Selecting DaoOpt solver configuration for MPE problems

Summer project report
Submitted by: Abhisaar Sharma
Submitted to: Dr. Rina Dechter

Abstract

We address the problem of selecting the best configuration out of a set of configurations for DaoOpt - an AND/OR Branch-and-Bound search based solver for combinatorial optimization problems expressed as MPE queries over graphical models. DaoOpt takes different parameters for solving a problem, the running time of a problem depends on the configuration chosen for it. We want to learn for a given problem instance which configuration is the fastest to solve it and predict the configuration likely to solve a new problem in the least amount of time. The results indicate that our predictor is able to improve the running time of the problems.

1 Introduction

We start by describing the basic supervised learning terminology. The learning domain is an arbitrary set X from which samples $x \in X$ are drawn, according to an unknown distribution. Samples are represented by a set of features which we extract from it. The set of target values Y gets assigned to each sample by a function $f : X \to Y$. Given a set S of training data, $(x_j, y_j) \in X \times Y$, the learning task is to use the training data to find a function $f^* : X \to Y$, called a model that can be used to predict for a new sample $x \in X$ it's target y using the sample features $\varphi(x)$. If Y is continuous or quantitative we have a regression learning problem. Let C be a set of configurations and $c \in C$ be a particular configuration. We train $|C|$ different models f_c^* on (X, Y_c), where target Y_c is configuration dependent. In our case the target Y_c is the time taken to solve a problem under a configuration c. To predict the configuration likely to solve a new problem x in the least amount of time, we choose $\underset{c}{\operatorname{argmin}} f_c^*(\varphi(x))$.

2 Methodology

The experiments were done on a problem base which included different problem classes such as Grid-Networks, Linkage analysis(pedigree), Protein folding, Image alignment etc[1] . and past UAI problems. We used CVO to generate variable orderings on the problem instances. CVO is a tool for generating variable orderings using min-fill heuristic and implements the Iterative Greedy Variable Ordering Algorithm. Let $C(X_i)$ denote the context of a variable X_i in a given AND/OR search space and let D_i denote the domain of variable X_i. The state space size, is then expressed as $\sum_{i=1}^{n} |D_i| \cdot \prod_{X_j \in C(X_i)} |D_j|$. Variable Elimination complexity is defined as the log of state space size. The following figure shows the histogram of the induced width of the problems and log State Space Size on the orderings generated by CVO after running it for 10^5 iterations per problem instance. The engineered features $\varphi(x)$ for the learning task are listed subsequently.

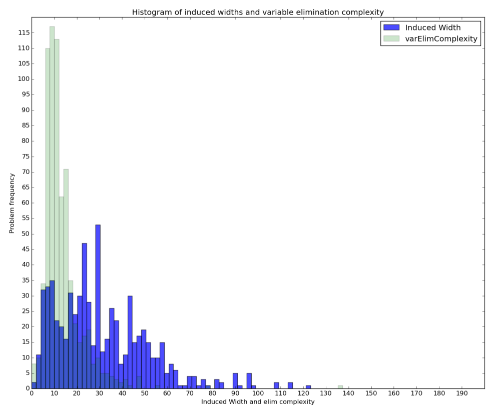


Figure 1: Histogram of induced widths and log State Space Size for the problem instances. The orderings are generated by running CVO for 10^5 iterations using min-fill heuristic.

[1]Problem Instances at `http://graphmod.ics.uci.edu/~abhisaar/mpeProblems/`

Problem Features

1. Width - Induced width of the problem
2. NumFactors - Number of factors in the problem
3. NumVar - Number of variables in the problem
4. PseudotreeDepth - Depth of the generated pseudo-tree
5. LowerBound - The minimum theoretical induced width
6. Complexity - log size of junction tree to store the problem
7. MaxFactorScope - Maximum scope of all the factors
8. MaxVarDomain - Maximum domain size of all the variables
9. MeanFactorScope - Mean value of the scopes of all the factors
10. MeanVarDomain - Mean of domain size of variables
11. MedianVarDomain - Median domain size of the variables
12. MinVarDomain - Minimum domain size of the variables
13. DevFactorScope - Standard deviation in factor scopes
14. DevVarDomain - Standard deviation in variable domain size

The following set of configurations were tailored which are suitable for solving easy to hard problems. The starting configurations have weaker heuristics but are faster to compute, making them suitable for easier problems.

Table 1: List of configurations for running the experiments differentiated by SLS iterations and time per iteration, number of MPLP iterations and the Max-time for running MPLP, number of JGLP iterations and the Max-time for running JGLP and the Ibound of each problem. Max Ibound indicates the solver will run with the highest possible Ibound given the memory limit.

Config	ibound	SLS it/time	MPLP it/time	JGLP it/time
2	Max-3	0/0	0/0	0/0
3	Max-1	0/0	0/0	0/0
4	Max-1	1/2	1/5	0/0
5	Max	10/5	1000/30	500/30
6	Max	20/5	1500/45	750/45
7	Max	20/10	2000/60	1000/60
8	Max	20/10	3000/90	1500/90
9	Max	30/10	4000/120	2000/120
10	Max	50/10	8000/240	4000/240

Initially, configurations also had different number of iterations to generate the orderings in the parameters, which were removed due to the large variance in the resulting running times. The set of 1,008 problems was run on each of these configurations with a time-out of 24 hours. The variable orderings for the problem were generated by running CVO for 10,000 iterations. The resulting data was used to train predictors. We studied the performance of lasso regression, ridge regression, Boosted decision tree and k-nearest neighbours regression. Using 5 fold cross-validation, we selected the hyper-parameters for each of the regressors. The problems that did not finish to completion were associated with a fixed time-out value.

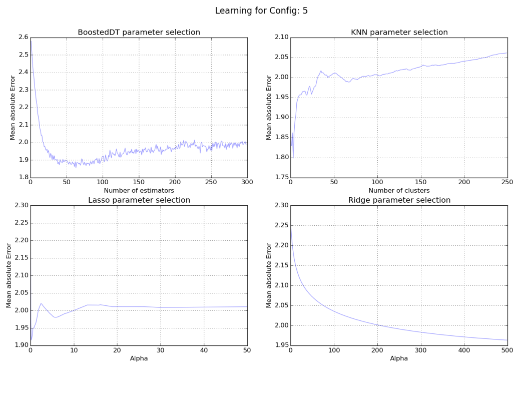


Figure 2: Parameter selection for Configuration 5. Clockwise from top-left we plot the Mean Absolute Prediction Error versus (a) Number of estimators for the Boosted Decision Tree Regression (b) Number of nearest neighbours for the KNN Regression (c) Regularization parameter for Ridge Regression (d) Regularization parameter for Lasso Regression.

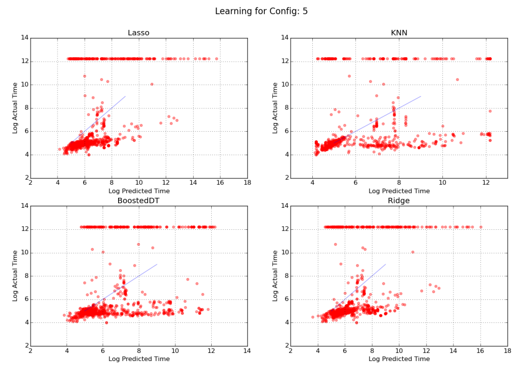


Figure 3: Time prediction for Configuration 5. Clockwise from top-left we plot the log time taken to solve a problem versus (a) Log predicted time for the Lasso Regression (b) Log predicted time for the KNN Regression (c) Log predicted time for the Ridge Regression (d) Log predicted time for the Boosted Decision Tree Regression.

3 Results

For evaluating the performance of our configuration selection, we compare the running times of each problem instance when using the configuration picked by our predictor, versus a fixed configuration from the set of configurations. Problems which were not solved completely within the 24 hour time-out are assigned a completion time of 86400 seconds. In general, we observed that configurations with higher FGLP/MPLP/SLS iterations perform slower but solve more problems. Configurations which don't use these pre-processing routines tended to solve easier problems much faster but timed out on harder problems. As measures to illustrate this trade-off, we compute for each configuration the average adjusted time, defined as the average of sum of times taken to solve problems finished to completion and the number of unsolved problems, denoted by U multiplied by a fixed penalty t_p (86400 secs).

$$AverageAdjustedTime = 1/N(\sum_i (t_i) + U * t_p) \quad (1)$$

$$AverageTime = \sum_i t_i/(N - U) \quad (2)$$

where the i^{th} problem was solved to completion. Improving both these metrics could indicate that we are solving more problems in lesser amount of time. We measure the number of exact matches, which happen if the predictor chose the configuration which would have taken the least amount of time, or if the predictor assigned a configuration 9 or 10 for problems that could not be solved.

We also analyse the pairwise performance of each individual configuration against our models predicted configuration. We compute the average sum of time differences from choosing our predictors configuration versus a fixed configuration over all instances that solved to completion. We also measure the average adjusted time gain per problem, where we also include unsolved problems, as described above. Additionally we measure for how many problem instances the model's predicted configuration performed better than a given configuration.

Table 2: Prediction Performance of configurations versus the learnt predictors and an oracle on various metrics.

Config	Unsolved Problems	Average Time	Average Adj. Time	Exact Matches
5	222	404	19400	-
6	222	430	19420	-
7	214	673	18927	-
8	214	655	18913	-
9	211	827	18793	-
10	212	1231	19197	-
BDT	211	491	18528	684
Lasso	214	491	18784	799
Ridge	214	450	18752	797
Knn	220	370	19202	771
Oracle	210	401	18371	-

The results in table 2 show that Lasso predictors have the most number of exact matches. The performance of Boosted decision trees was enhanced using feature selection. It solves the same number of problems as the hardest configurations, while reducing the average time needed to solve a problem by more than half.

The following sections show the comparison of the predictors performance versus the configurations.

3.1 Lasso Predictor Results

Table 3 compares the performance of the lasso predictor and the fixed configurations on the problem instances. The lasso predictor performs better or equal on most of the problem instances across each configuration and results in a time gain per problem.

Table 3: Comparison of Lasso predictor and the fixed configurations. The predictor performs better across all the configurations.

Config	Lasso Better or Equal	Lasso Worse	Time Gain /problem	Adj. Time Gain /problem
5	777	14	10 sec	616 sec
6	771	21	35 sec	636 sec
7	776	16	183 sec	144 sec
8	772	21	166 sec	129 sec
9	780	16	312 sec	9.4 sec
10	786	8	741 sec	414 sec

3.2 Decision Tree Predictor Results

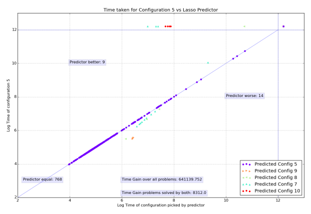
Table 4 compares the performance of the Decision tree and the configurations on the problem instances. The Decision tree is able to solve the same number of problems as the hardest configurations. However the number of exact matches and the number of problem instances where it was better than a given configuration are lesser than the lasso predictor.

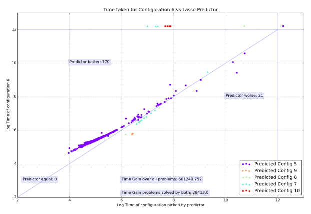
Table 4: Comparison of Decision Tree predictor and the fixed configurations.

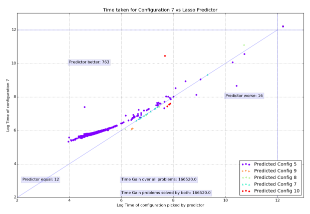
Config	BDT Better or Equal	BDT Worse	Time Gain /problem	Adj. Time Gain /problem
5	662	132	-28 sec	873 sec
6	715	79	-2.4 sec	893 sec
7	742	52	183 sec	400 sec
8	759	35	168 sec	386 sec
9	774	21	314 sec	266 sec
10	789	5	741 sec	670 sec

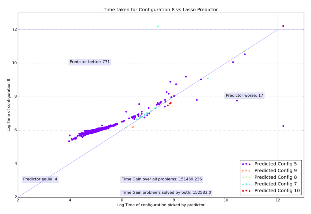
3.3 Ridge and KNN Predictor Results

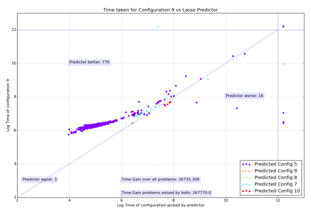
The performance of Ridge regression and KNN regression was not as good as the Lasso regression and KNN. Their prediction results are included in the Appendix.


(a) Lasso predictor vs Configuration 5


(b) Lasso predictor vs Configuration 6


(c) Lasso predictor vs Configuration 7


(d) Lasso predictor vs Configuration 8


(e) Lasso predictor vs Configuration 9

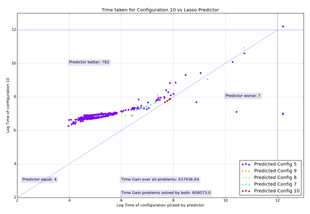
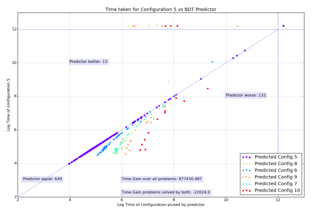
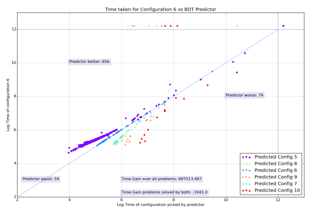
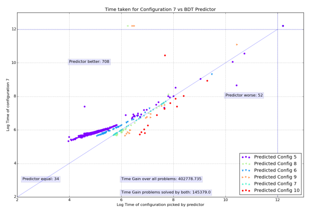
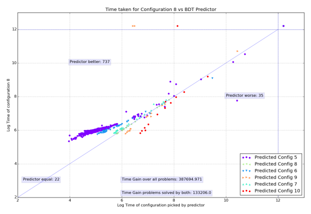

(f) Lasso predictor vs Configuration 10

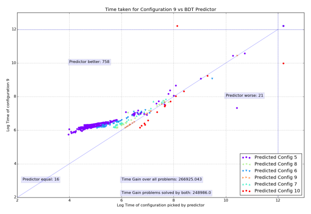
Figure 4: Plots showing log time for solving the problem instance using the Lasso model's predicted configuration versus the fixed configurations for all the problem instances. Lasso predictor's performance is better than the individual configurations for majority of the problems resulting in both an average and adjusted time gains.


(a) BDT predictor vs Configuration 5


(b) BDT predictor vs Configuration 6


(c) BDT predictor vs Configuration 7


(d) BDT predictor vs Configuration 8


(e) BDT predictor vs Configuration 9

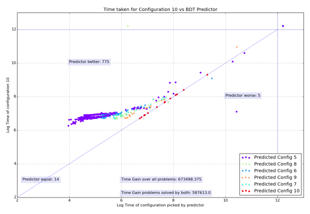

(f) BDT predictor vs Configuration 10

Figure 5: Plots showing log time for solving the problem instance using the predictors configurations versus the fixed configurations for all the problem instances. BDT solves the most number of problem instances, only 1 problem less than the oracle's performance in the number of problems solved completely.

4 Conclusion

We have showed that Lasso Regression and Boosted Decision Trees are able to predict the configurations likely to solve a new problem instance and in lesser time than using a fixed set of configurations over all the problems, which do not adjust according to the problem instance. Our predictors use the problem features to predict which configuration can solve a problem fastest, leading to large time improvements for our solver.

5 Side Activities

The other side activities done during the project involved comparing the performance of DaoOpt and Toolbar[2]

which showed Toolbar performed better in most problem classes on their benchmarks. DaoOpt was run with 500 FGLP iterations, 10 seconds as FGLP total time, 250 JGLP iterations, 10 seconds as JGLP total time, 5 SLS iterations

[2]Results available at `http://graphmod.ics.uci.edu/~abhisaar/plots/toolbox_BTD_plt/`

with 2 seconds as time per SLS iteration. However their performance still needs to be tested on our benchmarks.

An interesting observation was the lower bound on the induced width generated by CVO had the most feature weight in the lasso predictor, with the problem induced width as the second. To improve the learning performance, adding the feature $LowerBound - UpperBound$ was considered, where these bounds are MPE solution bounds. However it did not lead to any increase in prediction accuracy. What remains to be explored is if the normalized measure $(L-U)/L$ leads to any increase in prediction performance. A reasonable notion of MPE lower bounds developed was running SLS for a short period of time, in case it does not find a lower bound we could take the log of minimum non-zero values out of each factor table and add them to generate a loose lower bound. If a solution exists it will always be above this lower bound - hence it can be used as a reference.

We also tried to see if our predictor would lead to an increase in the any-time performance. We plotted the log solution cost versus log time for all the problems and compared the area of a solution generated by a configuration to that of our predictor. The area of a solution was calculated with the lower bound as described above with the log time window of 0 to 12. The results indicated the predictor could not lead to increases in the anytime area. It would be interesting to see if we can train a predictor for selecting a configuration to maximize the anytime performance.

Another side activity was observing performance of orderings which minimize a primary criteria and then a secondary criteria, for example finding orderings which minimize induced widths and amongst such orderings selecting ones with minimum state space size. Though we could not see a specific pattern in these with the limited examples we tested on, problems of a specific type seemed to always work better when the same criteria were being minimized. It may be possible to predict on which criteria we should generate the ordering based on the problem.

6 Appendix

6.1 KNN Predictor Results

Table 5 compares the performance of the KNN predictor and the fixed configurations on the problem instances. KNN is not able to recognise hard problems and has the highest number of unsolved problems.

6.2 Ridge Predictor Results

Table 6 compares the performance of the Ridge predictor and the fixed configurations on the problem instances. The performance of Ridge is worse than Lasso predictor, although it solves more problems than the KNN predictor.

Table 5: Comparison of KNN predictor and the fixed configurations.

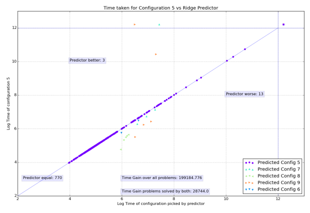
Config	KNN Better or Equal	KNN Worse	Time Gain /problem	Adj. Time Gain /problem
5	773	13	37 sec	198 sec
6	778	18	62 sec	218 sec
7	771	22	173 sec	-273 sec
8	774	19	223 sec	-289 sec
9	780	15	382 sec	-409 sec
10	781	13	781 sec	-4 sec

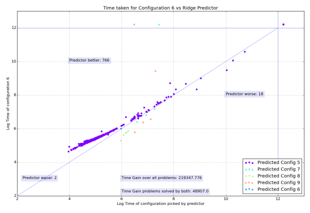
Table 6: Comparison of Ridge predictor and the fixed configurations.

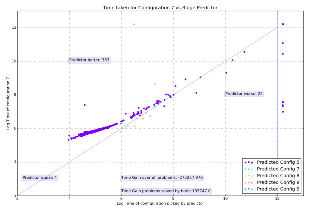
Config	Ridge Better or Equal	Ridge Worse	Time Gain /problem	Adj. Time Gain /problem
5	779	12	11 sec	532 sec
6	772	19	36 sec	552 sec
7	775	17	184 sec	60 sec
8	771	22	166 sec	45 sec
9	779	17	312 sec	-75 sec
10	785	9	742 sec	330 sec

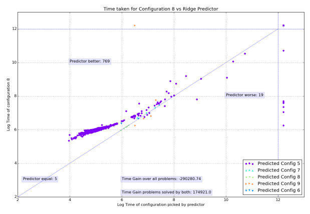
Parameter Selection and Prediction Plots

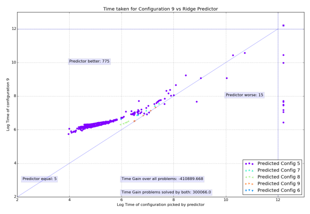
We plot the mean absolute prediction error versus the hyper-parameter for each of the predictor's configuration. Also shown are the plots of the predicted log time versus the actual log time taken to solve a problem by the predictor using the hyper-parameter giving the best cross-validation performance.


(a) KNN predictor vs Configuration 5


(b) KNN predictor vs Configuration 6


(c) KNN predictor vs Configuration 7


(d) KNN predictor vs Configuration 8


(e) KNN predictor vs Configuration 9

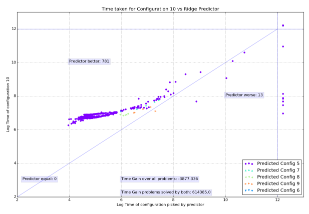
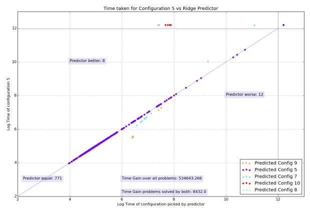
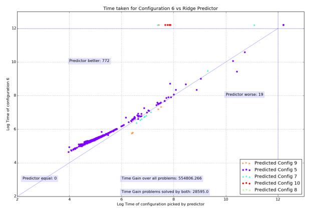
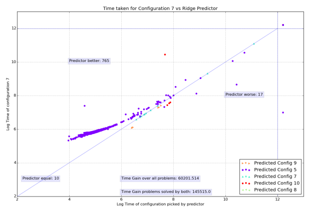
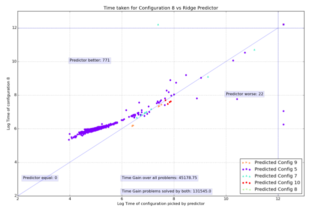

(f) KNN predictor vs Configuration 10

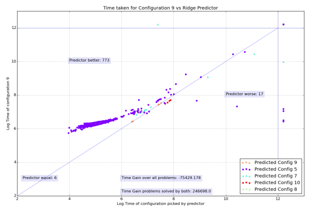
Figure 6: Plots showing log time for solving the problem instance using the KNN predictor's configurations versus the fixed configurations for all the problem instances.


(a) Ridge predictor vs Configuration 5


(b) Ridge predictor vs Configuration 6


(c) Ridge predictor vs Configuration 7


(d) Ridge predictor vs Configuration 8


(e) Ridge predictor vs Configuration 9

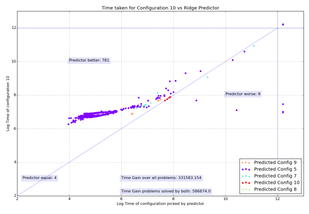

(f) Ridge predictor vs Configuration 10

Figure 7: Plots showing log time for solving the problem instance using the Ridge predictor's configurations versus the fixed configurations for all the problem instances.

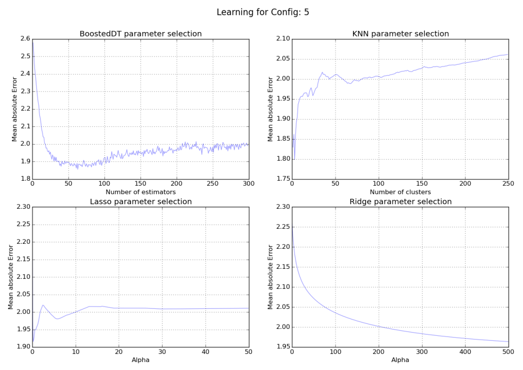


Figure 8: Parameter selection for configuration 5

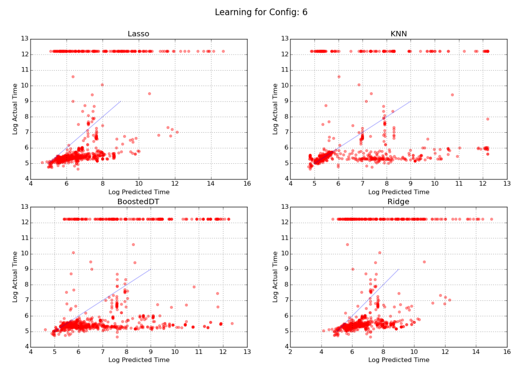


Figure 11: Time prediction for configuration 6

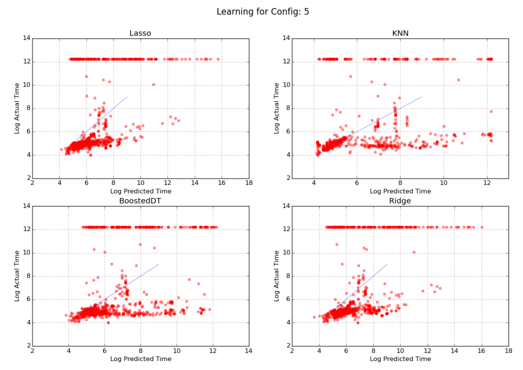


Figure 9: Time prediction for configuration 5

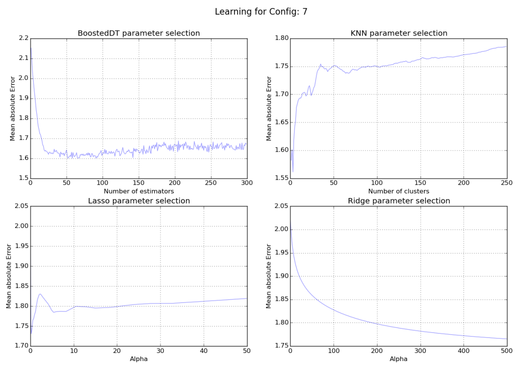


Figure 12: Parameter selection for configuration 7

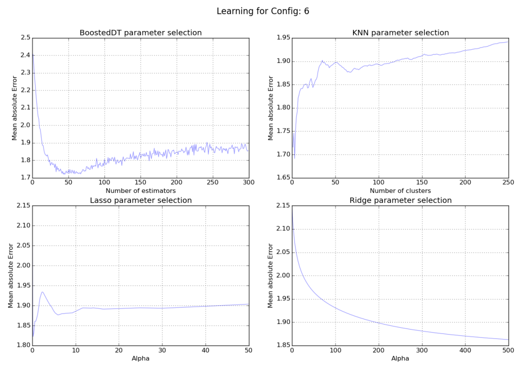


Figure 10: Parameter selection for configuration 6

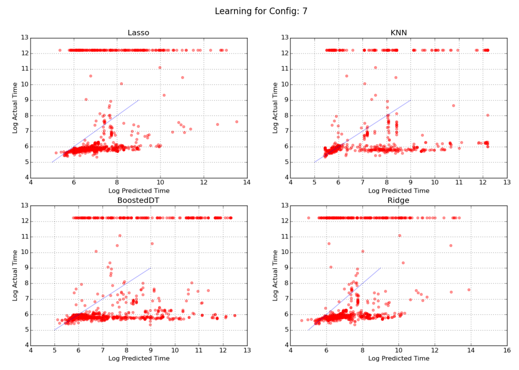


Figure 13: Time prediction for configuration 7

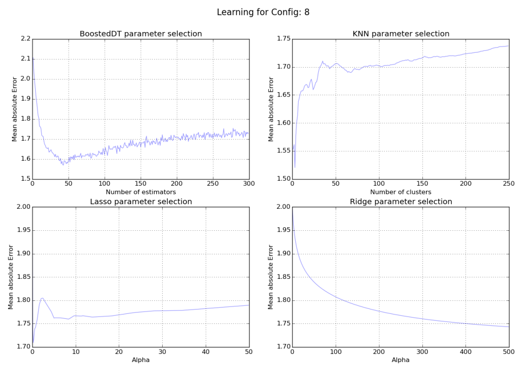


Figure 14: Parameter selection for configuration 8

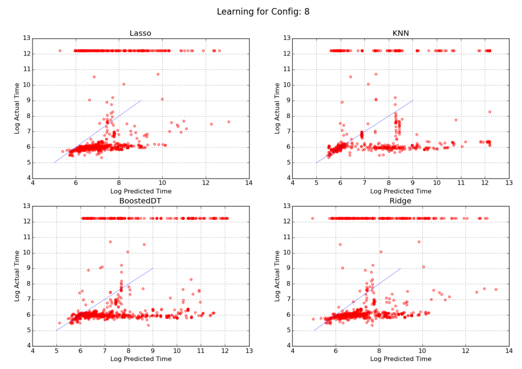


Figure 15: Time prediction for configuration 8

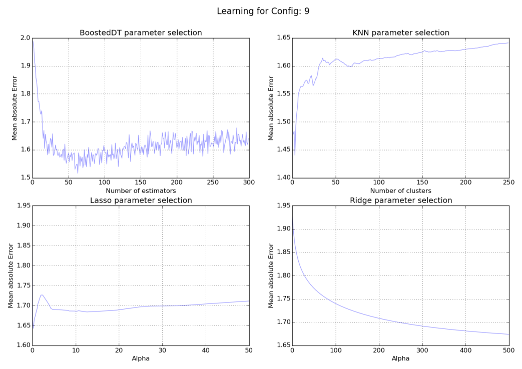


Figure 16: Parameter selection for configuration 9

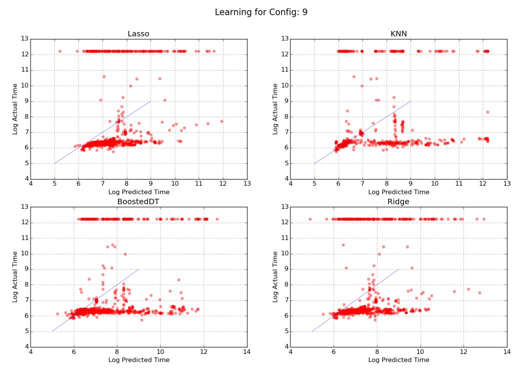


Figure 17: Time prediction for configuration 9

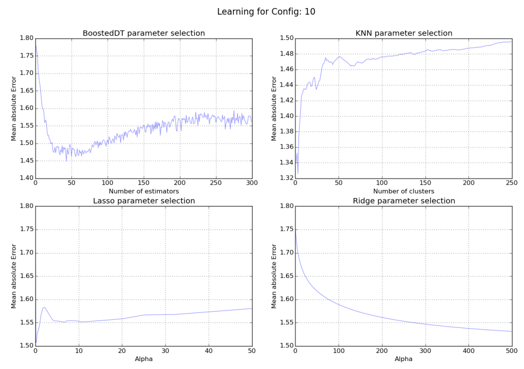


Figure 18: Parameter selection for configuration 10

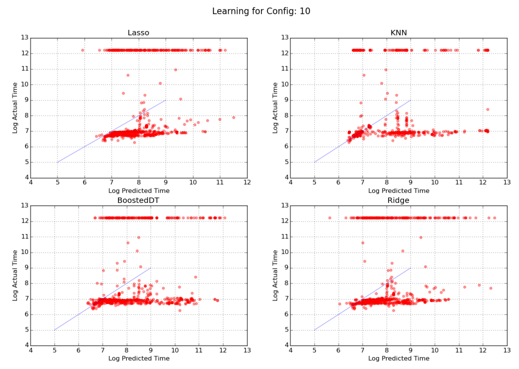


Figure 19: Time prediction for configuration 10