COL764: Assignment 3
IIT Delhi

Archisman Biswas, 2021MT10254

26 October 2024

Contents

1 Pointwise Learning to Rank (LETOR)

In pointwise learning to rank, each document-query pair is treated as an individual regression problem where a relevance score is predicted. The models minimize a loss function that measures the difference between the predicted and true relevance scores.

1.1 Support Vector Regressor (SVR)

The Support Vector Regressor solves the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} L_\epsilon(y_i, f(\mathbf{x}_i)),$$

where $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ is the regression function, L_ϵ is the ϵ-insensitive loss function given by

$$L_\epsilon(y_i, f(\mathbf{x}_i)) = \max(0, |y_i - f(\mathbf{x}_i)| - \epsilon),$$

and C controls the trade-off between the flatness of the model and the amount up to which deviations larger than ϵ are tolerated.
Library used for implementation: `SVR` from `sklearn.svm`

1.2 Gradient Boosted Decision Tree (GBDT)

Gradient Boosted Decision Trees optimize an ensemble of trees by minimizing a differentiable loss function $L(y, \hat{y})$ over the model's predictions:

$$\hat{y} = \sum_{m=1}^{M} \gamma_m h_m(\mathbf{x}),$$

where M is the number of trees, h_m are the individual decision trees, and γ_m are the weights found by line search to minimize

$$L(y_i, \hat{y}_i^{(m-1)} + \gamma_m h_m(\mathbf{x}_i)),$$

where $\hat{y}_i^{(m-1)}$ is the prediction from the first $m-1$ trees.
Library used for implementation: `GradientBoostingRegressor` from `sklearn.ensemble`

1.3 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron is a feedforward neural network that minimizes a loss function $L(y, \hat{y})$, where the predicted relevance score is given by:

$$\hat{y} = \sigma\left(\mathbf{W}^{(L)}\sigma\left(\cdots\sigma\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \cdots + \mathbf{b}^{(L)}\right)\right),$$

with L layers, weight matrices $\mathbf{W}^{(l)}$, bias vectors $\mathbf{b}^{(l)}$, and an activation function σ, typically ReLU $\sigma(z) = \max(0, z)$.
Library used for implementation: `MLPRegressor` from `sklearn.neural_network`

2 Hyperparameter Tuning

Initially, the hyperparameters are set to their default values. Each hyperparameter is then tuned by evaluating the model's performance across a range of possible values. For that a **sequential search** is done on the parameters, where one at a time is tuned while the others are constant (at their default values). In this part, there is not much optimization for runtime required, since the number of training examples is on the lower side.

Evaluation metric

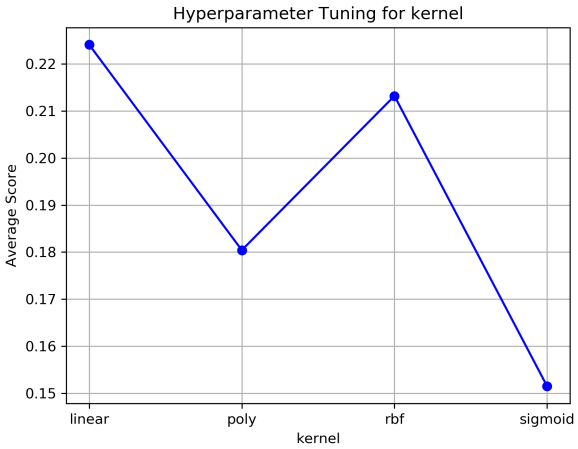
For each model, compute the average of $\min_i \text{avg\_nDCG}[i]$ across the four folds, where avg_nDCG[i] for a fold represents the nDCG[i] values averaged for a query.

2.1 Support Vector Regressor

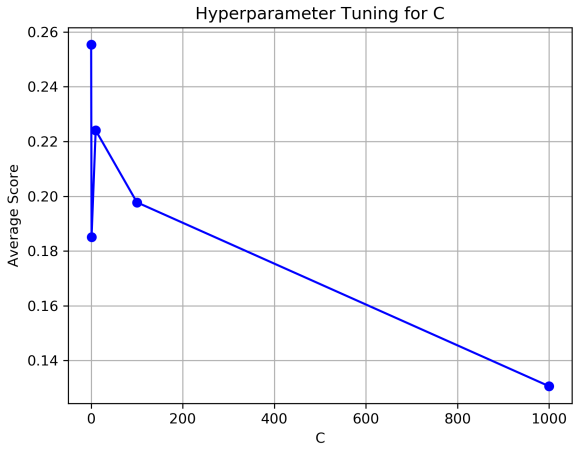
kernel	C	epsilon	degree	max_iter	shrinking	gamma
'rbf'	1.0	0.1	3	-1	True	'auto'

Table 1: Hyperparameters to be tuned (with default values)

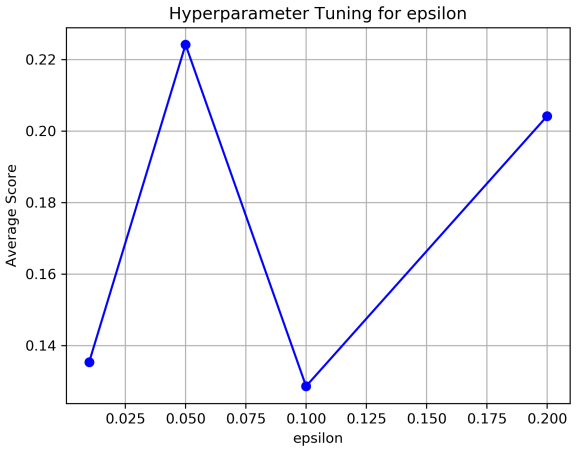
2.1.1 TD2003 Dataset



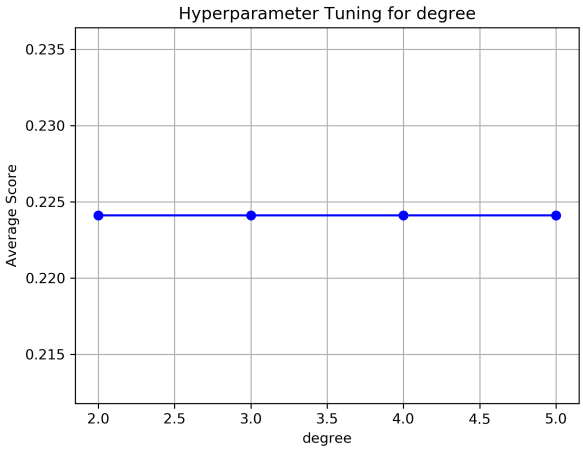
kernel



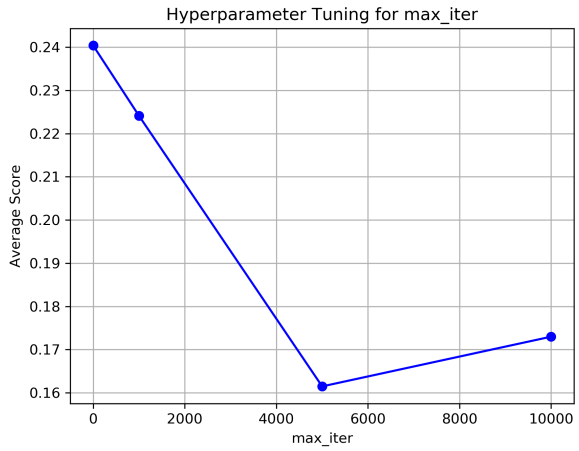
C



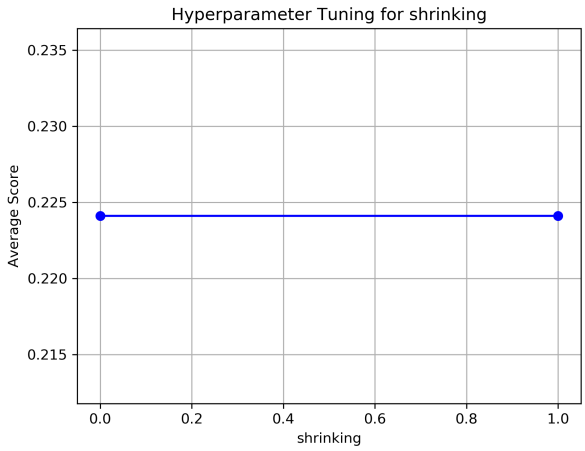
epsilon



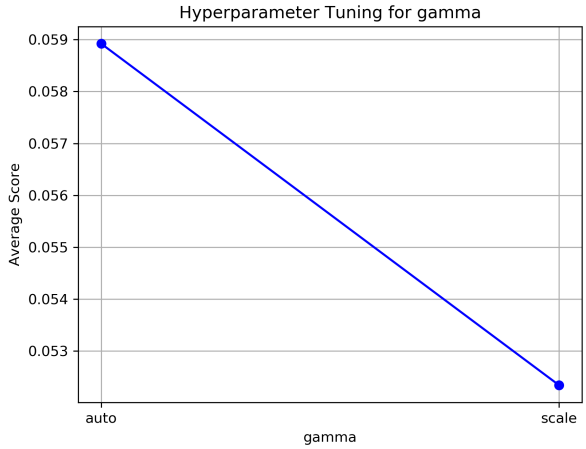
degree



max_iter



shrinking



gamma

Figure 4: Results on different hyperparameter values

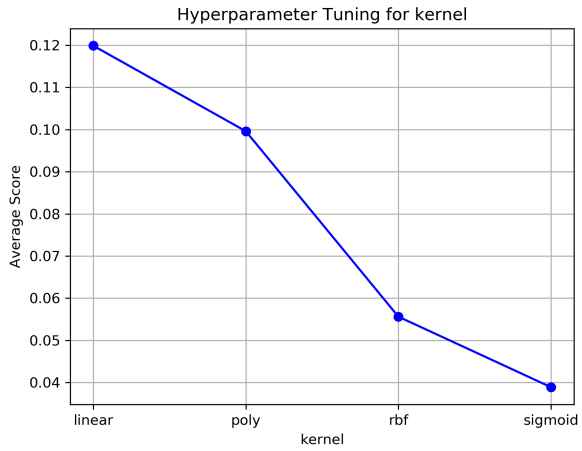
Therefore, based on the above results we choose the following values:

kernel	**C**	**epsilon**	**degree**	**max_iter**	**shrinking**	**gamma**
'Linear'	10.0	0.05	3	-1	True	'auto'

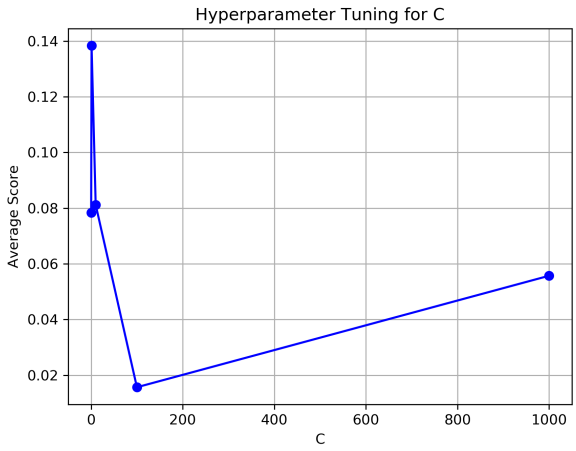
Table 2: SVR hyperparameters for TD2003

Note: `C=10.0` has been chosen after taking into account the consistency in the pattern showed.

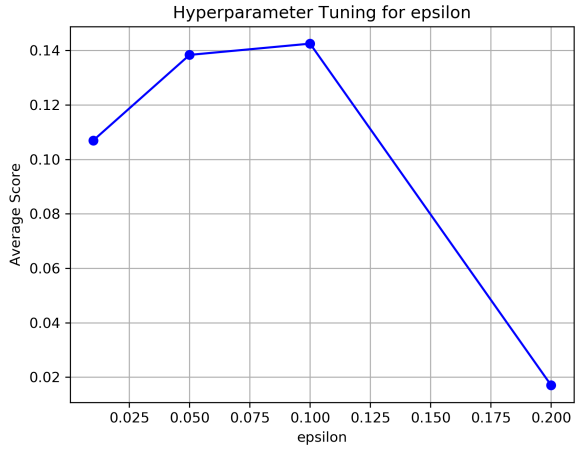
2.1.2 TD2004 Dataset



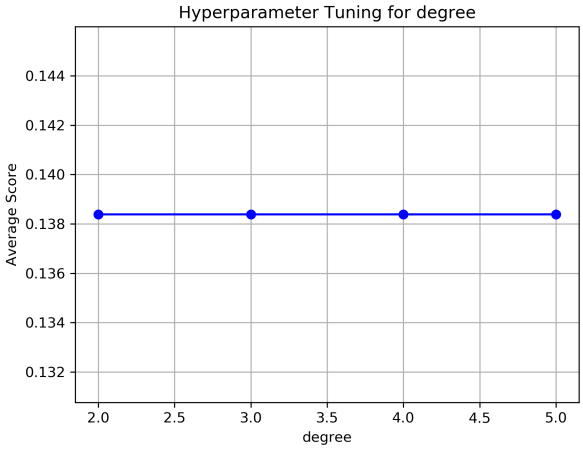
kernel



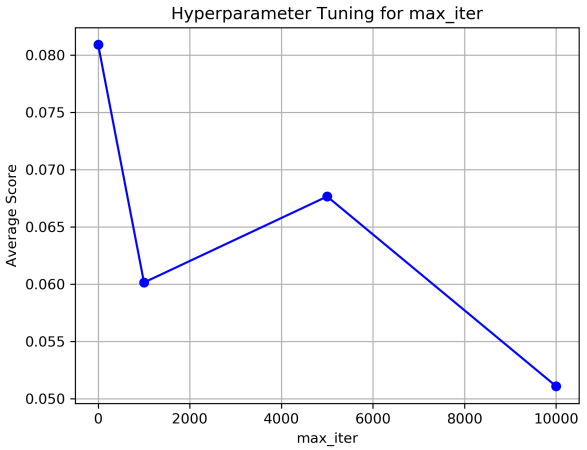
C



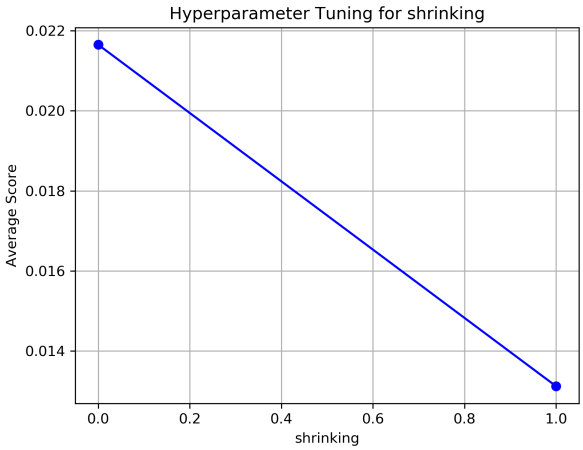
epsilon



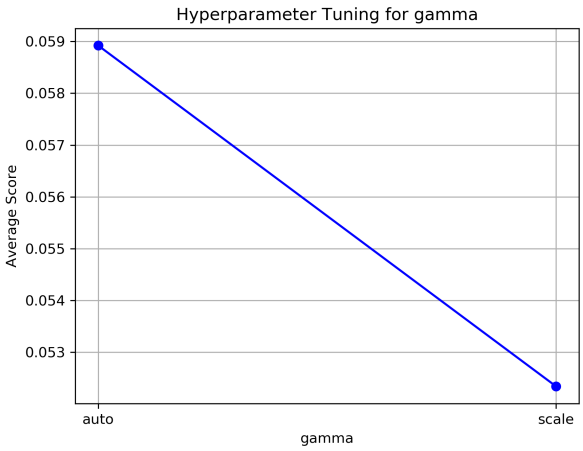
degree



max_iter



shrinking



gamma

Figure 8: Results on different hyperparameter values

Therefore, based on the above results we choose the following values:

kernel	C	epsilon	degree	max_iter	shrinking	gamma
'Linear'	10.0	0.1	3	-1	True	'auto'

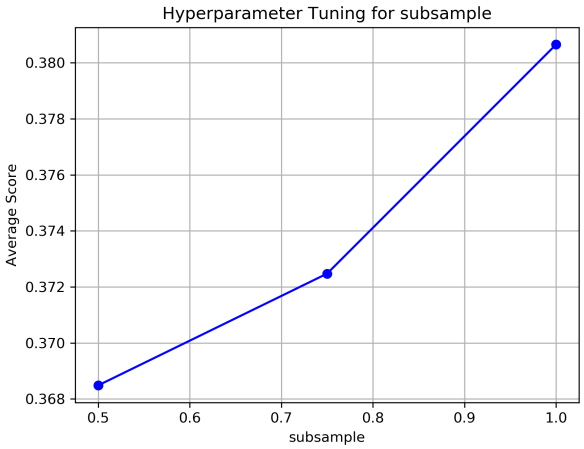
Table 3: SVR hyperparameters for TD2004

2.2 Gradient Boosted Decision Tree

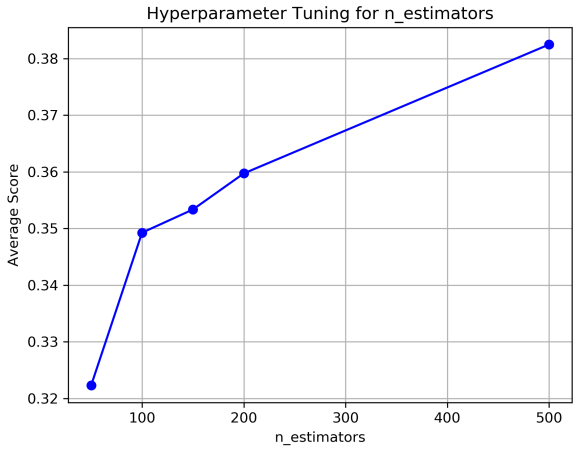
subsample	n_estimators	learning_rate	max_depth	criterion[1]	random_state	max_features	alpha
1.0	100	0.1	3	'friedman_mse'	None	'auto'	0.0

Table 4: Hyperparameters to be tuned (with default values)

2.2.1 TD2003 Dataset

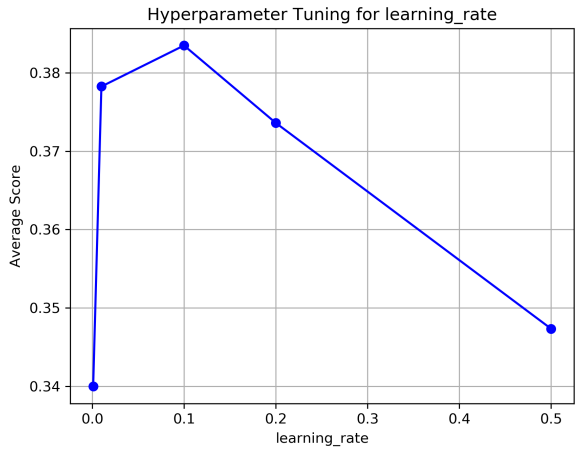


subsample

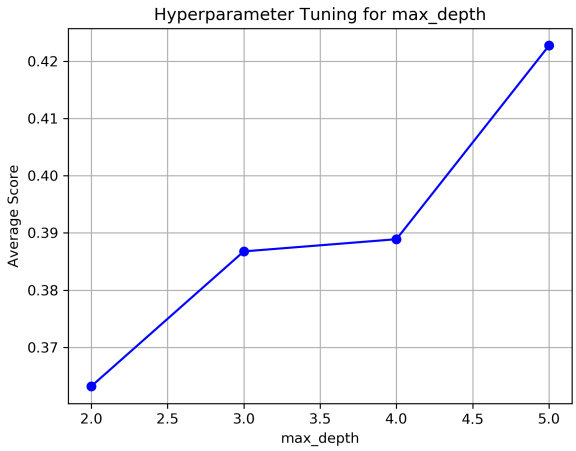


n_estimators

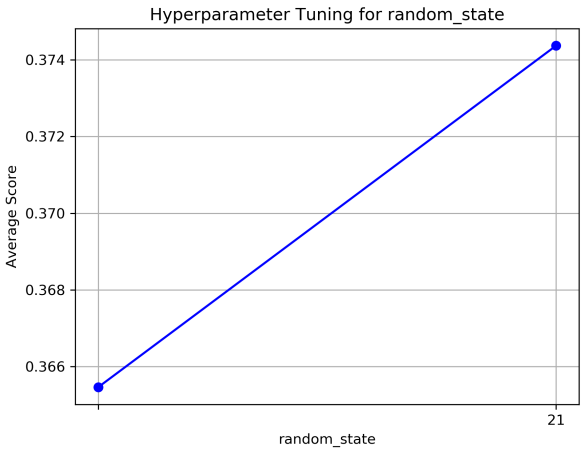
[1] Fixed



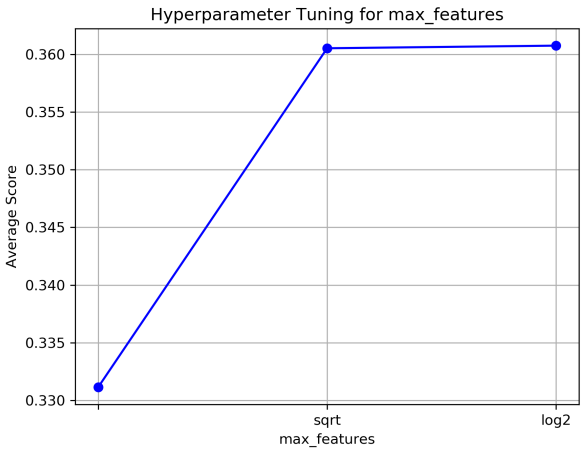
learning_rate



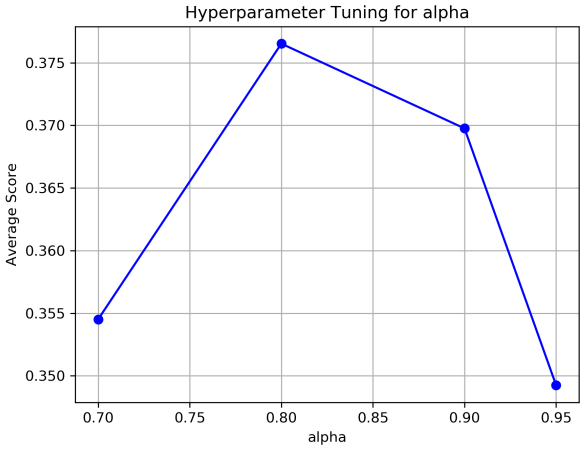
max_depth



random_state



max_features



alpha

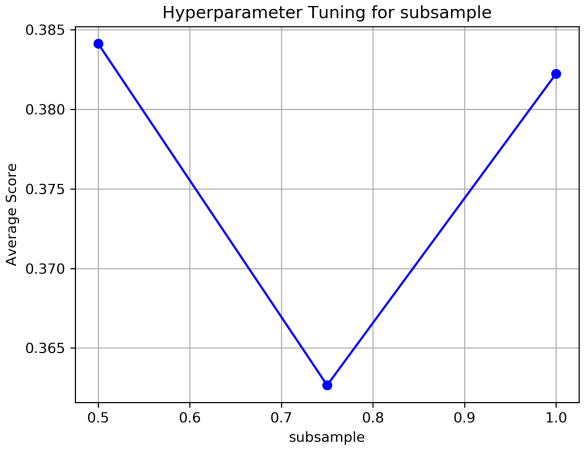
Figure 12: Results on different hyperparameter values

Therefore, based on the above results we choose the following values:

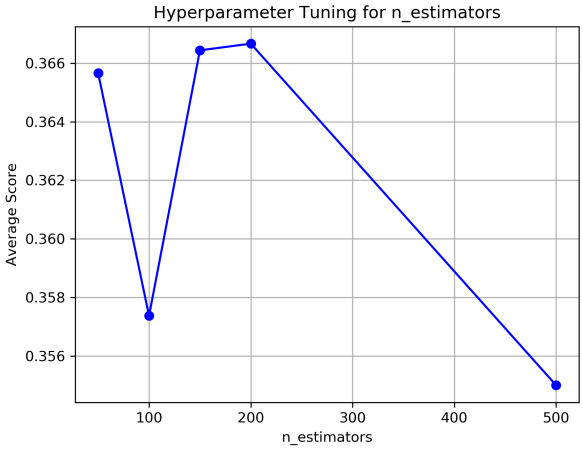
subsample	n_estimators	learning_rate	max_depth	criterion	random_state	max_features	alpha
1.0	500	0.1	5	'friedman_mse'	21	'sqrt'	0.8

Table 5: GBDT hyperparameters for TD2003

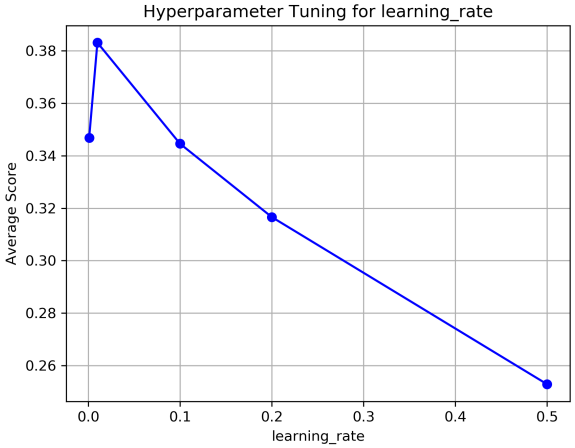
2.2.2 TD2004 Dataset



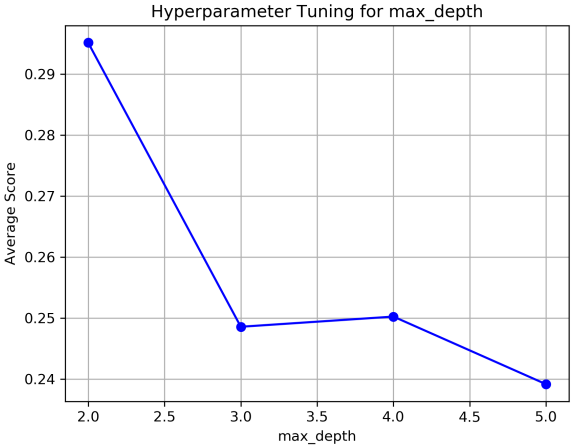
subsample



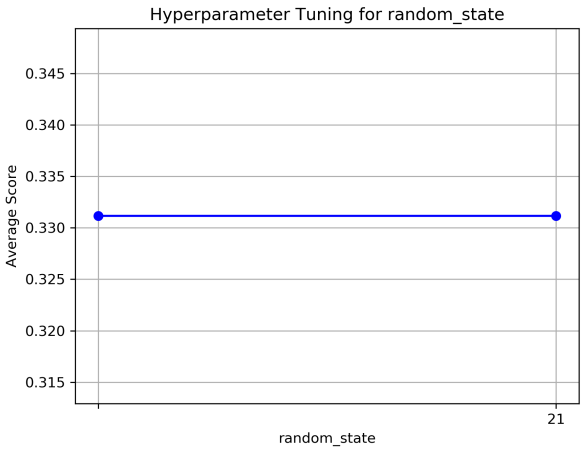
n_estimators



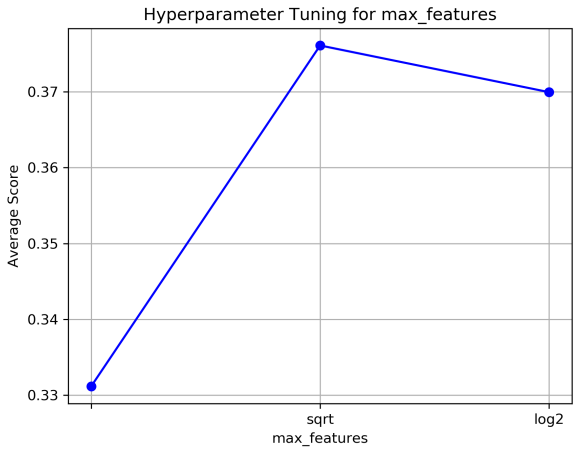
learning_rate



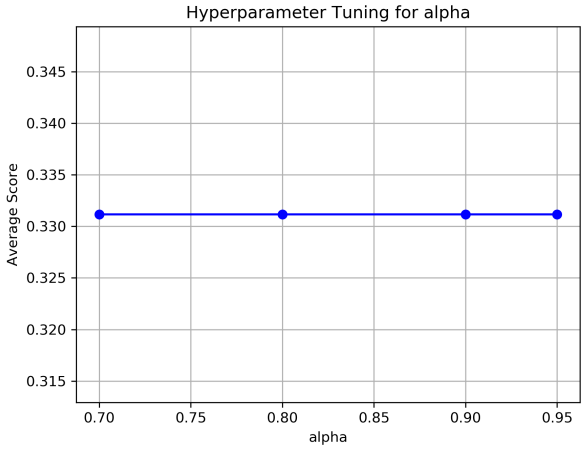
max_depth



random_state



max_features



alpha

Figure 16: Results on different hyperparameter values

Therefore, based on the above results we choose the following values:

subsample	**n_estimators**	**learning_rate**	**max_depth**	**criterion**	**random_state**	**max_features**	**alpha**
1.0	200	0.01	2	'friedman_mse'	21	'sqrt'	0.8

Table 6: GBDT hyperparameters for TD2004

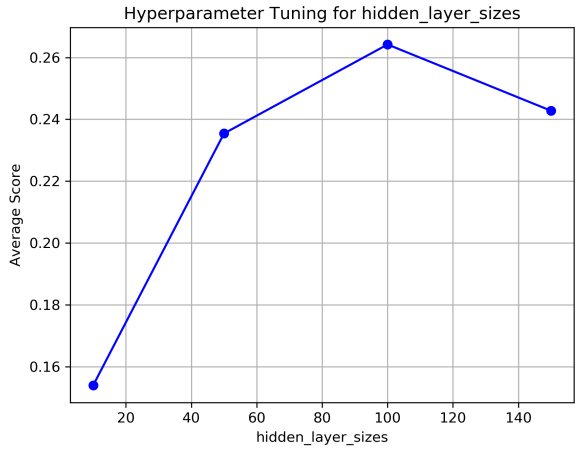
Note that ties are resolved based on results shown in **TD2003** dataset.

2.3 Multi-Layer Perceptron

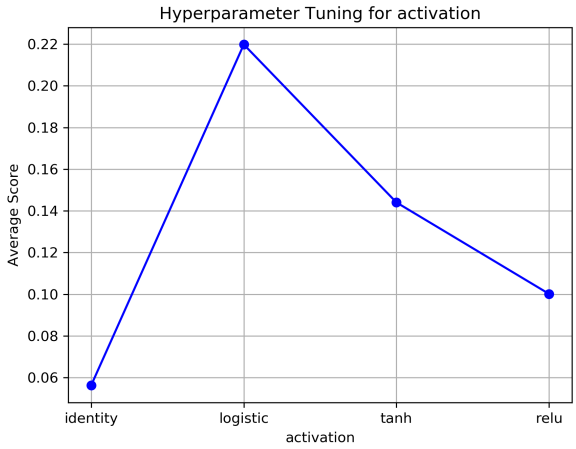
hidden_layer_sizes	activation	solver	max_iter	alpha	learning_rate	learning_rate_init	power_t	random_state
(100,)	'relu'	'adam'	200	0.0001	'constant'	0.001	0.5	None

Table 7: Hyperparameters to be tuned (with default values)

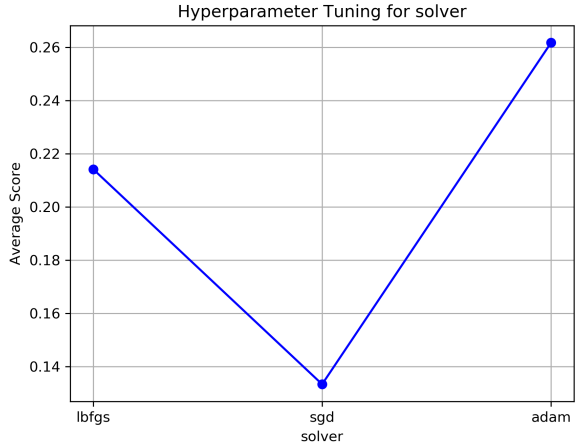
2.3.1 TD2003 Dataset



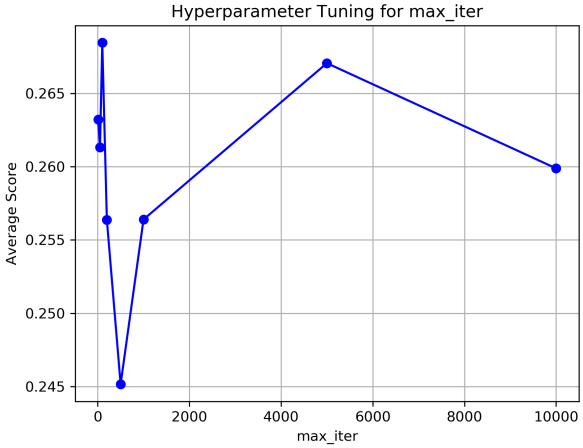
hidden_layer_sizes



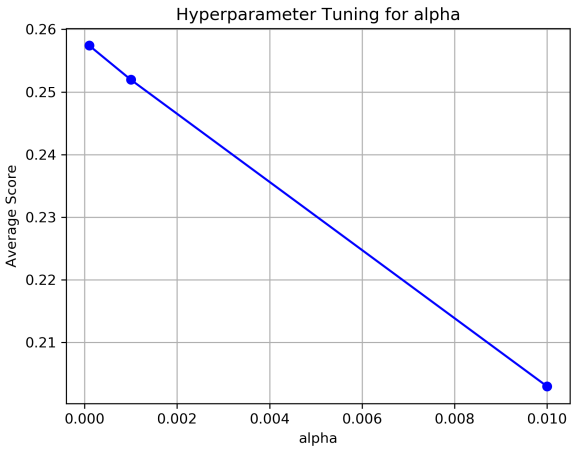
activation



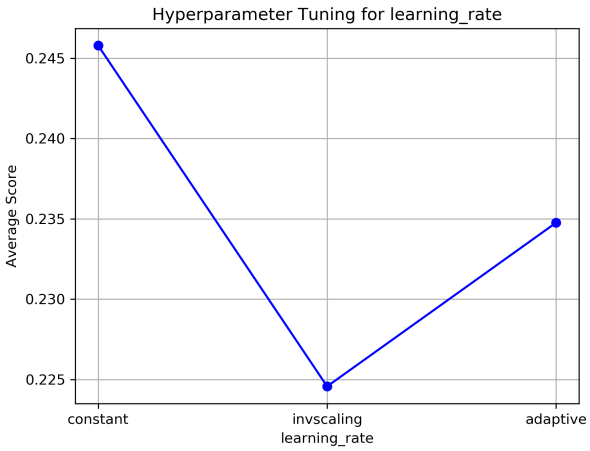
solver



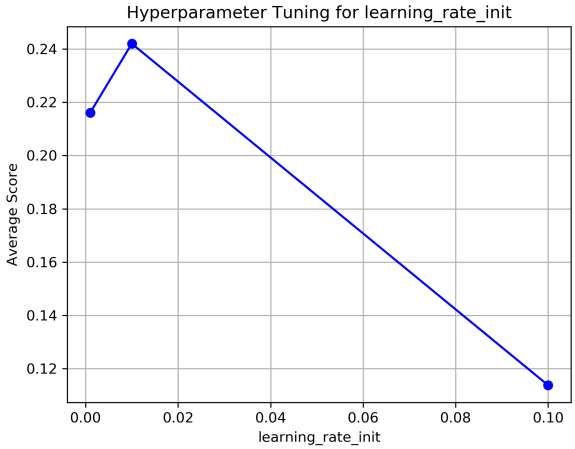
max_iter



alpha



learning_rate



learning_rate_init

Figure 20: Results on different hyperparameter values

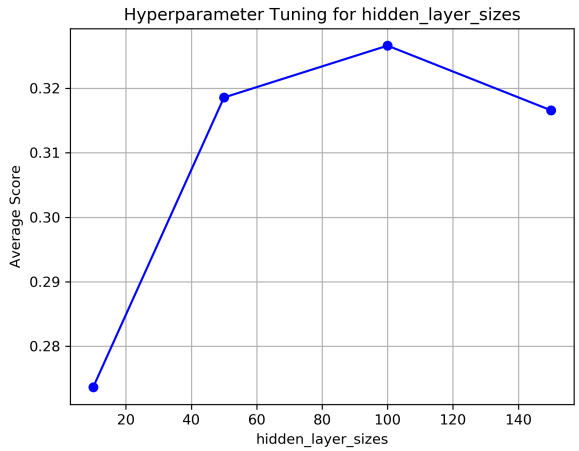
Therefore, based on the above results we choose the following values:

hidden_layer_sizes	**activation**	**solver**	**max_iter**	**alpha**	**learning_rate**	**learning_rate_init**	**power_t**	**random_state**
(100,)	'logistic'	'adam'	5000	0.0001	'adaptive'	0.01	0.5	60

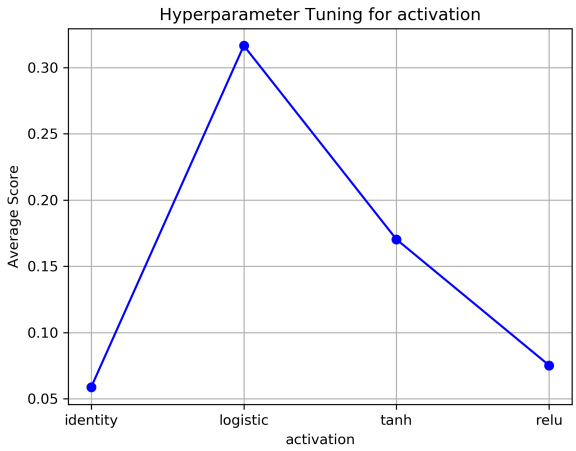
Table 8: MLP hyperparameters for TD2003

Note that `max_iter=200` has been chosen according to the pattern observed (in here as well in TD2004) and computation limits. `random_state` has been chosen to be some integer at random so that the quoted results are reproducible. `power_t`, the exponent for inverse scaling learning rate is fixed since it is only used when solver='sgd'. `learning_rate` has been set to **adaptive**, since it improves the runtime and does not differ too much to the maximum case. In that case, it will also be consistent with the results shown by TD2004.

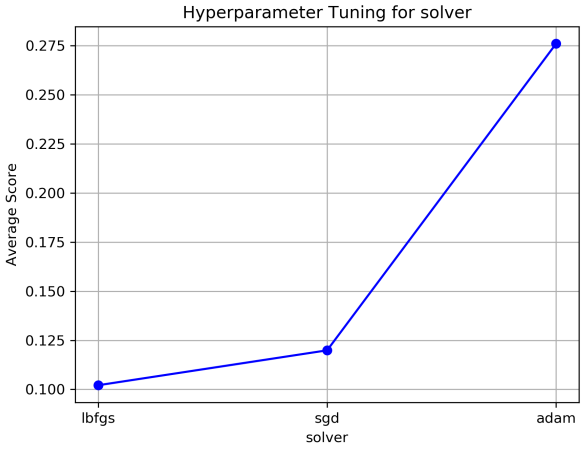
2.3.2 TD2004 Dataset



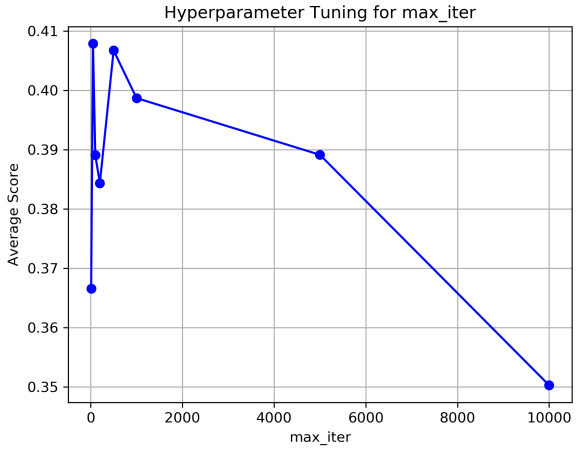
hidden_layer_sizes



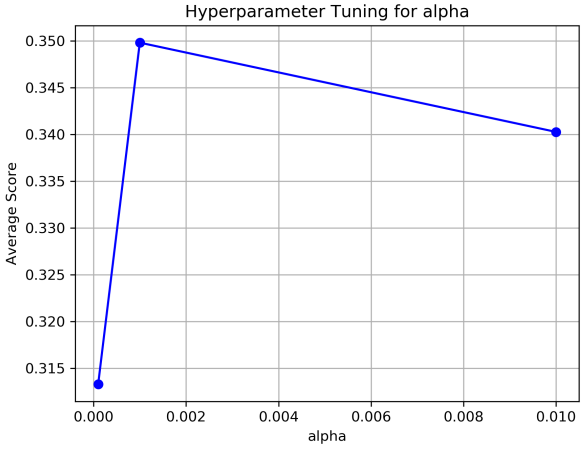
activation



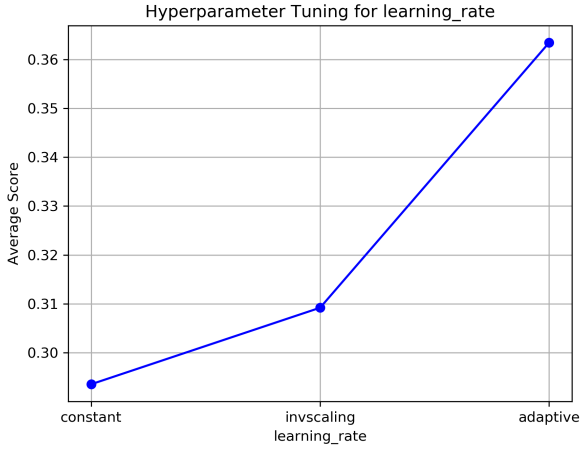
solver



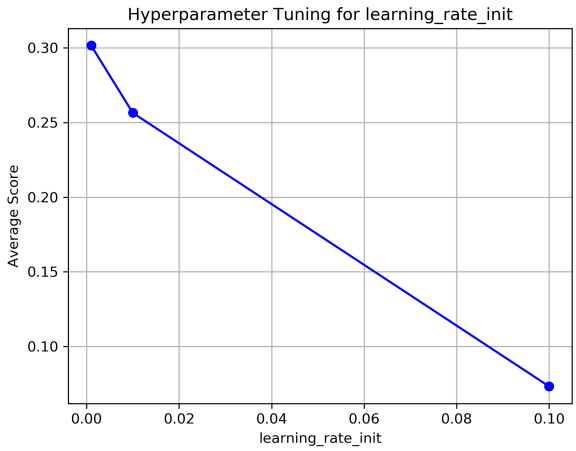
max_iter



alpha



learning_rate



learning_rate_init

Figure 24: Results on different hyperparameter values

Therefore, based on the above results we choose the following values:

hidden_layer_sizes	**activation**	**solver**	**max_iter**	**alpha**	**learning_rate**	**learning_rate_init**	**power_t**	**random_state**
(100,)	'logistic'	'adam'	5000	0.001	'adaptive'	0.001	0.5	0

Table 9: MLP hyperparameters for TD2004

Note that `max_iter=5000` has been chosen to out of consistency with the previous part, and satisfactory performance in this part too. The `random_state` choice is again to ensure reproducibility of the results. `power_t`, the exponent for inverse scaling learning rate is fixed since it is only used when solver='sgd'.

3 Feature Processing

In this part, tried out feature scaling, feature engineering, dimensionality reduction etc.
For feature scaling, min-max and standard scaling has been tried out, out of which the former fared better. MinMaxScaler has been used for implementation.

Feature engineering and dimensionality reduction details are as follows:

$$f_1 = \log(\text{BM25 score})$$
$$f_2 = \text{BM25 score of the anchor} \times \text{BM25 score of the title}$$
$$f_3 = \text{HITS authority} \times \text{HITS hub}$$
$$f_4 = \text{tfidf body} \times \text{tfidf anchor}$$

0	Original
1	Add feature f_1
2	Add features f_1, f_2, f_3, f_4
3	200 Add features f_2, f_3, f_4
4	0.001 Add feature f_2
5	Apply PCA (n_components=30)
6	Apply t-SNE (n_components=2)
7	Apply truncatedSVD (n_components=30)
8	Apply FastICA (n_components=30)

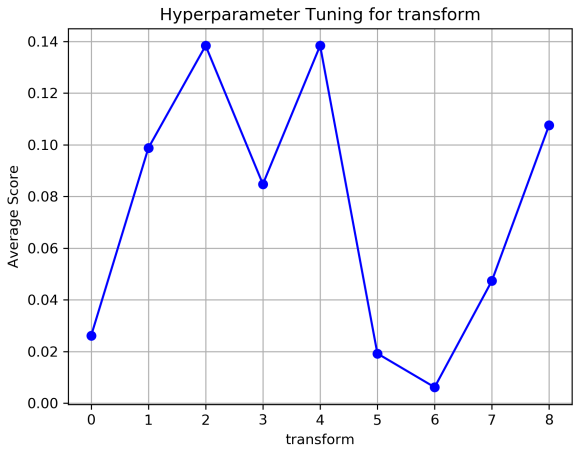
Table 10: Codes used for feature engineering and dimensionality reduction

1, 2, 3, 4 is feature engineering, while **5, 6, 7, 8** is dimensionality reduction. Both cannot be done together since the original features lose their meaning after dimensionality reduction. Hence, all the 9 cases could be compared on a single graph.

Note

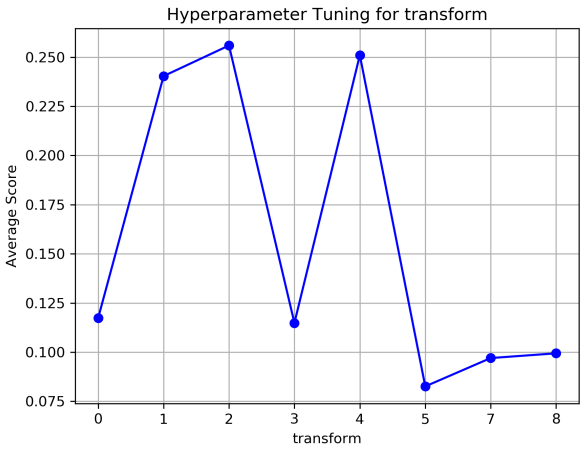
6 is removed from the experiments due to very low evaluation score and long computation time

3.1 Support Vector Regressor

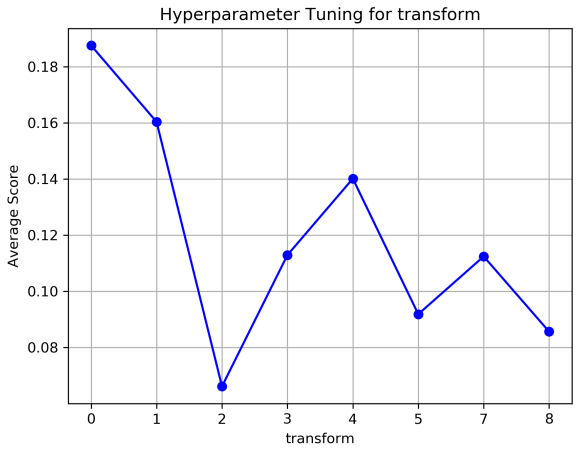


TD2003

Figure 25: Experimentation results of SVR before min-max scaling



TD2003

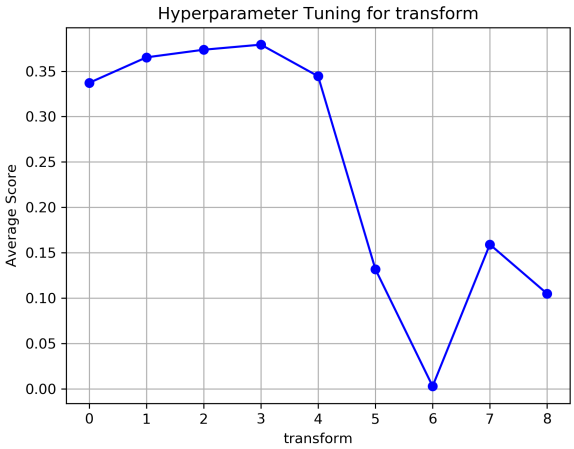


TD2004

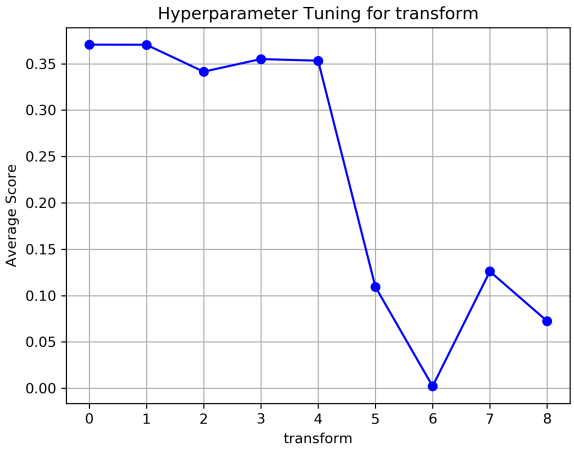
Figure 26: Experimentation results of SVR after min-max scaling

For scaling, we hence conclude that min-max scaling is much **better** for SVR. For feature engineering, after averaging out the results in both datasets, we notice that **1** (Adding feature f_1) suits the best.

3.2 Gradient Boosted Decision Tree

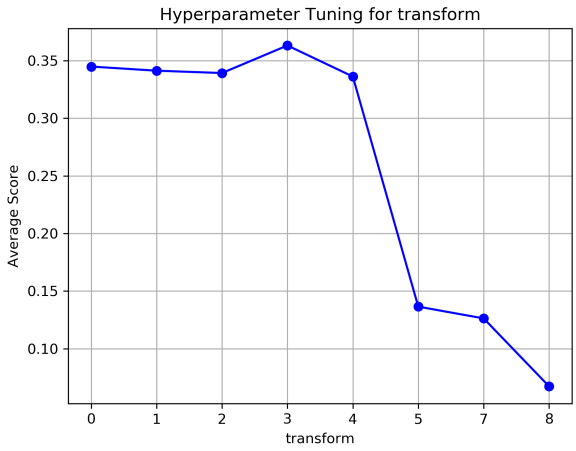


TD2003



TD2004

Figure 27: Experimentation results of MLP before min-max scaling

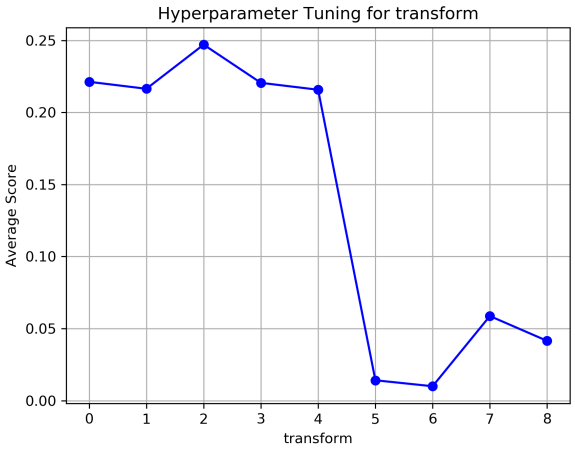


TD2003

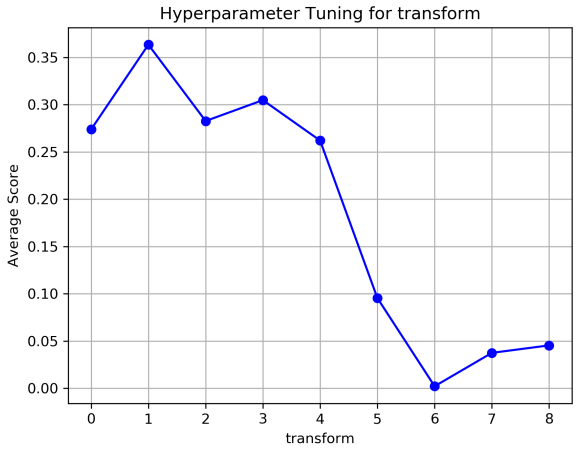
Figure 28: Experimentation results of GBDT after min-max scaling

Decision Trees are known to be prone to scaling and hence, as it can be seen min-max scaling **does not** work well. For feature engineering, after averaging out the results in both datasets, we notice that **1** (Adding feature f_1) suits the best.

3.3 Multi-Layer Perceptron

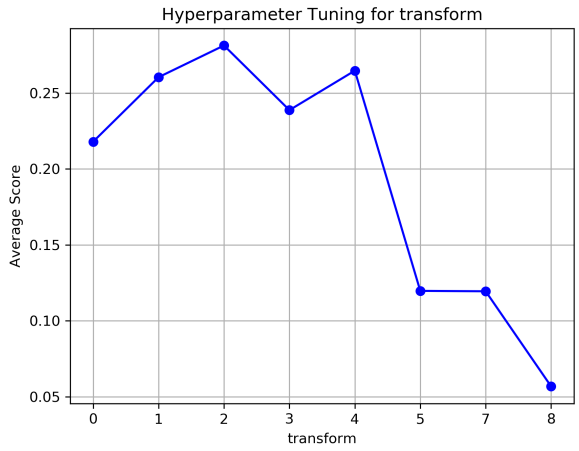


TD2003

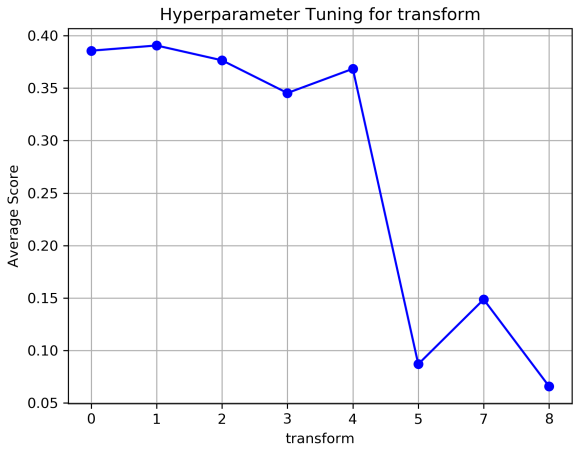


TD2004

Figure 29: Experimentation results of MLP before min-max scaling



TD2003



TD2004

Figure 30: Experimentation results of MLP after min-max scaling

For scaling, it is not satisfactorily clear if max-scaling is that good, so it is **not** done. For feature engineering, after averaging out the results in both datasets, we notice that **1** (Adding feature f_1) suits the best.

4 Results

Note

All following measures are given w.r.t programs run on Apple Silicon M2 chip

4.1 Support Vector Regressor

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.2555	0.2583	0.3399
val	0.4069	0.3778	0.3413
test	0.0339	0.0220	0.0352
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.2436	0.2197	0.2327
val	0.2276	0.1477	0.1213
test	0.0214	0.0139	0.0277
Fold 3			
train	0.2698	0.2350	0.2328
val	0.3276	0.3275	0.3782
test	0.1069	0.0863	0.0822
Fold 4			
train	0.2670	0.2212	0.2336
val	0.1946	0.2362	0.3491
test	0.2230	0.2144	0.2216

Table 11: nDCG results for TD2003

	Train Time (s)	**Test Time (s)**
Fold 1	8.7133	0.2224
Fold 2	20.8312	0.3307
Fold 3	16.5113	0.1682
Fold 4	19.0213	0.2418

Table 12: Train and test times for TD2003

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.1808	0.1876	0.2646
val	0.1244	0.1058	0.1622
test	0.0000	0.0000	0.0106
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.2408	0.2600	0.3795
val	0.0753	0.0797	0.1238
test	0.0000	0.0000	0.0197
Fold 3			
train	0.2345	0.2702	0.3753
val	0.0498	0.0511	0.1058
test	0.0185	0.0120	0.0408
Fold 4			
train	0.2196	0.2448	0.3425
val	0.2780	0.2676	0.3957
test	0.0000	0.0000	0.0200

Table 13: nDCG results for TD2004

	Train Time (s)	**Test Time (s)**
Fold 1	47.5260	0.3589
Fold 2	48.3123	0.3662
Fold 3	88.5335	0.3064
Fold 4	53.4189	0.2962

Table 14: Train and test times for TD2004

4.2 Gradient Boosted Decision Trees

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	1.0000	1.0000	0.9999
val	0.4213	0.3822	0.3681
test	0.2700	0.2467	0.3080
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.9994	0.9922	0.9953
val	0.3661	0.3266	0.3679
test	0.2369	0.2303	0.2987
Fold 3			
train	1.0000	0.9910	0.9918
val	0.1863	0.1732	0.2610
test	0.2406	0.2426	0.3281
Fold 4			
train	0.9877	0.9809	0.9823
val	0.1489	0.1917	0.3115
test	0.2552	0.2943	0.4093

Table 15: nDCG results for TD2003

	Train Time (s)	**Test Time (s)**
Fold 1	12.1395	0.0395
Fold 2	12.3275	0.0275
Fold 3	12.9135	0.0135
Fold 4	13.5633	0.0633

Table 16: Train and test times for TD2003

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.5046	0.5274	0.6226
val	0.3221	0.3483	0.4859
test	0.4531	0.4227	0.5038
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.5290	0.5381	0.6343
val	0.4208	0.4166	0.5057
test	0.2619	0.3278	0.4395
Fold 3			
train	0.4444	0.4701	0.5601
val	0.2921	0.3489	0.4487
test	0.3894	0.4247	0.5299
Fold 4			
train	0.4746	0.4685	0.5686
val	0.4667	0.4952	0.5965
test	0.3141	0.3388	0.4869

Table 17: nDCG results for TD2004

	Train Time (s)	**Test Time (s)**
Fold 1	3.2815	0.0815
Fold 2	3.2546	0.0546
Fold 3	3.6659	0.0659
Fold 4	3.5339	0.0339

Table 18: Train and test times for TD2004

4.3 Multi-Layer Perceptron

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.3231	0.3076	0.3969
val	0.3365	0.3204	0.3103
test	0.2935	0.2802	0.3540
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.4828	0.4240	0.4991
val	0.3008	0.2826	0.3960
test	0.1817	0.1799	0.2872
Fold 3			
train	0.4699	0.4096	0.4636
val	0.1384	0.1243	0.2400
test	0.1612	0.2283	0.3189
Fold 4			
train	0.4380	0.3575	0.4402
val	0.1447	0.2104	0.3228
test	0.2700	0.3007	0.3582

Table 19: nDCG results for TD2003

	Train Time (s)	**Test Time (s)**
Fold 1	0.4859	0.0138
Fold 2	0.4884	0.0153
Fold 3	0.5116	0.0131
Fold 4	0.5297	0.0163

Table 20: Train and test times for TD2003

Fold 1			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.4665	0.4909	0.5906
val	0.3099	0.2942	0.4352
test	0.3934	0.4369	0.4960
Fold 2			
	nDCG[5]	**nDCG[10]**	**nDCG[100]**
train	0.4285	0.4472	0.5596
val	0.4543	0.4559	0.5220
test	0.3270	0.3797	0.4775
Fold 3			
train	0.3375	0.3720	0.4844
val	0.3701	0.3875	0.4712
test	0.3615	0.3855	0.5002
Fold 4			
train	0.4086	0.4128	0.4969
val	0.3649	0.3827	0.4973
test	0.2129	0.2530	0.3633

Table 21: nDCG results for TD2004

	Train Time (s)	**Test Time (s)**
Fold 1	0.7266	0.0207
Fold 2	0.7213	0.0204
Fold 3	0.7468	0.0192
Fold 4	0.7324	0.0216

Table 22: Train and test times for TD2004

5 Bonus Task

In this part, a few adjustments have been done. First of which, only the **first 60** features for each query-document pair vector. This greatly helps in the runtime as well as the accuracy is also good. Some other methods like **PCA** was also tried after incorporating all features, but that did not really yield much satisfactory results.
The following section details out the hyperparameters used and where they differ from the ones used in the previous parts.

5.1 Hyperparameters

In the **Support Vector Regressor** model, the dimensionality is reduced to **50** using **PCA**, resulting in an improved nDCG value. Additional optimizations, such as setting `max_iter=1000` to address runtime concerns, were also implemented. The final average nDCG value demonstrates satisfactory performance following these optimizations.

kernel	C	epsilon	degree	max_iter	shrinking	gamma
'Linear'	1.0	0.001	3	1000	True	'auto'

Table 23: SVR hyperparameters

In the **Gradient Boosted Decision Tree** model, due to the increased training size, `n_estimators` has been reduced to **100**, while `learning_rate` has been increased to **0.1**. These adjustments enhance runtime performance and yield a satisfactory average nDCG.

subsample	n_estimators	learning_rate	max_depth	criterion	random_state	max_features	alpha
1.0	200	0.1	2	'friedman_mse'	21	'sqrt'	0.8

Table 24: GBDT hyperparameters

In the **Multi-Layer Perceptron** model, no scaling is applied. The primary changes compared to the previous parameters are a reduction in the maximum number of iterations to **500** and an increase in the initial learning rate to **0.01**. These adjustments were made because the number of examples is significantly larger, which helps improve the runtime while still achieving satisfactory results, with an average nDCG greater than **0.7**.

hidden_layer_sizes	activation	solver	max_iter	alpha	learning_rate	learning_rate_init	power_t	random_state
(100,)	'logistic'	'adam'	500	0.01	'adaptive'	0.001	0.5	0

Table 25: MLP hyperparameters

5.2 Results

	nDCG[5]	nDCG[10]	nDCG[100]
SVR	0.5554	0.6269	0.7688
GBDT	0.6792	0.7250	0.8252
MLP	0.6821	0.7283	0.8281

Table 26: nDCG results on validation set

	Train Time (s)	Validation Time (s)
SVR	19.7673	3.0059
GBDT	86.0809	0.2542
MLP	57.9594	0.1773

Table 27: Time taken by the models

List of Tables

List of Figures