**Preprocessing**

import os

import csv

import nltk

import numpy as np

import pandas as pd

import seaborn as sb

from nltk.stem import SnowballStemmer

from nltk.stem.porter import PorterStemmer

from nltk.tokenize import word\_tokenize

test\_filename = 'Dataset/News\_Data/test.csv'

train\_filename = 'Dataset/News\_Data/train.csv'

valid\_filename = 'Dataset/News\_Data/valid.csv'

train\_news = pd.read\_csv(train\_filename)

test\_news = pd.read\_csv(test\_filename)

valid\_news = pd.read\_csv(valid\_filename)

def data\_obs():

print("training dataset size:")

print(train\_news.shape)

print(train\_news.head(10))

print(test\_news.shape)

print(test\_news.head(10))

print(valid\_news.shape)

print(valid\_news.head(10))

data\_obs()

def create\_distribution(dataFile):

return sb.countplot(x='Label', data=dataFile, palette='hls')

create\_distribution(train\_news)

create\_distribution(test\_news)

create\_distribution(valid\_news)

def data\_qualityCheck():

print("Checking data qualitites...")

train\_news.isnull().sum()

train\_news.info()

print("check finished.")

test\_news.isnull().sum()

test\_news.info()

valid\_news.isnull().sum()

valid\_news.info()

data\_qualityCheck()

def stem\_tokens(tokens, stemmer):

stemmed = []

for token in tokens:

stemmed.append(stemmer.stem(token))

return stemmed

def process\_data(data,exclude\_stopword=True,stem=True):

tokens = [w.lower() for w in data]

tokens\_stemmed = tokens

tokens\_stemmed = stem\_tokens(tokens, eng\_stemmer)

tokens\_stemmed = [w for w in tokens\_stemmed if w not in stopwords ]

return tokens\_stemmed

def create\_unigram(words):

assert type(words) == list

return words

def create\_bigrams(words):

assert type(words) == list

skip = 0

join\_str = " "

Len = len(words)

if Len > 1:

lst = []

for i in range(Len-1):

for k in range(1,skip+2):

if i+k < Len:

lst.append(join\_str.join([words[i],words[i+k]]))

else:

lst = create\_unigram(words)

return lst

porter = PorterStemmer()

def tokenizer(text):

return text.split()

def tokenizer\_porter(text):

return [porter.stem(word) for word in text.split()]

**Features Extraction**

import os

import numpy as np

import nltk

import nltk.corpus

import pandas as pd

import module1 as DataPrep

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.pipeline import Pipeline

from nltk.tokenize import word\_tokenize

from gensim.models.word2vec import Word2Vec

countV = CountVectorizer()

train\_count = countV.fit\_transform(DataPrep.train\_news['Statement'].values)

print(countV)

print(train\_count)

def get\_countVectorizer\_stats():

train\_count.shape

print(countV.vocabulary\_)

print(countV.get\_feature\_names()[:25])

tfidfV = TfidfTransformer()

train\_tfidf = tfidfV.fit\_transform(train\_count)

def get\_tfidf\_stats():

train\_tfidf.shape

print(train\_tfidf.A[:10])

tfidf\_ngram = TfidfVectorizer(stop\_words='english',ngram\_range=(1,4),use\_idf=True,smooth\_idf=True)

tagged\_sentences = nltk.corpus.treebank.tagged\_sents()

cutoff = int(.75 \* len(tagged\_sentences))

training\_sentences = DataPrep.train\_news['Statement']

print(training\_sentences)

def features(sentence, index):

return {

'word': sentence[index],

'is\_first': index == 0,

'is\_last': index == len(sentence) - 1,

'is\_capitalized': sentence[index][0].upper() == sentence[index][0],

'is\_all\_caps': sentence[index].upper() == sentence[index],

'is\_all\_lower': sentence[index].lower() == sentence[index],

'prefix-1': sentence[index][0],

'prefix-2': sentence[index][:2],

'prefix-3': sentence[index][:3],

'suffix-1': sentence[index][-1],

'suffix-2': sentence[index][-2:],

'suffix-3': sentence[index][-3:],

'prev\_word': '' if index == 0 else sentence[index - 1],

'next\_word': '' if index == len(sentence) - 1 else sentence[index + 1],

'has\_hyphen': '-' in sentence[index],

'is\_numeric': sentence[index].isdigit(),

'capitals\_inside': sentence[index][1:].lower() != sentence[index][1:]

}

def untag(tagged\_sentence):

return [w for w, t in tagged\_sentence]

#Using Word2Vec

with open("glove.6B.50d.txt", "rb") as lines:

w2v = {line.split()[0]: np.array(map(float, line.split()[1:]))

for line in lines}

class MeanEmbeddingVectorizer(object):

def \_\_init\_\_(self, word2vec):

self.word2vec = word2vec

self.dim = len(word2vec.itervalues().next())

def fit(self, X, y):

return self

def transform(self, X):

return np.array([

np.mean([self.word2vec[w] for w in words if w in self.word2vec]

or [np.zeros(self.dim)], axis=0)

for words in X

])

**Classification**

import os

import pickle

import numpy as np

import pandas as pd

import module1 as DataPrep

import matplotlib.pyplot as plt

import module2 as FeatureSelection

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.pipeline import Pipeline

from sklearn.naive\_bayes import MultinomialNB

from sklearn.linear\_model import LogisticRegression

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import KFold

from sklearn.metrics import confusion\_matrix, f1\_score, classification\_report

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import learning\_curve

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import average\_precision\_score

nb\_pipeline = Pipeline([

('NBCV',FeatureSelection.countV),

('nb\_clf',MultinomialNB())])

nb\_pipeline.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_nb = nb\_pipeline.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_nb == DataPrep.test\_news['Label'])

logR\_pipeline = Pipeline([

('LogRCV',FeatureSelection.countV),

('LogR\_clf',LogisticRegression())

])

logR\_pipeline.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_LogR = logR\_pipeline.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_LogR == DataPrep.test\_news['Label'])

svm\_pipeline = Pipeline([

('svmCV',FeatureSelection.countV),

('svm\_clf',svm.LinearSVC())

])

svm\_pipeline.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_svm = svm\_pipeline.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_svm == DataPrep.test\_news['Label'])

random\_forest = Pipeline([

('rfCV',FeatureSelection.countV),

('rf\_clf',RandomForestClassifier(n\_estimators=200,n\_jobs=3))

])

random\_forest.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_rf = random\_forest.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_rf == DataPrep.test\_news['Label'])

def build\_confusion\_matrix(classifier):

k\_fold = KFold(n\_splits=5)

scores = []

confusion = np.array([[0,0],[0,0]])

for train\_ind, test\_ind in k\_fold.split(DataPrep.train\_news):

train\_text = DataPrep.train\_news.iloc[train\_ind]['Statement']

train\_y = DataPrep.train\_news.iloc[train\_ind]['Label']

test\_text = DataPrep.train\_news.iloc[test\_ind]['Statement']

test\_y = DataPrep.train\_news.iloc[test\_ind]['Label']

classifier.fit(train\_text,train\_y)

predictions = classifier.predict(test\_text)

confusion += confusion\_matrix(test\_y,predictions)

score = f1\_score(test\_y,predictions)

scores.append(score)

return (print('Total statements classified:', len(DataPrep.train\_news)),

print('Score:', sum(scores)/len(scores)),

print('score length', len(scores)),

print('Confusion matrix:'),

print(confusion))

build\_confusion\_matrix(nb\_pipeline)

build\_confusion\_matrix(logR\_pipeline)

build\_confusion\_matrix(svm\_pipeline)

build\_confusion\_matrix(random\_forest)

#classifier

nb\_pipeline\_ngram = Pipeline([

('nb\_tfidf',FeatureSelection.tfidf\_ngram),

('nb\_clf',MultinomialNB())])

nb\_pipeline\_ngram.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_nb\_ngram = nb\_pipeline\_ngram.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_nb\_ngram == DataPrep.test\_news['Label'])

logR\_pipeline\_ngram = Pipeline([

('LogR\_tfidf',FeatureSelection.tfidf\_ngram),

('LogR\_clf',LogisticRegression(penalty="l2",C=1))

])

logR\_pipeline\_ngram.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_LogR\_ngram = logR\_pipeline\_ngram.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_LogR\_ngram == DataPrep.test\_news['Label'])

svm\_pipeline\_ngram = Pipeline([

('svm\_tfidf',FeatureSelection.tfidf\_ngram),

('svm\_clf',svm.LinearSVC())

])

svm\_pipeline\_ngram.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_svm\_ngram = svm\_pipeline\_ngram.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_svm\_ngram == DataPrep.test\_news['Label'])

random\_forest\_ngram = Pipeline([

('rf\_tfidf',FeatureSelection.tfidf\_ngram),

('rf\_clf',RandomForestClassifier(n\_estimators=300,n\_jobs=3))

])

random\_forest\_ngram.fit(DataPrep.train\_news['Statement'],DataPrep.train\_news['Label'])

predicted\_rf\_ngram = random\_forest\_ngram.predict(DataPrep.test\_news['Statement'])

np.mean(predicted\_rf\_ngram == DataPrep.test\_news['Label'])

build\_confusion\_matrix(nb\_pipeline\_ngram)

build\_confusion\_matrix(logR\_pipeline\_ngram)

build\_confusion\_matrix(svm\_pipeline\_ngram)

build\_confusion\_matrix(random\_forest\_ngram)

print('Naive-bayes Classifier:')

print(classification\_report(DataPrep.test\_news['Label'], predicted\_nb\_ngram))

print('LogisticRegression Classifier:')

print(classification\_report(DataPrep.test\_news['Label'], predicted\_LogR\_ngram))

print('SVM Classifier:')

print(classification\_report(DataPrep.test\_news['Label'], predicted\_svm\_ngram))

print('RandomForest Classifier:')

print(classification\_report(DataPrep.test\_news['Label'], predicted\_rf\_ngram))

DataPrep.test\_news['Label'].shape

parameters = {'rf\_tfidf\_\_ngram\_range': [(1, 1), (1, 2),(1,3),(1,4),(1,5)],

'rf\_tfidf\_\_use\_idf': (True, False),

'rf\_clf\_\_max\_depth': (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

}

gs\_clf = GridSearchCV(random\_forest\_ngram, parameters, n\_jobs=-1)

gs\_clf = gs\_clf.fit(DataPrep.train\_news['Statement'][:10000],DataPrep.train\_news['Label'][:10000])

gs\_clf.best\_score\_

gs\_clf.best\_params\_

gs\_clf.cv\_results\_

parameters = {'LogR\_tfidf\_\_ngram\_range': [(1, 1), (1, 2),(1,3),(1,4),(1,5)],

'LogR\_tfidf\_\_use\_idf': (True, False),

'LogR\_tfidf\_\_smooth\_idf': (True, False)

}

gs\_clf = GridSearchCV(logR\_pipeline\_ngram, parameters, n\_jobs=-1)

gs\_clf = gs\_clf.fit(DataPrep.train\_news['Statement'][:10000],DataPrep.train\_news['Label'][:10000])

gs\_clf.best\_score\_

gs\_clf.best\_params\_

gs\_clf.cv\_results\_

model\_file = 'Train\_model.sav'

pickle.dump(logR\_pipeline\_ngram,open(model\_file,'wb'))

**Prediction**

import numpy as np

import pandas as pd

import pickle

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import MultinomialNB

from sklearn.linear\_model import LogisticRegression

from sklearn import svm

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import KFold

from sklearn.metrics import confusion\_matrix, f1\_score, classification\_report

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import learning\_curve

var = input("Enter the news text:")

print("verify: " + str(var))

def detecting\_fake\_news(var):

load\_model = pickle.load(open('Train\_model.sav', 'rb'))

prediction = load\_model.predict([var])

prob = load\_model.predict\_proba([var])

return (print("The news statement is ",prediction[0]),

print("The prediction probability score is:",prob[0][1]))

if \_\_name\_\_ == '\_\_main\_\_':

detecting\_fake\_news(var)