



**UNIVERSIDAD
DE ANTIOQUIA**

1 8 0 3

ENTREGA FINAL PROYECTO INTRODUCCIÓN A LA I.A. PARA CIENCIAS E INGENIERÍAS

Costa Rican Household Poverty Level Prediction

POR:

Estebal Alzate Pérez
Alejandro Mesa Mesa
Maria Paula Sedano Sarmiento

PROFESOR:

Raúl Ramos Pollán

Medellín - Antioquia

28 de mayo de 2023

INTRODUCCIÓN

La inteligencia artificial ha revolucionado numerosos aspectos de nuestras vidas, y su aplicación en la resolución de problemas sociales y económicos se ha convertido en un campo de creciente interés. En este contexto, el presente trabajo se enfoca en abordar un desafío predictivo de gran relevancia: la estimación del nivel de pobreza de los hogares en Costa Rica.

El banco interamericano de desarrollo necesita ayuda con la calificación de los ingresos de algunas de las familias más pobres del mundo. Esto con el objetivo de definir qué familias necesitan más ayudas o subsidios, ya que por lo general las familias más pobres no pueden proveer la información necesaria sobre sus ingresos y gastos. El problema predictivo se centra en Costa Rica, pero se puede aplicar a cualquier parte del mundo.

La información necesaria se saca del dataset de Kaggle, [Costa Rican household poverty prediction](#), que consiste en un dataset con 142 columnas que se resumen en atributos observables de las casas y las condiciones de vida de las familias a las que se les realiza el estudio, un total de 33413, de las cuales 9557 tienen un nivel de pobreza ya definido para servir como entrenamiento.

A cada hogar evaluado se le asigna un número entre 1 y 4, donde 1 significa pobreza extrema, 2 significa pobreza moderada, 3 es familia vulnerable y 4 es familia no vulnerable. La métrica utilizada para evaluar el desempeño del modelo predictivo será el “valor F1”, que es una medida basada en la precisión y la sensibilidad a partir de la recuperación de un conjunto de datos.

$$F_1 = 2 \frac{\text{precisión} * \text{sensibilidad}}{\text{precisión} + \text{sensibilidad}} \quad (1)$$

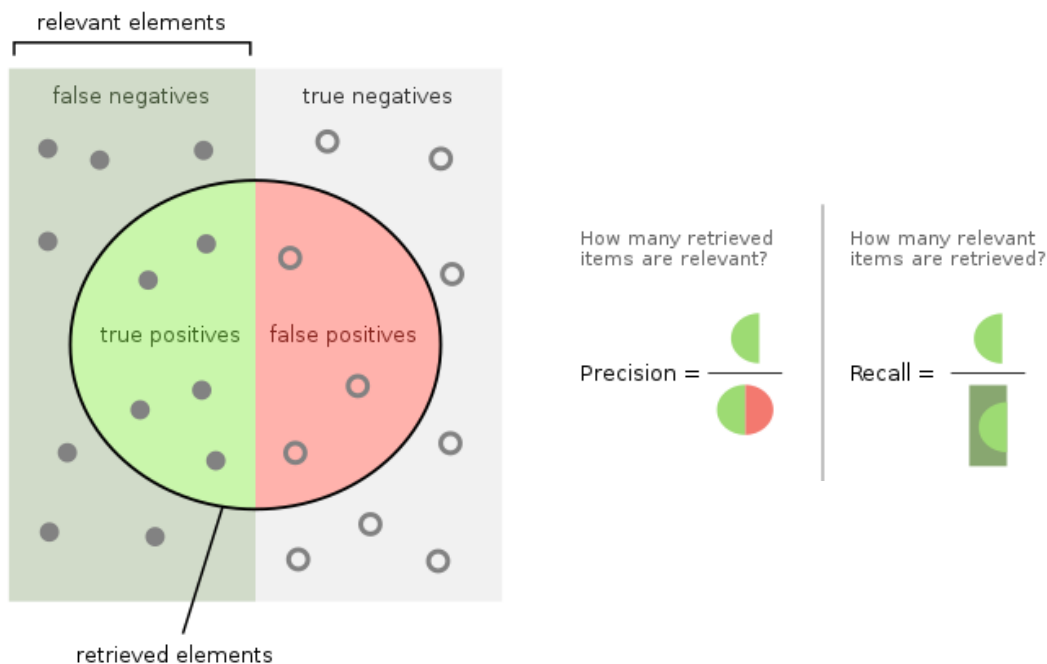


Figura 1. Valor F1. [3]

En este caso, el valor F1 es una medida más estricta que solamente la precisión, debido a que no solo debe considerar la concordancia entre los datos predichos y los reales, sino que es importante tener en cuenta la importancia y el peso de los falsos positivos y los falsos negativos. Al ser una medida estricta, no se esperan valores demasiado altos, sumado a que es apenas un inicio como equipo principiante en el tema de la inteligencia artificial y el análisis predictivo, se considera que un F1 de 0,3 será un valor aceptable.

A lo largo de este informe, describiremos el proceso seguido para abordar el problema predictivo, incluyendo el análisis exploratorio de los datos, la selección y entrenamiento de modelos de aprendizaje automático, y las técnicas utilizadas para mejorar el rendimiento del modelo. Además, discutiremos los resultados obtenidos y ofreceremos conclusiones sobre la viabilidad y utilidad de la inteligencia artificial en la predicción del nivel de pobreza en los hogares, no solo de Costa Rica, sino que puede ser aplicable a cualquier parte del mundo.

EXPLORACIÓN DESCRIPTIVA DEL DATASET

Los datos están compuestos por 2 archivos de tipo csv: un *train*, con 9557 filas y 143 columnas y un *test*, con 23856 filas y 142 columnas. Cada fila representa el conjunto de datos correspondiente a un hogar, y cada columna una característica de estos. El *train* cuenta con una columna adicional, *target*, que representa datos reales del nivel de pobreza de los hogares disponibles en este archivo; esta columna toma un valor entre 1 y 4, siendo 1 el nivel más crítico de pobreza y 4 significa que el hogar está fuera de peligro.

Entre las columnas presentes en el dataset, se encuentran una cantidad de datos que describen el estado físico de la casa mediante una calificación cuantitativa de diversos aspectos, el estado de los habitantes, su escolaridad, disponibilidad de aparatos electrónicos en el hogar, además si existe o no hacinamiento.

Para empezar a abordar los datos, lo primero que se hace es juntar ambos datasets en uno solo para asegurarse de que todas las manipulaciones que se hagan sean sobre ambos. Una vez hecho esto, se da una primera mirada mediante la función `df.info()`.

```
1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33413 entries, 0 to 33412
Columns: 143 entries, Id to Target
dtypes: float64(9), int64(129), object(5)
memory usage: 36.5+ MB
```

Figura 2. Información del dataset.

Se observa que se tienen 9 columnas de tipo flotante o real, que corresponden a datos numéricos, 129 columnas de tipo entero, de las cuales probablemente muchas representan booleanos, es decir, valores de 0 o 1 y 5 columnas de tipo objeto que pueden ser datos de tipo *string* u otros valores que normalmente no se pueden usar dentro de un problema de *Machine learning*. Para obtener más detalles sobre los datos, se usa la función `df.describe()`.

```
1 df.describe()
```

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...
count	9.150000e+03	33413.000000	33413.000000	33413.000000	33413.000000	33413.000000	33413.000000	7945.000000	33413.000000	33413.000000	...
mean	1.720308e+05	0.047077	4.955706	0.027055	0.993326	0.960464	0.237782	1.364003	0.407775	1.562595	...
std	1.550035e+05	0.211808	1.519659	0.162247	0.081423	0.194868	0.425731	0.714483	0.704245	1.003650	...
min	0.000000e+00	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	...
25%	8.000000e+04	0.000000	4.000000	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	1.000000	...
50%	1.350000e+05	0.000000	5.000000	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	1.000000	...
75%	2.000000e+05	0.000000	6.000000	0.000000	1.000000	1.000000	0.000000	2.000000	1.000000	2.000000	...
max	2.852700e+06	1.000000	15.000000	1.000000	1.000000	1.000000	1.000000	6.000000	6.000000	8.000000	...

8 rows x 138 columns

Figura 3. Descripción de las columnas.

Al usar esta función, tenemos una idea de cuáles son los valores booleanos, en donde el máximo es 1 y el mínimo es 0, o columnas en donde existen valores nulos, pues la cuenta total de datos presentes en la columna es menor al número de columnas. Sin embargo, es una función con mucho detalle específico y no deja ver a grandes rasgos características o tendencias del dataset. Para tener una visión más amplia de tendencias o una visualización gráfica rápida de los datos, se utiliza la función `df.hist()`, que muestra una representación gráfica de la frecuencia con la que aparecen los valores existentes dentro de cada columna.

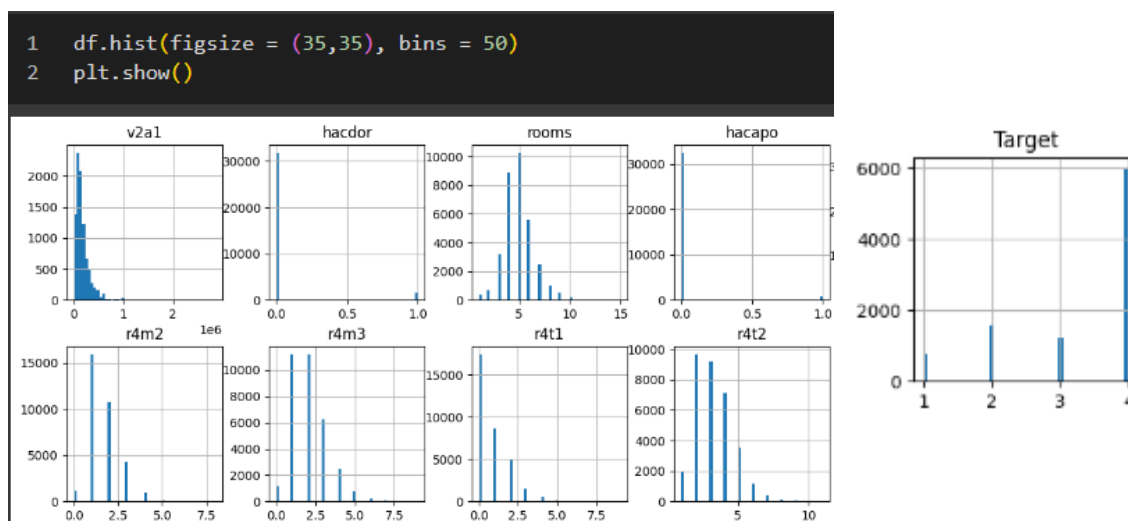


Figura 4. Frecuencia de los datos por columnas.

En la figura, se pueden ver a grandes rasgos la frecuencia con la que ocurren algunos sucesos representados por las columnas del dataset. Pero realmente el dato importante extraído de esta función es la distribución del *target* entre los datos del *train*, pues se muestra que la mayoría de datos existentes no se encuentran en una situación vulnerable, es decir, tienen calificación de 4.

Teniendo información sobre las características de los datos, se procede a hacer una limpieza de datos nulos, identificando sus causas y las mejores y más lógicas maneras de llenarlos. Los primeros valores nulos se encuentran en la columna 'v2a1', que dice cuánto dinero se paga de renta en cada hogar. Al compararlo con las filas 'tipovivi', se observa que solo existen valores nulos cuando la vivienda es propia, precaria u otra, por lo cual estos valores se llenan con 0. La siguiente columna con valores nulos es 'v18q1', que dice cuántas

tablets hay en el hogar; al compararla con 'v18q', que dice si en el hogar hay tablets, se observa que solo hay valores nulos cuando en el hogar no hay tablets, por lo que estos valores pueden llenarse con 0. Las otras columnas con valores nulos son 'meaneduc', que muestra la media de educación en el hogar, y 'rez_esc', que muestra los años atrás en escolaridad del individuo; estas columnas se pueden filtrar comparando con la edad de los individuos y aplicando la limpieza correspondiente.

Continuando con la limpieza de datos, se da una mirada a las columnas redundantes, que tienen una correlación muy alta entre sí o que directamente son iguales. Se encuentran 5 columnas que cumplen estas condiciones correspondientes al tamaño del hogar y a la edad cuadrática; se eliminan las columnas para dejar solo una por tipo de dato. Adicionalmente, el dataset contiene columnas diferenciadas entre hombres y mujeres que habitan en la casa, lo cual se demuestra que no tiene relevancia en el nivel de pobreza, por lo cual se eliminan estas columnas.

Teniendo ya un dataset más simplificado y sin valores nulos, se procede a relacionar los datos disponibles con el target, para tener una mirada más amplia de cómo influye cada variable y qué peso tiene con respecto al target. Para esto, se realizaron mapas de calor que comparaban varias columnas que establecían diferentes categorías de datos con el target, para ver qué características eran más comunes entre cada tipo de familia de acuerdo a su nivel de pobreza.

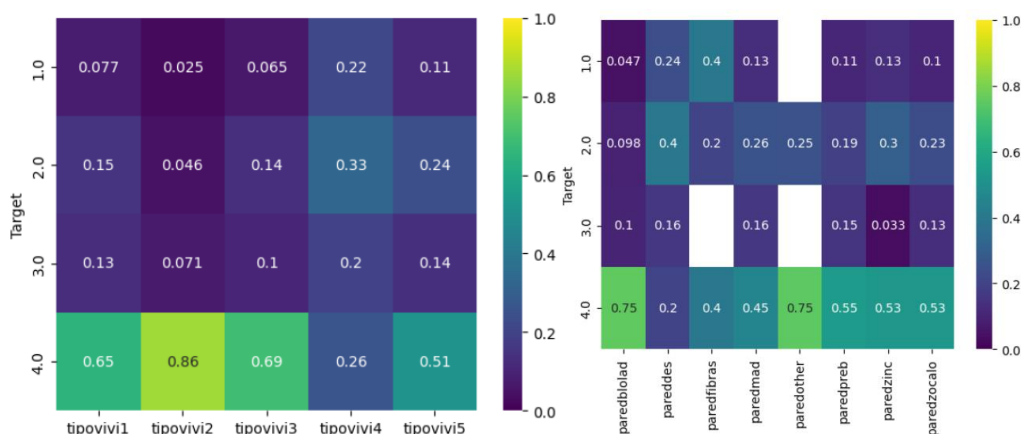


Figura 5. Mapas de calor relacionados al target.

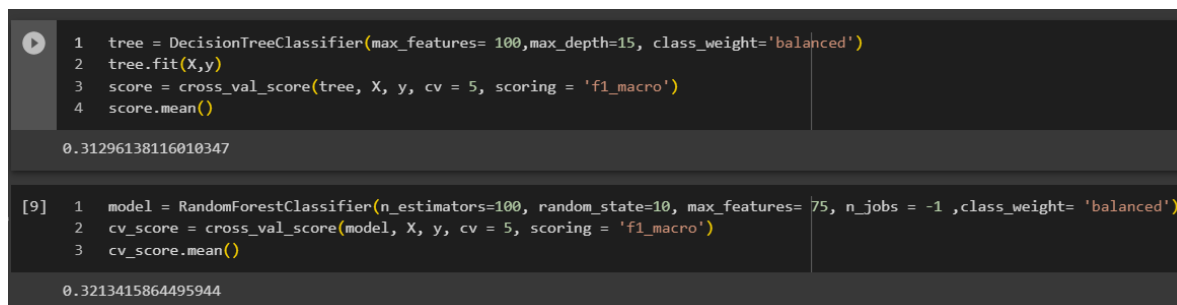
En la figura 5 se pueden observar ejemplos de esto, donde se observa, por ejemplo, que el tipo de vivienda 2 es más común en el target 4 que en los otros, mientras que el tipo 4 está más repartido entre todos los targets. Se realizó este análisis con el tipo de vivienda, la cantidad de tablets en el hogar, el tipo de pared, tipo de piso, tipo de techo, el abastecimiento de agua, la electricidad, el sanitario, la energía para cocinar, el método de eliminación de basura, el estado general de la casa, el estado civil de los individuos, el nivel de institucionalidad y el lugar de ubicación de la casa. A grandes rasgos, se observa que no necesariamente los hogares en un target de 1 tienen condiciones mucho peores que los demás y que los hogares en target 4 suelen tener repartidos todos los tipos de clasificación, aunque se considere que estos sean malos.

ITERACIONES DE DESARROLLO

Para todos los métodos utilizados, se hizo una validación de la métrica mencionada anteriormente, 'valor F1' mediante la función 'cross_val_score' de scikit learn, donde se utilizó un hiperparámetro `cv = 5`. Este parámetro implementa una validación cruzada del modelo dividiéndolo en 5 partes, evaluando el modelo 5 veces con diferentes combinaciones de datos y mirando su rendimiento con respecto a la métrica en cada iteración. Se escogió este valor para el parámetro `cv` debido a que a números muy grandes, el costo computacional empieza a ser muy alto y el tiempo de ejecución del algoritmo incrementa considerablemente, además de que utilizar un número muy alto puede hacer incurrir en una mayor varianza de los datos, al trabajar con conjuntos más pequeños y dispersos entre sí.

Selección inicial de modelos de Machine learning

Inicialmente, se opta por utilizar los modelos de árbol de decisión y RandomForest, donde se evaluará primeramente la métrica de desempeño y cómo cambia al variar los hiperparámetros escogidos. En la figura 6 se muestra la implementación de estos 2 modelos, con sus resultados bajos unos hiperparámetros dados. Para obtener los resultados mostrados en la figura, se variaron los hiperparámetros de `max_features` y `max_depth` en el árbol de decisión y `n_estimators` y `max_features` en el RandomForest. En ambos se utilizó el parámetro `class_weight = 'balanced'`, que permite ajustar los pesos de los diferentes target para tener un mejor desempeño de la métrica.



```
1 tree = DecisionTreeClassifier(max_features= 100,max_depth=15, class_weight='balanced')
2 tree.fit(X,y)
3 score = cross_val_score(tree, X, y, cv = 5, scoring = 'f1_macro')
4 score.mean()

0.31296138116010347

[9] 1 model = RandomForestClassifier(n_estimators=100, random_state=10, max_features= 75, n_jobs = -1 ,class_weight= 'balanced')
2 cv_score = cross_val_score(model, X, y, cv = 5, scoring = 'f1_macro')
3 cv_score.mean()

0.3213415864495944
```

Figura 6. Primera implementación de modelos.

En cuanto al primero de estos, se observó que al variar el número de `max_features` con un `max_depth` constante por debajo de 10, se obtenían valores para la métrica más bajos, del orden de 0,299 a 0,301; mientras que al aumentar el `max_depth` a un valor más allá de 10, para cualquier valor de `max_features` escogido , el desempeño del modelo aumenta hasta 0,313, lo cual resulta bastante decente.

En cuanto al modelo con RandomForest, se observó que al variar los hiperparámetros a valores altos, el costo computacional aumenta considerablemente; sin embargo, con un mayor número de `n_estimators` y de `max_features`, el puntaje de la métrica tiene a aumentar hasta valores de 3,3, lo cual resulta bastante mejor en comparación con el modelo de árbol de decisión. Al obtener un mejor resultado para este modelo, se estudió el efecto de variar el valor semilla `random_state` en un intervalo entre 100 y 200, buscando un mejor

desempeño de la métrica, obteniendo un resultado empírico de 160. Para probar este modelo, se utilizó una matriz de confusión que permite ilustrar la métrica de una buena manera, mostrando los falsos positivos, los falsos negativos y los valores acertados en la predicción.



Figura 7. Matriz de confusión.

En la matriz se puede observar que, aunque la métrica da un valor que a simple vista parece pequeño, en realidad el porcentaje de acierto resulta ser bastante alto como se ve al comparar los valores sobre la diagonal principal con los demás.

Tras comprobar el efecto en el cambio del `random_state`, se procede a comparar el RandomForest con otros modelos disponibles en las bibliotecas de scikit learn, en este caso, la regresión logística y el modelo gaussiano de Naive-Bayes. En la figura 8 se ve la implementación del código correspondiente.

```

1 clf1 = LogisticRegression(random_state=200,solver='newton-cg',class_weight= 'balanced',max_iter=1000)
2 clf2 = RandomForestClassifier(random_state=160, n_jobs=4, n_estimators=700, class_weight= 'balanced')
3 clf3 = GaussianNB()
4
5 eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3)],voting='hard')
6
7 for clf, label in zip([clf1, clf2, clf3, eclf], ['Logistic Regression', 'Random Forest', 'naive Bayes', 'Ensemble']):
8     scores = cross_val_score(clf, X, y, scoring='f1_macro', cv=10)
9     print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))

```

Accuracy: 0.32 (+/- 0.09) [Logistic Regression]
 Accuracy: 0.37 (+/- 0.04) [Random Forest]
 Accuracy: 0.30 (+/- 0.02) [naive Bayes]
 Accuracy: 0.34 (+/- 0.05) [Ensemble]

Figura 8. Comparación con otros modelos.

La regresión logística mostró una gran cantidad de advertencias con respecto al número de iteraciones y problemas de convergencia, por lo que fue necesario cambiar el parámetro del solucionador varias veces hasta encontrar el indicado, en este caso 'newton-cg', además de ajustar el máximo de iteraciones; en el RandomForest se utilizó la mejor combinación de hiperparámetros encontrada (no definitiva) y en el modelo gaussiano se

dejaron los parámetros por defecto. Además, se utilizó una técnica de ensemble, que combina los resultados de los modelos anteriores en busca de obtener resultados más confiables y precisos en la predicción.

En la figura se observa que el modelo con mejor resultado siguió siendo el RandomForest, con una mejora considerable con respecto a las iteraciones anteriores al utilizar los mejores hiperparámetros. También se observa que la regresión logística, aunque tiene un resultado decente, tiene una desviación estándar muy alta. El método gaussiano resulta no ser muy confiable en este caso, mientras que el ensemble no pudo superar los resultados del RandomForest, ni tampoco disminuyó la desviación.

Curvas de aprendizaje

Para diagnosticar posibles procedimientos erróneos a la hora de implementar los modelos, se utilizan las curvas de aprendizaje que se implementan mediante el módulo `learning_curves` de `sk.learn.model_selection`. En la figura 9 se muestra la implementación del código de esta.

```
1 def plot_learning_curve(estimator, X, y, cv, train_sizes):
2     train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, train_sizes=train_sizes, scoring='f1_macro')
3
4     train_scores_mean = np.mean(train_scores, axis=1)
5     train_scores_std = np.std(train_scores, axis=1)
6     test_scores_mean = np.mean(test_scores, axis=1)
7     test_scores_std = np.std(test_scores, axis=1)
8
9     plt.figure(figsize=(10, 6))
10    plt.title("Curva de Aprendizaje")
11    plt.xlabel("Tamaño del Conjunto de Entrenamiento")
12    plt.ylabel("Precisión")
13    plt.grid()
14
15    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color="r",)
16    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color="g",)
17    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Precisión en Entrenamiento")
18    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Precisión en Prueba")
19    plt.legend(loc="best")
20    plt.show()
```

Figura 9. Implementación de la curva de aprendizaje.

La curva recibe los parámetros del modelo, el dataset, las predicciones, el valor de `cv` mencionado anteriormente, el número de muestras que se tomarán para obtener los puntos de la curva y la métrica de evaluación. Además, incluye los valores con su desviación estándar. Primero, se analiza la curva de aprendizaje para el modelo con RandomForest como se observa en la figura 10.

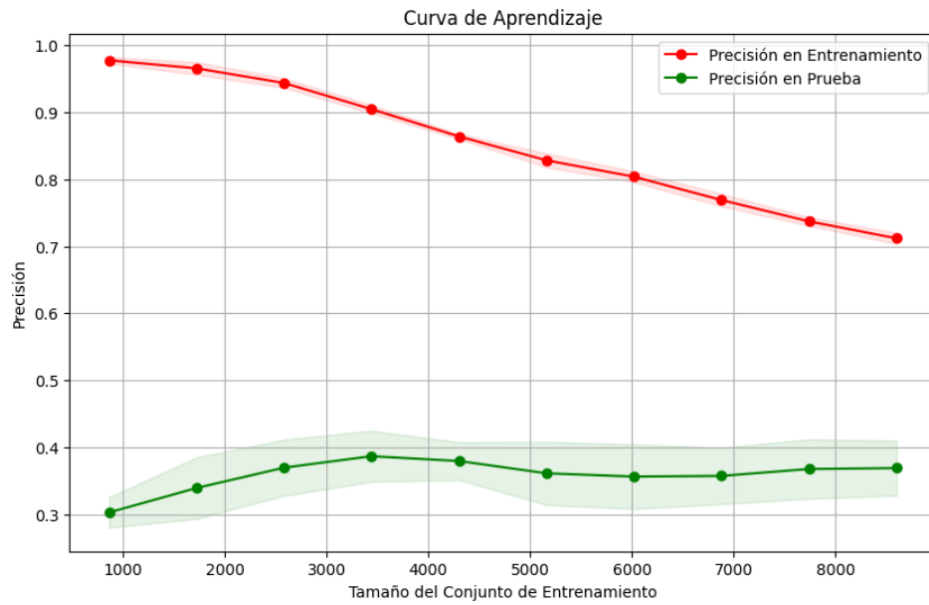


Figura 10. Curva de aprendizaje para modelo con RandomForest.

En la curva se observa claramente un caso de overfitting de los datos, pues hay una brecha significativa entre los datos de entrenamiento y los de prueba. Para mejorar esto, se propone disminuir la complejidad del modelo, aunque esto sacrifique un poco el desempeño de la métrica. En la figura 11 se muestra el desempeño de la nueva curva al disminuir los `n_estimators`.

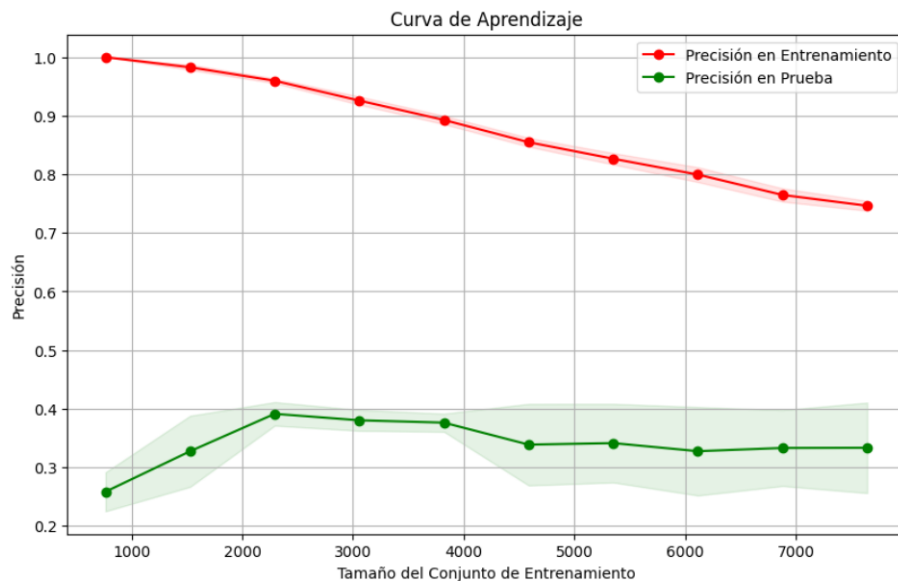


Figura 11. Nueva curva de aprendizaje para RandomForest.

Se observa que a pesar de haber disminuido la complejidad del modelo, la curva sigue teniendo la misma tendencia de overfitting, que incluso ha aumentado la separación entre los puntos de prueba y de entrenamiento. Según la documentación, esto puede deberse a varias razones: una cantidad insuficiente de datos, desequilibrio de clases, características irrelevantes o ruido, hiperparámetros incorrectos o en últimas, una mala elección del

modelo. En este caso, sabiendo que se cuenta con una gran cantidad de datos de entrenamiento, se han filtrado los datos en la exploración descriptiva, y se han explorado la mayoría de hiperparámetros disponibles, se abordan los problemas de desequilibrio de clases y una posible mala elección del modelo.

En cuanto al desequilibrio de clases, no hay mucho que se pueda hacer, debido a que inicialmente el problema se presenta bastante desbalanceado en cuanto a la distribución de los resultados finales del target, teniendo una tendencia muy alta hacia el target 4 con muchas combinaciones posibles de datos del dataset. Luego, para comprobar si ha sido una mala elección del modelo, se prueban las curvas de aprendizaje de distintos modelos disponibles en las librerías de scikit learn para luego comparar sus desempeños tanto en cuanto a la métrica como en problemas que puedan presentar en la curva de aprendizaje.

Inicialmente se prueba con un modelo LinearSVC de la biblioteca Support Vector Machine de sklearn, tomando hiperparámetros C de 1 y de 3. Este hiperparámetro busca ajustar de mejor manera los datos a su correspondiente target, a cambio de un mayor costo computacional y una posibilidad de que exista overfitting de los datos. En la figura 12 se observa la comparación entre las curvas de aprendizaje con valores de C de 1 y 3 para el modelo con LinearSVC.

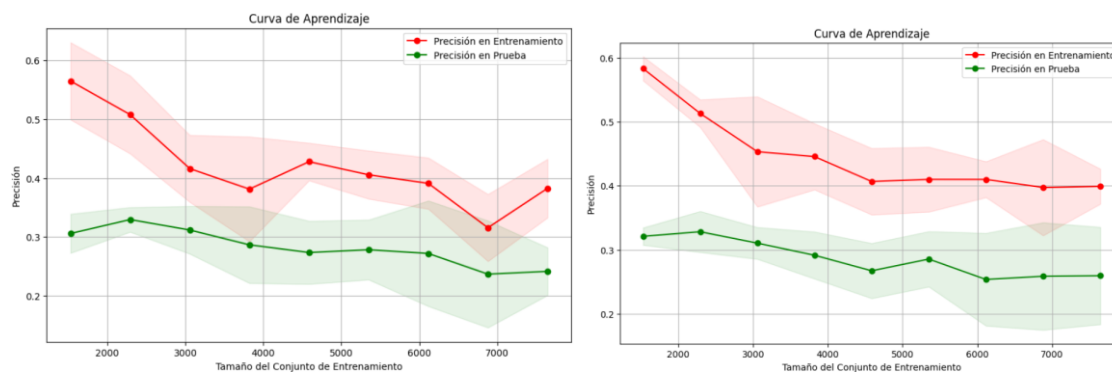


Figura 12. Curvas de aprendizaje para modelo con LinearSVC y diferentes C: a) 3 y b) 1.

En la primera curva se obtiene un mejor ajuste, con un porcentaje menor de overfitting, que tiende a converger cuando el número de muestras es ligeramente menor de 7000, pero que diverge nuevamente. En comparación con la segunda curva, tiene un valor de la métrica de desempeño ligeramente más bajo, pero se compensa con un mejor desempeño en la curva de aprendizaje. Según la tendencia observada, podría mejorar si se usan valores de C mayores.

Luego, se utiliza un modelo de LinearDiscriminantAnalysis (LDA), que tiene como objetivo reducir la dimensionalidad del problema. Para este se utilizan los hiperparámetros por defecto al considerarse apropiados para un problema con una dimensionalidad tan alta. En la figura 13 se observa el comportamiento de la curva de aprendizaje.

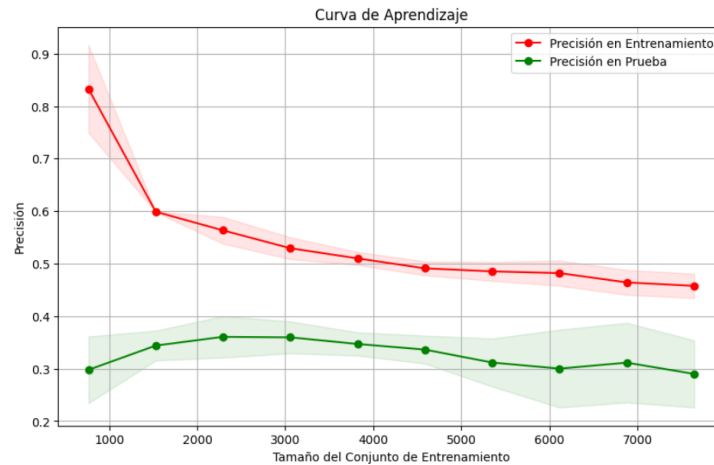


Figura 13. Curva de aprendizaje para LDA.

De la figura se observa una tendencia que empieza convergente hasta presentar un overfitting relativamente pequeño y un resultado para la métrica de aproximadamente 0,3, lo cual resulta aceptable como candidato para ser el modelo a utilizar.

El siguiente modelo evaluado fue un KNeighborsClassifier (KNN), que compara los datos con los valores más cercanos para obtener resultados y predicciones. Para este modelo se utilizaron hiperparámetros de n de 5, 10 y 20, como se detalla en la figura 14.

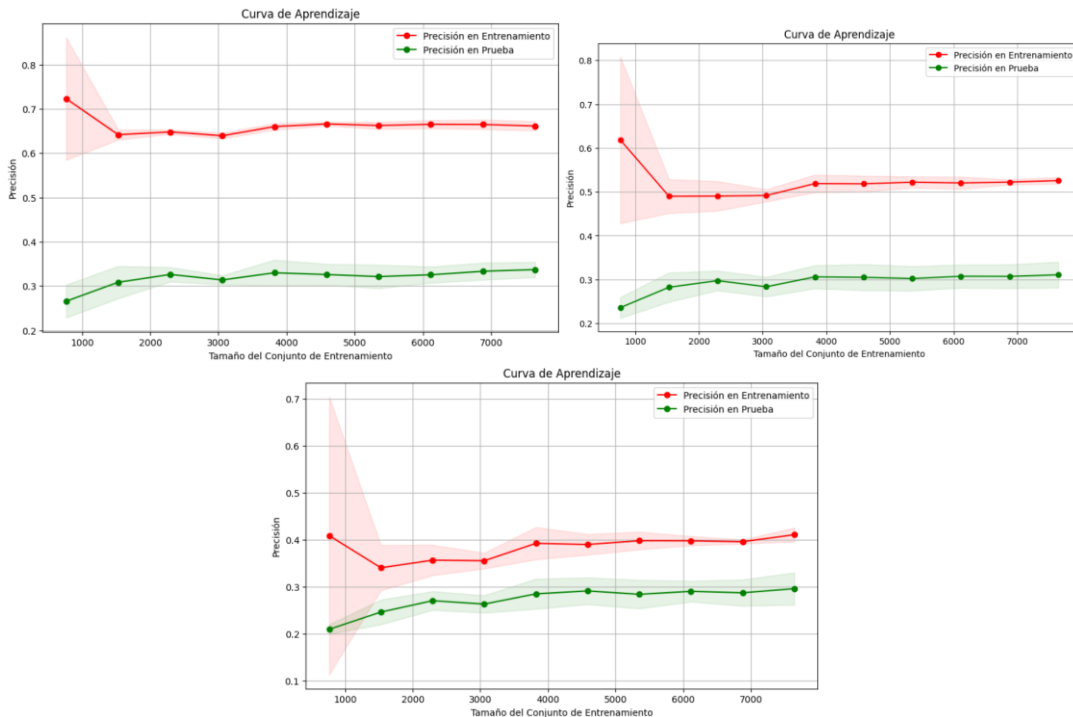


Figura 14. Curvas de aprendizaje para KNN con valores de n de a)5, b)10 y c)20.

Se observa que con un mayor número del hiperparámetro n , se tiene una curva de aprendizaje mucho mejor, con un sobre ajuste más pequeño, pero con la desventaja de que

la métrica de desempeño resulta ser algo menor en comparación con valores más pequeños.

Luego, se volvió al modelo de árbol de decisión para evaluar su curva de aprendizaje, usando un modelo simplificado y uno con hiperparámetros de mayor complejidad. En la figura 15 se muestran los resultados.

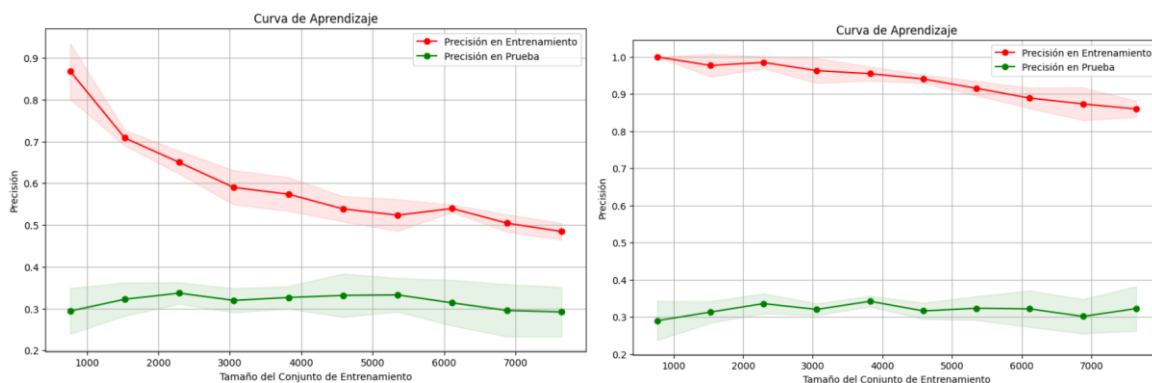


Figura 15. Curvas de aprendizaje para árbol de decisión a) simplificado y b) complejo.

En este modelo se observa claramente el resultado de simplificar el modelo para disminuir en gran medida el overfitting. Sin embargo, como se vio en la primera parte de la implementación de los modelos, la métrica de desempeño disminuye considerablemente.

Finalmente, habiendo evaluado una gran cantidad de modelos, se realizó una comparación entre los resultados obtenidos en cuanto a la métrica de desempeño y su desviación estándar, como se muestra en la figura 16.

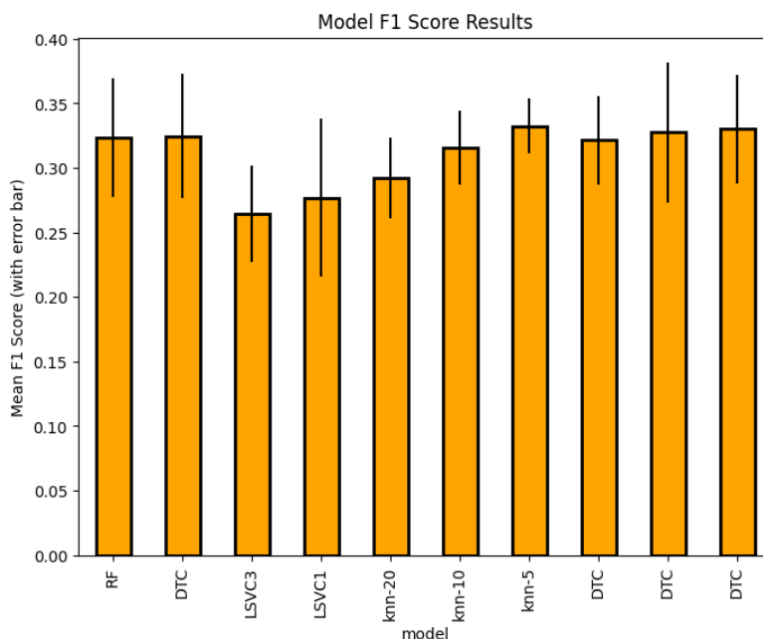


Figura 16. Comparación entre desempeño de los modelos.

En la figura no están incluidos todos los modelos evaluados, pero da una mirada general de cómo se comporta cada uno. Es importante tener en cuenta que no solo la métrica de desempeño afecta el resultado, sino posibles sobreajustes o inclinaciones hacia ciertos datos en las curvas de aprendizaje.

RETOS Y CONSIDERACIONES DE DESPLIEGUE

Para evaluar el desempeño mínimo con el cual se consideraría aceptable el modelo, se comparó con los modelos disponibles en Kaggle implementados con los competidores, donde la mayoría obtiene puntajes en 0,35 y 0,45. Al ser un equipo principiante en el tema, se consideró que un rendimiento de los modelos de aproximadamente 0,3 sería aceptable. A pesar de esto, se alcanzó un rendimiento máximo de 0,37 utilizando RandomForest, aunque debe ser tomado con cuidado, ya que presenta un overfitting muy alto.

El principal reto para el despliegue de este proyecto fue el poco conocimiento que se tenía sobre el tema de el análisis predictivo en general, los modelos, los parámetros que reciben y la interpretación de los resultados. Con ayuda de la documentación, la información disponible en el curso y repositorios de terceros, se logró implementar satisfactoriamente un análisis predictivo.

Para futuros análisis, es importante tener una mejor organización de los datos, pues existían muchas columnas redundantes, con valores nulos que perfectamente podían estar llenos y, al parecer según los resultados obtenidos, datos que no aportan o que generan ruido en la implementación de modelos de Machine learning.

CONCLUSIONES

- Se evidencia la importancia de realizar una buena exploración de los datos, buscando eliminar columnas que generen ruido a la hora de implementar el modelo y simplificar al máximo posible para obtener un mejor desempeño del modelo.
- No siempre aumentar la complejidad de un modelo garantiza un mejor desempeño. Se evidenció que puede generar problemas de overfitting y reducir el desempeño de los modelos cuando se aumenta la cantidad de datos que utilizan.
- Se evidencia la importancia de las curvas de aprendizaje para diagnosticar posibles sobreajustes o inclinaciones hacia cierto conjunto de datos en la implementación del modelo. Esto permite tener una mejor visión de cómo modificar los hiperparámetros usados para obtener mejores resultados.

BIBLIOGRAFÍA

- [1] ASHRAE – Costa Rican Household Poverty Level Prediction | Kaggle (2018). Retrieved 28 May 2023, from <https://www.kaggle.com/competitions/costa-rican-household-poverty-prediction/data>
- [2] Scikit-learn. Machine learning in Python. <https://scikit-learn.org/stable/>
- [3] Aziz, A. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool (2015). doi: [10.1186/s12880-015-0068-x](https://doi.org/10.1186/s12880-015-0068-x).