CAPSTONE PROJECT: Fake News Detection

Machine Learning Engineer Nanodegree

June 17, 2021

I. Definition

Project Overview

Fake news became a critical problem in today's political and social life. Disinformation continues to feed intolerance and polarize our societies. In the age of information wars and foreign meddling in domestic politics through online disinformation, it is important to develop tools and strategies for how to sift through fake and potentially harmful online content.

In this project, I plan to build a classification model able to distinguish fake news from real ones. The project will use the XGBoost classifier model to accomplish this objective. This project is inspired by the Data Flair example[1] and a Medium article[2] on detecting Fake/Real News.

The dataset is obtained from Kaggle and contains labelled news in between 31 March 2015 and 19 February 2018. There are 44,898 articles in total, 23,481 of them fake. The distribution of fake versus real news, therefore, is balanced for this dataset (52.3% of articles is fake news).

It contains text variables for the titles and the contents of the articles. Two other variables are also contained in the dataset: (1) the type of article and (2) the date of the article. These two were excluded from the analysis. In particular, the type of the article was excluded from the analysis because the categorization of fake news and real news did not overlap. Inclusion of this feature into a training model would bias the training outcomes.

Problem Statement

The objective is to build a classifier model distinguishing between fake and real news. It is a binary (0/1) classification problem. The model's objective is to determine which news articles are fake and which are real based solely on their texts (pre-processed and vectorized). The target variable distinguishes between "fake" and "real" news.

[1] https://data-flair.training/blogs/advanced-python-project-detecting-fake-news/

[2] https://medium.com/@ODSC/deep-learning-finds-fake-news-with-97-accuracy-d774ca977b0d

The particular step-by-step tasks that will be achieved in the project are the following:

1. Download and clean the fake/real news data from Kaggle: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset
2. Implement two vectorizers: TF-IDF and Count Vectorizer
3. Test which of the two vectorizers result in better model performance with the benchmark model
4. Train a benchmark XGBoost classifier, report its performance
5. Tune the XGBoost model's hyperparameters
6. Test model performance on the test part of the data
7. Report accuracy of the trained model

Metrics

At the stage of hyperparameter tuning, *GridSearchCV* will be used. Here, the Area Under the Receiver Operating Characteristic Curve (ROC AUC) will be used for evaluation of the hyperparameters performance and the choice of the best of them. For the *XGBoost* classifier model itself, the main evaluation metric will be accuracy, but precision and recall will be reported as well.

accuracy = (true positives + true negatives)/sample size

These metrics are common in evaluating the performance of classifier models. The main goal will be to increase the accuracy and reduce error with no specific preference for reducing false negatives or false positives.

II. Analysis

Data Exploration

The project will use the data from Kaggle on fake and real news. The data can be downloaded from here: https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset. This data consists of two parts, one is called 'real', and the other one—'fake.' This is an appropriate dataset for the use for classification of fake vs real news. In fact, it is intended for exactly that reason.

In this section, I will visualize some of the articles text data. Because text visualization is more effective on already cleaned text, this chapter includes the text visualizations after the article texts were cleaned (removed stopwords, punctuation, urls, emojis, multiple whitespaces, etc.).

Two main visualizations will be used in this chapter: word clouds and frequency bar charts.

Exploratory Visualization 1: Word Clouds

The plot in Fig. 1 combines all pre-processed texts in the dataset and builds a word cloud, which summarizes the 100 most frequent words (separately without collocations). The texts here are pre-processed (but not yet stemmed) as described in the Data Preprocessing section below. Words that are larger in size are those that appeared more often than those in smaller font.

Fig 1. shows that the most common occurrences were in mentioning Trump, said, state, new, and president.

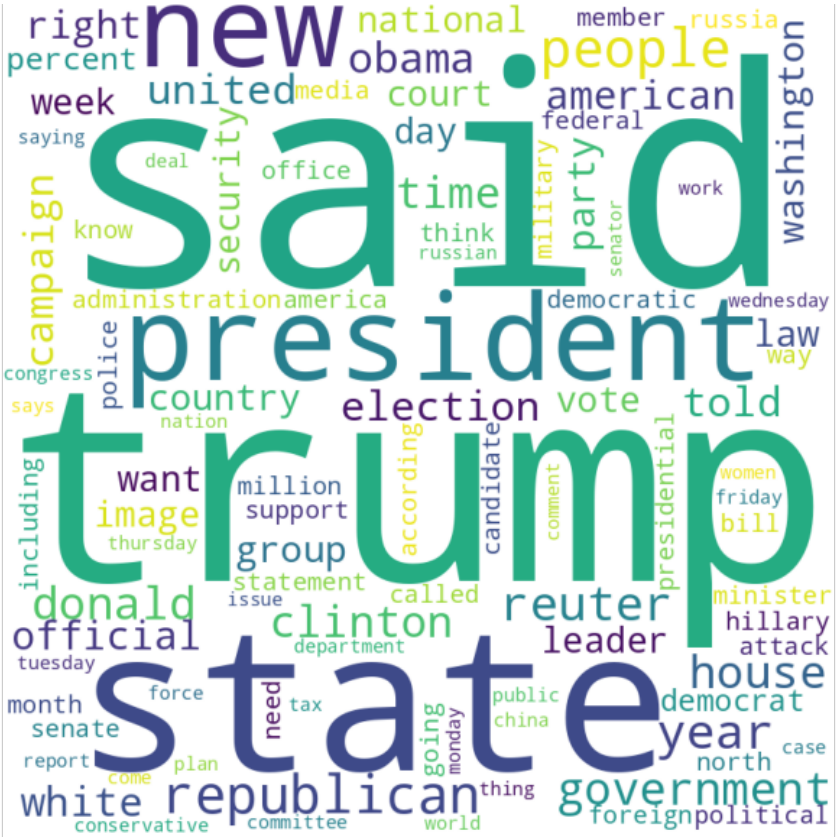


Figure 1 Word Cloud for pre-processed text

Exploratory Visualization 2: Bar Chart

Fig 2. summarizes the 20 most frequent words in a bar chart. The words mentioned above as the most frequent ones appear in the first five positions, but we also see that

the mention of trump and the word ‘said’ appeared in the article texts at least twice as often as any of the most frequent words.

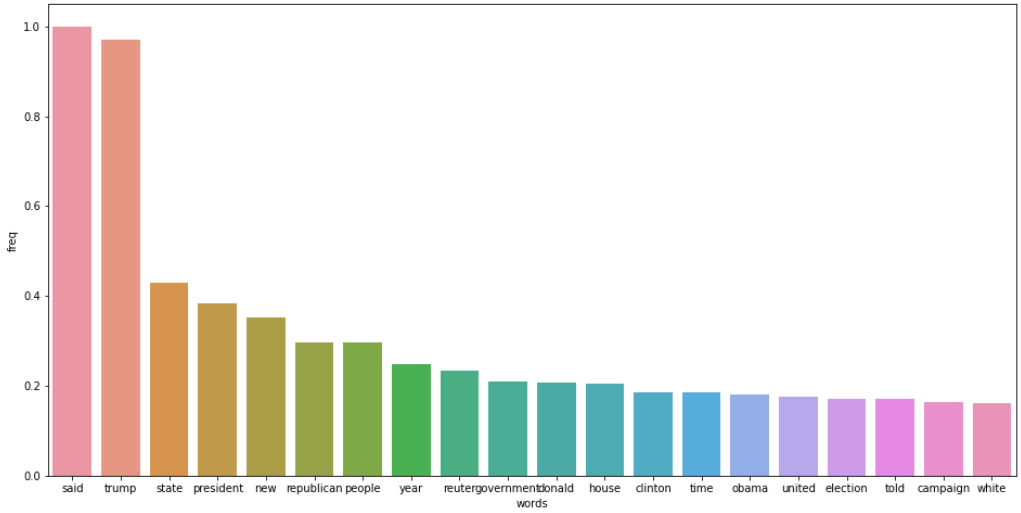


Figure 2 20 most frequent words

Among the least frequent word in the top 100 frequent words, we can also see words like ‘say’, ‘conservative’, ‘Monday’, ‘comment’, and ‘women’ (see Fig. 3).

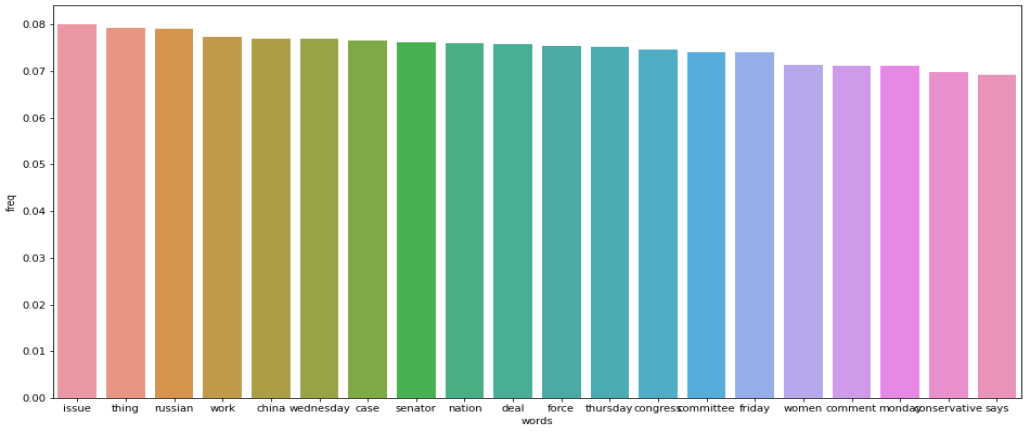


Figure 3 20 least frequent words in the top 100 frequent words

Algorithms and Techniques

The inputs for the decision of the pre-processing techniques are based on term frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF). Term frequency is the number of times the word appears in a document:

$$TF_{i,j} = n_{i,j} / \sum_k n_{i,j}$$

The TF-IDF is a measure of the originality of the word based on the number of frequency of the word in the document and the number of documents that the word appears, which is calculated by:

$$w_{i,j} = TF_{i,j} * \log(N/df_i)$$

The project will use either tf-idf or count vectorizer (depending on their performance) with the XGBoost classifier. The XGBoost is a scalable end-to-end boosting system.[3] It employs gradient tree boosting (gradient boosting machine, gradient boosted regression tree).

The main evaluation metric will be accuracy, but precision and recall will be reported as well. These metrics are common in evaluating the performance of classifier models. Seeds will be used to make the results replicable. XGBoost has been very successful on Kaggle. The authors (TIanqi Chen and team) report that in 2015, 17 out of 29 winning Kaggle solutions used XGBoost.

Benchmark

The model with TF pre-processed input and XGBoost will serve as a benchmark for further choice of vectorizer and model hyperparameter tuning. I chose to use this initial XGBoost model because I wanted to use it as a base for hyperparameter tuning. For the purposes of this project (choose best parameters with XGBoost classifier for fake news detection), this choice of this benchmarking is justified.

The initial model parameters are:

```
XGBClassifier(objective='binary:logistic',
              base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=np.nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1, random_state=42,
```

[3] https://arxiv.org/pdf/1603.02754.pdf

reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
subsample=1, tree_method='exact', validate_parameters=1,
verbosity=None, use_label_encoder=False, eval_metric='logloss')

The benchmark classifier basically uses the default parameters, such as max-depth of 6 or the learning rate of 0.3.

III. Methodology

Data Preprocessing

For preprocessing the data, first, I check if there are any duplicates. I check for the duplicates of articles using the column 'text'. The resulting number of observation in the final analytical sample after dropping the duplicates is 38,646 articles. As a result of this operation, the percentage of the fake articles drops to 45.16%, which is also quite balanced, so the chosen model performance evaluation metric (accuracy) is still applicable.

Then, I use Python packaged called NeatText.[4] It is a package created to clean unstructured text data and to reduce the noise, such as special characters and stopwords. In particular, I remove punctuation marks, contractions, emojis, numbers, special characters, html tags, stopwords (in English), urls, and emails. I also transform the text to lowercase. The text preprocessed in this way was used for creating visualizations in the Exploratory Visualization sections above.

In the next step, I use PorterStemmer from nltk package to extract stems of the words in the texts. Stemming allows decreasing the number of words in the vocabulary by combining, for example, verbs in different tenses to the same word stem.

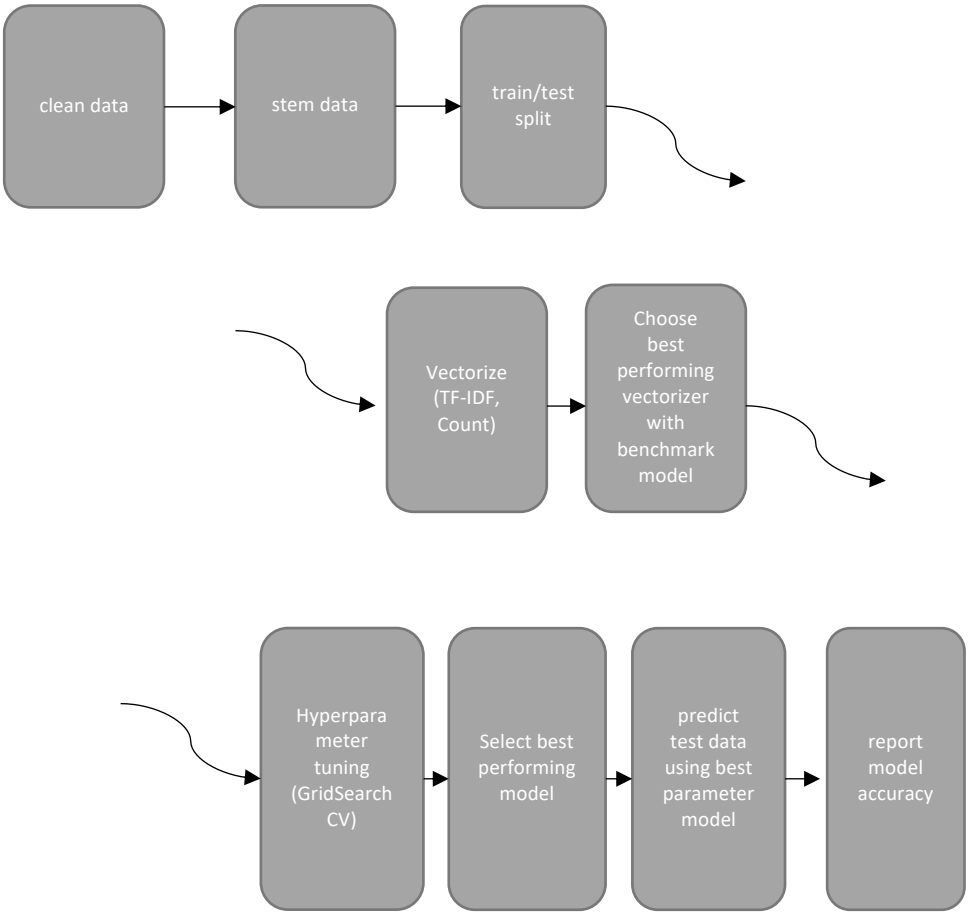
After stemming, the dataset was divided into two parts: training data and testing data with corresponding labeling subsets (the y-data). The testing proportion comprised 0.2 of the full dataset.

Implementation

The text analysis problems require the transformation of the text in a way that can letter be fed to models. Among possible vectorization possibilities, the project chose two common ones, in particular, TF-IDF and Count vectorization for data preprocessing. Because this is a classification problem, XGBoostClassifier can be used as one of the possible models for classification.

[4] https://pypi.org/project/neattext/

Starting from the benchmark model, I tested both TF-IDF and Count Vectorizer in their performance with the XGBoostClassifier benchmark model. Out of the two, the Count Vectorizer pipeline performed better on average (0.99 versus 0.97).



Refinement

GridSearchCV will be used for hyperparameter tuning. In tuning, Area Under the Receiver Operating Characteristic Curve (ROC AUC) will be used for scoring. For the model itself, the main evaluation metric will be accuracy, but precision and recall will be reported as well. These metrics are common in evaluating the performance of classifier models.

The most complications were in the hyperparameter tuning process because it was the most time-consuming step of the process. On the bright side, hyperparameter tuning does not have to be done very often but only perhaps once for certain data.

Using the Benchmark model mentioned in the Analysis section as the starting point, I further improve the model by tuning hyperparameters.

In the first round of hyperparameter tuning, I chose five hyperparameters to tune: ‘learning_rate’, ‘gamma’, and ‘reg_lambda’’. As a result of search for the best parameters, I found that the following specifications were the best among tested in the first round: {'gamma': 1.0, 'learning_rate': 0.3, 'reg_lambda': 1.0}. These results showed that learning_rate and reg_lambda hyperparameters were already optimally set in the benchmark model. The only parameter which was set lower was gamma. It showed that the maximum of the tested set [0, 0.25, 1.0].

In the second round, when testing the gamma parameter of the following values: [1, 5, 10], the best parameter was identified as gamma=1. So, in the final model, I adjusted gamma from 0 to 1.

IV. Results

Model Evaluation and Validation

The metric that is used to evaluate the model performance is its accuracy. The benchmark model already performed very well with the count vectorizer at 99.52% of accuracy. After hyperparameter tuning, the model accuracy improved by 0.01%. Considering that the benchmark model already performed very well, this is still an improvement on the initial model.

Fig 4. shows the confusion matrix for the predictions of the final model. There were 13 false positives and 23 false negatives (identified as real but, in fact, fake).

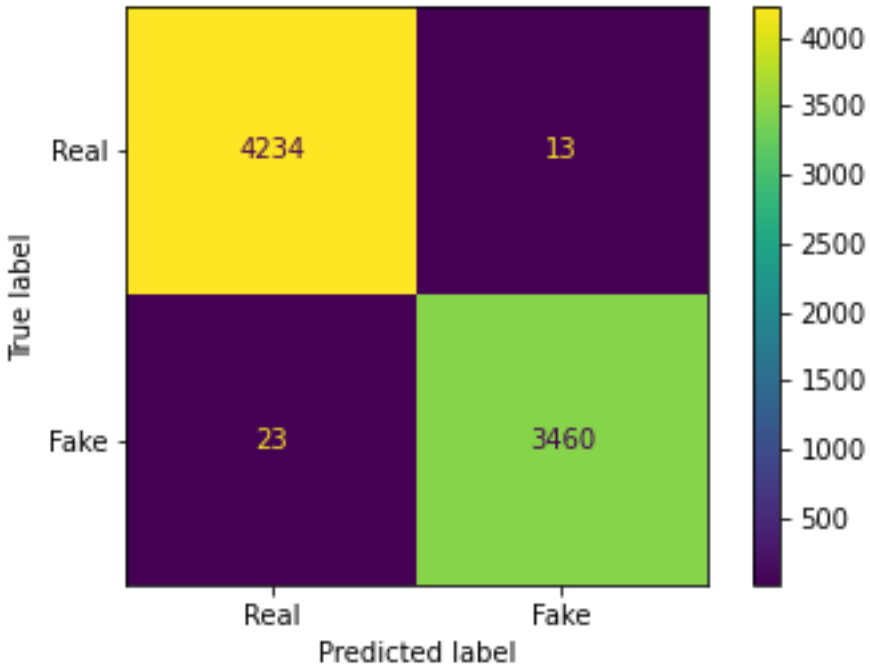


Figure 4 Confusion Matrix

Justification

The best performing model after 2 rounds of hyperparameter tuning was the XGBoost classifier with the following hyperparameters:

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
gamma=1, gpu_id=-1, importance_type='gain',
interaction_constraints='', learning_rate=0.3, max_delta_step=0,
max_depth=6, min_child_weight=1, missing=nan,
monotone_constraints='()', n_estimators=100, n_jobs=16,
num_parallel_tree=1, random_state=42, reg_alpha=0, reg_lambda=1,
scale_pos_weight=1, seed=42, subsample=1, tree_method='exact',
use_label_encoder=False, validate_parameters=1, verbosity=None)

This model performed with 99.53% accuracy. Although the benchmark model already performed well together with the count vectorizer (term frequency) at 99.52% accuracy, the hyperparameter tuning allowed to improve the performance by 0.01%. This accuracy is quite high for the models that are trying to classify fake and real news.