4641 Final Project

William Touchstone, wtouchstone3, 903394771

April 2020

1 Introduction to the Data Set

This data set was obtained through UC Irvine's public machine learning data set database. It describes the outcomes of various chess games in a situation where white has a King and a Rook remaining and black has only a king. The 6 attributes are the file (row) and rank (column) of each of the three pieces. The label for these is how many turns it took for white to win (using minimax optimal play). This is calculated up to sixteen moves, however once the number of moves have exceeded this the game is declared a draw.

I find this data set interesting because I am terribly awful at chess. It amazes me the amount of study and technique that has gone into the game, and I thought it would be interesting to investigate this technique in a rather simple scenario.

In order to measure performance I will be using a simple method: simply consider a correct classification correct or incorrect. A sum will be held of correct classifications on test data (where a 1 is correct and a 0 is incorrect) and this sum will be divided by the total number of items to be classified. Throughout this paper I will be discussing score, *not* loss.

2 Description of Algorithms to be Used

1. Random Forests with Bagging: I will be using SKlearn's Bagging Classifier for this. This bagging classifer defaults to a decision tree as its base algorithm. It uses bootstrap samples to create random forests of small decision trees. Being an ensemble method, it will create multiple less accurate predictors and take the "average" of the results for a more accurate result. Bagging, or boostrap aggregation, takes the smaller, simpler trees (which are not very accurate), and tallies the "votes" of all the small trees, and outputs the majority vote as the classification for a certain input. The hyperparameter of interest here will be "n_estimators" - this parameter represents the number of trees in the randomized forest.

2. Support Vector Machine with a non-linear kernel: A support vector machine (with a linear kernel) will try to draw a line to separate the data. This works poorly on non-linearly separable data. To fix this we use a non-linear kernel: essentially a away to transform the data to make it linearly separable. We fit the SVM to it, then apply the inverse of the kernel and we (hopefully) get a good non-linear classifier. The hyperparameters of interest are c (regularization parameter), type of kernel (including RBF, polynomial, and sigmoid), and for polynomial, the degree of the polynomial function used.

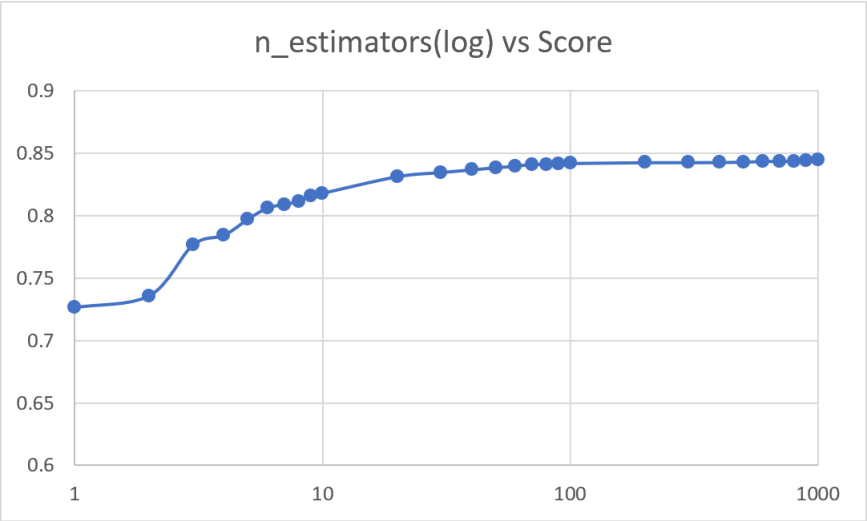
3. Neural Network with 2+ hidden layers: A neural network is a method of learning that uses a model of biological neurons to classify data. The input is sent to a series of neurons on a layer, and then utilizing a weighted sum and an activation function, these neurons will output a certain value to the next layer. On the next layer these inputs will once again be manipulated with a wieghted sum and an activation function and sent to an output. Eventually, depending on the number of layers, there will be a final node that will sum and output the single value for which lable it believes that instance of data should belong to. For this experiment I will be using SKlearn's Neural Network MLPClassifier. This classifier uses backward-propagation to iteratively improve its results. This means that the error whenever the network classifies incorrectly is sent backwards, and the weights in the network are updated. Eventually, we are left with a network that can simply take in inputs and (hopefully) classify it as the correct ouput.

   For this experiment, the hyperparameters I will be varying are the number of hidden layers in the network, and the number of nodes in each hidden layer.

3 Tuning Hyperparamters

Before I jump in, here is my evidence for the fact that this data is not linearly separable. Using a LinearSVC, I tried to classify the data and look at the performance of this method. Attempting to classify the data linearly resulted in a score of 0.1926. I would consider this to be a sufficiently low score for this data to not be linearly separable.

1. Random Forests with Bagging: I tuned across multiple values of n_estimators, ranging from values 1, 2, ... 10, 20, ..., 100, 200, ... 1000. This was done to ensure a small percentage change for value over value, while still accommodating a wide range of values. The graph with the results is shown below:



Random Forests with bagging, n_estimators on log scale vs. Score

As we can see, the chart begins to level off, starting at about n_estimators = 100, and continuing slow leveling off from there. However, n_estimators = 1000 *did* produce the best results, and the training time for it on the data set was relatively insignificant, less than a second, so I will be using n_estimators = 1000 for my comparisons.

2. Support Vector Machine with Non-Linear Kernels: The hyperparameters I was interested were the type of kernel, the degree of the polynomial (if the polynomial kernel was selected) and the slack (C) value. Once again, these comparisons were completed with a GridSearch cross-validation

These charts are in groups of C values, in order of C = 1, 5, 10, 50, 100, 500, 1000

Degree	RBF	Polynomial	Sigmoid
2	0.52873828	0.42314235	0.09811226
3	0.52873828	0.46729695	0.09811226
4	0.52873828	0.42554796	0.09811226
5	0.52873828	0.36807134	0.09811226
6	0.52873828	0.35608708	0.09811226
7	0.52873828	0.33906747	0.09811226
8	0.52873828	0.32824055	0.09811226
9	0.52873828	0.3056492	0.09811226

Degree	RBF	Polynomial	Sigmoid
2	0.52873828	0.42314235	0.09811226
3	0.52873828	0.46729695	0.09811226
4	0.52873828	0.42554796	0.09811226
5	0.52873828	0.36807134	0.09811226
6	0.52873828	0.35608708	0.09811226
7	0.52873828	0.33906747	0.09811226
8	0.52873828	0.32824055	0.09811226
9	0.52873828	0.3056492	0.09811226

Degree	RBF	Polynomial	Sigmoid
2	0.59249706	0.42840006	0.09873598
3	0.59249706	0.3934676	0.09873598
4	0.59249706	0.35203213	0.09873598
5	0.59249706	0.34227383	0.09873598
6	0.59249706	0.33314024	0.09873598
7	0.59249706	0.32565541	0.09873598
8	0.59249706	0.32507572	0.09873598
9	0.59249706	0.3240508	0.09873598

Degree	RBF	Polynomial	Sigmoid
2	0.6165568	0.39195341	0.09895876
3	0.6165568	0.35385848	0.09895876
4	0.6165568	0.33380785	0.09895876
5	0.6165568	0.33149219	0.09895876
6	0.6165568	0.32195756	0.09895876
7	0.6165568	0.32480775	0.09895876
8	0.6165568	0.32119993	0.09895876
9	0.6165568	0.31496085	0.09895876

Degree	RBF	Polynomial	Sigmoid
2	0.64319991	0.31447182	0.09886967
3	0.64319991	0.31424938	0.09886967
4	0.64319991	0.32708145	0.09886967
5	0.64319991	0.32164519	0.09886967
6	0.64319991	0.29740544	0.09886967
7	0.64319991	0.30925869	0.09886967
8	0.64319991	0.30520446	0.09886967
9	0.64319991	0.30533686	0.09886967

Degree	RBF	Polynomial	Sigmoid
2	0.61214589	0.29348717	0.09895875
3	0.61214589	0.29562424	0.09895875
4	0.61214589	0.30573806	0.09895875
5	0.61214589	0.307699	0.09895875
6	0.61214589	0.29892138	0.09895875
7	0.61214589	0.31870475	0.09895875
8	0.61214589	0.31104079	0.09895875
9	0.61214589	0.318659	0.09895875

Degree	RBF	Polynomial	Sigmoid
2	0.53956457	0.20976614	0.09900331
3	0.53956457	0.2639454	0.09900331
4	0.53956457	0.28510964	0.09900331
5	0.53956457	0.30765551	0.09900331
6	0.53956457	0.30676415	0.09900331
7	0.53956457	0.3171002	0.09900331
8	0.53956457	0.3081003	0.09900331
9	0.53956457	0.31148622	0.09900331

Degree	RBF	Polynomial	Sigmoid
2	0.51688671	0.20829588	0.09904786
3	0.51688671	0.26407907	0.09904786
4	0.51688671	0.28854104	0.09904786
5	0.51688671	0.30409051	0.09904786
6	0.51688671	0.30150631	0.09904786
7	0.51688671	0.29700521	0.09904786
8	0.51688671	0.31857079	0.09904786
9	0.51688671	0.30569461	0.09904786

As seen in the charts, the best solution is an RBF kernel with a C value of 100. This is what I will be using for my comparisons. As can also be seen in the charts, the degree does not effect the RBF or Sigmoid kernels. This degree variable is strictly used for the polynomial kernels. Because of its performance I will be using an RBF kernel with a C value of 100 for my comparisons between algorithms.

3. Neural Network: The hyperparameters I chose to vary here were the number of hidden layers and the number of nodes in each of the layers (NPL). The following table shows the score for this GridSearch.

Number of layers	NPL = 2	NPL = 5	NPL = 10	NPL = 20	NPL = 200	NPL = 500
3	0.22419544	0.34700041	0.41187105	0.49652589	0.64373646	0.65585497
4	0.22116244	0.31441041	0.40794599	0.49652938	0.65228737	0.68098311

As can be seen, a lower number of nodes per layer actually works better when the number of layers is 3 instead of 4. This trend reverses as the nodes per layer increases past 20. Due to computational difficulty on my end, I couldn't test more than this set of hyperparameters, and I will be using 4 hidden layers of 500 nodes for the comparison. I also changed the max_iters variable, since at its default value the neural network failed to converge.

4 Comparing Algorithm Performance

Algorithm	Mean Performance	Standard Deviation of Score	95% Confidence Interval
Random Forest, Bagging	0.8102	0.0192	[0.7784, 0.8420]
SVM with RBF Kernel	0.6661	0.0291	[0.6186, 0.7154]
Neural Network	0.8329	0.0228	[0.7947, 0.8711]

Observations: On this dataset, a SVM utilizing an RBF kernel with a slack value of 100 results in the worst performance of the algorithms tested. Utilizing a 95% confidence interval, it can be established that the difference between the SVM and other two algorithms *is* statistically significant enough to be considered as performing worse. This, however, is not the case when comparing the Bagging and Neural Network algorithms. The two had mostly similar means, and both means were withing the other's 95% confidence interval. From this we cannot say there is a statistically significant difference in the performance of these models.

However, in terms of time efficiency, Random Forests with Bagging performed *significantly* better. Bagging ran in under 5 seconds, whereas training the network took tens of minutes to complete.

5 Conclusion

I believe with the non-statistically significant differences between the performance of the algorithms, Random Forests with Bagging is a better option for classifying items in the same domain as the dataset. It trains significantly faster, and performs rather comparably. Also of note is that the Random Forest with bagging produced a set of predictions that had a lower variability (that is to say lower standard deviation across folds) than the neural network, though not by much. Regardless, I would consider this a trait that goes in bagging's favor.

6 Acknowledgements

The libraries I used: SKlearn, numpy

For the most part all I used on this assignment was the SkLearn API. It contained most of the information I needed. There were also points where I went back and looked at the lecture slides. I also HEAVILY utilized overleaf's LaTeXhelp pages, so I hope that this document has been very easy to read and well laid out.

Since this is the final assignment, I'd just like to say thanks to Dr. Hrolenok and all the TAs. I've had Dr. Hrolenok for two semesters in a row (3600 and 4641) and I've really enjoyed being in both the classes. Also shoutout to the TAs staying active on Piazza, especially Max. I don't know how terribly I'd have done in this class if it weren't for you guys' help on Piazza.