**Feature Engineering ICE 1**

**Github link -** [**https://github.com/SoumyaBhandari/Feature-Engineering-fall2021**](https://github.com/SoumyaBhandari/Feature-Engineering-fall2021)

**Team Members**

* Soumya Bhandari
* Divya Geethanjali
* Amaanullah Shareef
* Syed Araib Karim
* Shabab AL Gamdi

## Code

### Creating Dataset

import urllib.request,sys,time

from bs4 import BeautifulSoup

import requests

import pandas as pd

import bs4

import requests

### Creating the function to scrap the web page.

def scrapWebPage(url, articleType): response = requests.get(url)

if response is not None: html = bs4.BeautifulSoup(response.text, 'html.parser')

#title = html.select("#firstHeading")[0].text

`paragraphs = html.select("p")`

`articleText = ""`

`for para in paragraphs:`

`articleText += para.text.strip('\n')`

`#print (para.text)`

# just grab the text up to contents as stated in question`

`#intro = '\n'.join([ para.text for para in paragraphs[0:5]])`

`#print (intro)`

`data = {'category': articleType, 'content': articleText}`

`return data`

### Webpages used for data collection

alllinks = []

alllinks.append(scrapWebPage("https://www.forbes.com/sites/bernardmarr/2020/07/17/5-ways-self-driving-cars-could-make-our-world-and-our-lives-better/?sh=17181db742a3","positive"))

alllinks.append(scrapWebPage("https://www.startupgrind.com/blog/5-reasons-you-should-embrace-self-driving-cars/","positive"))

alllinks.append(scrapWebPage("https://www.vox.com/future-perfect/2020/2/14/21063487/self-driving-cars-autonomous-vehicles-waymo-cruise-uber","negative"))

alllinks.append(scrapWebPage("https://www.iotforall.com/dangers-of-autonomous-vehicles-worldwide-adoption","negative"))

alllinks.append(scrapWebPage("https://www.theatlantic.com/technology/archive/2018/12/7-arguments-against-the-autonomous-vehicle-utopia/578638/","negative"))

alllinks.append(scrapWebPage("https://www.mosaic51.com/featured/11-benefits-of-self-driving-cars-how-will-your-life-improve/","positive"))

### Assigning category to new dataset

import csv def savingToDataframe(alllinks): rows = [] fields = ["File\_Name","Content","Category","Complete\_Filename"] for i in range(0, len(alllinks)): Content = alllinks[i] filename = "FileName\_"+ str(i) category = "tech" Complete\_Filename = filename rows.append([filename, Content,category, Complete\_Filename]) df = pd.DataFrame(rows, columns=fields) print(df) return df

df = savingToDataframe(alllinks)

df.head()

df.to\_csv('dataset\_file.csv', index=False)

### Assigning the column name to the dataset

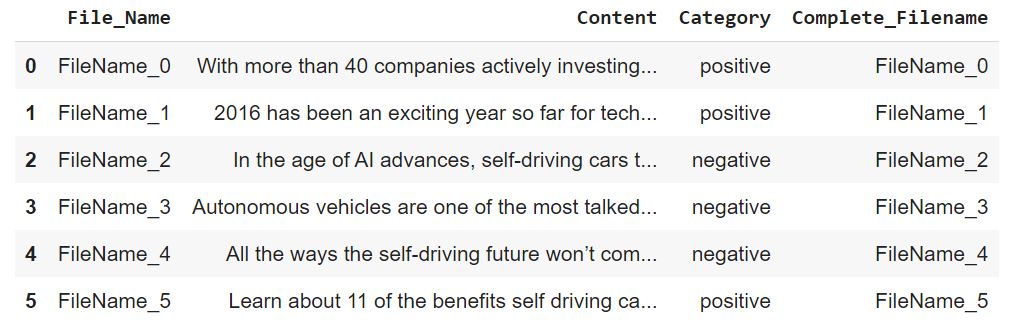
import pandas as pd

fields = ["File\_Name","Content","Category","Complete\_Filename"]

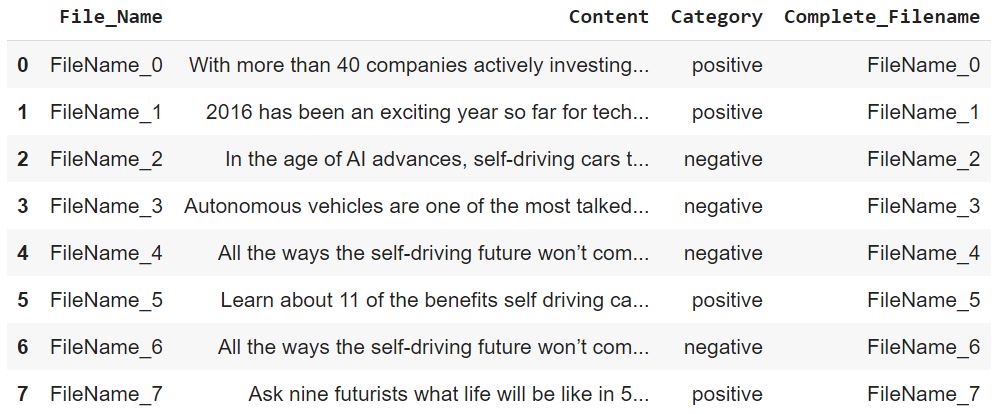
df = pd.read\_csv('dataset\_file.csv', names=fields,header=0)

print(df)

df.head(6)



df.tail(10)



pip install vega==1.3

## EDA Analysis

import pandas as pd

import matplotlib.pyplot as plt

import pickle

import seaborn as sns

sns.set\_style("whitegrid")

import altair as alt

import warnings

warnings.filterwarnings("ignore")

### Number of articles in each category

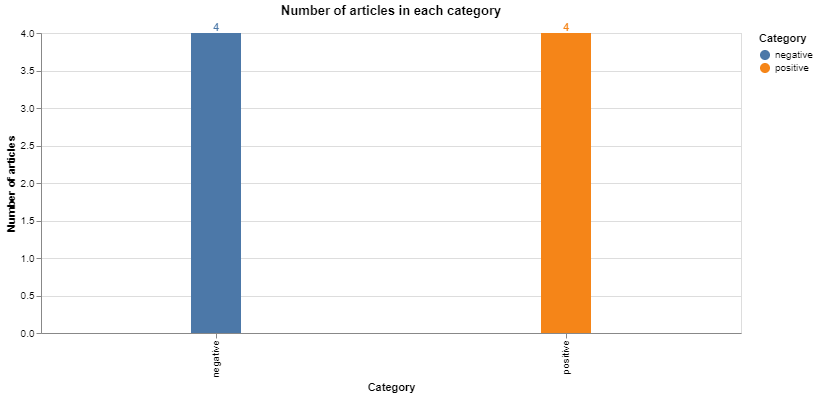
#### Bar chart

bars = alt.Chart(df).mark\_bar(size=50).encode( x=alt.X("Category"), y=alt.Y("count():Q", axis=alt.Axis(title='Number of articles')), tooltip=[alt.Tooltip('count()', title='Number of articles'), 'Category'], color='Category'

)

text = bars.mark\_text( align='center', baseline='bottom', ).encode( text='count()' )

(bars + text).interactive().properties( height=300, width=700, title = "Number of articles in each category", )



#### Percentage of articles in each category

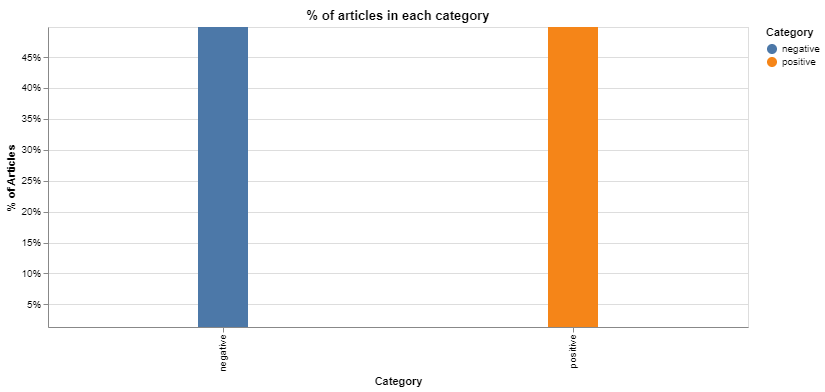
df['id'] = 1

df2 = pd.DataFrame(df.groupby('Category').count()['id']).reset\_index()

bars = alt.Chart(df2).mark\_bar(size=50).encode( x=alt.X('Category'), y=alt.Y('PercentOfTotal:Q', axis=alt.Axis(format='.0%', title='% of Articles')), color='Category' ).transform\_window( TotalArticles='sum(id)', frame=[None, None] ).transform\_calculate( PercentOfTotal="datum.id / datum.TotalArticles" )

text = bars.mark\_text( align='center', baseline='bottom', #dx=5 # Nudges text to right so it doesn't appear on top of the bar ).encode( text=alt.Text('PercentOfTotal:Q', format='.1%') )

(bars + text).interactive().properties( height=300, width=700, title = "% of articles in each category",

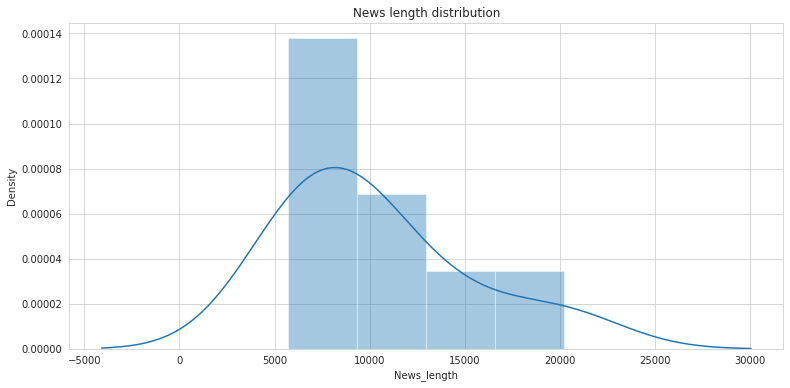


#### News length by category

#### Displot showing the new length of the article

plt.figure(figsize=(12.8,6))

sns.distplot(df['News\_length']).set\_title('News length distribution');



#### Calculating the statistics of the articles

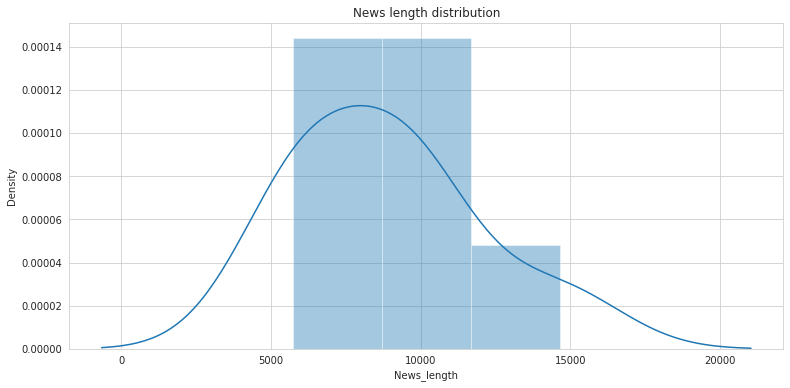
df['News\_length'].describe()

quantile\_95 = df['News\_length'].quantile(0.95)

df\_95 = df[df['News\_length'] < quantile\_95]

plt.figure(figsize=(12.8,6))

sns.distplot(df\_95['News\_length']).set\_title('News length distribution');



plt.figure(figsize=(12.8,6))

df\_more10k = df[df['News\_length'] > 10000]

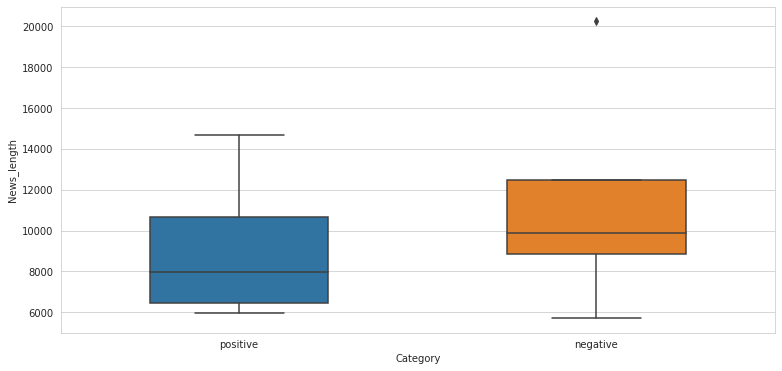
len(df\_more10k)

Output : 2

df\_more10k['Content'].iloc[0]

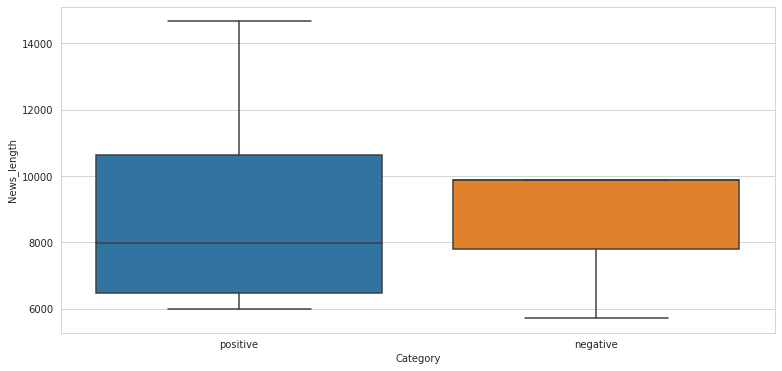
Output : In the age of AI advances, self-driving cars turned out to be harder than people expected. Finding the best ways to do good. \n When it comes to self-driving cars, the future was supposed to be now. In 2020, you’ll be a “permanent backseat driver,” the Guardian predicted in 2015. “10 million self-driving cars will be on the road by 2020,” blared a Business Insider headline from 2016. Those declarations were accompanied by announcements from General Motors, Google’s Waymo, Toyota, and Honda that they’d be making self-driving cars by 2020. Elon Musk forecast that Tesla would do it by 2018 — and then, when that failed, by 2020. But the year is here — and the self-driving cars aren’t. Despite extraordinary efforts from many of the leading names in tech and in automaking, fully autonomous cars are still out of reach except in special trial programs. You can buy a car that will automatically brake for you when it anticipates a collision, or one that helps keep you in your lane, or even a....

sns.boxplot(data=df, x='Category', y='News\_length', width=.5);



plt.figure(figsize=(12.8,6))

sns.boxplot(data=df\_95, x='Category', y='News\_length');



with open('News\_dataset.pickle', 'wb') as output: pickle.dump(df, output)

with open('News\_dataset.pickle', 'rb') as data: df = pickle.load(data)

df.head()



### Feature Engineering

import pickle

import pandas as pd

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.feature\_selection import chi2

import numpy as np

with open('News\_dataset.pickle', 'rb') as data: df = pickle.load(data)

df.head()



### Visualizing news sample content

df.loc[1]['Content']

**Output**

2016 has been an exciting year so far for technologies that, until recently, have been confined to the world of science fiction dreams. Whether it’s 3D printing revolutionizing the Healthcare industry or the widespread commercial availability of VR technology, we’re continuing to witness amazing\xa0innovations that even just a few years ago seemed like distant possibilities at best.Another technological advancement causing widespread wonder is self-driving cars. We may not have arrived at a Jetsons-like future of flying automated vehicles just yet, but the self-driving car is getting closer and closer to becoming a widespread, viable method of transportation. And, while many are skeptical about the future of driverless and self-driving cars on the road, there’s actually a good deal of data and research to support the idea that self-driving cars may actually be better for our future.With that in mind, here are five reasons to embrace self-driving cars:It’s tough at first to wrap our min

#### Text and Character Cleaning

# \r and \n

df['Content\_Parsed\_1'] = df['Content'].str.replace("\r", " ")

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace("\n", " ")

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace(" ", " ")

#" when quoting the text"

df['Content\_Parsed\_1'] = df['Content\_Parsed\_1'].str.replace('"', '')

# Lowercasing the text

df['Content\_Parsed\_2'] = df['Content\_Parsed\_1'].str.lower()

#### getting rid of punctuation

punctuation\_signs = list("?:!.,;")

df['Content\_Parsed\_3'] = df['Content\_Parsed\_2']

for punct\_sign in punctuation\_signs: df['Content\_Parsed\_3'] = df['Content\_Parsed\_3'].str.replace(punct\_sign, '')

df['Content\_Parsed\_4'] = df['Content\_Parsed\_3'].str.replace("'s", "")

#### Stemming and Lemmatization

# Downloading punkt and wordnet from NLTK

nltk.download('punkt')

print("------------------------------------------------------------")

nltk.download('wordnet')

#### Saving the lemmatizer into an object

wordnet\_lemmatizer = WordNetLemmatizer()

for row in range(0, nrows):

`lemmatized\_list = []`

`text = df.loc[row]['Content\_Parsed\_4']`

`text\_words = text.split(" ")`

`for word in text\_words:`

`lemmatized\_list.append(wordnet\_lemmatizer.lemmatize(word, pos="v"))`

`lemmatized\_text = " ".join(lemmatized\_list)`

`lemmatized\_text\_list.append(lemmatized\_text)`

df['Content\_Parsed\_5'] = lemmatized\_text\_list

#### Downloading the stop words list

nltk.download('stopwords')

#### Loading the stop words in english

stop\_words = list(stopwords.words('english'))

stop\_words[0:10]

Output ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]

df['Content\_Parsed\_6'] = df['Content\_Parsed\_5']

for stop\_word in stop\_words:

`regex\_stopword = r"\b" + stop\_word + r"\b"`

`df['Content\_Parsed\_6'] = df['Content\_Parsed\_6'].str.replace(regex\_stopword, '')`

#We have some dobule/triple spaces between words because of the replacements. However, it's not a problem because we'll tokenize by the spaces later.

#As an example, we'll show an original news article and its modifications throughout the process:

df.loc[2]['Content']

df.loc[2]['Content\_Parsed\_1']

df.loc[2]['Content\_Parsed\_2']

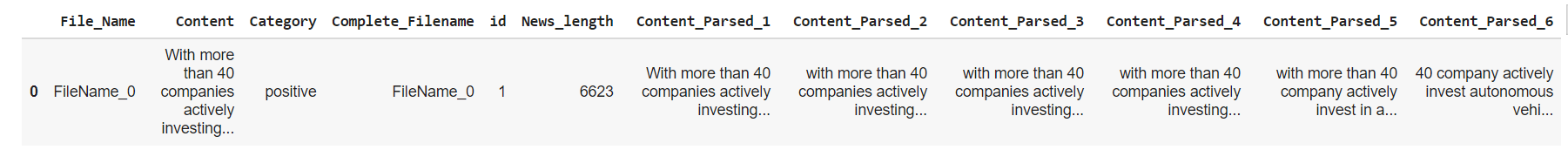
df.loc[2]['Content\_Parsed\_3']

df.loc[2]['Content\_Parsed\_4']

df.loc[2]['Content\_Parsed\_5']

df.loc[2]['Content\_Parsed\_6']

df.head(1)



# Label coding

#We'll create a dictionary with the label codification:

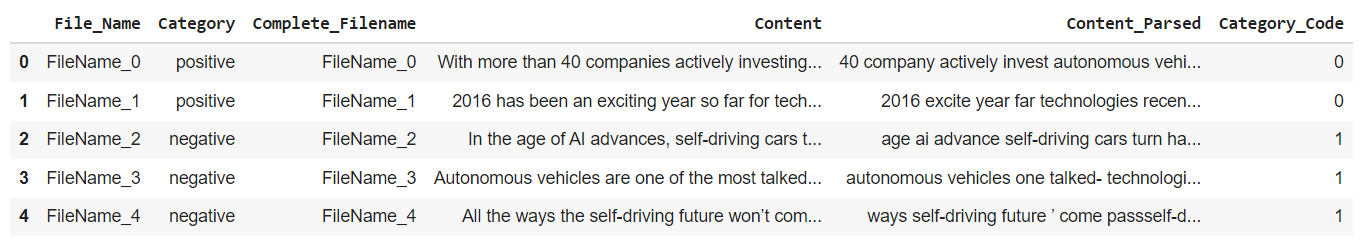
category\_codes = { 'business': 0, 'entertainment': 1, 'politics': 2, 'sport': 3, 'tech': 4 }

# Category mapping

df['Category\_Code'] = df['Category']

df = df.replace({'Category\_Code':category\_codes})

df.head()



#Train - test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['Content\_Parsed'], df['Category\_Code'], test\_size=0.15, random\_state=2)

X\_train.shape X\_test.shape

y\_test.shape

# Parameter election

ngram\_range = (1,2)

max\_df = 10

min\_df = 1.

max\_features = 300

tfidf = TfidfVectorizer(encoding='utf-8', ngram\_range=ngram\_range, stop\_words=None, lowercase=False, max\_df=max\_df, min\_df=min\_df, max\_features=max\_features, norm='l2', sublinear\_tf=True)

features\_train = tfidf.fit\_transform(X\_train).toarray()

labels\_train = y\_train

print(features\_train.shape)

features\_test = tfidf.transform(X\_test).toarray()

labels\_test = y\_test

print(features\_test.shape)

from sklearn.feature\_selection import chi2

import numpy as np

for Product, category\_id in sorted(category\_codes.items()): features\_chi2 = chi2(features\_train, labels\_train == category\_id) indices = np.argsort(features\_chi2[0]) feature\_names = np.array(tfidf.get\_feature\_names())[indices] unigrams = [v for v in feature\_names if len(v.split(' ')) == 1] bigrams = [v for v in feature\_names if len(v.split(' ')) == 2] print("# '{}' category:".format(Product)) print(" . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-5:]))) print(" . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-2:]))) print("")

bigrams

Output : ['driving cars', 'self driving', 'autonomous vehicles', 'driving car']

# X\_train

with open('X\_train.pickle', 'wb') as output: pickle.dump(X\_train, output)

# X\_test

with open('X\_test.pickle', 'wb') as output: pickle.dump(X\_test, output)

# y\_train

with open('y\_train.pickle', 'wb') as output: pickle.dump(y\_train, output)

# y\_test

with open('y\_test.pickle', 'wb') as output: pickle.dump(y\_test, output)

# df

with open('df.pickle', 'wb') as output: pickle.dump(df, output)

# features\_train

with open('features\_train.pickle', 'wb') as output: pickle.dump(features\_train, output)

# labels\_train

with open('labels\_train.pickle', 'wb') as output: pickle.dump(labels\_train, output)

# features\_test

with open('features\_test.pickle', 'wb') as output: pickle.dump(features\_test, output)

# labels\_test

with open('labels\_test.pickle', 'wb') as output: pickle.dump(labels\_test, output)

# TF-IDF object

with open('tfidf.pickle', 'wb') as output: pickle.dump(tfidf, output)

# Model Fitting

## Random Forest

#### First create the base model to tune

rfc = RandomForestClassifier(random\_state=8)

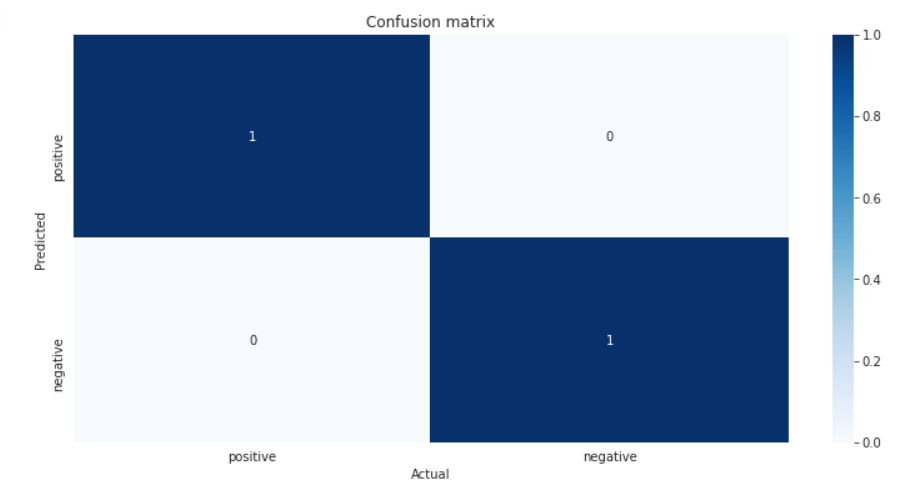
#### Definition of the random search

random\_search = RandomizedSearchCV(estimator=rfc, param\_distributions=random\_grid, n\_iter=50, scoring='accuracy', cv=3, verbose=1, random\_state=8)

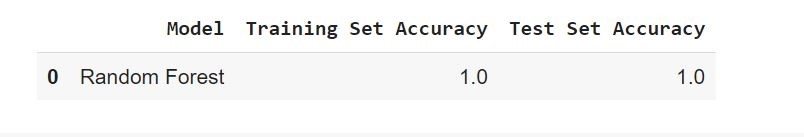
# Fit the random search model random\_search.fit(features\_train, labels\_train)

#### Confusion Matrix

aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category confusion\_matrix(labels\_test, rfc\_pred) plt.figure(figsize=(12.8,6)) sns.heatmap(conf\_matrix, annot=True, xticklabels=aux\_df['Category'].values, yticklabels=aux\_df['Category'].values, cmap="Blues") plt.ylabel('Predicted') plt.xlabel('Actual') plt.title('Confusion matrix') plt.show()



base\_model = RandomForestClassifier(random\_state = 8) base\_model.fit(features\_train, labels\_train) accuracy\_score(labels\_test, base\_model.predict(features\_test)) best\_rfc.fit(features\_train, labels\_train) accuracy\_score(labels\_test, best\_rfc.predict(features\_test))



## KNN

n\_neighbors = [1,2,3] param\_grid = {'n\_neighbors': n\_neighbors}

knnc = KNeighborsClassifier() cv\_sets = ShuffleSplit(n\_splits = 3, test\_size = .33, random\_state = 8)

grid\_search = GridSearchCV(estimator=knnc, param\_grid=param\_grid, scoring='accuracy', cv=cv\_sets, verbose=1)

grid\_search.fit(features\_train, labels\_train) print("The best hyperparameters from Grid Search are:") print(grid\_search.best\_params\_) print("") print("The mean accuracy of a model with these hyperparameters is:") print(grid\_search.best\_score\_)

#### Training accuracy

print("The training accuracy is: ") print(accuracy\_score(labels\_train, best\_knnc.predict(features\_train))) Output - The test accuracy is 1.00

#### Test accuracy

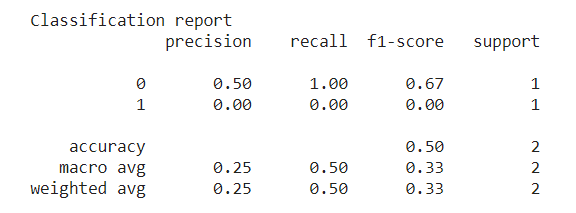
\_Code') conf\_matrix =

print("The test accuracy is: ") print(accuracy\_score(labels\_test, knnc\_pred))

Output - The test accuracy is 1.00

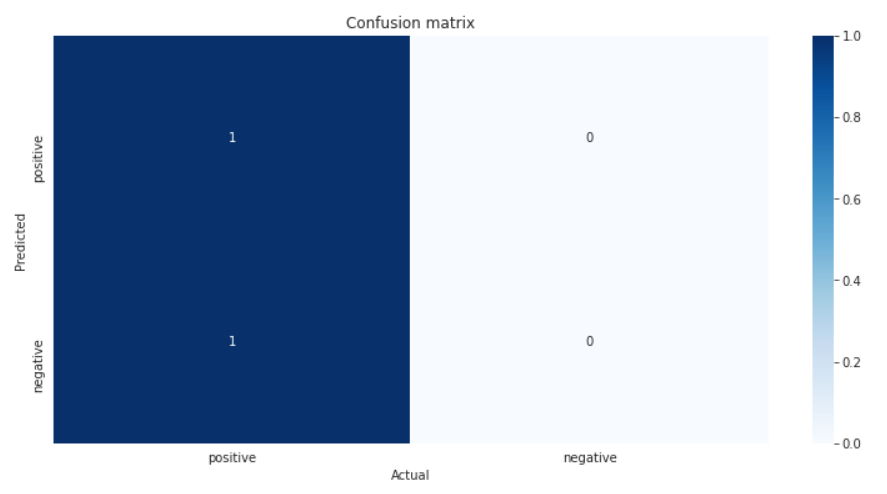
#### Classification report

print("Classification report") print(classification\_report(labels\_test,knnc\_pred))



#### Confusion Matrix

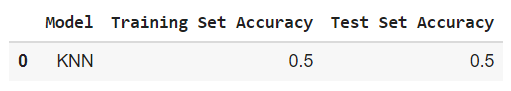
aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category\_Code') conf\_matrix = confusion\_matrix(labels\_test, knnc\_pred) plt.figure(figsize=(12.8,6)) sns.heatmap(conf\_matrix, annot=True, xticklabels=aux\_df['Category'].values, yticklabels=aux\_df['Category'].values, cmap="Blues") plt.ylabel('Predicted') plt.xlabel('Actual') plt.title('Confusion matrix') plt.show()



d = { 'Model': 'KNN', 'Training Set Accuracy': accuracy\_score(labels\_train, best\_knnc.predict(features\_train)), 'Test Set Accuracy': accuracy\_score(labels\_test, knnc\_pred) }

df\_models\_knnc = pd.DataFrame(d, index=[0])

df\_models\_knnc



## Support Vector Machine

#Cross Validation Hyper Parameter tuning

svc\_0 =svm.SVC(random\_state=8)

print('Parameters currently in use:\n') pprint(svc\_0.get\_params())

### Best hyperparameters resulting from the Random Search

print("The best hyperparameters from Random Search are:") print(random\_search.best\_params\_) print("") print("The mean accuracy of a model with these hyperparameters is:") print(random\_search.best\_score\_)

Output : The best hyperparameters from Random Search are: {'probability': True, 'kernel': 'rbf', 'gamma': 100, 'degree': 3, 'C': 0.001}

The mean accuracy of a model with these hyperparameters is: 0.6666666666666666

#### Training accuracy

print("The training accuracy is: ") print(accuracy\_score(labels\_train, best\_svc.predict(features\_train)))

#### Output

The training accuracy is: 1.0

#### Test accuracy

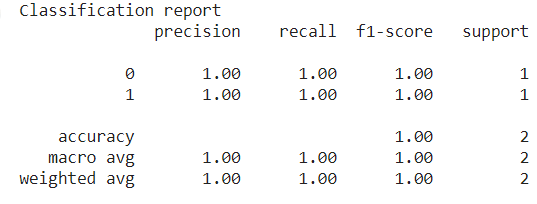
print("The test accuracy is: ") print(accuracy\_score(labels\_test, svc\_pred))

#### Output

The test accuracy is:

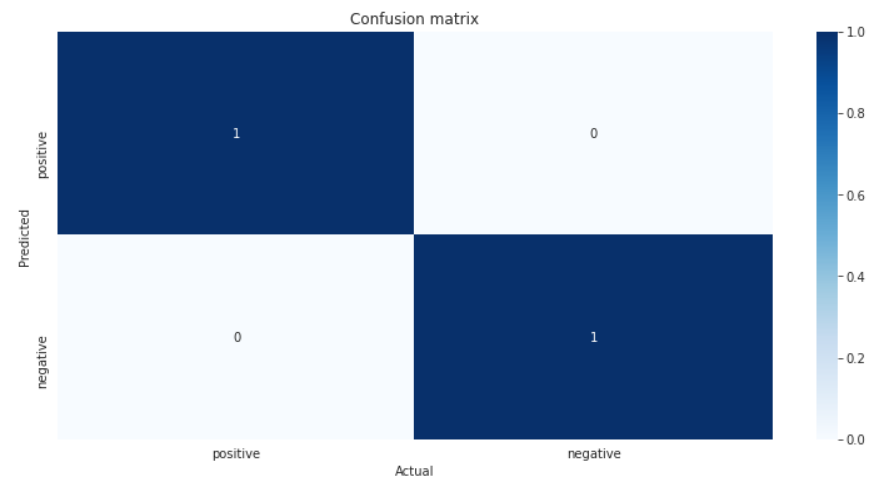
### Classification report

print("Classification report") print(classification\_report(labels\_test,svc\_pred))



### Confusion Matrix

aux\_df = df[['Category', 'Category\_Code']].drop\_duplicates().sort\_values('Category\_Code') conf\_matrix = confusion\_matrix(labels\_test, svc\_pred) plt.figure(figsize=(12.8,6)) sns.heatmap(conf\_matrix, annot=True, xticklabels=aux\_df['Category'].values, yticklabels=aux\_df['Category'].values, cmap="Blues") plt.ylabel('Predicted') plt.xlabel('Actual') plt.title('Confusion matrix') plt.show()



base\_model = svm.SVC(random\_state = 8) base\_model.fit(features\_train, labels\_train) accuracy\_score(labels\_test, base\_model.predict(features\_test))

#### Output :

0.5

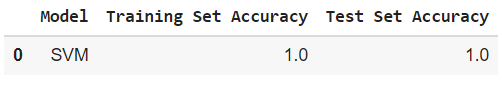
best\_svc.fit(features\_train, labels\_train) accuracy\_score(labels\_test, best\_svc.predict(features\_test))

Output : 1.0

d = { 'Model': 'SVM', 'Training Set Accuracy': accuracy\_score(labels\_train, best\_svc.predict(features\_train)), 'Test Set Accuracy': accuracy\_score(labels\_test, svc\_pred) }

df\_models\_svc = pd.DataFrame(d, index=[0])

df\_models\_svc



### Best Model Selection

path\_pickles = ""

list\_pickles = [ "df\_models\_knnc.pickle", "df\_models\_rfc.pickle", "df\_models\_svc.pickle" ]

df\_summary = pd.DataFrame()

for pickle\_ in list\_pickles:

path = path\_pickles + pickle

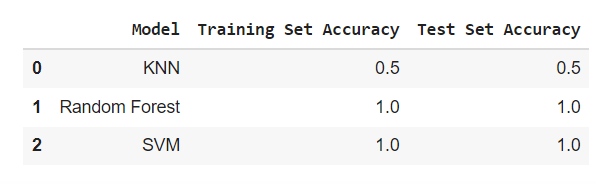
with open(path, 'rb') as data:

`df = pickle.load(data)`

df\_summary = df\_summary.append(df)

df\_summary = df\_summary.reset\_index().drop('index', axis=1)

df\_summary



df\_summary.sort\_values('Test Set Accuracy', ascending=False)

