Poker Hand Classification
CS419: Course Project

Shubham Agrawal (180040100) | Bhavini Jeloka (18D100007)
Mitalee Oza (180100067) | Latika Patel (180100062)

I. Introduction

The applications of machine learning have been spreading over various domains. From developing autonomous vehicles and drones to weather and temperature prediction, we see the wide ranging utility of this concept. Another such example is that of game play. After the large scale success of reinforcement learning and artificial intelligence on Atari and Go by Google's DeepMind, there has been an increased focus in games and game play. This could include predicting the outcomes of certain games (thereby calculating the probability of winning) or developing an agent to mimic a player.

One such game is that of poker. Poker is a card game where the player's winning chances strongly relates to the type of hand or cards the players have. The rules are based on rankings of card combinations and categories, thereby making machine learning techniques an obvious choice for analysis and prediction. To this extent, it is often modelled as a classification problem. Although the problem is suited for such a classification environment, there are several challenges associated with the dataset and its training. Neural networks have often been employed for this cause. Both supervised and unsupervised learning techniques have been used in the past for such classification. Feed forward backpropagation and self organising maps have been used earlier.

As a part of an introductory course to this domain, we try to obtain the best model possible for classification. We work with UCI's standard poker dataset where the five cards in a hand decide the rank. We extensively search for the best possible hyperparameters for the chosen model in order to obtain maximum accuracy.

II. Objectives

We intend to tackle the problem of poker hand classification using Machine learning techniques to come up with a simplified model which can predict the poker hand and determine the winning probability. There are no datasets available for the winning probabilities corresponding to poker hands. Hence we generate training and test data set using a deterministic method. We implement our entire project using the *sklearn* library available for python.

III. Approach

To achieve our objectives, we split the problem into two. The first part involves the poker hand classification based on a given training data set. This is followed by the second part which revolves around predicting the winning probabilities.

A. Classification Problem

1) Model Selection and Hyperparameter Tuning: In order to perform our analysis, we pick three popular models to choose from. These three models along with their detailed descriptions are as follows:

- **Neural Networks** - A neural network learning is a learning system that makes the use of a network of functions to understand and translate a data input of one form into a desired output, usually in another form.
- **Logistic Regression** - It is a supervised learning classification algorithm that can be used to predict the probability of the target. For our case, we use multinomial classification where there can be more than 2 outputs (which makes sense in our case because we have more than 2 hands in poker).
- **Support Vector Classification** - It is a supervised learning algorithm for both classification and regression problems. The objective of the SVMs is to find a hyperplane in the N-dimensional space that distinctly classifies the

data points. SVM in its most basic form supports binary classification but not multiclass classification. For multiclass classification, the problem is broken down into multiple binary classifications.

For hyperparameter tuning, we use the grid search approach. In this approach, we perform a search over the parameter values for a given model. The optimal value is chosen after cross validation over the grid. The hyperparameters with the best accuracies are printed for each model. We compare these values across models and then select the best model-hyperparameter pair.

2) Model Evaluation: In order to evaluate the performance of our model, we use the F1 score. It is used to compare two classifiers by taking the harmonic mean of the precision (P) and the recall (R). Mathematically,

$$F1 = \frac{2(P \times R)}{P + R} \tag{1}$$

Here, precision refers to the ratio of correctly classified positive samples to a total number of classified positive samples while recall measures the model's ability to detect positive samples. Therefore the F1 score takes into account both of these parameters. One of the advantages of using the F1 score is that it can be used to tackle the class imbalance problem. The Class Imbalance Problem occurs when the class distributions of the dataset is highly imbalanced. This generally lowers the predictive accuracy for the infrequent class. However, the F1 score gives equal weight to all classes regardless of their dominance thereby providing accurate results.

B. Winning probability prediction

This task has further been divided into 2 parts. First is generation of training and test data. The second is building a regression model and determining its accuracy on the test data generated.

We have assumed a 3 player game. To find the winning probability corresponding to a single hand, ideally we should consider all possible instances of hands with the other two players. The probability would then be the number of wins divided by the total number of instances. But this method is computationally expensive. The number of computations for the winning probability of a single hand itself is atleast of the order 10^{10}. Hence, for each hand, we have randomly generated a certain number of instances of hands with the other players. Broadly speaking, greater the number of instances, lesser the error of our computed probability with the true probability.

1) Model Selection and Hyperparameter Tuning: In order to perform our analysis, we pick two popular models to choose from. These two models along with their detailed descriptions are as follows:

- Linear Regression - This model fits a linear model to the data such that the residual sum of squares between the observed targets and the targets predicted linear approximation is minimized. In this model no hyper parameters are required to define the model. Mathematically,

$$min_w ||Xw - y||_2^2$$

- Ridge Regression - The model is similar to linear regression, but now it also imposes a penalty on the size of the coefficients, that is, it minimizes the penalized sum of squares. This model has one hyper parameter, α.

$$min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

We optimize the value of the hyperparameter α to get the best possible model for the data.

IV. Dataset

The dataset includes 10 features corresponding to the 5 cards a player has and the poker hand label. Shown below are the first 5 rows of the dataset:

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

Fig. 1. Data

Each card is represented by two of these ten features - one defining the suit (converted to an integer between 1 to 4) and one representing the associated number (converted to an integer between 1 to 13). Each pair of consecutive columns denotes

the suit and the rank of a single card in hand. The label (last column) represents the ranking of the poker hand where a higher label implies a higher rank (and therefore, a greater probability of winning given that hand). These ranks are as follows (as taken from the description given in the dataset, "The first percentage in parenthesis is the representation within the training set. The second is the probability in the full domain."):

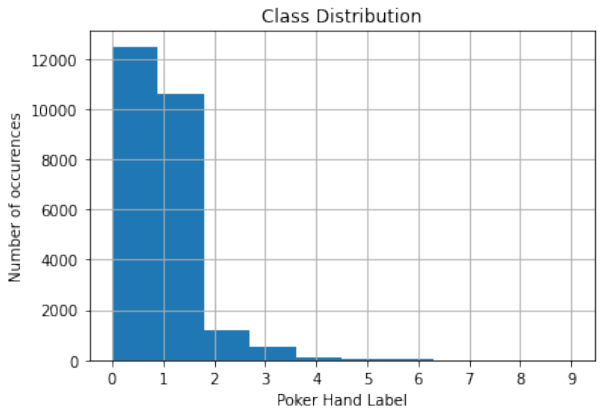

Fig. 2. Training Data

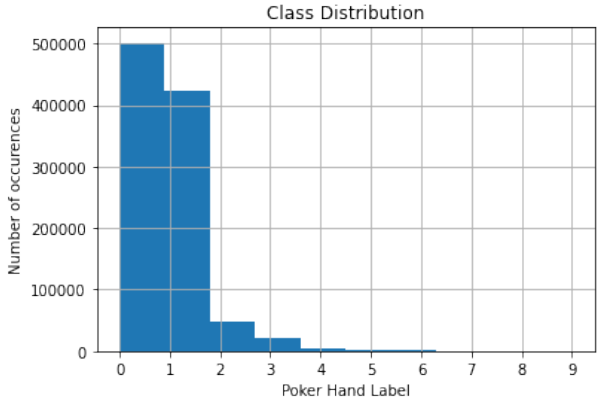

Fig. 3. Testing Data

0: Nothing in hand (worst case), 12493 instances, (49.95202%/50.11773%)
1: 1 pair, 10599 instances, (42.37905%/42.256903%)
2: 2 pairs, 1206 instances, (4.82207%/4.753902%)
3: 3 of a kind, 513 instances, (2.05118%/2.112845%)
4: Straight, 93 instances, (0.37185%/0.392465%)
5: Flush, 54 instances, (0.21591%/0.19654%)
6: Full house, 36 instances, (0.14394%/0.144058%)
7: Four of a kind, 6 instances, (0.02399%/0.02401%)
8: Straight flush, 5 instances, (0.01999%/0.001385%)
9: Royal flush, 5 instances, (0.01999%/0.000154%)

V. Results

A. Classification Problem

The results obtained after performing grid search (from a predetermined search space) on the training dataset gives us the following accuracy values. Figure 4 is a visual representation of the same.

Model	Accuracy
Support Vector Classification	0.5409
Logistic Regression	0.4972
Neural Networks	0.9775

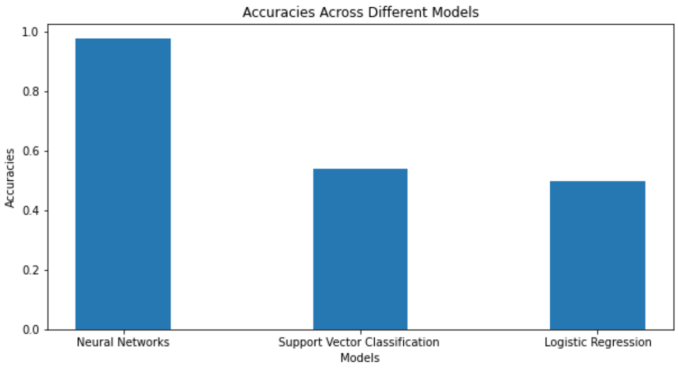

Fig. 4. Performance Evaluation

The corresponding grid search parameters are:

- Support Vector Classification: Regularization parameter - 1.45, kernel - rbf
- Logistic Regression: Regularization parameter - 0.032
- Neural Networks: Activation - tanh, L2 penalty - 0.0001, Hidden layers and neurons - (64,64)

Seeing that neural networks are clearly the superior choice. Deploying the model, we get the results on the test sets as follows:

	Mean Squared Error	F1 Score
Testing data	0.40	0.44

B. Winning Probability Prediction

Least Square error and Ridge Regression Method ($\alpha = 10000$) were used to model the probability. Both the methods give an error of 0.07 on the testing data.

VI. Observations and Discussion

As mentioned earlier, the data is extremely unbalanced and that makes it particularly challenging for classification algorithms. As a result the accuracy obtained by the SVM and Logistic Regression models are not very high and in fact, lower than 60%.

To overcome this problem of unbalanced data and the non linear nature of the problem, we decided to implement Neural Network Models for the classification task. For Neural networks, we proceeded with the hyper-parameter tuning in the following way -

1. First, we compared the accuracy for models by increasing the number of layers. A comparison of (10),(10,10) and (10,10,10) configurations showed us that the maximum accuracy of 0.5789 was obtained for 3 layered - (10,10,10) model. While this was better than SVM and Logistic Regression, this was still a low value for accuracy of the model.

2. Next, we tried to increase the complexity of the model by increasing the number of neurons for each layer. A (100,100,100) configuration gave an accuracy of 0.5984. This was against our expectations as even after such a great increase in number of neurons, the accuracy only slightly improved. The reason for this maybe because too many neurons could have led the model to overfit the data and hence the lower value of accuracy.

3. Mitchell suggests [1] that a network of 3 layers (1 output and 2 hidden layers) can be used to model any arbitrary function. With this in mind and the observations from the trials done so far, we decided to experiment with 2 layered NN models with different number of neurons to find the model with accuracy at least better than 95%. After some iterations, we observed that a network of (64,64) architecture gave the best result of 0.9775 out of 4 different configurations (64,64),(75,75),(85,85),(100,100).

The configuration (64,64) was chosen for classifying the poker hand for the testing dataset.

An F1 score of 0.44 was obtained. Since F1 score gives each class similar importance irrespective of the number of occurences of that class in the dataset, it is a better measure of the accuracy of the model.

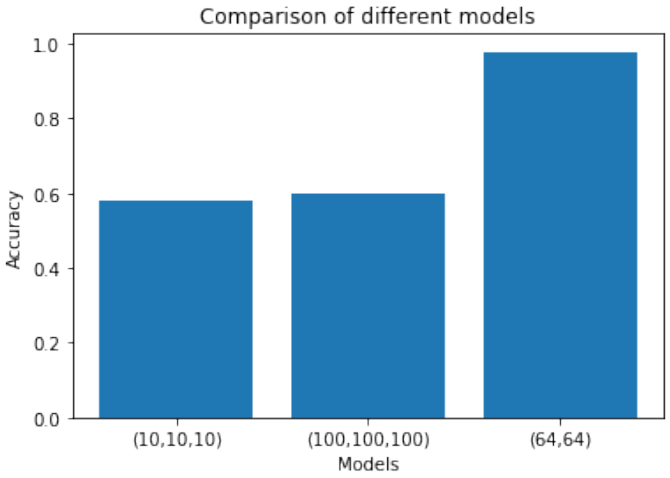

Fig. 5. Comparison of Neural Networks

This was the best F1 score that we obtained out of all the configurations experimented. A low F1 score of 0.44 shows us that our model is unable to capture all the classes. The higher poker hands which have lower number of occurrences are mis-classified and hence, we obtain a relatively lower F1 score.

VII. Conclusions and Future Work

Our classification results strongly support neural networks as a model for handling large and complex datasets in the real world. Even within neural networks, we emphasise on the importance of the hyperparameters, i.e, the number of hidden layers and nodes. In the future, we can further fine tun our results by increasing the search space for the hyperparameters. We can also expand our research to partially observable scenarios where we are not aware of all the cards of the hand.

For the winning probability, we realise that both the methods yield equivalent results and can therefore be used interchangeably. However, this is probably due to the low number of players. By increasing the number of players, we might see significant variations.

Naturally, simulating an actual poker game without any relaxations on the rules will be the next step forward.

VIII. Acknowledgements

We would like to thank Prof. Abir De for giving us the opportunity to actually execute the concepts learnt in class. Moreover, we are very grateful to the TAs Shubham and Ipsit for their prompt responses to all the doubts asked.

IX. References

[1] T. Mitchell, "McGraw Hill series in computer science," in Machine Learning, New York, McGraw-Hill., 1997, p. 105.