The Coursework 2 of Information Retrieval and Data Mining

ABSTRACT

Information retrieval models are an important part of many applications such as search, question answering, recommendation, etc. In this assignment, a number of information retrieval models are built that address the problem of paragraph retrieval, all of which are capable of efficiently returning a sorted list of short texts (i.e. paragraphs) relevant to a given query, and furthermore, are used to evaluate their retrieval Ranking performance metrics are designed to evaluate their retrieval performance.

KEYWORDS

information Retrieval Model,logistic regression,LambdaMART,neural Networks

ACM Reference Format:

. 2022. The Coursework 2 of Information Retrieval and Data Mining. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this assignment, a number of information retrieval models for solving the paragraph retrieval problem are established, and the retrieval effect is evaluated. In the second part, we establish metrics for evaluating information retrieval models: Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG), and evaluate the performance of the BM25 retrieval model built in previous work. In the third part, we build word embedding vectors of training data and validation data based on the Word2Vec model, and train a logistic regression model to predict the relevance before query and paragraph. In Section IV, we use the LambdaMART model to build a similar query ranking model and evaluate it. In Section V, we build three different types of neural networks and conduct a comparative analysis of their query performance.

2 THE ESTABLISHMENT OF METRICS

In task1.py, we established metrics for evaluating the information retrieval model: mean precision (MAP) and Normalized Discounted Cumulative Gain (NDCG), and evaluated the effect of the BM25 retrieval model established in the previous job.

2.1 Mean Average Precision

In order to calculate the evaluation index MAP, we designed the CalculateAveragePrecision function. Specifically, this function implements the calculation of the average precision of each query in query data. For the first 100 retrieved records corresponding to each query, we have calculated and accumulated average precision. Finally, we add it to query data as a new column.

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

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Parameters of the BM25 model

Type	Value
k1	1.2
k2	100
b	0.75
N	955211
avdl	56.82515067351611
train_data amount	None
validation_data amount	113040

Table 2: Retrieval performance of the BM25 model

Metrics	Score
MAP	0.014220879722022317
NDCG	0.03290452438605122

2.2 Normalized Discounted Cumulative Gain

Similarly, in order to calculate the evaluation index NDCG, we designed the CalculateNDCG function. Specifically, this function implements the calculation of the DCG value of each query in query data. For the first 100 retrieved records corresponding to each query, we have calculated and accumulated the DCG. Finally, the normalized NDCG value is added to query data as a new column.

2.3 Retrieval evaluation results of the BM25 model

In this assignment, we build the BM25 model for query ranking. Specifically, we designed TextPreprocessing and GenerateInvertedIndex functions for text preprocessing, and designed the CalculateBM25Similarity and BM25Model functions to build a bm25 model based on the query results in validation_data.tsv.The parameters of the model are shown in Table 1.It is worth mentioning that this model is not trained with training data.

Afterwards, for the query ranking result bm25_Df based on the BM25 model, we calculated the MAP and NDCG values for it using the CalculateAveragePrecision function and the CalculateNDCG function. The results are shown in the table 2.

The results shown in Table 2 show that the BM25 model has relatively good retrieval performance, and the designed metrics can effectively measure the retrieval results of the model.

3 THE ESTABLISHMENT OF LOGISTIC REGRESSION MODEL

In task2.py, we build word embedding vectors for training data and validation data based on the Word2Vec model, and train a

Table 3: Parameters of the Word2Vec model

Type	Value of queries	Value of passage
vector size	300	300
workers	4	4
min count	2	4
window	1	10
sample	0.001	0.001

logistic regression model to predict the relevance between query and paragraph. Finally, the sorting results are stored in LR.txt.

3.1 The establishment of word embedding model

For subsequent training of various information retrieval models, we need to perform word embedding processing on queries and paragraphs in the form of characters to generate word vectors for training and validation. Therefore, we choose Word2Vec model for word embedding.

First, we read the data in train_data.tsv and validation_data.tsv. In order to save the embedding processing time, it is stipulated in train_data and validation_data that the first 2 million rows of data and the first 500,000 rows of data are selected as samples of the dataset respectively, which is about 50% of the total data volume of the dataset.

After that, we use the TrainingVectorModels function, which is used to train the Word2vec model on the training data. Specifically, we use the GenerateSentences function to generate the corresponding sentences library for the queries and paragraphs in the training data, and then train two Word2Vec models: model_queries and model_passage. The parameters of these two models are shown in Table 3.

Subsequently, in order to unify the size of the word vector to facilitate subsequent training, the CalculateAverageVector function was designed. This function is based on the models model_queries and model_passage, and uses the GetAvgFeatureVecs function to calculate the average vectors corresponding to train_data and validation_data, respectively. Finally, X_train, y_train, X_validation, y_validation are returned for subsequent training.

It is worth mentioning that since we selected 300 features in the previous Word2Vec model training, the feature dimension of the generated query and passage word vectors is 300. After that, X_train and X_validation are horizontally spliced from a large number of queries and paragraphs, so their feature dimension is 600.

3.2 The training of logistic regression model

Based on _train and _train in word vector format, we manually designed and implemented a logistic regression model (LogisticModel function) for training it.

Specifically, the input parameters of the model are X_train, Y_train, learning_rate, num_iterations, and the return values are W, b, costs. It means that the function will perform num_iterations iterations, with learning_rate as the learning rate, perform forward and reverse gradient descent calculations in the Propagate function, thereby updating the solution weight W and offset b, and at the same time, record and return each time Loss costs when iterating. The parameters of this model are shown in Table 4.

Table 4: Parameters of the logistic regression model

Type	Value
learning rate	0.1
num iterations	100
X_train size	(2000000,600)
y_train size	(2000000,1)

The Predict function predicts the possible y values corresponding to the input X_validation based on the previously obtained weight W and offset b of the logistic regression model. We finally got y_validation_prediction, which is also the basis for subsequent retrieval and sorting, namely score.

Table 5: Retrieval performance of the logistic regression model

Metrics	Score
MAP	0.007348505368642836
NDCG	0.06719549196758456

3.3 The retrieval evaluation results for logistic regression models

To build a logistic regression-based retrieval model, we designed the GenerateSortedDataFrame function. Specifically, the prediction y_validation_prediction is used as the basis for sorting, and retrieved and sorted the passage corresponding to each query in the first 2,000,000 pieces of data in validation_data.tsv.

In addition, we recorded the first 100 passages of each query (some queries were less than 100) and stored them in LR.txt.

After that, we calculated the MAP value and NDCG value of the sorting result LogisticRegression_Df using the CalculateAveragePrecision function and CalculateNDCG function in the second part. The ranking retrieval performance evaluation results of the logistic regression model are shown in Table 5.

Obviously, after 100 iterations, this logistic regression model performs well in predicting the correlation between passage and query. Its NDCG value reaches about 0.06719, which means that most of the related items predicted by it are ranked first, but its MAP value is only about 0.007348, which means that its predicted value is only 1.0 or 0.0. Records with likelihood of relevance are simply ranked first, ignoring high and low relevance.

3.4 The effect of learning rate on training loss

In order to analyze the influence of the learning rate on the training loss, we selected the learning rate of 1, 0.5, 0.1, and 0.01 to train the logistic regression model for 100 iterations, and plotted the training loss costs with the iterations. The change trend comparison chart is shown in Figure 1.

As can be seen from Figure 1, the learning rate significantly affects the speed at which the loss function converges. When the number of iterations is 100 and the learning rate is 0.01, the training

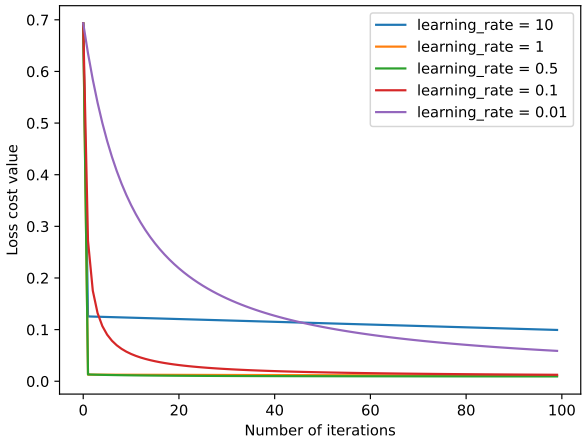

Figure 1: Convergence comparison of losses under different learning rates

loss converges slowly and does not really converge until the end of training, while the logistic regression model with a learning rate of 0.1 converges after about the 35th iteration. About 0.012040. As the learning rate increases, so does the drop in training loss. When the learning rate is 1 and 0.5, after the 10th iteration, their loss value has dropped below about 0.01203, which means a very fast convergence rate. However, when the learning rate is 10, the loss cost of the model quickly converges to 0.13273 and then never converges to the minimum value (about 0.009), which means that the excessive learning rate makes the learning curve jump repeatedly and is difficult to converge. to the global minimum.

In this assignment, since the cost function of logistic regression is convex, it is guaranteed to find the global minimum by gradient descent. In the actual training environment, the loss functions of other models may be non-convex functions. At this time, if the learning rate is too large, it may not converge, but diverge. On the other hand, a learning rate that is too small means that the loss drops very slowly during training, and a non-convex function means that a learning rate that is too small may cause the loss to converge to a local minimum rather than a global minimum. In short, a suitable learning rate is very necessary, which is why the learning rate of 0.1 is finally selected.

4 THE ESTABLISHMENT OF LAMBDAMART MODEL

In task3.py, we read the word embedding vectors of training data and validation data based on the Word2Vec model in task2.py. Afterwards, based on the XGBoost library, a LambdaMART model is trained to predict the relevance between queries and paragraphs. Finally, the sorting results are stored in LM.txt.

Table 6: Parameters of the LambdaMART model

Type	Value
$\max_depth$	6
eta	3
objective	rank:pairwise
X_train size	(2000000,600)
y_train size	(2000000,1)

Table 7: Retrieval performance of the LambdaMART model

Metrics	Score
MAP	0.013950970993741459
NDCG	0.07633866034463212

4.1 The establishment of word embedding model

For subsequent training of various information retrieval models, we need to perform word embedding processing on queries and paragraphs in the form of characters to generate word vectors for training and validation. Therefore, we read the word vector data based on the Word2Vec model in the third part.

First, the training data in X_train_task2.txt and y_train_task2.txt and the validation data in X_validation_task2.txt and y_validation_task2.txt are loaded. After that, since the XGBoost library has special requirements for the format of the input files, we established task3_train_data.txt and task3_validation_data.txt files to store training data and validation data suitable for XGBoost format. We then traversed the four data from task2.py, including X_train, and refactored its format to fit the XGBoost library. Finally, the files are stored in task3_train_data.txt and task3_validation_data.txt.

4.2 The training of LambdaMART model

Based on task3_train_data.txt and task3_validation_data.txt, a LambdaMART Model is built for training it. After several adjustments, the parameters of the ranking model with the best performance are shown in Table 6.

After that, we make predictions on the possible y values corresponding to the input X_validation. We finally got y_validation_prediction, which is also the basis for subsequent retrieval and sorting, namely score.

4.3 Retrieval Evaluation Results of the LambdaMART Model

In order to establish a retrieval model based on logistic regression, we used the GenerateSortedDataFrame function in the third part to retrieve and sort the passage corresponding to each query in the first 2,000,000 pieces of data in validation_data.tsv.

In addition, we recorded the first 100 passages of each query (some queries were less than 100) and stored them in LM.txt.

After that, we calculated the MAP value and NDCG value of the sorting result LogisticRegression_Df using the CalculateAveragePrecision function and CalculateNDCG function in the second part.

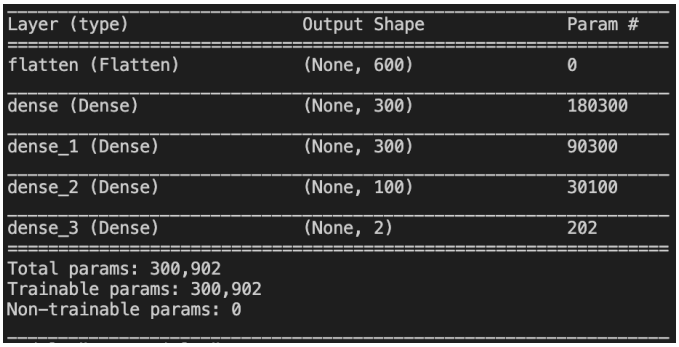

Figure 2: The struct of ANN

Table 8: Parameters of the ANN model

Type	Value
loss	sparse$_c ategorical_crossentropy$
optimizer	nadam
metrics	accuracy
epochs	10
X_train size	(2000000,600,1)
y_train size	(2000000,1)

The ranking retrieval performance evaluation results of the logistic regression model are shown in Table 7.

As can be seen from Table 7, the LambdaMART model performs relatively well in predicting the correlation between paragraphs and queries.

5 THE ESTABLISHMENT OF NEURAL NETWORK MODEL

In task4.py, we implemented three different types of neural networks, and repeatedly trained and tuned them using the word vector data in the third part, and used the MAP and NDCG in the second part as metrics to measure Its retrieval performance as a ranking model. Ultimately, recurrent neural networks achieve the best ranking retrieval results in this task.

5.1 The establishment of artificial neural network

The establishment of artificial neural network

First, we built a simple artificial neural network (ANN) based on the keras library of tensorflow, the structure of which is shown in Figure 2.

As shown in Figure 2, the artificial neural network has an input layer, three hidden layers with 300, 300, and 100 nodes respectively, and an output layer with softmax, which is used to predict the probability that a single record is 0 or 1. Next, we loaded the Word2Vec-based word vectors generated in the third section as input to train the artificial neural network. Its training parameters are shown in Table 8.

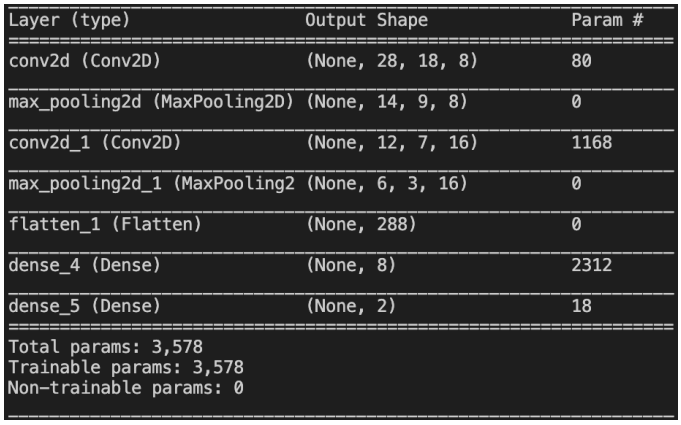

Figure 3: The struct of CNN

Table 9: Parameters of the CNN model

Type	Value
loss	sparse$_c ategorical_crossentropy$
optimizer	nadam
metrics	accuracy
epochs	10
X_train size	(2000000,30,20,1)
y_train size	(2000000,1)

5.2 The establishment of convolutional neural network

After that, we built a complex convolutional neural network (CNN) based on the keras library of tensorflow, whose structure is shown in Figure 3.

As shown in Figure 3, the convolutional neural network has a Conv2D layer and a MaxPooling2D layer for summarizing features and reducing the matrix size, followed by a Conv2D layer and a MaxPooling2D layer to further abstract the features, followed by three layers The fully connected network consists of a Flatten layer for stretching the matrix, a hidden layer with 8 nodes, and an output layer with softmax to predict the probability of a single record being 0 or 1.

Next, we loaded the Word2Vec-based word vectors generated in the third section as input to train the convolutional neural network. Its training parameters are shown in Table 9.

5.3 The establishment of recurrent neural network

Finally, since word vectors have a certain degree of correlation in the context of the front and back, this means that the specific memory unit in the recurrent neural network will have a unique advantage in its prediction. Therefore, based on the keras library of tensorflow, a sequential classifier with a recurrent neural network (RNN) architecture is built, and its structure is shown in Figure 4.

As shown in Figure 4, the recurrent neural network has a SimpleRNN input layer with 32 nodes, a SimpleRNN layer with 32

```
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 30, 32)            1696
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 3,809
Trainable params: 3,809
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: The struct of RNN

Table 10: Parameters of the RNN model

Type	Value
loss	$\text{binary}_c rossentropy$
optimizer	rmsprop
metrics	accuracy
epochs	10
X_train size	(2000000,30,20)
y_train size	(2000000,1)

Table 11: Retrieval performance of the neural network model

Type	MAP	NDCG
ANN	0.007348505368642836	0.06719549196758456
CNN	0.00818577462324807	0.06873133113456102
RNN	0.010222043285007037	0.07161612203338809

nodes, and a sigmoid output layer with one layer to predict the probability that a single record is 1. Next, we loaded the Word2Vec-based word vectors generated in the third section as input to train the RNN. Similar to a normal NN, except that backpropagation is done through time, i.e. the layer order gradients to the assigned weights to match the output of the layers. Its training parameters are shown in Table 10.

5.4 Retrieval Performance of Neural Networks

Similar to in Sections 3 and 4, we use the outputs of these three different neural networks as the ranking basis for our retrieval model. After sorting the validation data, the MAP and NDCG values for each neural network are calculated, as shown in Table 11.

As shown in Table 11, the RNN-based sequential classifier has the best retrieval performance. In terms of MAP value, its value is 0.010222043285007037, which exceeds the logistic regression model, ANN and CNN, which represents the excellent performance of the unique memory function of the recurrent neural network in sorting multiple sets of word vectors. Its NDCG value reaches 0.07161612203338809, which is slightly better than ANN's 0.06719549196758456 and CNN's 0.06873133113456102. The above data proves that the RNN-based sequential classifier has excellent retrieval and ranking performance. Therefore, the ranking results of the RNN retrieval model are recorded and stored in NN.txt.

5.5 The analysis of neural network structure and retrieval effect

The reasons why RNN has the best performance in this text classification problem are mainly divided into the following points

First of all, the data set used in this training is 2 million, which is relatively large, which also prevents the RNN from overfitting due to the small amount of data, and the effect is relatively better.

Secondly, in the data set preprocessing in the third part, the minimum frequency selected for the passage word embedding is only 4, which means that the existence of a large number of low-frequency words brings interference and difficulties to the training of the three neural networks.

After that, CNN and RNN have different applicability. CNN is good at learning and capturing spatial features, and is more widely used in the field of image processing. However, RNN is good at capturing time-series features. Since there is often a connection between text contexts, this means that RNN has a superior structure in the fields of text analysis and information retrieval.

6 REFERENCES

[1]Vallet D, Fernández M, Castells P. An ontology-based information retrieval model[C]//European Semantic Web Conference. Springer, Berlin, Heidelberg, 2005: 455-470.

[2]Bordogna G, Pasi G. An ordinal information retrieval model[J]. International Journal of Uncertainty, Fuzziness and Knowledge-Bas

[3]Jun-feng S, Wei-ming Z, Wei-dong X, et al. Ontology-based information retrieval model for the semantic web[C]//2005 IEEE International Conference on e-Technology, e-Commerce and e-Service. IEEE, 2005: 152-155.

[4]Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv:1409.2329, 2014.

[5]Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[C]//Interspeech. 2010, 2(3): 1045-1048.

[6]Lukoševičius M, Jaeger H. Reservoir computing approaches to recurrent neural network training[J]. Computer Science Review, 2009, 3(3): 127-149.

[7]Burges C, Svore K, Bennett P, et al. Learning to rank using an ensemble of lambda-gradient models[C]//Proceedings of the learning to rank Challenge. PMLR, 2011: 25-35.

[8]Gu J, Wang Z, Kuen J, et al. Recent advances in convolutional neural networks[J]. Pattern Recognition, 2018, 77: 354-377.

[9]O'Shea K, Nash R. An introduction to convolutional neural networks[J]. arXiv preprint arXiv:1511.08458, 2015.