Predicción de pérdida de clientes en Empresas de Servicios de Telecomunicaciones (Churn)

Reducir las salidas y desercione de clientes se ha convertido en una alta prioridad para la mayoría de los proveedores de servicios de comunicaciones a medida que los mercados maduran y la competencia se intensifica.

En este documento usaremos una base de datos de una empresa de telecomunicaciones anónima disponibilizada por IBM.

El principal objetivo es crear un model de aprendizaje automático basado en KNN para predecir la pérdida o salida de clientes en una empresa de telecomunicaciones.

Librerías

In [ ]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
```

Base de datos

Este conjunto de datos contiene un total de 7043 clientes y 21 características de los mismos. De las entradas, 5174 son clientes activos y 1869 son clientes que la empresa ha perdido. Observe que el conjunto de datos está desbalanceado pues por cada cliente perdido existe casi 3 clientes activos. La variable de salida para nuestro modelo de machine learning será `Churn`.

In [ ]:
```
# importamos dataset
DATA_PATH = "https://raw.githubusercontent.com/carlosfab/dsnp2/master/datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv"
df = pd.read_csv(DATA_PATH)

# vemos las primeras 5 filas
df.head()
```

Out[ ]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	No	No	Month-to-month	Yes
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	No	No	One year	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	No	No	Month-to-month	Yes
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	No	No	One year	No
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	No	No	Month-to-month	Yes

5 rows × 21 columns

Detalles de la base de datos

- `customerID` - Customer unique identifier
- `gender` - Customer gender - ['Female' 'Male']
- `SeniorCitizen` - Elderly or retired person, a senior citizen is someone who has at least attained the age of 60 of 65 years
- `Partner` - - ['No' 'Yes']
- `Dependents` - If customer has dependents - ['No' 'Yes']
- `Tenure` - Customer lifespan (in months)
- `PhoneService` - - ['No' 'Yes']
- `MultipleLines` - - ['No' 'No phone service' 'Yes']
- `InternetService` - - ['No' 'No internet service' 'Yes']
- `OnlineSecurity` - - ['No' 'No internet service' 'Yes']
- `OnlineBackup` - - ['No' 'No internet service' 'Yes']
- `DeviceProtection` - - ['No' 'No internet service' 'Yes']
- `TechSupport` - - ['No' 'No internet service' 'Yes']
- `StreamingTV` - - ['No' 'No internet service' 'Yes']
- `StreamingMovies` - - ['No' 'No internet service' 'Yes']
- `Contract` - Type of contract - ['Month-to-month' 'One year' 'Two year']
- `PaperlessBilling` - - ['No' 'Yes']
- `PaymentMethod` - payment method - ['Bank transfer (automatic)', 'Credit card (automatic)', 'Electronic check', 'Mailed check']
- `MonthlyCharges` - Monthly Recurring Charges
- `TotalCharges` - Life time value
- `Churn` - Churn value, the targer vector - ['No' 'Yes']

Limpieza del Dataset

In [ ]:
```
def get_df_size(df, header='Dataset dimensions'):
  print(header,
        '\n# Attributes: ', df.shape[1],
        '\n# Entries: ', df.shape[0],'\n')

get_df_size(df)

#df.info()

# reemplaza valores en blanco por NaN
df_clean = df.replace(r'^\s*$', np.nan, regex=True)

# reemplaza valores faltantes en TotalCharges por la mediana de TotalCharges.
total_charges_median = df_clean.TotalCharges.median()
df_clean['TotalCharges'].fillna(total_charges_median, inplace=True)
df_clean['TotalCharges'] = df_clean['TotalCharges'].apply(pd.to_numeric)

#CustomerID lo retiramos porque no es una caracteristica
df_clean = df_clean.drop('customerID', axis=1)
df_clean.describe()

print("Churn No Instances: ", df_clean[df_clean['Churn'] == 'No'].shape[0])
print("Churn Yes Instances: ", df_clean[df_clean['Churn'] == 'Yes'].shape[0])
```

```
Dataset dimensions
# Attributes:  21
# Entries:  7043

Churn No Instances:  5174
Churn Yes Instances:  1869
```

Preparación de la Base de Datos

In [ ]:
```
binary_feat = df_clean.nunique()[df_clean.nunique() == 2].keys().tolist()
numeric_feat = [col for col in df_clean.select_dtypes(['float','int']).columns.tolist() if col not in binary_feat]
categorical_feat = [ col for col in df_clean.select_dtypes('object').columns.to_list() if col not in binary_feat + numeric_feat ]
df_proc = df_clean.copy()
#Etiquetas para caracteristicas binarias
le = LabelEncoder()
for i in binary_feat:
  df_proc[i] = le.fit_transform(df_proc[i])
  print(i, '\n', np.unique(df_proc[i].values))
#Dummy variables
df_proc = pd.get_dummies(df_proc, columns=categorical_feat)
get_df_size(df, header='Original dataset:')
get_df_size(df_proc, header='Processed dataset:')
df_proc.head()
```

```
gender
 [0 1]
SeniorCitizen
 [0 1]
Partner
 [0 1]
Dependents
 [0 1]
PhoneService
 [0 1]
PaperlessBilling
 [0 1]
Churn
 [0 1]
Original dataset:
# Attributes:  21
# Entries:  7043

Processed dataset:
# Attributes:  41
# Entries:  7043
```

Out[ ]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn	...	StreamingMovies_No	StreamingMovies_No internet service	StreamingMovies_Yes	Contract_Month-to-month	C
0	0	0	1	0	1	0	1	29.85	29.85	0	...	True	False	False	True	
1	1	0	0	0	34	1	0	56.95	1889.50	0	...	True	False	False	False	
2	1	0	0	0	2	1	1	53.85	108.15	1	...	True	False	False	True	
3	1	0	0	0	45	0	0	42.30	1840.75	0	...	True	False	False	False	
4	0	0	0	0	2	1	1	70.70	151.65	1	...	True	False	False	True	

5 rows × 41 columns

División en conjunto de entrenamiento y test

In [ ]:
```
# dividimos df_proc en caracteristicas y salida
X=df_proc.drop('Churn', axis=1)
y=df_proc['Churn']

# Dividimos el conjunto de entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
```

Modelo de Machine Learning

Balanceo de datos

In [ ]:
```
# submuestreo -> under sampling
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=1)
X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
get_df_size(X_train, header='Before balancing:')
get_df_size(X_train_rus, header='After balancing:')

# verificamos que las categorias estén balanceadas
np.unique(y_train_rus, return_counts=True)
```

```
Before balancing:
# Attributes:  40
# Entries:  5634

After balancing:
# Attributes:  40
# Entries:  2990
```

Out[ ]:
```
(array([0, 1]), array([1495, 1495]))
```

Normalización de datos

In [ ]:
```
# standardizing X_train and X_test
scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
X_train_rus = scaler.fit_transform(X_train_rus)
X_test = scaler.transform(X_test)
```

Entrenamiento con KNN con valores predeterminados

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics as metrics
knn = KNeighborsClassifier() #Representar, valor de k predeterminado es 5 vecinos, ponderac uniforme (sin ponderac)

knn.fit(X_train_rus, y_train_rus) #entrenamiento

#performance conjunto de test
pred_test = knn.predict(X_test)
acc_test = metrics.accuracy_score(y_test, pred_test)
recall_test = metrics.recall_score(y_test, pred_test)
print("Recall test = ", recall_test, ", acc test = ", acc_test)

#performance conjunto de entrenamiento
pred_train = knn.predict(X_train_rus)
acc_train = metrics.accuracy_score(y_train_rus, pred_train)
recall_train = metrics.recall_score(y_train_rus, pred_train)
print("Recall train = ", recall_train, ", acc train = ", acc_train)
```

```
Recall test =  0.7727272727272727 , acc test =  0.6827537260468417
Recall train =  0.8568561872909699 , acc train =  0.8113713274458194
```

Actividad: Optimización de Hiperparámetros KNN

Use 5-fold cross-validation (ver `GridSearchCV` y `StratifiedKFold`) para optimizar los siguientes hiperparámetros de KNN en el conjunto de entrenamiento balanceado (`X_train_rus` , `y_train_rus`):

1. Ponderación de vecinos: sin ponderación, ponderación del inverso de la distancia
2. Número de vecinos: rango 1 a 100.
3. Métricas de distancia: euclideana, manhattan, coseno

Para la optimización de hiperparámetros utilice como métrica al recall.

Una vez optimizados los hiperparámetros, reentrene su clasificador de kNN con los mejores hiperparámetros encontrados. Finalmente, evalúe este último clasificador en el conjunto de test ('X_test', 'y_test').

In [ ]:
```
# k-fold cross validation (GridSearchCV y StratifiedKFold)
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

#1. Ponderación de vecinos: sin ponderación, ponderación del inverso de la distancia
#2. Número de vecinos: rango 1 a 100.
#3. Métricas de distancia: euclideana, manhattan, coseno

#Definimos la grilla de parámetros
param_grid = {'n_neighbors': np.arange(1, 100),
              'weights': ['uniform', 'distance'],
              'metric': ['euclidean', 'manhattan', 'cosine']}
#Definimos el modelo
knn = KNeighborsClassifier()
#Definimos el método de validación cruzada
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
#Definimos la búsqueda por grilla
knn_grid = GridSearchCV(knn, param_grid, cv=cv, scoring='recall', n_jobs=-1)
#Ajustamos el modelo
knn_grid.fit(X_train_rus, y_train_rus)
#Mejores parámetros
print("Mejores parámetros: ", knn_grid.best_params_)
#Mejor score
print("Mejor score: ", knn_grid.best_score_)
#Mejor modelo
best_model = knn_grid.best_estimator_
#Predicciones
pred_test = best_model.predict(X_test)
#Performance
acc_test = metrics.accuracy_score(y_test, pred_test)
recall_test = metrics.recall_score(y_test, pred_test)
print("Recall test = ", recall_test, ", acc test = ", acc_test)

# performance conjunto de entrenamiento
pred_train = best_model.predict(X_train_rus)
acc_train = metrics.accuracy_score(y_train_rus, pred_train)
recall_train = metrics.recall_score(y_train_rus, pred_train)
print("Recall train = ", recall_train, ", acc train = ", acc_train)

# performance conjunto de test
pred_test = best_model.predict(X_test)
acc_test = metrics.accuracy_score(y_test, pred_test)
recall_test = metrics.recall_score(y_test, pred_test)
print("Recall test = ", recall_test, ", acc test = ", acc_test)
```

```
Mejores parámetros:  {'metric': 'euclidean', 'n_neighbors': 99, 'weights': 'distance'}
Mejor score:  0.8648411633100621
Recall test =  0.8395721925133069 , acc test =  0.6678495386799148
Recall train =  0.9973244147157191 , acc train =  0.9986622073578595
Recall test =  0.8395721925133069 , acc test =  0.6678495386799148
```

In [ ]: