CS178 Final Project

Predicting Poker Hand

Wenhan Kong,

William Sun,

Wei Qiu,

Xuhui Zhu

1. Problem Statement

According to the American Game Association, USA gambling revenues reach up to $150 billion every year. In five American adults , there is one person is addicted to gambling. The 2017 gross revenues of Poker rooms in the state was $117.7 billions. Texas Hold' em poker is the most popular poker game of all poker variations. This complex game requires grambler having amount of skill and strategy. People wonder if the gambling depends on luck or skill? It turns out that actually the gambling is not heavily depend on luck. There is a scientific measure can be used here which is machine learning of predicting poker hands. Texas is played by dealing each player 2 cards (face down), called the hole cards, and dealing 5 community cards (face up), on the table. The player makes a poker hand using any combination of the 3 cards dealt to them, and the 5 cards on the table. The player with the best combination of five cards wins the pot. The order of poker hands are ranked from the lowest possibility -- nothing in hand which means five unmatched cards, to the highest possibility -- royal flush which means five cards of same suit, ranked from ace through ten. The strength of a hand can be categorized in to ten classes, which also map to our data predictive classes. In this project, we will use the materials we have learned to analyze how to predict the poker hands by the given cards.

2. Data Set

In the UCI Machine Learning Dataset, we found this Poker Hand Dataset as our final project data. There are totally 25010 training data and 1000000 testing data in the dataset.

Each record is an example of a hand consisting of five playing poker cards drawn from a standard deck which contains 52 cards. The order of cards is important, which is why there are 480 possible Royal Flush hands as compared to 4 (one for each suit explained in more detail below). [1] This dataset was first released in 2007, it was considered to be generated with intention that it be difficult to discover the underlying concepts yet easy to analyze them. Because of the way the problem is represented, it is difficult to discover rules that can correctly classify poker hands.[2] Thus we take the advantages of this particular dataset for analysis to gain a better insight into how various machine learning algorithms actually work.

Feature

There are eleven attributes for each record. Every card can be distinguished by their suit and rank so that there are ten predictive attributes for suit and rank of five predictive cards totally. The last attribute describes the “ Poker Hand”. (**Figure 1**)

Attribute	Information
1	Suit of card 1 Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
2	Rank of card 1 Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
3	Suit of card 2 Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
4	Rank of card 3 Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
5	Suit of card 3 Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
6	Rank of card 4 Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
7	Suit of card 4 Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
8	Rank of card 5 Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
9	Suit of card 5 Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
10	Rank of card 2 Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
11	Class “Poker Hand” Ordinal (0-9) 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

(Figure 1):Table of Attribute Information

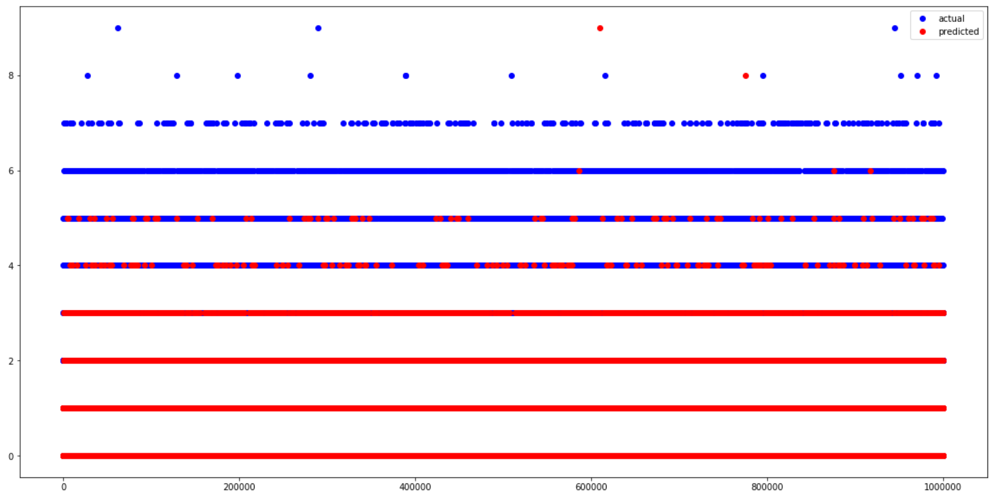
3. Experience

1) Baseline

In this project we choose dummy estimator in scikit-learn library that belongs to sklearn.dummy class.[3] A dummy estimator is implemented with several simple strategies for classification. Here we set "most_frequence" which always predicts the most frequent label in training as the first parameter. Unsurprisingly the dummy classifier gives a unremarkable result. The accuracy score provided by DummyClassifier.score() is only 0.51209. And the confusion matrix shows that almost all labels predicted are class 0 (nothing in hand). An algorithm with lower predictive accuracy always choose the majority, while a higher score indicates that some correct choices are made during prediction. Therefore, the goal of our chosen algorithms is to achieve at least 51.2% accuracy.

2) Linear Regression

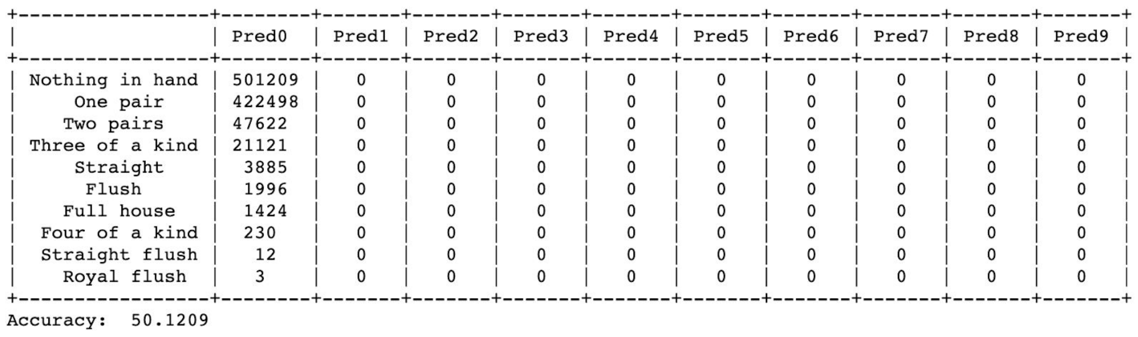
We take the first attempt of testing using linear regression. By training scikit-learn library's class linear_model.LinearRegression()[4], we find that the regression line lies mainly on the majority class 0 (nothing in hand) in test data space. With this observation, we have a chance to peek at the data distribution of poker hand dataset. As shown in the figure(3), most classes players held are class 0(nothing in hand) and class 1(one pair). Therefore regression model is almost impossible to establish connection with rare cases such as class 9(royal flush), which only occurs 3 times out of 1,000,000 instances. In the final result, linear regression gives 37.9% accuracy in training data, and 40.2% accuracy in test data. Since the score is far behind the baseline, linear regression in this case is considered to be a failure.



(Figure 3) Linear Regression: Predicted vs Actual

3) Logistic Regression

Then, we use Logistic Regression as the second attempt. However, the result is still disappointing, the prediction totally lies in class 0 (nothing in hand) in the test data space. The accuracy of training data is 61.6902, and accuracy of testing data is only 50.12%(Figure 4). As we can see from the figure, most players held are class 0 (nothing in hand) and class 1 (one pair). Therefore the regression model gets rid of the possibility of rare cases like class 9 (royal flush). This is similar to Linear Regression, which we need more advance model to achieve higher accuracy.



```
+-----------------+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|                 | Pred0  | Pred1 | Pred2 | Pred3 | Pred4 | Pred5 | Pred6 | Pred7 | Pred8 | Pred9 |
+-----------------+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
| Nothing in hand | 501209 |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|    One pair     | 422498 |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|    Two pairs    | 47622  |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
| Three of a kind | 21121  |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|    Straight     |  3885  |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|      Flush      |  1996  |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|   Full house    |  1424  |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|  Four of a kind |  230   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|  Straight flush |   12   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
|   Royal flush   |   3    |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |   0   |
+-----------------+--------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
Accuracy:  50.1209
```

(Figure 4): Logistic Regression:Predicted table

4) SVM

The Support Vector Machine is a geometrically motivated algorithm. We are trying to find a particular optimal separating hyperplane, where we define “optimal” in the context of support vectors of poker hand instances. In Scikit-learn library, svm.SVC(probability=”True”)[5] is used, for svm and following models, probability attribute is set to be True for later “area under curve” calculation. Also, the default linear kernel is used in our first attempt. It is worth mentioning that linear SVM works slowly in large sample data, since the distance between each vectors needs to be calculated. According to classification report, linear svm only predicts correctly on class 0(nothing in hand) and class 1 (one pair) with 67% f1-score and 41% f1-score respectively. Notably, SVM don’t penalize examples for which the correct decision is made with sufficient confidence. [6] This may be good for generalization, as shown in the graph, but performs poorly on predicting rare instance. The final accuracy score is 55.7%, which is slightly higher than the baseline. The result of linear SVM indicates that poker hand dataset is not linear separable, therefore more complex algorithms need to be applied.

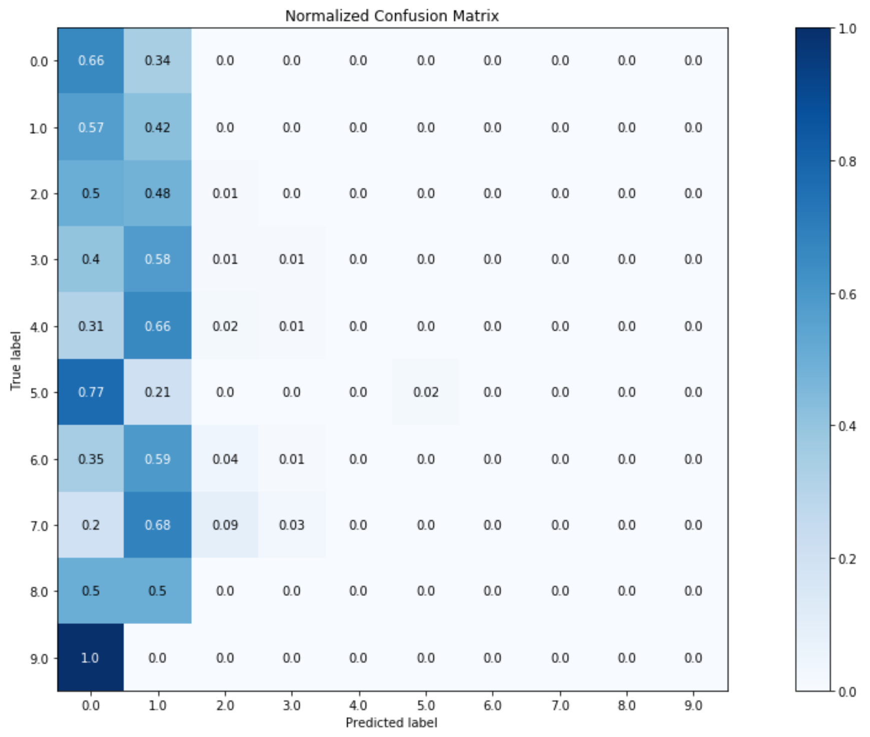
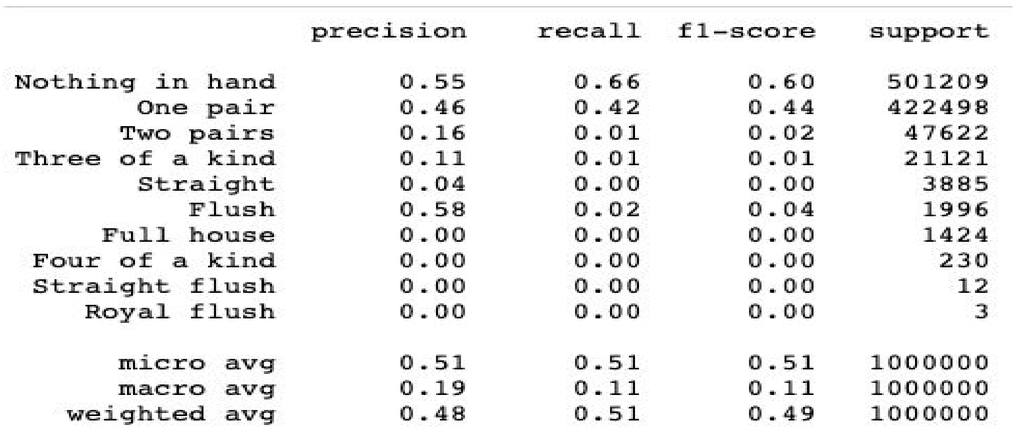


Figure 5: SVM Normalized Matrix



```
                  precision    recall  f1-score   support

  Nothing in hand       0.55      0.66      0.60    501209
         One pair       0.46      0.42      0.44    422498
        Two pairs       0.16      0.01      0.02     47622
  Three of a kind       0.11      0.01      0.01     21121
         Straight       0.04      0.00      0.00      3885
            Flush       0.58      0.02      0.04      1996
       Full house       0.00      0.00      0.00      1424
   Four of a kind       0.00      0.00      0.00       230
   Straight flush       0.00      0.00      0.00        12
      Royal flush       0.00      0.00      0.00         3

        micro avg       0.51      0.51      0.51   1000000
        macro avg       0.19      0.11      0.11   1000000
     weighted avg       0.48      0.51      0.49   1000000
```

Figure 6: SVM

5) KNN

For quicker training time, we choose KNN to deal with the nonlinear dataset. KNeighborsClassifier()[7] in sklearn.neighbors is used. The number of neighbors is set to default 5. The prediction result shows that KNN also suffers from high dimensionality of data. Our theory is that the computations of distance metric in KNN dramatically increase with the dimension of data points. The final result is 51.2% on accuracy.

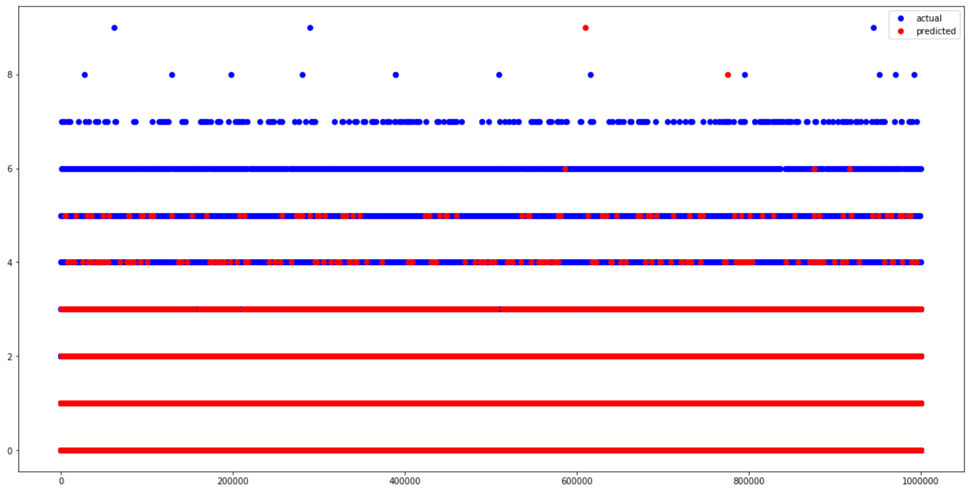
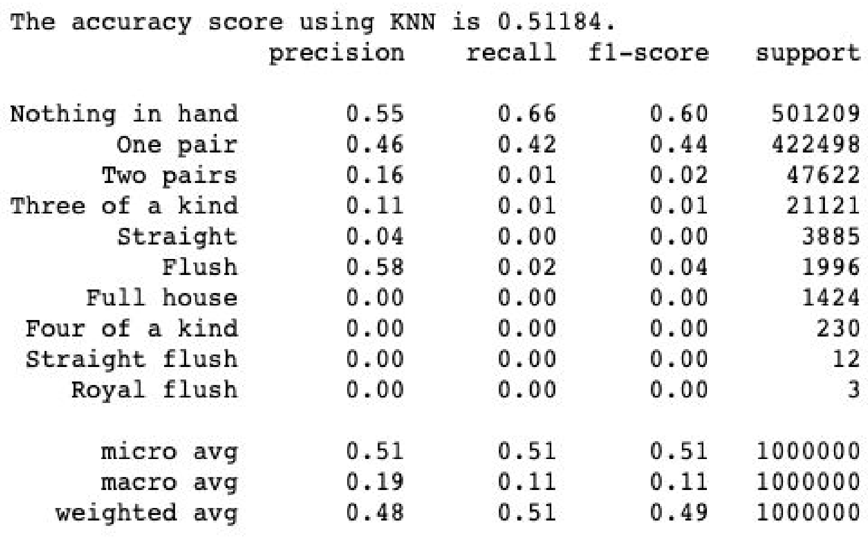


Figure 7: KNN: Predicted vs Actual



```
The accuracy score using KNN is 0.51184.
                 precision    recall  f1-score   support

Nothing in hand       0.55      0.66      0.60    501209
       One pair       0.46      0.42      0.44    422498
      Two pairs       0.16      0.01      0.02     47622
Three of a kind       0.11      0.01      0.01     21121
       Straight       0.04      0.00      0.00      3885
          Flush       0.58      0.02      0.04      1996
     Full house       0.00      0.00      0.00      1424
 Four of a kind       0.00      0.00      0.00       230
 Straight flush       0.00      0.00      0.00        12
    Royal flush       0.00      0.00      0.00         3

      micro avg       0.51      0.51      0.51   1000000
      macro avg       0.19      0.11      0.11   1000000
   weighted avg       0.48      0.51      0.49   1000000
```

Figure 8:KNN Accuracy Table

6) Decision Tree

Decision tree is a non-parametric supervised learning method. The Scikit-learn library, tree.DecisionTreeClassifier(maxt_depth = n), is used.[8] We try to find a proper depth for our tree. After analyzing we choose 7 as max depth for our decision tree classifier. As shown in the figure 9, the tree starts overfitting when max depth is bigger than 7. The decision tree model has a decent accuracy on class 0(Nothing in hand), class 1(One pair) and class 5(Flush). With 63%, 41% and 32% fl-score respectively. The final accuracy score is 52.7%, which indicates a more precise algorithm is needed.

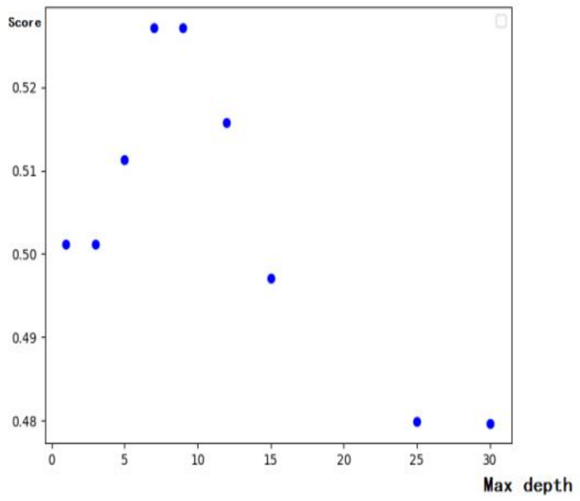
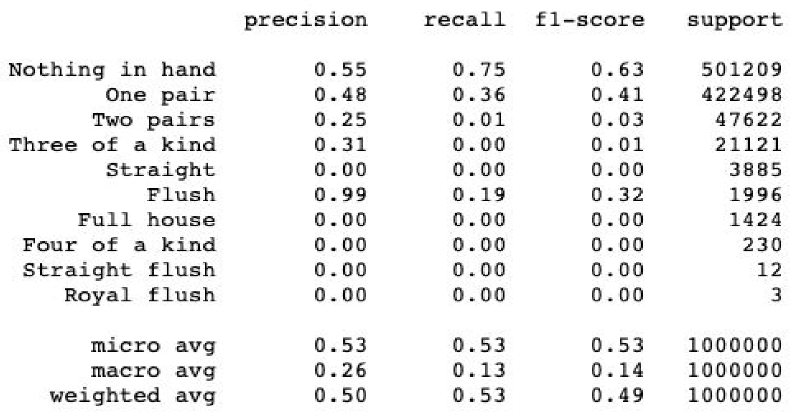


Figure 9: Decision Tree Score and Max depth

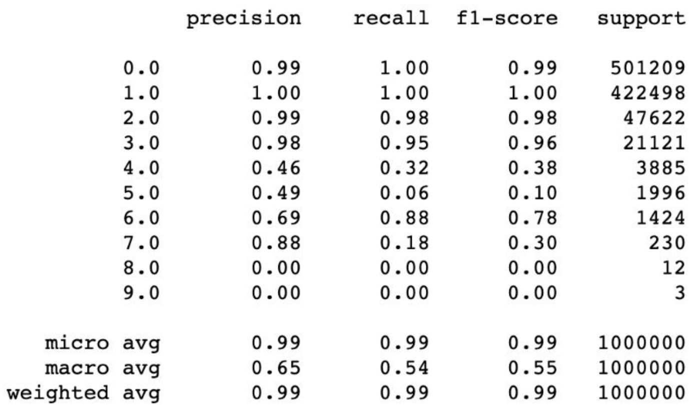


```
                  precision    recall  f1-score   support

 Nothing in hand       0.55      0.75      0.63    501209
        One pair       0.48      0.36      0.41    422498
       Two pairs       0.25      0.01      0.03     47622
 Three of a kind       0.31      0.00      0.01     21121
        Straight       0.00      0.00      0.00      3885
           Flush       0.99      0.19      0.32      1996
      Full house       0.00      0.00      0.00      1424
  Four of a kind       0.00      0.00      0.00       230
  Straight flush       0.00      0.00      0.00        12
     Royal flush       0.00      0.00      0.00         3

       micro avg       0.53      0.53      0.53   1000000
       macro avg       0.26      0.13      0.14   1000000
    weighted avg       0.50      0.53      0.49   1000000
```

Figure 10: Decision Tree Accuracy Table

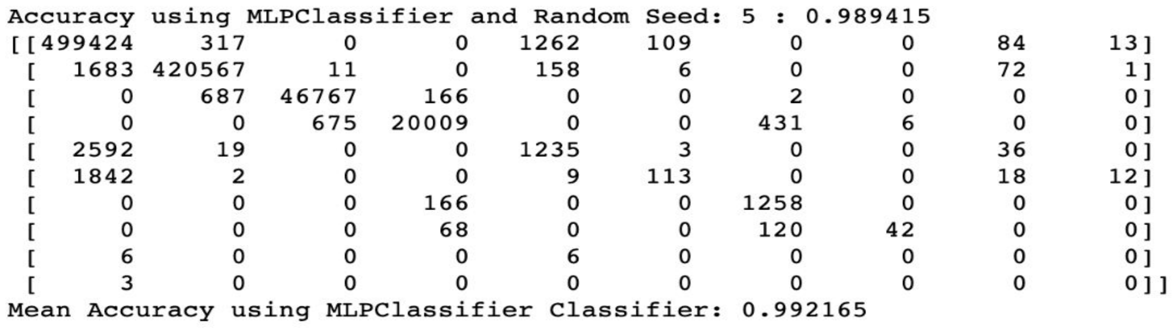
7) Multilayer Perceptron

Multilayer Perceptron (MLP) is a class of feedforward artificial neural network. MLP utilizes a “supervised learning technique called backpropagation for training.”[9] This model optimizes the log-loss function using LBFGS or stochastic gradient descent.[10] This model’s multiple layers and non-linear activation distinguish from a linear Perceptron model. Therefore, MLP model is not linearly separable. We use five different random seeds (1 - 5) to apply the model. The mean accuracy of five times training gives us 99.22% accuracy. The MLP classifier is setted as MLPClassifier(solver = 'adam', alpha = 1e-5, hidden_layer_sizes = (64, 64), activation = 'tanh', learning_rate_init = 0.02, max_iter = 2000, random_state = s). We kept alpha low in order to assign less penalty, therefore, the class samples can be retained in the output and won’t be ignored by weight. We used adam as solver which is a gradient-based stochastic optimizer, and it is able to converge large data set faster. We choose tanh as activation function so that the training process will not get stuck for long. Moreover, the output range between -1 and 1. With these setting, the learning rate achieves such a high score. What’s more, we can see the model predicts six class 7 with accuracy 88% even though it only appears few times(Figure 11 & 12).



```
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99    501209
         1.0       1.00      1.00      1.00    422498
         2.0       0.99      0.98      0.98     47622
         3.0       0.98      0.95      0.96     21121
         4.0       0.46      0.32      0.38      3885
         5.0       0.49      0.06      0.10      1996
         6.0       0.69      0.88      0.78      1424
         7.0       0.88      0.18      0.30       230
         8.0       0.00      0.00      0.00        12
         9.0       0.00      0.00      0.00         3

   micro avg       0.99      0.99      0.99   1000000
   macro avg       0.65      0.54      0.55   1000000
weighted avg       0.99      0.99      0.99   1000000
```

Figure 11: Multilayer Perceptron Confusion Matrix



```
Accuracy using MLPClassifier and Random Seed: 5 : 0.989415
[[499424    317      0      0   1262    109      0      0     84     13]
 [  1683 420567     11      0    158      6      0      0     72      1]
 [     0    687  46767    166      0      0      2      0      0      0]
 [     0      0    675  20009      0      0    431      6      0      0]
 [  2592     19      0      0   1235      3      0      0     36      0]
 [  1842      2      0      0      9    113      0      0     18     12]
 [     0      0      0    166      0      0   1258      0      0      0]
 [     0      0      0     68      0      0    120     42      0      0]
 [     6      0      0      0      6      0      0      0      0      0]
 [     3      0      0      0      0      0      0      0      0      0]]
Mean Accuracy using MLPClassifier Classifier: 0.992165
```

Figure 12: Multilayer Perceptron Confusion Matrix

4. Group work: decomposition

For this final project, We had totally six meeting during the project creating process. At first two meeting we brainstormed about the topic and decided predicting poker hand as our final topic. There are four members in our team: Wenhan Kong, William Sun, Wei Qiu, and Xuhui Zhu. We divided the whole project evenly to every member. Wenhan is the facilitator who maintains the pace of meetings of the whole team. Every member is responsible of 1-2 model: Wenhan (**baseline and linear SVM)**, William Sun (**Logistic Regression and MLP)**, Wei Qiu (**Linear regression)** and Xuhui Zhu (**Decision Tree and KNN).**

5. Result

After we build all classification models: Baseline, Linear Regression, Linear SVM, Logistic Regression, KNN, Decision Tree and Multilayer Perceptron. Here is a table that compares the accuracy with all classification models.

Classifier	**Accuracy**
Baseline	**50.1%**
Linear Regression	**40.2%**
Linear SVM	**55.7%**
Logistic Regression	**51%**

KNN	**51.2%**
Decision Tree	**52.7%**
Multi-layer Perceptron	**99%**

Figure 13: All models Comparison Table

6. Conclusion

For a nonlinear, high dimensional and large dataset, linear algorithms and algorithms with distance computation fail due to time and space complexity. From the accuracy table above we can draw a conclusion that MLP is the best model to predict poker hand among the methods we are using. As the figure ROC of MLP shown, despite the comparatively small dataset such as the class 9 (straight flush) and class 10 (royal flush), the neural network does well in classifying most hands. However, it is important to note that winning at poker depends as much on our own strategy as it does our opponent's. Therefore, even the best classifier with the most accurate prediction does not guarantee your win.

7. Bibliography

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

[2] Robert Cattral, Franz Oppacher. Discovering Rules in the Poker Hand Dataset.

[3]Scikit Learn Documentation of dummy estimators
https://scikit-learn.org/stable/modules/model_evaluation.html#dummy-estimators

[4]Scikit Learn Documentation of Linear Regression
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[5]Scikit Learn Documentation of Support Vector Classification
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6]Kevin Swersky.
http://www.cs.toronto.edu/~kswersky/wp-content/uploads/svm_vs_lr.pdf

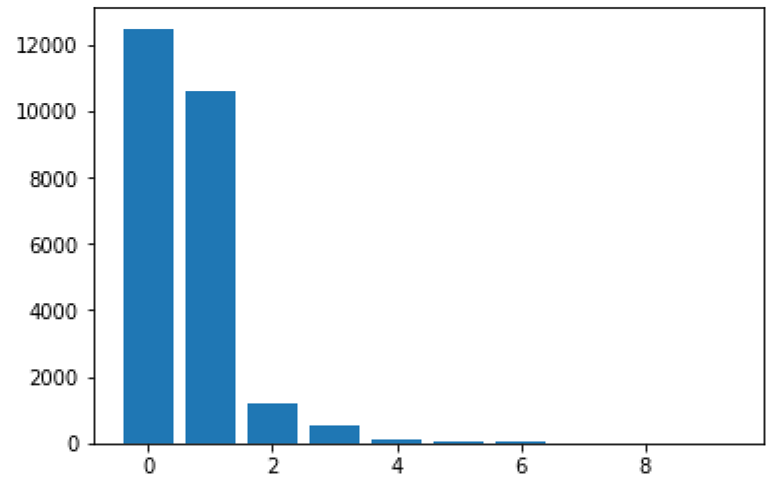
[7]Scikit Learn Documentation of K Neighbors Classifer
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[8]Scikit Learn Documentation of Decision Tree Classifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

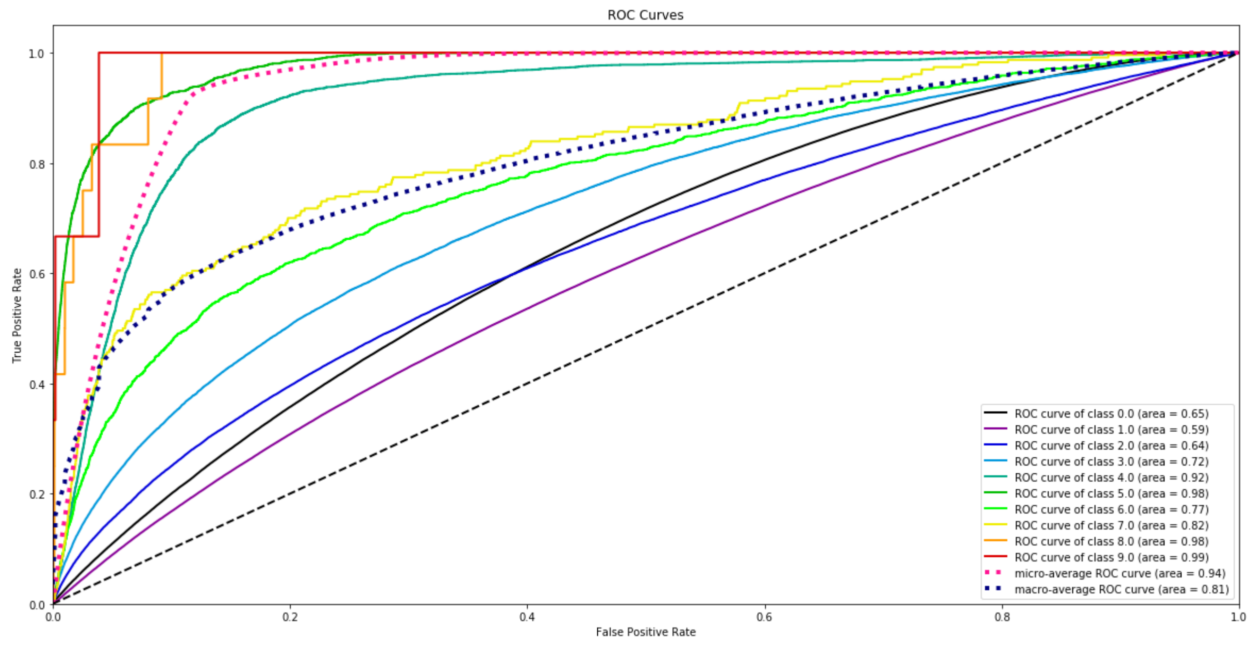
[9] Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961

[10]Scikit Learn Documentation of MLP Classifier
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

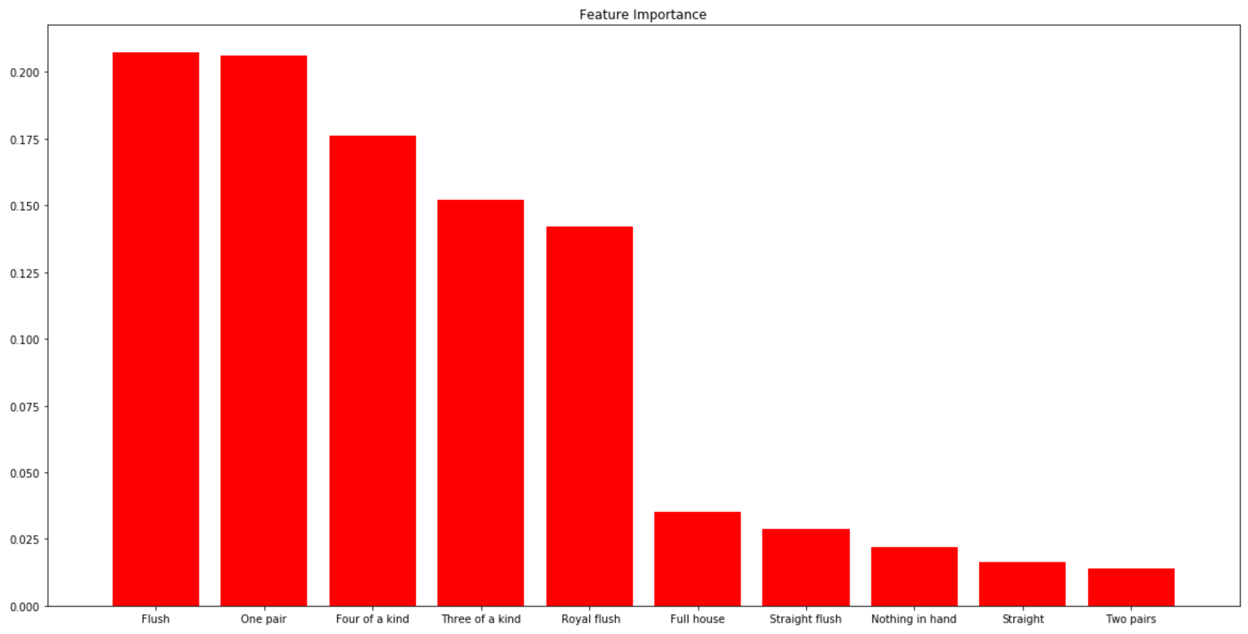
Appendix



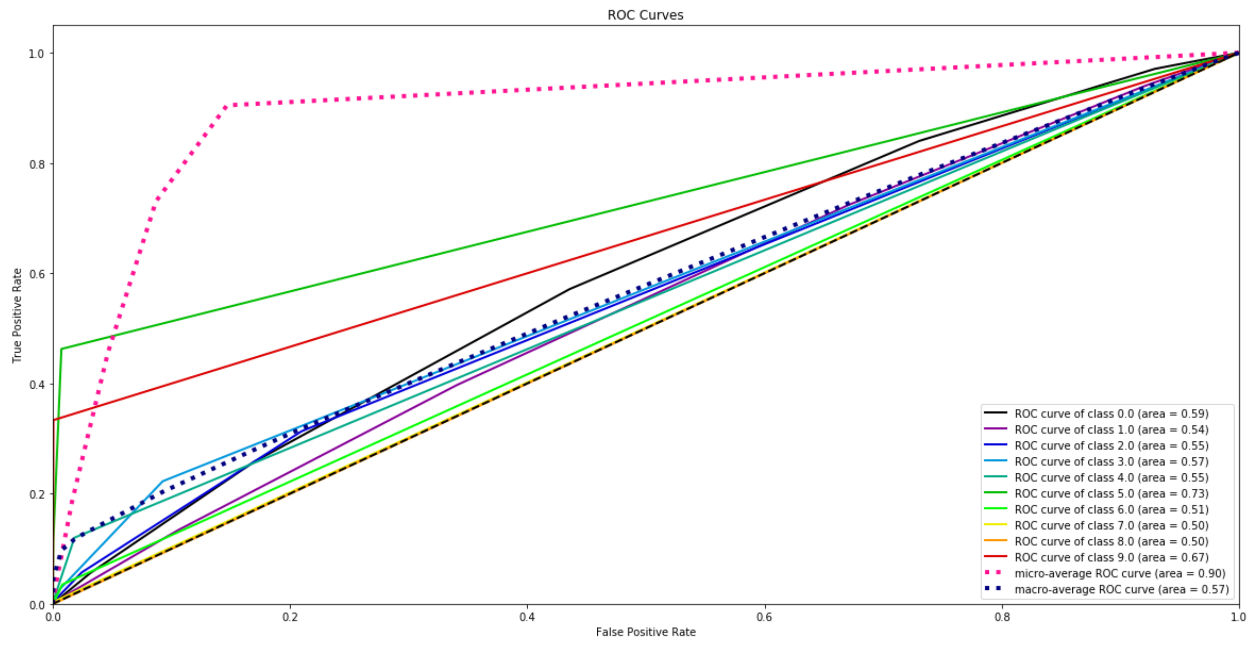
Train Data shape



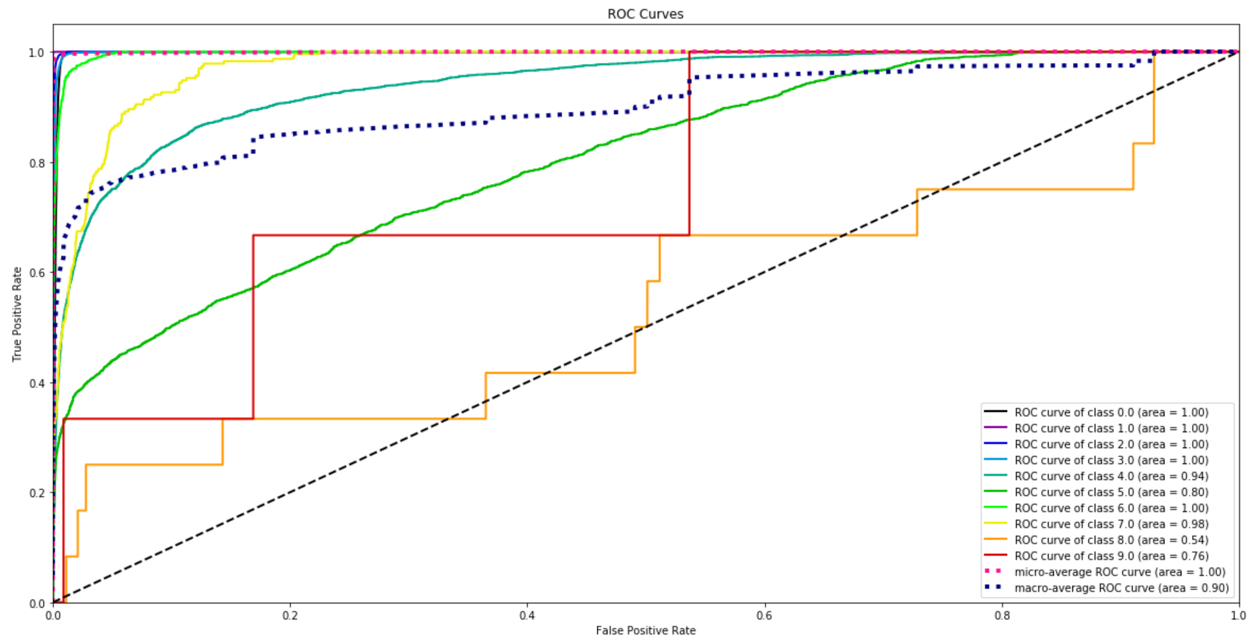
SVM ROC



Decision Tree Feature importance



Decision Tree ROC



MLP ROC