APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

Project Report:

SVM, Decision Tree, Boosting & Experimentations

Under the guidance of:

Professor Muhammad Sabir

Submitted by:
Pranav Shil
M.Sc. Business Analytics
Track: Data Science
Pranav.shil@utdallas.edu

Date: 10/15/2018

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

I have used Python a scripting language for this assignment and all of my work has been conducted and performed using Jupyter notebook.

Most of the packages are inbuilt and available in jupyter notebook but I had to download and install packages like graphviz and sklearn**.**

The data sets used here are publicly available and you can download those datasets through the links attached below:

Student Math grade dataset -

https://archive.ics.uci.edu/ml/datasets/Student+Performance

Attribute Information:

Attributes for both student-mat.csv (Math course) and student-por.csv (Portuguese language course) datasets:
1 school - student's school (binary: 'GP' - Gabriel Pereira or 'MS' - Mousinho da Silveira)
2 sex - student's sex (binary: 'F' - female or 'M' - male)
3 age - student's age (numeric: from 15 to 22)
4 address - student's home address type (binary: 'U' - urban or 'R' - rural)
5 famsize - family size (binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3)
6 Pstatus - parent's cohabitation status (binary: 'T' - living together or 'A' - apart)
7 Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
8 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 â€“ 5th to 9th grade, 3 â€“ secondary education or 4 â€“ higher education)
9 Mjob - mother's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
10 Fjob - father's job (nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other')
11 reason - reason to choose this school (nominal: close to 'home', school 'reputation', 'course' preference or 'other')
12 guardian - student's guardian (nominal: 'mother', 'father' or 'other')
13 traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15 failures - number of past class failures (numeric: n if 1<=n<3, else 4)
16 schoolsup - extra educational support (binary: yes or no)
17 famsup - family educational support (binary: yes or no)
18 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
19 activities - extra-curricular activities (binary: yes or no)
20 nursery - attended nursery school (binary: yes or no)
21 higher - wants to take higher education (binary: yes or no)

22 internet - Internet access at home (binary: yes or no)
23 romantic - with a romantic relationship (binary: yes or no)
24 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
25 freetime - free time after school (numeric: from 1 - very low to 5 - very high)
26 goout - going out with friends (numeric: from 1 - very low to 5 - very high)
27 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
28 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
29 health - current health status (numeric: from 1 - very bad to 5 - very good)
30 absences - number of school absences (numeric: from 0 to 93)

these grades are related with the course subject, Math or Portuguese:
31 G1 - first period grade (numeric: from 0 to 20)
31 G2 - second period grade (numeric: from 0 to 20)
32 G3 - final grade (numeric: from 0 to 20, output target)

Wisconsin Breast Cancer Diagnosis dataset –

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

In Both dataset – Main objective is to successfully understand and apply various machine learning algorithms and classify the dataset accurately.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

Let's start with student Dataset and see how different algorithms work for this data set and how accurate results we can produce using them.

Student Math Grade Prediction:

Preamble: I am dropping the variables G1 and G2 before any scaling or transformation as asked by our professor

Data Preprocessing: In this dataset there are 13 ordinal categorical variables, 16 nominal categorical variables and 2 numerical variables.

Steps :

- All the nominal categorical variables have been converted to dummy variables using a pandas get_dummies package
- Target or response variable 'G3' has been prepared for classification case by dissecting it into two types based on value greater or lower than its median value and labeled as Good Grade and Bad Grade respectively which was then again changed to 1 and 0 respectively for gradient boosting classifier.
- Normalize the two numerical variables 'age' and 'absences'
- Divide the data set into train and test set using train_test_split package in sklearn.model_selection

Congratulations Now data is ready for further crunching and prediction

Case 1 :

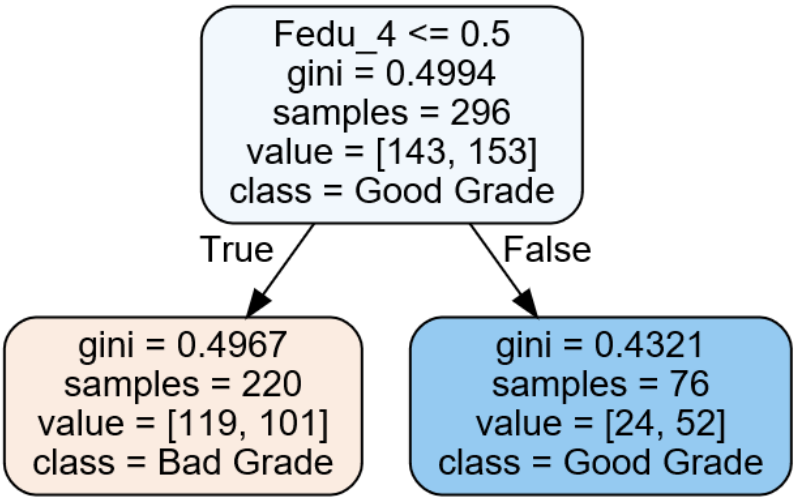
Decision Tree Classifier

I have used ipywidgets package for interactive output where we can tune the decision tree and see the decision tree with all those tuned parameters. The confusion matrix, test set accuracy as well as train set accuracy will be automatically printed in the output which had made Pruning and tuning very simple. You can see and see in the code. I have defined a plot_tree function and called it in the interactive function with arguments as option in the widget.

- criteria can be changed from gini to entropy
- splitter can be chosen between best and random
- depth of the tree can be selected from 1 to 12(as per code) but can be changed if required which is very simple
- The minimum no. of samples required to split an internal node can be tuned too
- Minimum no. of samples required to be at the leaf node can be also adjusted.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT


```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Acuracy on training set: 57.770
Acuracy on test set: 49.495

 Confusion Matrix:

[[36  7]
 [43 13]]

 Classification Report:

             precision    recall  f1-score   support

  Bad Grade       0.46      0.84      0.59        43
 Good Grade       0.65      0.23      0.34        56

avg / total       0.57      0.49      0.45        99
```



This is the overview of that interactive model mentioned above.

I will be mentioning all the best model as per all the various criteria and parameters whenever I will compare**.**

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

crit gini
split best
depth 9
min_split 3
min_leaf 7

```
Controlled Decision Tree : Change the above parameters and observe the
 To see the effect of pruning, change the tree depth and notice the ch


Acuracy on training set: 78.378
Acuracy on test set: 62.626

 Confusion Matrix:

[[28 15]
 [22 34]]
```

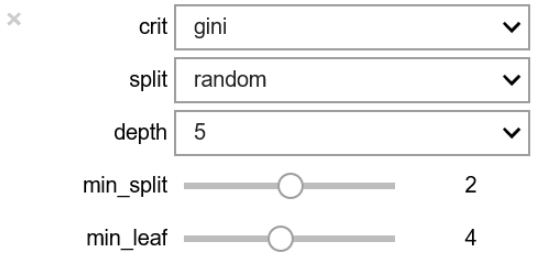
```
 Classification Report:

             precision    recall  f1-score   support

  Bad Grade       0.56      0.65      0.60        43
 Good Grade       0.69      0.61      0.65        56

avg / total       0.64      0.63      0.63        99
```

This is the optimum accuracy after tuning the parameters. As you know pruning is the cutting of tree or assigning a certain depth to the tree, I have adjusted it to 9 which I decided by having a look at the learning curve

crit gini
split best
min_split 2
min_leaf 5

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

When the splitter was changed to random instead of best and min_sample_leaf was set to 4, the test set accuracy increased to 70.707% and optimum tree depth turn out to be 5




```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Acuracy on training set: 68.581
Acuracy on test set: 70.707

 Confusion Matrix:

[[33 10]
 [19 37]]
```

Now let's change the criteria to entropy and find the optimum accuracy by tuning.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

crit: entropy
split: random
depth: 7
min_split: 2
min_leaf: 4

```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Acuracy on training set: 75.000
Acuracy on test set: 69.697

 Confusion Matrix:

[[34  9]
 [21 35]]
```


```
-     --

Classification Report:

             precision    recall  f1-score   support

  Bad Grade       0.62      0.79      0.69        43
 Good Grade       0.80      0.62      0.70        56

avg / total       0.72      0.70      0.70        99
```

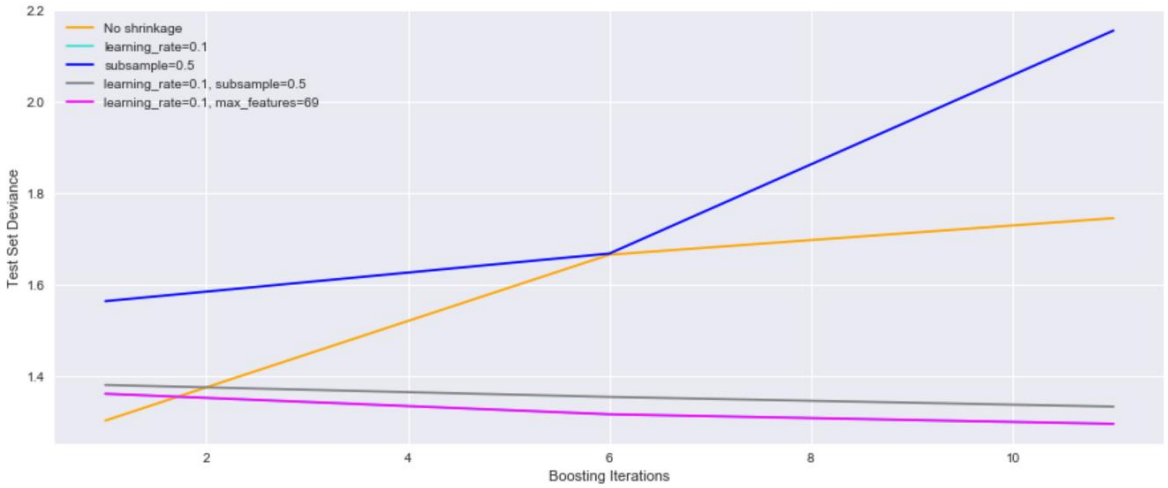
With proper tuning we got the best test set accuracy of 69.697% for entropy criteria at tree depth = 7 and other parameters shown above in the snip attached.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

Case 2:

Gradient Boosting Classifier:

I have used Gradient boosting classifier for further improving the accuracy of the model and enhance the performance.



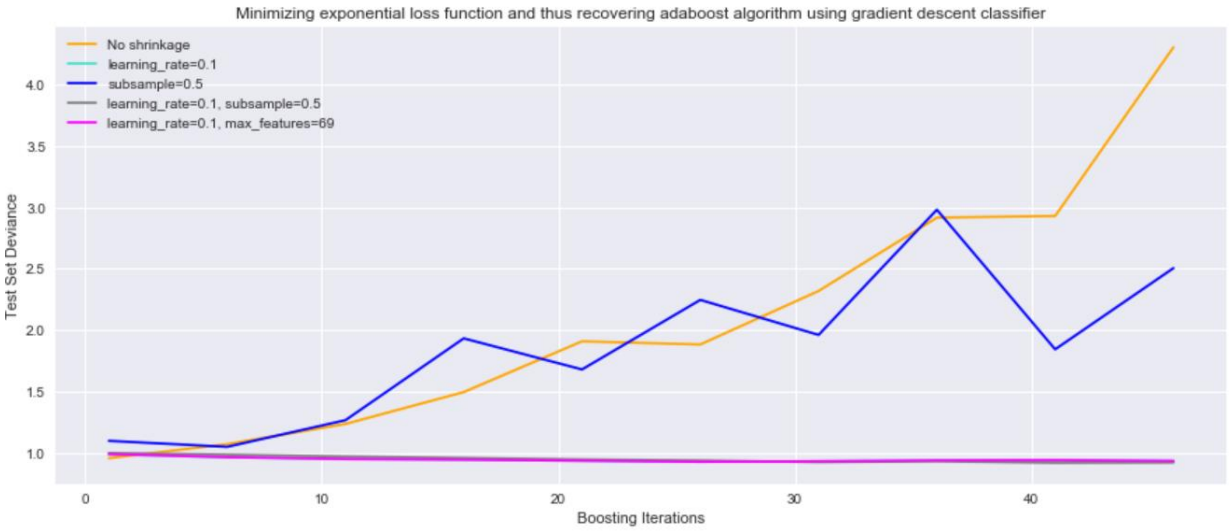
By applying gradient boosting classifier, I have plotted a graph to understand how the test se deviance varies with the no. of boosting iterations. We can see the trends in the graph above: As the boosting iterations increase the test set deviance decrease except for no shrinkage cure and subsample = 0.5 which tends to increase. We will discuss about the variation and tuning further below. One thing to notice here is that the loss function which is minimized is the 'deviance'.

Note:
No shrinkage = sub_sample = 1, learning_rate = 1
Wherever there is no mention on of subsequent parameter(sub_sample or learning_rate) it is equal to 1.

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=None,
              max_features=69, max_leaf_nodes=4, min_impurity_split=1e-07,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=15,
              presort='auto', random_state=100, subsample=1.0, verbose=0,
              warm_start=False)

Acuracy on training set: 71.284
Acuracy on test set: 68.687

 Confusion Matrix:

[[28 15]
 [16 40]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.64      0.65      0.64        43
          1       0.73      0.71      0.72        56

avg / total       0.69      0.69      0.69        99
```

Accuracy obtained is 71% for train set and 69% for test set. Here the criteria used is friedman mean squared error. I have changed it to mean absolute error and mean squared error but friedman mse turned out to be the best one comparatively.

I have changed the loss function to exponential in gradient boosting classifier which in turn recovered Adaboost algorithm where we minimize the exponential loss function. So, let's see how the trend varies in loss function = 'exponential'.

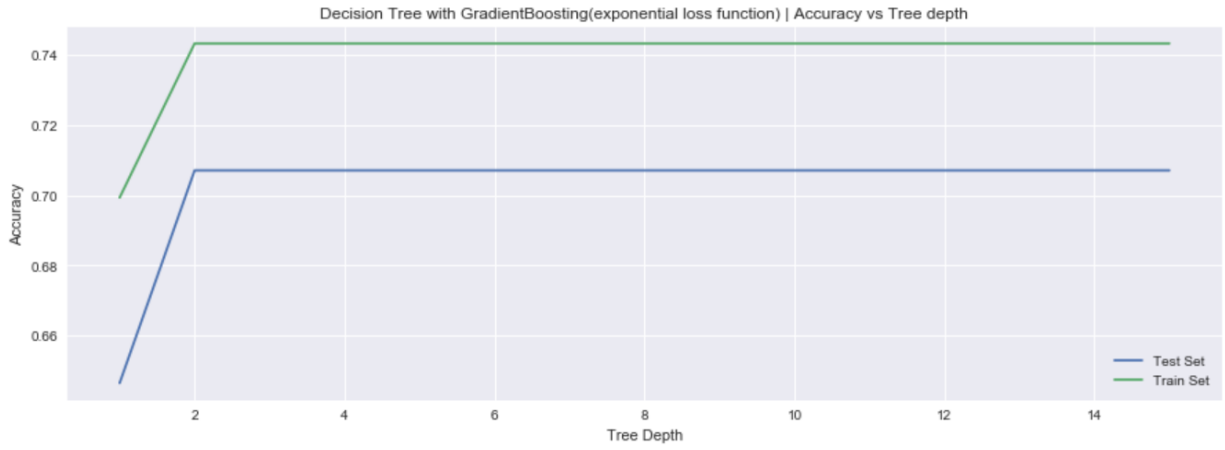


The trends are quite clear that with subsample = 1 and learning rate = 1, the loss functions is not able to converge and jumps from one crest to another but cannot reach the minima. Whereas with learning rate of o.1 and subsample =0.5 or learning rate =0.1 and max features = 69 seems to have a negative trend and loss function decreases or it converges to the minima with the increase of no. of iterations which makes sense.

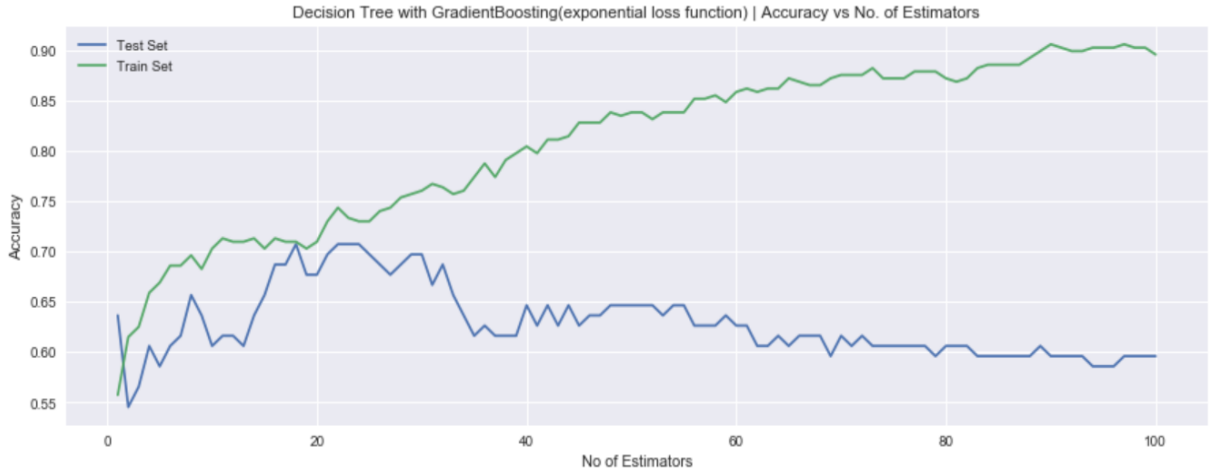
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='exponential', max_depth=15,
              max_features=69, max_leaf_nodes=4, min_impurity_split=1e-07,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=50,
              presort='auto', random_state=100, subsample=1.0, verbose=0,
              warm_start=False)

Acuracy on training set: 83.784
Acuracy on test set: 64.646

 Confusion Matrix:

[[28 15]
 [20 36]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.58      0.65      0.62        43
          1       0.71      0.64      0.67        56

avg / total       0.65      0.65      0.65        99
```

We should observe one thing here that maximum tree depth is set to 15, learning rate is set to 0.1 and no. of estimators is set to 50. But we must validate all these first, right?

Hence our next step would be to optimize all those parameters.



This graph shows that maximum tree depth to be chosen as 2 would be a right choice. You would argue that how did you come to this conclusion without checking the no. of estimators. Let me mention it without further delay that I have checked it for few different no. of estimators and each of them resulted to the same conclusion that optimum tree depth should be 2 in case of gradient boosting classifier.



If you observe the graph above carefully you can see that at no. of estimators = 22, the accuracy is the highest for the test and train set together. Hence optimum no. of estimators in this case is equal to 22

From the graph above we can see the variation of accuracy with change in alpha or learning rate. And the optimum value of alpha would be equal to 0.1 in this case. Let's see the classification report after using the optimized parameters in the gradient boosting classifier.

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='exponential', max_depth=2,
              max_features=None, max_leaf_nodes=4,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=10, min_weight_fraction_leaf=0.0,
              n_estimators=22, presort='auto', random_state=100,
              subsample=1.0, verbose=0, warm_start=False)

Acuracy on training set: 74.324
Acuracy on test set: 70.707

 Confusion Matrix:

[[29 14]
 [15 41]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.66      0.67      0.67        43
          1       0.75      0.73      0.74        56

avg / total       0.71      0.71      0.71        99
```

Compare and Contrast:

- If you scroll above and look for the best accuracy prediction, you will find decision tree classifier with gini criterion, max_tree_depth = 5, min_split = 2, min_leaf = 4 and splitter = random having a training set accuracy of 68.581% and test set accuracy of 70.707%. we have already resolved the issue of overfitting by pruning the decision tree. Hence this is a very good model.
- Now when we see the gradient boosting classified model with the optimized parameters, it appears to be the best model so far, since it being an ensemble of regression trees, it is devoid of overfitting and also the accuracy obtained through the model makes it pretty awesome. Although the test set accuracy is same as the decision tree classifier best model mentioned above, it has greater training set accuracy of 74.324%

Case 3:

SVM Classifier

Steps to preprocess the Data is similar to what we did in the case of decision tree classification.

Scroll above to see

After the data preprocessing we directly dive into selecting kernel and producing the confusion matrix with support vector classifier. I have set the class_weights = ‘balanced’ to balance the weights of the respective classes.Kernel is selected as ‘linear’, ‘rbf’ and ‘poly’. Argument degree is only mentioned in case of kernel poly.

Let’s have a look at the classification report in each case.

Kernel = ‘linear’

```
Train accuracy = 79.71%
Test accuracy = 59.66%

 Confusion Matrix:

[[25 22]
 [26 46]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.49      0.53      0.51        47
          1       0.68      0.64      0.66        72

avg / total       0.60      0.60      0.60       119
```

Kernel = ‘rbf’

```
Train accuracy = 94.2%
Test accuracy = 58.82%

 Confusion Matrix:

[[25 22]
 [27 45]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.48      0.53      0.51        47
          1       0.67      0.62      0.65        72

avg / total       0.60      0.59      0.59       119
```

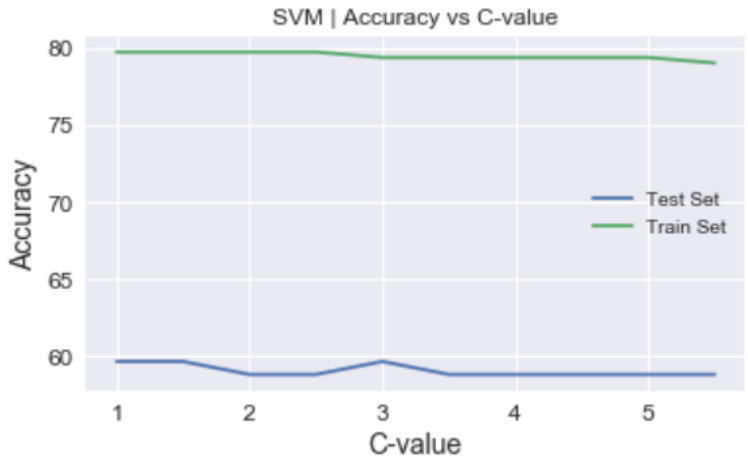
APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

Kernel = 'poly'

```
Train accuracy = 94.2%
Test accuracy = 62.18%

 Confusion Matrix:

[[18 29]
 [16 56]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.53      0.38      0.44        47
          1       0.66      0.78      0.71        72

avg / total       0.61      0.62      0.61       119
```

Next step would be tuning the penalty parameter c and optimizing the model for each kernel. Let's start with linear kernel.

Kernel = 'linear'



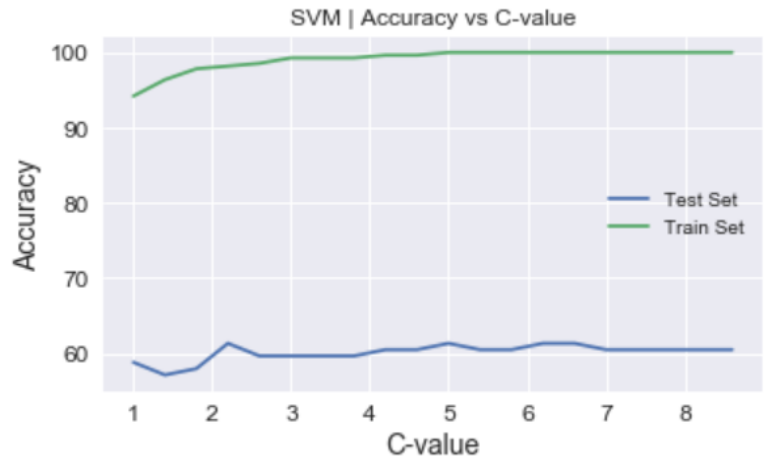
This graph shows us that at c = 1.5 the test set accuracy starts to decrease hence we can choose c = 1.4 as the penalty parameter and improve the accuracy of the linear kernel svm.

The classification report after tuning C is given below:

```
SVC(C=1.4, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=111, shrinking=True,
  tol=0.001, verbose=False)

Train accuracy = 80.07%
Test accuracy = 59.66%

 Confusion Matrix:

[[25 22]
 [26 46]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.49      0.53      0.51        47
          1       0.68      0.64      0.66        72

avg / total       0.60      0.60      0.60       119
```

If we compare the basic linear svm with c =1 with tuned linear svm (c=1.4) , we can say that even though the test set accuracy hasn't increased but the train set accuracy has increased with the increment of penalty parameter .

Kernel = 'rbf'



This graph shows us that at c = 6.2 the test set accuracy and the train set accuracy are at their peaks at the same time. Hence by selecting the penalty parameter as 6.2 we can improve the accuracy of the rbf kernel svm.

The classification report after tuning C is given below:

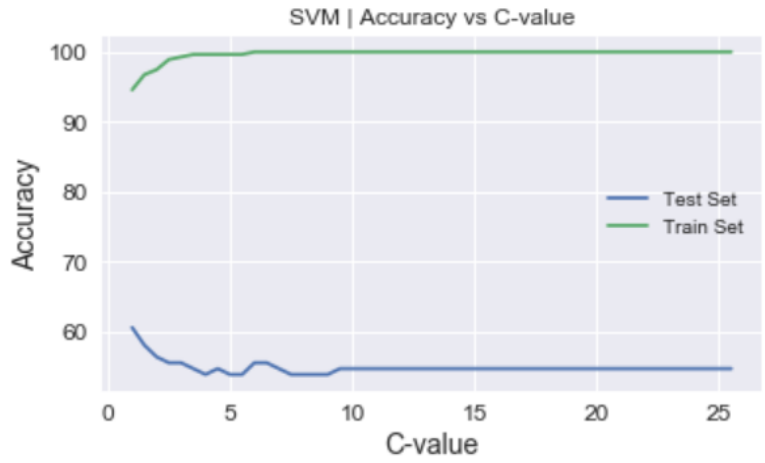
```
SVC(C=6.2, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=111, shrinking=True,
  tol=0.001, verbose=False)

Train accuracy = 100.0%
Test accuracy = 61.34%

 Confusion Matrix:

[[25 22]
 [24 48]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.51      0.53      0.52        47
          1       0.69      0.67      0.68        72

avg / total       0.62      0.61      0.61       119
```

As compared to the untuned svm with kernel function rbf, this tuned model has increased test accuracy from 58.82% to 61.34% and train set accuracy from 94.2% to 100%. It seems there is overfitting in this case.

Maximizing the margin is not the only trick (although it is very important). If a non-linear kernel function is used, then the smoothness of the kernel function also has an effect on the complexity of the classifier and hence on the risk of over-fitting. If you use a Radial Basis Function (RBF) kernel, for example, and set the scale factor (kernel parameter) to a very small value, the SVM will tend towards a linear classifier. If you use a high value, the output of the classifier will be very sensitive to small changes in the input, which means that even with margin maximization, you are likely to get over-fitting.

Unfortunately, the performance of the SVM can be quite sensitive to the selection of the regularization and kernel parameters, and it is possible to get over-fitting in tuning these hyper-parameters via e.g. cross-validation. The theory underpinning SVMs does nothing to prevent this form of over-fitting in model selection.

Kernel = 'poly'



In this case the degree of poly kernel is set to 2 since it is the best amongst other degree values. As we can see, with the increase of C value, the test set accuracy decreases drastically, and it is evident that with poly kernel here C = 1 (default) is the best penalty parameter value to optimize the model. Hence, we have a test set accuracy of 62.18% and train set accuracy of 94.2%

Note: Other factors like gamma should be tuned too for better optimization of svm models.

Conclusion:

If I have to compare between the SVM classifier , Decision Tree classifier and Boosted Classifier (Gradient Boosting classifier in our case).

I would always go with ensemble methods like gradient boosting classifier where it takes multiple additive model of regression trees and result into a model which is devoid of overfitting problem and bias problem.

The results obtained in all those cases verify that the model obtained by optimizing the gradient boosting classifier on the data set was the best amongst all with test set accuracy of 70.71% and train set accuracy of 74.32%.

Wisconsin Breast Cancer Diagnosis Data Set:

Data preprocessing:

- Check if there are null values in the data set. I have checked and found no null values in it except in the variable 'unnamed' which was dropped.
- Since all the variables here are numerical there is no requirement to make dummy
- Scaling the variable values is very important. Normalize it.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

- Split the data set into train and test set after dropping the id variable which cannot be a predictor variable.

Case 1 :

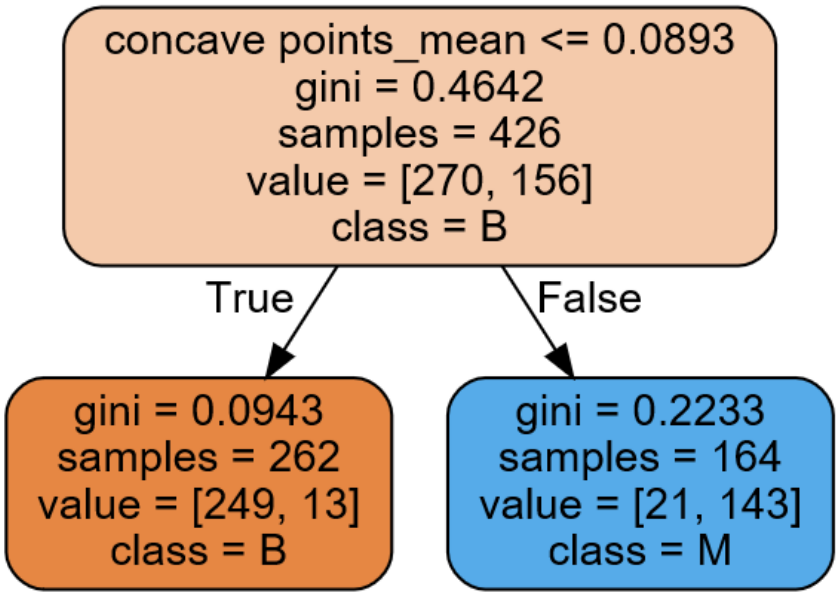
Decision Tree Classifier

I have used ipywidgets package for interactive output where we can tune the decision tree and see the decision tree with all those tuned parameters. The confusion matrix, test set accuracy as well as train set accuracy will be automatically printed in the output which had made Pruning and tuning very simple. You can see and see in the code. I have defined a plot_tree function and called it in the interactive function with arguments as option in the widget.

- criteria can be changed from gini to entropy
- splitter can be chosen between best and random
- depth of the tree can be selected from 1 to 25(as per code) but can be changed if required which is very simple
- The minimum no. of samples required to split an internal node can be tuned too
- Minimum no. of samples required to be at the leaf node can be also adjusted.


```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Acuracy on training set: 92.019
Acuracy on test set: 90.210

 Confusion Matrix:

[[80  7]
 [ 7 49]]
```

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT



Main objective is to classify the cases of patient's cancer accurately as Malignant or Benign.
Now let's see how accurate we can predict the diagnosis of the cancer.

This graph shows us that with the increase of the depth of tree, the accuracy increases until depth = 4 and then saturates. I have tried to tune the min_split as well as min_leaf but the optimum depth is 4 in this case where criteria is Gini criteria and splitter is best.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

crit gini
split best
depth 4
min_split 2
min_leaf 5

```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Acuracy on training set: 97.887
Acuracy on test set: 93.007

 Confusion Matrix:

[[84  3]
 [ 7 49]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.97	0.94	87
1	0.94	0.88	0.91	56
avg / total	0.93	0.93	0.93	143

Test set accuracy is 93.007% and train set accuracy is 97.887%

Here we can see at tree depth = 6 the accuracy on train as well as test set seems to saturate. The parameter tuned are as follows:

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

crit: gini
split: random
depth: 6
min_split: 3
min_leaf: 7

```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Accuracy on training set: 95.775
Accuracy on test set: 94.406

 Confusion Matrix:

[[85  2]
 [ 6 50]]


  Classification Report:

             precision    recall  f1-score   support

          0       0.93      0.98      0.96        87
          1       0.96      0.89      0.93        56

avg / total       0.94      0.94      0.94       143
```

Accuracy on test set and train set are 94.406% and 95.775% respectively.

Both of the models above were based on gini criterion with best split and random split and tuned min_leaf and min_split.

Let's check how accurate our results get if we change the criterion to entropy.

We can clearly see above in the graph that at tree depth = 4 the accuracy of both train and test set is the highest. The parameters tuned for this model is shown below:

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

crit: entropy
split: best
depth: 4
min_split: 2
min_leaf: 7

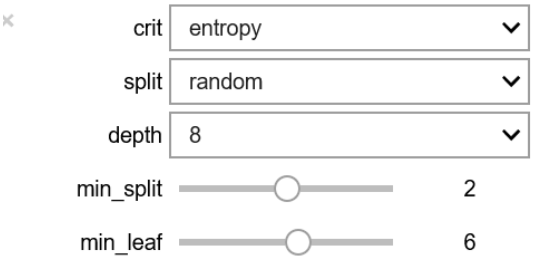
```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Accuracy on training set: 98.122
Accuracy on test set: 93.706

 Confusion Matrix:

[[85  2]
 [ 7 49]]

Classification Report:

             precision    recall  f1-score   support

          0       0.92      0.98      0.95        87
          1       0.96      0.88      0.92        56

avg / total       0.94      0.94      0.94       143
```

Criteria is selected as entropy and min_split = 2, min_leaf= 7,tree depth at 4 and splitter is selected as 'best' and we got test set accuracy of 93.706% and train set accuracy of 98.122%.

Let's check the same with splitter as random.

With splitter as random we find the max tree depth should be taken as 8 with min_split and min_leaf as 2 and 6 respectively.

The tuned parameters are show below as follows:




```
Controlled Decision Tree : Change the above parameters and observe the difference.
 To see the effect of pruning, change the tree depth and notice the change in various metrics


Accuracy on training set: 95.775
Accuracy on test set: 94.406

 Confusion Matrix:

[[84  3]
 [ 5 51]]


 Classification Report:

             precision    recall  f1-score   support

          0       0.94      0.97      0.95        87
          1       0.94      0.91      0.93        56

avg / total       0.94      0.94      0.94       143
```

The accuracy is found to be 94.406 % for the test set and 95.775% for the train set. I am not mentioning everything again and again since we have gone through the parameters tuning and resultant accuracy.

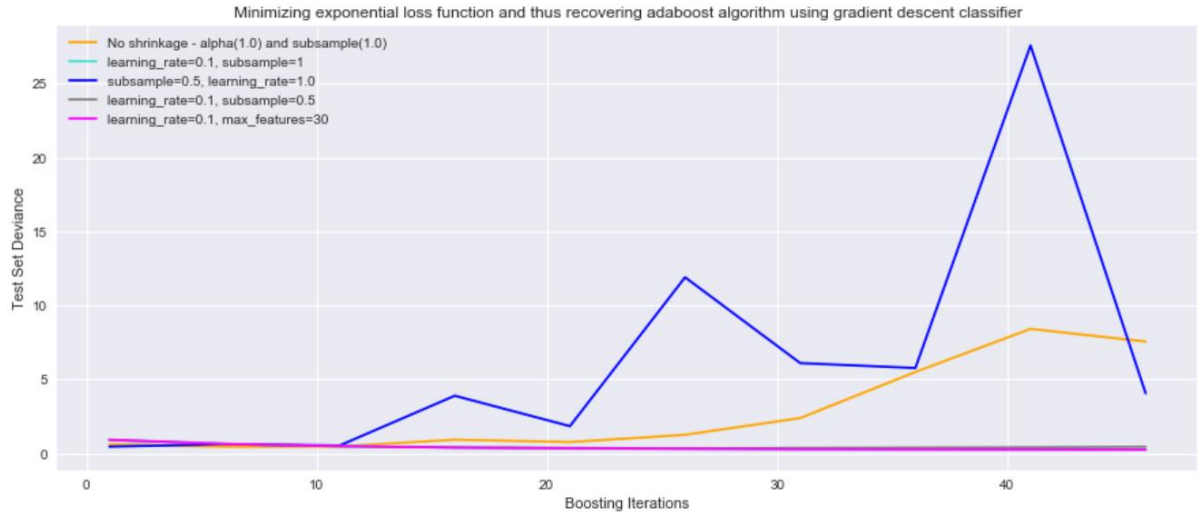
One thing that we see is that model test set accuracy with some tuning of max_leaf and max_split is same for gini criteria and entropy criteria , both with random split, which is interesting. So if we have to choose one among them we should select the one with gini criterion because the depth of the tree is way less than the one with entropy criterion and more depth in tree increases the computation power and make it more expensive.

Case 2:

Gradient Boosting Classifier:

Gradient boosting classifier has proved to be promising in student math grade prediction data set. So let’s try this out here in this data set as well.

APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT



Above graph shows 5 cases with each one having different combination of learning rate and subsample and their variation with respect to the boosting iterations. Blue curve seems to suffer from large learning rate and cannot converge to minima. Violet, grey and light blue seems to converge and result in low test deviation. Here we are reducing the exponential loss function, hence it is recovering the adaboost algorithm which does the same.

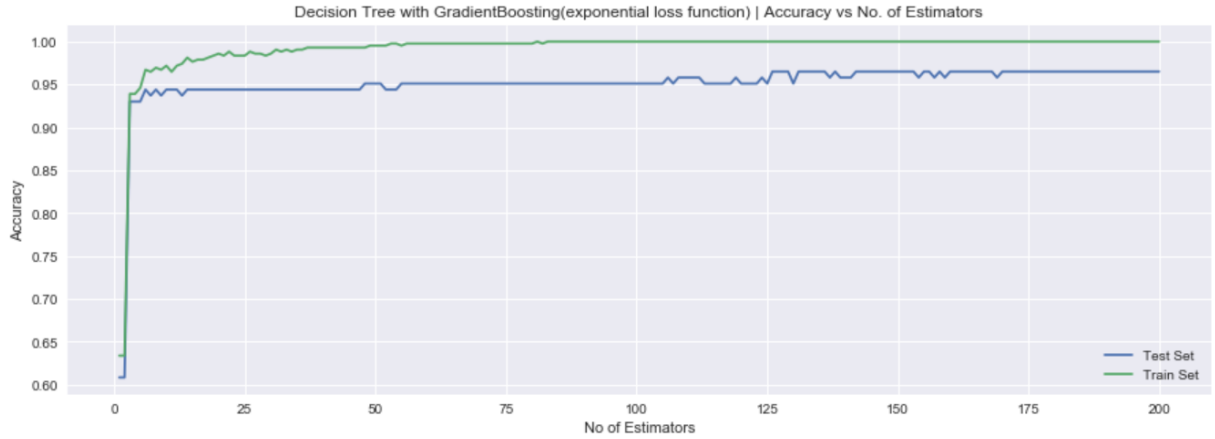
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='exponential', max_depth=15,
              max_features=30, max_leaf_nodes=4, min_impurity_split=1e-07,
              min_samples_leaf=1, min_samples_split=10,
              min_weight_fraction_leaf=0.0, n_estimators=50,
              presort='auto', random_state=100, subsample=1.0, verbose=0,
              warm_start=False)

Acuracy on training set: 99.531
Acuracy on test set: 95.105

 Confusion Matrix:

[[85  2]
 [ 5 51]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.94      0.98      0.96        87
          1       0.96      0.91      0.94        56

avg / total       0.95      0.95      0.95       143
```

With friedman_mse criterion, loss = ‘exponential’, max_depth = 15 , learning rate = 0.1, no. of estimators = 50, min_samples_split = 10, min_sample_leaf = 1 and other default parameters I managed to get an accuracy of 95.105% on test set and train set accuracy of 99.531% through gradient boosting classifier.

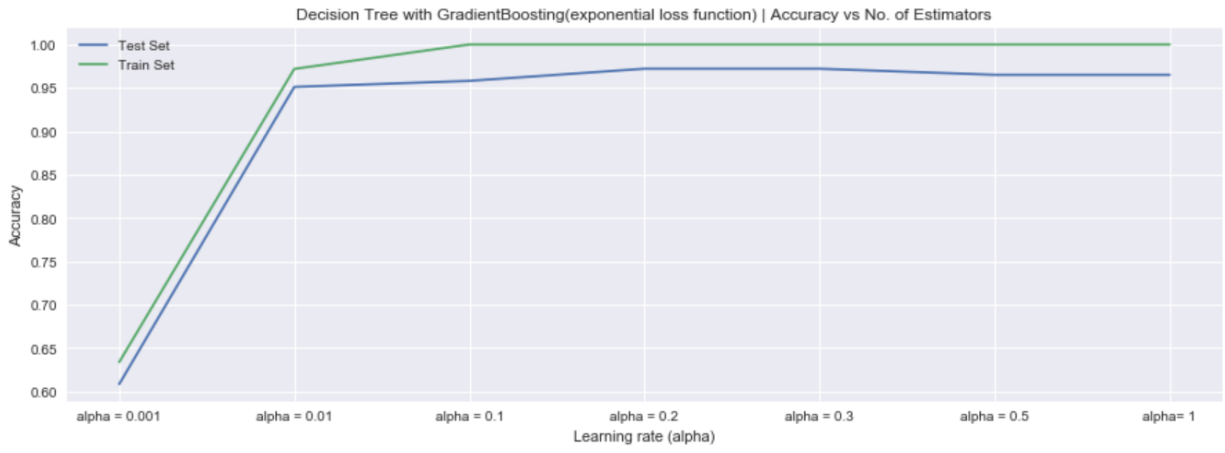
APPLIED MACHINE LEARNING : CLASSIFICATION PROJECT

Optimizing model by tuning the parameters:

This graph shows that the accuracy decreases with the increase of depth of tree. Hence optimum no. of depth in the trees would be 1 only.



This graph shows the optimum no. of estimators would be 128.



This graph shows that optimum alpha or learning rate is 0.2 .

Let's apply the gradient boosting classifier with all these discovered optimized parameters and see how much we can improve the accuracy.

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.2, loss='exponential', max_depth=1,
              max_features=None, max_leaf_nodes=4,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=10, min_weight_fraction_leaf=0.0,
              n_estimators=128, presort='auto', random_state=100,
              subsample=1.0, verbose=0, warm_start=False)

Acuracy on training set: 100.000
Acuracy on test set: 97.203

 Confusion Matrix:

[[87  0]
 [ 4 52]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.96      1.00      0.98        87
          1       1.00      0.93      0.96        56

avg / total       0.97      0.97      0.97       143
```

It seems like there is over fitting issue, but ensemble methods are not susceptible to overfitting issues. Gradient boosting with optimized parameters has delivered a better result over all if we compare it to decision tree with or without pruning.

Accuracy on test and train set are 97.203% and 100% respectively.

Case 3:

SVM classifier:

Steps to preprocess the Data is similar to what we did in the case of decision tree classification.

Scroll above to see

After the data preprocessing we directly dive into selecting kernel and producing the confusion matrix with support vector classifier. I have set the class_weights = 'balanced' to balance the

weights of the respective classes. Kernel is selected as ‘linear’, ‘rbf’ and ‘poly’. Argument degree is only mentioned in case of kernel poly.

Let’s have a look at the classification report in each case.

Kernel = ‘linear’

```
Train accuracy = 98.49%
Test accuracy = 97.08%

 Confusion Matrix:

[[101   2]
 [  3  65]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.97      0.98      0.98       103
          1       0.97      0.96      0.96        68

avg / total       0.97      0.97      0.97       171
```

Kernel = ‘rbf’

```
Train accuracy = 97.99%
Test accuracy = 98.25%

 Confusion Matrix:

[[101   2]
 [  1  67]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.99      0.98      0.99       103
          1       0.97      0.99      0.98        68

avg / total       0.98      0.98      0.98       171
```

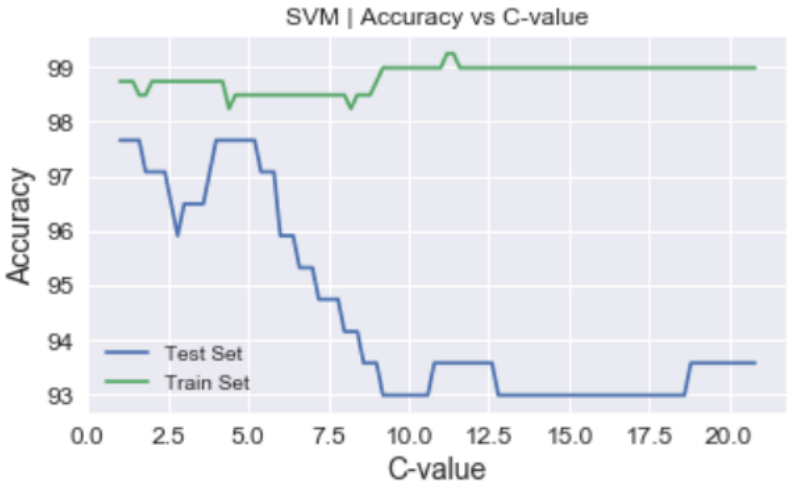
Kernel = 'poly'

```
Train accuracy = 93.22%
Test accuracy = 92.98%

 Confusion Matrix:

[[103   0]
 [ 12  56]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.90      1.00      0.94       103
          1       1.00      0.82      0.90        68

avg / total       0.94      0.93      0.93       171
```

Among all of these three kernel functions used, kernel = 'rbf' seems to be the best which predicts the test data with 98.25% accuracy.

Tuning the Penalty parameter could give us some better results:Penalty parameter C (= 1 , default)

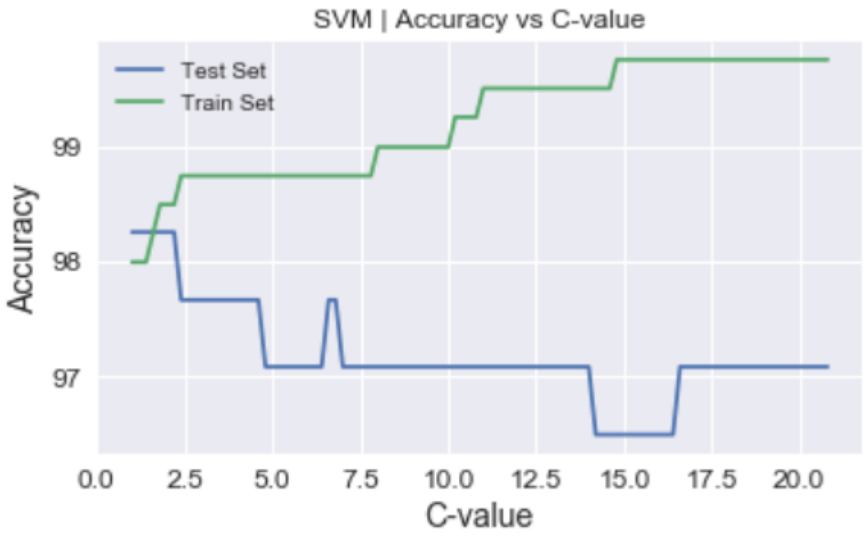
Kernel = 'linear' :



From the graph above we can set the penalty parameter C = 1.2 which looks to be the optimum value for high accuracy.

```
SVC(C=1.2, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=111, shrinking=True,
  tol=0.001, verbose=False)

Train accuracy = 98.74%
Test accuracy = 97.66%

 Confusion Matrix:

[[101   2]
 [  2  66]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.98      0.98      0.98       103
          1       0.97      0.97      0.97        68

avg / total       0.98      0.98      0.98       171
```

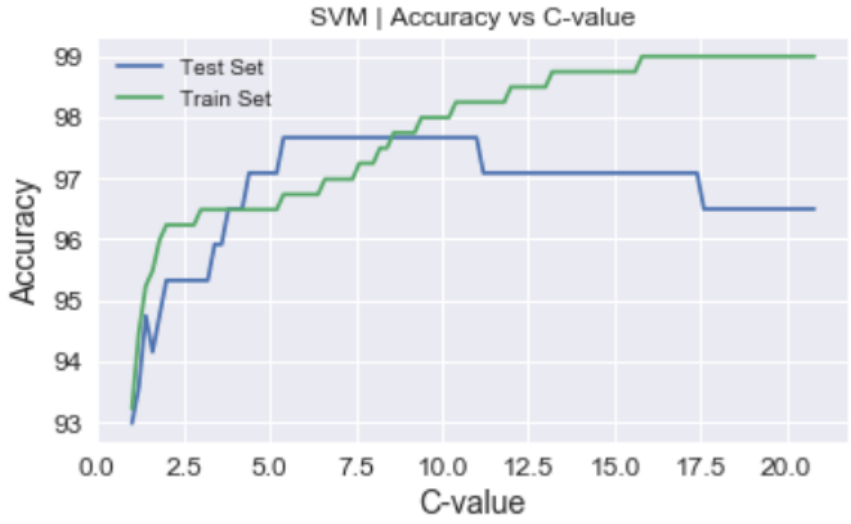
Kernel = 'rbf' :



From the graph above we can set the penalty parameter C = 2.3 which looks to be the optimum value for high accuracy.

```
SVC(C=2.3, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=111, shrinking=True,
  tol=0.001, verbose=False)

Train accuracy = 98.74%
Test accuracy = 98.25%

 Confusion Matrix:

[[101   2]
 [  1  67]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.99      0.98      0.99       103
          1       0.97      0.99      0.98        68

avg / total       0.98      0.98      0.98       171
```

Kernel = 'poly' :



Degree is set to 3, after observing changes w.r.t. degree value from 1 to 20. Degree = 3 is the best suitable one. Degree in all other kernels except poly is ignored by svc.

From the graph above we can set the penalty parameter C = 11 which looks to be the optimum value for high accuracy.

```
SVC(C=11, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=111, shrinking=True,
  tol=0.001, verbose=False)

Train accuracy = 98.24%
Test accuracy = 97.66%

 Confusion Matrix:

[[102   1]
 [  3  65]]

 Classification Report:

             precision    recall  f1-score   support

          0       0.97      0.99      0.98       103
          1       0.98      0.96      0.97        68

avg / total       0.98      0.98      0.98       171
```

Among all these C tuned svm classifiers, svm with kernel rbf are the most suitable ones for the data set and It has the most accurate results when compared to untuned and tuned C SVM classifiers. Tuned C svm classifier with kernel rbf has test set accuracy of 98.25% and train set accuracy of 98.74%.

Conclusion:

- C tuned SVM classifiers provide better accurate model in comparison to Decision tree and gradient boosted classifiers.
- Even though arguably gradient boosting classifier is devoid of overfitting and bias, we get better results through tuned C SVM with kernel = radial basis function(rbf) and It appears to be a right model for this data set among SVM's , Decision Tree classifiers and gradient boosting classifier.

Note: we could also tune gamma to further evaluate the model and improvise the predictive power through optimization.