NLP_project8_Shell without outputs_Edouard_Toutounji_june_20_2020

October 27, 2025

#NLP - Project 8 : Edouard Toutounji : June_19_2020

0.1 1- Essential preprocessing libraries

[ ]:
```
# NLP - Project8 : Edouard Toutounji June 20_2020

# 1- Essential preprocessing libraries

import contractions
import re, string, unicodedata
import contractions
from bs4 import BeautifulSoup

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder

import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

0.2 2- Data load

[ ]:
```
# 2.1 - Data load

from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv ('/content/drive/My Drive/Colab Notebooks/Tweets.csv')
data.shape
data.info()
```

```
data.head()
data.isnull().sum(axis=0)
```

[ ]:
```
# 2.2 'airline_sentiment' needs to be encoded to integers

data['airline_sentiment'].value_counts()
```

[ ]:
```
labelencoder = LabelEncoder()
data['airline_sentiment_coded'] = labelencoder.
 ↪fit_transform(data['airline_sentiment'])
data.airline_sentiment_coded.value_counts()
```

[ ]:
```
# 2.3 Display Expansion and keeping only the 2 columns needed


# display's expansion
pd.set_option('display.max_colwidth', None)


# keep columns in question
data = data[['airline_sentiment_coded','text']]
data.head(10)
```

0.3 3- Elementary preprocessing functions , and then an encapsulating normaling function.

[ ]:
```
# 3- Elementary preprocessing functions ,  and then and encapsulating normaling␣
 ↪function.



stopwords = stopwords.words('english')
lemmatizer = WordNetLemmatizer()

 # # # # # # # # # # # # # # # # # # # # # # # #
 # The 9 elementary functions:

def strip_html(words):
    temp = BeautifulSoup( words , 'html.parser')
    return temp.get_text()

def replace_contractions(words):
    temp = contractions.fix(words)
    return temp

def remove_numbers(words):
    temp = re.sub(r'\d+', '', words)
    return temp
```

```python
def tokenize_text(words):
    temp = nltk.word_tokenize(words)
    return temp

def remove_non_ascii (words):
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii','ignore').
 ↪decode('utf-8', 'ignore')
        new_words.append(new_word)
    return new_words

def to_lowercase(words):
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation (words):
    new_words = []
    for word in words:
        new_word = re.sub( r'[^\w\s]','',word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def remove_stopwords (words):
    new_words = []
    for word in words:
        if word not in stopwords:
            new_words.append(word)
    return new_words

def lemmatize_words (words):
    new_words = []
    for word in words:
            new_words.append(lemmatizer.lemmatize(word))
    return new_words

# # # # # # # # # # # # # # # # # # # # # # # #


# The encapsulating Normalising function

def normalize(words):
```

```python
    words = strip_html(words)
    words = replace_contractions(words)
    words = remove_numbers(words)
    words = tokenize_text(words)
    words = remove_non_ascii (words)
    words = to_lowercase(words)
    words = remove_punctuation (words)
    words = remove_stopwords (words)
    words = lemmatize_words (words)
    return ' '.join(words)


# normalise will iterate on all the cells of the data['text'] column

for i,row in data.iterrows():
    words = data.at[i , 'text']
    words = normalize (words)
    data.at[i , 'text'] = words

data.head(10)
```

##4- Libraries for vectorisation and then ML classification

[ ]:
```python
# 4- Libraries for vectorisation and then ML classification

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

from sklearn.ensemble import RandomForestClassifier
```

0.4 5- Random Forest Model using CountVectoriser

[ ]:
```python
# 5- Random Forest Model using CountVectoriser

vectorizer = CountVectorizer( max_features = 2000)

X = vectorizer.fit_transform(data['text'])
X = X.toarray()
```

```
X.shape

y = data['airline_sentiment_coded']
y.shape


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 7)



# Fitting the model
forest = RandomForestClassifier(n_estimators = 20 , n_jobs=4)
forest.fit(X_train, y_train)



# Accuracy Score on the whole Data
print('Accuracy score on the whole Data')
print(np.mean(cross_val_score(forest, X, y , cv =20)))




# Confusion matrix
y_pred = forest.predict(X_test)
cm = confusion_matrix (y_test, y_pred)
print(cm)

df_cm = pd.DataFrame( cm, index = [i for i in '012'] , columns = [i for i in␣
 ↪'012'])
plt.figure( figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

##6- Random Forest Model using TfidfVectoriser

[ ]:
```
# 6- Random Forest Model using TfidfVectoriser

vectoriser = TfidfVectorizer( max_features = 2000)

X = vectorizer.fit_transform(data['text'])
X = X.toarray()
X.shape

y = data['airline_sentiment_coded']
y.shape


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state = 7)
```

```python
# Fitting the model
forest = RandomForestClassifier(n_estimators = 20 , n_jobs=4)
forest.fit(X_train, y_train)


# Accuracy Score on the whole Data
print('Accuracy score on the whole Data')
print(np.mean(cross_val_score(forest, X, y , cv =20)))



# Confusion matrix
y_pred = forest.predict(X_test)
cm = confusion_matrix (y_test, y_pred)
print(cm)

df_cm = pd.DataFrame( cm, index = [i for i in '012'] , columns = [i for i in␣
↪'012'])
plt.figure( figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

##7- Final thoughts on CountVectoriser vs. TfidfVectoriser

Tfdif is barely slighlty better at classifying the text.

Accuracy scores: * CountVectorizer: 0.7155737704918033 * TfIdfVectorizer: 0.7185109289617486

Also the results took a long time to be processed on Colab with the hyperpapramters above.

The initial trials for both Vectotization approaches was initially run with half the above: * max_features = 1000 * n_estimators = 10 * cv = 10

The results were faster but slighly less in terms of accuracy, both approaching the 70% correct classification rate.

Accuracy scores: * CountVectorizer: 0.6895491803278688 * TfIdfVectorizer: 0.6962431693989071

Thank you GL team, the journey was not easy but so much worth it!

Edouard Toutounji

[ ]:

```python

```