1 NTLK Setup and Design Choices

This project uses Python 3.7, along with the NLTK library [1]. The installation instructions can be found for whichever platform is being used in the reference; once installed, enter a python console and execute the following:

```
>>> import nltk
>>> nltk.download()
```

A window should appear (after a short delay), with a list of downloads. For Windows, change the download directory to "C:\nltk_data", for Mac to "/usr/local/share/nltk_data" and for Linux to "/usr/share/nltk_data". Select all the packages and press "Download".

The data is parsed using a custom function, and fed into three lists of dictionaries: TrainSet, ValSet and TestSet. Each dictionary therein contains the full text of the headline and body (which from here we shall refer to as the query and document, respectively, in accordance with literature). Dictionaries were chosen rather than lists, due to the $\mathcal{O}(1)$ lookup time, and because in Bag of Words (BoW) representations of documents the order of words is not considered.

With word stemming performed by the Lancaster algorithm of the NLTK library, preprocessing can take up to 20 minutes; for this reason, during testing, a directory of the objects which are most time-consuming to create was used, which could be loaded using by switching the "Test" parameter in the "Initialise" function to True. The size limit on submissions has made this feature impossible, so all dictionaries must be built. Preprocessing includes finding counts for all unique words both for each individual document and query and globally, as well as finding the inverse document frequency for every unique word.

Numpy vector operations were used which gave a considerable speed boost to gradient descent and calculating the cost function for regression.

2 Language Models, Vector Spaces, Distances and Regression

The first challenge was to set up vector representations of the queries and documents in order to ascertain whether or not they were related. There are several possible metrics, to choose from; cosine similarity and KL-divergence were chosen, in part for simplicity and in part for their excellent performance and low cost.

2.1 Cosine Similarity

In the BoW representation of a string, each unique word is assigned an index in a vector and the number of occurrences of each word is recorded at their index. If the query vector is $\mathbf{u}$ and the document vector is $\mathbf{v}$, the cosine similarity $\cos\theta$ is:

$$\cos\theta = \frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}$$

The mean results on the training set are summarised in Table 1. Even from this extremely simple measure, the difference between values for related and unrelated query-document pairs is stark; there is no distinction by this metric, however, between 'agree', 'disagree' and 'discuss'. A plot of the distributions can be found in Fig. 2.1; The extremely irregular pattern for 'disagree' is due to the low number data available for that category.

2.2 KL Divergence

KL divergence is a measure of the difference between two distributions. Fig. 2 shows the distribution of Dirichlet-smoothed KL divergences for the different stance categories in the training set. The same BoW vector representations used for cosine similarity were used here. Again, the difference between the distributions of unrelated and related query-document pairs is obvious, but there is nothing to distinguish 'agree', 'disagree' and 'discuss'. KL Divergence is defined as:

$$D_{KL} = -\sum_{w\in Q} P(w|\theta_Q)\log P(w|\theta_D)$$

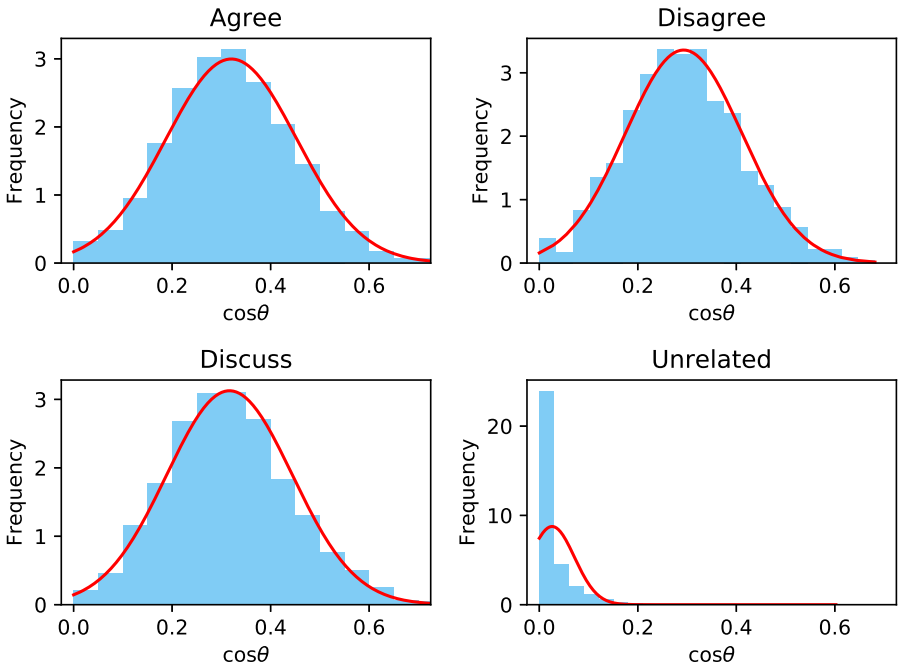

Figure 1: Histograms and best fit lines for cosine similarities

where θ_Q is the language model based on the query and θ_D is the language model based on the document. With Dirichlet smoothing this becomes:

$$D_{DSKL} = -\sum_{w \in Q} P(w|\theta_Q) \cdot \log\left(\lambda P(w|\theta_D) + (1-\lambda)P(w|\theta_C)\right)$$

$$\lambda = \frac{N}{N+\mu}$$

where a smaller document length N gives a larger weighting to the language model based on the collection, θ_C. μ is a parameter to be optimised; a larger μ gives a larger smoothing effect. $\mu = 1000$ was used here, which is relatively modest.

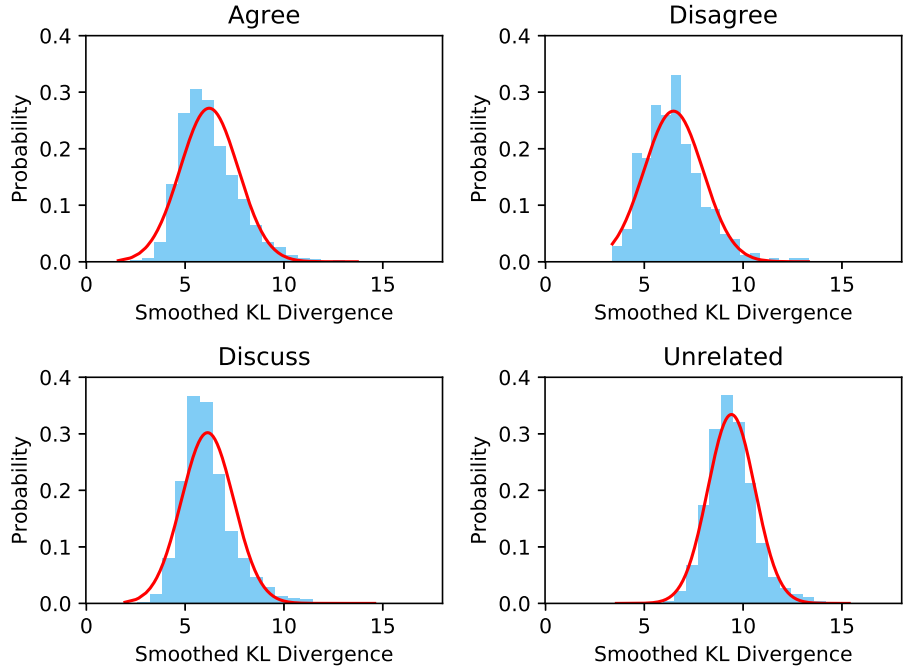

Figure 2: Histograms and best fit lines for Dirichelt-smoothed KL divergence

2.3 Query Likelihood

Query Likelihood is less frequently used than KL-divergence, however it is included here as a feature to be used in linear and logistic regression as it does provide some predictive power. Dirichlet smoothing similar to that described above was used here as well. A table of mean query likelihoods on the training set can be seen in Table 1.

Stance	$\cos\theta$	QL	BM25
Agree	0.3026	-53.32	24.45
Disagree	0.2806	-53.77	22.84
Discuss	0.3032	-52.03	21.28
Unrelated	0.0145	-63.16	0.862

Table 1: Cosine similarity, query likelihood and BM25 scores for training set.

2.4 BM25

BM25 is a popular BoW ranking algorithm which takes into account the inverse document frequency of each query term to give words which encode more information a higher weight. The specific form used in this project is:

$$\text{BM25}(D,Q) = \sum_{w \in Q} \text{IDF}(w) \cdot \frac{f(w,D)\cdot(k_1+1)}{f(w,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avdl}}\right)}$$

The mean BM25 scores are summarised in Table 1. *avdl* is the mean document length, and the parameter k_1 was set to the standard value of 1.2.

The features described above (cosine similarity, KL divergence, query likelihood and BM25 score), along with a vector of the counts for the query and document of the 2000 most common words in the corpus to create an overall feature space of $\mathbf{x}_i \in \mathbb{R}^{4004}$ were used as the feature vector for regression. Softmax multiclass regression with regularisation was also implemented.

In order to maximise efficiency whilst avoiding oscillation, a variable step length/learning rate has been used. Whenever the cost decreases from one step to the next, the training rate is increased; whenever the cost increases, the weights from the previous step are restored and the step length is decreased. This ensures that the local minimum is found in the least possible number of iterations. This was done using a fixed learning rate caused either oscillation (non-convergence, too high) or prohibitively slow convergence (too low).

2.5 Linear Regression

Binary linear regression was performed for each of the three stances. The labels assigned were either 1 for membership of the class (stance) under investigation or 0 for other classes. After finding the weights for individual binary classifiers, the weights from the classifier that gave the highest prediction were used to assign a class to each data point. Gradient descent was used to find the local minimum, and a series of random starting points along with measuring validation error was used to try to ensure the local minimum was also the global minimum - or at least the best local minimum in the region.

The bias term was learned implicitly by adding an extra column of ones to the feature vector. Confusion matrices with the predictions made by this model can be found in Fig. 3.

2.6 Logistic Regression

The problems that occur when using linear regression for classification can be tackled at least to some extent by changing to logistic regression. Here, the prediction is made with a sigmoid function and the cost function is modified to give a steep penalty for incorrect predictions:

$$h_\theta(\vec{x}) = \frac{1}{1+e^{-\theta^T\vec{x}}}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} y_i \ln(h_\theta(\vec{x}_i)) - (y_i - 1)\log(1 - h_\theta(\vec{x}))$$

Again, the standard method for multiclass regression was used. A confusion matrix and a normalised version are presented in Fig. 4

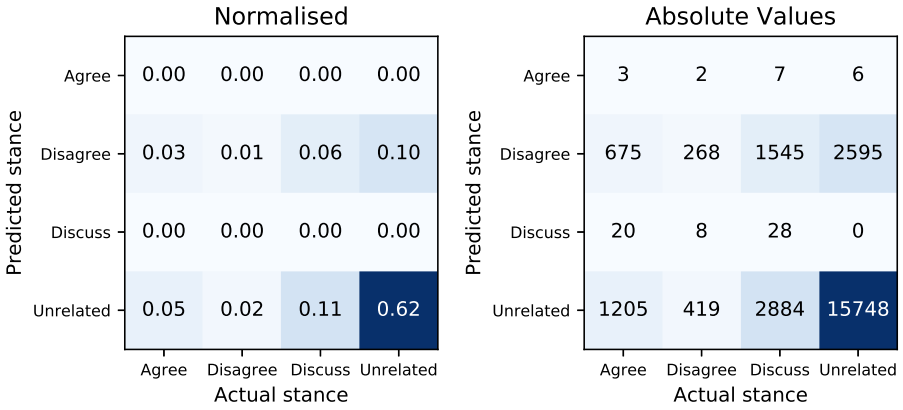

Figure 3: Absolute and normalised confusion matrices for linear regression with weights calculated from the training set and applied to the test set.

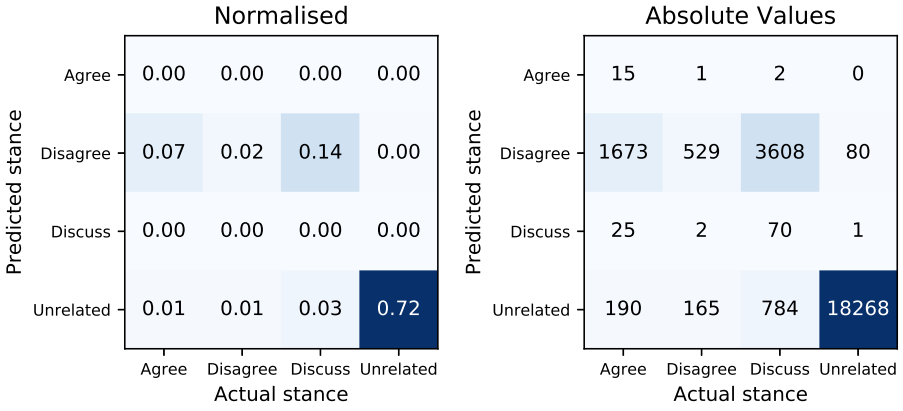

Figure 4: Absolute and normalised confusion matrices for logistic regression with weights calculated from the training set and applied to the test set.

2.7 Softmax Regression

A classifier more suited to multi-class classification is softmax regression. Here there are four outputs, each one signifying the probability that $\vec{x}_i$ is belongs to each of the four categories. We take the category with the highest probability. The relevant equations are:

$$h(\theta_i, \vec{x}_j) = \frac{e^{\theta^T \vec{x}_j}}{\sum_{k=0}^{3} e^{\theta_i^T \vec{x}_k}}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=0}^{3} \mathcal{I}(y_i, j) \cdot \log h(\theta_j, \vec{x}_i) + \frac{\lambda}{2} \sum_{i=1}^{n} \sum_{j=0}^{3} \theta_{ij}$$

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \vec{x}_i \cdot (\mathcal{I}(y_i, j) - h(\theta_j, \vec{x}_i))$$

where the stances are given labels 0, 1, 2 and 3. $\mathcal{I}(i, j)$ is a boolean comparison function which has the value of 1 if $i = j$ and 0 if $i \neq j$. λ is the regularisation parameter, because without it softmax regression is overparametrised, and its inclusion guarantees that the problem is convex (unlike in linear regression), so any local minimum found also a global minimum. The sum of the probabilities for each $\vec{x}_j$ over the four classes is equal to one. A confusion matrix for softmax regression is displayed on Fig. 5.

None of the regression models with the limited features performed well in terms of differentiating between the first three stances ('agree', 'disagree' and 'discuss'). However, they did manage to filter related from unrelated query-document pairs. This is to be expected, as all of the features chosen were similar for related articles.

The more important feature (between cosine similarity and KL divergence) is cosine similarity. This can be seen because the range of weights associated with cosine similarity is 0.0298; the range for KL divergence is only 0.0222. KL divergence values are typically 2 to 3 orders of magnitude higher than cosine similarities, a difference

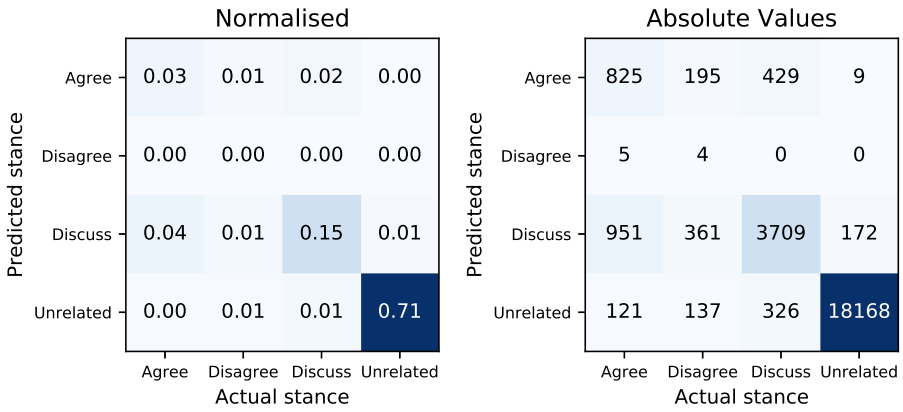

Figure 5: Absolute and normalised confusion matrices for softmax regression with weights calculated from the training set and applied to the test set.

Stance	Linear	Logistic	Softmax
Agree	0.00	1.64	43.4
Disagree	38.5	7.59	0.01
Discuss	0.01	0.02	83.1
Unrelated	85.8	99.6	99.0

Table 2: Percentage accuracy for different types of regression.

which is not accounted for by this small difference in weights, so we can conclude that cosine similarity is a more important parameter.

3 Literature Review and Improvements

A number of articles have been released concerning stance detection on this dataset. The consensus among these is that while sorting the related from the unrelated is trivial, in order to sort between 'agree', 'disagree' and 'discuss', some form of recurrent neural network (RNN) with multiple hidden layers is required. In the model developed by Mrowca and Wang [2], 100 Short-Term Memory (LSTM) units are used in a RNN with and a softmax top layer, optimised with respect to cross-entropy loss. The most features most important to the success of the model were reported to be so-called 'hand' features (word co-occurance, ngram and chargram counts). It achieved good scores for predicting 'agree' (67.7%), 'discuss' (81.4%) and 'unrelated' (97.6%) but still performed poorly for 'disagree' (31.3%).

This trend (poor performance for the 'discuss' class) was repeated across different research groups. A team from UCL [3] used a much simpler architecture - the BoW vector representations of both the query and the document, using a 5000 of the most common words in the English language, along with the TF-IDF cosine similarity were fed through just one ReLU hidden layer of 100 units to a linear layer and finally to a softmax top layer for the class probability output. This model managed accuracies of 44.04%, 6.70%, 81.38% and 97.90% for 'agree', 'disagree', 'discuss' and 'unrelated' respectively.

In both cases, the poor performance for 'disagree' was partly accounted for by the extremely low number of articles with that label, resulting in there not being enough training data for an accurate model.

Linear models are not complex enough to capture the extremely complex relationship between words and sentiment. Regression could be improved by implementing non-linear regression with a Gaussian or polynomial kernel and a non-linear SVM would also likely be able to determine whether or not a query and document are related. To really get useful results it would be necessary to implement a RNN, for example LSTM, with a softmax top layer giving a probability for each class.

References

[1] NLTK Natural Language Processing Library. https://www.nltk.org/data.html, 2018.

[2] Damian Mrowca, Elias Wang, and Atli Kosson. Stance detection for fake news identification.

[3] Benjamin Riedel, Isabelle Augenstein, Georgios P. Spithourakis, and Sebastian Riedel. A simple but tough-to-beat baseline for the fake news challenge stance detection task. *CoRR*, abs/1707.03264, 2017.