Information Retrieval and Data Mining: Coursework 2

21069508*
ucabc19@ucl.ac.uk
University College London
London, UK

ABSTRACT

This report aims to deliver the details of processing coursework 2 of COMP0084. A detailed description of text ranking and retrieval task using different methods (BM25, logistic regression, LambdaMart, Transformer based Neural Network) with different metrics (Average Precision, NDCG) is presented, and the experiment results are displayed in this report.

KEYWORDS

BM25, Transformer, NDCG, Average Precision, Logistic Regression

ACM Reference Format:
21069508. 2022. Information Retrieval and Data Mining: Coursework 2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In information Retrieval and Data Mining tasks, ranking task is one of the most important task that are required by information retrieval server. A number of methods can be applied to realize the goal of retrieving passages by query. This report describes the method of retrieving and ranking passages using methods like BM25, Logistic Regression, transformer-based Neural Network. Two metrics, Average Precision (AP), and Normalized Discounted Cumulative Gain(NDCG) are used to measure the performance of ranking model. Experiment settings are displayed and performances are described at the end of this report.

This report is divided into four parts for each task, respectively. First part will describe how to use Average Precision (AP) and Normalized Discounted Cumulative Gain(NDCG) to evaluate the performance of BM25. Second part will describe Logistic Regression. Third part is used for detailing the parameter fine-tuning process of LambdaMart. The last part displays the detail of using transformer-based neural network for ranking task.

2 BM25

BM25 [5] is the probabilistic model that used for information retrieval. We will not focusing on the detail of implementation of BM25 but the metrics used for evaluating BM25.

2.1 Average Precision

Average Precision (AP) is one of the metric used for evaluating the result of ranking model. My implementation is just using the ranked true label to calculate AP. For example, if I am about to predict 5 data points with their true labels $L = [1, 1, 0, 0, 0]$, I will use BM25 model to obtain their scores first. For example, their scores are $S = [0.1, 0.2, 0.3, 0.4, 0.5]$. Then I will use S to rank L and derive a new label list $L^{'}$, which is $[0, 0, 0, 1, 1]$. In this way, new label list $L^{'}$ contains both the information of predicted order and the truth labels. Thus, new label list $L^{'}$ is used to derive AP. In this specific example, the result is derived as:

$$AP = \frac{\frac{0}{1} + \frac{0}{2} + \frac{1}{3} + \frac{2}{4} + \frac{3}{5}}{5}$$

2.2 Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain(NDCG) is another metric used for evaluating the performance of retrieval model. Before introducing NDCG, Discounted Cumulative Gain(DCG) is used as a metric for evaluation. The calculation of DCG is presented as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log(1+i)}$$

where rel_i is the relevance score of the retrieved passage, which is $L^{'}_i$ in our case. P is the particular total rank that we want to evaluate. In all experiments (BM25, Logistic Regression, LambdaMart, Neural Network), I retrieved whole amount of passages given for one query, i.e. 1000 passages for 1 query in most cases. However, DCG can be unbounded in some cases, i.e. for different query, DCG can be differed in a wild range. Thus, a normalized metric is desired for evaluation. In this case, NDCG is introduced by dividing DCG with the optimal DCG, which is shown as:

$$NDCG = \frac{DCG}{DCG_{optim}}$$

Optimal DCG is obtained by sorting all documents based on decreasing relevance scores and computing the DCG value on the ranking obtained. Same as AP, NDCG also only requires the information of predicted order as well as the truth labels. Thus, we can just input our $L^{'}$ and derive the results. Note that if none of the relevant passages is retrieved, then NDCG is set to 0 in this case.

2.3 Performance of BM25

In the experiment of BM25, I simply took the parameter setting from the last coursework, i.e. not training on train data. I used BM25 to evaluate on validation data, and I obtained the performance of my BM25 model via AP and NDCG. The AP of my BM25 model is 0.2108, and the NDCG of my BM25 model is 0.3212.

3 LOGISTIC REGRESSION

3.1 Input representation of LR

In my experiment, I use tokenizer and Mini-BertModel from HuggingFace[1].

Tokenizer. The tokenizer from huggingface is used for tokenizing raw texts into several pieces of tokens. This tokenizer use WordPiece[7] method to tokenize words and mapping each token into numbers. Tokenizer often uses space in raw text to tokenize words. Note that sometimes one word will also be tokenized to several tokens. For instance, "smaller" will be tokenized to "small" and "##er". "##" means that tokens with "##" are not regarded as the start of complete words. The advantage of using WordPiece is that, most interactive information can be preserved in the way. For example, "smaller" and "small" has similar meaning in English, but if we mapping them to different numbers, i.e. "small" to 1 and "smaller" to 2, then these two words are orthogonal to each other, thus we are wasting space while losing interactive information. After tokenizing raw text into tokens, tokenizer uses a dictionary (internal process) to map each token to a number $n \in \mathbb{N}$. Note that I do not add special tokens to the raw text, since logistic regression does not requires special tokens to initialize the model. I pad each query to length equal to 50 and each passage to 300. Tokenizer returns the encoded texts and corresponding attention mask. Each element in attention mask represents whether the element i is padded or not. For example, if our maximum length is 5, then "I like apple" is encoded as $[1, 2, 3, 0, 0]$ and corresponding attention mask is represented as $[1, 1, 1, 0, 0]$.

Embedding Representation. After obtaining encoded representation of raw texts, I fed each encoded tokens to BertModel to obtain embedding representation. BertModel is a pre-trained model, which uses a large amount of training data for Masked Language Model (MLM) task and Next Sentence Prediction (NSP) task. The benefit of using Pre-trained language model is that, we are leveraging the knowledge of model obtained from these large amount of training data. Plus, BertModel is able to obtain the interactive information between texts. For each token, BertModel will output a embedded vector with embedding size equal to 128. Each text will used averaged embedding representation. Note that, padded tokens will not attend the process of averaging by using attention mask obtained from the previous processing of tokenizing texts. Finally, embedded representations of query and passage are concatenate together to form the final training data. In our case, final train data has dimension of $128 \times 2 = 256$. Note that for logistic regression, I append vector $\mathbb{1}^m$ (m is the size of training data) to the end to X to form a biased logistic regression model.

3.2 Basic Rule of Logistic Regression

Logistic Regression (LR) [3] is a machine-learning based model, which assigns weights to each input feature and use a sigmoid function to constrain the output within range $[0, 1]$. The whole process can be represented as:

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

[1]https://huggingface.co/docs/transformers/index

Then the loss function I used in my experiment is cross entropy loss, which is shown as:

$$L(w) = -\frac{1}{m}\sum_{(} y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

where $\hat{y} = \frac{1}{1+e^{-(w^T x + b)}}$, m is the number of training samples. Then, according to the loss function, we can take the gradient of $L(w)$ and derive the update function as:

$$\frac{\partial L(w)}{\partial w} = \frac{1}{m}\sum_{i}^{m}(y_i - \hat{y}_i)x_i$$

where y_i is the i-th target, x_i is the i-th instance in training data, $\hat{y}_i$ is the i-th prediction made by LR. Thus, w can be updated as $w = w - lr\frac{\partial L(w)}{\partial w}$.

3.3 Comparison between different learning rate

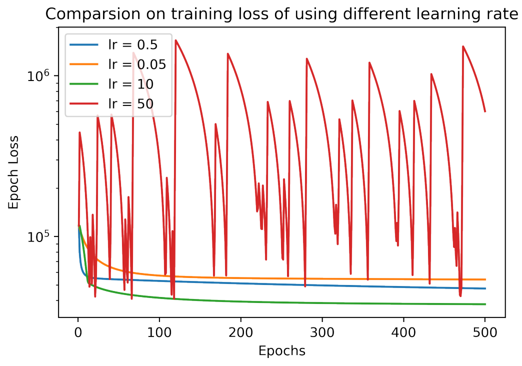

Figure 1: Different learning rates lead to different convergence level as well as different convergence time

From figure 1 we can see that it is important to choose an appropriate learning rate. Large learning rate will leads to the saddle point when implementing optimization, thus left with a local-minima. Even in some case, large learning late leads to the extreme fluctuation, just as figure 1 shows. While small learning rate will eventually converge in most case, however consuming more steps, i.e. large amount of time to converge.

3.4 Performance of Logistic Regression

In my experiment of LR, I use whole training data to train my LR model, and I test all validation data, i.e I retrieved whole amount of passages given for one query, i.e. 1000 passages for 1 query in most cases. The AP of my LR model is 0.0136 and NDCG of my LR model is 0.1331.

4 LAMBDAMART

LambdaMart [1] is a method of learning to rank. It turns the problem of ranking into the problem of fitting decision trees. Decision trees are normally fitted by using Gradient Boosting, while in LambdaMart, it uses a special "Lambda" to replace the gradient. In my experiment, I use xgbranker from XGBoost [2] to implement LambdaMart ranking algorithm.

4.1 Input representation of LambdaMart

I use the same representation of Logistic Regression when I implement LambdaMart. In other words, tokenization and embedding representations are obtained exactly from the previous codes. However, in LambdaMart we are required to fine-tune our hyperparameters. Thus, when I fine-tune hyperparameters, I use negative sampling to sample training data in order to speed up the processing of finding the optimal hyperparameter settings. For each query, I choose the ratio of postive samples and negative samples to be 5. After finding the optimal hyperparameters, I use this setting to train on the sampled training data and validate on validation data.

4.2 Hyperparameter Fine-tuning

I use bayesian optimization from [4] to fine-tuning the hyperparameters of LambdaMart. In the experiment, I choose to fine-tune "learning rate", "subsample", "colsample by tree", "max depth" and "num estimators". "learning rate" represents the learning rate of our ranking model. "subsample" represents the ratio of training data we will use to train each tree. "colsample by tree" represents the fraction of features used for training each tree. "max depth" represents the maximum depth we will allow our trees to have. "num estimators" represents how many trees we want in our model. I split my training set in to 5 folds and use cross validation to derive the averaged value of Average Precision coming from each tuning as the target to optimize using bayesian optimization. I run bayesian optimization for 10 times to find the optimal setting within predefined hyperparameter space.

The optimal value I found for the hyperparameters are, learning rate = 0.03, subsample = 0.9, colsample by tree = 0.69, max depth = 10, num estimators = 377.

4.3 Performance of LambdaMart

In my experiment of LambdaMart, I use whole training data to train my LambdaMart model, and I test all validation data, i.e I retrieved whole amount of passages given for one query, i.e. 1000 passages for 1 query in most cases. The AP of my LambdaMart model is 0.0358 and NDCG of my LambdaMart model is 0.1684.

5 NEURAL NETWORK

5.1 Input representation of Neural Network

I use the same tokenization of Logistic Regression when I implement Neural Network. However, given that the large amount of hyperparameters within neural network, I choose to use all the embedded representation of each token instead of using averaged embedded representation, which is more flexible when doing backward propagation. Same as Logistic Regression, attention masks are obtained for attention used in transformer- based neural network. I use negative sampling to sample training data in order to speed up the processing of training neural network. For each query, I choose the ratio of postive samples and negative samples to be 5. After the convergence of loss, I use trained network to validate on validation data.

5.2 Structure of Neural Network

In my experiment, I use transformer-based neural network to train my training data. The structure of my network is shown as: Figure

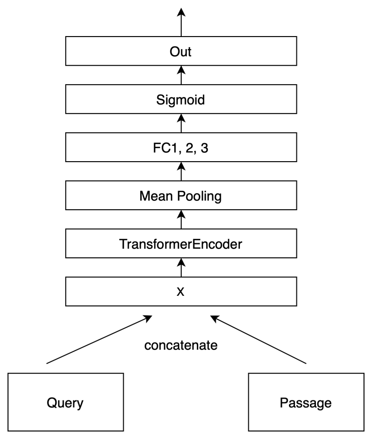

Figure 2: Network structure

2 shows the structure of my neural network. Note that our target is to predict the relevance between passage and query. Thus, it is natural to compute the attention between query and passage. In this case, I concatenate the input query and passage to form a full vector, the size of this full vector is (batch size, sequence length, embedding size). Then attention is computed within this full vector by using transformer encoder layer[6]. This stage is aimed for extract information of relationship between query and passage by using global attention within each token of two inputs. Further feed forward layers in transformer encoder is used to further learn more information of this concatenated vector. After encoded by transformer encoder, I use a mean pooling layer to pool each full vector on the dimension of sequence length, i.e. the size of full vector is (batch size, embedding size). This stage is aimed for preparing the vector for output. Then I use three fully-connected layers with activation function (ReLU) to extract information and decrease vector dimension to 1. Finally, I use sigmoid function to constain my output in range $[0, 1]$. Same as Logistic Regression, queries are padded to length of 50, and passages are padded to length of 300. This is because most queries and passages have lengths within that two ranges. Embedding size of one token of queries and passages is 128. I use 2 transformer encoder layer to encode my inputs. The dimension of feed forward layers inside transformer is set to be 64. Learning rate is set to 10^{-4}. I use Adam optimizer to optimize my network. I train my network for 150 epochs.

Note that the other structure (shown as figure 3) is also tried in my experiment. The intuition behind this design is that I wonder the attention mechanism between query and passage will really work. Thus I separate them when they are fed into transformer encoder. The result of this model is worse than the previous, which roughly prove that attention between query and passage will surely help model to grasp interactive information between queries and passages.

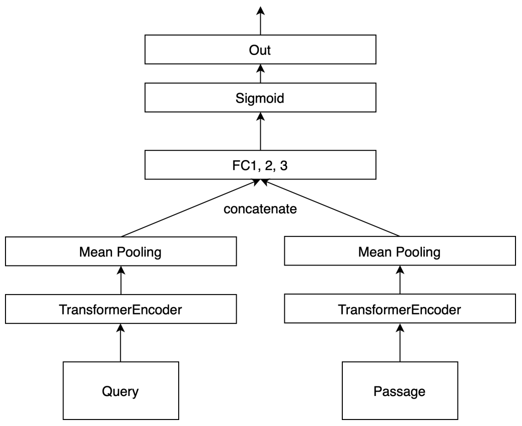

Figure 3: Network structure2

5.3 Performance of Neural Network

In my experiment of neural network, I use whole sub-sampled training data with positive and negative sample ratio equal to $\frac{1}{5}$ to train my neural network model, and I test all validation data, i.e I retrieved whole amount of passages given for one query, i.e. 1000 passages for 1 query in most cases. The AP of my neural network model is 0.0362 and NDCG of my neural network model is 0.1687.

REFERENCES

[1] Christopher J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview.

[2] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[3] D. R. Cox. 1958. The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* 20, 2 (1958), 215–242. http://www.jstor.org/stable/2983890

[4] Fernando Nogueira. 2014. Bayesian Optimization: Open source constrained global optimization tool for Python. https://github.com/fmfn/BayesianOptimization

[5] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (apr 2009), 333–389. https://doi.org/10.1561/1500000019

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[7] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144