IRDM CW2 Report

1. Under Sampling

There were 2 main problems with the dataset to complete this coursework.

Firstly, the given datasets is that there are too many number of datasets. The training dataset has around 4 000 000 number of data and the validation dataset has approximately 1 000 000 of data. This causes a problem when preprocessing the data and calculating the embeddings due to their computation time.

Secondly, , the dataset is extremely imbalanced ; there are 4 359 542 non-relelvant and 4797 relevant passages in the dataset. This would make a trained model undergo overfitting and biased. (more reasons should be here)

Therefore, I under-sampled both train and validation data so that both preprocessing and the embedding can be done in a feasible time.

In order to down-sample the dataset, I selected all relevant and 100 randomly chosen non-relevant passages for each query. I applied this method to both the training and validation dataset. This would reduce the dataset appropriately while maintaining all quries. This would also mitigate the imbalance aspect of the datasets.

These down-smapled train and validation datasets were used throughout the coursework from Subtask 1 to Subtask 4.

1. Evaluating Retrieval Quality

I implemented 2 metrics for evaluating a ranked list: 1. Average precision@k, 2. NDCG.

As the question asks, I acquired a ranked list from the validation data given using BM25 and computed the performance BM25.

1. Average precision@k

I implemented this metric in a fucntion called ‘average\_precision\_cal’ which takes a ranked list as a parameter.

I calculated the average precision of the ranked list by looping over the ranked list. During the loop, if a passage is a relevant passage, I calculate the precision of the list at that stage and add to a variable called ‘precision\_sum’. For example, if the third passage is a relevant passage and it has had 1 relevant passage on its previous steps, the precision at this point would be 2/3 and this will be added to ‘precision\_sum’.

When the loop is over, the average precision is calculated by dividing ‘precision\_sum’ by the length of the ranked list.

As for k, I used k = 100.

To calculate the mean average precision for all the queries, I computed the average precision@100 of a ranked list for each query, added them up, and divided it by the number of queries.

1. NDCG

I implemented NDCG in a fucntion called ‘get\_NDCG’ which takes a ranked list as a parameter.

Firstly, I implemented a function which calculates the IDCG. To get IDCG value, I created an ideal ranked list from the ranked list given as a parameter by sorting the list by the ‘relevancy’ column. Then, by looping over the sorted list and using the equation below(Figure 1), I found the value of IDCG.

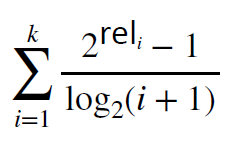


Figure 1

As for DCG, like what I did to get IDCG, I looped over the ranked list given as the parameter and used the same equation to get DCG value. After the loop, I divided DCG value by IDCG value to get NDCG value.

As what I did to compute the mean average precision, I calculated NDCG of a ranked list for each query added them up, and divided it by the number of queries to get mean NDCG.

1. Logistic Regression (LR)

For logistic regression, I created 3 features: 1. Cosine similarity(dense vectors) 2. BM25 3. Wind Mover’s Distance.

1. Cosine Similarity of Dense Vectors

I chose this feature because it is an effective metrics to determine if two documents are similar especially if the vectors are pretrained embeddings such as Glove. The Euclidean distance could be used to compare 2 vectors mathematically, but cosine similarity is far more effective in NLP.

First, I converted all the queries and the passages into a dense vector representation using Glove 300D pre-train embedding. My query vector has a shape of (64, 300), where 64 is the length of a query including the padding and 300 is the dimension of each word. Similarly, my passage vector has a shape of (150, 300) where 150 is the length of a passage including the padding and 300 is the dimension of each word. Then, by computing the average vector of each query and passage using np.mean, each query and passage can be represented by a single vector. Cosine similarity of a pair of a query and a passage can be found by applying these 2 average vectors to the cosine similarity formula.

I implemented the cosine similarity formula in a function called ‘cosine\_sim\_formula.

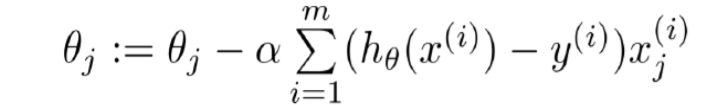
1. BM25

I used BM25 as one of the features because BM25 is one of the most commonly used ranking metrics in information retrieval field. The implementation is the same as the one I used in Subtask 1.

1. Word Mover’s Distance

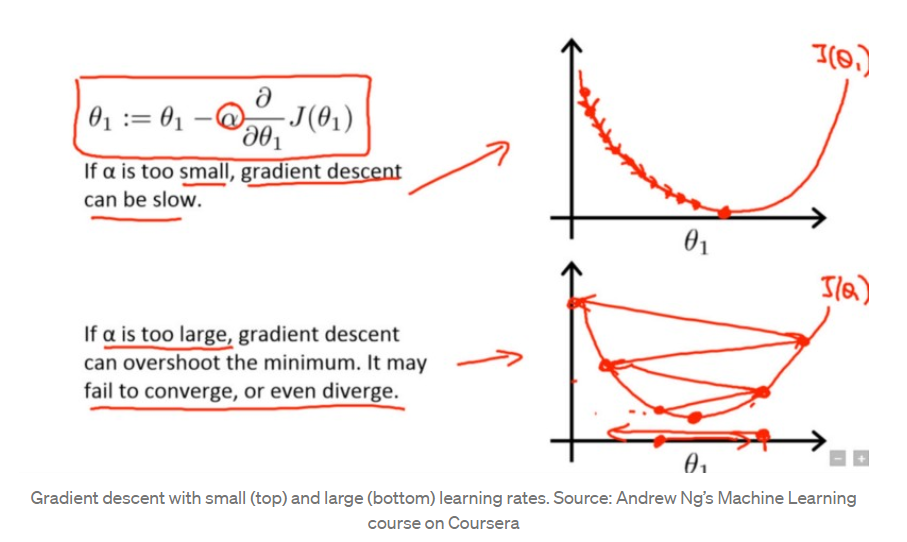
This metric is able to detect similarities between sentences that have different words but semantically have similar meaning using the word embedding space. Therefore, I believed that it would be an appropriate feature for a ranking model. I used ‘model. Wmdistance’ function from ‘gensim’ library to compute this metrics.

To implement the logistic regression, I defined a class called ‘LogisticRegression’ in which a method called ‘fit’ is responsible for training the model and adjusting the weights accordingly. Since my data has 3 features as I described above, the weights of this model has 3 weights and they are initially initialized randomly using the normal distribution and the bias is initialized to 0. Furthermore, I defined the sigmoid function as the prediction function and the cross-entropy function as the loss function of the model. The model is trained using full-batch gradient descent which means that we use the entire dataset to update our weights in each epoch. Each parameter(weights and bias) are updated using the following logistic regression derivative function.



In addition, for each epoch, I shuffled the train dataset in order to acquire better results. This shuffling is done by a function called ‘shuffleIdx’.

The learning rate is a hyper-parameter that controls the step size at each iteration while moving towards a minimum of a loss function. This means that the training loss will converge slowly if the learning rate is small., but it may not converge or even diverge if the learning rate is too big. The following image explains the phenomenon when the learning rate is too small and too large.



1. **LambdaMart Model (LM)**

Firstly, in order to implement LambdaMArt model with Xgboost library, the input data has to be converted into a specific form which the library accepts. The queries in the train data has to be clustered within the dataset and this converted dataset has to be passed to the model with a list that contains a number of passages per query. Therefore, the train set is sorted by the ‘qid’ column so that all queries are clustered. Then I counted the number of passages per query and save it in a list. The same steps are applied to validation dataset as well for evaluations. These inputs are passed to the model to train. As for the ‘objective’ parameter, I used ‘rank:ndcg’ as this is the evaluation function for the pairwise LambdaMart model. As for hyperparameters such as the number of boosted round, I used grid search method to optimize the model and get the best hyper-parameter values.

As for the features, I used the same 3 features that I used in the logistic regression (Cosine Similarity, BM25, Wind Mover’s Distance).

1. **Neural Network Model (NN)**

In q4, we expect you to justify your implementation, e.g. what you implement? why you implemented your architecture over different ones? What are the pros/cons of your architecture? We are not asking you to implement specific architecture, but we want you to discover useful models from existing literature.

For this task, I implemented a pairwise Ranknet model using the Tensorflow/Keras library.

Because this is a pairwise ranking model, I performed an input processing. For each query in the train dataset, 2 corresponding passages were extracted and their feature values are stored separately in a list (xi, xj). Then their probability(label) is calculated depending on their passage score(the relevancy value in my case). Let us assume that for a query, there is a pair of passages A and B. If the passage score of A is larger than that of B, then the probability is 1, if it is the opposite situation, then the probability is 0. Lastly, if the scores are the same, then the probability is 0.5.

**References**

https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10