Analysis and Prediction of Poker Hand Strength

Sahir Noor Ali - K132047

1. Setting the Research Goal

We have selected the UCI Poker Hand Dataset as our Data Science Project Data. The training and testing data can be found here [1].

The first work related to a similar data set was done by Cartel et al. [2] by designing a data mining system that utilizes machine learning algorithms and extends it to learn strong rules in order to classify hard data sets like Poker Hand data set. Dişken [3] in his work has also used the same data set but have presented the use of artificial neural networks to predict the poker hand strength. A tutorial utilizing Java Neural Network Framework (Neuroph) to classify this multivariate data set is given here [4].

The main goal of this research project is to analyze the poker hands and predict the strength of the given hands (each instance of the data set) by using Neural Networks. This data set has been categorized, by the authors of the above mentioned researches, as a hard data set for classification. Thus, our main objective is to make the use of neural networks to make the machine learn from the information of the provided 5 cards in hand (each color suit and the card in hand) and improve its predictability on an unseen data.

2. Retrieving Data

The data retrieval process of the Data Science process for us was relatively easier as the Poker Hand data set is publicly available via UCI Machine Learning Repository website specifically at [1]. The training data is only 600KBs while the testing data is of 24MBs in size. The description of the data set is clearly written and specifies each attribute and its functioning in a very clear manner. The description also states that it's a challenging/hard data set for classification algorithms. Moreover, it presents a statistical overview of the distribution of the classes in the training data set.

3. Data Preparation

The data set is described in the Figure 7 (appendix)**.** It contains no missing values thus, we can take a sigh of relief. We thought at first to reduce the number of features from the dataset and thus, we applied RandomForestClassifier and printed out the feature importance. The following result was yield:

```
[ 0.07137824  0.1326106   0.06795146  0.13056928  0.06787164  0.1429566
  0.07011656  0.12741501  0.05694591  0.13218469]
```

Figure 1: Feature Importance

We can clearly see that each feature is important and at least contributes more than 5% to the result. This totally relates to the practical scenario of poker where we cannot remove a card or ignore its suit when we try to determine the strength of the hand. Thus, we must not reduce the number of features.

However, for us to apply the MLPClassifier, we must scale the data as the classifier is very sensitive to feature scaling. For this reason, we have applied the StandardScalar preprocessing [5] on our data set. And if we look closely, the even columns (color suits) have a range of values (1-4) whereas, the odd columns have a range of values (1-13) thus, we are better off with scaling the data. We first fit the training data via StandardScalar and then we transform the training and testing data.

4. Data Exploration

It's time to explore our data set. The table below describes the head (first 5 samples) of the training data:

	0	1	2	3	4	5	6	7	8	9	10
0	1	10	1	11	1	13	1	12	1	1	9
1	2	11	2	13	2	10	2	12	2	1	9
2	3	12	3	11	3	13	3	10	3	1	9
3	4	10	4	11	4	1	4	13	4	12	9
4	4	1	4	13	4	12	4	11	4	10	9

Table 1: First 5 Samples of the Training Data

Each pair of consecutive column denotes the suit color (1: Hearts, 2: Spades, 3: Diamonds and 4: Clubs) and the rank (1: Ace, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: Jack, 12: Queen and 13: King) of single card in hand. The last column is the class label or strength of the poker hand (0: Nothing, 1: 1 Pair, 2: 2 Pairs, 3: 3 of a Kind, 4: Straight, 5: Flush, 6: Full House, 7: 4 of a Kind, 8: Straight Flush, and 9: Royal Flush).

The training set contains 12493, 10599, 1206, 513, 93, 54, 36, 6, 5, 5 samples for classes 0 to 9 respectively. That is a total of 25010 samples. A histogram to show this distribution is given below:

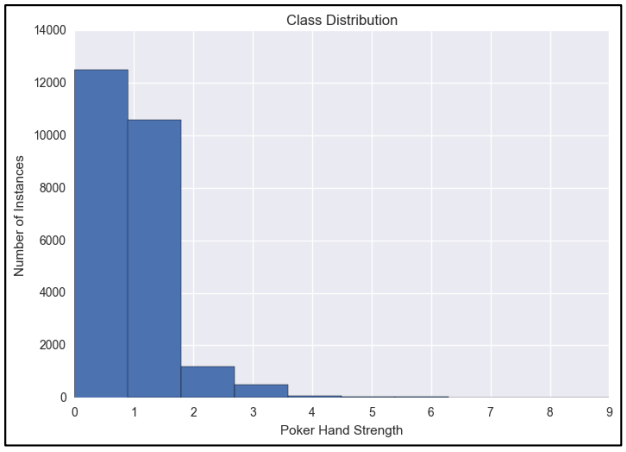

Figure 2: Histogram to show the Class Distribution of Training Data

We can note here that the last three classes are invisible in the histogram above, it is because they cannot be fitted on the scale of thousands yet we know that they exist under a unit of tens.

The testing set contains 501209, 422498, 47622, 21121, 3885, 1996, 1424, 230, 12 and 3 samples for classes 0 to 9 respectively. That is a total of 1,000,000 testing samples. A histogram to show this distribution is given below:

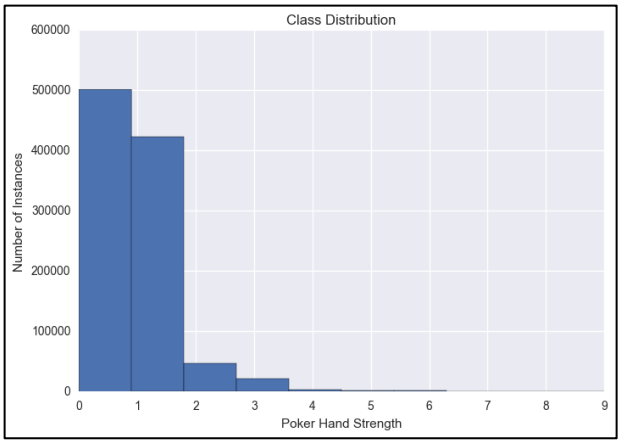

Figure 3: Histogram to show the Class Distribution of Testing Data

As we can see the distribution is very similar to the training data yet contains huge number of samples. The joint plot of the training data in the Figure 8 (appendix) illustrates that there exists no or little relationship among the features and thus, useless to visualize them. This also gives us the insight that the data is not linear and thus, Neural Networks works tremendously well in approximating a nonlinear function.

5. Data Modeling

Data modeling is an iterative step in the Data Science process. Here we explain the three major components of the Data Modeling step which are Model and Variable Selection, Model execution, Model diagnostic and model comparison.

5.1 Model and Variable Selection

We have selected all the variables, as explained in the Data Preparation step, as each attribute contributes to the prediction because the order of the cards in the hand

and the color/suit of each is heavily dependent on the strength of the hand and thus, cannot be removed.

We have selected the Multi-Layer Perceptron Classifier (a feed forward Neural Network), which is a standard neural network algorithm for classification problems, as our predictive model which has stood out immensely well as compared to traditional classification methods and has outperformed them all (explained in model comparison).

5.2 Model Execution

We have executed the MLPClassifier that belongs to the sklearn.neural- network class [6]. The Multi-Layer Perceptron uses (at least) three layers underneath which are the input, hidden and output layers. The hidden layers are user customizable and for each hidden layer specified, the user can also select the number of neurons to put in that respective layer. This is the part where we use the trial and error approach to converge to good results.

We have used two hidden layers where the first hidden layer contains 64 neurons and the second hidden layer also contains 64 neurons. The input features (10) are equal to the number of neurons in the input layer and as well as in the output layer. Thus, our neural architecture can be put in a tuple as [10, 64, 64, 10]. We have made sure that our model doesn't over fit the data by using the following equation:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Figure 4: Upper Bound for the Hidden Layer Neurons [7]

Where N_s is the number of training samples (25010 in our case), N_i is the number of neurons in the input layer (10 in our case), N_o is the number of neurons in the output layer (10 in our case) and α is a random scaling constant (values from 2-10). Thus, if we take 5 as the scaling constant, we get 250 neurons for the hidden layer N_h. But since we did not get good results in a single layer, we utilized two layers with 64 neurons each. Which gives a total of 128 neurons that is far less than 250 neurons. Thus, ensuring that our model doesn't over fit the data.

The MLPClassifier is executed with following settings: The solver used is 'adam', alpha is kept at 10^{-5} , hidden layer sizes: (64,64) that is 2 layers with 64 neurons each, activation function 'tanh', constant learning rate of 0.02, maximum iterations: 2000 and the model is executed 5 times for randomization (random seed).

The alpha is kept low so as to assign less penalty so that the class samples, which are rare in the data, can be retained in the output and can't be ignored via weights. The adam solver is used which is a gradient based stochastic optimizer and it has an ability to converge faster in the large data sets. Two sets of hidden layer are used to approximate a nonlinear function as a single layer cannot approximate this data set very well. The number of neurons in each layer is restricted as discussed earlier to avoid overfitting. Tanh activation function is used for its ability of not getting stuck in the training for longer and its output values range between -1 and 1 thus, it maps really well with the scaled data set and properly take every class into account. The learning rate was kept a bit higher than default so as to increase the step-size and by increasing the default maximum iterations, we were able to get very good results. Thus, selecting the hidden layers, specifying its neurons, setting up the number of iterations and the learning rate are the major factors that we have tuned to achieve 96% (mean) accuracy.

5.3 Model Diagnostic and Comparison

We have increased the overall accuracy achieved for this data set. Dişken [3] was able to achieve 92.4% accuracy with 50 neurons and 2000 iterations yet we have successfully achieved 96% mean accuracy with our proposed neural network architecture with 2000 iterations. We present the comparision of our used model with other traditional classification models in the table below:

S.No	Classifier	Accuracy
1	Bagging	59%
2	Random Forest	57%
3	AdaBoost	49%
4	KNN	55%
5	Naïve Bayes	50%
6	Decision Tree	48%
7	Linear SVM	50%
8	OutputCodeClassifier with Linear SVM	60%
9	OneVsAll with Linear SVM	TLE
10	Multi-Layer Perceptron	96%

Table 2: Model Comparison

All models in the table above except the MLP are used in their default setting as provided by sklearn library [8]. The Gaussian version of Naïve Bayes is used and the OneVsAllClassifier took hours yet couldn't converge to the result (TLE - Time Limit Exceed). The OutputCodeClassifer actually makes a binary code book for each class and runs the given model for each class while this is not the case in OneVsAllClassifier. As we can see the traditional models have failed to map good results for this data set, this is the reason we opted for a Neural Network.

A confusion matrix and classification report of the executed model with a random seed of 5 are given below:

```
[[497014   2867      0      0    532    713      0      0     78      5]
 [  9901 411016   1114     81    150    113      1      0    109     13]
 [     2   2654  42416   2517      2      0     19     12      0      0]
 [     0     44   1948  18989      0      0     55     85      0      0]
 [  2979    121      0      0    750      8      0      0     26      1]
 [  1798      8      0      0      4    179      0      0      3      4]
 [     0      0     26    981      0      0    405     12      0      0]
 [     0      0      0    160      0      0     33     37      0      0]
 [     3      0      0      0      6      2      0      0      1      0]
 [     1      0      0      0      0      1      0      0      1      0]]
```

Figure 5: Confusion Matrix

	precision	recall	f1-score	support
0	0.97	0.99	0.98	501209
1	0.99	0.97	0.98	422498
2	0.93	0.89	0.91	47622
3	0.84	0.90	0.87	21121
4	0.52	0.19	0.28	3885
5	0.18	0.09	0.12	1996
6	0.79	0.28	0.42	1424
7	0.25	0.16	0.20	230
8	0.00	0.08	0.01	12
9	0.00	0.00	0.00	3
avg / total	0.97	0.97	0.97	1000000

Figure 6: Classification Report

.

It can be seen that the model works outstandingly well for most of the major classes. Yet it fails to predict the last rare classes correctly. Due to its great results for the major classes, it gives out a great accuracy. A future improvement to this research can involve dealing with these rare classes and devising a methodology to learn better for these rare instances.

6. Presentation & Automation

The code written for this research is very well formatted and commented. The code is tidy and readable for everyone. There is only a single code file for the data preparation, model execution and the model comparison which applies good coding tactics to present everything in a short and concise manner.

References

[1] "Poker Hand Data Set," UCI Machine Learning Repository, [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Poker+Hand.

[2] F. O. D. D. ROBERT CATTRAL, "Evolutionary Data Mining With Automatic Rule Generalization," *Recent Advances in Computers, Computing and Communications,* pp. 296-300, 2002.

[3] G. Dişken, "PREDICTING POKER HAND'S STRENGTH WITH ARTIFICIAL NEURAL NETWORKS".

[4] N. Ivanić, "PREDICTING POKER HANDS WITH NEURAL NETWORKS," Neuroph, [Online]. Available: http://neuroph.sourceforge.net/tutorials/PredictingPokerhands/Predicting%20poker%20hands%20with%20neural%20networks.htm.

[5] "Standard Scalar," [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[6] "sklearn.neural_network.MLPClassifier," scikit-learn, [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.

[7] "How to choose the number of hidden layers and nodes in a feedforward neural network?," Stack Exchange, [Online]. Available: https://stats.stackexchange.com/a/136542.

[8] "Scikit Learn Home," [Online]. Available: http://scikit-learn.org/stable/.

Appendix

	0	1	2	3	4	5	6	7	8	9	10
count	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000	25010.000000
mean	2.506756	6.995242	2.497721	7.014194	2.510236	7.014154	2.495922	6.942463	2.497321	6.962735	0.621152
std	1.116483	3.749805	1.121767	3.766974	1.123148	3.744974	1.116009	3.747147	1.118732	3.741579	0.788361
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	2.000000	4.000000	1.000000	4.000000	2.000000	4.000000	1.000000	4.000000	1.000000	4.000000	0.000000
50%	3.000000	7.000000	2.000000	7.000000	3.000000	7.000000	2.000000	7.000000	3.000000	7.000000	1.000000
75%	4.000000	10.000000	4.000000	10.000000	4.000000	10.000000	3.000000	10.000000	3.000000	10.000000	1.000000
max	4.000000	13.000000	4.000000	13.000000	4.000000	13.000000	4.000000	13.000000	4.000000	13.000000	9.000000

Figure 7: Data Set Description

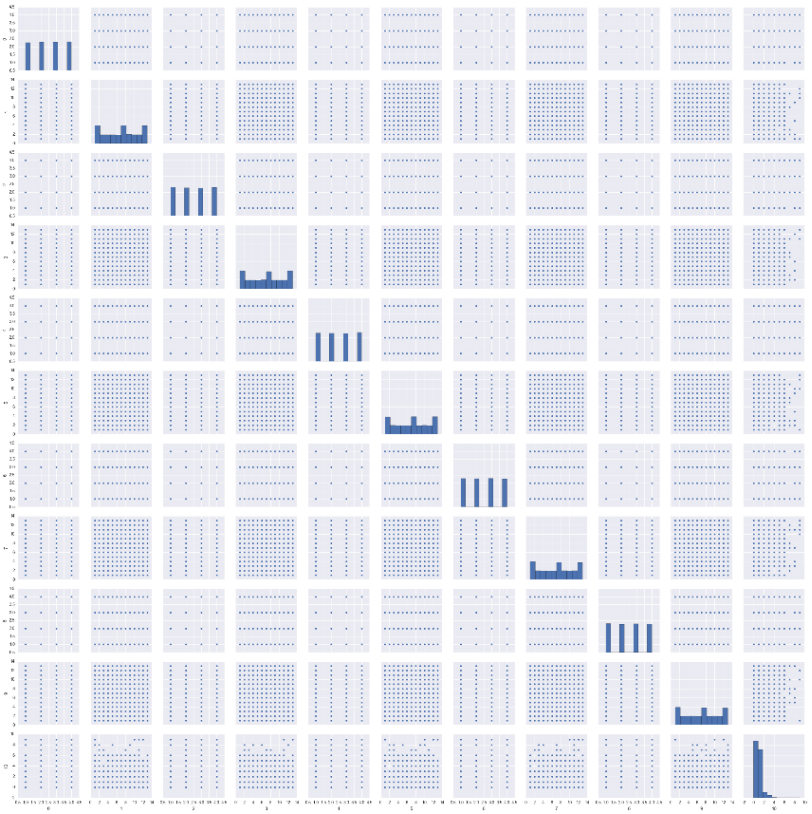

Figure 8: Pair Plot