ISyE 6740 – Summer 2020

Final Report

**Team Member Name**: Ian Emmons (iemmons3)

**Project Title**: Unreliable News Detection

**Problem Statement**

Unreliable news publication has become a topic of interest in the media. Social media has made it easier than ever for users to spread potentially dubious information to their networks. Further, some false claims can be overtly detrimental to the health and well-being of social media consumers. For example, disinformation on the effects of vaccines or effectiveness of some treatments or preventative measures of COVID-19 (Coronavirus) could lead to poor personal and societal health outcomes.

Social media companies, such as Facebook or Twitter, constantly face pressure to censor untrustworthy material that is shared on their platforms. However, with 2.6 billion monthly users, human review of shared content for Facebook would be untenable. A machine learning model to classify untrustworthy and trustworthy content would be a necessity to tackle this problem.

This work attempts to devise a model capable of classifying untrustworthy and trustworthy news articles using only the articles’ titles and text.

**Data Sources**

In this project, a Fake News dataset from kaggle.com was used for model training and initial model performance evaluation.1 The data contained the following predictors:

* Id
* Title
* Author
* Text

The data also contained labels for whether each article was deemed reliable or unreliable. The documentation of the data set lacked any indication of how the labels were generated. The implications of this will be discussed further in the Evaluation and Final Results section.

A second set of data from kaggle.com was used as a secondary test set. It included similar predictors. The articles were labeled either “True” or “False”. These labels were generated using fact checks from politifact.com.2 Though true/false might have slightly different implications that reliable/unreliable, for purposes of this analysis, they were considered analogous.

For both data sets, any articles with missing text or title were excluded. Additionally, any articles with less than 30 words in the text were excluded. The project intent was to classify articles; anything shorter than 30 words was considered out of scope.

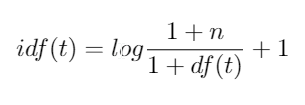
The first data set was partitioned into a training and test set. 80% (14628 articles) were used for training, while the remaining (3657) articles were used for model testing. The entire second data set was used for testing.

**Methodology**

*Variable Selection and Feature Engineering*

With the provided data, only the title and text attributes were used as predictors. The author was excluded from all modeling. Though author would probably be a highly informative predictor of whether a given article was reliable, using it in the model might render the model totally ineffective at classifying articles with either no author given or an unknown author not seen in the training set.

In order to transform the textual data into vectors for model training, two well-established methods were used: count vectorization and term frequency-inverse document frequency vectorization (TF-IDF). TF-IDF is a technique used to emphasize words in an article that are least common in the whole list of articles (the corpus). So, for example, the word “yesterday” might end up with a smaller weight because it is likely to occur in many news articles, while a word like “launch” may have a higher weight if it appears in an article, since it would be less common in the corpus. Equation 1 shows the formula used to calculate inverse document frequency for a given word in a single document. *n* represents the total number of documents, while *df(t)* is the number of documents containing the word *t*. This quantity is then multiplied by term frequency to calculate TF-IDF. For both vectorizing methods, existing libraries were used to strip out common, un-informative words, such as “and” or “is”.



**Equation 1. Inverse Document Frequency calculation.**

Each article has two strings associated: a title and text. Since these have different meanings, they were transformed into vectors separately. During model selection, using only title, only text, and both title and text vectors was evaluated. When using both title and text, the two vectors are simply concatenated. In the training data set, the number of features – unique words – in the title vectors was 18,068, while the number of features in the text vectors was 143,479.

With many features, singular value decomposition was employed in order to attempt to reduce the training vectors into fewer features. After decomposition, the predictors were scaled. Scaling was not performed when using the full feature set, as the full feature set is a sparse matrix and storing the full feature set in a non-sparse matrix was not compuationally feasible. Though singular value decomposition generally improved model fit time, performance was slightly degraded, according to 5-fold cross-validation (more discussion follows).

*Initial Model Comparison*

First, in order to establish a baseline for model performance, naïve Bayes classifiers were fit to the count-vectorized data. These were expected to perform relatively poorly since Bayes classifiers assume class conditional independence of predictors. In text classification, this is likely untrue. However, multinomial naïve Bayes models using document titles, text, and both yielded cross-validation scores of 0.902, 0.914, and 0.927, respectively. The models performed decently well. However, more models were evaluated to improve on the naïve Bayes performance.

The following models were compared: logistic regression with no penalty term, logistic regression with *L1* regularization, logistic regression with *L2* regularization, a neural network with rectified linear unit neurons, a random forest classifier, and *k-*nearest neighbors with 20 neighbors.

Both count and TF-IDF vectorization were evaluated. Models were trained on either article text, article title, or both. Singular value decomposition was used in order to reduce the number of features from 18,068 features to 10, 100, or 1000 features. When singular value decomposition was used with the combination of text and title data, the title and text vectors were concatenated before decomposition.

All possible combinations (168) were tested. Unless otherwise mentioned, default parameters were used in Python’s scikit-learn module for initial evaluation. In order to evaluate initial model performance, cross-validation was performed with 5 folds on the training data. Details on all models’ performance may be found in the supplementary material. Results with an asterisk did not converge but are still likely close to the true prediction accuracy. Bolded results are investigated further in the hyperparameter tuning step.

Overall, the best results were produced when using TF-IDF vectorization and using a combination of title and text. This makes intuitive sense since using both title and text allows maximal characteristics of the articles to be considered. And TF-IDF is well-understood to be effective at accentuating a given articles most important words. Across most model types, performance degraded slightly as the number of dimensions was reduced with singular value decomposition. The one exception to this rule is *k-*nearest neighbors, which performed extremely poorly when trained on all features, but was competitive with other methods at low dimensionality. This is likely because most other models determine which features are most important, while *k-*nearest neighbors assumes all features have equal weight.

*Hyperparameter Tuning*

Two models were carried into parameter tuning due to their effectiveness. First, the neural network model with 100 features. Though several neuron function and regularization parameters were tried, few had much impact on overall model efficacy (Table 1).

|  |  |
| --- | --- |
| Model | Cross-validation Score |
| Rectified Linear Unit (lambda=1) | 0.966 |
| Rectified Linear Unit (lambda=1e-2) | 0.967 |
| Rectified Linear Unit (lambda=1e-3) | 0.968 |
| Rectified Linear Unit (lambda=1e-4) | 0.969 |
| Rectified Linear Unit (lambda=1e-5) | 0.97 |
| Rectified Linear Unit (lambda=1e-6) | 0.968 |
| Rectified Linear Unit (lambda=1e-7) | 0.969 |
| Rectified Linear Unit (lambda=1e-8) | 0.968 |
| Logistic (lambda=1) | 0.96 |
| Logistic (lambda=1e-2) | 0.97 |
| Logistic (lambda=1e-3) | 0.969 |
| Logistic (lambda=1e-4) | 0.969 |
| Logistic (lambda=1e-5) | 0.968 |
| Logistic (lambda=1e-6) | 0.969 |
| Logistic (lambda=1e-7) | 0.969 |
| Identity (lambda=1e-4) | 0.965 |
| Hyperbolic Tangent (lambda=1e-4 | 0.968 |

**Table 1. Parameter tuning with neural network model**

Next, the regularization parameter was optimized for the logistic regression model with *L1* regularization. This method is helpful for reducing the number of parameters, as it tends to force some coefficients toward zero. It differs from *L2* regularization, which reduces all parameters equivalently. Table 2 shows the search for an optimal regularization parameter. The parameter itself is inversely related to regularization strength; in other words, a lower parameter leads to stronger regularization. The best parameter was found to be 0.1.

|  |  |
| --- | --- |
| Regularization Penalty | Cross-validation Score |
| 10 | 0.969 |
| 1 | 0.972 |
| 0.1 | 0.974 |
| 0.01 | 0.948 |
| 0.9 | 0.972 |
| 0.8 | 0.972 |
| 0.7 | 0.972 |
| 0.6 | 0.972 |
| 0.5 | 0.973 |
| 0.4 | 0.973 |
| 0.3 | 0.973 |
| 0.2 | 0.974 |
| 0.1 | 0.974 |
| 0.05 | 0.971 |
| 0.025 | 0.964 |
| 0.01 | 0.948 |

**Table 2. Parameter tuning of *L1* logistic regression model.**

Finally, though it was not a type of model initially considered, some tuning was done on an elasticnet logistic regression model. This is a model that mixes *L1* and *L2* regularization and often can outperform both. This model has two parameters: the regularization penalty, which is equivalent to the parameter used in *L1* regularization, and the *L1* ratio, which determines the relative strength of *L1* and *L2* regularization. A ratio of 0 would be equivalent to normal *L2* regularization, while a ratio of 0.5 would have even weighting for *L1 and L2* regularization. Table 3 shows the simultaneous optimization of both parameters. Though many combinations of parameters produce very similar results, we decided to use regularization penalty 0.05 and *L1* ratio 0.5. With this data set a more regularized model seems desirable to help reduce model variance and hopefully make the model perform better on new data. The elasticnet model slightly outperformed the *L1-*only model. Though this model was given 1000 possible features, the fitted model only has 661 features with non-zero coefficients.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| L1 Ratio | Regularization Penalty | 0.01 | 0.05 | 0.1 | 0.2 | 0.35 |
| 0.1 |  | 0.975 | 0.976 | 0.974 | 0.973 | 0.972 |
| 0.2 |  | 0.973 | 0.976 | 0.975 | 0.973 | 0.973 |
| 0.3 |  | 0.969 | 0.976 | 0.975 | 0.974 | 0.973 |
| 0.4 |  | 0.967 | 0.975 | 0.975 | 0.975 | 0.974 |
| 0.5 |  | 0.963 | 0.976 | 0.975 | 0.975 | 0.975 |
| 0.6 |  | 0.96 | 0.975 | 0.975 | 0.975 | 0.975 |
| 0.7 |  | 0.957 | 0.974 | 0.976 | 0.975 | 0.975 |
| 0.8 |  | 0.954 | 0.973 | 0.975 | 0.975 | 0.975 |
| 0.9 |  | 0.951 | 0.972 | 0.975 | 0.975 | 0.975 |

**Evaluation and Final Results**

The fitted elasticnet logistic regression model was fitted on the training data (80% of first data set). Importantly, the TF-IDF vectorizers and the singular value decomposition objects were also fitted only on training data. This allows evaluation of how the model might perform on totally new data, which may even have new terms not in the initial training set. Using this model to predict the labels on the test data yielded a prediction accuracy and AUC of 0.977.

Finally, the model was used to predict the labels from the second data set. Again, these articles were labeled “True” or “False”, as opposed to “Reliable” or “Unreliable”. Model performance on this secondary data set was rather poor, with prediction accuracy and AUC of 0.700 and 0.699, respectively. Without further analysis, it is difficult to say whether this diminished accuracy was due to differences in labeling criteria or differences in the nature of the two sets. Perhaps the second data set sampled a far different set of publications than the first and the model could not perform on the novel article structure.

Next, a new logistic regression model was fitted on 80% of the second data set using the same hyperparameters as previously used. Using this model to predict the labels of the final 20% of articles, this model performed exceedingly well, with prediction accuracy and AUC of 0.991. This further suggests that perhaps labeling differences contributed to the poor performance when using a model trained on one data set to predict labels on the second.

**Future Work**

Some work was considered but not done in this project. First, it might be helpful to attempt to extract phrases from source documents as features, rather than simply looking at word frequencies. Additionally, it could be helpful to provide some analysis of which words or phrases were the most indicative of a given article being reliable or unreliable. This could probably be achieved using the logistic regression model.

With more subject matter experts, it may be helpful to have them label articles, rather than relying on the data sets found on Kaggle. Reliability and unreliability are in the eyes of the beholder. If implementing a tool like this for a social media company, it would be important to use the company’s labels in the training data set.

**Supplementary Materials**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Logistic Regression - no penalty | Logistic Regression - L1 | Logistic Regression - L2 | Neural Network | Random Forest | K-Nearest Neighbors |
| Title Only | SVD 10 | 0.899\* | 0.899 | 0.899\* | 0.906 | 0.91 | 0.901 |
|  | SVD 100 | 0.91 | 0.911 | 0.91\* | 0.909 | 0.917 | 0.816 |
|  | SVD 1000 | 0.922\* | 0.917 | 0.914 | 0.919 | 0.908 | 0.769 |
|  | All features | 0.933\* | 0.928 | 0.929 | 0.927 | 0.93 | 0.776 |
| Text Only | SVD 10 | .809\* | 0.81 | .81\* | 0.859 | 0.853 | 0.841 |
|  | SVD 100 | .904\* | 0.907 | .907\* | 0.917 | 0.894 | 0.808 |
|  | SVD 1000 | 0.936\* | 0.936 | 0.933\* | 0.924 | 0.836 | 0.522 |
|  | All features | 0.935\* | 0.953 | 0.951\* | 0.957 | 0.91 | 0.434 |
| Title and Text | SVD 10 | 0.81\* | 0.811 | 0.811 | 0.862\* | 0.855 | 0.844 |
|  | SVD 100 | 0.912\* | 0.915 | 0.915 | 0.928\* | 0.897 | 0.814 |
|  | SVD 1000 | 0.953\* | 0.952 | 0.949\* | 0.936 | 0.844 | 0.521 |
|  | All features | 0.946\* | 0.973 | 0.971\* | 0.968 | 0.943 | 0.435 |

**Table 1. Initial Model Performance with Count Vectors**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Logistic Regression - no penalty | Logistic Regression - L1 | Logistic Regression - L2 | Neural Network | Random Forest | K-Nearest Neighbors |
| Title Only | SVD 10 | 0.891 | 0.891 | 0.913\* | 0.913 | 0.911 | 0.904 |
|  | SVD 100 | 0.909\* | 0.909 | 0.907\* | 0.916 | 0.915 | 0.803 |
|  | SVD 1000 | 0.916\* | 0.916 | 0.914 | 0.914 | 0.905 | 0.573 |
|  | All features | 0.929\* | 0.921 | 0.918 | 0.906 | 0.933 | 0.57 |
| Text Only | SVD 10 | 0.848 | 0.848 | 0.848 | 0.895\* | 0.887 | 0.871 |
|  | SVD 100 | 0.932\* | 0.932 | 0.932 | 0.945 | 0.919 | 0.869 |
|  | SVD 1000 | 0.949\* | 0.944 | 0.943 | 0.95 | 0.9 | 0.451 |
|  | All features | 0.962\* | 0.937 | 0.948 | 0.961 | 0.011 | 0.432 |
| Title and Text | SVD 10 | 0.927 | 0.927 | 0.927 | 0.951\* | 0.944 | 0.939 |
|  | SVD 100 | 0.964\* | 0.963 | 0.963 | **0.969** | 0.954 | 0.897 |
|  | SVD 1000 | 0.971\* | **0.97** | 0.967 | 0.973 | 0.94 | 0.479 |
|  | All features | 0.979\* | 0.967 | 0.971 | 0.97 | 0.946 | 0.434 |

**Table 2. Initial Model Performance with TF-IDF Vectors**

**References**

[1] “Fake News”. Kaggle.com. <https://www.kaggle.com/c/fake-news/data>

[2] “Fake and real news dataset”. Kaggle.com. <https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset>