COMP0084 Information Retrieval and Data Mining Coursework 2

Abstract

This project investigates a range of passage reranking techniques in information retrieval. Starting with a classical BM25 model as a baseline, we progressively explore more advanced models, including a logistic regression classifier trained on averaged GloVe Embeddings, a LambdaMART model using gradient boosting with features, and two neural network architectures - a feedforward multilayer perceptron and a Siamese bidirectional LSTM. Each model is evaluated using Mean Average Precision (mAP) and Normalised Discounted Cumulative Gain (mNDCG) on a validation set.

1 Introduction

This project is structured into four subtasks, implementing and evaluating three supervised re-ranking models: Logistic Regression (LR), LambdaMART (LM), and a Neural Network (NN). Each model is trained using the labelled dataset provided in train_data.tsv, and their performance is measured on validation_data.tsv using mAP and mNDCG.

In addition to validation, all three models are used to re-rank passages for the test queries contained in test-queries.tsv, with initial candidates drawn from candidate_passages_top1000.tsv. The top 100 ranked passages per query are compiled into output files named LR.txt, LM.txt, and NN.txt respectively, corresponding to the outputs of each model.

2 Task 1 Evaluating Retrieval Quality with BM25

Our BM25 approach first preprocesses the text by lowercasing, tokenizing, and applying stemming. An inverted index is then built from the processed passages to allow efficient computation of BM25 scores for each (query, passage) pair. The BM25 formula (with parameters k_1 = 1.2, k_2 = 100, and b = 0.75) is applied to produce a ranked list of passages per query. The scores take into account term frequency in both the query and passage and the average document length, discounting lower-ranked terms logarithmically.

In the following passage, average precision (AP) and discounted cumulative gain (DCG) metrics are implemented to compute the performance of BM25 on the validation data.

2.1 Mean Average Precision (MAP)

For an individual query, AP is calculated by measuring the precision at each rank position where a relevant document appears and then averaging these precision values. This means that instead of looking at just one threshold, AP considers the entire ranked list. It mimics a user's behaviour who is interested in retrieving all relevant documents.

Essentially, it approximates the area under the un-interpolated precision-recall curve for that query. It gives more weight to relevant documents that appear at higher ranks. The formula is shown below:

$$AP(q_j) = \frac{1}{m_j} \sum_{k=1}^{m_j} Precision@k * Relevance@k$$

MAP is the mean of the AP values over a set of queries, therefore, roughly the average area under the precision-recall curve for a set of queries. It provides a single-figure measure of quality across recall levels and good discrimination power and stability, making it one of the most popular metrics in offline evaluation.

$$MAP(Q) = \frac{1}{|Q|} \sum_{|Q|}^{j=1} AP(q_j)$$

One limitation of MAP is that test queries must be large and diverse enough to be representative of system effectiveness across different queries. In our case, the dataset contains 4,590 unique queries, which is large and diverse enough to ensure that the computed MAP is both robust and reliable.

2.2 Normalised Discounted Cumulative Gain (NDCG)

NDCG is a metric that evaluates the ranking quality by considering both the relevance and the position of the documents. The Discounted Cumulative Gain (DCG) for a query is calculated as:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

where rel_i is the relevance score at rank i. Since DCG values vary across queries, the Ideal DCG (IDCG) is computed for a perfect ranking, and NDCG is defined as:

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

This normalization ensures that NDCG values range between 0 and 1, facilitating meaningful comparisons across queries.

2.3 BM25 Performance

The BM25 model was evaluated on the validation set with the following results:

mAP	mNDCG
0.24197	0.38409

Table 1: Performance of BM25 on Validation Data

These results indicate that the BM25 model achieves a moderate level of effectiveness, with a MAP of approximately 24% and an NDCG of approximately 38%. While these metrics demonstrate the baseline quality of the BM25 approach, they also highlight that there is room for improvement, particularly in the ranking quality as reflected by NDCG.

3 Task 2 Logistic Regression

3.1 Model Introduction

Logistic Regression (LR) is a classical model used for binary classification tasks. At its core, logistic regression models the probability that an input belongs to a given class by applying the sigmoid function to a linear combination of input features.

Mathematically, given an input feature vector $x \in \mathbb{R}^n$ and parameters $\theta \in \mathbb{R}^n$, the hypothesis function is defined as:

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

This output $h_\theta(x)$ represents the probability that x belongs to the positive class (or the relevant passage to the query in our application).

The training process involves optimising the model parameters using a loss function. For logistic regression, binary cross-entropy (log loss) is used:

$$\mathcal{L}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log h_\theta(x^{(i)}) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta(x^{(i)})\right)\right]$$

Gradient descent is then applied to update the weights iteratively:

$$\theta := \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta)$$

where η is the learning rate, and the gradient is computed as:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

This formulation lays the foundation for our implementation that is applied to determine the relevance of a passage to a query.

3.2 Data Preparation

For our task, textual data undergoes an extensive preprocessing and feature representation pipeline. Initially, both the query and the passage text are converted to lowercase and processed to remove any non-alphabetic characters through regular expressions. Using NLTK's tokenisation tools, these cleaned texts are split into individual tokens. To ensure that only informative words contribute to the final representation, common stopwords are removed from the token lists.

Word Embedding. Once the text has been tokenised and cleaned, each remaining word is mapped to its corresponding vector from a pre-trained GloVe embedding model (glove.6B.200d). The word vectors for a given query or passage are then averaged to yield a single vector representation.

Concatenation and Bias Addition. The next step involves combining these representations. The average embedding for the query is concatenated with that of the passage, and a constant bias term (typically 1) is prepended to the resulting vector. This process transforms each query–passage pair into a numerical feature vector suitable for input to our logistic regression model.

Downsampling. A critical aspect of our preprocessing is the downsampling of training data to address the severe imbalance between positive and negative examples. Preliminary data exploration revealed a dramatic imbalance – there were over 4.3 million negative records compared to roughly 4,800 positive ones. Therefore, for each query, the algorithm first retains all positive examples, then samples up to a fixed number of negatives. In our current setting, the parameter max_negatives is set to 20. This means that for a given query with P positive examples, the maximum number of negatives sampled is computed as $\max\left(0, \text{max\_negatives} - P\right)$.

3.3 LR Implementation

We initialize the parameter vector θ to zeros and then employ mini-batch gradient descent to optimize the parameters. At each epoch, the data is shuffled, and the model processes it in batches. For each mini-batch, the dot product X_batch θ is calculated, and the sigmoid function is applied to obtain predictions. The model then computes the binary cross-entropy loss and updates θ by subtracting the product of the learning rate and the gradient. The training process includes an early stopping mechanism that halts training when the relative improvement in loss between epochs falls below a threshold, ensuring that the model does not overfit. Multiple learning rates are tested (ranging from 1×10^{-6} to 1×10^{-3}) to analyze their effects on the convergence speed and stability of training. The training loss curves for each learning rate are plotted in Fig 1.

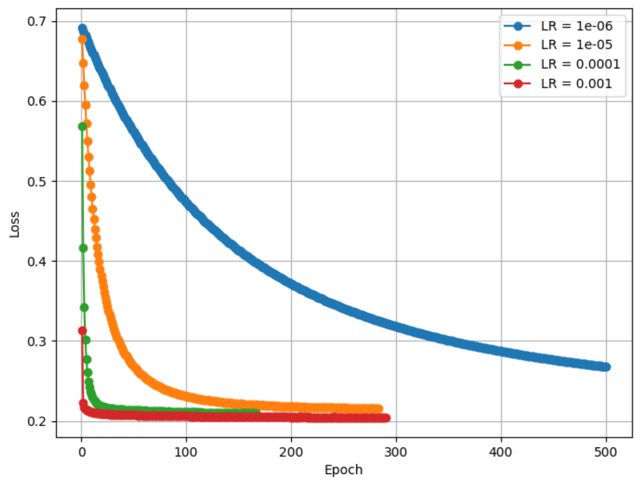

Figure 1: Training Loss vs Epoch for different learning rates

The experimental results reveal a clear relationship between the learning rate and the training dynamics. In our tirals, a very small learning rate of 1×10^{-6} yielded a gradual decrease but suffered from very slow convergence. This indicates that though it avoids overshooting and ensures stability, it still limits the model's ability to traverse the loss efficiently within a fixed epoch (n=500). In contrast, learning rates of 1×10^{-5} and 1×10^{-4} achieved a more pronounced reduction during the early stages. Notably, the learning rate of1×10^{-3} demonstrated the fastest initial decrease in loss.

A further learning rates comparison was carried out:

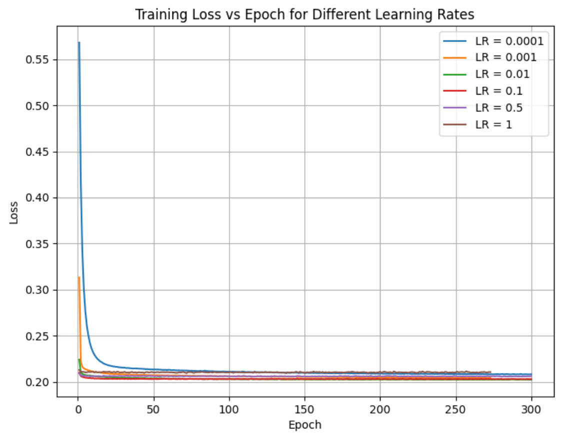

Figure 2: Training Loss vs Epoch for different learning rates

3.4 LR Performance

The BM25 model was evaluated on the validation set with the following results:

mAP	**mNDCG**
0.0120	0.1309

Table 2: Performance of BM25 on Validation Data

4 Task 3 LambdaMART Model

This section presents a ranking model based on XGBoost's LambdaMART algorithm. The model is designed to re-rank candidate passages for each query by learning a ranking function from training data. In this implementation, we leverage both semantic representations through Glove embeddings (providing a robust 200-dimensional representation) and four additional features that capture surface-level similarities between queries and passages. The additional features include query length, passage length, word overlap count, and Jaccard similarity.

4.1 Feature Extraction and LM Implementation

The data preprocessing pipeline consists of the following steps:

Text Preprocessing and Embedding. Queries and passages are tokenized using NLTK. Stopwords are removed to focus on informative terms. Average Glove embeddings for both the query and the passage are computed. Additionally, the element-wise difference between these embeddings is also calculated.

Additional Feature Extraction. Four extra features are computed:

(1) **Query Length:** Count of non-stopword tokens in the query.
(2) **Passage Length:** Count of non-stopword tokens in the passage.
(3) **Word Overlap Count:** Number of common tokens between the query and passage.
(4) **Jaccard Similarity:** Ratio of the intersection to the union of the token sets from the query and passage.

Feature Transformation. The final feature vector for each query–passage pair is obtained by concatenating the embedding-based features (query embedding, passage embedding, and their difference) with the four additional scalar features. This combined representation is then used as input for the ranking model.

Downsampling. The training data is also pre-processed by downsampling negatives (with `MAX_NEGATIVES = 20`) likewise and the observations are grouped by query identifiers to align with the requirements of the XGBRanker (LambdaMART) model.

4.2 Hyper-Parameter Tuning

For optimizing the performance of the LambdaMART model, we employed a grid search strategy over a set of hyperparameters:

Hyperparameters	Values
Learning Rate	0.01, 0.1, 0.3
Maximum Depth	3, 5, 7
Number of Estimators	100, 300, 500

Table 3: Grid Search of Hyperparameters

The model objective was set to `rank:pairwise` to adopt a pairwise ranking approach. For each hyper-parameter combination, the model was trained on the training set and evaluated on the validation set using mAP and mNDCG as performance metrics.

4.3 LM Performance

The performances of different hyper-parameter configurations before and after adding the features were compared, and the top five results based on validation mAP are presented in Table 4. The best configuration achieved an mAP of 0.0552 and an mNDCG of 0.1908 before feature extractions.

Learning Rate	Max Depth	Estimators	**mAP**	**mNDCG**
0.3	3	500	0.0552	0.1908
0.3	7	500	0.0514	0.1863
0.3	5	500	0.0506	0.1847
0.1	5	500	0.0489	0.1840
0.1	3	500	0.0491	0.1842

Table 4: Top-performing configurations without features

Performance Improvement. Additional four features are computed and the top five results are presented in Table 5. The best configuration achieved an mAP of 0.2072 and an mNDCG of 0.3439 before feature extractions.

Learning Rate	Max Depth	Estimators	**mAP**	**mNDCG**
0.1	7	300	0.2072	0.3439
0.1	7	500	0.2028	0.3406
0.1	5	500	0.1998	0.3377
0.1	5	300	0.1985	0.3360
0.1	3	500	0.1963	0.3345

Table 5: Top-performing configurations wit features

Overall, these experiments demonstrate the potential of the LambdaMART model when enhanced with additional query–passage interaction features. Future work may focus on refining feature selection, exploring alternate ranking objectives, and integrating more advanced neural representations to further boost performance.

5 Task 4 Neural Network Model

In this task, two distinct neural architectures were implemented and evaluated: a feedforward multilayer perceptron (MLP) and a bidirectional LSTM (BiLSTM) in a Siamese setup. Both models were developed using PyTorch and trained on preprocessed query-passage pairs with GloVe embeddings as input features. The purpose of this dual-architecture setup was to compare the effectiveness of shallow vs. sequential models in a document re-ranking setting.

5.1 Feedforward Neural Network (MLP)

The first model is a straightforward feedforward neural network trained on concatenated average word embeddings of the query and the passage. We begin by extracting a fixed-size vector representation of each query–passage pair.

(1) Use pretrained GloVe embeddings (200-dimensional) to represent each query and passage.
(2) Concatenate the averaged query embedding, averaged passage embedding, and a simple overlap feature (or any additional scalar feature) to produce a single input vector of dimension $2 \times 200 + 1 = 401$.
(3) Feed this 401-dimensional vector into a stack of fully connected layers:

- Layer 1: Linear$(401 \longrightarrow 128)$ with ReLU activation.
- Layer 2: Linear$(128 \longrightarrow 64)$ with ReLU activation.
- Output Layer: Linear$(64 \longrightarrow 1)$ with Sigmoid activation.

The use of ReLU is straightforward, offering computational simplicity and avoiding vanishing gradients. The final Sigmoid outputs a value in (0,1), which we interpret as the predicted probability that the passage is relevant to the query. We train the network using binary cross-entropy (BCELoss), with an Adam optimizer at a learning rate of 0.001.

Why it's appropriate? As illustrated in Figure 3, this architecture clearly delineates the sequential flow—from the initial 401-dimensional input through successive nonlinear transformations to the final classification output. The simplicity and rapid training of this MLP make it a suitable baseline, especially when dealing with fixed-size, low-dimensional inputs such as averaged embeddings. Its structure allows rapid experimentation and provides a useful performance reference when evaluating deeper models.

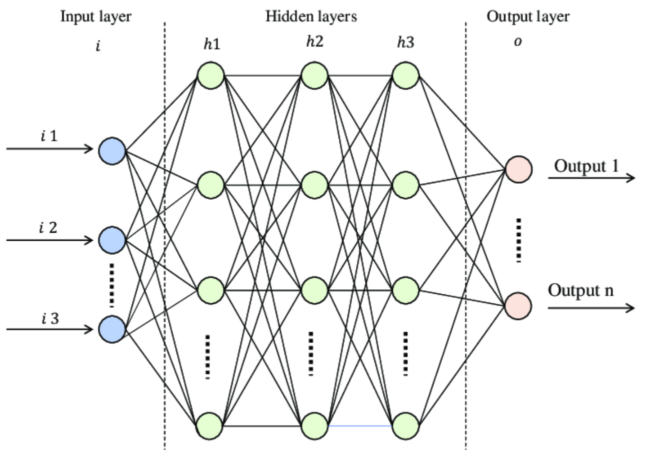

Figure 3: Architecture Diagram of a FNN model [1].

5.2 Recurrent Neural Network (Siamese BiLSTM)

The second model adopts a Siamese architecture with a shared bidirectional LSTM encoder. Instead of averaging the embeddings, each query and passage is treated as a sequence of word embeddings. The BiLSTM processes these sequences to capture contextual and sequential dependencies. The encoded query and passage representations are then compared using absolute difference and elementwise product, before being passed to a classifier network.

As shown in Figure 4, this recurrent architecture is unrolled over time, where each input x_t updates the hidden state h_t, which then produces an output y_t, capturing sequential dependencies across the entire sequence.

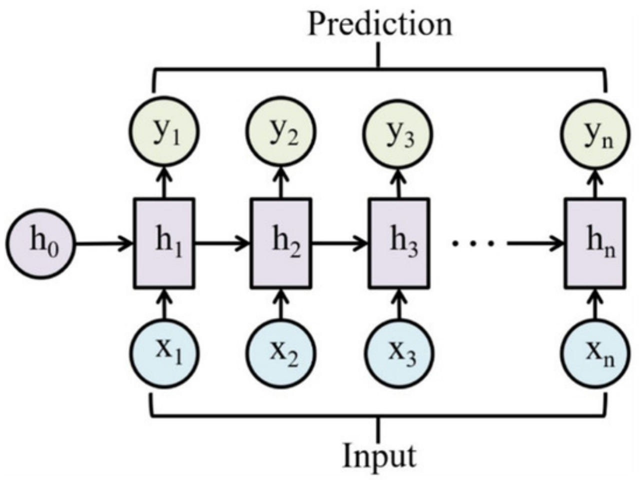

Figure 4: Architecture Diagram of a RNN model [2].

Shared BiLSTM Encoder. We pass each tokenised query (up to a fixed length) through a bidirectional LSTM with a hidden size of 128. This yields a 2×128-dimensional final hidden state per query.

The same shared BiLSTM encoder is used for passages, but with a larger maximum length (e.g., up to 100 tokens). This encoding step helps capture word-order information, context, and phrase-level interactions.

Interaction Features. Once we have the final vector for the query (q_{repr}) and for the passage (p_{repr}), each of size $[\text{batch}, 2 \times 128]$, we compute:

- Absolute Difference $|q_{\text{repr}} - p_{\text{repr}}|$
- Element-wise Product $(q_{\text{repr}} \times p_{\text{repr}})$

Classifier. We concatenate $\left[q_{\text{repr}}, p_{\text{repr}}, |q_{\text{repr}} - p_{\text{repr}}|, q_{\text{repr}} \times p_{\text{repr}}\right]$, resulting in a $[\text{batch}, 8 \times 128]$ tensor. This combined vector is fed into a fully connected classifier:

- $\text{Linear}(8 \times 128 \longrightarrow 128)$, ReLU, Dropout(0.3)
- $\text{Linear}(128 \longrightarrow 1)$, Sigmoid

Similar to the FNN, the output of the Sigmoid is interpreted as a probability of relevance, and we train with BCELoss using the Adam optimizer.

This model is significantly more expressive than the MLP. It retains word order, allowing it to distinguish between phrases like "not relevant" and "relevant". The BiLSTM is also able to model long-range dependencies that are crucial for understanding how a passage answers a query.

Why it's appropriate? The BiLSTM was chosen because ranking tasks often depend on sequence-level understanding. Unlike MLP, which sees only global similarity, the BiLSTM can learn how meaning unfolds over word sequences. It is particularly well-suited for reranking tasks where phrasing and position may change the intent.

5.3 NN Performance

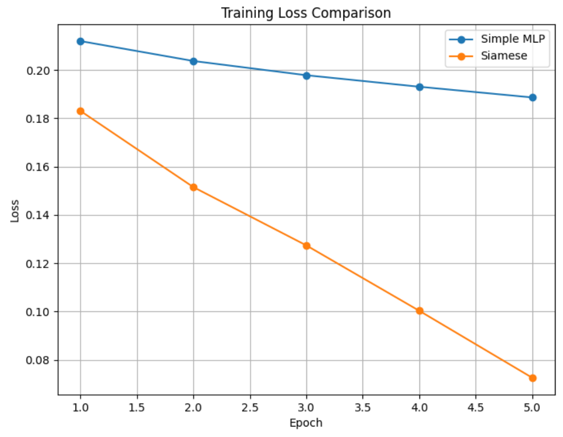

Figure 5: FNN vs RNN

To better understand the learning dynamics of the two neural architectures, we compared their training loss curves across five epochs (Figure 5). The Simple MLP showed a slow and steady decline in training loss, reducing from 0.212 to 0.189 over the five training epochs. In contrast, the Siamese BiLSTM exhibited a significantly faster convergence, with its training loss dropping from 0.183 to just 0.073. This rapid decline suggests that the Siamese model was able to more effectively capture meaningful patterns from the data in fewer epochs.

The final validation results support this observation. While the MLP achieved a modest mAP of 0.0342 and NDCG of 0.1661, the Siamese BiLSTM delivered markedly stronger performance, reaching 0.1488 mAP and 0.2867 NDCG.

Model	mAP	mNDCG
FNN	0.0342	0.1661
RNN	0.1488	0.2867

Table 6: Performance of Neural Networks on Validation Data

The combination of faster convergence and higher retrieval effectiveness demonstrates that the BiLSTM's ability to model sequence-level dependencies and contextual interactions plays a crucial role in improving ranking quality.

References

[1] S. M. Bhagya P. Samarakoon, M. A. Viraj J. Muthugala, Anh Vu Le, and Mohan Rajesh Elara. htetro-infi: A reconfigurable floor cleaning robot with infinite morphologies. *IEEE Access*, 8:69816–69828, 2020.

[2] Ibomoiye Domor Mienye, Theo G. Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9), 2024.