Information Retrieval Coursework 2

ABSTRACT

The main objective of this paper is to implement three different information retrieval models that can effectively rank the relevance of passages with respect to a given query. More specifically, this paper investigates the performance of Logistics Regression, LamdaMART, and LSTM.

KEYWORDS

information retrieval, neural networks, logistics regression

ACM Reference Format:
. 2023. Information Retrieval Coursework 2. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Data

- test-queries.tsv - a tab separated file, where each row contains a test query identifier (qid) and the actual query text.
- candidate-passages-top1000.tsv - a tab separated file with an initial selection of at most 1000 passages for each of the queries in test-queries.tsv. The format of this file is <qid pid query passage>, where pid is the identifier of the passage retrieved, query is the query text, and passage is the passage text (all tab separated). The passages contained in this file are the same as the ones in passage-collection.txt. However, there might be some repetitions, i.e. the same passage could have been returned for more than 1 query.
- train_data.tsv - a tab separated file which contains the columns <qid pid query passage relevance>. This is a file that is used for training the models
- validation_data.tsv - a tab separated file which contains the columns <qid pid query passage relevance>. This file contains the validation data which is used to evaluate the performance of the trained model

1.2 Text Processing

Each of the queries and passages in the available data files need to be pre-processed in order to be fed as inputs to the models. For this reason, as an initial step of this project, a pre-processing function is developed. The function takes as input a unified string, then replaces all new lines and punctuation marks with empty spaces, and finally converts all letters to lowercase. This standardized form of the text is then separated into tokens based on empty spaces i.e. the output of the function is a list of tokens.

2 TASK 1 - EVALUATING RETRIEVAL QUALITY

For this task, two evaluation metrics are developed. One of the metrics is the mean Average Precision defined as:

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: mAP and nDCG score at rank 3, 10 and 100

Rank	mAP score	nDCG score
3	0.177	0.194
10	0.214	0.272
100	0.227	0.340

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

where n is the number of classes and AP_k is the average precision of class k. The average precision is calculated as the mean of the precision scores after each relevant document is retrieved.

The second metric used for evaluation is the Normalized Discounted Cumulative Gain (nDCG) defined as:

$$nDCG = \frac{DCG}{optDCG}$$

where DCG is the discounted cumulative gain and $optDCG$ is the optimal discounted cumulative gain. The DCG is calculated as follows:

$$nDCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log(1+i)}$$

where p is the rank and rel is the relevance score. This metric rests on the assumption that highly relevant documents are more useful than marginally relevant documents and that the lower the rank of the a relevant document, the less useful it is to the user.

The optimal nDCG, $optDCG$, is obtained by sorting the documents for each of the corresponding queries based on their relevancy and applying the nDCG function.

As part of this task these two metrics were used to evaluate the BM25 model which was developed as part of the previous coursework. Before running the model the query and passage columns were pre-processed using the text-preprocessing function. This function created 2 new columns one for the query and one for the passage which contained the tokenized version of each of them. After this step, the BM25 model was ran on the validation_data.tsv file.

Once the BM25 model was ran, the mAP and nDCG were used to evaluate the model at rank 3, 10, and 100 for the 1148 unique queries. Table 1 summarizes the obtained results.

3 TASK 2 - LOGISTICS REGRESSION

As part of task 2, a logistics regression information retrieval model was implemented. The model was trained on the train_data.tsv file and was tested and evaluated on the validation_data.tsv file. To train and test this model the full train and the full validation set were used in this task.

To prepare the inputs for the model, firstly, the unique passages and queries in both the train and test data were prepossessed using

the text-preprocessing function such that they are obtained in a tokenized form. After this step, for each word in the query and for each word in the passage, a word embedding was created. For this task, Google's pretrained word2vec model was used. This model represents each word as a vector of length 300. Each word in each query as well as each word in each passage was then represented by a vector of 300. If a word was not present in Google's pre-trained embedding model, that word was assigned a vector of 300 with all 300 values set to 0.

Following this step, each query is a vector of size $n_i * 300$, where n_i is the number of words in the query, and each passage is of size $m_i * 300$, where m_i is the number of words in that passage. The next step involved averaging out the word embeddings for each query to get a single word2vec embedding of size 300. The same averaging was performed on the passages such that each passage is an average of its word embeddings and is now a single vector of size 300.

Please note that the average embeddings of the passages for both the train and validation dataset are saved at this stage since they take a long time to compute. This allows for easier use of the embeddings for the following tasks. This means task 2 code has to be run first in order to generate these files. The embeddings for the passages coming from the train data are saved under the file name "passages" (a feather file), and the embeddings for the passages from the validation data are saved under the file name "passages_v".

Once the average embeddings for both the queries and the passages were calculated a cosine similarity score between the average query and average passage embeddings was computed as follows:

$$sim(q,d) = cos(\vec{v_q}, \vec{v_d}) = \frac{\vec{v_q}^T \vec{v_d}}{\|\vec{v_q}\|\|\vec{v_d}\|}$$

where $\vec{v_q}$ is the vector of the average query embedding and $\vec{v_d}$ is the average of the passage embedding.

The cosine similarity is then used as an input feature to the logistics regression model.

The logistic regression model used is defined as:

$$f_w(x) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \in [0,1]$$

where $\boldsymbol{w}$ is the weight vector associated with each feature, $\boldsymbol{x}$ is the input vector which in this case is the vector that contains the cosine similarity score, and b is the bias term. The sigmoid function is the defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

The loss function in this model is the binary cross entropy loss defined as follows:

$$L(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m} \boldsymbol{y} log(\hat{\boldsymbol{y}}) + (1-\boldsymbol{y}) log(1-\hat{\boldsymbol{y}})$$

where $\boldsymbol{y}$ is the vector containing the true relevance labels and $\hat{\boldsymbol{y}}$ is the vector that contain the predicted relevance labels obtained using the logistics regression function above defined as $f_w(x)$.

The weight and the bias are updated using a gradient decent method as follows:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

where θ is the parameter, which in this case is the weight or the bias, η is the learning rate, ∇L is the gradient of the loss function.

The model is trained on the entire train_data.tsv file. The model was trained for 5000 epochs on 5 different learning rates, particularly for the following learning rates: 0.001,0.01,0.1,0.8,1.5

The loss against the number of epoch for each of these learning rates can be seen in Figure 1.

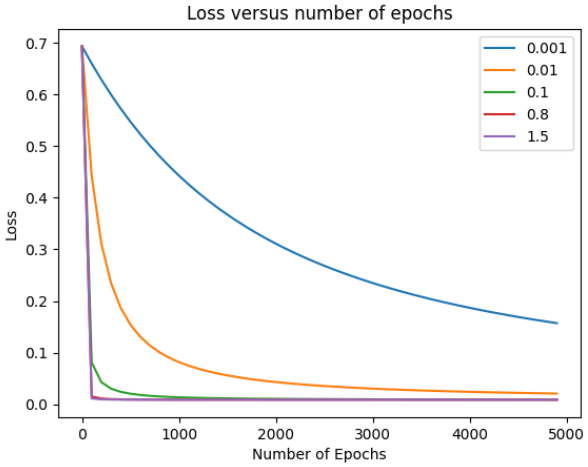

Figure 1: Loss versus epochs for 5 different learning rates

The learning rate that was chosen to train the final model is 0.1.

After the model is trained, it is tested on the entire validation_data.tsv. After applying the model on the validation_data.tsv the obtained mAP at rank 100 is 0.00939 and the nDCG is 0.02087. As observed, this model performs significantly worse compared to the BM25 model in task 1.

Some possible reasons that could lead to a poor performance of this model are the following:

- the data is highly unbalanced with many more irrelevant passages compared to relevant ones. A technique of sampling that would provide a balanced sample may improve the performance of this model.
- Logistics regression is a linear classifier and as such assumes linear relationship which may not be the case with the data presented.

The final logistics regression model was applied to the candidate_passages_top1000.tsv file to re-rank the passages. For each query, the top 100 passages were retrieved and this file is saved under the name 'LR.txt'.

4 TASK 3 - LAMBDAMART MODEL

As part of Task 3, the LambdaMART model was implemented. The model was trained on the train_data.tsv file and was tested and evaluated on the validation_data.tsv file. To train and test this model the full train and the full validation set were used in this task.

The train and validation datasets were preprocessed in the same way as in Task 2. For each query and each passage, the average embedding was computer using Google's pretrained word2vec model. After the average embeddings were computed, a cosine similarity score between the average query and average passage embedding was calculated.

The cosine similarity score between the query and the passage embedding is used as an input feature into the LambdaMART model.

The LambdaMART model has a pairwise objective and as such it compares how relevant a certain document is relative to another. In this way, documents or passages, are ranked based on their relative relevance to each other.

More formally, the pairwise objective can be expressed as follows:

Given a set of $< q, d_i, d_j >$, predict the more relevant document.

The feature vectors for q, d_i and q, d_j can be expressed as $\vec{x_i} and \vec{x_j}$ respectively

and the model scores can be represented as $s_i = f(\vec{x_i})$ and $s_j = f(\vec{x_j})$ respectively

The pairwise loss function then becomes:

$$L_{pairwise} = \phi(s_i - s_j)$$

The LambdaMart model is based on the pairwise RankNet model which minimizes the above mentioned pairwise objective. The LamdaMART model uses the idea of boosting to achieve this.

The LambdaMART model was implemented using the XGBoost gradient boosting library. To specify the LamdaMART model, XGBRanker was used, and the objective parameter was set to 'rank:pairwise'.

Hyper-parameter tuning was performed on 'gamma', 'max_depth', 'subsample', and the 'learning rate' hyperparameters. The hyperparameter search was conducted using Bayesian optimization. This approach was preferred over grid search and randomized search since it allowed for the evaluation of the model at each step and based on past information determined which parameters to select next to build the new model. The library used to do this is hyperopt. As part of this optimization, a set of ranges for each parameter was chosen and an objective function to be minimized was defined. In this case, the objective function was defined to minimize the nDCG value. The optimization was run for 100 trials.

The best hyperparameters based on this search were:

- gamma: 29.587
- max_depth: 1
- subsample: 0.742
- learning_rate: 1.06
- n_estimators: 110

where gamma is the minimum loss reduction required to make a further partition on a leaf node of the three, max_depth is the maximum depth of the tree, subsample is the ratio of the training instances, and n_estimators is the number of gradient boosted trees.

After the best hyperparametrs were determined, the LambdaMART was trained using these parameters. Once the model was trained, it was then applied on the validation dataset. The obtained mAP at rank 100 for this model is 0.165 and the obtain nDCG at rank 100 for this model is 0.169.

The LamdaMART model performs worse compared to the BM25 model. One reason for this could be that the LamdaMART model requires careful feature engineering. To improve this model, more features could be introduced as part of the model. In this case, cosine similarity was used as an input feature. Perhaps a different feature, or a collection of features could provide better performance of the model.

The LamdaMART model, however, performs better in comparison to the logistics regression model. A reason for this could be that the LamdaMART model is a model which is specifically designed for learning to rank, as opposed to the logistics regression which is a more general algorithm. Additionally, LamdaMART, as a tree-based model, is better at capturing non-linear relationships.

Finally, the LamdaMART model was used to rerank the passages in the candidate_passages_top1000.tsv file. For each query, the top 100 passages were retrieved and this file is saved under the name 'LM.txt'.

5 TASK 4 - NEURAL NETWORK MODEL

As part of task 4 an LSTM neural network architecture was implemented. For this task a subset of the train dataset was used for training. The sampling was done such that for each query, for each of its relevant passages, 5 irrelevant passages were sampled.

After the trained dataset was sampled, the query and passages were tokenized using the same pre-processing steps as described in the previous tasks. Same as in the previous tasks, average embeddings for both the query and passages were calculated using Google's pretrained word2vec model. Following this step, the average query and average passage embeddings were concatenated. Since the Google's pretrained word2vec model generates emebddings in the form of a vector of length 300, the average query and the average passage embeddings are vectors of length 300 each. Once concatenated, the final feature input becomes a vector of length 600. The input to the model as such is a $n * m$ vector where n is the number of rows/data points, and m is the number of features which in this case is 600.

The neural network architecture implemented in this part is an LSTM layer followed by a Hidden Layer which is then followed by a Fully Connected layer. The architecure can be viewed in Figure 2. As observed at the bottom of the figure, initially the query and passages are represented as text values. After this step, preprocessing is done, and embeddings are generated such that average query and average passage emeddings are calculated. Since Google's pretrained word2vec model was used, the query and the passage embeddings are each vectors of length 300. After this step, the query and passage average embeddings are concatenated to obtain a single vector of 600. This is the vector which is then inputted into the model.

The LSTM takes an input of size 600 and transforms it to the hidden layer size of 128. Following the hidden layer, the next and final layer is the fully connected layer which takes in the size of 128 and outputs a final score of size 1.

The loss minimized is the mean squared error loss defined as:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where θ represent the parameters, y_i the true values and $\hat{y}_i$ the predicted values.

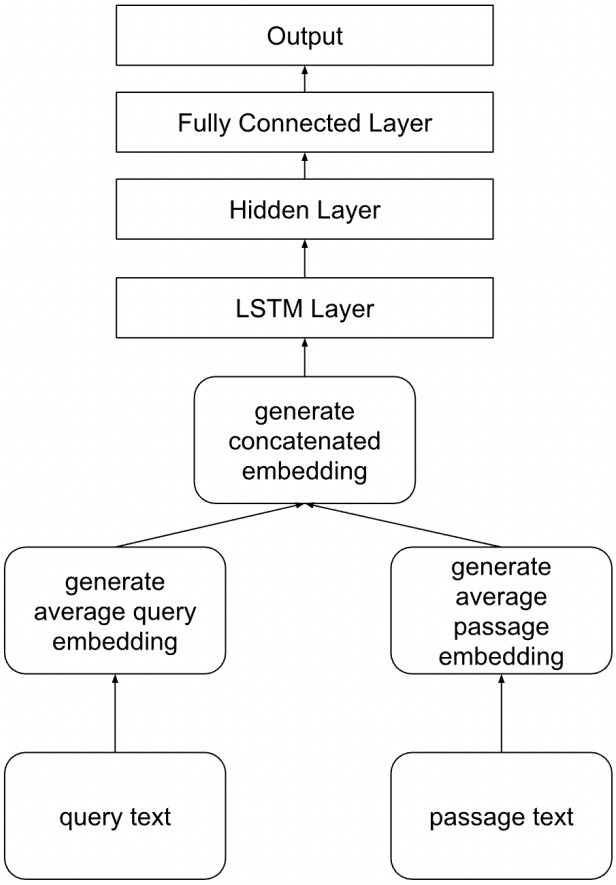

Figure 2: NN Architecture

The model was trained over 100 epochs and a learning rate of 0.001. These values were determined empirically.

The LSTM model was chosen due to its ability to recognise patterns and analyze data sequentially. It is a model that is capable of memorizing and keeping relevant information, while discarding irrelevant information.The LSTM model is good at capturing the meaning in a sentence. Finally, LSTM is a model that can capture complex relationship between the inputs and the output format.

Once trained the model was applied on the entire validation dataset. The obtained mAP at rank 100 for this model is 0.026 and the obtain nDCG at rank 100 for this model is 0.074.

The model performs better than the logistics regression, which could be expected given that the LSTM is a model which can capture more complex textual representations as opposed to the logistics regression model which can only capture linear relationships and is not a model designed for handling textual data.

On the other hand, this model performs worse compared to the BM25 and the LamdaMART models. There are several reasons as to why this might be the case:

- Feature Engineering - the input features of this model could have been chosen better. For example, instead of averaging out the query and document ebeddings, the entire emeddings could have been passed as an input which would have helped the model learn more complex relationships. This is an approach Palangi et al. adopted in order to generate deep sentece embeddings using LSTM-RNN model. This model sequentially takes each word, extracts the informatio, and represents it into vector space. [2]
- Objective Function - the model could be posed with a pairwise objective, and the features could be passed as pairs of documents for a given query in order to asses the relative relevance of the documents. Nogueira and Cho have implemented such a pairwise function for passage re-ranking using BERT and have achieved a state of the art model. [1]
- Hyperparameter tuning - a more extensive hyperparameter tuning could be performed to improve the model

Finally, the LamdaMART model was used to rerank the passages in the candidate_passages_top1000.tsv file. For each query, the top 100 passages were retrieved and this file is saved under the name 'NN.txt'.

REFERENCES

[1] Rodrigo Nogueira and Kyunghyun Cho. 2020. PASSAGE RE-RANKING WITH BERT. *arhiv* (April 2020). https://doi.org/10.48550/arXiv.1901.040853

[2] Hamid Palangi; Li Deng; Yelong Shen; Jianfeng Gao; Xiaodong He; Jianshu Chen; Xinying Song; Rabab Ward. 2016. Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval. *IEEE* 24, 4 (April 2016), 694 – 707. https://doi.org/10.1109/TASLP.2016.2520371