Automated Fact Checking

COMP0084 Information Retrieval and Data Mining

18070639
Computer Science
University College London
London, United Kingdom
qiqi.zeng31@gmail.com

ABSTRACT

This research in automated fact checking is increasingly important as the fake information online is rampant today. This report researches and analyses the widely used methods for automated fact checking, including the methods for retrieving relevant documents, telling relevant sentences and predicting truthfulness. The methods to solve these problems will be described in detail.

1 Introduction

The increasing concern with online misinformation triggered the research in automated fact checking. This report will represent a complete process of fact checking, including data analytics, document retrieval, relevant sentences retrieval and truthfulness prediction.

The data used in this project is the publicly available Fact Extraction and Verification (FEVER) dataset. This dataset provides a collection of documents collected from Wikipedia pages ('wiki-pages' file), which is used as the corpus in this project. In addition, there are 185,445 claims, of which the labeled file 'train.jsonl' and 'dev.jsonl' are available for training and evaluating models, and a test file 'test.jsonl' for final test. The programing language used in this project is Python3.

2 Methods and Results

2.1 Subtask 1: Text Statistics

To have a general understanding of the verification corpus, this task counted the frequency of word occurrence in the collection of documents. In the first step, all of the sentences in the 'text' filed was loaded from the wiki-pages file and was divided into single terms. Specifically, the term dividing method was based on the space and punctuations but leaves the abbreviations like 'U.S.A.' and optional internal hyphens, such as "can't". To avoid duplicately counting the same word, each term had been converted to lower case and lemmatized. Finally, there are 4,104,303 terms derived. Then all the words were collected and the frequency of them were counted. The visualization of the word count distribution is showed below:

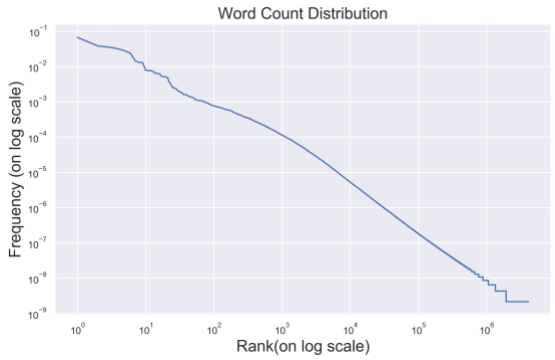

Figure 1: Word Count Distribution

As this figure shows, it is distributed fitting a straight line on the log scale, revealing that only few words occur very often and most of the terms seldom occur, which is a distinct feature of power law distribution. By calculating the product of frequency and rank, the result showed an approximately a constant at around 0.08. These features demonstrate that the word count distribution is likely to follow the Zipf's law distribution.

As Zipf's law is an example of a power law distribution, it follows the formula:

$$\log\big(P(k)\big) \sim -\alpha \cdot \log(k). \quad (1)$$

The parameter α can be calculated by Least Square method. Using its derivation formula:

$$-\alpha = ((X^T \cdot X)^{-1} \cdot X^T y \quad (2)$$

In this case, the X is rank, and y is word frequency. The α=1.34.

2.2 Subtask 2: Vector Space Document retrieval

In this subtask, the top 5 relevant documents in Wike pages were selected out according to the cosine similarity of sentence vectors, which is represented based on term frequency-inverse document frequency (TF-IDF), between the claims and documents. Before calculating TF-IDF, an inverted index dictionary of corpus was calculated to accelerate the calculations later. Traversing each sentence in Wike pages documents, sentences were divided into words using almost the same method in subtask 1. The only

difference is that stop words was removed in this task to reduce the data noise. Then traverse each word in that sentence to check if it is in the pre-built dictionary. If not, adding the new word as the dictionary key, and save its documents id and the word count in that sentence as the dictionary values. Then this dictionary was saved as backup (This inverted index documents would be applied to subtask 2,3 and 4).

The term frequency (TF) of claims is firstly calculated using the formula: TF(t) = Number of times term t appears in a document / Total number of terms in the document (Here a claim is the document). The same formula was also applied to calculate the TF for all the documents in the documents collection, based on the words that appeared in claim. In this case, the number of times for a term appears in a document had already been saved to inverted index dictionary, so it could be got from dictionary value without calculation, which greatly reduced the computational cost.

However, the term frequency does not suggest the importance of a word. The most significant word should be the word that seldom appears in corpus but appears frequently in this document. Thus, the inverse document frequency (IDF) was introduced to balance this problem: IDF(t) = log (Total number of documents / Number of documents with term t in it + 1). As the word vectors are the same in both claims and documents, IDF was calculated only once for both claims and documents, using the inverted index dictionary.

Finally, the TF-IDFs for claims and documents were got by multiply IDF with TF of claims and documents respectively. The similarity between documents and claims was calculated by the cosine similarity:

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \quad (3)$$

where A is the TF-IDF represented vector for claims and B for documents. The top 5 similar documents were selected out by choosing the most similar 5 documents, and the retrieval results has been saved to 'q2.csv'.

2.3 Subtask 3: Probabilistic Document Retrieval

In addition to the vector space document retrieval, this subtask provides an alternative method, query-likelihood unigram language model, to retrieve the documents. To avoid zero-score due to one word missing from document, three smoothing methods were applied to this model and their performance were compared

The query-likelihood model measures the probability of generating a given query, which can be calculated using the formula:

$$P(Q|M_D) = \prod_{i=1}^{k} P(q_i|M_D) \quad (4)$$

In this project, the inverted index dictionary was applied to improve the computational efficiency. Traversing each document, the probability of a claim generated against each document is calculated by multiplying the words count of claim words appeared in the documents and dividing by the total length of the document. However, for many documents, a missing claim word in that documents led to the zero probability as a whole, preventing the comparison among them.

A solution is smoothing. The Laplace Smoothing gives weights to every word so that the probability for those documents with a missing word would not be zero. For the probability of single word against a single document, the formula is

$$P(w|D) = \frac{tf_{w,d}+1}{|D|+|V|} \quad (5)$$

where V is the vocabulary size, D is the document length and the tf had been saved in inverted index dictionaty. To avoid overflow problem, accumulation on log values was used instead of multiplicative multiplication on original values. Another two smoothing methods were Jelinek-Mercer Smoothing and Dirichlet Smoothing, following the formula:

$$P(w|D) = \lambda \cdot \frac{tf_{w,d}}{|D|} + (1-\lambda)\frac{cqi}{|C|} \quad (6)$$

where the C is the number of total words in the collection, cqi is the word frequency in collection. The λ is 0.5 for Jelinek-Mercer Smoothing, and N/(N+μ) for Dirichlet Smoothing, where the N is the length of a single document and μ is the average document length.

The retrieval result suggested the problem of zero-scores. Due to the missing words in many documents, many of them got zero probabilities, which led to the same result for them even though they have different relevance to a same claim. Thus, only 3 out of 10 retrieval results include the documents that mentioned in the 'evidence' label of claims, which suggested a quite non-relevant result indirectly. However, with smoothing, the zero-score was avoided, and the documents were more comparable. The best performed result from Dirichlet Smoothing includes 7 documents that mentioned in the 'evidence' label of claims, and this method will be applied to subtask 4.

2.4 Subtask 4: Sentence Relevance

The forth subtask started to tell the relevance of a sentence in a document using logistic regression model, and the training result would be evaluated based on evaluation metrics.

Collecting data is the first step for this task. The sentence was divided into positive side (1), which means the relevant sentences, and negative side (0), meaning the non-relevant sentences. The positive sentences were got from the 'evidence' of claims dictionary in train.jsonl, and the negative sentences were derived by negative sampling method. Specifically, the negative sampling firstly retrieved the top 5 relevant documents for the first 1000 claims in train data, because sentences in these documents would be more similar to positive sample, therefore the information contains in the training data could be more detailed and the model might better distinguish the sentences label in validation data and test data. To balance the training data, the negative sentences ended by punctuation were random picked from these top 5 relevant documents for each claim, with the same amount of positive data (so that the positive: negative = 1:1).

Then, to make these samples trainable, those sentences would be converted to vectors and the Glove pre-trained embedding were introduced to improve the training efficiency. In this project, the 300d glove model were applied. Loading the embedding model, the words in each claim and sample sentence were embedded into a 300d vectors. When using embedding words, the new words were added to the original Glove model if they were not included in before. These words were not excluded from training data because some terms such as name of person, name of place, and years, play important roles in relevance classification. Sentence vectors were constructed by sum pooling these word vectors, represented as 300-d vectors. The claim vector and document sentence vector were connected by horizontally concatenation, deriving an 600d vector as logistic regression model input.

The logistic regression model was trained by updating the gradient using the formula:

$$\theta_j := \theta_j :- \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (7)$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (8)$$

and the training process was supervised by the loss using the cost function:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] \quad (9)$$

However, as log is applied to the cost function, the overflow problem might happen due to some extreme value, leading to Nan for cost computation. A formula deformation solved this problem. Through a series of deductions, the new formula was derived: using

$$J(\theta) = \log(1 + \exp(-x)) + (1 - y) * x \text{ when } x \geq 0 \quad (10)$$

and

$$J(\theta) = \log(\exp(x) + 1) - y * x \text{ when } x < 0 \quad (11)$$

In order to standardize the range of scalars in input vectors, scalars in input vectors were normalized using formula:

$$new\ x_k^{(i)} = \frac{x_k^{(i)} - mean\ of\ x_k}{standard\ deviation\ of\ x_k} \quad (12)$$

and to overcome the overfitting problem, the cost function and gradient were regularized using formulas:

$$new\ J(\theta) = J(\theta) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \quad (13)$$

and

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} J(\theta) + \frac{\lambda}{m} \theta_j \quad (14)$$

respectively.

To train this model, several learning rates were tried to optimize the performance of this model. The optimization results are showed below:

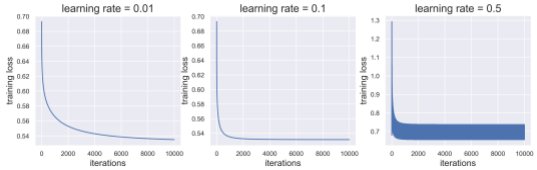

Figure 2: Impact of Learning rate on training performance

On the training data, the loss decreases steadily on the left two figures. When learning rate = 0.01, the training loss drops relatively slow. It takes about 10000 iterations to converge to the minimum, at a value just blow 0.54. When rise the learning rate to 0.1, the model learns much quicker and drops to minimum with only around 2000 iteration. However, it cannot be set to high as well. As the right figure shows, the training loss drops quickly at first with a leaning rate equal to 0.5, however, it then fluctuates dramatically and cannot converge to minimum. This is because when the learning rate is too large, gradient decent can overshot the minimum, failing to converge. Thus, it is important to find an appropriate learning rate.

Setting the learning rate to 0.1, all the initial theta to 0 and iterate 10000 times, a logistic regression model was derived. To evaluate the model performance, a balanced validation data with a same class ratio with train data would be applied. In subtask 5, the model performance on unbalanced data will be discussed. The negative sampling was used to produce the validation data (original validation sentences was the first verifiable claims from 'dev.jsonl'). Putting the train data and validation data into the trained model, predicting its relevance and comparing the predicting results with their labels which were contained in claims data, the accuracy, recall and precision of the model were calculated. The accuracy was 0.74 on training data and 0.75 on validation data. The recall and precision were 0.75 and 0.76 respectively on train data and 0.69 and 0.72 on validation data. Since these metrics on validation data were similar to those on train data, it seems that this model is not encountered with overfitting problem. However, as the accuracy is not above 0.9 on training data, this model could still be improved. For future work, more data can be collected, and the model complexity can be improved.

2.5 Subtask 5: Relevance Evaluation

This task will apply recall, precision and F1 score to evaluate the sentence retrieval performance on development set. The precision measures the proportion of retrieved documents that are relevant, using the formula:

$$precision = \frac{\text{number of retrieved docs that are relevant}}{\text{number of retrieved docs,and recall}} \quad (15)$$

and recall measures the proportion of relevant documents actually retrieved, using the formula:

$$recall = \frac{\text{number of retrieved docs that are relevant}}{\text{total number of relevant docs}} \quad (16)$$

In many cases, recall and precision give opposite result, specifically, keep model unchanged, the recall can be improved by retrieve more documents, but at the expense of precision. Therefore, F1 score:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (17)$$ is introduced to balance this problem.

The performance of the model on balanced data has been discussed on subtask 4. However, the balanced data can only be used to evaluate the prediction ability of model, in practical application, the data are always unbalanced, with only a small amount of relevant data. Therefore, in this task, the model was evaluated also based on unbalanced validation data. Instead of negative sampling, all the possible negative sentences were picked out, contributing to the positive to negative ratio of 1:14. In this case, the accuracy become meaningless, because it can reach 0.93 when the model predict all the relevance indicator to 0. With this unbalanced validation data, the precision is still relatively high, at 0.83, which means the retrieval results can be quite reliable based on this model, and most of the results it retrieved are relevant. However, the recall drops from 0.75 to 0.13, lowering the F1 score to 0.22. This suggested that, in practical application, this model seems to miss many relevant documents even though the retrieved documents were relevant.

2.6 Subtask 6: Truthfulness of claims

With the retrieved relevant sentences, the truthfulness of a claim can be predicted based on these sentences. In this subtask, as the 'train.jsonl' and 'dev.jsonl' are labeled, the 'evidence' field in claim data was used directly to construct training data and validation data. When predict the truthfulness of claims on test data, the relevant sentences should be selected firstly using the methods of subtask 2,3 and 4, and then take the steps described below.

Before applying neural network to assess truthfulness, the words in claims and document sentences were embedded to trainable vectors respectively (represented by the index in temporary vocabulary list), and all of these sentences were padding to the same length. Claims and evidence sentences were connected by horizontally concatenation. Later these index-represented word vector would be convert to actual vectors as model input, using the 300d Glove model. Again, this pre-trained model is additable for missing words so that important factors such as names of places and years would be considered in prediction model.

The neural network chosen for this truthfulness prediction task was bidirectional Long Short-Term Memory model (Bi-LSTM). The experiment of many previous researchers has demonstrated that the LSTM performed well on many natural language processing tasks, and this will be mentioned in subtask 7. As the sequence of the words in claims and sentences is significant, this sequential model can be very good choice for this task. Improving RNN model, LSTM solves the problem of vanishing gradient, and it can share features of two terms in a sentence, even though their positions are far apart. For a word in a sentence, its previous words and following words are same important, however, the traditional LSTM only go through the words from the first to the last. In this subtask, the traditional LSTM were improved by applying Bi-LSTM, so that the model would go through the word from both the first to the last and the last to the first.

As for implementation, the package of Keras was applied. A three-recurrent-layer LSTM network is constructed with 50 neurons in each layer. The activation function in the last layer used is sigmoid and the loss function is binary crossentropy. The 'AdaGrad' was chosen as optimizer. During the training process, to prevent overshooting, the learning rate would be reduced when a loss on validation data had stopped improving. To avoid overfitting, the regularization and dropout was applied to the model. In addtion, the early stop method were also used and the training process would be stopped with 3 epochs of no reduce in validation loss. The batch size is 1000. The performance of the trained model is evaluated by accuracy, recall and precision. As the data is quite unbalanced, the accuracy is meaningless. The precision is 0.86 on the training data and 0.69 on validation data. The recall is 0.96 on train data and 0.92 on validation data. This result shows a quite high recall and a relatively low precision, which means it is reliable when this model predicts a positive value for truthfulness, however, a small amount truthful statement can be predicted to be fake news. The record of training loss in the last three epochs shows a downward trend on training data and an upward trend on validation data, which illustrated that the model is still a bit overfitting and that might be the reason why the recall and precision were both lower on validation data compared with that on train data. For further work, more data should be collected, and the weight of regularization and dropout should be increased.

3 Literature Review (Subtask 7)

Many researchers have researched on the field of automated fact checking. Alhindi, Petridis and Muresan (2018) [1] stated that the checking process include the retrieving evidence from the trusted sources, understanding the context and concluding the information form the evidence.

In terms of the evidence retrieving, there are many methods, in addition to using the documents of wikipage, Vlachos and Riedel (2014) [2] provided an alternative method of using the construction of a dataset using statements fact checked by journalists available online. Compare with using fixed document, this method can get the latest information and is more rationally. For the further work, the method of construction of a dataset can be updated to this method.

When understanding the context and concluding the information form the evidence, basic machine learning models and neural network would be very helpful. In the experiment of Alhindi, Petridis and Muresan(2018) [1], they compared the predictive model using the Logistic Regression (LR), Support Vector Machines (SVM) and Bi- Directional Long Short-term Memory (BiLSTM). The results show that the BiLSTM provide the best performance almost among four different situations, especially on the train data. This might be because the BiLSTM have more complex network, so that it can include more information, such as word sequences.

REFERENCES

[1] T. Alhindi and S. Petridis and S. Muresan (2018) Where is your Evidence: Improving Fact-checking by Justification Modeling, https://aclweb.org/anthology/W18-5513

[2] A. Vlachos and S. Riedel (2014) Fact Checking: Task definition and dataset construction, https://www.aclweb.org/anthology/W14-2508