Machine Learning Capstone Project

Definition

Project Overview

The project centers around a very popular Natural language processing problem regarding fake and true news detection. I will be using a specific data-set that will allow me to create algorithm able to determine if an article is fake news or true.

Problem Statement

The problem is how we can determine if a news article is fake or true and determine this with consistently high accuracy. Obviously the goal would be to construct an algorithm that would provide us with viable results for the problem. There are several ways we can achieve this ;

➔ We can use Term frequency-Inverse document frequency to train the model. There might be some terms that occur frequently across all documents and these may tend to overshadow other terms in the feature set. This will be particularly effective for words that don't occur as frequently but might be more interesting and effective as features to identify specific categories.

➔ Another more common method is to use pre-trained word embeddings. Instead of converting every word in a number, words are converted to a tensor representation. So when a 4-dimensional embedding is used, every (unique) word is converted to a combination of four numbers. After the words are converted into word embeddings, the words are fed into a neural network.

Metrics

To evaluate the performance of each model, we used accuracy score metric, where TP= True Positives ; TN= True Negatives ; FP= False Positives ; FN= False Negatives.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}.$$

I felt confident with this metric due to the fact that the two datasets are pretty balanced.

Analysis

Data Exploration

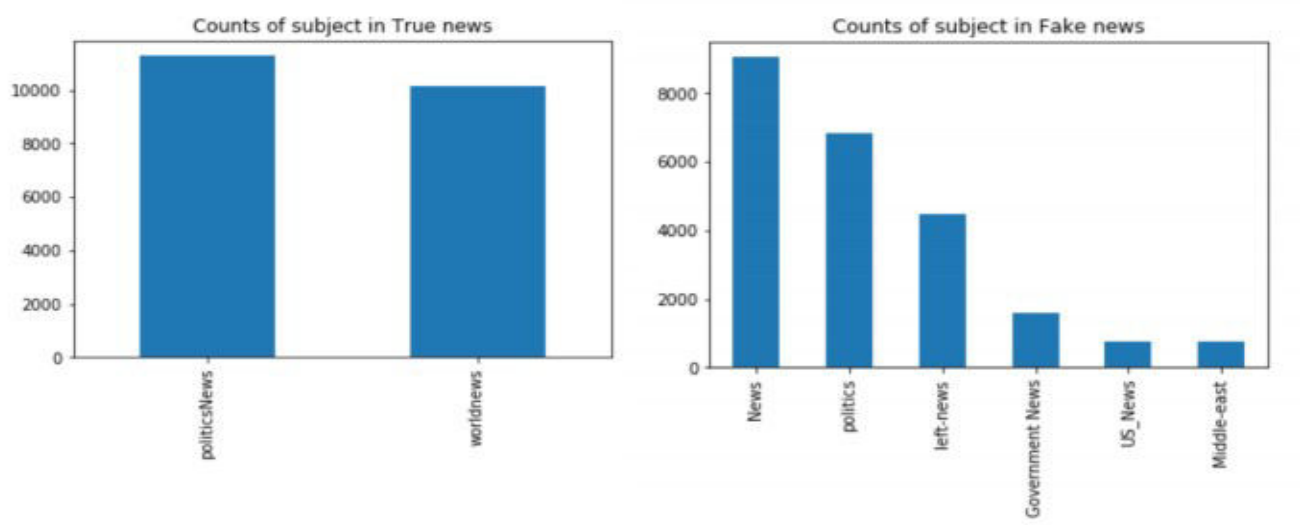
Using Pandas.read_csv(), I was able to import the True and Fake datasets and further examine the features : title, text, subject and date.

```
true.head()
```

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

Exploratory Visualization

In order to visualize the distribution of data, I made plots on 'subject' from both datasets. I noticed that in the true data-set, the subjects are almost evenly divided into 'politicsnews' and 'worldnews'. And in the fake data-set the subjects are spread across multiple classes.



One of the unusual things I discovered in the data is the datetime format. Pandas did not catch the datetime format in the first image, so I parsed it, which I will explain in the next steps.

Algorithms and Techniques

The classification technique I used was **Naive-Bayes** because of the basic idea of Naive-Bayes model is that all features are independent of each other, which would work well with text classification due to the fact that the presence of a particular feature in a class is unrelated to the presence of any other feature. The following classifier was used in estimating the target.

$$\hat{y} = argmax\, P(y) \prod_{i=1}^{n} P(x_i|y)$$

This link shows a thorough breakdown of the technique;
https://people.engr.tamu.edu/huangrh/Spring18/l3_textClassification_naivebayes.pdf

I also decided to go with a Biderectional LSTM Layer and a sigmoid output.. This will allow the LSTM to learn what comes next and the cause of it, because what comes before and after the text is important as well. This can provide additional context to the network and result in faster and even fuller learning on the problem.

Benchmark

As a Benchmark, I am targeting what the Authors of the articles listed in my proposal achieved using linear support vector machine (LSVM), which was an accuracy score of 92%.

Methodology

Data Pre-processing

The first Data pre-processing steps I took was to add a label to the datasets and link them to form a complete dataset, then rearrange the entire dataset to uncover any one sidedness.

```
true['targetClass'] = 0
fake['targetClass'] = 1
```

```
df = pd.concat([true,fake])
```

```
#Shuffle the dataframe to randomize things up
df=df.sample(frac=1, random_state=1).reset_index(drop=True)
```

The pre-processing steps taken ;

- ➔ Parse dates so that we can extract the features from the dates.
- ➔ Text cleaning: lowercase, filtering numbers, URLS and hashtags.
- ➔ Text tokenization and Tf-Idf vectorization.

I used the parse() function To parse the dates. After several value errors due to some rows having date fields containing texts or urls, the function created was able to return NaN if the dates could not be parsed.

The dates were correctly parsed using this;

```
df['date'] = df['date'].apply(parse_date)
```

```
df['date'].head()
```

```
0   2017-04-02
1   2017-07-26
2   2016-02-07
3   2017-11-30
4   2016-12-27
Name: date, dtype: datetime64[ns]
```

Subject categories were remapped to: politics and general.

```
#Create a single view of multiple mappings
from collections import ChainMap
to_politics = ["politicsNews","left-news","Government News","politics"]
to_general = ["worldnews", "News","US_News", "Middle-east"]
full_map = ChainMap(dict.fromkeys(to_politics,'politics'),dict.fromkeys(to_general,'general'))
```

In order to clean the text, I built a function that takes the input text and returns the cleaned text as output.

```
def process_text(text, length=False, stem=False):

    try:

        stop_words = set(stopwords.words('english'))

    except:

        nltk.download('stopwords')

        stop_words = set(stopwords.words('english'))
    if stem:
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(word.lower()) for word in text.split() if (word.isalpha()) and (word not in stop_words)]
    else:
        tokens = [word.lower() for word in text.split() if (word.isalpha()) and (word not in stop_words)]

    cleaned_text = ' '.join(tokens)

    if length:
        length_of_text = len(tokens)
        return cleaned_text,length_of_text
    else:
        return cleaned_text
```

I used NLTK's PorterStemming algorithm to filter stop words from texts, and then concluded the data processing with term frequency–inverse document frequency Vectorization for the titles and article texts.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vect_text = TfidfVectorizer(max_features=2500).fit(cleaned_df['cleanedText'])
vect_title=TfidfVectorizer(max_features=250).fit(cleaned_df['cleanedTitle'])
```

```
text_df = pd.DataFrame(vect_text.transform(cleaned_df['cleanedText']).toarray().astype(np.float16), column
s=vect_text.get_feature_names())
title_df = pd.DataFrame(vect_title.transform(cleaned_df['cleanedTitle']).toarray().astype(np.float16), col
umns=vect_title.get_feature_names())
```

After I linked the tf-idf terms to rest of the datetime features, I split them into train and test sets and then uploaded them to the S3 bucket.

```
pd.concat([y_train,X_train],axis=1).to_csv(path+'_train.csv',index=False,header=False)
```

```
pd.concat([y_test,X_test],axis=1).to_csv(path+'_test.csv',index=False,header=False)
```

```
sagemaker_session.upload_data(bucket=bucket, key_prefix=prefix, path=path+'_train.csv')
sagemaker_session.upload_data(bucket=bucket, key_prefix=prefix, path=path+'_test.csv')
```

For the pre-processing steps involved for the **LSTM model**, I focused only on the article texts as feature

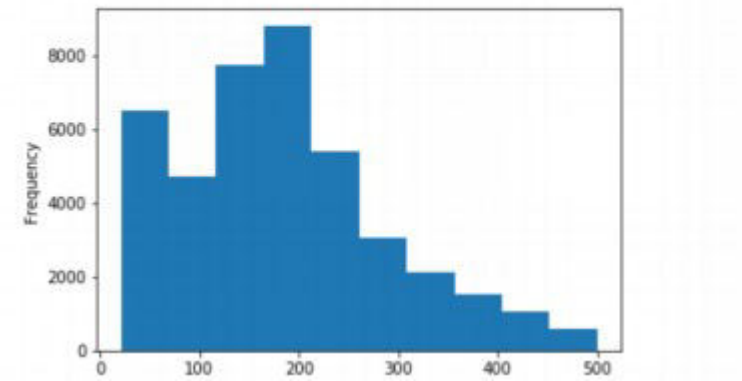
The pre-processing steps taken ;

➔ filtered texts with length below 20 and above 500 words to avoid empty or too long sequences
➔ split the data in Train, Validation and Test datasets with train_test_split from Sklearn
➔ applied a Tokenizer from keras to the Training set
➔ I padded all sequences with a max_len of 500 (filtering size):
➔ Linked labels with integer-encoded sequences and uploaded all the datasets to S3 bucket

Filtered text plot.

```
df_filt['articleLength'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbdc2f3f98>
```



Tokenizer applied to training set

```
text_tokenizer = Tokenizer(num_words=80000)
#title_tokenizer = Tokenizer(num_words=1000)

text_tokenizer.fit_on_texts(X_train['article'].astype(str))
#title_tokenizer.fit_on_texts(X_train['title'].astype(str))
```

padded all sequences with a max_len of 500

```
X_train = sequence.pad_sequences(X_train,maxlen=500, padding='post')
X_val = sequence.pad_sequences(X_val,maxlen=500, padding='post')
X_test= sequence.pad_sequences(X_test,maxlen=500, padding='post')
```

Implementation

In order To train a Naive Bayes model, I added a training script to Amazon SageMaker, which is located in ‘source_train/’, called ‘train_sklearn_nb’.py. I then incorporated the MultinomialNB After adding the arguments to the argumentParser ;

```
clf = MultinomialNB()

clf.fit(train_X,train_y)
```

I then created an Estimator and Launched the training job by passing the s3 location of the training data tothe fit() method ;

```
from sagemaker.sklearn import SKLearn
```

```
model = SKLearn(entry_point='train_sklearn_nb.py',
                source_dir='source_train',
                role=get_execution_role(),
                train_instance_count=1,
                train_instance_type='ml.m4.xlarge',
                )
```

```
key='udacityCapstone/data/vectorized_traindata.csv'
train_path = f's3://{bucket}/{key}'

input_channels = {"train":train_path }
```

```
model.fit(input_channels)
```

And finally, the deployed endpoints;

```
predictor = model.deploy(initial_instance_count=1,
                         instance_type='ml.c4.xlarge')
```

Int the case of the LSTM Model, using the tf.keras sequential API. I went with a vocabulary size of 8000 words, input length of 500 and embedding dimension of 128. irepeated this for the Bidirectional LSTM Layer and the Dense Layer.

```
def RNN():
    model = Sequential()
    layer = model.add(Embedding(80000,128,input_length=500))
    layer = model.add(Bidirectional(LSTM(128)))
    layer = model.add(Dense(128,name='FC1'))
    layer = model.add(Activation('relu'))
    layer = model.add(Dense(1,name='out_layer'))
    layer = model.add(Activation('sigmoid'))
    return model
```

The model was then saved and fitted.

```
model.fit(train_X,
          train_y,
          batch_size=256,
          epochs=args.n_epochs,
          validation_data=(val_X, val_y))


model_path ='/opt/ml/model'
model.save(os.path.join(model_path,'bi_lstm/1'), save_format='tf')
```

Refinement

A refinement I would have definitely considered. Would have been to increase the number n-grams in the tfiidf vectorizer and vocab size. Th e reason being that uisng only 1-grams will not allow us catch inter word dependencies which is crucial in text classification. In regard s to the RNN, initially the accuracy did not go aboev 50 % and was stuck here, in both the train and validation set. So, by adding Biderectional LSTM Layer, I noticed the training loss start to go down.

Results

Model Evaluation and Validation

The initial accuracy score was a low (57.4%) with the test data-set. Not exactly what I was looking for.

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
```

```
accuracy_score(true_labels,preds)
```

```
0.5739750445632799
```

But, when I started the LSTM, this time I came up with a validation set for the model just so we were sure it was not overfitting to the training set.

```
loss: 0.0037 - accuracy: 0.9992 - val_loss: 0.0464 - val_accuracy: 0.9877
```

As you can see here, a function was defined to adjust the probability threshold, due to the fact that the outputs were giving probabilities instead of class membership.

```
def threshold(x):
    if x < 0.5:
        return 0
    else:
        return 1
```

```
preds_df['preds']=preds_df['preds'].apply(threshold)
```

```
target_preds = pd.concat([y_test,preds_df], axis=1)
```

Here the results after testing the accuracy on the test data-set given ;

```
print(accuracy_score(target_preds['targetClass'],target_preds['preds']))
```

```
0.986639753940792
```

.

Justification

So Ultimately, I was able to beat the benchmark with an accuracy score of 98.6% on the sample test set perfectly balanced with biases removed. When it came to the un-threshholded predictions, the output of the final dense layer are quite far from the margin of .5, so small changes do not affect the prediction of the network. However, I do understand that given the limited dataset focusing specifically on world news and politics, somewhat limits the models range of performance. Obviously, adding more training data with an expanded range of subjects would remedy this issue. All in all, I feel satisfied with the results.