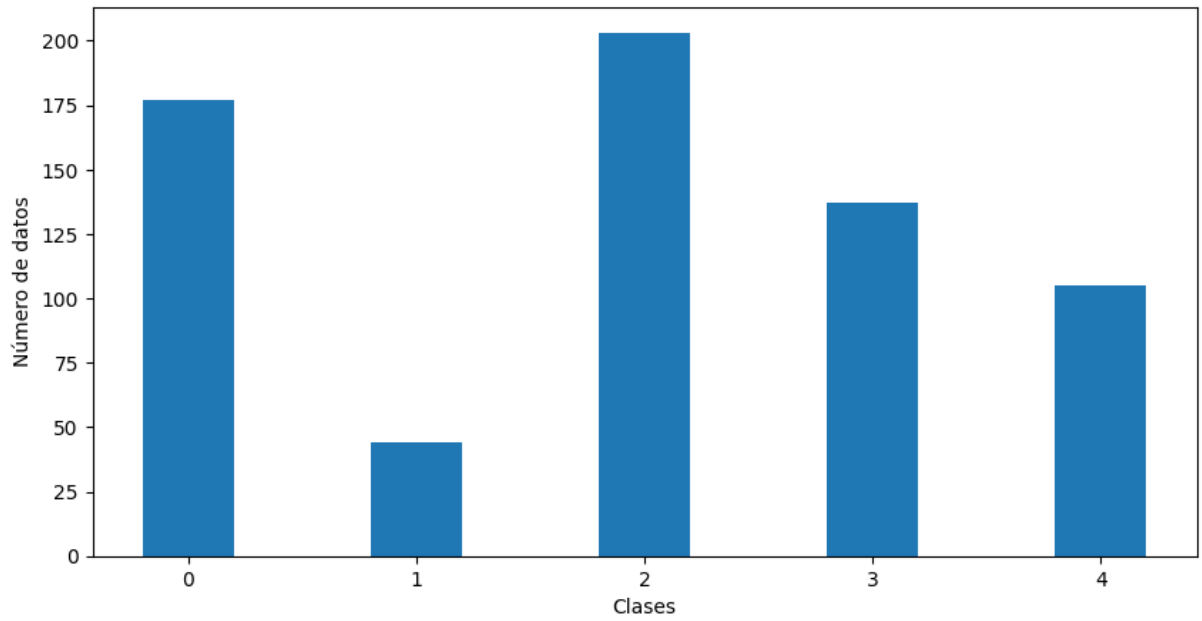
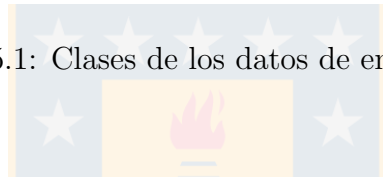


Figura 5.2: Clases de los datos de prueba

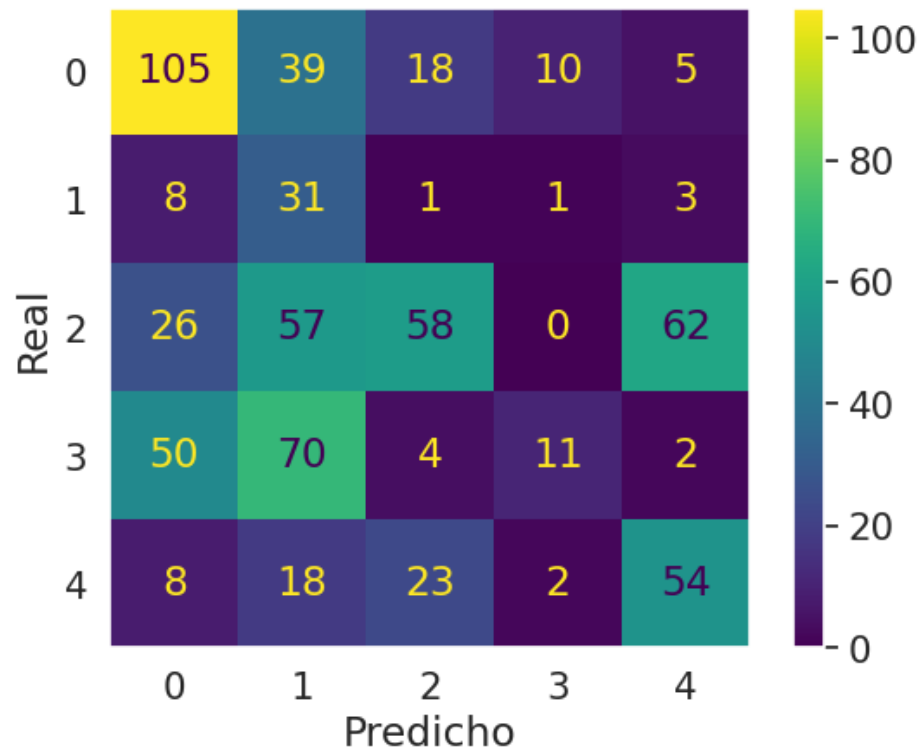


Figura 5.3: Matriz de confusión

Notamos que hay una gran diferencia en la proporción de los datos pertenecientes a la clase 1, en el conjunto de entrenamiento y en el de prueba. Esto explica que nuestro modelo haya clasificado múltiples datos en la clase 1 equivocadamente, sobre todo aquellos pertenecientes a las clases 2 y 3.

Modelo	F1-score promedio
SBS	0.819
VBS	0.877
APBS	0.828
Auto-SKlearn	0.408

Tabla 5.2: Comparación del F1-score promedio

Auto-SKlearn se probó con las mismas instancias y se le dio el tiempo suficiente para completar una iteración de su algoritmo, lo cual toma alrededor de 30 segundos. APBS obtuvo un F1-score promedio más alto que la primera iteración de Auto-SKlearn.

Además, la selección de un algoritmo con Alexnet tomó en promedio 4.535 segundos, y la caracterización de cada imagen tomó en promedio 1.421 segundos, así que APBS obtuvo sus resultados en menos de 6 segundos.

Al evaluar nuestros resultados en las instancias de prueba con la métrica \hat{m} , APBS obtuvo una puntuación de 0.845. Por lo tanto, su predicción es mejor que simplemente usar el SBS por sí solo. APBS también obtiene mejores resultados que Auto-SKlearn para tiempos cortos en los que solo se entrena un pipeline de clasificación binaria.



Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

Hemos creado un modelo de selección automática de algoritmos para el problema de clasificación binaria, al que llamamos Approximate-PCA Based Selector (APBS). Para ello, generamos instancias artificiales del problema de clasificación binaria y las caracterizamos como un espacio de $256 \times 256 \times 8$ bloques. Luego, construimos un portafolio de algoritmos, desde donde se seleccionan los algoritmos para cada instancia. Finalmente, elegimos Alexnet como nuestro selector y lo entrenamos y probamos con las instancias caracterizadas. Nuestros resultados muestran que APBS tiene un rendimiento superior a los pipelines individuales de clasificación binaria y al modelo del estado del arte Auto-SKlearn, dado un presupuesto corto de tiempo, de aproximadamente 30 segundos o menos. Esto confirma nuestra hipótesis. Usando una caracterización rápida y eficaz de las instancias de problemas de clasificación binaria pudimos proveer suficiente información a una red neuronal convolucional, permitiendo que nuestro modelo mejorara el estado del arte en selección automática de algoritmos para clasificación binaria, obteniendo mejores resultados que la primera iteración de Auto-SKlearn. La medición de la dispersión de la proyección en 3 dimensiones de los puntos que componen una instancia de clasificación binaria por medio de approximatePCA probó ser una caracterización rápida que entrega suficiente información a nuestra red neuronal convolucional. Además, pudimos utilizar AlexNet para explotar esta caracterización y así realizar selección automática de algoritmos para clasificación binaria en menos de 6 segundos. Este trabajo está limitado a conjuntos de datos artificiales, por lo que en el futuro quisieramos recolectar instancias de problemas de clasificación binaria reales, además de mejorar nuestro generador de instancias para que cree problemas de clasificación binaria mas parecidos a los problemas reales. Nos gustaría también mejorar la caracterización de las instancias para poder trabajar con problemas de clasificación binaria que contengan atributos categóricos. También estamos interesados en agregar un método de clasificación binaria no supervisado como una pipeline en nuestro portafolio y ver

cómo se desempeña nuestro modelo al entrenarlo y probarlo con instancias de problemas de clasificación binaria que contengan diferentes proporciones de muestras no etiquetadas.



Bibliografía

- [1] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.
- [2] Huerta I. I., Neira D. A., Ortega D. A., Varas V., Godoy J., and Asín-Achá R. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. In *Expert Systems with Applications*, volume 187, 2022.
- [3] D. S. Hochbaum and P. Baumann. Sparse computation for large-scale data mining. *IEEE Transactions on Big Data*, 2(2):151–174, 2016.
- [4] M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Proceedings of the Neural Information Processing Systems*, pages 2962—2970, 2015.
- [5] Vedant Bahel, Sofia Pillai, and Manit Malhotra. A comparative study on various binary classification algorithms and their improved variant for optimal performance. pages 495–498, 01 2020.
- [6] Gürol Canbek, Tugba Taskaya Temizel, and Şeref Sağiroğlu. Ptopi: A comprehensive review, analysis, and knowledge representation of binary classification performance measures/metrics. *Sn Computer Science*, 4, 2022.
- [7] Gürol Canbek, Tugba Taskaya Temizel, and Şeref Sağiroğlu. Benchmetrics: a systematic benchmarking method for binary classification performance metrics. *Neural Computing and Applications*, 33:14623 – 14650, 2021.
- [8] Neela Rayavarapu Shilpa Hudnurkar. Binary classification of rainfall time-series using machine learning algorithms. In *International Journal of Electrical and Computer Engineering (IJECE)*, pages 1945–1954, 2022.
- [9] Bao Zhang Zhongguo Wang. Toxic comment classification based on bidirectional gated recurrent unit and convolutional neural network. In *ACM Transactions on Asian and Low-Resource Language Information Processing*, pages 1–12, 2022.
- [10] Prasenjit Das Chetan Sharma Shankar Shambhu, Deepika Koundal. Binary classification of covid-19 ct images using cnn. In *International Journal of E-Health and Medical Communications*, pages 1–13, 2022.
- [11] Wei Wang, Lei Feng, Yuchen Jiang, Gang Niu, Min-Ling Zhang, and Masashi Sugiyama. Binary classification with confidence difference, 2023.

- [12] A. D. Jesus, A. Liefvooghe, B. Derbel, and L. Paquete. Algorithm selection of anytime algorithms. In *GECCO '20: Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 850–858, 2020.
- [13] Qingliang Zeng Diya Shang, Hua Sun. A reinforcement-algorithm framework for automatic model selection. In *IOP Conference Series Earth and Environmental Science*, 2020.
- [14] Abdullah Mueen Cynthia Freeman, Ian Beaver. Improving univariate time series anomaly detection through automatic algorithm selection and human-in-the-loop false-positive removal. In *The International FLAIRS Conference Proceedings*, 2021.
- [15] M. Girish Chandra Karumanchi Ravikumar Naveen Kumar Thokala, Kriti Kumar. Long-term forecasting of heterogenous variables with automatic algorithm selection. In *Advances in Computational Intelligence - Lecture Notes in Computer Science*, pages 186–197, 2019.
- [16] Sim K. Hart E. Alissa, M. Automated algorithm selection: from feature-based to feature-free approaches. In *J Heuristics 29*, pages 1–38, 2023.
- [17] Raphael Patrick Prager. Improving automated algorithm selection by advancing fitness landscape analysis, 2023.
- [18] Gürol Canbek, Seref Sagiroglu, Tugba Taskaya Temizel, and Nazife Baykal. Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights. pages 821–826, 10 2017.
- [19] Somesh Das Gupta. Discriminant analysis. In Stephen E. Fienberg and David V. Hinkley, editors, *R.A. Fisher: An Appreciation*, pages 161–170, New York, NY, 1980. Springer New York.
- [20] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [21] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, dec 2006.
- [22] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [23] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. volume 3408, pages 345–359, 04 2005.
- [24] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

- [25] Lindauer M., Van Rijn J. N., and Kotthoff L. The algorithm selection competitions 2015 and 2017. In *Artificial Intelligence*, pages 86–100, 2019.
- [26] Jonathon Shlens. A tutorial on principal component analysis, 2014.
- [27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.
- [29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.
- [30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [32] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). In *Proc. of LION'11*, pages 507—523, 2011.
- [33] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. Analysis of the automl challenge series 2015–2018. *Automated Machine Learning*, pages 177–219, 2019.
- [34] Matthias Feurer, Jost Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), 2015.
- [35] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. 2021.
- [36] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F Hutter. Practical automated machine learning for the automl challenge 2018. In *ICML 2018 AutoML Workshop*, 2018.