METCS521 FINAL PROJECT SUBMISSION

Fake News Detector

Authors: Ross Sheng and Futaba Kikuchi

Introduction:

In this project, we explored supervised machine learning algorithms to detect fake and real news. We looked at a csv dataset, then vectorized the data so it could be used in a classifier model to predict if a given news article is real or fake. To complete this main objective for this project, we created two Jupyter scripts that accomplish the following:

1. Download the dataset
2. Divide the data into train and test sets with Sklearn model selection
3. Vectorize the content with Sklearn feature extraction vectorizer and naïve bayes algorithms
4. Apply a classifier and return its accuracy for its predictions

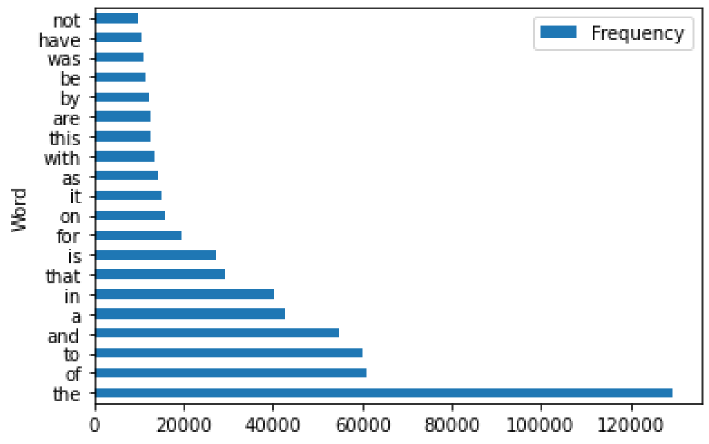
Initially, we used two different methods and classifier models to fulfill one of our extensions as well as to see which one we thought would be more suitable in better achieving our objective. By comparing two different classifier models, we were able to see how different classification algorithms impacted the accuracy of being able to predict data based on modeling.

Downloading the Datafile:

Our script that downloads the datafile is scipt.ipynb. We import pandas to be able to manipulate the data from our csv files so that it could be stored in a data frame table.

Initial Data Analysis:

In our first few weeks trying to gain an understanding of what is going on within our data, we looked to create many graphs such as seeing if there was an even number of fake/real news or using select keywords to see if they had an even distribution with fake/real news. The most interesting graph we came to find was when we tried to the most occurring words within the text of our data frame.



As we split each word and counted the most frequent words, we saw that stop words that didn't add any semantic meaning to the text like 'the', 'and', 'to' and 'of' occurred hundreds of thousands of times. This let us realize how important it was that when we were vectorizing and training our classifier, we must remove these stop words as efficiently as possible. This was done in the Data_Analysis.ipynb file.

Comparing Classifier Models (Extension 1):

With our extension, we wanted to compare two classifier models: The Passive Aggressive Classifier from sklearn.linear_model and the MultinominalNB classifier from sklearn.naive_bayes. When we compared the two, we used the same method of natural language processing where we finished vectorizing the dataset with a TF-IDF vectorizer that filters the list of words with stopwords and then converted that text into vectors of numbers. We then gained an understanding of what went on in each of the classifier models and compared the output accuracy of each model. With the Multinominal Naïve Bayes classifier, it is a classifier that is based on supervised learning. It takes training data and its independent variables, and tries to determine its dependent variables (fake/real). It determines the classification of its independent variable (text of articles) with a simple algorithm where the prediction of the class = the number of variables associated with that class (fake/real) divided by the total number of independent variables. Then with the Passive Aggressive classifier model, we learned that this is a real time data collection model that updated the classifier for every example that is fed into it. As the model is constantly being updated with its predictions based on the training data set, we initially believed the Passive Aggressive classifier would be a better approach in terms of its accuracy to predict future data. After training the classifier models, we got its accuracy score from sklearn metrics, and our initial guess turned out to be correct. The Passive Aggressive model returned a score of **94.24%** accuracy, while the Multinomial Nive Bayes classifier returned a score of **82.64%.** As we compared these two models, we decided that it would be best to continue with our second extension with the Passive Aggressive classifier model. We furthered our understanding of how accurate the Passive Aggressive classifier model was by getting a classification report to evaluate its performance in its prediction with test data. As we see a weighted average precision score of 0.95, we were confident that the classifier would be able to do a good enough job in predicting future input data.

```
              precision    recall  f1-score   support

        FAKE       0.95      0.95      0.95       638
        REAL       0.95      0.94      0.95       629

    accuracy                           0.95      1267
   macro avg       0.95      0.95      0.95      1267
weighted avg       0.95      0.95      0.95      1267
```

We then used the pickle module to dump our classification model and data into a pkl file so that it can be read by our simple web application. This was all done in the files machine learning 2.ipynb

Creating a Simple Application to Upload and Test Results (Extension 2):

With our second extension, we decided to create a simple web application to upload a URL of an article link that would determine if the news was real or fake. A substantial amount of research was done to complete this task as neither of us in the group had much experience with web app development.

References for us to learn how to use certain modules, and html include:
https://www.w3schools.com/html/

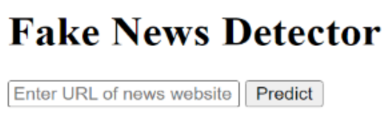
https://www.fullstackpython.com/flask-app-flask-examples.html

https://pythonbasics.org/what-is-flask-python/

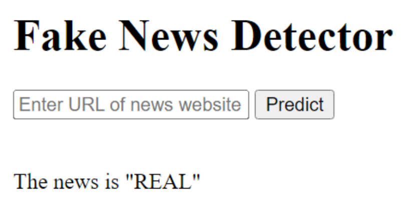
https://www.youtube.com/watch?v=GS_ylghUtLQ&ab_channel=MachineLearningHub.

https://medium.datadriveninvestor.com/improve-the-text-classification-results-with-a-suitable-preprocessing-step-gridsearchcv-and-f19cb3e182a3

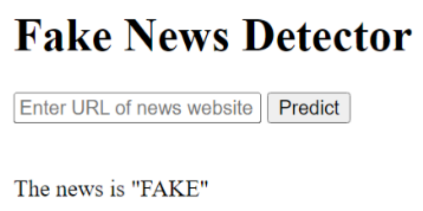
The files we implemented for this application include fake_news_application.py, model.pkl, and a templates folder that includes our style reference file and html file. First with the fake_news_application, we used new modules that include: flask (simple python web app framework), pickle (serialize our classification model from our machine learning 2.ipynb file to be unserialized and read in the fake_news_application so that it could predict whether the links to articles we feed into the application would be fake or real), newspaper, and articles (modules to be able to break down and parse the article links so that it could be fed to our classifier models). Though a detailed explanation of each line of the code is written in the README file, we basically defined methods that initializes the application with flask, reads the classifier model with pickle, parses the URL text with newspaper and article, and prepares the application to be run with os within main. Then with online guides as a reference, we created a template file that holds the style css file as well as the html file where we defined an input (URL) and output (FAKE or REAL). After running our fake_news_application.py on python, we receive this link: http://127.0.0.1:5000 and when we open it, we get this below:


Fake News Detector

We used https://beforeitsnews.com/v3/top50/ to get fake news, and CNN to get real news. As a result, we get the result below when we input the URL of a fake article and a real article:


Fake News Detector

The news is "REAL"


Fake News Detector

The news is "FAKE"

Conclusion:

As we set out an objective to be able to train and test a classifier model, use the classifier model to predict if a given source of news is fake or real, we believed we accomplished this task very well. In addition to being able to complete our two extensions of comparing classifier models and creating a simple web application to predict results, we were able to learn about the basics of natural language processing and how text must be converted to a vector of numbers to be able to be read by an algorithm to be analyzed by the classifier model. Additionally, we learned how different modules could be used to help accomplish a very daunting task. Though html was very foreign to us, we were still able to accomplish this task by looking up references online.