CNN News category classifier Blackbox analysis

Abstract:

From ways of communicating instantly with news sources and content generated first-hand in image, video, and text form, to the publication of content across an ever-increasing array of channels. In particular, news organizations are increasingly leveraging AI to change the way news is generated, produced, published, and shared. Nowadays, as the publication of digital news articles has been increased through these organisations. This is where the latest technology like prediction using various modelling techniques or identification of similar content through topic modelling can be used to make use of resouces and time. This report is about how a classifier model can be used by the news agencies to classify their text articles into certain categories. Also, gives a critical analysis on blackbox of the created model to analyse how efficient the model can classify the given news text in the real world with showcasing the potential issues of the critique of context used.

In this report, we will analyze how a Naive Bayes classifier can be used to analyze the news article into given categories using neural networks. Also, testing the created model with existing data gives insights into what kind of pattern can be identified by the model, so that it can be trustworthy to implement. Deploying the model without analyzing, what types of news it can categorize and what it cant, leads to major problems and loss of credibility to major news agencies. For example, classifying some category as crime, which is not, leads to a different perspective of approach and affects the stakeholders and organizations mentioned in the news article. The explainability of this neural network gives some insights where and which situations the model can predict incorrectly about the given data. So, the news agencies take note of the report and provide highly precise data, like news headlines and descriptions, etc, for classification.

The layout of this notebook is organised as follows:

- Loading Of Libraries
- Loading Up Training/Test Data
- Preprocessing Of Text Data
- Analysis With Our Naive Bayes Classifier
- Evaluation Of Our Model's Performance
- AI Explainability Assessment With LIME
- Potential Issuses
- Conclusion

Loading Of Libraries

In [1]:

```
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install gensim
!{sys.executable} -m pip install pyLDAvis
!pip install kaggle
```

```
Collecting gensim
  Using cached gensim-3.8.3-cp37-cp37m-manylinux1_x86_64.whl (24.2 MB)
Requirement already satisfied: six>=1.5.0 in /opt/conda/lib/python3.7/site
-packages (from gensim) (1.14.0)
Requirement already satisfied: scipy>=0.18.1 in /opt/conda/lib/python3.7/s
ite-packages (from gensim) (1.4.1)
Requirement already satisfied: numpy>=1.11.3 in /opt/conda/lib/python3.7/s
ite-packages (from gensim) (1.18.1)
Processing /home/jovyan/.cache/pip/wheels/bb/1c/9c/412ec03f6d5ac7d41f4b965
bde3fc0d1bd201da5ba3e2636de/smart_open-2.0.0-py3-none-any.whl
Collecting boto3
  Downloading boto3-1.14.5-py2.py3-none-any.whl (128 kB)
     |████████████████████████████████| 128 kB 9.9 MB/s eta 0:00:01
Collecting boto
  Using cached boto-2.49.0-py2.py3-none-any.whl (1.4 MB)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-p
ackages (from smart-open>=1.8.1->gensim) (2.22.0)
Collecting botocore<1.18.0,>=1.17.5
  Downloading botocore-1.17.5-py2.py3-none-any.whl (6.3 MB)
     |████████████████████████████████| 6.3 MB 15.2 MB/s eta 0:00:01
Collecting jmespath<1.0.0,>=0.7.1
  Using cached jmespath-0.10.0-py2.py3-none-any.whl (24 kB)
Collecting s3transfer<0.4.0,>=0.3.0
  Using cached s3transfer-0.3.3-py2.py3-none-any.whl (69 kB)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.7/site-packages (from requests->smart-open>=1.8.1->gensim) (2019.11.28)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/opt/conda/lib/python3.7/site-packages (from requests->smart-open>=1.8.1->
gensim) (1.25.7)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/pyt
hon3.7/site-packages (from requests->smart-open>=1.8.1->gensim) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/lib/python3.7/
site-packages (from requests->smart-open>=1.8.1->gensim) (2.8)
Collecting docutils<0.16,>=0.10
  Using cached docutils-0.15.2-py3-none-any.whl (547 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /opt/conda/l
ib/python3.7/site-packages (from botocore<1.18.0,>=1.17.5->boto3->smart-op
en>=1.8.1->gensim) (2.8.1)
Installing collected packages: docutils, jmespath, botocore, s3transfer, b
oto3, boto, smart-open, gensim
  Attempting uninstall: docutils
    Found existing installation: docutils 0.16
    Uninstalling docutils-0.16:
      Successfully uninstalled docutils-0.16
Successfully installed boto-2.49.0 boto3-1.14.5 botocore-1.17.5 docutils-
0.15.2 gensim-3.8.3 jmespath-0.10.0 s3transfer-0.3.3 smart-open-2.0.0
Processing /home/jovyan/.cache/pip/wheels/3b/fb/41/e32e5312da9f440d34c4eff
0d2207b46dc9332a7b931ef1e89/pyLDAvis-2.1.2-py2.py3-none-any.whl
Requirement already satisfied: numpy>=1.9.2 in /opt/conda/lib/python3.7/si
te-packages (from pyLDAvis) (1.18.1)
Requirement already satisfied: wheel>=0.23.0 in /opt/conda/lib/python3.7/s
ite-packages (from pyLDAvis) (0.34.1)
Requirement already satisfied: jinja2>=2.7.2 in /opt/conda/lib/python3.7/s
ite-packages (from pyLDAvis) (2.11.0)
Collecting pytest
  Downloading pytest-5.4.3-py3-none-any.whl (248 kB)
     |████████████████████████████████| 248 kB 11.4 MB/s eta 0:00:01
Requirement already satisfied: numexpr in /opt/conda/lib/python3.7/site-pa
ckages (from pyLDAvis) (2.7.1)
Requirement already satisfied: pandas>=0.17.0 in /opt/conda/lib/python3.7/
site-packages (from pyLDAvis) (0.25.3)
```

```
Processing /home/jovyan/.cache/pip/wheels/3c/33/97/805b282e129f60bb4e87cea
622338f30b65f21eaf65219971f/funcy-1.14-py2.py3-none-any.whl
Requirement already satisfied: joblib>=0.8.4 in /opt/conda/lib/python3.7/s
ite-packages (from pyLDAvis) (0.14.1)
Processing /home/jovyan/.cache/pip/wheels/56/b0/fe/4410d17b32f1f0c3cf54cdf
b2bc04d7b4b8f4ae377e2229ba0/future-0.18.2-py3-none-any.whl
Requirement already satisfied: scipy>=0.18.0 in /opt/conda/lib/python3.7/s
ite-packages (from pyLDAvis) (1.4.1)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/lib/python3.
7/site-packages (from jinja2>=2.7.2->pyLDAvis) (1.1.1)
Requirement already satisfied: importlib-metadata>=0.12; python_version <
"3.8" in /opt/conda/lib/python3.7/site-packages (from pytest->pyLDAvis)
(1.5.0)
Requirement already satisfied: more-itertools>=4.0.0 in /opt/conda/lib/pyt
hon3.7/site-packages (from pytest->pyLDAvis) (8.2.0)
Collecting pluggy<1.0,>=0.12
  Using cached pluggy-0.13.1-py2.py3-none-any.whl (18 kB)
Requirement already satisfied: packaging in /opt/conda/lib/python3.7/site-
packages (from pytest->pyLDAvis) (20.1)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.7/site-pa
ckages (from pytest->pyLDAvis) (0.1.8)
Collecting py>=1.5.0
  Downloading py-1.8.2-py2.py3-none-any.whl (83 kB)
     |████████████████████████████████| 83 kB 4.6 MB/s  eta 0:00:01
Requirement already satisfied: attrs>=17.4.0 in /opt/conda/lib/python3.7/s
ite-packages (from pytest->pyLDAvis) (19.3.0)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/conda/lib/py
thon3.7/site-packages (from pandas>=0.17.0->pyLDAvis) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/si
te-packages (from pandas>=0.17.0->pyLDAvis) (2019.3)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-
packages (from importlib-metadata>=0.12; python_version < "3.8"->pytest->p
yLDAvis) (2.1.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packag
es (from packaging->pytest->pyLDAvis) (1.14.0)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.
7/site-packages (from packaging->pytest->pyLDAvis) (2.4.6)
Installing collected packages: pluggy, py, pytest, funcy, future, pyLDAvis
Successfully installed funcy-1.14 future-0.18.2 pluggy-0.13.1 py-1.8.2 pyL
DAvis-2.1.2 pytest-5.4.3
WARNING: pip is being invoked by an old script wrapper. This will fail in
a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on fixing th
e underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of runni
ng pip directly.
Processing /home/jovyan/.cache/pip/wheels/aa/e7/e7/eb3c3d514c33294d77ddd5a
856bdd58dc9c1fabbed59a02a2b/kaggle-1.5.6-py3-none-any.whl
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.7/site-
packages (from kaggle) (1.14.0)
Collecting urllib3<1.25,>=1.21.1
  Using cached urllib3-1.24.3-py2.py3-none-any.whl (118 kB)
Requirement already satisfied: certifi in /opt/conda/lib/python3.7/site-pa
ckages (from kaggle) (2019.11.28)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-p
ackages (from kaggle) (2.22.0)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.
7/site-packages (from kaggle) (2.8.1)
Processing /home/jovyan/.cache/pip/wheels/7c/26/30/5f3d95da00fe94d0c4a5ec5
b4ffd2e1ae18545f5fa61752e52/python_slugify-4.0.0-py2.py3-none-any.whl
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packa
```

```
ges (from kaggle) (4.42.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/pyt
hon3.7/site-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/lib/python3.7/
site-packages (from requests->kaggle) (2.8)
Collecting text-unidecode>=1.3
  Using cached text_unidecode-1.3-py2.py3-none-any.whl (78 kB)
Installing collected packages: urllib3, text-unidecode, python-slugify, ka
ggle
  Attempting uninstall: urllib3
    Found existing installation: urllib3 1.25.7
    Uninstalling urllib3-1.25.7:
      Successfully uninstalled urllib3-1.25.7
Successfully installed kaggle-1.5.6 python-slugify-4.0.0 text-unidecode-1.
3 urllib3-1.24.3
```

In [3]:

```
# Numerical Data Manipulation libraries
import pandas as pd
import numpy as np
import statistics as stat
import re

# Figure Plotting libraries
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns
sns.set()

# Naive Bayes libraries
import sklearn
from sklearn.naive_bayes import BernoulliNB      # Naive Bayes Classifier based on a Be
rnoulli Distribution
from sklearn.naive_bayes import GaussianNB       # Naive Bayes Classifier based on a Ga
ussian Distribution
from sklearn.naive_bayes import MultinomialNB    # Naive Bayes Classifier based on a Mu
ltinomial Distribution

# Machine Learning libraries
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from tensorflow.keras.layers import Dense, Input, Flatten
from tensorflow.keras.layers import Conv2D, Embedding, Dropout,Conv1D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import OneHotEncoder, StandardScaler,MinMaxScaler

# Text Analysis libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline

from gensim.models import TfidfModel
from gensim.corpora import Dictionary
from gensim.utils import tokenize
from gensim.utils import simple_preprocess
from gensim.corpora.textcorpus import remove_stopwords
from gensim.summarization import keywords
from gensim.models.ldamodel import LdaModel

import pyLDAvis
import pyLDAvis.gensim

# callbacks
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from lime.lime_text import LimeTextExplainer
```

```
from matplotlib import cm
import json

import os
```

```
Using TensorFlow backend.
```

Various libraries are imported for retrieving data from the source and data processing before feeding into the classifier to creation of model, evaluation and testing the model. Libraries like Pandas, numpy, json,re used for data loading and splitting up the training and testing data. Where sklearn, Keras, tensorflow and gensim are used to creating of Naïve Bayes model, converstion of text into vectors for mathematical purposes.

Loading Up Training/Test Data

For developing the high accurate model, which distinguish the given news articles into categories, a wide number of data is required for proper classification. The “News category dataset” contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost. The model trained on this dataset could be used to identify tags for untracked news articles or to identify the type of language used in different news articles.This dataset has wide number of categories which are mapped to the headlines and descriptions, this will be sorted and used in this modelling technique.

In this section, using the kaggle API, the dataset is accessed through in the json format. The kaggle website demands a key to access its API`s. This token key can be generated through the website by clicking on the create api token key option from the kaggle profile.

In [4]:

```
os.environ['KAGGLE_USERNAME'] = 'dineshduraisamy'
os.environ['KAGGLE_KEY'] = 'dece0dbf10a1761e26c6ca05210ef9aa'
```

In [5]:

```
import kaggle as kg

kg.api.authenticate()
kg.api.dataset_download_files(dataset="rmisra/news-category-dataset", unzip=True)
```

In [7]:

```
# load the complete dataset
with open('News_Category_Dataset_v2.json', 'r',) as f:
    news_list = f.readlines()

# convert each line (string) to json (dict)
news_json = list(map(json.loads,news_list))

print("Number of stories: ",len(news_json))

# view the first 10 elements in the list
news_json = news_json[0:100000]
news_json[0:2]
```

```
Number of stories:  200853
```

Out[7]:


```
[{'category': 'CRIME',
  'headline': 'There Were 2 Mass Shootings In Texas Last Week, But Only 1
On TV',
  'authors': 'Melissa Jeltsen',
  'link': 'https://www.huffingtonpost.com/entry/texas-amanda-painter-mass-
shooting_us_5b081ab4e4b0802d69caad89',
  'short_description': 'She left her husband. He killed their children. Ju
st another day in America.',
  'date': '2018-05-26'},
 {'category': 'ENTERTAINMENT',
  'headline': "Will Smith Joins Diplo And Nicky Jam For The 2018 World Cu
p's Official Song",
  'authors': 'Andy McDonald',
  'link': 'https://www.huffingtonpost.com/entry/will-smith-joins-diplo-and
-nicky-jam-for-the-official-2018-world-cup-song_us_5b09726fe4b0fdb2aa54120
1',
  'short_description': 'Of course it has a song.',
  'date': '2018-05-26'}]
```


From the above JSON file, we can see that each news artile has columns like category, headlines, short descriptions, author, link, and date of publication. In this analysis, the major contribution features are headline and short descriptions. The classes or labels in which the neural network function is the categories for classification of news articles. So, the news agencies can input their headlines and article text for predicting their certain category.

Preprocessing Of Text Data

This dataset contains numerous information about different news articles. But we will be using only the top 20 published categories in the total dataset. This is because, as the number of categories increases the accuracy of prediction becomes lesser in the real scenario. Also, it is not necessary to use all the categories in the prediction model.

In [9]:

```
news_data_frame = pd.DataFrame(data = news_json)
news_data_frame.drop(["authors","link"], axis = 1, inplace = True)
news_data_frame = news_data_frame[0:100000]
news_data_frame[0:5]
```

Out[9]:

	category	headline	short_description	date
0	CRIME	There Were 2 Mass Shootings In Texas Last Week...	She left her husband. He killed their children...	2018-05-26
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	Of course it has a song.	2018-05-26
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	The actor and his longtime girlfriend Anna Ebe...	2018-05-26
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	The actor gives Dems an ass-kicking for not fi...	2018-05-26
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	The "Dietland" actress said using the bags is ...	2018-05-26

In [11]:

```
#getting the top 20 categories from the dataset
Top_category = news_data_frame[["category","headline"]].groupby("category").count().sor
t_values(["headline"],ascending=False).head(20)
categories = sorted(Top_category.index.tolist())
categories
```

Out[11]:

```
['BLACK VOICES',
 'BUSINESS',
 'COMEDY',
 'CRIME',
 'ENTERTAINMENT',
 'GREEN',
 'HEALTHY LIVING',
 'IMPACT',
 'MEDIA',
 'PARENTS',
 'POLITICS',
 'QUEER VOICES',
 'RELIGION',
 'SPORTS',
 'STYLE',
 'TASTE',
 'THE WORLDPOST',
 'WEIRD NEWS',
 'WOMEN',
 'WORLD NEWS']
```

These are the categories that will be used as labels in this model which are sorted from the top 20. The neural networks cannot able to read data in the text format. As it is a machine it can only understand the numerics to identity the pattern. The headlines and the short descriptions from the dataset are combined to form a numeric identifier.

Insights from 1st model (Updating dataset)

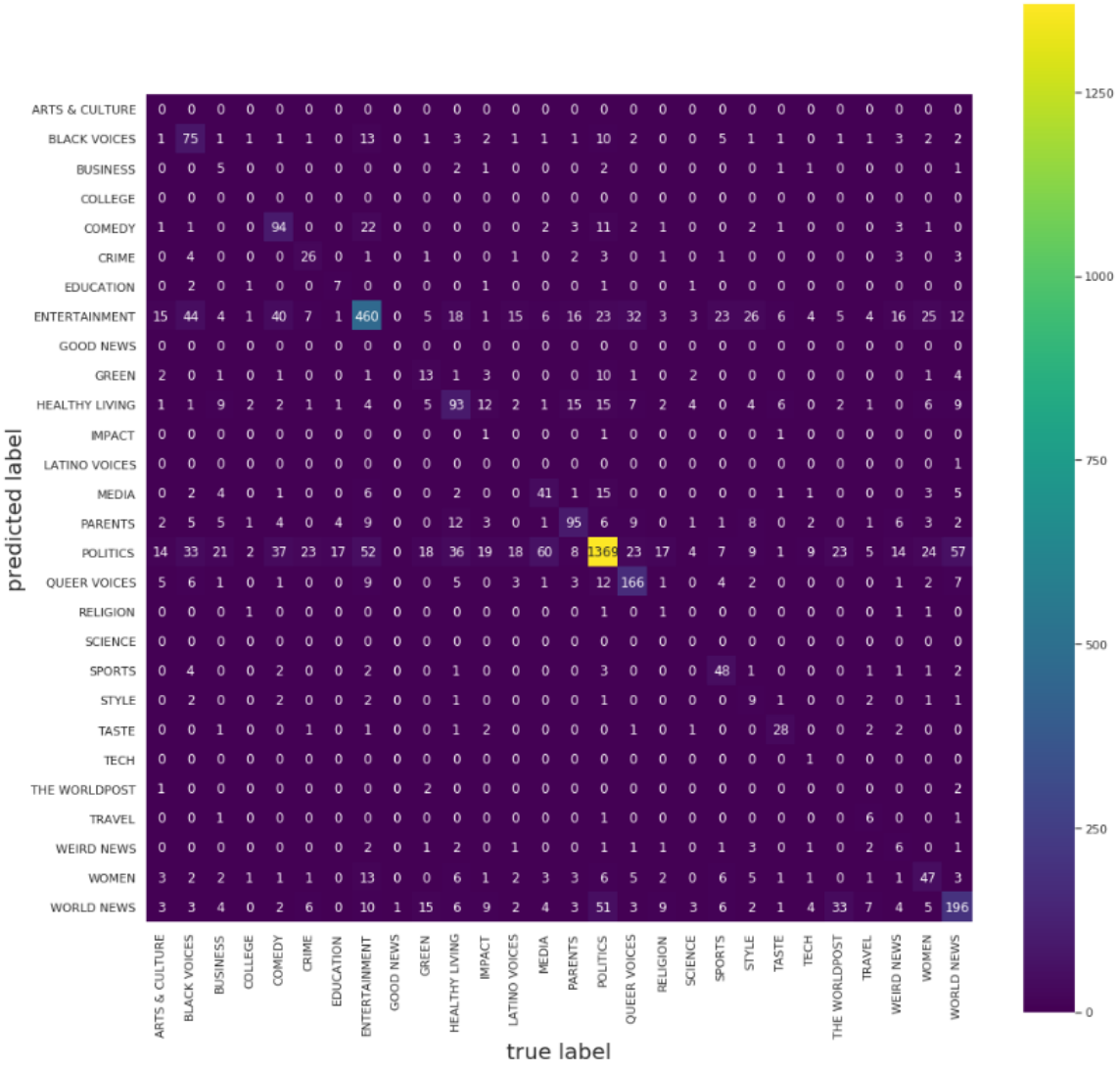
At first, the model has been created with the first 30000 data. The accuracy of the model was around 97.6% but by analyzing the confusion matrix, it comes to light that the dataset is imbalanced i.e, some of the news articles in the dataset has more number of data compared to other categories of news articles. The below-shown figure is the output of the confusion matrix of the first model. As the first 30000 data of the news category dataset has around 27k data only in politics, the training and the test data was more with the “POLITICS” category. All other categories only have very less training and test data compared to “POLITICS” and “ENTERTAINMENT” categories.

In the real world, the news articles can be from any category with number of data. But our 1st model has been created with more prediction for specific category. This lead to a false classification of a category if it is implemented, with more prediction of articles into “POLITICS” rather than its true category.

In [12]:

```
#![title](ConfusionMatrixNormalDataset.png)
from IPython.display import Image
Image("confusionmatrix2.PNG")
```

Out[12]:

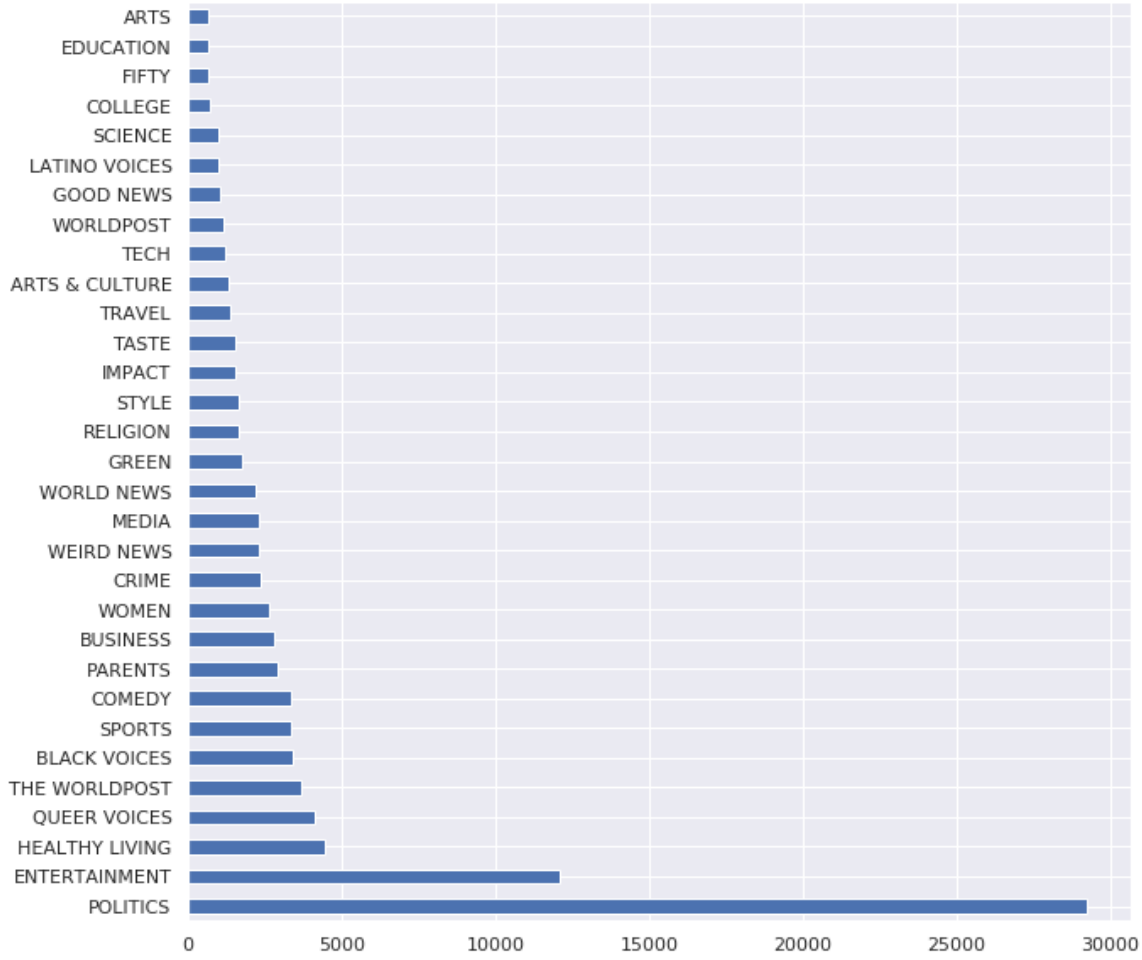


In [13]:

```
plt.figure(figsize=(10,10))
news_data_frame['category'].value_counts().plot(kind='barh')
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faadad97350>
```



The above figure shows the distribution of news articles in each category. It depicts that the “POLITICS” has very large data than the other categories. If the model has been created with this type of dataset, it takes most of the words into the “POLITICS” than predicting its real category. The model needs to created with balanced data, which is same number of data in all the categories.

So, there is a need of selection of appropriate data in each category before converting them into numeric identifiers after tokenizing. Here, the dataset has been sorted from maximum number of articels produced in each category and selected only the top 20 for this analysis. Each category had 1500 data i.e, articles, that brings the whole dataset to 30000 article in tolal.

In [14]:

```
# for each, create the 'story' by adding together the headline and the short_descriptio
n
news_data_frame["story"]  = [story['headline']+' - '+story['short_description'] for sto
ry in news_json]


news = pd.DataFrame(columns=news_data_frame.columns)
for category in categories:
    filtered_json = news_data_frame[news_data_frame["category"] == category].head(1500)
    news = pd.concat([news, pd.DataFrame(filtered_json)],ignore_index = True)

# look at first 10
news_data_frame = news
print("Number of stories: ",len(news_data_frame["story"]))
```

```
Number of stories:  30000
```

Before converting the stories into separate words, to maximise the accuracy of our prediction. The words like occurs often which doesnot add value to the analysis will be removed, example (is, the, a). This is because we want our classification model to identify the category using the distinct words rather than the words which is present in all the artilcles.

In [15]:

```
# do this for whole dataframe
news_data_frame['terms'] = [remove_stopwords(simple_preprocess(story,min_len=3)) for st
ory in news_data_frame['story']]
```

In [17]:

```
word_counts = {}

for terms in news_data_frame["terms"]:
    for word in terms:
        if word in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1

# sort the word_counts by counts
sorted_counts = {k: v for k, v in sorted(word_counts.items(), key=lambda item: item[1],
reverse=True)}
```

In [18]:

```
news_data_frame["terms"][:5]
```

Out[18]:

```
0    [warriors, coach, steve, kerr, calls, nfl, ban...
1    [historic, victory, barbados, elects, female, ...
2    [police, killed, black, americans, moment, col...
3    [bet, chairman, ceo, debra, lee, stepping, lee...
4    [women, pulled, stops, land, dream, job, jessi...
Name: terms, dtype: object
```

The above words show the final words from the article which will be used in our model for creating and these words are saved into the “terms” column in our dataset. It also shows the number of times the words occurred in the whole dataset, for example, “trump” words occurs 3855 times in our dataset.

Convert Text into Tokens

In the next step, we will be using the tokenizer to convert the words into numeric identifiers using the this count. So, it can relate similar words into certain categories. The Tokenization process in text analytics converts each single word of the vocabulary of your corpus into a unique integer, which will the unique identifier that represents that word.

After tokenizing the dataset, there are 33503 unique tokens i.e, unique words.

In [19]:

```
MAX_SEQUENCE_LENGTH = 100
MAX_NB_WORDS = 35000

tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(news_data_frame["terms"])

word_index = tokenizer.word_index
print('\nFound %s unique tokens.\n' % len(word_index))

sequences = tokenizer.texts_to_sequences( terms for terms in news_data_frame["terms"])
```

```
Found 33502 unique tokens.
```

In [20]:

```
# example of a sequence
print("sequence 0:")
print (sequences[0][:])
# length of sequence
print ("Total words in sequence 0: " + str(len(sequences[0][:])) + "\n")

print("sequence 1:")
print (sequences[1][:])
# length of sequence
print ("Total words in sequence 1: " + str(len(sequences[1][:])))
```

```
sequence 0:
[2521, 944, 797, 7195, 131, 312, 235, 614, 347, 6259, 20188, 525, 3307, 7
9, 862, 12642, 944, 4]
Total words in sequence 0: 18

sequence 1:
[845, 753, 15375, 6693, 265, 677, 458, 5891, 20189, 20190, 3784, 3308, 46,
15376, 79, 2228]
Total words in sequence 1: 16
```

Convert Sequences of Tokens to Fixed Lengths

Each sequence of words has a different length. This represents a problem because Convolutional Neural Nets require the size of the input to be fixed. This is quite challenging because textual data does not have a predefined structure. To address this problem, we need to fix the length for each sequence (in this case we will fix it to 1000) and do padding in the data: we start to fill the sequences with zero's and we add the content's of the sequence until the maximum fixed length is reached.

In [21]:

```
# add zeros to the begining of the sequence until the maximum length is reached
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

# let's take a look at the first padded sequence:
print (data.shape)
print (data[0][:])
len(data)
```

```
(30000, 100)
[    0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0  2521   944
   797  7195   131   312   235   614   347  6259 20188   525  3307    79
   862 12642   944     4]
```

Out[21]:

```
30000
```

The classes which is the categories needs to be matched with the order of padded data. So, we are encoding the categories as per the sequence in the dataset by 0s and 1s.

In [22]:

```
labels = news_data_frame["category"].tolist()
```

In [23]:

```
# convert labels
enc = OneHotEncoder()
Y = enc.fit_transform(np.array(labels)[:, np.newaxis]).toarray()
Y.shape[1]
```

Out[23]:

```
20
```

Split Data into Training, Test and Validation Sets

Now that we took a look at our data and that we separated the data into a variable with the prediction, y, and another variable with the features, data, we need to split our data into two sets: a training set (used to estimate our model), and a test set (used to evaluate how good our model is).

In [24]:

```
# Split the data set into training, testing and validation sets
X_train, X_test, Y_train, Y_test = train_test_split(data, Y, test_size=0.3,random_state
= 500)
X_validation, X_test, Y_validation, Y_test = train_test_split(X_test, Y_test, test_size
=0.5, random_state=2)
```

In [21]:

```
print("training set dimensions:")
print(X_train.shape)

print("\nlabels training set dimensions:")
print(Y_train.shape)

print("\ntest set dimensions:")
print(X_test.shape)

print("\nlabels test set dimensions:")
print(Y_test.shape)
```

```
training set dimensions:
(21000, 100)

labels training set dimensions:
(21000, 20)

test set dimensions:
(4500, 100)

labels test set dimensions:
(4500, 20)
```

Analysis With Our Naive Bayes Classifier

To capture relationships in words in the news articles, that are very difficult to capture, the embedding layer is used. An embedding is a mapping of a discrete categorical variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

In other words, this layer can help us find the most similar words in our classification task. The Dropout layer has been added to the network to reduce the over fitting of training and test accuracy and the flatten layer is used to converting the multidimensional layers to one dimensional to the output. There 20 neurons as per the number of categories and the sigmoid is the activation function used because it exists between (0,1). Therefore, it is especially used in our model where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. Due to this the out of the predicted category can have certain level on confidence which is probability of predicting the news category will be 0 to 1. Using all these the model has been created and fitted without training and validation dataset.

In [22]:

```
# defining a very simple model
model = Sequential()

input_dim = len(word_index) + 1
model.add(Embedding(input_dim,100, input_length=MAX_SEQUENCE_LENGTH))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(Y.shape[1], activation='sigmoid'))

# compile the model
model.compile(optimizer='nadam', loss='binary_crossentropy', metrics=['accuracy'])

# summarize the model
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          3350300
_________________________________________________________________
dropout (Dropout)            (None, 100, 100)          0
_________________________________________________________________
flatten (Flatten)            (None, 10000)             0
_________________________________________________________________
dense (Dense)                (None, 20)                200020
=================================================================
Total params: 3,550,320
Trainable params: 3,550,320
Non-trainable params: 0
_________________________________________________________________
None

/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/training/tra
cking/data_structures.py:720: DeprecationWarning: Using or importing the A
BCs from 'collections' instead of from 'collections.abc' is deprecated sin
ce Python 3.3,and in 3.9 it will stop working
  if not isinstance(wrapped_dict, collections.Mapping):
```

In [23]:

```
# define the callebacks to take into consideration during training
# stop training when convergence is achieved after 10 iterations
early_stop = EarlyStopping(monitor='val_loss', patience=15, verbose=1, mode='min')
# save the model after every epoch
callbacks_list = [early_stop]

# fit the model
history = model.fit(X_train, Y_train, epochs=5, validation_data=(X_validation, Y_valida
tion), callbacks=callbacks_list, verbose=1)
```

```
Train on 21000 samples, validate on 4500 samples
Epoch 1/5
21000/21000 [==============================] - 20s 968us/sample - loss: 0.
1900 - accuracy: 0.9494 - val_loss: 0.1647 - val_accuracy: 0.9507
Epoch 2/5
21000/21000 [==============================] - 19s 914us/sample - loss: 0.
1309 - accuracy: 0.9557 - val_loss: 0.1289 - val_accuracy: 0.9564
Epoch 3/5
21000/21000 [==============================] - 18s 842us/sample - loss: 0.
0885 - accuracy: 0.9673 - val_loss: 0.1166 - val_accuracy: 0.9603
Epoch 4/5
21000/21000 [==============================] - 18s 840us/sample - loss: 0.
0612 - accuracy: 0.9779 - val_loss: 0.1146 - val_accuracy: 0.9613
Epoch 5/5
21000/21000 [==============================] - 18s 833us/sample - loss: 0.
0418 - accuracy: 0.9864 - val_loss: 0.1201 - val_accuracy: 0.9609
```

Evaluation Of Our Model's Performance

In [24]:

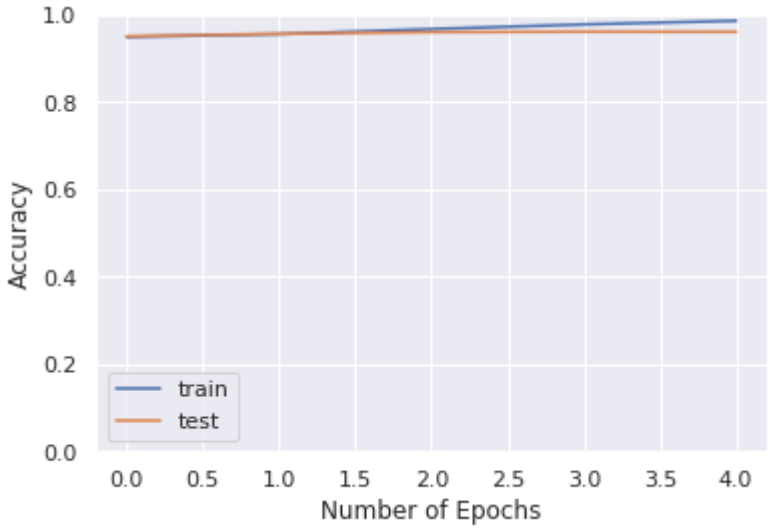
```
# evaluate loaded model on test and training data
train_loss,train_acc= model.evaluate(X_train, Y_train, verbose=1)
test_loss,test_acc = model.evaluate(X_test, Y_test, verbose=1)
print('[Accuracy] Train: %.3f, Test: %.3f' % (train_acc, test_acc))
print('[Loss] Train: %.3f, Test: %.3f' % (train_loss, test_loss))
```

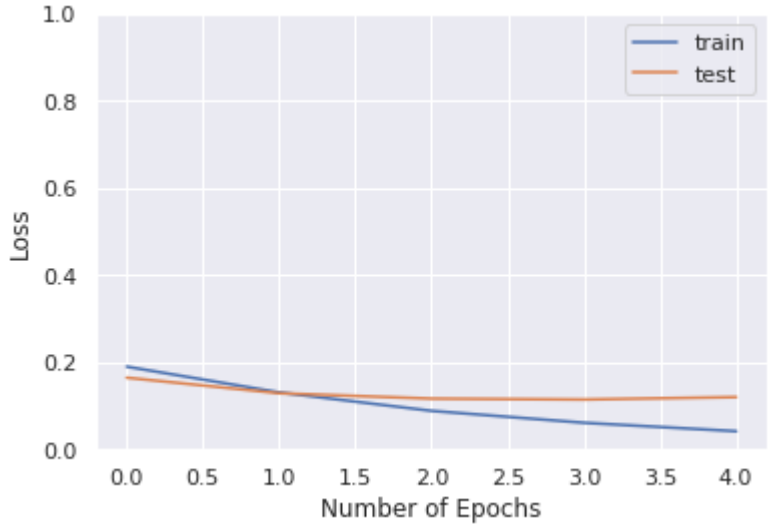
```
21000/21000 [==============================] - 1s 37us/sample - loss: 0.02
78 - accuracy: 0.9924
4500/4500 [==============================] - 0s 37us/sample - loss: 0.1182
- accuracy: 0.9612
[Accuracy] Train: 0.992, Test: 0.961
[Loss] Train: 0.028, Test: 0.118
```

In [25]:

```
# get model's training history
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy')
plt.xlabel('Number of Epochs')
plt.ylim([0, 1])
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss')
plt.xlabel('Number of Epochs')
plt.ylim([0, 1])
plt.legend()
plt.show()
```





The above graphs show an evaluation of the accuracy and loss of the created model. Following insights can be gained from the plotted graphs :

• The model starts with high accuracy in the 1st epochs, which is because of the removal of stopwords from the news articles. This shows our model can identity distinct patterns from the start of the iteration i.e, distinct words in each news category.

• The max accuracy achieved by our train and test dataset is around 99.2% and 96.1%. That is very high precision. There is a slightly over the training accuracy, but it is not a major issue.

• From the accuracy graph, we can say that our model is neither over fitting nor under fitting.

• From the loss graph, the training loss steadily decreases which showcase that our model using the dataset efficiently to classify.

• The variance between the training and testing from the graph shows how much the data has been differentiated by over model over the end of the epochs. Larger the variance larger overlapping of words which can be hard to predict by our classifier.

AI Explainability Assessment With LIME

Here, we are going to analyze the BlackBox of our model using the Lime package. This can be done using our developed code below. This gives a detailed analysis like what makes our model conclude to the particular category. This can be done by breaking done, how our algorithm functions. It takes each word and relates to a certain category. If the words are mapped to many categories, it makes use of the weightage of other words. So, from this code, we can visualize, how much each word contributes to a certain category, additionally how much confidence our model showcases our prediction. It also shows the top 4 category from the maximum level of evidence of top 12 words it predicts.

In [26]:

```
# We'll initiate our explainer here, as we can parlay its functionality with the predic
tion process easier
explainer = LimeTextExplainer(class_names=categories)
class_colors = ['#d70000', '#008700', '#d7af00', '#005fff']


def draw_ev_chart(evidence, col, title_text):
    plt.figure(1, figsize=(10, len(evidence)/3.5))
    plt.barh(np.array(range(0,len(evidence)))+0.5,[t[1] for t in evidence], align='cent
er',height=0.5, color=col)    # notice the 'height' argument
    plt.yticks(np.array(range(0,len(evidence)))+0.5, [t[0] for t in evidence], weight=
"normal", size="10")
    plt.xticks(weight='bold', size='8')
    plt.gca().axvline(0,color='k',lw=3)
    plt.title(title_text, weight="normal", size="10")
    plt.grid(False)
    plt.show()

def print_colorised(input_text, arg_categories, weightings, exp):
    print("\n\n\n")


    plt.figure(1, figsize=(4, len(weightings)/2))
    plt.barh(np.array(range(0,len(weightings)))+0.5, weightings, align='center',height=
0.5, color=class_colors)    # notice the 'height' argument
    plt.yticks(np.array(range(0,len(weightings)))+0.5, categories, weight="normal", siz
e="10")
    plt.xticks(weight='bold', size='8')
    plt.gca().axvline(0,color='k',lw=3)   # poor man's zero level
    plt.title("Class Weightings", weight="normal", size="10")
    plt.grid(False)
    plt.show()

    print("\n\n\n")

    ww = sorted(weightings, reverse = True)
    index_orders = []
    while(len(ww) > 2):
        index_to_drop = ww.index(max(ww))
        index_orders.append(index_to_drop)
        del ww[index_to_drop]
    index_orders.extend([1,0])

    word_class_list = []
    for l in range(0, len(arg_categories)):
        word_class_list.append([w[0] for w in exp.as_list(label=l)])
    reset_color_black = '\033[0m'
    constructed_replace_list = []
    for cl_v in range(0, len(word_class_list)):
        cl = index_orders[cl_v]
        for w in range(0, len(word_class_list[cl])):
            constructed_replace_list.append([len(word_class_list[cl][w]), word_class_li
st[cl][w], "\u001b[%sm" % (41+cl)])
    constructed_replace_list.sort(key=lambda x: x[0])
    output_text = input_text
    pattern_around = "(?<![a-zA-Z])"
    pattern_after = "(?![a-zA-Z])"
    for w in constructed_replace_list:
        output_text = re.compile(pattern_around+w[1]+pattern_after).sub( w[2]+w[1]+rese
```

```
t_color_black, output_text)
    print(output_text)

def c_fn(text):
    s = tokenizer.texts_to_sequences(text)
    d = pad_sequences(s, maxlen=MAX_SEQUENCE_LENGTH)
    return model.predict(d)

def run_explainability_test(input_sentence):
    # tokenize
    sequences = tokenizer.texts_to_sequences(input_sentence)
    # make padding
    data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    exp = explainer.explain_instance(input_sentence, c_fn, num_features=12, num_samples
=len(data), labels=list(range(20)))
    # get prediction
    weightings = [sum([x[1] for x in exp.as_list(label=y)]) for y in range(0,len(catego
ries))]
    print( "\n\n\nPredicted values: " + categories[weightings.index(max(weightings))])
    sorted_weightings = sorted(weightings, reverse=False)
    sorted_categories = [categories for _,categories in sorted(zip(weightings,categorie
s))]
    print_colorised(input_sentence, categories, weightings, exp)

    sorted_exp = sorted(range(len(weightings)), key=lambda i: weightings[i])[-4:]
    sorted_exp = list(reversed(sorted_exp) )
    for value, class_n in enumerate(sorted_exp):
        draw_ev_chart(exp.as_list(label=class_n),class_colors[value],"Evidence Of %s" %
(categories[class_n]))

def predict(input_sentence):
    exp = model.predict(input_sentence)
    weightings = [sum([x[1] for x in exp.as_list(label=y)]) for y in range(0,len(catego
ries))]
    result = categories[weightings.index(max(weightings))]
    return result
```

To form the confusion matrix, to identify how our model predicts by comparing true labels and its predicted labels, we are decoding the encoded test data labels for comparision in the same order as the prediction labels. And also we are reshaping the test features to initiate the prediction and collected all the results in the order similar to the true test labels.

In [27]:

```
x_train = np.reshape(X_train, (len(X_train), 1, X_train.shape[1]))
x_test = np.reshape(X_test, (len(X_test), 1, X_test.shape[1]))

results = []
for index in range(0,len(x_test)):
    weightings = model.predict(x_test[index])
    r = categories[(weightings[0].tolist()).index(max(max(weightings)))]
    results.append(r)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/engin
e/training_v2_utils.py:544: DeprecationWarning: Using or importing the ABC
s from 'collections' instead of from 'collections.abc' is deprecated since
Python 3.3,and in 3.9 it will stop working
  if isinstance(inputs, collections.Sequence):
```

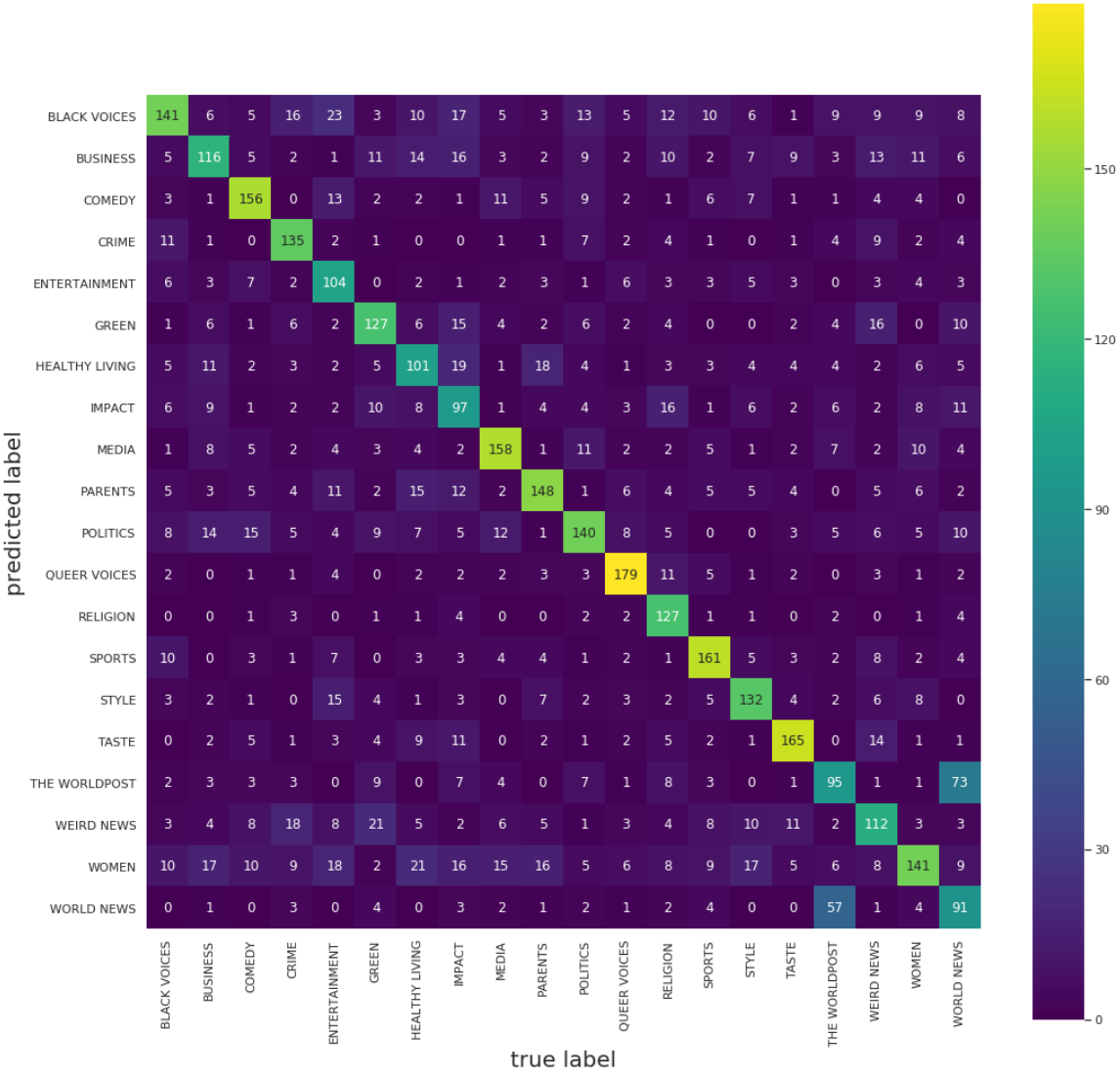
In [28]:

```
def decode(datum):
    return np.argmax(datum)

Y_test_decoded = []
for i in range(0,len(Y_test)):
    datum = Y_test[i]
    decoded_datum = decode(Y_test[i])
    Y_test_decoded.append(categories[decoded_datum])
```

In [30]:

```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(Y_test_decoded, results)
plt.figure(figsize= (17,17))
#sns.heatmap(mat.T, square=True, fmt='d', cbar=True, xticklabels=categories,
            #yticklabels=categories, annot=True)
sns.heatmap(mat.T, square=True, cbar=True, xticklabels=categories,
            yticklabels=categories, annot=True, cmap=cm.viridis, fmt= 'g')
plt.xlabel('true label',fontsize= 20)
plt.ylabel('predicted label',fontsize= 20);
```


predicted label
true label
BLACK VOICES
BUSINESS
COMEDY
CRIME
ENTERTAINMENT
GREEN
HEALTHY LIVING
IMPACT
MEDIA
PARENTS
POLITICS
QUEER VOICES
RELIGION
SPORTS
STYLE
TASTE
THE WORLDPOST
WEIRD NEWS
WOMEN
WORLD NEWS
150
120
90
60
30
0

Insights from the confusion matrix are as follows:

• The diagonal of different color depicts that there is more number of model prediction are accurate compared with the true labels.

• Comparing this model matrix with the previous matrix we have tested, there is almost the same number of distribution of news articles in our dataset. Unlike the previous confusion matrix, this model has been tested with all categories of almost the same number of data

• From this matrix, we can say that our model can predict all categories with some level of confidence rather than predicting with high confidence only on certain news categories.

• There are also some overlapping prediction of topics like “The world post” and “World news”. It is understandable that the articles from both of these topics we similar words around the world.

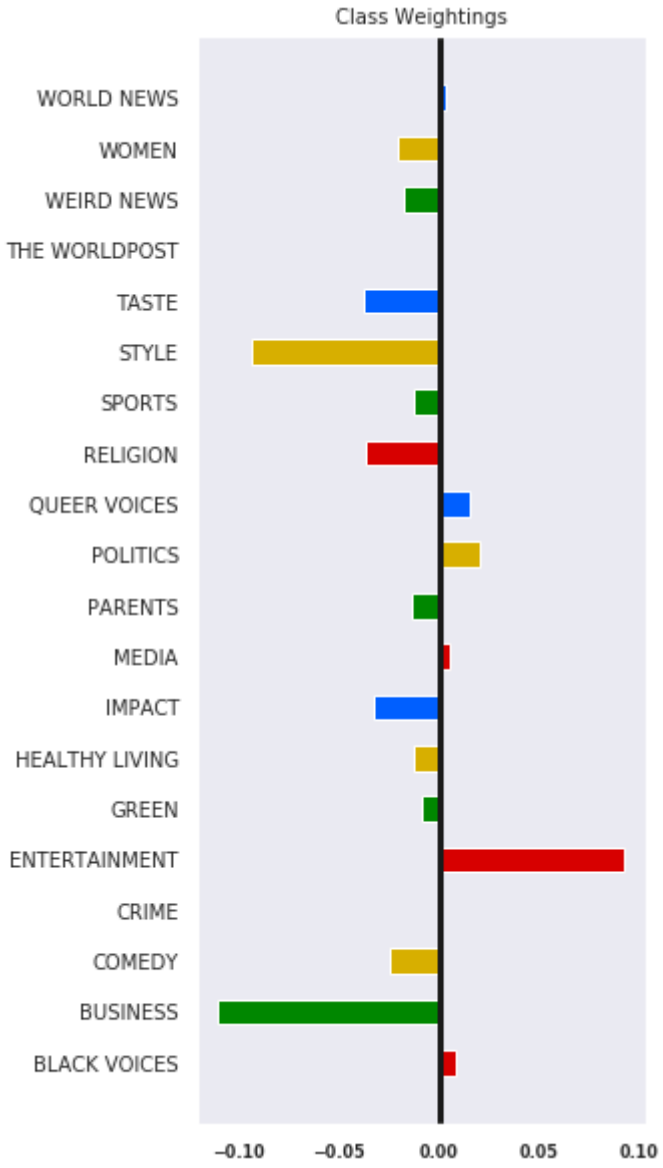
• The category “Queer Voices” has a high number of correct predictions, it shows that the words present in the Queer Voices articles are more distinct than the other articles in the dataset.

Now, lets analyse the type of content of how the classifier works with the given dataset. So the given sentence is about SPORTS and we are trying to predict the news category using our created model. Using our created visualization code we can able to see which category contain how much confidence with respect to the given data. And also, we have developed for getting insights from the words related to which category. A word can be related to many categories but the level of confidence varies between the categories and from our analysis we can see what are words contributing for high level of confidence in the predicted category. Then we can analyse that the prediction of distinct word can contribute to some relation to the classified category.

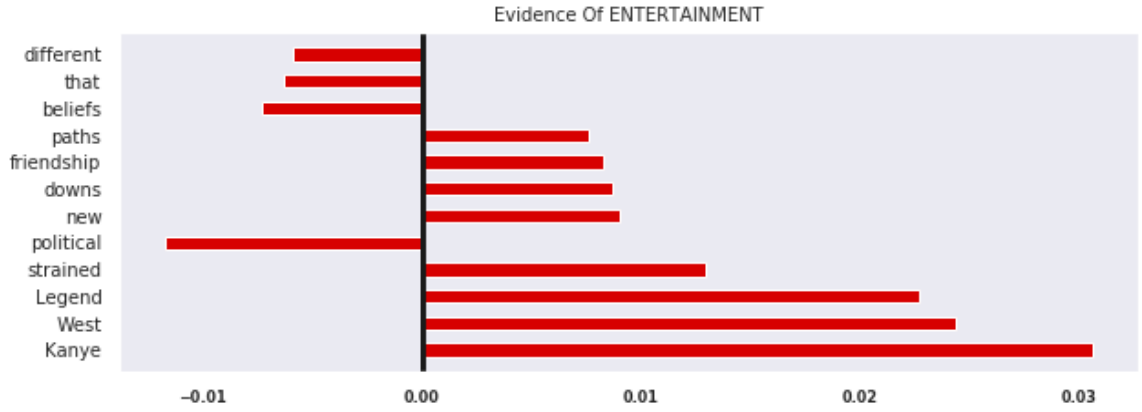
In [34]:

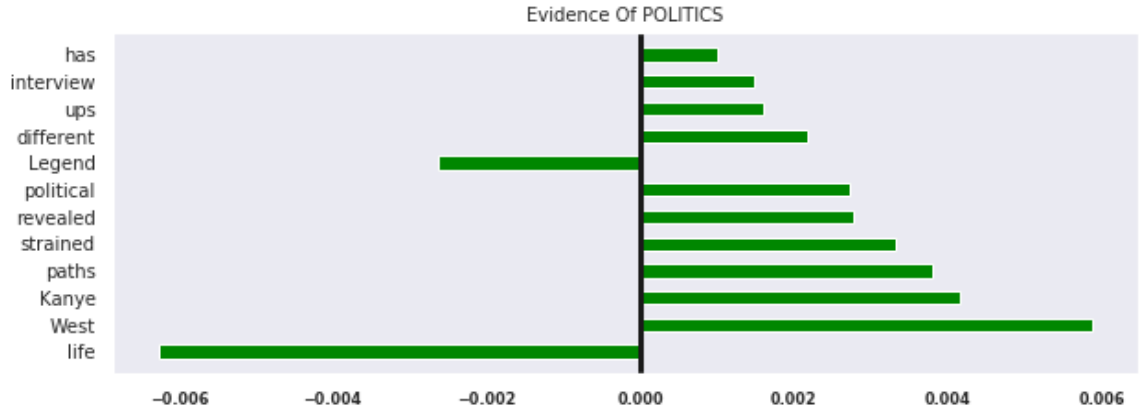
```
# A sentence about sports
input_sentence = """After many ups and downs, Legend has revealed in a new interview th
at his friendship with Kanye West wasn’t strained by their different political beliefs,
but rather with their life paths."""
run_explainability_test(input_sentence)
```

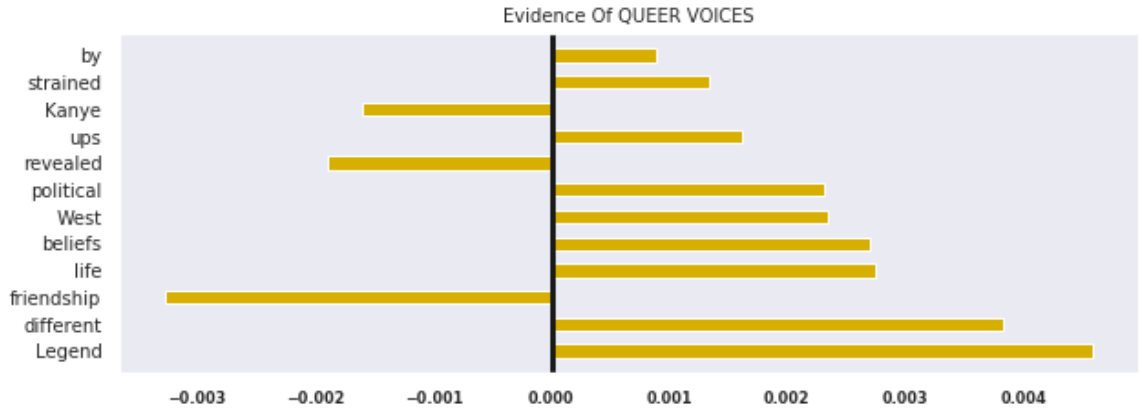
```
Predicted values: ENTERTAINMENT
```

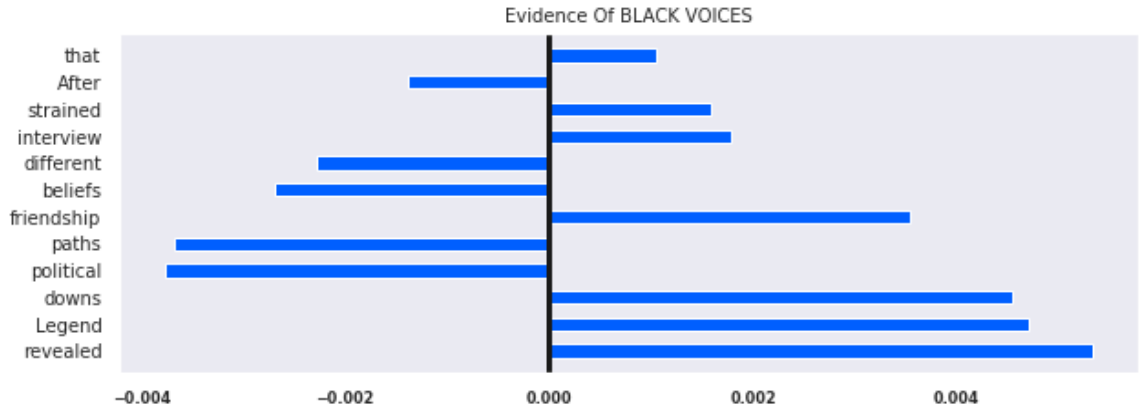


```
After many ups and downs, Legend has revealed in a new interview that his
friendship with Kanye West wasn’t strained by their different political be
liefs, but rather with their life paths.
```







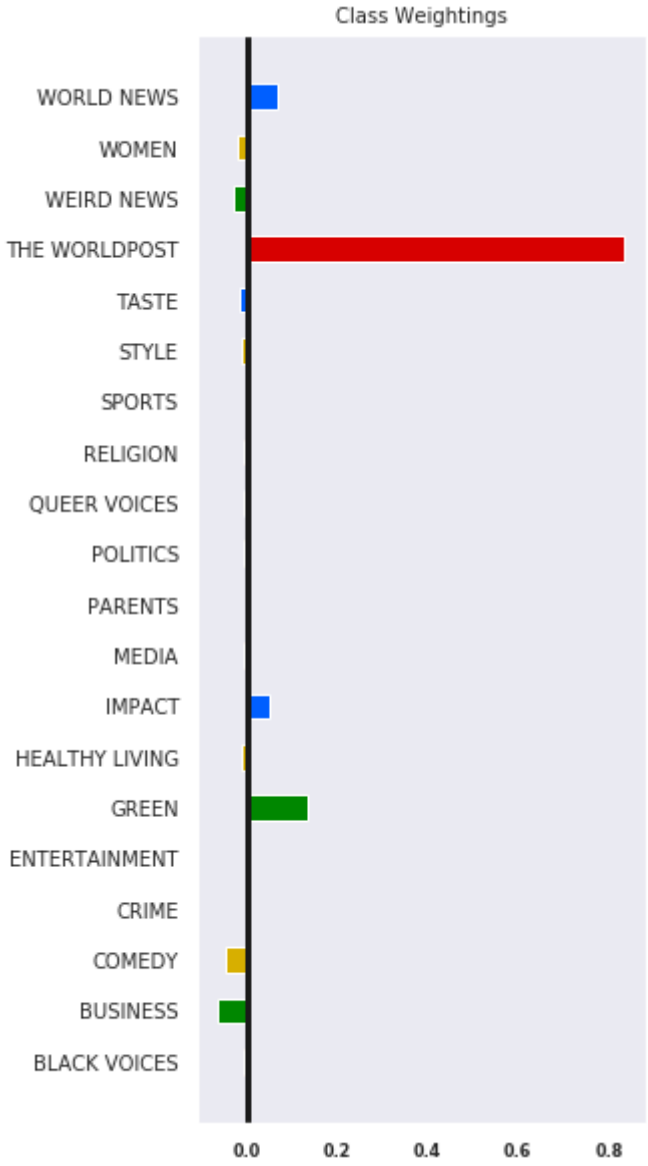


Our model predicted the given article in the correct category i.e, sports. The classifier shows a clear cut of evidence about sports compared to other categories. The level of confidence is also high in the article. The confidence level is the sum of all the word's probability of being the certain category. So, here the sum of all the words contributed to the sports category leads to the predicted category. The words like “final”, “credited”, “coach” has a higher probability of occurring in the sports category which is reasonable.

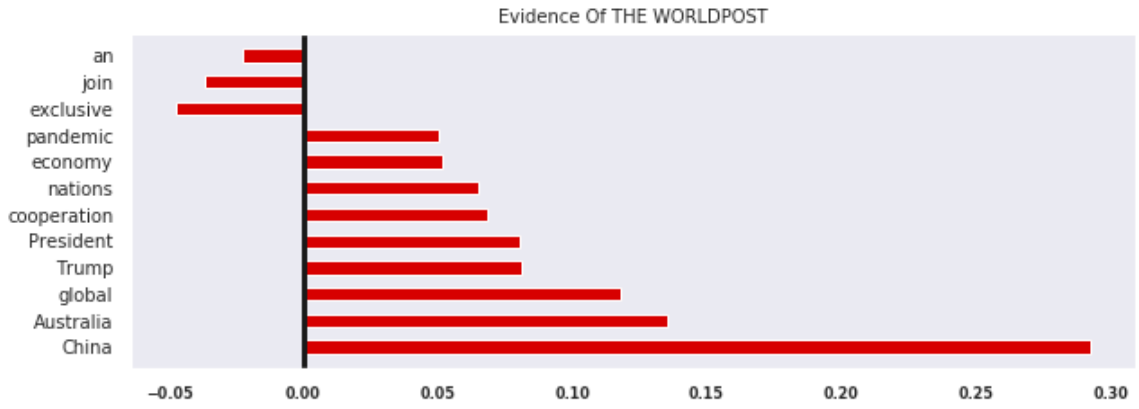
In [112]:

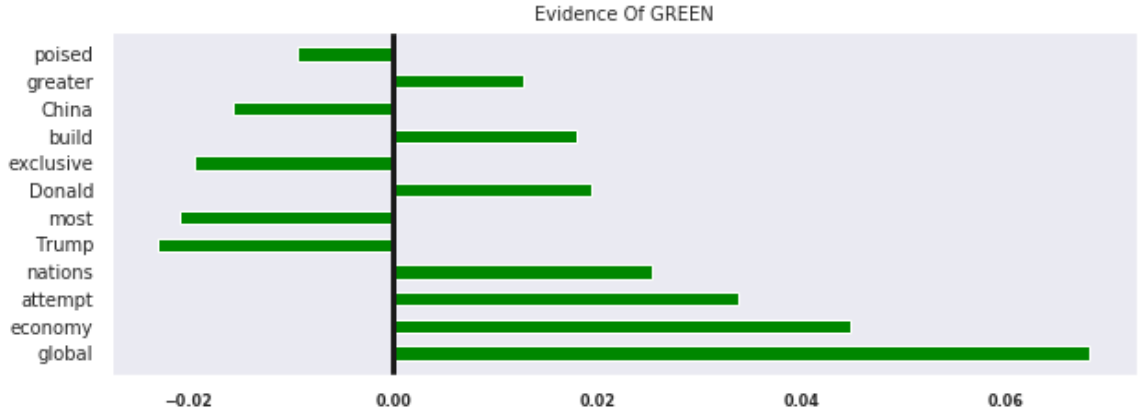
```
# 2nd sentence about WorldPost
input_sentence = """Australia is poised to join the world's most exclusive political or
ganisation after US President Donald Trump called for an expansion of the Group of 7 na
tions without China in an attempt to build greater cooperation over restoring the globa
l economy following the coronavirus pandemic."""
run_explainability_test(input_sentence)
```

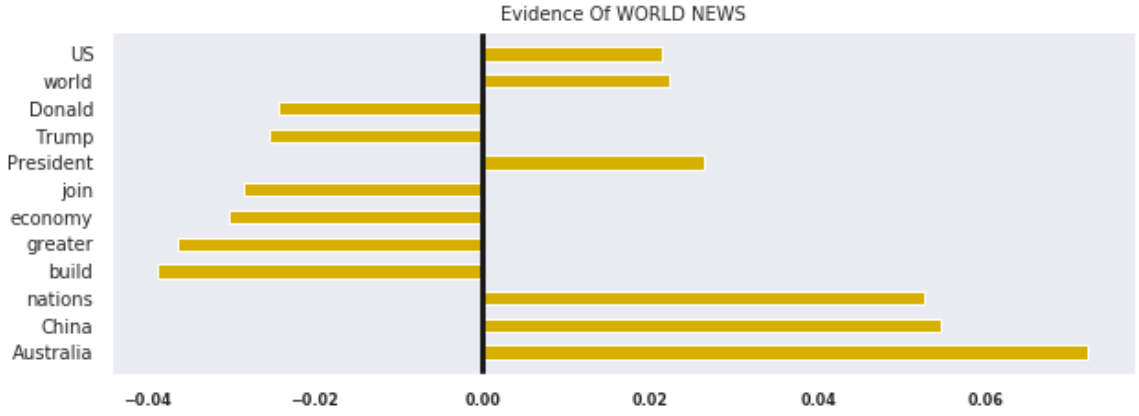
```
Predicted values: THE WORLDPOST
```

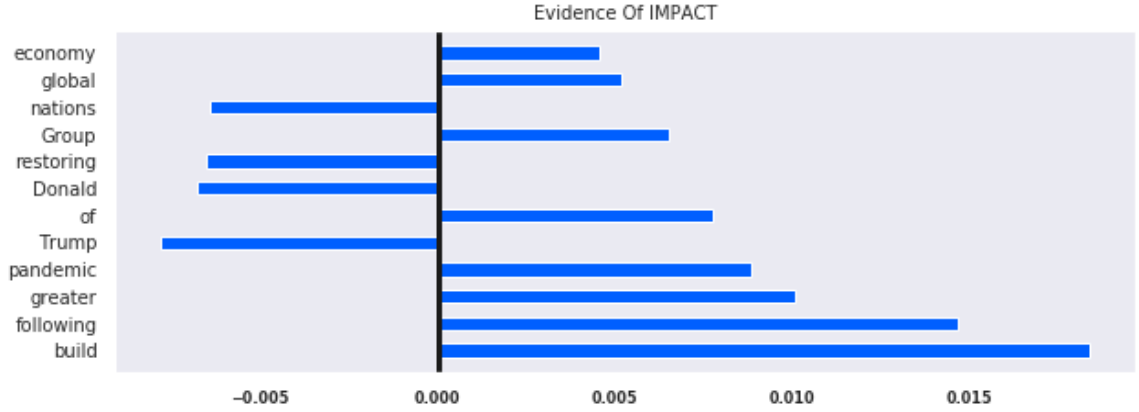

Class Weightings
WORLD NEWS
WOMEN
WEIRD NEWS
THE WORLDPOST
TASTE
STYLE
SPORTS
RELIGION
QUEER VOICES
POLITICS
PARENTS
MEDIA
IMPACT
HEALTHY LIVING
GREEN
ENTERTAINMENT
CRIME
COMEDY
BUSINESS
BLACK VOICES
0.0
0.2
0.4
0.6
0.8

```
Australia is poised to join the world's most exclusive political organisat
ion after US President Donald Trump called for an expansion of the Group o
f 7 nations without China in an attempt to build greater cooperation over
restoring the global economy following the coronavirus pandemic.
```







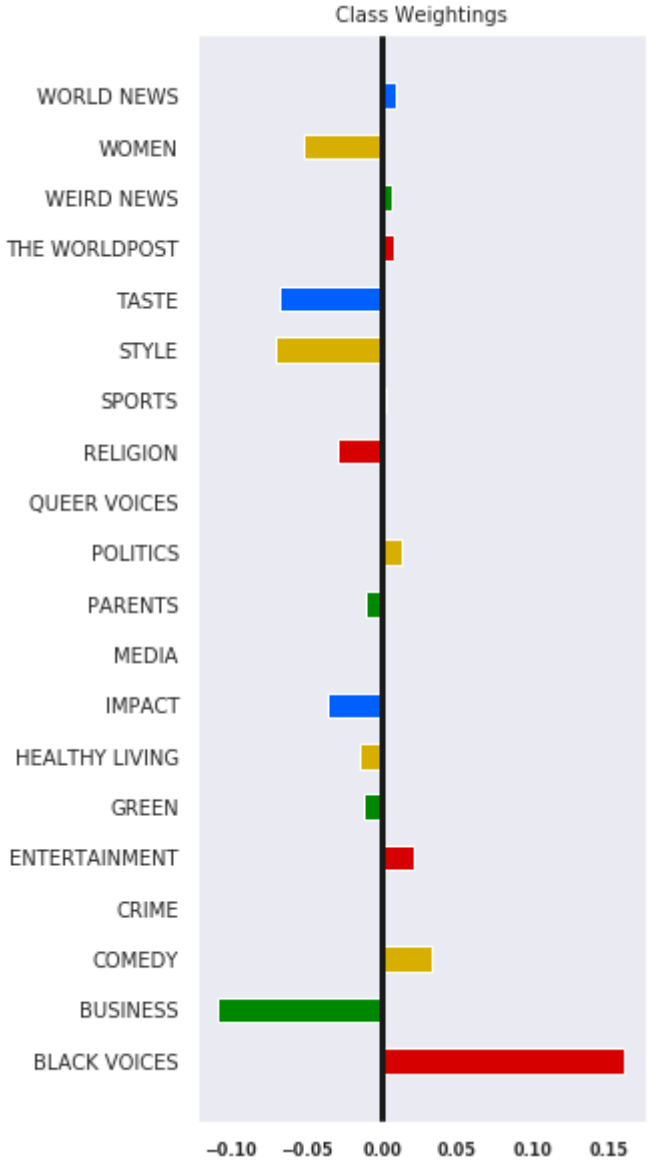


The sentance was about world post and the model predicts the category world post with good classification of words like country names “china”, “australia”, “global”, “nations” etc. Also it has very high amount of evidence with the number of words associated to the given article.

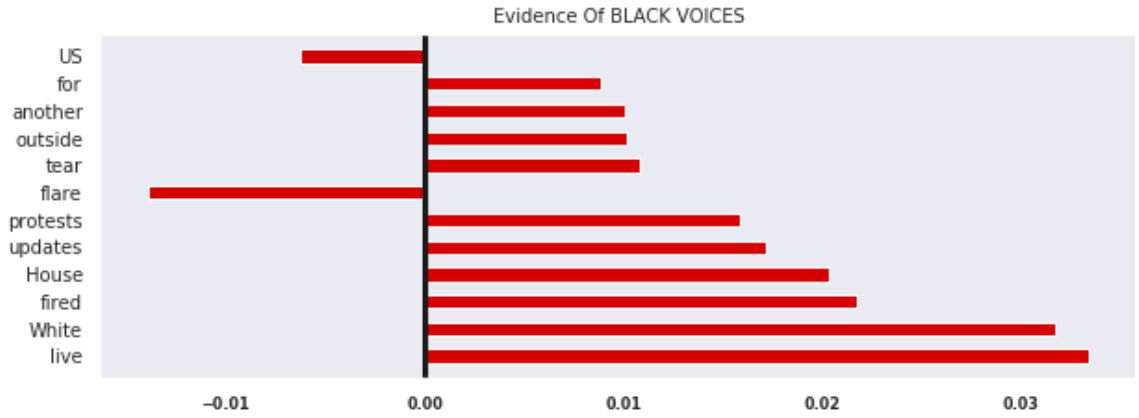
In [99]:

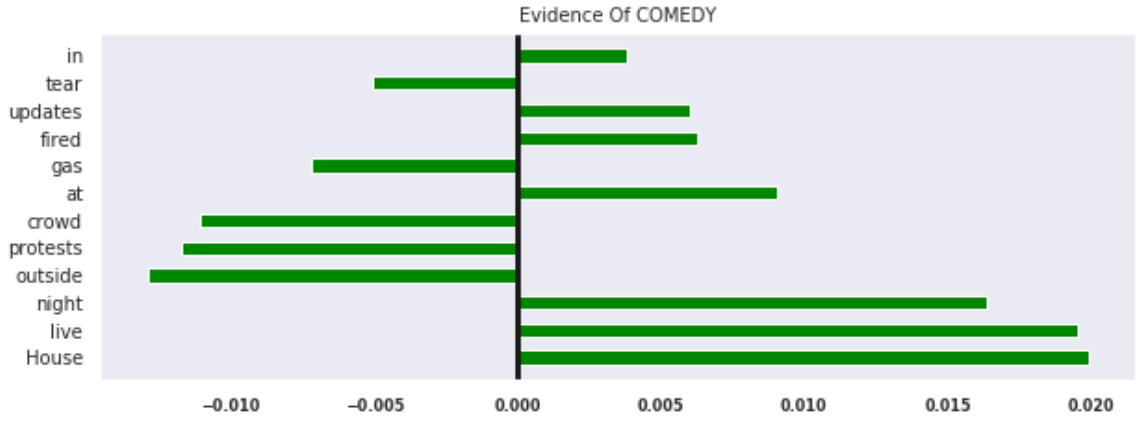
```
# 3nd sentence about Crime
input_sentence = """protests flare in the US for another night, tear gas fired at crowd
outside the White House, live updates"""
run_explainability_test(input_sentence)
```

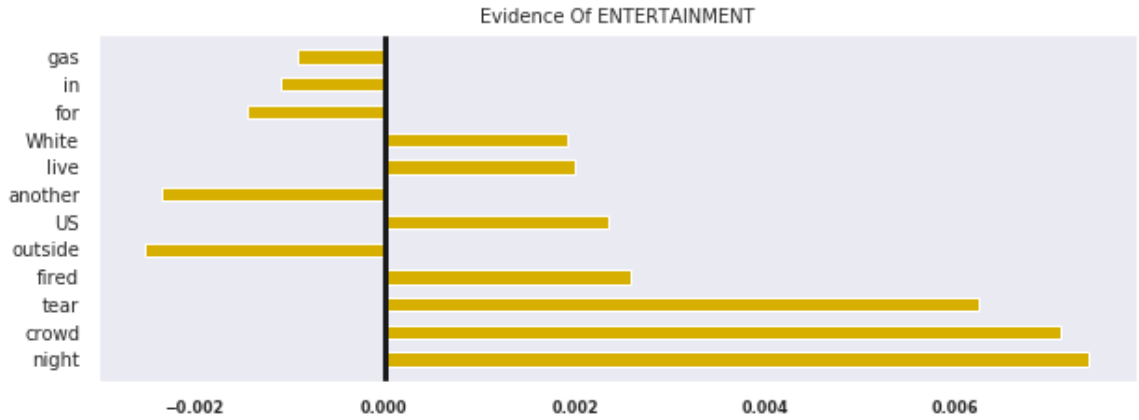
```
Predicted values: BLACK VOICES
```

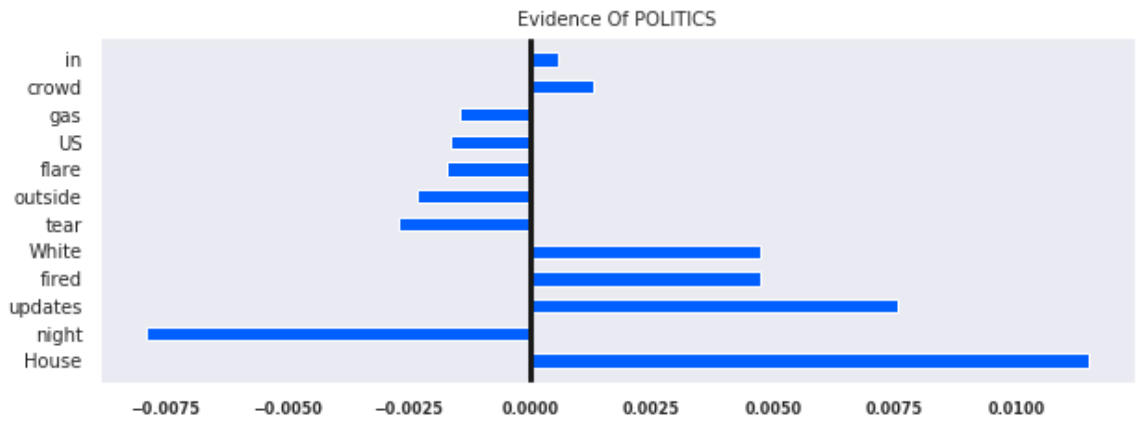



```
protests flare in the US for another night, tear gas fired at crowd outsid
e the White House, live updates
```


Evidence Of BLACK VOICES
US
for
another
outside
tear
flare
protests
updates
House
fired
White
live
−0.01
0.00
0.01
0.02
0.03


Evidence Of COMEDY
in
tear
updates
fired
gas
at
crowd
protests
outside
night
live
House
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020


Evidence Of ENTERTAINMENT
gas
in
for
White
live
another
US
outside
fired
tear
crowd
night
−0.002
0.000
0.002
0.004
0.006


Evidence Of POLITICS
in
crowd
gas
US
flare
outside
tear
White
fired
updates
night
House
−0.0075
−0.0050
−0.0025
0.0000
0.0025
0.0050
0.0075
0.0100

From the above example, it was recent news about the protest happening in the US. The true category of the article is “Crime”, but our predicted category was “Black voices”. The following are the insights gained from our the above input in the analysis of this model:

• No words are logically meant to be in the category “Black Voices”.

• The level of confidence is not that high, but it has nothing to do with the predicted category. Anyhow, in the real implecation, the model gives an output as Black voices, which is wrong.

• The words which are most contributed are “fired”, “live”, “White”, “protests” and “tear”. These words are not linked to the “Black voices” but our model shows these are related to this category with some level of evidence.

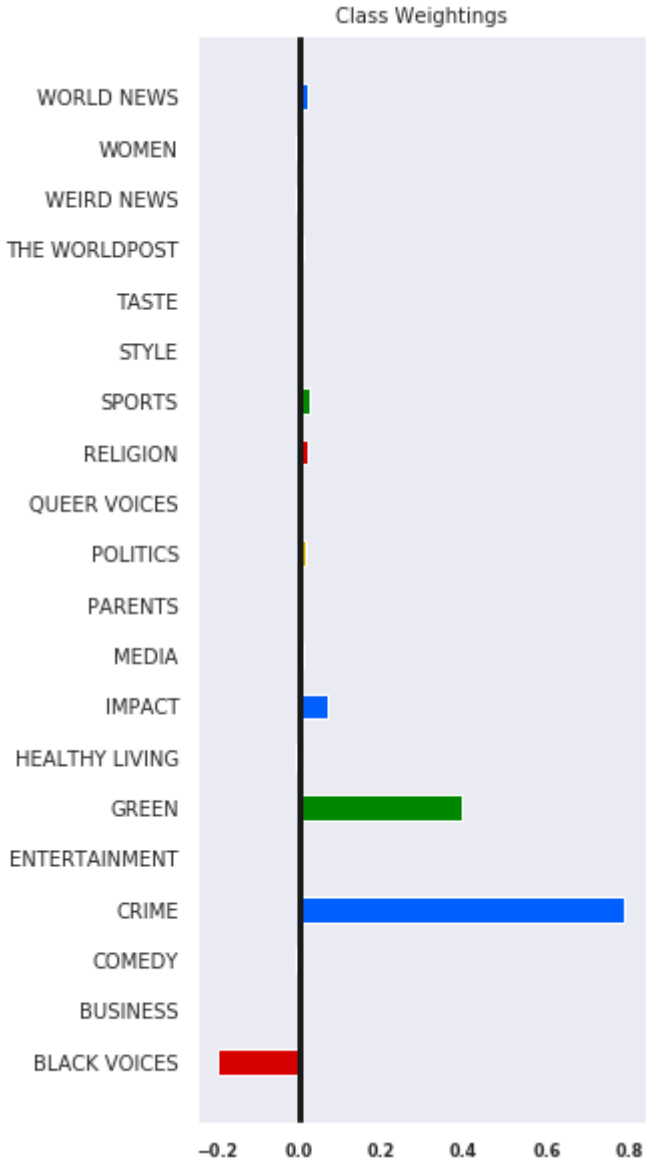
• These words are logically linked to the crime category. But our model classifies incorrectly.

• Let's also consider the number of words. This input article is just a headline without a detailed description compared to our previous category.

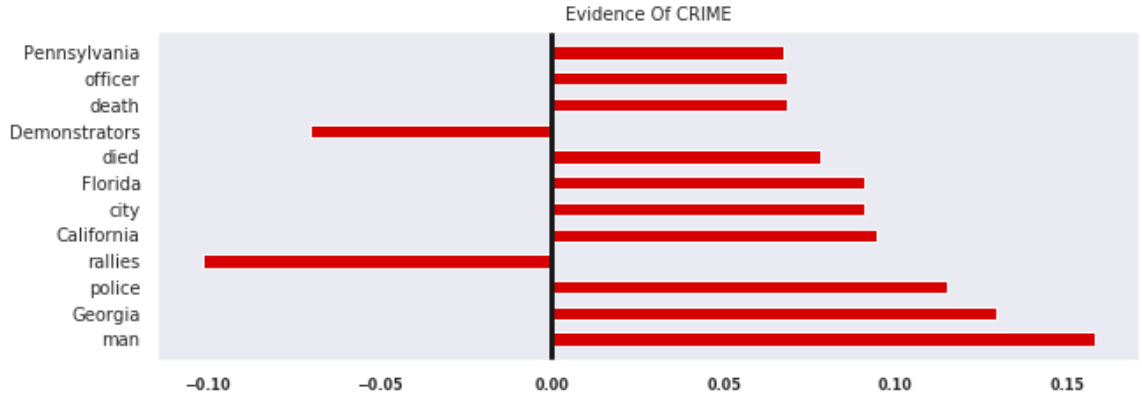
In [92]:

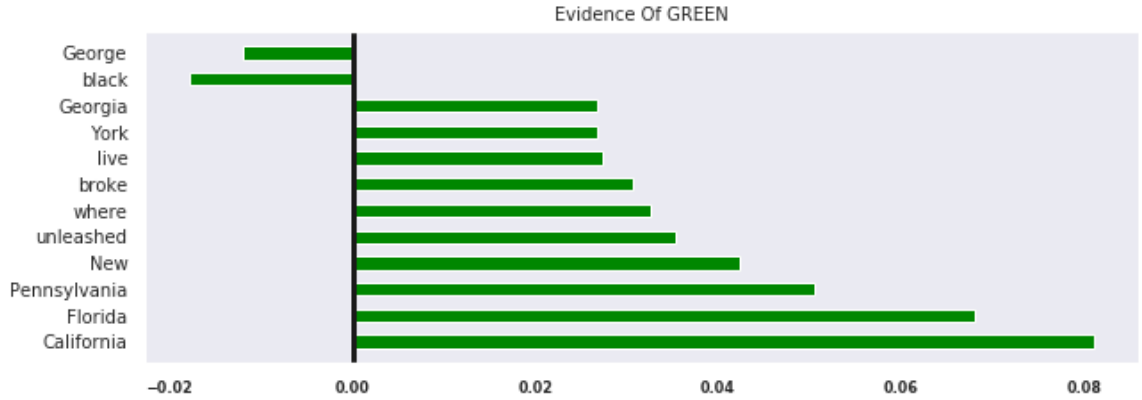
```
# 4th - same as above sentence with detailed description
input_sentence = """protests flare in the US for another night, tear gas fired at crowd
outside the White House, live updates!The death this week of an unarmed black man at th
e hands of a Minnesota police officer has unleashed a wave of protests across the Unite
d States.
Demonstrators first turned out in the city of Minneapolis where George Floyd died, but
 rallies quickly broke out in Georgia, California, New York, Kentucky, Florida and Penn
sylvania."""
run_explainability_test(input_sentence)
```

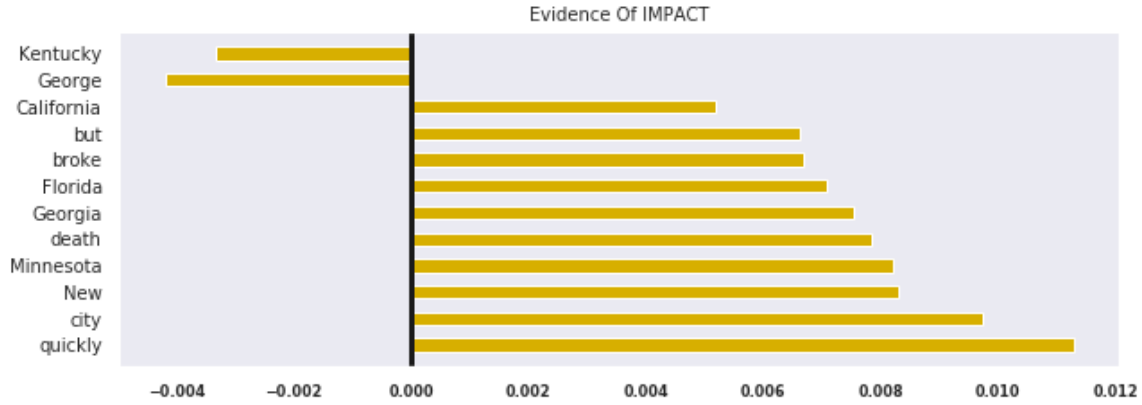
```
Predicted values: CRIME
```

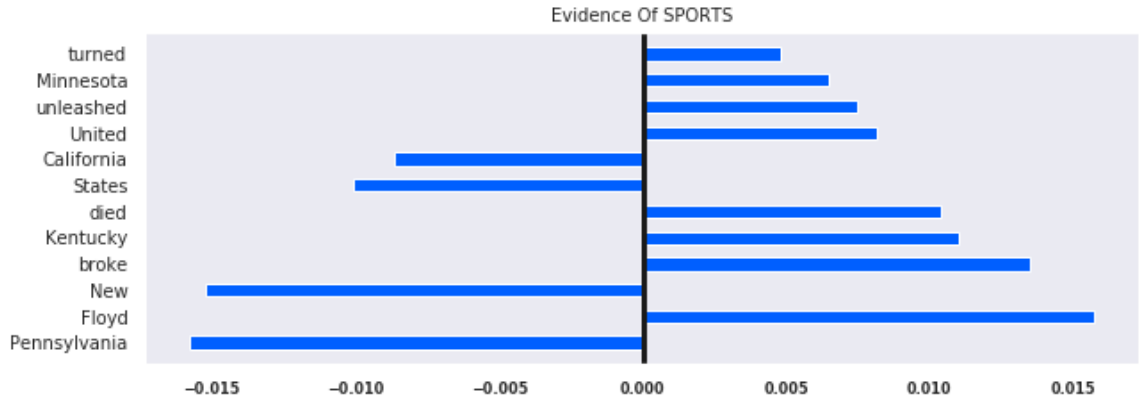

Class Weightings
WORLD NEWS
WOMEN
WEIRD NEWS
THE WORLDPOST
TASTE
STYLE
SPORTS
RELIGION
QUEER VOICES
POLITICS
PARENTS
MEDIA
IMPACT
HEALTHY LIVING
GREEN
ENTERTAINMENT
CRIME
COMEDY
BUSINESS
BLACK VOICES
−0.2
0.0
0.2
0.4
0.6
0.8

```
protests flare in the US for another night, tear gas fired at crowd outsid
e the White House, live updates!The death this week of an unarmed black ma
n at the hands of a Minnesota police officer has unleashed a wave of prote
sts across the United States.
Demonstrators first turned out in the city of Minneapolis where George Flo
yd died, but rallies quickly broke out in Georgia, California, New York, K
entucky, Florida and Pennsylvania.
```









This was the same article with detailed description. But the number of words in higher than the previous input. Following are the insights collected from the analysis:

• It shows a significant confidence level that the given article is about crime.

• The context of the words mapped to the crime are logically correct and it is reasonable, unlike our previous insight.

• The words connected to crime are “police”, “man”, “died”, “death”, “officer” etc. These words show high level of probability of being in the crime category.

• Other categories also shows more number of logically related words to its categories.

• The data provided is more compared to our last input. This might have something to do the correct confidence level.

But why?

Let's analyse further what makes the model to predict the above story in the wrong classification. We can analyze using the number of words contributed in the Black Voice category.

In [21]:

```
BlackVoicesTerms = news_data_frame[news_data_frame['category'] == "BLACK VOICES"]["term
s"]

Black_voices_word_counts = {}

for terms in BlackVoicesTerms:
    for word in terms:
        if word in Black_voices_word_counts:
            Black_voices_word_counts[word] += 1
        else:
            Black_voices_word_counts[word] = 1

# sort the word_counts by counts
Black_voices_sorted_counts = {k: v for k, v in sorted(Black_voices_word_counts.items(),
key=lambda item: item[1],reverse=True)[:20]}

Black_voices_sorted_counts
```

Out[21]:

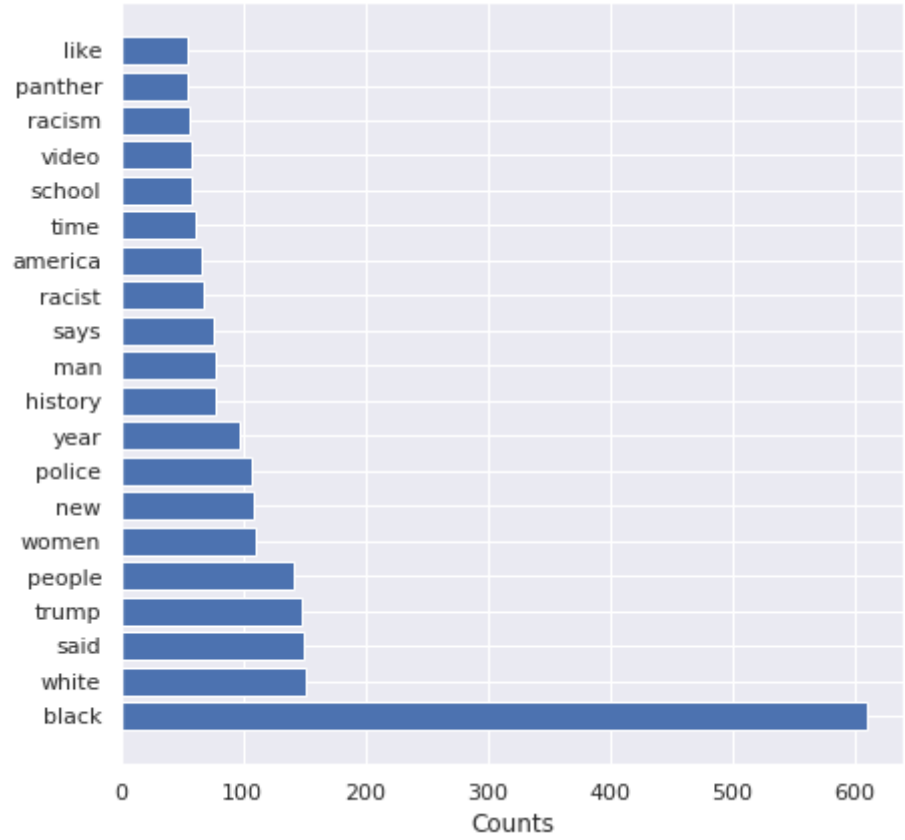
```
{'black': 610,
 'white': 152,
 'said': 149,
 'trump': 148,
 'people': 141,
 'women': 111,
 'new': 109,
 'police': 107,
 'year': 97,
 'history': 78,
 'man': 77,
 'says': 76,
 'racist': 68,
 'america': 66,
 'time': 61,
 'school': 58,
 'video': 58,
 'racism': 56,
 'panther': 55,
 'like': 55}
```

In [64]:

```
plt.figure(figsize=(7,7))
plt.barh(list(Black_voices_sorted_counts.keys()), list(Black_voices_sorted_counts.values()))
plt.xlabel(xlabel="Counts")
```

Out[64]:

```
Text(0.5, 0, 'Counts')
```



From the above graph, it shows the words which occurs most in the Black voice category while creating the model. These words are from our dataset provided at the starting of the development of the model. Words like “white”, “Police” should not be a part of this category logically, which makes the dataset one-sided. We can see that our dataset provided which has more contexts like protesting, police, white, fired etc., Our model couldn’t find enough evidence of other categories because of fewer data. That’s made our model predict it in the wrong category but with less confidence.

Combining all the insights and mapping it to business concern:

• The model showed very high accuracy and loss using the training and test datasets for evaluation. This shows the model can classify and identify distinct words from the articles for each category.

• Looking at the variance of the loss between training and testing dataset. They are very low, which depicts that the words used for training and testing data are not overlapping. The model distinguishes the words precisely and relates to certain categories.

• From the confusion matrix, we can come to the result that the model can classify all categories instead of working better on certain categories.

• From the explainability, most the time, 3 of 4 analysis, the model predicts categories and classify at a higher rate of evidnce. These words are also making logical connections to the topic of prediction.

• But when the number of data given to the model is lower, i.e, inputing with less number of words. The model was not functioning quite well compared to a large number of data. It classifies words into wrong categories and the words are not logically making any connection to the topic of prediction. Also, the level of confidence is very low for the whole category.

Potential Issues

If this model has been deployed in the news agencies, news syndicates website, mobile application, etc., it would classify the news article most of the time to its true labels. But, if the provided data is insufficient the model will predict given text in the wrong category. Example, what we have seen in our analysis before like the text is nothing to do with the Black Voice category but it matches words like “fired”, “tear”, “white” and “protests”. The level of confidence may very low but, at this point, it classifies into the wrong category. This can be avoided if we implement certain conditions like the prediction should showcase certain evidence before deciding on the category. This may affect the perspective of the readers about the news and also the important stakeholders involved in the article. Also, the agencies can loss important values like customers, credits etc.

Conclusion

To conclude, the news agencies can use this model for classifying the news category. But, they should be taking into account the amount of data they provide from the article to our model. More data reduces the risks of predicting it in the wrong category. From the explainability of our model, to avoid these risk of minimal data in the future, news agencies can use the more filtered dataset with words logically and meaningfully connecting to the category. Also, as mention above, we can improve the algorithm by using more condition on finalizing the category with a certain level of confidence. The interesting note to make is that, from our analysis, we can also see that news articles are categorized in a bit biased way like protests, arrests, crime words in the article to Black Voices. These words are more to do with the Crime category rather than Black Voices. These should be taken into account by the news agencies in the future.

In [ ]: