Information Retrieval and Data Mining Report

Jianyu Liu
University College London
London, UK
jianyu.liu@ucl.ac.uk

1 SUBTASK1: TEXT STATISTICS

In order to count frequencies of every term in the collection documents of 'wiki-pages', we need to remove the symbols that are not terms, convert uppercase letters to lowercase letters and split the sentences into terms. According to the reading of 'wiki-pages', we could find that both of 'text' and 'lines' contain the contents of 'wiki-pages'. For 'lines', it including the sentence number, the synonym of some terms and it is split by '\t' and '\n'. This information has a negative impact on the frequency of statistics. So I use 'text' for term frequency statistics.

By cleaning the 'text' part of the 'wiki-pages', we can get the count and the frequency of each term. By rank the terms in order of decreasing frequency, we can obtained the rank of each term. Zipf's Law refers to the theory that rank of a word times its frequency is approximately a constant. For English, this constant is around 0.1. The result of top 10 words of frequency is shown in the Table 1.

Table 1: Freq, rank(r), Pr(%) and r*Pr of top 10 terms

term	Freq	rank	Pr(%)	r*Pr
the	30746674	1	5.794718997	0.05794719
of	16098465	2	3.034021858	0.060680437
in	14520729	3	2.736671426	0.082100143
an	12695179	4	2.39261635	0.095704654
a	10797630	5	2.034991872	0.101749594
is	7881498	6	1.485398589	0.089123915
to	6671325	7	1.257321482	0.088012504
was	5991677	8	1.129230581	0.090338446
as	3604873	9	0.67939791	0.061145812
by	3515931	1	0.66263532	0.066263532

From the table we could find that rank of a word times its frequency is not approximately a constant. The range of r*Pr is from around 0.058 to around 0.102 and this range includes 0.1. The average r*Pr of top 500 terms is around 0.078 which is close to 0.1. This difference may be due to the omit of word 'the' in some sentences and the number of words is not big enough because the total number of words in 'wiki-pages' is 530598188.

By plot the curve of term probability and rank, we can further an-alyze the deviation. The graph 1 show the curve of term probability and rank of top 100 terms and the curve of $r * Pr = 0.1$.

From the graph we could find that the the word frequency statistics match the expected curve trend. To some extent, we can believe our term frequencies result compliance with the Zipf's Law. The parameters for Zipf's Law of top 500 terms is save in the document

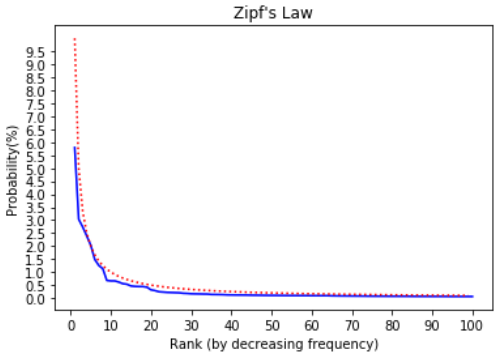

Figure 1: The curve of term probability and rank of top 100 terms

'Q1_term_frequency.csv' and the parameters of all the terms can be run by the code provided in 'Subtask1.ipynb'.

2 SUBTASK2: VECTOR SPACE DOCUMENT RETRIEVAL

In order to improve the efficiency of the calculation, we first preprocessing the data to get the lexicon which needed for the inverted index.

To solve the problem of limited memory, the first step is to use the function 'extract_wiki' to obtain a dictionary which the key is the 'id' of all the documents in the 'wiki-pages' and the value is the 'text' of that document. Then, by using the function 'create_dictory', we could collect the document 'id' of each term in the 'text'. Since the order in which terms appear in the 'text' is not applied in the subsequent calculations, the lexicon for inverted index only including the index term and the 'id' of documents it appear. The output of this function is a dictionary which the key is the term and the value is the 'id' of the documents including this term. From this step, we could also summary that the total number of documents in 'wiki-pages' is 5396106.

By using the lexicon of inverted index, we could calculate the TF-IDF value of each terms in each claim and document. The term frequency for claim refers to the frequency at which a word appears in this claim which is a sentence. The term frequency for document refers to the frequency at which a word appears in this document which is a paragraph. We calculate the term frequency by dividing the number of occurrences of the term in the claim or document by the total number of words appearing in the sentence or paragraph. In order to improve the accuracy, we removed most of the punctuation marks except '-' because '-' is always used in

some proper noun. Also we did not change in capitalization for the same reason.

Inverse Document Frequency (IDF) is used to filter out common words and retain important terms. We could calculate the count of each term in documents by calculate the length of the value of that term in the lexicon of inverted index. The equation of IDF 1 is shown as follow.

$$IDF = log(\frac{number\ of\ documents}{(1 + count\ of\ the\ term\ in\ documents)}) \quad (1)$$

By using IDF and TF of each terms in each claims and documents, we could get the TF*IDF value of each term. The top 5 TF-IDF terms in the 10 claims and the TF-IDF of these terms is save in the 'Q2_claim_TF-IDF.csv'.

To find the five most similar documents of each claim. We compute the cosine similarity between the claim and each document. By using the TF-IDF value of top 5 terms in each claim or document, we obtain a five-dimensional vector which each dimension is the TF-IDF value of a term. The numerator of the cosine similarity equation 2 is the sum of the numbers multiplied by each dimension of the two vectors, and the denominator is the product of the length of two vectors.

$$cosinesim(w, v) = \frac{\sum_{i=1}^{N} w_i \cdot v_i}{\sqrt{\sum_{i=1}^{N} w_i^2} \cdot \sqrt{\sum_{i=1}^{N} v_i^2}} \quad (2)$$

By using the data pre-processing method, TF-IDF value and the equation of cosine similarity. We could obtain the 5 most similar documents and the cosine similarity value of the 10 claims.

The Table 2 show part of the result of vector space document retrieval and the full result of the 10 claims and the five most similar documents with the claim is save in the 'Q2_vector_space.csv'.

Table 2: Document id of top 2 similar documents of cosine similarity method

claim id	doc id_1	doc id_2
75397	New_Amsterdam···	Nikolaj_Coster-Waldau
150448	The_O.C._season_3	Genre_fiction
214861	History_of_art	Leo_Zogmayer
156709	Nate_Butler	Cheetah-licious_Christmas
129629	Homeland_TV_series	Gideon_Raff

From the result above, we could find accuracy of vector space document retrieval is acceptable. These 10 claims have 14 documents that are their evidence and from the top 5 documents we find, we could find 5 of them and the accuracy is 0.36. By viewing the documents we obtain, we could some of the cosine similarity is over 0.9 but it is not including in the evidence. When we read the 'text' of this document, we could find that people can infer the claim from the document but they are sometimes not intuitive. For example, we have a claim say 'The Boston Celtics play their home games at TD Garden'. And from the document 'TD_Garden' we retrieval by cosine similarity, we could find the sentence 'TD Garden is the home arena for the Boston Bruins of the National Hockey League and the Boston Celtics of the National Basketball Association.' and people can understand this two sentences have the similar meaning but this document is not including in the evidence. Overall, the result of our method is acceptable and it could find the documents have the similar meaning.

3 SUBTASK3: PROBABILISTIC DOCUMENT RETRIEVAL

In addition to cosine similarity, we can also use probabilistic retrieval models to retrieve the five most similar documents. To distribute the probability over words, we can use the unigram language model. By using query-likelihood unigram language model, we can rank documents by the probability that the query could be generated by the document model. For the data pre-processing part, we just need a dictionary which the key is the term and the value is the frequency of the term in the 'wiki-pages'.

3.1 Query-likelihood unigram language model

For the query-likelihood unigram language model, we need to calculate the probability of each term in the claim. In the original unigram query-likelihood model, the probability of a term is the count of the word in the document divided by the total number of word. So for the term in the claim but not exist in the document, the probability is 0. In order to increase the accuracy of the model, I smoothing the model with background. For the term not exist in the document, the p value is the probability it exist in the 'wiki-pages', or 0.001 if it also do not exist in the wiki-pages. By calculating the logarithm of the sum of probability of each term in the claim to the document, we could get the query-likelihood unigram language model mark and the higher the score, the higher the similarity between the claim and document. By descending order the documents, we could get the top five similar documents of each claim.

The Table 3 show part of the result of query-likelihood unigram language model and the full result of the 10 claims and the five most similar documents with the claim is save in the 'Q3_unigram.csv'.

Table 3: Document id of top 2 similar documents of query-likelihood unigram language model

claim id	doc id_1	doc id_2
75397	New_Amsterdam···	Ved_verdens_ende
150448	Bedside_Press	Appticles
214861	History_of_art	Fine_art
156709	Empire_Girls···	All_You've_Got
129629	Homeland_season_6	Homeland_season_2

The accuracy of query-likelihood unigram language model based on the 10 claims is 0.43 which is better than the cosine similar method. The amount of data is too small, so to compare the two method, we need to test more claims.

3.2 Laplace Smoothing

Laplace Smoothing is always used to improve the performance of the model. This method plus 1 to every count but it gives too much weight to unseen terms. I also smoothing the model with background based on the Laplace Smoothing. The Table 4 show part of the result of query-likelihood unigram language model with Laplace

Smoothing and the full result of the 10 claims and the five most similar documents with the claim is save in the 'Q3_laplace.csv'.

Table 4: Document id of top 2 similar documents with Laplace Smoothing

claim id	doc id_1	doc id_2
75397	New_Amsterdam···	The_Other_Woman···
150448	Only_Much_Louder	Tosyn_Bucknor
214861	History_of_art	The_arts
156709	Empire_Girls···	Loni_Love
129629	List_of_Homeland···	Homeland_season_2

The accuracy of Laplace smoothing based on the 10 claims is 0.5 which is the best in the 4 methods. We can consider using the output of Laplace smoothing as the input of the following sub tasks.

3.3 Jelinek-Mercer Smoothing

Jelinek-Mercer Smoothing is a kind of interpolation method which use background probabilities. In this model, for the term exist in the 'wiki-pages', the probability of it including two parts. One is the frequency of the term in the document multiplied by α, constant which is independent of document and query query, and another part is the frequency of a term in the background, in this project is 'wiki-pages', multiplied by the 1- α. The sum of this two parts represents the probability of this word. By calculating the logarithm of the sum of probability of each term in the claim to the document and descending order the scores, we could get the top five similar documents of each claim based on the Jelinek-Mercer Smoothing. The Table 5 show part of the result of query-likelihood unigram language model with Jelinek-Mercer Smoothing and the full result of the 10 claims and the five most similar documents with the claim is save in the 'Q3_jelinek_mercer.csv'.

Table 5: Document id of top 2 similar documents with Jelinek-Mercer Smoothing

claim id	doc id_1	doc id_2
75397	'New_Amsterdam···	Nikolaj_Coster-Waldau
150448	Only_Much_Louder	Joel_Spolsky
214861	History_of_art	The_arts
156709	Empire_Girls-COLON···	All_You've_Got
129629	Homeland_season_2	Homeland_season_3

The accuracy of Jelinek-Mercer smoothing based on the 10 claims is 0.43 which is the same as the original query-likelihood unigram language model. But we could find in the result that this model find many film which have the save name as the evidence but the year of the film is different.

3.4 Dirichlet Smoothing

To avoid the problem of Jelinek-Mercer smoothing that longer documents provide better estimates, we could also use Dirichlet Smoothing which provide less smoothing. First, we need to calculate the μ which is number of word occurrences in the 'wiki-pages' divided by the number of documents. The probability of a term in the claim including two parts. One is the number of words in the claim divided by the sum of it and μ, and multiplied by the count of this word in the claim. Another part is the μ divided by the sum of the number of words in the claim and μ, and multiplied by the frequency of the term in the 'wiki-pages'.The sum of this two parts represents the probability of this word. By calculating the logarithm of the sum of probability of each term in the claim to the document and descending order the scores, we could get the top five similar documents of each claim based on the Dirichlet Smoothing. The Table 6 show part of the result of query-likelihood unigram language model with Dirichlet Smoothing and the full result of the 10 claims and the five most similar documents with the claim is save in the 'Q3_dirichlet.csv'.

Table 6: Document id of top 2 similar documents with Dirichlet Smoothing

claim id	doc id_1	doc id_2
75397	New_Amsterdam···	The_Other_Woman···
150448	Joel_Spolsky	Brett_Atwood
214861	History_of_art	The_arts
156709	Empire_Girls-COLON···	Loni_Love
129629	List_of_Homeland···	Homeland_TV_series

The accuracy of Dirichlet smoothing based on the 10 claims is 0.43 which is the same as the orginal query-likelihood unigram language model.

The result of different smoothing method and original query-likelihood unigram language model is similar. This may be because I smooth the original model with background and the test set is too small. The accuracy of this four model is over 0.4 which is acceptable.

4 SUBTASK4: SENTENCE RELEVANCE

We need four steps for sentence relevance. First we need to retrieval the five most similar sentences of the claims using for training. And we need to label the sentences in these documents. From the train set, we can extract the evidence document id and the sentence number. With the help of it, we can label the sentences in the document with '1' if it is the evidence sentence, otherwise we label the sentence with '0'. Secondly, we need to represent the claim and the sentences based on a word embedding method. Thirdly, we need to sampling the data from the embedding result and use it as the training set of the logistic regression model. Finally, we can use the model to predict the data in the validation set and compare the output with the label to evaluate the performance of the logistic regression model.

In order to achieve the above steps, we need to pre-process the data. We need similar process for training set and development set and we are here to use the training set as an example. Firstly, we need to find the five most similar documents of each claim. By comparing the accuracy of each method, we use the Laplace smoothing of query-likelihood unigram language model in this project. We retrieval the first 150 claims in the train set and 113 of them including at least one evidence. We also need to use the function 'Subtask4_pre_train_3' to output a dictionary which the

Jianyu Liu
University College London
London, UK
jianyu.liu@ucl.ac.uk

key is the document 'id' that appear in any of the claim's five similar documents and the value is 'lines' in wiki-pages. We use the 'lines' instead of 'text' because the sentence number in the evidence corresponds to the position of the sentence in the 'line' instead of the position of the sentence in the 'text'. By reading the data, we could find that this is because some sentence in the 'line' is empty. The function 'Subtask4_pre_train_4' provide a list which including the claim, evidence document id and the sentence number and it could help us to label the data.

4.1 Word Embedding

In the word embedding process, we use the gensim package to read 'GoogleNews-vectors-negative300.bin'. This allow us to use the Word2vec embedding method [9]. The output of the Word2vec is a 300-dimensional vector. To calculate the vector of a sentence, We sum the output of the vector representation of each word one by one at the corresponding dimension and then scaling the vector to unit length. Next, we connect the vector of the claim with the vector of each of the sentence in the five most similar documents and label the relationship by '1' or '0'. Finally, we obtain a 601-dimensional vector which the first 300 dimensions is the embedding result of claim, following 300 dimensions is the embedding result of sentence and the last dimension is the label. The embedding result is save in the 'traindata_Subtask4.txt' and 'devdata_Subtask4.txt'

4.2 Data sampling

To sampling the data and used as the input of logistic regression model, we tried no sampling, under sampling and over sampling. In our training set, we have 153 positive samples and 3201 negative samples. In the logistics regression model, we tried to optimize the model by stochastic gradient descent and randomly select samples which could help us reducing cycle fluctuations. The hyper-parameters in the model is the learning rate and max iteration.

4.3 Logistic regression model Evaluation

Due to the severe class imbalance in our data set, the negative samples are much more abundant than the positive samples. The horizontal axis of the ROC curve is false positive rate, and the vertical axis is true positive rate. The robustness of the ROC curve lead to the area under the curve ROC (AUC) does not significant change in the unbalanced samples. So in the Subtask4, we evaluate our model by the ROC curve.

By using different sampling methods, stochastic gradient descent and hyper-parameters, we can get the root-mean-square error(RMSE), ACC (accuracy), AUC, train loss and test loss of each model. The best result we get use the under sampling to obtain 153 positive samples and 153 negative samples, stochastic gradient descent, learning rate is 0.0001 and max iteration is 5000. The RMSE of the model is 0.70, the accuracy is 0.51, the AUC is 0.64. The under sampling training data is save in the 'train_x.dat' and 'train_y.dat'.

From the ROC curve we could find the AUC is 0.64, which is significant higher than 0.5 and the result is much better than random result.

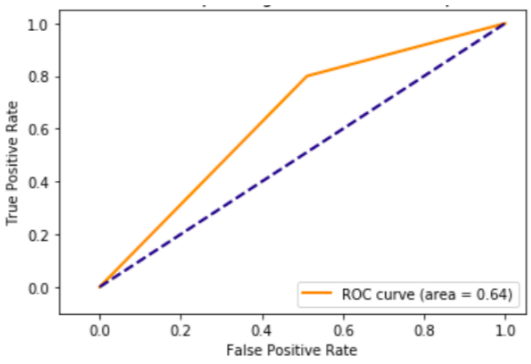

Figure 2: The ROC curve and evaluation result

4.4 Effect of Learning Rate

Based on this model, we keep the factors other than the learning rate unchanged, and try to analysis the effect of learning rate on the model training loss. From the result we could find that as the learning rate decrease, the speed of train loss reduction is gradually decreasing. When the learning rate is over 1, the train loss will keep reducing and fluctuating around the minimum train loss. The training time do not change significantly because the training data is too small.

5 SUBTASK5: RELEVANCE EVALUATION

Recall, precision and F1 metrics is always used to evaluate the classification model. In our model, the precision is the number of true prediction on the positive sample (TP) divided by the total number of positive prediction. The Recall is the TP divided by the total number of positive sample. The F1 is double the recall multiplied by precision, and divided this number by recall plus precision.

Base on the algorithm above, we could get the evaluation result of our model. The precision is 0.08, the recall is 0.8 and the F1 score is 0.15. This result show that our model give much more positive prediction than the real situation. This problem can result in many sentences which is not evidence that retrieved as the evidence sentence.

6 SUBTASK6: TRUTHFULNESS OF CLAIMS

By using neural network, we want to judge whether the evidence sentence supports or refutes the claim. For the data pre-processing, it is similar like the process in the Subtask4. The only different is the output is a 601-dimensional vector which the first 300 dimensions are the embedding result of the claim, the following 300 dimensions are the embedding result of the evidence sentence and the last dimension is the label of truthfulness. For the 'SUPPORTS' sentences, the label is '1' and for the 'REFUTES' sentences, the label is '0'. The embedding result is saved in 'traindata_Subtask6.txt' and 'devdata_Subtask6.txt'

6.1 Customize my Estimator

In order to architect my own neural network, I used Tersorflow to customize my Estimator instead of using DNNclassifier function in Tensorflow.

The input layer deliver the input data features into the feature column, and later it will be passed to the hidden layer. In the hider layer, we use the cycle to architect the hider layer which allow us to define the number of hider layer and the number of neurons by ourselves. In the output layer, we do not activation the following layer so it can be the output.

The loss function can calculate the degree of deviation of our model. The larger the loss function, the larger the deviation of our mode from the true result and the less accurate of our model. In order to reduce loss, we can use the optimization methods to optimize the model. In our model, we use the cross-entropy loss function provided by Tensorflow to calculate the degree of loss because this method is very effective for classification problems.

For the optimizer in our training process, we tried Adam algorithm [8], Adaptive Gradient algorithm, RMSProp algorithm, Momentum algorithm and gradient descent algorithm. After comparing the model performance, we use the Adam algorithm as the optimizer in our model. This algorithm combine the advantage of RMSProp algorithm and Momentum algorithm.

6.2 Evaluation

In the evaluation process, we output the accuracy, AUC, precision and the recall of our model. For the data sets, the number of training data is 7632 and the number of test data is 588. The test set is approximately 8 percent of the training set.

We get a good performance in the parameters as follow. The loss function is cross-entropy loss. The optimizer is Adam algorithm and the learning rate is 0.001. The number of hidden layer is 2 and for each layer the number of neurons is 512. The batch size is 128 and the steps is 2500. The evaluation result is shown in the Table 7

Table 7: The evaluation result of neural network

metrics	value
AUC	0.65
accuracy	0.65
final train loss	0.05
test loss	2.14
precision	0.6
recall	0.83

The result is acceptable and the AUC is higher than 0.5 which is mean that the model is better than random sampling.

6.3 Model motivation

The following is some motivation behind my proposed neural architecture.

We use cross-entropy loss because it describe the distance between two probability distributions and it always perform well in classification problem. We use the Adam algorithm as the optimizer in our model because it is very efficiency and independent adaptive learning rate is designed for different parameters by calculating the first moment estimation and second moment estimation of the gradient.

The learning rate I tried from 0.0001 to 0.8 and it effect the speed of train loss reducing. The smaller the learning rate, the slower speed of reducing loss. A suitable learning rate can not only avoid missing any local optima but also avoid over the local optima directly. So I try the learning rate for 0.01 base on the validation of the model.

The reason of that the number of neurons in each hidden layer is 512 is that the dimension of input data is 600 and the number of neurons is around 75 percent of the dimension of input data. Increasing the number of hidden layers can reduce network errors and improve accuracy, but it also complicates the network, which increases the training time of the network and the tendency to over-fitting. I tried the model with 1, 2, 4 and 8 hidden layers and finally I find two hidden layers have a better performance in my model.

In conclusion, the novelty of my network architecture is the usage of most property loss function and optimizer on a multi-layer perceptron. By adjusting the hyper-parameter based on the validation process, we get a good performance in our model.

7 SUBTASK7: LITERATURE REVIEW

In the past few years, the amount of information on the Internet has been increasing, and the convenience of network sharing enables the rapid transmission of information on the Internet. At the same time, the number of fake or misleading contents on the Internet has increased, which makes the fact checking and misinformation detection more and more important. While it has received a lot of attention in the context of journalism, fact checking and misinformation detection is important for other domains like product reviews and scientific publications.

The Fact Extraction and Verification (FEVER) shared task provides a data set for fact checking and misinformation detection, which allow us testing the ability of the model on a corpus of around 5.4M Wikipedia articles.

7.1 Baseline model

The original dataset description paper provide a basic fact checking model [11]. This model including three components: document retrieval, sentence selection and recognizing textual entailment. For the document retrieval, the model use the document retrieval component from the DrQA system [3] which using cosine similarity between binned unigram and TF-IDF vectors to find the k nearest documents for a query. For the sentence selection, the model ranks sentences by TF-IDF similarity between claim and each sentence in the retrieved document. Validation accuracy on the development set is used for tune a cut-off. For the textual entailment, the model use a multi-layer perceptron with a single hidden layer which uses TF-IDF between the claim and evidence sentences as features to judgment the truthfulness of claims.

The advantage of this model is the model is the model is simple and straightforward. Using TF-IDF reduce the negative effects of meaningless high-frequency words in a document. The problem of this model is the efficient of the original retrieval component is around 55% which is mean that the accuracy reduce rapidly and

Jianyu Liu
University College London
London, UK
jianyu.liu@ucl.ac.uk

the beginning and the model. Also, the multi-layer perceptron in the model is so simple that the aggregating retrieved evidence by concatenating all the sentences into a single paragraph. The performance of this model is acceptable which the precision is 11.28%, recall is 47.87%, F1 is 18.26% and the label accuracy is 48.84%.

7.2 HexaF

The Four Factor Framework For Fact Finding (HexaF) model [12] is an improved fact checking model which is more effective in FEVER data. The model including four components: document retrieval, sentence retrieval, multi-layer perceptron and aggregation. Document retrieval aims to find the 'id' of documents in 'wiki-pages' in the claim, and then ranks the articles based on capitalisation, sentence position and token match features. For the sentence retrieval, token matches with the claim and position in the article is used to retrieved the most relevant sentences from the top ranked documents. For multi-layer perceptron, Enhanced Sequential Inference Model (ESIM) [4] is adopted being used and the input is each of the retrieved sentences paired with the claim and the output giving a prediction for each pair of claim and the potential evidence sentence. For the aggregation part, HexaF use a multi-layer perceptron to re-ranked the potential evidence sentences so that the evidence which is consistent with the final prediction are placed at the top.

The advantage of HexaF is it building a dictionary of documents titles and construct an initial list of potential document. Logistic regression model is used based on the position and capitalisation within the claim, presence of stop words. The model take into account the position of the sentences which allows the model to judged the meaning of the sentence by the order of sentences. Replace the symbol and using stop word dictionary also help the model. HexaF found that the evidence sentences appear at the beginning of the documents more often than other positions and use this to improve the accuracy of logistic regression model. Instead of concatenates all evidence sentences and feeds them into the NLI model, HexaF generate NLI predictions individually for each predicted evidence, thus processing them in parallel. They also find that the evidence sentences always including a pronoun referring to the subject of the article in the 'line', which is ignored in my model, and HexaF prepend the corresponding title of the article to the sentence along with a separator.

The problem of this model is it do not take into account the optimization of numerical expressions. Some numerical information like the year of the event is an important part to help the model judge the sentence. It is do not use pre-trained word embeddings or document embeddings to dealing with complex sentences.

HexaF is very efficiency on the FEVER data set which the precision is 22.16%, recall is 82.84%, F1 is 34.97% and the label accuracy is 67.62%. The recall and the accuracy of the model is really efficiency and this model outperforms baseline model on every metrics.

7.3 NSMN

The Neural Semantic Matching Network (NSMN) cite[10] is the model got highest score based on FEVER. NSMN model is a modification of the ESIM. Based on the validation results, the model add shortcut connections from input to matching layer and change output layer to only max-pool plus one affine layer with rectifier activation. This model is used in the each parts of the fact checking process. The matching score between the claim C_i and the j-th document is computed as: 3 4

$$\left\langle m^+, m^- \right\rangle = NSMN\left(c_i, \left[t_j, s_j^0\right]\right) \tag{3}$$

$$p\left(x = 1 | c_i, j\right) = \frac{e^{m^+}}{e^{m^+} + e^{m^-}} \tag{4}$$

For the document retrieval, the system used NSMN for semantic understanding because they found that 10% of the documents have extra information in their 'id' which is using to disambiguation. To narrow down the search space, keyword matching is using to retrieved the 8 most similar documents of a claim. Then they identify the documents not 'disambiguative' and add to the result list. For the 'disambiguative' document, they use NSMN model to identify the documents with $p\left(x = 1 | c_i, j\right)$ and m^+ value. In the sentence selection, all the sentences in the retrieved documents is comparing with the claim by NSMN. $p\left(x = 1 | c_i, j\right)$ and m^+ of each pair of claim and sentence is calculated and the sentences with $p\left(x = 1 | c_i, j\right)$ lower than the threshold is excluded from evidence sentences. The rest of the combinations of claim and sentences are sorted by m^+ and the top 5 sentences is identified as the evidence sentences. In the claim verification part, NSMN with additional token-level features is used for logical inference from evidence to the claim. Without using GloVe and ELMo embeddings, a 30-dimension indicator features regarding ontological information and a 5-dimension real-value embeddings to number token is used in this process.

The advantage of this system including three parts. Firstly, instead of using traditional information retrieval method like TF-IDF, this system use NSMN to optimize the semantic matching at the granularity of sentences. From the result we could find that this model if significantly more efficient than term-weighting based methods. Secondly, the system combined the claim verification module with evidence retrieval module and improve the embedding of numerical information and increase the effect of ontological information. Thirdly, the system allow input the claim and evidence sentence directly to the neural systems by formalize the different parts of similar textual semantic matching problem. Normalized semantic relatedness score is used in the model as 2-dimension features for each token to support connecting the different parts strongly.

Overall, the NSMN model support the work of different part of the system. And some extra information like the word using for disambiguation and embedding the numerical data is also used to improve the model. The performance of keyword method is significantly better than the traditional TF-IDF method and the accuracy is 99.86% in top 5 documents and for the top 10 documents is 90.69%. The accuracy of this model is the highest and the accuracy is 68.21%. The precision is 42.27%, the recall is 70.91% and the F1 value is 52.96%.

7.4 Knowledge Graph

In addition to the above models, knowledge graph has also been gradually applied to fact checking and misinformation detection. Knowledge graph is used to translate the relations among entities as

changing from head entity to tail entity. TransE is the representative of this type of model. TransE [2] learns vector embedding for both entities and relationships. For a triple (h, r, t) which including the relation r between head entity h and tail entity t. The basic idea of TransE model is that this triple induce a functional relation corresponds to a translation of the embedding of entities in vector space. (Equation 5).

$$h + r \approx t \tag{5}$$

By constructing the knowledge graph, we can predict the tail by the head and relationship, so as to verify the authenticity of a sentence. For the truth new statement, if it is an edge of the Knowledge Graph or if there is a short path linking its head to its tail within the Knowledge Graph, we expect it to be true. If there have neither edges nor short paths that connect head and tail, it is not true. More in general, fact checking and misinformation detection is a kind of special case of link prediction in knowledge graphs.

Wikipage Knowledge Graph (WKG) [5] is used to architect fact checking system based on DBpedia database [1]. The WKG including 3 million entity nodes linked by around 23 million edges and it only only facts within infoboxes. For example, the triple (head, relation, tail) of the sentence 'Nikolaj Coster-Waldau worked with the Fox Broadcasting Company.' out to be the (Nikolaj Coster-Waldau, work, Fox Broadcasting Company) and this triple can help us check the fact of similar claims. The AUC of WKG in subject-predicate-object statements is 0.55 and for statements with 'degree' predicate if 0.65 and it is significantly better than the random result.

8 SUBTASK8: FINAL MODEL

In the final model, my novelty of the fact checking system architecture is to use a improved convolutional neural networks for sentence classification [7], which trained the vectors on 100 billion words of Google News [9].

For the document retrieval, I used the TF-IDF and cosine similarity to retrieve the five most similar documents of the claims in the test set like the baseline model. For the sentence relevance, I ranked the sentences by the cosine similarity value of each sentences in the similar documents with the given claim. For the truthfulness of claims , I used the convolutional neural networks for sentence classification. The model including a convolutional neural networks with one convolution layer on top of word vectors obtained from an unsupervised neural language model. The output is a 300-dimensional vector and were trained using the continuous bag-of-words architecture. Initialized randomly is used for the words not present in the set of pre-trained words. The model architecture is a slight improvement of the CNN architecture of Collobert [6]. The model uses multiple filters to obtain multiple features. These features become the penultimate layer and are passed to a fully connected softmax layer. The probability distribution over labels is the output.

Another novelty of my model is my model can distinguish 'SUPPORT', 'REFUTES' and 'NOT ENOUGH INFO'. The output of the original model is 'POS' and 'NEG'. By judge the output value of the model and data validation, I use the value 0.2 as the threshold to identify 'NOT ENOUGH INFO' claim.

For the model, we use rectified linear units and the filter windows is 3, 4, 5 with 100 feature maps each. The dropout rate of the model is 0.5 and the size of mini-batch is 50. These values were chosen via a grid search on the SST-2 dev set and the code is available from: https://github.com/gaussic/text-classification.

By using this model on our test set, the accuracy of predict the two-class truthfulness of claims is 75.64% and this accuracy is significantly better than the result in Subtask6 which use the neural network based on my customized Estimator which accuracy is 65%. By integrate this model with previous work, we could get the final prediction of the label and evidence of the test set. The final result is saved in the 'Q8_result.txt'.

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2787–2795.

[3] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 1870–1879. https://doi.org/10.18653/v1/P17-1171

[4] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for Natural Language Inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 1657–1668. https://doi.org/10.18653/v1/P17-1152

[5] Giovanni Luca Ciampaglia, Prashant Shiralkar, Luis M Rocha, Johan Bollen, Filippo Menczer, and Alessandro Flammini. 2015. Computational fact checking from knowledge networks. *PloS one* 10, 6 (2015), e0128193.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, Aug (2011), 2493–2537.

[7] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[10] Yixin Nie, Haonan Chen, and Mohit Bansal. 2018. Combining Fact Extraction and Verification with Neural Semantic Matching Networks. *arXiv preprint arXiv:1811.07039* (2018).

[11] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355* (2018).

[12] Takuma Yoneda, Jeff Mitchell, Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. 2018. UCL Machine Reading Group: Four Factor Framework For Fact Finding (HexaF). In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*. Association for Computational Linguistics, Brussels, Belgium, 97–102. https://www.aclweb.org/anthology/W18-5515