Supervised Learning Experiments

- **Why did I choose these datasets?**

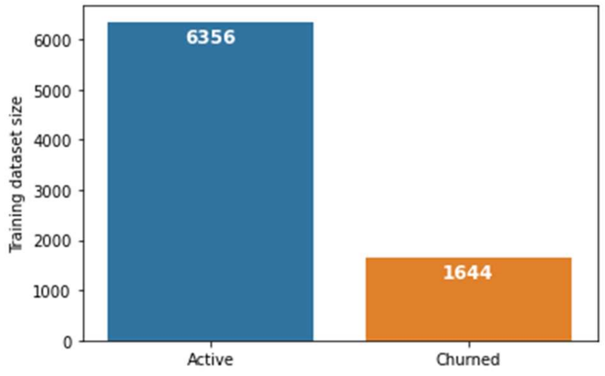
1. **Dataset 1 - Churn for Bank Customers**
   This dataset consists of customer's data of whether they are churned or not. It consists of 13 features and one column to predict whether the customer is churned (i.e. 1) or not (i.e. 0). As there are many features it was interesting to see which feature affects churning of the customer. Also, as the dataset is not balanced, it was interesting to see how each model learns and predicts. It has 2 categorical features, it was a good example to use one hot encoding. Also one of the features "Balance" ranged from 0 to 250898. So having features with wide ranges also can affect the predictions
2. **Dataset 2 - White Wine Quality**
   This dataset consists of 11 features which together decide the quality of white wine ("quality"). It ranges from a score of 0 (lowest quality) to 10 (excellent quality). This dataset is highly unbalanced, so it was interesting to decide whether few scores needed to be grouped together or what approach is to be followed. Also unlike previous dataset, this dataset has features of chemical compositions like chlorides, citric acid etc which are not commonly known and it's hard to depict how each feature affects the predictions.

Having different sizes of dataset was helpful to understand how the size of dataset affects the predictions. So I chose Dataset 1 with 10,000 rows and Dataset 2 with 4,898 rows.

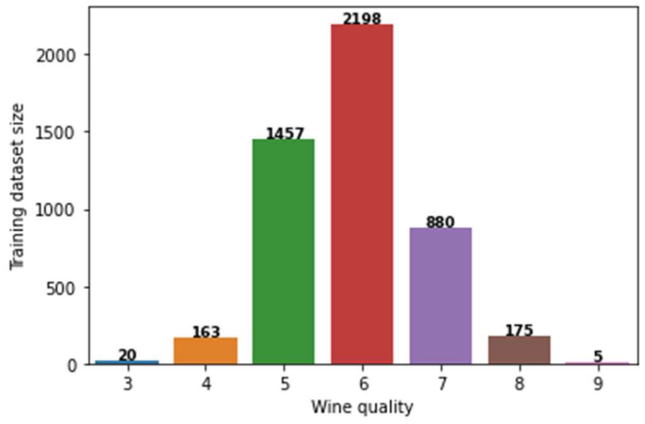
- **Important steps I followed for all algorithms**

1. The datasets were split to make the training set to be 80% and testing set to be 20%.
2. Algorithms like SVM, Neural Networks, KNN needed the dataset to be scaled as they are sensitive to the distance of data points from each other. Rest algorithms did not need scaling as they are not sensitive to variance in data (such as trees). Also scaling was performed separately after data was divided into train and test data to avoid data leakage issues.
3. I performed label encoding on "Geography" and "Gender" columns in Dataset 1 as they were categorical features. Then one hot encoding was applied so that the algorithms don't treat the label encoded features as high value high weightage and low value low weightage.
4. As the range of each hyperparameter is huge, it is difficult to try each value and find the best parameters. So I plotted a validation curve to understand the behavior of algorithms for different values. Also high training accuracy does not mean the model is good. So to avoid overfitting and underfitting, after plotting each validation curve, a range of values was selected to further check which is the best value.
5. I made sure that training and test accuracies were not very far from each other as it indicates the model is not properly trained. (i.e. training accuracy increasing while testing accuracy decreasing).
6. After finding the best parameters, I applied k-folds to make sure that the testing accuracy is not very far from the average accuracy of k-folds which again makes sure that the model is not overfitted.
7. For validation curves I have used f1-score as the performance metric and for learning curves, SVM kernels/ activation functions for NN, K-folds and final model I used accuracy as the performance metric.

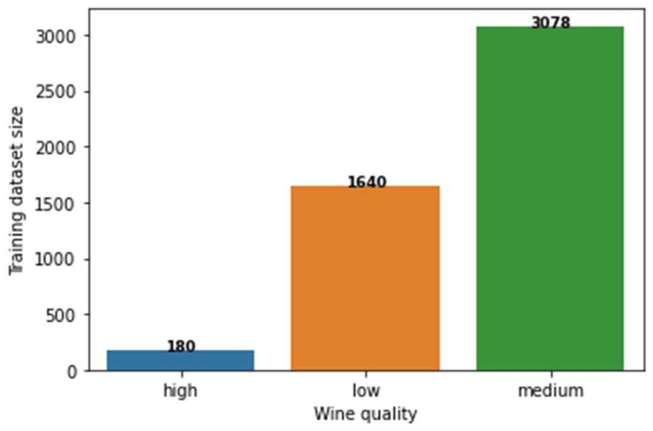
- **Results of data visualization and overall key points observed**
  Dataset 1 contains much greater data for Active customers.



For dataset 2, as seen in the first graph below, the distribution of data is very unbalanced, and the accuracy of algorithms was not exceeding ~55%. Therefore, I clubbed wine quality scores in low (wine quality <= 5), medium (6<=wine quality<=7), high (wine quality > 7). After doing so, there is still unbalanced data, but it is much better than the previous one.



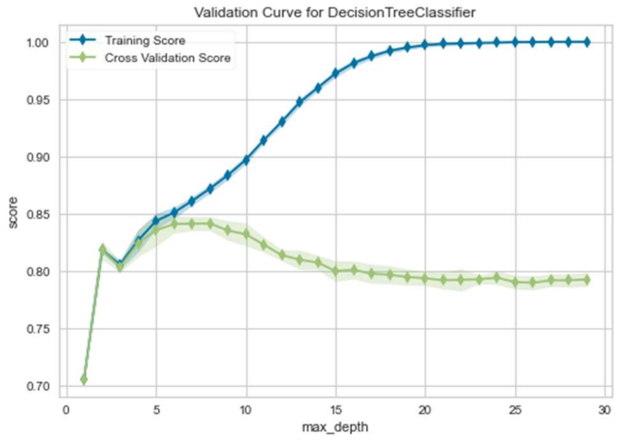
Original distribution of classes



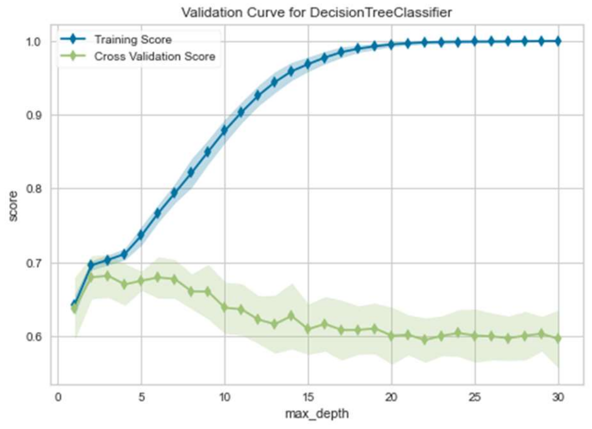
Revised distribution of classes

As the classes are not well distributed, it is observed from all 5 algorithm's confusion matrices that the models are predicting the "medium" quality wine much better than that of "high" and "low". With additional data for these classes, the models will be able to predict much better.

1. **<u>Decision trees with some form of pruning:</u>**
   Initially I ran DecisionTreeClassifier() without changing anything to see how the model is performing. For both datasets, the training accuracy was 100% indicating that the model was overfitted. I chose the hyperparameter max_depth. If the value is too small, it under-fits the model and if it's too large the model becomes more complex and does not capture needed patterns. So I plotted the curve for values from 0 to 30. For smaller values for both datasets the model was underfitting and for values above 15, the training data was achieving 100% accuracy indicating the model was overfitted. So for both datasets I passed max_depth below 6 so that both training and testing results were well fitted and not very far from one another.

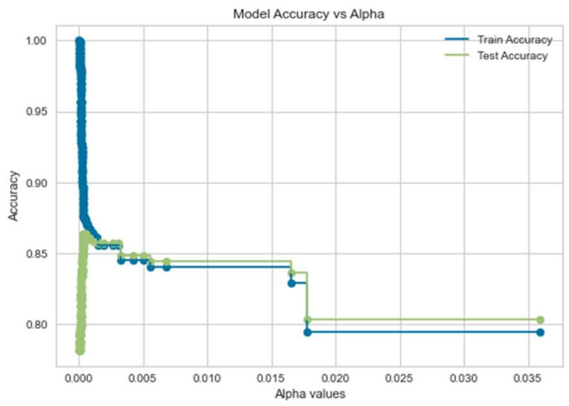


Hyperparameter curve (max_depth) Dataset 1

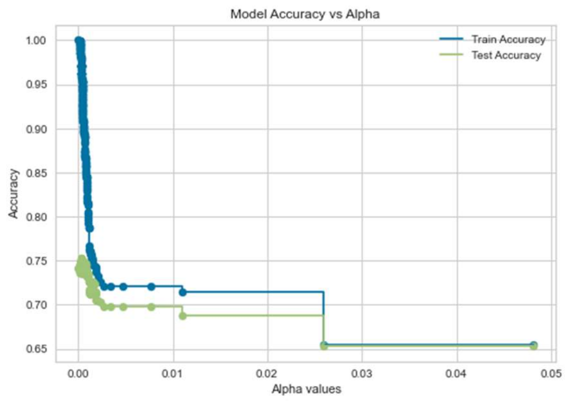


Hyperparameter curve (max_depth) Dataset 2

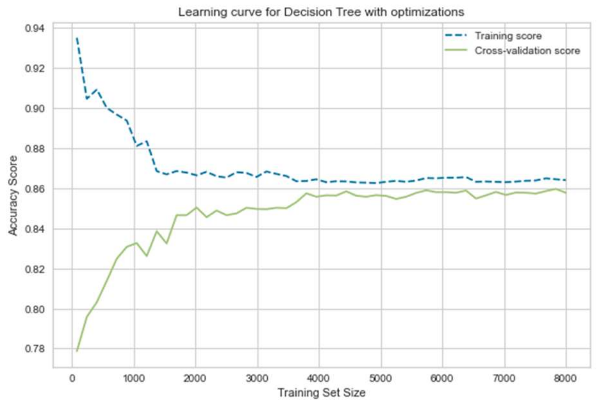
I also tried post pruning to see if ccp_alpha values were having any role in increasing the accuracy. But as seen in the graphs, for very small values the accuracy increased by around 1% for Dataset 1 and did not affect Dataset 2 a lot.



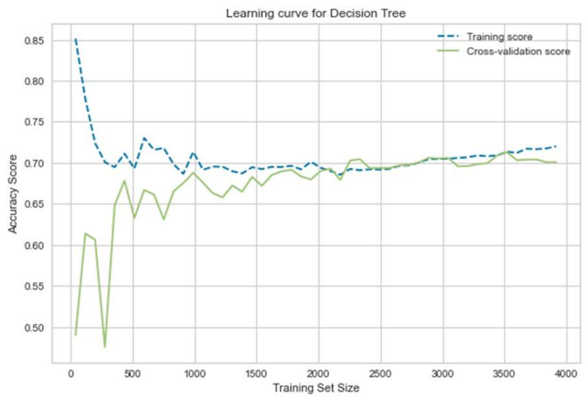
Dataset 1



Dataset 2



Learning curve for Dataset 1



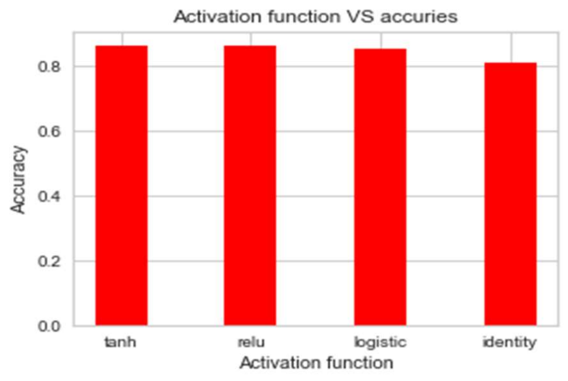
Learning curve for Dataset 2

Dataset	**Training Accuracy**	**Testing Accuracy**	**K-folds Accuracy**
Dataset 1	86.39%	86.35%	85.96%
Dataset 2	72.63%	70.30%	70.80%

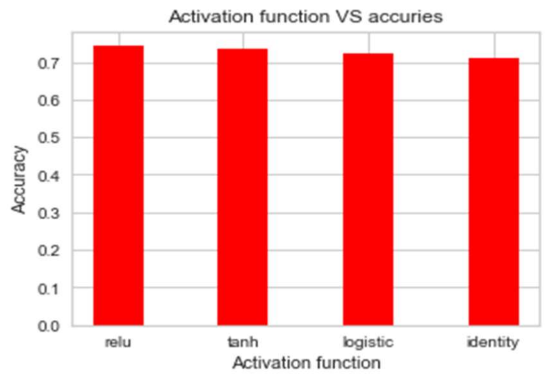
Table: Decision Tree accuracies with hyperparameter optimization

2. Neural networks

I chose to tune the activation function and hidden layers size. After plotting activation functions, I passed “tanh” and “relu” to GridsearchCV as they had the highest accuracies for both Datasets.

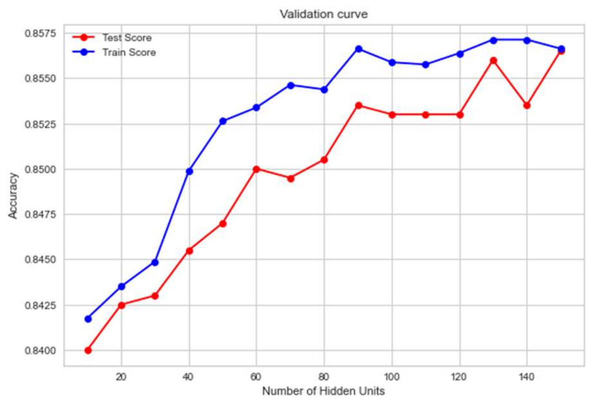


Activation functions Dataset 1

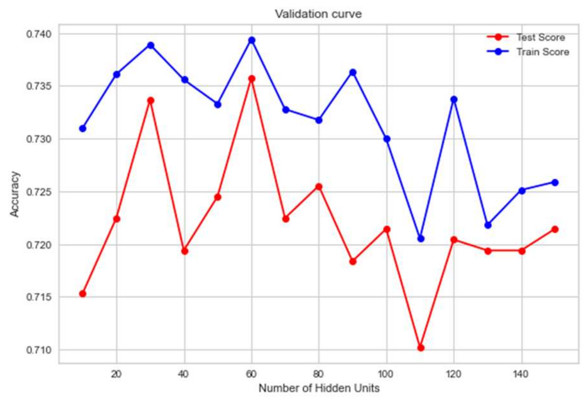


Activation functions Dataset 2

After plotting a hyperparameter curve for hidden_layer_sizes ranging from 10 to 150, I again chose a few hidden_layer_sizes where the accuracies were highest and passed to GridsearchCV.

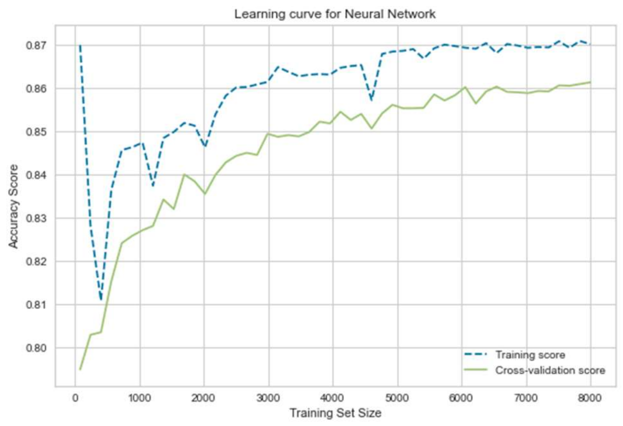


Hyperparameter curve (hidden_layer_sizes) Dataset 1

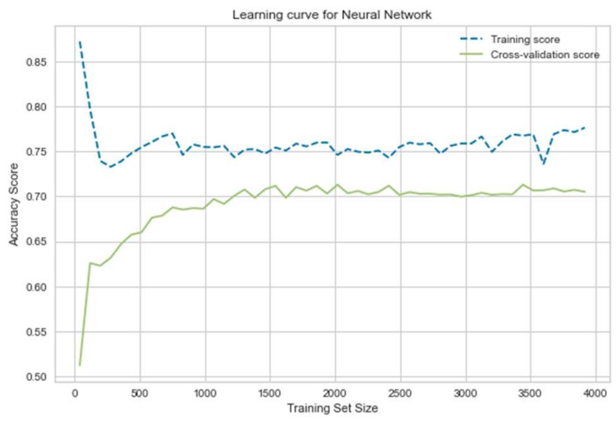


Hyperparameter curve (hidden_layer_sizes) Dataset 2

For both datasets’s activation function performing best was “tanh”.



Learning curve for Dataset 1



Learning curve for Dataset 2

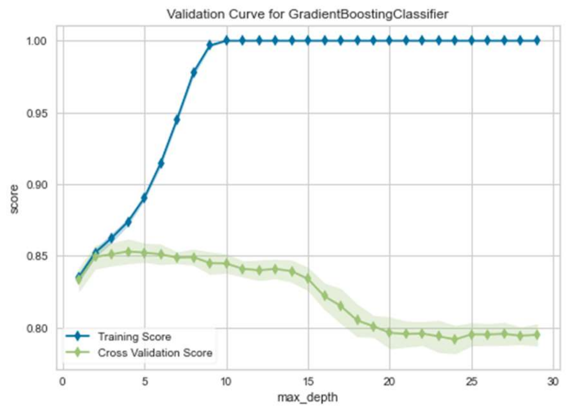
Dataset	**Training Accuracy**	**Testing Accuracy**	**K-folds Accuracy**
Dataset 1	86.93%	86.15%	86.27%

Dataset 2	76.49%	75.40%	72.35%

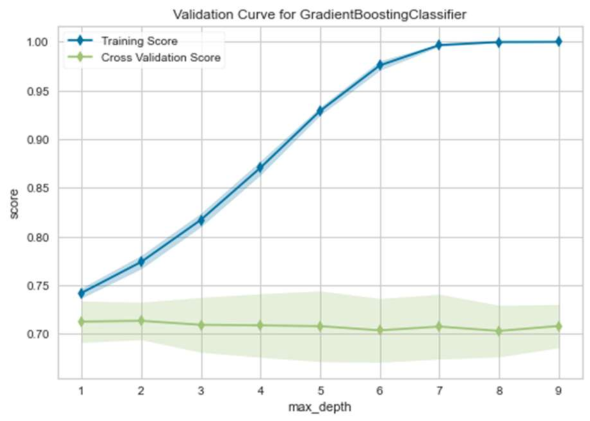
Table: Neural Network accuracies with hyperparameter optimization

3. Boosting (Gradient boosting)

For both datasets as seen in the hyperparameter curve for max_depth, the models start to overfit after a certain value.So they needed to be excluded from the GridsearchCV. Also, the accuracy of training keeps increasing while that for testing decreases, indicating the model is overfitted. This is due to the fact that after a certain depth, the testing set starts to ignore important patterns in between due to large depth.

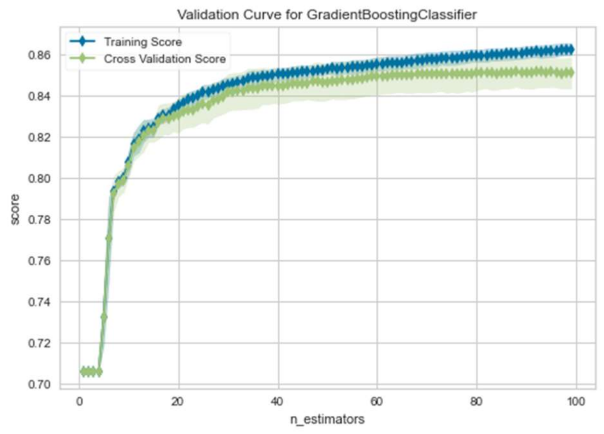


Hyperparameter curve (max_depth) Dataset 1

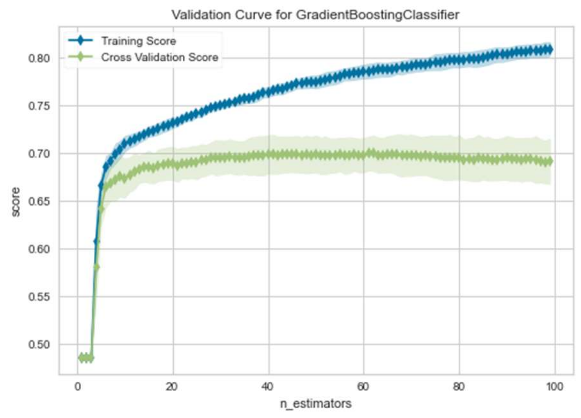


Hyperparameter curve (max_depth) Dataset 2

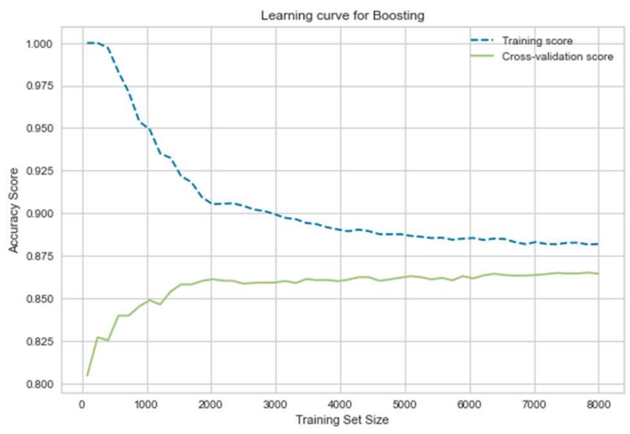
For n_estimators, as it is more resistant to overfitting, overfitting is not observed, but as the size of n_estimators increases, it increases the response time of the algorithm. So the value should be not very high but also there should be a minimum difference between training and testing accuracy.



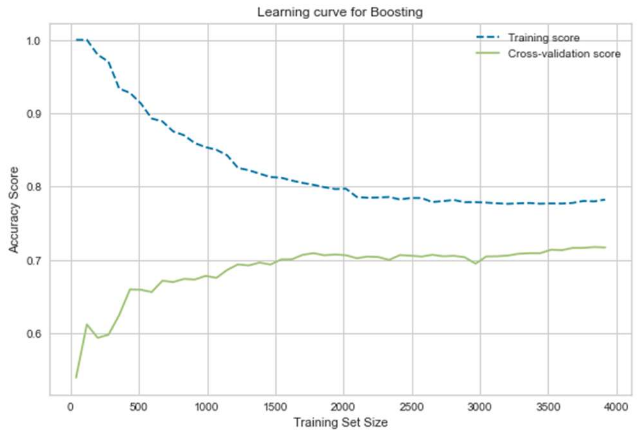
Hyperparameter curve (n_estimators) Dataset 1



Hyperparameter curve (n_estimators) Dataset 2



Learning curve for Dataset 1



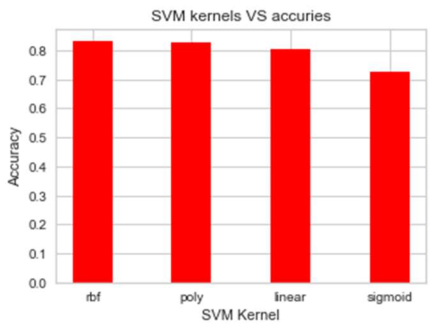
Learning curve for Dataset 2

Dataset	Training Accuracy	Testing Accuracy	K-folds Accuracy
Dataset 1	88.35%	86.85%	86.36%
Dataset 2	77.36%	72.85%	72.34%

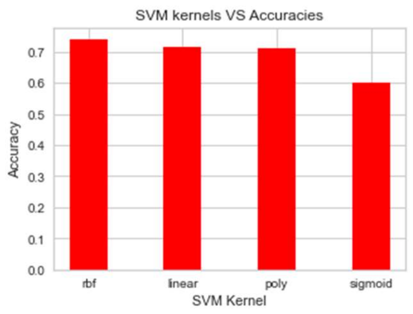
Table: Gradient boosting accuracies with hyperparameter optimization

4. <u>Support Vector Machines</u>

RBF and Linear kernels work better for Dataset 2. For Dataset 1 “poly” shows higher accuracy than “linear”.

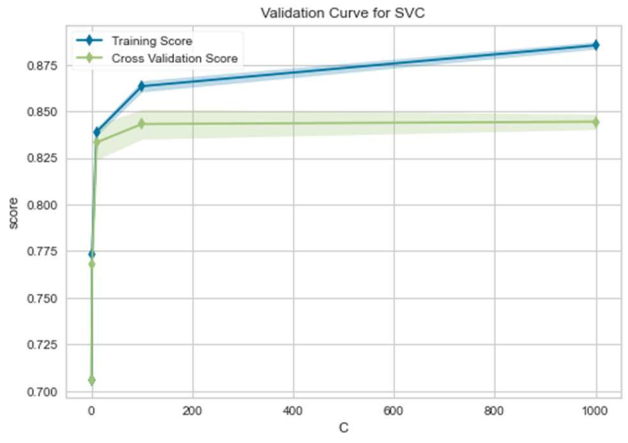


SVM Kernel performance Dataset 1

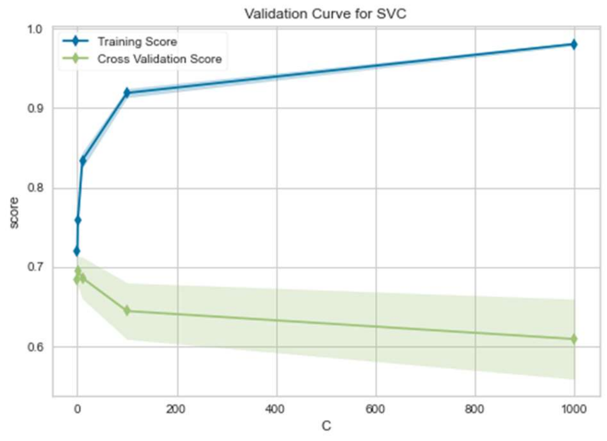


SVM Kernel performance Dataset 2

For Dataset 1, C value chosen is 1000 which means there is a small margin hyperline and that for Dataset 2 is 1 which means there is a larger margin hyperline. So as the C value for Dataset 1 is large, misclassification is less. For Dataset 2 as C value is small, there might be more misclassification but it is still less than remaining values of C as for larger values you can see in the second graph that the accuracy of the testing set starts decreasing.

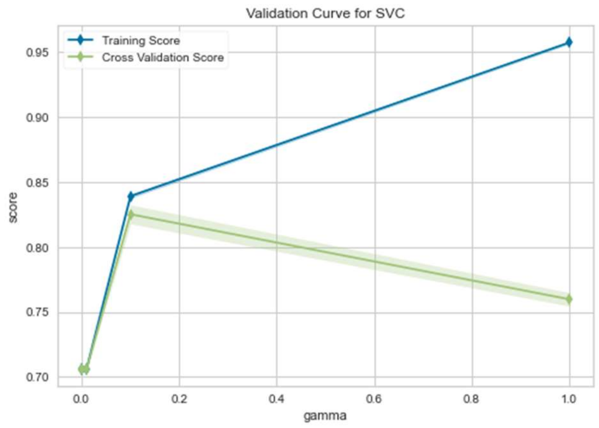


Hyperparameter curve (C) Dataset 1

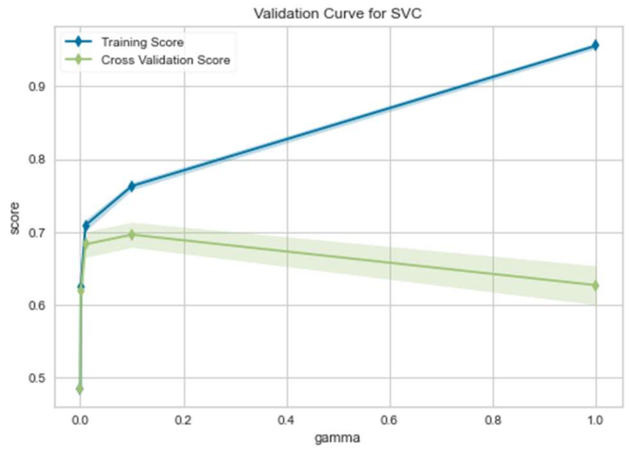


Hyperparameter curve (C) Dataset 2

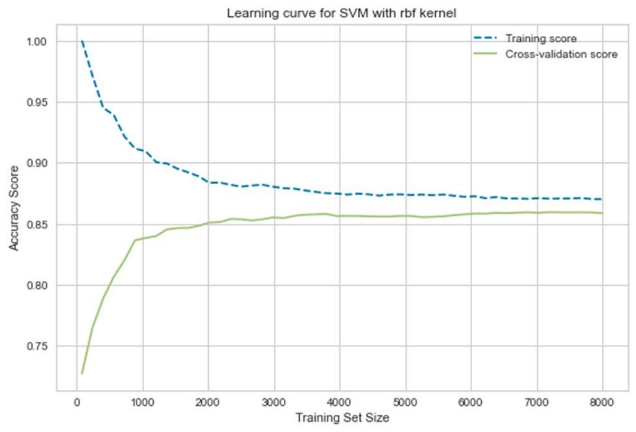
Gamma is only useful for the “rbf” kernel. For both the datasets the the optimal value is low, indicating that more training points are grouped together



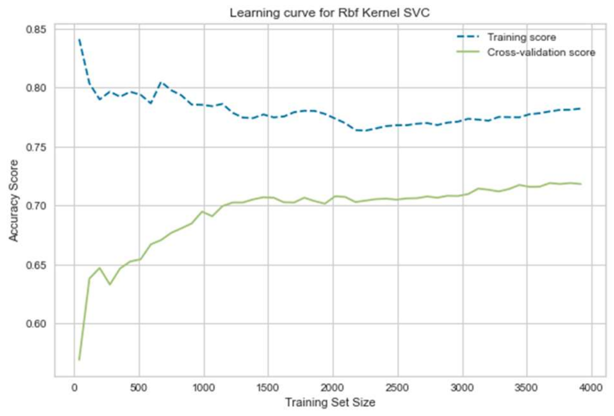
Hyperparameter curve (gamma) Dataset 1



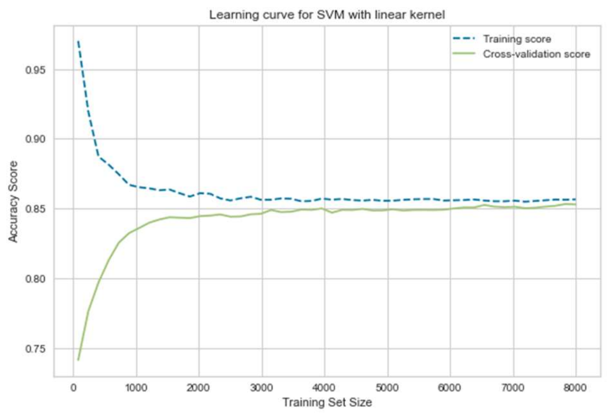
Hyperparameter curve (gamma) Dataset 2



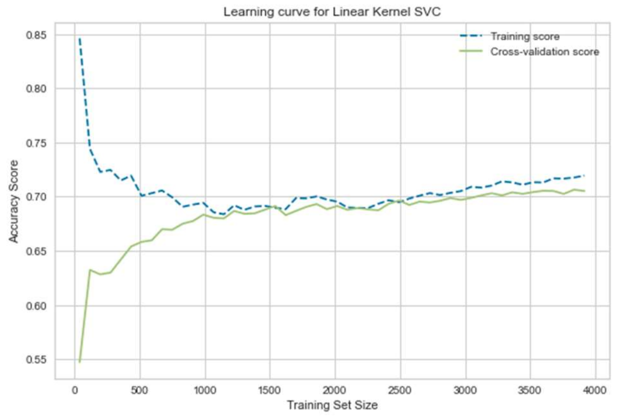
Learning curve (RBF kernel) Dataset 1



Learning curve (RBF kernel) Dataset 2



Learning curve (Linear kernel) Dataset 1



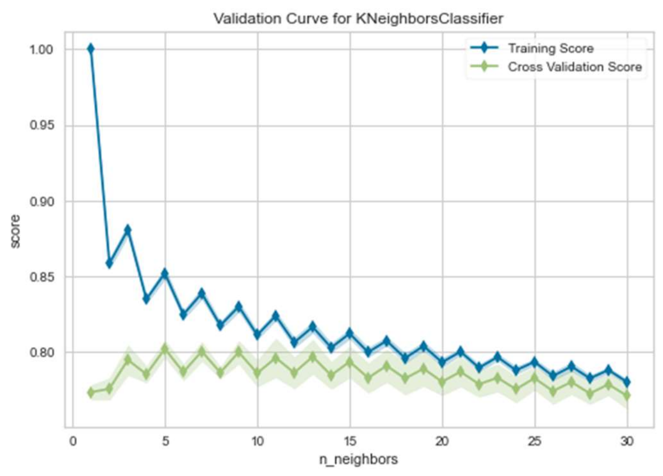
Learning curve (Linear kernel) Dataset 2

Dataset	**Kernel**	**Training Accuracy**	**Testing Accuracy**	**K-folds Accuracy**
Dataset 1	RBF	86.93%	85.9%	85.93%
	Poly	85.76%	85%	85.35%
Dataset 2	RBF	77.79%	74.59%	72.17%
	Linear	72.12%	71.32%	71.25%

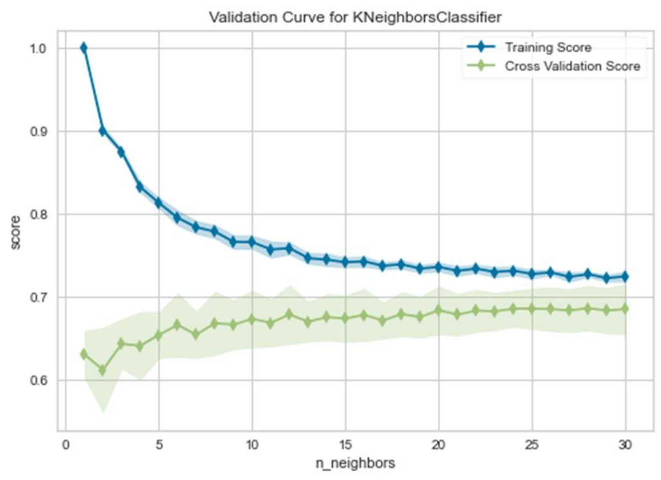
Table: SVM accuracies with hyperparameter optimization

5. <u>k-nearest neighbors</u>

The K value in KNN refers to the number of nearest neighbors used in the voting process. As seen in the graphs below, having very less value of k results in inaccurate results and noise will have higher influence on the predictions. And very large values can lead to misclassification.

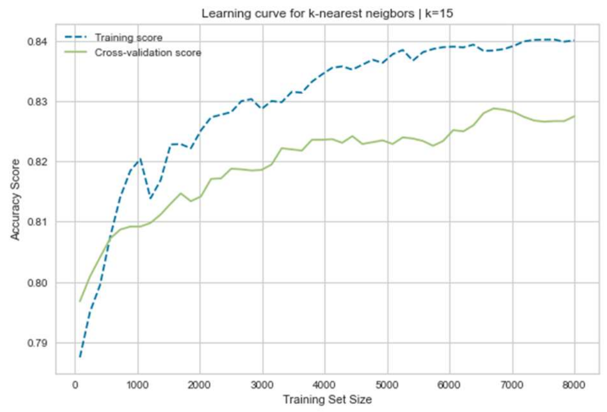


Hyperparameter curve (n_neighbors) Dataset 1

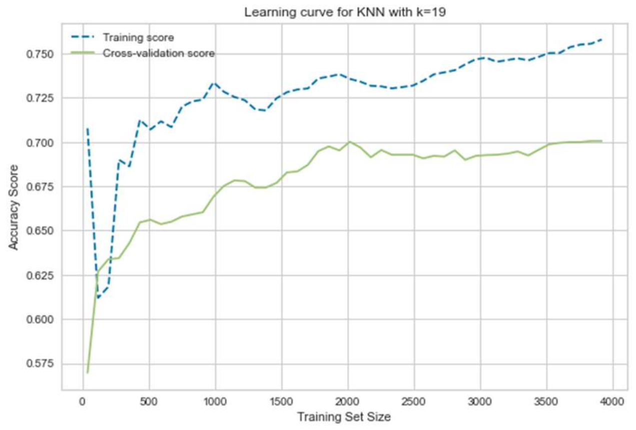


Hyperparameter curve (n_neighbors) Dataset 2

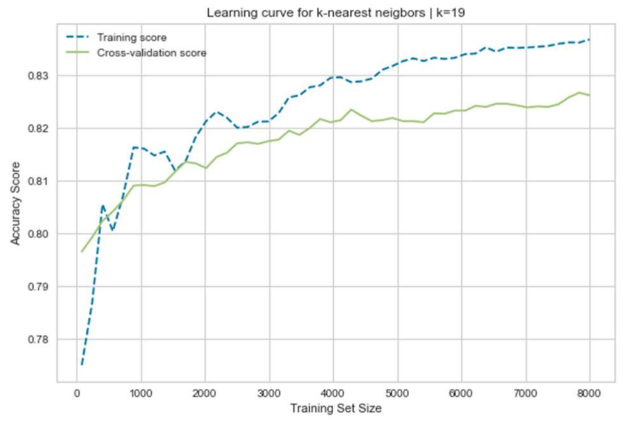
I used two values for n_neighbors for both datasets from mid range which gives me almost similar accuracies. Lower and higher values were underfitting and overfitting the model.



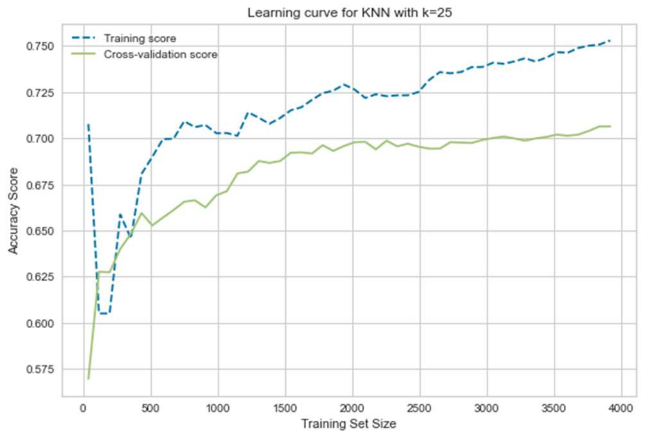
Learning curve (k = 15) Dataset 1



Learning curve (k = 19) Dataset 2



Learning curve (k = 19) Dataset 1



Learning curve (k = 25) Dataset 2

Dataset	**N_neighbors = 5 (default)**	**Training Accuracy**	**Testing Accuracy**	**K-folds Accuracy**
Dataset 1	default	86.63%	83%	82.30%
Dataset 2	default	81.92%	71.22%	67.70%

Table: KNN with default parameters

With the default n_neighbors as 5, it is seen that the training and testing accuracies are higher than after hyperparameter optimization shown in the table below. But after running k-folds it is evident that hyperparameter optimization is needed. Also for Dataset 2 there is a huge difference between training and testing accuracies, indicating that the model is overfitted and also the k-folds accuracy has dropped by a huge number.

Dataset	**n_neighbors**	**Training Accuracy**	**Testing Accuracy**	**K-folds Accuracy**
Dataset 1	15	84%	83.3%	82.69%
	19	83.58%	82.75%	82.66%
Dataset 2	19	75.93%	72.65%	71.90%
	25	75.93%	72.65%	72.42%

Table: KNN accuracies with hyperparameter optimization

- **Results**

Dataset 1

Algorithm	Model parameters	Accuracy
Decision Tree	max_depth = 6,ccp_alpha = 0.0005	85.96%
Neural Networks	hidden_layer_sizes = (60,) activation = 'tanh'	85.88%
Gradient Boosting	max_depth = 5, n_estimators = 46	**86.36%**
Support Vector Machines	kernel='rbf',C=1000,gamma=0.01	85.93%
k-Nearest Neighbors	n_neighbors = 15, p = 1	**82.69%**

The best performing algorithm is Gradient Boosting with 86.36% accuracy.

Dataset 1

Algorithm	Model parameters	Accuracy
Decision Tree	max_depth = 3, ccp_alpha = 0.00001	**70.80%**
Neural Networks	hidden_layer_sizes = (40,) activation = 'tanh'	72.35%
Gradient Boosting	max_depth = 3, n_estimators = 44	72.33%
Support Vector Machines	kernel='rbf',C=1,gamma=0.1	**73.17%**
k-Nearest Neighbors	n_neighbors = 25,p=1	72.41%

The best performing algorithm is Support Vector Machines with 73.17% accuracy.

As Dataset 2 was more unbalanced than Dataset 1, the accuracy for Dataset 2 is much lower than that of Dataset 1.

- **Links referred**

1. https://www.kaggle.com/mathchi/churn-for-bank-customers
2. https://www.kaggle.com/piyushagni5/white-wine-quality
3. https://www.kaggle.com/questions-and-answers/174904
4. https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3
5. https://www.analyticsvidhya.com/blog/2020/10/cost-complexity-pruning-decision-trees/
6. https://machinelearningmastery.com/data-preparation-without-data-leakage/
7. https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
9. https://scikit-learn.org/stable/common_pitfalls.html
10. https://neerajkumar.org/writings/svm/
11. https://medium.com/@pushkarmandot/what-is-the-significance-of-c-value-in-support-vector-machine-28224e852c5a
12. https://machinelearningmastery.com/a-simple-intuition-for-overfitting/
13. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
14. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
15. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
16. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
17. https://scikit-learn.org/stable/modules/neural_networks_supervised.html
18. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
19. https://towardsdatascience.com/understanding-the-confusion-matrix-from-scikit-learn-c51d88929c79