Information Retrieval and Data Mining Coursework 2

ABSTRACT

This project is dedicated to the development of information retrieval (IR) models that enhance the ranking of a set of passages in response to a specific query. Three re-ranking models with different methodologies are built: Logistic Regression, LambdaMART, and a Neural Network approach. Each model is trained by feeding data from "train_data.tsv". Their performance is assessed by computing **Mean Average Precision (mAP)** and **mean Normalised Discounted Cumulative Gain (mNDCG)**, with the data from "validation_data.tsv". Additionally, the three models are applied to all queries from "test-queries.tsv" and the passages from "candidate_passages_top1000.tsv" to score each passage's relevance to a specific query. The top 100 passages for each query are then recorded in "LR.txt" for Logistic Regression, "LM.txt" for LambdaMART, and "NN.txt" for the Neural Network model.

1 RETRIEVAL QUALITY EVALUATION

1.1 Average Precision (AP)

Average Precision (AP) is one of the metric to evaluate the performance of a retrieval model. For a given query, AP approximates the Area Under curve for precision-recall, thus offering a straightforward evaluation method: the higher the AP, the more effective the IR model. The formula for AP is as follows:

$$AP = \frac{1}{N} \sum_{k=1}^{K} Precision(k) \times rel(k) \tag{1}$$

where k is the rank, K is the number of all passages in the list, N is the number of relevant passages, and $rel(k)$ is either 0 or 1, with 1 indicating passage at rank k is relevant to the query.

Mean Average Precision (mAP) is calculated by taking the mean of the AP values across all queries.

In my implementation, I constructed a two-dimensional dictionary for relevance lookup, with the structure **dict[qid][pid] = relevance**. When relevance score is calculated from IR models, a dataframe containing qid, pid, and the score is sorted in descending order by the score within each qid group. The mAP is then computed by iterating through the sorted dataframe, returning the mAP for the IR model.

1.2 Normalised Discounted Cumulative Gain (NDCG)

Normalised Discounted Cumulative Gain (NDCG) is another frequently used metric for comparing IR models. Unlike AP, NDCG takes into consideration with a scale of relevancy scores, offering the ability to fine-tune which passages are more relevant than others. Similar to AP, a higher NDCG value indicates a better-performing IR model. The formula for NDCG is shown below:

$$NDCG = \frac{DCG}{IDCG} \tag{2}$$

$$DCG = \sum_{k=1}^{K} \frac{rel(k)}{\log_2(k+1)} \tag{3}$$

where $rel(k)$ is the scale of relevancy scores, 0 or 1 in this project.

The Ideal Discounted Cumulative Gain (IDCG) is computed using the same formula as DCG, shown in Equation 3, but with the results arranged in the ideal order.

1.3 Performance of BM25

In this project, the BM25 model developed from Coursework 1 is reused. Its output serves as a benchmark for evaluating the performance of other models.

Table 1 shows the mAP and mNDCG of BM25 calculated using the data from "validation_data.tsv".

Table 1: Performance of BM25

Model	mAP	mNDCG
BM25	0.2399	0.3826

2 LOGISTIC REGRESSION

2.1 Down Sampling

Since the train dataset is too large for my computer, to reduce the computational demands, a subset of data is generated in which all the relevant passages are retained while random irrelevant passages are removed, as known as negative class downsampling. In the new dataset, each query has no more than 100 passages including relevant passages. As majority of queries have number of relevant passages less than 5, the positive-to-negative sample ratio is now approximately 1:20, a significant reduction from the original ratio of about 1:200. This downsampling method not only reduces the dataset size but also mitigates the issue of imbalance, leading to a logistic regression model that is less likely to be biased towards irrelevant passages.

2.2 Word Embedding

To prepare the data for the regression model, all text must be converted into vector format. For this project, words are transformed into word embeddings using the GloVe (Global Vectors for Word Representation) format, which is well-suited for diverse datasets and topics (our dataset is also diverse).

The initial step in this conversion process involves tokenising sentences using a function developed in Coursework 1, removing all stopwords, and applying lemmatization. Next, the pretrained GloVe model named "glove.6B.100d", which was trained on the 2014 Wikipedia dataset by Google, is used to convert words to 100-dimensional word vectors. The model is donwloaded as a ".txt" file and loaded into Python by gensim API, with functions glove2word2vec() and KeyedVectors.load_word2vec_format(). The

model serves as a dictionary, it looks up the corresponding word vector for a given word.

Once all the words in the passages have been converted into GloVe vectors, we calculate the mean of these embeddings, resulting in average embeddings which represent sentences. The average embeddings of a query and passage are then concatenated, and an intercept term with value 1 is added to created a feature vector with shape (1, 201), which is fed into regression model in later stages.

However, GloVe has a shortcoming - it does not return anything for an out-of-vocabulary word (unseen word). To mitigate this shortcoming, my first approach was to drop out the out-of-vocabulary word (OOV). But there is a query in the dataset which is comprised entirely of OOV words, leading to an empty feature (null vector). Since the regression model requires consistent input shapes, this null vector from OOV query caused the logistic regression unable to be trained. To overcome this issue, I changed the way to handle OOV - a random 100-dimensional vector with values ranging from -1 to 1 is assigned to OOV. This ensures that every query and passage has a feature representation.

2.3 Logistic Regression implementation

Since the labels of the data can be either relevant (1) or irrelevant (0) only, the problem of passage reanking can be treated as binary classification problem. With a given query, the query-passage pair with higher probability is considered to be more relevant.

The logistic regression model is trained to minimise the loss function as shown in Equation 4,

$$C(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})] \quad (4)$$

where

$$\hat{p} = \sigma(\theta^{\mathrm{T}}x) = \frac{1}{1+\exp(-\theta^{\mathrm{T}}X)} \quad (5)$$

x is the input with shape (<number of sample>, 201), m is the batch size, y is the label with either 0 or 1, and θ is the weight of each term.

In each iteration, the regression model learns from the trainig set and improve its weights by gradient descent algorithm. Gradient descent helps minimising the cost function by calculating the gradient of the cost function, logistic weight terms are updated in each iteration by substracting the product of learning rate and gradient descent, which is shown in Equation 6 . The formula of gradient descent for logistic regression is also shown in Equation 7:

$$\theta_{n+1} = \theta_n - lr \times \frac{\partial C}{\partial \theta} \quad (6)$$

where lr is the learning rate and

$$\frac{\partial C}{\partial \theta} = \frac{1}{m}X^{\mathrm{T}}[\frac{1}{1+e^{-\theta^{\mathrm{T}}X}} - y] \quad (7)$$

To save the computer's memory, the logistic regression model is trained with mini-batch with batch size 5000 per iteration. More specifically, in each epoch, the model is first trained with the first 5000 records, then the next batch of 5000, until all the samples are fed in.

To fine-tune the logistic regression model and investigate the effect of learning rate on the model, 5 logistic regression models with differnt learning rates are trained and their losses are plotted in Figure 1. It is obvious that the model with higher learning rate converges more rapidly and tends to be 0 with fewer epochs. Although higher learning rate can speed up the training process, it increases the risk of overfitting as the model updates its weights significantly over each iteration and starts to memorise the result at later epochs.

To avoid overfitting, hyperparameters with $epoch = 100$, (learning rate) $lr = 0.001$ and $batch\_szie = 5000$ are selected.

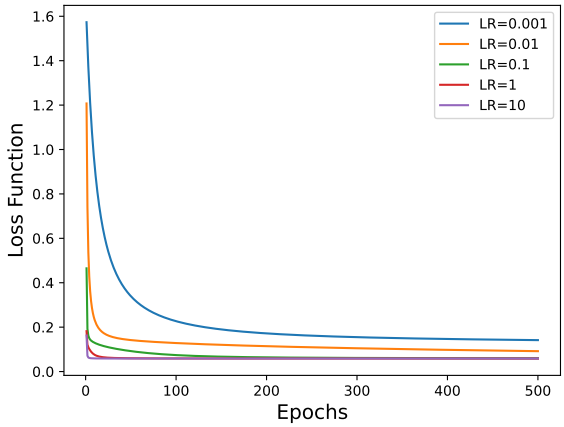

Figure 1: Effect of Learning Rate on Logistic Regression Loss

2.4 Performance of Logistic Regression

The mAP and mNDCG of Logistic Regression model are computed and shown in Table 2.

Table 2: Performance of Logistic Regression Model

Model	mAP	mNDCG
LR	0.0135	0.1333

3 LAMBDAMART

3.1 LambdaMART Introduction

LambdaMART is a pairwise ranking model that evaluates the degree of relevance between every pair of samples in the training set, yielding a relevance score for each query-passage pair. This model is particularly well-suited for re-ranking tasks.

3.2 Input Representation

The input representation used to train this LambdaMART model is similar to the one used for logistic regression in the previous section. The only difference is that intercept term with value 1 is not required to be concatenated in the input vector. In other words, the input vector is now composed of two average embedding, one from query and one from passage. Hence, each of the input vector is in shape (1,

200). Additionally, as written on LambdaMART tutorial, the input training set must be sorted by "qid". As an extra requirement, a 1-dimensional vector named "qid" which contains all qid in ascending order is also pass into the LambdaMART model.

3.3 LambdaMART Hyperparameters Fine Tuning

In this project, LambdaMART is first set up with consistent variable, with $tree\_method$ = "$hist$", $device$ = "$cuda$", $objective$ = "$rank:pairwise$" and $lambdarank\_pair\_method$ = "$mean$". Then, a grid search is performed to find the optimal hyperparameters. The grid is defined as shown in Table 3. After training 48 different configurations and validated with "validation_data.tsv", the optimal hyperparameters in the grid are found and are recorded in Table 3.

Table 3: LambdaMART Hyperparameters Grid

Variable	Grid Values	Optimal Value
learning_rate	[0.1, 1]	0.1
alpha	[0, 0.1]	0
gamma	[0, 0.1, 1]	0
max_depth	[6, 7]	6
n_estimators	[100, 200]	100

3.4 Performance of LambdaMART

The mAP and mNDCG of LambdaMART model are computed and shown in Table 4.

Table 4: Performance of LambdaMART Model

Model	mAP	mNDCG
LM	0.0181	0.1439

4 NEURAL NETWORK

4.1 Selection of Neural Network

In this section, Recurrent Neural Network (RNN) is selected to predict the relevance between each pair of query and passage. RNN is chosen due to its ability to process sequential data. Since Neural Language Processing (NLP) data is often considered as sequential data, in which the meaning of a word in a sentence is affected by words located before and after itself, RNN is has outstanding performance on handling this kind of relationship. Unlike other models that necessitate input data to be in the form of fixed-size vectors, RNNs are flexible, allowing for inputs of varying dimensions. This flexibility enables the conversion of sentences into extended vectors that comprise the embeddings of all constituent words, rather than relying on a singular, fixed-size average embedding. This richer input format furnishes RNNs with a more detailed dataset, enhancing their ability to accurately predict relevance compared to models limited to average embeddings.

4.2 Input Representation

To utilise the advantages of RNN, in theory the passages should be represent as a list of word embedding representing all the words. For example: [<query word embeddings>, <passage word embeddings>], where <passage word embeddings> is <word embedding of word 1>, <word embedding of word 2>, ..., <word embedding of word n>. However, due to the extremely high computational demands and limited Random Access mMemory (RAM) availabe on my computer, I was unable to feed training in this format to the RNN. Therefore, the same methodology of average embedding from LambdaMART is reused in this section.

4.3 Recurrent Neural Network Architecture

RNN is a special neural network in which the output from previous step is also fed as input to the current step. This architecture makes RNNs particularly adept at processing sequential data, such as text for prediction and classification tasks. Figure 2 shows the architecture of a simple RNN. The key feature of an RNN, highlighted by the component labeled "A", is that the output of this layer not only progresses to the next layer but is also looped back as input to itself in the subsequent step. By expanding the RNN structure, it becomes more clear how the output from one step is fed back into the network at the next step.

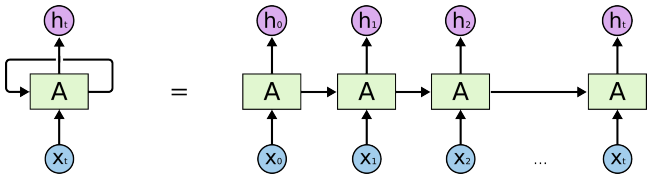

Figure 2: Unrolled Recurrent Neural Network. Illustration by Christopher Olah. (https://colah.github.io/posts/2015-08-Understanding-LSTMs/?ref=blog.paperspace.com)

However, RNN suffers from short-term memory, it fails to maintains the important information found in earlier steps to later ones. To overcome this issue, Long Short-Term Memory (LSTM) is applied. LSTM acts as a memory cell (as shown in Figure 3), it remembers the useful information while forgetting less relevant information. With this working principle, RNN with LSTM can better store the useful information discovered in earlier steps and apply to later steps.

Furthermore, for NLP tasks, sometimes it might help if information from future (next word) is also taken into account during prediction. Bidirectional layer is introduced to cater the needs. Therefore, Bidirectional layer is also included in our RNN model and the architecture of RNN is shown in Figure 4.

4.4 Performance of RNN

The mAP and mNDCG of RNN model are computed and shown in Table 5.

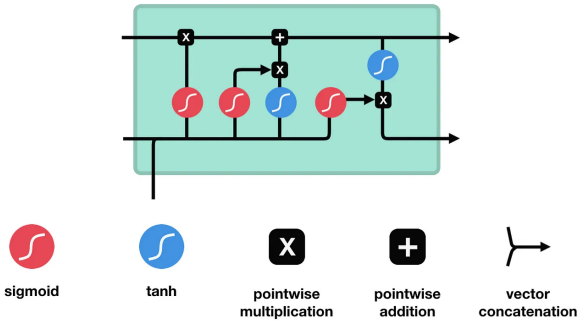

Figure 3: LSTM Cell Overview. Illustration by Michael Phi. (https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21)

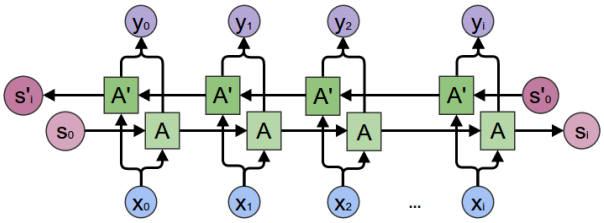

Figure 4: Bidirectional Recursive Neural Network. Illustration by Christopher Olah. (https://colah.github.io/posts/2015-09-NN-Types-FP/)

Table 5: Performance of RNN Model

Model	mAP	mNDCG
LM	0.0112	0.1293

5 LIMITATION

Overall, the performance of Logistic Regression, LambdaMART and Recurrent Neural Network do not have an excellent perforamce on mAP and mNDCG. This might be caused by the following reasons:

1) The training dataset is quite small and the subset of the training dataset is used.

2) The training dataset is highly imbalanced, which leads to poor performance on determining relevant passage.

3) Average embedding is extensively used for training, since many semantic information is lost during the process of averaging, the IR model cannot extract sufficient feature from the training sets.

4) For RNN, since the computational time for one epoch takes around 10 minutes, for demonstraction purpose, the number of epoch is selected to be 1 only. However, with more epoch, the RNN might be able to provide better prediction.