COMP0084 Coursework 2

ACM Reference Format:
. 2018. COMP0084 Coursework 2. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

0.1 Data

Please note that for all of the tasks we use a sampled subset of the training data. It consists of 25% of the data points sampled randomly.

1 TASK 1

In this task we write code which is used to evaluate algorithms designed to rank a group of passages to a certain query. In particular, we write code to calculate the Normalized Discounted Cumulative Gain (NDGC) and the Average Precision.

1.1 Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain measures the gain of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom, with the gain of each result discounted at lower ranks. The cumulative gain is calculated until a certain position. In this task we calculate it until the 100-th position.

The way we calculate NDGC is via this formula:

$$\text{NDGC}_p = \frac{\text{DGC}_p}{\text{IDGC}_p} = \sum_{i=1}^{p} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)} \frac{1}{\sum_{i=1}^{|\text{REL}_p|} \frac{\text{rel}_i}{\log_2(i+1)}}$$

where IDGC is the ideal DCG, $|\text{REL}_p|$ represents the list of relevant documents (ordered by their relevance) in the corpus up to position p, and rel is the relevancy score. Since the relevancy score takes values 0 or 1, we sometimes get an IDGC equal to 0, causing division by zero. In order to mitigate that, we add 0.1 to IDGC.

The above is a formula for one query, and since we're interested in evaluating the algorithm's performance on the whole dataset, we calculate the NDCG for each query and take the average to get the final result.

1.2 Average Precision

We calculate the average precision via the following formula:

$$\frac{\sum_{k=1}^{n} P(k) \times (k)}{\#\text{ relevant documents}}$$

where $P(k)$ is the precision at cut-off k in the list and $rel(k)$ is an indicator function equaling 1 if the item at rank k isa relevant document, zero otherwise.

As we did with NDGC, we average over all queries to get the final result.

In this task we evaluated the performance of the BM25 algorithm. We got Average precision: 0.004104084110119765 and NDGC_{100}: 0.08548352967618224.

2 TASK 2

In this task we are going to implement a logistic regression model to predict the relevancy of a passage to a given query. We are first going to pre-process the query and passages using the Word2Vec alogrithm, secondly we are going to calculate the cosine similarity of all the query and passage pairs, and lastly, based on these scores, we are going to predict the relevancy, which is a binary variable taking values from 0 to 1.

2.1 Word2Vec

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. It transforms each word into a vector of length n. In our case $n = 32$. We have tested the performance of the logistic regression model with vectors of size $10, 20, 30, 40, 50, \ldots, 400$ and found that the best performance was among the number from 20 to 30, so then we narrowed down the subset of lengths to be $20, 22, 26, \ldots, 40$ and found that 32 performed best. We chose the Word2Vec algorithm since it produces vectors which, when plugged into a function such as the cosine similarity function (as a pair), we can infer the semantic similarity between those 2 pairs of vectors.

2.2 Inputs of the model

After making a Word2Vec model and getting the vector representations of each word of dimension 32, we made the query and passages to be represented as vectors themselves by taking the average of the element-wise sum of the words (vectors) which the query and passages contain. Since for each query passage pair we want to predict the relevancy score, and since each pair represents a a pair of vectors, we decided to calculate the cosine similarity score of each such pair and use it as an input for the logistic regression model. Alternative ways to represent the inputs of the model is to again take the average of the element-wise sum of the pair of vectors and have inputs of dimension 32. Another way is to concatenate the 2 vectors and the input dimension would be 64. We made the decision to go with the cosine similarity score as the input since the Word2Vec model allows us to do so.

2.3 Logistic Regression model

Our logistic regression model takes as input a number $c \in [-1, 1]$ and produces an output a number $p = \frac{1}{1+\beta_0+\beta_1 c} \in [0, 1]$, which represents the probability of the given pair being relevant, i.e. $p = \Pr(\text{passage is relevant to the given query})$. We learn the parameters using gradient descent on the loss function, which is

$\sum_{i=1}^{m}(y_i - p_i)^2$. In one epoch we calculate the predictions using the current parameters, then we calculate the gradient of the loss function with respect to the parameters, and we subtract the respective gradient from the parameters. This update happens 5 times as we do 5 epochs. Note that the number of epochs is chosen to be 5 since after 5 epochs the relative performance plateaus. Afer getting the parameters, we predict on the unseen data and calculate the relevant metrics.

2.4 Ranking and results

After predicting the relevancy score for each query and passage pair, we rank all of the passages within a query group (all passages which have the same query) by the predicted score in descending order and calculate the NDGC_{100} and average precision. The results are: NDGC_{100} = 0.019388443026138028 and average precision = 0.0026570115421879017. We changed the learning rate to take values in the set $\{0.001, 0.01, 0.1\}$ and the results did not change, as the parameters converged after the first few epochs at every value of the learning rate.

3 TASK 3

In the third task we implement the LambdaMart ranking algorithm for re-ranking passages from the XGBoost library. The algorithm takes in as input features and a ranking score, in our case it is the relevancy. Within a query group it then compares the performance of a specific ranking if a certain pair of passages would exchange places, and it does so for each pair. It is essentially an algorithm where ranking is transformed into a pairwise classification problem.

The features we use are the cosine similarity scores of each query and passage pair. We derive those scores the same way we derived them in task 2, using the Word2Vec vector representations of the queries and passages. In order to get the best results we did a grid search over the parameters. The parameters we searched over are: step-size shrinkage ‘eta‘, learning rate ‘learning_rate‘, the subsample ratio of columns when constructing each tree ‘max_depth‘, Maximum depth of a tree ‘max_depth‘ and the number of trees ‘n_estimators‘. First we search over a wide range of estimators and look for the combination which yields the best performance, which we measure by the average of the average precision and NDGC metrics. The intervals we search for are the following: ‘n_estimator‘ $\in \{40, 50, 60, 70, 80, 90, 100, 120, 130, 140, 150, 160\}$, ‘colsample_bytree‘ $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, ‘eta‘ $\in \{0.01, 0.05, ..., 0.95\}$, ‘max_depth‘ $\in \{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$ and ‘learning_rate‘ $\in \{0.001, 0.01, 0.1, 0.5\}$. We get that the intervals for which the best results are obtained are the following: ‘n_estimator‘ $\in [60, 70]$, ‘colsample_bytree‘ $\in [0.8, 0.9]$, ‘eta‘ $\in [0.25, 0.35]$, ‘max_depth‘ $\in [12, 14, 16]$ and ‘learning_rate‘ = 0.01. We did a secondary search over these new intervals with smaller step sizes between each value and found that the best results are obtained when ‘n_estimator‘ = 64, ‘colsample_bytree‘ = 0.86, ‘eta‘ = 0.27, ‘max_depth‘ = [14] and ‘learning_rate‘ = 0.01.

The results we got are: average precision = 0.0010701885637380108, NDGC = 0.05951808706416718.

4 TASK 4

In this task we try to rank passages using a neural network to predict the relevancy of each passage to a certain query. We use the Tensorflow framework. The inputs are again the cosine similarity scores. The network we use to predict the relevancy cosists of 3 layers - 1 input, 1 hidden and 1 output layer.

After predicting the relevancy score for each query and passage pair, we rank all of the passages within a query group (all passages which have the same query) by the predicted score in descending order and calculate the NDGC_{100} and average precision. The results are: NDGC_{100} = 0.010767578150323599 and average precision = 0.002604688285073202.