CS-7641 Assignment 1 Report – Supervised Learning

Puneeth Kumar Chana Reddy
preddy61@gatech.edu

***Abstract*—In this report, we will be exploring five supervised learning algorithms namely Decision Trees, Neural Networks, Boosted Trees, Support Vector Machines, and k-nearest neighbors on two classification data sets to analyse their performance. Furthermore, every algorithm will be tuned to provide optimal performance on each of the data sets to compare and discuss their effectiveness and shortcomings.**

I. Stroke Prediction Dataset

According to the World Health Organization (WHO) stroke is the second leading cause of death globally, responsible for approximately 11% of total deaths. As they say "prevention is better than cure", it makes an important case of identifying people with risks to prevent them from developing the stroke. This data set is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status.

A. Data set description

This is binary classification data set which contains 4908 rows and 11 columns. About 95% of the the data are "no stroke" (Class 0) and the remaining are "stroke" (Class 1) which makes this data set a highly imbalanced one. Furthermore, the feature space consists of 5 continuous, 4 categorical variables.

B. Data set preprocessing

- The missing data in the "bmi" column were removed. They account for about 4% for the entire dataset.
- The categorical column "smoking_status" contains "never_smoked" and "unknown" that were converted to "never_smoked" based on the other features of the instances.
- The categorical columns in the feature space are One Hot Encoded for the algorithms to make fair predictions without introducing any bias.
- The numerical columns like "bmi" and "avg_glucose_level" are on different scales, hence all the numerical feature are scaled using "MinMaxScaler" to bring them on to the same scale before feeding them to algorithms

C. Metric

As this is a highly imbalanced data set, we will be using PR-AUC score for comparing the algorithms, especially focusing on the recall value of class 1 ('stroke'). This will show us the extent to which a particular algorithm is able to classify the "stroke" label correctly from the test data set. The idea is to tune the algorithms so as to increase the recall value of class 1 (stroke) but keeping the precision as high as possible .

II. FIFA 19 complete player Dataset

Data set contains detailed attributes for every player registered in the latest edition of FIFA 19 database. We will be using the features of the data set to classify players into multiple classes

A. Data set description

This is multi class classification data set which contains 18207 entries with 89 columns. Several features are combined based on their positions into four classes namely "Midfielder", "Attacker", "Defender" and "Goal Keeper". This transformation reduces the feature space to 29 columns and the predictable classes are more or less evenly distributed in the entire dataset. Furthermore, the feature spaces consists of 29 continuous variables.

B. Data set preprocessing

- The missing data in all the column were removed, as they accounted for just 0.3% of the data set.
- All the numerical features are scaled using "StandardScaler" to bring them on to the same scale before feeding them to algorithms

C. Metric

As this is a balanced data set, we will be using F1_macro score to compare the algorithms. F1-score, or the macro-F1 for short is an arithmetic mean of the per-class F1-scores.

This data set brings to light four significant difference compared to the Stroke Prediction Data set.

- This is a balanced data set compared to Stroke Prediction Data set which was highly imbalanced
- This is a multiclass classification where as Stroke Prediction Data set was a binary one
- This data set contains 89 features and 18000 instances compare to Stroke Prediction Data set that has 10 features and 5000 instances
- As this is balanced data set, all the classes are treated equally but in the Stroke Prediction Data set, the 'stroke' class is more important than the 'no stroke' class. Failing

to correctly classify 'stroke' can lead to life threating scenarios.

Both the data sets are then split into 70% training/validation set and the remaining 30% is kept aside as a testing set for checking the performance of the final model. To make better use of data, a 5 fold stratified cross validation is performed. The stratification and shuffling is necessary to keep the ratio of the predictor labels same across training and testing set. Furthermore, we use sklearn's pipeline mechanism to avoid the data leakage into the testing set. Data is first split and then the training data is passed through the pipeline containing pre-processing steps and then to the actual classifier for the training. Similarly, the test data follows the same pre-processing pipeline but predict the outcomes.

III. Stroke Prediction Data set

A. Decision Trees Classifier

The analysis started with using the vanilla implementation of decision trees from sklearn with *class_weights='balanced'* to penalize the algorithm for mis-classifying the "stroke" class. Setting *ccp_alpha=0.0* the PR-AUC score was 11.5% and F1-Score for "stroke" class was 10%. The learning curve depicted a very high variance and low bias, as it was evident from a high training score and low cross validation score.

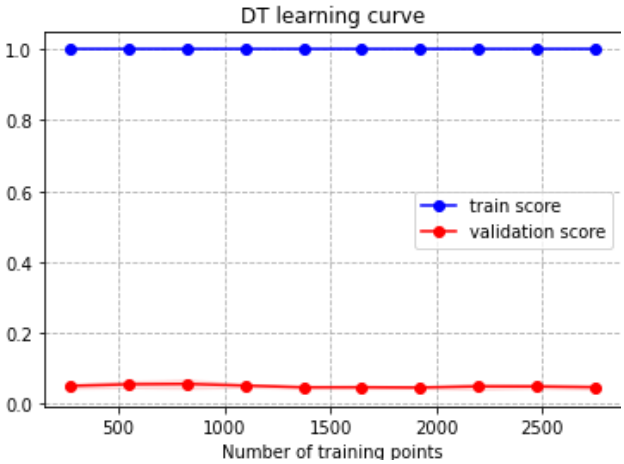


Fig. 1: Learning curve of the decision tree before tuning

From the figure, it shows that the classifier had over fit the data. Probing max_depth attribute showed a high value of 21 for fewer features in the data. Additionally, only 3 out 10 features appeared to be of importance, further confirming the above over fitting effect. Therefore, the tree had to be pruned to reduce the complexity to reduce over fitting. So, I plotted a validation curve against couple of hyper parameters namely *ccp_alpha* and *max_depth* to see their affect on the performance.

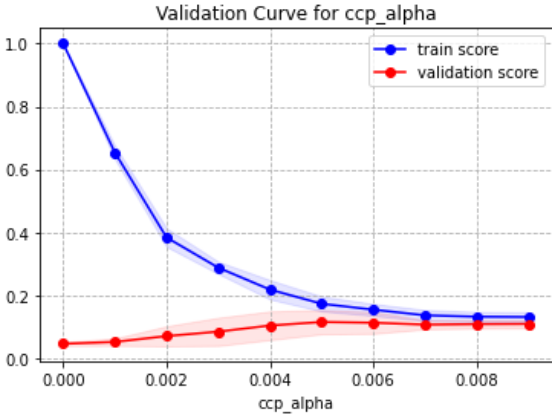


Fig. 2: Validation curve of the decision tree against *ccp_alpha*

Analyzing train/validation scores show that low *ccp_alpha* values tends to over fit and higher values tend to under fit, But the values between 0.002-0.006 increased the validation score.

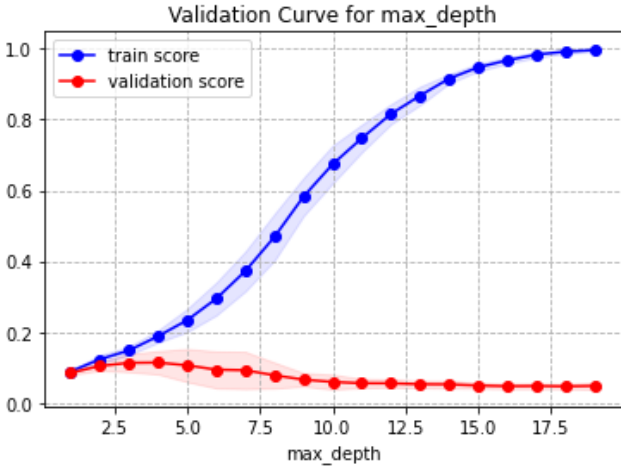


Fig. 3: Validation curve of the decision tree against *max_depth*

Analyzing train/validation scores show that low *max_depth* tends to under fit and high value tends to over fit.

A grid search was then performed which gave me ***max_depth=3***, ***ccp_alpha=.004*** and ***criterion='entropy'***. Plotting a learning curve after training the model with best parameters reduced the variance. Also *max_depth=3* shows that there are a handful of features that are of importance to classify the labels.

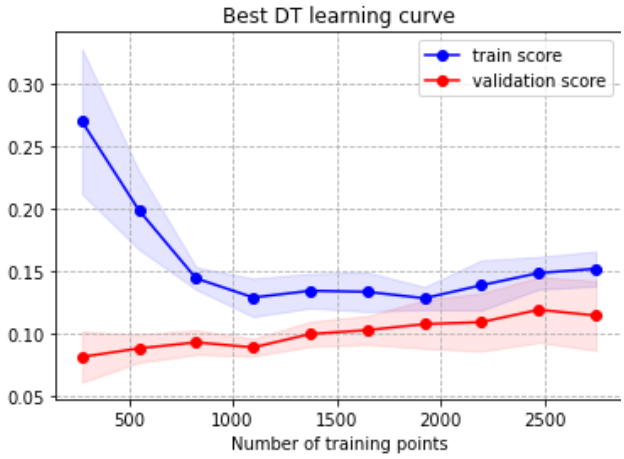


Fig. 4: Validation curve of the decision tree post tuning

After tuning the classifier, I tested it on the test data and found that PR-AUC score increased to 31.5% and F1-Score for ”stroke” class was increased to 21%. Also, the recall for stroke class increased from 10% to 81%. This means the tuned classifier was able generalize better to detect 81% of ”stroke” samples from the test data.

	precision	recall	f1-score	support
0	0.99	0.74	0.85	1410
1	0.12	0.81	0.21	63
accuracy			0.75	1473
macro avg	0.56	0.78	0.53	1473
weighted avg	0.95	0.75	0.82	1473

B. Boosting - Ada Boost Classifier

The analysis started with using the vanilla implementation of Ada Boost classifier with a decision tree *max_depth=1* and *class_weights='balanced'* as a weak learner from sklearn. The learning curve depicted a high bias, as it was evident from a low training score and low cross validation score, as the base estimator was under fitting the data. The PR-AUC score was 17% and F1-Score for ”stroke” class was 21%. Also, the recall score for stroke class increased to 81% in comparison to 10% of the vanilla implementation of the decision tree classifier above, which confirmed that ensemble methods worked better than single decision tree for binary classification.

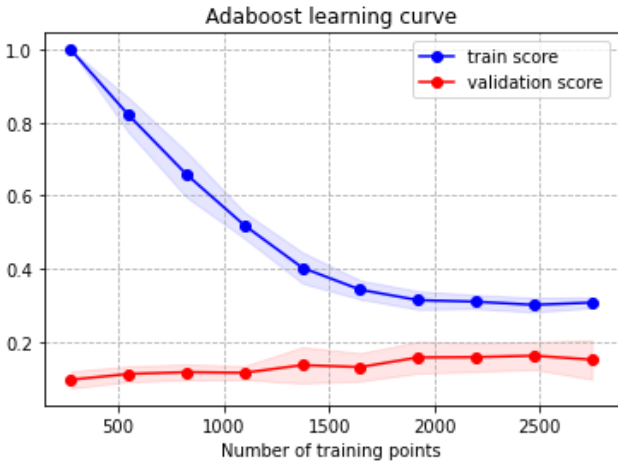


Fig. 5: Validation curve of the Ada Boost before tuning

Complexity of the model had to be increased to counter the high bias without increasing the complexity of the base learner.
I plotted a validation curve against couple of hyper parameters namely *n_estimators* and *learning_rate* to see their affect on the performance.

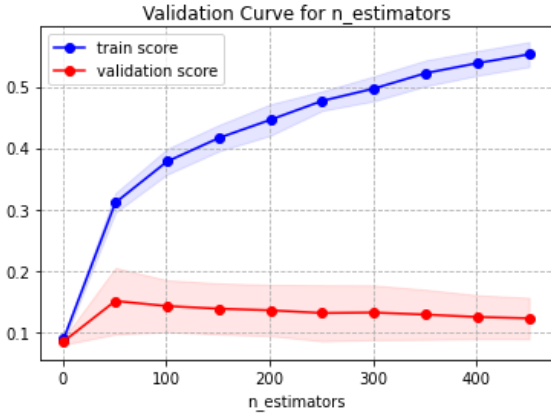


Fig. 6: Validation curve of Ada boost against *n_estimators*

Analyzing train/validation scores show that low *n_estimators* tends to under fit for low values. The training and validation score seems to deviate above 100 whereas, around 50-80 the validation score increased.

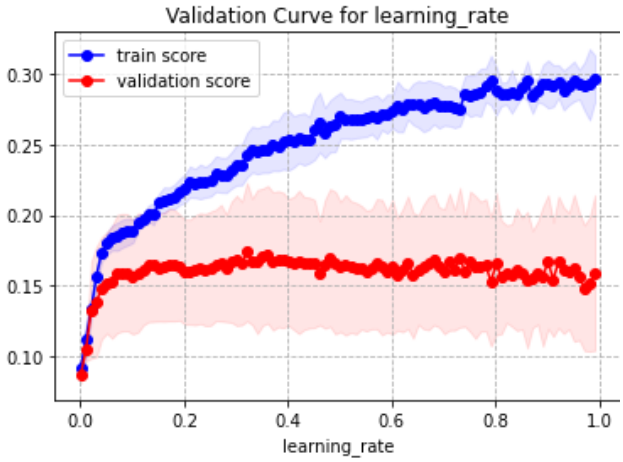


Fig. 7: Validation curve of Ada boost against *learning_rate*

Analyzing train/validation scores show that higher *learning_rates* decreased the validation score. If you have a lower learning rate then you could have more estimators since each individual estimator is contributing less.
To make trade off between the parameters, a grid search was performed which gave me ***n_estimators=91*** and ***learning_rate=.1809***
Plotting a learning curve after training the model with best parameters is shown below.

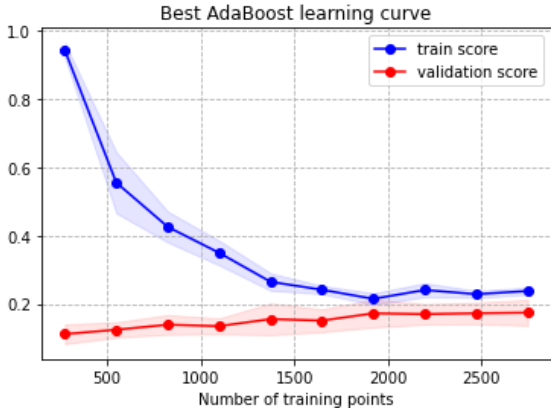


Fig. 8: Learning curve of Ada Boost after tuning

After tuning the classifier, I tested it on the test data and found that PR-AUC score increased to 18.5% and F1-Score for ”stroke” class was increased to 20%. Also, the recall for stroke class increased from 60% to 84%. This means the tuned classifier was able to detect 84% of ”stroke” samples from the test data.

	precision	recall	f1-score	support
0	0.99	0.70	0.82	1410
1	0.11	0.84	0.20	63
accuracy			0.71	1473
macro avg	0.55	0.77	0.51	1473
weighted avg	0.95	0.71	0.79	1473

C. k-Nearest Neighbors

The analysis started with using the vanilla implementation of k-nearest neighbours classifier. There is no *class_weights* parameter to penalize the algorithm for mis-classifying the ”stroke” class. The learning curve depicted a high variance. The PR-AUC score was around 10% and F1-Score for ”stroke” class was 5%. Also, the recall score for stroke class was just 3%.
These scores were way too less in comparison to decision tree classifier and boosting algorithm. Furthermore, this effect could also be attributed to the imbalances in the data, as the classifier couldn't account for mis-classfication errors of the minor class.

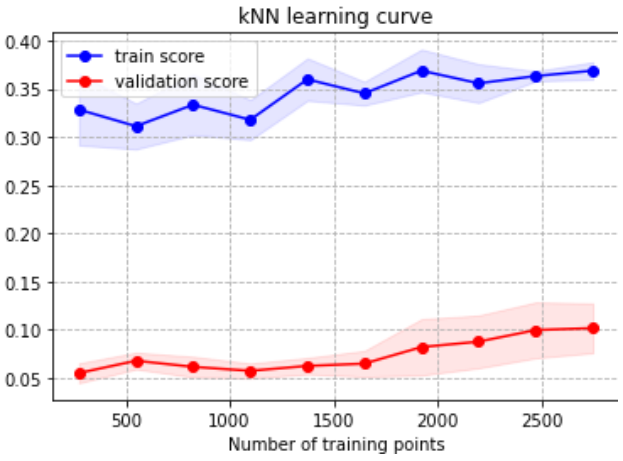


Fig. 9: Learning curve of k-NN before tuning

Complexity of the model had to be decreased to counter the high variance to see if it did any better.
I plotted a validation curve against hyper parameter namely *k* to see its effect on the performance.

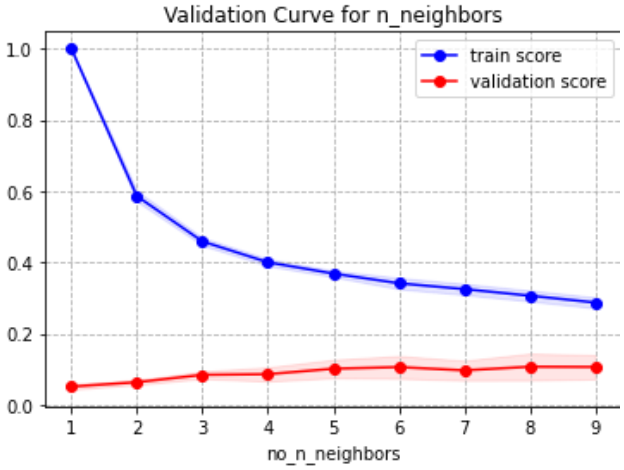


Fig. 10: Validation curve of k-NN against *k*

Analyzing train/validation scores shows that changing *k* tends to over fit for values of k<3 as shown above.
As the value of K increased the mis-classification of the ’stroke’ samples increased. This can be attributed to the fact that, a higher proportion of ’no stroke’ samples around the ’stroke’ sample causes the algorithm to mis-classify the ’stroke’ class.
Performing a grid search then gave me ***k=4*** and ***weights='uniform'***. Plotting a learning curve after training the model with best parameters is shown below.

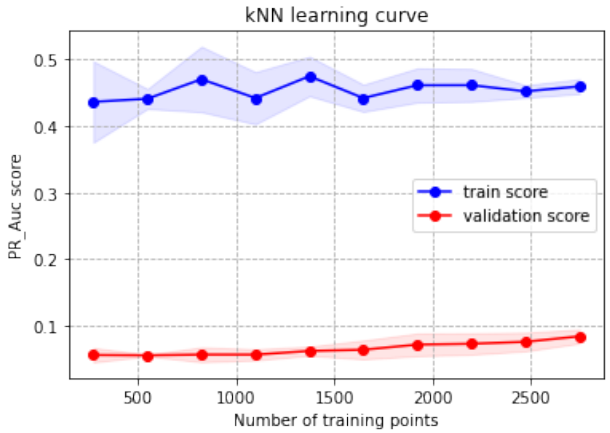


Fig. 11: Learning curve of k-NN after tuning

After tuning the classifier, I tested it on the test data and found that PR-AUC score slightly increased to 12.5% and F1-Score for ”stroke” class was increased to 11%. Also, the recall for stroke class increased from 3% to 11%. Furthermore, the tuned k-NN classifier was not able to penalize for mis-classifcation of ’stroke’ class causing such a poor performance.

	precision	recall	f1-score	support
0	0.96	0.96	0.96	1410
1	0.10	0.11	0.11	63
accuracy			0.92	1473
macro avg	0.53	0.53	0.53	1473
weighted avg	0.92	0.92	0.92	1473

To overcome the problem of imbalanced data, I performed SMOTE on the train data to increase the instances of the

"Stroke" class to see the effect on the algorithm. After tuning, I was able to achieve high recall value of 79% for the minority class but the overall F1-score still remained at 12%

	precision	recall	f1-score	support
0	0.98	0.54	0.70	1410
1	0.07	0.73	0.12	63
accuracy			0.55	1473
macro avg	0.52	0.64	0.41	1473
weighted avg	0.94	0.55	0.68	1473

D. Support Vector Machines

The analysis started with using the vanilla implementation of SVC classifier with *kernel='rbf'* and *class_weights='balanced'*. The learning curve depicted a very high bias, as it was evident from a low training score and low cross validation score. The PR-AUC score was 15.8% and F1-Score for "stroke" class was 18%. Also, the recall score for stroke class was 73% and is one of the highest among other classifiers.

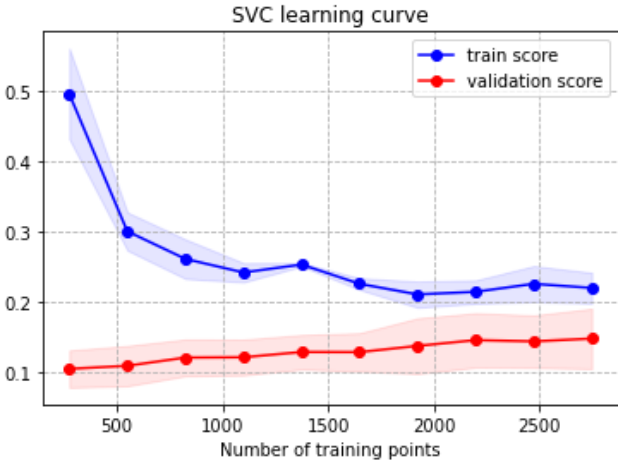


Fig. 12: Learning curve of SVC before tuning

- **kernel='rbf'**
  Complexity of the model had to be increased to counter the high bias.
  I plotted a validation curve against *gamma* hyper parameter to see its affect on the performance.

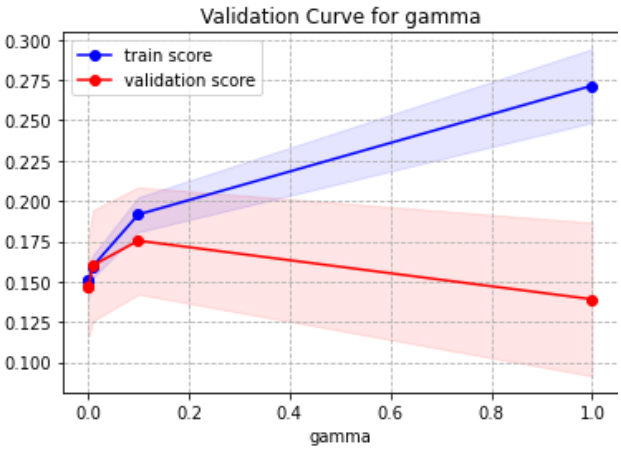


Fig. 13: Validation curve of SVC against *gamma*

High values of gamma tends to over fit the model. As gamma gets larger, the effect of c becomes negligible. Therefore, both c and gamma parameters need to be optimized simultaneously.
Performing a grid search on these parameter gave me ***gamma=0.1*** and ***C=2.1***. Plotting a learning curve after training the model with best parameters is shown below

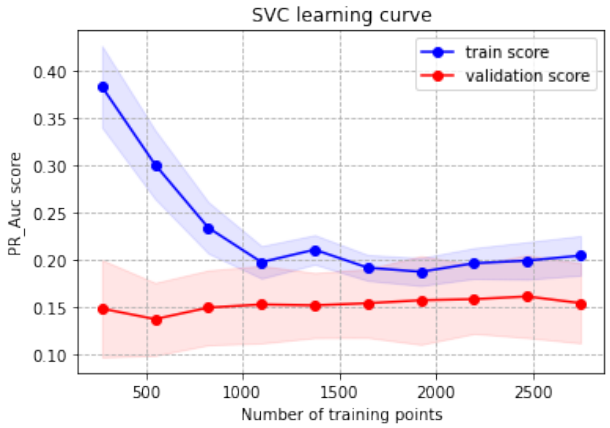


Fig. 14: Learning curve of SVC after tuning
After tuning the classifier, I tested it on the test data and found that PR-AUC score increased to 18% and F1-Score for "stroke" class was increased to 20%. Also, the recall for stroke class increased from 73% to 79%.

	precision	recall	f1-score	support
0	0.99	0.72	0.83	1410
1	0.11	0.79	0.20	63
accuracy			0.72	1473
macro avg	0.55	0.76	0.51	1473
weighted avg	0.95	0.72	0.80	1473

- **kernel='poly'** To see the effect of a different kernel on the data, I switched classifier's kernel to 'poly' and *degree* of 3 which did not perform well.

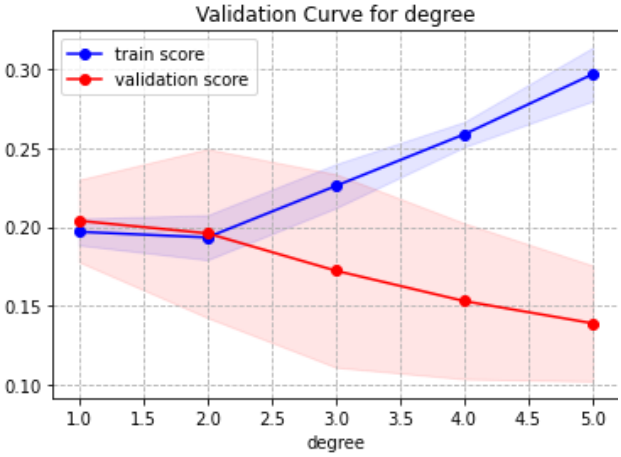


Fig. 15: Validation curve of SVC poly against *degree*
As the *degree* parameter increased, the classifier performed poorly on the test data but performed well on the training data. This points to the fact that, classifier tries to fit all the outliers reducing the margin of the

support vectors, thereby over fitting the model.

I therefore performed a grid search which gave me ***degree=1*** which would behave like a linear kernel. The F1-Score for "stroke" class was around 20% which is almost equal to the 'rbf' kernel

	precision	recall	f1-score	support
0	0.99	0.72	0.83	1410
1	0.12	0.84	0.21	63
accuracy			0.73	1473
macro avg	0.55	0.78	0.52	1473
weighted avg	0.95	0.73	0.81	1473

E. Neural Networks - MLP

The analysis started with using the vanilla implementation of MLP classifier with *max_iter=300* from sklearn. The learning curve depicted a very high bias, as it was evident from a low training score and low cross validation score. The PR-AUC score was 14.7% and F1-Score for "stroke" class was very low at 6%.

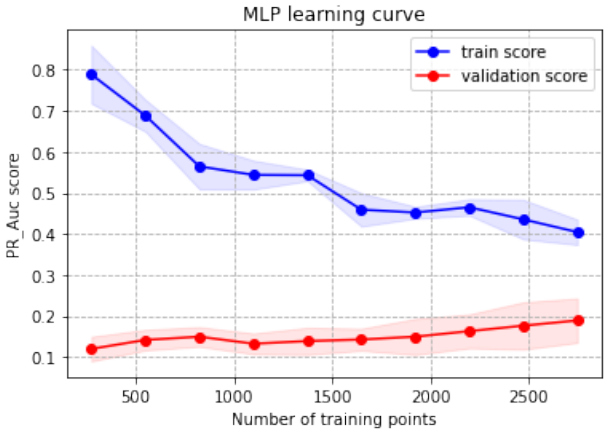


Fig. 16: Learning curve of MLP before tuning

Plotting the loss curve shows that the loss decreased initially but as the number of iterations increased, loss decreased gradually.

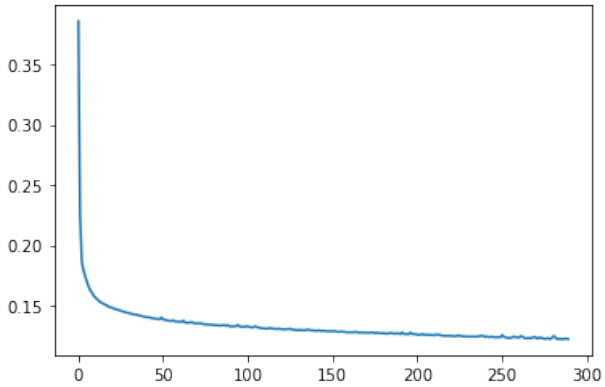


Fig. 17: Loss curve of MLP before tuning

Complexity of the model had to be increased to counter the high bias. Increasing the hidden layers or increasing the units per hidden layer further increased the non linearity of the network but did not have much effect. So, regularization parameter *alpha* was tuned to reduce bias.
I plotted a validation curve against *alpha* to see its affect on the performance.

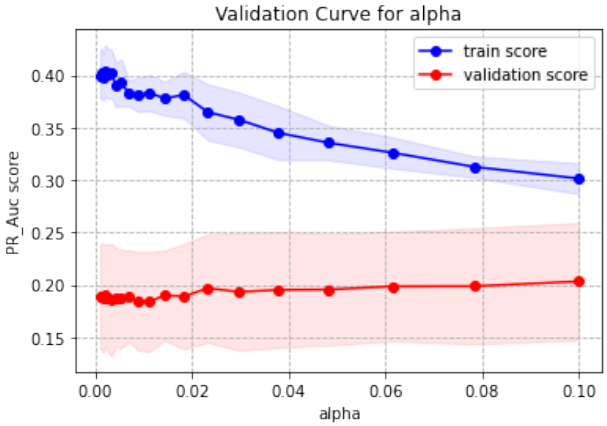


Fig. 18: Validation curve of MLP against *alpha*

Analyzing train/validation scores show that low *alpha* tends to over fit for low values and as the number increases the classifier seem to under fit.

A grid search of various parameters then gave me *alpha=0.00001* and *learning_rate=constant* and *hidden_layer_size=(50,)*. So, reducing the *alpha* and halving the units in the hidden layer improved the complexity of the model to some extent but could not perform well overall. In this scenario, having more data here would be helpful for the MLP to learn.
Plotting a learning curve after training the model with best parameters is shown below.

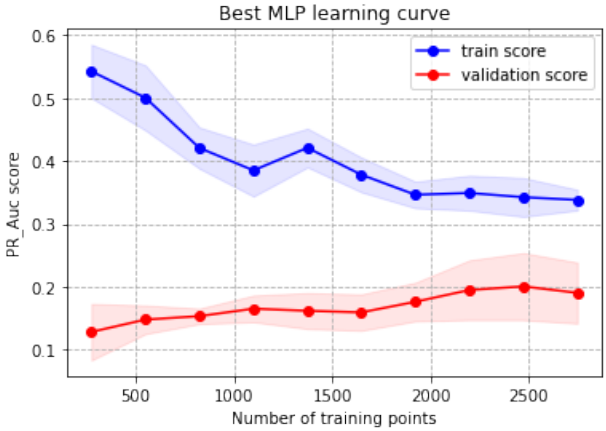


Fig. 19: Learning curve of MLP after tuning

Also, the loss reduced to values lower than pre-tuned version as shown below

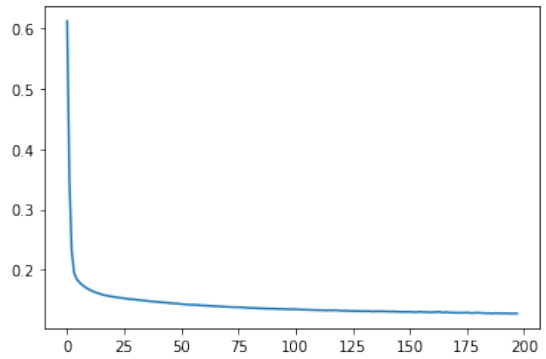


Fig. 20: Loss curve of MLP after tuning

After tuning the classifier, I tested it on the test data and found that the PR-AUC score increased to 16.9% but F1-Score for "stroke" class remained at 6%.

MLP: PR-Auc= 0.169

	precision	recall	f1-score	support
0	0.96	1.00	0.98	1410
1	0.50	0.03	0.06	63
accuracy			0.96	1473
macro avg	0.73	0.52	0.52	1473
weighted avg	0.94	0.96	0.94	1473

To overcome the problem of imbalanced data, I performed SMOTE on the train data to increase the instances of the "Stroke" class to see the effect on the algorithm. After tuning, I was able to achieve good recall value of 43% for the minority class but the overall F1-score still remained at 16%. But still it was way lower than other classifiers except k-NN.

	precision	recall	f1-score	support
0	0.97	0.82	0.89	1410
1	0.10	0.43	0.16	63
accuracy			0.80	1473
macro avg	0.53	0.62	0.52	1473
weighted avg	0.93	0.80	0.86	1473

IV. FIFA 19 COMPLETE PLAYER DATASET

A. Decision Trees Classifier

The multi class classification analysis started with using the vanilla implementation of decision trees from sklearn. Setting *ccp_alpha=0.0*, the F1-score average for all classes stood at 83.7%. The learning curve depicted a very high variance and low bias, as it was evident from a high training score and low cross validation score.

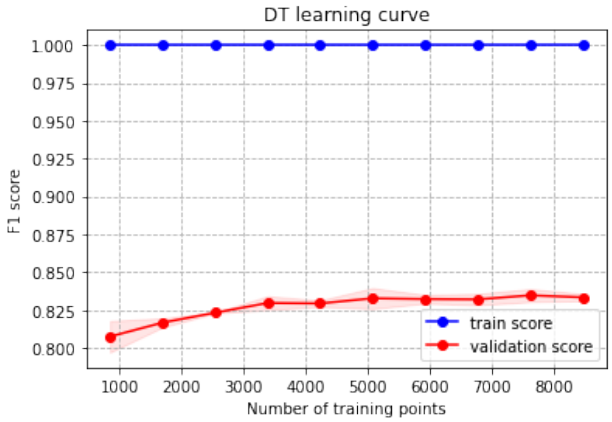


Fig. 21: Learning curve of the decision tree before tuning

From the figure, it shows that the classifier had over fit the data. Probing max_depth attribute showed a high value of 22. I plotted correlation map of features and found that few of them were positively and negatively correlated to each other e.g short-passing vs ball-control and strength vs acceleration and so on. Therefore, the tree had to be pruned to reduce the complexity to reduce over fitting. So, I plotted a validation curve against couple of hyper parameters namely *ccp_alpha* and *max_depth* to see their affect on the performance.

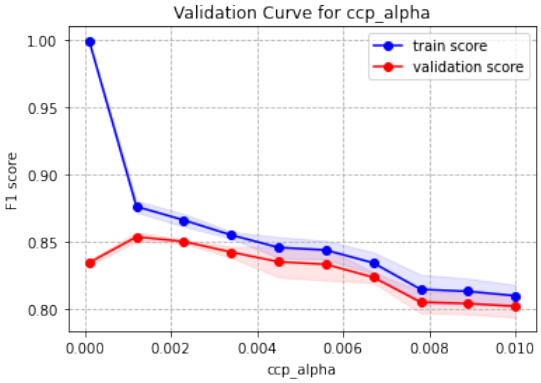


Fig. 22: Validation curve of the decision tree against *ccp_alpha*

Analyzing train/validation scores show that low *ccp_alpha* tends to over fit and high value tends to under fit, But the values between 0.000-0.002 shows the increase in validation score.

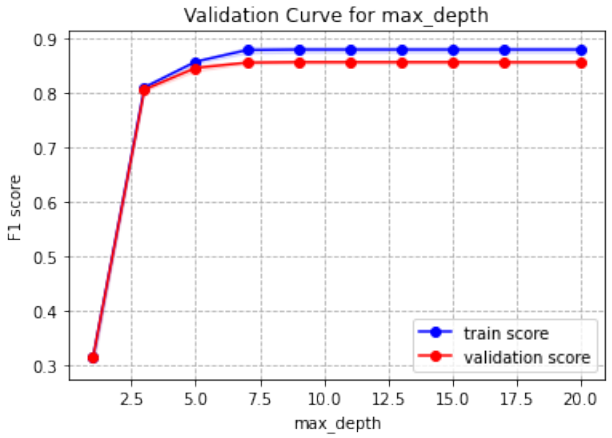


Fig. 23: Validation curve of the decision tree against *max_depth*

Analyzing train/validation scores show that low *max_depth* tends to under fit and high value tends to over fit.
A grid search was then performed which gave me *max_depth=9*, *ccp_alpha=.000612* and *criterion='gini'*. Plotting a learning curve after training the model with best parameters reduced the variance. Also *max_depth=9* shows that there are highly correlated features that did not bring much information gain.

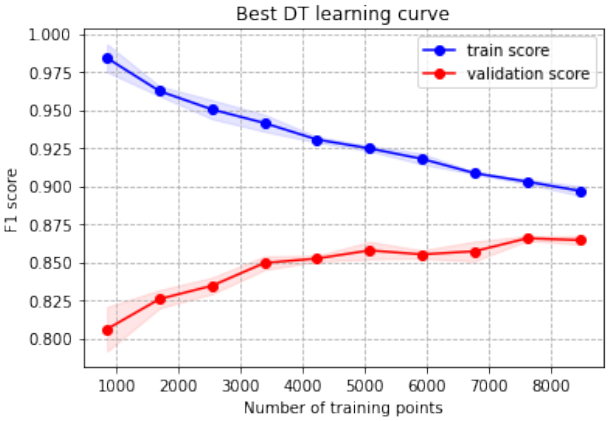


Fig. 24: Learning curve of the decision tree after tuning

After tuning the classifier, I tested it on the test data and found that F1 score increased to 86.1% and F1-Score for all classes increased by 2-3% compared to non pruned tree.

DT: F1 score= 0.861

	precision	recall	f1-score	support
Attacker	0.79	0.75	0.77	1025
Defender	0.86	0.91	0.88	1760
Goalkeeper	1.00	1.00	1.00	608
Midfielder	0.80	0.78	0.79	2052
accuracy			0.84	5445
macro avg	0.86	0.86	0.86	5445
weighted avg	0.84	0.84	0.84	5445

B. Boosting - Ada Boost Classifier

The analysis started with using the vanilla implementation of Ada Boost classifier with a decision tree *max_depth=2* as a weak learner from sklearn. The learning curve depicted a high bias, as it was evident from a low training score and low cross validation score as the number of samples increased, as the base estimator was under fitting the data. The F1 score macro was 63% which is less than the vanilla implementation of the decision tree classifier above. I tried to adjust *n_estimators* and *learning_rate* but the classifier did not show any improvement.

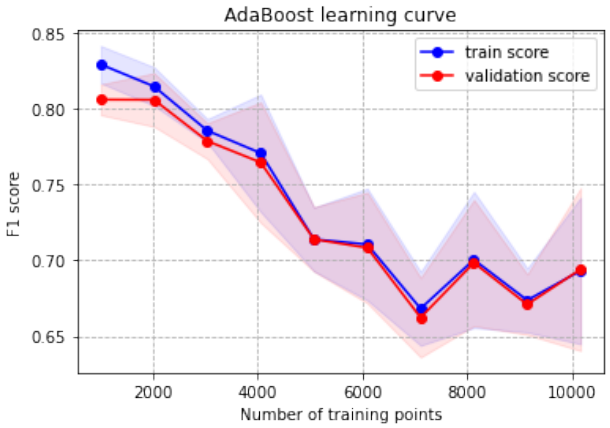


Fig. 25: Validation curve of the Ada Boost before tuning

Complexity of the model had to be increased to counter the high bias by increasing the complexity of the base learner.
I plotted a validation curve against hyper parameter *ccp_alpha* of the base estimator to see the effect. From the below figure it shows that pruning too much made the classifier not learning anything. I decide to use *ccp_alpha=0.005* for further analysis.

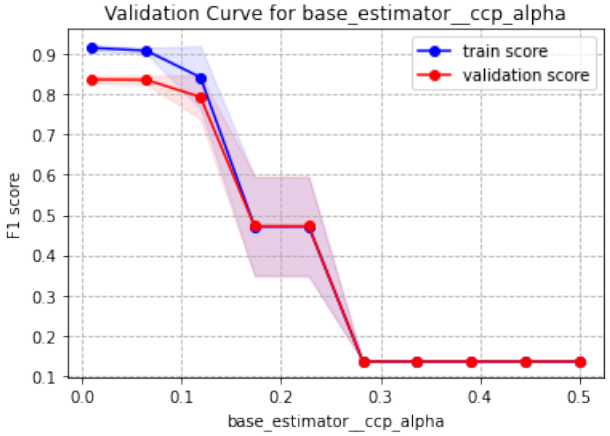


Fig. 26: Validation curve of Ada boost against *ccp_alpha*

Furthermore, as the base learner was strong, I plotted the validation curve against *learning_rate* of the classifier to see the effect.

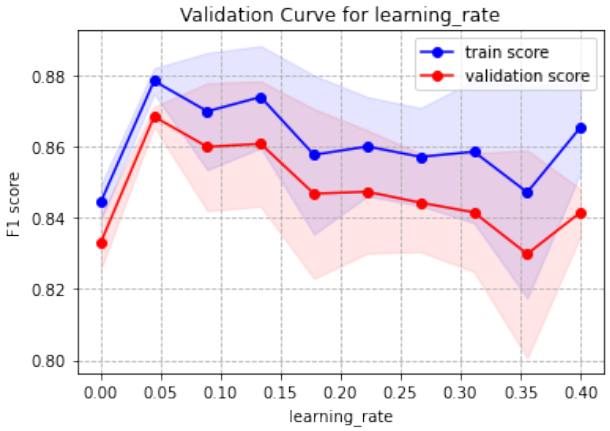


Fig. 27: Validation curve of Ada boost against *learning_rate*

Analyzing train/validation scores show that higher *learning_rate* decreased the validation score. To make trade

off between the other parameters like *n_estimators* and *learning_rate* for a chosen *ccp_alpha=0.005*, a grid search was performed which gave me *n_estimators=21* and *learning_rate=.14*
Plotting a learning curve after training the model with best parameters is shown below.

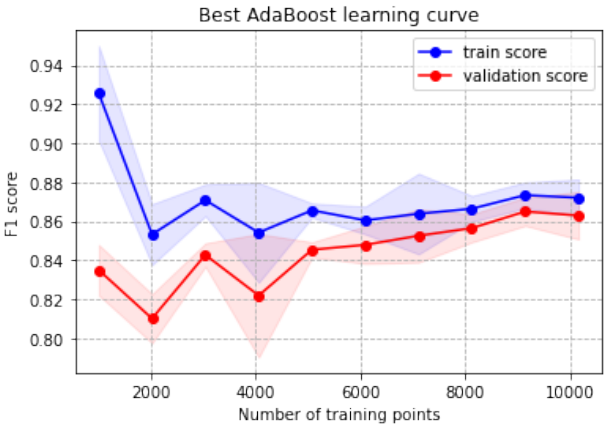


Fig. 28: Learning curve of Ada Boost after tuning

After tuning the classifier, I tested it on the test data and found that F1-Score increased to 86% which is good improvement. Also, decrease in *n_estimators* to 21 from the initial value of 50 showed adding more estimators were doing no good to improve the performance.
The performance ada boost with a relatively strong base estimator was similar to that of the tuned decision tree.

AdaBoost: F1 score= 0.861

	precision	recall	f1-score	support
Attacker	0.86	0.71	0.78	1025
Defender	0.90	0.83	0.87	1760
Goalkeeper	1.00	1.00	1.00	608
Midfielder	0.75	0.86	0.80	2052
accuracy			0.84	5445
macro avg	0.88	0.85	0.86	5445
weighted avg	0.85	0.84	0.84	5445

C. k-Nearest Neighbors

The analysis started with using the vanilla implementation of k-nearest neighbours classifier. The learning curve depicted a high variance. The F1 score was around 87% which is higher than decision tree classifier and boosting algorithm.

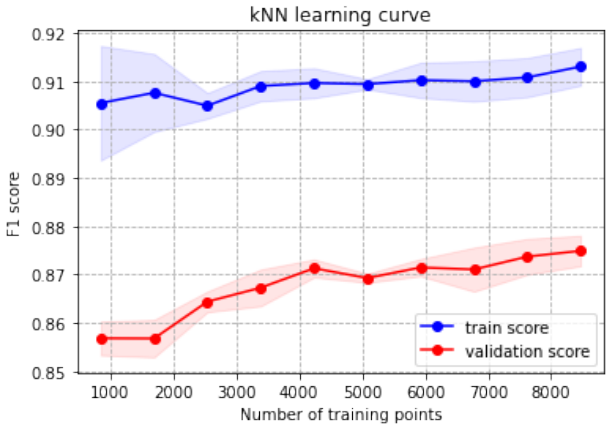


Fig. 29: Learning curve of k-NN before tuning

Complexity of the model had to be decreased to counter the high variance to see if it did any better.
I plotted a validation curve against hyper parameter namely *k* to see its effect on the performance.

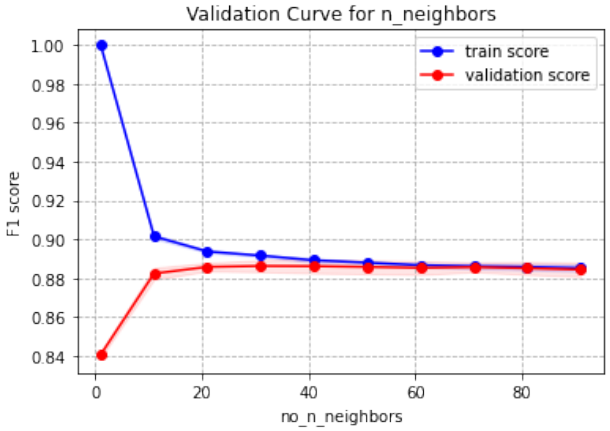


Fig. 30: Validation curve of k-NN against *k*

Analyzing train/validation scores show that changing *k* tends to over fit for most of values of k as shown above.
Performing a grid search then gave me *k=24* and *weights='uniform'*. Plotting a learning curve after training the model with best parameters is shown below.

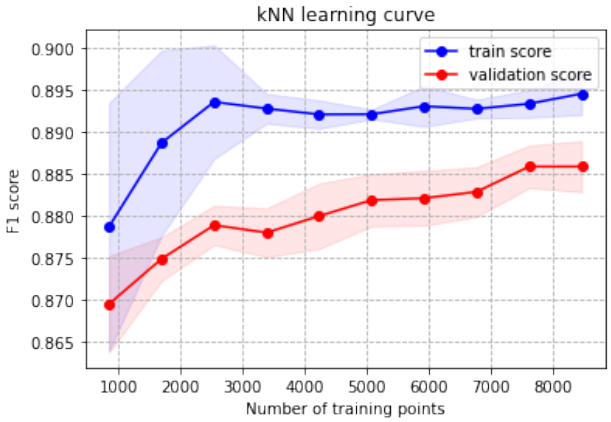


Fig. 31: Learning curve of k-NN after tuning

After tuning the classifier, I tested it on the test data and found that F1 score slightly increased to 88.2% which is around 2% better than the previous classifiers.

kNN: F1 score= 0.872

	precision	recall	f1-score	support
Attacker	0.81	0.74	0.77	1025
Defender	0.90	0.90	0.90	1760
Goalkeeper	1.00	1.00	1.00	608
Midfielder	0.80	0.83	0.81	2052
accuracy			0.86	5445
macro avg	0.88	0.87	0.87	5445
weighted avg	0.86	0.86	0.86	5445

D. Support Vector Machines

The analysis started with using the vanilla implementation of SVC classifier with *kernel='rbf'* . The learning curve of the vanilla implementation already shows good fit. Having more data here would nicely converge training and the validation scores. The F1 score was 89.7% and is one of the highest among other classifiers.

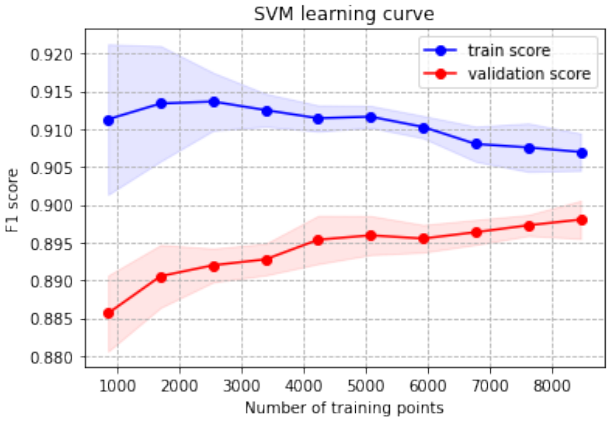


Fig. 32: Learning curve of SVC before tuning

- **kernel='rbf'**
  Complexity of the model had to be increased to counter the high bias.
  I plotted a validation curve against *gamma* hyper parameter to see its affect on the performance.

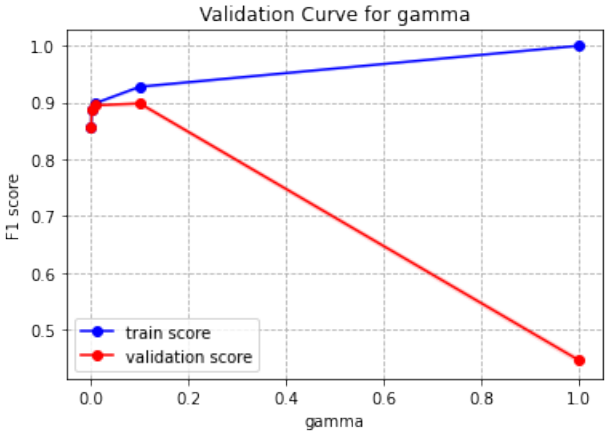


Fig. 33: Validation curve of SVC against *gamma*

High values of gamma tends to over fit the model. As gamma gets larger, the effect of c becomes negligible. Therefore, both c and gamma parameters need to be optimized simultaneously.

Performing a grid search on these parameter gave me *gamma=0.01* and *C=10*. Plotting a learning curve after training the model with best parameters is shown below

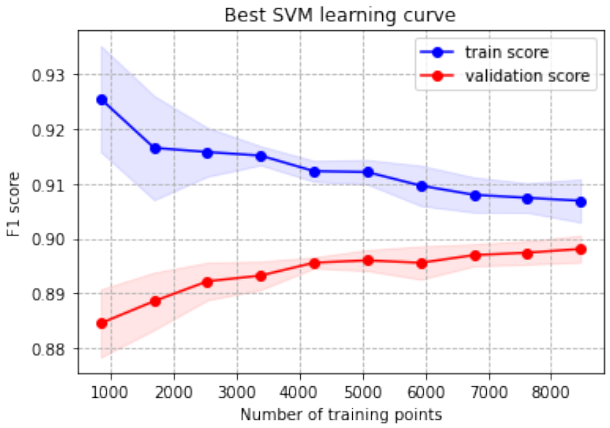


Fig. 34: Learning curve of SVC after tuning

After tuning the classifier, I tested it on the test data and found that F1 score increased to 89.9%, a marginal increase from its vanilla implementation.

SVM: F1 score= 0.898

	precision	recall	f1-score	support
Attacker	0.89	0.74	0.81	1025
Defender	0.93	0.93	0.93	1760
Goalkeeper	1.00	1.00	1.00	608
Midfielder	0.83	0.89	0.86	2052
accuracy			0.89	5445
macro avg	0.91	0.89	0.90	5445
weighted avg	0.89	0.89	0.89	5445

- **kernel='linear'** To see the effect of a different kernel on the data, I switched classifier kernel to 'linear' and plotted a validation curve to see the effect of C. As see below, a higher C over fits the data.

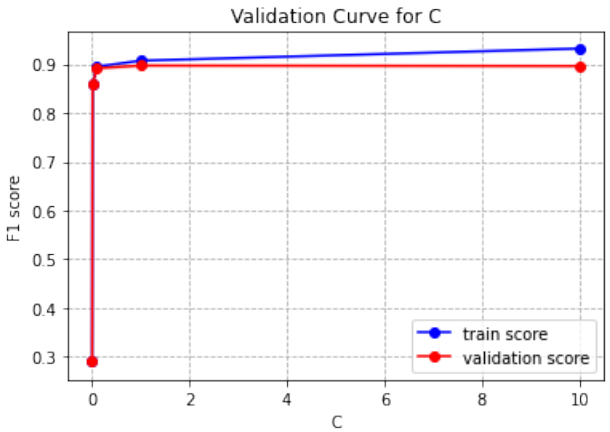
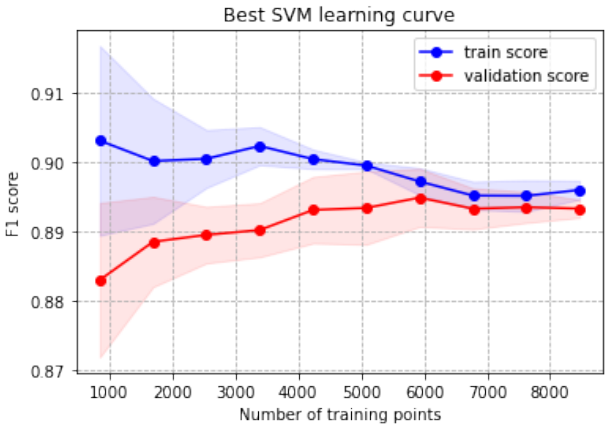


Fig. 35: Validation curve of SVC linear against *C*

I therefore performed a grid search which gave me *C=0.1*. The F1-Score for decreased slightly by 0.7% compared to the 'rbf' kernel. A 'linear' kernel was not able to classify some non linear elements of the data.



E. Neural Networks - MLP

The analysis started with using the vanilla implementation of MLP classifier with one hidden layer of 100 units from sklearn. The learning curve depicted a high variance, as it was evident from a high training score and low cross validation score. The F1 score was 88.5% which is the highest among other classifiers

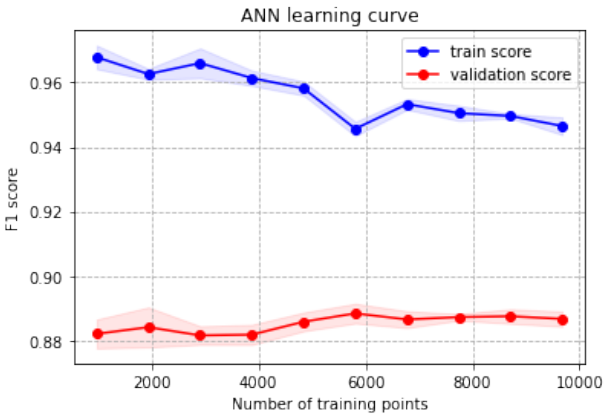


Fig. 36: Learning curve of MLP before tuning

Plotting the loss curve show that the loss decrease quickly, but as the number of iterations increase, loss decreases gradually.

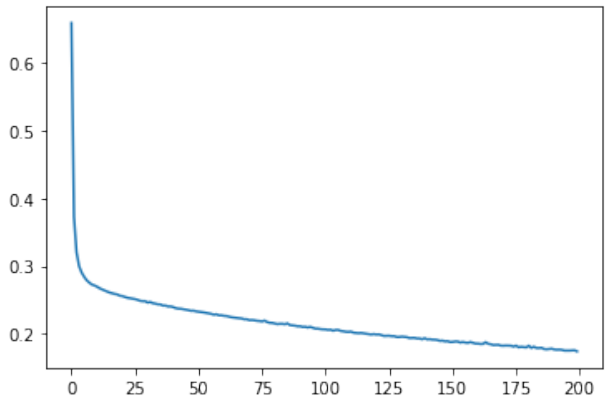


Fig. 37: Loss curve of MLP before tuning

Complexity of the model had to be decreased to counter the high variance. Increasing the layers or increasing the units per layer further increased the non linearity of the network that did not help in reducing the over fit. So, regularization parameter *alpha* had to be tuned to reduce variance.

I plotted a validation curve against *alpha* to see its affect on the performance.

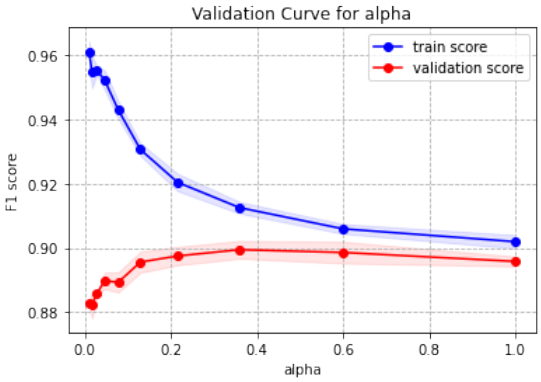


Fig. 38: Validation curve of MLP against *alpha*

Analyzing train/validation scores show that low *alpha* tends to over fit for low values and as the number increases the classifier seem to under fit.

A grid search of various parameters then gave me *alpha=0.3* and *learning_rate=constant* and *hidden_layer_size=(25,)*. Plotting a learning curve after training the model with best parameters is shown below.

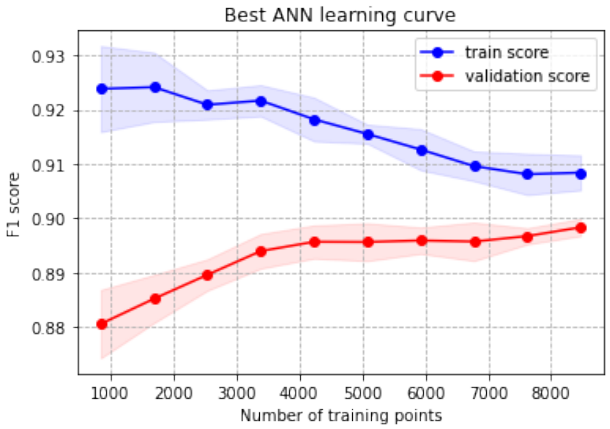


Fig. 39: Learning curve of MLP after tuning

Also, the loss curve reduces smoothly after tuning as shown below

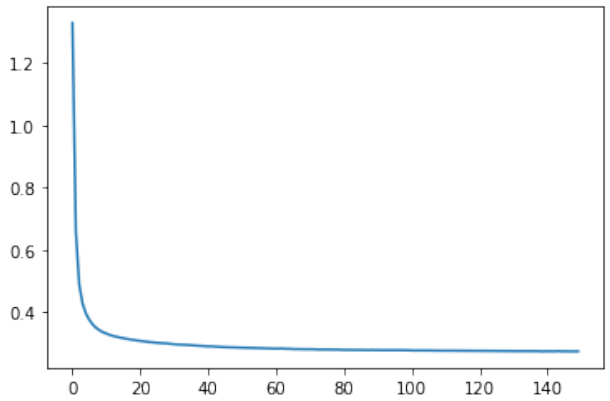


Fig. 40: Loss curve of MLP after tuning

After tuning the classifier, I tested it on the test data and found that F1 score increased to 90%. Tuned MLP was the best among all the other classifiers on this data set.

```
ANN: F1 score= 0.897
              precision    recall  f1-score   support

    Attacker       0.88      0.74      0.80      1025
    Defender       0.92      0.94      0.93      1760
  Goalkeeper       1.00      1.00      1.00       608
  Midfielder       0.83      0.89      0.86      2052

    accuracy                           0.89      5445
   macro avg       0.91      0.89      0.90      5445
weighted avg       0.89      0.89      0.89      5445
```

V. Summary

A. Stroke Prediction Dataset

Classifier	PR-AUC Score	Recall Score (stroke class)
Decision Tree	31.4%	81%
Ada Boost	18.5%	84%
k-NN	12.5%	11%
SVM	18.1%	79%
Neural Net	16.9%	6%

In the stroke prediction data set, the performance of k-NN and neural net was the worst, this could also attribute to the fact that these algorithms could not penalize the mis-classification of the minority class. On the other hand, decision tree performed the best of the lot.

B. FIFA 19 complete player Dataset

Classifier	F1 Score	F1 Score weighted
Decision Tree	86.1%	84%
Ada Boost	86.1%	84%
k-NN	88.7%	87%
SVM	89.8%	89%
Neural Net	89.8%	89%

In the fifa data set, the performance of k-NN and neural net and SVM with 'rbf' were the best. When the data is balanced and has some non linearity, then these algorithms performed better than decision tree and boosting.

C. Performance Stroke Prediction Data set

Classifier	Training score(sec)	Testing score(sec)
Decision Tree	0.025	0.002
Ada Boost	0.44	0.056
k-NN	0.035	0.078
SVM	1.20	0.136
Neural Net	24.0	0.078

D. Performance FIFA 19 complete player Data set

Classifier	Training score(sec)	Testing score(sec)
Decision Tree	0.13	0.018
Ada Boost	15.0	0.071
k-NN	0.30	2.784
SVM	12.0	0.932
Neural Net	13.0	0.002

In both the data sets, the train and test performance have similar pattern. Neural net, Ada boost and SVM takes more time to train than predict, On the other hand, k-NN takes more time to predict compared to train. Decision tree performed well on both train and test fronts.

References

[1] "Stroke-Prediction-Dataset", Kaggle Available: https://www.kaggle.com/fedesoriano/stroke-prediction-dataset. [Accessed: 01-Feb-2022]

[2] "FIFA 19 complete player Dataset," Kaggle Available: https:https://www.kaggle.com/karangadiya/fifa19. [Accessed: 01-Feb-2022]

[3] "scikit-learn documentation" Available: https://scikit-learn.org/stable/