COMP0084_ICA2

Information Retrieval and Data Mining

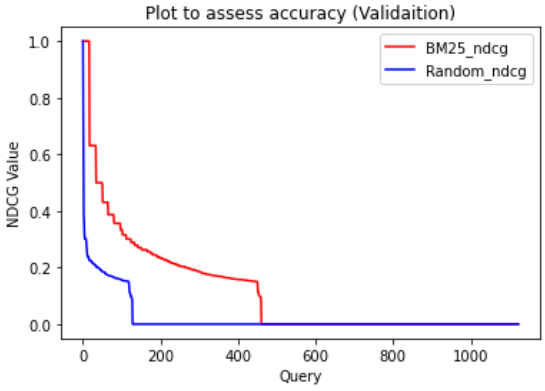
Student number: 20101171
Department: Statistical Science
UCL
ucakdli@ac.uk.com

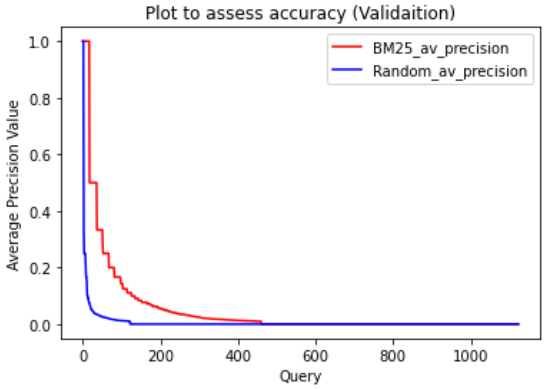
EVALUATING RETRIEVAL QUALITY

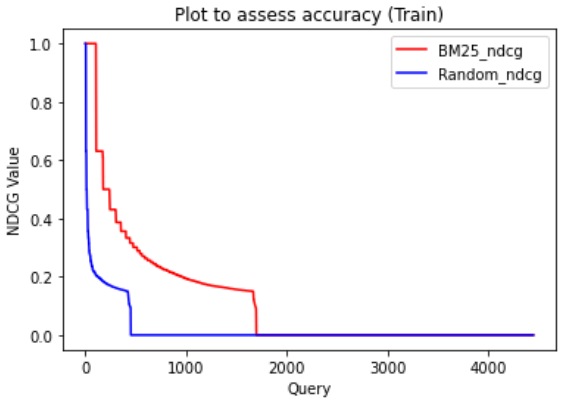
In the ICA 1, BM25, as a kind of retrieval method was built and it is implemented on the two datasets (train dataset & validation dataset) of ICA 2. The text preprocessing method before implementing the BM25 is still same as before, which contains deleting stopwords, stemming and so on. To evaluate the result, average precision and normalized discounted cumulative Gain (NDCG) metrics are used and precision is at rank 100, considering the document sizes are mainly about 1000 and the following subtasks are to rank top 100. It could be observed that in some queries the document size is lower than 100 so that in such cases accuracy is not measured as the relevant term could be ranked into top 100 no matter using which model. Also, there are still some queries having document size between 100 and 1000, they are not processed and are still measured here as the number of them and their influence is small.

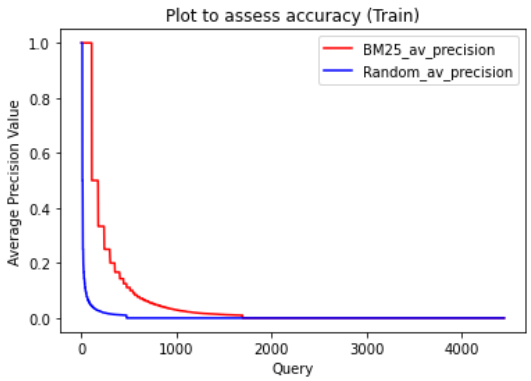
To assess the quality of retrieval methods needs to compare their average precision (AP) / NDCG value. However, using the statistical summary and doing hypothesis test is not chosen here as the simple statistical summary like mean would lose too much information and the hypothesis tests like t-test are not intuitive enough. Therefore, the way here for comparison is to plot the AP/NDCG value of different queries generated by different methods in descending order. The used dataset, the kind of accuracy measurement and retrieval methods are noted in title, y label and legend, respectively.

To assess the behavior of BM25 retrieval method, the way is to set a 'null group' which just ranks the documents in each query randomly this method to rank is useless indeed. The four plots to compare BM25 with the 'null group' based on AP/NDCG in validation/train datasets are shown below.









From the above figures, it could be seen that BM25 could select relevant items in more queries while the 'null group' could merely achieve that in about 1/10 queries, which just follows the statistical expectation (in most queries, only one document is relevant). Also, in the queries whose relevant item could be selected, the AP/NDCG value of BM25 is much higher. As a result, such finding supports BM25 in ICA1 to be an actual useful retrieval method. In addition, the shape of lines between plots using validation and train datasets adopting the same accuracy measurement looks similarly, which suggests the validation data and train data are collected from the same resource.

Last thing to mention is that ranking passages in validation dataset by BM25 does not use any information from train dataset. Therefore, the result of BM25 is useful as a comparison in the following subtasks whose models are trained by train dataset.

LOGISTIC REGRESSION

(Ahead of further interpretation, it is necessary to mention that the legend "LM_av_precision" of the plots in this section should be "LR_av_precision" actually. And "trained by validation" is also a mistake. It should be "Trained by balanced data". Just a typing error.)

To represent words in the query/document, word2vect as a kind of embedding method is selected. Although ELMO was regarded as the best word embedding method in 2018, the reason why it is not chosen here is that in ELMO, the vector representation of a word is based on the before/after content. But during averaging the word embeddings, that information may either be messed up (bad result) or still be well reserved (good result). As things are unpredictable, ELMO is not used here. Compared with FastText/Glove, word2vect is more typical and it has pre-trained package which could be directly used. So, it is chosen here.

The used word2vect here is pretrained by google[1] on part of Google News dataset (about 100 billion words). The word2vect model contains 300-dimensional vectors for 3 million words and phrases.

Before implementing the word embeddings, texts need to be pre-processed while the processing is quite different from ICA1. The plan is based on a website in Kaggle [2], which states two golden principles: Don't use standard preprocessing steps like stemming or stopword removal when you have pre-trained embeddings (The second is omitted here). The author suggests that without using stopword, stemming or tf-idf for feature extraction could help loose value information, which would be helpful for NN to figure things out. Even if he mentioned so, some basic issues are still considered:

1.The below picture shows the top5 similar words of swimming, which supports the feasibility not to do stemming as without doing stemming could still have similar word vector input.

```
In [226]: model_word2vec.most_similar('swimming', topn=5)
Out[226]:
[('swim', 0.8685395121574402),
 ('Swimming', 0.759138286113739),
 ('swimmers', 0.7531777024269104),
 ('swims', 0.7336981296539307),
 ('swimmer', 0.7183950543403625)]
```

2.And during investigating the stopword, it is found that cosine similarity between " 're " and "are" is much smaller than the "swimming" result above, which suggests to replace " 're " with "are". And this kind of processing is also implemented on " 'll" , " 'd " and " 've".

3.The capital letter is not turned into small letter as, for example, china and China are two different things.

After finishing the text processing, words are represented by vectors which are then averaged to represent the query/document (the problem here about simply averaging would be discussed later). To input the query vector and the passage vector into the logistic regression model, they are just concatenated together, which means, for example, the size of vector in google pretrained model is 300 and the size of the concatenated vector would be 600. The reason to do so is to do the additive interaction between two vectors and let logistic regression to decide the parameters. In addition, BM25 information is also added into that vector. Finally, a column filled with value 1 is added into feature variable matrix, which is bias term indeed. Another thing needs to mention is that the logistic regression model is run in torch with CPU (GPU could not be used due to the limited memory, about 6GB).

However, the constructed model merely follows the change in BM25 and after deleting the BM25 information, the accuracy of this model is just similar to the random ranking mentioned before, which suggests the model is useless. Reasons are considered and are listed in the below:

1.It is noticed that there are several sentences in the passage. During averaging the embeddings, too many useless words are added together and generate big noise. One method may be helpful is called smooth inverse frequency[3]. It is to calculate the PCA of the matrix formed by multiple sentence vectors and then eliminate the first principle for each sentence vector in order to reserve the own feature of this sentence. And this method was tested to be useful in matching condition.

2.Similarly to the first reason, compared with loosing the value information (mentioned in the "golden principle"), without deleting stopwords or doing stemming actually covers the important information.

3. The word2vect model used is pretrained, whose topic is different from this task. As a result, the word2vect should be trained specifically based on the content of this task.

4.Using word2vector here is not good enough as it could not reflect the meaning of sentence in terms of order after averaging the embeddings. For instance, "dog likes cat" would have the same output with "cat likes dog" here. As a result, "Bert" as the embedding may be better here.

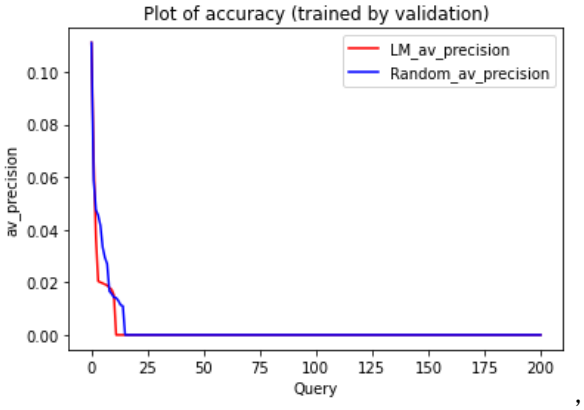
5.The training dataset is not balanced as there are much more irrelevant terms (relevancy=0) than relevant terms (relevancy=1).

6.The vectors concatenated together as the input may not fit the logistic regression. Instead, some other forms like cos-similarity may be a better input.

The above reasons may not only have additive effects but also interaction influence. To test the above reasons, a small sample is used, which is just the validation dataset. The new train set contains 947 unique queries while the new validation set contains 201 unique queries. As it is mentioned before that the character of validation dataset is coincident with the train dataset, this sample is reasonable to be used. And in this part, some codes could not be directly used because of format issue and so on. Reserving them is to show the evidence of trials.

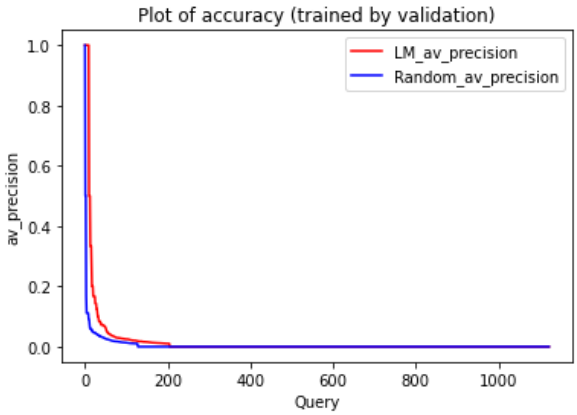
The first reason is not tested as it is out of my ability and the fourth is not tested as it had better focus on one embedding method and dig out the key reason why this method fails.

For the second reason, such procedures like stemming are added again but it has tiny influence to the final result. For the third reason, word2vect is constructed using the content of this dataset and setting vector size for a word to be 50, window to be 5 and words whose frequency are smaller than 1 not to be counted. Still, nothing changes. For the sixth reason, instead of the concatenated vector, cos-similarity is inputted into the model as the feature. But, result is still bad. The below is an example about the accuracy plot when the model fails.

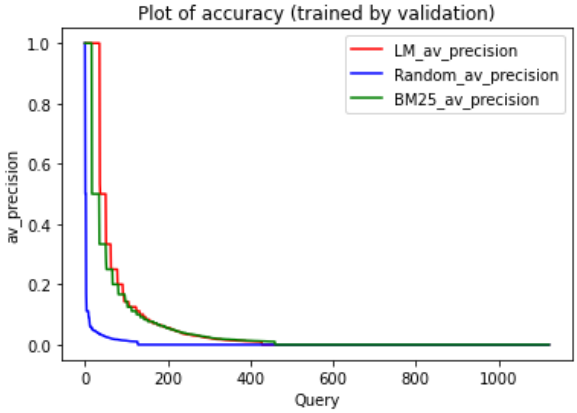


,

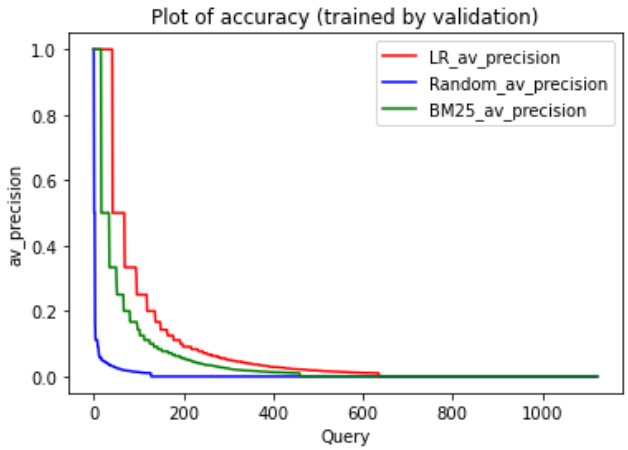
However, for the fifth reason, after the train set is balanced (number of terms with relevancy equal to 0 is set to be 3 times of number of terms with relevancy equal to 1), it seems that the result of this LM model with concatenated vector as input could be better than the result of random rank. As after balancing the data, the training set is quite small, this whole balanced training is regarded as the input rather than sample mentioned before. The plot is shown below.



Considering the mentioned sixth reason, cos-similarity is inputted only with the bias term. It seems the result is much better, which suggests the sixth reason is also sensible.

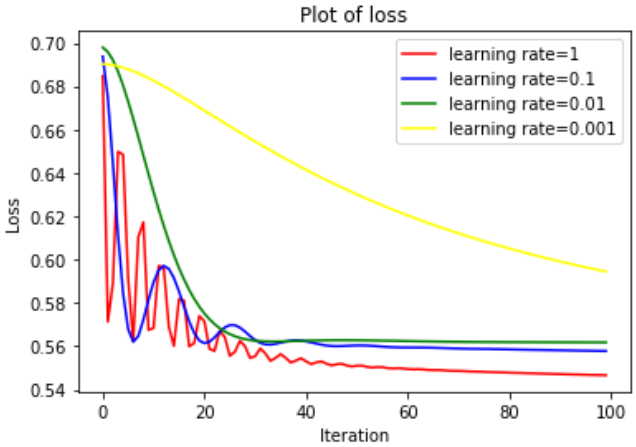


After adding the BM25 term, the result could still be improved:



In conclusion, the model with 3 feature variables (BM25, cos-similarity between query vector and document vector, bias) is treated as final model for best performance. And it seems that the unbalanced training set should be the key reason as the parameters of such model are mainly infected by the irrelevant terms due to the large quantity. Alternatively, predicting the relevant term correctly would not have much influence on loss function.

Finally, with regard to analyzing the effect of learning rate on the loss, as the logic is similar among different experiments, here is just an example of the model with concatenated vector and balanced train set as input. From the below plot, it could be seen that when learning rate is small, the training loss decays quite slow. When the learning rate is relatively large, the training loss would fluctuate but could still decay and converge in general. However, if the learning rate(lr) is too large (e.g. lr=10), the loss could not converge. It is plotted as it would make the plot messy. As a result, a moderate learning rate should be selected. (Need to mention that the accuracy of models constructed with different learning rate is not compared here as it is not required and not what we focus on this part.)



(P.S. As the question require with query/passage embeddings as input, if the cos-similarity version is not available, there is also a version named 'LR_1.txt' with BM25 and concatenated vector as input and could be regarded as the final result.)

LAMBDAMART MODEL

Firstly, the data set used should be clarified: Although there is argument insider the xgboost function to play more weight on term with relevancy equal to one, using the whole train set is limited by the RAM of laptop. As a result, the train set used in this part is same with part 2, which is a balanced train set. To adjust parameters, cross-validation using the balanced train set is not considered as test set is actually unbalanced. Parameters group behaves well during cross-validation may not still behaves well in the unbalanced test set. Instead, a small sample (containing about 150 unique queries) from the whole test set (containing about 1148 unique queries) is used to adjust the parameter. In addition, when outputting the final result, this validation dataset is contained as it is quite small compared with the test dataset. It is not strict here but just for convenience. If needed, it could be deleted.

Secondly, the feature variables inputted should be explained: As mentioned in logistic regression, BM25 and cos-similarity are considered as input in prior and it has been verified that, compared with the concatenated query/doc vector, cos-similarity could better represent the similarity even in LambdaMART model. However, using just two dimensional vectors to represent the whole query/doc embedding with 600 dimension may lose much information. As a result, the Euclidean distance is added to compensate the information about relationship between query vector and doc vector. Also, the number of words in the query or doc are also regarded as feature variable in order to reflect information about query/doc. One thing may be useful but is not tried here is, similar to the SIF mentioned in part2, using the second and third principle vector to represent the query/doc matrix and regarding them as feature variable.

Thirdly, the parameters tuned should be interpreted here: Considering the variance-bias tradeoff, for avoiding overfitting, using validation data set to adjust the parameters is quite important. The methodology used is grid search and the metric is still decided to be ndcg top 100. Ndcg is for consistency and top100 is because it is finally needed.

Four main parameters are considered here:

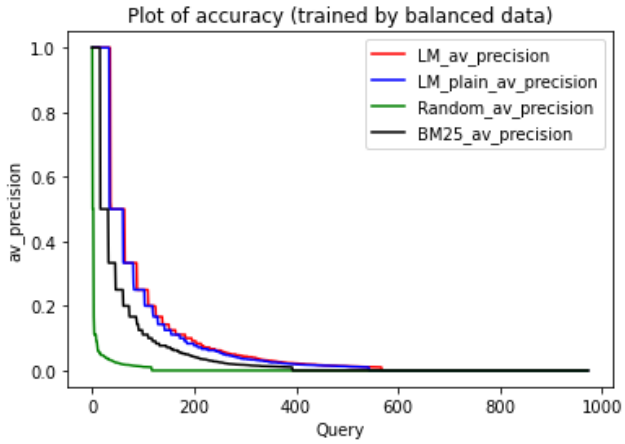
1. To select a best learning rate such that the loss function could converge, learning rate is set to be 0.1, 0.3(default) and 0.6 within the 100 iterations.
2. To decide the maximal complexity of the model, the maximal depth of trees is set to be 4, 6(default) and 8.

3. The Minimum loss reduction (regularization parameter) required to make a further partition on a leaf node of the tree, short as 'gamma', is set to be 0(default) 5, 10 and 20.

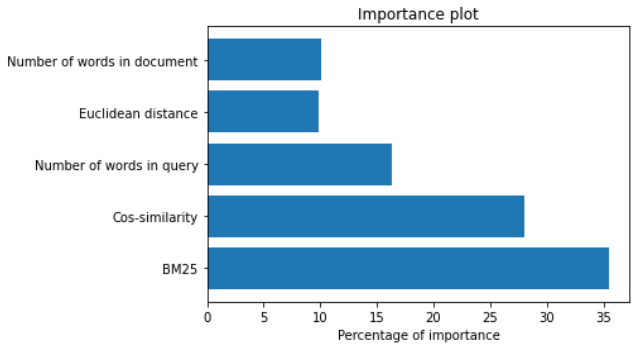
4. Minimum sum of instance weight (hessian) needed in a child used in tree partition step, short as minimal child weight, is set to be 0, 0.1, 1(default) and 5.

As a result, the number of parameter combinations is 144 and the optimal combination is: learning rate=0.3(default), maximal depth of trees=6(default), gamma=10 and minimal child weight=1(default). Note that none of the other parameters locate in the boundary of the selections, which suggest this parameter combination is reasonable in some degrees.

The below figure shows the average precision mentioned in part1. Validation dataset used for adjusting parameters is not plotted and ndcg plot is omitted here as it reflects the similar tendency. And the red line with legend "LM_av_precision" represents the final model while the blue line with legend "LM_plain_av_precision" represents the model with default parameter set. It could be seen that the final model behaves much better than the random rank and BM25 which indeed is an unsupervised method. In addition, although only 1 parameter is adjusted actually in the end, this final model with parameter tuned still behaves slightly better.



In the end, it is meaningful to check the importance of the feature variables. Using the average gain across all splits that the feature is used in as judgement, it shows the importance of BM25, cos-similarity, Euclidean distance, number of words in query and number of words in document in the following plot, respectively. As expected, BM25 with cos-similarity plays the most important role. Surprisingly, number of words in query seems to be more important and it suggests more information about query had better be digged out.



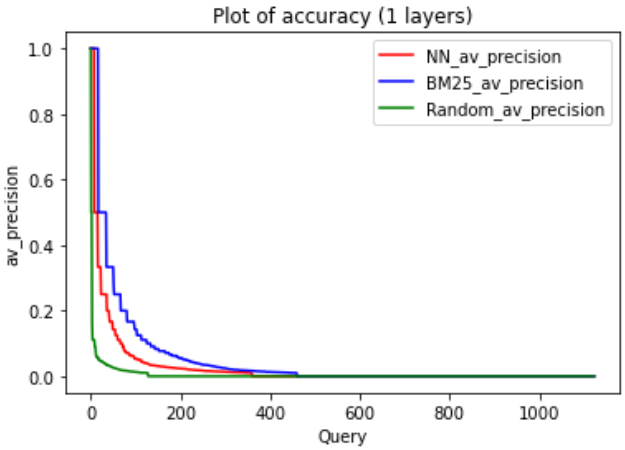
NEURAL NETWORK

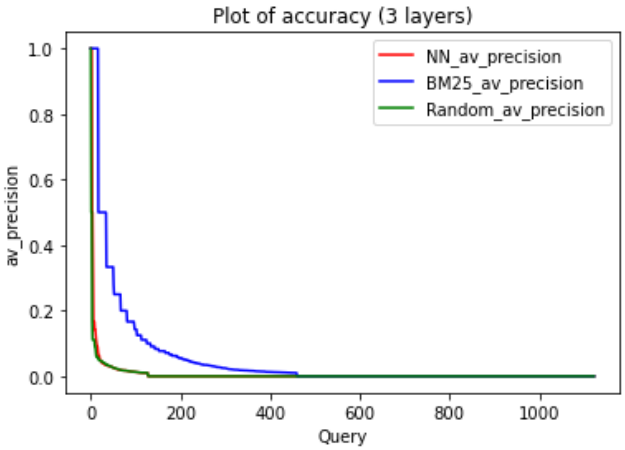
PART1:

To simulate the idea of Deep Structured Semantic Model (DSSM), two DNN models are applied to query embedding (word2vect from part2) and document embedding separately. Using DNN here is to discover hidden patterns of embeddings and find the intricate relationships among the different dimensions of the query/document embeddings. Due to the multiple layers, it could help conduct the non-linear separation. The outputs of two models are two vectors with multiple but same dimensions and the absolute cos-similarity of these two outputs is finally calculated. Then, to reduce the difference between the final cos-similarity and the target relevancy, the parameters in the two DNN keep update so that find the best one. The criteria of "difference" is actually the binary cross-entropy, which is suitable for classification problems as this case it's to classify relevant/irrelevant. Similar to before, the training data has been balanced which means the ratio of irrelevant terms to relevant terms is 3 under each query. In addition, this model may be not only useful for distinguishing the relevancy but also useful for reduce the dimension of word2vect (similar to PCA).

Inside the DNN models, there are several layers and the linear transformation between different layers. In addition, the dimension of query/doc embeddings is actually 300, which means there are enough space to do so. And in the first layer, normalization is applied and the bias term is added. In each layer, there is also activation function (relu and tanh are selected here) dropout is conducted to prevent overfitting.

However, the result is quite poor, it could be seen from the below plot that the accuracy decreases as the number of hidden layer increases. Even if much parameter adjustment has been tried, the result still has no much improvement. In addition, this decrease feature also represents the structure of this model is not useful.





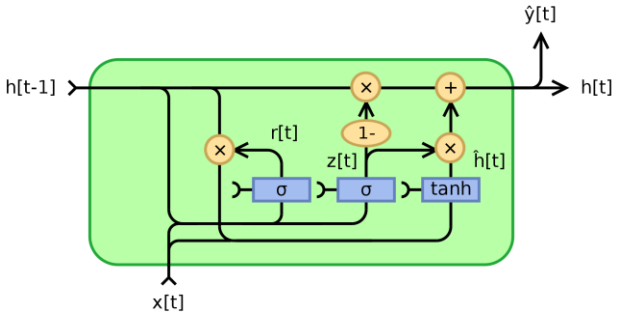
PART2:

To compensate for the failure in part1, the real DSSM is tried here in part2. Due to the complexity of the whole model, part of the codes is referred from github[4]. The reason for this model is mainly for digging about this field deeper.

In terms of the training data, due to the vast quantity of computation but the limited computing power of laptop, a small sample is selected and it is the same with one used in logistic model, which is just the validation dataset. Therefore, the new train set contains 947 unique queries while the new validation set contains 201 unique queries. As it is mentioned before that the character of validation dataset is coincident with the train dataset, this sample is reasonable to be used. And the training data is still balanced here.

The real DSSM used here has the similar structure to the model implemented in part1. Instead of using the word2vect embedding, bag of words is implemented here. It is to split the word in the sentence after the sentence is split. Then, each word is changed to its index of where it locates in the bag. Here, an existed dictionary about the bag is used. For example, the bag of words embedding of "what law repealed prohibition" is [2054, 2375, 21492, 13574].

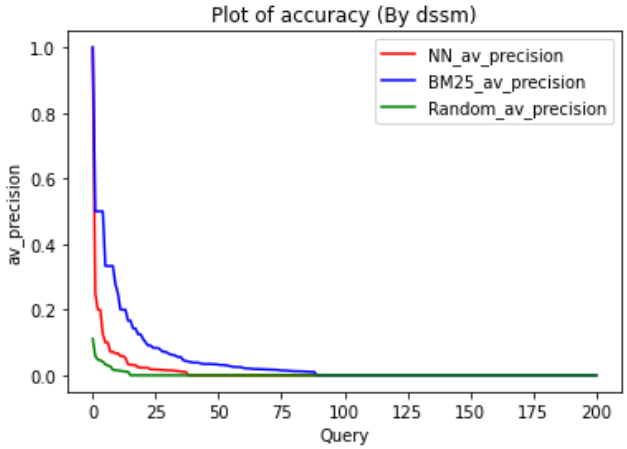
Instead of the DNN mentioned before, the neuro network to process the query/doc embedding is Gated Recurrent Units (GRU) which is a gating mechanism in recurrent neural networks. It is similar to the LSTM with a forget gate but with fewer parameters as it lacks an output gate. A plot could clearly show its structure here:



Instead of the cos-similarity mentioned before, the Manhattan distance is used here to pair the outputs from the GRU model as the first published paper about DSSM. It has the form:

$$Distance = \sum_{i=1}^{n} |x_i - y_i|$$

Due to the small quantity of the balanced training data set, the result is not ideal but still much more useful than the random rank (see below). And It reflects the architecture of the whole model is reasonable so that the predicted score of this one is regarded as the final output rather than the score in part1.



REFERENCES

[1] https://code.google.com/archive/p/word2vec/
[2] https://www.kaggle.com/christofhenkel/how-to-preprocessing-when-using-embeddings
[3] Arora, S., Liang, Y., & Ma, T. (2017). A SIMPLE BUT TOUGH-TO-BEAT BASELINE FOR SENTENCE EMBEDDINGS. ICLR.
[4] https://github.com/InsaneLife/dssm