COMP9417 Assignment 2:

Fake News Challenge Stage 1 (FNC-I):

Stance Detection

Andy (studentID) Edrian (studentID) MJ (studentId)

**1. Introduction**

**1.1. The problem.** Fake news is news created to deliberately misinform or deceive readers. Detecting fake news is currently a major issue for social media companies such as Facebook, who are investing vast amounts of resources to find ways to detect and prevent its circulation. The Fake News Challenge Stage 1 (FNC-1) held from late 2016 to mid-2017, was a competition based on tackling one aspect of fake news detection – stance detection. Stance detection involves comparing the perspectives (or stances) of two pieces of text relative to a topic, claim or issue (reference FNC website). This involves comparing a headline and body of text and determining one of four relationships between them: agree, disagree, unrelated, neutral. This would prove useful in identifying what different news organizations are saying about a topic. This challenge seeks to explore ways in which we can automate this process.

**1.2. The approach.** We engineered features to capture the different aspects of the relationship between the headline and body of an article. The features we engineered all aim to address one or more of the target labels i.e. agree, disagree, unrelated, neutral. We then train an XGBoost classifier on some training data published by the FNC-I website(reference) and validate our model on some test data also provided by the website(reference).

**1.3. Important aspects.** The most important aspects of our project with respect to machine learning is the task of feature engineering. Features must capture the relationship between two natural language texts which relate to the stance classifications. The most successful participants of this competition had implemented deep neural networks. Since this is out of the scope of this course, we attempt to apply concepts and traditional machine learning approaches learned throughout COMP9417 such as decision tree learning, logistic regression, and boosting.

**2. Implementation**

**2.1. Data pre-processing.** Our implementation begins by pre-processing the headline-article pairs in the training set. This stage consists of four stages: tokenizing, stemming, stop-word removal, and n-gram conversion.

**2.1.1. Tokenizing.** Headline and body texts are split into individual words, delimited by space.

**2.1.2. Stemming.** Tokens are then stemmed using an English language stemmer from the Python NLTK module.

**2.1.3. Stop-word removal.** Stop words based on the Python NLTK stop-word collection are removed from each text.

**2.1.4. N-gram conversion.** Finally, a text is converted into a list of unigrams, bigrams and trigrams which are groupings of n-number of successively occurring tokens.

**2.2. Feature engineering.** The most important part of our task is to engineer features which reflect some aspect of the relationship between the headline and article body. The features we chose are mostly based on the competition winner’s approach, but we have also incorporated a couple of our own features. These features include: TF-IDF cosine similarity, SVD cosine similarity, word2vec, sentiment analysis, and body text length. The process of engineering each feature are explained below.

**2.2.1. TF-IDF cosine similarity.** Representing headlines and body texts as TF-IDF allows us to extract keywords from both and compare the topical similarity between headlines and body texts. We expect this measure to be particularly useful in gauging the relevance of headlines to body texts. This feature would help discriminate *unrelated* stances from related stances (i.e. *agree*, *disagree*, and *discuss*).

**2.2.2. SVD cosine similarity.** Since TF-IDF vectors are rather sparse, we convert them into a smaller, denser feature space using Latent Semantic Analysis and Singular Value Decomposition. We then measure their similarity in the context of the latent topic feature space. As for TF-IDF, we expect that SVD cosine similarity will help to distinguish between unrelated and related texts based on latent topics.

The following table depicts the significant difference between unrelated and non-unrelated stance values for TF-IDF and SVD features. It is apparent that unrelated stances have a far lower TF-IDF and SVD than non-unrelated stances on average. From the distribution of SVD similarity values visualized in Appendix (?) we also observe that unrelated stances are more likely to have lower values, as opposed to other stances which are more likely to be higher.

|  |  |  |
| --- | --- | --- |
|  | TF-IDF mean value | SVD mean value |
| Unrelated | 0.0035 | 0.1617 |
| **Agree** | **0.1246** | **0.7080** |
| **Disagree** | **0.1118** | **0.7482** |
| **Discuss** | **0.1096** | **0.6784** |

**2.2.3. word2vec.** Explain it here.

**2.2.4. Sentiment analysis.** In order to capture *agree*, *disagree* and *discuss* stances, we introduce a sentiment analysis feature that compares how close the overall sentiment measure of the headline and body texts are. Sentiment analysis measures range between -1 and 1, where values closer to 1 are positive; values closer to -1 are negative; and values around 0 are neutral. Instead of measuring the sentiment of entire texts, we select the ten tokens with the highest TF-IDF values to represent the original text. Our implementation uses a sentiment analyser from the Python NLTK module.

**2.2.5. Body text length.** We expect that the length of a body text gives some indication of how much of the headline is being discussed. Generally, longer body texts are more likely to elaborate on the headline. Therefore, this feature may prove useful in identifying *discuss* stances.

**2.3. Model implementation.** The models we chose to implement are 1) an XGBoost model, 2) random forest model, and 3) a multiclass logistic regression model. Each model is trained on all of the features specified in the previous section.

**2.3.1 XGBoost model.** Extreme Gradient Boosting (XGBoost) is a boosting algorithm that is relatively much faster than other gradient boosting algorithms due to memory-management optimizations, increased emphasis on regularization, and the ability to scale to large datasets. Our decision to implement XGBoost was inspired by the competition winner who had implemented a hybrid approach combining an XGBoost decision tree model with a deep neural network model. As deep neural networks is out of the scope of this course, we decided to just implement an XGBoost decision tree model. We implemented the XGBoost classifier using the Python xgboost module. We also conducted hyper-parameter tuning using the Sci-Kit Learn GridSearch method.

**2.3.2 Random forest model.** We implement yet another ensemble learning method - a random forest classifier. This model involves the creation of multiple decision trees where each tree is based on a randomly selected subset of the original feature set. The benefit of this model is that it avoids overfitting as the height of decision trees is low, thus improving its ability to generalize to unseen data. We implement our classifier using the RandomForestClassifier method from the sklearn.ensemble module.

We chose this model because the training set is imbalanced and we needed a way to minimize the risk of overfitting. We also saw randomization as a possible solution that improves upon the traditional decision tree model since decision trees are created using a greedy algorithm and may not achieve the global minima.

**2.3.3 Multiclass logistic regression model.** Finally, we implement a multi-class logistic regression model. The model is based on a “one-vs-rest” approach whereby four binary logistic regression models are created, one for each stance label. Each binary model takes the instances belonging to one class and treats the instances belonging to every other class as belonging to the same class. When classifying a new test instance, it is passed into each of the four binary models and receives a probability value. The resulting classification is determined by the binary model which yields the highest probability. We implement this approach using the Sci-Kit Learn LogisticRegression method and we also use the GridSearch method to find the best set of parameters.

Among all the traditional machine learning models learnt in this course, logistic regression seemed to be the most obvious one to use in this problem. The probabilistic element of the model is intuitive and is relatively less complex than neural network-based approaches.

3. Experimentation

References

Appendices

Appendix 1 – Distributions of SVD similarity values for each stance

|  |  |
| --- | --- |
|  |  |
|  |  |