Report of COMP0084 Coursework2

ABSTRACT

This project develops information retrieval models to solve the problem of passage retrieval. To improve the basic models implemented in the first part of the coursework, this part of project constructs Logistic Regression Model, LambdaMART Model, and Neural Network Model to predict the relevance of queries and passages. Finally, the performance of the models are evaluated by Mean Average Precision(MAP) and Normalized Discounted Cumulative Gain(NDCG).

ACM Reference Format:

. 2022. Report of COMP0084 Coursework2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Techniques of information retrieval are applied widely on searching engines, question answering, and recommendations. The task of this part of work is to improve the basic information retrieval models implemented in the first part of the coursework. The models implemented in the project are Logistic Regression Model, LambdaMART Model, and Neural Network Model. This report describes the input processing, features or representations chosen, hyper-parameter tuning method, and neural architecture chosen while implementing the models. The methods used to evaluate the performance of the models is based on Mean Average Precision and Normalized Discounted Cumulative Gain which are developed based on the BM25 model constructed in the first part of the coursework.

2 TASK1

In this part, the metrics of Mean Average Precision(MAP) and Normalized Discounted Cumulative Gain(NDCG) are computed to evaluate the performance of BM25 model implemented in the first part of the coursework.

2.1 Introduction to matrics

AP is the average of precisions at all relevant documents.

$$AP_k = \frac{\sum_{i=1}^{k} precision_{i_{rel}}}{n_{rel_k}} \tag{1}$$

where AP_{kq} is the average precision of relevant documents at rank k retrieved by a query, $precision_{i_{rel}}$ is the precision value of the i relevant document at rank i, n_{rel_k} is the total number of relevant documents at rank k.

Unpublished working draft. Not for distribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00

https://doi.org/10.1145/nnnnnnn.nnnnnnn

Precision at rank i can be computed by the following equation:

$$precision_i = \frac{n_{rel_i}}{n_{retr_i}} \tag{2}$$

where n_{rel_i} is total number of retrieved and relevant documents at rank k, n_{retr_i} is total number of retrieved documents at rank k.

MAP is the average AP values over all queries. The value of MAP is larger, the model has better performance.

$$MAP = \frac{\sum_{i=1}^{q} AP_{kq}}{n_q} \tag{3}$$

where n_q is the total number of queries.

DCG is the total gain accumulated gain at a particular rank k,

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i - 1}}{\log(1+i)} \tag{4}$$

NDCG is the normalised DCG against the best possible DCG result (the perfect ranking) for the query. The value of NDCG is between 0 and 1. The NDCG value is more closed to 1, the model is better.

$$NDCG = \frac{DCG}{optDCG} \tag{5}$$

2.2 Implementation

The computation of the two metrics on the BM25 model is conducted in 3 .py files.

The file task1_preprocess.py carries out data preparations for BM25 computation, which involves text preprocessing, inverted index generation, term frequency computations of the validation data set.

The file task1_bm25_retrieval.py computes the BM25 score for each query and passage pair, and output qid_pid_score csv file for evaluation.

The task1_evaluation.py file computes the MAP and NDCG values for BM25 ranked results on the validation data set. The method of MAP computation and NDCG computation are encapsulated as functions to ease the evaluation of other models.

2.3 Evaluation

Implement the above .py files in order, we can get the result of MAP and NDCG of the BM25 retrieval model, which is around 0.237 and 0.355 respectively.

3 TASK2

In this part, Word2Vec is used to generate embeddings of words, which then are utilised to compute query/passage embeddings by averaging embeddings of all the words in that query/passage. Then Logistic Regression(LR) is trained using feature vectors derived from query and passage embeddings. Finally, the performance of LR model is evaluated by the MAP and NDCG methods on task1.

Unpublished working draft. Not for distribution.

Table 1: Training Loss

Learning Rate	Training Loss
0.01	NA
0.005	NA
0.003	0.124296
0.001	0.146415
0.0008	0.050395
0.0005	0.030997
0.0001	0.030278
0.00001	0.030278

3.1 Dataset processing

There are three phases which uses the train data set and validation data set as input in task2.

In the phase of generating word embeddings, I use all queries and passages in the train data set in order to get word embeddings in more comprehensive contexts.

In the phase of training the LR model, I extract a sample of train data set to lessen computing time. The sample consists of all 4797 relevant query-passage pairs with the value of "relevancy" 1 and 1 million irrelevant query-passage pairs with the value of "relevancy" 0. Thus, the quantity of the sample is near to the whole validation data set which will be used to evaluated the performance of the model in the evaluation phase.

3.2 Model Implementation

The file task2_embeddings.py is implemented to generate embeddings of queries and passages by averaging embeddings of all words in each query and passage based on Word2Vec. An embedding array's size is 1*100. At the end of the implementation, the query embeddings, passage embeddings, and the sample of train data will be stored locally for the implementation of LR model.

After getting query embeddings and passage embeddings, the file task2_LogisticRegression.py. trains and evaluates the LR model. The vector I choose to represent the relevance of a pair of query and passage is an array which appends the embedding of a query and that of the corresponding passage. Thus the size of the relevance vector is 1*200. The vectors of all pairs in the train data sample compose a matrix with size number of pairs * 200, which is used as the LR model input x while training. The LR model input y is an array with size of number of pairs * 1 which stores the label of relevancy for all query-passage pairs. The evaluation is based on the whole validation data set and the observation of the impact of iteration numbers on the training loss are also conducted in the file.

3.3 Evaluation

By changing the learning rate and implement the LR model, I get the training loss of each learning rate shown on Table1.

From the result, we can find that the learning rate has an direct impact on the training loss on the condition that the iteration number is 2000 and the learning rate is between 0.003 and 0.00001. As the learning rate declines, the the training loss shrinks. But when the learning rate gets to an extremely low value, its change has less impact on the training loss. On the other hand, if the learning rate is too large, the loss function cannot converge, which results to a NA value of loss.

The mAP and NDCG value of the LR model(iteration number=2000, learning rate=0.0005) is around 0.0133 and 0.0374 respectively, which is not improved compared to the result of BM25 model. The elements impact the performance of the model involve the quality and scale of train data and validation data, the features chosen, the iteration number and learning rate of the LR model. Therefore, the model could be optimised at a large scale.

4 TASK3

In this task, the LambdaMART(LM) learning-to-rank algorithm from XGBoost gradient boosting library is used to learn a model that can re-rank passages.

4.1 Dataset processing

In this section, I use the same features as the task2 to train model, the appended embeddings of query and passage pairs. The corresponding data (x_train and y_train used to train model; x_valid and y_valid used to evaluate model) are stored locally on task2.

The whole validation data set is used to evaluate the performance of the model.

4.2 Model Implementation

The task3_LambdaMART.py file conducts the LambdaMART model hyper-parameter tuning to find a best model and use the model to rank passages, the result of which is evaluated by MAP and NDCG methods.

The hyper-parameter tuning method I use is based on the CV attribute from the XGBoost library. At first, I define a dictionary to store parameters I want to tune and another dictionary to store the optimal parameters, and an initial MAP value to benchmark the performance of the model. Then, in a for-loop, the model will be trained based on each set of the parameters. At the end of each training, the MAP of current model will be compared to the stored MAP value, so that the model with the maximum MAP value will be stored as the best model.

Then the best model is utilised to predict labels on validation data set, the result of which is evaluated by MAP and NDCG methods on task1.

4.3 Evaluation

The MAP value of the LM Model is 0.0071, the NDCG value is 0.0255. The performance of the model is influenced by values of parameters as well as quality and scale of the training data, so it may have a better performance after more tuning of parameters or use a more appropriate sample of training data to train.

5 TASK4

This part of work utilises Tensorflow to build a neural network based model with the ability of re-ranking passages. The performance of the model is evaluated by MAP and NDCG on the validation data set.

Unpublished working draft. Not for distribution.

5.1 Dataset processing

In this section, I use the same features as the task2 and task3 to train model, the appended embedddings of each query and passage pair. The corresponding data (x_train and y_train used to train model; x_valid and y_valid used to evaluated model) are stored to the local on task2.

The whole validation data set is used to evaluate the performance of the model.

5.2 Model Implementation

The task4_NeuralNetwork.py constructs the LSTM(Long-Short Term Memory RNN) model using Tensorflow. The reason I choose the LSTM architecture is that it is advantageous in natural language processing. Compared with CNN, RNN actually improves the context dependency and solves the similar context dependency to a certain extent. However, LSTM solves the dependency between words even at a long distance, making the recurrent neural network have a deeper memory function. Therefore, LSTM is assumed to perform well on the provided data set which contains a large amount of long texts data.

5.3 Evaluation

The MAP value of the LSTM Model is 0.0133, the NDCG value is 0.0366. I assume that we can improve the performance of the model by setting more appropriate parameters or providing features containing more information of the queries and passages, such as BM25, cosine similarity, etc.

6 CONCLUSION

The project constructs Logistic Regression Model, LambdaMART Model, and Neural Network Model to re-rank passages for each query. The methods of MAP and NDCG derived based on the previous BM25 model are used to evaluate the new implemented models.

The models constructed in the part of the coursework generally have a performance that still needs improving since they depends largely on the quality of the sample data set, the parameters set to train the model, and features chosen.

Unpublished working draft.
Not for distribution.