Information Retrieval and Data Mining CW2

ABSTRACT

The goal of the assignment was to develop information retrieval models to solve the problem of passage retrieval. This means that the most can effectively and efficiently return a ranked list of passages relevant to a given query.

KEYWORDS

information retrieval, data mining, BM25, logistic regression, LambdaMART, XGBoost, neural networks

ACM Reference Format:

. 2018. Information Retrieval and Data Mining CW2. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

This document contains the choices and steps taken to build and evaluate the retrieval quality of three learning to rank models, namely Logistic Regression (LR), LambdaMART (LM) and a Neural Network (NN).

Given training and validation data containing a relevance score for each query-passage pair, the models are trained to calculate a predicted relevance score for an unseen query-passage pair on test data. The score is then used to rank (in descending order) the top 100 relevant passages for each query within the test data. For each model, there are three inputs/features. The three features provide a representation for each of the three standard groups of features in document retrieval: a dynamic feature (cosine similarity), a query dependent feature (query length) and a query independent feature (passage/document length).

Once a model has been trained and the validation data rankings have been calculated, normalized discounted cumulative gain (NDCG) and mean average precision (MAP) were used to evaluate the retrieval quality of the trained model.

Finally, the test data was ranked and a text file was produced for each model to show the associated passage id, rank, score and the algorithm name for the top 100 ranked passages of each query id.

2 EVALUATING RETRIEVAL QUALITY

Within this section of the assignment, *validation_data.tsv* was used to compute the performance of the BM25 model computed in Coursework 1. The metrics that were used were MAP and NDCG.

To briefly revisit the steps taken to compute the BM25 score, first the validation data was preprocessed. The preprocessing function first uses case-folding. This ensures all the words in a string (the passage) are in lowercase, to ensure capitalised words at the beginning of a sentence match with a lowercase word in a query if this occurs. Even though this may lead to unintended matches for proper nouns, it avoids the expectation that queries will be capitalised correctly.

Next, the passage was tokenised into individual terms using the RegexpTokenizer from nltk, because it forms tokens out of alphabetic sequences, money expressions, and any other non-whitespace sequences. The RegexpTokenizer was chosen as it incorporates several rules of linguistics to split the sentence into the most optimal tokens. After this, all non-alphabetic tokens were removed using Python's .isalpha() method.

Finally, English stopwords were removed using the nltk.corpus stopwords.

To compute the MAP and NDCG metrics, the BM25 score was calculated for each for the query-passage pairs by creating and inverted index and setting k1 = 1.2, k2 = 100, and b = 0.75 as was required in Coursework 1.

The top 100 passages retrieved by BM25 score are kept for each query.

2.1 Average Precision

As its name suggests, to calculate the mean average precision of a retrieval model the sum of the precision values of all the relevant documents retrieved is computed and then this is divided by the total number of relevant documents for the associated query. The average precision values for all unique queries is totalled and divided by the count of unique queries to obtain the mean average precision. Precision itself is calculated by dividing the number of true positives (relevant documents) by the total number of documents (relevant and non-relevant).

The mean average precision for the validation set is calculated and reported for each ranking model (including BM25) in this report.

2.2 Normalized Discounted Cumulative Gain

NDCG is another popular metric for evaluating retrieval quality. The NDCG metric represents how close a ranking algorithm is to the perfect ranking algorithmn. This is shown by its calculation which is DCGp/IDCGp which represents the discounted gain at rank p divided by the ideal discounted gain at rank p. If the ranking for a query is perfect then a NDCG score of 1 will be calculated.

To calculate the DCG the sum of the ratio of the gain and the discount is computed. For the purpose of this report the non-linear gain formula was used. This is because the non-linear gain function gives higher values to documents with higher relevance scores, but it increases at a decreasing rate. The decision was taken as I believe it is a more accurate representation of relevance, since the difference in the quality of a document or passage in each perceived

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

unit of relevance is smaller at low levels of relevance and greater at high levels of relevance.

The mean of the NDCG values is taken to get the average NDCG value for the model.

2.3 Results

For the BM25 ranking on the validation set the MAP score is 0.2755 and the average NDCG score is 0.1572.

3 LOGISTIC REGRESSION

Logistic regression (LR) is a statistical model used for binary classification tasks. In information retrieval logistic regression is used to classify whether a passage or document is relevant or not. In this case, the model outputs the probability score of a passage being relevant. If the score is above 0.5 then the document is considered relevant, otherwise it is deemed not relevant. The probability scores are later used for ranking purposes. The model is then trained using a target variable 'relevance' which is either 0 or 1.

Within the training data provided the number of positive samples is very low for each unique query (0.01% of total passages for each query is positive). To mitigate the likelihood of the model overfitting, negative sampling was used before calculating embedding. The motivation behind negative sampling in information retrieval is to improve the accuracy of the model by training it to recognise what is relevant and non-relevant. This was executed by selecting a random positive (relevant) example for each unique query and then selecting up to 10 non-relevant samples for the query. This creates a subset of the training data where 9% of the samples were positive.

After the subset was created passages and query embeddings were computed using FastText for the training subset, validation and text data. The embeddings were used as input to all three ranking models. FastText is a variant of Word2Vec that uses character n-grams to learn word embeddings. It captures morphological information by representing each word as a bag of character n-grams.

For each token, 300 dimension embeddings were created. This lead to the passage_embeddings vector for the training data, for example, having the shape (50341,300) after negative sampling and FastText embedding. FastText was chosen as the embeddings algorithm as it is meant to capture syntactic and semantic relationships between words which seemed the most appropriate for this coursework.

I used the FastText 'cc.en.300.bin' model because it is a popular pre-trained model that provides word embeddings for English. The model is trained on a dataset of web pages, which is appropriate given the query-passage relationship explored in this coursework. As requested, the embeddings were averaged across all of the words in each query or passage.

For the input features the cosine similarity of each associated query and passage embeddings was calculated. The cosine similarity function for Coursework 1 was utilised for this task. In addition to this feature, the query length and document length were calculated as was mentioned in the introduction.

All three arrays of the features were then horizontally stacked to create the input array for the logistic regression model. The same was executed for the validation and test sets. Likewise, for the training and validation sets, the associated 'ground truth relevance score' was stored in a separate array. These steps were repeated for the LambdaMART and neural network models that follow.

For the creation of the logistic regression class a learning rate of 0.01 and 1000 iterations were the hyperparameters chosen. Gradient descent for the binary cross entropy loss function was calculated manually as the task specified the logistic regression implementation had to be our own. The weights and biases were updated after each iteration.

When using the predict method within the logistic regression class, the output probabilities were rounded to provide class predictions. These class predictions were returned alongside the output probabilities to give the option of checking the predictions compared to the ground truth, or ranking the probabilities in order which is needed for the evaluation metrics.

After trialling learning rates of 0.0001, 0.001, 0.01 and 0.1, I found the best learning rate to be 0.01, it provided a high validation accuracy of 99.89%. It also provided the strongest MAP and average NDCG scores shown below (although they are still very weak scores). I believe the reason 0.01 was the best learning rate was because it allowed the model to coverge quicker during the 1000 iterations compared to 0.001 and 0.0001, but did not cause values that were too large for the sigmoid function like the learning rate of 0.1 did.

3.1 Results - Logistic Regression

For the LR ranking on the validation set the MAP score is 0.1118 and the average NDCG score is 0.0299. The results are significantly worse than BM25.

4 LAMBDAMART MODEL

LambdaMART is a learning to rank algorithm that is an extension of multiple additive regression trees (hence the MART in its name). The main adaptation is that the gradient is calculated using LambdaRank, which accounts for the trade-off between document/passage relevance and their respective rank positions in the list. By accounting for all the data within the list, it can choose splits and leaf values that are better overall, rather than just for the individual query.

The inputs/features for the LambdaMART Model were computed in the exact same way as in logistic regression, so the process to derive the embeddings, cosine similarity, query length and document length will not be repeated.

To train the LambdaMART model for ranking, specific steps must be taken. A DMatrix must be made for the training and testing data in XGBoost. For the training DMatrix, the stacked feature array is included alongside the ground truth labels and feature names

['cosine_similarity', 'query_length', 'passage_length']. For the test DMatrix, the ground truth labels are excluded as they are unknown.

Next, a list of the query_id's and their frequency was created so that the set group method could be used on the DMatrix. This allows XGBoost to know which rows belong to each sequence, thus representing the number of passages per query_id.

Finally, the .train function was called with params = ['objective': 'rank:ndcg', 'eval_metric': 'ndcg', 'eta': 0.001, 'max_depth': 5"], rank was set to ndcg so that it executes listwise ranking in which NDCG was maximised.

Similarly to the logistic regression model, I tested various 'eta' (learning rate) values and max depths by holding one constant and changing the other and then recording the results. The best values for the hyperparameters that I tested are the ones above.

4.1 Results - LambdaMART Model

For the LM ranking on the validation set the MAP score is 0.1245 and the average NDCG score is 0.0233. The MAP result is an improvement on the logistic regression result, however the average NDCG is slightly worse. This implies that the LambdaMART model is better at retrieving relevant documents, but it is worse at placing relevant documents at the top of the list.

5 NEURAL NETWORK

5.1 Architecture

For the neural network model (NN) I have used a sequential feedforward network, which has for fully connected (dense) layers. The hidden layers have 20 neurons each and are activated by the widely used ReLU activation function after each fully connected layer. The ReLU activation function means that any negative input value is converted to 0. Each hidden layer is followed by a dropout layer, which randomly sets 20% of the input units to zero during training. This is to prevent overfitting and improve generalistion performance. The single output neuron has a sigmoid activation function which transforms the output range to between 0 and 1, this allows the output to be interpreted as a probability of the whether the passage is relevant to the query or not. The loss function used was binary cross entropy as provides a way to compare the predicted relevance scores with the ground truth. Since the document can only be relevant or not relevant (1.0 or 0.0) the binary cross entropy loss can be used. The loss function encourages the model to assign high scores to relevant documents and low scores to non-relevant documents. Additionally, BCE loss is easy to optimise using back propagation. The optimiser chosen was the Adam optimiser. The choice came because it is efficient and optimises the learning rate in an adaptive manner by computing the learning rate for each parameter, which it adjusts during training. I chose this simple feedforward network as I wanted a model that would be computationally efficient (taking a similar time and compute as LambdaMart and logistic regression) while also being able to test how a neural network compares in terms of performance.

5.2 Data

While most of the input data is exactly the same as for the logistic regression and LambdaMART models, oversampling and undersampling were also introduced. As mentioned earlier the training data was highly imbalanced so a combination of RandomUnderSampler and RandomOverSampler from Imbalanced-learn was used for the neural network to prevent the likelihood of overfitting. RandomUnderSampler deletes rows of the majority class (not relevant), while RandomOverSampler duplicates rows of the majority class. I was hesitant to use these tools to a large extent as I wanted to maintain some diversity within the training data, so I used the over sampler with a ratio of 0.4 and the under sampler with a ratio of 0.5. This lead to there being one positive sample for every three negative samples, a significantly more balance dataset than the original.

Pipeline was used to execute the steps sequentially to the training data.

Before computing any other preprocessing I split the training data into 90% training data and 10% validation data so that the validation accuracy could be tracked while training the model.

Additionally, Scikit-Learn's StandardScaler was used to standardise the data before inputting it to the neural network. This was done to avoid numerical instability and the make the model more robust. I was sure to only fit the StandardScaler on the training data to prevent data leakage. However, the transform was used on the training, validation and test data.

Otherwise the data input into the neural network was the same as the other models. The three features used remained as cosine similarity of the embeddings, query length and passage length.

5.3 Hyperparameter Tuning

The number of epochs of and batch size were my chosen hyperparameters. A smaller batch size means that the parameters are updated more often as they are updated after each batch. The 10% validation data split from the training data was used to tune these hyperparameters. After testing the accuracy on the validation data of batch sizes [100,150,200,250,300] and number of epochs [4,8,10,12] the best performing model had a batch size of 150 and 10 epochs with a validation accuracy of 67.6%.

5.4 Results - Neural Network

For the NN ranking on the validation set the MAP score is 0.1222 and the average NDCG score is 0.0263. The MAP result is an improvement on the logistic regression result, however worse than LambdaMart. The average NDCG is the best of the three models. This implies that the Neural Network model is best overall model with the highest ranking results, despite only being a simple feedforward network.