Manikkandan1404 /
FakeNewsDetectionusingNLP-phase-

Code Issues Pull requests Actions Projects Wiki Security Insights Set

FakeNewsDetectionusingNLP-phase- / main.py

Manikkandan1404 Add files via upload now

222 lines (178 loc) · 13.5 KB

Code Blame Raw

```python
# %% [markdown]
# *Fake News Detection*
#
# It has become humanly impossible to identify fake news on the online portals across the globe.The sheer vol
#
# The most crucial thing here is data which has been already available in the kaggle. We will be using differ

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:19:57.550035Z","iopub.execute_input":"2023-10-24
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

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

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

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:19:57.573835Z","iopub.execute_input":"2023-10-24
!pip install gensim # Gensim is an open-source library for unsupervised topic modeling and natural language p
import nltk
nltk.download('punkt')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
import nltk
import re
from nltk.corpus import stopwords
import seaborn as sns
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS

import plotly.express as px
```

```python
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

# %% [markdown]
# *Import the data & Clean ups*

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:12.47781Z","iopub.execute_input":"2023-10-24T
#importing data
fake_data = pd.read_csv('/kaggle/input/fake-and-real-news-dataset/Fake.csv')
print("fake_data",fake_data.shape)

true_data= pd.read_csv('/kaggle/input/fake-and-real-news-dataset/True.csv')
print("true_data",true_data.shape)

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.043858Z","iopub.execute_input":"2023-10-24
fake_data.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.066765Z","iopub.execute_input":"2023-10-24
true_data.head(5)

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.0866Z","iopub.execute_input":"2023-10-24T2
#adding additonal column to seperate betwee true & fake data
# true =1, fake =0
true_data['target'] = 1
fake_data['target'] = 0
df = pd.concat([true_data, fake_data]).reset_index(drop = True)
df['original'] = df['title'] + ' ' + df['text']
df.head()

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.737583Z","iopub.execute_input":"2023-10-24
df.isnull().sum()

# %% [markdown]
# *Data Clean up*
# - create a function here that will be responsible to remove any unneccesary words (Stopwords) from the data

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.781215Z","iopub.execute_input":"2023-10-24
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 2 and token not in stop_words
            result.append(token)

    return result

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.79379Z","iopub.execute_input":"2023-10-24T
# Transforming the unmatching subjects to the same notation
df.subject=df.subject.replace({'politics':'PoliticsNews','politicsNews':'PoliticsNews'})

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.819181Z","iopub.execute_input":"2023-10-24
sub_tf_df=df.groupby('target').apply(lambda x:x['title'].count()).reset_index(name='Counts')
sub_tf_df.target.replace({0:'False',1:'True'},inplace=True)
fig = px.bar(sub_tf_df, x="target", y="Counts",
             color='Counts', barmode='group',
             height=350)
fig.show()

```

```
# %% [markdown]
# - The data looks balanced and no issues on building the model

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:14.938929Z","iopub.execute_input":"2023-10-24
sub_check=df.groupby('subject').apply(lambda x:x['title'].count()).reset_index(name='Counts')
fig=px.bar(sub_check,x='subject',y='Counts',color='Counts',title='Count of News Articles by Subject')
fig.show()

# %% [markdown]
#

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:15.056981Z","iopub.execute_input":"2023-10-24
df['clean_title'] = df['title'].apply(preprocess)
df['clean_title'][0]

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:18.393978Z","iopub.execute_input":"2023-10-24
df['clean_joined_title']=df['clean_title'].apply(lambda x:" ".join(x))

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:18.444948Z","iopub.execute_input":"2023-10-24
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords = stop_words).generate(" ".join(df[
plt.imshow(wc, interpolation = 'bilinear')

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:40.057751Z","iopub.execute_input":"2023-10-24
maxlen = -1
for doc in df.clean_joined_title:
    tokens = nltk.word_tokenize(doc)
    if(maxlen<len(tokens)):
        maxlen = len(tokens)
print("The maximum number of words in a title is =", maxlen)
fig = px.histogram(x = [len(nltk.word_tokenize(x)) for x in df.clean_joined_title], nbins = 50)
fig.show()

# %% [markdown]
# *Creating Prediction Model*

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:57.606819Z","iopub.execute_input":"2023-10-24
X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_title, df.target, test_size = 0.2,random_
vec_train = CountVectorizer().fit(X_train)
X_vec_train = vec_train.transform(X_train)
X_vec_test = vec_train.transform(X_test)

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:20:59.134487Z","iopub.execute_input":"2023-10-24
#model
model = LogisticRegression(C=2)

#fit the model
model.fit(X_vec_train, y_train)
predicted_value = model.predict(X_vec_test)

#accuracy & predicted value
accuracy_value = roc_auc_score(y_test, predicted_value)
print(accuracy_value)

# %% [markdown]
# *Create the confusion matrix*

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:21:01.621433Z","iopub.execute_input":"2023-10-24
cm = confusion_matrix(list(y_test), predicted_value)
plt.figure(figsize = (7, 7))
sns.heatmap(cm, annot = True,fmt='g',cmap='viridis')

```

```
# %% [markdown]
# - 4465 Fake News have been Classified as Fake
# - 4045 Real News have been classified as Real

# %% [markdown]
# *Checking the content of news*

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:21:02.055182Z","iopub.execute_input":"2023-10-24
df['clean_text'] = df['text'].apply(preprocess)
df['clean_joined_text']=df['clean_text'].apply(lambda x:" ".join(x))

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:22:28.723648Z","iopub.execute_input":"2023-10-24
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 2000 , width = 1600 , height = 800 , stopwords = stop_words).generate(" ".join(df[
plt.imshow(wc, interpolation = 'bilinear')

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:23:30.622781Z","iopub.execute_input":"2023-10-24
maxlen = -1
for doc in df.clean_joined_text:
    tokens = nltk.word_tokenize(doc)
    if(maxlen<len(tokens)):
        maxlen = len(tokens)
print("The maximum number of words in a News Content is =", maxlen)
fig = px.histogram(x = [len(nltk.word_tokenize(x)) for x in df.clean_joined_text], nbins = 50)
fig.show()


# %% [markdown]
# *Predicting the Model*

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:27:23.232691Z","iopub.execute_input":"2023-10-24
X_train, X_test, y_train, y_test = train_test_split(df.clean_joined_text, df.target, test_size = 0.2,random_s
vec_train = CountVectorizer().fit(X_train)
X_vec_train = vec_train.transform(X_train)
X_vec_test = vec_train.transform(X_test)
model = LogisticRegression(C=2.5)
model.fit(X_vec_train, y_train)
predicted_value = model.predict(X_vec_test)
accuracy_value = roc_auc_score(y_test, predicted_value)
print(accuracy_value)

# %% [code] {"execution":{"iopub.status.busy":"2023-10-24T21:30:15.444444Z","iopub.execute_input":"2023-10-24
prediction = []
for i in range(len(predicted_value)):
    if predicted_value[i].item() > 0.5:
        prediction.append(1)
    else:
        prediction.append(0)
cm = confusion_matrix(list(y_test), prediction)
plt.figure(figsize = (6, 6))
sns.heatmap(cm, annot = True,fmt='g')

# %% [code]
```