

ASSIGNMENT 2: SUPPORT VECTOR MACHINE, DECISION TREE & BOOSTING

Introduction:

The goal of the project is to implement Support Vector Machine, Decision Tree and Boosting algorithm on two different datasets to predict target variable and compare the performance of algorithms based on various hyper-parameters. To understand the behavior of algorithms better, the datasets are chosen based on:

- **High observations and Less Features:** Categorical(High/Low) GPU run time(in milliseconds) of matrix product matrix product where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 261400 possible parameter combinations.
- **More features and Less Observations:** IBM Employee Attrition based on features such as performance, tenure, education, job satisfaction etc.

GPU Run-time Data

Data Description:

This data set measures the running time of a matrix-matrix product $A*B = C$, where all matrices have size 2048 x 2048, using a parameterizable SGEMM GPU kernel with 241600 possible parameter combinations. For each tested combination, 4 runs were performed, and their results are reported as the 4 last columns. All times are measured in milliseconds

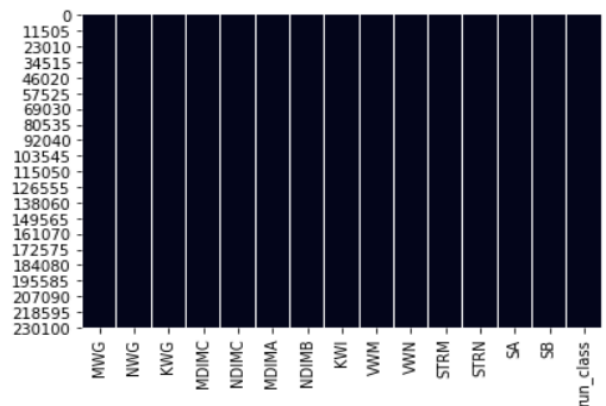
Dataset consists of 241600 observations on 18 variables of which the first 14 variables are independent features and the last 4 variables are the run-times of 4 different runs measured in milliseconds. For the goal of the project, the average run-time is calculated and categorized into two classes based on the median run time.

Data Analysis:

Correlation Plot

	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	run_class
MWG	1	0.0006	0.0093	0.11	-0.0086	0.16	0.015	0	0.35	-0.0008	0.41
NWG	0.0006	1	0.0093	-0.0086	0.11	0.015	0.16	0	-0.0008	0.35	0.23
KWG	0.0093	0.0093	1	0.15	0.15	-0.035	-0.035	-0	-0.012	-0.012	-0.022
MDIMC	0.11	-0.0086	0.15	1	-0.21	0.2	0.085	-0	-0.13	0.011	-0.19
NDIMC	-0.0086	0.11	0.15	-0.21	1	0.085	0.2	-0	0.011	-0.13	-0.15
MDIMA	0.16	0.015	-0.035	0.2	0.085	1	0.088	-0	-0.2	-0.019	-0.013
NDIMB	0.015	0.16	-0.035	0.085	0.2	0.088	1	-0	-0.019	-0.2	-0.037
KWI	0	0	-0	-0	-0	-0	-0	1	-0	-0	-0.0052
VWM	0.35	-0.0008	-0.012	-0.13	0.011	-0.2	-0.019	-0	1	0.0012	0.18
VWN	-0.0008	0.35	-0.012	0.011	-0.13	-0.019	-0.2	-0	0.0012	1	0.084
run_class	0.41	0.23	-0.022	-0.19	-0.15	-0.013	-0.037	-0.0052	0.18	0.084	1

Missing Value Plot



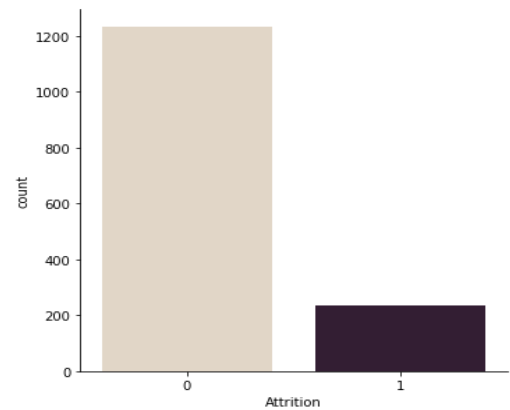
Null values: The data looks clean and the missing value plot has no gaps i.e., the data has no missing values.

Correlation: Since we are experimenting on SVM and Decision Tree algorithms, the correlation between the features is an important observation. For SVM, the effect is similar to that of multicollinearity in linear regression for linear kernel. For decision trees, the algorithm doesn't make assumptions on the relationship between features and if features are highly correlated, no information will be gained, and features gets ignored for split.

IBM HR Analytics Employee Attrition & Performance

Data Description:

- This is the fictional data created by IBM data scientists regarding their employees and their attrition.
- It is interesting to know what causes employees to quit even though some are highly paid and satisfied with the job.
- The dataset describes the details of IBM employees such as age, department, work experience, job satisfaction working hours etc.
- Dataset consists of 1470 observations with 34 independent variables and attrition as dependent variable.
- There is an imbalance in the target variable. Oversampling or under-sampling technique will be performed based on experimentation and evaluation using decision tree



Data Preparation:

Among 34 explanatory features, 6 features are nominal and remaining features are numerical. Since python-sklearn doesn't handle categorical inputs, using pandas-get_dummies for nominal variables and LabelEncoder for ordinal variables. So, the transformed dataset has 1470 observations 52 features. The data looks clean and doesn't have any null values for imputation.

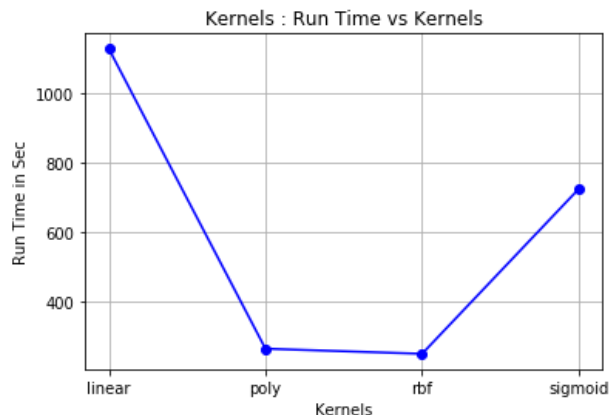
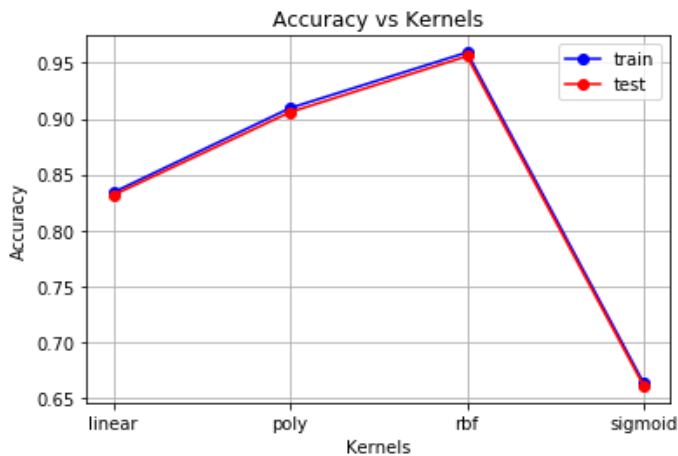
EXPERIMENTATION – SVM for GPU Run-time Data

Overview:

For SVM, different kind of experiments are performed – Varying Kernels, Varying degree of polynomial for polynomial kernel, varying gamma values for RBF kernel, varying penalty parameter C for RBF kernel, K-fold cross validation with varying kernel and number of folds.

- ❖ Theoretically, SVM works well on small data with lots of features, and not too many instances, but not for big data because storing the kernel matrix requires memory that scales quadratically with the number of data points.
- ❖ Training time for traditional SVM algorithms also scales super linearly with the number of data points
- ❖ Same can be observed from the run-time plots of different experimentations below.

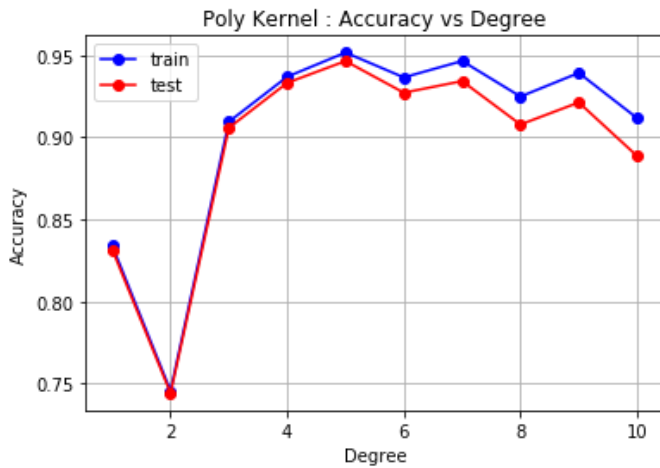
Experiment 1: Varying Kernels



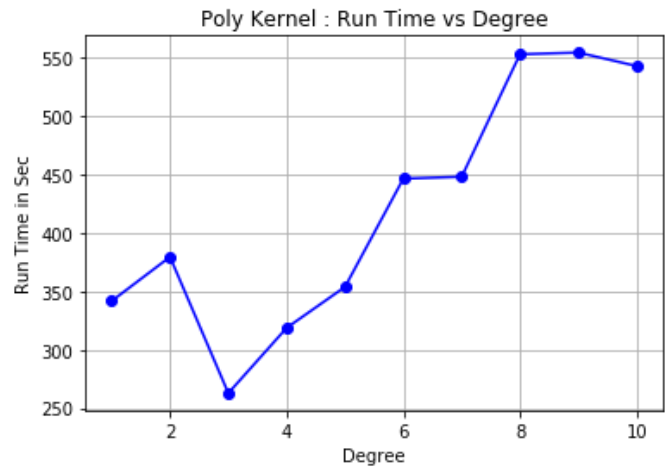
- ❖ Among the kernels chosen, RBF performed better for training data and test data while sigmoid performed worse.
- ❖ It can be inferred that data is non-linearly separable since RBF kernel performed lot better than linear kernel

- ❖ From the above plot, we can observe that poly(3rd order) and rbf kernels run faster.
- ❖ This might be due data is well separated using rbf, the optimization process is able to find the boundary much more easily than linear. Rbf in this case needs to search less space than linear so rbf takes less time.

Experiment 2: Varying Polynomial Degree



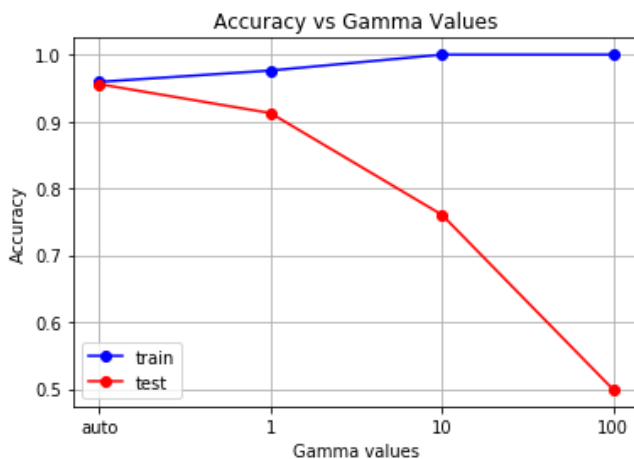
- ❖ Clearly it can be observed that, the relationship between features and target variable is not linear as the train and test accuracy increases with increasing the degree of polynomial.



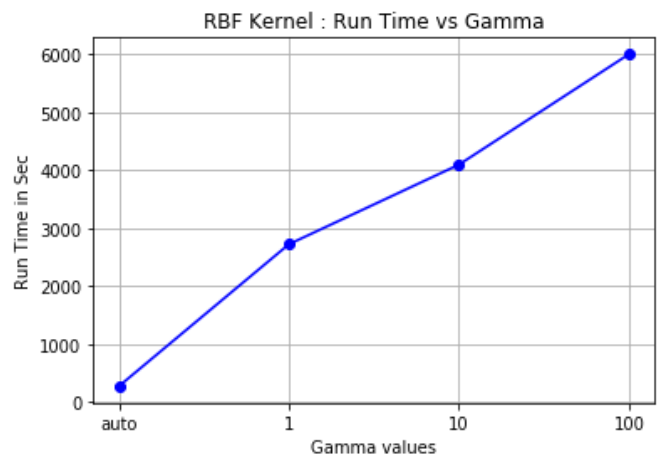
- ❖ Optimal degree can be observed at 5 and it is also not overfitting since the in-sample and out-sample errors are similar

Experiment 3: Varying 'gamma' parameter for rbf kernel

Theoretically, higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.



- ❖ As we increase the gamma value the training accuracy increases but out-sample error is decreased. Hence overfitting

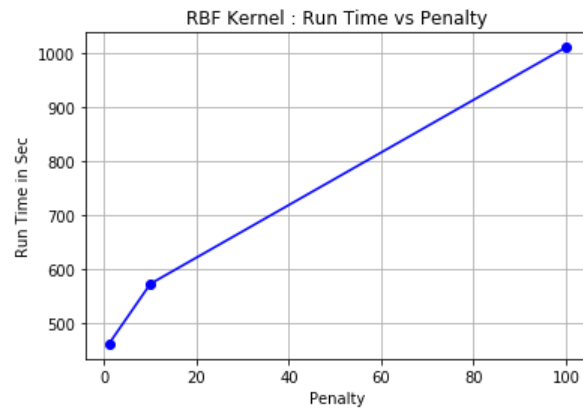
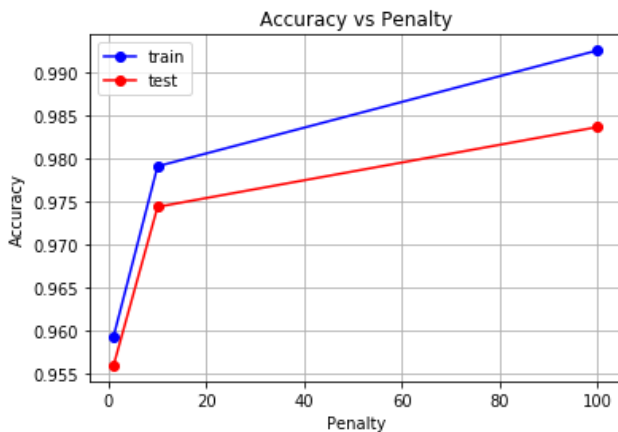


- ❖ It can be observed that small gamma will give low bias and high variance while a large gamma will give higher bias and low variance.

Experiment 4: Varying 'Penalty parameter C of error term' for rbf kernel

Theoretically, C is the cost of misclassification.

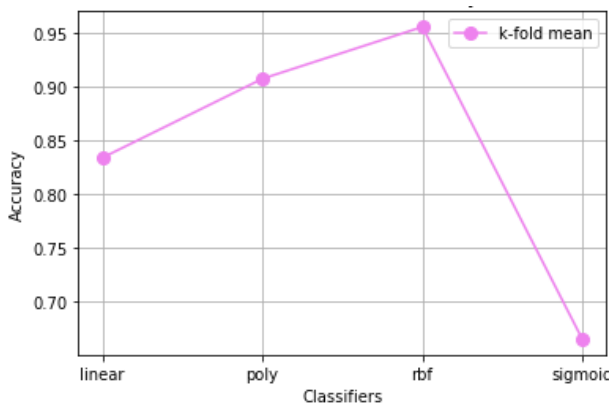
- ❖ A large C gives low bias and high variance. Low bias because it penalizes the cost of misclassification a lot.
- ❖ A small C gives you higher bias and lower variance.



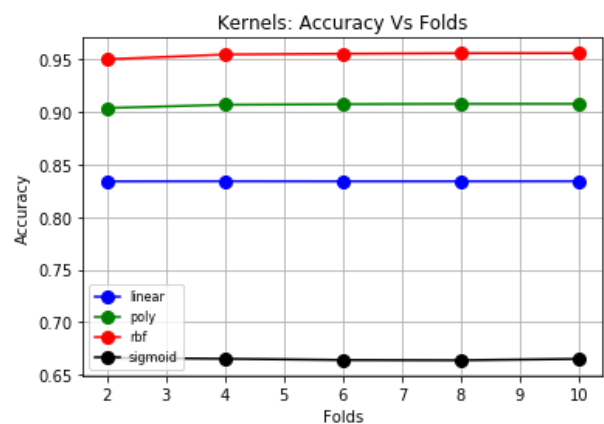
- ❖ Small C makes the cost of misclassification low ("soft margin"), thus allowing more of them for the sake of wider "cushion".
- ❖ Large C makes the cost of misclassification high ("hard margin"), thus forcing the algorithm to explain the input data stricter and potentially overfit.
- ❖ Even though we increase the C value, the test error didn't drop. This can be implied that the data is well separated
- ❖ Only trade-off is run-time is increased with increasing C value.

Experiment 5: K-fold Cross validation by varying kernels and varying folds

Varying Kernels:



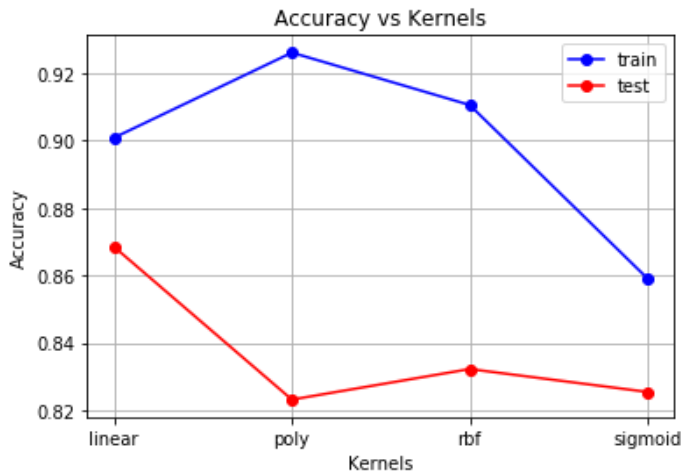
Varying number of folds:



- ❖ From the above experiments, we observed that data is well separated. So cross validation didn't improve much on the normal fit for any of the kernels.
- ❖ There is no much variation in the performance w.r.t folds for any of the kernels

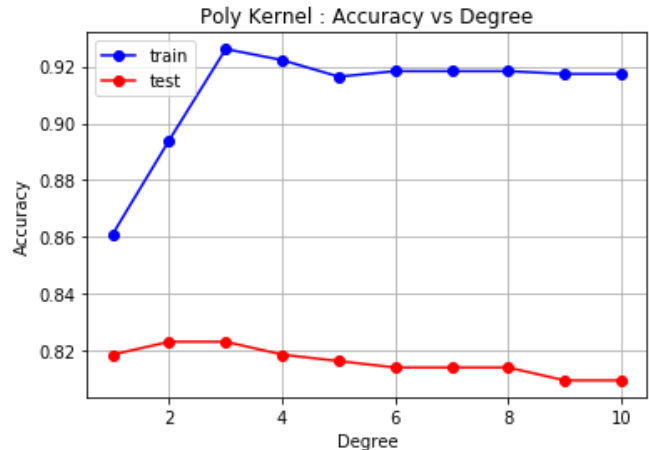
EXPERIMENTATION – SVM for IBM Data

Experiment 1: Varying Kernels



- ❖ Among the kernels chosen, Polynomial kernel(3rd order) performed better for training data and worse for test data i.e., overfitting
- ❖ But the algorithm is performing relatively better with linear kernel for train and test data.

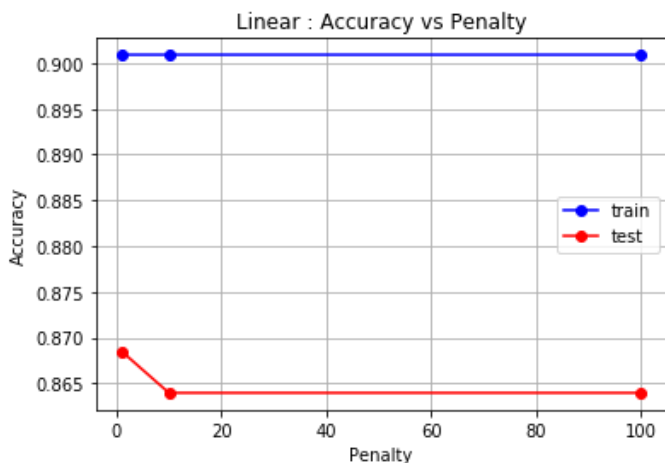
- ❖ Experimenting with Poly kernel by varying degree to see if it performs better than linear kernel



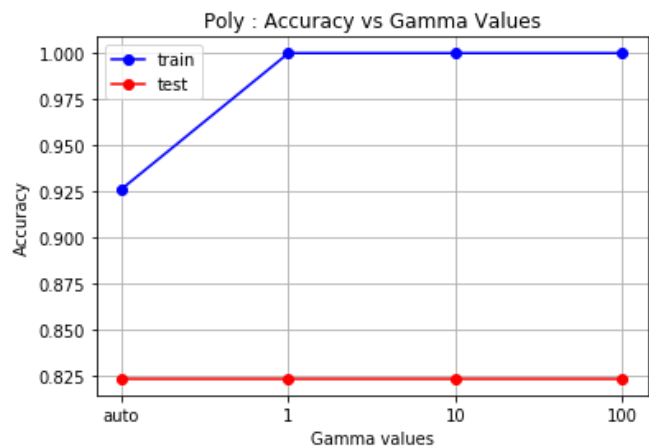
- ❖ From the above plot we can confirm that, SVM performs worse with poly kernel of higher degrees.
- ❖ So, it can be concluded that the data might be linearly separable.

Experiment 2: Varying Gamma for poly kernel and C parameters for linear kernel:

Now, let's experiment with parameters of linear kernel to increase the out-sample accuracy and also by changing gamma values for poly kernel

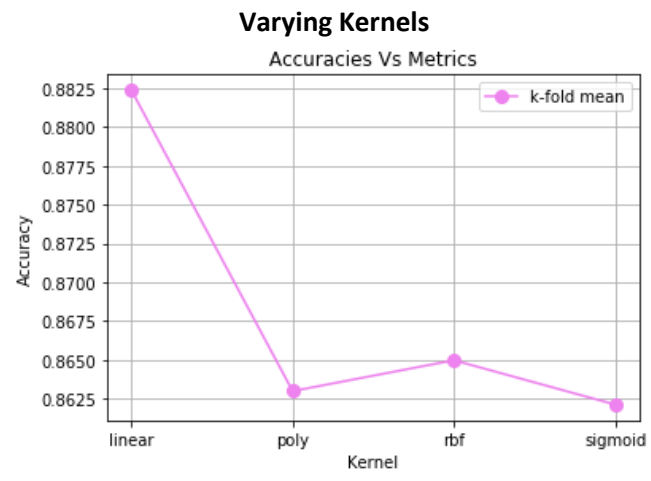


- ❖ By increasing the misclassification penalty C, there is no change in train accuracy, but the test accuracy is decreased.

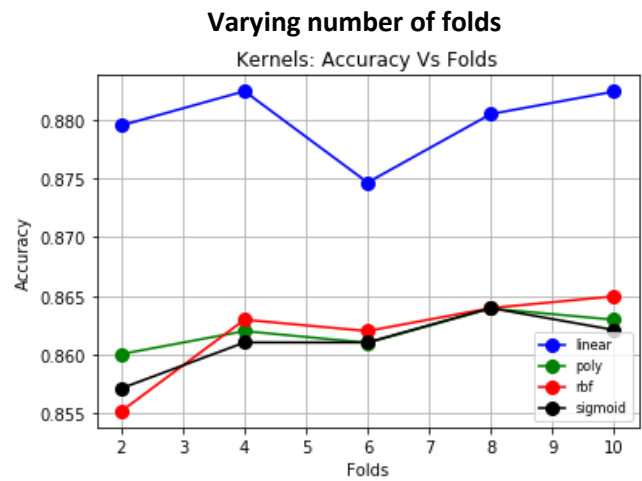


- ❖ For the poly kernel, there is no change in test accuracy with increasing gamma values.
- ❖ It can be concluded that linear kernel is the best kernel for this data

Experiment 2: Cross Validation:



- ❖ From the above experiments, we observed that data is well separated linearly. So cross validation didn't improve much on the normal fit for any of the kernels.



- ❖ With increasing number of folds, the accuracy increased for poly, rbf and sigmoid kernels but it is negligible

EXPERIMENTATION – Decision Tree and Boosting

GPU RUN-TIME DATA

Overview:

Decision tree is simple to understand and provide a clear visual to guide the decision-making process. But it has serious disadvantages : Overfitting, Bias error and Variance error.

To overcome those disadvantages, boosting technique is implemented. **Boosting** is based on weak learners (high bias, low variance). In terms of decision trees, weak learners are shallow trees, sometimes even as small as decision stumps (trees with two leaves). Boosting reduces error mainly by reducing bias (and also to some extent variance, by aggregating the output from many models).

For this project, let's compare the performance of decision tree by performing these experiments – Varying Criterion (Gini /entropy), varying training size, K-fold cross validation with varying metric & number of folds and pruning by reducing depth of tree, restricting the size of sample leaf and reduce the number of leaf nodes and compare with the boosted tree which is implemented by gradient to optimize the loss function.

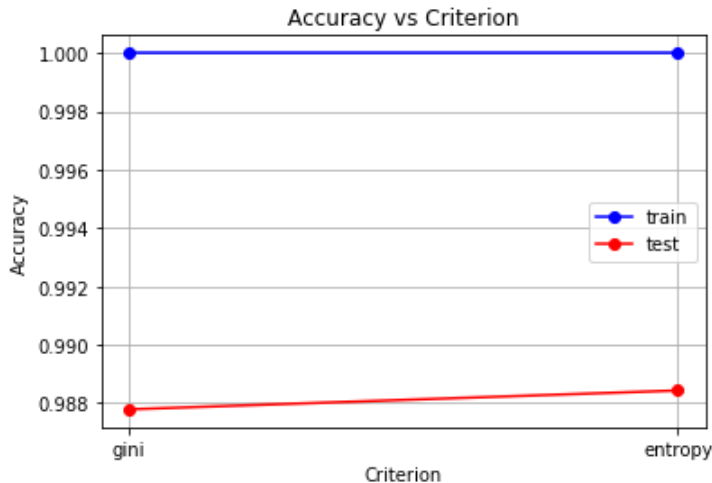
For boosting, I performed hyper-parameter tuning using XGBoost & LightGBM:

XGBoost	LightGBM
<pre>'model': 'XGBoost', 'parameter search time': '5:54:32.412000', 'accuracy': 0.992, 'test auc score': 1.0, 'training auc score': 1.0, 'learning_rate': 0.13509889841790165, 'm_child_weight': 1.0, 'max_depth': 12.0, 'n_estimators': 525.0, 'subsample': 0.9}}</pre>	<pre>'model': 'LightGBM', 'parameter search time': '0:43:41.275000', 'accuracy': 0.992, 'test auc score': 1.0, 'training auc score': 1.0, 'learning_rate': 0.061170438915614314, 'max_depth': 13.0, 'min_split_gain': 0.00012548669447202954, 'n_estimators': 1000.0, 'num_leaves': 117.0</pre>

Let's confirm these using different experiments and learning curves below:

Experiment 1: Decision Tree : Varying Criterion and Varying Train Size

Different Criterion



- ❖ It is evident from SVM that dataset is clearly separable in higher dimension, so it is expected that Decision tree performs much better which can be observed from the above graph.
- ❖ There is no much difference in the accuracy w.r.t criterion and tree didn't overfit as we can observe much higher accuracy with testing data.

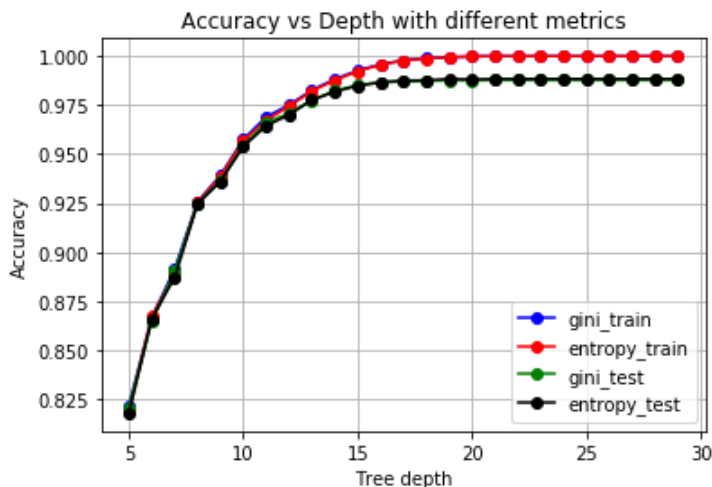
Varying Train Size



- ❖ It can be inferred from the above plot that generalized tree is formed with adding more data as the out-sample accuracy is increasing
- ❖ The selected metric didn't have much impact on the accuracy

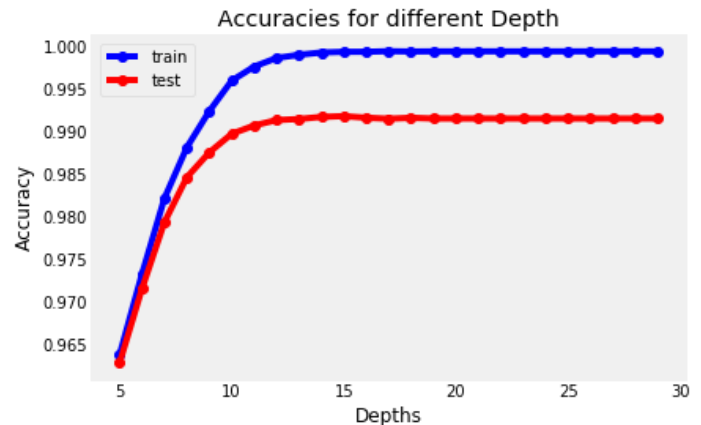
Experiment 2 : Pruning for Decision Tree vs Boosting for varying depth

Decision Tree



- ❖ Here we are observing less accuracy for low depth decision tree because decision tree too little flexibility to capture the patterns and interactions in the training data
- ❖ As the tree depth increases, the decision tree might simply overfit the training data without capturing useful patterns. But here test accuracy is also increasing, so the tree is generalizing rather than overfitting.

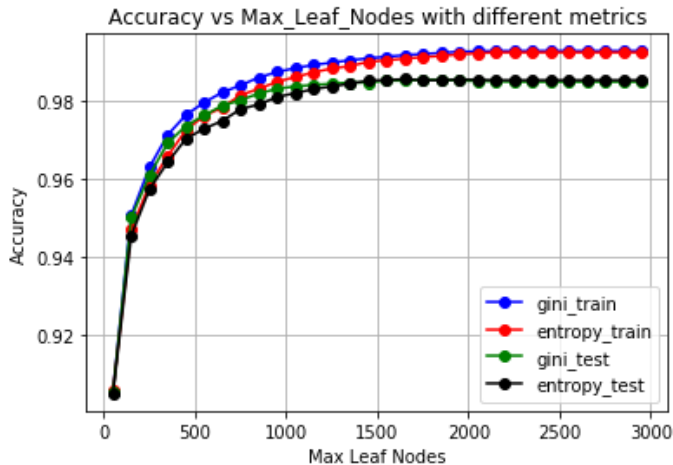
Boosted Tree



- ❖ Comparing the both plots, we can clearly observe that boosted tree has much higher accuracy for low depth trees since they are weak learners.
- ❖ But as the tree depth increases, boosted tree is performing worse on the test data.
- ❖ **Optimal value of depth for boosted is between 12 or 13 which we found by hyper-parameter tuning above**

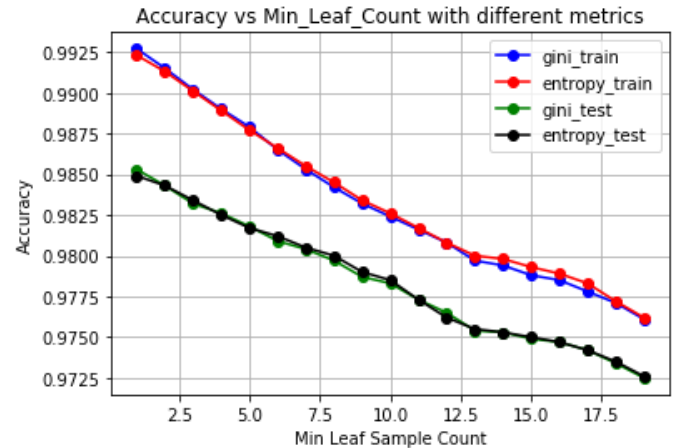
Experiment 3: Decision Tree Pruning by limiting max leaf nodes and sample leaf size

Varying Max Leaf Nodes



- ❖ By increasing the number of leaf nodes, we are allowing decision to grow. Hence increasing the accuracy of train.
- ❖ Optimal value can be observed at 1500

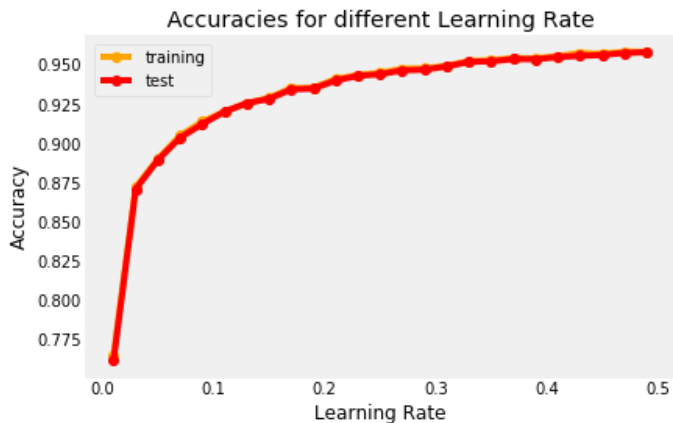
Varying sample leaf size



- ❖ As the number of samples required to split an internal node decreases, the algorithm restricts many splits thereby many misclassifications.
- ❖ Hence, there is decrease in accuracy for train and test data

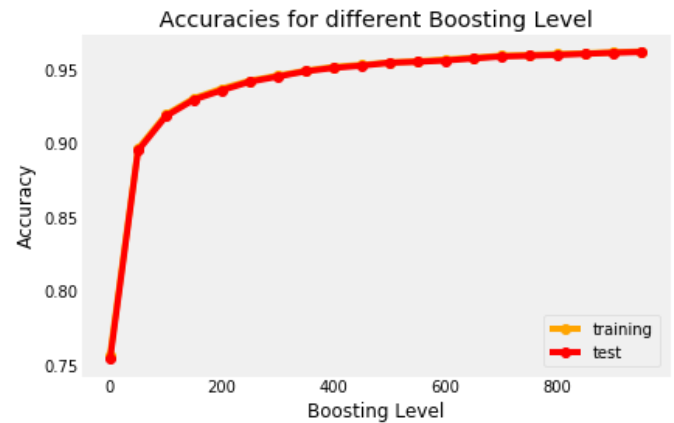
Experiment 4: Boosting – Learning Rate and Boosting Level (No of trees or rounds)

Learning Rate



- ❖ Learning rate is the amount of overfitting in a way you could say with each new tree
- ❖ So if it is very low (slow, gradual learning from one tree to next) obviously you will need a lot of trees to get a final reasonable model.
- ❖ If you want to fit faster, or rather overfit itself, set a higher eta and low number of trees
- ❖ **Optimal learning can be found between 0.2 and 0.3 from the learning curve above**

N_estimator



- ❖ There is no much increase in the accuracy beyond n_estimator beyond 500.
- ❖ The reason is in the way that the boosted tree model is constructed, sequentially where each new tree attempts to model and correct for the errors made by the sequence of previous trees. Quickly, the model reaches a point of diminishing returns.
- ❖ The same can be confirmed from the hyper-parameter tuning where the n_estimator is 525

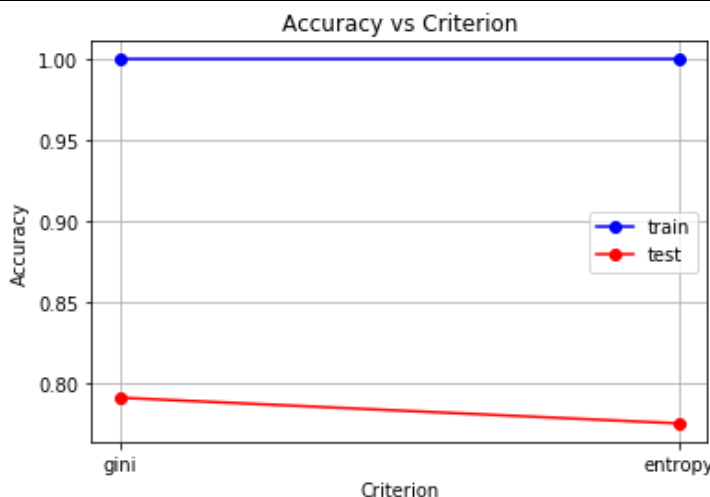
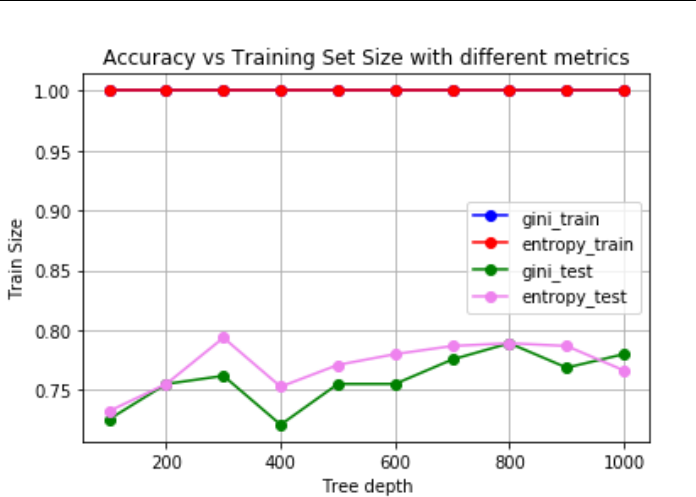
EXPERIMENTATION – Decision Tree and Boosting with IBM Data

For booting, I performed hyper-parameter tuning using XGBoost & LightGBM:

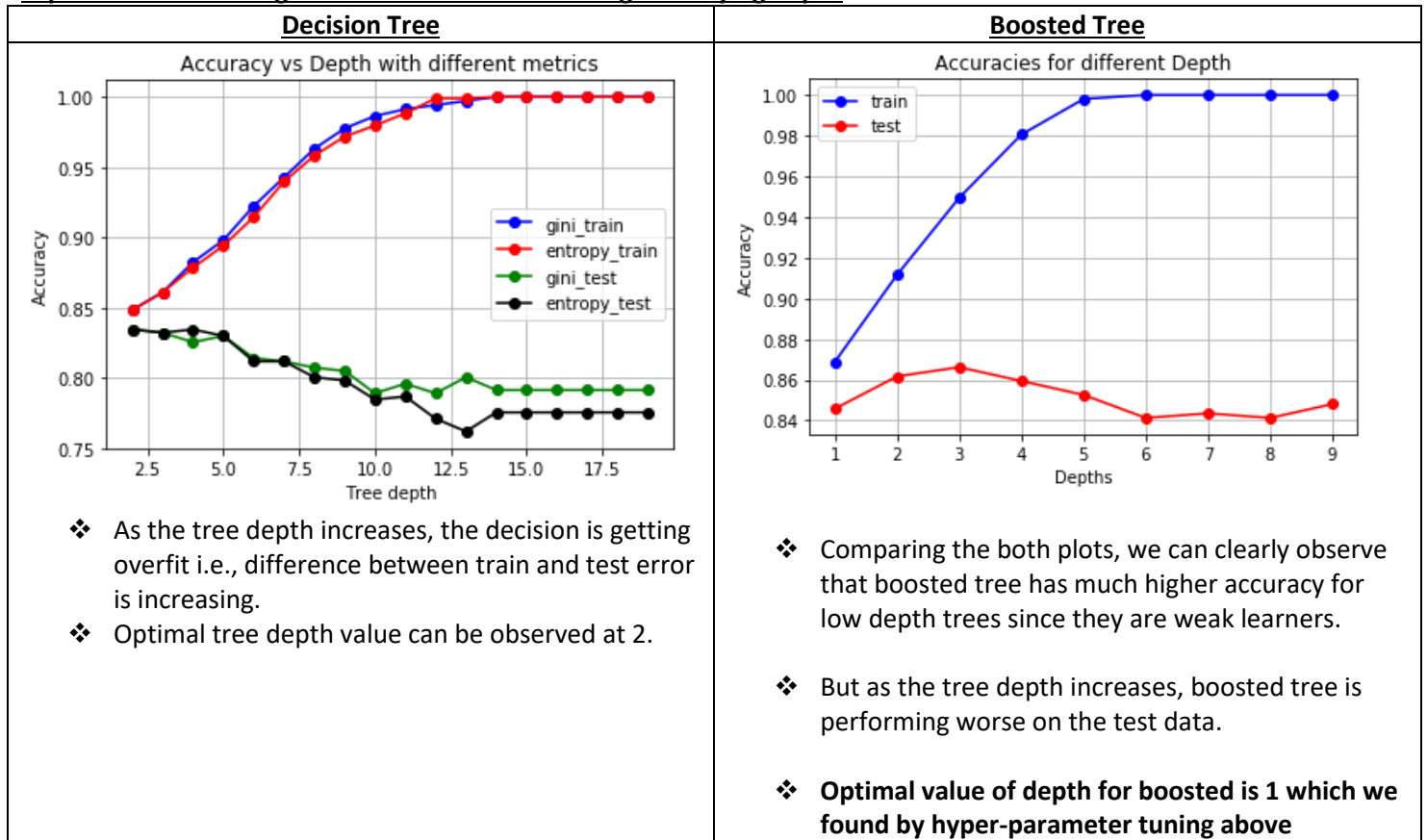
XGBoost	LightGBM
<pre>{'model': 'XGBoost', 'parameter search time': '0:01:14.320000', 'accuracy': 0.896, 'test auc score': 0.868, 'training auc score': 0.906, 'learning_rate': 0.2562928065031515, 'm_child_weight': 5.0, 'max_depth': 1.0, 'n_estimators': 675.0, 'subsample': 0.55</pre>	<pre>{'model': 'LightGBM', 'parameter search time': '0:00:41.729000', 'accuracy': 0.875, 'test auc score': 0.854, 'training auc score': 0.92, 'feature_fraction': 0.9015923386943969, 'learning_rate': 0.22912214347416732, 'max_depth': 1.0, 'min_split_gain': 0.07873357374370113, 'n_estimators': 525.0</pre>

Let's confirm these using different experiments and learning curves below:

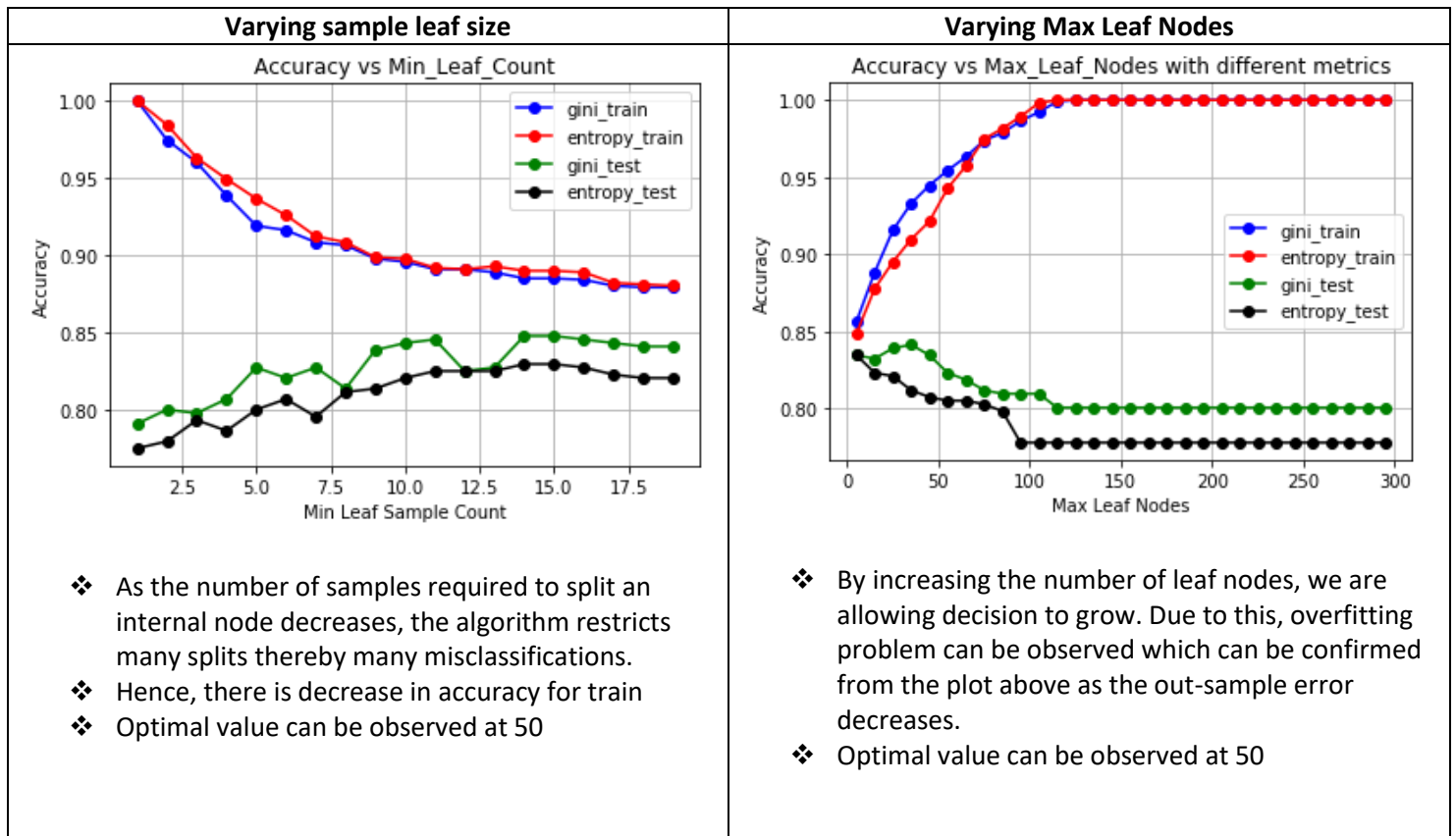
Experiment 1: Decision Tree : Varying Criterion and Varying Train Size

<u>Different Criterion</u>	<u>Varying Train Size</u>																																							
<p>Accuracy vs Criterion</p>  <table><caption>Accuracy vs Criterion Data</caption><tr><th>Criterion</th><th>train</th><th>test</th></tr><tr><td>gini</td><td>1.00</td><td>0.79</td></tr><tr><td>entropy</td><td>1.00</td><td>0.77</td></tr></table> <ul style="list-style-type: none">❖ There is no difference in accuracy for any of the metrics, but the test error is low for both metrics.❖ It might be due to not restricting any parameters for the tree thereby overfitting	Criterion	train	test	gini	1.00	0.79	entropy	1.00	0.77	<p>Accuracy vs Training Set Size with different metrics</p>  <table><caption>Accuracy vs Training Set Size Data</caption><tr><th>Tree depth</th><th>gini_train</th><th>entropy_train</th><th>gini_test</th><th>entropy_test</th></tr><tr><td>200</td><td>1.00</td><td>1.00</td><td>0.73</td><td>0.74</td></tr><tr><td>400</td><td>1.00</td><td>1.00</td><td>0.72</td><td>0.75</td></tr><tr><td>600</td><td>1.00</td><td>1.00</td><td>0.75</td><td>0.78</td></tr><tr><td>800</td><td>1.00</td><td>1.00</td><td>0.78</td><td>0.79</td></tr><tr><td>1000</td><td>1.00</td><td>1.00</td><td>0.78</td><td>0.77</td></tr></table> <ul style="list-style-type: none">❖ It can be inferred from the above plot that decision tree is able to perfectly categorize the train data with smaller data❖ Test accuracy is increasing with adding more data to train i.e., better fit	Tree depth	gini_train	entropy_train	gini_test	entropy_test	200	1.00	1.00	0.73	0.74	400	1.00	1.00	0.72	0.75	600	1.00	1.00	0.75	0.78	800	1.00	1.00	0.78	0.79	1000	1.00	1.00	0.78	0.77
Criterion	train	test																																						
gini	1.00	0.79																																						
entropy	1.00	0.77																																						
Tree depth	gini_train	entropy_train	gini_test	entropy_test																																				
200	1.00	1.00	0.73	0.74																																				
400	1.00	1.00	0.72	0.75																																				
600	1.00	1.00	0.75	0.78																																				
800	1.00	1.00	0.78	0.79																																				
1000	1.00	1.00	0.78	0.77																																				

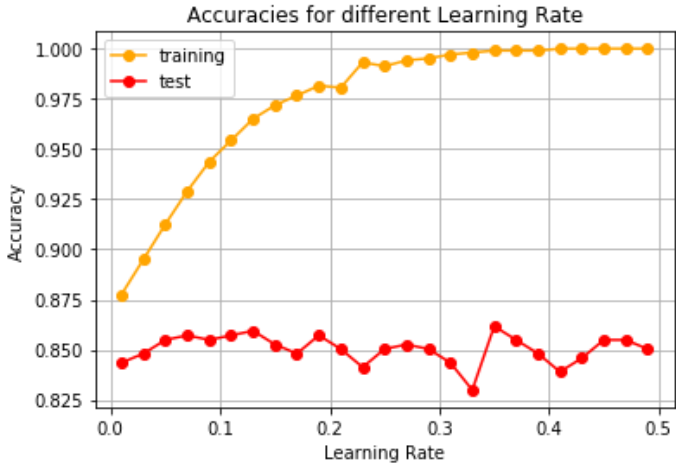
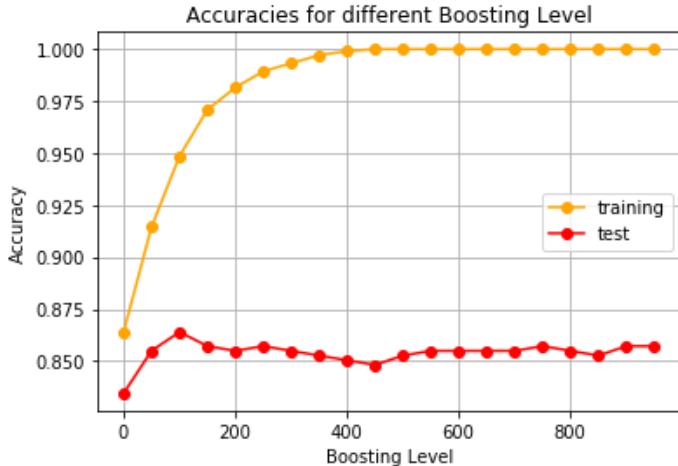
Experiment 2 : Pruning for Decision Tree vs Boosting for varying depth



Experiment 3: Decision Tree Pruning by limiting max leaf nodes and sample leaf size



Experiment 4: Boosting – Learning Rate and Boosting Level (No of trees or rounds)

Learning Rate	N_estimator
<p>Accuracies for different Learning Rate</p>  <ul style="list-style-type: none"> ❖ Learning rate is the amount of overfitting in a way you could say with each new tree ❖ So if it is very low (slow, gradual learning from one tree to next) obviously you will need a lot of trees to get a final reasonable model. ❖ If you want to fit faster, or rather overfit itself, set a higher eta and low number of trees ❖ Optimal learning can be found between 0.2 and 0.3 from the learning curve above 	<p>Accuracies for different Boosting Level</p>  <ul style="list-style-type: none"> ❖ There is no much increase in the accuracy beyond n_estimator beyond 500 ❖ The same can be confirmed from the hyper-parameter tuning where the n_estimator is 675

Learning Outcomes:

GPU run-time data:

- ❖ SVM and Decision Tree performed equally better for large data dataset but the only drawback is that run-time for SVM is much higher compared to Decision Tree
- ❖ Since both the data sets have clear boundary or separation, boosting didn't improve much on basic decision tree

IBM HR Attrition Data:

- ❖ For smaller dataset, SVM performed worse than decision tree
- ❖ Decision tree performed better on IBM dataset since a tree cuts the space based on the information content. It divides recursively so even with a small data set you get an informative separation.
- ❖ SVM divides the space once, based on parameters for all points, learned via a cost function. That line is as good as the data it can fit