COMP0084: Information Retrieval and Data Mining Coursework-2

ABSTRACT

The coursework aims to implement and evaluate different methods for passage re-ranking in an information retrieval system. The main objective is to improve the relevance of retrieved passages by re-ranking them based on their relevance to the query. The implemented methods include deep learning models, such as Logistic Regression and Feed-forward neural networks, as well as traditional IR models, such as BM25 and TF-IDF. The performance of the different methods will be evaluated using standard IR metrics, such as precision, recall, and F1-score, on a dataset of queries and relevant passages. The results of the evaluation will provide insights into the effectiveness of the different methods for passage re-ranking in an information retrieval system, and can be used to guide future improvements in this field.

KEYWORDS

Information Retrieval, **Ranking System**, **Neural Networks**

ACM Reference Format:
. 2018. COMP0084: Information Retrieval and Data Mining Coursework-2. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (IRDM Coursework).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Information retrieval systems are software tools that help users find relevant information from large collections of documents. These systems work by processing queries, which are typically expressed as keywords or natural language phrases, and matching them against a database of documents. The process of processing queries involves various techniques such as parsing, stemming, and indexing to extract meaningful information from the query and to identify relevant documents that match the query.

Once the system has identified a set of documents that match the query, it ranks them according to their relevance to the query. This process is known as reranking and is typically based on a combination of factors such as the frequency of the query terms in the document, the location of the query terms within the document, and the overall relevance of the document to the query. Reranking can significantly improve the effectiveness of an information retrieval system by ensuring that the most relevant documents are presented to the user at the top of the search results.

Overall, information retrieval systems play a critical role in modern information access and are widely used in various domains such as web search, e-commerce, and scientific research. As the volume of digital information continues to grow rapidly, the development of effective information retrieval systems has become increasingly important for enabling users to find the information they need quickly and easily.

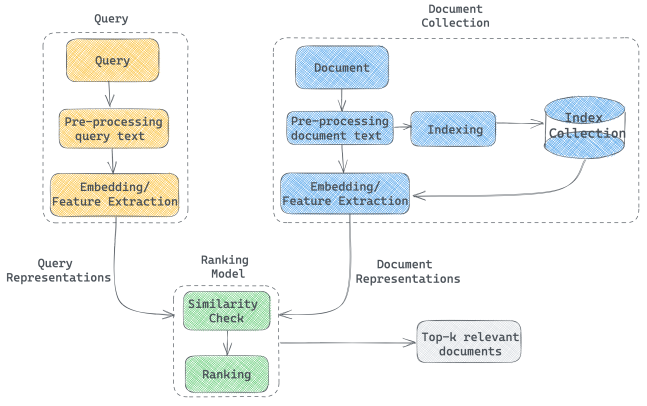


Figure 1: A conventional Information Retrieval System

2 EVALUATING RETRIEVAL QUALITY

We have implemented methods to compute the average precision and NDCG metrics of a ranking model. BM25 has been used as the baseline model to implement these metrics. The implementations have been calculated on **validation data**.

2.1 Preprocessing

Text preprocessing is a crucial step in information retrieval, as it involves transforming raw text data into a more structured format that can be easily analyzed by machine learning algorithms. It helps with Noise reduction, Standardization, Feature extractiona and Efficiency improvement. For task 1, the validation data has been loaded and preprocessed with steps like

- Lowercase Conversion
- Punctuation Removal
- Apostrophe Removal
- URL Removal
- White-space Removal
- Tokenisation
- Stop-word Removal

The source passages and queries underwent cleaning, which involved several steps. Firstly, the entire text was converted to lowercase to avoid any case sensitivity issues. Punctuation marks and apostrophes were removed to reduce noise in the data, and URLs were split into separate words while removing prefixes such as *https* and *www* to allow for hyperlinks in search results. Finally, any white spaces were removed from the data. The passages were then tokenized into unigrams for individual access in the future.

2.2 Ranking Model

BM25 has been used as the ranking model for task 1. BM25 is a probabilistic model that aids in information retrieval by taking into account various factors such as relevance, term frequencies, and average document lengths. BM25 considers the average document length of the collection to mitigate the impact of individual document length. BM25 has control parameters, including k1 and b, which regulate the influence of term frequency and document length on information retrieval. This ensures that documents with high occurrences of a term or lengthier than average do not receive extra weightage. The BM25 score between a query and a document can be expressed as follows:

$$score(q, D) = \sum_{i=1}^{N} IDF(q_i) \frac{f(q_i, D)(k1 + 1)}{f(q_i, D) + k1(1 - b + b(\frac{|D|}{avgdl})} \quad (1)$$

The model takes parameters defined in IRDM coursework-1.The passages in **validation data** have been ranked by BM25.

2.3 Average Precision

Precision is the fraction of relevant documents retrieved out of the total number of documents retrieved, and it is a key performance indicator for information retrieval systems. Average Precision is calculated by taking the precision at each rank and averaging them over all relevant documents. Average precision is commonly used in information retrieval research to evaluate the performance of different retrieval models and algorithms. A function has been set-up in the code that incorporates below formulae to calculate the precision of each query-passage set, then the average precision of query. Finally, the mean average precision is calculated for all the queries.

Precision measures the proportion of retrieved documents that are actually relevant to the user's query.

$$Precision = \frac{ActualRelevantDocs}{PredictedRelevantDocs} \quad (2)$$

Average Precision is calculated by averaging the precision values at each relevant document rank, where relevant documents are those that are deemed relevant by the user.

$$AP = \frac{\sum_{doc=1}^{N} Precision * Relevancy}{RelevantDocs} \quad (3)$$

Mean Average Precision is the average of the average precision values over all queries in a given dataset.

$$MAP = \frac{\sum_{query=1}^{M} AP}{TotalQueries} \quad (4)$$

2.4 Normalised Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) is a metric for evaluating the effectiveness of information retrieval systems. The basic idea behind NDCG is that highly relevant documents should be ranked higher than less relevant documents. The metric calculates the gain of each document, which is determined by the relevance of the document to the query, and then discounts the gain based on the rank of the document.

Gain is a measure of the relevance or usefulness of a document for a given query, and is is calculated based on a document's relevance score. Highly relevant documents are given a higher gain than less relevant ones. The formula for Gain for a document as ma fucntion of relevancy is given as,

$$Gain = 2^{rel} - 1 \quad (5)$$

Discounted Cumulative Gain (DCG) is a measure of the relevance of a ranked list of documents for a given query. It is calculated by summing the gains of each document, discounted by the rank of the document. DCG for top-k ranked documents by the ranking model is given by,

$$DCG(k) = \sum_{i=1}^{k=actualorder} \frac{Gain}{log_2(i+1)} \quad (6)$$

Ideal Discounted Cumulative Gain (IDCG) represents the maximum possible score for a given list of documents. It is calculated by sorting the documents by their relevance scores and then computing the DCG for that sorted list. IDCG for top-k ideally ranked documents is given by,

$$IDCG(k) = \sum_{i=1}^{k=idealorder} \frac{Gain}{log_2(i+1)} \quad (7)$$

Normalized Discounted Cumulative Gain (NDCG) metric is calculated by comparing the Discounted Cumulative Gain (DCG) of a retrieval system's ranked list of results with an ideal DCG (IDCG) that represents the maximum possible score for that list.

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)} \quad (8)$$

Results: A function has been implemented in the code that uses the relevancy information provided in the dataset to calculate DCG and IDCG. NDCG is then calculated by taking the ratio of the DCG of the top-k documents retrieved by the ranking model on the basis of BM25 score and the top-k documents based on the relevancy provided in the dataset. The code is stored in the file name *task1.py*. For the validation set, we have compared performance of different k values, i.e. different number of top ranked documents.

3 LOGISTIC REGRESSION (LR)

Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict a binary outcome. It is a type of binary classification model that can be used to predict the probability of a document being relevant or not relevant to a given query. The basic idea of logistic regression is to transform a linear regression model that predicts a continuous output variable into a model that predicts a binary output variable. This is achieved by applying a sigmoid function to the linear combination of input variables along with certain weights and a bias, which maps the

Table 1: BM25 Performance on Validation Data for differnt k values

Metric	k	Value
MAP	3	0.1805
NDCG	3	0.2509
MAP	10	0.2188
NDCG	10	0.4535
MAP	100	0.2305
NDCG	100	0.7624

continuous output range to the range [0,1].

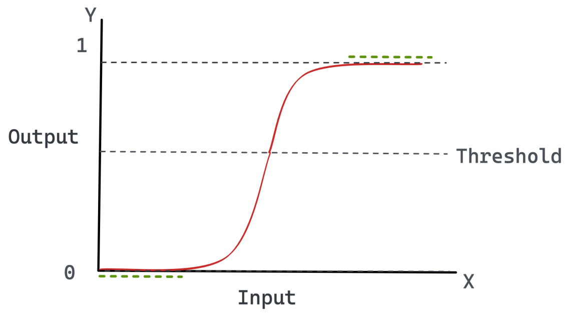
Linear model is a linear combination of independent variables' set **X** which is multiplied by a set of weights and a bias term **w** and is given as,

$$z = \mathbf{Xw} \tag{9}$$

$$z = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{10}$$

Sigmoid Function is used to convert the continuous output of the linear model to a binary output by passing the linear model output through an exponential function. The output of the sigmoid function g(z) is interpreted as the probability of the positive class, i.e. if the probability of a document being relevant to a query is greater than a threshold, then it is classed as relevant, otherwise it is not relevant.

$$sigmoid = \frac{1}{(1 + e^{-z})} \tag{11}$$



·.

Figure 2: Sigmoid (Logistic) Function as the s-curve

Based on the above formulae, a logistic regression class has been defined in the code, that takes input and output features to formulate a linear model as mentioned in eq (9) and (10). The linear model is then transformed to a sigmoid function using eq (11). The sigmoid (also called logistic) model is then fit on the entire training data, to tune the weights and bias values. The model fitting is done by updating the weights such that the linear combination of the features is close to the output value. It also includes multiplying with learning rate, which is the number of steps the model is taking to reach a global minima. With a set of updated weights, to predict the classes of the test data.

3.1 Model

Dataset is a subset of the training data that has been extracted through **Negative Sampling**. The function uses the concept of negative sampling to generate negative examples for a given query ID (qid) and improve the ranking of relevant documents. The sampling function extracts 20 negative samples against every 10 positive samples. This helps reduce the computational load.

Preprocessing has been performed on the sampled data, and the steps are similar to task 1, including Lowercase conversion, Punctuation Removal, Apostrophe Removal, URL Removal, White-space Removal, Tokenisation, Stop-words Removal. Both queries and passages have gone through these steps of preprocessing to form a clean and tractable collection of words.

Embedding is a technique used to represent data in a low dimensional vector space, to capture semantic relationships between various features of the data. Our task 2 uses **Word2vec** embedding which is an embedding model that is trained using a neural network architecture that learns to predict the likelihood of a word given its context, resulting in a compact, semantically meaningful representation of words. Word2Vec embedding model has been accessed through **gensim** library. The model is trained on our sampled data and a unique embedding for each word in sampled data is generated. The same trained model is used to generate embedding for our validation data and test data in upcoming steps of the task.

Features are the most crucial part of a machine learning models. For our logistic regression model, the input features that have been extracted are query length, document length and cosine similarity between the query vector and passage vector. Target feature is pssage relevancy.

- **Query length and Document length** have been chosen citing their prominent presence and relevance in ranking models like vector-space models and probabilistic models (seen in coursework 1). These two parameters provide information about the complexity and specificity of the query and passage, respectively.
- **Cosine Similarity** between a query vector and a passage vector can be used to quantify the degree of relevance between them. It measures the cosine of the angle between the two vectors, where a value of 1 indicates that the vectors are identical, and a value of 0 indicates that they are orthogonal (completely dissimilar). Thus, the model can better capture the semantic similarity between the query and passage. This makes it a tangible input feature for our model.

- **Relevancy** of the passages has been used as the *output feature* because the aim of the ML model is to predict the rank the document by predicting their relevance to queries. The value of relevance is either 0 or 1 in our data, where 1 signifies the passage being relevant and 0 shows that the passage is not significant.

Architecture A Logistic Regression class has been created and underlying functions namely *sigmoid, fit, predict* have been defined for the Logistic Regression model. The model uses different values of *learning rate* to get the best performing model. The model then uses the **fit** method to train with the sampled training data. The evaluation metric has been defined as **Precision** because tracking the number of truly relevant documents is of maximum interest for our problem. The predictions produced by the *predict* function are used to classify documents as relevant or not-relevant. Since the output produced is continuous, we add a binary classification, with a threshold on predicted relevancy that decides if the document is to be classified as not-relevant(0) or relevant(1). On the basis of the range of prediction values generated, we have set the **threshold=0.1**. The predicted values ranged mostly close to zero, as the training data contains a large number of documents with relevancy=0 which has introduced a bias in the model. Thus, analysing the number of true relevant documents via a confusion matrix, a threshold of 0.1 has been decided. This threshold preserves the number of more no. of relevant documents that other higher values.

Hyperparameters are the control parameters that tune the performance of the model. The following hyperparmeters have been specified in our model,

- **Iterations** is the number of times a training is performed in a loop until the model converges to the optimal set of parameters that minimize the cost function. The number of iterations have been kept at *n_iter=1000* as the performance was not changing much beyond the value.
- **Learning Rate** is a hyperparameter that determines the step size at which the algorithm updates the model parameters during the training process. If the learning rate is too high or too low, it can fail to track the global minima and impact the model performance. We have tried different learning rates with our logistic regression model on training data and validation data and the performances are mentioned in the table. 0.01 is the optimal learning rate for our model.

Table 2: Effect on Learning rate on Logistic Regression model

Learning Rate	Precision	F1	Accuracy
0.001	0.0011	0.0021	0.7903
0.010	0.00127	0.0025	0.8448
0.1	0.0	0.0	0.9989
1	0.0	0.0	0.9989
10	0.001095	0.00218	0.001095

Results: The code is stored in the file name *task2.py*. For test data, a tab-seperated file named **LR.txt** has been generated that has the passages of *candidate_passages_top1000.tsv* re-ranked using the logistic regression model. For the validation set, the learning rate has been taken as 0.01 and the performance metrics are shown below.

Table 3: Logistic Regression Performance on Validation Data

Metric	Value
MAP	0.0069
NDCG	0.1037

4 LAMBDA_MART (LR)

LambdaMART is an algorithm, which combines the ideas of gradient boosting and lambda ranking. The algorithm works by building a series of decision trees, where each tree is trained to predict the difference in relevance between pairs of documents. The model uses NDCG as the optimisation metric. The model minimizes the expected value of a cost function by penalizing the deviation of the predicted pairwise differences from the true pairwise differences.

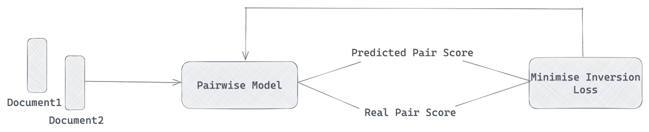


Figure 3: LambdaMart Learn-to-rank Model

Figure 3 shows the pseudo code for lambdamart model. When a document with a lower relevance score is ranked higher than a document with a higher relevance score, then it is considered an inversion. The inversion loss is the sum of the weights of all pairwise inversions, where the weight of each inversion is defined as the difference in the relevance scores between the two documents involved in the inversion. The inversion loss is a measure of the total difference in relevance scores between the predicted ranking and the true ranking. LambdaMART model minimises the inversion loss during training by updating the model parameters based on the gradients of the inversion loss with respect to the predicted relevance scores and the model parameters.

4.1 Model

Dataset is a subset of the training data that has been extracted through **Negative Sampling**, same as Task 2.

Preprocessing has been performed on the sampled data, and the steps are similar to task 1 and 2.

Embedding has been are generated for the sampled data, and the steps are similar to task 2.

Features have been are generated for the sampled data, and the steps are similar to task 2.

Architecture An *XGBRanker* object has been created using *xgboost* library and a list of plausible parameters have been defined for the

```
Algorithm: LambdaMART
set number of trees $N$, number of training samples $m$, number of leaves per tree $L$,
learning rate $\eta$
for $i = 0$ to $m$ do
    $F_0(x_i) = \text{BaseModel}(x_i)$    //If BaseModel is empty, set $F_0(x_i) = 0$
end for
for $k = 1$ to $N$ do
    for $i = 0$ to $m$ do
        $y_i = \lambda_i$
        $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$
    end for
    $\{R_{lk}\}_{l=1}^{L}$    // Create $L$ leaf tree on $\{x_i, y_i\}_{i=1}^{m}$
    $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$    // Assign leaf values based on Newton step.
    $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$    // Take step with learning rate $\eta$.
end for
```

Figure 4: LambdaMart Learn-to-rank Pseudo Code [1]

LambdaMART model. The model performs hyperparemeter tuning to get the best set of values for the model. The model then uses the **fit** method to train with the training and validation set extracted from the sampled data. The evaluation metric has been defined as **ndcg**. The model returns 100 evaluation results for 100 rounds of boosting and the last result of each boosted set is taken as the final evaluation value.

Hyperparameter Tuning has been performed manually using a *for loop.* Algorithms like GridSearchCV seemed incompatible with xgboost algorithms. Upon tuning the hyperparameters, following set of values were given as the optimal ones, where *max_depth* is the maximum allowed depth of the decision trees, *eta* is the learning rate of the model and *eval_metric* is the metric for evaluating the validation set which is 'ndcg' in our case. The optimal parameters achieved through hyperparameter tuning are: 'max_depth' = 3 'eta' = 0.2 'objective = rank:ndcg
The best score achieved is 0.9399 on training data.

Result The code is stored in the file name task3.py. For test data, a tab-seperated file named LM.txt has been generated that has the passages of *candidate_passages_top1000.tsv* re-ranked using the LambdaMart model. For the validation set, the performance metrics are shown below:

Table 4: LambdaMART Performance on Validation Data

Metric	Value
MAP	0.0078
NDCG	0.0775

5 NEURAL NETWORK

Neural networks are particularly effective in IR because they can automatically learn from the data without the need for handcrafted features or prior knowledge. Neural networks learn to identify important features in text documents that are relevant to a given query.

Feed-forward Neural Networks is our model of interest which are called so because these models do not form a cycle with previous nodes. Feed-forward neural networks consist of multiple layers of interconnected nodes, with each node processing the inputs from the previous layer using a set of weights and biases. The output of the final layer represents the predicted output of the model. The predicted output is then used to classify the documents as relevant or non-relevant. Feed-forward network capture non-linearity in the data very well (like our text data), and is good at generalising as compared to other more complex models.

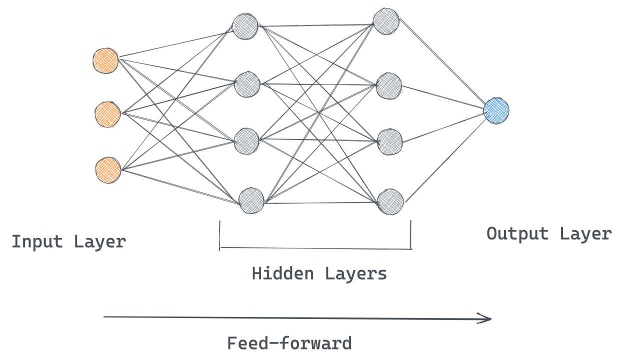


Figure 5: Feed-forward Neural Network

5.1 Model

Dataset is a subset of the training data that has been extracted through **Negative Sampling**, same as Task 2,3.

Preprocessing has been performed on the sampled data, and the steps are similar to task 2 and 3.

Embedding has been are generated for the sampled data, and the steps are similar to task 2,3.

Features have been are generated for the sampled data, and the steps are similar to task 2,3.

Architecture: A *Sequential* object has been created using *tensorflow-keras* library and the layres of the neural network have been defined. A sequential model is a linear stack of layers in a neural network, where the output from each layer is passed as input to the next layer. Our model consists of,

- Input/Hidden Layer1: Dense(nodes=128, activation='relu')
- Hidden Layer2: Dropout(0.2)
- Hidden Layer3: Dense(64, activation='relu'))
- Hidden Layer4: Dropout(0.2)
- Hidden Layer5: Dense(32, activation='relu'))
- Output Layer: Dense(nodes=1, activation='sigmoid')

Dense Layer is a type of layer in a neural network where each neuron is connected to every neuron in the previous layer densely.

It performs a linear transformation of the input and applies an activation function to produce the output.
Dropout Layer is a regularisation layer in a neural network that randomly drops out a percentage of the neurons in the layer during training. This helps prevent overfitting by forcing the network to learn more robust and generalisable features.

The model has first layer as a dense layer with 128 nodes, where the features extracted in the previous step are also passed. The input then passes through other hidden layers to achieve optimal set of parameters. The final output is received from the output layer. The documents are then ranked on the basis of the prediction score.

Hyperparameters

- **Nodes** are junctions for information flow in neural networks. They receive inputs from other nodes or the input layer, perform a computation using weights and biases, and produce an output that is passed to the next layer or the output layer. The input layer in our model has 128 nodes, whereas the output layer has 1 node. The intermediary layers have a sequentially decreasing number of nodes to converge the input.
- **Activation Function** is the non-linearity applied to the linear functions in order to capture the patterns more accurately. Activation functions are applied at each layer to transform the input from the previous data. Our model has used *relu* as the activation for input and hidden layers and *sigmoid* as the activation for the output layer.
  ReLU or Rectified Linear Unit only passes through positive values and sets negative values to zero. ReLU has been used because it prevents the vanishing gradient problem, thus helping the model to learn better. It is given by,

  $$ReLU = max(0, x) \tag{12}$$

  Sigmoid function as discussed in task 2 (Logistic Regression) is a non-linear transformation and is appropriate for binary classification tasks. The exponential nature of sigmoid function restricts the values between a range(0,1). Thus, it is a valid choice for our problem of classifying documents based on relevancy.

  $$sigmoid = \frac{1}{(1 + e^{-z})} \tag{13}$$

- **Optimiser** *Adam* has been used as the optimiser because it is an improved version of the traditional stochastic gradient method of optimisation. Adam updates the parameters based on previous learning rates, while also keeping a track of the momentum

Results: The code is stored in the file name task4.py. For test data, a tab-separated file named NN.txt has been generated that has the passages of *candidate_passages_top1000.tsv* re-ranked using the Feed-forward neural network model. For the validation set, the performance metrics are shown below:

Table 5: Neural Network Performance on Validation Data

Metric	Value
MAP	0.0100
NDCG	0.1372

REFERENCES

[1] Christopher J. C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report. Microsoft Research. http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf