IR CW2

19095174

ACM Reference Format:
19095174. 2023. IR CW2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 TASK 1

Copied and pasted BM25 function and hyperparameters over from CW1, and ran the BM25 over on the validation data to rerank the passages per query based on BM25 scores. The purpose was to get the ranking rather than the scores themselves, since MAP and NDCG is based on ranking of documents. The BM25 function returns the top 500 documents based on score for each query. Hence I have calculated the MAP@k and NDCG@k scores for k = 10, 100, 500 in Table 1 below

2 TASK 2

For the logistic regression task, I had opted for gensim's pretrained GloVe package as it embedded words to 100 dimensions, which I felt was suitable and not too large for the task, having tried different packages that gave too large of word vectors, such as Spacy's 300 dimension word vectors from its pretrained model, which would take more computational power to handle (e.g. to find cosine similarity, etc).

Due to the very large sample size of the training data (> 4 million rows), I used a random sampler that randomly selected a number of queries

After taking the word embedded for each word within the queries and passages, I then took the average of all word vectors elementwise to get an average word embedding representation for each query and vector. This would serve to capture the 'overall meaning' of the sentence, since each dimension captures some sort of semantic meaning.

After which, I took the cosine similarity for each query-passage pair via taking the normalised dot product of the average embeddings. This was the X input that was used to fit the logistic regression.

Due to the large size of the validation dataset, the time taken to embed the queries and passages were 257 seconds or more than 4 minutes long.

The @k MAP and NDCG are the following in Table 4

As for the effect of learning rate on model training loss, I tested out learning rates of : 1e-6, 1e-5, ... 1e-2 and recorded the model loss, then plotting out the relationship between the two variables as in Figure 1, where we can see that the optimal learning rate is 1e-4.

Table 1: BM25

k	MAP	NDCG
10	0.5	0.260
100	0.0769	0.318
500	0.182	0.341

Table 2: Logistic Regression

k	MAP	NDCG
10	0.25	0.00364
100	0.0139	0.0209

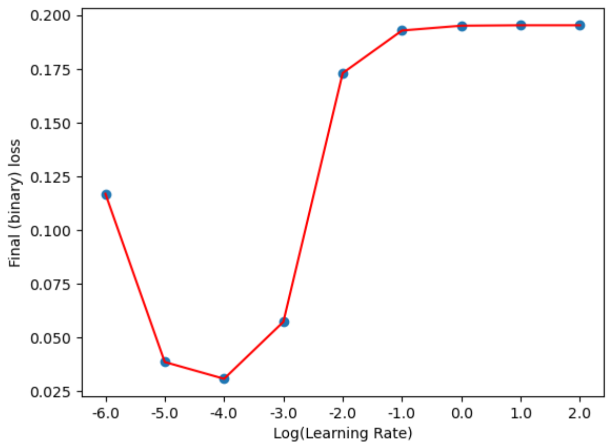

Figure 1: Training loss against Learning rate

3 TASK 3

For task 3, I had used the averaged word embedding vector representation from Task 2 for the X input to the XGB model. *objective = 'ndcg'* was used for a listwise Learning-to-Rank model that would maximise NDCG from ranking the passages per query, as opposed to pairwise methods.

The X data input is the summation of the averaged word embedding of query and document, as I wanted to 'combine' the information of the two as a single input data.

In terms of model hyperparameters, I have chosen the objective to be 'rank:ndcg' so as to achieve listwise learning-to-rank such that the ndcg of the entire rank is optimised. I set the intial max_depth (tree complexity) to be 6, and the n_estimators (number of trees) to be 100, and the eta as 0.05. As the model will be trained on the rankings of all passages per query, it was important to specify the grouping argument 'groups' in the input xgb.DMatrix data object before fitting into the model.train() method.

Using sklearn's GridSearchCV module which uses gridsearch to iteratively search for optimal hyperparameters , according to

Table 3: LambdaMart Learning-to-rank

k	MAP	NDCG
10	0.125	0.00593
100	0.083	0.0247

which gives the best precision (used ‘scoring = ’precision’‘ to select the best hyperparameters), it was revealed that the following were optimal:

- Learning rate : 0.005
- Max_depth : 5
- n_estimators : 75
- Sub_sample : 0.5

4 TASK 4

For task 4, I have used the averaged embedded vector representation of both queries and passages separately as inputs into the neural network. I have also added in Lidstone smoothing QLM scores, to provide a different perspective on the data and queries via *exact matching* rather than *semantic representation* that the former provides. As mentioned in COMP0084 Lecture week 6 and similar to the DeepRank paper by Guo et al (2019) [1] that combined the use of BM25 scores with vector embeddings, along other features, as inputs to the model, this combines the simplicity of exact matching algorithms that can complement vector represenations in special cases such as rare word matching. I had used the Lidstone smoothing QLM code from CW1 for the purposes of Task 4.

The model used is a basic feedforward neural network with a single convolutional layer at the beginning.

The architecture of the neural network consists of :

(1) Initial convolution layer for both query and passage vector, with a single in channel, 10 filters/kernels, and a post max-pooling step of size 2
    - Single fully connected layer for query and passage, after max pooling
(2) Single fully connected layer for Lidstone scores
(3) Finally, joining all 3 hidden layer outputs thus far from queries and passages and scores, we concatenate them for 2 more hidden layers and a final sigmoid activation function, since the output is binary

The rationale behind using a convolution layer is to find patterns in the text for both the query and passage, which could potentially pick out topic word/phrases, and reduce the dimension for the subsequent fully connected layers.

I am hoping that the model separately analyses the query and passage, then joins them together finally to be able to spot relationships between query and passage that determine relevancy (or not).

I had attempted to include some exact matching scores, to be able to combine both vectorial embedding information and exact matching information to get a better model and better predictions, similar to Guo et al (2019). However, I was unable to fix the model architecture, and hence settled for without the exact matching scores as inputs for the neural network.

Table 4: Neural Network

k	MAP	NDCG
10	0.25	0.00364
100	0.0139	0.0209

As we can see, compared to the LambdaMart, the Neural Network which takes in more information through Lidstone scores performs better on MAP@10, but worse on all other metrics i.e. MAP@100, NDCG@10, NDCG@100.

It can also be seen that the metric scores from the Neural Network are the same as those of the Logistic Regression, which shows that the two models are working very similarly.

REFERENCES

[1] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. A deep look into neural ranking models for information retrieval. *CoRR*, abs/1903.06902, 2019.