COMP0084 Information Retrieval and Data Mining

Chung-Yu Wei (24057783)
MSc Data Science & Machine Learning
ucabc47@ucl.ac.uk

Abstract

This project develops and evaluates multiple models for passage retrieval. In Task 1, we implement BM25 and define evaluation metrics (mAP and NDCG). Task 2 builds a logistic regression model using word embeddings and investigates learning rate effects. Task 3 employs LambdaMART with 3D feature vectors and hyperparameter tuning via cross-validation. Finally, Task 4 develops a neural network using a feed-forward architecture in PyTorch for passage re-ranking.

1 Introduction

1.1 Dataset

The project mainly focusing on two files. `train_data.tsv` and `validation_data.tsv` where the training and validation sets are provided respectively with a format `<qid pid queries passage relevancy>`. In the following parts of this report, all of our learning models (task2-task4) are trained in the train data and then evaluated with metrics in validation data.

1.2 Metrics 1: Average Precision (AP)

Average Precision is computed by averaging the precision values at each rank where a relevant document is retrieved. Let $\mathrm{rel}(k)$ be an indicator function that is 1 if the item at rank k is relevant and 0 otherwise, and let $P(k)$ be the precision at rank k. Then, for a ranking of n documents, AP is defined as:

$$\mathrm{AP} = \frac{\sum_{k=1}^{n} \left[P(k) \cdot \mathrm{rel}(k)\right]}{\sum_{k=1}^{n} \mathrm{rel}(k)}. \tag{1}$$

This metric summarizes how well the retrieval system orders the relevant documents among the top-ranked positions, giving higher scores to systems that retrieve relevant documents earlier.

To evaluate performance across a set of queries Q, the Mean Average Precision (MAP) is computed by taking the mean of AP over all queries:

$$\mathrm{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{AP}(q). \tag{2}$$

These metrics reward systems that retrieve relevant documents early and measure how well a retrieval system ranks relevant items for each query.

1.3 Metrics2: Normalized Discounted Cumulative Gain (nDCG)

Normalized Discounted Cumulative Gain (nDCG) is a ranking metric that evaluates the quality of a ranked list by considering both the relevance and the position of documents. The Discounted Cumulative Gain (DCG) is computed as:

$$\mathrm{DCG} = \sum_{k=1}^{n} \frac{2^{\mathrm{rel}(k)} - 1}{\log_2(k+1)},$$

where n is the number of retrieved documents. The Ideal DCG (IDCG) is the DCG obtained with an optimal ranking (i.e., documents sorted in descending order of relevance). The normalized version is then given by:

$$\mathrm{nDCG} = \frac{\mathrm{DCG}}{\mathrm{IDCG}}.$$

Mean nDCG (mNDCG) is defined as the average nDCG over all queries in the set Q:

$$\mathrm{mNDCG} = \frac{1}{|Q|} \sum_{q \in Q} \mathrm{nDCG}(q).$$

This metric rewards retrieval systems that rank highly relevant documents at top positions.

2 Task1: Evaluating Retrieval Quality.

In this task, our objective is to assess the effectiveness of a passage retrieval system using a baseline BM25 model, and our code performs the following steps:

Data Preprocessing: We preprocess the validation dataset by converting text to lowercase, removing punctuation and stopwords, and applying tokenization and lemmatization. This prepares both queries and passages for further analysis.

Inverted Index Construction: An inverted index is built from the tokenized passages, mapping each token to the passages in which it appears along with its frequency. This index is essential for efficient BM25 score computation.

BM25 Scoring: For each query, the BM25 model computes a relevance score for every candidate passage using the inverted index and document length statistics. The scores are used to rank the passages.

Evaluation: The ranked lists are then evaluated against the ground truth relevance labels in the validation data. The evaluation uses metrics mAP and mNDCG to assess retrieval performance at cutoff 20 and 100.

Table 1: BM25 Performance at Different Cutoffs

Cutoff	mAP	mNDCG
Top-100	0.2412	0.3505
Top-20	0.2345	0.3120

As shown in Table 1, including more passages (Top-100) yields a slightly higher mAP and mNDCG than restricting the ranking to the Top-20. Both metrics reward retrieval systems that return relevant documents at higher ranks, these results provide a baseline for subsequent learning-based models.

3 Task2: Logistic Regression (LR)

Below are the following key steps we do in this section:

Data Subsampling and Preprocessing: Due to the large training dataset with many non-relevant examples, we retain all positive instances (relevancy > 0) and randomly sample 5% of the negatives (relevancy = 0) using a fixed seed for reproducibility. After merging and shuffling, we clean the queries and passages by function `preprocess_text`.

Embedding Preparation: We construct a unified corpus from the cleaned queries and passages to train a Word2Vec model using the skip-gram architecture. Each word in the corpus is mapped to a 100-dimensional vector. For each query and passage, we compute its embedding by averaging the vectors of all tokens in the text. This results in a fixed-length representation for both queries and passages.

Feature Matrix Construction: For every query–passage pair, the corresponding feature vector is created by concatenating the query embedding and the passage embedding. This yields a feature vector of 200 dimensions that encapsulates information from both components, serving as the input to the logistic regression model.

Logistic Regression Training: We implement logistic regression from scratch using gradient descent with binary cross-entropy loss. For each training example, with feature vector $\mathbf{x}$ (formed by concatenating the query and passage embeddings) and binary label $y \in \{0, 1\}$, the model computes the predicted relevance probability as:

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^\top \mathbf{x} + b)}},$$

where $\mathbf{w}$ is the weight vector, b is the bias term, and $\sigma(\cdot)$ is the sigmoid activation function.

The binary cross-entropy loss for a single training instance is defined as:

$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right].$$

We update the model parameters via gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L,$$

where α is the learning rate. To understand the impact of the learning rate on convergence, we perform multiple experiments with different values, tracking the training loss over iterations.

Figure 1 shows the training loss curves for three different learning rates over 3000 iterations. A higher learning rate (0.01) converges more quickly to a lower loss, while smaller learning rates (0.001 and 0.0001) converge more slowly but steadily.

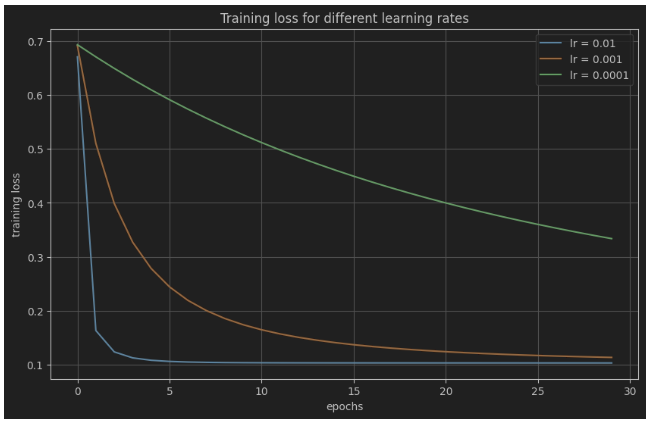

Figure 1: Training Loss for Different Learning Rates.

Validation and Ranking: The trained logistic regression model is applied to the validation set to compute prediction scores. Candidate passages are ranked per query based on these scores and evaluated using metrics mAP and mnDCG.

Table 2: Logistic Regression Performance at Cutoff = 100

Metric	Value
mAP	0.008647071369628893
mNDCG	0.02986902103150239

Table 2 shows that our logistic regression model, evaluated with a cutoff of 100, achieved an mAP of approximately 0.00865 and an mNDCG of about 0.02987. These low scores suggest that the model struggles to rank relevant passages near the top of the list. This limited performance may be due to the simplicity of the linear model, challenges in the feature representation from averaged embeddings, or the effects of subsampling in the training data. Future work could involve enhancing feature extraction or exploring more complex models to improve ranking effectiveness.

4 Task3: LambdaMART Model (LM).

In this task, we develop a LambdaMART model using XGBoost to re-rank passages based on a compact 3-dimensional feature representation. The key steps are as follows:

3D Feature Construction: For each query–passage pair, we compute a feature vector consisting of three components:

- The **dot product** between the query and passage embeddings, which captures their similarity.
- The **norm** (Euclidean length) of the query embedding.
- The **norm** of the passage embedding.

These features are combined into a single vector and assembled into a feature matrix along with the corresponding relevance labels.

Group Formation and Cross-Validation: To ensure that passages are evaluated in the context of their respective queries, we group instances by query identifiers. A custom grouping function calculates the number of passages per query, which is essential for ranking tasks. We then perform a 5-fold cross-validation, splitting the data while keeping query groups intact. In each fold, an XGBoost DMatrix is created with group information, and the model is trained with the objective set to `rank:ndcg`. The mNDCG across folds is used as the evaluation metric.

Hyperparameter Tuning: We define a grid in Table 3 over key hyperparameters including learning rate, maximum tree depth, minimum child weight, L2 regularization strength (`reg_lambda`), and subsample ratio. Each candidate parameter set is evaluated using cross-validation, and the best performing set (i.e., the one with the highest average validation NDCG) is selected. After 162 iterations.

Table 3: Hyperparameter Grid for LambdaMART Model

Parameter	Values
learning_rate	{0.01, 0.05, **0.1**}
max_depth	{3, 4, **5**}
min_child_weight	{**1**, 2}
reg_lambda	{0, **0.05**, 0.1}
subsample	{0.7, 0.85, **1.0**}

The optimal hyperparameters, as shown in Table 3 in bold, in addition resulted in a best cross-validation NDCG of **0.9280**. This high NDCG value indicates that the LambdaMART model effectively ranks the passages, placing the most relevant documents at the top. These parameters are used to train the final model on the full training set, ensuring robust re-ranking performance.

Final Model Training and Evaluation: With the optimal hyperparameters, the final LambdaMART model is trained on the complete training set. The features are standardized and queries are grouped appropriately. The trained model is then used to predict scores on the validation set. These scores are employed to rank passages for each query, and the ranking quality is measured using mAP and mNDCG.

Table 4: LambdaMART (3D Features) Final Model Performance

Metric	LambdaMART Value
mAP	0.06293
mNDCG	0.11581

Table 4 shows that the model achieves an mAP of approximately 0.06293 and an mNDCG of about 0.11581. In comparison, the logistic regression model yielded significantly lower performance (mAP around 0.00865 and mNDCG around 0.02987). This substantial improvement indicates that the non-linear learning-to-rank approach of LambdaMART is more effective at capturing the complex interactions between query and passage representations, leading to a better ranking of relevant passages.

5 Task4: Neural Network Model (NN).

In this final task, we build a neural network-based model using PyTorch to re-rank candidate passages using the same subsampled training data. Each query–passage pair is represented by the concatenation of their respective embeddings, yielding a fixed-length feature vector. The key steps are as follows:

Hyperparameter Choices: We design a feed-forward network (NNModel) with the following layers and hyperparameters:

- **Hidden Layer Sizes (256, 128, 64):** These sizes balance model capacity and generalization. Larger layers (e.g., 256 units) capture richer features, while progressively reducing the dimensionality (to 128 and then 64) compresses representations and helps avoid overfitting.
- **Dropout Rate (0.5):** A dropout probability of 0.5 is commonly used to provide strong regularization, preventing co-adaptation of neurons and reducing overfitting in fully-connected layers.
- **Learning Rate (0.0001):** This relatively low rate was chosen to ensure stable convergence when using the Adam optimizer, especially given multiple layers and dropout. Higher learning rates risk overshooting minima, while lower rates can slow convergence.
- **Weight Decay (4e-4):** Adding a small L2 penalty to the weights further regularizes the model. The value 4e-4 is typical in practice for neural networks, striking a balance between under- and over-regularization.
- **Batch Size (512):** This size offers a good compromise between computational efficiency and the stability of gradient estimates. Smaller batches can lead to noisy updates, whereas much larger batches may require more memory and slow convergence.
- **Number of Epochs (100):** This provides sufficient training iterations for the model to converge on the dataset without excessively long training times. Empirically, the loss curve in Figure 2 suggests that 100 epochs are adequate to achieve stable performance.

Dataset and DataLoader Creation: We implemented a custom dataset class (`PassageDataset`) to wrap the training feature matrix and corresponding binary labels. The dataset is then loaded via PyTorch's DataLoader with a batch size of 512 to enable efficient mini-batch training.

Training Procedure: The network is trained for 100 epochs using the Adam optimizer with a learning rate of 0.0001 and a weight decay of 4e-4. The loss function employed is the binary cross-entropy loss with logits (`BCEWithLogitsLoss`), which internally applies a sigmoid activation to produce probability estimates. The training loss is monitored and plotted over epochs to assess convergence.

As illustrated in Figure 2, the Neural Network shows a consistent downward trend in loss, decreases steadily from around 0.090 to 0.080, indicating successful convergence, demonstrating that the model effectively learns the relationship between queries and passages with each training iteration. The combination of mini-batch training, the Adam optimizer, and dropout layers helps prevent

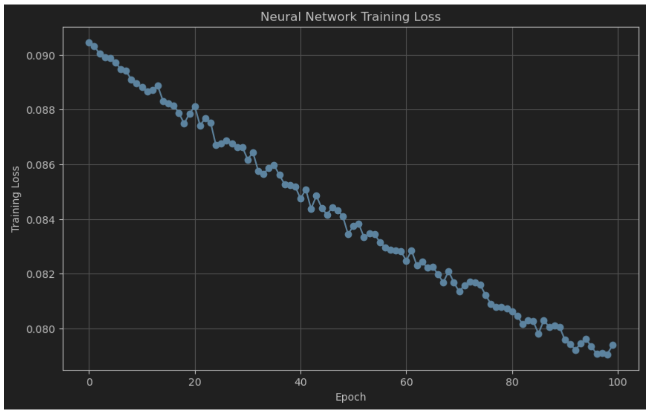

Figure 2: Neural Network Training Loss

overfitting and enables the model to capture non-linear interactions between query and passage embeddings.

Validation and Ranking: After training, the model is evaluated on the validation set. The trained network produces relevance scores (via sigmoid activation) for each query–passage pair. These scores are used to rank the passages per query, and the top-100 passages are selected to form a ranking dictionary. Table 5 shows the neural network's performance. The model achieves an mAP of approximately 0.0525 and an mNDCG of about 0.1218, indicating that it effectively ranks relevant passages higher than less relevant ones. Compared to simpler methods (e.g., logistic regression), these metrics highlight the benefit of a non-linear architecture that can better capture complex interactions between query and passage representations.

Table 5: Neural Network Model Performance on Validation Set

Metric	Value
mAP	0.05250
mNDCG	0.12180

Future enhancements could include:

- Exploring deeper or alternative architectures (e.g., CNNs, RNNs, or transformers) to capture richer contextual information.
- Incorporating pre-trained contextual embeddings (e.g., BERT) to improve input representations.
- Further hyperparameter tuning and additional regularization (e.g. batch normalization) to boost generalization.

6 Submission of Test Results

In the final step, we generate three separate submission files for the models developed in Tasks 2–4: the Logistic Regression (LR) model, the LambdaMART (LM) model, and the Neural Network (NN) model. Each file contains the re-ranked results for the test queries found in `test-queries.tsv`, based on the top-1000 candidate passages provided in `candidate_passages_top1000.tsv`.

6.1 Generated Submission Files

- **LR.txt:** Contains the re-ranked list of passages for each test query using the Logistic Regression model.
- **LM.txt:** Contains the re-ranked list of passages for each test query using the LambdaMART model.
- **NN.txt:** Contains the re-ranked list of passages for each test query using the Neural Network model.

Each file has 19290 rows with format `<qid1 A2 pid1 rank1 score1 algoname2>` ranks up to 100 passages per query, ensuring that the best-scoring passages are placed at the top of the list. These three files constitute the final submission for Tasks 2–4.