Supervised Learning Assignment

Mehmet Oguz Kazkayasi
CS7641 – Assignment 1
mkazkayasi3@gatech.edu

Abstract

Five supervised learning algorithms (Decision Trees, Neural Networks, Boosted Trees, Support Vector Machines, and k-nearest neighbors) are implemented and optimized on two different classification datasets with different properties. The difference among the classifiers are observed and analyzed.

1. Introduction to Datasets

1.1. The UFC Fight Dataset

The first dataset is the UFC Fight Dataset. UFC (The Ultimate Fighting Championship) is the largest mixed martial arts (MMA) promotion company in the world and features on its roster the highest-level fighters in the sport [1]. The dataset contains fights since 2013 with summed up entries of each fighter's round by round record preceding that fight.

The dataset contains 3592 entries and 156 features. The features are information about the two fighters (red and blue) and 1 feature displays if the match is a title-decider. The target is the **winner** of each game. And to make it more realistic, I have removed the *number_of_rounds* feature from the dataset, so that the prediction is made before the game starts.

In UFC, the favorite fighter always gets to be the RED corner. This results in an imbalanced dataset, in which 66.26%, the **Winner** is RED. Because of this property, to predict BLUE winners correctly means more than predicting RED winners. BLUE fighter offers better rates in odds, too.

The attributes consists of both continuous and binary ones (because some of the attributes are OneHotEncoded). And the classification is binary.

As the metric during hyper-parameter optimization process, f1_score (macro) is used in the UFC Fight dataset. f1_score (macro) calculates f1_score for both classes (RED and BLUE). Then, it takes their average. So, the class with less representative data (BLUE in this case), gets more importance.

1.2. The Wine Dataset

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from **three different cultivars**. The analysis determined the quantities of 13 constituents found in each of the three types of wines. It is gathered from OpenML.org datasets [3].

All 13 features are continuous and the classification is multi-class classification consists of 3 different classes. This dataset has 1M entries and it is reduced to 5000 entries to make classification and optimization process faster and to reveal the differences among classifiers. (first 5000 is selected)

Using the features, the algorithms are used to predict the cultivar of each wine. The dataset is reasonably balanced and no class dominates any other.

This dataset presents **four** significant difference comparing to the UFC Fight Dataset. The first one is the Wine Dataset is balanced, unlike the UFC Dataset. The second is the number of features are much less (13 vs. 156). And the third one is the Wine Dataset is multi-class classification, whereas the UFC Dataset is binary. The last difference is that UFC Dataset is very difficult to predict, unlike The Wine Dataset which is easier to get over 90% accuracy.

As the metric during hyper-parameter optimization process, f1_score (weighted) is used in the Wine dataset. f1_score (weighted) calculates f1_score for all classes (1 – 2 – 3). Then, it takes their weighted average according to their data count.

2. Pre-Processing of Datasets

There were no negative attribute in any of the datasets. So, In both datasets, normalization

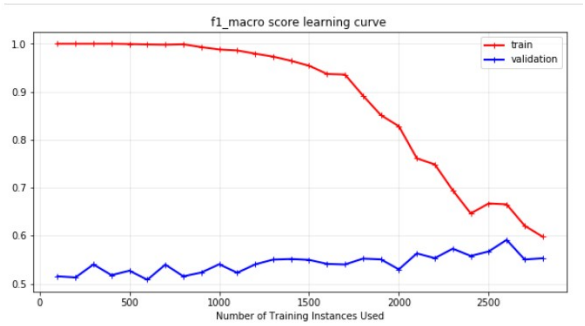
(*MinMaxScaler function*) is used to bring the features between 0 and 1.

Then the datasets are divided into two stratified splits, where 80% of them are in training/validation and 20% of them are kept in testing data. Stratified splits are used because the testing dataset should represent the same ratio on targets with the training dataset. So, our results are not biased to one class more.

3. Decision Trees

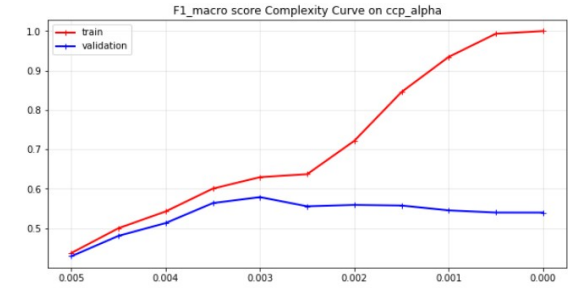
3.1. The UFC Fight Dataset

Started with *ccp_alpha=0.005*, the first f1_macro score I get is 0.5832 on the UFC Dataset.

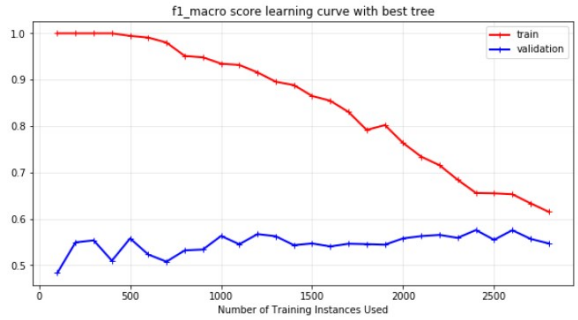


When I draw the learning curve I realized that there is too much of bias and little variance comparing to the bias. This means the algorithm can't learn the data quite well. So, I had to increase the complexity of the algorithm. I made a couple GridSearch's where I draw the learning curve in the end. It was suggested to use *ccp_alpha=0.003 and criterion=`gini` by GridSearch.*

I plotted two complexity curves using ccp_alpha and max_depth. Max-depth (pre-pruning) didn't work well (f1_macro=0.5696 best at depth 4). The plot on ccp_alpha is:



This complexity plot shows that before ccp_alpha=0.003, the model can't learn the data quite well (under-fitting). It is very high bias, even the training data has very low f1_macro score. And after especially 0.0025, the model over-fits to the data and the f1_macro score keeps increasing significantly, while validation score goes down slightly.



Using the best parameters (ccp_alpha=0.003 and criterion=`gini`), it can be seen that both the training and validation dataset's f1_scores increased by around 2%. The algorithm is still high bias and comparingly low variance, but since it's a difficult classification problem single decision trees can go so far.

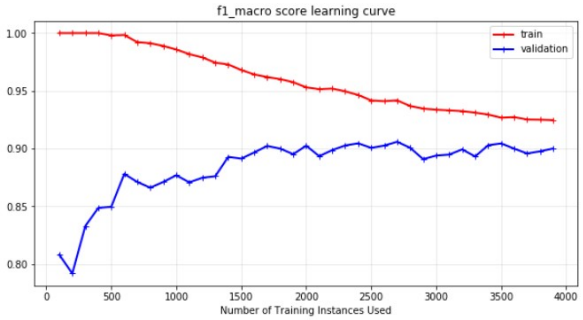
After last learning curve analysis, I have tested the best Decision Tree on the testing dataset, which I kept aside during all process.

	precision	recall	f1-score	support
0.0	0.50273	0.37860	0.43192	243
1.0	0.71828	0.80882	0.76087	476
accuracy			0.66342	719
macro avg	0.61051	0.59371	0.59640	719
weighted avg	0.64543	0.66342	0.64970	719

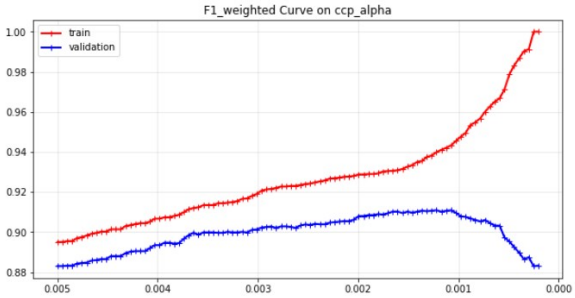
The f1_macro result is 0.5964 and it can be seen that over 50% of BLUE predictions are correct, which can be valuable for betters because BLUE fighter offers more than doubling the bet money in general.

3.2. The Wine Dataset

Started with *ccp_alpha=0.005*, the first f1_weighted score I get is 0.9033 on the UFC Dataset.

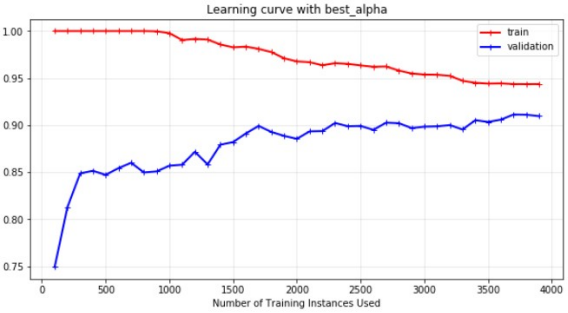


The learning curve shows that the first decision tree gets a reasonable amount of bias and variance. To optimize, I made GridSearch's on both sides (more complex and less complex). After a couple GridSearch's where I draw the learning curve in the end. It was suggested to use *ccp_alpha=0.0014 and criterion=`gini` by GridSearch. This tree much more complex comparing to the UFC dataset.* I think that is because Wine dataset is much more predictable and easier to learn.



The complexity curve on ccp_alpha parameter shows that the best parameter is 0.00122 and it can be seen that after 0.001, the f1_scores of training and validation sets gets significantly different (over-fit).

I have used the best parameter to draw a learning curve to analyze the difference between the first learning curve.



It can be seen that the bias of the algorithm is reduced significantly, whereas the variance stays almost same. This is because I increased the complexity of the algorithm carefully and didn't let any over-fitting occur.

After last learning curve analysis, I have tested the best Decision Tree on the testing dataset, which I kept aside during all process.

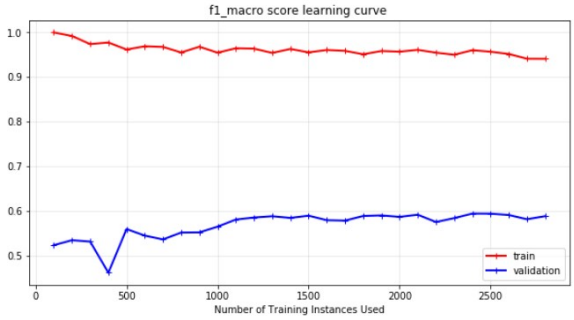
	precision	recall	f1-score	support
1	0.91139	0.91429	0.91284	315
2	0.92727	0.88586	0.90609	403
3	0.86622	0.91844	0.89157	282
accuracy			0.90400	1000
macro avg	0.90163	0.90619	0.90350	1000
weighted avg	0.90505	0.90400	0.90412	1000

The weighted f1_score reaches to 0.90412 on testing dataset, whereas validation set had 0.91330, where I did 5-fold cross validation during GridSeach's. This significant difference could result from the difference between the testing and training dataset. The classes are divided using stratified splitting but the features could be different to make this 1% difference.

4. Neural Networks

4.1. The UFC Fight Dataset

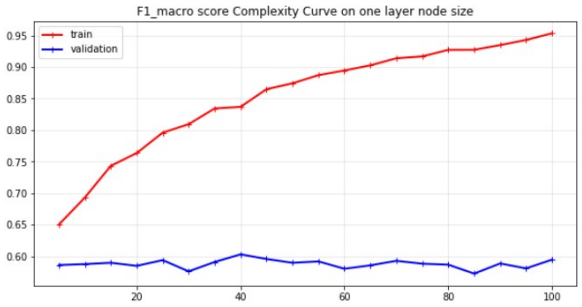
Started with the sklearn's basic MLPClassifier with one hidden layer of 100 nodes. It got a promising 0.5947 f1_macro score on cross-validation.



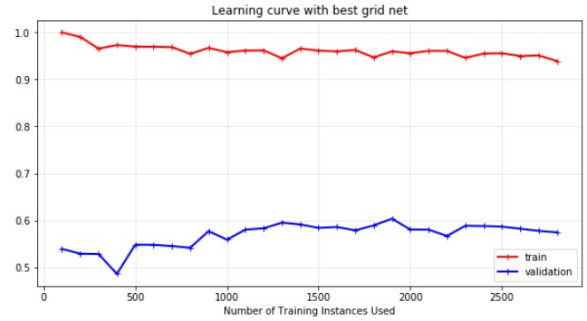
As I get the learning curve for this basic network, it can be seen that the bias is very low and the variance is quite high. This means the algorithm over-fits and we need to reduce the variance. To reduce variance I ran a GridSearch with lower number of nodes and two alpha values for regularization. I didn't need to go for multiple layers because of the low bias.

After several GridSearch's on parameters "hidden_layer_sizes", "alpha", "max_iter",

"beta_1", I used the best of them to draw a complexity curve on "hidden_layer_sizes" with one layer.



It can be seen from the complexity curve that when the hidden node number is increased, the algorithm over-fits the data and reduces the bias, but simultaneously the variance increases. So, validation score doesn't increase. So, I did a complexity curve on alpha parameter, too. But, its result didn't affect the validation score significantly. In the end, I have used the parameters I got from the GridSearch's.



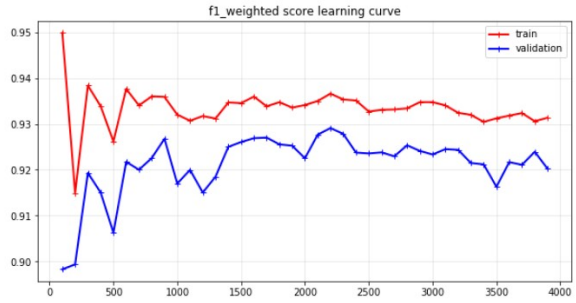
In the end, the best network I get from GridSearch's gave similar validation and training result. Since I couldn't decrease the variance I used this network on testing set.

	precision	recall	f1-score	support
0.0	0.53807	0.43621	0.48182	243
1.0	0.73755	0.80882	0.77154	476
accuracy			0.68289	719
macro avg	0.63781	0.62252	0.62668	719
weighted avg	0.67013	0.68289	0.67362	719

As a result, I got much better result on BLUE (0) f1_score comparing to Decision Tree. Better for both precision and recall. Also, the f1_macro got over 62.5%. This is because Neural Network had really low bias. It can be said that Neural Networks are able to fit into the dataset really well even with one layer of hidden layer.

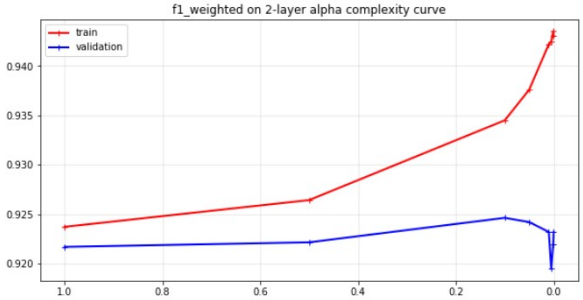
4.2. The Wine Dataset

Started with the sklearn's basic MLPClassifier with one hidden layer of 100 nodes. It got a promising 0.9255 f1_weighted score on cross-validation.



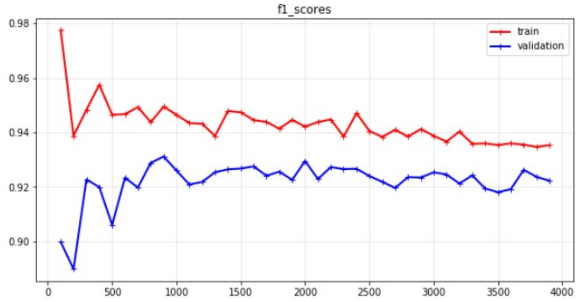
The learning curve shows that the first neural network gets a reasonable amount of bias and variance. To optimize, I made GridSearch's on both sides (more complex and less complex). So, I have used 2-layer network, unlike UFC Dataset case. I have aimed to reduce the bias more as well as variance by setting other parameters such as *alpha*, *learning_rate*.

I have checked two-layer networks using GridSearches with different *alpha* and *learning_rate* parameters. These GridSearches pushed me to increase the number of hidden layer nodes (to decrease the bias) and also increase the alpha parameter (to not increase the variance) simultaneously. In the end, I have decided to use (60, 60) hidden layer set with 0.001 learning_rate_init and 'invscaling' learning rate during the learning process. And I have used complexity analysis to decide on the alpha value.



It can be seen that at 0.1 alpha value, the best validation result is obtained. After 0.1, the algorithm over-fits to the training data by reducing the bias and increasing the variance.

I have plotted the learning curve of the best neural network *I got from the GridSearch's.*



As we can see, comparing to the first learning curve, the algorithm has lower bias and lower variance. When I tested the algorithm on the testing set:

	precision	recall	f1-score	support
1	0.94249	0.93651	0.93949	315
2	0.94815	0.95285	0.95050	403
3	0.93972	0.93972	0.93972	282
accuracy			0.94400	1000
macro avg	0.94345	0.94303	0.94323	1000
weighted avg	0.94399	0.94400	0.94399	1000

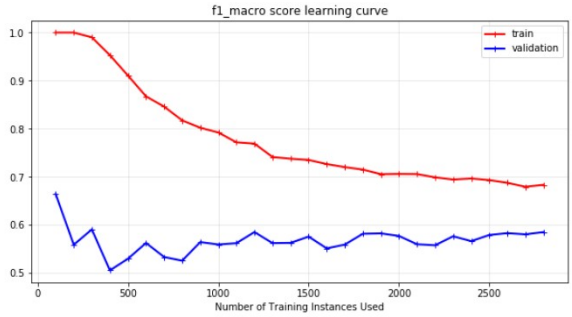
It got 0.9432 f1_macro score, which is around 3.5% higher than Decision Tree. This is because Neural Network had a lower variance because it generalized better due to the regularization. And the testing result got even better than training and validation dataset, interestingly. This could be because the training dataset had more instances to train the algorithm. But, mostly I believe it is caused by the variance.

5. Boosting

In this section, I have used AdaBoost in sklearn library.

5.1. The UFC Fight Dataset

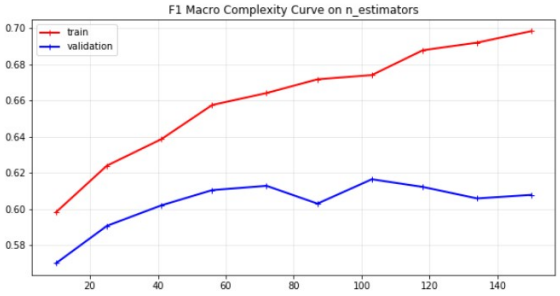
Started with the sklearn's basic AdaBoostClassifier, boosting got 0.5861 f1_macro score on cross-validation. As I draw the learning curve for this classifier:



The boosting algorithm has a lower variance comparing to the Decision Tree, but its variance is a bit higher. The low bias seems promising because if the variance could be lowered as in the Decision Tree, it will bring a really good result.

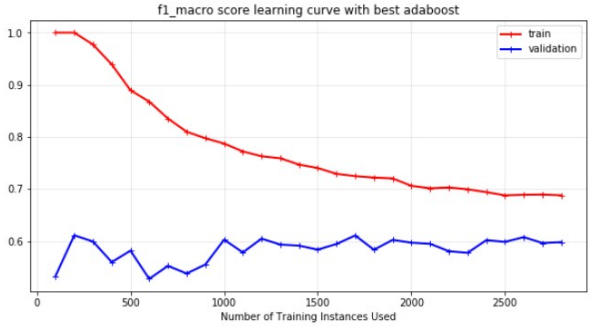
Because of that I delved into GridSearch's, drawing the learning curve of the best_estimator in the end of each. I used the "number of estimators", "max_depth" and "splitter (best or random)" of the base estimator (Decision Tree) in GridSearch.

In the end, I get 100 estimators with 0.55 learning_rate and the Base Estimator is depth=1 and splitter="random" with class_weight="balanced".



I draw a complexity curve on the number of estimators. It can be seen that with few decision trees, the model can't fit well to the data and after 100, it over-fits and training score keeps increasing steeply.

I have selected 100 trees and draw a learning curve with the best parameters I got from GridSearch's and other learning curves.



As it can be seen, the bias of the AdaBoost has been lowered and the variance kept almost same. The model is still high-bias and low variance but got better results in f1_macro score.

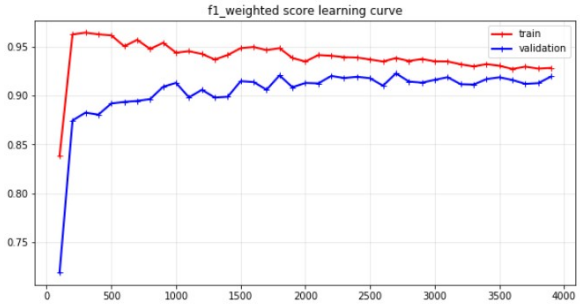
In the end, I have used the testing set and get these values:

	precision	recall	f1-score	support
0.0	0.46821	0.66667	0.55008	243
1.0	0.78284	0.61345	0.68787	476
accuracy			0.63143	719
macro avg	0.62552	0.64006	0.61898	719
weighted avg	0.67651	0.63143	0.64130	719

As it can be seen, the f1_macro result got pretty high and especially the f1-score of the BLUE (0) is much higher comparing to other algorithms. On the other hand, the main metric I have used (f1_macro) is lower than Neural Networks and this is because Neural Network got really low bias by fitting the training data well.

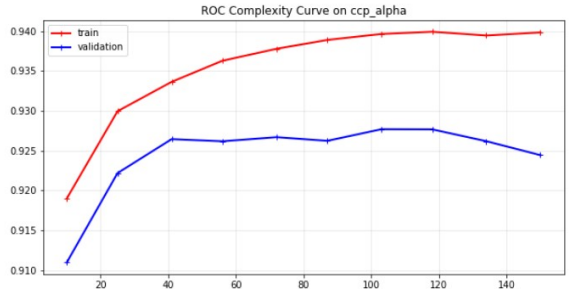
5.2. The Wine Dataset

Started with the sklearn's basic AdaBoostClassifier, boosting got 0.92 f1_weighted score on cross-validation. As I draw the learning curve for this classifier:

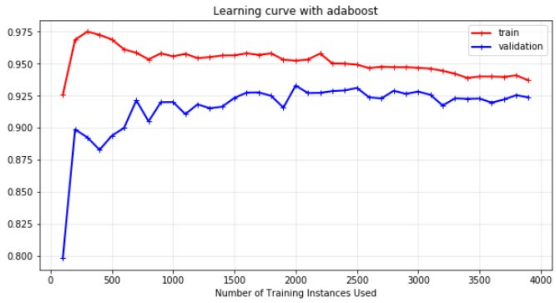


It can be seen that there is really little variance in this order. So, I have gone into GridSearch to find ways to reduce the bias by choosing the best hyper-parameters in max_depth, splitter, n_estimator, and criterion parameters. When none of these parameters help me to reduce the bias, I have made a grid search with more n_estimators and lower learning_rate.

By increasing the n_estimators and lowering the learning rate, the bias is reduced a little bit. Then I made a complexity analysis on n_estimators:



In this graph, it can be seen that after 120, the validation score drops significantly. And, under 40, the training and validation scores are both low, which shows under-fitting. I have chosen 110 estimators and draw another learning curve:



As it can be seen comparing to the first learning curve, the variance is increased a bit but the bias is reduced more. The classification score is higher. I have used this hyper-parameters on testing dataset and I got these results:

	precision	recall	f1-score	support
1	0.90578	0.94603	0.92547	315
2	0.94148	0.91811	0.92965	403
3	0.91727	0.90426	0.91071	282
accuracy			0.92300	1000
macro avg	0.92151	0.92280	0.92194	1000
weighted avg	0.92340	0.92300	0.92299	1000

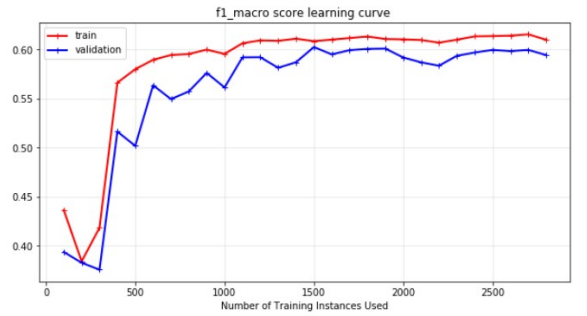
I have got 0.923 as a result, which is almost 1.5% more than single Decision Tree result, but lower than the Neural Network result because Neural Network

had less bias due to its flexibility to fit to training dataset.

6. Support Vector Machines

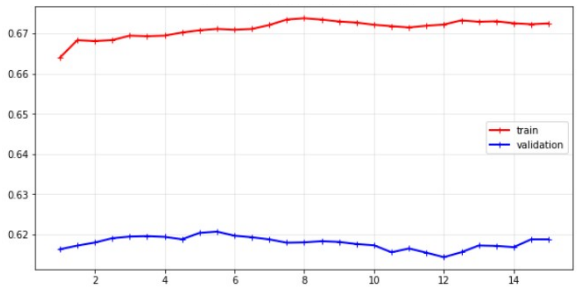
6.1. The UFC Fight Dataset

Starting with the sklearn's basic SVC with 'rbf' kernel and C=1.0, I have added class_weight: balanced parameter and got 60.27% f1_macro score on cross-validation. As I draw the learning curve for this classifier:



This learning curve shows that the algorithm under-fits the data because for training the f1_macro score is really low and it gets increased with more data. There is really large bias in this algorithm. And I tried to fix it with different kernel's and C parameter with gamma for less regularization. I needed more complex algorithms.

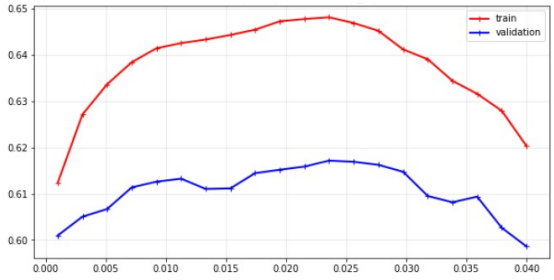
First GridSearch suggested linear kernel and I have done a complexity analysis on C parameter to see how it fits to the training and validation dataset.



Even though almost all of the C parameters resulted in similar values, the best two results are from 5 and 5.5, so that I chose the 5.5 parameter. But, in this complexity analysis, there can't be said much about the bias-variance trade-off.

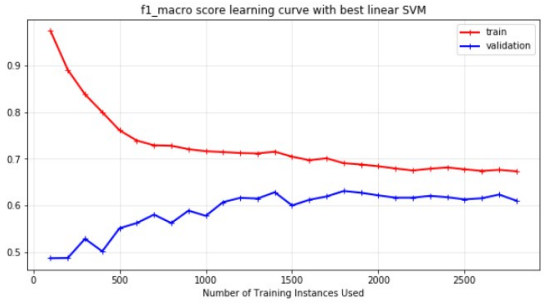
The second kernel I chose was sigmoid kernel. I have optimized the gamma and C parameters on sigmoid kernel and done a complexity analysis on C parameter.

I got C parameter as 11.0, which is higher than the linear kernel and did a complexity analysis on parameter gamma to make the best fit to the data.



In this complexity analysis of gamma parameter, the training and validation set behaved similar on values. So, I picked up the best one (0.025).

The learning curve of the best linear SVM:



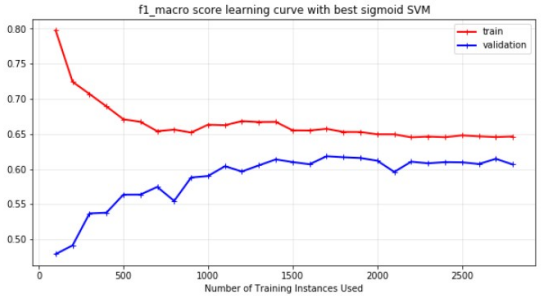
As it can be seen the model is high-bias and lower variance comparing to the Neural Networks. I believe The linearity of the SVM kernel has a role in this.

So, I used linear SVM algorithm on the testing dataset. The linear kernel got

	precision	recall	f1-score	support
0.0	0.46791	0.72016	0.56726	243
1.0	0.80290	0.58193	0.67479	476
accuracy			0.62865	719
macro avg	0.63541	0.65105	0.62102	719
weighted avg	0.68968	0.62865	0.63845	719

The f1_macro score is 0.621 for the linear kernel and it is only worse than what neural networks gave. It can be said that SVM fit the training data quite well, too.

The learning curve of the best sigmoid SVM:



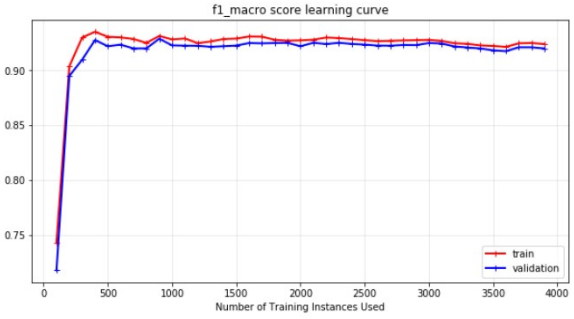
It can be seen that the sigmoid SVM has a lower variance and a higher bias. It was unexpected for me because I believe sigmoid would fit the training dataset better. On the other hand, it depends on the dataset, as well. The dataset could be better for linear classifiers. I have tested it on testing dataset.

	precision	recall	f1-score	support
0.0	0.46381	0.71193	0.56169	243
1.0	0.79769	0.57983	0.67153	476
accuracy			0.62448	719
macro avg	0.63075	0.64588	0.61661	719
weighted avg	0.68485	0.62448	0.63441	719

The result on f1_macro metric is slightly worse than the linear kernel. This is because sigmoid kernel got a larger bias and variance is similar in both.

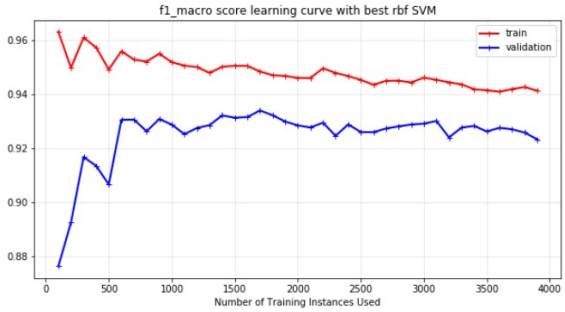
6.2. The Wine Dataset

Started with the sklearn's basic SVC, I got 0.92 f1_weighted score on cross-validation. As I draw the learning curve for this classifier:



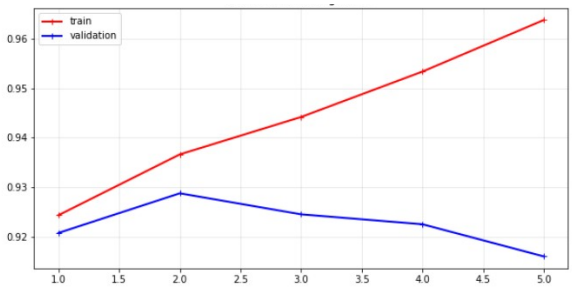
Even though I get a good f1_macro score, the learning curve suggests that the SVM under-fits the dataset because there is no difference between the training and the validation scores. So, I needed to increase the complexity of the SVM algorithm.

I have used the GridSearch and I got the 'rbf' kernel as the most promising one (among rbf, linear, and poly). I have used GridSearch on C and gamma parameters. In the end, the best results are obtained using C=4 and gamma=0.54. The learning curve using this values:



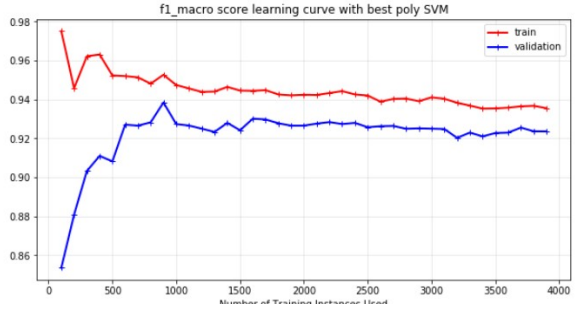
It can be seen that the bias is reduced significantly at the expense of adding some variance using the rbf kernel.

The second kernel I have used on the Wine Dataset is poly (chosen by GridSearch among linear, poly, and sigmoid). I have used GridSearch on degree, C, and gamma parameters. In the end, the best results are obtained using C=6 and gamma=0.54. And I have done complexity analysis on degree parameter.



It can be seen that after degree 2 the kernel overfits to the training dataset and f1_score of the validation set drops down.

The learning curve using this values:



Comparing to the rbf kernel, it can be seen that poly kernel resulted in more variance and less variance.

So, I have used both kernels on the testing dataset. The rbf kernel got:

	precision	recall	f1-score	support
1	0.92260	0.94603	0.93417	315
2	0.94472	0.93300	0.93883	403
3	0.93190	0.92199	0.92692	282
accuracy			0.93400	1000
macro avg	0.93307	0.93367	0.93330	1000
weighted avg	0.93414	0.93400	0.93400	1000

f1_macro score of 0.933, which is better than all classifiers except neural networks, so far.

The poly kernel got:

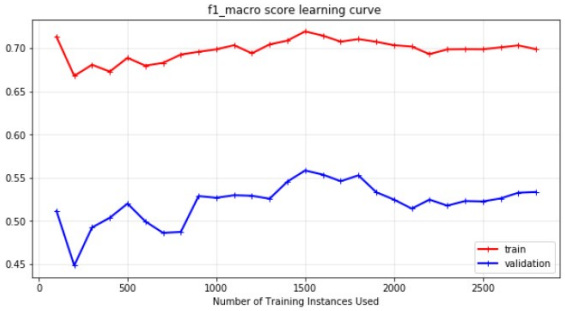
	precision	recall	f1-score	support
1	0.91950	0.94286	0.93103	315
2	0.94015	0.93548	0.93781	403
3	0.93841	0.91844	0.92832	282
accuracy			0.93300	1000
macro avg	0.93269	0.93226	0.93239	1000
weighted avg	0.93315	0.93300	0.93300	1000

almost the same result on f1_macro result. The difference between their variance and bias neutralized in the testing dataset and both kernels got almost same score on Wine dataset.

7. k-Nearest Neighbors

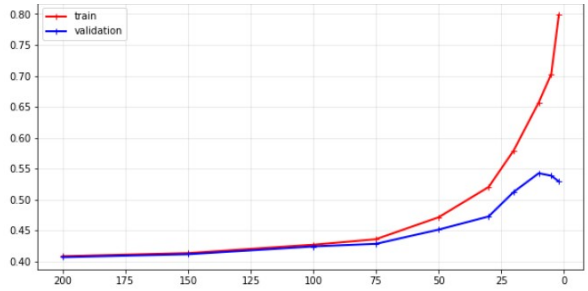
7.1. The UFC Fight Dataset

Starting with the sklearn's KneighborsClassifier's original parameters, I got 53.8% f1_macro score on cross-validation, which seems lower than all other algorithms. As I draw the learning curve for this classifier:



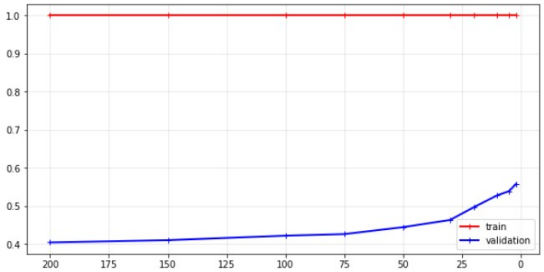
This algorithm has both high bias and high variance. So, I decided to use both 'distance' and 'uniform' weights and plot complexity analysis plots.

I have used the Uniform weights first, and got this chart:



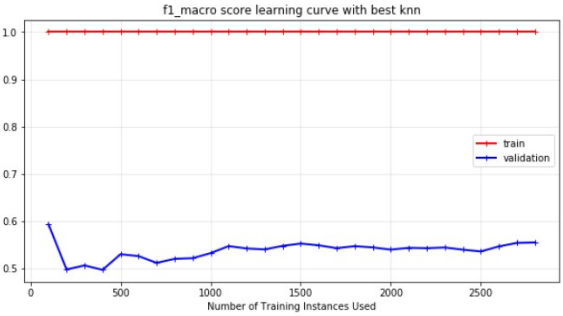
10-nn gave the best result and as the complexity rises from 10-nn, it can be seen that algorithm overfits to the training dataset and starts to decrease validation f1_score. The best classification f1_score on uniform weights knn is 0.5425, which is pretty low.

Then I have used the inverse-Distance weights and got this chart:



In the chart, training data always have 1 as f1_score because instances have 0-distance with the closest dataset and it has infinite effect on that instance. So, it works as 1-nn on uniform knn. It can be seen that the best result is got using 2 neighbors on distance-based weights. The best validation score is 0.5579 so I have used this one as the final classifier.

The learning curve using this classifier:



It can be seen that with data, the f1_score increases slightly. Because training dataset always gets 1 score, it is not possible to talk about bias-variance tradeoff in this algorithm.

The results from testing dataset:

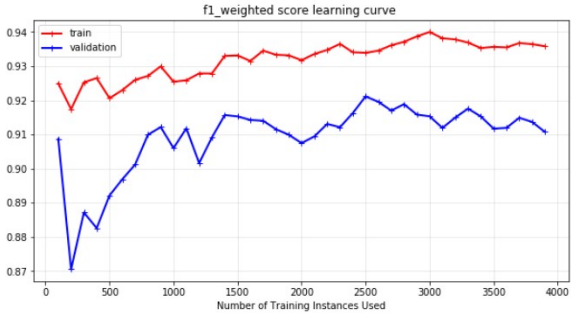
	precision	recall	f1-score	support
0.0	0.38528	0.36626	0.37553	243
1.0	0.68443	0.70168	0.69295	476
accuracy			0.58832	719
macro avg	0.53485	0.53397	0.53424	719
weighted avg	0.58332	0.58832	0.58567	719

```
[[ 89 154]
 [142 334]]
```

f1_macro score at 0.53, which is way lower than all other algorithms. This is because this dataset has 156 features and they all weigh uniformly on the decisions of KNN algorithm even though many of them has little say on the result.

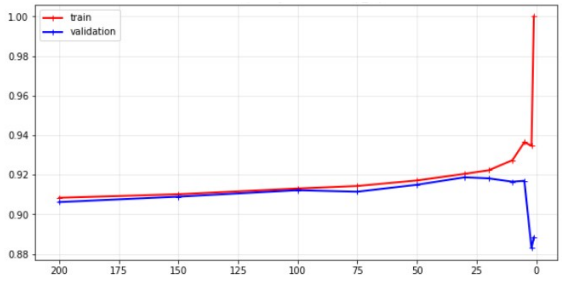
7.2. The Wine Dataset

Starting with the sklearn's KneighborsClassifier's original parameters, I got 91.6% f1_weighted score on cross-validation, which is much better comparing the UFC dataset. As I draw the learning curve for this classifier:



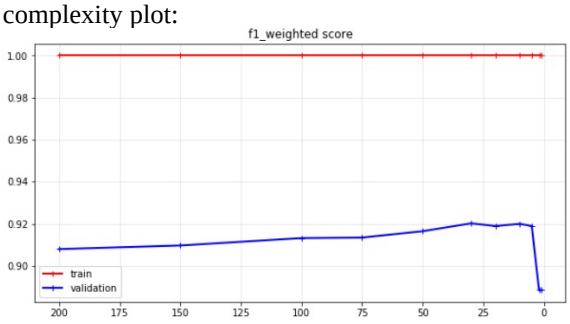
The algorithm seems to have some bias and some variance, but not too much of any of them. So, I will use both uniform and distance-based weights and draw complexity analysis plots to get the minimum combination of bias and variance.

Using uniform-weights, I got this complexity curve:

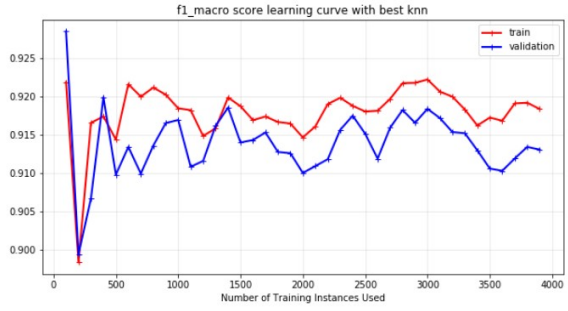


It can be seen that after 30 neighbors, the model over-fits to the training data and loses its f1_score on validation dataset. The best result is 0.9187.

Using the distance-based weights I got this complexity plot:



Also similarly, the distance-based KNN got the best result on 30 neighbors. And its f1_weighted score is 0.9202, which is almost same with the uniform KNN. So, I used the uniform-weight KNN to analyze the learning curve better.



Using 30 neighbors, it can be seen that the values of both training and validation dataset are pretty close (as the y-axis is limited from 0.9 to 00.925). As a result of optimizing the hyper-parameters, we have reduced the variance significantly and reached this result. On the testing dataset:

	precision	recall	f1-score	support
1	0.91718	0.94921	0.93292	315
2	0.93052	0.93052	0.93052	403
3	0.93727	0.90071	0.91863	282
accuracy			0.92800	1000
macro avg	0.92832	0.92681	0.92735	1000
weighted avg	0.92822	0.92800	0.92792	1000

The best KNN gave 0.928 as weighted f1_score, which is quite close to other algorithms, even though not the best. This is because the wine dataset has much less features comparing to the UFC dataset.

7. Conclusion

Five supervised learning algorithms are ran in two distinct datasets and the results are as followed.

The UFC Fighting Dataset

Algorithm	**BLUE precision**	**f1-macro**
Decision Trees	0.5027	0.5964
Neural Network	0.5381	0.6267
AdaBoost	0.4682	0.6190
SVM	0.4679	0.6210
KNN	0.3853	0.5342

It can be seen that Neural Networks and SVM's got over 62% on f1-macro and AdaBoost gets pretty close. This means that the dataset has a very complex nature and complex algorithms (or combination of simple ones) are needed to learn the dataset. On the other hand, Neural Network gave the best precision on BLUE predictions, which would be the best useful metric for betting on UFC fights.

KNN algorithm did really poorly and this is because the dataset has 156 features and it is not possible for KNN to distinguish more significant features.

The Wine Dataset

Algorithm	**Accuracy**	**f1-weighted**
Decision Trees	0.9040	0.9035
Neural Network	0.9440	0.9432
AdaBoost	0.9230	0.9219
SVM	0.9340	0.9333
KNN	0.9280	0.9274

In the wine dataset, even though there are less features, the classification seems to be a complex one again. More flexible algorithms such as Neural Network and SVM fits better than other datasets on wine dataset. An interesting result I got was the KNN algorithm, which performed better than AdaBoost. This is because all the features seem to be significant and there are little features comparing to the UFC Fighting Dataset.

8. References

[1] "Ultimate Fighting Championship," Wikipedia, 09-Feb-2020.[Online].Available:https://en.wikipedia.org/wiki/Ultimate_Fighting_Championship. [Accessed: 10-Feb-2020].

[2] K. Aggarwal, "UFC Fight Data," Kaggle, 11-Dec-2018. [Online].Available:https://www.kaggle.com/calmdownkarm/ufcdataset. [Accessed: 10-Feb-2020].

[3] J. Vanschoren, "wine," OpenML. [Online]. Available: https://www.openml.org/d/187. [Accessed: 10-Feb-2020].