Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Puebla

Inteligencia Artificial para la ciencia de datos TC3007C

Reto

Momento de Retroalimentación: Reto Modelo y Refinamiento

Fernando Jiménez Pereyra	A01734609
Daniel Flores Rodríguez	A01734184
Alejandro López Hernández	A01733984
Daniel Munive Meneses	A01734205

8 de noviembre de 2022

Modelos seleccionados

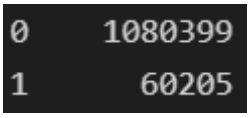
Modelo a emplear	Características	Requerimientos	Hiperparámetros
Random Forest	Consta de un conjunto grande de árboles de decisión ensamblados. Cada árbol es considerado como una clase con diferentes características que arrojan una predicción, y aquella clase con una predicción más asertiva será la que el modelo tome como la predicción final.	Las predicciones y errores de cada árbol individual debe tener poca correlación entre sí y respecto a otros árboles	Profundidad máxima de los árboles, cantidad máxima de hojas por árbol, distribución de pesos de cada hoja de árboles
Regresión Logística	Muestra la relación entre una característica, para posteriormete calcular la probabilidad de un resultado determinado.	Unicamente es valido para modelo en los que la variable dependiente es dicotómica (binaria).	El intercepto y los pesos que multiplican cada una de las variables independientes.
MLP Classifier	En un MLP, los perceptrones (neuronas) se apilan en múltiples capas. Cada nodo de cada capa está conectado a todos los demás nodos de la siguiente capa. No hay conexión entre los nodos dentro de una sola capa. Una MLP es una Red Neuronal Completamente (Densamente) Conectada (FCNN). Entonces, usamos la clase Dense() en Keras para agregar capas. En un MLP, los datos se mueven desde la entrada hasta la salida a través de capas en una dirección (hacia adelante). Un MLP también se conoce como Feed-Forward Neural Networks (FFNN) o Deep Feed Forward Network (DFFN) en alguna literatura. Un MLP es un tipo de modelo secuencial. Entonces, usamos la clase Sequential() en Keras para construir ML	MLPClassifier entrena iterativamente ya que en cada paso de tiempo se calculan las derivadas parciales de la función de pérdida con respecto a los parámetros del modelo para actualizar los parámetros. También se puede agregar un término de regularización a la función de pérdida que reduce los parámetros del modelo para evitar el sobreajuste. Esta implementación funciona con datos representados como matrices numpy densas o matrices scipy dispersas de valores de punto flotante.	Número de capas ocultas en la red Número de nodos en cada capa oculta Tipo de función de pérdida Tipo de optimizador Tipo de métrica de evaluación Tipo de función de activación en cada capa oculta Tamaño del lote Número de épocas Tasa de aprendizaje

Set de datos a utilizar

El set de datos empleado principal empleado durante el reto refiere a una lista de clientes de una compañía de telecomunicaciones; en la cual se desea saber si un cliente cancelaría su servicio con la empresa (churn) o no, de acuerdo a su perfil.

El set de datos cuenta con 1140604 filas que contienen información de 28 columnas; 5 variables categóricas por 23 variables numéricas. Respecto a la proporción de clases

de predicción, siendo no churn un valor a 0 y churn valor a 1, los clientes se distribuyen de la siguiente manera:



Lo que representa que apenas el 5.27% de clientes haría churn originalmente. Tal y como está el set de datos, es probable que algunos modelos presenten problemas debido a que es un problema de una proporción muy desbalanceada entre clases de predicción.

Link al dataset: https://www.kaggle.com/datasets/mark18vi/telecom-churn-data

Métricas de comparación entre modelos

Matriz de confusión

Una matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real., o sea en términos prácticos nos permite ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos. Para este caso de clasificación en el cuál la predicción se basa principalmente en resultados de 0 o 1, la matriz de confusión a analizar será de dimensión 2x2; en donde se tendrán 4 sectores correspondientes a positivos verdaderos, positivos falsos, negativos falsos y negativos verdaderos.

K-Folds Validation

La Validación Cruzada o k-fold Cross Validation consiste en tomar los datos originales y crear a partir de ellos dos conjuntos separados: un primer conjunto de entrenamiento

(y prueba), y un segundo conjunto de validación. Luego, el conjunto de entrenamiento se va a dividir en k subconjuntos y, al momento de realizar el entrenamiento, se va a tomar cada k subconjunto como conjunto de prueba del modelo, mientras que el resto de los datos se tomará como conjunto de entrenamiento.

Este proceso se repetirá k veces, y en cada iteración se seleccionará un conjunto de prueba diferente, mientras los datos restantes se emplearán, como se mencionó, como conjunto de entrenamiento. Una vez finalizadas las iteraciones, se calcula la precisión y el error para cada uno de los modelos producidos, y para obtener la precisión y el error final se calcula el promedio de los k modelos entrenados.

Matriz de probabilidades de predicción

Es una matriz que refiere a la probabilidad de predicción de cada una de las clases por fila. En este caso, se refiere a la probabilidad que tiene cada cliente de hacer o no hacer churn. Esta matriz se compone de dos valores en un rango de 0 a 1. Esta matriz se emplea para determinar rangos de probabilidades bajas, medias o altas de que un cliente haga churn dados porcentajes introducidos por algún usuario. La función que engloba este aspecto es la siguiente

```
#function to set each client according to low, mid or high churn chance
(given low and high percentages by the user)
def showProbabilities(low,mid,high):
    clients_permanent = []
    clients_low = []
    clients_mid = []
    clients_high = []
    i = 0
    #for each client in the data set
    for client in proba_matrix:
        #get all their data and their churn chance into one list
        client_index = x_test.index[i]
        client_info = x_test.loc[client_index].values
```

```
        client_info = np.append(client_info,client[1])
           #store client data into profiles(permanent, low, mid, high)
list
        if client[1] < low:
            clients_permanent.append(client_info)
        elif client[1] < mid:
            clients_low.append(client_info)
        elif client[1] < high:
            clients_mid.append(client_info)
        else:
            clients_high.append(client_info)
        i += 1
    return clients_permanent, clients_low, clients_mid, clients_high
```

Implementación Random Forest

La implementación del modelo Random Forest quedó de la siguiente manera y con los siguientes hiper parámetros: profundidad máxima por árbol 3, random state de 1, sin definición de hojas máximas por árbol ni distribución de pesos definida El resto de hiper parámetros necesarios para el modelo quedaron en su configuración por default

```
random_forest = RandomForestClassifier(max_depth = 3, random_state = 1,
max_leaf_nodes = None, class_weight = None)
```

K Folds Validation

El K Folds validation, tanto para el Random Forest como para el resto de modelos, fue implementado a manera de 5 divisiones del data set original, así como también un acomodo aleatorio entre los datos. Posteriormente, se calculó el score que tendría el ajuste del modelo en cada una de las divisiones para comprobar que su comportamiento fuera estable.

```
kfold = KFold(n_splits = 5, random_state=42, shuffle=True)
cv_results = cross_val_score(random_forest, x,y, cv = kfold,
scoring='accuracy', verbose = 10)
```

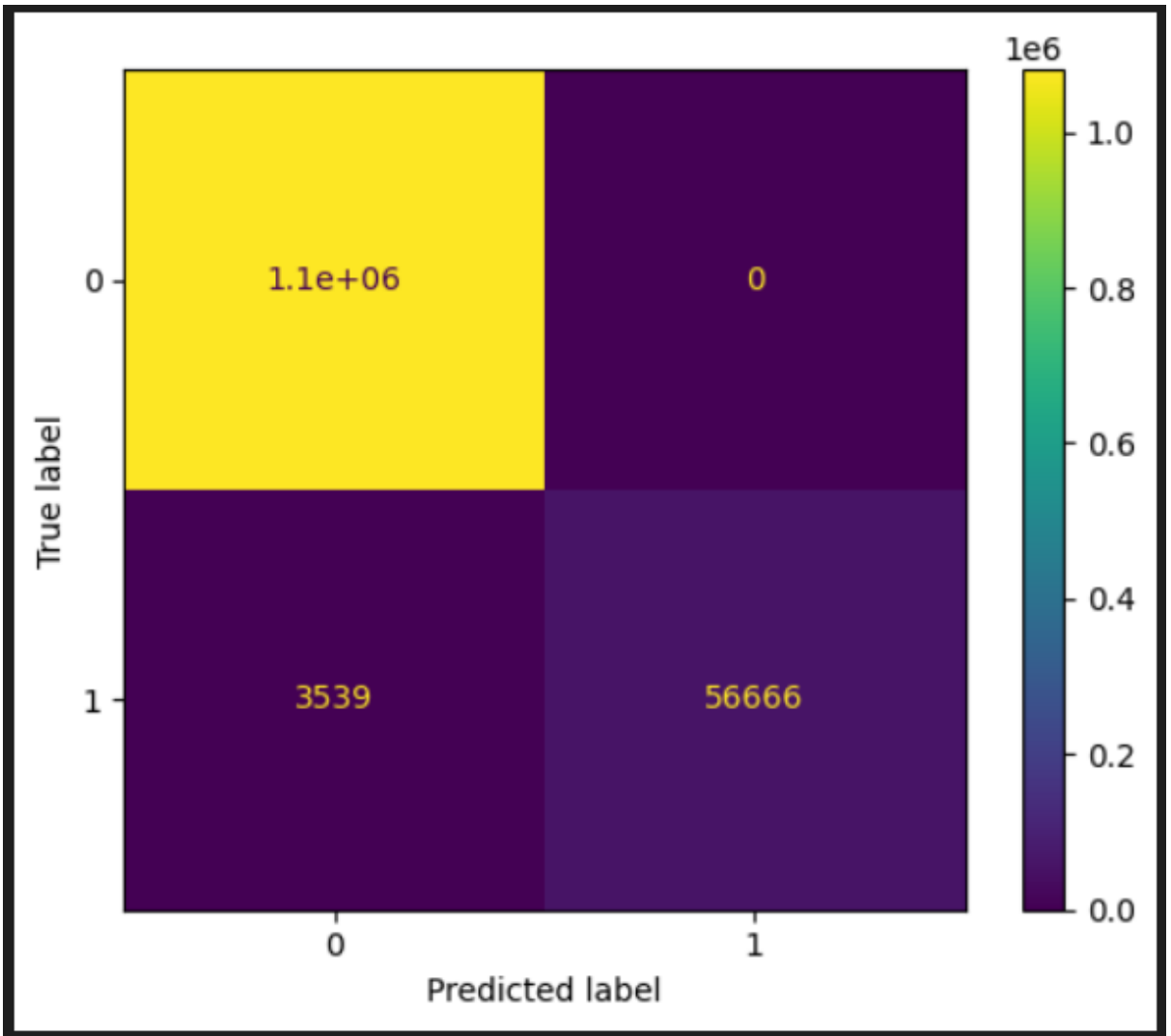
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] START ....................................................................
[CV] END ............................... score: (test=0.997) total time= 2.0min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.0min remaining:    0.0s

[CV] END ............................... score: (test=0.998) total time= 1.8min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  3.8min remaining:    0.0s

[CV] END ............................... score: (test=0.998) total time= 1.8min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  5.6min remaining:    0.0s

[CV] END ............................... score: (test=0.994) total time= 1.9min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  7.5min remaining:    0.0s

[CV] END ............................... score: (test=0.998) total time= 2.2min

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  9.7min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  9.7min finished
```

Como se puede observar, en las 5 divisiones se mantuvo un puntaje bastante alto con ligeras variaciones; por lo que en este rubro el modelo mantiene cierta tendencia sin importar la división en los datos tanto para train como para test. Posteriormente, se obtuvo la media de los puntajes entre las 5 divisiones, así como también su desviación estándar.

```
print(cv_results.mean(), cv_results.std())
✓ 0.1s
0.9968972587657191 0.0013753030947746782
```

Matriz de confusión

Una vez haciendo el ajuste del modelo con un set de datos de entrenamiento y elaborando una predicción a través de cross validation, se obtuvo la matriz de confusión. El cuadrante (0,0) representa a los Positivos Verdaderos, el cuadrante (1,0) a los Positivos Falsos, el cuadrante (0,1) a los Negativos Falsos, y el cuadrante (1,1) a los Falsos Positivos); considerando que por Positivo se refiere a la clase no churn y Negativo a churn.




```
a,b,c,d = cm.ravel() ···
```

```
(1080399, 0, 3539, 56666)
```

Como se puede observar, la gran mayoría de las predicciones apuntan hacia los positivos verdaderos; mientras que en el caso de los negativos, casi todos se encuentran en los negativos verdaderos. Por lo que el modelo de random forest predice en un 100% a los clientes que no harán churn, y predice a los clientes que harán churn en un 94.12% aproximado de confianza.

```
Precission in true positives (A): 1.0  Precission in true negatives (D): 0.9412175068515904
```

Probabilidades de clasificación

Por otro lado, los resultados de las probabilidades de cada clase a partir de la función previamente mostrada son los que se muestran a continuación, teniendo todavía un gran concentrado de clientes en el apartado no churn; mientras que el resto se va diluyendo en rangos de bajo, medio o alto el porcentaje de probabilidad que tienen de cancelar su servicio con la empresa.

```
clients_permanent,    clients_low,    clients_mid,    clients_high    =
showProbabilities(.30,.60,.80)
```

```
1080398  clients have no churn chances
59755  clients have low churn chances
420  clients have mid churn chances
31  clients have high churn chances
```

Implementación Regresión Logística

La implementación de la regresión logística se implementó de la siguiente manera, optando por dejar todas las configuraciones predeterminadas que la librería scikit learn otorga con tal de poder tener un modelo más neutro y ver su comportamiento con el set de datos; el único parámetro modificado es la declaración de iteraciones

máximas para la conversión del algoritmo de solución empleado por la regresión logística, en este caso se trata del algoritmo lbfgs

```
logistic_regression = LogisticRegression(random_state=0, max_iter = 300)
```

K Folds Validation

La implementación del K Folds validation se mantiene igual que en el Random Forest, 5 splits aplicado a todo el set de datos.

```
kfold = KFold(n_splits = 5, random_state=42, shuffle=True)
cv_results = cross_val_score(logistic_regression, x,y, cv = kfold, scoring='accuracy', verbose = 10)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] START ....................................................................
[CV] END ............................... score: (test=0.947) total time=   7.9s
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    8.0s remaining:    0.0s

[CV] END ............................... score: (test=0.948) total time=   8.4s
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:   16.6s remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time=  18.6s
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   35.3s remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time=   9.9s
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:   45.4s remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time=  13.6s

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   59.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   59.1s finished
```

A través de los splits, el comportamiento de la regresión logística se mantiene prácticamente constante; dando un acierto del 94.72%

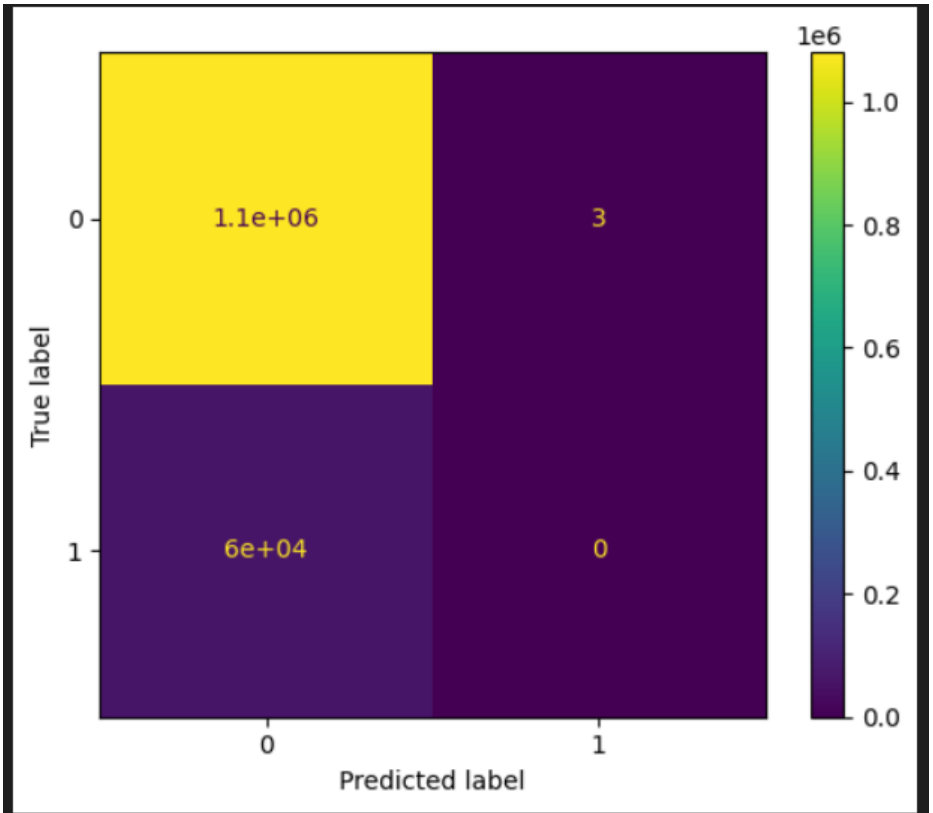
```
print(cv_results.mean(), cv_results.std())
```

✓ 0.6s

```
0.9472139316931178 0.0005364297345475543
```

Matriz de confusión

Sin embargo, al observar la matriz de confusión se puede observar que prácticamente está omitiendo los negativos verdaderos en su totalidad; es decir, no está considerando correctamente en la clasificación a aquellos clientes que harían churn. Esto se da debido al desbalance considerado de datos respecto a la clasificación como tal.




```
a,b,c,d = cm.ravel() ···
```

```
(1080396, 3, 60205, 0)
```

```
Precission in true positives (A): 0.9999972232480778  Precission in true negatives (D): 0.0
```

Probabilidades de clasificación

A raíz de lo observado en la matriz de confusión, se puede deducir que las probabilidades estarán prácticamente erróneas, teniendo prácticamente a todos los clientes en la probabilidad de churn nulo.

```
1140590  clients have no churn chances
14  clients have low churn chances
0  clients have mid churn chances
0  clients have high churn chances
```

Implementación MLP Classifier

La red neuronal empleada como tercer modelo consta de 2 capas gemelas de 50 nodos, 80 iteraciones máximas para la conversión del algoritmo de solución Adam, random state de 1 y learning rate inicial de 0.001.

```
clf = MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=80, activation = 'relu', solver = 'adam', random_state=1, learning_rate_init= 0.001)
```

K Folds Validation

La implementación del K Folds validation se mantiene igual que en los dos modelos anteriores, 5 splits aplicado a todo el set de datos.

```
kfold = KFold(n_splits = 5, random_state=42, shuffle=True)
cv_results = cross_val_score(clf, x,y, cv = kfold, scoring='accuracy', verbose = 10)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] START ....................................................................
[CV] END ............................... score: (test=0.947) total time=11.6min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 11.6min remaining:    0.0s

[CV] END ............................... score: (test=0.948) total time= 9.7min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 21.3min remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time=12.5min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 33.8min remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time= 7.0min
[CV] START ....................................................................

[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 40.9min remaining:    0.0s

[CV] END ............................... score: (test=0.947) total time=14.4min

[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 55.3min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 55.3min finished
```

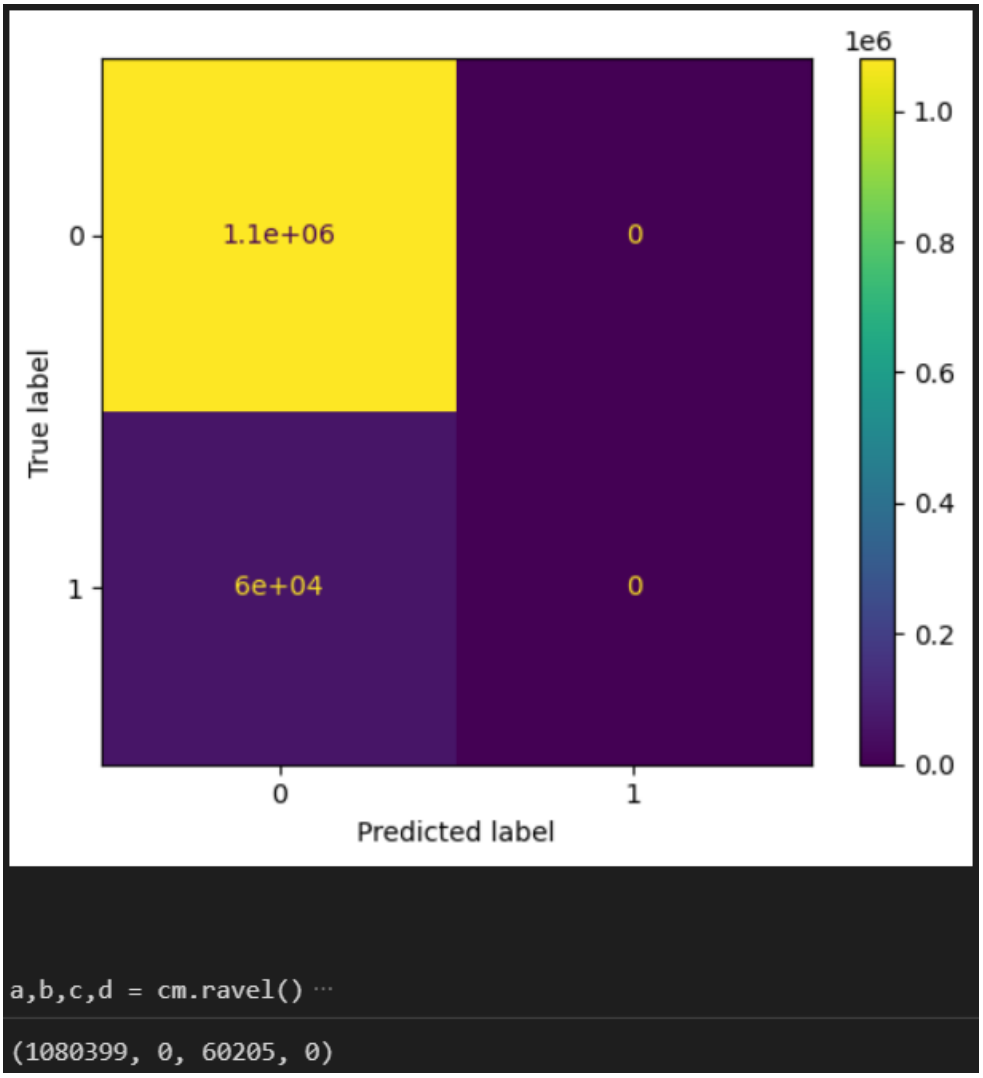
A través de los splits, el comportamiento de la red neuronal se mantiene prácticamente constante; dando un acierto del 94.72%. Al tener exactamente el mismo porcentaje que la regresión logística, se intuye que ambos modelos tienen el mismo rendimiento erróneo a pesar de contar con un porcentaje tan alto.

```
print(cv_results.mean(), cv_results.std())
✓ 0.1s
0.9472165618762224 0.0005373920116063629
```

Matriz de confusión

Al observar la matriz de confusión se puede observar que prácticamente está omitiendo los negativos verdaderos en su totalidad; es decir, no está considerando

correctamente en la clasificación a aquellos clientes que harían churn, tal y como la regresión logística. Esto se da debido al desbalance considerado de datos respecto a la clasificación como tal.




```
Precission in true positives (A): 1.0  Precission in true negatives (D): 0.0
```

Probabilidades de clasificación

A raíz de lo observado en la matriz de confusión, se puede deducir que las probabilidades estarán prácticamente erróneas, a todos los clientes en la probabilidad de churn nulo.

```
1140604  clients have no churn chances
0  clients have low churn chances
0  clients have mid churn chances
0  clients have high churn chances
```

Conclusiones

Evidentemente se trata de un set de datos limitado en cuanto a la cantidad de filas que maneja, sin embargo resulta interesante trabajar con él debido a las características que presenta al tener muchas variables numéricas y pocas categóricas. A raíz de ello, es probable que muchos de los modelos vistos puedan presentar overfitting al momento de ajustarse a este dataset, o presentar poco funcionamiento debido al desbalance de los datos.

Tanto la regresión logística como la red neuronal clasificatoria presentaron problemas al trabajar con las características del set de datos en cuestión. Al haber un desbalance importante entre las clases disponibles, ambos modelos prácticamente estaban omitiendo todos los casos en los cuales la clase a predecir resultaba churn. Si bien mostraba una efectividad bastante alta e igual para ambos modelos, no representaba nada si se indaga un poco más en la matriz de confusión y en las probabilidades de cada clase.

Por otro lado, el Random Forest resultó una mejor opción que los modelos previamente mencionados ya que, al basarse en árboles de decisión nulamente correlacionados entre sí, forzosamente debe considerar todas las clases presentes en la predicción. Por lo tanto, el Random Forest resulta mucho mejor para set de datos con desbalance significativo.