Machine Learning Capstone Project

Definition

Project Overview

This project deals with fake/true news detection. It can be inserted undoubtedly in the context of Natural Language Processing problems.

While I was navigating on Kaggle, I found this interesting dataset:
https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset/kernels

The dataset is made of 2 CSV files (true, fake news) which store title, article, date and subject of the articles.

Problem Statement

So, the problem can be stated in the following way: Given the text of an article, I want the algorithm to be able to predict whether it refers to True or Fake news.

Metrics

First, I had thought of using F1-Score or ROC-AUC as methods for scoring my algorithm, but given the balance of the 2 datasets, then I opted for a simple Accuracy Metric:

Acc = (True Positives + True Negatives)/(True Positives + True Negatives + False Positives + False Negatives)

Analysis

Data Exploration and Exploratory Visualization

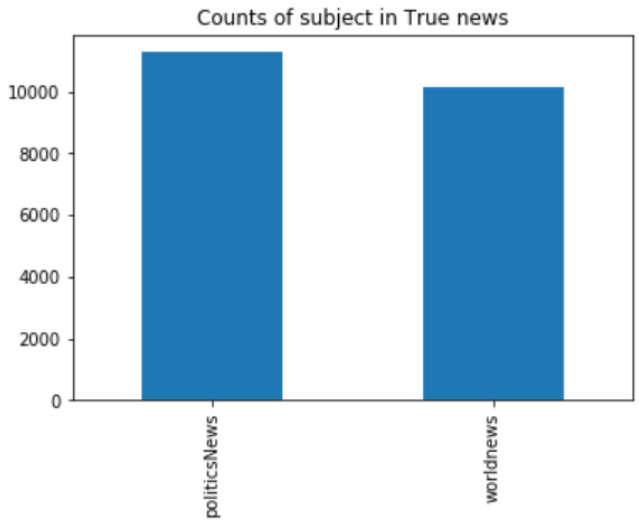
I started exploring the datasets in the notebook called “Data Exploration and S3 Integration.ipynb”.

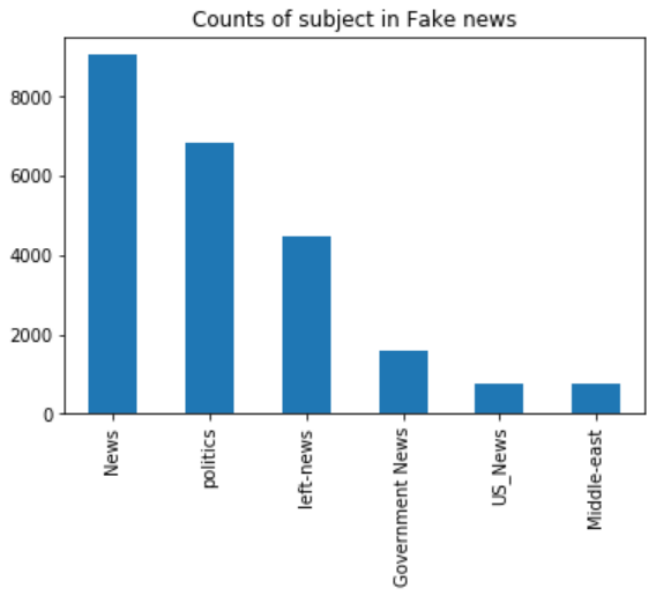
I first imported the True and Fake datasets with Pandas.read_csv() and checked the features: title, text, subject and date.

```
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

I plotted distributions of the 'subject' feature from both datasets and noticed that whereas in the True data, the subjects are almost evenly divided into 'politics' and 'world' news, in the Fake data article subjects are spread across several classes.





An anormality I found in the data is the datetime format, as you can see in the first image. Pandas did not catch the datetime format, so I needed to parse it in some specific way which i'll discuss in next steps.

Algorithms and Techniques

The algorithms I used for this project are two, and really diverse in nature.

The first approach is a Naive Bayes approach with Multinomial target. As it is very known, Naive Bayes models do work fine with text classification tasks in which features (word counts or tf-idf counts) are supposed to be independent. I chose this a baseline model because it was approximately simple to implement (remembering that SageMaker requires a custom training script for Sklearn models) which I wanted then to compare to a more sophisticated model, a Neural Network with a Recurrent layer.

In fact, the second model I chose was a Neural Network with a Bidirectional LSTM layer and a single sigmoid output, to present the binary classification task. I'll discuss issues and pros of the two models later on.

Benchmark

As a naive benchmark, I chose to take the accuracy score obtained by the Authors of the articles listed in my Capstone Proposal. In particular, they report an accuracy score of 90% by using a Linear Support Vector Machine.

Methodology

Data Preprocessing

Regarding Data preprocessing, I took several steps and choices depending on the model I was working with.

The first thing I did was add a label to the datasets and concatenate them in a complete dataset and also to reshuffle the whole dataset in order to be sure to uncover any possible bias.

```
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 preprocessing steps I took for Naive Bayes are the following:

- Parse dates in order to be able to extract features from date
- Text cleaning: lowercase, filtering numbers, URLS and hashtags, stemming
- Text tokenization and Tf-Idf vectorization

To parse dates, I needed to use the parse() function from parser module of dateutil package.

By applying it to the DataFrame column directly I obtained a value error, as there were some rows where the 'date' field was filled with URLs and texts instead of dates. So, I built a function to be able which returns NaN if the date cannot be parsed:

```
def parse_date(x):

    try:
        return parser.parse(x)

    except:
        return np.nan
```

And so I was able to parse dates correctly like 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]
```

And then I dropped rows with NaN date, which were about 10.

I also decided to remap all subject categories into two: 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'))
```

Going with text cleaning, I built a function which takes as input a 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 PorterStemmer and stopwords to stem and filter stop words from texts.

The last stage of data processing for the Naive Bayes model was Tf-Idf Vectorization for titles and article texts, with 250 and 2500 max features, respectively.

```
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())
```

Finally, I concatenated tf-idf terms to rest of the datetime features:

```
full_df = pd.concat([cleaned_df,text_df,title_df],axis=1)
```

Split it into train and test sets:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, random_state=1, stratify=y, test_size=0.3)
```

And then uploaded them to S3:

```
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)
```

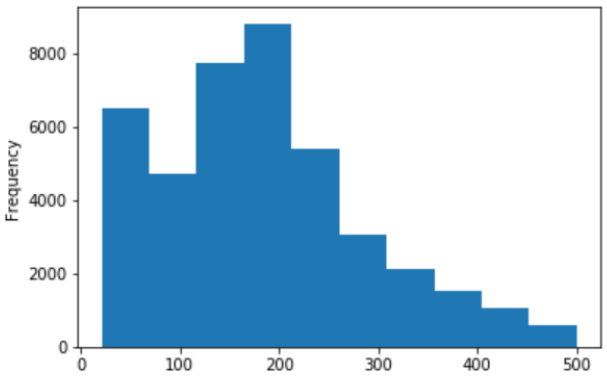
```
sagemaker_session.upload_data(bucket=bucket, key_prefix=prefix, path=path+'_train.csv')
sagemaker_session.upload_data(bucket=bucket, key_prefix=prefix, path=path+'_test.csv')
```

For what concerns the preprocessing steps for the LSTM model, I considered only the article texts as feature over which:

- I filtered texts with length below 20 and above 500 words to avoid empty sequences or too long sequences

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

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



- I split the data in Train, Validation and Test datasets with train_test_split from Sklearn
- I applied a Tokenizer from keras to the Training set which then I used to transform also Validation and Test Datasets (To avoid data Leakage)

```
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))
```

- I padded all sequences with a max_len of 500 (filtering size):

```
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')
```

- And finally I concatenated labels with integer-encoded sequences and uploaded all the datasets to S3.

Implementation

Multinomial Naive Bayes

To train a Naive Bayes model, I needed to provide a training script to Amazon SageMaker, which is located in 'source_train/', called 'train_sklearn_nb'.py.

After adding all the arguments to the argumentParser, I simply instantiated the MultinomialNB:

```
clf = MultinomialNB()


clf.fit(train_X,train_y)
```

And, on the SageMaker Instance, I created an Estimator and Launched the training job by passing the s3 location of the training data to the 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)
```

Which then I deployed as an endpoint:

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

Recurrent Neural Net

For what concerns the Neural Network, I chose the following structure (refer to train_keras_lstm.py) by using the tf.keras Sequential API:

```
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
```

As you can see, I chose a vocabulary size of 80000 words (same as the Tokenizer), an input_length of 500 (max_len of pad_sequences) and an embedding dimension of 128, which then I repeated for the Bidirectional LSTM layer and for the following Dense layer.

Then I fitted and saved the model:

```
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')
```

On the instance side:

Model Training

```
In [24]: input_channels = {"train":train_data, "validation":val_data}
```

```
In [25]: from sagemaker.tensorflow import TensorFlow
```

```
In [29]: # create a TensorFlow estimator
estimator = TensorFlow(entry_point='source_train/train_keras_lstm.py',
                       train_instance_type='ml.p2.xlarge',
                       train_instance_count=1,
                       role=role,
                       framework_version='2.1.0',
                       py_version='py3',
                      hyperparameters={"n_epochs":3})
```

So, you can see that I used a ml.p2.xlarge instance (1 GPU) with Tensorflow 2.

Refinement

A step I would have taken as a refinement would have been increasing the number of n-grams in the Tf-Idf Vectorizer and also its vocabulary size, because using only 1-grams does not allow catching inter-word dependencies which is essential in text classification, but it didn't really represent a problem as I wanted Naive Bayes to be a first SageMaker implementation.

As for the Recurrent Neural Network, the first implementation did not have a Bidirectional layer but only a unidirectional LSTM. I noticed that the loss wasn't decreasing and also the accuracy was stuck at 50% both on train and validation set. Then, I decided adding Bidirectionality to the LSTM and I noticed that the training loss decreased smoothly.

Results

Model evaluation and validation

As explained before, I used accuracy score as a metric for validating my model because I didn't want to come up with complex metrics, having a pretty well-balanced dataset after all.

When evaluating the accuracy on a test set, I obtained a low accuracy, about 57%:

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

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

```
0.5739750445632799
```

```
#This model did not perform as I wished.
```

It did not perform as I wished, but wasn't really a problem.

When I came up with the LSTM, I also provided the model a validation set in order to be sure that the model wasn't overfitting to the training set.

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

To test this model, I couldn't directly use predictions as the sigmoid outputs gives probabilities instead of class membership, so I defined a function to threshold the predictions:

```
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)
```

And testing the accuracy on the test dataset provided:

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

```
0.986639753940792
```

So I beat the benchmark and I found myself satisfied with the overall project.

Overall, I am satisfied with the results that I obtained. It must be said, though, that the dataset might show itself some kind of sampling bias, so in a real world context one would need much more data and cross-validation strategies to infer the robustness of the model, but in the scope of the project I believe to have reached a satisfactory result both in terms of solution and implementation using SageMaker for my first time.