Course Project: Automated Fact Checking

COMP0084: Information Retrieval and Data Mining
Student number: 14010014

ABSTRACT

This report follows the course project description on automated fact checking. The aim of the project is to develop information retrieval and data mining methods to assess veracity of claims using over 5 million Wikipedia pages as documents. First, we develop methods for relevant document retrieval, considering both vector space transformation and probabilistic approaches. Second, we apply a logisitc regression to retrieve the relevant sentences from the previously retrieved documents. Lastly, we train a Bi-directional LSTM (BiLSTM) with Glove embedded claims and relevant sentences to predict whether the claims are true or false. We build on this model by including part-of-speech (POS) tagging and hyperparameter optimization. Due to computational limitations, we train the final model on 3,000 claims from the training set with 8 epochs and obtain validation accuracy of 69%. We make further recommendations for the improvement of this model, had we had the processing power to go use more claims in training and more time to develop the network architecture.

1 INTRODUCTION

The awareness of misinformation through has been rising rapidly over the last few years, as has interest in fake news detection and automated fact checking. With more information in circulation through social media and high speed internet connections, the truthfulness of news and cliams is rising to the top of information retrieval agenda. There are multiple shared tasks designed to encourage development of models in this field. We use the Fact Extraction and VERification (FEVER) dataset [1]. This dataset contains 185,445 claims, manually generated and verified against the introductions of Wikipedia pages and includes over 5 million Wikipedia pages as the documents in the corpus.

As a preliminary, we consdier the text statistics to show that the distribution of words in the corpus follows Zipf's law, which addresses Question 1 in Section 2. The end goal of this task is to label the claims with the correct truthfulness class, and return sentences identified as relevant as the evidence from the corpus. Thefore, the task is broken down into three segments. Section 3 considers Relevant Document Retrieval methods, where Questions 2 and 3 are answered in subsections 3.1 and 3.2, respectively. Section 4 considers Evidence Sentence Selection, where Questions 4 and 5 are answered in Sections 4.1 and 4.2, respectively. Lastly, Section 5 addresses the Claim Veracity Prediction, answering Question 6. Finally, Questions 7 and 8 are addressed in Sections 6 and 7, respectively. This report is complimentary to the Python code provided as part of the assignment.

2 TEXT STATISTICS

To compute the text statistics we consider Zipf's law, which states that for a given corpus the frequency of any word is inversely proportional to its rank in the frequency table. For example, it is typical that the word 'the' appears most frequently, and will occur twice as often as the second most frequent word, usually 'of'.

The application to the corpus of Wikipedia pages, we observe that the distribution of words broadly follows the law. In order to calculate this, we counted all the words in all documents using a counter, which stores the words and their frequencies as a dictionary. To obtain the sample probabilities, we divided the frequency of each word by the total number of words in the corpus. The plot on the left hand side in Figure 1 plots the rank of the word against its frequency for the first 1,000 words. We see a Zipfian distribution, where the highest ranked words have significantly higher frequency and the freqency quickly decays to 1.

We can summarise the law more formally by considering the probability of words, which are $k \Pr_r \approx c$, where k is the rank, $\Pr_r$ is the sample probability, calculated by the frequency of the word divided by the total number of words in the corpus, and some constant c. It has been shown that $c \approx 0.1$ for the English language.

We can show that our corpus has similar values for the constant by considering the transformed graph of log probability against log rank. This linearalises the relationship between the two variables, to which we fit a linear regression by hand, $\log(Pr) = \beta_0 - \beta_1 \log(k)$. If our corpus follows Zipf's law, then $\exp(\beta_0) \approx 0.1$ and $\beta_1 \approx -1$. Plotting the transformed variables and the regression, we obtain the right hand side plot in Figure 1. The estimated coefficients for the linear regression are $\beta_0 = 0.38$ and $\beta_1 = -1.27$, showing that this corpus does broadly follow Zipf's law.

3 RELEVANT DOCUMENT RETRIEVAL

The end goal is to transform the sentences in the claims and documents to figure out how similar they are. There are two methods we consider. The first considers vectors, the other is a probabilistic approach.

3.1 Vector Space Document Retrieval

This method uses text statistics on the words in the claim to vectorise them. We only consider the words in the claims throughout this process. This saves time but also does not change anything since the words not in the claim would have a dot product of 0 so would not be considered anyway.

3.1.1 TF-IDF. To vectorise the claims and documents, we consider Term Frequency - Inverse Document Frequency (TF-IDF). TF gives us the relative frequency of the word in each document in the corpus to the total number of words in that document. It is essentially the sample probability of that word occuring in that document. Equation 1 depicts this calculation, where $n_{i,j}$ is the count of word i in document j.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \tag{1}$$

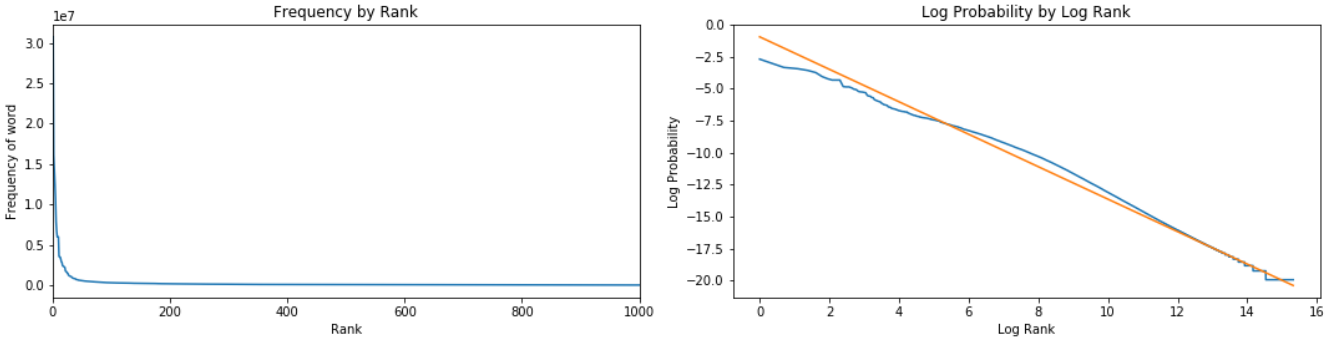


Figure 1: Zipf's law

However, the most common words like 'the' and 'of' will have the highest TF but that will be true across all documents. Therefore, we consider the rarity of the words we are vectorising across all the documents in the corpus. This is what the IDF calculates, and is shown in Equation 2, where N is the total number of documents and df_i is the number of documents containing word i.

$$idf_i = \log\left(\frac{N}{df_i}\right) \tag{2}$$

Now, we multiply the two together so that the term frequency is weighted by the 'rarity' of the word. The more rare the word is, the higher the TF-IDF. This final calculation is given by Equation 3.

$$tfidf_{i,j} = tf_{i,j} * idf_i \tag{3}$$

3.1.2 Cosine Similarity. Having calculated the vector space representation of the query terms and documents, we can measure the similarity between the two items by considering the cosine of the ange between the two vectors of an innter product space. For vector $A_i = tfidf_{i,c}$ for claim c and vector $B_i = tfidf_{i,j}$ for document j, the cosine similarity between the claim and the document is given by equation 4.

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{4}$$

This method tells us how similar the two vectors are. The maximum similarity is obtained when the two vectors are parallel, and maximium dissimilarity when they are orthogonal. When we consider the top five documents, we keep the ones with the highest cosine similarity for each claim.

Implementing this methodology for our training set, we had to be mindful of the memory and disk resource restrictions. In light of this, we performed this method by batching within the document collection, meaning we iterated over each of the claims whilst parsing through the 109 files in the document collection individually. We also removed stop words and counted the similarity over only the claim words in the documents.

Additionally, we included the title of the documents as an extra filter in choosing the relevant documents. If the title did not include any of the claim words, it was discarded. Of course, this does not hold for all claims, but it sped up the process and overall the recall of relevant documents was higher, so we thought that was a worthy trade-off. In the same vein, we removed the documents which are disambiguation pages on Wikipedia, meaning that they are lists of other documents with very similar titles. These usually do not include information relevant to fact checking.

Although we are not optimising using the development set, we found it informative to consider the recall scores of relevant documents for the training and development data. Implementing the aforementioned method and the alterations resulted in a recall of 22% on the training claims and 58% on the development claims. Finally, we provide a csv file with the output of the relevant document ids for each of the ten claims provided.

3.2 Probabilistic Document Retrieval

For the probabilistic document retrieval, we construct a unigram language model which considers the probability distribution over the words in the corpus. The probabilities of each word appearing in each document is estimated by the proportion of the word frequency to the total number of words in the document. If the word does not appear in the document, the probability is 0.

If the number of occurance of words in D are $(m_1, m_2, ..., m_{|V|})$, with $\sum_i m_i = N$, where $|V|$ is the number of unique words in the entire collection, the maximium likelihood estimates are given by

$$\left(\frac{m_1}{|D|}, \frac{m_2}{|D|}, ..., \frac{m_{|V|}}{|D|}\right) \tag{5}$$

In this task, for each given claim, we extract each word in the claim. For each document, we then count the number of times each claim word appears and multiply these for each word in the claim. We repeat this for all claims across all documents. We then have a probability value of a claim appearing in a particular document. The highest probabilities are the ones where the claim words have been mentioned most relative to all the words in the document, hence these documents are considered 'most relevant'.

There are several problems with this approach. Firstly, it requires all the words in the query to be present in the document to obtain a non-zero probability. This is a strong penalty for not containing specific words, but the document may still be relevant. The smoothing methods in the following subsections consdier how to deal with this problem.

Additionally, there are two strong assumptions used that allow us to multiply the sample probability of word occurance in the document. Firstly, the words are independent and secondly, that the order of the words does not matter. These issues are not addressed in the smoothing procedures, but we consider this in section 7, where we apply methods which take into account semantics and context.

We found that this method with no smoothing had a recall of 0, meaning that in no document were all the words of a claim present. Hence, we did not continue with this method, but only used the smoothed versions going forward.

3.2.1 Laplace Smoothing. This method adds 1 to every count of the claim word in each document, such that no claim word has count 0. Then, the probabilities are adjusted so that they sum to 1. This then discounts the sample probabilities for words that are seen in the document text. This is analogous to having a uniform prior over the claim words.

Following on from Equation 5, the Laplace smoothed estimates maximum likelihood estiamtes are given by Equation 6.

$$\left(\frac{m_1+1}{|D|+|V|}, \frac{m_2+1}{|D|+|V|}, ..., \frac{m_{|V|}+1}{|D|+|V|}\right) \tag{6}$$

3.2.2 Jelinek-Mercer Smoothing. This method addresses the problem of giving too much weight to unseen words by doing something less arbitrary. It is the weighted probability of the word appearing in the document and the probability of the word appearing in the whole corpus. This means that if the word is very rare in the whole corpus, each document should not be penalised for not including it. However, if the word is very common in the corpus, then the overall probability is penalised more if the document does not contain the word. The model equation is given by

$$\lambda * P(w|D) + (1-\lambda) * P(w|C) \tag{7}$$

where $P(w|D)$ is the probability of the claim word occuring in the document, which is the same as in the original method, and $P(w|C)$ is the probability of the word occuring in the entire corpus, and λ is a hyperparamter to be tuned.

When implementing this method, we set $\lambda = 0.5$. However, given more time, we would have cross-validated this parameter by considering the recall of the train and development data. As before, we found it informative to consider the recall of the relevant documents over the training claims, which was 67%, and the development was 62%.

3.2.3 Dirichlet Smoothing. The formula for this method is

$$\frac{N}{(N+\mu)} * P(w|D) + \frac{\mu}{(N+\mu)} * P(w|C) \tag{8}$$

where $P(w|D)$ and $P(w|C)$ are defined as above, N is the document length, and μ is a constant hyperparameter. Here, the difference is that we take into account the document length, as longer documents are more likely to contain more of the words, simply because they are longer. As before, the five documents with the highest probability per claim are returned as the most relevant documents.

4 EVIDENCE SENTENCE SELECTION

We have so far discussed two methods to retrieve the most relevant documents given a claim, but to be able to determine whether or not a claim is true, we need to be able to identify which sentences in the relevant documents are the relevant ones to support the classification of the claim.

4.1 Sentence Relevance

To extract relevant sentences from unseen queries, we train a logistic regression, which, given documents previously identified as relevant, attributed a probability of a particular sentence from the given document being relevant, meaning it can be used as evidence to support or refute the claim. We train this with the given test set of claims, for which we are able to retrieve relevant docuemnts, as discussed in section 3. Breaking these down into sentences, we embed them and the claims using pre-trained GloVe embeddings with 50 dimensions. As an input to the model, we stack the claim word embedding and the relevant sentence embeddings, and initialise the weights at 0.

For the labels, the training data set gives us the documents and sentances which are relevant and allow us to determine the veracity of the claim. For this model, we ignore how the claim is classified and istead focus only on which sentences are listed as evidence. Even if the claim is false, the evidence provided is still relevant. We show the calculation for the probability of a given sentence being relevant in equation 9, where $\boldsymbol{\theta}^{\boldsymbol{T}}$ represents the weights to be trained and the sigmoid function 'squashes' the results to be between 0 and 1. Equation 10 shows the log-loss function we have used to evaluate the model, which was minimised in every iteration to find the minimum.

$$\begin{aligned} h_\theta(x) &= g(\boldsymbol{\theta}^{\mathbf{T}}\mathbf{x}) \\ g(z) &= \frac{1}{1+e^{-z}} \end{aligned} \tag{9}$$

$$C(\theta) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x)) \tag{10}$$

We fitted this model to obtain the weights which assigns a probability to each sentence, $P(relevant = 1) = p$, by iterating until the update in the weights is negligible and we are in a minimum. For this, we use Newton's Gradient Descent method, where the update on the weights is calculated by the negative gradient divided by the Hessian. Although typically the Hessian has high time and space complexity of $O(N^2)$, since we are using a small subset of the claims (only 10), calculating this matrix was not too demanding and we were able to converge to a minimum. Therefore, we did not consider learning rates set as hyperparameters, as this method is faster and has the potential to converge in significantly fewer steps than first-order methods.

Using the fitted weights for the mdoel, we calculated evaluation metrics using the development set of claims to retrieve the most relevant sentences given the embeddings of the documents identified as relevant from the previous section. We considered the method which had the highest recall of relevant documents, which was probabilistic retrieval with Jelinek-Mercer smoothing.

To obtain the predicted labels for the development dataset, we set a boundary at $p = 0.5$. If the probability that the sentence is

relevant is below 0.5, we attribute the label to be 0. Otherwise, it is labelled as relevant and the corresponding label is 1. Then, we were able to calculate the evaluation metrics for the relevant sentence retrieval.

4.2 Relevance Evaluation

We consider three metrics for the evaluation of the retrieval methods. We also report the scores for the best document retrieval method, the probabilistic retrieval with Jelinek-Mercer smoothing.

4.2.1 Recall. We calculate the recall as the fraction of the relevant documents that are successfully retrieved in the model, as described in Equation 11. For Jelinek-Mercer Smoothing, recall is 67%.

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|} \quad (11)$$

4.2.2 Precision. We calcualte the precision as the fraction of retrieved documents that are relevant to the claim, as described in Equation 12. For Jelinek-Mercer Smoothing the precision is 1.9%.

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|} \quad (12)$$

4.2.3 F1 score. Considering recall and precision metrics alone may be misleading, as there is a trade-off between the success of these metrics. A model which retrieves all documents will have a high recall score but a very low precision score. The F1 score combines both recall and precision metrics to account for this trade-off. For Jelinek-Mercer Smoothing, the F1 is 3.7%.

$$F_1 = \left(\frac{recall^{-1} + precision^{-1}}{2}\right)^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (13)$$

5 CLAIM VERACITY PREDICTION

The logistic regression is able to retrieve the relevant sentences from the documents for each claim. We now need to verify if the claim is Supported, or Refuted, based on these sentences. We implement a Bi-directional Long-Short Term Memoery (BiLSTM) network. We first consider the theory behind the LSTM, then the application to this task, and finally the performance in labelling the claims.

5.1 Method

A BiLSTM is an example of a Recurrent Neural Network (RNN), which, when inputting new information, looks over information it has seen before. They are networks with loops, which allow information to persist and for the networks to 'remember'. These networks are typically used for sequence learning, like text prediction or translation, whereby each word is an input. Once we have seen the first word and put it through a layer in the network, we can use the output of this layer to firstly predict the outcome but also to act as an input to the layer for the next word.

As the sequences of words, or even sentences, grow longer, in pratice, RNNs are not capable to connect information that is far apart and are not able to handle 'long-term dependencies'. Instead, Long-Short Term Memory (LSTM) networks, initially introduced by Hochreiter & Schmidhuber [2] are used, which are capable of dealing with the problems faced by RNNs.

LSTMs have multiple neural network layers in every chain and cell state, which is the key of this type of network. Figure below shows this.[1]

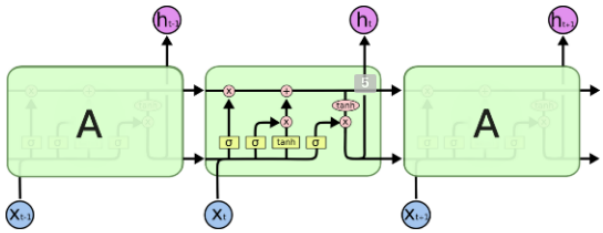


Figure 2: Each chain contains four interacting layers and the cell state.[1]

The cell state is edited by the network as it progresses through structures called 'gates', which are a way to optimially let through information. They are composed of a sigmoid neural net layer and a pointwise multiplication operation. Since the sigmoid returns numbers between zero and one, it describes how much of each component of the input should be let through; a value of zero means the cell state is unaffected, and a value of one means that all the information is added to the cell state.

The LSTM has three of these gates. The first, the 'forget gate layer', decides what information to throw away from the cell state. It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (14)$$

Next, we have to decide what information to store in the cell state. This happens in two parts; firstly, the 'input gate layer' decides which values to update, and secondly, the tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the sate.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \quad (15)$$

Now we can update the old cell state, C_{t-1} into the new cell state, C_t. To do so, we multiply the old state by f_t, forgetting things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$.

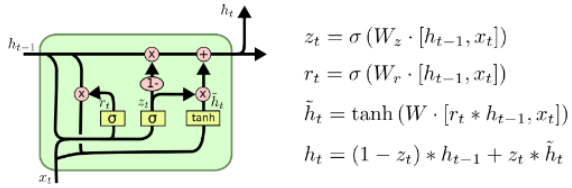
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (16)$$

Finally, we decide on the output, which will be a filtered version of the cell state. First, we use a sigmoid to decide what part of the cell to output. Then, we use tanh, to output between -1 and 1, and multiply by the output of the sigmoid gate, so that we only output parts we want to.

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \quad (17)$$

[1]Figures on this page attributed to Christopher Olah *https://colah.github.io/posts/2015-08-Understanding-LSTMs/*

There are many variants of the LSTM model, one of which is the Gate Recurrent Unit (GRU) [3]. It combines the forget and input gates into a single 'update gate'. It also merges the cell state and the hidden stae. The model is simpler and has fewer parameters to train.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: Gated Recurrent Unit, or GRU, combines the forget and input gates, and cell state and hidden state.[1]

Another variant is the BiLSTM, which essentially does the LSTM sequence twice: once going through the network and once going the opposite way, building two independent LSTM networks. These are then aggregated using an attention function, usually concatenation, then put through densely-connected layer and a two-way output to get a prediction for each of the classifications.

It seems like we should look towards attention functions when considering LSTMs, as we may only want the network to pick information to focus on from a larger collection. We consider this further in sections 6 and 7.

5.2 Implementation

Using an LSTM is a natural choice for this application as we are dealing with sequential data in the form of words and sentences. The sequences of words are then 'remembered' by the network, and words which have been present long ago in the chain may become useful later on. Each cell in the chain will therefore filter out words which are not useful and keep those which are. A BiLSTM is an even better choice as the network is able to aggregate the most imporant words going forwards and going backwards through the sequence.

The inputs for this model are given by the claims, their true label and the sentences that are provided as evidence to support the label. This information is available in the training data. The data is then processed by tokenizing and indexing the words, in accordance with the embedding matrix created with GloVE.

Initially, we create two input channels, one for the claims and one for the evidence for the claims, and each of these goes through an embedding layer using the pre-trained GloVE embeddings of 50 dimensions. We then perform a 'Siamese' BiLSTM with 64 units, such that the claims and evidence sentences go through independent BiLSTMs. Then, we concatenate the outputs from these layers and put them through a dense layer with 64 units and a ReLU activation function. Finally, we use a sigmoid at the output layer to return a two-way classification for whether the claim is true or false.

5.3 Performance

Resource contraints had meant that we were able to train the network on only 3,000 claims and over 3 epochs. Of course, more training data trained over more epochs should constitute better performance. We find that the model achieved 55% validation loss and 68% validation accuracy.

6 RELATED WORK

For inspiration and further development of the machine learning models we have used, we consider three segments of information retrieval literature. This paper's contribution is to combine the techniques we see in the fact-checking tasks and natural language inference, which we measure the performance of in section 7. Firstly, we consider the published work on the same fact checking task, FEVER. Secondly, we consdier other and similar datasets

6.1 FEVER

The current state-of-the-art paper on the FEVER task uses the same three-stage approach as we have, but the authors implement only a Neural Semantic Matching Network (NSMN) for all three stages of the task [4]. Firstly, the document retrieval selects documents from the corpus using semantic matching, as well as commonly-used keyword matching, for improved document ranking. Secondly, for sentence selection, they conduct semantic matching between each sentence from the retrieved documents in the first stage and the claim. Lastly, the claim verifier integrates upstream semantic relatedness features from the second stage and injects ontological knowledge from WordNet into a similar NSMN for natural language inference (NLI), and train it to classify the veracity of the claim. These methods perform with 64.23% accuracy on the test set, almost two-fold the baseline model which receives a score of 33%.

An interesting aspect of their method is that they do not use any intermediate term representations (such as word embeddings) as inputs to the network, and use only the raw textual input. The authors show that the neural network is able to learn the word embeddings itself without the need for user-specified methods. However, this has limitations as the network is not able to fully represent the sentence words. For example, as reported in the paper, it gets confised over the lexical clue of 'capital' when trying to verify whether Berlin is the capital of Germany. This suggests further work is needed to integrate more world knowledge into the model for NLI. This problem may be also addressed by using word embeddings, since words such as 'capital' and 'Berlin' would have a similar embedding.

The second place for this task is held by the UCL machine reading group, who achieve 62.52% accuracy on the test set [6]. For the second stage, they use 300 dimensional GloVe word embeddings as input to their Enhanced Sequential Inferences Model (ESIM), which employs a BiLSTM to encode premise and hypothesis, and also encodes local inferences and information so that the model can effectively exploit sequential information. However, they are limited by the word-level embedding, as they are not contex-dependent.

Other methods considered by teams participating in this task are entity-linking approaches, done by matching the titles of the Wikipedia documents with the claim words [5], Decomposable Attention and a transformer network with pre-trained weights [7].

6.2 Other Automatic Fact Checking Datasets

There are other fact-checking tasks similar to FEVER which employ information retreival techniques to fact-check claims. One of these is CLEF, and we focus on the leading paper on this task. The CLEF 2008 Fact Checking Lab addresses the need to fact-check during political debates. The task is split into first checking the 'check-worthiness' of the claims that are made during the debate, meaning whether the claims are factually dubious. Second, the factuality of the claims is checked given the results from the first task.

The current leading paper used a Recurrent Neural Newtork (RNN) that instead of using pre-trained embeddings, learns sentence embeddings by itself, which are then used to predict the check-worthiness of a sentence [8]. Their network architecture is similar to the one we have used, but it uses GRUs instead of LSTMs due to the significantly smaller training dataset.

The sentence embedding then encodes their semantic and syntactic dependencies, leading to multi-representations of each word. They additionally use pre-trained *word2vec* embeddings. Their approach is therefore on sentence-level features, since while they consider the context of words within a sentence through semantic matching, they do not consider the context of sentences within each document. Considering paragraph- or document-level features may improve on their methods, but this requires a large amount of training data, and more complex semantic and syntactic dependency matching. They also do not proprocess raw sentences (except for tokenization), and do not remove stopwords for all features apart from TF-IDF calculations.

6.3 Natural Language Inference

The literature on natural language inference faces an unsolved problem of predicting the relation between evidence and the claim with ground truth evidence. As we have mentioned before, a prominant part of the literature in this area is focused on attention functions, which are an alternative way to learn semantics for sentence matching to embedding that we have performed in this task. These attention mechanisms are able to capture interactions between words and sentence pairs [11]. We consider two papers that we believe could be combined to provide state-of-the-art approaches in application to the FEVER task.

The first paper develops an attention function for the network to docus on similarities between the premise and hypothesis on the Stanford Natural Language Inference Dataset [13]. Their state-of-the-art results achieves an accuracy of 88.6% with a carefully designed sequential inference model basted on chain LSTMs. The last part of their network is the inference composition, for which they use a composition layer to compose the local inference information. They develop a 'soft alignment layer', which computes the attention weights as the similarity of a hidden state tuple between a premise and a hypothesis. They find that each attention function contributed to an improvement to their final performance.

Building on this, multiway attention network (MwAN) uses numerous attention functions to match two sentences at word level [9]. They use the soft alignment alongside three other attention functions as a layer in a bi-direction RNN after the embedding the two sentences. They then apply another bi-directional RNN to pass through the combined representation, which finally goes through attention-pooling and the output layer. They find that using the MwAN is substantially better than using any single attention function.

7 PROPOSED IMPROVEMENTS ON THE MACHINE LEARNING MODELS

We propose improvements on all three parts of the method we have implemented to conduct this fact checking exercise.

7.1 Document Relevance

The Jelinek-Mercer smoothing method had given us the best scores for recall of relevant documents and sentences, however, it is still a unigram model, as it is assuming independence of each word in the claim. As we have mentioned before, this is a simplifying assumption, as it would be difficult to construct dependencies between each word in each sentence. Instead, we could use bi-grams or skip-gram models which would enable us to compute the probability of pairs or sets of words appearing together, affecting the probability of words occuring in each document.

In a similar vein, it could be helpful to do topic modelling for each of the claims and documents, so that we have a more accurate representation of the probability of a claim word appearing in a document. For example, if the document is about the film franchise *Avengers*, we know it will be classified as movie or more specifically superhero film. Therefore, a claim containing an actor, such as *Robert Downey Jr. is an actor* will be much more likely to appear in documents that are classified as films rather than documents about historical events.

Finally, a further improvement in the input of words, which is relevant for all parts of this task, would be to lemmatize words rather than use the full tokenized versions. Lemmatizing words removes word endings and groups words together that have the same stem. It is easier then to have closer matches between words, and therefore closer embeddings, cosine similarities and so on.

7.2 Logistic Regression

The performance of the model is limited by the number of claims used from the training data to train the weights. Obviously, using more data would lead to better performance. However, there are also other considerations that we can have for this model. When we make the prediction about the relevance of the sentence with the development claims, we specify a threshold for the probability leading to a hard assignment of either 0, not relevant, or 1, relevant. Optimising this threshold for the AUC could give us better results for recall and/or precision and therefore the classification of the claims.

7.3 Neural Network

There are three main aspects of the neural network that can be improved: inputs, architecture and optimisation.

7.3.1 Inputs. As we have seen in section 6, this tasks benefits from multi-representation of the inputs and sentence-level context. Therefore, we included part-of-speech (POS) tagging as a secondary

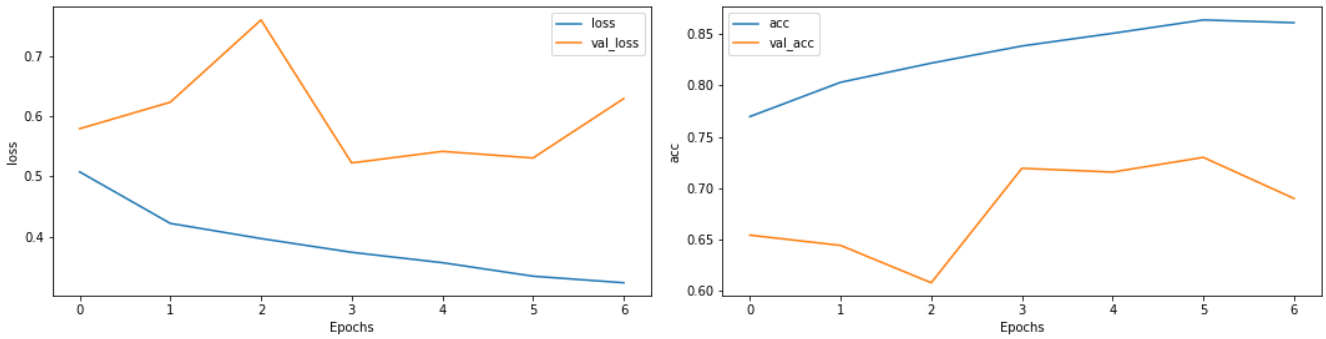


Figure 4: Performance output for the BiLSTM neural network

input to the Neural Network. We included in the claim and evidence sentence channels the part-of-speech tags with a seperate embedding with trainable weights.

Since we embed each word individually, we have word-level context. We do not aggregate to get sentence- or document-level context. For this, we could use trained embeddings such as *doc2vec*. An improvement in semantic matching above topic modelling and attention functions that is particularly relevant for this aspect is the use of dependency trees to find which words depend on each other the most as a way of illustrating the 'importance' of words in a text or sentence. For this, we would be able to use the spaCy package in Python to construct these trees and include them as another representation into the network.

7.3.2 Architecture. When using a sequential network, there is always the choice between using GRUs instead of LSTMs. As we have seen in section 5, GRU memory units are a lot simpler and therefore have fewer parameters to train. In general, we have seen numerous existing works which use GRUs instead of LSTM units. Therefore, we could compare the performance across the two types of memory units in the RNN.

Secondly, specifying a dropout rate is important in order to avoid overfitting. A droupout means that for every forward and backwards pass through the network, a node in a layer is not included in the calculation for that pass with a certain probability [14]. Since we are using a fairly large network, dropout also makes it faster to train.

Most importantly, including sentence matching using attention functions between the dense layers and BiLSTMS would be a move forward towards significantly improved techniques for this task. Based on the literature, we have seen that multi-way attention functions have the ability to dramatically improve performance. To our knowledge, this method has not yet been implemented for fact checking, and would be a novel approach. This is because we would be better able to capture the similarities between the claim sentences and the sentences retrieved from the relevant documents, which would make for better classification and predictions.

7.3.3 Hyperparameter optimisation. We used the scikit-optimize python package to optimise out hyperparameters when defining the neural network. The parameters we optimised were embedding dimension for the part-of-speech tags, the number of hidden units in the BiLSTM layers, the number of units in the dense layer and the number of epochs. We specified a range for each of these parameters for which training the network was feasible and ran the optimisation for over 20 iterations. We found that 7 epochs was the optimal number, with embedding dimensions of 35 and 34, bilstm hidden layers of 31 and 23, 97 in dense, and 0.34 dropout rate.

7.4 Performance

We were able to make some adjustments to the network reported in section 5, including considerations of overfitting and semantic matching. To do this, we added dropout, POS tagging and hyperparameter optimisation. We used a similar structure to the network as previously, but this time we have two sets of input channels into the network, as well as differnt values for the optimized number of units in each layer. The new neural network architecture is shown in figure 5. We report the performance of the neural network with the adjustments made above to the one proposed and tested in section 5 in Table 1.

	val. loss	val. acc.
GloVE embeddings	55%	68%
GloVE, POS, dropout	62%	69%

Table 1: Performance metrics for neural network

Figure 4 shows the performance plots as we train the neural network. Although we did not improve the validation accuracy between the two models, we find that the addition of the extra features in inputs and architecture has decreased overfitting in the model, and we would expect this model to outperform the previous one had we been able to train them on more claims from the training data over a longer period of time.

8 DISCUSSION AND CONCLUSION

The given task was to verify given claims via a three-stage approach of document retrieval, sentence selection, and claim verification using the FEVER dataset consisting of Wikipedia pages. First, we investigated several document retrieval methods, which we tested against recall and found that the Jelinek-Mercer smoothing on probabilistic retrieval was best. We used this method to then implement a logistic regression to identify which sentences within each of the documents were the relevant ones to be used as evidence. Then, we

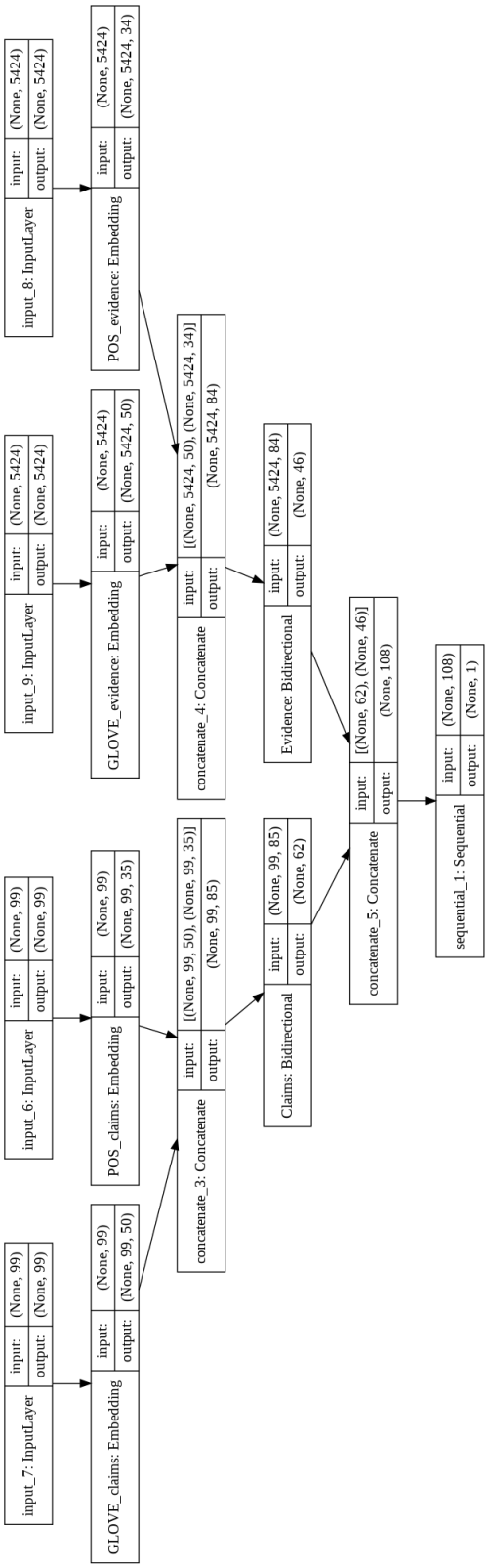


Figure 5: BiLSTM Neural Network Architecture

designed and trained a Bi-LSTM to detect whether or not each claim was true or false, based on the evidence given. We improved upon a standard neural network by using multi-representations for the words, including POS tagging, and hyperparameter optimisation. We found that this network performed with a validation accuracy of 69%.

In order to perform this task to the fullest, we would have to significantly speed up the performance of our document retireval methods. We managed to reach a speed of 10 claims per 2.5 minutes, but with almost 20,000 test claims, it would have taken over a month to retrieve the relevant documents for all of the claims. Moreover, training the network would have taken significantly longer than the hours it takes for 3,000 training claims. Therefore, we propose additional ways in which our implementations of the models can be improved, most notably the use of multi-way attention functions which, to the best of our knowledge, have not been applied to this task.

REFERENCES

[1] Thorne J., Vlachos A., Christodoulopoulos A., Mittal A. *FEVER: a large-scale dataset for fact extraction and verification.* Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1, (2018)

[2] Hochreiter S., Schmidhuber J. *Long Short-Term Memory.* Neural Computation, 9(8): 1735-1780, 1997

[3] Cho K., van Marrienboer B., Gulcehre C., Bougares F., Schwenk H., Bahdanau D., Bengio Y., *Learning Phrase Representations using RNN EncoderâĂŞDecoder for Statistical Machine Translation.* Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (2014)

[4] Nie Y., Haonan C., Bansal M. *Combining Fact Extraction and Verification with Neural Semantic Matching Networks.* Association for the Advancement of Artificial Intelligence, 2019

[5] Hanselowski A., Zhang H., Li Z., Sorokin D., Schiller B., Schulz C., Gurevych I. *UKP-Athene: Multi-Sentence Textual Entailment for Claim Verification.* Proceedings of the First Workshop on Fact Extraction and VERification (FEVER), pp 103âĂŞ108

[6] Yoneda T., Mitchell J., Welbl J., Stenetorp P., Riedel S. *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER).* 'UCL Machine Reading Group: Four Factor Framework for Fact Finding (HexaF)'. pp 97-102, Belgium, 2018

[7] Malon C. *Team Papelo: Transformer Networks at FEVER.* Proceedings of the First Workshop on Fact Extraction and VERification (FEVER), pages 109âĂŞ113, Belgium, 2018

[8] Zuo C., Karakas AI, Banerjee R. *A Hybrid Recognition System for Check-worthy Claims Using Heuristics and Supervised Learning.*

[9] Tan M., dos Santos C., Xiang B., Zhou B. *Improved Representation Learning for Question Answer Matching.* pp 464âĂŞ473. (2016)

[10] Man C., Xu K., Gregory K. *CALYPSO: A Neural Network Model for Natural Language Inference.* Technical report. (2017)

[11] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomes A.N., Kaiser L., Polosukhin I. *Attention is all you need.* 31st Conference on Neural Information Processing Systems (2017)

[12] Zhou X., Zafarani R. *Fake News: A Survey of Research, Detection Methods, and Opportunities.* ACM Comput. Surv. 1, 1 (2018)

[13] Chen Q., Zhu X., Ling Z., Wei S., and Jiang H. *Enhancing and combining sequential and tree lstm for natural language inference.* arXiv preprint arXiv:1609.06038, (2016).

[14] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15 (2014), pp.1929-1958

[15] Olah C., *https://colah.github.io/posts/2015-08-Understanding-LSTMs/.* Accessed 26 April 2019