Entrega final del proyecto

POR:

Juan Sebastián Ortiz Tangarife

MATERIA:

Introducción a la Inteligencia Artificial

PROFESOR:

Raúl Ramos Pollan

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Universidad de Antioquia

Facultad de Ingeniería

Medellín 2023

INTRODUCCIÓN

Se realizará un modelo de Machine Learning en el que se intentará predecir si un cliente va a realizar o no un reclamo en cierta empresa. Para este trabajo se utilizará un dataset que contiene 957919 filas, las cuales cumplen con el requerimiento del proyecto de ser mayor a 5000. Sin embargo, en el momento de revisar los datos faltantes y las columnas categóricas, se evidencia que no se cumple con el requerimiento planteado por el profesor ya que los datos faltantes eran menor al 5% y ninguna columna del dataset era categórica.

Para darle solución a este problema, tuvo que realizarse una simulación en la cual se eliminaron datos de forma aleatoria, también se escogieron columnas al azar para hacerles la conversión a variable categórica y de esta forma cumplir con los requerimientos del proyecto.

Se realizó también un análisis exploratorio de los datos para darnos cuenta específicamente cuantos datos faltantes habían, cuantas columnas categóricas con sus respectivos valores existían y diversos análisis como por ejemplo en de la matriz de correlaciones para ver si había algún tipo de relación entre distintas variables del dataset que pudieran entorpecer nuestro modelo final de predicción.

Luego de haber hecho dicho análisis, había que arreglar el dataset y hacerle el debido procesamiento para entregarlo limpio, ya que como vimos anteriormente, tenían muchos datos faltantes y además, las variables categóricas no estaban trabajadas de la forma correcta.

Una vez tenido el dataset limpio, se escogieron dos algoritmos de algoritmos predictivos supervisados y dos algoritmos no supervisados, a los cuales se le estimaron los mejores hiper parámetros para que nuestro algoritmo de predicción fuera lo más acertado posible, y también a cada uno de esos algoritmos, se les realizó curvas de aprendizaje de las cuales hablaremos más adelante.

PREPROCESAMIENTO DEL DATASET

El reto comienza cuando vemos por primera vez el dataset y nos damos cuenta que tenemos 118 columnas que debemos trabajar. A las columnas les miraron cuántos datos faltantes tenían cada una y se evidenció que todas tienen datos faltantes, sin embargo, no los suficientes como para cumplir con el 5% del total en mínimo 3 columnas.

```
f1      15247
f2      15190
f3      15491
f4      15560
f5      15405
        ...
f114    15438
f115    15559
f116    15589
f117    15407
f118    15212
Length: 118, dtype: int64
```

Datos faltantes

Como tenemos 957919 filas en total, si realizamos el cálculo del 5% vemos que este valor es 47895, sin embargo, como vimos en la gráfica de “datos faltantes” estas filas a duras penas llegan al valor de 16000 datos faltantes, por lo tanto es necesario simular datos faltantes hasta cumplir los 48000 necesarios como mínimo.

```
dfAux=df.copy()
columna_seleccionada=["f1","f2","f3"]

for col in columna_seleccionada:
  datosEliminar=int(len(dfAux[col])*0.05)
  num_filas=len(dfAux)

  indices_aleatorios=np.random.choice(num_filas, size=datosEliminar)
  for indice in indices_aleatorios:
    dfAux.at[indice,col]=np.nan

k=dfAux.isna().sum()
k[k!=0]

ccols=[i for i in dfAux.columns if not i in dfAux._get_numeric_data()]
print(ccols)
```

Con ese pequeño algoritmo, se eliminaron de forma aleatoria el 5% de los datos de la columna f1, f2 y f3, cumpliendo de esta manera con el requisito planteado por el profesor.

```
f1      61176
f2      61159
f3      61368
f4      15560
f5      15405
        ...
f114    15438
f115    15559
f116    15589
f117    15407
f118    15212
Length: 118, dtype: int64
```

Como podemos observar, las columnas f1, f2 y f3 ahora si superan los 48000 datos faltantes necesarios.

El otro requerimiento era que se necesitaba que al menos el 10% de las columnas fueran categóricas, pero en el momento de mirar cuantas columnas categóricas tenía nuestro dataset, observamos que no tenía ninguna.

```
25] ccols = [i for i in dfAux.columns if not i in dfAux._get_numeric_data()]
    print (ccols)

    []
```

Columnas categóricas

Por tanto, para categorizar el 10% del total de columnas, que en mi caso serían 12 columnas (120*0,1=12) se procedió a realizar una función que categoriza con 1, 2 o 3 de la siguiente manera:

Si el número es menor al 33% de la longitud del intervalo en el que se mueve la variable, se categoriza como un 1; si está entre el 33,3% y el 66,6% se categoriza como un 2; y si es mayor al 66,6% se categoriza como un 3.
Por ejemplo, si tenemos una variable que toma valores entre 0 y 100, cuando tome el valor de 24, va a categorizarse como un 1.

```
def categorizarValor(valor,a,b):
  if valor<a:
    return "1"
  elif a<=valor<=b:
    return "2"
  else:
    return "3"
```

```
dfaux2=dfAux.copy()
columnasCategorizar=["f6","f10","f20","f30","f41","f51","f65","f72","f81","f90","f101","f110"]

for columna in columnasCategorizar:
  minimo=dfaux2[columna].min()
  maximo=dfaux2[columna].max()
  dfaux2[columna]=dfaux2[columna].apply(categorizarValor, args=(minimo+((maximo-minimo)*0.33),maximo-((maximo-minimo)*0.33)))
```

Una vez aplicada dicha función a las 12 columnas, se verificó que efectivamente esas 12 columnas si son ahora categóricas

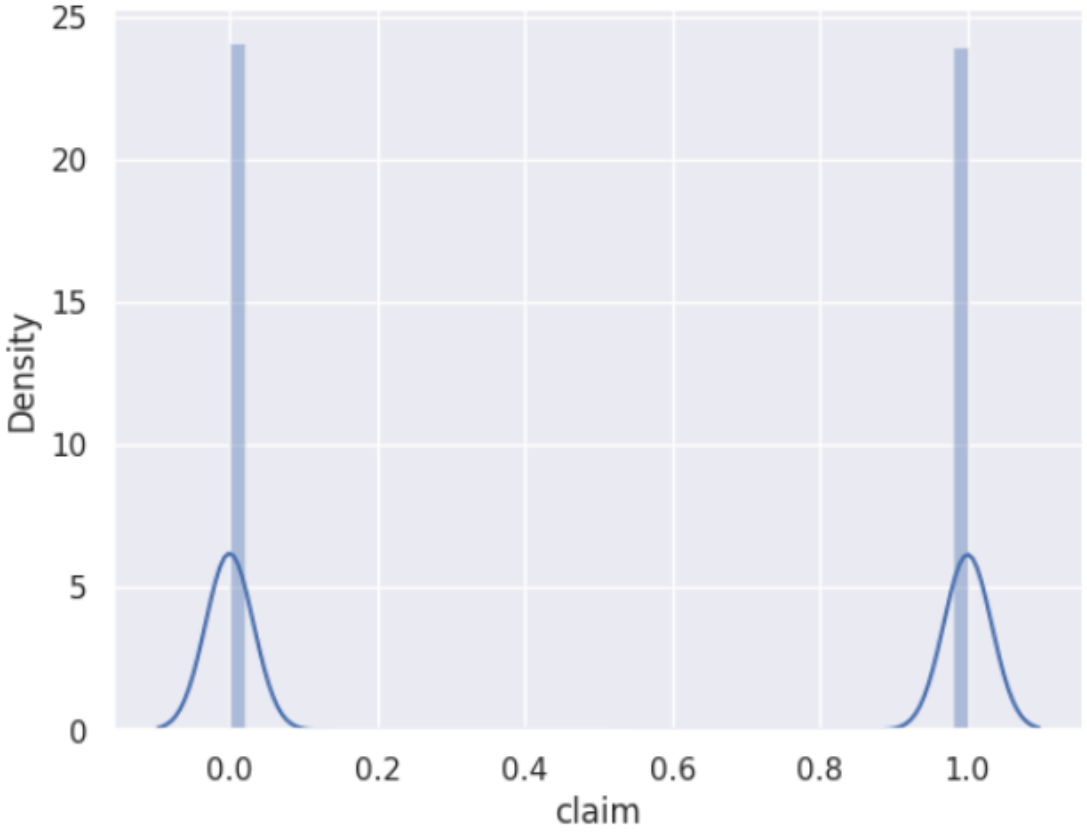
```
ccols = [i for i in dfaux2.columns if not i in dfaux2._get_numeric_data()]
print (ccols)
```

```
['f6', 'f10', 'f20', 'f30', 'f41', 'f51', 'f65', 'f72', 'f81', 'f90', 'f101', 'f110']
```

Por último se exportó el csv para poder empezar a trabajar con ese nuevo Data Frame ya preprocesado.

ANÁLISIS EXPLORATORIO DE LOS DATOS

Lo primero que se hizo fue inspeccionar la variable objetivo llamada Claim para ver qué valores toma y se evidencia que solo puede tomar dos valores: 0 y 1



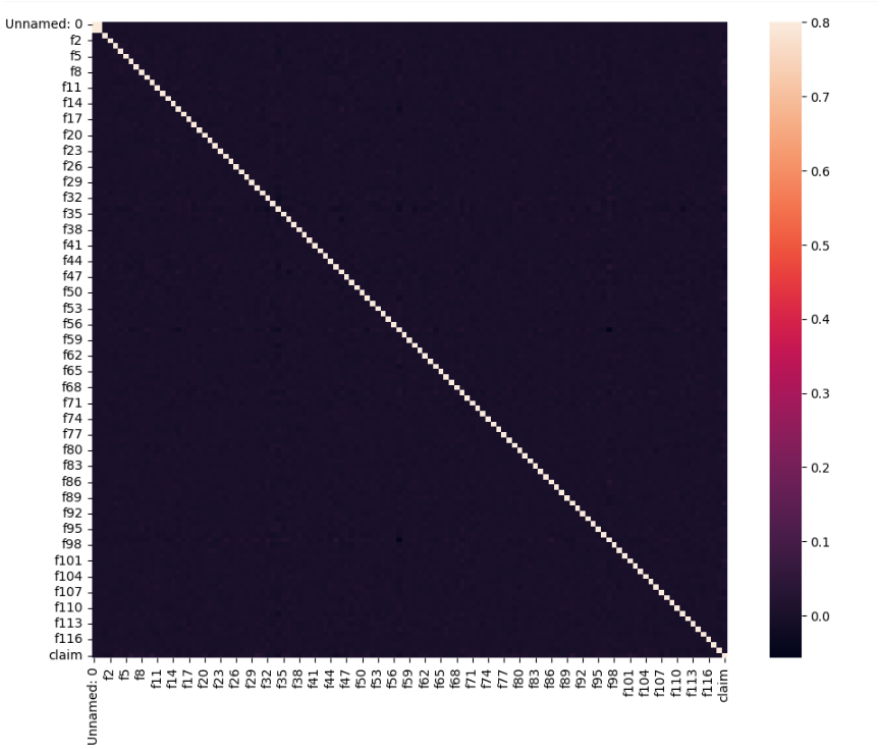
También se mostró que tipo de variable estaba contenido en cada una de las columnas del dataset y como era de esperarse, solamente tenemos de tipo “Object” a las columnas que les hicimos la conversión a categórica. En la siguiente imagen no se mostrarán todas las columnas puesto que son muchas.

```
 f4 float64
 f5 float64
 f6 object
 f7 float64
 f8 float64
 f9 float64
f10 object
f11 float64
f12 float64
f13 float64
f14 float64
f15 float64
f16 float64
f17 float64
f18 float64
f19 float64
f20 object
```

Se realizó estadística descriptiva a las columnas numéricas del dataset para extraer información relevante tal y como lo es la media, la desviación estándar, mínimos, máximos y los cuantiles.

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	884509.0	4.422540e+05	2.553359e+05	0.000000e+00	2.211270e+05	4.422540e+05	6.633810e+05	8.845080e+05
id	884509.0	4.422540e+05	2.553359e+05	0.000000e+00	2.211270e+05	4.422540e+05	6.633810e+05	8.845080e+05
f1	828031.0	9.021637e-02	4.358299e-02	-1.438400e-01	7.023850e-02	9.013900e-02	1.165300e-01	4.151700e-01
f2	828058.0	3.459269e-01	1.462597e-01	-1.904400e-02	2.828500e-01	3.891000e-01	4.584600e-01	5.189900e-01
f3	827785.0	4.066250e+03	6.413559e+03	-9.421700e+03	4.183000e+02	1.279000e+03	4.442200e+03	3.954400e+04
...	...	...	...	...	...	...	...	...
f115	870160.0	1.208859e+00	1.149654e-01	9.052700e-01	1.146800e+00	1.177200e+00	1.241900e+00	1.874800e+00
f116	870098.0	4.272355e+16	6.725324e+16	-8.396800e+15	2.321125e+14	1.327400e+16	5.275200e+16	3.249900e+17
f117	870241.0	3.958861e+03	3.155640e+03	-4.152400e+02	1.307000e+03	3.227600e+03	6.137200e+03	1.315100e+04
f118	870505.0	5.595644e-01	4.085156e-01	-1.512400e-01	2.768600e-01	4.736500e-01	7.466300e-01	2.743600e+00
claim	884508.0	4.984172e-01	4.999978e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00

También se realizó un mapa de calor para ver la correlación entre variables o columnas y mirar si existe algún problema de correlación.



Claramente se puede observar que la correlación entre columnas es menor al 10% lo cual es bueno ya que no vamos a tener información redundante que nos pueda llegar a afectar el futuro modelo que creemos.

Ahora, por parte de las columnas categóricas, miramos qué tipo de categorías guardan cada una de las columnas y como era de esperarse (porque así fue como lo hicimos en el preprocesamiento del dataset) esas columnas tienen solamente 3 tipos de valores: Bajo, Medio y Alto.
A la columna f6 se le hizo un análisis más profundo en el cual nos muestra cuántos datos son bajo, cuántos son medio y cuántos son alto.

```
['f6', 'f10', 'f20', 'f30', 'f40', 'f50', 'f60', 'f70', 'f80', 'f90', 'f100', 'f110']
        f6 ['Alto' 'Bajo' 'Medio']
       f10 ['Alto' 'Bajo' 'Medio']
       f20 ['Alto' 'Bajo' 'Medio']
       f30 ['Alto' 'Bajo' 'Medio']
       f40 ['Alto' 'Bajo']
       f50 ['Alto' 'Bajo' 'Medio']
       f60 ['Alto' 'Bajo' 'Medio']
       f70 ['Alto' 'Bajo']
       f80 ['Alto' 'Bajo' 'Medio']
       f90 ['Alto' 'Bajo' 'Medio']
      f100 ['Alto' 'Bajo' 'Medio']
      f110 ['Alto' 'Bajo' 'Medio']
Medio    798424
Alto      63834
Bajo      22251
Name: f6, dtype: int64
```

LIMPIEZA DE LOS DATOS

Para que un modelo de Machine Learning funcione y prediga de forma adecuada, es necesario entregarle los datos de la forma más limpia posible, es decir, sin datos faltantes, realizar el encoding de las variables categóricas y normalizar el dataset.

Nuestro dataset tiene varios problemas y el primero que se arregló fue el de los datos faltantes. La técnica utilizada para solucionar esto fue rellenar todos esos datos faltantes con la media de la columna.

```
#Reparar valores faltantes en columnas
k = dfClean.isna().sum()
k[k!=0]

ccols = [i for i in dfClean.columns if not i in dfClean._get_numeric_data()]
print (ccols)

for i in dfClean.columns:
    if i!="id" and i!="claim" and i not in ccols:
        media=dfClean[i].mean()
        dfClean[i]=dfClean[i].fillna(media)


k = dfClean.isna().sum()
k[k!=0]
```

```
['f6', 'f10', 'f20', 'f30', 'f40', 'f50', 'f60', 'f70', 'f80', 'f90', 'f100', 'f110']
Series([], dtype: int64)
```

Se puede observar que ahora no tenemos ningún dato faltante.

Lo último que se arregló fue realizar el encoding de las variables categóricas, es decir, que si tenemos una columna con valores bajo, medio y alto, debemos crear tres columnas nuevas para cada valor de bajo, medio y alto, y llenarlo con un 0 o un 1 según corresponda. Para ésto se utilizaron dummies que simplificaron demasiado todo este proceso.

```python
#Crear encoding para variables categóricas

ccols = [i for i in dfClean.columns if not i in dfClean._get_numeric_data()]
print (ccols)

for i in dfClean.columns:
    if i not in dfClean._get_numeric_data():
        dummies=pd.get_dummies(dfClean[i], prefix=i, prefix_sep="_", dtype=int)
        dfClean=pd.concat([dfClean,dummies],axis=1)
        dfClean = dfClean.drop(i, axis=1)


columnaRespuesta=dfClean["claim"]
dfClean=dfClean.drop("claim",axis=1)
dfClean["claim"]=columnaRespuesta

dfClean=dfClean.drop("Unnamed: 0",axis=1)
dfClean=dfClean.drop("id",axis=1)
print(dfClean.columns)
```

```
['f6', 'f10', 'f20', 'f30', 'f40', 'f50', 'f60', 'f70', 'f80', 'f90', 'f100', 'f110']
<ipython-input-3-6a66e1bace31>:15: PerformanceWarning: DataFrame is highly fragmented.
  dfClean["claim"]=columnaRespuesta
Index(['f1', 'f2', 'f3', 'f4', 'f5', 'f7', 'f8', 'f9', 'f11', 'f12',
       ...
       'f90_Alto', 'f90_Bajo', 'f90_Medio', 'f100_Alto', 'f100_Bajo',
       'f100_Medio', 'f110_Alto', 'f110_Bajo', 'f110_Medio', 'claim'],
      dtype='object', length=141)
```

Aquí se puede apreciar que la cantidad de columnas cambió, ahora tenemos 141 puesto que creamos columnas nuevas.

ALGORITMOS PREDICTIVOS SUPERVISADOS

Una vez tenemos el dataset limpio, se separan las columnas de características que se guardan en la variable X y la columna de nuestra variable respuesta llamada Claim que se guarda en la variable Y. Luego, se dividen estas variables, 80% para el train y un 20% para el test.

```
X = df.drop('claim', axis=1)  # Features
y = df['claim']  # Target variable

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

En el momento de realizar el algoritmo predictivo supervisado, se escogieron dos modelos de clasificación que permite dar solución al problema de predecir si un cliente va o no a hacer un reclamo en una empresa. Los algoritmos que se escogieron fueron el de regresión logística y el del árbol de decisiones.

- **REGRESIÓN LOGÍSTICA:**
  La primera vez que se corre el modelo de regresión logística, se hizo con los parámetros por default que trae el modelo, es decir, no se cambia ningún hiper parámetro y el resultado, por desgracia, da un accuracy del 49%, lo que indica que es incluso peor que lanzar una moneda al aire para predecir si el cliente va a realizar un reclamo.
  Para solucionar este problema, se deben buscar los mejores hiper parámetros que nos incrementen el accuracy. Para ello se utiliza una técnica gracias a una librería que se llama RandomizedGridSearchCV, la cual, por medio de un diccionario, se ponen distintos valores de hiper parámetros, el modelo itera sobre cada posible combinación y entrega cual es la mejor.

```
# Inicializar el modelo de regresión logística
model = LogisticRegression()

param_grid = {
    'fit_intercept': [True, False],
    'C': [0.01,  1],  # Regularization parameter
    'penalty': ['l1', 'l2'],  # Regularization type
    'solver': ['liblinear'],  # Solver algorithm
}

#Inicializar GridSearchCV para encontrar la mejor combinación de hiperparámetros
grid_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=5, cv=3, scoring='accuracy', random_state=42)
grid_search.fit(X_train, y_train)

# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_
print("Mejores hiperparámetros:", best_params)
```

Por último, se utiliza esta combinación para hacerle el fit al modelo de regresión logística y lastimosamente, el modelo no mejoró mucho.

```
# Entrenar el modelo con los mejores hiperparámetros
best_lr_model = LogisticRegression(**best_params)
best_lr_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = best_lr_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5048625171787958
```

- **ÁRBOL DE DECISIÓN**

Se hizo exactamente el mismo proceso que en el modelo de regresión logística y de nuevo tenemos un accuracy muy pobre

```
dt_model = DecisionTreeClassifier(random_state=42)

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5)
}

#Inicializar GridSearchCV para encontrar la mejor combinación de hiperparámetros
grid_search = RandomizedSearchCV(estimator=dt_model, param_distributions=param_grid, n_iter=7, cv=3, scoring='accuracy', random_state=42)
grid_search.fit(X_train, y_train)

# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_
print("Mejores hiperparámetros:", best_params)

# Entrenar el modelo con los mejores hiperparámetros
best_dt_model = DecisionTreeClassifier()#**best_params
best_dt_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred2 = best_dt_model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred2))
```

```
Accuracy: 0.5143147889761962
```

Sin embargo, cabe resaltar que hubo mejoría en comparación al modelo anterior.

ALGORITMOS NO SUPERVISADOS

Para los algoritmos no supervisados se utilizó redes neuronales y KMeans.

- ## REDES NEURONALES
  Al igual que en los algoritmos supervisados, también se utilizó el RandomizedSearchCV para la búsqueda de los mejores hiper parámetros. También dio un accuracy muy malo, similar a los supervisados.

```
mlp_model = MLPClassifier()

# Definir el espacio de búsqueda para hiperparámetros

param_dist = {
    'hidden_layer_sizes': [(50,),(100,),(50,50),(100,50,25)],
    'activation': ['relu', 'tanh', 'logistic'],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'max_iter': randint(50, 200)
}

# Inicializar RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=mlp_model, param_distributions=param_dist, n_iter=10, cv=3, scoring='accuracy', random_state=42)

# Realizar la búsqueda aleatoria en el espacio definido
random_search.fit(X_train, y_train)

# Obtener los mejores hiperparámetros
best_params = random_search.best_params_
print("Mejores hiperparámetros:", best_params)

# Entrenar el modelo con los mejores hiperparámetros
best_mlp_model = MLPClassifier(**best_params)
best_mlp_model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = best_mlp_model.predict(X_test)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

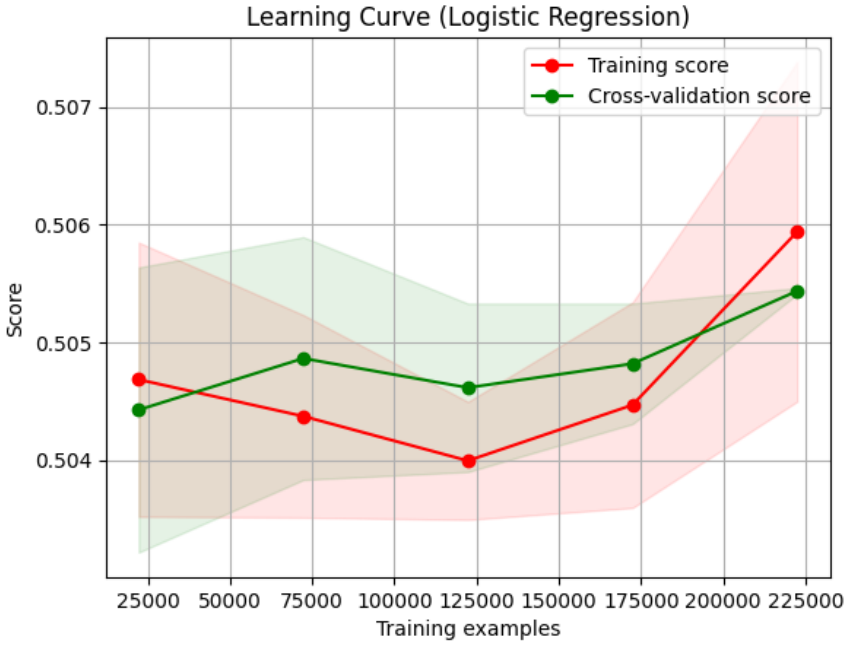
```
Accuracy: 0.5006583054012516
```

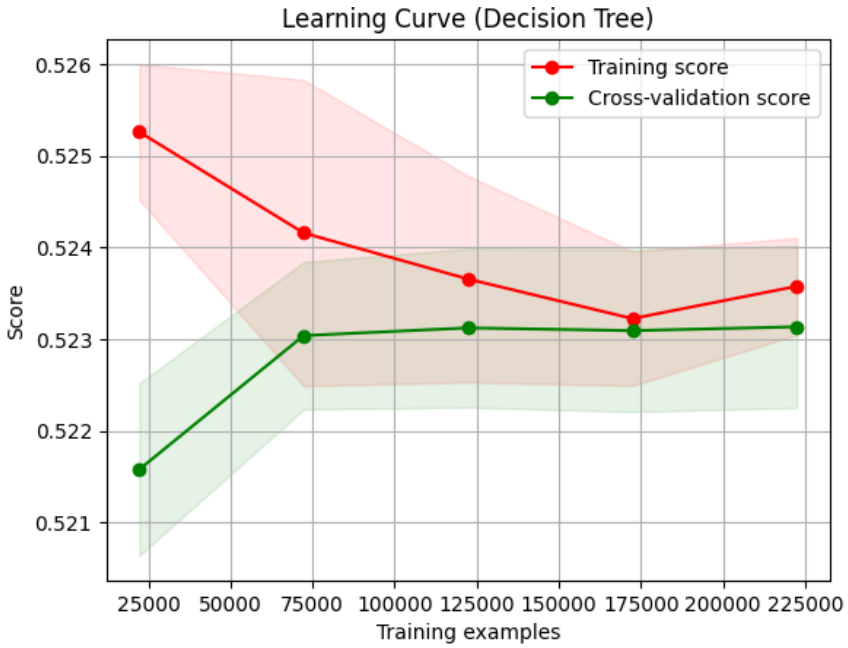
- ## KMEANS
  Como usando los mejores hiper parámetros sigue dando un accuracy malo, y los tiempos de ejecución son muy largos, se decidió realizar los KMeans con los hiper parámetros por default para ahorrar tiempo

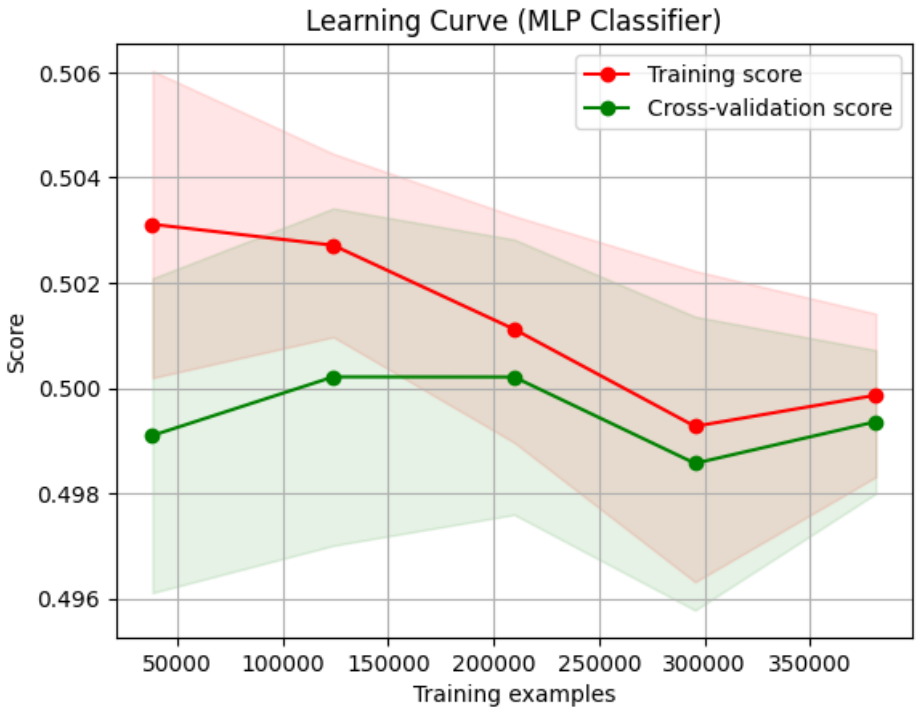
```
# Preprocesamiento: Escalar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Aplicar K-Means con el número de clusters deseado
n_clusters = 1 # Puedes ajustar este valor según tus necesidades
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
labels = kmeans.fit_predict(X_scaled)

# Evaluar el rendimiento del clustering usando el accuracy
# Aquí, necesitarías comparar los clusters asignados por K-Means con las etiquetas reales
# Sin embargo, en clustering no hay una "verdad" conocida, por lo que el accuracy puede no ser la mejor métrica
# Puedes explorar otras métricas como la pureza o el índice de Rand ajustado.
# Aquí, solo se proporciona como una métrica simple para ilustrar el concepto.
accuracy = accuracy_score(y, labels)
print("Accuracy del clustering:", accuracy)
```

```
Accuracy del clustering: 0.5015002035241145
```

CURVAS DE APRENDIZAJE







La curva de aprendizaje de regresión logística tiene una tendencia a incrementar más si se tienen más datos.
Por otra parte tenemos al árbol de decisión que se estabiliza alrededor de los 75000 datos, es decir, que a partir de este número es irrelevante la información nueva que reciba.
Por último, el modelo de redes neuronales tiende a ser cada vez peor con más datos, alcanzando su mejor puntuación entre los 120000 y los 200000 datos, pero si incrementamos más, su rendimiento empieza a empeorar.

EVALUACIÓN DIAGNÓSTICA

1. **Diagnóstico:** Todos los modelos implementados tienen problema de sub-ajuste ya que tanto la curva de training score como la de cross validation score tienen un valor muy bajo de aproximadamente 0.5. También puede ser señal de que tenga un problema de sobreajuste severo pero es muy improbable ya que los modelos que se usaron tienen hiper parámetros muy poco complejos, lo cual debería rechazar la hipótesis de que estén sufriendo un problema de sobreajuste.

2. **Recomendación:** Dado que los modelos tienen problema de sub-ajuste, es recomendable incrementar la complejidad de los modelos, cambiando los hiper parámetros de tal forma que el modelo pueda aprender mejor.
   Aunque, de manera subjetiva, y con mi poco conocimiento que tengo sobre Machine Learning, si tuviera que decir cual es el principal problema que están sufriendo los modelos, es que desde el inicio se hizo un borrado de datos que pudo alterar significativamente la información relevante que está entregando el dataset. También, al realizar el cambio de variable numérica a variable categórica, también hubo una pérdida de información relevante a una escala inmensa ya que fueron el 10% de las columnas a las que se le hizo ese tratamiento.
   Una solución, que no se puede hacer para la entrega de este proyecto, es realizar nuevamente la limpieza de datos pero trabajando la información como realmente es, si es numérica tratarla como numérica, sin crear categorías que puedan generar una pérdida de información. Seguramente así el accuracy de los modelos va a incrementar, no creo que sea perfecto, pero más que un 50% es seguro.

3. **Evaluación:** Dado que tenemos un problema de clasificación binaria, donde el 50% de accuracy sería lanzar una moneda al aire para decidir si una persona va a realizar o no un reclamo, es inadmisible este modelo de predicción. Para que pueda ser desplegado en producción debe como mínimo tener un accuracy del 75% y tener ¼ de posibilidades de que el modelo de un resultado erróneo. Mientras está en producción debe monitorearse en cada momento las ventas de la empresa porque si en algún momento empiezan a caer abruptamente pero el modelo parece funcionar de forma adecuada, quiere decir que predecir los reclamos o bien no son relevantes para la empresa, o se están adoptando estrategias equivocadas para mantener a los clientes satisfechos.

CONCLUSIONES

- La fase más importante del proceso de Machine Learning es la limpieza de los datos ya que si se hace de forma incorrecta, puede llegarse a dar una pérdida de información relevante para el modelo que puede llegar a dar un bajón en el puntaje de accuracy.
- Pasar una columna numérica a categórica repercute de forma negativa en el modelo ya que tendremos gran pérdida de información.
- Entre más complejos son los hiper parámetros del modelo, más tiempo tardó el algoritmo en ejecutarse.
- Un modelo muy complejo puede llegar a dar problemas de sobreajuste.
- Modelos poco complejos y con poca información relevante en los datos (mi caso) generan problemas de sub-ajuste.