COMP0084 Information Retrieval and Data Mining (2023/24) Coursework 2 - Report

Yuen Chung CHAN (23077688)
MSc Data Science & Machine Learning
University College London

ABSTRACT

This report documented our work on coursework 2 for the module COMP0084 Information Retrieval and Data Mining. The coursework consists of 4 tasks. In Task 1, we implemented two evaluation metrics for a retrieval systems and evaluated on a BM25 model. Subsequently, Tasks 2 through 4 focused on three different models for document relevancy ranking, specifically Logistic Regression, LambdaMART, and a Two-tower Neural Network architecture. Each model was evaluated on its efficacy in ranking.

1 DATASET

This coursework employed two distinct datasets: `train_data.tsv` and `validation_data.tsv`, containing 4,364,340 and 1,103,040 records respectively. Each record includes several fields: qid (Query ID), pid (Passage ID), query (the textual query input), passage (the corresponding textual passage), and a binary indicator of relevance to the query. In this coursework, all learning-based models were trained using records in `train_data.tsv` and all models were evaluated using records in `validation_data.tsv` after training.

1.1 Sub-sampling the Training Set

The original training set has 4,590 different queries but was very unbalanced - for each query, there were only 1 to 2 relevant records among a vast majority of 4,359,542 irrelevant ones, making a total of just 4,797 relevant records. To address this issue and improve the training process efficiency of machine learning models, we used sub-sampling. This technique kept all the relevant records but only randomly selected 5 irrelevant records per query. As a result, a smaller and more balanced training set with 27,726 records in total was created. This sub-sampled set was then used to train all the learning-based models in the coursework.

While it was possible to include more irrelevant records per query in the training set, we opted to limit this number to 5 per query due to the limitation of available computing resources for this coursework.

2 TASK 1 - EVALUATION METRICS AND BM25

In this task, we implemented two ranking evaluation metrics, Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG), from scratch. The implemented metrics were used to evaluate the performance of the probabilistic model BM25 on the validation dataset.

2.1 Average Precision

Average Precision (AP) is a metric that evaluates the quality of the ranked retrieval results of a single query. It is calculated by averaging the precision scores at each rank where a relevant document is retrieved. Mathematically, AP is defined as:

$$AP = \frac{\sum_{k=1}^{n} P(k) \times rel(k)}{\text{number of relevant documents}}$$

where n is the length of rankings, $P(k)$ is the precision at cut-off k in the list, $rel(k)$ is an indicator function equals to 1 if the item at rank k is relevant, and 0 otherwise. An extension of AP that provides a single figure measure of quality across multiple queries is Mean Average Precision (mAP). It is calculated as the mean of the AP scores for each query, offering a comprehensive overview of the retrieval system's performance over a collection of queries. Mathematically, mAP is defined as:

$$mAP = \frac{1}{Q} \sum_{q=1}^{Q} AP_q$$

where Q is the total number of queries, and AP_q is the AP for the q^{th} query.

2.2 Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) accounts for the position of the relevant documents in the ranking order, attributing higher scores to hits at top ranks. Unlike AP, NDCG can handle relevance score instead of binary relevance only. The Discounted Cumulative Gain (DCG) is defined as:

$$DCG = \sum_{i=1}^{n} \frac{rel_i - 1}{\log_2(i+1)}$$

where n is the length of rankings, rel_i is the relevance score of the result at position i. NDCG normalizes the DCG value by comparing it to the ideal DCG (IDCG), which represents the highest possible DCG given the set of relevance judgments:

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

To assess the performance of a retrieval system, one can use Mean Normalized Discounted Cumulative Gain (mNDCG) over a collection of queries defined as

$$mNDCG = \frac{1}{Q} \sum_{q=1}^{Q} NDCG_q$$

where Q is the total number of queries, and $NDCG_q$ is the NDCG for the q^{th} query.

2.3 BM25 Performance

We evaluated the BM25 model performance on the validation set using mAP and mNDCG. The results is presented in Table 1. The implementation of BM25 is same as that in coursework 1.

Table 1: Performance of BM25 on Validation Data

mAP	mNDCG
0.1871	0.3266

3 TASK 2 - LOGISTIC REGRESSION

The focus of this task is on representing query text and passage text using word embeddings model and develop a logistic regression model for ranking.

3.1 Model

Logistic regression is a binary classification model that predicts the probability of the positive class. Given a input feature vector $\vec{x} \in \mathbb{R}^D$, the model's prediction $\hat{y} \in (0, 1)$ is formulated as:

$$\hat{y} = \sigma(\vec{w} \cdot \vec{x} + b)$$

where $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function, $\vec{w} \in \mathbb{R}^D$ is the model's weight vector and $b \in \mathbb{R}$ is the bias term. The bias term can be absorbed into the feature vector, leading to a simpler formulation

$$\hat{y} = \sigma(\vec{w} \cdot \vec{x})$$

where $\vec{x}$ and $\vec{w}$ both have a dimension of $D + 1$.

3.1.1 *Loss Function.*

To obtain the optimal weights of LR, a common loss function to minimise is the Binary Cross Entropy loss (BCE) defined as

$$\text{BCE}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

where y is the true label (0 or 1) and $\hat{y}$ is the predicted probability of the positive class. In this coursework, as the training data was still imbalanced after the sub-sampling, we adopted the Weighted BCE loss (WBCE) defined as

$$\text{WBCE}(y, \hat{y}) = -w_{\text{pos}} \cdot y \log(\hat{y}) - w_{\text{neg}} \cdot (1 - y) \log(1 - \hat{y})$$

This loss function modifies the standard binary cross entropy loss by introducing weights for the positive and negative classes, denoted as w_{pos} and w_{neg}, respectively. In this coursework, w_{pos} was set as the ratio of negative records to positive records and w_{neg} was set as 1.

3.1.2 *Mini-batch Gradient Descent.*

We utilised mini-batch gradient descent on WBCE to optimise the model weights. For a batch of N data, we computed the weighted mean of WBCE as the batch loss as follows:

$$L_{\text{batch}} = \frac{1}{\sum_{i=1}^{N} w_{\text{pos}} \cdot y_i + w_{\text{neg}} \cdot (1 - y_i)} \sum_{i=1}^{N} \text{WBCE}(y_i, \hat{y_i})$$

The gradient of the loss with respect to the model weights $\vec{w}$ was computed for each mini-batch. The weights, initialised with zeros, were then updated by:

$$\vec{w} \leftarrow \vec{w} - \alpha \frac{\partial}{\partial \vec{w}} L_{\text{batch}}$$

$$\frac{\partial}{\partial \vec{w}} L_{\text{batch}} = \frac{\sum_{i=1}^{N} (w_{\text{pos}} \cdot y_i + w_{\text{neg}} \cdot (1 - y_i))(\hat{y_i} - y_i)\vec{x_i}}{\sum_{i=1}^{N} w_{\text{pos}} \cdot y_i + w_{\text{neg}} \cdot (1 - y_i)}$$

where α is the learning rate. These updates were performed iteratively over mini-batches of the training data until convergence was reached or a predetermined number of iterations was completed.

3.2 Input Processing

3.2.1 *Text Preprocessing and Tokenization.*

Preprocessing and tokenization of text were conducted using PyTorch's `torchtext` library [1]. We employed its basic English tokenizer [2] which works as follows:

1. Convert all text to lowercase.
2. Implement basic text normalization, which includes:
   - Add spaces before and after punctuation marks.
   - Remove double quotes (").
   - Replace HTML break tags (<br>) with a single space.
   - Replace semicolons (;), colons (:), and consecutive spaces with a single space.
3. Split into tokens based on white spaces.

3.2.2 *Text Embedding.*

For the purpose of computing embeddings for passages and queries, we utilized the pretrained GloVe-42B 300D model [1]. Our implementation incorporates the accessible API in `torchtext` for downloading and employing a variety of pretrained word embeddings.

Text embedding is collected as follows: for each passage and query, we first tokenize the text into individual words as mentioned. Subsequently, each word's embedding is retrieved from the GloVe-42B 300D dataset. The embedding for the entire text is then calculated as the mean of these word embeddings.

Specifically, given a tokenized text $T = \{W_1, W_2, ..., W_L\}$, the embedding for T, denoted as $\vec{E_T}$, is computed using the equation:

$$\vec{E_T} = \frac{1}{L} \sum_{i=1}^{L} \vec{E_{W_i}}$$

where $\vec{E_{W_i}}$ is the GloVe embedding for word W_i. This procedure is applied uniformly to both passage and query texts.

3.2.3 *Input Feature Construction.*

For each record in our dataset, the input feature vector is constructed by concatenating the embeddings of the query with those of the passage, supplemented by the inclusion of a bias term. Given a query embedding $\vec{E_q}$ of dimension 300 and a passage embedding $\vec{E_p}$ also of dimension 300, the feature vector $\vec{x}$ is formed as

$$\vec{x} = [\vec{E_q} \; ; \vec{E_p} \; ; 1]^\top .$$

Here, the semicolon represents the concatenation operation, and the number 1 represents the bias term added as the final element of the vector. This 601-dimensional vector serves as the input to our logistic regression model.

Next, we performed standardization, a common data processing technique in machine learning, on those 600 embedding dimension. It involves rescaling these dimensions to have a mean of zero and a standard deviation of one. By standardizing the features, we ensure that a model treats all features equally.

[1] https://pytorch.org/text/stable
[2] https://pytorch.org/text/stable/data_utils.html#get-tokenizer

3.3 Training and Evaluation

3.3.1 Training. The model was set to train for a maximum of 2000 epochs. This number of epochs was chosen given the complexity of the task and the size of the dataset. A tolerance parameter was set to monitor the improvement of the loss. This parameter, set at 10^{-6}, ensures that training will be halted once the relative change in loss between epochs falls below this threshold and prevents over-fitting. Furthermore, the training employed a mini-batch size of 64.

Four models were trained on using three different learning rates $[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]$. The models with learning rates 10^{-3} and 10^{-4} converged after 360 epochs and 1359 epochs respectively, while the other two models had not converged within 2000 epochs.

3.3.2 Evaluation. The best model was chosen according to their F1 scores on the training data. The model with learning rate 10^{-3} was considered the best as it has the highest F1 score on the training set.

We evaluated this model performance on the validation set using mAP and mNDCG. The results is presented in Table 2. It is noted that the model performs significantly worse than the BM25 model.

Table 2: Performance of Logistic Regression on Validation Data

mAP	mNDCG
0.0236	0.1484

The poor performance might be attributed to the generation of passage and query embeddings, specifically the averaging of word embeddings to represent these texts. While this approach simplifies the representation of texts into fixed-dimensional vectors, it leads to a significant loss of information.

Averaging word embeddings to form a single vector for an entire passage or query effectively collapses all word-level information into a unified representation. This process disregards the syntactic structure and semantic relationships between words within the text.

Furthermore, this method of embedding generation does not account for the weighting of words based on their importance or relevance within the text. All words contribute equally to the final embedding, diluting the influence of key terms in the query.

3.4 Effects of Learning Rate

The learning rate is a hyperparameter in our model training. To investigate the influence of different learning rates on our model's training, we show the training loss curves of the four trained models in Figure 1.

The curve corresponding to the learning rate of 10^{-3} (red line) shows a rapid decrease in loss, indicating that the model is quickly learning from the data. It flattened out and converged within a smaller number of epochs.

The curve for the learning rate of 10^{-4} (green line) descends more gradually compared to the 10^{-3} learning rate and took more epochs to converged. However, it converged at a loss value that is slightly lower, making it a potential candidate model that should be tested on the validation set.

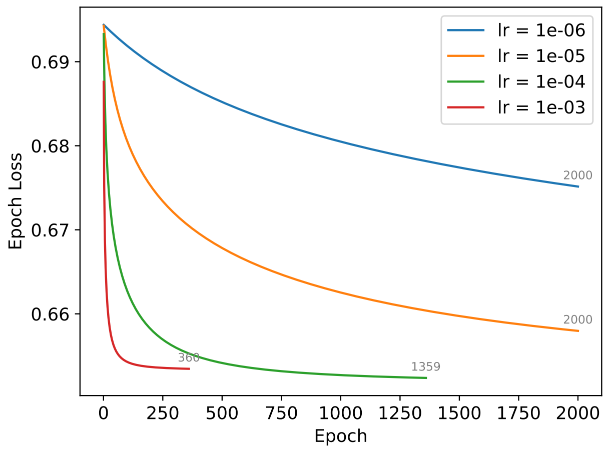

Figure 1: Training loss of LR models with different learning rate.

Lastly, the curves for the other two models of learning rates 10^{-5} and 10^{-6} shows slow descent. Both model did not converge within the 2000 epochs window. The slope of this curve is less steep, indicating very gradual learning. At the end of 2000 epochs, the losses remains above that of the other two rates, and the curve is flattening out. The learning is too slow for practical purposes within the given epoch constraint.

4 TASK 3 - LAMBDAMART

This task implements LambdaMART [3] learning to rank algorithm from `XGBoost` [4] gradient boosting library.

4.1 Model

LambdaMART is an algorithm that combines the principles of LambdaRank and MART (Multiple Additive Regression Trees) to create a powerful learning-to-rank model that optimizes ranking measures directly [2]. In `XGBoost`, one can set the model's objective as `rank:ndcg` to use the LambdaMART algorithm.

4.2 Input Features

Instead of using the same input features as task 2, different features were constructed in this task. Upon experimenting with various combinations of query and passage embeddings, it was revealed that utilizing the raw embeddings directly as input did not yield satisfactory results. This discovery led to a refined feature selection strategy, where the inputs were shrank down to the dot product of the query and passage embeddings and the norms of each embedding. The dot product serves as a measure of the semantic similarity between the query and passage, while the inclusion of the norms provides a sense of the magnitude of the embeddings. Using the query embedding $\vec{E_q}$ and the passage embedding $\vec{E_p}$ obtained in

[3]https://xgboost.readthedocs.io/en/latest/tutorials/learning_to_rank.html
[4]https://xgboost.readthedocs.io/en/latest/index.html

task 2, the feature vector $\vec{x}$ is constructed as

$$\vec{x} = [\vec{E_q} \cdot \vec{E_p} \quad ||\vec{E_q}||_2 \quad ||\vec{E_p}||_2]^\top,$$

where $||\cdot||_2$ is the Euclidean norm. This 3-dimension feature vector after standardisation was passed as the LambdaMART model input.

4.3 Training and Evaluation

4.3.1 Training. The training process of the LambdaMART algorithm was executed using a 5-fold cross-validation to ensure the model's generalizability. The hyperparameter tuning was conducted via a grid search over the set of parameters presented in Table 3. The low dimenionality of model input enables an efficient training process for searching a large grid.

Table 3: Hyperparameter grid for LambdaMART model training.

Hyperparameter	Values
Learning rate	0.01, 0.05, **0.1**, 0.3
Tree depth	3, 4, **5**
Minimum instances for split	**1**, 2, 3
L2 regularization strength	0, **0.05**, 0.1
Row subsampling rate	0.7, **0.85**, 1

The model demonstrating the highest performance across the cross-validation folds, based on the grid search, was selected to advance to the validation phase. Over 324 sets of hyperparameters, the set that gave the highest mNDCG during cross-validation was selected as the best model for evaluation. This set of hyperparameters is highlighted in bold in Table 3.

4.3.2 Evaluation. A model using the best hyperparameters was retrained on the full training data and evaluated on the validation data. The results is presented in Table 4.

Table 4: Performance of LambdaMART on Validation Data

mAP	mNDCG
0.0436	0.1756

In terms of mAP, the LambdaMART performs 88% better than the logistic regression model. While the improvement in mNDCG is not as significant, the LambdaMART model's mNDCG is also 18% higher than that of the logistic regression model.

5 TASK 4 - NEURAL NETWORK

In this task, we incorporates a neural network architecture known as the two-tower model. This architecture is specifically designed to handle and compare two different types of input data: query and passage texts. It is a modern architecture that is commonly adopted for recommendation and retrieval systems [3].

5.1 Network Architecture

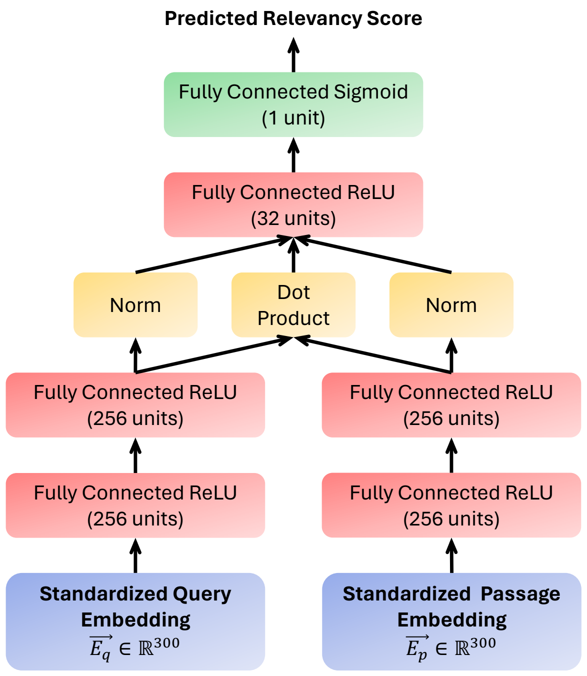
Predicted Relevancy Score
Fully Connected Sigmoid (1 unit)
Fully Connected ReLU (32 units)
Norm
Dot Product
Norm
Fully Connected ReLU (256 units)
Fully Connected ReLU (256 units)
Fully Connected ReLU (256 units)
Fully Connected ReLU (256 units)
Standardized Query Embedding $\vec{E_q} \in \mathbb{R}^{300}$
Standardized Passage Embedding $\vec{E_p} \in \mathbb{R}^{300}$

Figure 2: Two-tower Neural Network Architecture adopted in Task 4

Figure 2 shows the network structure. In short, each tower acts as an encoder of the embedding vector and produces an encoded hidden state. The two hidden states from the two towers are used to produce a feature vector following the procedures in task 3, i.e. the norm of each state and their dot product. These features are kept due to the significant improvement shown in task 3. The whole network is structured as follows:

- **Input Embeddings:** The model receives two inputs: the standardized query embedding $E_q \in \mathbb{R}^{300}$ and the standardized passage embedding $E_p \in \mathbb{R}^{300}$, both represented by 300-dimensional vectors.
- **Two-Tower Layers:** Each input is passed through its respective tower, consisting of two fully connected layers with ReLU activations, containing 256 units each. These layers aim to non-linearly map the raw standardized embeddings into a transformed feature space.
- **Combined Features:** The transformed features from the query and passage towers are combined by computing their dot product, alongside with the individual norms of both transformed feautures.
- **Additional Transformation:** The combined features and norms are further processed by another fully connected layer with ReLU activation and 32 units, creating the information for final relevancy prediction.
- **Prediction Output:** The last transformation layer's output is fed into a fully connected layer with a sigmoid activation

function consisting of a single unit. This layer outputs the predicted relevancy score, a value between 0 and 1, predicting the relevancy between the query and the passage.

The network as well as the whole training process were implemented using the PyTorch [5] library.

5.2 Training and Evaluation

5.2.1 Training. The model was trained using the Adam optimiser [4] to minimise the weighted binary cross entropy loss discussed in Section 3.1.1. It was set to be trained for 30 epochs with batch size and learning rate of 1024 and 10^{-3} respectively.

5.2.2 Evaluation. We evaluated this model performance on the validation set using mAP and mNDCG. The results is presented in Table 2. The two-tower model performs slightly better than the LambdaMART model in terms of both metrics.

5.3 Future Work Suggestion

To further improve the model, there are several directions:

- **More complex features:** Instead of using the dot product of the two hidden states, one may try more features such as the cosine similarity between the two vectors.
- **More complex tower:** Instead of using two fully connected layers for the two towers, one may employ encoders with self-attention mechanism.

Table 5: Performance of Two-tower Neural Network on Validation Data

mAP	mNDCG
0.0442	0.1781

REFERENCES

[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[2] Chris J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, June 2010.

[3] Ji Yang, Xinyang Yi, Derek Zhiyuan Cheng, Lichan Hong, Yang Li, Simon Xiaoming Wang, Taibai Xu, and Ed H. Chi. Mixed negative sampling for learning two-tower neural networks in recommendations. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 441–447, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] https://pytorch.org/docs/stable/index.html