CS7641: Supervised Learning

Kamolphan Liwprasert (kliwprasert3@gatech.edu)

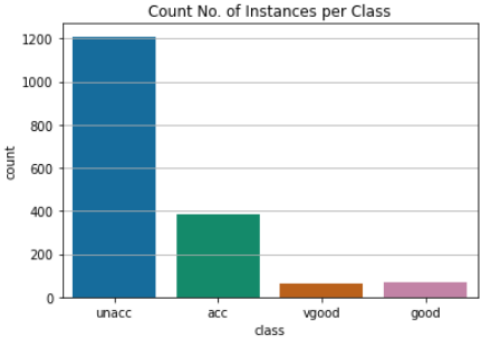
Note: this analysis is partly based on my previous semester (kliwprasert3, Fall 2020) [6]

1 Multi-class Classification Problem: CAR EVALUATION

1.1 Problem Overview and Dataset Selection

The Car Evaluation dataset is a multi-class classification problem for evaluating a car condition from a set of features in terms of prices and technical characteristics. This dataset is used for testing constructive induction and structure discovery methods as the features are derived from two main categories, Price and Tech (figure 1 - left). Car Evaluation data had been originally used in Machine Learning based on Function Decomposition paper [2]. The dataset is available on UCI Machine Learning Repository [1].

CAR car acceptability
. PRICE overall price
. . buying buying price
. . maint price of the maintenance
. TECH technical characteristics
. . COMFORT comfort
. . . doors number of doors
. . . persons capacity in terms of persons to carry
. . . lug_boot the size of luggage boot
. . safety estimated safety of the car



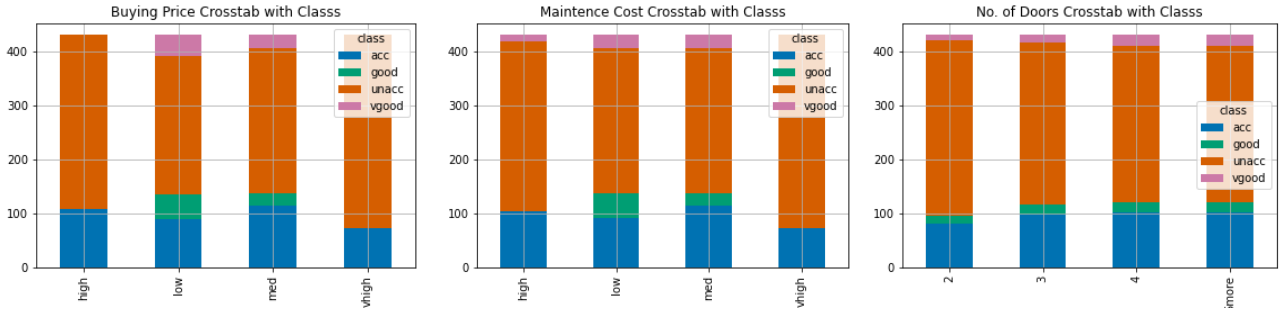
Left: Concept structure of features. Source: UCI repository.
Right: Imbalance target class plot

The dataset contains 1,728 unique instances and 6 category features. The target classes are 4 classes of the car evaluation; *unacceptable, acceptable, good,* and *very good*. Those target classes are imbalanced with the majority of the *unacceptable* class (figure 1 - right). The features are straightforward and do not need much of a feature engineering process. In terms of the data quality, this dataset has no missing value and is decent in number of instances and features. They are easy to understand and can be converted to ordinal numbers without the complex label encoding.

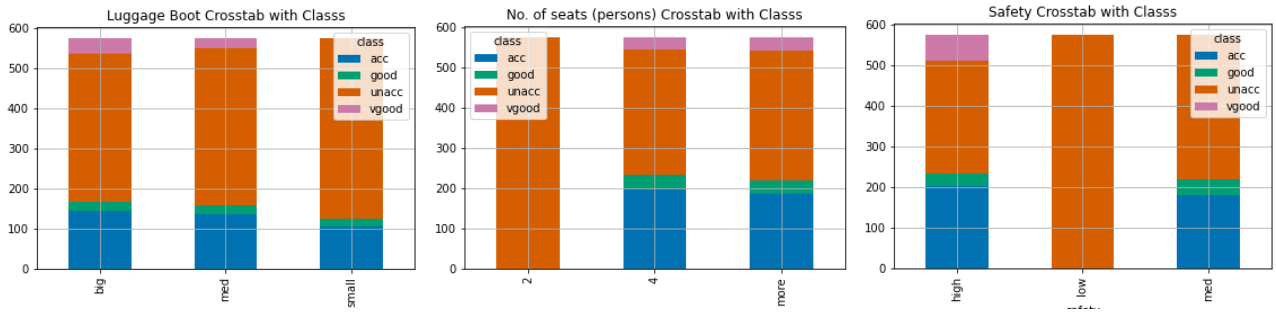
For my personal reason, my family has a small used car business in Thailand. I am quite familiar and interested in this domain. The result provides some insights in car evaluation that could be considered about customer decision factors. Hence, I think that it is a good start for a multi-class classification problem with imbalanced target classes.

1.2 Exploratory Data Analysis

To explore the data, I use the crosstab function (cross tabulation analysis) to plot the ratio of target classes for all features. The plot below shows the similar patterns among features. However, the *very good* class samples are very small.



The two features, *no. of seats* and *safety*, show that there are clear cut rules of a specified value such as all low safety cars are unacceptable, same to two-seat cars.

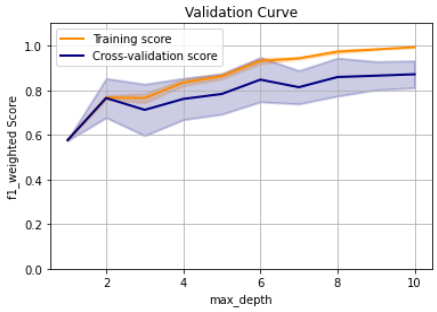


1.3 Train - Test Splitting and Choosing Metrics

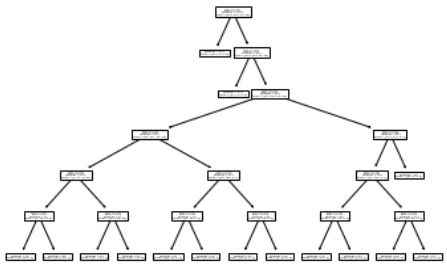
As mentioned in the previous section that the data is imbalanced, I used a stratified option and split the data into 70:30 (train:test) dataset, and used stratified 10-folds cross validation. **Weighted F1** score is used as the main metrics. It is recommended for the imbalance dataset because of the adjusted weight by combining precision and recall.

1.4 Decision Trees

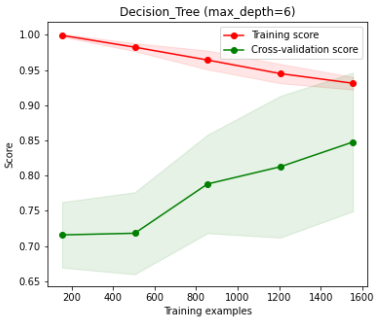
I used pre-pruning techniques to limit the **maximum level of tree depth** and **max leaf nodes**. So that it will not be too overfitting from over complicated rules. The splitting measure is the Gini index because it calculates the probability of impurity in classification. During the training, I choose the best max_depth first, then explore the max leaf nodes.



From the Validation Curve on the left, which plots F1 score on different max_depth parameters from 1 to 10, it shows that at **max_depth = 6** can get the best cross-validation compared to the earlier. And this makes sense because there are only 6 features in this dataset, the more levels than the number of features would be unnecessary as it overfits towards the too complex tree at the end, where the cross-validation score does not significantly increase anymore.



Tree Structure (max_depth = 6)

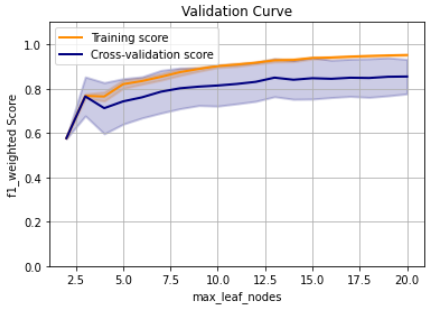


Learning curve(max_depth=6)

```
max_depth =  6
F1 Score:  0.898909060755405
Training Accuracy:  0.9172870140612076
Testing Accuracy:  0.9017341040462428
Confusion Matrix
[[352  10   1   0]
 [ 22  89   2   2]
 [  0   7  11   3]
 [  0   4   0  16]]
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       363
           1       0.81      0.77      0.79       115
           2       0.79      0.52      0.63        21
           3       0.76      0.80      0.78        20

    accuracy                           0.90       519
   macro avg       0.82      0.77      0.79       519
weighted avg       0.90      0.90      0.90       519
```

Comparing the metrics from 6-depth and 5-depth trees on the right, the 5-level tree's confusion matrix shows that it is missing one class output. This can infer that the max_depth of the decision tree cannot be lower than 6 in this classification problem. Otherwise, it does not have enough variance to fit all classes. The learning curve shows that CV scores are generalized well.

```
max_depth =  5
F1 Score:  0.857956771216
Training Accuracy:  0.874
Testing Accuracy:  0.8689
Confusion Matrix
[[330  32   0   1]
 [  8 105   0   2]
 [  0  11   0  10]
 [  0   4   0  16]]
```



To further prune the decision tree, I explored the effect of **max leaf nodes for max_depth = 6.** The max leaf nodes parameter ranges from 2 to 20. This plot shows that the tree needed at least max leaf nodes = 8 to cover all the target classes in the confusion matrix with weighted F1 = 0.8630. The weighted F1 is gradually increasing till **max leaf nodes = 14**, the weighted F1 score is saturated at 0.8989 and does not improve anymore. So that it can achieve the same accuracy as no max leaf limit. The wall clock time for decision tree fitting and cross validation is about 1.3 seconds on average.

1.5 Neural Networks

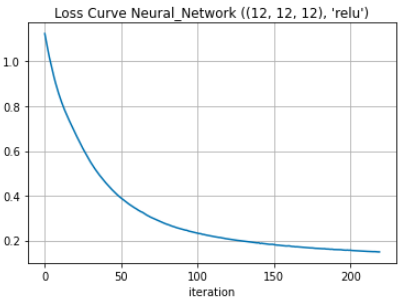
In this task, I used multi-layered perceptron classifiers with one and more hidden layers. The Scikit-learn's *MLPClassifier* optimizes the log loss function using stochastic gradient descent [5]. The activation function is *ReLU* (Rectified Linear Unit) because the computation is more simple and easier to converge than the sigmoid function that requires exponential calculation. The *MinMaxScaler* is used to preprocess and standardize the training input. The neural networks were trained with 42 different network structures varied in number of one, two, and three hidden layers. The learning rate is constant with a default value of 0.001. The maximum iteration is 1,000 with early stopping if there is no change within 100 iterations. The model training took between 150 - 700 iterations.

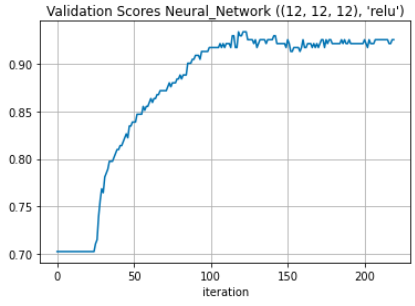
The neural networks with one hidden layer performed poorly in F1 score. The hidden layer size under 256 can output only two majority target classes out of four. This is in contrast with the loss curve and validation score of the neural networks that did not take the imbalance target classes into account and resulted in low weighted F1 score. The best number of nodes is 256 with weighted F1 score = 0.693. However, with a larger size of 512, the F1 score was dropped by 0.1 because of overfitting.

The two hidden layer neural network performance result was interesting. With network size (6, 6) can output all four target classes with F1 score = 0.669 which is close to one hidden layer of (256). The F1 scores look better than one hidden layer, but none of them has an F1 score more than 0.69. The larger network sizes focus on reducing the loss, hence output only two classes which are prone to be overfitting.

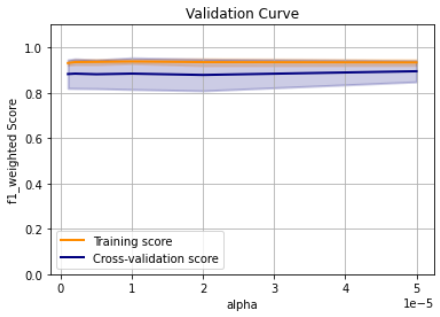
On the other hand, the three hidden layer neural network with **network size of (12, 12, 12)** was the only one that has **F1 score = 0.708** which is the highest among other network sizes. The network can output all target classes as well. The rest of them performed similarly with two hidden layer networks. Though, the performance is much lower than decision trees.

```
hidden_layer_sizes =  (12, 12, 12) , activation =  relu
F1 Score:  0.7080846170113124
Training Accuracy:  0.7237386269644334
Testing Accuracy:  0.6994219653179191
Confusion Matrix
[[281  77   2   3]
 [ 39  73   3   0]
 [  2  10   6   3]
 [  0   7  10   3]]
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.77      0.82       363
           1       0.44      0.63      0.52       115
           2       0.29      0.29      0.29        21
           3       0.33      0.15      0.21        20

    accuracy                           0.70       519
   macro avg       0.48      0.46      0.46       519
weighted avg       0.73      0.70      0.71       519
```





The loss curve shows that the error was going down by each iteration during the neural network training. As opposed to the validation score that goes higher by iteration and achieves more accuracy in validation steps. From the confusion matrix can be seen that the model still misclassified the minority classes.



For the attempt to generalize the neural network models, the 10 **alpha** values for *L2 regularization* has been chosen between $1e^{-6}$ to $1e^{-1}$, though there is no significant improvement between each value. So I scoped down the alpha to be around the default value of $1e^{-5}$ and the change does not improve the weighted F1 metric of the model as shown on the left figure.

The wall clock time for neural network training ranges between 13 seconds to more than a minute for complex networks. The average training and validation time is 36 seconds.

1.6 Boosting

Adaptive Boosting *(AdaBoostClassifier)* is used in this task. It consists of multiple weak learners of decision trees by the number of "***n_estimators***". I compared the effect of the number of weak learners and the **max_depth** of each decision tree. To keep the weak learner simple, the pre-pruning of max_depth between 1 and 3 (default) is applied to prevent high complexity trees. The n_estimators range between 3 to 2,000 in 20 different values.

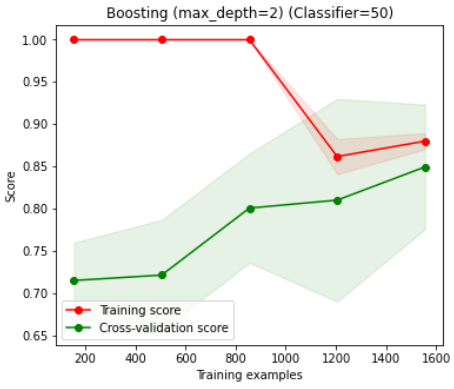
For the number of the weak learner with max_depth = 1, the model has a decent weighted F1 score of 0.753 given the low number of n_estimators as 3, which performed better than the neural network. The F1 and training score does not change after n_estimators more than 100. In this case, the highest F1 score is 0.872 when n_estimators is 60.

When the max_depth = 2, the F1 score of n_estimators = 3 started at 0.843. It reaches the highest F1 score at 0.8903 when n_estimators is 50 which has a similar score to the decision tree. Similar to the previous case, the F1 score of n_estimators does not improve 80, it is stable around 0.88.

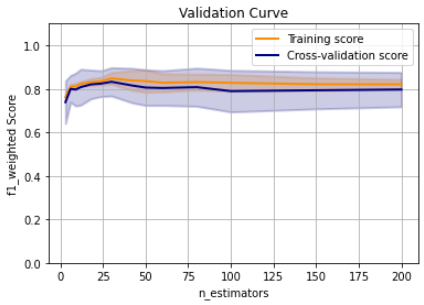
For max_depth = 3, it performed a little bit worse than max_depth = 2 with F1 score around 0.86. However, when n_estimators = 800, the F1 score has been gradually improved to be 0.8904 and stable after that. I think that this setting has too many complex trees to achieve the simple tasks. Hence, **max_depth = 2 with 50 estimators** is the best optimal value.

The average wall clock time for each Ada boosting training and CV is 10.6 seconds.

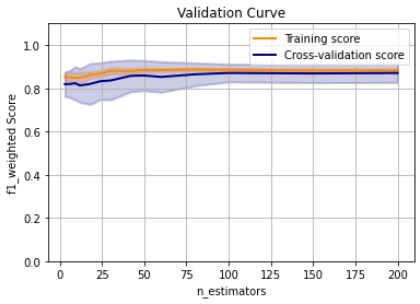
```
AdaBoost (max_depth = 2): n_estimators =  50
F1 Score:  0.8903058046144933
Training Accuracy:  0.8759305210918115
Testing Accuracy:  0.8863198458574181
Confusion Matrix
[[325  37   1   0]
 [  8  94   3  10]
 [  0   0  21   0]
 [  0   0   0  20]]
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.90      0.93       363
           1       0.72      0.82      0.76       115
           2       0.84      1.00      0.91        21
           3       0.67      1.00      0.80        20

    accuracy                           0.89       519
   macro avg       0.80      0.93      0.85       519
weighted avg       0.90      0.89      0.89       519
```



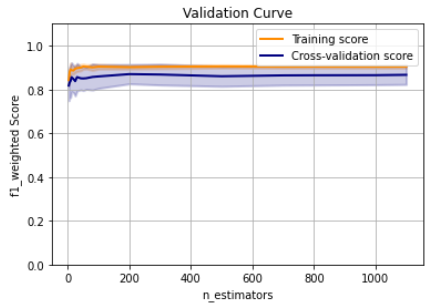
The validation curves showed the training score and CV score for each n_estimators for each setting of weak learners.



(max_depth = 1)



(max_depth = 2)

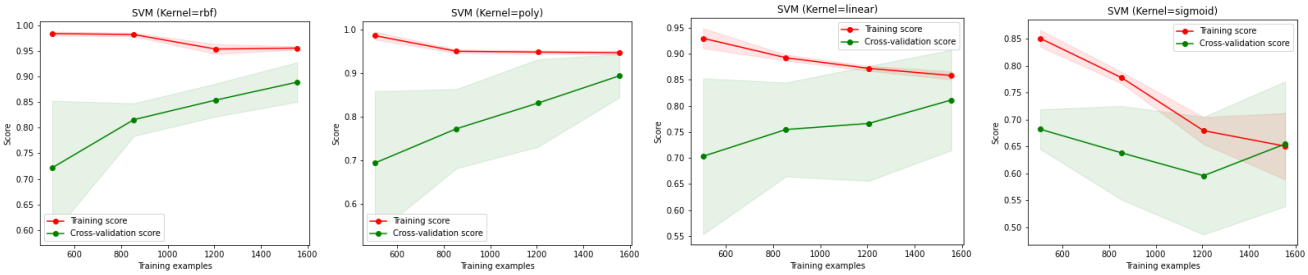


(max_depth = 3) *rescaled to account for n_estimators = 800

1.7 Support Vector Machines

In the Support Vector classification task, four different kernel functions have been explored; *linear*, *polynomial*, *rbf*, and *sigmoid*. The decision function shape is one-vs-rest (ovr) for the multi-class classification.

For each kernel sorted by the F1 score, the *rbf* kernel performed best at 0.921, followed by *polynomial* at 0.919, then *linear* kernel at 0.823, and lastly *sigmoid* at 0.50. The *rbf* and *poly* kernel performed well with higher scores than the decision tree because both functions support higher dimensional mapping better than linear function. It also showed that the *sigmoid* kernel only supports binary classification because it outputs only two classes. The plots below show training / CV scores.

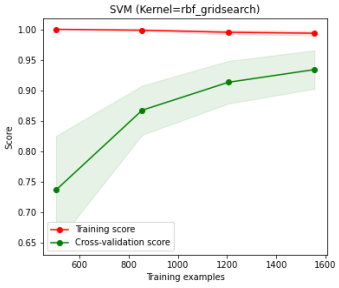


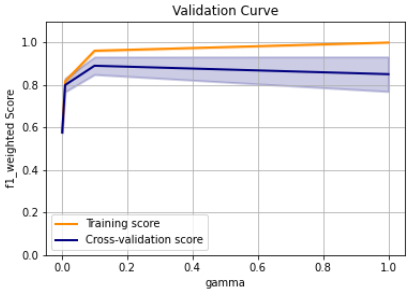
The ***rbf*** kernel is chosen to further explore the effect of **hyperparameter tuning** using Grid Search cross validation. The parameters for *rbf* are *C*, the regularization parameter, and *Gamma*, the kernel coefficient. I used the different magnitudes of C from 0.1 to 100, and Gamma from 0.00001 to 10 that combined into 24 model candidates. The result from Grid Search improves the F1 score to 0.9863 with C = 100, and Gamma = 0.1 as shown in the training plot below.

```
Best Estimators:
 SVC(C=100, gamma=0.1)
F1 Score:  0.9863600083942065
Training Accuracy:  0.9991728701406121
Testing Accuracy:  0.9865125240847784
Confusion Matrix
[[363   0   0   0]
 [  5 110   0   0]
 [  0   2  19   0]
 [  0   0   0  20]]
Classification Report
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       363
           1       0.98      0.96      0.97       115
           2       1.00      0.90      0.95        21
           3       1.00      1.00      1.00        20

    accuracy                           0.99       519
   macro avg       0.99      0.97      0.98       519
weighted avg       0.99      0.99      0.99       519
```

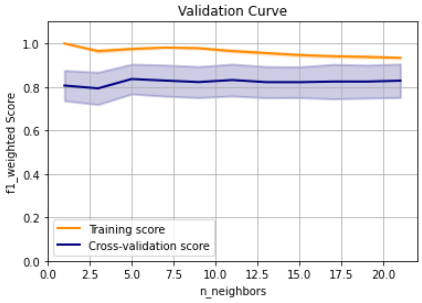
The validation curve below shows the effect of different Gamma values from 0.001 to 1 where 0.1 gives the highest CV score based on weighted F1.

The average wall clock time for SVM training and prediction is 1.5 seconds





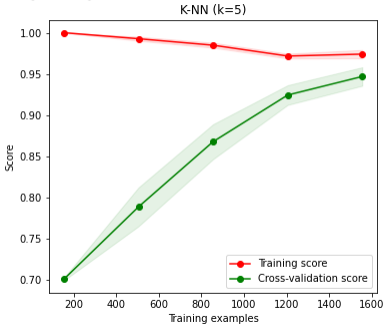
1.8 K-Nearest Neighbors



The different k values are used to explore the effect of the parameter. I have tried k in odd numbers from 1 to 19 to search for the best k that maximize F1 score.

From the validation curve on the left, k = 5 has the highest F1 score at 0.93493 before slightly declining from being too generalized with high k-value. If the k is too low, it will have higher variance and be more dependent on the closest data points. Hence, k = 5 is a good balance for this dataset.

```
K =  5
F1 Score:  0.9349326069234694
Training Accuracy:  0.9727047146401985
Testing Accuracy:  0.9383429672447013
Confusion Matrix
[[357   6   0   0]
 [  8 106   1   0]
 [  1  11   9   0]
 [  0   4   1  15]]
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       363
           1       0.83      0.92      0.88       115
           2       0.82      0.43      0.56        21
           3       1.00      0.75      0.86        20

    accuracy                           0.94       519
   macro avg       0.91      0.77      0.82       519
weighted avg       0.94      0.94      0.93       519
```



The results from multiple k values in the k-NN are consistent and performing quite well compared to the other algorithms. The k-NN is the fastest in the training time as well. The average wall clock time in prediction and evaluation takes less than a second (0.98 s).

1.9 Conclusion

Car Evaluation	Decision Tree	Neural Network	AdaBoosting	SVM	k-NN
Main Params	Max_depth = 6 Max_leaf_nodes = 14	Hidden layer size (12,12,12), ReLU	Max_depth = 2 N_estimators = 50	**RBF kernel** **C = 10, gamma = 0.01**	k = 5
Weighted F1	0.89891	0.70808	0.89030	**0.98636**	0.93493
Wall clock time	~1.3 seconds	~36 seconds	~10.6 seconds	~1.5 seconds	**~ 1 second**

SVM with *rbf* kernel has the best performance among five algorithms and is considered a fast algorithm. The C and gamma of the rbf kernel can be tuned to achieve the highest weighted F1 score. With 10-fold cross validation, it helps reduce the bias but it cannot guarantee not overfitting if the *rbf* kernel is overly complex. **K-NN** with k=5 has the second best performance. The K-NN model can be well adaptive to the newer data and does not require model training. The prediction time is the fastest of all models. Though, it depends on the amount of data points. K-NN could be slower if there are a lot of data points. Overall, both of them are pretty good algorithms for imbalanced dataset like Car Evaluation.

Decision Tree and Adaptive Boosting can achieve similar scores due to the same tree based estimator. **Decision Tree** is a simple model with high interpretability and easy to understand the criteria of tree splitting. This property is useful in business settings where people could apply the same decision logic. The model training and prediction is fast as well with pre-pruning conditions. **Adaboost** and weak learners are decent in this task, but they are 10 times slower than a decision tree. I think that Histogram-based Gradient Boosting could perform better than AdaBoosting due to higher complexity.

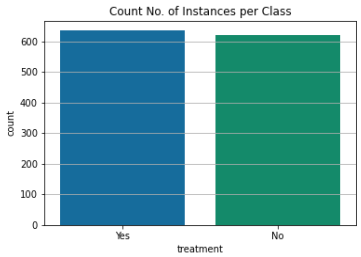
In contrast, **Neural Network** performs worst in the car evaluation task which has imbalance target classes. Not only does the model take longer time to train, but also has low interpretability to determine the business logics or insights from the black box model. Therefore, it is not suitable for the Car Evaluation task.

---

2 Binary Classification Problem: MENTAL HEALTH IN TECH SURVEY (2014)

2.1 Problem Overview and Dataset Selection

Mental Health in Tech survey (2014) is a survey conducted by OSMI, Open Sourcing Mental Illness Ltd. It contains 1,259 instances and 27 features. Although this data is a survey, many people have used this dataset to classify if a person needs mental health treatment from a set of features. The *treatment* column has only two possible values, *yes* and *no*. Therefore, the task of this problem is a binary classification to predict the target class *treatment*. The Mental Health in Tech Survey (2014) data is available on Kaggle [3].




```
print("Len(): ", len(data['Gender'].str.lower().unique()))
data['Gender'].str.lower().unique()
```

```
Len():  43
array(['female', 'm', 'male', 'male-ish', 'maile', 'trans-female',
       'cis female', 'f', 'something kinda male?', 'cis male', 'woman',
       'mal', 'male (cis)', 'queer/she/they', 'non-binary', 'femake',
       'make', 'nah', 'all', 'enby', 'fluid', 'genderqueer', 'female ',
       'androgyne', 'agender', 'cis-female/femme', 'guy (-ish) ^_^',
       'male leaning androgynous', 'male ', 'man', 'trans woman', 'msle',
       'neuter', 'female (trans)', 'queer', 'female (cis)', 'mail',
       'a little about you', 'malr', 'p', 'femail', 'cis man',
       'ostensibly male, unsure what that really means'], dtype=object)
```

Left: Plot of number of instances per class showing balanced data.

Right: The example input in the Gender column that varies in many forms.

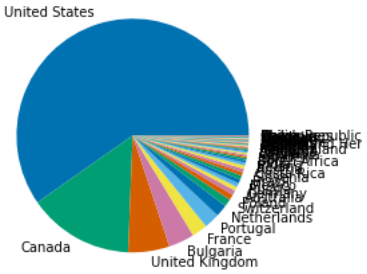
The Mental Health in Tech survey dataset is decent in number of instances and features. The columns have more variety than the Car Evaluation dataset. But the target class is balanced which is good for binary classification. In terms of the data quality, this dataset has some missing values that needed some imputation. Some columns are not necessary in training the model that can be dropped, such as States and comment. One of the most challenging tasks is data

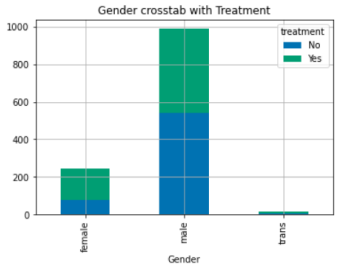
preprocessing. For example, there are 43 different unique values in the Gender column that need to be categorized into 3 types. Then, the label encoding can be done after preprocessing.

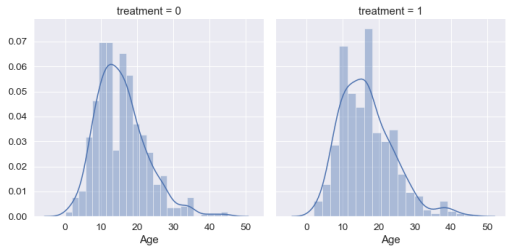
The reason that I chose this dataset is because I have a friend who got depressed while working in a tech company. She did not seek treatment, but we discussed a lot about mental health in the workplace. So this became the area of my interest to understand more and try to find a better way to improve the work environment in tech companies. I have tried the latest mental health data released in 2019, but the survey was getting more complex with 82 feature columns. Hence, I preferred the data in version 2014 as it is the simpler version, so I can focus more on the analysis and modeling part.

2.2 Exploratory Data Analysis

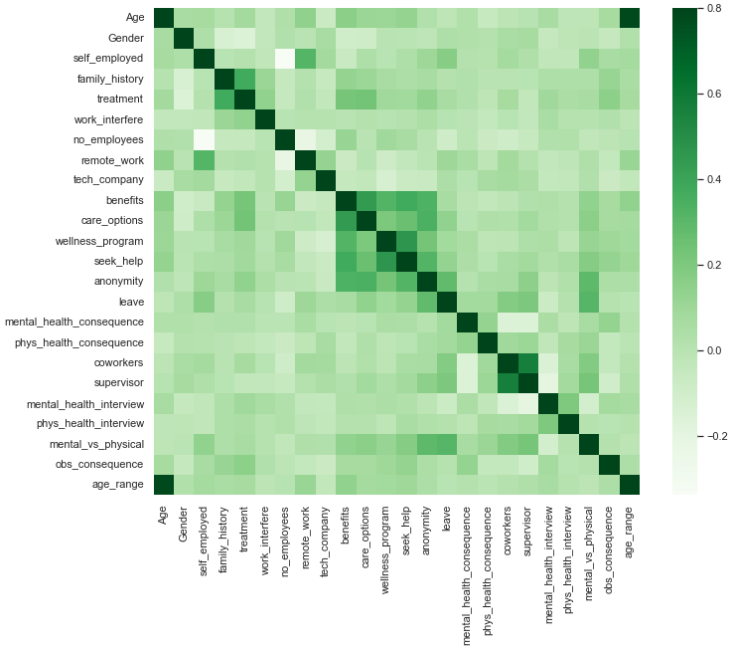
During the Exploratory Data Analysis (EDA) process, these are some of insights about the dataset:



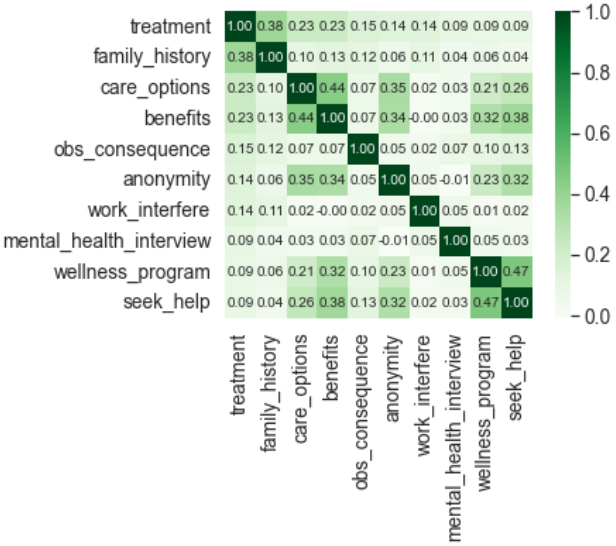




- Majority of the participants were from the U.S., followed by Canada, the U.K. So, I chose to drop this column.
- Male is the majority Gender in this survey. The target class distributions are quite balanced in most of the features
- There are differences in age distribution for those who have treatment and haven not as shown on histograms.



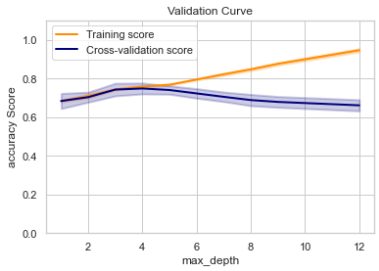
The two figures on the left and down are the correlation plot of the Mental Health in Tech dataset



2.3 Train - Test Splitting and Choosing Metrics

I used 80:20 for train_test_split in order to get more training data and used 10-fold cross validation. *MinMaxScaler* is used to standardize the training input.. Since the class label is balanced, I use accuracy score and ROC score as main metrics for binary classification. The ROC curve is often used in clinical tests as it can visualize both true positive rate (sensitivity) and false positive rate (1 - specificity). The ROC area under curve (AUC) is also another metric to compare.

2.4 Decision Trees



The pre-pruning techniques to apply to decision trees are **maximum level of tree depth** and **max leaf nodes**. Gini index is used as the splitting criteria.

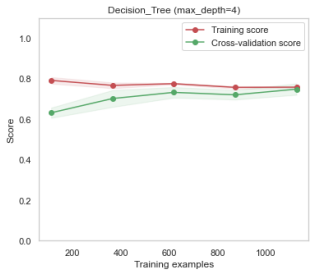
First, I set the *max_depths* ranging from 1 to 12. The validation curve from different max_depth values shows that the best max_depth is between 4 and 5. Then it will get overfitting as the cross-validation goes down because of the high variance and low bias. The tree will be too complex with the higher value of max_depth.

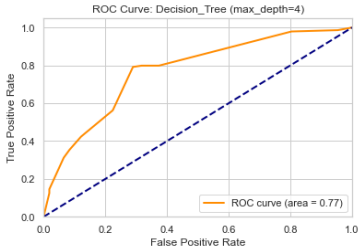
Comparing between max_depth = 4 and 5. The decision tree with max_depth = 4 has a bit better accuracy and F1 score, but slightly lower on ROC score and the area under the curve than another.

```
max_depth =  4
Accuracy:  0.7569721115537849
ROC AUC score:  0.7509735202492213
F1 Score:  0.7889273356401383
Training Accuracy:  0.7564870259481038
Testing Accuracy:  0.7569721115537849
Confusion Matrix
[[ 76  31]
 [ 30 114]]
Classification Report
              precision    recall  f1-score

           0       0.72      0.71      0.71
           1       0.79      0.79      0.79

    accuracy                           0.76
   macro avg       0.75      0.75      0.75
weighted avg       0.76      0.76      0.76
```

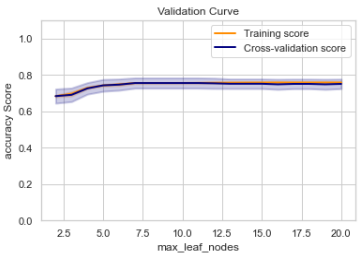
```
max_depth =  5
Accuracy:  0.7529880478087649
ROC AUC score:  0.7547053478712358
F1 Score:  0.7753623188405797
Training Accuracy:  0.7604790419161677
Testing Accuracy:  0.7529880478087649
Confusion Matrix
[[ 82  25]
 [ 37 107]]
Classification Report
              precision    recall  f1-score

           0       0.69      0.77      0.73
           1       0.81      0.74      0.78

    accuracy                           0.75
   macro avg       0.75      0.75      0.75
weighted avg       0.76      0.75      0.75
```

Considering max_depth = 5, the training accuracy is starting to get higher than testing accuracy which can be the sign of starting to overfit. So, **max_depth = 4** is a better tree pruning criteria for the highest accuracy at 0.757 and the ROC score at 0.751.





The average wall clock time for decision tree training and validation is 3.63 seconds. The longer time to fit due to 4 times higher number of features than the Car Evaluation dataset.



Considering the number of *max leaf nodes* when max_depth is 4, The validation curve shows that the accuracy of the decision tree is not significantly different or the hyperparameter directly causes overfitting. The change is quite flat after 5 max leaf nodes. The best **max_leaf_nodes is at 10**.

2.5 Neural Networks

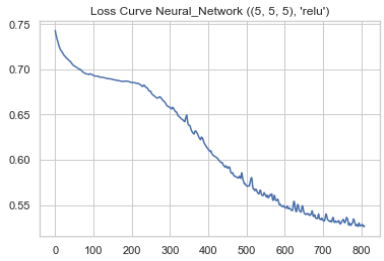
To see the effect of hidden layer sizes and the accuracy of binary classification from neural network models, 31 different network sizes are trained with *MLPClassifier with ReLU activation function* varied in different sizes of one, two, and three hidden layers. The learning rate is constant with a default value of 0.001. The maximum iteration is 1,500 with early stopping if there is no change within 100 iterations. The model training took between 140 - 800 iterations.

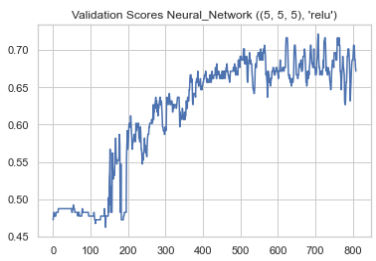
For the neural network with one hidden layer, with sizes ranging from 3 to 512, the accuracy and ROC scores are around 0.63 - 0.7. As the network becomes more complex, the increase in accuracy and ROC, but it could lead to overfitting. The best parameter for one hidden layer network is 256, with accuracy at 0.705 and ROC score at 0.705.

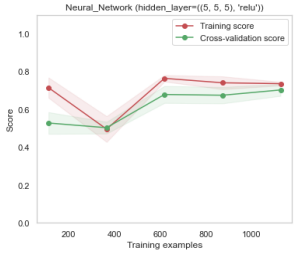
The two hidden layer neural networks with size from (3, 3) to (512, 512) performed similarly to the one hidden layers. The accuracy and the ROC scores are around 0.65 - 0.7. The best network size is (128, 128) with accuracy at 0.705 and ROC score at 0.702, slightly dropped from the one hidden layer (256).

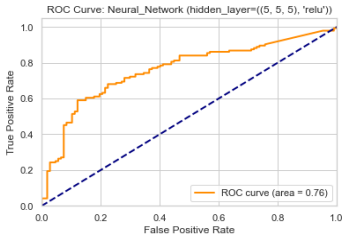
The results from the three hidden layer networks are interesting. **The highest accuracy and ROC score network is (5, 5, 5)** with accuracy = 0.713 and ROC score = 0.712 which is the best performing neural network and has close performance to the decision tree. The number of nodes in the hidden layer is not too complex either. For the simplest network of (2, 2, 2) can get accuracy and ROC score at 0.67 which is better than the starting accuracy of the previous number of hidden layers. The higher network size slightly loses the accuracy in cross validation due to being more likely to overfit with the training data from the complex structure.

The plots below are from neural network size (5, 5, 5) showing the loss curve and the validation scores (accuracy) during the model training, the validation score comparing between training and CV, and lastly, the ROC curve with AUC = 0.76.









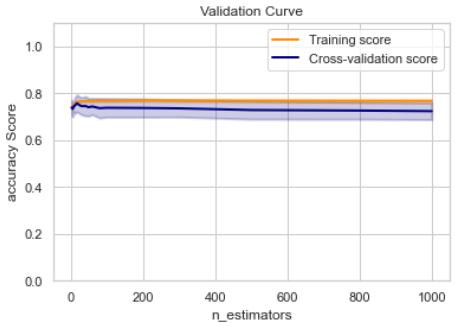
I have explored the different effects of learning rate options between *constant* (default at 0.001), *invscaling*, and *adaptive*. Three of them have the same output of accuracy, ROC score, and same plots. So the different learning rate does not affect the converging neural network in this case.

The wall clock time for training and validation for the network size (5, 5, 5) is 6.65 seconds. It took a shorter time to train due to the number of target classes that have only two classes (no, yes) for binary classification.

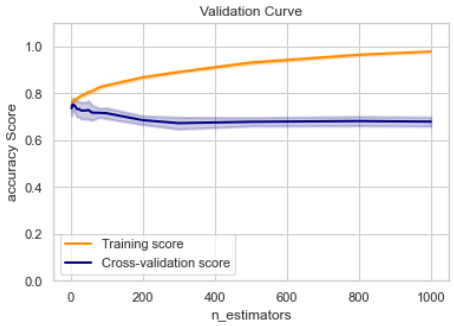
2.6 Boosting

Similar to the Car Evaluation task, Adaptive Boosting (AdaBoost) is used for classifying using a number of weak learner estimators with different pre-pruning settings of decision trees. The **number of estimators** range between 3 to 2,000 in 20 different values. The **max_depth** of decision trees are between 1 and 3.

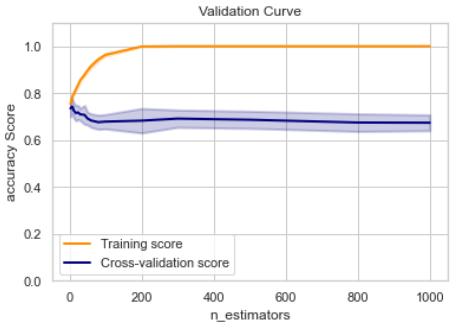
The validation curve below shows the difference in n_estimators for each max_depth. For max_depth = 2 and 3 can clearly see the pattern of overfitting where the training scores get higher while declining in CV scores. Especially, the max_depth =3 quickly reached the max score when the number of estimators was 200, much faster than max_depth=2. Therefore, the decision tree with **max_depth = 1 is the most suitable pre-pruning form**. Because of binary classification, the one level tree can cover both two target classes.



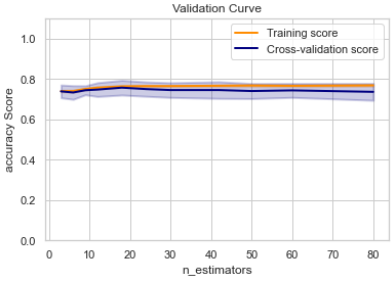
(max_depth = 1)



(max_depth = 2)



(max_depth = 3)



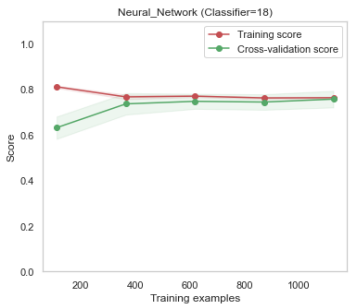
Taking a closer look at the validation curve of max_depth = 1, the best number of estimators is between 10 and 20. The accuracy and the ROC score is between 0.74 - 0.76 which is higher than the decision tree.

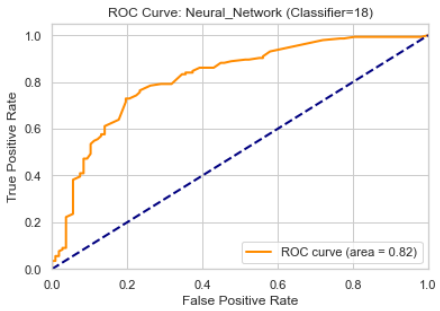
The best **number of estimators** that have the highest accuracy is **18** with accuracy = 0.757, ROC score = 0.762, and the ROC AUC = 0.82.

The average wall clock time for training and validation is 0.94 seconds.

```
AdaBoost: n_estimators =  18
Accuracy:  0.7569721115537849
ROC AUC score:  0.7617795950155762
F1 Score:  0.7749077490774907
Training Accuracy:  0.7574850299401198
Testing Accuracy:  0.7569721115537849
Confusion Matrix
[[ 85  22]
 [ 39 105]]
Classification Report
              precision    recall  f1-score   support

           0       0.69      0.79      0.74       107
           1       0.83      0.73      0.77       144
```

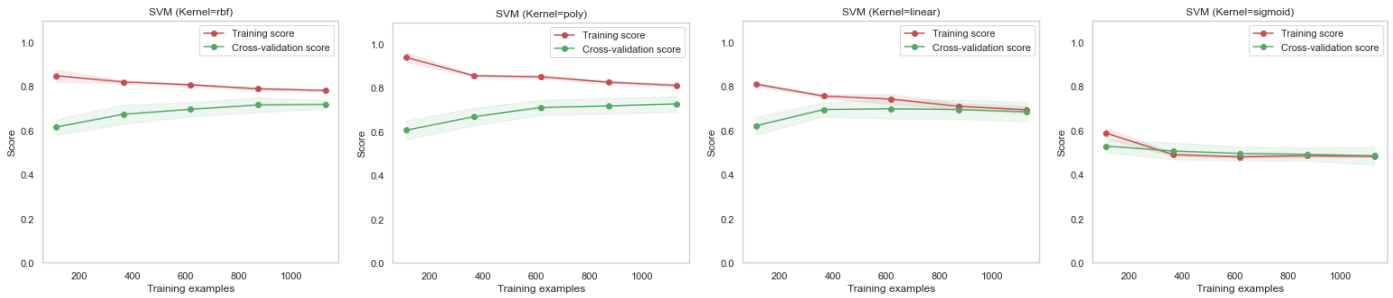
The plots below show the training curve and ROC curve of AdaBoost with n_estimatos = 18 and the weak learner decision trees have max_depth = 1.



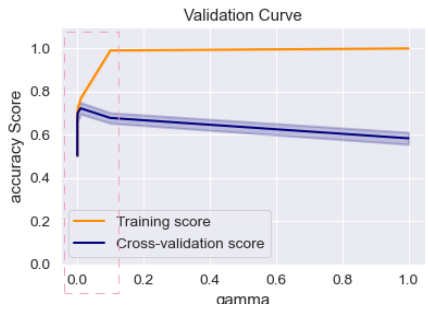


2.7 Support Vector Machines

The SVM models are trained with different Kernels; *linear*, *polynomial*, *sigmoid* and *rbf*. The best performing kernels sorted by accuracy and the ROC score are rbf, poly, linear, and sigmoid respectively. The sigmoid kernel is not suitable for this task because the hyperplane of data cannot be separable with sigmoid function, unlike the other kernel.



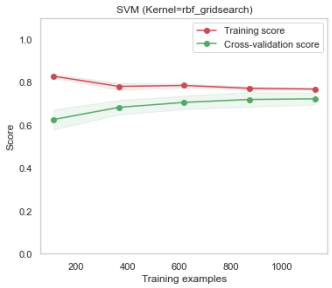
The accuracy of the **rbf kernel**, which is the highest of all four kernels, is 0.713, the ROC score is 0.721, and the ROC AUC is 0.78. I used *GridSearchCV* to further search on the hyperparameter tuning and run a validation curve on it. The best hyperparameters for the rbf SVM kernel are **C = 10** for regularization parameter, and **Gamma coefficient = 0.01**.

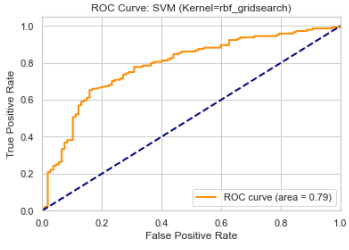


↓ (zoom in to 0.01 - 0.1)

To explore more on the effect of gamma in the Validation Curve; The best gamma value is 0.01 as shown in the accuracy metrics, training and ROC curves below. After the higher gamma values may lead to overfitting.

```
Accuracy:  0.7250996015936255
ROC AUC score:  0.7340018172377986
F1 Score:  0.7376425855513309
Training Accuracy:  0.7764471057884231
Testing Accuracy:  0.7250996015936255
Confusion Matrix
[[85 22]
 [47 97]]
Classification Report
              precision    recall  f1-score   support

           0       0.64      0.79      0.71       107
           1       0.82      0.67      0.74       144

    accuracy                           0.73       251
   macro avg       0.73      0.73      0.72       251
weighted avg       0.74      0.73      0.73       251

Cross Validation Score
[0.75428427 0.80090726 0.79314689 0.83717358 0.81157194
 0.7703533  0.8218126  0.73451101 0.80107527]
Best Params :  {'C': 10, 'gamma': 0.01}
```

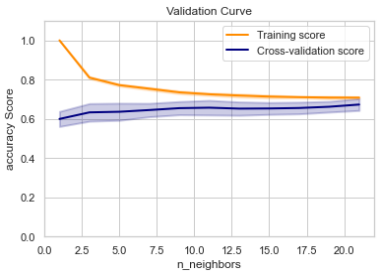




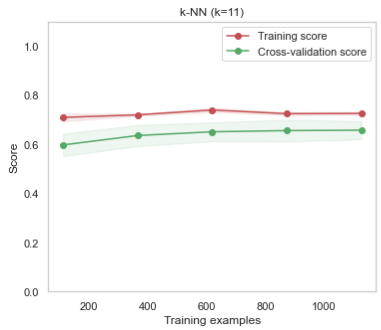
The plot on the left shows that 0.01 is the best value to get both training score and cross-validation score. Before the cross-validation score continues dropping.

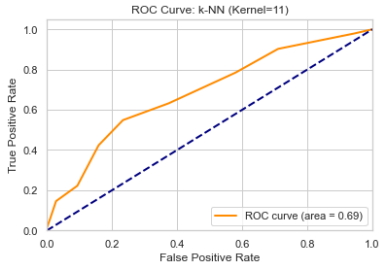
The average wall clock time for SVM model training and validation is 2.5 seconds.

2.8 K-Nearest Neighbors



In the k-NN algorithm, the different values of *k* (*n_neighbors*) are used to estimate the output from the *k* nearby data points. From the Validation Curve that shows the accuracy from the odd number of k from 1 to 19, the accuracy of the lower number of K is not performed well on cross-validation due to high variance from the closest data points. According to the validation curve, it suggests that the proper value of k should be around 9 to 13. Having a high number of K can be too generalized and result in underfitting as "a high value of k leads to high bias and low variance" [4].





The best accuracy and ROC score is from **k = 11**. The accuracy metrics and training and ROC curves as shown below. However, the ROC AUC is the lowest of all algorithms and the accuracy is too low to be useful.

```
K =  11
Accuracy:  0.6334661354581673
ROC AUC score:  0.6493380062305295
F1 Score:  0.6290322580645162
Training Accuracy:  0.719560878243513
Testing Accuracy:  0.6334661354581673
Confusion Matrix
[[81 26]
 [66 78]]
Classification Report
              precision    recall  f1-score   support

           0       0.55      0.76      0.64       107
           1       0.75      0.54      0.63       144

    accuracy                           0.63       251
   macro avg       0.65      0.65      0.63       251
weighted avg       0.67      0.63      0.63       251
```

The average wall clock time in prediction and evaluation of k-NN is pretty fast, it is slightly lower than a second (0.94 s).

2.9 Conclusion

Mental Health	Decision Tree	Neural Network	AdaBoosting	SVM	k-NN
Main Params	Max_depth = 4 Max_leaf_nodes = 10	Hidden layer size (5,5,5)	**N_estimators = 18** **Max_depth = 1**	RBF kernel C = 10, gamma = 0.01	k = 11
Accuracy	0.75697	0.71314	**0.75698**	0.72510	0.63347
ROC score	0.75097	0.71158	**0.76178**	0.73400	0.64933
ROC AUC	0.77	0.76	**0.82**	0.79	0.69
Wall clock time	~3.63 seconds	~6.65 seconds	**~0.94 seconds**	~ 2.5 seconds	**~0.94 seconds**

In Mental Health Survey binary classification, the AdaBoost is the best performing algorithm, followed by the decision tree which has the similar accuracy and ROC score, but slightly lower in ROC AUC. In this case, it can show that the **weak learners with max_depth of one are working well in binary classification** tasks and can beat complex decision trees. Using the AdaBoosting in Mental Health Survey classification seems to be the most effective in terms of accuracy, time, and computing power with multiple weak learners. The tree based model is also good for model understanding.

The SVM and Neural Network model performances are similar in from the above table. SVM has a slightly higher performance in all accuracy, ROC score, and the ROC AUC. It is also faster to train than a neural network model.

Therefore, SVM is more preferable than Neural Network because of the explainability with functions rather than a black box model.

The k-NN is the worst performing of all algorithms in terms of accuracy and the ROC score. In terms of speed, k-NN can achieve the same wall clock time with AdaBoosting with the current amount of data. But with low accuracy, the algorithm is not suitable for use in clinical tests.

---

3 REFERENCES

1. Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Available at: <https://archive.ics.uci.edu/ml/datasets/car+evaluation> [Accessed 9 September 2020].
2. Zupan, B., n.d. Car Dataset - Function Decomposition Page At AI Lab, FRI. [online] File.biolab.si. Available at: <https://file.biolab.si/biolab/app/hint/car_dataset.html> [Accessed 12 September 2020].
3. Kaggle.com. 2016. *Mental Health In Tech Survey*. [online] Available at: <https://www.kaggle.com/osmi/mental-health-in-tech-survey> [Accessed 31 August 2020].
4. En.wikipedia.org. 2020. *Bias–Variance Tradeoff*. [online] Available at: <https://en.wikipedia.org/wiki/Bias–variance_tradeoff> [Accessed 11 September 2020].
5. Scikit-learn.org. *"Sklearn.neural_network.MLPClassifier."* [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html> [Accessed September 18 2022].
6. Kamolphan Liwprasert (kliwprasert3). Fall 2020. Georgia Tech's CS7641: Supervised Learning Analysis.