COL764 Assignment 3

Raj Chanda (2021ME10983)

Modelling of the problem and loss functions

The problem is modelled as a regression problem – for every query_id and doc_id, there is a feature vector available, which is used to get the relevance score. Since the assignment required pointwise learning to rank approach, these relevance scores are then sorted in decreasing order to get the predicted order of documents.

The loss function was initially chosen to be Root Mean Square Error (RMSE). Inspiration was taken from the book Learning to Rank by TY Liu, which mentions the following result for pointwise learning to rank

$$(1 - nDCG) \leq K * \frac{\sqrt{\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2}}{n}, \quad K > 0$$

This result suggests that 1-nDCG will always be upper bounded by RMSE.
But after experimentation it was observed that RMSE scores were not good indicator for the behaviour of regression model, since the target values are either 0 or 1, with percentage of 0's > 99%.

At the end, nDCG@10 was used to judge the regression models for pointwise learning.

Loss Function used	1- nDCG@10

Hyperparameter selection

For Hyperparameter selection of the three models given, first I had to decide on the type of aggregation of nDCG scores to do, to uniformly judge the performance of the models. For this, I took the following approach -
Every fold contains some set of queries. nDCG@10 is calculated for every query of the fold, and average of the nDCG@10 scores are taken to aggregate the result for each fold. This will give an average nDCG score for every fold.
To get the best hyperparameter, we have to further aggregate the fold scores into one single score, to get a score value for each hyperparameter. For this aggregation, either Min, Max, Average can be taken.

The following are results of MLP Regressor

Min Aggregation Results

0	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2436897552046847
1	{'n_estimators': 500, 'learning_rate': 0.001, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 2}	0.237073926816093
2	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2361739752605594
3	{'n_estimators': 300, 'learning_rate': 0.1, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2269803638849694
4	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.2269474443526744
5	{'n_estimators': 200, 'learning_rate': 0.1, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2266907266502659
6	{'n_estimators': 500, 'learning_rate': 0.001, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2248073038179276
7	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.2231019432925202
8	{'n_estimators': 200, 'learning_rate': 0.1, 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.2230627290266503
9	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 2}	0.2222751677905837

Max Aggregation Results

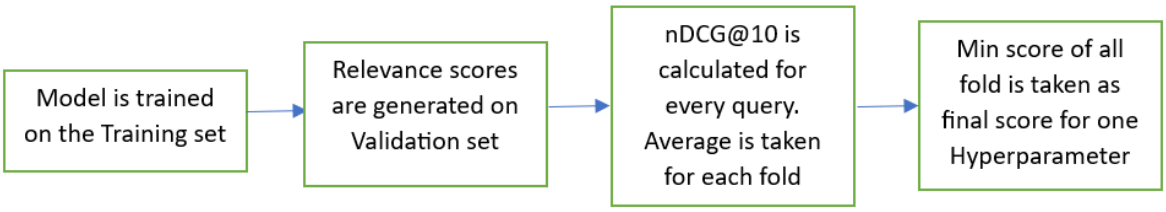
0	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.4564985856115363
1	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.4560858577261965
2	{'n_estimators': 200, 'learning_rate': 0.1, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.4557802884572481
3	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.4425047916121346
4	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.4424263852527688
5	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.4388200659762919
6	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.4351678440938052
7	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 2}	0.4348334193116237
8	{'n_estimators': 500, 'learning_rate': 0.1, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.4296932534463659
9	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.4281182879290228

Average Aggregation Results

0	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.3247790968132317
1	{'n_estimators': 200, 'learning_rate': 0.1, 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.3175867449623882
2	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.3168806601593198
3	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.31539517494097347
4	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.3069487094718889
5	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.30443625480761766
6	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.3027532574671973
7	{'n_estimators': 300, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.3022893611930236
8	{'n_estimators': 200, 'learning_rate': 0.01, 'max_depth': 5, 'min_samples_split': 5, 'min_samples_leaf': 4}	0.30059398349258837
9	{'n_estimators': 500, 'learning_rate': 0.01, 'max_depth': 7, 'min_samples_split': 10, 'min_samples_leaf': 4}	0.29868813542420336

I have taken the Min aggregation for this stage, since if I take the hyperparameter with the highest Min value, I am setting a higher lower bound for the performance of the model on the Test set.

The hyperparameter with highest score is finalised for testing on the Test Set of every fold.



For finding optimal the optimal hyperparameters, I made my own custom Grid Search. A grid of Parameters was passed, and the scores were stored by running the above method.

Support Vector Regressor

Parameter Grid used for GBR is as follows

Hyperparameter	Values
kernel	[rbf, linear]
C	[0.1, 1, 10, 100]
gamma	[scale, auto, 0.01, 0.1, 1]
epsilon	[0.01, 0.1, 0.5]

Optimal Hyper-parameter: -

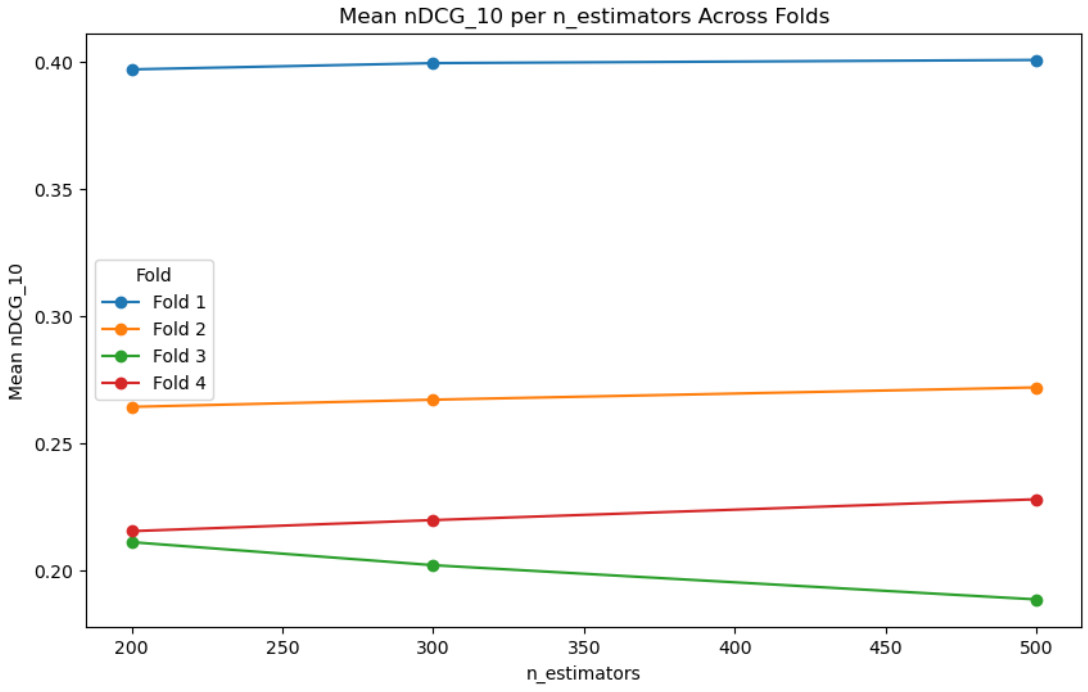
TD2003	{'kernel' : 'rbf', 'C' : 0.1, 'gamma' : 0.01, 'epsilon' : 0.01}
TD2004	{'kernel' : 'rbf', 'C' : 0.1, 'gamma' : 0.01, 'epsilon' : 0.01}

Gradient Boosting Regressor

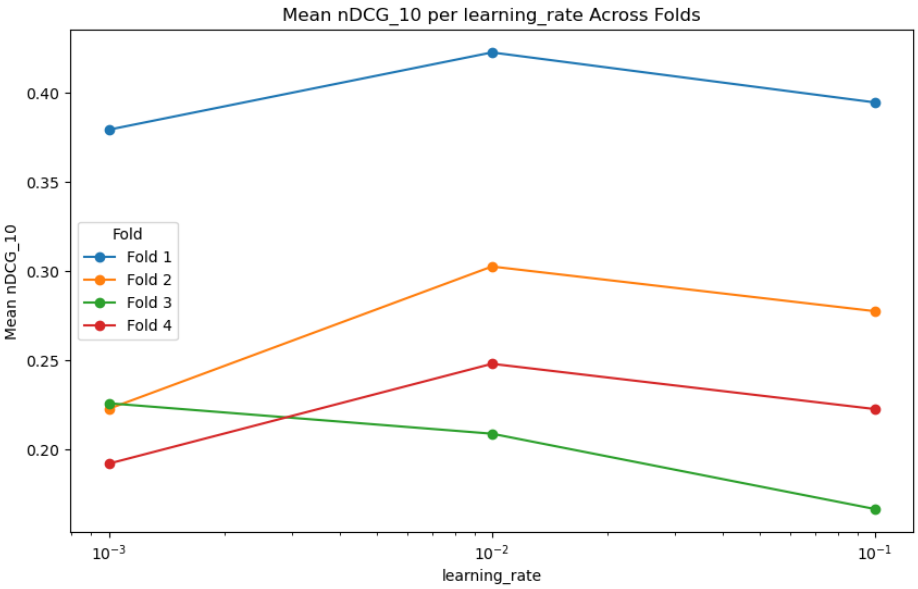
Parameter Grid used for GBR is as follows

Hyperparameter	Values
n_estimators	[100, 200, 300, 500]
learning_rate	[0.001, 0.01, 0.1]
max_depth	[5, 7, 9]
min_samples_split	[5, 7, 10]
min_samples_leaf	[2, 4, 6]

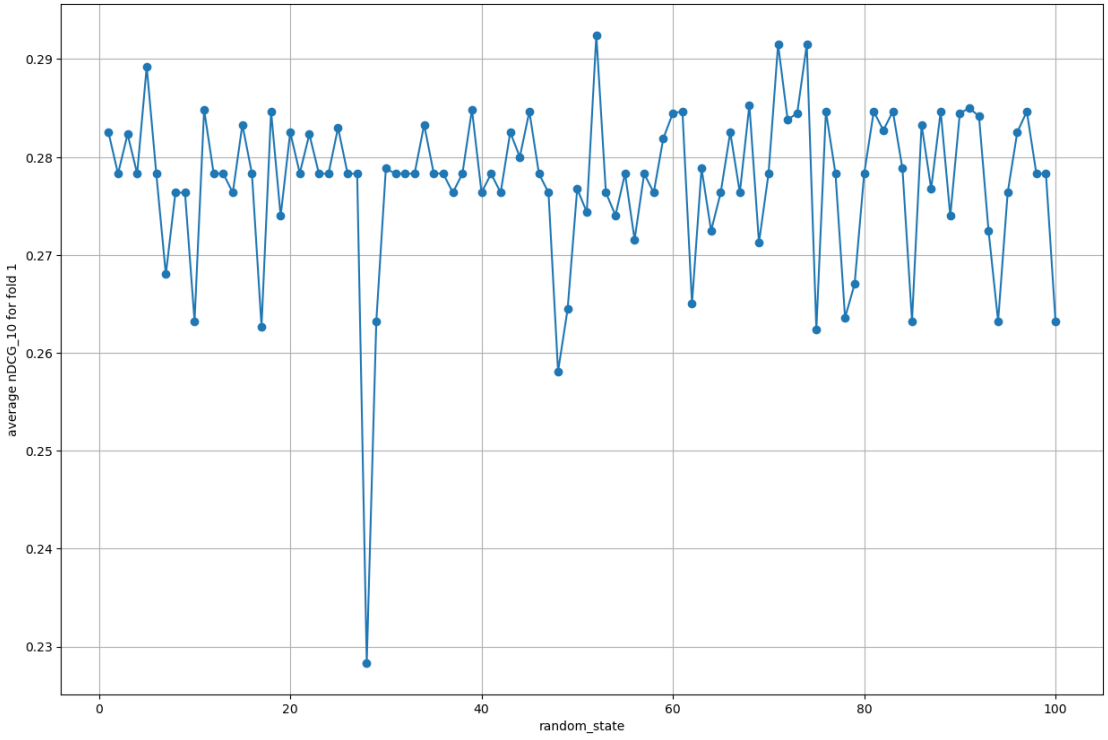
Variation with 'n_estimators'



Variation with ‘learning_rate’



Variation with random_state



Other hyperparameters have straight uniform behaviour

Optimal Hyper-parameter: -

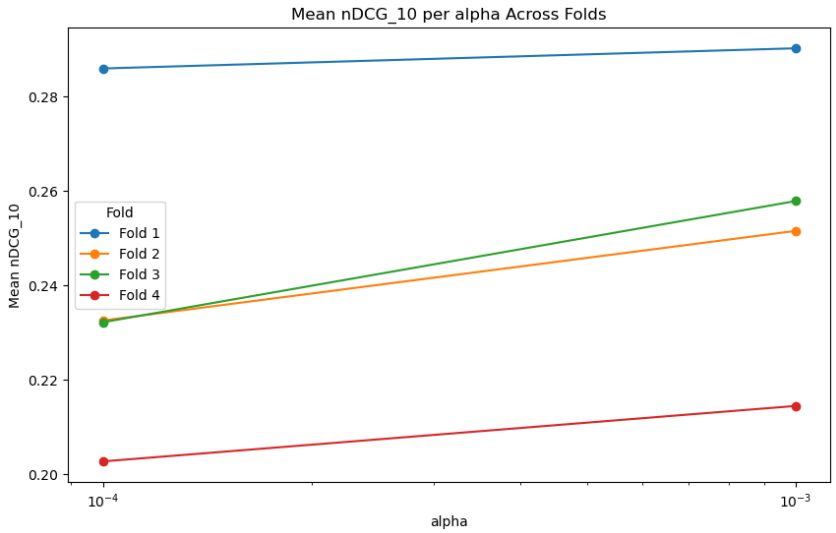
TD2003	{‘n_estimators’ : 300, ‘learning_rate’ : 0.01, ‘max_depth’ : 5, ‘min_samples_split’ : 5, ‘random_state’ : 52}
TD2004	{‘n_estimators’ : 300, ‘learning_rate’ : 0.01, ‘max_depth’ : 5, ‘min_samples_split’ : 2, ‘random_state’ : 52}

MLP Regressor

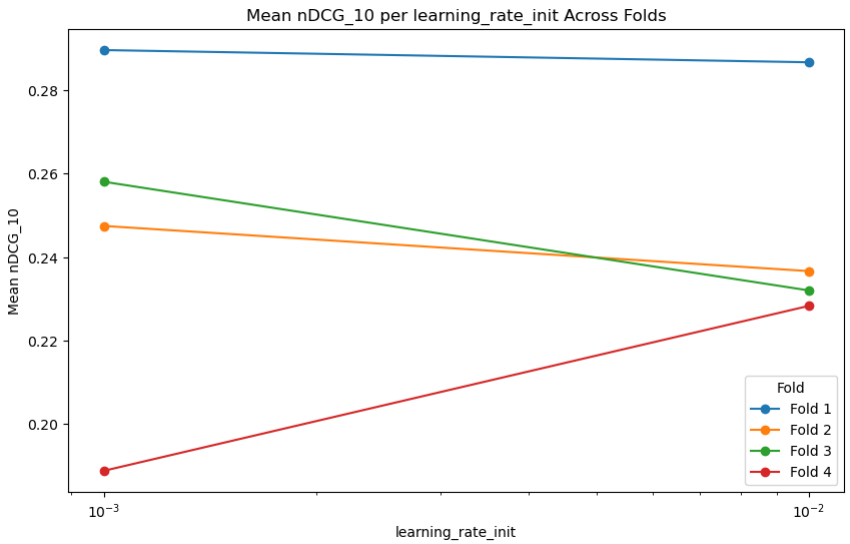
Parameter Grid used for GBR is as follows

Hyperparameter	Values
hidden_layer_sizes	[(100,), (50, 50), (100, 50), (100, 100)]
activation	[tanh, relu]
solver	adam
alpha	[0.0001, 0.001, 0.01]
learning_rate_init	[0.001, 0.01]
max_iter	[1000, 1500]

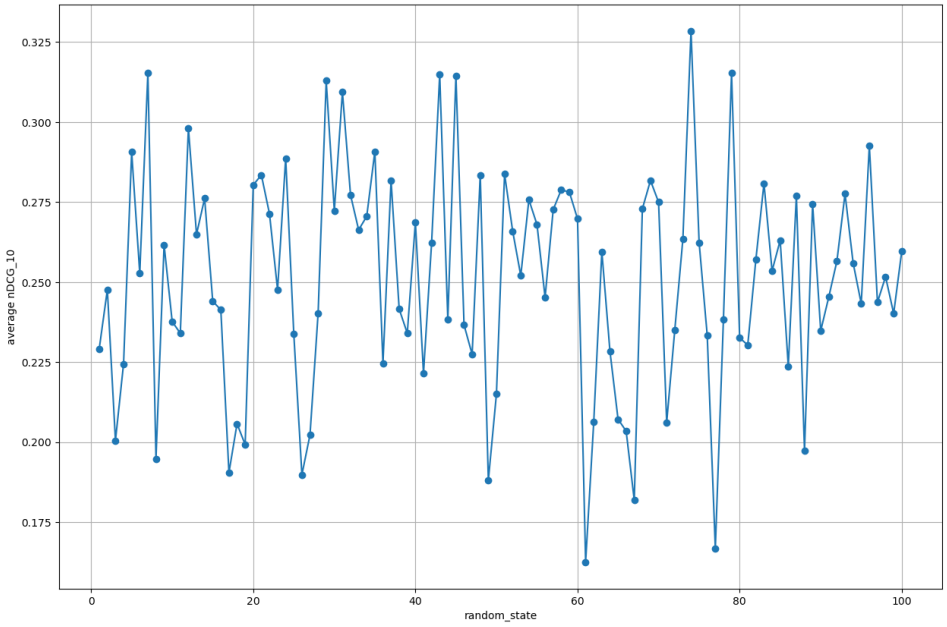
Variation with ‘alpha’



Variation with ‘learning_rate_init’



Variation with ‘random_state’



Optimal Hyper-parameter: -

TD2003	{‘hidden_layer_sizes’: (100, 50), ‘activation’ : ‘relu’, ‘solver’ : ‘adam’, ‘alpha’ : 0.001, ‘learning_rate_init’: 0.01, ‘max_iter’ : 1000, ‘random_state’ : 74}
TD2004	{‘hidden_layer_sizes’: (100, 100), ‘activation’ : ‘tanh’, ‘solver’ : ‘adam’, ‘alpha’ : 0.01, ‘learning_rate_init’: 0.001, ‘max_iter’ : 1500, ‘random_state’ : 79}

Performance on Train, Validation and Test

Following results are nDCG@10

SVR 2003 and 2004

fold	training	validation	test
1	0.774	0.322	0.202
2	0.765	0.206	0.238
3	0.732	0.279	0.194
4	0.748	0.181	0.328

fold	training	validation	test
1	0.725	0.336	0.314
2	0.731	0.363	0.257
3	0.705	0.235	0.355
4	0.708	0.435	0.255

GBR 2003 and 2004

fold	training	validation	test
1	0.729	0.439	0.274
2	0.723	0.311	0.227
3	0.674	0.216	0.170
4	0.728	0.331	0.387

fold	training	validation	test
1	0.803	0.393	0.359
2	0.825	0.398	0.318
3	0.779	0.363	0.380
4	0.780	0.401	0.370

MLP 2003 and 2004

fold	training	validation	test
1	0.559	0.333	0.284
2	0.552	0.303	0.419
3	0.508	0.327	0.318
4	0.565	0.138	0.293

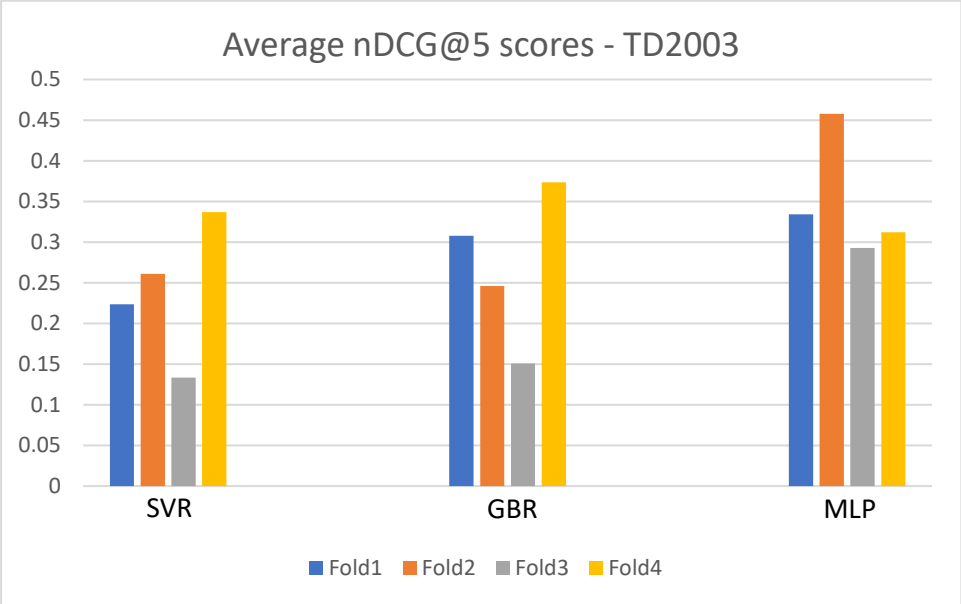
fold	training	validation	test
1	0.500	0.434	0.454
2	0.542	0.470	0.413
3	0.432	0.392	0.489
4	0.493	0.506	0.379

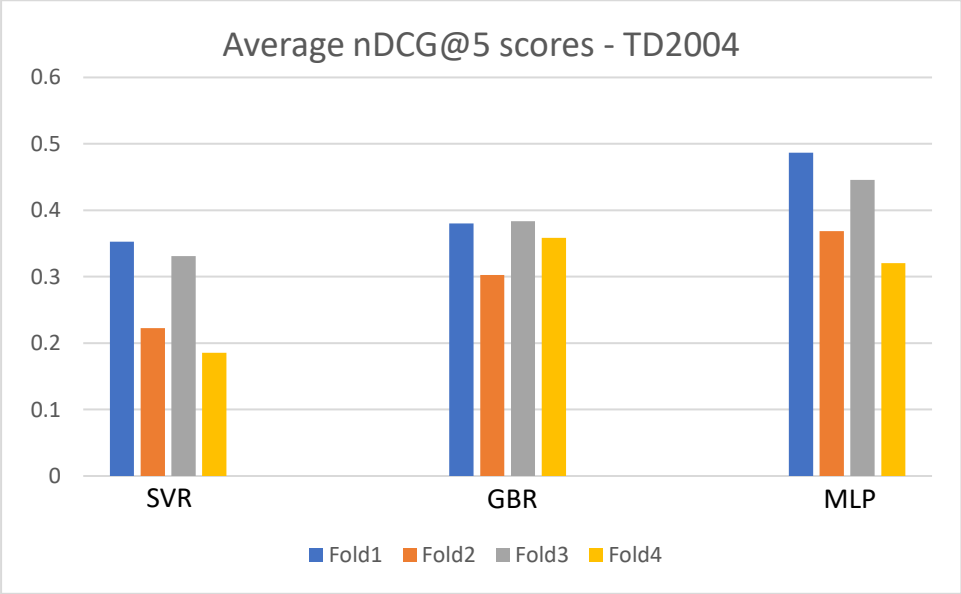
Further comparisons have been done on the test set only for all the folds

Two sets of comparisons have been done – Dataset based and Model based

Dataset based comparisons of Test set results

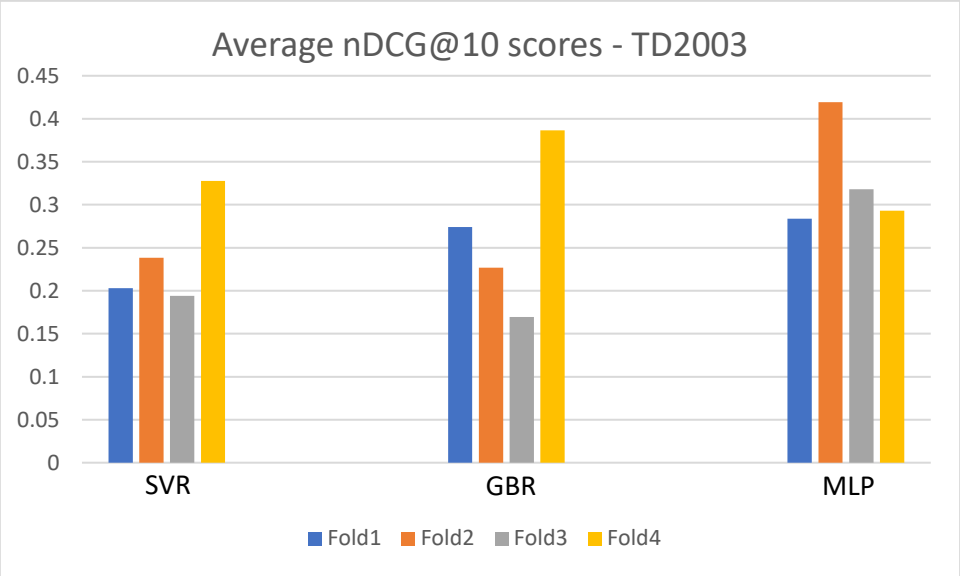
Comparison of average nDCG@5 on TD2003 and TD2004 Dataset folds for the three models.

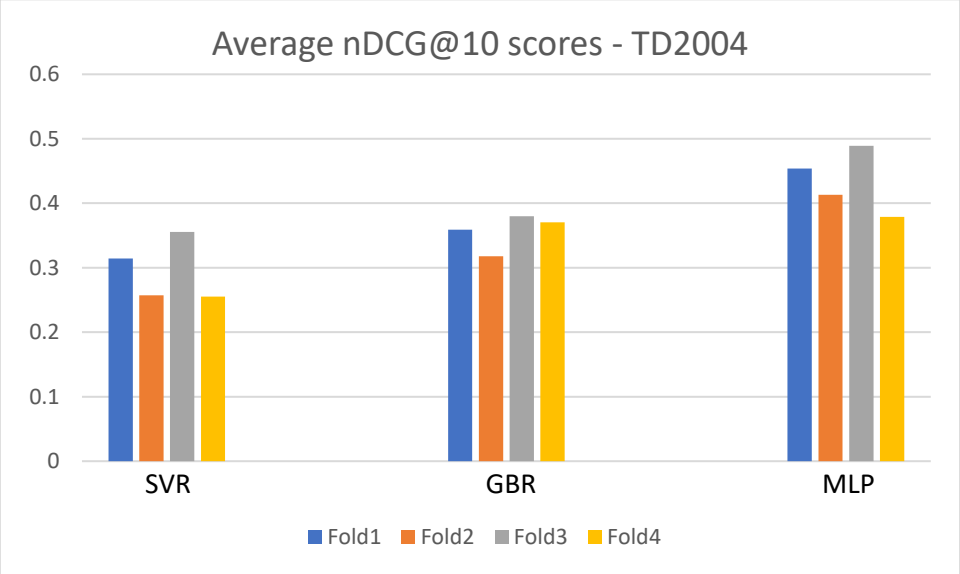




Observations for nDCG@5 – MLP has outperformed both SVR and GBR in all the folds of TD2003 and TD2004. GBR has performed better than SVR on TD2004.

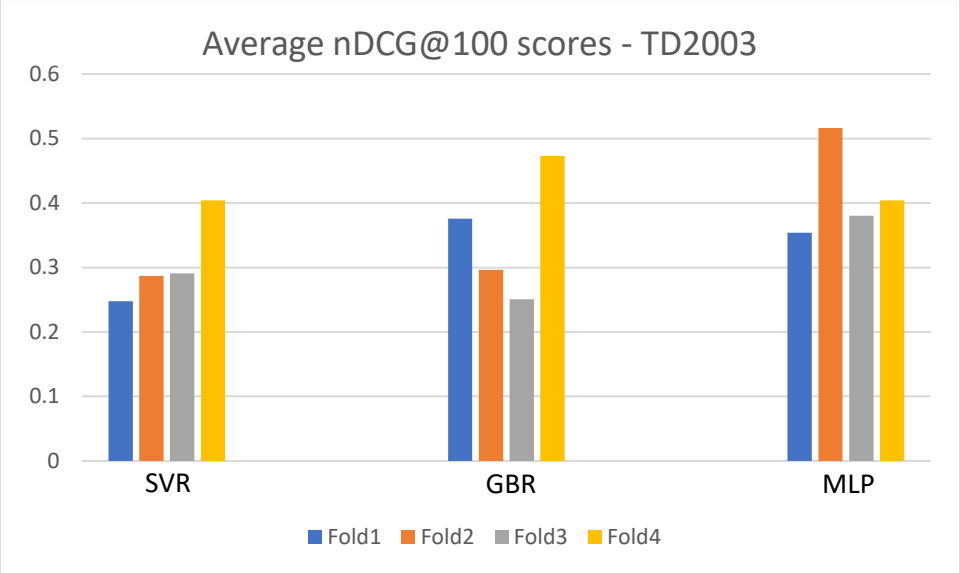
Comparison of average nDCG@10 on TD2003 and TD2004 Dataset folds for the three models.

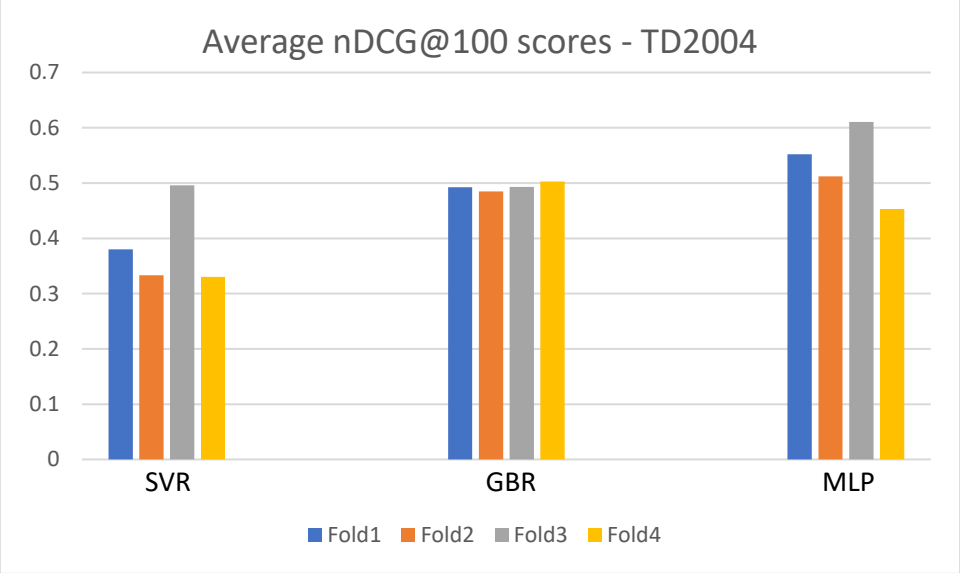




Observations for nDCG@10 – GBR has performed better than SVR on nDCG@10 metric. Fold 4 of TD2003 has higher scores compared to fold 1, 2 and 3. The scores of MLP are not consistent, as compared to GBR.

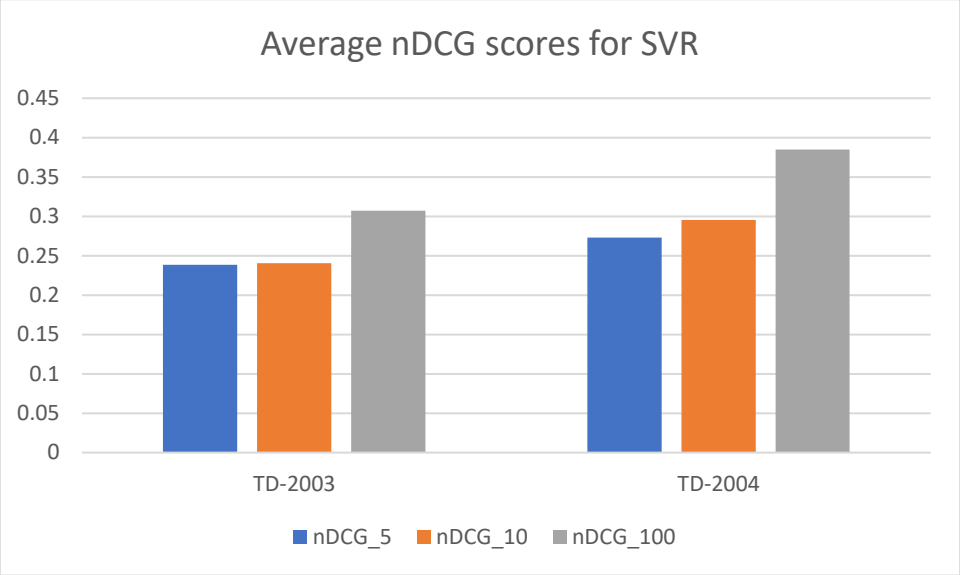
Comparison of average nDCG@100 on TD2003 and TD2004 Dataset folds for the three models.

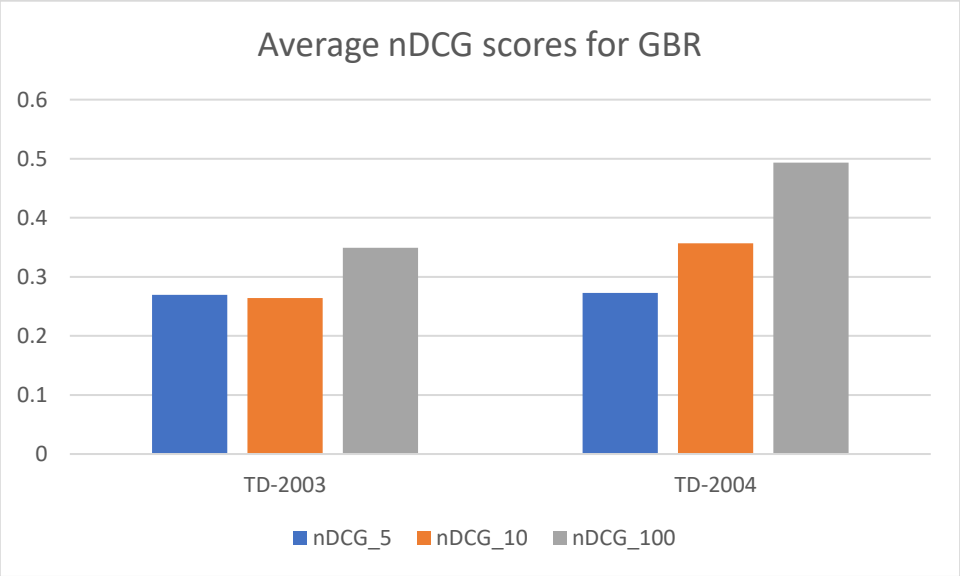


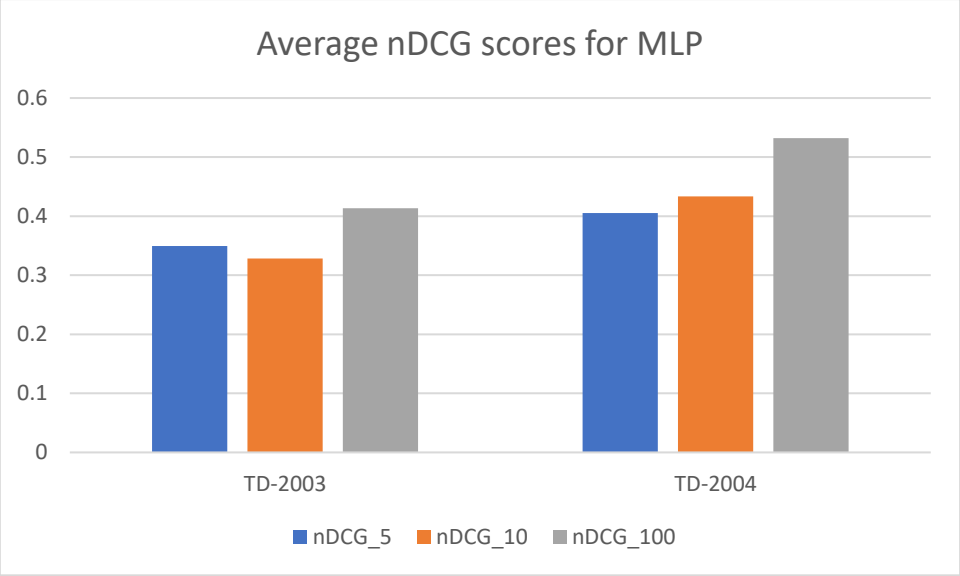


Similar observations for nDCG@100. The scores are significantly higher than nDCG@5 and 10. Reason might be models are able to rank the relevant documents in the first 100 docs, than in first 5 or 10.

Model based comparisons of Test set results



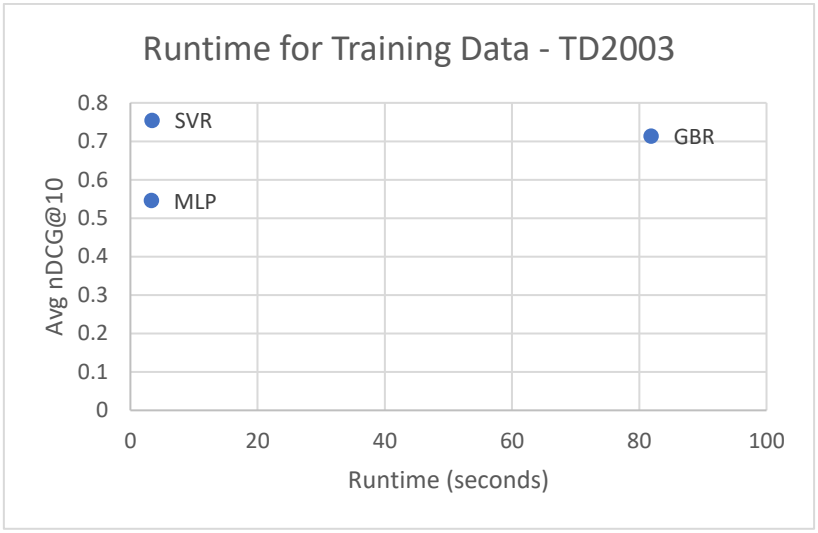




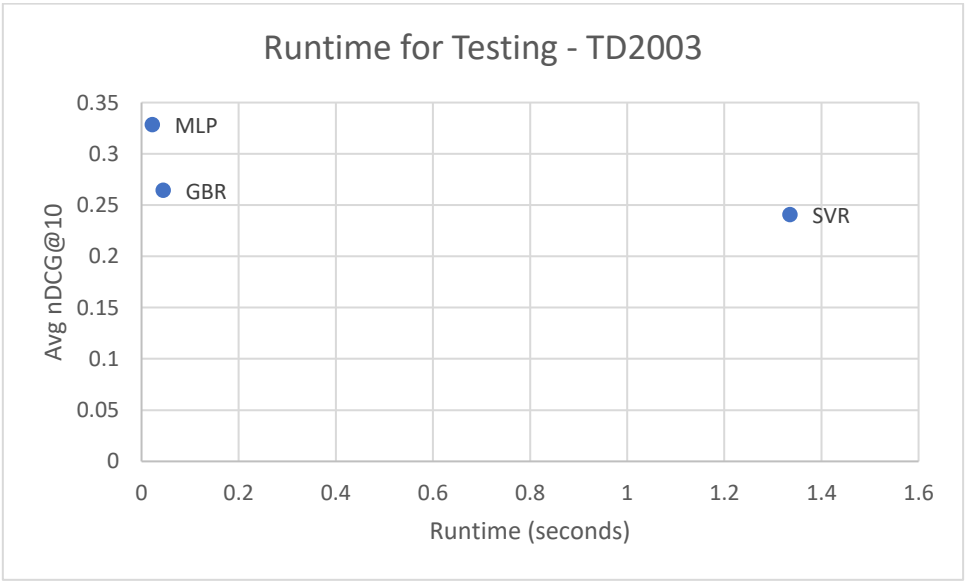
Observed Runtime on Training and Testing

Following are the results I have gathered on the runtime

Training Data Runtime (in seconds)															
SVR						GBR						MLP				
fold	1	2	3	4		fold	1	2	3	4		fold	1	2	3	4
2003	2.829878	3.093733	3.874145	3.897288		2003	79.92686	80.77762	83.14168	83.62878		2003	2.954693	2.900589	3.646047	3.708158
2004	3.694993	4.070441	3.671863	4.237405		2004	131.3617	129.4274	128.7131	133.3435		2004	12.44646	14.32512	12.92337	14.04004

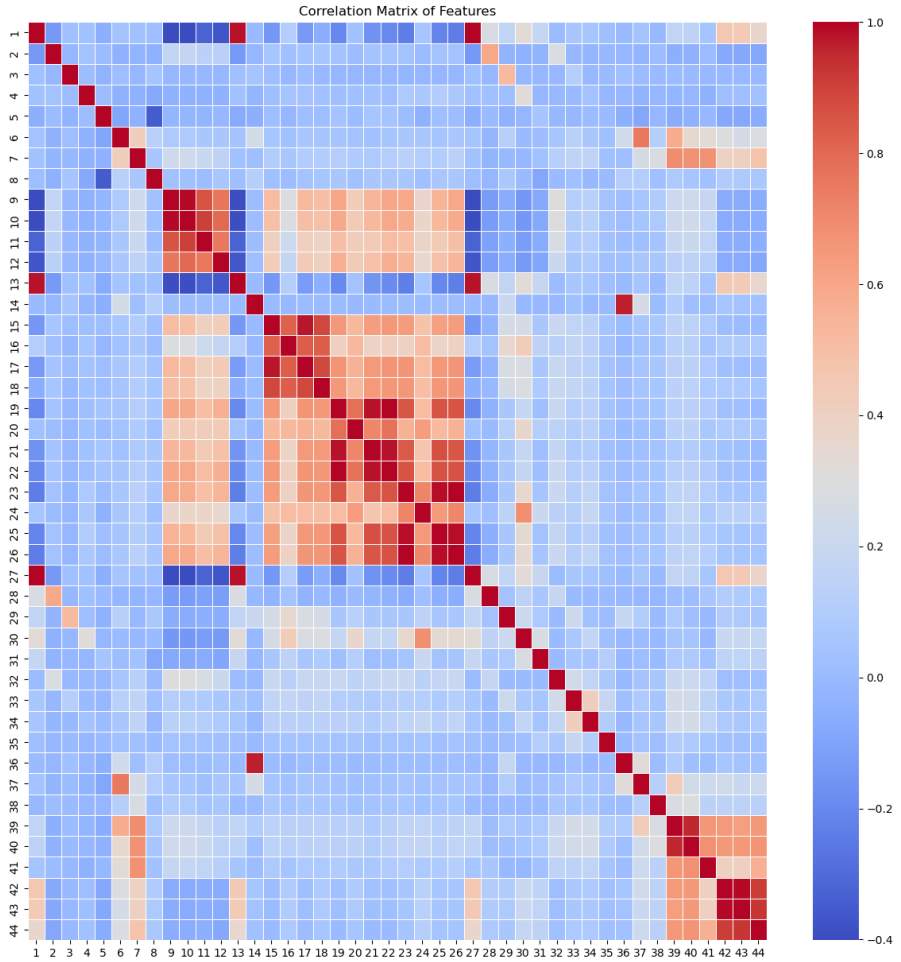


Testing Data Runtime (in seconds)															
SVR						GBR						MLP				
fold	1	2	3	4		fold	1	2	3	4		fold	1	2	3	4
2003	0.909593	1.300818	1.497937	1.630484		2003	0.040065	0.047525	0.044977	0.045099		2003	0.022006	0.025092	0.019001	0.022044
2004	1.578266	1.719227	1.56863	1.763113		2004	0.074637	0.118692	0.091532	0.057511		2004	0.07158	0.068053	0.069603	0.067492



Choices made to improve runtime

1. Since I was performing my own custom GridSearch and that I aggregated the scores using min of avg nDCG score aggregation, if at any point in the iterations, score of a fold for a hyper-parameter came to be less than the best score of past iterations, then no further folds were evaluated using that hyper-parameter.
   This optimised the runtime of the search, from almost 10 hrs on GBR to 3 hrs.
2. Remove features with high correlations
   Observe that many features are highly correlated.



I removed features ["1", "17", "9", "22", "19", "26", "44"] to decrease this correlation amount and increase the runtime efficiency of the models.