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 posteriormente calcular la probabilidad de un resultado determinado.	Únicamente es válido para modelo en los que la variable dependiente es dicotómica (binaria).	El intercepto y los pesos que multiplican cada una de las variables independientes.
Red Neuronal Convolucional (CNN)	La CNN es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos y "ver". Se puede implementar en 1 dimensión para el análisis de clasificación poco demandante como texto, o en 2 dimensiones para análisis de patrones o imágenes Cuenta con gran robustez para el análisis de los datos La cantidad de capas, la jerarquía y su estructura queda totalmente a consideración del usuario	CNN 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.	• Definición de la red en capa de input, capas convolutivas, MaxPooling, Flatten y capas de activación • Tamaño de batch • Épocas • Verbosidad • Datos de validación • Algoritmo de distribución de pesos • Algoritmo de pérdida • Métricas a considerar para cada época

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. Cabe considerar que este set de datos ya pasó por un proceso de limpieza de datos previamente mencionado, el cual basa su funcionamiento en label encoding e imputación por PCA.

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:

```
0    1080399
1      60205
```

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

Métodos de validación de datos

K-Folds Cross 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.

Data Balancing

Un conjunto de datos desbalanceado es uno donde el número de observaciones pertenecientes a un grupo o clase es significativamente mayor que las pertenecientes a las otras clases. En este caso, la gran mayoría de muestras cuentan con un target de 0; es decir, de clientes que potencialmente no harían churn. Empleando la función StandardScaler de la librería scikit learn estandariza los datos eliminando la media y escalando los datos de forma que su varianza sea igual a 1, lo cual sería una técnica para poder evitar un set de datos tan desbalanceados y modelos sesgados por el mismo desbalance.

Métricas de comparación entre modelos

Accuracy en test (Validation score)

La fracción de predicciones que el modelo realizó correctamente. Se representa como un porcentaje o un valor entre 0 y 1. Es una buena métrica cuando tenemos un conjunto de datos balanceado, esto es, cuando el número de etiquetas de cada clase es similar.

Mean Squared Error (MSE)

MSE básicamente mide el error cuadrado promedio de nuestras predicciones. Para cada punto, calcula la diferencia cuadrada entre las predicciones y el objetivo y luego promedia esos valores. Cuanto mayor sea este valor, peor es el modelo.

Mean Absolute Error (MAE)

El error absoluto medio o MAE es un puntaje lineal, lo que significa que todas las diferencias individuales se ponderán por igual en el promedio. Por ejemplo, la diferencia entre 10 y 0 será el doble de la diferencia entre 5 y 0.

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
```

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
```

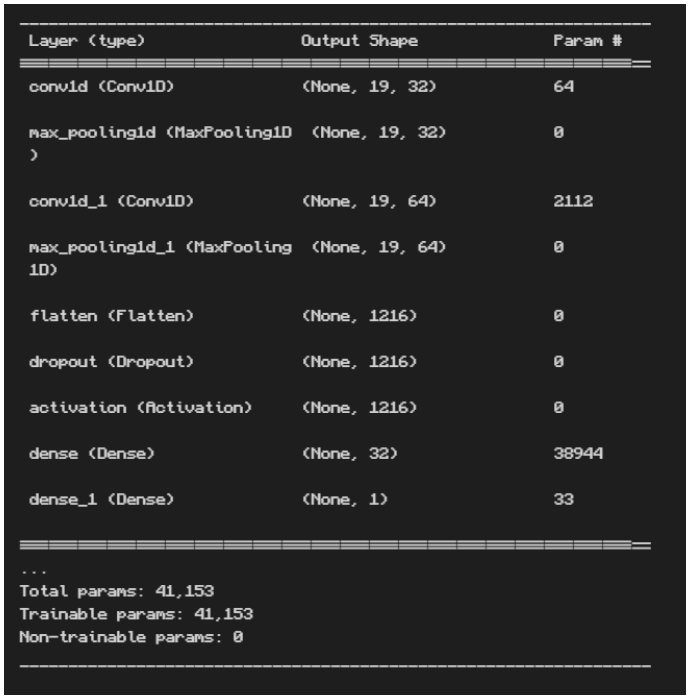
A través de los splits, el comportamiento de la regresión logística se mantiene prácticamente constante; dando un acierto del 94.72%

```
print(cv_results.mean(), cv_results.std())
✓ 0.6s
0.9472139316931178 0.0005364297345475543
```

Implementación CNN

La red neuronal empleada como tercer modelo consta con una entrada de tamaño (19,1) correspondiendo a la cantidad de columnas con el set de datos Se tienen dos capas convolutivas de 1 dimensión, seguidas de MaxPooling con tamaño 1. En adición de ello, se aplica Flatten y se determina un Dropout de 0.5 para evitar tanto overfitting. Asimismo, la función de activación corresponde a ReLU y la última casa emplea sigmoide, debido a las características de clasificación binaria (0 o 1 respecto al churn). La implementación y el resumen del modelo se muestra a continuación.

```
K.clear_session()
model = Sequential()

model.add(Conv1D(filters = 32, kernel_size = 1,padding = 'Same',activation ='relu', input_shape=(19, 1)))
model.add(MaxPooling1D(pool_size=(1)))

model.add(Conv1D(filters = 64, kernel_size = 1,padding = 'Same',activation ='relu'))
model.add(MaxPooling1D(pool_size=(1)))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Activation('relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=['accuracy'])
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 19, 32)            64

max_pooling1d (MaxPooling1D  (None, 19, 32)            0
)

conv1d_1 (Conv1D)            (None, 19, 64)            2112

max_pooling1d_1 (MaxPooling  (None, 19, 64)            0
1D)

flatten (Flatten)            (None, 1216)              0

dropout (Dropout)            (None, 1216)              0

activation (Activation)      (None, 1216)              0

dense (Dense)                (None, 32)                38944

dense_1 (Dense)              (None, 1)                 33

=================================================================
...
Total params: 41,153
Trainable params: 41,153
Non-trainable params: 0
_________________________________________________________________
```

Comparativa entre modelos antes de Data Balancing

Modelo	Score (%)	MSE train	MSE test	MAE train	MAE test
Random Forest	99.52	0.0048	0.0047	0.0048	0.0047
Regresión Logística	94.72	0.04924	0.04922	0.1003	0.1004
CNN 1D	94.75	0.04915	0.04916	0.10047	0.10045

Comparativa entre modelos tras Data Balancing

Modelo	Score (%)	MSE train	MSE test	MAE train	MAE test
Random Forest	99.52	1.5487x10^-29	1.5455x10^-29	1.4389x10^-15	1.5445x10^-15
Regresión Logística	100	1.7132x10^-30	1.6947x10^-30	3.2427x10^-15	3.2384x10^-16
CNN 1D	100	1.5487x10^-29	1.5455x10^-29	1.4389x10^-15	1.4245x10^-15

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 convolutiva 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. Ya con el balance de datos, ambos modelos presentaron unos mejores coeficientes y rendimiento; específicamente la CNN.

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. En adición a ello, el tiempo de ajuste que se tiene es corto en comparación al CNN, el cual puede terminar de ajustarse en 30 minutos; mientras que el random forest sólo en 4 minutos. Considerando que parte del reto es elaborar un sistema el cual permite entrenar un modelo en una plataforma web cuando se le introduzcan nuevos sets de datos, el tiempo de ajuste es fundamental: por lo que el Random Forest resulta una mejor elección para implementarse en este proyecto.