from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

import pandas as pd

In [3]:

linkcode

df\_full = pd.read\_csv('../input/consumercomplaintsdata/Consumer\_Complaints.csv')

df\_full.head()

*# How many unique financial products (the second column) are we talking about here*

df\_full['Product'].nunique()

*# The shape of the full, unmodified data*

print('Shape of data',df\_full.shape)

*# The idea is to demonstrate a workflow, so we will work with a smaller portion of the data*

*# First, we retain only the columns relevant to our present purpose*

df=df\_full[['Consumer complaint narrative','Product']]

print('Shape of data',df.shape)

Shape of data (903983, 2)

*# Next, we get rid of nulls*

print('Before dropping the nulls')

display('Null count', df.isna().sum())

print('Total rows of data', len(df))

df.dropna(inplace=True)

print('='\*80)

print('After dropping the nulls')

display('Null count', df.isna().sum())

print('Total rows of data', len(df))

df=df.head(1000).reset\_index(drop=True)

display(df.head())

display(df.tail())

print('Shape of data',df.shape)

Shape of data (1000, 2)

df.tail()

*# Kinds of products on which complaints are generated*

df['Product'].nunique()

df['Consumer complaint narrative'][0]

#Categories of products - the classes for which we will predict

list(df.Product.unique())

df['Product'].value\_counts()

#Train-test split

25% of the total data is used as validation data while the remaining as training.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

df['Consumer complaint narrative'], df['Product'],

test\_size=0.25, random\_state=0, stratify=df['Product'])

print(f'Training utterances: **{**len(X\_train)**}** of shape **{**X\_train.shape**}**')

print(f'Validation utterances: **{**len(X\_test)**}** of shape **{**X\_test.shape**}**')

*# NOTE: The features occupy a single column*

Training utterances: 750 of shape (750,)

Validation utterances: 250 of shape (250,)

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

display(y\_test.value\_counts())

#Calculating tf-idf scores

#Calculating tf-idf scores for each unique token in the dataset and creating frequency chart for #each utterance in the dataset.

In [18]:

#linkcode

*# instantiate the vectorizer object*

vectorizer = TfidfVectorizer(stop\_words= 'english')

*# convert the documents into a matrix*

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

X\_train\_vec, X\_test\_vec

# **#Feature Selection**

**#SelectKBest** Select features according to the k highest scores.

**#Chi-square test** measures dependence between stochastic variables, so using this function #“weeds out” the features that are the most likely to be independent of class and therefore #irrelevant for classification.

from sklearn.feature\_selection import SelectKBest, chi2

n\_features=100

ch2 = SelectKBest(chi2, k=n\_features)

X\_train\_sp = ch2.fit\_transform(X\_train\_vec, y\_train)

X\_test\_sp = ch2.transform(X\_test\_vec)

X\_train\_sp, X\_test\_sp

*# Converting the sparse matrix to a dense one to visualize it.*

cols = list(range(n\_features))

X\_train\_dense = pd.DataFrame(data=X\_train\_sp.toarray(), columns=cols)

X\_test\_dense = pd.DataFrame(data=X\_test\_sp.toarray(), columns=cols)

print(X\_train\_dense.shape, X\_test\_dense.shape)

X\_train\_dense

*# Now we have train and test data as vectors*

*# Let us also convert the target data appropriately*

encoder = LabelEncoder()

y\_train\_num = encoder.fit\_transform(y\_train)

y\_test\_num = encoder.transform(y\_test)

y\_train\_num.min(), y\_train\_num.max(), y\_test\_num.min(), y\_test\_num.max() *# sanity check*

*# What does the target look like, after encoding. Check out the first n datapoints*

n=5

print('Text Encoding')

print('-'\*50)

for p,q **in** zip(y\_train[:n].values,y\_train\_num):

print(f'**{**q**}** **{**p**}**')

*# Now, if you are fussy and want to see exactly what kind of encoding has happened.*

mapping = {l: i for i, l **in** enumerate(encoder.classes\_)}

mapping

# **#Our data is ready for modelling**

#We want to train a model such that looking at the complaint text, it should be able to determine #which category of complaint it deals with.

#linkcode

rf\_model = RandomForestClassifier(n\_estimators=200, random\_state=42, n\_jobs = -1)

scores = cross\_val\_score(rf\_model,

X\_train\_dense,

y\_train\_num,

cv=5,

n\_jobs = -1,

scoring = 'accuracy')

scores.mean()

rf\_model.fit(X\_train\_dense, y\_train\_num)

preds=rf\_model.predict(X\_test\_dense)

print('Predictions ready')

#Predictions ready

#linkcode

*# What does a prediction look like - let's take the first one*

preds[0]

*# Let's revert back to the categories we understand*

preds=encoder.inverse\_transform(preds)

preds[0]

Out[27]:

# **#brief look at the predictions made**

In [28]:

linkcode

report = pd.DataFrame(columns=['Complaint','Actual Product','Prediction'])

report['Complaint'] = X\_test

report['Actual Product'] = y\_test

report['Prediction'] = preds

report

*## How accurate is this model?*

report['Correct'] = (report['Actual Product'] == report['Prediction']).astype('int')

display(report)

print(f'Accuracy: **{**100\*report.Correct.sum()/report.Correct.count()**}** %')

*# Another way to crunch numbers*

r = pd.DataFrame()

r['Correctly Predicted'] = report.groupby('Actual Product').sum()['Correct']

r['Overall Predicted'] = report.groupby('Prediction').count()['Correct']

r['Actuals'] = report.groupby('Actual Product').count()['Correct']

r

## **# brief look at a confusion matrix**

In [32]:

linkcode

from sklearn.metrics import confusion\_matrix

def plot\_confusion\_matrix(cm,labels,size=10, rotate\_labels=False):

*'''*

*This function receives a confusion matrix object and plots it out using seaborn*

*'''*

import seaborn as sns

import matplotlib.pyplot as plt

font\_specs = {"size": 20, 'fontweight':'bold'}

title\_specs= {"size": 16, 'fontweight':'bold'}

figsize = size

fig, ax = plt.subplots(figsize = (figsize,figsize), facecolor = '#ebebeb', frameon = True, edgecolor = 'black')

ax = sns.heatmap(cm,annot=True, cbar = False, cmap = 'Blues',linewidths=5,

linecolor='#ebebeb', annot\_kws=font\_specs, fmt='g')

plt.xlabel('Predicted', fontdict = font\_specs, labelpad=-(figsize\*65))

plt.ylabel('Actual', fontdict = font\_specs, labelpad=15)

ax.xaxis.set\_ticklabels(labels)

ax.yaxis.set\_ticklabels(labels)

if rotate\_labels:

ax.set\_xticklabels(labels, rotation=90, ha='center')

ax.set\_yticklabels(labels, rotation=0, ha='right')

ax.tick\_params(labelbottom=False, labeltop=True, labelsize = 12, colors ='#151736' )

plt.title('CONFUSION MATRIX',loc = 'right', pad = figsize\*4 , fontdict = title\_specs)

plt.show()

print('custom function defined')

custom function defined

cm = confusion\_matrix(y\_test, preds, labels=encoder.classes\_)

plot\_confusion\_matrix(cm=cm,labels=encoder.classes\_, size=12, rotate\_labels=True)