2020

Real vs Fake New Classifier

Deepthi Sukumar
Grand Valley State University

Real Vs Fake News Classifier

By

Deepthi Sukumar

April,2020

REAL VS FAKE NEWS CLASSIFIER

By
Deepthi Sukumar

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

April. 2020

Dr Yonglei Tao	04/22/2020
Your Professor	**Date**

Table of Contents

Abstract

Fake News and Hoaxes started since the internet era. The fake news trend started mainly to deceive readers, increase readership and is often used as a means of psychological warfare. Advances in technology and the spread of news through different types of media, without actually verifying the facts, have increased the spread of fake news today. The main purpose of this project is to come up with a classifier which can differentiate fake news from the real news.

Introduction

The effects of fake news have increased exponentially in the recent past and something must be done to prevent this from continuing in the future. The dangerous effects of fake news, as previously defined, are made clear by events such as in which a man attacked a pizzeria due to a widespread fake news article. This story along with analysis provide evidence that humans are not very good at detecting fake news, possibly not better than chance . As such, the question remains whether machines can do a better job. A machine can solve the fake news problem using supervised learning that extracts features of the language and content only within the source in question, without utilizing any fact checker or knowledge base.

Real vs Fake news Classifier is based on naïve bayes algorithm. It was coded in python using Jupyter notebook IDE. Guardian newspaper provides an API(application program interface) which enables to populate the model with up-to date news. The training data was collected using the API, and has three features – Title, text and label. In the next step, preprocessing of the dataset like removing stop words, punctuation marks, missing fields was done. Naïve Bayers was used to classify the real news from the fake news. Multinomial NB feature of naïve which is suitable for classification with discrete features (e.g., word counts for text classification) was used, and the classifier was able to differentiate the news with an accuracy of over 85%. The classifier was tested by feeding news manually, and it was able to classify if the news was REAL or FAKE.

Background and Related Work

In machine learning, naive Bayes classifiers are the part of simple machine learning. There are a number of algorithms that focus on common principle. so it is not the only algorithm for training such classifiers. To check if the news is fake or real naive Bayers can be used.
The following formula for naive Bayes classification uses the probability of the previous event and compares it with the existing event. Each and every probability of the event is calculated and at last the overall probability of the news as compared to the dataset is calculated.
Therefore on calculating the overall probability, we can get the approximate value and can detect whether the news is real or fake.

$P(A|B) = P(B|A) \cdot P(A) / P(B)$, (1)
Finding the probability of event, A when event B is TRUE
$P(A)$ = PRIOR PROBABILITY
$P(A|B)$ = POSTERIOR PROBABILITY FINDING PROBABILITY:
$P(A|B1) = P(A1||B1). P(A2||B1). P(A3||B1)$ (2) $P(A|B2) = P(A1||B2). P(A||B2). P(A3||B2)$ (3) If the probability is 0
P (Word) = Word count +1/ (total number of words+ No. of unique words)
Therefore, by using this formula one can find the accuracy of the news.

Program Requirements

Real vs Fake news Classifier is based on the Machine learning naïve bayes algorithm, giving an accuracy of about 85% with the learning rate of 0.1. Further for test purposes, if a news is fed to the classifier,the classifier distinguishes the news to be a real or Fake news
Retrieval of Training Dataset is crucial in the process. It is important to distinguish the features and label the dataset inorder to feed to the classifier. Once the Dataset is retrieved, it is preprocessed.

Preprocessing data is a normal first step before training and evaluating the data. It is crucial that data is formatted properly, and meaningful features are included in order to have sufficient consistency that will result in the best possible results. or computer vision machine learning algorithms, pre-processing the data involves many steps including normalizing inputs and dimensionality reduction. The goal of these is to take away some of the unimportant distinguishing features between different texts, and to remove duplicate texts. There are portions of text that are not beneficial in the task of labeling the text as real or fake.

Implementation

TRAINING DATASET

The lack of manually labeled fake news datasets is certainly a bottleneck for advancing computationally intensive, text-based models that cover a wide array of topics. The Dataset for my classifier was collected from Guardian Newspaper API interface.

Guardian allows users to download news from an API interface daily. New York Times also provides an API for daily download of news.

PREPROCESSING

The CSV file was preprocessed , by first removing all the missing values. I also cleaned my file removing any Punctuation marks,and stopwords. Stemming and lemmatization processes could also be performed on the fields for extra cleanup. It takes a longer time to download, hence a copy of a file which was downloaded from the API was saved to use as the training dataset.

TECHNOLOGY

The whole project was coded in python using Jupyter Notebook IDE.

CLASSIFIER DETAILS

Naive Bayes, which is a popular algorithm in Machine Learning was used to find the accuracy of the news using multinomial NB.
It is a kind of algorithm used in text classification. The use of a token is correlated with the news that may be fake or not fake in naive Bayes classifier and then the accuracy of the news is calculated by using Bayes theorem.

Results, Evaluation, and Reflection

The classifier works fine with smaller datasets, but if the Dataset size is increased and when FAKE/REAL any one of them outweighs the other (for example if FAKE = 100 articles, and REAL = 30 Articles), there has been a significant reduction in accuracy.

Also, a google Collaberator would have been a good option for Data storage. Since I had to use my laptop to process a huge volume of data, I had to create a subset of the larger dataset. The Collaberator will be able to faster handle large volumes of data

Conclusions and Future Work

The Efficiency can be improved using about five classifier models like Bernoulli's Bayers, Passive aggressive classifier, SGD classifier, Support Vector Machines, logistic Regression, Logistic Regression CV, which can perform better classification and can give a better accuracy. Using these classifiers, if the outputs are (REAL,REAL,FAKE,FAKE,REAL), then the output would be REAL as it is the majority. Apart from the classifier, we can also build a fact detector and a stance detector. Combination of all these tools would be the best way to classify the news accurately

Bibliography

Conroy, N. Rubin and Chen. Y, "CIMT Detect: A Community Infused Matrix-Tensor Coupled Factorization," 52(1), pp.1-4, 2018

Markines, B. Cattuto, C., & F. Menczer, "Hybrid Machine- Crowd Approach," (pp. 41-48), April 2018.

H. Shaori, W. C. Wibowo, "Fake News Identification Characteristics Using Named Entity Recognition and Phrase Detection," 2018, 10th ICITEE, Universitas Indonesia.

[Kai Shu, Suhang Wang, Huan Liu, "Understanding User Profiles on Social Media for Fake News Detection," 2018, MIPR.

Stefan Helmsetter, HeikoPaulheim, "Weakly Supervised Learning for Fake News Detection on Twitter," IEEE, May 2018, ASONAM

Appendices

<u>API INTERFACE</u>

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().system('pip install html2text')
get_ipython().system('pip install python-twitter')
get_ipython().system('pip install xlsxwriter')
get_ipython().system('pip install openpyxl')
get_ipython().system('pip install beautifulsoup4')
get_ipython().system('pip install schedule')


# In[2]:


import json
import requests
import html2text,pandas as pd
import csv,xlsxwriter
from bs4 import BeautifulSoup
from openpyxl import load_workbook
import re
import schedule
import time
import xlwt


def guardian():


    API_KEY = "9a0083e8-554d-4306-be70-46d527d8903c"

    API_ENDPOINT = 'http://content.guardianapis.com/search'


    my_params = {
        'from-date': "",
        'to-date': "",
        'order-by':"oldest",
        'show-fields': "all",
        'page-size':10 ,
        'api-key': API_KEY,
        'lang':"en",
    }

    my_params['from-date'] = "2020-01-26"
    my_params['to-date'] = "2020-01-27"

    body_list=[]
    topic=[]
    startrow=None
    current_page = 1
    total_pages = 1

    while current_page <= total_pages:
            print("...page", current_page)
            my_params['page'] = current_page
            resp = requests.get(API_ENDPOINT, my_params)
            print(resp.url)
            print("\n")
            data = resp.json()
            current_page += 1
            total_pages=data['response']['pages']

            results=len(data["response"]["results"])

            for i in range(0,results):
                d=data["response"]["results"][i]["fields"]["bodyText"]
                body_list.append(d)
                topic.append("REAL")

            list_of_tuples = list(zip(body_list, topic))

            df = pd.DataFrame(list_of_tuples,index=None, columns=['text','label'])
            df.dropna(inplace=True)


            df.to_excel("guardian.xlsx",index=False,header=False)



    return(print("good"))

guardian

# In[ ]:


res=schedule.every().day.at("12:00").do(guardian)

while True:
    schedule.run_pending()
    time.sleep(1)
```

CLASSIFIER

```
#!/usr/bin/env python
# coding: utf-8

# In[12]:


#Importing the libraries which are required.
import pandas as pd
import numpy
import nltk
from nltk.corpus import stopwords
import string
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[13]:


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

# I AM USING MY OWN CSV FILE, because the API takes much time to download the latest news,
hence saved some for my sample
# In[14]:


df = pd.read_excel('guardian.xlsx') # If you have your input file as .xlsx (excel sheet) format
#df = pd.read_csv('fakerealnews.csv') # If you have your input file as .csv format
df = pd.DataFrame(df)
df.head()


# In[15]:


df.info()


# In[16]:


df.shape


# In[17]:


df.groupby("label").describe()

# IF YOU DON'T SEE ANY VERTICAL BAR IN THE BELOW GRAPH, THEN YOU DON'T HAVE
ANY MISSING VALUES IN ANY OF THE FIELDS
```

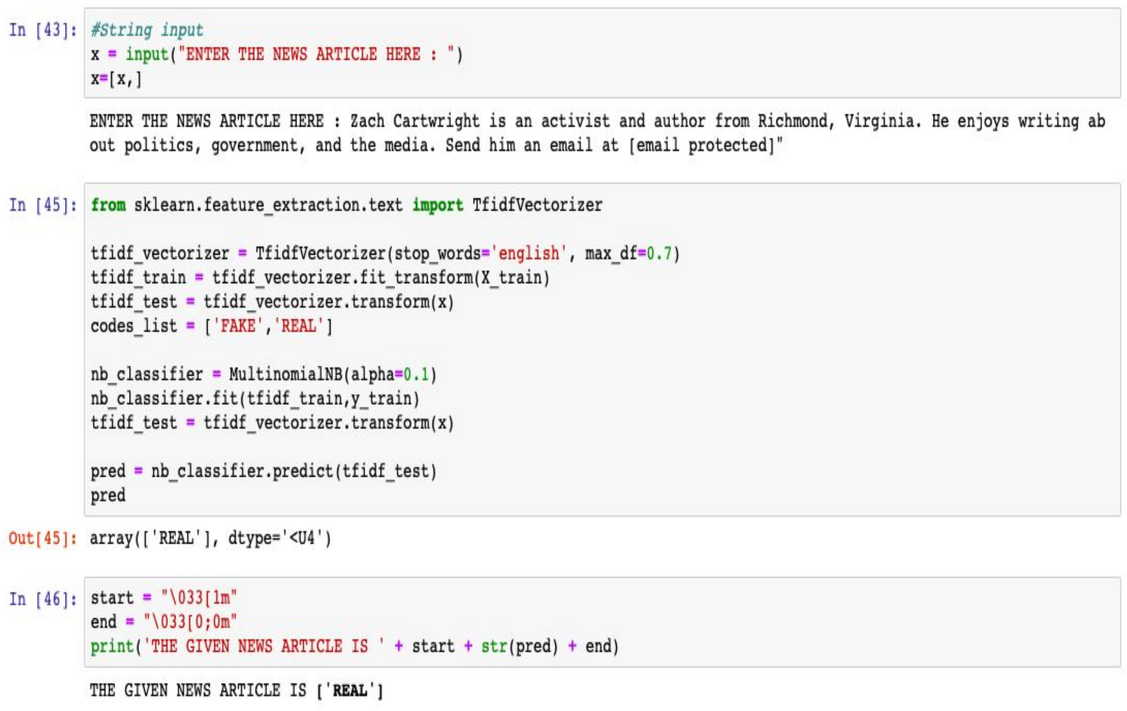
```
# In[18]:


import seaborn as sns
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='YlGnBu')

# HERE WE DROP ALL THE DUPLICATES ROWS IF EXISTS IN THE DATAFRAME
# In[19]:


df.drop_duplicates(inplace = True)
df.shape

# THE BELOW SET OF FUNCTIONS CLEANS THE CODE . I.E FROM PUNCTUATION REMOVAL
, STOPWORDS REMOVAL , STEMMING PROCESS , LEMMATIZATION PROCESS
# In[20]:


all_punctuations = string.punctuation + '‘’,:”][],'

def punc_remover(raw_text):
    no_punct = "".join([i for i in raw_text if i not in all_punctuations])
    return no_punct


# In[21]:


def stopword_remover(no_punc_text):
    words = no_punc_text.split()
    no_stp_words = " ".join([i for i in words if i not in stopwords.words('english')])
    return no_stp_words


# In[22]:


lemmer = nltk.stem.WordNetLemmatizer()
def lem(words):
    return " ".join([lemmer.lemmatize(word,'v') for word in words.split()])


# In[23]:


def text_cleaner(raw):
    cleaned_text = stopword_remover(punc_remover(raw))
    return lem(cleaned_text)


# In[24]:


df['SECTION_CLEANED'] = df['text'].apply(text_cleaner)
```

```
#df.to_excel("output.xlsx")  # TO SAVE THE FINAL CLEANED COPY IF I NEED


# In[25]:


df = pd.DataFrame(df)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df['SECTION_CLEANED'],df['label'],test_size=0.20,random_state=53,shuffle=True)


# In[26]:


from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer = CountVectorizer(stop_words = "english")
count_train = count_vectorizer.fit_transform(X_train.values)
count_test = count_vectorizer.transform(X_test.values)


# In[27]:


print(count_train)
print('\n')
print("feature_names",count_vectorizer.get_feature_names()[:10])


# In[28]:


# Import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train.values)
tfidf_test = tfidf_vectorizer.transform(X_test)
print(tfidf_train)
#this command gives importance for each and every word
print(tfidf_train[0:3])


# In[29]:


from sklearn.naive_bayes import MultinomialNB
import sklearn.metrics as metrics


# In[30]:
```

```
#USING NAIVE BAYES MODEL TO PREDICT ON TFIDFVECTORIZER CALCULATED VALUES

from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

nb_classifier = MultinomialNB(alpha=0.1)

nb_classifier.fit(tfidf_train,y_train)
pred = nb_classifier.predict(tfidf_test)

# Calculate the accuracy score: score
score = metrics.accuracy_score(y_test,pred)
print("Score from TFIDF Vectorizer", round(score*100,3) ,"%")

# Calculate the confusion matrix: cm
cm = metrics.confusion_matrix(y_test,pred, labels=['FAKE','REAL'])
print(cm)


# In[31]:


# Get the class labels: class_labels
class_labels = nb_classifier.classes_
print("class_labels" , class_labels)
print(" ")


# In[32]:


# Extract the features: feature_names
feature_names = tfidf_vectorizer.get_feature_names()
print("feature_names" , feature_names)
print(" ")


# In[33]:


# Zip the feature names together with the coefficient array and sort by weights: feat_with_weights
feat_with_weights = sorted(zip(nb_classifier.coef_[0], feature_names))

# Print the first class label and the top 20 feat_with_weights entries
print(class_labels[0], feat_with_weights[:20])
print(" ")

# Print the second class label and the bottom 20 feat_with_weights entries
print(class_labels[0], feat_with_weights[-20:])


# In[34]:
```

```
tfidf_vectorizer = TfidfVectorizer(analyzer='word',stop_words="english",lowercase = True)
tfidf_vectorizer.fit_transform(X_train.values.tolist())

print(tfidf_vectorizer.vocabulary_)

#  IF YOU WANT TO CATEGORIZE LOTS OF NEWS COLUMNS, JUST GIVE THE INPUT AS A
DATAFRAME LIKE THE ABOVE

# OR
# IF YOU HAVE ONLY ONE PIECE OF ARTICLE TO TEST REAL OR FAKE, THEN USE THE
BELOW SET OF CODES TO CLASSIFY THEM.
# In[35]:


# Import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vocab = tfidf_vectorizer.vocabulary_
tfidf_vectorizer = TfidfVectorizer(stop_words='english',vocabulary=vocab)


# # ENTER THE NEWS COLUMN HERE TO PREDICT 'REAL' OR 'FAKE'

# In[43]:


#String input
x = input("ENTER THE NEWS ARTICLE HERE : ")
x=[x,]


# In[45]:


from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
tfidf_test = tfidf_vectorizer.transform(x)
codes_list = ['FAKE','REAL']

nb_classifier = MultinomialNB(alpha=0.1)
nb_classifier.fit(tfidf_train,y_train)
tfidf_test = tfidf_vectorizer.transform(x)

pred = nb_classifier.predict(tfidf_test)
pred


# In[46]:


start = "\033[1m"
end = "\033[0;0m"
print('THE GIVEN NEWS ARTICLE IS ' + start + str(pred) + end)
```

OUTPUT



```
In [43]: #String input
         x = input("ENTER THE NEWS ARTICLE HERE : ")
         x=[x,]

         ENTER THE NEWS ARTICLE HERE : Zach Cartwright is an activist and author from Richmond, Virginia. He enjoys writing ab
         out politics, government, and the media. Send him an email at [email protected]"

In [45]: from sklearn.feature_extraction.text import TfidfVectorizer

         tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
         tfidf_train = tfidf_vectorizer.fit_transform(X_train)
         tfidf_test = tfidf_vectorizer.transform(x)
         codes_list = ['FAKE','REAL']

         nb_classifier = MultinomialNB(alpha=0.1)
         nb_classifier.fit(tfidf_train,y_train)
         tfidf_test = tfidf_vectorizer.transform(x)

         pred = nb_classifier.predict(tfidf_test)
         pred

Out[45]: array(['REAL'], dtype='<U4')

In [46]: start = "\033[1m"
         end = "\033[0;0m"
         print('THE GIVEN NEWS ARTICLE IS ' + start + str(pred) + end)

         THE GIVEN NEWS ARTICLE IS ['REAL']
```