CS4641 Project 1 Analysis Report

Kristian Suhartono / 903392481

Datasets

DOTA 2 Games Results: This dataset are actual match results from a popular online MOBA game called DOTA 2. Each datapoint consisted of which team won the match, some cluster and location information (which was dropped because I didn't deem it related to the problem I want to solve), and a which heroes were selected in the match. There are always 10 distinct heroes every match. The challenge that I attempted with this dataset is to attempt to classify which team would win the game given the set of heroes that were selected for the game. Which is not a simple task due to the large hero pool, 103 heroes, which can really make it hard for the classifier to predict. 92649

```
 1    48782
-1    43867
Name: -1, dtype: int64
```

The amount of data for each class, 1 being one team winning and -1 the other team winning

Poker Hands : A dataset of a poker hands, which consists of 5 cards each with their suit and rank, and the corresponding label of what poker hand this is as is described below. The problem that is given is that given a set of 5 cards, we have to classify which class of poker hand does this set belong to given the set labelling below.

```
S1 "Suit of card #1"
Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
C1 "Rank of card #1"
Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)

...

S5 "Suit of card #5"
C5 "Rank of card #5"
```

Input (X)

```
0: Nothing in hand; not a recognized poker hand
1: One pair; one pair of equal ranks within five cards
2: Two pairs; two pairs of equal ranks within five cards
3: Three of a kind; three equal ranks within five cards
4: Straight; five cards, sequentially ranked with no gaps
5: Flush; five cards with the same suit
6: Full house; pair + different rank three of a kind
7: Four of a kind; four equal ranks within five cards
8: Straight flush; straight + flush
9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush
```

```
0    12493
1    10599
2     1206
3      513
4       93
5       54
6       36
7        6
9        5
8        5
Name: hand, dtype: int64
```

Output (y)

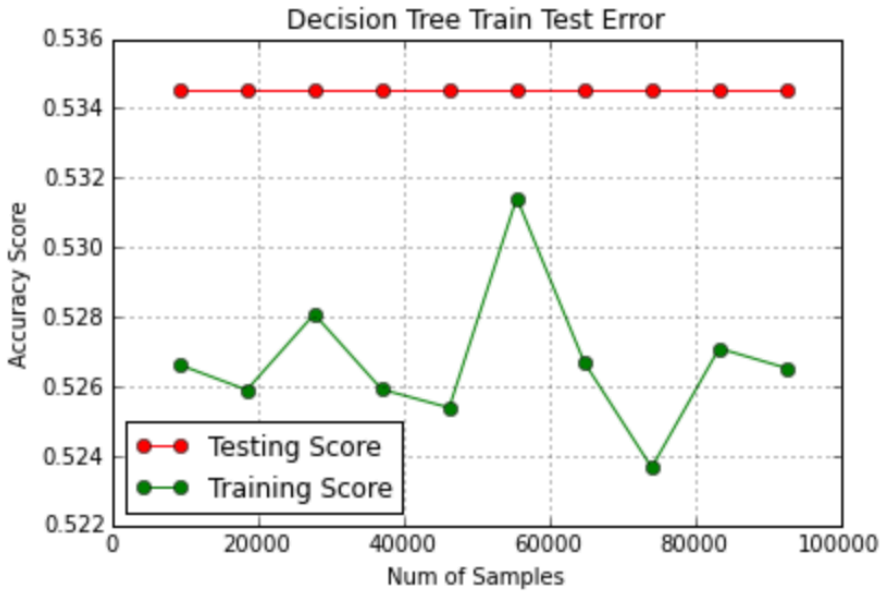
Why are these interesting datasets?

The Dota 2 match dataset is interesting to me personally as a player of the game. There is a term called "meta-game" that is often described as a set of heroes that would have higher likelihoods of being able to win a match due to how powerful that hero is in the current patch of the game. From some live match data, our model may be able to classify and correctly predict which heroes are more likely to bring victory to a certain team. Sure there are a lot of

other factors to consider in such a game, but if a certain hero combination worked this kind of model can be used to find out which hero combinations would be able to bring you that victory. Such information may be useful for players of the game, for the casual people, it's nice to be able to win matches! While for the pro players scene, it can be valuable information that can be used to analyze opponents, and create strategical advantages to ensure that their team can win a match. With respect to machine learning, this is interesting because there are so many attributes in the problem. With a hero pool of 103 which is constantly being added upon, the problem is extremely big. When you have 103 heroes and 10 heroes can be picked in a match, there can be 23591276125340 combinations!

The poker hand dataset is interesting as game playing AIs are quite a subject of research, as people are still trying to research if there is a way that a machine can beat a human in playing a game. With games where the possibilities of movement are small such as checkers certainly this is very easy. But games such as chess and Go, albeit Go having been classified as solved by AlphaGo, it is very hard to be able to be able to certainly beat a human. Poker being one of these games where there are many things to consider to reach the objective of winning a game of poker. The first step to be able to do that is certainly by being able to understand what sort of hand do you have. In respect to machine learning, this dataset is interesting because it'll allow me to find out whether a machine can figure out the rules of the game given past information. Rules of the game in this case being what sort of hand does this set of cards classify to. It would also be interesting that there is a much larger test dataset provided, however the only way to check the percentage of correct predictions would be by submitting the prediction results to Kaggle. This test dataset is biased unlike the training set which is more evenly distributed, so it can be very interesting to see whether the rules that model learned is general enough to be able to figure out the problem. Also, the given number of examples are extremely low for this dataset, especially for the last classes, it is interesting to find out how the models would perform with this spread of data

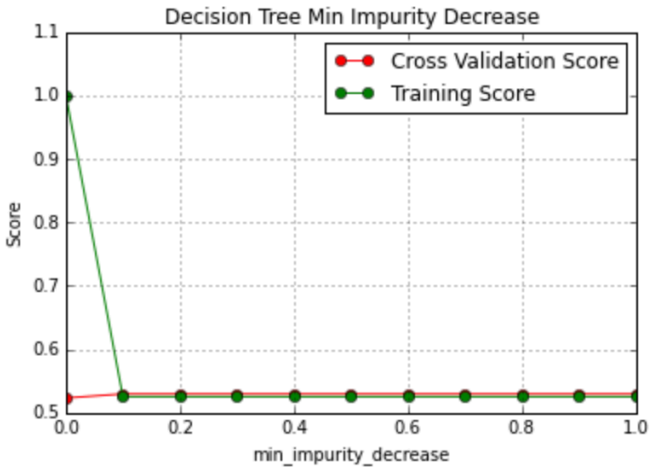
Decision Trees




```
1    10293
Name: team, dtype: int64
```

Dota 2 Dataset:

Conclusively, Decision Trees perform very badly on this problem. With a Test Error that was basically a straight line describing the actual ratio of the dataset (53% for a team winning and 47% for the other team winning). This result was due to the classifier predicting only that one team wins all the time. This can be caused by the fact this problem cannot be simply defined as a tree due to the massive amounts of possibilities, as in it needs to be very specific in having seen a certain pattern of certain heroes winning a game multiple times in order to be able to understand that this set of heroes would win a game. Whereas in decision trees, there are decisions that lead to another decision where the consequences may generally not end up being the same thing. As can be seen in the training score, even with the training dataset, the D-tree is still confused as to what should it predict here.




```
print(dtGS.best_score_)
print(dtGS.best_estimator_.presort)
print dtGS.cv_results_['mean_test_score']
print dtGS.cv_results_['mean_train_score']
```

```
0.525267134377
True
[ 0.52526713  0.52362655]
[ 1.  1.]
```

Samples of the results of tuning the parameters, as we can see here cross validation results show that the score barely has any improvements

Another note is that tuning the parameters barely made any changes to the accuracy of the model, most of the parameter changes brought over 3dp (decimal places) improvements in accuracy, meaning only 0.1% increases or none at all. This can be attributed to again the same fact that this algorithm just doesn't fit this problem at all.

Almost unsurprisingly, pruning fails to have any effect too. Since the decision tree is unable to classify it in the beginning, removing sections of the tree obviously doesn't make much of a difference.

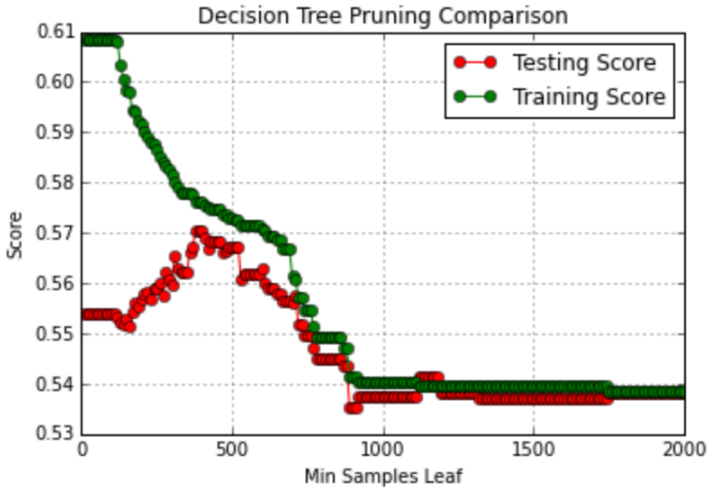
Poker Dataset:


```
0     1784
1      717
Name: hand, dtype: int64
             precision    recall  f1-score   support

          0       0.58      0.82      0.68      1252
          1       0.56      0.37      0.45      1075
          2       0.00      0.00      0.00       104
          3       0.00      0.00      0.00        49
          4       0.00      0.00      0.00         8
          5       0.00      0.00      0.00         3
          6       0.00      0.00      0.00         6
          7       0.00      0.00      0.00         2
          8       0.00      0.00      0.00         1
          9       0.00      0.00      0.00         1

avg / total       0.53      0.57      0.53      2501
```

Performs much better than the Dota2 Dataset, here however, we can see that most of the classifications are only made for the first 2 classes. Being the most common, the D-Tree probably generalized most of it's predictions to those cases after the pruning of the tree. The pruning of the tree might have caused the more finer information that might have led to the tree understanding the other 8 classes to be omitted. Interestingly there is a dip in accuracy when 0.8 of the dataset is used to train, this might have been the other classes (not 0 or 1) that appeared more and lowered the confidence of the classifier in its prediction rate of 0s and 1s, leading it to be more confused on what is 0 and 1, lowering the accuracy. Note that there is a visible trend that the score follows each other, I believe this means that the classifier is not at all overfitted, and rather as a result of the low amounts of data, it is already general by default and might even be underfitting.

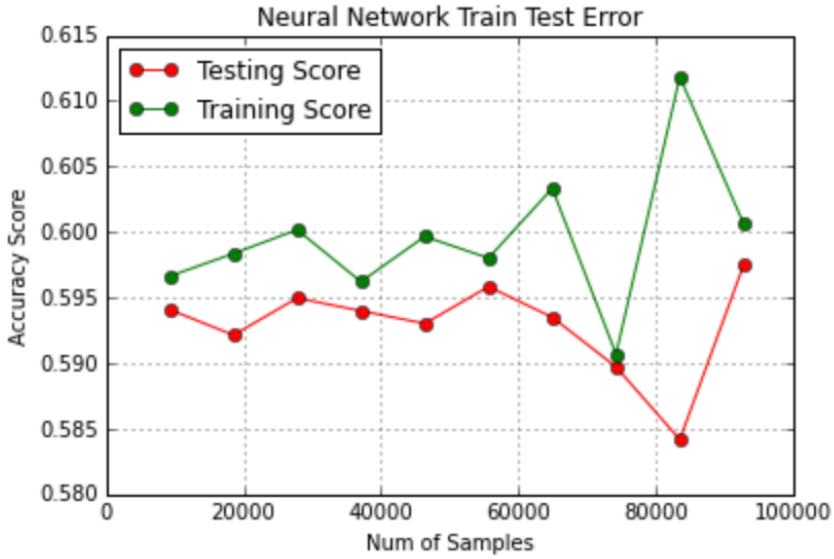


As can be seen here, the train-test score converges the more leave samples are pruned, however, the accuracy of the prediction seems to max out at 380 (which is the value I'm using). I guess the most likely explanation is, after that threshold, the model starts to over-generalize and it starts losing accuracy in prediction.

Decision Trees are the best in runtime for both the datasets. Averaging in under a minute of runtime for training and predicting.

Neural Nets

Dota2 Dataset:

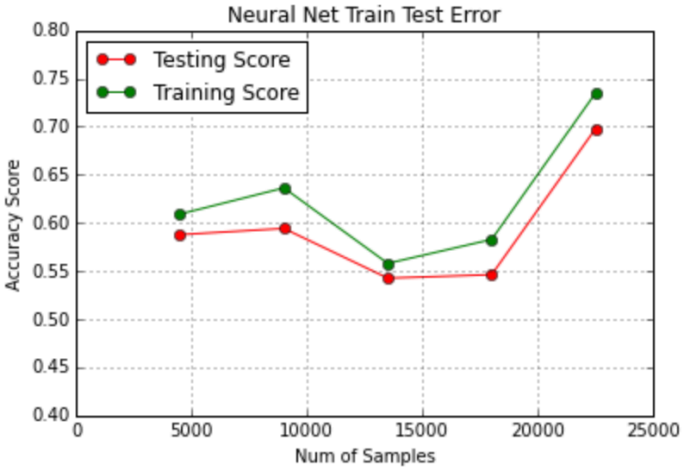


Here again, we see that the training and testing score attempts to converge in most of the times with the best accuracy and convergence achieved at the usage of the whole dataset. Right before that though, with using 0.9 of the dataset, we can see a spike in training score and a drop in testing score, which definitely suggests that it overfitted to the training set there, but the next 0.1 data had managed to generalize it more.

Tuning parameters have variable results on the dataset:

- Hidden layer size: test score mostly follows the train score and goes up and down with no descriptive trend, may suggest that the size doesn't affect as much here.
- Alpha : converges to a score and then stays mostly similar.
- Max iteration: Huge accuracy spike at higher max_iteration numbers. May suggest overfitting.

Poker dataset:




```
1    1412
0    1065
2      11
3       6
5       5
7       1
6       1
Name: hand, dtype: int64
```

	precision	recall	f1-score	support
0	0.79	0.67	0.73	1241
1	0.59	0.78	0.67	1059
2	0.18	0.02	0.03	128
3	0.33	0.04	0.07	52
4	0.00	0.00	0.00	6
5	0.80	0.80	0.80	5
6	0.00	0.00	0.00	8
7	0.00	0.00	0.00	1
8	0.00	0.00	0.00	1
avg / total	0.66	0.67	0.65	2501

Again the testing score follows a trend of following the accuracies of the training score. However, we can see a much more diverse spread of results in the predictions, in that the

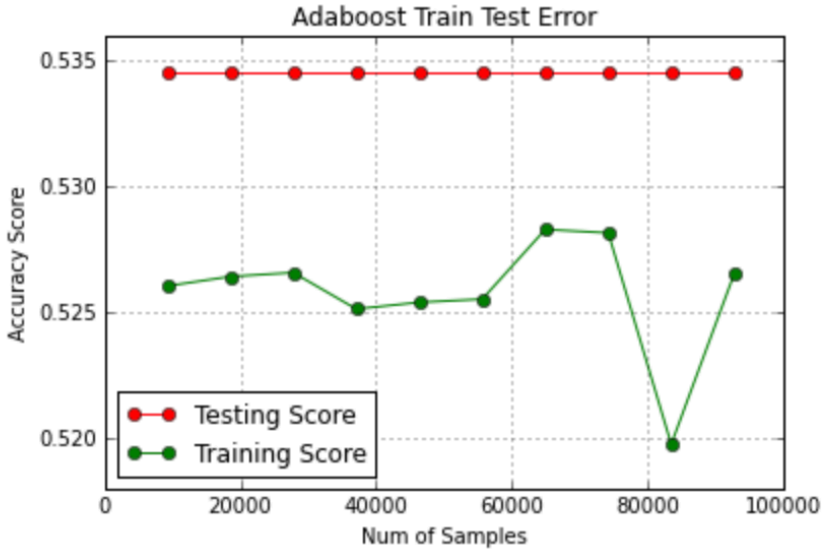
neural net is able to make more correlations among the classes and is able to even try predicting some of the rarer cases. This can be attributed to the fact that the definition of a neural network itself fits this problem well, the algorithm is able to express the problem well, and thus is able to make better guesses.

Tuning of parameters results:

- Alpha: again a trend of following the training score, no major interesting spikes may show that the alpha value doesn't affect the overfitting or underfitting of the data in this case.
- Hidden layers size: A continuous uptrend of accuracy until (500,) layer size then a drop and the curve tapering off after that. Might be that too big of a hidden layer also starts confusion in the classification data.
- Max_iterations: No interesting uptrend or downtrend, again follows the training score. Might suggest that the Neural net already converges before the tested max iteration numbers.

For both datasets, training time is majorly affected by hidden_layer_size, which makes sense as the larger the hidden layer the longer it takes to compute. A bit slower than decision trees, but still respectably fast.

Boosting (adaboost):
Dota2 Dataset:



Unsurprisingly, boosting applied on the decision tree had no effect at all in increasing the accuracies of the model. An inability of the model to classify the classes correctly is not helped at all by boosting. Increasing the number of estimators also practically has no effect at all to the accuracy. Nor does tuning any of the parameters in that matter. Again this just shows how bad decision trees are at solving this specific problem.

Poker Dataset:


```
0     1233
1     1003
2      243
3       22
Name: hand, dtype: int64
             precision    recall  f1-score   support

          0       0.56      0.56      0.56      1252
          1       0.46      0.43      0.45      1075
          2       0.07      0.16      0.10       104
          3       0.00      0.00      0.00        49
          4       0.00      0.00      0.00         8
          5       0.00      0.00      0.00         3
          6       0.00      0.00      0.00         6
          7       0.00      0.00      0.00         2
          8       0.00      0.00      0.00         1
          9       0.00      0.00      0.00         1

avg / total       0.48      0.47      0.48      2501
```

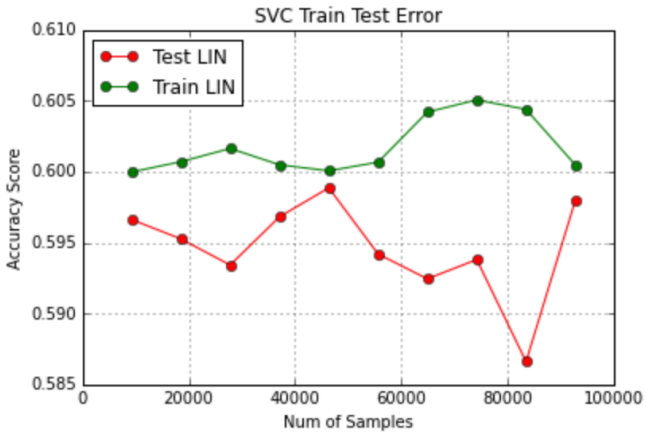
A similar trend with the decision trees, a gradual decrease of accuracy with a spike near the end that bumped up accuracy. A better spread of classifications in that it attempts to classify some as class 2 and class 3, which should be caused by the aspects of boosting itself that focuses on parts where the classifier is doing bad in currently. Comes at an accuracy cost as we can see, as it become less good in predicting correct results in classes 0 and 1 since it likely focused less on these and tried to predict the other classes more.

Effects of parameter tuning:

- N_estimators: the pattern follows a sort of logarithmic curve as the number of estimators increases. With slight variations, it remains true that the more estimators there are, the better the boosting performs.
- Learning rate: caused a gradual almost linear increase in training score, however, test score is unaffected.

General notes about ADABoost is that, unsurprisingly, number of estimators correlate directly with the running time, and that running time in itself is dependent on the type of weak classifier that it uses. So the runtime is similar to the runtime of neural nets.

SVM:
Dota2




```
             precision    recall  f1-score   support

         -1       0.57      0.50      0.54      4791
          1       0.61      0.67      0.64      5502

avg / total       0.59      0.59      0.59     10293

 1    6089
-1    4204
Name: team, dtype: int64
```

SVM's accuracy rate is actually pretty good here, Using the default rbf kernel, it would seem that SVM is able to reach a pretty high accuracy, one that is close to the accuracy of MLP. This shows that SVM might actually be suited for solving this problem due to the fact that we only have to separate between two classes which is something that SVMs are by definition the best at. The only problem is that with this dataset SVM has ridiculously long training times (training takes hours to complete). I wasn't able to tune much really because the kernel function cross validations lasted for 20 hours and nothing really happened. As such, I could only do predictions with rbf kernel and a train test error with a linear kernel. It showed that the linear one seems to perform similar to the rbf, with a minor improvement. This is probably due to the large feature space, causing both kernel functions have a similar performance. A better hand authored kernel function might raise the score.

Poker:


```
0    1850
1     651
Name: hand, dtype: int64
```

	precision	recall	f1-score	support
0	0.57	0.85	0.69	1241
1	0.57	0.35	0.43	1059
2	0.00	0.00	0.00	128
3	0.00	0.00	0.00	52
4	0.00	0.00	0.00	6
5	0.00	0.00	0.00	5
6	0.00	0.00	0.00	8
7	0.00	0.00	0.00	1
8	0.00	0.00	0.00	1
avg / total	0.52	0.57	0.52	2501

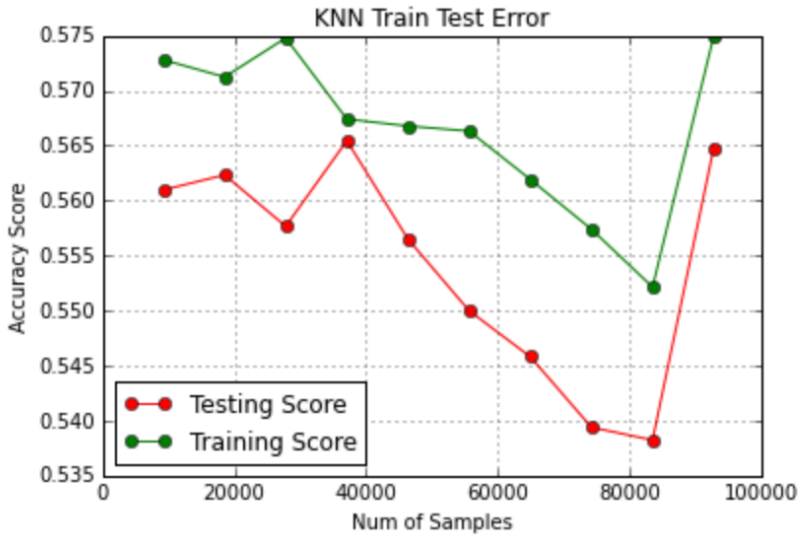
As can be seen, the SVC score is much better with the rbf kernel compared to the linear kernel. This just purely traces back to the fact that rbf kernel is able to wrap around the classes much better than a linear kernel function. Similar to previous datasets, there is again another dip when using 0.8 of the dataset which I'll again attribute to confusion caused by more data. As we can also see, most of the predictions are once again mostly the first two classes. It is expected that an SVM should perform poorly here, due to the fact that the classes have distinct features that has to be learnt, something an SVM is unable to do.

Attempted a C parameter tune which virtually had no effect on the accuracy of the set. The C itself is a penalty parameter to the error term, I guess since the dataset is balanced, changing the C value had little to no effect.

A major issue with SVM is that the training times were exceedingly slow for the Dota2 dataset, reaching up to 20 hours (and still didn't finish) when I tried to benchmark the different kernel functions. I guess this was expected due to the fact that the number of distances between points that needed to be calculated increased significantly as the number of features

increased. It wasn't as bad for the Poker dataset, however for poker it was still significantly slower than the previous 3 classifiers.

KNN:
Dota2:

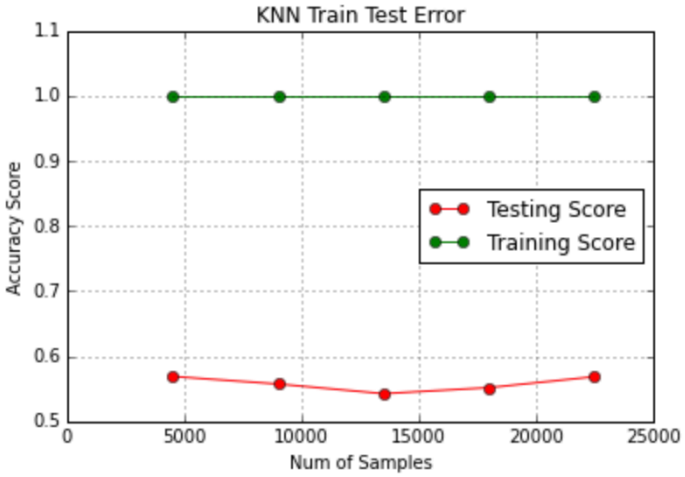


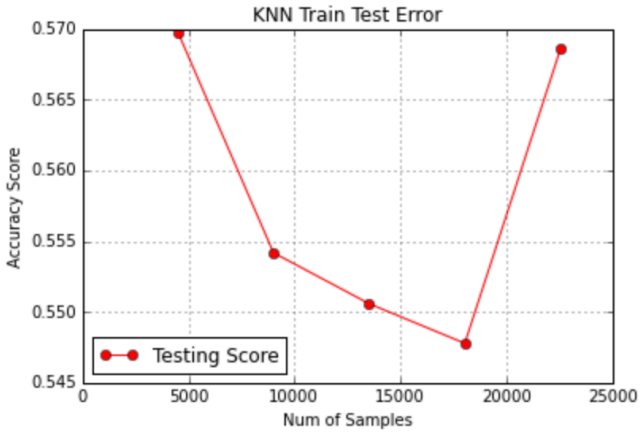
My main analysis of KNN in this context is that it is slow to cross validate. In an attempt to do parameter GridSearch with 5-fold cross validation, for tuning n-neighbors alone took around 5-6 hours on this. Which is kind of expected on this dataset given the feature space (again causing problems), causing the prediction time to take a while due to the amount of data and feature space. Performance of the system however doesn't show anything special. It performs better than most but still has a lower accuracy. However, KNN manages to fit quite well in this problem, it still follows the trend that the other algorithms has been following over time.

One interesting point is that as the number of neighbours increased, the train and cross validation scores converges and looks like a logarithmic curve. As n grows, the closer they are to each other. This may be attributed to the fact that the more neighbours it has to compare with, the closer it is to being more nicely fit with the dataset.

Tuning parameters involved tuning the leaf size which practically had no effect to accuracy as this only possibly affected training time and space usage, which did not feel that big either. Another is the power parameter for the distance metric (Minkowski metric), using Euclidean distance as a metric appeared to yield the best results. This can be explained by the dimensionality of the representation space, Euclidean distance should be able to express the distance better than Manhattan distance.

Poker Dataset:





In poker however, KNN has a perfect training score, but that result has no relation with the test error which seem to just hover in an area. This suggests that the data may just be too hard to evaluate with KNN. An interesting thing is the best accuracy was given to the dataset by an N value of 10. However, this raises the issue of amount of data. As expected, KNN would require n numbers of neighbours to be able to sort of predict how a class looks like. If we look at the dataset distribution, the n values of the last few datasets are even under 10, making it unlikely that KNN would be able predict those classes. So supposedly, KNN here was only able to accurately predict the first few classes, and couldn't do anything for the later cases simply because of the lack of data. Thus in this case, KNN isn't very suitable due to the lack of data. Interestingly however, it was somehow able to get perfect scores in train score which is weird. To me the most likely explanation was that the KNN overfitted to the training data, more tuning to the parameters might be required.

Tuned parameters were n-neighbours which gradually increases accuracy until 100 neighbours, and starts dropping accuracy after that. Likely to be caused by the amount of data in the dataset. This time, the power parameter was the best when using Manhattan distance as a metric. Not sure why this is the case, maybe because the features were more clumped up together and as such, Euclidean might have given same numbers of distance for a lot of them.

Final notes:
For both datasets, it seems that the neural network is the best algorithm to solve these problems, averaging with a better prediction class amounts and higher accuracies on the board, also with rather quick runtimes. The worst for the poker dataset would likely to be KNN due to the fact that it just won't be able to represent the problem unless we have more data. SVM might be solvable using other kernel functions to represent the data. It's harder for the Dota2 dataset however since Decision Trees and its boosted versions practically worked equally bad in classification, however SVM was also ridiculously slow in this implementation, but in terms of possibility of improvement, Decision Trees are just the absolute worst in this case it seems.