Stance Detection on News Articles

Swapnil Taneja
swtaneja@eng.ucsd.edu

Yash Agarwal
y2agarwa@eng.ucsd.edu

Vamsidhar Kamanuru
vkamanur@eng.ucsd.edu

ABSTRACT

The project is based on finding out whether a given headline aligns with its corresponing article. Any given headline for a news article should align with its corresponding article. Any relationship between a headline and an article is potentially useful for identifying irrelevant context among the articles that can actually be published. News organizations and social media can utilize the results for improved and targeted information for audiences. Checking whether the headline is related or unrelated to the body can also be used as a first step towards detection of fake news.

1 INTRODUCTION

This paper explores a series of approaches for the alignment problem. The alignment problem is described as the task of identifying whether a headline is related to an article. We use the dataset provided in the Fake News Challenge[2] and explore the approaches in performing this binary classification problem. Specifically, our task is to identify a tuple (headline, article) as either "related" or "unrelated". We modify the current data so as to make it suitable for the task. In the methodologies section we explore feature engineering and also model selection . We find the accuracy of the models and document in the section below.

2 RELATED WORK

One way to look at the task at hand in text classification. In this field, [9] used a convolutional neural network to classify sentences. One can think of convolutional filters extracting different n-grams depending on the filter size which are known to be useful for text classification. One closely related task is stance detection in Twitter Tweets [7]. In this task, a stance target - e.g., a politician - is given and the goal is to estimate whether a given Tweet is in favor, against or neutral towards the given target. This is similar to deciding whether a given article is related or unrelated to a given headline. [8] tries to solve the Twitter task by using a bidirectional LSTM to read the tweet conditioned on the target. Revisiting Text classification in [10], apart from using shallow lexico-syntactic features and words (such as uni-grams, bi-grams, ngrams) demonstrated the gain in fake review detection performance by drawing from features based on Context Free Grammars (CFG) parse trees. Towards this approach they had experimented using four different production rules, each having their own advantages. [11] have explored fictitious opinions which are written to sway consumer judgment . One possible solution elucidated in [11] is to use the output of the Linguistic Inquiry and Word Count software tool as a feature

3 EXPLORATORY DATA ANALYSIS

3.1 Data Statistics

In our data , we have 49972 stances and 1683 articles available for train. Every stance is a dictionary that has the following keys - "Body ID", "Headline " and "Stance" . A typical stance looks like - *'Body ID': 712, 'Headline': "Police find mass graves with at least '15 bodies' near Mexico town where 43 students disappeared after police clash", 'Stance': 'unrelated'* . Articles are stored in a dictionary having Body ID for keys. Each Body ID corresponds to an article for the headline . Maximum characters in an article is 27579 . Maximum characters in a headline are 225. Minimum characters in a headline are 9 and minimum characters in a body are 38. An analysis on common words resulted in the following - Some headlines do not have any words in common with their body(article) . Maximum length of common words is 32 and minimum is zero. 10829 pairs of (headline,body) do not have any word in common.

Our test data has 25413 stances and 904 articles. Maximum number of characters in a headline is 245 and maximum number of characters in a body is 19815. Minimum number of characters in a headline is 9 and minimum number of characters in a body is 29. Maximum length of common words is 26 between a pair (headline,body) and minimum number of common words is zero. There are 6609 pairs of (headline,body) that have zero number of words in common.

The training data has 12587 data points that are related and test data has 6367 data points that are related. The data is cited here [2]. Therefore we can see that we have an unbalanced data set which means that we will have to use measures in addition to accuracy to evaluate the efficacy of our classifiers. For this purpose we will use the Balanced Error Rate (BER).

4 CLASSIFICATION TASK,EVALUATION METRICS AND FEATURES FOR SKLEARN CLASSIFIERS

Predictive Task : We have the 49972 data points for the training dataset and 25413 additional data points for testing. We have the corresponding output labels ("related"/"unrelated"). We chose to evaluate our models on two accuracy metrics - 1. *Accuracy* (percent correct classification) and *Balanced Error Ratio* (BER) .
Accuracy = Numerber of correct classifications/Total number of classifications.
BER = 1 - 0.5(TPR + TNR)
where,
TPR = TP/(TP+FN)
TNR = TN/(TN+FP)
TP = True Positives
FN = False Negatives
TN = True Negatives
FP = False Positives

We chose these metrics to get a holistic understanding of the results of our dataset. Now, we did evaluate our dataset and there are disproportionate number of unrelated and related labels. Hence, we also decided to have BER as a metric. Based on the work cited here [4], we allow their approach to serve as our baseline. They use Gradient Boosting Classifier to achieve a 75.20-79.53 percent accuracy on 4 stances.

Some of the properties of the data inspired our model. Since the number of data points having stance related have common words, we chose to introduce that as one of our features. . This involved removing stop words as well. We introduced a bunch of features after analyzing the data . The first is the binary occurrence of the tokens in the title. This feature calculates how many times a token (in the title) appears in the body text. After preprocessing the data, we were able to get the count of such tokens. Also, we added additional count of such tokens that appear early in the body text. It is possible that some tokens in text appear in the end of the body. Such tokens will not have much effect on the relatedness of head and body. In the next feature , we removed the stop words from the headlines and calculated the count of tokens (in headlines) in body . The next features involved removing stop words from the headline and body. We calculated the consecutive length of strings of specified length - 2, 4, 8 and 16 from the headline. And checked if these strings appear in the body. And incremented the corresponding count . In getting these features , we ignored the concept of words and simply calculated combinations of strings having broken words. In the next step, we also considered words into the picture. So , just the concatenation of words appear how many times in the body.

After concatenating all these features , we ran it on a number of classifiers from the sklearn library. The results are seen from the table 1

These features were also inspired by the baseline submitted on the challenge. [2]

5 DESCRIPTION OF MODELS

5.1 SkipThought

In this part, we generated feature vectors from the encoder of the Skip thought model referenced here [3]. We used their pre-trained model and generated 4800 dimensional vector for each sentence. So, for each headline we simply encoded the vectors using the model. For each body , we split the body into sentences and summed up the 4800 dimensions for each sentence in body. Assumption was that since skip thought vectors capture the relationships between sentences, they should provide latent representations for headline vs body. So, we then explored just the cosine similarity between headline and body. As it turns out this was time consuming and is really not scalable. It took more than 3 hours to generate 49972 X 4800 dimensional vector . And due to their large size, we faced Memory crash. After dealing with these issues, we ran the simple similarity based model and achieved an accuracy of 74.85 percent. This is because the predictor turns out to be a trivial predictor. The reason for lower accuracy is that when we sum up the vectors for sentences in a body, we lose semantic meaning to some extent. Also, not having many sentences in headlines provides us less information about the headlines. There can be improvements in this method. We can concatenate the headline and body features to make a 49972 X 9600 matrix. And feed it to the Classifiers listed in table 1.

Table 1: Performance of classifiers on test data

Classifier	Accuracy	BER(Balanced Error Ratio)
Gradient Boosting	0.9231889	0.1041
SVM	0.923346	0.1067
AdaBoost	0.921221	0.1131
ExtraTreesClassifier	0.91547633	0.1149
RandomForestClassifier	0.91850627	0.1109
DecisionTreeClassifier	0.8916302	0.1556
LogisticRegression	0.92173297	0.12069
VotingClassifier	0.9221658206	0.11351

Classifier: On the general features, we ran a bunch of classifiers. The reason we chose to use these classifiers is because of linear separability in the data. Higher the relationship among the headlines and bodies, higher the separation of data points from the rest. Therefore, SVM classifier, Logistic Regression were the first ones we tried. Each of the features, we generated serve as attributes for Random forest and Decision Trees and can provide better separation guidelines at each level. Which is why AdaBoost and Gradient boosting ensembles of weak learners perform good . Lastly , we added a Voting Classifier for all the above stated classifiers and observed the performance.

5.2 SGD Classifier

In this part we computed the following features:
1. Term Frequency(TF) for the headline
2. Term Frequency for the body
3. TFIDF for the headline
4. TFIDF for the body
5. Cosine similarity between TFIDF of the body and the headline.
and ran SGD Classifier on them.

All of the above features are computed using scikit-learn. TF is calculated for the top 8000 frequently appearing words in the training set and a set of stop words are excluded. TFIDF are calculated on this top 8000 occurring vocabulary on the training set. Finally TF and TFIDF resulted in 16001 feature vector which are fed to the models for classification.
Then we implemented SGD Classifier using scikit-learn with Hinge loss function which outputs a Linear SVM classifier to classify the data 'Relevant' and 'Irrelevant' Stances. For SGD Classifier and MLP , we chose 50 percent of test data for validation and the rest for test.

Hyperparameter tuning: TF vectors are calculated with different number of frequently appearing words in the training set. Choosing 5000 most frequently appearing words made the computation time optimal.

The SGD Classifier is trained with a range of regularization constants. A regularization constant of 0.0001 gave the least Classification error on the validation set.

Table 2: SGD Classifier:Performance on Validation Set

Accuracy	BER(Balanced Error Ratio)
0.9152	0.08961

Validation Accuracy:
The validation accuracy(classification accuracy) is 0.9152

The SGDClassifier is trained with a range of regularization constants. As evidenced by Figure 1 showing the SGD Classification Error with respect to the regularization constant,a regularisation constant of 0.0001 gave the least Classification error on the validation set.

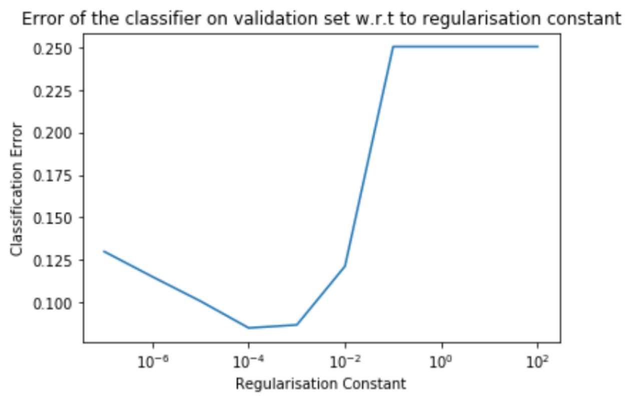

Figure 1: SGD Classification Error with respect to the regularization constant.

Table 3: SGD Classifier:Performance on Test Set

Accuracy	BER(Balanced Error Ratio)
0.9046	0.09839

5.3 Multi layer Perceptron

We use the same features as we did for Stochastic gradient Descent in our Multi Layer Perceptron Classification. In the paper [5] , a simple Multi Layered Perceptron model is explored. We use their model and features to see how well it performs on our dataset. The MLP has a single hidden layer with 70 neurons. Relu activation is kept. In the paper, they explore four stances namely -discuss, agree, disagree, unrelated. In our model, we keep the softmax layer but mapping the outputs to just "related" or "unrelated". We do this by mapping "disagree" and "unrelated" to "unrelated" And "related" to every other stance. Figure 2 shows the MLP Architecture for stance detection. Figure 3 is a graph showing the loss with the number of iterations.

Table 4: MLP Classifier:Performance on Validation Set

Accuracy	BER(Balanced Error Ratio)
0.9285	0.08399

Table 5: MLP Classifier:Performance on Test Set

Accuracy	BER(Balanced Error Ratio)
0.9216	0.08931

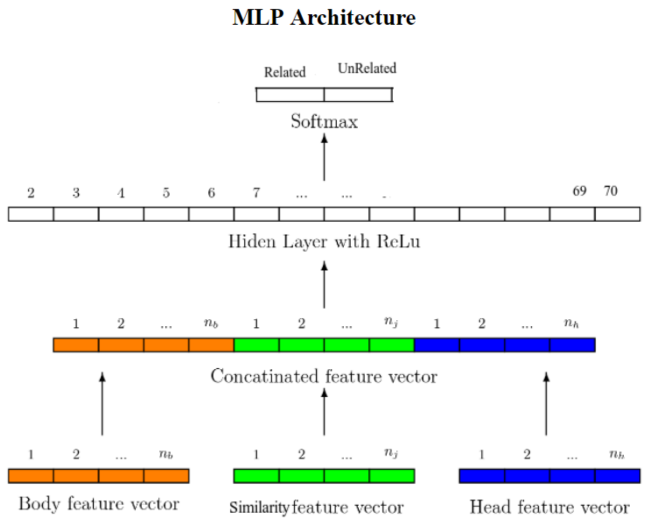

Figure 2: MLP Architecture for stance detection

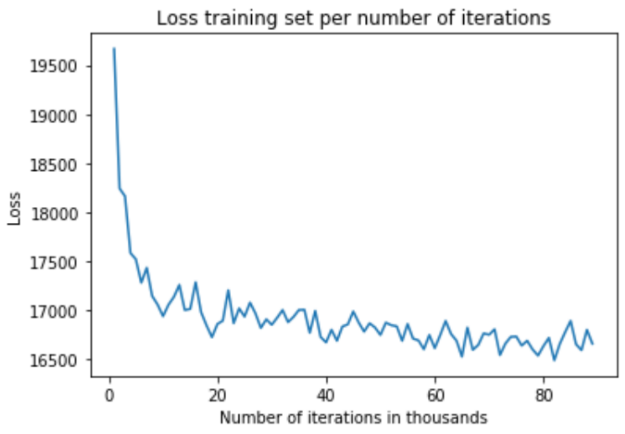

Figure 3: Loss with number of iterations

Table 6: Latent Semantic Analysis: Training Set

Optimal Threshold	Accuracy	BER(Balanced Error Ratio)
0.18	0.902705515088	0.1640461569606586

5.4 Latent Semantic Analysis

Latent Semantic Analysis, or LSA is a technique used to analyze documents so as to understand the underlying meaning or concepts of those documents. It is a very powerful technique with respect to our task here as we can use it to analyse the underlying meaning of the headline and body and then compare the extracted meanings to check the stance. LSA accomplishes this by mapping documents and the important words in those documents into a concept space and then performing the comparison in that concept space.

STEP 1 in Latent Semantic Analysis is to create the word by document matrix. In this matrix, each index word is a row and each title is a column. Each cell contains the number of times that word occurs in that title.

STEP 2 Modify the Counts with TFIDF. The raw matrix counts are modified so that rare words are weighted more heavily than common words. This is done using TFIDF (Term Frequency -Inverse Document Frequency). Under this method, the count in each cell is replaced by the following formula.

TFIDFi,j = (Ni,j / N*,j) * log(D / Di) where

Ni,j = the number of times word i appears in document j (the original cell count).

N*,j = the number of total words in document j (just add the counts in column j).

D = the number of documents (the number of columns).

Di = the number of documents in which word i appears (the number of non-zero columns in row i).

STEP 3 Performing Singular Value Decomposition (SVD). We perform SVD to find a reduced dimensional representation of our matrix that emphasizes the strongest relationships and throws away the noise. The count matrix A is decomposed such that

A=USVt

The U matrix gives us the coordinates of each word on our "concept" space, the Vt matrix gives us the coordinates of each document in our "concept" space, and the S matrix of singular values gives us a clue as to how many dimensions or "concepts" we need to include.

STEP 4 Evaluate Cosine Similarity We then proceed to find the cosine similarity between the vector representations of the headline and body in Vt.

STEP 5 Find Threshold. The threshold for similarity which will determine whether the stance is related or unrelated, is computed using the training set. The above 4 steps are run over the training set for different threshold values. We obtained 0.18 as the optimal threshold value which gave the highest Training Accuracy and lowest Training BER.

Strengths of LSA

1. LSA extracts the underlying concepts and meaning of documents in terms of vectors whose elements describe the significance of the important words extracted from documents. This gives us a quantifiable way of looking at a document and comparing it with other documents.

2. The space into which we map the documents and important words has vastly fewer dimensions compared to the original matrix. Not only that, but these dimensions have been chosen specifically because they contain the most information and least noise. This makes the new concept space ideal for running further algorithms.

Weaknesses of LSA

1. LSA does not handle polysemy very well. It does not take into consideration that the same word can have different meaning in different situations.

2.LSA assumes a Gaussian distribution and Frobenius norm which may not fit all problems.

Table 7: Latent Semantic Analysis: Performance on Test Set

Optimal Threshold	Accuracy	BER(Balanced Error Ratio)
0.18	0.8768740408452	0.193171996543

6 CONCLUSIONS

Using accuracy as the performance metric we conclude that SVM gives the best results on general features. While taking the BER as the performance metric, we get the best results from the Multi Layer Perceptron architecture. Also additionally, MLP gives nearly the same accuracy. MLP gives 0.9216 accuracy while SVM gave 0.923346 accuracy. But the BER of the MLP Classifier is better, 0.08399 compared to the 0.1067 of SVM. Therefore we can conclude that MLP and SVM perform the best. We believe the reason for the better accuracy on SVM is because of the linear separability in the features. SVM performs because data points are linearly separable. It can be argued that the classifiers listed in table 1 give great accuracy as the data is linearly separable. And MLP gives us the best BER because the hidden layer capture the notion of data being disproportionate. LSA features provide a better accuracy than baseline and BER from a simple similarity based model. From the model parameters, we understand that lambda lower than 0.0001 gives us higher complexity which is why error rate is high . And higher lambda gives us higher error as the model can not capture the right weights. Skip thought vectors and their cosine similarity do not give great results and the reason for that is skip thoughts are very high dimensional vectors (4800). Cosine similarity of lower dimensions may capture better features than the cosine similarity of very large dimensions.

A REFERENCES

1. sklearn - http://scikit-learn.org/stable/modules/classes.html
2. Fake News Challenge - http://www.fakenewschallenge.org/
3. SkipThoughts - Ryan Kiros , Yukun Zhu ,Ruslan Salakhutdinov , Richard S. Zemel , Antonio Torralba ,Raquel Urtasun , Sanja Fidler
4. baseline fnc - https://github.com/FakeNewsChallenge/fnc-1-baseline
5. MLP - Benjamin Riedel1 , Isabelle Augenstein12, Georgios P. Spithourakis1, Sebastian Riedel1
6. LSA - https://technowiki.wordpress.com/2011/08/27/latent-semantic-analysis-lsa-tutorial/
7. S. M. Mohammad, S. Kiritchenko, P. Sobhani, X. Zhu, and C.

Cherry, "Semeval-2016 task 6: Detecting stance in tweets," Proceedings of SemEval, vol. 16, 2016.
8. I. Augenstein, T. Rocktaschel, A. Vlachos, and K. Bontcheva, "Stance detection with bidirectional conditional encoding," arXiv preprint arXiv:1606.05464, 2016.
9. Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014
10. Song Feng, Ritwik Banerjee, and Yejin Choi. Syntactic stylometry for deception detection. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pages 171âĂŞ175. Association for Computational Linguistics, 2012.
11. Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T Hancock. Finding deceptive opinion spam by any stretch of the imagination. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pages 309-319. Association for Computational Linguistics, 2011.