**Desvalance, en campo churn**

df.Churn.value\_counts()

**Output**

No 5174

Yes 1869

Hay muchos mas “No” eso hace desbalance, 73% es No

**Build a model**

**import** tensorflow **as** tf

**from** tensorflow **import** keras

**from** sklearn.metrics **import** confusion\_matrix , classification\_report

**def** ANN(X\_train, y\_train, X\_test, y\_test, loss, weights):

model **=** keras**.**Sequential([

keras**.**layers**.**Dense(26, input\_dim**=**26, activation**=**'relu'),

keras**.**layers**.**Dense(15, activation**=**'relu'),

keras**.**layers**.**Dense(1, activation**=**'sigmoid')

])

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

**if** weights **==** **-**1:

model**.**fit(X\_train, y\_train, epochs**=**100)

**else**:

model**.**fit(X\_train, y\_train, epochs**=**100, class\_weight **=** weights)

print(model**.**evaluate(X\_test, y\_test))

y\_preds **=** model**.**predict(X\_test)

y\_preds **=** np**.**round(y\_preds)

**#from** sklearn.metrics **import** confusion\_matrix , classification\_report

print("Classification Report: \n", classification\_report(y\_test, y\_preds))

**return** y\_preds

y\_preds **=** ANN(X\_train, y\_train, X\_test, y\_test, 'binary\_crossentropy', **-**1)

**Mitigating Skewdness of Data**

df1.Churn.value\_counts()

Output (desbalanceado)

No 5174

Yes 1869

Hay muchos mas “No” eso hace desbalance, 73% es No

***# Class count***

count\_No, count\_Yes **=** df1**.**Churn**.**value\_counts()

#count\_No = 5174

#count\_Yes = 1869

***# Divide by class***

**df\_NO** **=** df2[df2['Churn'] **==** 0]

**df\_YES** **=** df2[df2['Churn'] **==** 1]

**Method 1: Undersampling**

**Reduce los datos de prueba, si hay 5174 NO y 1869 YES, los reduce a ambos a 1495**

reference: <https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>

***# Undersample 0-class and concat the DataFrames of both class***

**df\_Temp** **=** **df\_NO.**sample(count\_Yes)

**df\_Final** **=** pd**.**concat([**df\_Temp**, **df\_YES**], axis**=**0)

df\_Temp.shape #7032

df\_Final.shape #3738 (1869 yes, 1869 no)

***# =================================================================***

X **=** **df\_Final.**drop('Churn',axis**=**'columns')

y **=** **df\_Final**['Churn']

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

random\_state**=**15, stratify**=**y)

*# Number of classes in training Data*

y\_train**.**value\_counts()

Output

1 1495 NO

0 1495 YE

Balanceado

**Printing Classification in the last, Scroll down till the last epoch to watch the classification report**

y\_preds **=** ANN(X\_train, y\_train, X\_test, y\_test, 'binary\_crossentropy', **-**1)

Check classification report above. f1-score for minority class 1 improved from **0.57 to 0.76**. Score for class 0 reduced to 0.75 from 0.85 but that's ok. We have more generalized classifier which classifies both classes with similar prediction score

**Method2: Oversampling**

**Aumenta los datos de prueba, le reduce un poco al que tiene mas y le aumenta al que tiene menos, mejora exactitud**

***# Oversample 1-class and concat the DataFrames of both classes***

**df\_Temp** **=** **df\_YES.**sample(count\_No, replace=True)

**df\_Final** **=** pd**.**concat([df\_NO, **df\_Temp**], axis**=**0)

print('Random over-sampling:')

print(df\_test\_over.Churn.value\_counts())

Output

Random over-sampling:

0 5163

1 5163

***# =================================================================***

X **=** **df\_Final.**drop('Churn',axis**=**'columns')

y **=** **df\_Final**['Churn']

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

random\_state**=**15, stratify**=**y)

*# Number of classes in training Data*

y\_train**.**value\_counts()

Output

1 4130 NO

0 4130 YES

Balanceado

Check classification report above. f1-score for minority class 1 improved from **0.57 to 0.76**. Score for class 0 reduced to 0.75 from 0.85 but that's ok. We have more generalized classifier which classifies both classes with similar prediction score

**Method3: SMOTE**

To install imbalanced-learn library use **pip install imbalanced-learn** command

X **=** df2**.**drop('Churn',axis**=**'columns')

y **=** df2['Churn']

**from** imblearn.over\_sampling **import** SMOTE

**smote = SMOTE(sampling\_strategy='minority')**

**X\_sm, y\_sm = smote.fit\_sample(X, y)**

y\_sm**.**value\_counts()

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(**X\_sm, y\_sm**, test\_size**=**0.2,

random\_state**=**15, stratify**=**y\_sm)

In [63]:

*# Number of classes in training Data*

y\_train**.**value\_counts()

y\_preds **=** ANN(X\_train, y\_train, X\_test, y\_test, 'binary\_crossentropy', **-**1)

SMOTE Oversampling increases f1 score of minority class 1 from **0.57 to 0.81 (huge improvement)**

Also over all accuracy improves from 0.78 to 0.80

**Method4: Use of Ensemble with undersampling**

df2**.**Churn**.**value\_counts()

X **=** df2**.**drop('Churn',axis**=**'columns')

y **=** df2['Churn']

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, random\_state**=**15, stratify**=**y)

In [68]:

y\_train**.**value\_counts()

Out[68]:

model1 --> class1(1495) + class0(0, 1495)

model2 --> class1(1495) + class0(1496, 2990)

model3 --> class1(1495) + class0(2990, 4130)

In [73]:

df3 **=** X\_train**.**copy()

df3['Churn'] **=** y\_train

In [74]:

**df\_0 =** df3[df3**.Churn==0**]

**df\_1** **=** df3[df3**.Churn==1**]

**def** **train\_batch**(df\_majority, df\_minority, start, end, loss, weights):

df\_train **=** pd**.**concat([df\_majority[start:end], df\_minority], axis**=**0)

X\_train **=** df\_train**.**drop('Churn', axis**=**'columns')

y\_train **=** df\_train**.**Churn

**#model**

model **=** keras**.**Sequential([

keras**.**layers**.**Dense(26, input\_dim**=**26, activation**=**'relu'),

keras**.**layers**.**Dense(15, activation**=**'relu'),

keras**.**layers**.**Dense(1, activation**=**'sigmoid')

])

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

**if** weights **==** **-**1:

model**.**fit(X\_train, y\_train, epochs**=**100)

**else**:

model**.**fit(X\_train, y\_train, epochs**=**100, class\_weight **=** weights)

print(model**.**evaluate(X\_test, y\_test))

y\_preds **=** model**.**predict(X\_test)

y\_preds **=** np**.**round(y\_preds)

**#from** sklearn.metrics **import** confusion\_matrix , classification\_report

print("Classification Report: \n", classification\_report(y\_test, y\_preds))

**return** y\_preds

y\_pred1 **=** **train\_batch**(**df\_0**, **df\_1**, **0, 1495**, 'binary\_crossentropy', **-**1)

y\_pred2 **=** **train\_batch**(**df\_0**, **df\_1**, **1495, 2990**, 'binary\_crossentropy', **-**1)

y\_pred3 **=** **train\_batch**(**df\_0**, **df\_1**, **2290, 4130**, 'binary\_crossentropy', **-**1)

len(y\_pred1)

y\_pred\_final **=** y\_pred1**.**copy()

**for** i **in** range(len(y\_pred1)):

n\_ones **=** y\_pred1[i] **+** y\_pred2[i] **+** y\_pred3[i]

**if** n\_ones**>**1:

y\_pred\_final[i] **=** 1

**else**:

y\_pred\_final[i] **=** 0

cl\_rep **=** classification\_report(y\_test, y\_pred\_final)

print(cl\_rep)

f1-score for minority class 1 improved to 0.62 from 0.57.

The score for majority class 0 is suffering and reduced to 0.80 from 0.85 but at least there is some balance in terms of prediction accuracy across two classes