INFORMATION RETRIEVAL AND DATA MINING STANCE DETECTION PROJECT

NIREN PATEL

This paper looks at using techniques learnt from the Information Retrieval and Data Mining (COMPGI15) to detect the stance between pairs of Headlines and Article Bodies in news articles. Similarity measures such as that of Cosine and KL Divergence are calculated, and combined with Machine Learning Techniques to decide whether Body Articles; 'agree with', 'disagree with', 'discuss' or are "unrelated to" their respective headlines.

Niren Patel
UCL Data Science and Machine Learning MSc

Literature Review

Fake news has become rampant in the media. Building models to detect fake news can separate reputable sources from poor ones. Poor sources can often waste the time of an individual, whilst in some cases may lead to deception, political or financial issues. [1] Cosine similarity [2] and KL Divergence [3] are two prominent methods for comparing the similarities/differences between texts and are explored further in this paper. With the influx of more data, these models have been refined and become more powerful in clustering texts into distributions for machine learning algorithms to classify. [4]

Data Exploration

To begin with, four files were provided for this project, this included:'train_stances', 'train_bodies', 'test_stances' and 'test_bodies'. These datasets were first combined based on their matching ID's so there were effectively only two data frames; which would make the data more manageable and so bodies and headlines could be compared with more ease. The combined training dataset was then split into two datasets for hyperparameter tuning purposes; the split resulted in a 9:1 ratio for Training:Validation. The ratios of the different stances are shown in Table 1. As can be seen from Table 1, the proportion of different stances for each dataset is very similar. There is a clear class imbalance between the 'Unrelated' stance and the other three stances; suggesting it will be easier to predict 'Unrelated' stances due to their volume.

Table 1 - Ratio Statistics for Training and Validation Datasets for the four different stances.

Stance	Training Dataset	Validation Dataset
Agree	7.395%	7.044%
Disagree	1.659%	1.881%
Discuss	17.823%	17.871%
Unrelated	73.123%	73.204%

Vector Representation and Cosine Similarity

Before Vector Representation and Cosine Similarity was performed, it was imperative that strings in 'Headline' and 'articleBody' fields were made lowercase. This is because 'Regular Expression' functions in Python, generally count words of different casing as individual entities. Once Vectors had been created, Equation 1 was used to compute the Cosine Similarities between the pairs of Headlines and Vectors. Where, for two vectors A and B, the numerator represents the dot product of A and B, and the denominator represents the multiplication of the magnitude of the two vectors. The average Cosine Similarity for each stance for the training data is shown in Table 2. A lower value for Cosine Similarity, represents a lack of words that occur in both the headline and article body pairs.

In addition to this, 'stop-word removal' and 'stemming' was also performed to gauge what effect this could have on Cosine Similarity. The dictionaries associated with these techniques were downloaded from the NLTK Python package. The results are also shown in Table 2. As shown in the table, stop-word removal has a noticeable effect on the unrelated stance, almost reducing Cosine Similarity to 0. This is expected as, stop-words generally provide little relevance and provide little meaning in the context of news articles. The removal of stop-words also helped in making average Cosine Similarities more distinct for the other stances. The introduction of 'stemming' however, had very little impact; this is most likely to be the result of headlines and article bodies having few ordinary words and instead containing proper nouns that don't have related synonyms. For the rest of this paper, only text that has undergone 'stop-word' removal will be considered. Furthermore, TF-IDF (Term Frequency – Inverse Document Frequency) was not considered for the weighting of terms in Cosine Similarity calculation, because the texts after stop-word removal were likely to be of similar proportions.

$$Cosine\ Similarity = \mathrm{Cos}(\theta) = \frac{A \cdot B}{|A|\,|B|} \qquad (1)$$

Table 2 – Table showing the effects of different text-transformation techniques on average Cosine Similarity for Training Dataset

Stance	Regular	Stop-word Removal	Stop-word Removal and Stemming
Agree	0.253	0.294	0.280
Disagree	0.243	0.276	0.261
Discuss	0.243	0.296	0.282
Unrelated	0.097	0.0154	0.015

Language Model Representation and Kullback-Leibler Divergence

A unigram approach was used to model the language for this project. This is because, in text comparison problems such as the one posed by this project, bigram models are less likely to be effective for comparing similarities between two different sized texts, especially due to the briefness of headlines. The effectiveness of a Bigram Model, and other N-gram models, for short queries are more effective in applications that require the prediction of the next word. e.g. word suggestions when typing on mobile. [5] However, when comparing the texts of two large documents, N-gram models may be a powerful tool in text comparison.

Language Model

To obtain a Unigram model, punctuation was first removed from both the headline and article body pairs across all datasets. The text was then tokenised to create a "Bag-of-Words" model. This was achieved by using the tokenization package present in the NLTK python package. Before KL-Divergence could be performed, a probability distribution model for the unique vocabulary of words that occur in both headline and articles needed to be created. One representing the likelihood of a query given a document (query-likelihood model)

and one representing the likelihood of a document given a query (document-likelihood model). However, since article bodies are unlikely to contain all the unique words in a headline, and vice-versa; smoothing needs to be performed on both texts to account for probabilities that are not present in both texts.

Smoothing Techniques

Four different smoothing techniques were exhausted for this process. Line-search was performed across more sophisticated smoothing models, and the effectiveness was based on average KL-Divergence scores across different stances. Efficiency in compute time was also considered.

Laplace Smoothing

Firstly, Laplace Smoothing was considered. The equation used for Laplace Smoothing is shown in Equation 2. Where text represents the text to compare against, w represents the word, which probability is needed to be calculated. For this technique each unique word in the vocabulary of the two texts is given a count of 'one' regardless of whether it appears in both texts or not. The word is then counted in the text and given a count, based on the number of times it occurs in a document. The occurrences are then divided by the total text size and unique vocabulary words. The performance of this smoothing technique, as well as the others used are discussed in the smoothing performance section.

$$P(w|text) = \frac{tf_{w,text} + 1}{|text| + |V|} \qquad (2)$$

Discounting (Lidstone Correction)

Since Laplace smoothing is thought to give too much weight to "unseen" terms (terms that don't occur in both documents, but only appear in one). A similar approach to Laplace smoothing, is discounting, which adds a smaller number, ε, instead of 1. The term, ε, requires tuning, usually between the value of 0 and 1, then is normalised to a probability. [6] The equation for this method is shown in Equation 3.

$$P(w|text) = \frac{tf_{w,text} + \varepsilon}{|text| + \varepsilon|V|} \qquad (3)$$

Jelinek-Mercer Smoothing

Jelinek-Mercer Smoothing is an interpolation smoothing method that prevents uniform probabilities of unseen terms, unlike that of the previous two smoothing terms mentioned. This provides a more accurate depiction of the likelihood of terms occurring, but also includes probability of the word occurring in the whole collection. λ must be tuned to find an optimal value that gives a probable weight to each term, so the distribution sums to 1. The equation for this technique is shown in Equation 4. It is important to note: that this discounting method essentially considers TF-IDF as it also takes into the likelihood of the term appearing in the collection.

$$P(w|text) = \lambda * \frac{tf_{w,text}}{|text|} + (1-\lambda) * \frac{tf_{w,collection}}{|collection|} \qquad (4)$$

Dirichlet Smoothing

Dirichlet Smoothing was also considered as another interpolation smoothing method for preventing uniform probabilities of unseen terms. This is shown in Equation 5, Dirichlet smoothing, is essentially an extension to Jelick-Mercer Smoothing, with λ being expanded; so, the term is also dependent on the text size (sample size). Where N represents the size of the sample. This method also considers TF-IDF, because it considers the likelihood of the term occurring in the collection.

$$P(w|text) = \frac{N}{N+\mu} * \frac{tf_{w,text}}{|text|} + \frac{\mu}{N+\mu} * \frac{tf_{w,collection}}{|collection|} \qquad (5)$$

Smoothing Performance

All four of these smoothing techniques were applied to both the headline (query) and article body (document) pairs, to obtain the query-likelihood and document-likelihood distributions.

The most reliable results were achieved with Laplace and Discounting methods. This is most likely to be the consequence of lack of hyperparameter tuning. For the interpolation methods, since there are many queries and document with sizes of varying proportions, it becomes too difficult to just tune one hyper-parameter (even with normalisation after determining initial distribution values). Consequently, each document and headline impression would require its own tuned parameter to be effective; which would increase the compute time by k*N (with k representing the number of individual impressions, and N; the number of hyperparameter values to iterate over). Furthermore, probabilities can often be miniscule for terms that don't appear often, and the logarithm values should be taken. Therefore, it can be concluded that Dirichlet and Jelinek-Mercer smoothing methods are more suited to problems which involve ranking documents. Nonetheless, all smoothing techniques and their best optimised performances are shown in Table 3; based on how well they separate stances on average KL divergence, for the training set. The equation for KL divergence is given in Equation 6: where Mq represents the Query Likelihood Model and Md represents the Document Likelihood Model. [7]

Initially, one would have expected KL Divergence to have a higher score for unrelated, since the distribution of related words would mainly be unique, and a KL Divergence score of 1 generally represents models with completely different distributions, whilst a score of 0 represents identical distributions. However, intuitively having a KL divergence where the average KL divergence is lower in

comparison to other stances, makes sense because both Laplace and Discounting as smoothing techniques assign uniform probabilities, for 'unseen' terms. That of which, the unrelated Stance is abundant in. It is therefore, advantageous to use Laplace smoothing KL Divergence Values, since they provide the best distinction between stances, in comparison to other smoothing methods.

$$KL(M_d \| M_q) = \sum_{t \in V} P(t|M_q) \log \frac{P(t|M_q)}{P(t|M_d)} \quad (6)$$

Table 3 – Table showing the average KL Divergence for different stances of the Training Dataset for different smoothing techniques

Stance	Laplace Smoothing	Discounting (Lidstone Correction)	Jelinek-Mercer Smoothing	Dirichlet Smoothing
Agree	0.0364	0.0144	0.0131	0.0155
Disagree	0.0389	0.0159	0.0135	0.0158
Discuss	0.0445	0.0173	0.0154	0.0164
Unrelated	0.0015	0.0078	0.0110	0.0130

Alternative Distances

The Euclidian distance was proposed as an alternative feature, where there was intersection between vector words. The results are shown in Table 4 in comparison to Cosine Similarity as a feature, as well KL Divergence with Laplace smoothing as a feature for the training set. The Euclidian Distance is computed by finding the square root of the sum of squared distances between corresponding terms that appear in both vectors. The Euclidian Distance was chosen as an alternative feature as it is generally a powerful evaluation tool, when comparing words that appear more frequently than others in their corresponding texts. However, it doesn't consider the length of documents. [6]

Table 4 - Table comparing the average scores for different distances against the different stances on the training Set

Stance	Cosine Similarity	Laplace Smoothing	Euclidian Distance
Agree	0.294	0.0364	18.593
Disagree	0.276	0.0389	19.864
Discuss	0.296	0.0445	21.382
Unrelated	0.015	0.0015	20.987

Two Most Important Chosen Features

For logistic and linear regression, it was decided that Cosine Similarity and KL Divergence would be selected as they had the best performance in distinguishing between classes (based on Table 4), in comparison to Euclidian Distance. Machine Learning models described in the next two sections are more likely to perform more effectively for these two features; as machine learning models make generalisations based on the distribution of points against features. Figure 1(shown on final page) shows the distribution of the different training set impressions with different classes. It is clear from the graph, most impressions belonging to the unrelated class, form a strong boundary layer against all other stances, with data focused more towards the lower left corner of the axis. This suggests that the individual prediction accuracy for the Unrelated class will be higher than other classes. For the other stances, machine learning models may struggle. However, as Table 4 suggests, differences in the feature values between classes may help models.

Logistic Regression

A One Vs. All approach was taken for using logistic regression for this multi-class detection problem. Dummy features were created for each stance; creating four individual binary classification problems (one for each class). Each of these four binary classifiers were trained on the training set, whilst hyper-parameters were tuned on the validation set to increase accuracy and reduce error. Each binary classifier would then make probability predictions on the test set for each impression of being the class in question. For each impression, the highest probability of the four predictions from the models would be taken and be given the corresponding predicted class. These classes would then be compared to the actual test class and an accuracy score would be given based on correct predictions. For this task, the tutorial found at [8] was used to help me implement the model into python. The model was altered considerably, to meet the demands of this project e.g. removal of cross-validation function and removal of functions to predict probabilities.

Sigmoid Function

$$p(class = 1) = \frac{1}{1 + e^{-(W_0 + W_1 KLDiv + W_2 CosineSim)}} \quad (7)$$

Equation 7 shows the function used to predict the probability of an impression belonging to the positive class. The Sigmoid Activation Function is used to introduce non-linearity to the model and is effective, as it can only produce outputs that are between 0 and 1. Hence, why it is the standard for logistic regression models. W_0 represents the bias weight, whilst $W_1$and W_2 represent the weights for the features, KL Divergence and Cosine Similarity, respectively. To obtain optimal weight values, the optimisation method, gradient descent is used.

Stochastic Gradient Descent

Stochastic gradient descent was used to optimise the weight coefficients. Two parameters, learning rate and epochs, were the two hyperparameters to tune. The learning rate is used to reduce the amount by which a coefficient is corrected, every time it is updated, whilst, the epochs specifies how many times to loop over the training data. [8] This technique works because, it works to minimise the

error until convergence occurs. Smaller learning rate values perform more favourably, however, they increase compute time drastically. A higher number of epochs is usually beneficial to reducing the error. However, an epoch value too large can cause issues such as overfitting to the validation set.

The Effect of Learning Rates and number of epochs

Grid-Search was performed across a combination of different learning rates and epoch numbers. Table 5 shows the effects of learning rate on accuracy for 100 epochs. Learning rate has very little effect on accuracy, however, for larger values of learning rate, it is likely that accuracy would decrease drastically. The performance of the logistic regression model considered Mean Squared Error, in addition to the learning rate, to determine optimal hyper-parameters for testing purposes.

Table 5 - Table showing the effect of learning rate on different binary classification models (100 epoch sub sample)

Model	Stance: Agree	Stance: Disagree	Stance: Discuss	Stance: Unrelated
Learning Rate = 0.01	92.90%	98.12%	88.07%	96.52%
Learning Rate = 0.05	92.68%	98.12%	88.07%	96.64%
Learning Rate = 0.1	92.66%	98.12%	88.11%	96.70%
Learning Rate = 0.5	92.76%	98.12%	88.10%	96.88%

Evaluation of Logistic Regression Results

The logistic regression model had an accuracy of 86.53% correct classifications, this is a suitable metric since the task is only looking at classifying. The value of 85.53% exceeds a 'benchmark' of predicting all impressions as Unrelated, which would have achieved an accuracy of around 73%. This model could still be improved, but new 'unseen' data would be needed, to prevent inevitable overfitting, to the test data that has already been predicted on. Both the Cosine Similarity and KL Divergence have been moderately successful in predicting stances, due to the clusters they form.

Linear Regression

Multiple linear regression was also performed, using a One Vs. All Technique; like the aforementioned logistic regression model. The tutorial followed for this model can be found in [9].

Linear Function

Equation 8 shows the function to calculate the continuous value. Linear regression isn't expected to explicitly produce a probability value from 0 to 1, unlike the sigmoid function. However, because all target/output values are either 0 or 1, a predicted probability is obtained. Gradient Descent will once again be needed to optimise the values of the weights, W_0 to W_2, to reduce the Mean Square Error.

$$p(class = 1) = W_0 \ + \ W_1 * KLDiv \ + \ W_2 * CosineSim \quad (8)$$

Stochastic Gradient Descent

Stochastic Gradient Descent was also used to reduce the Root Mean Square Error (RMSE) as low as possible. Once again, Grid Search was used to obtain the best hyperparameters for use in the test set.

The Effect of Learning Rates and number of epochs

For the linear regression, both the number of epochs and learning rate had very little effect on the RMSE for all the binary linear classification models. Changes were only noticeable in orders of magnitudes after four decimal places. Therefore, the learning rate was set to 0.001 and the number of epochs was set to 350 for all models.

Evaluation of Linear Regression Results

The linear regression model achieved an accuracy of 85.17%; this is suitable as an evaluation metric because the task is only looking at classifying, rather than predicting the probability of something occurring. This model also performs better than a benchmark prediction. However, the accuracy result of the linear regression model being close to that for the logistic regression model, suggests the logistic regression model was not tuned rigorously enough. As generally, logistic regression models perform better than linear regression for classification problems.

Importance of Features

Both KL Divergence and Cosine Similarity are of similar importance to the models. However, if more compute power was available, Jelinek-Mercer or Dirichlet smoothing may have been used to compute KL divergence. Furthermore, TF-IDF weight addition to the cosine similarity calculations may have led to more desirable stance prediction. However, a score of 86.53% on the most basic of regression models, exemplify the potential and importance of the features.

Proposing of Improvement of Machine Learning Models

To improve the machine learning models, numerous ready-to-use packages are available from recognised libraries such as Scikit-learn and gradient boosting (XGBoost). Ready-to-use models would have allowed the comparison of multiple models for a fraction of the time. Furthermore, to prevent the overfitting that was likely occurring on the validation set, k-fold cross validation could have been used instead, to produce more robust and reliable training models. Most importantly. Allowing sentiment analysis of words, would have allowed more deeper predictions from the three 'agree', 'disagree' and 'discuss' stances, as it is possible some toxic words, are prevalent

in articles that are of the stance ‘disagree’. Although, implementing these models manually, give a deeper understanding of how these processes work. It would now be more efficient to use automatic built in libraries for functions such as TF-IDF. Furthermore, instead of using One Vs. All, there are models that can provide multiclassification, saving further time.

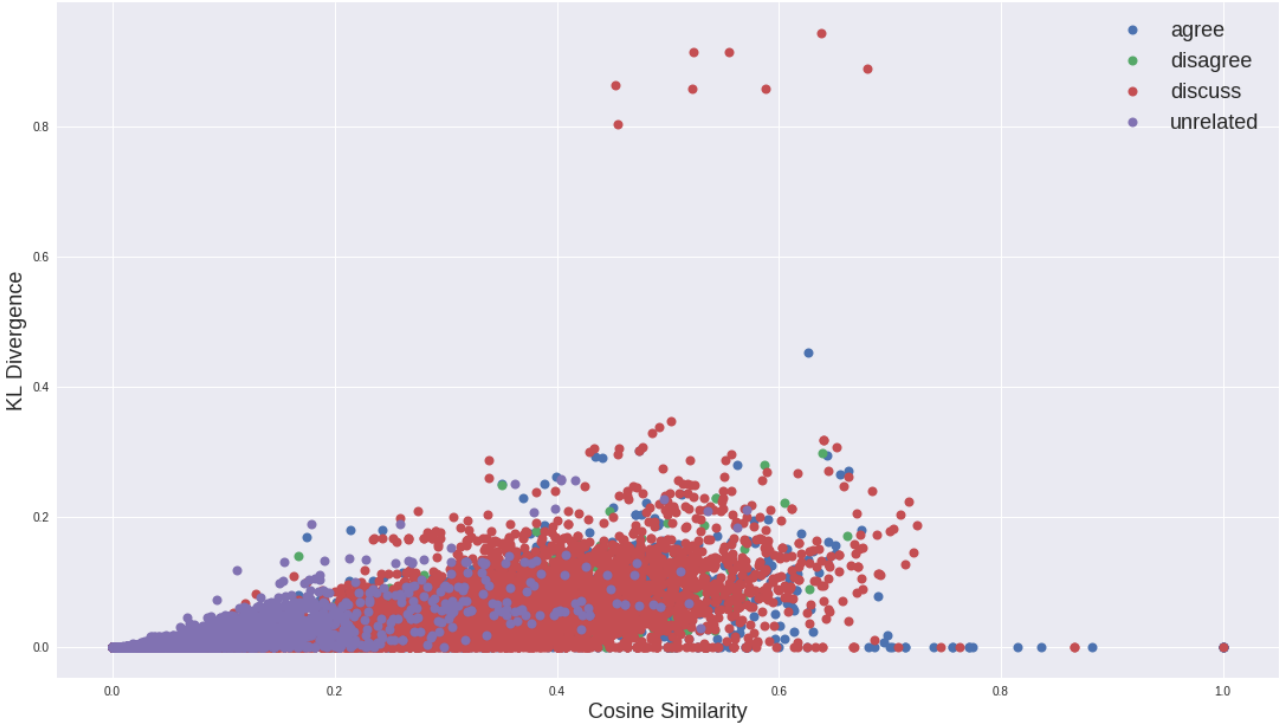

Figure 1 – Distribution of different stances plotted against KL Divergence and Cosine Similarity

References

[1] Chai XW. Fake News Stance Detection. Stanford

[2] Huang A. Similarity measures for text document clustering. In Proceedings of the sixth New Zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand 2008 Apr 14 (pp. 49-56).

[3] Dhillon IS, Mallela S, Kumar R. Enhanced word clustering for hierarchical text classification. InProceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining 2002 Jul 23 (pp. 191-200). ACM.

[4] Baker LD, McCallum AK. Distributional clustering of words for text classification. InProceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval 1998 Aug 1 (pp. 96-103). ACM.

[5] Lesher GW, Moulton BJ, Higginbotham DJ. Effects of ngram order and training text size on word prediction. In Proceedings of the RESNA'99 Annual Conference 1999 Jun 25 (pp. 52-54).

[6] Lidstone smoothing – Aiaioo Labs Blog [Internet]. Aiaioo.wordpress.com. 2018 [cited 3 April 2018]. Available from: https://aiaioo.wordpress.com/tag/lidstone-smoothing/

[7] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008

[8] 1. Brownlee J. How To Implement Logistic Regression With Stochastic Gradient Descent From Scratch With Python - Machine Learning Mastery [Internet]. Machine Learning Mastery. 2018 [cited 6 April 2018]. Available from: https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/

[9] 1. Brownlee J. How to Implement Linear Regression with Stochastic Gradient Descent from Scratch With Python - Machine Learning Mastery [Internet]. Machine Learning Mastery. 2018 [cited 7 April 2018]. Available from: https://machinelearningmastery.com/implement-linear-regression-stochastic-gradient-descent-scratch-python/

[10] Conroy NJ, Rubin VL, Chen Y. Automatic deception detection: Methods for finding fake news. Proceedings of the Association for Information Science and Technology. 2015 Jan 1;52(1):1-4.