Optimización automática de hiperparámetros en la creación de modelos para un problema de minería de datos

Daniel Sebastián Ochoa Urrego

24 de noviembre de 2023

Resumen

In this article, we will be creating an algorithm to automatize the preprocessing and modeling stages of the CRISP-DM methodology. To do so we will use the Grid Search algorithm to tune the hyperparameters of the Random Forest, KNN, SVM, Bayesian and ANN models, while also automatically imputing the data in search of the best model possible basing the results in the accuracy score of the models. Here we will find that while this can in deed improve the model, in most cases we won't have an improvement bigger than a couple of percentage points.

Keywords— Random Forest, KNN, SVM, Bayesian models, ANN, Preprocessing, Modeling

1. Introducción

El problema concreto que trataremos en este proyecto se basará en el data set "census", en este se nos presenta con información básica de residentes de Estados Unidos como la edad, la educación, estado civil, entre otros, y con estos debemos predecir si las personas ganaran más o menos de 50,000 dólares al año, usando modelos basados en algoritmos de Bosque aleatorio, KNN, SVM, Clasificación Bayesiana y ANN.

Durante todo el curso, el profesor nos ha comentado como a la hora de hacer minería de datos no hay una respuesta definitiva sobre como se deberían hacer los procesos de esta. Ya que todos los problemas en este ámbito son únicos, jamás tendremos una respuesta certera sobre como abordarlos. Por ejemplo, no hay una respuesta sobre si los datos deberían ser normalizados o discretizados a la hora de hacer el preprocesamiento de los datos.

Otro momento donde problemas como estos aparecen es en la etapa de modelado, cuando ya tenemos nuestros datos tratados surgen las preguntas ¿Qué modelo deberíamos usar? ¿Cuáles son los mejores hiperparámetros que le podemos pasar a este modelo para que tenga la mejor precisión posible? Ya que no hay respuesta definitiva para estas preguntas, y (hasta donde entendí) no hay

una manera de ver que se debería usar más allá de prueba y error, pensé que la mejor manera de encontrar ese caso ideal sería simplemente automatizar esa prueba y error hasta encontrar la mejor configuración posible del modelo.

De hecho, investigando para la realización de este artículo, descubrí que es una práctica muy común el intentar optimizar los modelos variando los hiperparámetros. Los algoritmos más comunes que cubre este artículo incluyen el BabySitting, GridSearch, Random Search y Gradient-Based Optimization.[2]

2. Estado del Arte

Esta sección se dividirá en 3, en la primera explicaremos cada algoritmo de optimización encontrado, en la segunda hablaremos sobre los hiperparámetros que se consideran más importantes en cada modelo y en la tercera hablaremos sobre trabajo que ya se ha hecho sobre el data set usado.

2.1. Algoritmos de Optimización

Al momento de iniciar este proyecto, tenía 2 ideas para desarrollar la mejora de hiperparámetros, pensé en que podría usar un algoritmo de fuerza bruta o uno más inteligente que buscara mejorar la precisión maximizándola de manera similar a como se minimiza una función de costo en redes neuronales. Al investigar (Y para sorpresa de nadie) descubrí que dichos algoritmos ya existen.

Los algoritmos de optimización los dividen en 2, los que usan y no usan otro modelo. Su diferencia está en el enunciado, para mejorar la selección de hiperparámetros podríamos usar otro modelo cuyo trabajo sea buscar ese conjunto óptimo de parámetros. O podríamos usar otro tipo de algoritmos como el BabySitting, GridSearch, Random Search y Gradient-Based Optimization.[2]

2.1.1. BabySitting

Este es el método que todos hemos hecho durante el curso hasta ahora, consiste en cambiar manualmente los hiperparámetros hasta encontrar el modelo que más lo satisfaga a uno o hasta llegar a la fecha de entrega. Inicialmente, este tiene muchos problemas, ya que consume mucho tiempo, por lo que muchas veces se tienen demasiados hiperparámetros, o se tienen relaciones no lineales entre la precisión y los hiperparámetros. Por lo que con base en estos problemas es que se empezaron a buscar mejores alternativas. [2]

2.1.2. GridSearch

Este algoritmo en resumidas cuentas es la aceración al problema por fuerza bruta, consiste en definir un rango de valores para cada hiperparámetro y entrenar el modelo con todas las combinaciones posibles dentro de los rangos dados. Dentro de sus puntos buenos tiene que es muy probable encontrar el mejor modelo, pero es computacionalmente costoso.[2]

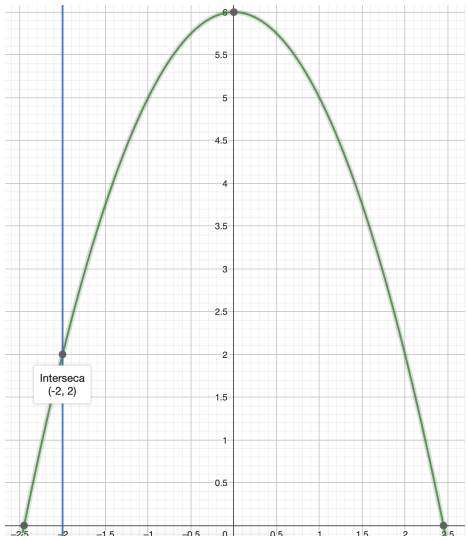

Figura 1: Función de precisión hipotética de un modelo con 1 hiperparámetro

2.1.3. Random Search

Este algoritmo nació como una propuesta de mejora a GridSearch. Nace de la idea que si entrenamos modelos con hiperparámetros al azar, luego de suficiente tiempo es probable que encontremos un modelo lo suficientemente cerca al mejor modelo que podríamos encontrar con GridSearch y sin gastar tantos recursos. [2]

2.1.4. Gradient-Based Optimization

Este algoritmo intenta buscar de una manera más inteligente ese punto óptimo, de una manera muy resumida, lo que hace es intentar maximizar esa función de precisión, variando un poco los parámetros y yendo en la dirección que mejora la precisión.[2]

Por ejemplo, asumiendo que tenemos un modelo que solo tiene un hiperparámetro, y que la función de precisión es la de la Figura 1, lo que haríamos al seguir este algoritmo sería lo siguiente

1. Escoger un valor al azar del hiperparámetro como valor inicial (En la figura se escoge -2)

2. Calcular el valor de la precisión en puntos cercanos a -2 por la izquierda y la derecha

3. Cambiar el punto inicial por el valor que mayor precisión haya tenido, en el ejemplo sería un punto a la derecha de -2

4. Repetir los pasos 2 y 3 hasta que los puntos cercanos calculados sean menores que el punto central, se llegaría a este punto cuando el punto central sea 0

Y para mayores dimensiones (Más hiperparámetros) se usa este mismo concepto, pero calculando la gradiente entre el punto central y los puntos adyacentes, y moviéndose hacia donde menor sea este valor.[2]

Ahora, como los anteriores algoritmos, este tiene sus pro y sus contra. el algoritmo es más rápido en encontrar el valor óptimo más cercano al punto inicial, pero también tiene sus desventajas, y es que si la función de presión no es convexa, es muy posible que solo se llegue a un máximo local, y no al máximo absoluto, además, de que este funciona solo para hiperparámetros que son continuos, puesto que con variables categorías no se podría calcular este gradiente.[2]

Luego de revisar estos algoritmos, llegue a la conclusión que el mejor enfoque sería usar un GridSearch. Babysitting quedo descartado, porque el enfoque que tome en este proyecto era automatizar esas modificaciones manuales que hicimos durante el curso. Random Search quedo descartado, porque creo que este solo es funcional cuando se entrenan los modelos las suficientes veces como para encontrar ese máximo, y como veremos más adelante, los rangos de hiperparámetros que tomaríamos no serían tan grandes como para justificar el uso de este modelo. Y por último Gradient-Based Optimization quedo descartado, porque muchos de los modelos que debíamos usar de Sklearn tenían como valores categóricos de hiperparámetros.

2.2. Hiperparámetros a optimizar por cada modelo

Ahora, luego de haber seleccionado GridSearch, nace un problema, varios de los modelos que usaremos de SKLearn tienen muchos hiperparámetros con los que jugar, por lo que si variamos todos ellos, el tiempo de búsqueda sería demasiado grande, por lo que para solucionar este problema, se buscó cuáles eran los hiperparámetros de cada modelo que más afectaban su precisión

2.2.1. Random Forest

Según lo visto en [4] y [5] se encontró que los hiperparámetros que más tienen un impacto en la precisión del modelo son el número de árboles, la profundidad máxima de los árboles, el número máximo de hojas permitidas de los árboles y el criterio de división de los datos.

2.2.2. KNN

Por obvias razones, uno de los hiperparámetros a variar será el número de vecinos a tomar en cuenta, y con base en [3] y [4] decidí que los otros hiperparámetros que se variarán será el tipo de distancia entre los vectores, y si la distancia con los vecinos tendrá un peso a la hora de tomar la decisión sobre la clase.

2.2.3. SVM

Con base en [4] y [6] decidí que los hiperparámetros que variaría serían el parámetro de regularización C y el coeficiente del kernel Gamma.

Además, también tenía pensado variar el kernel que usara el modelo, pues como nos dicen en [2] estos impactan directamente en el rendimiento, pero el hacer esto hacía que el tiempo de búsqueda del GridSearch fuera demasiado grande, por lo que por esto y porque como veremos más adelante el desempeño de este modelo no era el mejor comparado con los otros, decidí solo usar los hiperparámetros mencionados en el párrafo anterior.

2.2.4. Bayesian Clasifier

En SKLearn tienen implementados varios tipos de clasificadores Bayesianos, y ninguno tiene muchos hiperparámetros, además el tiempo de entrenamiento de este tipo de modelos era muy bueno, por lo que para estos modelos se entrenaron todos los tipos de modelos, variando todos los hiperparámetros de cada uno que afectaban al rendimiento.

2.2.5. ANN

Para este tipo de modelos no encontré artículos que mencionaran cuáles eran los hiperparámetros más importantes, por lo que pregunte a estudiantes de la especialidad en Inteligencia Artificial en la Escuela, y me dijeron que debía variar la función de activación, el máximo número de iteraciones permitido para ajustar los pesos, la tolerancia y el número máximo de rondas sin cambio de para definir la conversión de los pesos y la función que optimizará los pesos cada ronda

2.3. Trabajo sobre Census

Luego de tener definido que haremos en el proyecto, hay que revisar que trabajo previo se ha hecho sobre el data set en concreto en el que trabajaremos. Según lo encontrado, [1] fue el mejor modelo que se ha podido implementar hasta el momento, ellos usaron el modelo "Gradient Boosting Classifier", mejorando los hiperparámetros con GridSearch y obtuvieron una precisión de un 88 %.

Aunque fue un muy buen modelo, para este proyecto no podíamos usarlo, pues estamos limitados a usar los modelos mencionados en la introducción.

3. Descripción del problema y algoritmo

3.1. Tarea

En este trabajo se nos dio un objetivo claro, crear el mejor modelo posible para clasificar atributos de un data set sobre un censo realizado en Estados Unidos hace un tiempo. La manera en la que decidí abordar el problema, fue

```
criterion = ["gini", "entropy", "log_loss"]
depths = [None, 5, 10, 15, 20]
max_nodes = [None, 6, 7, 8, 9, 10, 13, 15]
n_arboles = [150, 200, 300, 400]
param_grid = ParameterGrid({'n_estimators': n_arboles,
                            'max_depth': depths,
                            'criterion': criterion,
                            'max_leaf_nodes': max_nodes})
```

Figura 2: Ejemplo de la creación de la grilla de parámetros para modelos Random Forest

automatizando la creación de los modelos variando los hiperparámetros, para luego de probarlos, y así encontrar el mejor modelo posible

3.2. Algoritmo

Para realizar la automatización se decidió ir por el camino del GridSearch, ya que, creo que era de los caminos más sencillos y funcionales que podía tomar para lograr ese modelo ideal.

Para hacerlo se usó la clase "ParameterGrid", en esta lo único que hacemos es crear un objeto de este tipo, y al constructor le pasamos un diccionario donde cada llave es el nombre de un parámetro del modelo y sus valores son una lista con los posibles valores con los que queremos probar nuestros modelos.

Luego de tener el objeto creado, lo único que debemos hacer es iterar sobre este con un ciclo for y así obtendremos en cada iteración una combinación de los parámetros pasados en la inicialización de la grilla.

Y por último, para tomar en cuenta diferentes preprocesamientos de datos en la construcción de los modelos, también se implementó un ciclo for donde hacemos una imputación de los datos por KNN con 3, 5 y 10 vecinos.

Además, durante el entrenamiento de cada modelo, también hacemos que prediga con los datos de prueba para descubrir su puntaje de precisión, esto para luego guardar la configuración del modelo junto con su puntaje y luego buscar con qué hiperparámetros se consiguió el mejor modelo.

Por lo que a grandes rasgos el algoritmo quedaría de la siguiente manera

```
for k in knn_neighbors: # Donde knn_neighbors es una lista con los
    posibles vecinos
    df = impute_knn(df, k)
    for params in param_grid:
        model = Modelo(**params)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
```

Atributo	Descripción
Id	Identificador del registro
Age	Edad de la persona
Workclass	Área profesional general donde se encuentra el trabajo de la persona
Fnlwgt	(Final Weight) El número de personas que el censo cree que representa los valores de la entrada
Education	Tipo de educación que consiguió la persona
Education-Num	Grado de educación de la persona
Marital-Status	Estado civil de la persona
Occupation	Área profesional específica donde se encuentra el trabajo de la persona
Relationship	Tipo de relación que tiene la persona
Race	Etnia de la persona
Sex	Sexo de la persona
Capital-Gain	Dinero entrante mensual de la persona
Capital-Loss	Dinero saliente mensual de la persona
Hours-Per-Week	Horas trabajadas por semana
Native-Country	País de origen
Income	Determina si la persona gana más o menos de 50.000 dólares al año

Cuadro 1: Descripción de datos en census

```
ac = acuraccy(y_pred, y_test)

save(params, ac) # Guardamos a conficuracion del modelo y su
    exactitud para al final buscar el mejor
```

4. Evaluación experimental

4.1. Datos

Para el proyecto usaremos un data set bien conocido, "census", en este tenemos información obtenida en un censo realizado en los Estados Unidos. Se compone de 16 atributos explicados en el Cuadro 1.

4.2. Metodología

Para el desarrollo de este proyecto se usó la metodología CRISP-DM, lo siguiente fue lo que se realizó en cada paso

- Entendimiento del negocio: Ya que este es un proyecto académico diría que no hay un negocio que entender, creo que lo que más se relacionaría con este paso en nuestro caso es el entendimiento del funcionamiento de la competencia en Kaggle.
- Entendimiento de los datos: Para este paso diría que lo que más hice fue el buscar información sobre los datos en diferentes plataformas, además de realizar mi propio análisis exploratorio de los datos, con el fin de entender bien con que datos estaba trabajando.
- Preparación de los datos: En esta etapa lo único que se hizo fue hacer las imputaciones automatizadas explicadas arriba. No se usaron más estrategias de preprocesamiento, pues por el tema de la automatización resultaba

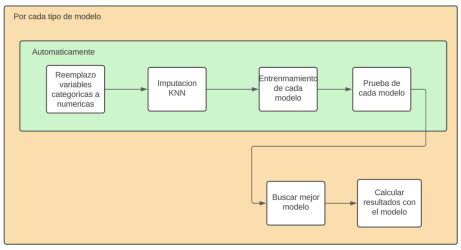

Figura 3: Flujode trabajo

muy pesado para la máquina en la que estábamos corriendo el algoritmo. Además, creo que sin estas otras técnicas, igualmente se obtuvieron resultados muy buenos en los modelos, aunque también cabe la posibilidad de que la precisión de estos mejorara al añadir otras técnicas.

- Modelado: Esta parte es en la que más nos enfocamos durante el desarrollo del proyecto, arriba tenemos toda la explicación de que se hizo para la automatización y optimización los diferentes modelos usados.
- Evaluación: Ya teniendo los modelos entrenados, se buscó cuál fue el mejor entre ellos basándose en el puntaje de precisión del modelo y con este se predecía la clase en el DataFrame de prueba.
- Despliegue: Por ahora no hicimos despliegue.

Y gráficamente el flujo del trabajo del proyecto podría verse representado como se ve en la Figura 3

4.3. Resultados

4.3.1. Random Forest

Al terminar el algoritmo de GridSearch graficamos la precisión de cada modelo y encontramos los resultados vistos en la Figura 4

En esta podemos ver que el mejor modelo tuvo una precisión de al rededor de un 85 % este se encontró cuando hacemos imputación por KNN con 10 vecinos y teníamos los siguientes hiperparámetros

- 300 árboles
- Perdida logarítmica como criterio de división de nodos
- 15 niveles como la profundidad máxima del árbol:

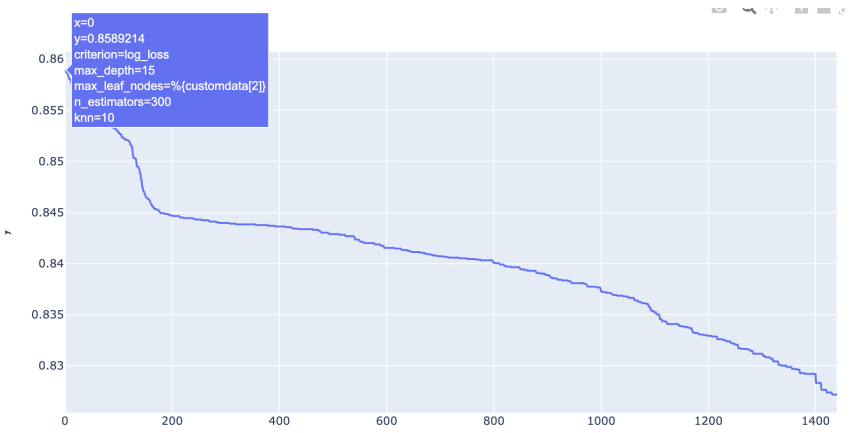

Figura 4: Rendimiento de modelos Random Forest

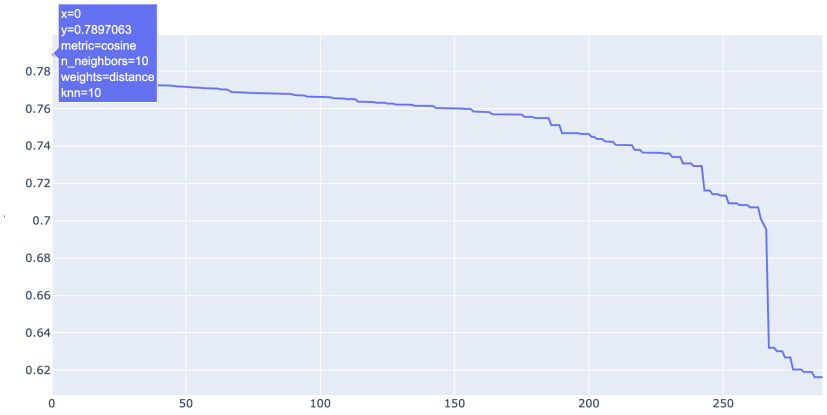

Figura 5: Rendimiento de modelos KNN

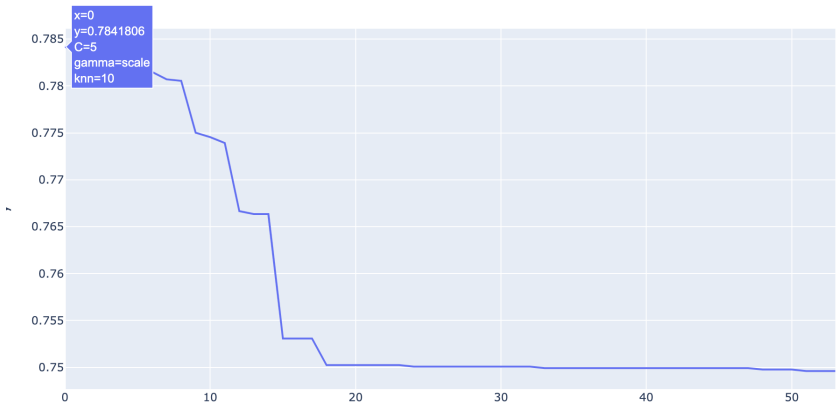

Figura 6: Rendimiento de modelos SVM

4.3.2. K Nearest Neighbors

Al terminar el algoritmo de GridSearch graficamos la precisión de cada modelo y encontramos los resultados vistos en la Figura 5

En esta podemos ver que el mejor modelo tuvo una precisión de al rededor de un 79 %. El modelo se encontró cuando hacemos imputación por KNN con 10 vecinos y teníamos los siguientes hiperparámetros

- Cálculo de las distancias de vectores con medida de Similitud de Coseno
- 10 vecinos a tomar en cuenta en el modelo:
- La cercanía del vecino tiene un peso sobre el cálculo de la clase

4.3.3. Support Vector Machine

Al terminar el algoritmo de GridSearch graficamos la precisión de cada modelo y encontramos los resultados vistos en la Figura 6

En esta podemos ver que el mejor modelo tuvo una precisión de al rededor de un 78 %. El modelo se encontró cuando hacemos imputación por KNN con 10 vecinos y teníamos los siguientes hiperparámetros

- Parámetro de regularización C igual a 5
- Coeficiente Gamma calculado automáticamente como el inverso del producto entre el número de atributos y la varianza de los datos de entrenamiento

4.3.4. Bayesian Classifier

Al terminar el algoritmo de GridSearch graficamos la precisión de cada modelo y encontramos los resultados vistos en la Figura 7

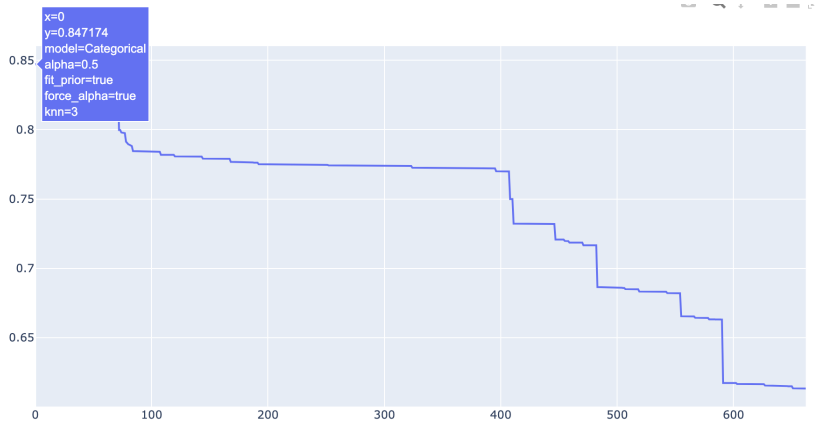

Figura 7: Rendimiento de modelos Bayesianos

En esta podemos ver que el mejor modelo tuvo una precisión de al rededor de un 85 %. El modelo se encontró cuando hacemos imputación por KNN con 3 vecinos y teníamos los siguientes hiperparámetros

- Usamos un modelo Bayesiano enfocado en predecir clases con variables categóricas en sus datos de entrenamiento
- Índice Alpha de 0.5
- Forzar Alpha a ser un número pequeño si es necesario
- Aprender la probabilidad de ocurrencia de clases anteriores

4.3.5. Artificial Neural Network

Al terminar el algoritmo de GridSearch graficamos la precisión de cada modelo y encontramos los resultados vistos en la Figura 8

En esta podemos ver que el mejor modelo tuvo una precisión de al rededor de un 78 %. El modelo se encontró cuando hacemos imputación por KNN con 3 vecinos y teníamos los siguientes hiperparámetros

- Tangente hiperbólica como función de activación
- 300 iteraciones máximas intentando mejorar el modelo
- Tolerancia de cambio igual a 0.001
- Máximo número de iteraciones donde no hay cambios y terminar el modelo igual a 15
- Función adam para optimizar los pesos del modelo

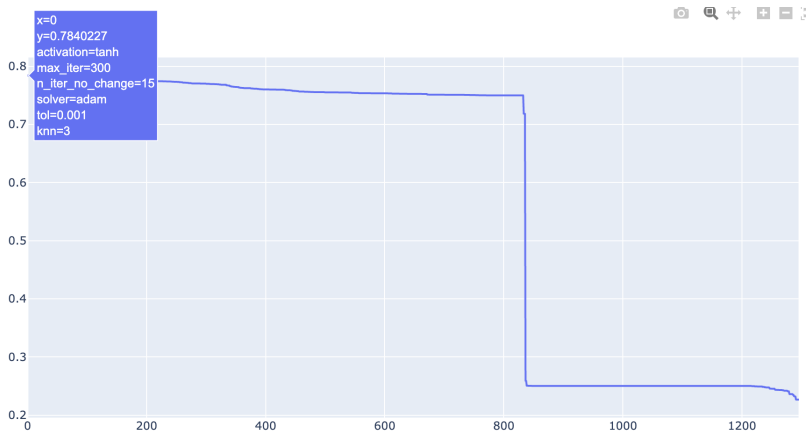

Figura 8: Rendimiento de modelos ANN

4.4. Discusión

4.4.1. Random Forest

El GridSearch tuvo mayor éxito con este modelo. El peor random forest tuvo una precisión del 82 %, lo cual es mucho más alto de varios de los modelos implementados más tarde, y el mejor de los modelos tuvo una precisión del 85 %, el máximo de precisión con el que llegaríamos en el artículo.

Además, al ver la Figura 4 también podemos observar un comportamiento particular al rededor de x = 190, en este punto hay un salto en el rendimiento de los modelos. Al analizar los resultados vi que era debido a la combinación de 2 hiperparámetros, cuando el máximo de hojas no se le especificaba al modelo, y cuando la profundidad máxima se le permitía crecer a más de 10 niveles.

En [5] vimos que este par de hiperparámetros eran de los que más afectaban el desempeño del modelo, y creo que podemos ver esa hipótesis comprobada en este trabajo.

4.4.2. K Nearest Neighbors

Este modelo fue de los que más varianza tuvo en la precisión al variar los hiperparámetros, pasamos de un modelo de 61 % a uno de 78 %. Aunque compararlo con otros modelos, este fue simplemente peor. Por lo que pese a que vería esto como un éxito a la hora de implementar el GridSearch por el salto de precisión, creo que el modelo de por sí no era el indicado para predecir información de este data set.

Además, al analizar la Figura 5, también podemos encontrar 2 comportamientos interesantes con el modelo

1. Los mejores modelos usaban la similitud del coseno como método para calcular la distancia

2. Los peores modelos, (Que tuvieron una precisión entre 61 % y 63 %) en su mayoría no les daba peso a las distancias entre vecinos y usaban distancias Manhattan.

4.4.3. Support vector Machine

En mi opinión, este fue de uno de los modelos con los que se fracasó al implementar el GridSearch. Por un lado, era muy demorado para entrenar, lo que hizo que no se pudieran variar muchos hiperparámetros y por el otro, la precisión de SVM era peor que la de los demás tipos de modelos, por lo que no se compensaba este tiempo de entrenamiento.

Aunque creo que son pocos valores para hacer un análisis de calidad, creo que en la Figura 6 también podemos un comportamiento interesante en el rendimiento de los modelos, al parecer para este caso, los peores predictores de SVM, se obtenían cuando gamma se calculaba como el inverso del número de atributos.

4.4.4. Bayesian Classifier

Este fue de otro de los modelos con los que más éxito tuvo al hacer GridSearch, pasamos de un modelo de 61 % de precisión a uno de casi 85 %.

Además, en la Figura 7 también podemos ver comportamientos interesantes

- Los peores clasificadores (Los que se encuentran en x >410) eran consistentemente los que usaban el modelo de Bernoulli
- La mayoría de clasificadores tenían una precisión al rededor de 77 %
- Los mejores clasificadores (Los que se encuentran en x <70) eran consistentemente los que usaban el modelo de Bernoulli alterado para variables categóricas. Lo cual creo que tiene sentido, ya que la mayoría de variables de los datos eran de este tipo.

Algo que también cabe destacar es que este modelo no funciona bien cuando en los datos de prueba aparecen valores que no estaban en los datos de entrenamiento. Por lo que para calcular la predicción lo que se hizo fue separar los datos de prueba en 2 grupos, los que tienen datos nuevos (grupo A) y los que no, (grupo B); luego hacer la preedición sobre B con el mejor modelo Bayesiano, y sobre A con el otro mejor modelo que se tenía (Random Forest).

4.4.5. Artificial Neural Network

Este creo que fue otro de los fracasos del proyecto. A primera vista, al observar la Figura 8 podemos ver una mejora en el rendimiento de casi un 50 %, pero si analizamos un poco los datos vemos que este cambio no se da por hacer GridSearch.

Al ver la gráfica de precisión vemos que los modelos caían dentro de dos categorías, los que tenían una precisión del 25 % y los de 75 %. Luego, al investigar

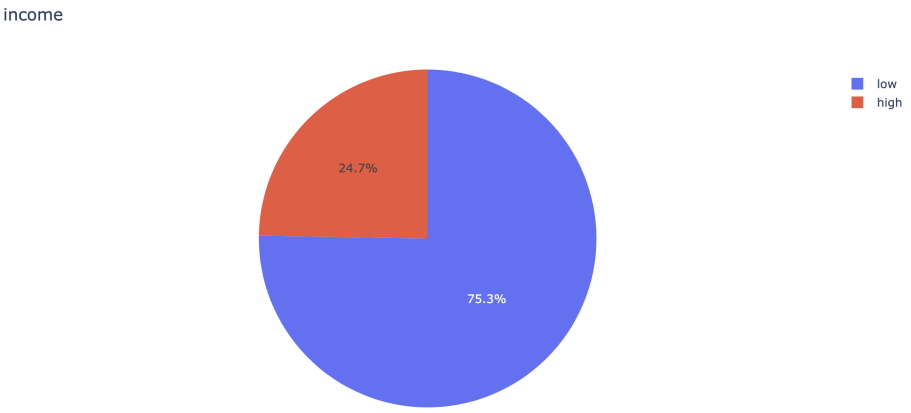

Figura 9: Distribución de clases de Income

un poco la distribución de las clases en los datos de prueba, encontré lo visto en la Figura 9.

Al ver la coincidencia entre la precisión de los modelos y la distribución de las clases, pensé en comprobar si tenían algo que ver, para hacerlo cree una ANN de prueba, con los mejores hiperparámetros la entrene y predije manualmente valores para analizar sus resultados, y vi que en entrenamientos consecutivos, los resultados no se mantenían consistentes, a veces los resultados de la red eran en su mayoría la clase "low" y a veces la clase "high" para los mismos datos de prueba.

Con esto creo encontré la respuesta de este salto en el rendimiento de los modelos. Pues asumiendo que tenemos un modelo que como resultado nos da una lista de solo la clase "low" la precisión de este modelo seria del 75 % porque al rededor de 75 % de los registros tienen esta clase y de igual manera para la clase "high".

5. Conclusiones

- La optimización automática de hiperparámetros probó ser muy útil en el desarrollo del proyecto, todos los modelos a excepción de la red neuronal lograron mejorar su precisión durante este procedimiento.
- Las redes neuronales tuvieron un fallo en su entrenamiento, como se explicó en la discusión, creo que tiene que ver con la distribución de los valores de la clase Income.
- Para este conjunto de datos, los modelos que mejor se desempeñó tuvieron fueron los Random Forest y los clasificadores Bayesianos categóricos.
- Y en contraste, los que peor desempeño tuvieron fueron los KNN y SVM.

- Creo que el proyecto fue un éxito al momento de compararlo con los resultados encontrados en [1], nuestro mejor modelo tenía una precisión del 85 %, tan solo un 3 % por debajo del mejor modelo hecho para estos datos.

Referencias

[1] Navoneel Chakrabarty and Sanket Biswas. A statistical approach to adult census income level prediction. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 207–212. IEEE, 2018.

[2] Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

[3] Raji Ghawi and Jürgen Pfeffer. Efficient hyperparameter tuning with grid search for text categorization using knn approach with bm25 similarity. *Open Computer Science*, 9(1):160–180, 2019.

[4] Daud Muhajir, Muhammad Akbar, Affindi Bagaskara, and Retno Vinarti. Improving classification algorithm on education dataset using hyperparameter tuning. *Procedia Computer Science*, 197:538–544, 2022.

[5] Erwan Scornet. Tuning parameters in random forests. *ESAIM: Proceedings and Surveys*, 60:144–162, 2017.

[6] Jacques Wainer and Pablo Fonseca. How to tune the rbf svm hyperparameters? an empirical evaluation of 18 search algorithms. *Artificial Intelligence Review*, 54(6):4771–4797, 2021.