Naman Thaker

20BCE529

IRS PRACTICAL 4

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
#import data
data = pd.read_csv('/kaggle/input/questions.csv')
```

```
#check slice of data
data.head()
```

```
# distribution of data for different class
data.is_duplicate.value_counts()
```

```
print('length of data: ', len(data))
print('shape of data: ', data.shape)
#no of entries in data is 405k which is quiet high
#so we can perform analysis on subset of data to save excecution time.
```

```
# taking subset of data
df = data[:50000]
```

```
# checking Nan values if any
df.isna().sum()
#we dont have any Nan values in subset of data
```

```
#printing few question pairs in the datset
for i in range(0,20):
    print(df.question1[i])
    print(df.question2[i])
    print()
```

```
from string import punctuation
import scipy #library for scientific calculations
import datetime
import nltk
from sklearn import re
from sklearn import pipeline
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, f1_score, accuracy_sc


# Cleaning text in question

# This list is so extensive that we can look back at it as a reference when
# needed for other regular expression related work.

SPECIAL_TOKENS = {'non-ascii': 'non_ascii_word'}

def normalized_text(text, stem_words=True):
    def pad_str(s):
        return ' '+s+' '

    if pd.isnull(text):  #If null
        return ''

    # Empty question

    if type(text) != str or text=='':  #if text type is not string
        return ''

    # Clean the text
    text = re.sub("\'s", " ", text) # we have cases like "Sam is" or "Sam's" (i.e. his) th
    text = re.sub(" whats ", " what is ", text, flags=re.IGNORECASE) # replace whats by wh
    text = re.sub("\'ve", " have ", text) # replace 've by have
    text = re.sub("can't", "can not", text) # replace can't by can not
    text = re.sub("n't", " not ", text) # replace n't by not
    text = re.sub("i'm", "i am", text, flags=re.IGNORECASE) # replace i'm by i am and igno
    text = re.sub("\'re", " are ", text) # replace 're by are
    text = re.sub("\'d", " would ", text) # replace 'd by would
    text = re.sub("\'ll", " will ", text) # replace 'll by will
    text = re.sub("e\.g\.", " eg ", text, flags=re.IGNORECASE) # replace e.g. by eg and ig
    text = re.sub("b\.g\.", " bg ", text, flags=re.IGNORECASE) # replace b.g. by bg and ig
    text = re.sub("(\d+)(kK)", " \g<1>000 ", text)
    text = re.sub("e-mail", " email ", text, flags=re.IGNORECASE)
    text = re.sub("(the[\s]+|The[\s]+)?U\.S\.A\.", " America ", text, flags=re.IGNORECASE)
    text = re.sub("(the[\s]+|The[\s]+)?United State(s)?", " America ", text, flags=re.IGNO
    text = re.sub("\(s\)", " ", text, flags=re.IGNORECASE)
    text = re.sub("[c-fC-F]\:\/", " disk ", text)
    # remove comma between numbers, i.e. 15,000 -> 15000
    text = re.sub('(?<=[0-9])\,(?=[0-9])', "", text)

    # add padding to punctuations and special chars, we still need them later
    text = re.sub('\$', " dollar ", text)
    text = re.sub('\%', " percent ", text)
    text = re.sub('\&', " and ", text)

    text = re.sub('[^\x00-\x7F]+', pad_str(SPECIAL_TOKENS['non-ascii']), text)

    # Indian Currency
    text = re.sub("(?<=[0-9])rs ", " rs ", text, flags=re.IGNORECASE)
    text = re.sub(" rs(?=[0-9])", " rs ", text, flags=re.IGNORECASE)

    # cleaning text rules from : https://www.kaggle.com/currie32/the-importance-of-cleanin
    text = re.sub(r" (the[\s]+|The[\s]+)?US(A)? ", " America ", text)
    text = re.sub(r" UK ", " England ", text, flags=re.IGNORECASE)
    text = re.sub(r" india ", " India ", text)
    text = re.sub(r" switzerland ", " Switzerland ", text)
    text = re.sub(r" china ", " China ", text)
    text = re.sub(r" chinese ", " Chinese ", text)
    text = re.sub(r" imrovement ", " improvement ", text, flags=re.IGNORECASE)
    text = re.sub(r" intially ", " initially ", text, flags=re.IGNORECASE)
    text = re.sub(r" quora ", " Quora ", text, flags=re.IGNORECASE)
    text = re.sub(r" dms ", " direct messages ", text, flags=re.IGNORECASE)
    text = re.sub(r" demonitization ", " demonetization ", text, flags=re.IGNORECASE)
    text = re.sub(r" actived ", " active ", text, flags=re.IGNORECASE)
    text = re.sub(r" kms ", " kilometers ", text, flags=re.IGNORECASE)
    text = re.sub(r" cs ", " computer science ", text, flags=re.IGNORECASE)
    text = re.sub(r" upvote", " up vote", text, flags=re.IGNORECASE)
    text = re.sub(r" iPhone ", " phone ", text, flags=re.IGNORECASE)
    text = re.sub(r" \0rs ", " rs ", text, flags=re.IGNORECASE)
    text = re.sub(r" calender ", " calendar ", text, flags=re.IGNORECASE)
    text = re.sub(r" ios ", " operating system ", text, flags=re.IGNORECASE)
    text = re.sub(r" gps ", " GPS ", text, flags=re.IGNORECASE)
    text = re.sub(r" gst ", " GST ", text, flags=re.IGNORECASE)
    text = re.sub(r" programing ", " programming ", text, flags=re.IGNORECASE)
    text = re.sub(r" bestfriend ", " best friend ", text, flags=re.IGNORECASE)
    text = re.sub(r" dna ", " DNA ", text, flags=re.IGNORECASE)
    text = re.sub(r" III ", " 3 ", text)
    text = re.sub(r" banglore ", " Banglore ", text, flags=re.IGNORECASE)
    text = re.sub(r" J K ", " JK ", text, flags=re.IGNORECASE)
    text = re.sub(r" J\.K\. ", " JK ", text, flags=re.IGNORECASE)

    # replace the float numbers with a random number

    text = re.sub('[0-9]+\.[0-9]+', " 87 ", text)

    #Removing Punctuations
    text = [word for word in text if word not in punctuation]
    text = ''.join(text)
    text = text.lower()
    # Return a list of words
```

```
#applying text cleaning function to question text
df['question1'] = df['question1'].apply(normalized_text)
df['question2'] = df['question2'].apply(normalized_text)
```

```
#checking the cleaned text to see changes made
for i in range(0,20):
    print(df.question1[i])
    print(df.question2[i])
    print()
```

```
#data head
df.head()
```

- 1. ▪ *** Bag of Words + XGBOOST**

```
#using word vector of word_count and frequency withpout capturing the meaning of word
#r'\w{1,}' indiactes 1 or more word

CV = CountVectorizer(analyzer='word', stop_words='english', token_pattern=r'\w{1,}')
q1_trans = CV.fit_transform(df['question1'].values)
q2_trans = CV.fit_transform(df['question2'].values)
```

```
#scipy.sparse.hstack will stack sparse matrix columnwise, and stacking them side by side

X = scipy.sparse.hstack((q1_trans, q2_trans))
y = df.is_duplicate.values
```

```
#splitting traing set and test set for training and validating model for classification
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.30, random_state = 42)
```

```
#gradient Boosting Model used
#start time
st = datetime.datetime.now()

classifier1 = XGBClassifier(max_depth=50, n_estimators=80, learning_rate=0.1, colsample_by
objective='binary:logistic', eta=0.3, silent=1, subsample=0.8)
#fitting the model
print(classifier1.fit(X_train, y_train))
#predicting if pair is duplicate or not
prediction_CV = classifier1.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, prediction_CV))
print("Accuracy score: \n", accuracy_score(y_test, prediction_CV))
print("Classification report:\n", classification_report(y_test, prediction_CV))
print("F1 Score:\n ",f1_score(y_test, prediction_CV))
```

```
et = datetime.datetime.now()
print("Code run-time: ", et-st)
```

* **TF-IDF (word level) + XGBOOST **

```
#TF-IDF vectorizer
#5000 features were used for tfidf vectorizer
#r'\w{1,}'  indicates more than 1 word

tfidf = TfidfVectorizer(analyzer='word', max_features=5000, token_pattern=r'\w{1,}')

q1word_trans = tfidf.fit_transform(df['question1'].values)
q2word_trans = tfidf.fit_transform(df['question2'].values)

X = scipy.sparse.hstack((q1word_trans,q2word_trans))
y = df.is_duplicate.values
```

```
# train-test-split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.33, random_state = 42)
```

```
# Xg Boost classifier for word level vectorizer

st = datetime.datetime.now()

classifier2 = XGBClassifier(max_depth=50, n_estimators=80, learning_rate=0.1, colsample_by
objective='binary:logistic', eta=0.3, silent=1, subsample=0.8)
#fitting the model with traing data
print(classifier2.fit(X_train, y_train))

#predicting the test data
prediction_tfidf = classifier2.predict(X_test)

#Performance evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, prediction_tfidf))
print("Accuracy score: \n", accuracy_score(y_test, prediction_tfidf))
print("Classification report:\n", classification_report(y_test, prediction_tfidf))
print("F1 Score:\n ",f1_score(y_test, prediction_tfidf))

et = datetime.datetime.now()
print("Code run-time: ", et-st)
```

- **ngram level TFIDF + XGBOOST

```
#TF-IDF ngram level vectorizer
#5000 features were used for tfidf vectorizer
#r'\w{1,}'  indicates more than 1 word
#ngram_range = (1,3) means 2 and 3 features are used
```

```
tfidf = TfidfVectorizer(analyzer='word',ngram_range=(1,3), max_features=5000, token_patter

q1ngram_trans = tfidf.fit_transform(df['question1'].values)
q2ngram_trans = tfidf.fit_transform(df['question2'].values)

X = scipy.sparse.hstack((q1ngram_trans,q2ngram_trans))
y = df.is_duplicate.values


# train-test-split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.30, random_state = 42)


# Xg Boost classifier for ngram_range=(1,3) level vectorizer

st = datetime.datetime.now()

classifier3 = XGBClassifier(max_depth=50, n_estimators=80, learning_rate=0.1, colsample_by
objective='binary:logistic', eta=0.3, silent=1, subsample=0.8)
#fitting the model with traing data
print(classifier3.fit(X_train, y_train))

#predicting the test data
prediction_tfidf = classifier3.predict(X_test)

#Performance evaluation
print("ngram_range Confusion Matrix:\n", confusion_matrix(y_test, prediction_tfidf))
print("ngram_range Accuracy score: \n", accuracy_score(y_test, prediction_tfidf))
print("ngram_range Classification report:\n", classification_report(y_test, prediction_tfi
print("ngram_range F1 Score:\n ",f1_score(y_test, prediction_tfidf))

et = datetime.datetime.now()
print("Code run-time: ", et-st)


#TF-IDF ngram level vectorizer
#5000 features were used for tfidf vectorizer
#r'\w{1,}'  indicates more than 1 word
#ngram_range = (2,3) means 2 and 3 features are used

tfidf = TfidfVectorizer(analyzer='word',ngram_range=(2,3), max_features=5000, token_patter

q1ngram_trans = tfidf.fit_transform(df['question1'].values)
q2ngram_trans = tfidf.fit_transform(df['question2'].values)

X = scipy.sparse.hstack((q1ngram_trans,q2ngram_trans))
y = df.is_duplicate.values


# train-test-split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.30, random_state = 42)


# Xg Boost classifier for ngram_range=(1,3) level vectorizer

st = datetime.datetime.now()
```

```
classifier4 = XGBClassifier(max_depth=50, n_estimators=80, learning_rate=0.1, colsample_by
objective='binary:logistic', eta=0.3, silent=1, subsample=0.8)
#fitting the model with traing data
print(classifier4.fit(X_train, y_train))

#predicting the test data
prediction_tfidf = classifier4.predict(X_test)

#Performance evaluation
print("ngram_range(2,3) Confusion Matrix:\n", confusion_matrix(y_test, prediction_tfidf))
print("ngram_range(2,3) Accuracy score: \n", accuracy_score(y_test, prediction_tfidf))
print("ngram_range(2,3) Classification report:\n", classification_report(y_test, predictio
print("ngram_range(2,3) F1 Score:\n ",f1_score(y_test, prediction_tfidf))

et = datetime.datetime.now()
print("Code run-time: ", et-st)
```

* Char level TFIDF + XGBOOST**

```
#TF-IDF ngram level vectorizer
#5000 features were used for tfidf vectorizer
#r'\w{1,}'  indicates more than 1 word
#ngram_range = (1,3) means 2 and 3 features are used
#char level analyzer is used

tfidf = TfidfVectorizer(analyzer='char',ngram_range=(1,3), max_features=5000, token_patter

q1char_trans = tfidf.fit_transform(df['question1'].values)
q2char_trans = tfidf.fit_transform(df['question2'].values)

X = scipy.sparse.hstack((q1char_trans,q2char_trans))
y = df.is_duplicate.values


# train-test-split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size = 0.30, random_state = 42)


# Xg Boost classifier for char level vectorizer

st = datetime.datetime.now()

classifier5 = XGBClassifier(max_depth=50, n_estimators=80, learning_rate=0.1, colsample_by
objective='binary:logistic', eta=0.3, silent=1, subsample=0.8)
#fitting the model with traing data
print(classifier5.fit(X_train, y_train))

#predicting the test data
prediction_tfidf = classifier5.predict(X_test)

#Performance evaluation
print("char level Confusion Matrix:\n", confusion_matrix(y_test, prediction_tfidf))
print("char level Accuracy score: \n", accuracy_score(y_test, prediction_tfidf))
print("char level Classification report:\n", classification_report(y_test, prediction_tfid
print("char level F1 Score:\n ",f1_score(y_test, prediction_tfidf))

et = datetime.datetime.now()
print("Code run-time: ", et-st)
```

Conclusion

```
### Challenge was to create the model which can classify between similar question asked on

### text processing used is so extensive that we can use it as a reference when
# needed for other regular expression related work.###

### Initially Bag of Words model is used for vectorization with XGBOOST for classification

### TFIDF vectorizer at word level, n-Gram level, Char level is used.

### Without sentiment capturing, best accuracy of 75.73% is achieved using
###extreme gradient boosting on character level TF-IDF....
```