Automated Fact Checking

Student number - 18071894
Information Retrieval and Data Mining (COMP0084)
Department of Computer Science, University College London

ABSTRACT

In recent years, there has been an increase in the spread of misinformation, in particular in 'fake news', on social media networks such as Twitter and Facebook. This has led to a growth in research in the field of automated fact checking with the motivation to combat this problem. In this paper, a model was built to detect whether a given claim can be supported or refuted, using the 5.5 million documents in the Wikipedia June 2017 dump as the data source against which the claim's veracity is checked. The claims were taken from the FEVER 1.0 dataset, which contains 145 thousand labelled training examples and 20 thousand of both labelled development examples and unlabelled test examples [1]. This task consisted of three main steps: relevant document retrieval, evidence sentence selection, and claim veracity prediction. The final model, the bidirectional long short-term memory recurrent neural network, obtained an accuracy of 66% on the development subset, compared to 58% for the baseline feedforward neural network model.

1 TEXT STATISTICS

The frequency of each word from each of the documents in the Wikipedia June 2017 dump was counted by converting the downloaded 'wiki-pages' folder into a list of files and then iterating through that list. After lowercasing the texts, a regular expression was used to tokenize each of the texts as this is more computationally efficient than the NLTK tokenizing tools. However, one drawback of using this regular expression is that it does not deal with apostrophes in an appropriate manner; a word containing an apostrophe will be split into two different tokens. For example, the word 'don't' would be split into the tokens 'don' and 't'.

Zipf's law states that given a large sample of words, the frequency of any word is inversely proportional to its rank in the frequency table, i.e. the frequency of the word with the n^{th} ranked frequency is inversely proportional to n. This is equivalent to saying that the probability of any word appearing (word's frequency divided by the total number of words in the collection) is inversely proportional to its rank:

$$r \cdot p = c \tag{1}$$

where r is the word's rank, p is the word's probability, and c is a constant (to be determined). Figure 1 shows the plots of log(frequency) against ranking and log(frequency) against log(ranking). The graph on the left shows the expected curve shape for Zipf's law, and that on the right shows an almost straight line with a gradient of minus one, which is also expected. Figure 2 shows the same relationships but for probability instead of frequency.

Three different methods were used to estimate the parameter, c: mean squared error, mean, and median. For mean squared error, the following derivation was used to calculate this estimate:

$$r \cdot p = c$$
$$log(r \cdot p) = log(c)$$
$$log(r) + log(p) = log(c)$$
$$\underbrace{\log(p)}_{y} = \underbrace{-}_{gradient} \underbrace{\log(r)}_{x} + \underbrace{\log(c)}_{intercept} \tag{2}$$

Therefore, c is equal to the exponential of the intercept of the line of best fit to the log(probability) against log(ranking) graph. The *polyfit* function from the NumPy library was used to calculate this intercept and a value of 1.2 was obtained for this estimate. After calculating the mean and median of the values for ranking multiplied by probability, 0.008 and 0.007 were obtained respectively as estimates for the constant. This constant should be roughly 0.1 in the English language and none of these estimates were close to this value. However after rounding all of the values for ranking multiplied by probability to one decimal place, it was observed that the 18 thousand most used words had this value.

2 VECTOR SPACE DOCUMENT RETRIEVAL

The aim of this subtask was to extract TF-IDF representations of ten claims from the training subset and all of the documents respectively, based on the document collection. Then, these representations were used to calculate, given a claim, its cosine similarity with each document and find the five most similar documents for that claim.

Term frequency-inverse document frequency (TF-IDF) is a measure of how important a word is to a document in a collection or corpus. The importance increases (proportionally) with the number of times the word appears in the claim or document, and decreases with the number of times the word appears in the document collection. TF is given by:

$$TF = \frac{n_w}{n_t} \tag{3}$$

where n_w is the number of times the word appears in the claim or document, and n_t is the total number of words in that claim or document. IDF is given by:

$$IDF = log_{10}\left(\frac{N_d}{N_w}\right) \tag{4}$$

where N_d is the total number of documents in the collection and N_w is the number of documents in which that word appears. The TF-IDF weight for a word in a claim or document is the product of its TF and IDF.

In order to obtain these representations for the ten claims, the training subset and a list containing the document IDs were loaded. Then, the ten claims were tokenized, using the same regular expression as before, and stemmed, using the *PorterStemmer* function

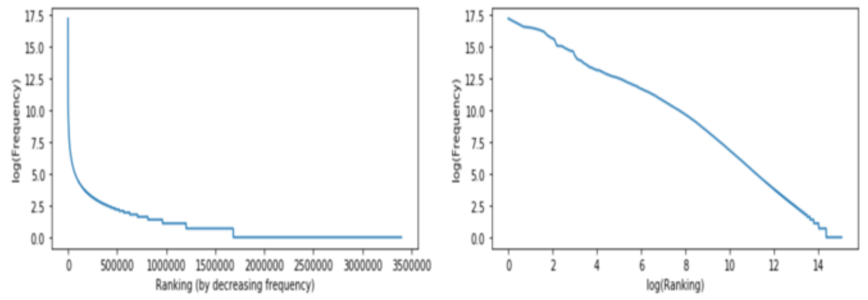


Figure 1: log(frequency) against ranking graph (left) and log(frequency) against log(ranking) graph (right).

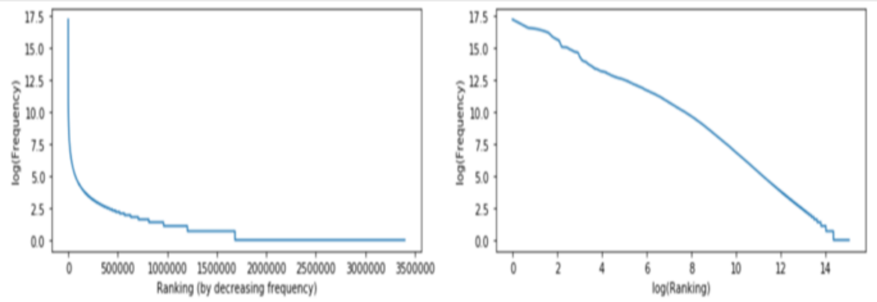


Figure 2: log(probability) against ranking graph (left) and log(probability) against log(ranking) graph (right).

from NLTK. Stemming is used to account for the fact that documents use different forms of words (such as 'playing', 'play' and 'played'). The Porter stemming algorithm is the most popular and focuses on removing suffixes from words.

Two inverted indexes were created; one for the documents and one for the claims. These were dictionaries, with the keys being a word. For the document inverted index, values were a set of tuples. Each tuple represented a different document, where the first element of each tuple was the document number, the second element was the number of times the word appears in the document, and the third element was the total number of words in the document. Since documents only needed to be represented based on the words that would have an effect on the cosine similarity with the claims, the keys of the inverted index were only the words that appeared in the ten claims. The inverted index for the claims had a similar structure. The keys were the same as that of the document inverted index and the values were also a set of tuples. In this case, each tuple represented a claim, where the first element of each tuple was the claim number, the second element was the number of times the word appears in the claim, and the third element was the total number of words in that claim.

These inverted indexes were used to form an IDF vector for each of the words and two TF matrices; one for the claims and one for the documents. Each row represented a claim or document and each column represented a word. Matrix-vector multiplication was performed to obtain TF-IDF matrices for the claims and the documents, which had the same dimensions as the TF matrices.

A cosine similarity matrix was formed by computing the cosine similarity between each row of the document TF-IDF matrix and each row of the claim TF-IDF matrix. The cosine similarity between arbitrary vectors A and B is given by:

$$cossim(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5}$$

This matrix consisted of ten rows and 5.5 million columns. From this matrix, the IDs of the five most relevant documents for each of the ten claims were retrieved by finding the top five values in each row, and then indexing the list of document IDs created earlier. This method produced desirable results as all of the retrieved documents were related to their corresponding claims.

This process was repeated but after removing the stopwords (i.e. the most common words in the English language such as 'the' and 'and') and relevant documents were retrieved again. The NLTK *english* stopwords set was used. This was done due to these words carrying less important meaning than keywords. This again produced desirable results. Comparing the two sets of results, there were only twelve different documents out of the fifty retrieved for each method (five for each of the ten claims).

3 PROBABILISTIC DOCUMENT RETRIEVAL

Probabilistic retrieval methods can be used as an alternative to vector space ones. They involve language models, which are probability distributions over sequences of text; i.e. how likely a sequence is to appear in a language. In this case, each document in the collection defines a language.

Firstly, a query-likelihood unigram language model was built, based on the document collection. A unigram language model is a probability distribution over individual words. A query-likelihood model ranks documents by the probability that the query could be generated by the document model. In this case, the query is the claim. To do this a probability matrix was built, where each row represented a claim and each column represented a document. Each element of the matrix represented a probability and was calculated by multiplying the TFs of each word from that row's claim for that column's document. It was assumed that each word in the claim was generated independently. Similarly, the five most relevant documents for each claim were found by taking the top five values in each row. This model failed to produce meaningful results. This is because the probability of a claim given a document in this model will equal zero unless all of the words in the claim are present in that document. Only a small number of retrieved documents contained all of the words in the claim. This is shown by only a minority of these retrieved documents being related to their corresponding claim. The rest of the retrieved documents were the same regardless of what the corresponding claim was.

It is unrealistic to assume that missing words have zero probability of occurring. This can be accounted for by smoothing. Smoothing estimates probabilities for missing or unseen words. This is done by discounting the probability estimates for words present in the document and assigning that remaining probability to the probability estimates for the missing words.

The first smoothing technique implemented was Laplace smoothing. This involves counting the occurrences of words in the document, adding one to every count and then renormalizing. If the word occurrences in a document are n_1,n_2,...,n_V, V is the number of unique words in the document collection (the vocabulary size), and $\mid D \mid$ is the length of the document, the Laplace estimates are given by:

$$\frac{n_1 + 1}{\mid D \mid + V} + \frac{n_2 + 1}{\mid D \mid + V} + \ldots \tag{6}$$

Again, a probability matrix was formed with the same structure as before and the five most similar documents were retrieved. This model produced slightly more meaningful results than the vanilla model, however some of the retrieved documents were the same for different claims, which should not happen as these claims are not related. This problem is likely to have arisen due to unseen words being treated equally. However, this should not be done as some words obviously appear more frequently than others.

To account for this, the next smoothing method implemented was Jelinek-Mercer smoothing. This method creates a mixture model; it mixes the probability of the word from the document with the background probability for that word (probability from the document collection). These estimates are given by:

$$\lambda \cdot TF_{doc} + (1 - \lambda) \cdot TF_{col} \tag{7}$$

where TF_{doc} is the TF for the word in the document and TF_{col} is the TF for the word in the collection (number of times the word appears in the collection divided by the total number of words in the collection). λ is a hyperparameter and was chosen to be 0.5. The documents were retrieved using the same methods as before and these documents were much more relevant. All of the documents were related to their corresponding claim.

The final smoothing method implemented was Dirichlet smoothing, which is closely related to Jelinek-Mercer smoothing but generally performs better for shorter queries (claims in this case). The hyperparameter is dependent on the length of the document and the average document length of documents in the collection. The estimates are given by:

$$\frac{N}{N + \mu} \cdot TF_{doc} + \frac{\mu}{N + \mu} \cdot TF_{col} \tag{8}$$

where μ is the average document length in the collection, N is the length of the document, and TF_{doc} and TF_{col} are the same as before. Again, the resulting retrieved documents were all related to the subject of their claims. However after reading the text from these documents, it was observed that the different retrieved documents were slightly more specific to the actual claims being made. Compared to the results for Jelinek-Mercer smoothing, there were nine different documents out of the fifty retrieved for each method.

4 SENTENCE RELEVANCE

The aim of this subtask was to build a logistic regression model that can correctly predict whether a sentence is relevant to a given claim. If a sentence is relevant, then it can go on to predict whether a claim can be supported or refuted. Pre-trained word embeddings were loaded, specifically the GloVe Wikipedia 2014 50d vectors. The 50-dimension vectors were chosen for computational efficiency. The average of all of the word vectors was used for unknown words, as recommended by the GloVe author, Jeffrey Pennington [2].

A feature matrix, where each row represented a claim and sentence pair, and a label vector, where each element represented a claim and sentence pair and was either 1 for relevant or 0 for nonrelevant, were built. This was done by extracting the relevant sentences from the evidence 'field' of each 'VERIFIABLE' claim in the training subset, i.e. the claims where there is sufficient evidence to support or refute it. For the purpose of the logistic regression model, only those evidences that did not require multiple sources to prove that claim were considered. For each pair, negative sampling was then performed, using the *sample* function from the Python library random, to retrieve a nonrelevant sentence from the same document and this formed a nonrelevant pair. This ensured that there was a balanced dataset (same number of relevant pairs and nonrelevant pairs).

After tokenizing all of the claims and sentences, for each row in the feature matrix, the embedding vectors for each word in the claim were summed and the same thing was done for its paired sentence. Then, the two resulting vectors were concatenated to form one 100-dimension vector. This, along with the label vector (containing 1s and 0s) would be the input for the logistic regression model. This process was repeated for the development subset for the evaluation of the model.

Firstly, the sigmoid, log loss and gradient functions were defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{9}$$

$$L(\hat{y}, y) = -\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{10}$$

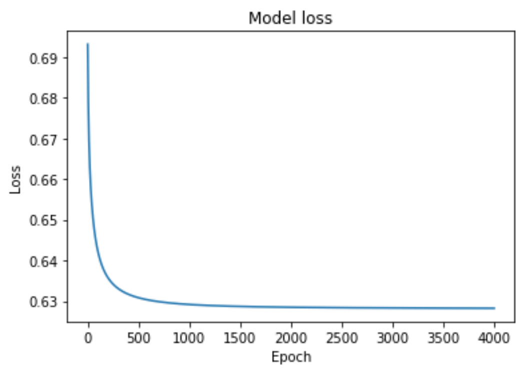


Figure 3: Training loss curve for logistic regression model

$$\nabla = \frac{1}{m} \cdot \sum_{i=1}^{m} (\hat{y}_i - y_i) \cdot x_i \tag{11}$$

where in this case, m is the number of examples, $\hat{y_i}$ is the prediction, y_i is the label and x_i are the features for example i. The sigmoid function squashes the input between zero and one. The model was trained by firstly initializing the weights as zero and adding a bias, and then making predictions by inputting the dot product of the feature matrix with the intialized weights through the sigmoid function. The weights were then updated by minusing the product of the gradient and the learning rate (which was to be specified). This was repeated using the new weights for a specified number of epochs (repetitions).

After investigating for a range of epochs and learning rates, the parameters chosen for the final model were 4,000 epochs and a learning rate of 0.001. If the training was run for too many epochs, the loss would continue to decrease however produced a lower accuracy on the development subset. This is due to overfitting; the model started to learn patterns from the training data that don't generalize well to other datasets. However, if the model was not trained for long enough, it underfitted (hasn't learned the relevant patterns in the training data). The learning rate controls how much the weights are adjusted with the gradient. When the model was trained with a learning rate of 0.01, the training loss failed to converge, and when it was trained with 0.1, it in fact diverged. The 0.0001 learning rate model's training loss converged albeit slowly. Therefore, the chosen learning rate was 0.001. The training loss curve is shown in Figure 3.

One evaluation metric used for the model involved retrieving the five most relevant documents for the first ten claims in the development subset. The retrieval method used for this was TF-IDF representation and cosine similarity. The fifty documents were split into sentences using the 'lines' fields in the data, and each sentence was tokenized. Following this, each sentence was converted into a vector by summing the word embedding vectors. This vector was concatenated with the summed vector for the corresponding claim. Inputting this vector with the weights from the chosen trained logistic regression model into the sigmoid function, predictions of whether or not each sentence is relevant to the claim were obtained. If the sigmoid output was greater than 0.5, the sentence was predicted to be relevant. The correct labels were obtained from the 'evidence' fields in the development subset for each of the claims. If a sentence was in the 'evidence' field, then that sentence was relevant, and if it was not, the sentence was nonrelevant. After comparing the predictions with the labels, an accuracy of 80.79% was achieved. However, only 0.76% of the sentences from the fifty retrieved documents were relevant to their corresponding claims according to the 'evidence' field from the data. Therefore in this case, a more appropriate metric would be recall (sensitivity), which is equal to the number of correctly predicted relevant sentences divided by the total number of relevant sentences. A high score of 0.82 was achieved for this.

5 RELEVANCE EVALUATION

In order to evaluate the performance of the model, the development subset was firstly transformed into a feature matrix and label vector in the same way as this was done for the training subset. Then predictions were made, using the weights from our chosen logistic regression model. The accuracy of the model, i.e. the percentage of correctly predicted sentences, was calculated by comparing the predictions to their corresponding labels and was 67.45%. Recall, precision and F1 were calculated using:

$$recall = \frac{TruePos}{TruePos + FalseNeg} \tag{12}$$

$$precision = \frac{TruePos}{TruePos + FalsePos} \tag{13}$$

$$F1 = 2 \cdot \frac{recall \cdot precision}{recall + precision} \tag{14}$$

where TruePos is the number of relevant sentences correctly predicted to be relevant, FalseNeg is the number of relevant sentences incorrectly predicted to be nonrelevant, FalsePos is the number of nonrelevant sentences incorrectly predicted to be relevant and TrueNeg is the number of nonrelevant sentences correctly predicted to be nonrelevant. For all three metrics, scores of 0.68 were achieved.

If the scores were below or close to 0.5, this would mean that the model had either learned a nonrelevant pattern from the training subset or was just randomly guessing. However, the accuracy along with the metric scores are all significantly above 0.5, which suggests that the model has sufficient ability to predict whether a sentence is relevant to a given claim.

6 TRUTHFULNESS OF CLAIMS

The aim of this task was to build a model, specifically a neural network, to predict whether a claim is 'supported' or 'refuted' (proven true or false) by a given relevant sentence. Neural networks are machine learning models who's architectures are inspired by that of the human brain. They are made up of nodes, which mirror the neurons in a biological brain. There are three types of nodes; input, hidden and output. The input nodes provide information from the outside world to the network and are together referred to as the input layer. The hidden nodes have no direct connection with the outside world and make up the hidden layer (or layers).

Within hidden layers, complex features are created, which are useful for predicting the output but may be difficult for a human to understand. The output nodes make up the output layer and are responsible for computations and transferring information from the network to the outside world.

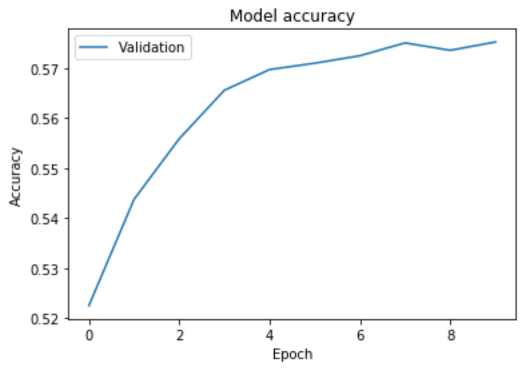
The claims with the label 'NOT ENOUGH INFO', i.e. claims that cannot be proven true and false, were filtered out. Then, the data was transformed into a feature matrix (100 columns representing the concatenated vectors and each row represents a claim-sentence pair) and label vector (1s and 0s for 'SUPPORTS' and 'REFUTES' respectively) with a similar method to how it was done for the logistic regression. The difference here was that the labels for the sentence pair were 'SUPPORTS' and 'REFUTES' rather than 'relevant' and 'nonrelevant', and no negative sampling was required. However, the dataset was very unbalanced. 73% of the claim-sentence pairs had the label 'SUPPORTS'. This can be problematic when training as the model could overfit to the more popular class and ignore the other class. After training, it was observed that when the model was trained using the unbalanced dataset, its predictions on the development subset all had the label 'SUPPORTS'. To combat this, undersampling was performed. A random sample of 'SUPPORTS' examples of the same size as the number of 'REFUTES' examples was obtained by again using the *sample* function. This was an appropriate method to balance the data since there was still a sufficiently large number of examples in the 'REFUTES' class.

The Keras library was used to build the neural network. The architecture consisted of one hidden layer between the input and output layers. The activation function chosen for the hidden layer was the rectified linear unit (ReLU), which is defined by:

$$f(x) = max(0, x) \tag{15}$$

Activation functions are used to introduce nonlinearity to the network. ReLU was chosen because of its computational efficiency and its fast convergence. Also, not all of the neurons are activated at once, which causes sparsity. This often results in models that overfit less and have better predictive power. After this hidden layer, dropout was implemented with a parameter of 0.5 This causes outputs of the hidden layer to be ignored with a probability of 0.5. This is a regularization technique, i.e. a method to prevent overfitting. The number of hidden nodes was first chosen to be 50, the mean of the number of input and output nodes. However, since dropout causes the outputs of the hidden layer to be randomly sampled, an improvement was made to the network by increasing the number of hidden nodes to 75 (closer to 100, the number of input nodes). The activation function chosen for the output layer was the sigmoid function as this is appropriate for binary classifiers (two classes). The *kernel_initializer* parameter, i.e. the way to set the initial weights, was set to *glorot_uniform* for each layer. This is an effective initialization method as it helps ensure that the neuron activation functions are not starting out in saturated or dead regions and therefore helps signals reach the network. It draws samples from a uniform distribution within the range [-x,x] where:

$$x = \sqrt{\frac{6}{in + out}} \tag{16}$$



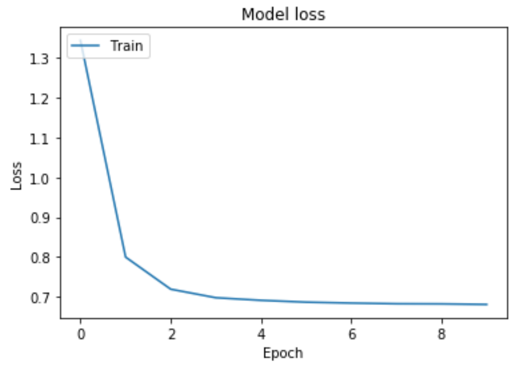


Figure 4: Validation accuracy and loss curves for feedforward neural network

where *in* and *out* are the number of input and output nodes for that layer. The training set was split into batch sizes of 125 to avoid any memory issues. These batches were shuffled before being inputted, to avoid any patterns or bias within the batches.

Similar to the logistic regression, the loss function chosen was the log loss. The optimizer chosen was stochastic gradient descent, due to its fast convergence and its compatibility with the ReLU activation function. The *nesterov* argument was set to *True*, which applies Nesterov momentum to the optimizer. This helps accelerate convergence in the relevant direction and dampen oscillations.

The model was trained for ten epochs and the training loss and validation accuracy curves are shown in Figure 4.

A feature matrix and label vector were obtained from the development subset and predictions were made using the trained model. An accuracy of 57.52% was achieved. Recall, precision and F1 were 0.55, 0.59 and 0.57 respectively. These scores are all above 0.5 but not significantly. This suggests that the model has an element of predictive power for this task, but not a strong one.

7 LITERATURE REVIEW

Traditionally, the task of fact-checking has been carried out by humans. For example, PolitiFact is a platform where specialist journalists look for statements from speeches, campaign manifestos and news stories to fact-check [3]. However, the rapidly increasing volume of information and data being released by journalists and politicians, combined with the speed of which these facts can be spread has caused the need for research into automated fact checking and verifying. Machine learning algorithms are trained to recognise patterns, therefore a large section of this research focuses on these techniques.

One of the more successful efforts to build an automated fact-checking system was done by Hassan et al. (2017) with their ClaimBuster platform [4]. This uses natural language processing, supervised learning and database query techniques to detect falsehood in political dialogues. It monitors speeches, news, debates and interviews to catch factual claims, and then instantly delivers the labels to readers and viewers. For claims where humans must be brought into the loop, it provides algorithmic and computational tools to assist the people in understanding and verifying the claims. It was trained on 21 thousand sentences spoken by US presidential candidates between 1960 and 2012. Their three-class classification used multinomial naive Bayes, support vector machines and random forests. The model trained on data specific to presidential debates and didn't generalise well to other areas of politics. This suggests that the model would generalise even worse to non-politics subjects.

Other similar platforms include Storyzy, which is a quote verifier [5]. It uses a database of 5 thousand 'reliable sources' from the mainstream media for verifying quotes. The potential drawback with this is that the mainstream media may be biased itself. It has been shown that the algorithm for this gets confused by longer quotes that have small discrepancies. Another platform is TruthTeller from The Washington Post, which performs algorithmic claim matching, similar to ClaimBuster [6]. However, it has a worse performance due to it carrying out straightforward string matching rather than attempting to interpret the semantics and structure of the claims.

Agirre et al. (2013) took the approach of matching a claim to one that has previously been fact-checked by a journalist [7]. They then found the label by assessing the semantic similarity between statements, using the k-nearest neighbors clustering method. A drawback of this approach is that it cannot be applied to new claims, such as breaking news, as these will not have been fact-checked yet. Therefore, this is only useful for claims that have been paraphrased or repeated. A suggested extension to this approach by Vlachos and Riedel (2014) is to assume that some text corpus, say Wikipedia, is the source of all true claims, including novel claims [8].

Yang et al. (2018) proposed a text and image convolutional neural network (CNN) for the task of detecting fake news [9]. It combined the text and image information with the corresponding latent and explicit features. Their approch had strong expandability and could easily absorb other featues of news. CNNs can be trained much faster than other deep learning frameworks, such as recurrent neural networks, as it can see the entire input at once. The reason CNNs are so useful for image-related tasks, such as facial or object recognition, is that the local structure is important; adjacent pixels hold useful semantic information. However, the drawback of using them for text classification is that in sentences, words don't have to be adjacent to be related. Therefore, it could have a worse performance in part-of-speech tagging (marking up a word as corresponding to a particular category of words with similar properties, based on both its definition and context) and entity extraction (identifying and categorising key elements from text into pre-defined categories, such as people and places).

Roy et al. (2018) proposed a deep ensemble framework with a Bi-LSTM (specific type of recurrent neural network) and CNN, and achieved state-of-the-art results for fake news detection and classification for US politics [10]. The input data was the statement, the speaker, the speaker's job title, the subject, the state information, the party affiliation, and the context (venue or location of the speech or statement). The statements were classified into six classes: 'true', 'mostly-true', 'half-true', 'barely-true', 'false', and 'pants-fire'. They achieved high precision but very low recall and F1; the model predicted a smaller number of instances as being 'true', but when it did predict this it was very accurate. This model's performance could be further improved by using a larger dataset, and also by including more features, such as a speaker's history of lying.

8 IMPROVEMENTS

An improvement made to the feedforward neural network was to instead implement a recurrent neural network (RNN), specifically a bidirectional long short-term memory model (BiLSTM). RNNs are designed to work with sequence prediction problems and have been proven to be successful in natural language processing tasks, i.e. tasks that involve working with sequences of words in text.

However, traditional RNNs suffer from 'short-term memory', which means that they struggle with carrying information from earlier time steps to later ones. This could have been problematic here if the claim or sentence in a pairing was long. RNN's can suffer from the vanishing gradient problem during backpropogation if the gradient becomes too small. This usually occurs in the earlier layers and because they will only get a small gradient update, they stop learning. The network could forget what it has previously seen.

LSTMs are designed to solve this problem. They are made up of a cell, an input gate, an output gate and a forget gate. The cell is the memory part of the LSTM architecture, and the three gates regulate the flow of information inside the LSTM unit. These gates learn what parts of the data in a sequence are relevant and important, and then decides whether to keep the information or forget it. The input gate controls the extent to which new information enters the cell, the forget gate controls the extent to which this new information remains in the cell, and the output gate controls how much of the information in the cell is used to compute the output activation. Each gate contains a sigmoid activation. BiLSTMs are an extension of this and run the sequence both from the past to the future and in the reverse direction. They are especially effective with tasks involving text as they do well at interpreting the context.

For training, the input data was taken from what was used for the feedforward neural network. However, each example (claim-sentence pair) was split into a sequence of length two using the NumPy function *reshape*. Each segment of the sequence had a length of 50 and represented either the claim or sentence. Since LSTMs tend to take longer to train than feedforward neural networks, it was run for fifteen epochs rather than ten.

Another improvement made to the model was changing the optimizer from stochastic gradient descent to the adaptive Adam optimizer. Adam has a nonconstant learning rate and combines the advantages of two other extensions of stochastic gradient descent; root mean square propogation and adaptive gradient algorithm. It finds individual learning rates for each parameter and works

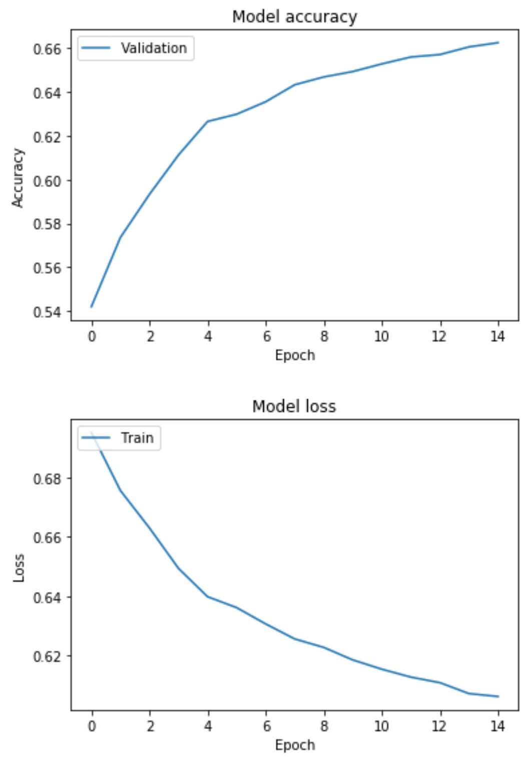


Figure 5: Validation accuracy and loss curves for BiLSTM model

well with sparse gradients. The algorithm calculates an exponential moving average of the gradient and the gradient squared. The values of 0.9 and 0.999 were chosen for the parameters $beta_1$ and $beta_2$ respectively, as recommended in the original Adam paper [11]. They control the decay rates of the moving averages. The training loss and validation accuracy curves are shown in Figure 5.

An accuracy of 66.26% was achieved on the development subset, a 9% increase from the feedforward neural network. Recall, precision and F1 were 0.64, 0.68 and 0.66 respectively. These scores are significantly above 0.5, which suggests the model has strong predictive power for this task.

The first ten claims from the unlabelled test subset's veracity was predicted. For each of the claims, its five most relevant documents were retrieved from the document collection. This was done using the TF-IDF and cosine similarity method. Each of the fifty documents were split into sentences and then tokenized. The loaded word embeddings for each word in a sentence and each word in a claim were summed and concatenated to obtain a vector of length 100 for each claim-sentence pair. Each pair was inputted through the previously trained logistic regression model to retrieve the predicted relevant sentences. Then, these relevant sentence-claim pairs were reshaped and inputted into the trained BiLSTM model to predict whether each claim could be supported or refuted. The predictions for each claim were averaged and if that average was greater than 0.5, then it was classified as 'SUPPORTS', and if it was less than 0.5, it was classified as 'REFUTES'. In the test subset submitted results in the 'predicted_evidence' field, only those predicted relevant sentences that gave the same result as the final prediction for each claim were included.

9 CONCLUSION AND FUTURE WORK

Future work would use the 300-dimension word embedding vectors from GloVe rather than the 50-dimension ones as these vectors carry more semantic meaning. The corpus used for the model could be expanded to include other sources as well as Wikipedia. This would mean that less claims would be non-verifiable, and for those that are verifiable, there would be more relevant sentences to support or refute that claim. A further improvement that could be made to the model would be to not sum the word vectors in each claim and sentence, but instead concatenate all of the vectors. Although this would cause a significant increase in computation time, it would maintain a lot more of the semantic meaning of the sentence, which could be lost from summing the vectors. Then, the BiLSTM can treat all of the separate vectors as a sequence and this should increase the predictive power of the model.

The dangerous consequences of the spread of misinformation increases the importance of research in the field of automated fact checking. The next steps are to address the weakness of these models' abilities to deal with new claims, such as breaking news.

ACKNOWLEDGMENTS

Emine Yilmaz and the other lecturers, for their insightful Data Mining and Information Retrieval lectures, and also to the teaching assistants for their kind assistance.

REFERENCES

[1] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. [online] http://fever.ai/resources.html [Accessed 1 Mar. 2019].
[2] Jeffrey Pennington. 2014. Groups.google.com. Google Groups. [online] Available at: https://stackoverflow.com/questions/49239941/what-is-unk-in-the-pretrained-glove-vector-files-e-g-glove-6b-50d-txt [Accessed 1 Apr. 2019].
[3] 2018. [online] Available at: https://www.politifact.com/truth-o-meter/article/2018/feb/12/principles-truth-o-meter-politifacts-methodology-i/Truth-O-Meter%20ratings [Accessed 5 Apr. 2019].
[4] Naeemul Hassan, Anil Nayak, Vikas Sable, Chengkai Li, Mark Tremayne, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph and Aaditya Kulkarni. 2017. [online] ClaimBuster: the first-ever end-to-end fact-checking system. Proceedings of the VLDB Endowment. 10. 1945-1948. 10.14778/3137765.3137815. [Accessed 5 Apr. 2019].
[5] Storyzy. 2012. [online] https://storyzy.com [Accessed 5 Apr. 2019].
[6] TruthTeller. 2013. [online] https://www.washingtonpost.com/news/ask-the-post/wp/2013/09/25/announcing-truth-teller-beta-a-better-way-to-watch-political-speech/ [Accessed 5 Apr. 2019].
[7] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez- Agirre and Weiwei Guo. 2013. [online] *sem 2013 shared task: Semantic textual similarity. In Proceedings of the Second Joint Conference on Lexical and Computational Semantics and the Shared Task: Semantic Textual Similarity, pages 32-43, Atlanta, GA. [Accessed 5 Apr. 2019].
[8] Andreas Vlachos and Sebastian Riedel. 2014. [online] Fact Checking: Task definition and dataset construction. Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science, pages 18-22, Baltimore, MD, USA. [Accessed 5 Apr. 2019].
[9] Yang Yang, Lei Zheng, Jiawei Zhang, Qingcai Cui, Xiaoming Zhang, Zhoujun Li and Philip S. Yu. 2018. [online] TI-CNN: Convolutional Neural Networks for Fake News Detection. https://arxiv.org/pdf/1806.00749.pdf [Accessed 5 Apr. 2019].
[10] Arjun Roy, Kingshuk Basak, Asif Ekbal and Pushpak Bhattacharyya. 2018. [online] A Deep Ensemble Framework for Fake News Detection and Classification. https://arxiv.org/pdf/1811.04670.pdf [Accessed 5 Apr. 2019].
[11] Diederik P. Kingma and Jimmy Lei Ba. 2015. [online] Adam: A Method for Stochastic Optimization. https://arxiv.org/pdf/1412.6980.pdf [Accessed 6 Apr. 2019].