COMP0084 Information Retrieval and Data Mining Coursework 2 (2024/25)

Abstract

This report presents an evaluation of four methods for assessing query relevance to passages in the Information Retrieval and Data Mining Coursework 2. The first approach examines BM25's effectiveness using mean Average Precision (mAP) and mean Normalized Discounted Cumulative Gain (mNDCG), evaluated on the validation set. Four additional models, Logistic Regression, LambdaMART, and two Neural Network models, were trained on the training set and evaluated on the validation set using the same metrics. The best-performing model based on mAP and mNDCG was the BM25 model. However, as an extension, a transformer model (*distilbert-base-uncased*) was explored and returned an mAP of 0.90 and mNDCG of 0.93 which outperformed BM25.

1 Datasets

This coursework used 4 different datasets. The first two, *train_data.tsv* and *validation_data.tsv*, were used for training and evaluating the logistic regression, LambdaMART, and neural network models, and *validation_data.tsv* was used to evaluate the BM25 model.

In order to train the non-BM25 models and evaluate the BM25 model, both of these datasets contained 5 columns: qid (query ID), pid (passage ID), query, passage, and relevancy. The query and passage columns consist of textual input. The relevancy column consists of either 1 or 0, denoting whether the passage was relevant to the query or not.

Two other datasets are also utilised: *candidate_passages_top1000.tsv* and *test_queries.tsv*. These are used for the three non-BM25 models post training and evaluation. Each query in *test_queries.tsv* corresponds to a certain number of passages in *candidate_passages_top1000.tsv* and each of the passages of each query is ranked in terms of its corresponding relevancy score from each of the non-BM25 models.

2 Dataset Processing and Sampling

Input Processing. To standardise query and passage processing across all models bar the extension model, the NLTK *word_tokenize* function was used [2]. This applies the *PunktSentenceTokenizer*, which splits input into sentences — preserving abbreviations, capitalisation, and punctuation patterns (e.g., "Dr. Smith is here. He arrived at 3 p.m." becomes two sentences, without splitting "Dr." or "p.m."). *word_tokenize* is then applied to each sentence to produce individual word and punctuation tokens.

For the extension transformer architecture model, the *AutoTokenizer* class from HuggingFace's Transformers library was used [9]. This tokenizer is model-specific and automatically loads the appropriate tokenisation scheme for the given transformer architecture. In both cases, it applies WordPiece tokenisation to the concatenated query-passage pair, splits the input into subword units, and assigns each token a unique integer ID (*input_ids*). It also constructs an *attention_mask*, which is integral for distinguishing real tokens from padding. The AutoTokenizer ensures that the input formatting (e.g., special tokens like [CLS] and [SEP], padding strategy, truncation, and vocabulary alignment) exactly matches the pretraining configuration of the transformer model, which is essential for utilising its learned contextual representations.

In addition to tokenisation, lemmatisation was applied using NTLK's *WordNetLemmatizer* [1] in all models except the transformer neural network model. Lemmatisation reduces each word to base form (e.g. "running" to "run", "better" to "good"), which helps reduce vocabulary size and increases the chance of matching semantically similar words across queries and passages in the upcoming models. This will have the ultimate effect of increasing the mAP and mNDCG performance of each retrieval model considered.

Sampling. Negative sampling [7] was used to address computational constraints and class imbalance. The training dataset contained 4364339 query-passage pairs, while the validation set had 1103039, making full-scale training expensive. Only 4797 training and 1208 validation pairs had a relevancy score of 1, resulting in an overwhelming number of non-relevant (score 0) pairs, which degraded model performance.

To mitigate this, all relevant pairs were retained, and ten times as many non-relevant pairs were randomly sampled. A fixed seed ensured consistency across models. This resulted in 4797 training pairs with a relevancy score of 1 and 47970 with a score of 0, and 1208 validation pairs with a relevancy score of 1 and 12080 with a score of 0. This signifcantly reduced computational cost.

3 Task 1: Evaluating BM25 using mAP and mNDCG

This section evaluates the BM25 ranking algorithm on the validation set using Mean Average Precision (mAP) [6] and Mean Normalized Discounted Cumulative Gain (mNDCG) [5], which assess how well relevant passages are ranked. These metrics will also be used to evaluate the non-BM25 models. The BM25 ranking model scores passages using a TF-IDF-based scheme that adjusts for document length. The implementation is the same as in coursework 1.

3.1 Mean Average Precision (mAP)

The first metric to convey how well BM25 ranks passages is the Average Precision (AP). AP measures the quality of ranked retrieval by computing the precision at each rank where a relevant document appears and averaging over the total number of relevant documents:

$$AP = \frac{\sum_{k=1}^{n} P(k) \times \text{rel}(k)}{N} \tag{1}$$

Where $\text{rel}(k)$ is 1 if the document at rank k is relevant, otherwise 0, N is the total number of relevant documents for the query, n is the total number of retrieved documents, and $P(k)$ is the precision at rank k, defined as:

$$P(k) = \frac{\#\text{ of relevant documents retrieved up to rank } k}{k} \tag{2}$$

Then, to calculate mAP, the mean of all the average precisions calculated for each query is calculated:

$$\text{mAP} = \frac{1}{Q}\sum_{q=1}^{Q} AP_q \tag{3}$$

Where Q is the total number of queries, and AP_q is the AP for the query q.

3.2 Mean Normalised Discounted Cumulative Gain

Mean Normalized Discounted Cumulative Gain (mNDCG) evaluates ranking effectiveness by giving more weight to relevant documents appearing earlier in the list [5]. Although it supports graded relevance, binary labels are used in this context. To define mNDCG, we begin with Discounted Cumulative Gain (DCG):

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \tag{4}$$

Where rel_i is the relevancy score of the passage at rank i, k is the number of passages considered for each query (set to 10), and the denominator, $\log_2(i+1)$ discounts the gain by the rank of the passage, i.

To then calculate the Normalised Discounted Cumulative Gain (NDCG) for each query, the NDCG is:

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k} \tag{5}$$

Where IDCG@k is ideal DCG at k - the DCG computed with the best possible ranking order.

Lastly, mNDCG is:

$$\text{mNDCG@}k = \frac{1}{Q}\sum_{q=1}^{Q} \text{NDCG@}k^{(q)} \tag{6}$$

Where Q is the total number of queries, and NDCG@$k^{(q)}$ is the NDCG@k score for query q.

3.3 BM25 Performance on the Validation Set

mAP	mNDCG
0.80	0.85

Above is the numerical performance of the BM25 model. Analysing these specific values using the equations previously defined, we begin with the mAP score. A value of 0.80 indicates that BM25 ranks relevant documents highly for most queries in the validation set. Since mAP rewards both the presence and the early ranking of relevant documents, such a high score implies that relevant passages are consistently being placed near the top of the returned list across many queries.

Similarly, the mNDCG@10 score (k will be set to 10 throughout) of 0.85 suggests that not only are relevant documents retrieved, but they are also ranked close to their ideal positions. Given that NDCG places greater emphasis on retrieving relevant documents earlier in the ranking, a value of 0.85 is near-optimal. A perfect score of 1.0 would mean that all relevant documents are ranked at the very top. Thus, BM25 appears to be doing an excellent job at ranking relevant passages prominently within the top-10 results.

Compared to benchmarks on standard datasets like MS MARCO and TREC, where BM25 typically achieves mAP scores in the range of 0.2–0.35 and NDCG scores around 0.3–0.4 [4, 10], the performance observed here is notably higher. This suggests that either the validation set is particularly well-suited to BM25's retrieval assumptions, or that the dataset's construction (e.g., the use of 10:1 negative sampling) may have made the task more favorable for a lexical model like BM25. Nonetheless, whether BM25 is ultimately the best model for this retrieval task can only be determined through comparison with the learning-to-rank models analysed in the upcoming sections.

4 Task 2: Logistic Regression

4.1 Methodology

Input Representation. In order to train the logistic regression model, each query and passage in the training data must be represented correctly. Each input sample is processed as described in the start of the paper. Each input sample consists of a query q and a passage p, both of which are converted into vector representations using pre-trained GloVe word embeddings[1] of dimension $d = 300$. Given a query q containing N_q words and a passage p containing N_p words, each word w_i is mapped to a corresponding 300-dimensional embedding $\mathbf{e}_i \in \mathbb{R}^{300}$ from the GloVe embedding matrix. To obtain fixed-length vector representations for queries and passages, the embeddings of all words in the text are averaged:

$$\mathbf{q} = \frac{1}{N_q}\sum_{i=1}^{N_q} \mathbf{e}_i, \quad \mathbf{p} = \frac{1}{N_p}\sum_{j=1}^{N_p} \mathbf{e}_j. \tag{7}$$

This results in two fixed-dimensional vectors, $\mathbf{q}$ for the query and $\mathbf{p}$ for the passage, each of size 300. The final input feature vector $\mathbf{x}$ is obtained by concatenating these representations:

$$\mathbf{x} = [\mathbf{q} \,||\, \mathbf{p}] \in \mathbb{R}^{600}. \tag{8}$$

Where the symbol || represents concatenation of two vectors.

This 600-dimensional input vector is then used as input to the logistic regression model, which predicts the relevance of the passage to the query. The justification of the use of GloVe embeddings stems from the assumption that semantically similar words have similar vector representations, aiding in effective ranking of passages.

Logistic Regression Model. Now that each query and passage has been converted into the necessary feature vector to be fed into the model, the model itself works as follows. The model is a logistic regression classifier, where the relevance of a passage to a given query is modeled as a probability. Given a query embedding $\mathbf{q} \in \mathbb{R}^{300}$ and a passage embedding $\mathbf{p} \in \mathbb{R}^{300}$ obtained from the GloVe embedding model, the input to the logistic regression model is the concatenated feature vector $\mathbf{x} = [\mathbf{q}||\mathbf{p}] \in \mathbb{R}^{600}$. The model consists of a single linear layer with weights $\mathbf{w} \in \mathbb{R}^{600}$ and a bias term b, followed by a sigmoid activation function:

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + b), \tag{9}$$

[1] https://nlp.stanford.edu/projects/glove/

Where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function that maps the output to a probability score in the range $[0, 1]$.

The model is trained using a weighted Binary Cross-Entropy (BCE) loss to account for class imbalance, ensuring that relevant passages (positive class) are given appropriate emphasis in the loss function. The trained model produces a ranking score for each query-passage pair, which is then used to order passages in descending order of predicted relevance.

Weighted Binary Cross-Entropy Loss. Although negative sampling is used to address the class imbalance between the relevant and non-relevant passages for each query, there is still a class imbalance of 10 non-relevant passages for every relevant passage. Consequently, a weighted version of the Binary Cross-Entropy (BCE) loss function is employed instead of the Standard BCE. The Standard BCE loss is defined as:

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)] \tag{10}$$

Where y_i is the ground truth relevancy label (0 or 1), and $\hat{y}_i$ is the model's predicted probability for the positive class. However, in imbalanced datasets, the model may become biased towards the majority class, leading to suboptimal performance. To mitigate this, class-specific weights are introduced, w_0 and w_1, which adjust the loss contribution of each class:

$$\mathcal{L}_{WBCE} = -\frac{1}{N}\sum_{i=1}^{N}[w_1 y_i \log \hat{y}_i + w_0(1-y_i)\log(1-\hat{y}_i)] \tag{11}$$

Where w_1 and w_0 are computed as:

$$w_1 = \frac{N}{N_1}, \quad w_0 = \frac{N}{N_0} \tag{12}$$

Here, N_1 and N_0 denote the number of positive and negative samples, respectively. This ensures that the contribution of both classes is balanced, preventing the model from favoring the majority class. The weighted BCE loss approach improves the model's ability to learn meaningful decision boundaries despite the imbalance in the training data compared to the standard BCE.

Training. The model training is performed using *stochastic gradient descent* (SGD) to minimize the WBCE loss. Given a dataset of N training samples, where each sample consists of an input feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and a binary label $y_i \in \{0, 1\}$, the model computes a predicted probability using the sigmoid function:

$$\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i + b) = \frac{1}{1+e^{-(\mathbf{w}^\top \mathbf{x}_i + b)}} \tag{13}$$

Where $\mathbf{w} \in \mathbb{R}^d$ is the weight vector and b is the bias term. The loss function used is the weighted binary cross-entropy (11).

During training, we update $\mathbf{w}$ and b iteratively using mini-batch stochastic gradient descent. For each mini-batch of size m (set to 32), the gradient of the loss with respect to the model parameters is computed:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}_j - y_j)\mathbf{x}_j, \quad \frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}_j - y_j) \tag{14}$$

The parameters are then updated using the learning rate η:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\frac{\partial \mathcal{L}}{\partial \mathbf{w}}, \quad b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b} \tag{15}$$

This process continues iteratively for a specified number of epochs, ensuring that the model gradually learns a weight vector that minimizes classification error while handling imbalanced classes effectively.

Ranking. Once the weights and bias term have been learnt during the training process, the model is assessed on the validation dataset to assess its performance. Given a query q_i with corresponding candidate passages $\{p_1, p_2, \ldots, p_k\}$, the input feature vector is formed by concatenating the query and passage embeddings just as in the training process. The model now computes a relevance score using the weight vector $\mathbf{w}$ and bias b:

$$r_{ij} = \sigma(\mathbf{w}^\top [\mathbf{x}_i, \mathbf{x}_j] + b) \tag{16}$$

These scores are then used to rank the passages for a given query in descending order. Now, just as in the BM25 task, these rankings are evaluated using mAP (3) and mNDCG (6). These measure how well the ranked passages align with the ground-truth relevancy labels from the validation dataset.

Hyper-parameter Tuning. The following plot displays that a learning rate of 0.001 reaches the minimum training loss out of all the tested learning rates. While taking longer to converge compared to a 0.01 learning rate, it ultimately converges around 700 epochs. Due to compute constraints and wanting to reduce overfitting, a learning rate of 0.001 will be used and trained for 250 epochs.

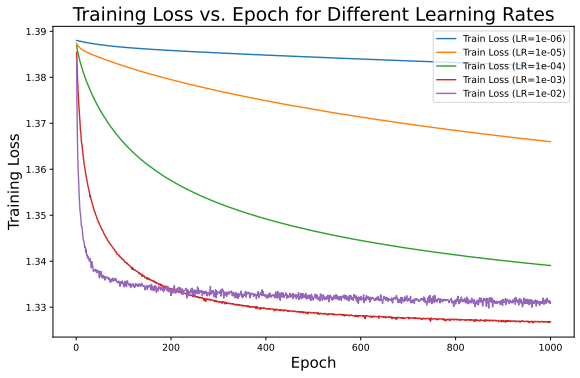

Figure 1: Training loss over epochs for various learning rates.

4.2 Logistic Regression Performance on the Validation Set

mAP	mNDCG
0.35	0.48

An mAP value of 0.35 and an mNDCG of 0.48 indicate that the Logistic Regression model performs significantly worse than the

BM25 baseline (mAP = 0.80, mNDCG = 0.85). This suggests that, despite being a supervised model, Logistic Regression is less effective at ranking relevant passages highly compared to the unsupervised BM25 algorithm. This raises the question: to what extent is this due to the model's simplicity as a learning-to-rank model and will LambdaMART and the neural network models perform better? Or, is it due to the strength of the BM25 algorithm in this specific instance when negative sampling has been used? Interestingly, further analysis shows that for logistic regression, as the negative ratio increases, both the mAP and mNDCG metrics degrade further:

Negative Ratio	mAP	mNDCG
15:1	0.28	0.36
20:1	0.24	0.31
30:1	0.18	0.29

The table above illustrates the effect of the negative sampling ratio on the performance of the logistic regression model in terms of mean Average Precision (mAP) and mean Normalized Discounted Cumulative Gain (mNDCG). As the ratio of negative to positive samples increases, we observe a consistent decline in both mAP and mNDCG scores. can be attributed to the increasing class imbalance introduced by higher negative ratios. While this is an unavoidable consequence of training on this large training set, it must be noted, if these learning-to-rank models were to be trained on the full training dataset, the results of mAP and mNDCG would be much lower.

5 Task 3: LambdaMART

5.1 Methodology

LambdaMART is a gradient-boosted tree-based learning-to-rank model that optimises the NDCG metric using pairwise preference learning [3]. The implementation follows these steps.

Feature Extraction. Each query-passage pair is converted into a vector $\mathbf{x} \in \mathbb{R}^{300}$ using the glove embeddings, as previously explained. Features are then obtained for the LambdaMART model, resulting in a feature vector $\mathbf{x} \in \mathbb{R}^{303}$ which capture similarity features between the query and passage vectors. First, cosine similarity:

$$S_{\cos}(q, d) = \frac{q \cdot d}{\|q\|\|d\|} \tag{17}$$

Second, Euclidean distance:

$$S_E(q, d) = \sqrt{\sum (q_i - d_i)^2} \tag{18}$$

Third, the dot product:

$$S_{\text{dot}}(q, d) = q \cdot d \tag{19}$$

The remaining 300 dimensions are due to the element-wise absolute difference:

$$|q - d| = [|q_1 - d_1|, |q_2 - d_2|, \ldots, |q_{300} - d_{300}|] \tag{20}$$

Learning to Rank. LambdaMART is designed to optimise the NDCG (5). LambdaMART uses pairwise gradients called 'Lambdas' to guide learning. These lambda gradients help the model learn to correctly order passages for each query by minimising the expected NDCG loss.

Model Training and Optimisation. The hyper-parameters used in the training of LambdaMART are as follows:

- **objective**: Specifies the learning task - optimising NDCG.
- **learning_rate**: This controls the step size shrinkage used in each boosting step.
- **max_depth**: The maximum depth of each decision tree.
- **n_estimators**: Specifies the total number of trees.
- **lambda**: L2 regularization term on leaf weights.
- **eval_metric**: The evaluation metric used during training: NDCG.
- **early_stopping_rounds**: Stops training if the model's performance on the validation set does not improve for a specified number of rounds, helping prevent overfitting.

While the hyper-parameters, objective, evaluation metric, and early stopping rounds (set to 10) were kept constant, the following hyper-parameters were varied. The best hyper-parameter combination are represented in bold.

Hyper-Parameters	Tested Values
Learning Rate	0.01, 0.05, **0.1**, 0.2
Max Depth	**4**, 6, 8
N Estimators	**100**, 300, 500
Lambda (L2 Regularisation)	**0**, 0.1, 1

5.2 LambdaMART Performance on the Validation Set

mAP	mNDCG
0.56	0.65

The LambdaMART model achieved an mAP of 0.56 and a mNDCG of 0.65 on the validation set. These values reflect a considerable improvement over the logistic regression problem, indicating that LambdaMART is more effective as a learning-to-rank model than logistic regression. This is to be expected; LambdaMART directly optimises ranking-based loss functions using gradient-boosted decision trees, enabling it to model feature interactions that logistic regression might not be able to capture.

However, LamdbaMART performs poorly against BM25. While this is not due to insufficient hyperparameter tuning as this has been addressed to a considerable extent, the lack of performance compared to BM25 could be contributed to suboptimal feature engineering - LambdaMART's success as a model is heavily dependent on the quality of input features. While the 4 features I have included outperform the first learning-to-rank model, logistic regression, more features may have to be considered to match the BM25 performance.

6 Task 4: Neural Network

In this section, two models are considered. First, a *Two-Tower* design is implemented, which encodes queries and passages independently, followed by the concatenation of the same training data representation in task 3, followed by a scoring function based on these learned similarity features. The second model, as an extension looking at the modern transformer architecture, looks at the *distilbert-base-uncased* Hugging Face transformer model which takes a concatenated query-passage pair as input and outputs contextual embeddings. The embedding of the *[CLS]* token is passed through a feedforward layer to produce a relevance score via sigmoid activation. The following section outlines the forward pass for each model.

Forward Pass (Two Tower). Let $\mathbf{q} \in \mathbb{R}^d$ and $\mathbf{p} \in \mathbb{R}^d$ be fixed-length embeddings for a query and a passage, respectively (derived from GloVe, with $d = 300$).

These vectors are passed through two separate feedforward towers:

$$\mathbf{h}_q = f_q(\mathbf{q}) \in \mathbb{R}^h$$
$$\mathbf{h}_p = f_p(\mathbf{p}) \in \mathbb{R}^h$$

Each tower is a two-layer MLP:

$$f(\mathbf{x}) = \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

where $W_1 \in \mathbb{R}^{h \times d}$, $W_2 \in \mathbb{R}^{h \times h}$, and $h = 256$.

Given the output embeddings $\mathbf{h}_q, \mathbf{h}_p \in \mathbb{R}^h$, we compute the following similarity features: cosine similarity (17), Euclidean distance (18), the dot product (19), and element-wise absolute difference between the query and passage (20). These are concatenated into a single feature vector:

$$\mathbf{z} = [S_{\text{dot}}(q,d), S_E(q,d), S_{\cos}(q,d), |q-d|] \in \mathbb{R}^{3+h}$$

The final scoring function is a four-layer perceptron:

$$\hat{y} = \sigma\left(W_4 \cdot \phi\left(W_3 \cdot \phi\left(W_2 \cdot \phi\left(W_1 \cdot \mathbf{z} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3\right) + \mathbf{b}_4\right)$$

where ϕ is the ReLU activation function and σ is the sigmoid function. The output $\hat{y} \in [0, 1]$ represents the predicted relevance score.

Forward Pass (distilbert-base-uncased). First, the query and passage text are concatenated into a single sequence using HuggingFace's *AutoTokenizer*, which selects the appropiate tokeniser based on the specified transformer model, *distilbert-base-uncased*. This tokeniser performs lowercasing for uncased models, WordPiece tokenisation, and returns the input in the form of an input ID and an attention mask. Importantly, the input IDs are integers representing the tokenised vocabulary, while the attention mask indicates which tokens are actual input (corresponding to "1") and which are padding (corresponding to "0"). Dimensionally this results in:

$$\textbf{input\_ids} \in \mathbb{Z}^{128}, \quad \textbf{attention\_mask} \in \{0,1\}^{128}$$

These inputs are passed into the DistilBERT encoder. Unlike traditional models that embed the query and passage separately (e.g., GloVe), DistilBERT processes the entire sequence jointly, preserving token order and allowing for contextual interaction between query and passage tokens through self-attention mechanisms [8].

Each token ID is first mapped to a dense vector embedding, producing:

$$\textbf{token\_embeddings} \in \mathbb{R}^{128 \times 768}$$

These embeddings pass through DistilBERT's 6-layer transformer stack, consisting of multi-head self-attention and feedforward layers, resulting in the contextualized output known as the last hidden state:

$$\textbf{last\_hidden\_state} \in \mathbb{R}^{128 \times 768}$$

From this output, the embedding corresponding to the [CLS] token (the first token in the sequence) is extracted:

$$\textbf{cls\_embedding} \in \mathbb{R}^{768}$$

This [CLS] embedding is then fed into a two-layer feedforward neural network (MLP) to compute a scalar relevance logit:

$$\mathbf{z}_1 = \mathbf{W}_1 \cdot \textbf{cls\_embedding} + \mathbf{b}_1, \quad \mathbf{z}_1 \in \mathbb{R}^{128}$$
$$\mathbf{a}_1 = \text{ReLU}(\mathbf{z}_1), \quad \mathbf{a}_1 \in \mathbb{R}^{128}$$
$$\textbf{logit} = \mathbf{W}_2 \cdot \mathbf{a}_1 + \mathbf{b}_2, \quad \textbf{logit} \in \mathbb{R}^1$$

The final score is obtained by applying the sigmoid activation function to the logit:

$$\textbf{score} = \sigma(\textbf{logit}) = \frac{1}{1 + e^{-\textbf{logit}}} \in (0,1)$$

This score represents the predicted probability that the passage is relevant to the input query.

Training Process. Both models are trained to minimise binary cross entropy (10) between predicted and true relevance. In both models, each query-passage pair goes through a forward pass to compute a relevance score, followed by loss computation and gradient backpropagation. Model weights are updated using the Adam optimiser:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{21}$$

Where m_t and v_t are moving averages of gradient and squared gradient, η is the learning rate (set to 2e-5), and ϵ is a small constant added for numerical stability.

The two-tower model is trained for 10 epochs with a learning rate of 1e-3 - more epochs resulted in excessive validation loss (overfitting). Due to compute constraints, *distilbert-base-uncased* will be trained for just 2 epochs with a learning rate of 2e-5.

Evaluation. Both models will be evaluated using mAP and mNDCG as in the three other tasks.

6.1 Neural Network Performance on the Validation Set

Model	mAP	mNDCG
Two-Tower	0.39	0.51
Distilbert-base-uncased	0.90	0.93

The Two-Tower neural network achieves an mAP of 0.39 and an mNDCG of 0.51 on the validation set. While it shows marginal improvement over the logistic regression model (mAP = 0.35, mNDCG = 0.48), its performance lags behind the LambdaMART model, which

achieves mAP = 0.56 and mNDCG = 0.65. This comparison highlights the relative strength of the LambdaMART learning-to-rank method; using gradient boosted trees with specifically feature engineered similarity features between query and passage is evidently more optimal than this specific neural network two-tower architecture where the similarity features are scored instead via a multilayer perceptron.

A key limitation of the Two-Tower model, and the LambdaMART and logistic regression models by extension, lies in its reliance on pre-trained GloVe embeddings. These are static and non-contextual, meaning that each word is represented by a fixed vector regardless of its surrounding context. As a result, the model cannot capture dynamic semantic relationships between tokens that are often critical in determining passage relevance. Additionally, its late-interaction architecture — where query and passage are encoded independently before being combined — limits the extent of cross-attention between the inputs.

In contrast, while the *distilbert-base-uncased* model does not strictly represent the same training data representations as task 3, it is included to convey the power of modern transformer architectures for information retrieval. This effectiveness is evident, achieving substantially higher performance (mAP = 0.90, mNDCG = 0.93). By embedding the query and passage together in a single input sequence and processing them through stacked self-attention layers, DistilBERT produces contextualised token representations that encode rich interactions between the two texts. This effect is limited by representing the query-passage input as pre-trained glove embeddings. The early interaction between the query-passage pair allows the model to produce a [CLS] token which better resolves ambiguities and understand relevance in context, leading to more accurate ranking predictions.

7 Conclusion and Future Work

This project investigated several ranking models for passage retrieval, ranging from traditional approaches to modern neural architectures. Among traditional baselines, BM25 performed competitively (mAP = 0.80, mNDCG = 0.85), serving as a strong benchmark. LambdaMART, which uses static GloVe embeddings and hand-crafted similarity features, offered moderate gains (mAP = 0.56, mNDCG = 0.65), outperforming logistic regression (mAP = 0.35, mNDCG = 0.48) due to its ability to model more complex interactions through gradient-boosted decision trees.

The Two-Tower neural model, also using GloVe embeddings, yielded slightly worse performance compared to LambdaMART (mAP = 0.39, mNDCG = 0.51). While these results fell short of BM25, it is plausible that both models could outperform BM25 if enhanced feature representations were used. For LambdaMART, the choice of similarity features may have limited its discriminative power. Similarly, the Two-Tower model relied on static GloVe embeddings which lack contextual sensitivity. Future work could involve incorporating contextual embeddings such as BERT or DistilBERT outputs into the LambdaMART or Two-Tower pipelines to provide richer input representations.

The transformer-based *distilbert-base-uncased* model achieved the best performance overall (mAP = 0.90, mNDCG = 0.93), highlighting the strength of contextual embeddings and attention mechanisms in capturing deep semantic relationships. However, this model is computationally expensive and slow to train and evaluate. A valuable direction for future work is to explore more lightweight transformer variants (e.g., *MiniLM*, *TinyBERT*) that can strike a balance between efficiency and ranking performance while still outperforming BM25.

Additionally, negative sampling was employed to create a balanced training set. It was observed that as the negative ratio is increased, performance degrades. The weighted BCE (11) does not combat this, and thus this is another source of arbitrary inconsistency for performance. Future efforts should investigate more intelligent methods of sampling which ensure a more consistent performance for mAP and mNDCG irregardless of the number of samples sampled.

In summary, the most traditional method of information retrieval, BM25, performs the best under the metrics of mAP and mNDCG when compared to logistic regression, LambdaMART, and the two-tower neural architecture. However, the transformer based *distilbert-base-uncased* suggests that more modern techniques of self-attention to produce contextual embeddings, as opposed to static pre-trained embeddings, should be used to predict relevancy.

References

[1] Steven Bird, Ewan Klein, and Edward Loper. 2006. NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. Association for Computational Linguistics, 69–72.

[2] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc. https://www.nltk.org/book/

[3] Christopher J. C. Burges. 2010. From RankNet to LambdaRank to LambdaMART: An Overview. In *Learning*, Vol. 11. 81.

[4] Yinqiong Cai, Jiafeng Guo, Yixing Fan, Yanyan Liu, and Xueqi Cheng. 2022. Hard Negatives or False Negatives: Correcting Pooling Bias in Training Neural Ranking Models. *arXiv preprint arXiv:2209.06366* (2022). https://www.researchgate.net/figure/Recall-performance-of-BM25-and-ANCE-on-the-MS-MARCO-and-TREC-DL-datasets_tbl1_363500260

[5] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446. doi:10.1145/582415.582418

[6] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press. https://nlp.stanford.edu/IR-book/

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 26. Curran Associates, Inc., 3111–3119. https://papers.nips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*. 5998–6008.

[9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 38–45. https://aclanthology.org/2020.emnlp-demos.6

[10] Xinyu Zhang, Nandan Thakur, Odunayo Ogundepo, Ehsan Kamalloo, David Alfonso-Hermelo, Xiaoguang Li, Qun Liu, Mehdi Rezagholizadeh, and Jimmy Lin. 2023. MIRACL: A Multilingual Retrieval Dataset Covering 18 Diverse Languages. *Transactions of the Association for Computational Linguistics* 11 (2023), 1114–1131. doi:10.1162/tacl_a_00595