Proyecto 1 - Clasificación

[*]Link del código: https://github.com/m41k1204/ml_classification

1ro Michael Hinojosa
Facultad de Computación
Ciencias de la Computación
michael.hinojosa@utec.edu.pe
Contribución: 100%

2do Mikel Bracamonte
Facultad de Computación
Ciencias de la Computación
mikel.bracamonte@utec.edu.pe
Contribución: 100%

3ro Michael Castillo
Facultad de la Computación
Ciencias de la Computación
michael.castillo@utec.edu.pe
Contribución: 100%

I. Introducción

En el siguiente documento se presentará el desarrollo y los hallazgos del Proyecto 1 de CS-3061 Machine Learning. El objetivo del proyecto es predecir el riesgo de default de clientes que aplican por préstamos comerciales, mediante la implementación de 3 modelos multiclases de clasificación: Regresión Logística, Máquinas de soporte vectorial y Árboles de decisión. Antes evaluar los modelos se realizará un exhaustivo preprocesamiento de la data y se implementarán algoritmos de selección de características: Information Gain y Random Forest Feature Importance. Asimismo se implementará un algoritmo de extracción de características: Principal Component Analysis (PCA). Con el objetivo de encontrar la mejor forma de controlar la dimensionalidad se compararán los algoritmos para encontrar el mejor. Por último, se manipularán los hiperparametros en cada modelo para poder encontrar los idóneos y se compararán los modelos entre sí para encontrar el mejor.

II. Análisis y preprocesamiento de datos

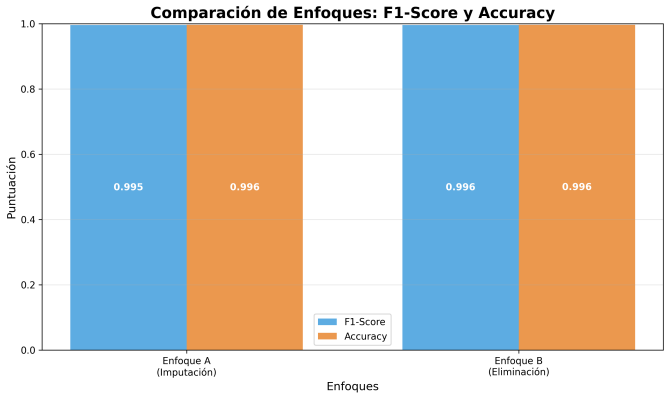
Antes de poder entrenar los modelos con los datos de entrenamiento y posteriormente probar nuestros modelos con los datos de testeo, tenemos que preprocesar los datos para evitar entrenar mal nuestros modelos.

A. Datos faltantes

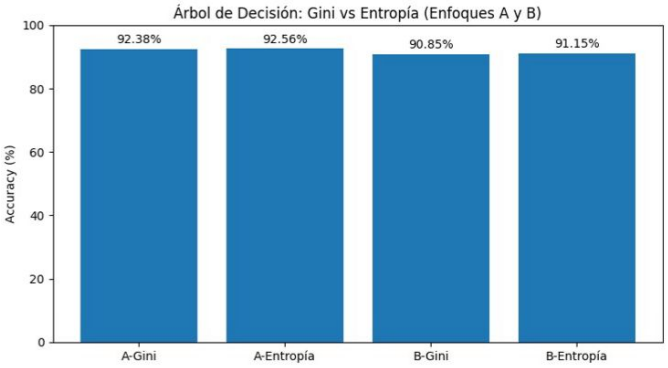
Una revisión rápida muestra que en la data de entrenamiento hay 3640 datos faltantes, mientras que en la data de testeo hay 1108 datos faltantes. Para solucionar ello se plantearon dos soluciones:

A. Imputación de datos, es decir agregar datos, la mediana en caso de variables numericas y la moda en caso de variables categóricas.
B. Eliminación de filas que contenían datos faltantes.

Al correr la Softmax Regression con ambos datos, no se identifican diferencias en los Enfoques.



Ambos tienen el mismo F1-Score y Accuracy. Sin embargo, al correr el Árbol de decisión con ambos Enfoques si se puede notar una leve diferencia a favor de el Enfoque A.



Es así, que se tomó la decisión de utilizar en su mayoría la imputación de datos.

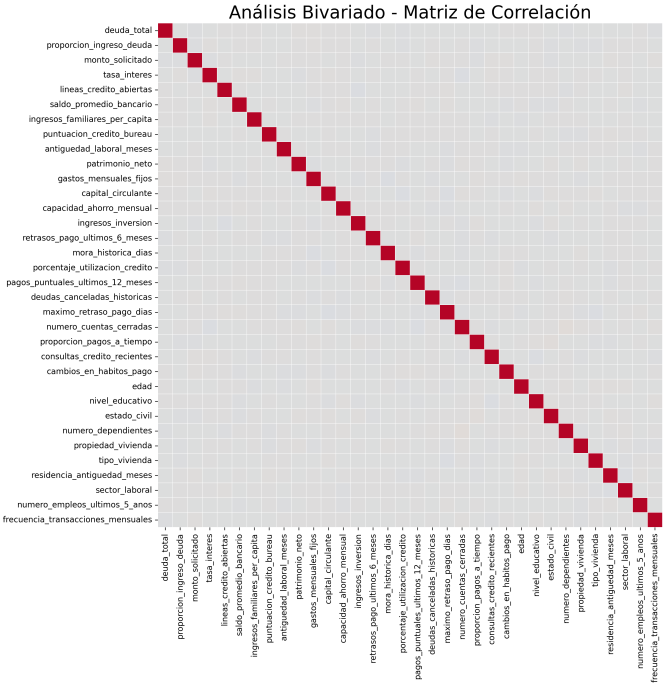
B. Normalización y estandarización de los datos

El siguiente paso es el de normalizar los datos. Para ello se utilizó el StandardScaler de la librería sklearn. Asímismo se agregó la columna de los bias. Por último se realizó un mapeo de las clases. ”Bajo” pasó a ser ”0”, ”Medio” pasó a ser ”1” y ”Alto” pasó a ser ”2”.

C. Exploración de Datos

Durante la exploración de datos se realizó un análisis bivariado para poder identificar la correlación entre las variables. Esta información nos ayuda a comprender las relaciones existentes entre las características del dataset y la variable

objetivo (nivel de riesgo), facilitando la selección de features relevantes y detectando posibles problemas de multicolinealidad que podrían afectar el rendimiento del modelo. Como se puede observar en el próximo gráfico hay una mínima correlación entre las variables, por no decir nula. Esto significa que no hay relaciones peligrosas para nuestros modelos.



Con un mejor entendimiento del dataset y teniendo los datos de entrenamiento y testeo listos para entrenar los modelos, podemos continuar con los 3 modelos desarrollados.

III. Regresión Logística

Este modelo es usado para predecir la probabilidad de pertenencia a una clase basándose en una o más variables independientes. La variable que se busca predecir es una variable dependiente categórica, donde se estima la probabilidad de que una instancia pertenezca a una clase particular.

La función de la regresión logística es una función sigmoide o logística con forma de "S" que encapsula los resultados entre [0,1], de modo que se pueda calcular la probabilidad que una instancia pertenezca a una de las clases categóricas.

A. Hipótesis

La hipótesis de regresión logística está definida por:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{1}$$

donde θ son los parámetros del modelo y x es el vector de características.

B. Función de Costo

La función de costo para regresión logística binaria utiliza la log-verosimilitud negativa:

$$J(\theta) = -\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] \tag{2}$$

donde m es el número de ejemplos de entrenamiento.

C. Optimización

Para minimizar la función de costo, se utiliza el descenso por gradiente. El gradiente está dado por:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)} \tag{3}$$

La actualización de los parámetros se realiza mediante:

$$\theta = \theta - \alpha \nabla_\theta J(\theta) \tag{4}$$

donde α es la tasa de aprendizaje. α es junto con las epocas los dos hiperparametros que se pueden alternar para encontrar el mejor modelo posible.

D. Problema de Multiclases

Dado que la regresión logística es binaria, se deben pleantear y realizar diversas estrategias para poder entrenar el dataset del proyecto. Esto debido a que la clasificación de riesgo crediticio presenta tres clases (Bajo, Medio, Alto). En ese sentido se pleantearon dos enfoques principales:

1) One-vs-All (OvA): En este enfoque se entrenan K clasificadores binarios independientes, uno para cada clase. Cada clasificador distingue una clase específica del resto de clases combinadas, transformando el problema multiclase en K problemas binarios.

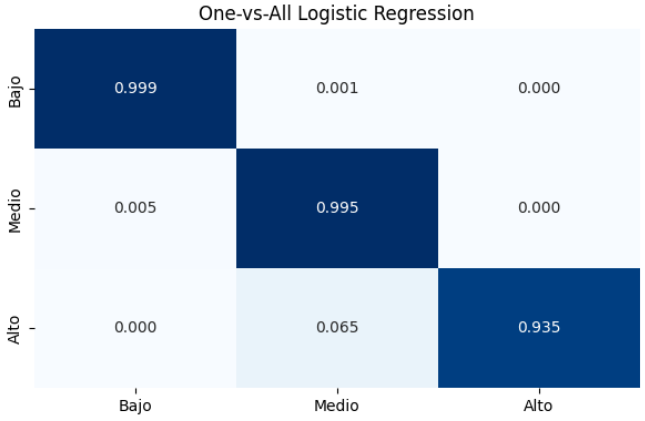
Para cada clase i, se crea una variable objetivo binaria:

$$y_{binary}^{(i)} = \begin{cases} 1 & \text{si } y = i \\ 0 & \text{si } y \neq i \end{cases} \tag{5}$$

Cada clasificador utiliza la función sigmoide estándar:

$$h_\theta^{(i)}(x) = \frac{1}{1 + e^{-\theta^{(i)T} x}} \tag{6}$$

Para realizar la predicción final, se evalúan todos los clasificadores y se selecciona la clase con mayor probabilidad.



La matriz de confusión de OvA Regression muestra que el

modelo entrena bien para personas con riesgo Bajo o Alto, pero el recall en Alto disminuye y el modelo se confunde levemente más entre personas de riesgo Medio y Alto.

2) Softmax Regression: La regresión softmax es una generalización natural de la regresión logística para clasificación multiclase [1]. Para K clases, la hipótesis produce un vector K-dimensional de probabilidades:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{\theta^{(j)T}x}} \begin{bmatrix} e^{\theta^{(1)T}x} \\ e^{\theta^{(2)T}x} \\ \vdots \\ e^{\theta^{(K)T}x} \end{bmatrix} \quad (7)$$

[1] La función de costo para softmax regression es:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \mathbf{1}\{y^{(i)}=k\} \log \frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}}\right] \quad (8)$$

[1] El gradiente para la clase k está dado por:

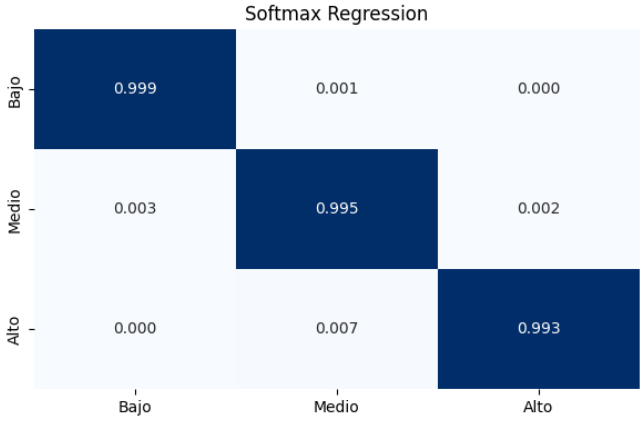
$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left[x^{(i)}\left(\mathbf{1}\{y^{(i)}=k\} - P(y^{(i)}=k|x^{(i)};\theta)\right)\right] \quad (9)$$

La clase predicha corresponde al índice con mayor probabilidad en el vector de salida softmax:

$$\hat{y} = \arg\max_{k\in\{0,1,2\}} P(y=k|x;\theta) \quad (10)$$

donde $k=0$ corresponde a riesgo "Bajo", $k=1$ a "Medio" y $k=2$ a "Alto".

Ambos enfoques permiten manejar eficientemente la clasificación multiclase del riesgo crediticio, proporcionando probabilidades interpretables para cada nivel de riesgo.



El modelo de Regresión Softmax presenta claras mejoras sobre la estrategia OvA. El recall en la clase Alto crece en aprox. medio punto porcentual y se acerca casi al 1.

IV. Máquina de soporte vectorial (SVM)

Este modelo de clasificación intenta hallar el mejor hiperplano que separe dos grupos de datos. Para lograr esto, se intenta maximizar la distancia entre el hiperplano y los puntos más cercanos de las clases, generando dos hiperplanos paralelos que se separan del principal hasta que se intercepte con algunos de los puntos. Estos puntos se denominan vectores de soporte.

El hiperplano está definido por la hipótesis:

$$(x_i * w^t + b) \quad (11)$$

El modelo SVM con el que se trabajó es una variante denominada Soft SVM. En este caso, se introduce un nuevo parámetro C, cuyo objetivo es permitir que algunos puntos sobrepasen el margen marcado por el hiperplano. De esta manera, el modelo puede ajustarse a datos más reales, donde estos no son necesariamente perfectamente separables.

De esta manera, nuestra función de pérdida se define de la siguiente forma:

$$\frac{1}{2}||w||_2^2 + C\sum_{i=0}^{n} max(0, 1 - y_i(x_i * w^t + b)) \quad (12)$$

Este modelo pasa por una etapa de entrenamiento, en la que queremos hallar los valores óptimos de w y b de manera que minimicemos la función de pérdida. Para esto usamos el descenso por gradiente, que consiste en actualizar los parámetros en la dirección opuesta de la gradiente de la función de pérdida. Para lograr esto, tenemos que hallar las derivadas de la función:

$$\frac{\partial L}{\partial w} = \begin{cases} w + C\sum_{i=0}^{n} -y_i x_i, & \text{si } y_i(x_i w^t + b) < 1, \\ w, & \text{en otro caso.} \end{cases}$$

$$\frac{\partial L}{\partial b} = \begin{cases} C\sum_{i=0}^{n} -y_i, & \text{si } y_i(x_i w^t + b) < 1, \\ 0, & \text{en otro caso.} \end{cases}$$

Al tener las derivadas, la actualización de los parámetros se hace de la siguiente manera:

$$w = \begin{cases} w - \alpha(w + C\sum_{i=0}^{n} -y_i x_i), & \text{si } y_i(x_i w^t + b) < 1, \\ w - \alpha * w, & \text{en otro caso.} \end{cases}$$

$$b = \begin{cases} b - \alpha(C\sum_{i=0}^{n} -y_i), & \text{si } y_i(x_i w^t + b) < 1, \\ b, & \text{en otro caso.} \end{cases}$$

Una vez obtenido el valor de w y b, podemos clasificar un punto dependiendo si está encima o debajo del hiperplano, de la siguiente manera:

$$\hat{y} = sign(w^T x + b) \quad (13)$$

Es decir, en base al signo del resultado, clasificamos el punto en una u otra clase.

Sin embargo, este modelo solo nos permite clasificar datos de manera binaria. En nuestro caso, se desea poder

clasificar los datos en 3 clases, por lo que esto no es suficiente. Para lograr esto, se usó el método de One-vs-All, en el que generamos 3 SVMs, uno que clasifique los elementos en los que pertenecen a la clase 0 y los que no pertenecen, otro que haga lo mismo pero para los que pertenecen a la clase 1 y los que no pertenecen, y finalmente lo mismo para la clase 2. Al clasificar un punto, calculamos el score del punto con respecto a cada uno de los 3 SVMs, de la siguiente manera:

$$score = w^T x + b \tag{14}$$

Luego, clasificamos al punto en la SVM que obtuvo el score más alto.

De esta manera, ya podemos entrenar nuestro modelo, y clasificar los datos de testing. Al hacerlo, obtenemos la siguiente matriz de confusión:

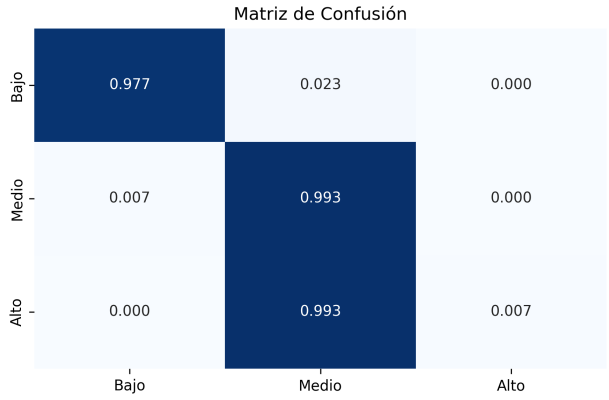


Fig. 1. Matriz de confusión del soft SVM

Para las clases 'Bajo', 'Medio' y 'Alto' se obtuvo un f1-score de 0.98, 0.87 y 0.01 respectivamente, un macro avg de 0.62, un weighted avg de 0.78 y un accuracy de 0.84.

Como se pudo observar, el resultado no fue ideal, ya que casi ningún dato se clasificó como 'Alto' y el modelo juntó a los datos 'Medio' y 'Alto' en la misma categoría. Esto indica que el modelo no pudo diferenciar entre esas dos categorías. Para solucionar esto, podemos revisar el cálculo del score. Como se vió anteriormente, el score se calculaba simplemente viendo si el dato estaba encima o debajo del hiperplano. Eso funciona correctamente si solo nos importa ese dato, pero en este caso, también nos importa saber qué tan lejos o cerca está el punto del hiperplano, y como se van a comprar los scores, esta métrica es mucho más útil, ya que terminaremos clasificando un dato viendo qué hiperplano está más lejos en la dirección positiva. Para esto, dividiremos el resultado entre la norma de W, obteniendo así la distancia al hiperplano:

$$score = \frac{w^T x + b}{||w||} \tag{15}$$

Entrenando el modelo con esta nueva ecuación para el score, obtenemos lo siguiente:

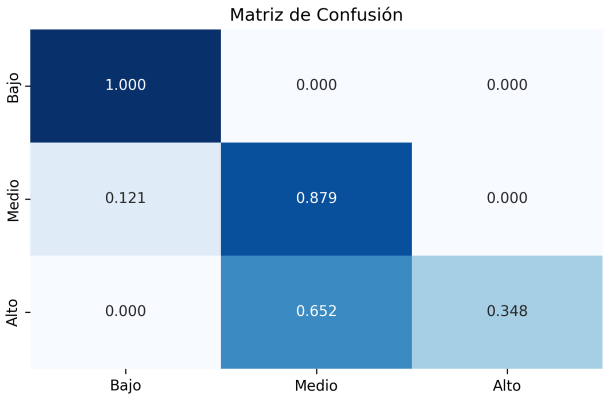


Fig. 2. Matriz de confusión del soft SVM usando la norma

Para las clases 'Bajo', 'Medio' y 'Alto' se obtuvo un f1-score de 0.90, 0.86 y 0.52 respectivamente, un macro avg de 0.76, un weighted avg de 0.82 y un accuracy de 0.84.

Con esto obtuvimos mejores resultados, y el modelo empieza a poder diferenciar entre la clase 'Medio' y 'Alto'. Sin embargo, todavía no los clasifica de manera completamente correcta. Para mejorar el modelo, podemos cambiar la forma en la que usamos los hiperparámetros. Actualmente, entrenamos los 3 SVMs con los mismos hiperparámetros, más importantemente, con los mismos valores de C. Lo que podemos hacer, es variar el valor de C dependiendo del SVM que estemos entrenando. El valor de C indica que tan estricto será el margen, si es un valor alto, hallará el hiperplano que separe ambas clases de manera casi perfecta, sin importa si eso implica que el margen sea extremadamente pequeño, y si es un valor muy bajo, entonces no le importará mucho que los puntos se pasen del margen, y priorizará encontrar el margen más grande. Viendo los datos que tenemos, se puede observar que los ejemplares de la clase 'Alto' son mucho más escasos, por lo que clasificar un dato incorrectamente es mucho más costoso. Si le aumentáramos el valor de C, forzaríamos a que estos datos se representen mejor. Entonces, una primera idea es hacer que las clases con más datos tengan un valor de C más bajo, y las clases con menos datos tengan un valor de C más alto. Para esto, agregaremos pesos que multiplicarán a cada C, calculados de la siguiente manera:

$$weight_i = \frac{n}{3 * c_i} \tag{16}$$

donde c_i es la cantidad de elemento en la clase i, y n es la cantidad de elementos totales.

Ahora, al entrenar al SVM iésimo, usaremos como hiperparámetro C al valor de C multiplicacdo por el peso iésimo. De esta manera, obtenemos la siguiente matriz de confusión al entrenar al nuevo modelo:

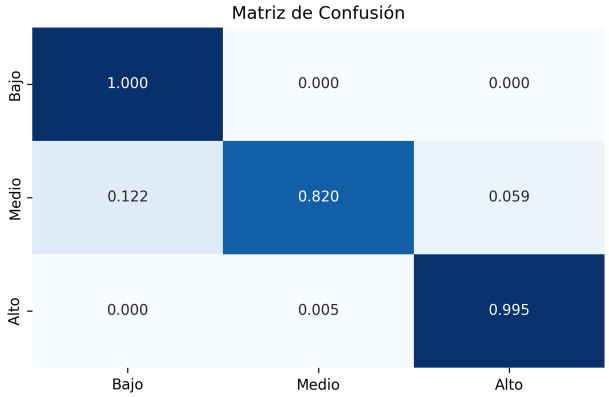


Fig. 3. Matriz de confusión del soft SVM usando pesos

Para las clases 'Bajo', 'Medio' y 'Alto' se obtuvo un f1-score de 0.90, 0.90 y 0.90 respectivamente, un macro avg de 0.90, un weighted avg de 0.90 y un accuracy de 0.90.

Como sigueinte optimización del modelo, podemos cambiar los valores de los pesos manualmente hasta obtener el mejor resultado. De esta manera se obtuvo la siguiente matriz de confusión:

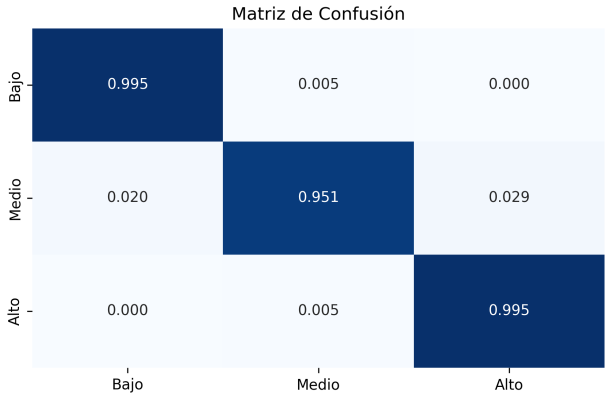


Fig. 4. Matriz de confusión del soft SVM optimizando los pesos manualmente

Para las clases 'Bajo', 'Medio' y 'Alto' se obtuvo un f1-score de 0.98, 0.97 y 0.95 respectivamente, un macro avg de 0.97, un weighted avg de 0.97 y un accuracy de 0.97.

Para terminar de optimizar el modelo, podemos aplicar algun tipo de reducción de dimensionalidad. En este caso, se aplicó la técnica de information gain, con k = 30. Con esto se obtuvo la siguiente matriz de confusión final:

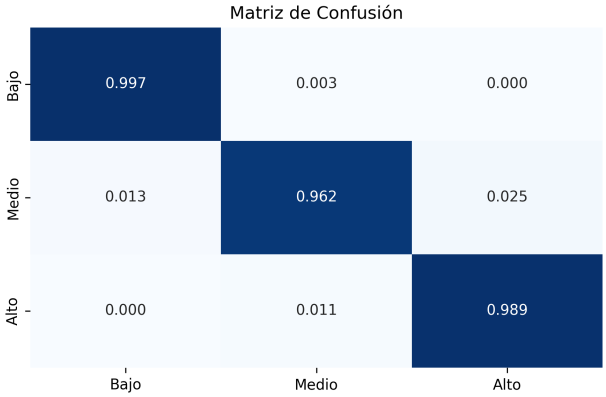


Fig. 5. Matriz de confusión del soft SVM con information gain

Para las clases 'Bajo', 'Medio' y 'Alto' se obtuvo un f1-score de 0.99, 0.98 y 0.96 respectivamente, un macro avg de 0.97, un weighted avg de 0.98 y un accuracy de 0.98.

V. Árbol de decisión

El algoritmo CART (Classification and Regression Trees) construye un árbol binario que divide recursivamente el conjunto de datos con el objetivo de reducir la impureza en cada nodo.

Un nodo S contiene un subconjunto de los datos y se caracteriza por la distribución de clases en su interior. Sea k el número de clases posibles y p_i la proporción de elementos de la clase i en S, de modo que $\sum_{i=1}^{k} p_i = 1$.

Para medir la impureza de un nodo se emplean distintas funciones de costo. Dos de las más utilizadas son:

- Índice de Gini:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{k} p_i^2.$$

  Este índice mide la probabilidad de clasificar erróneamente un ejemplo escogido al azar si se etiqueta de acuerdo con la distribución de clases en el nodo. Valores cercanos a 0 indican nodos puros (con predominio de una sola clase), mientras que valores cercanos al máximo $\frac{k-1}{k}$ indican una mezcla balanceada de clases.

- Entropía:

$$\text{Entropía(S)} = -\sum_{\text{i=1}}^{\text{k}} \text{p}_\text{i} \log_2(\text{p}_\text{i}).$$

  Esta medida, proveniente de la teoría de la información de Shannon, cuantifica la incertidumbre asociada a la clasificación de un ejemplo. Su valor mínimo es 0 (pura certeza en el nodo) y su máximo ocurre cuando todas las clases están igualmente representadas.

En cada división del árbol se busca la característica X_j y el umbral t que generen la mayor reducción de impureza:

$$\Delta I(S, X_j, t) = I(S) - \frac{|S_L|}{|S|} I(S_L) - \frac{|S_R|}{|S|} I(S_R),$$

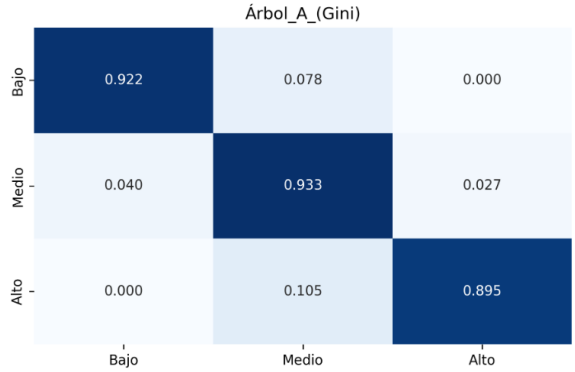


Fig. 6. Matriz de confusión del árbol de decisión con criterio Gini

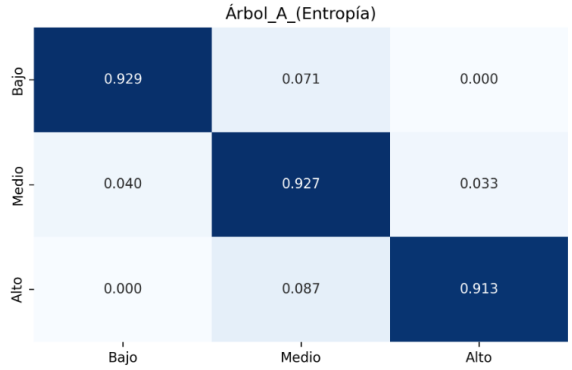


Fig. 7. Matriz de confusión del árbol de decisión con criterio Entropía

donde $I(\cdot)$ es la medida de impureza elegida, y S_L, S_R son los subconjuntos resultantes de aplicar el corte $X_j \leq t$ y $X_j > t$ respectivamente. El árbol se expande de manera recursiva hasta alcanzar un criterio de parada, como tamaño mínimo de muestra en nodo, impureza nula o una profundidad máxima.

De esta forma, cada nodo interno actúa como una regla de decisión que particiona el espacio de características, y las hojas corresponden a regiones del espacio donde se asigna una clase mayoritaria.

En este trabajo se entrenaron árboles de decisión bajo ambos criterios, Gini y Entropía, con el mismo enfoque de imputación de datos. Los resultados se presentan en las Figuras 6 y 7.

Como se observa en la Figura 6, el criterio Gini produce un modelo con un desempeño aceptable, pero presenta mayor confusión entre las clases 'Medio' y 'Alto'. En cambio, la Figura 7 muestra que el criterio de Entropía logra una separación más clara entre estas clases, reduciendo la tasa de error y aumentando la estabilidad de los resultados. Esto confirma lo señalado en la literatura: aunque ambos criterios tienden a dar resultados similares, la entropía puede capturar con mayor sensibilidad los cambios en la distribución de probabilidades, lo que la hace más adecuada en datasets donde existen clases con menor representación.

VI. MANEJO DE LA DIMENSIONALIDAD

Debido al alto número de features que contiene el dataset (35), surge la necesidad de buscar formas de mejorar los resultados de los modelos, implementando selección de características y extracción de características [2].

A. Selección de Características

1) Information Gain: Utiliza información mutua para evaluar la relevancia de cada característica respecto a la variable objetivo [2]. Este método mide la dependencia estadística entre variables, seleccionando las k características con mayor ganancia de información.

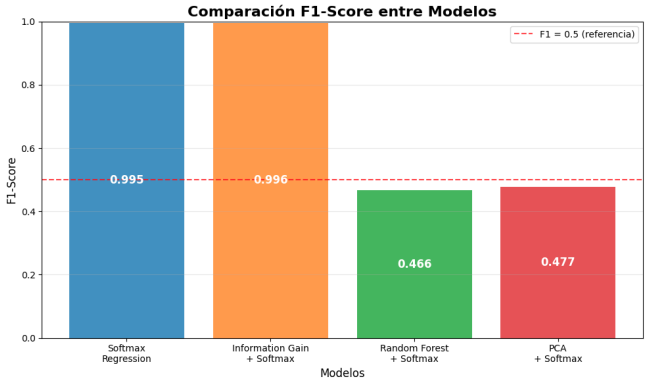
2) Random Forest Feature Importance: Emplea un ensemble de árboles de decisión para calcular la importancia de cada característica basada en la reducción de impureza que aporta [2]. Este enfoque selecciona las k características con mayor importancia, integrando la selección previo al entrenamiento del modelo.

B. Extracción de Características

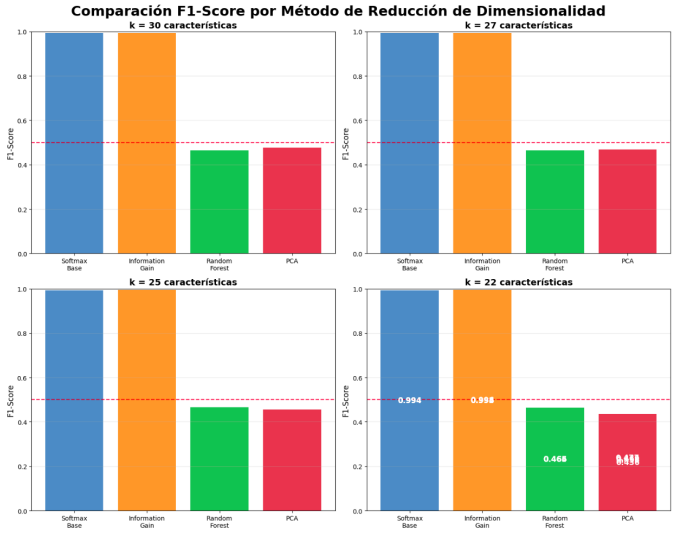
1) Principal Component Analysis (PCA): PCA crea nuevas características como combinaciones lineales de las originales, reduciendo la dimensionalidad mientras mantiene la información más relevante.

C. Resultados

El siguiente gráfico muestra los resultados luego de entrenar el Softmax Regression con datos sin reducción de la dimensionalidad y con datos luego aplicar los 3 algoritmos.



Tanto el Random Forest como el PCA tienen resultados desfavorables, mientras que el Information Gain mejora, aunque por muy poco. Para obtener los mejores resultados posibles se manipuló el hiperparametro "k", que determina los top "k" features que que se extraerán. Para el gráfico anterior se utilizó k = 25. En el siguiente gráfico se entrenaron 4 veces modelos, utilizando distintos ks: 30, 27, 25 y 22.



Para ningún ”k”, el Random Forest o PCA sobrepasan la linea punteada roja de 0.5. Aunque resulta interesante que mientras mayor es el ”k”, el PCA tiene resultados marginalmente mayores al Random Forest, mientras que si el ”k” es menor, la relacion se invierte y el Random Forest tiene mejores resultados.

En los árboles de decisión (CART) se adoptó el criterio de entropía por su ligero mejor desempeño al diferenciar la clase minoritaria. Para evaluar la sensibilidad del modelo se variaron tres hiperparámetros estructurales: profundidad máxima, mínimo de muestras para dividir un nodo y mínimo de muestras por hoja. El rendimiento fue consistente: *accuracy* entre 92.3% y 93.0%, F1 macro entre 0.921 y 0.929, y F1 de la clase minoritaria alrededor de 0.91. Esto indica que aumentar mucho la profundidad sólo produce mejoras marginales, mientras que controlar *min_split* y *min_leaf* aporta más estabilidad sin sacrificar desempeño, manteniendo un resultado global cercano al 93%.

TABLE I
SENSIBILIDAD A HIPERPARÁMETROS DEL ÁRBOL (CRITERIO ENTROPÍA, ENFOQUE A).

Depth	Split	Leaf	Acc. (%)	F1 macro	F1 “Alto”
12	4	2	92.30	0.921	0.905
14	6	3	92.86	0.927	0.912
16	8	4	93.04	0.929	0.9226
	Promedio		92.73	0.926	0.913

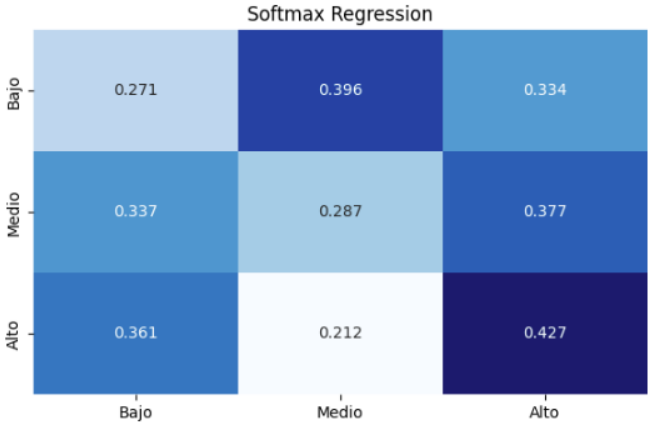
VII. COMPARACIÓN ENTRE MODELOS

Para evaluar el rendimiento de los diferentes algoritmos de clasificación implementados, se presenta una comparación de las métricas principales obtenidas por cada modelo en el conjunto de prueba.

Una diferencia interesante entre Softmax Regression y Soft svm son los valores del alpha y los epochs. El mejor alpha encontrado para soft svm fue de 0.000001 y valor adecuado para epochs fue de 500. Sin embargo para Softmax Regression fue todo lo contrario. El mejor alpha encontrado fue de 0.9 y un valor adecuado para las epochs fue de 10000.Si entrenamos la Softmax regression con los hiperparametros de la soft svm obtenemos la siguiente matriz de confusión.

TABLE II
COMPARACIÓN DE RENDIMIENTO ENTRE MODELOS DE CLASIFICACIÓN

Modelo	**F1-Score**	**Accuracy**	**Recall**
SVM	0.97	0.98	0.98
Softmax Regression	0.996	0.996	0.997
Árbol de Decisión	0.9226	0.9304	0.919



VIII. CONCLUSIONES Y PROPUESTAS DE MEJORA

Los resultados experimentales demuestran que Softmax Regression alcanza el mejor rendimiento con F1-Score de 0.996, superando a SVM (0.97) y árbol de decisión(0.93). La extensión multiclase mediante softmax mostró ventajas sobre One-vs-All en términos de recall para la clase Alto. Hallazgos clave:

- Information Gain fue la única técnica de reducción de dimensionalidad que mejoró el rendimiento base
- Los hiperparámetros óptimos difieren significativamente entre modelos (α=0.9, epochs=10000 para Softmax vs α=0.000001, epochs=500 para SVM)
- El desbalance de clases afectó particularmente al rendimiento de SVM hasta implementar pesos adaptativos

Si bien el Softmax Regression tuvo los mejores resultados, los resultados favorables también se dieron porque la data no era demasiado difícil de entrenar. Durante la investigación para este proyecto se encontró un modelo de Softmax Regression que implementa un sistema de pesos adaptativos [3], el cual podría mejorar el rendimiento en datasets más complejos con datos ruidosos o desbalanceados.

IX. NOTA

Para el desarrollo de este proyecto se utilizó inteligencia artificial para los gráficos.

Todos los Colabs estan en Referencias.

REFERENCES

[1] Stanford University, “Softmax Regression,” UFLDL Tutorial. [Online]. Available: http://uflDl.stanford.edu/tutorial/supervised/SoftmaxRegression/

[2] R. R. Zebari, A. M. Abdulazeez, D. Q. Zeebaree, D. A. Zebari, and J. N. Saeed, “A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction,” Journal of Applied Science and Technology Trends, vol. 1, no. 1, pp. 56-70, 2020.

[3] Y. Ren, P. Zhao, Y. Sheng, D. Yao, and Z. Xu, “Robust softmax regression for multi-class classification with self-paced learning,” in Proc. 26th Int. Joint Conf. Artificial Intelligence (IJCAI), 2017, pp. 2641-2647.

[4] W.-Y. Loh, “Fifty Years of Classification and Regression Trees,” *International Statistical Review*, vol. 82, no. 3, pp. 329–348, 2014. doi: 10.1111/insr.12016.

[5] M. Hinojosa, M. Bracamonte, and M. Castillo, “Softmax Regression Implementation,” Google Colaboratory. [Online]. Available: https://colab.research.google.com/drive/1RyjERph85JrlOyaNxtbzSl5d5wuH72-_

[6] M. Hinojosa, M. Bracamonte, and M. Castillo, “One-vs-All Logistic Regression Implementation,” Google Colaboratory. [Online]. Available: https://colab.research.google.com/drive/18vHnsSbzBZXozXnm3c4LcZfrRN1rf9SK

[7] M. Hinojosa, M. Bracamonte, and M. Castillo, “SVM Implementation,” Google Colaboratory. [Online]. Available: https://colab.research.google.com/drive/10TMleV5xaXeTqtGeK16uHG9DYZV1cRwT

[8] M. Hinojosa, M. Bracamonte, and M. Castillo, “Decision Tree Implementation,” Google Colaboratory. [Online]. Available: https://colab.research.google.com/drive/1KaaJI6RywampICRtyEd-26N64jNmDlYs