Poker Hand Classification

Xue Gu
xgu63@wisc.edu

Jing Huang
jhuang339@wisc.edu

Qiuhong Li
qli288@wisc.edu

Abstract

As Texas Hold'em poker, a betting game is increasingly prevalent in the United States, the need to quickly classify poker hands becomes more demanding. Regarding the numerous combinations of cards in hand, this classification can efficiently display whether it is a valid poker hands or not. Simultaneously, it helps learn the rule based problems and solve resource allocation in a network. For this project, we implement the learning algorithms including KNN, Decision Tree, Rabdom Forests and HistGradient-Boosting, Gradient Boosting, Multi-Layer-Perceptron. In order to check the generalization performance of our models, we utilize the Stratified 10-fold Cross-Validation in the evaluation stage. Additionally we perform mean score-based Grid search to gain more insight in the way each model fits our dataset, and how the data must be learned. Finally, we use a range of scoring methods from basic accuracy to ROC curves and AUC values to judge the performance of each model.

1. Introduction

Poker is a well known played betting game in the United States. There are many different kinds of poker games in the world, but all of the games need players to know poker hands, the combination of cards in hand, in order to know the most efficient way to show cards and win. In a poker of 52 standard deck, there are four suits (Spades, Hearts, Diamonds, and Clubs) and thirteen ranks (2 through 10, jack, queen, king and ace).

In this project, we will focus on Texas Hold'em poker. In this game, every player has two in hand (face down) and the dealer will put five cards face-down on the table. After one round, the first three cards will be turned face-side-up and in the subsequent rounds, the last three cards will be turned face-side-up one by one, making the maximum to four rounds. Between each round, all players (unless a player folded in a previous round) are able to place bets (fold, call or raise). The winning pot goes to the player still in the game with the strongest hand.[3] Hands are different combinations of five cards (two cards in hand and three cards from the five face-side-up cards on table). There are 10 different hands in total, and the order of strength of hands (For example, Nothing in Hand, One Pair, Two Pair, Three of a Kind, Straight, Flush, Full House, Four of a Kind, Straight Flush, Royal Flush). For this project, we only focus on whether there has valid poker hands. Specifically, we treat class 0 as nothing in hand versus class 1 as having poker hands no matter what kinds of poker hands they are.

Poker is widely regarded as the benchmark for AI in the space of incomplete-information game. Recently, researchers pay much attention to it and put much effort in designing poker bot to solve poker and other incomplete-information games. Because of the way poker-related problems is represented, it is not easy to discover the rules that can correctly classify poker hands. Noticing the popularity of poker games, we decided to design a machine learning algorithm involving techniques such as Rabdom Forests, gradient boosting to analyze the underlying concepts of poker hands. In this way, the performance and interpretation of poker players can be largely enhanced, thus achieving the entertainment and learning purpose.

In addition, classifying poker hands can also bring broader societal impact. Further advantages of the poker hands classification problems are that experiments with poker hand classification and its solution readily resonant with several other real-world problem domains such as resource allocation in a network, and that it helps learning of rule based problems even in unsupervised learning tasks.[8] The goal of our project was to identify whether there exist valid poker hands formed by five cards using machine learning techniques and find the algorithm with the highest accuracy.

2. Related Work

There have been many work done in the domain of poker hands classification with various learning paradigms and architectures. For example, Suraiya Jabin (2016 International Conference on Computing, Communication and Automation) utilized artificial neural networks using multi-layer feed-forward backpropagation to solve this real valued classification problem [7]. Her approach can generalize to poker hands of 5, 7... cards.

Another example is the NFSP proposed by Heinricha and Silver. Heinricha and Silver (2016) combined fictitious self-play with deep reinforcement learning methods to approach a Nash equilibrium, whereas common reinforcement learning methods is diverged. When they applied an imperfect-information version of poker Leduc poker, Neural Fictitious Self-Play (NFSP) approached the Nash equilibrium. They use a poker game of real world scale, Limit Texas Hold'em, the performance of human experts and state-of-art methods can be approached and this competitive strategy is learnt by NFSP [6].

The first related work is the project that our dataset comes from. It is a rule-based study and it used deep learning algorithms. The second one also used deep learning methods and based on Texas Hold'em, which is the same as our project. While our project uses methods mostly what we learned in class like K Nearest Neighbors, Decision Tree, Rabdom Forests and Gradient boosting.

3. Proposed Method

3.1. K Nearest Neighbors (KNN)

K Nearest Neighbors is one of the most frequently used classification techniques in machine learning. It is a non-parametric lazy learning method, which allows the users to solve classification and regression problems. The non-parametric property of KNN allows it to be easily implemented and applied to a wide range of fields. The lazy property means that the training phase is minimal. Specifically, in the training stage, KNN merely memorizes the datasets and defers all the computation untill prediction. In KNN algorithm, the Euclidean distance(the distance of the test observation with every observation in the training set) is computed. Then the first k nearest points are chosen after sorting the distances in increasing order, and the corresponding class is assigned to the majority class of these k observations. Besides, KNN makes no assumption regarding the distribution of data.

3.2. Decision Tree

Decision Tree is a supervised algorithm and mostly used for classification problems, but also for regression problems. It works for both categorical and continuous input and output variables. Decision Tree is a decision support tool that using tree-like graphs to describe the possible consequences. It splits the dataset into two or more sub-dataset based on largest information gain. In the Decision Tree plot, each node represents a feature, each link represents a decision and each leaf represents an outcome. When there are many features, it has a large amount of split, which results in complex tree and will be easy to overfit. Therefore, the performance can be improved by pruning, like setting a maximum depth of the model. The advantage of Decision Tree is the easy interpretation, which means that it can be understood even for people with no statistical knowledge and analytical background and the plot is very intuitive.

3.3. Random Forests

Random forests are among the most widely used machine learning algorithms, probably because it has relatively good performance "out of the box" and ease of use (do not require much hyperparameter tuning). It is a supervised learning algorithm and it is used for both classification and regression problems. There are a large amount of individual trees in Random Forests and these trees operate as an ensemble (bagging). It creates an uncorrelated forest of trees so that the prediction is more accurate than any individual tree by using bagging and feature randomness.

3.4. Gradient Boosting

In addition to random forests(bagging), gradient boosting is another supervised machine learning model popular to classification problem. It combines multiple weak learners into a strong learner. Rather than training the Decision Trees(the underlying model) in isolation like in random forests, boosted trees are trained sequentially and greedily to make incremental improvements(based on the learning rate) by correcting the "residual errors" or the misclassification error in this case from the predictions from previous models. The objective of gradient boosting can be generalized to minimize the training loss with an arbitrary regularization term.

XGBoost XGBoost or "Extreme Gradient Boosting" is an implementation of gradient boosting based on CART tree ensembles. The advantage of XGBoost is three-fold. By approximating to build to perfect tree that correctly classifies, boosting reduced bias. With built in random resampling and use of majority voting during training, boosting also helps fight the potential threat of overfitting. Last but not least, it is an efficient implementation that usually gives decently fast performance. Later, through grid search with cross validation, the best parameters for XGBoost in this project are chosen.

HistGradientBoosting HistGradientbBoosting or "Histogram-based Gradient Boosting " classification tree is another gradient boosting model implementation. In other words, it utilized similar boosted tree ensembles for prediction. Unlike XGBoost, its efficiency is rooted in the fact that HistGradientBoosting do not require sorting the feature values and instead use a data-structure called histogram. Although HistGradientBoosting is expected to have better predictive performance than that of XGBoost, according to the benchmarks provided online[1], this expectation is not the case for our project. Similarly, through grid search with cross validation, the best parameters for XGBoost in this project are chosen.

3.5. Multi-layer Perceptron

Multi-layer Perceptron(MLP) is a feedforward model from the artificial neural network framework that is able to handle classification . It consists of an input layer, an output layer and multiple hidden layers. The input layer literally takes the inputs or the features, while the output layer outputs the prediction of the model following the process from the hidden layers. While the hidden layers are not directly exposed to the inputs, it handles the inputs processed by the input layers with certain activation function. The MLP in our project utilizes three dense, nonlinear hidden layers(each with 100 nodes) with activation function “tanh”. The MLP is trained a supervised learning technique called backpropagation. The hyperparameters are tuned under the guidance from some commonly used machine learning conventions and grid search with cross validation.

4. Experiments

4.1. Dataset

In this research, the dataset from UCI Machine Learning Repository [2], in which each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. In theory, there are 10 types of valid poker hands as listed in the dataset. For example, full house and pairs are both valid poker hands. However, due to the extreme imbalance between valid and invalid poker hand classes, we will combine all 10 types of valid poker hands into one class against the invalid poker hands. This decision will help create balance between the two classes and thus avoiding bias towards the majority class. For predictive features, each of these five cards have suit and rank two attributes, hence in the training dataset, there are total 10 predictive attributes [5]. By analyzing these 10 attributes, we can get a model for predicting whether the poker hand is valid, which is a binary class attribute.

4.2. Experiment Setup

Data Preprocessing To address the imbalanced multi-class problem in the original dataset, all valid 9 types of poker hands(originally class 1-9) are pooled together into a single class against the original class 0, which now will be identified as invalid poker hand class 0. Fortunately, this data processing results in fairly balanced binary classes with class 0 (invalid poker hands) obtaining a class size of 12, 493 and class 1 (valid poker hands) with class size of 12, 516. The same applies for test dataset. The predictive features will remain unchanged for both datasets.

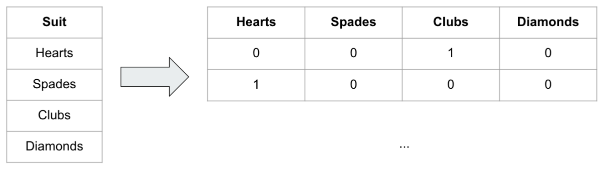

Figure 1: Example of One Hot Encoding transformation

Holdout Split We are given two datafiles, specifically one for training dataset and validation dataset, and one for test dataset. The training dataset and test dataset are independent. We use simple holdout method to split our first dataset into training subset(80%) and validation subset(20%) while maintaining the stratified ratio of the two classes in the training data. Overall, there are 20,008 instances in training subset, 5,002 instances in validation subset, and 1,000,000 instances in test.

One Hot Encoding One Hot Encoding(OHE) is a technique used to transform nominal features into separate binary features. It takes the original feature values and derives new indicator features that only takes in values 0 or 1 to indicate match from observations. Using One Hot Encoding allows us to detach the nominal notion from the categorical variables we are using. Since neither rank nor suit have a relevant ordering under the domain of our problem, in desire to remove the notion of order from the variables we one hot encode them, allocating a separate dimension for each possible value of each variable within the feature vector.

Figure 2 shows a snippet of how OHE works. This technique is used to improve the prediction accuracy for our training algorithms, especially for those that are tree based because it can provide better splits of the data. However,one disadvantage is its potentially significant increase in the dimensionality of the dataset. However, due to the small number of original features, this transformation only increased the dimensionality to 85 which proves to be not too concerning for this purpose of this project.

4.3. Learning

The training set has sample size 20,010. We split the training set into training subset and validation subset, as we mentioned in Holdout Split. Except for KNN, all other five methods first use scikit packages to train the model on train subset, make a prediction on validation subset and calculate the training accuracy. Then, use the model to make a prediction on test set and calculate the test accuracy. Since neither rank nor suit have a relevant ordering under the domain of our problem, we utilize OHE to change the training subset and validation subset, use the changed subsets to train the model, make a prediction and calculate the training accuracy. Then, we use the model to make a prediction on test set and calculate the test accuracy. For the Decision Tree, we utilize DecisionTreeClassifier library from Scikit-learn package; for Rabdom Forests, we utilize RandomForestClassifier library from Scikit-learn package; for XGBoost, we utilize XGBClassifier library from xgboost package; for HistGradientBoosting, we utilize HistGradientBoostingClassifier library from Scikit-learn package; for Multi-layer Perceptron, we utilize MLPClassifier library from Scikit-learn package. One exception of applying the above mentioned learning process is the KNN. The lazy property of KNN enables the algorithm to achieve the learning easily by merely memorizing the data points, as mentioned before. During the training stage, KNN does not apply the One Hot Encoding.

4.4. Evaluation

One of the most common evaluation metrics to estimate generalization performance of the models is cross validation. Specifically, this project utilized stratified 10 fold cross validation to evaluate how accurately the six predictive models will perform in "new" dataset. A 10-fold cross validation performs in ten iterations during which a training and validation subset will be randomly created with a 9:1 ratio (training vs validation), thus resulting in ten dependent test subsets. The motivation to use cross validation is that the models will be fitted to new training dataset at each iteration. Without cross validation, the model training is only limited to the in-sample training subset.

Due to our sample size without being extremely large, the stratified 10-fold cross validation was feasible without being extremely computationally expensive. The average prediction accuracy on the 10 new test sets from the 10 iterations is the performance metric used to assess the performance of the models. However, since the training data has been transformed using OHE technique to increase prediction accuracy, evaluation will also be performed after the transformations to assess the performance of models trained on data after OHE as well. The comparisons will be summarized in the results section.

4.5. Parameter Tuning

KNN In essence, the tuning phase of KNN is closely related to the generalization error. Choosing the parameter k is essential for KNN. We are facing a dilemma when k is too large, the training set is more likely to overfit. While k is too small, the model is underfit. According to our results, the test accuracy and train accuracy of larger k are increasing given the same number of instances for train and test dataset. Simultaneously, we do not want our model to be underfitted because of the large k. Remember that large k will lead to low variance but high bias. It might occur the situation that all test data points belong to the same class. In this way, we choose k=7 as our parameter for KNN, since the accuracy of the test data is increasing slower than before.

for RF.png

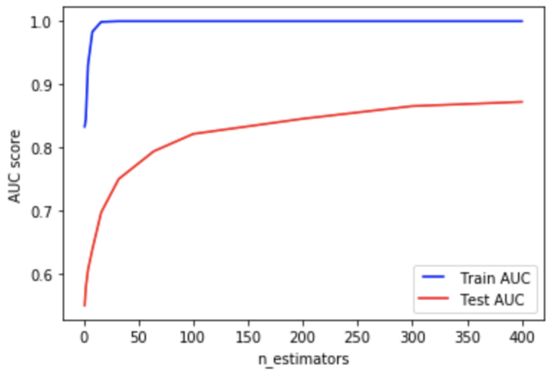

Figure 2: n_estimator parameter for Random Forests

Random Forests For the Random Forests, the n-estimator parameter represents the number of trees in the forest. Usually, the larger the number of trees, the better to learn the data. However, more trees will slow down the training process. So, I do a parameter search to find the sweet spot by drawing the ROC curve for the n-estimator parameter, we can find that after 250, the test AUC becomes flat. So we choose 250 for n-estimator to fit the model.

Grid Search Regarding with the Grid Search, we apply the GridSearchCV library. This library is basically a model selection utilizing cross-validation. After running GridSearchCV, we could find out the best hyperparameters and used it to fit on the training subset, make a prediction on test subset and calculate accuracy so that we are able to get an optimal model. We use Grid Search for Decision Tree, XGBoost, HistGradientBoost and MLP. For Decision Tree, since in DecisionTreeClassifier library, there are two choices for criterion, which is a function to measure the quality of a split, we do the Grid Search to find the one that can better fit the model. Also, since we have 10 features in our dataset, if we do not set the maximum depth of the Decision Tree, the tree will be very big and will take lots of time to run. So, after the Grid Search, the results shows that the best criteria is entropy with the depth equals to 20. For the XGBoost, max_depth is the maximum depth of a tree and min_child_weight is "the minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building pro-

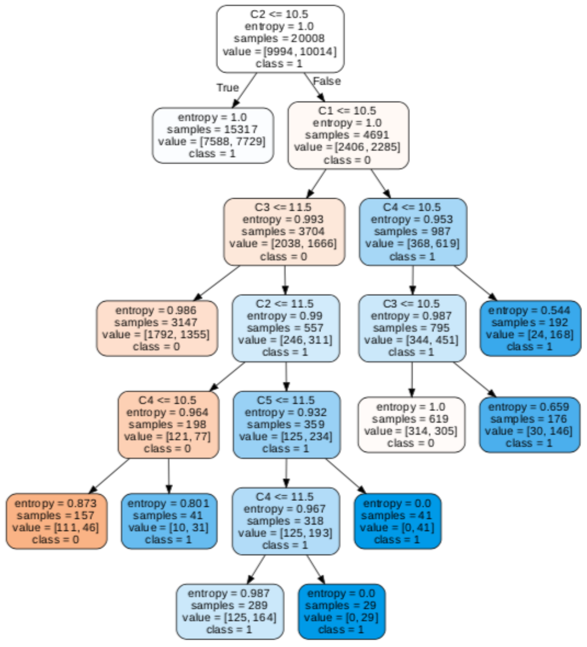

Figure 3: Decision Tree

Value of K	Train Acc	Validation Acc	Test Acc
k = 1	100.00%	59.54%	59.30%
k = 3	80.50%	61.16%	60.82%
k = 5	76.10%	62.14%	61.68%
k = 7	73.71%	62.63%	62.15%
k = 9	72.37%	63.65%	62.45%
k = 11	71.41%	63.89%	62.76%

Table 1: KNN prediction accuracy with different values of k.

cess will give up further partitioning".[4]. The results end in max_depth is 4 and the min_child_weight is 4. For HistGradientBoost, the setting of l2_regularization can reduce the complexity of the model and reduce the risk of overfitting. The min_samples_leaf is the minimum number of samples per leaf. If the dataset is very small, this value should be small. The result is that l2_regularization equals to 0 and min_samples_leaf equals to 3. For the MLP, we set the activation function of the gradient boost to tanh function, since the tanh function is a rescaling method of the sigmoid function. It optimizes the log-loss function. For the hidden layer parts, we select the 50 hidden units.

4.6. Software

Our project utilizes the following packages: Scikit-learn, NumPy, DataFrame, Pandas, Matplotlib, Mlxtend. We first apply the NumPy and DataFrame packages to preprocess the data. We used the scikit-learn package to implement learning algorithms such as Decision Tree and gradient boosting.

Model	Train Acc Pre-OHE	Train Acc OHE
Decision Tree	59.38%	62.95%
Rabdom Forests	68.90%	85.50%
XGBoost	89.82%	99.44%
HGBoost	76.05%	96.40%
MLP	97.00%	98.36%

Table 2: Training Accuracy before and after One Hot Coding

Model	Test Acc Pre-OHE	Test Acc OHE
Decision Tree	59.58%	63.19%
Rabdom Forests	69.20%	85.90%
XGBoost	89.87%	99.42%
HGBoost	75.43%	96.90%
MLP	96.64%	98.25%

Table 3: Testing Accuracy before and after One Hot Coding

4.7. Hardware

For the simple model, such as KNN, we merely implement the models on our own private laptops. One extraordinary thing we utilized for our complex model is the GPU resource. In this way, the process of waiting for results has been dramatically speeded up.

5. Results and Discussion

After stratified 10-fold cross validation, we obtain prediction accuracy before and after the transformation of the OHE as shown in table 4. It can be observed that all models have decent validation performance which indicates that all model generalizes well when tested on "new" data. It is also clear that OHE improved accuracy of the models, especially for models that are tree-based including XGBoost and HistGradientBoosting. The increase of accuracy after using OHE for models including XGBoost and random forests is expected due to the benefits of OHE. Through hyperparameter tuning using grid search with cross validation on most of our training algorithms, figure 6 and figure 7 shows the examples of heatmaps that represent the prediction accuracy with different combinations of parameters for the gradient boosting models used in this project.

We can see that, in general, as the min_child_weight and max_depth decreases, the prediction accuracy of the XGBoost model also increases. Max_depth specifies the

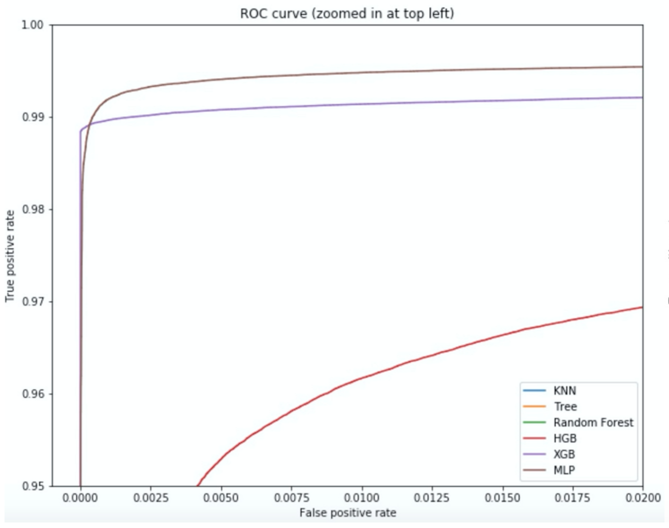

Figure 4: zoomed ROC

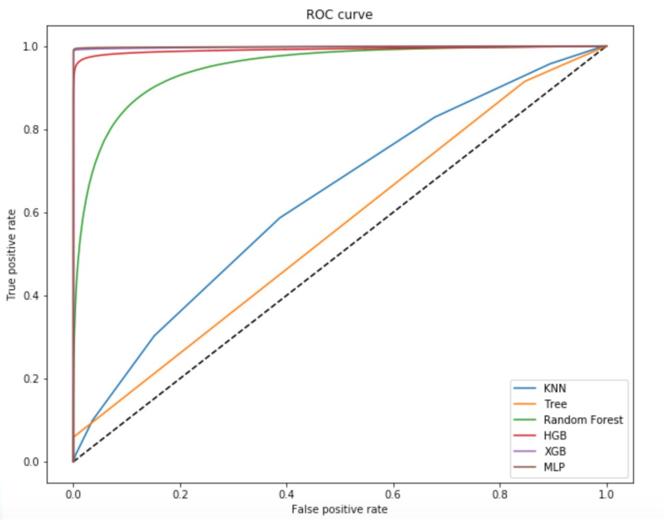

Figure 5: ROC

Model	10-fold Acc Pre-OHE	10-fold Acc OHE
KNN	61.57%	59.53%
Decision Tree	59.01%	63.21%
Rabdom Forests	68.90%	84.87%
XGBoost	88.28%	99.36%
HGBoost	72.78%	97.34%
MLP	94.51%	97.64%

Table 4: 10-Fold Accuracy before and after One Hot Coding

Model	AUC
KNN	0.60799
Decision Tree	0.53455
Random Forests	0.88550
XGBoost	0.99443
HGBoost	0.98326
MLP	0.99646

Table 5: Area Under Curve with different model

maximum depth of each tree, with larger values allowing for more splits and more complex individual learners, however higher values of this parameter run the risk of overfitting, or preforming bad outside of the training data. min_child_weight on the other hand controls further splitting of nodes, with higher values controlling growth of a tree, thus preventing complex learners from forming at higher values of the parameter. Similarly to XGBoost we can see that high performance can be achieved with low values of min_samples_leaf, which allows for more splitting within learners, but is offset by a high value of l2 regularization coefficient, which tames overfitting. It is interesting to note how the opposite strategy, of low l2 regularization and high min_child_weight also achieves decent performance. Similarly, for models that use grid search for hyperparameters tuning, the best parameters are chosen based on their averaged prediction accuracy over the cross validation performed for model selection.

From table **??**, it can be observed that overall MLP has the highest testing accuracy, followed by XGBoost, while decision tree model ranked last with only less than 60% test accuracy. It is interesting to note that HGBoost performed worse than XGBoost by obtaining a slightly lower test accuracy. It does not conform to our expectation of better perform from HGBoost based on the benchmarks summarized at [1]. The reason for this discrepancy is not yet clear and could be an interesting topic to explore in future work.

In addition, the ROC are plotted for each model as shown in figure 4 and figure 5. For HGB, XGB and MLP we get low False positive rate and higher false negative rate. From previous grid search results, we notice that for the MLP classier tanh works better than relu regardless of the number of nodes in each hidden layer. One of the reasons might be that tanh resembles logistic regression, or the sigmoid function. In fact tanh is like a scaled version of sigmoid function that can make decent binary separations. Since in our case the MLP classifier has to infer a simple "true/false" structure, an activation function such as "tanh" that essentially integrates a decision barrier makes sense to perform well. Furthermore, the area under the curve ROC AUC are summarized in table 5.

The confusion matrix for all six models on the test dataset is shown in figure 8, figure 9, figure 10, figure 11, figure 12 and figure 13. It can be observed that the top per-

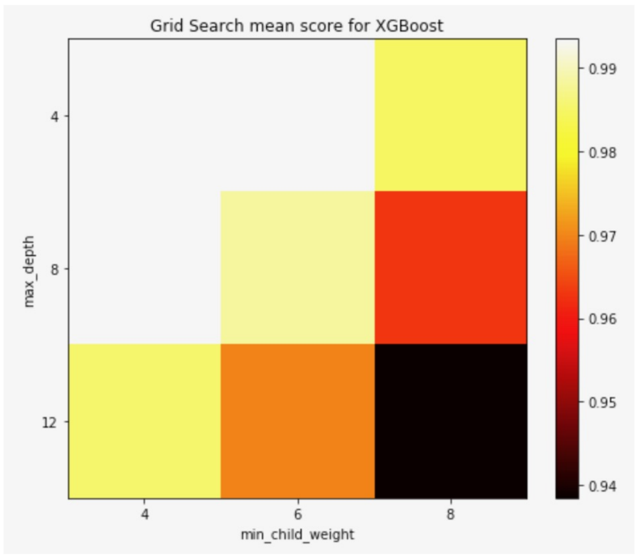

Figure 6: Grid search heatmap with XGBoost

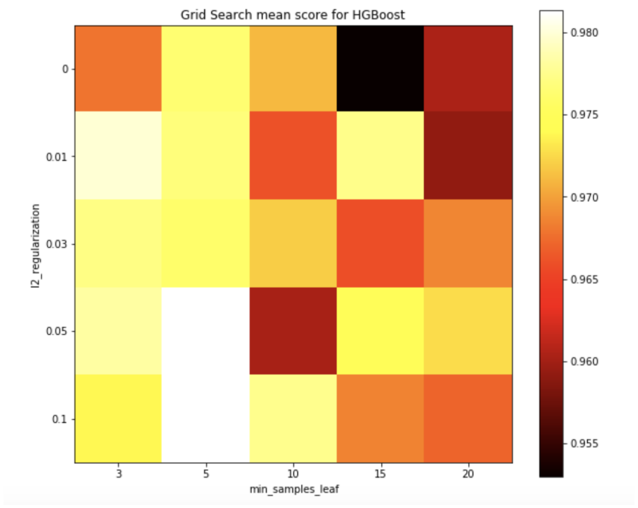

Figure 7: Grid search heatmap with HistGradientBoosting

forming models similarly demonstrated low false positive and higher false negative rate, both in the case of MLP and XGBoost.

The F1 scores for all six models are calculated in table 6. The F1 scores are one of the final gauges to measure the performance of our models on the test dataset that considers both recall and precision equally.

While this project successfully predicts the validity of the poker hands, there are still several limitations. By pooling all 9 types of the poker hands together as one group, it is almost impossible to know which poker hand is present for a certain observation, if present at all. Another limitation is the relative small size of the training dataset compared to the independent test dataset.

Model	F1 Acc
KNN	60.17%
Decision Tree	66.18%
Rabdom Forests	88.19%
XGBoost	98.30%
HGBoost	99.44%
MLP	99.64%

Table 6: F1 Accuracy for different models

6. Conclusions

In conclusion, we can see that both tree based models as well as deep learning based approaches can perform extremely well on the proposed problem, which is not unsurprising given the robust nature of the logic behind poker. However we were still able to gain several insights into modeling of the problem with different approaches, such as the importance of choosing the appropriate activation function for the multi-layer approach, as well as striking a balance between complexity and generalizability of tree based methods. More importantly each of us gained valuable experience working with numerous packages and models available through python's data science ecosystem, as well as plotting and analyzing the achieved results.

Future work on the matter could generalize the problem to a multi-class setting, to identify poker hands more precisely. This, however, is a much more daunting task, since the combinatorial nature of poker hands makes any representative sample of data extremely biased in favour of a few select hands, making it challenging to build a classifier which can classify the rarely occurring hands well, maintaining a good differentiation between the big classes. While different approaches to this issue exist, including weighted performance metrics, as well as altering sampling methodology for training, given a heavily imbalanced dataset such as ours none of them are perfect. Future work wold face these challenges.

7. Contributions

Xue Gu :

- Proposed methods: Gradient Boosting; Multi-Layer-Perceptron
- Experiments: Dataset; Data Preprocessing; OHE; Evaluation
- Results and Discussion
- Conclusions

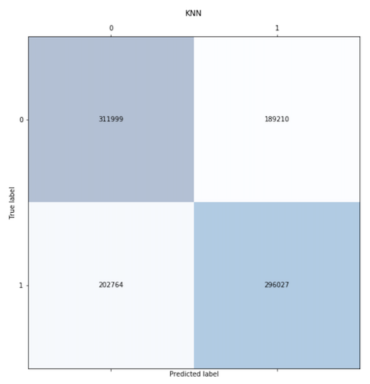

Figure 8: KNN confusion matrix

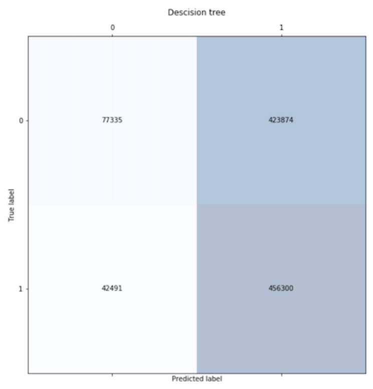

Figure 9: Decision Tree confusion matrix

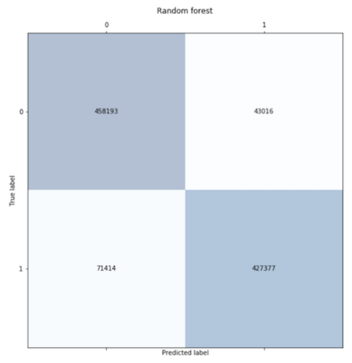

Figure 10: Random Forests confusion matrix

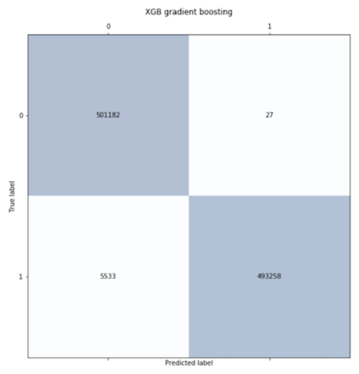

Figure 11: XGBoost confusion matrix

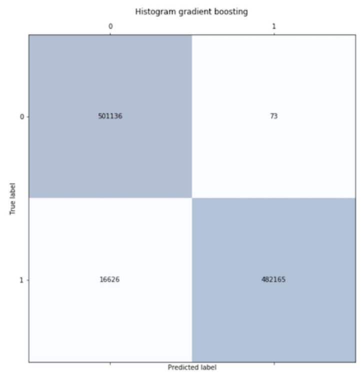

Figure 12: HGBoosting confusion matrix

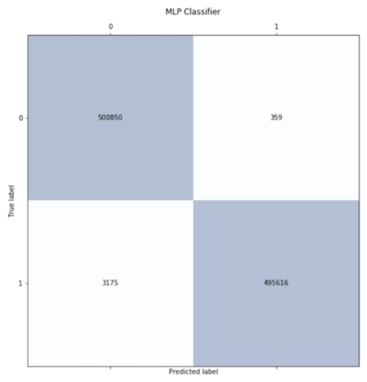

Figure 13: MLP confusion matrix

Jing Huang :

- introduction; Related Work
- proposed methods: Decision Tree; Random Forest
- experiments: Learning
- Parameter Tuning
- Conclusion

Qiuhong Li :

- Abstract
- proposed methods: KNN
- Experiments: Holdout Split; Parameter Tuning; Software; Hardware
- Conclusion

References

[1] https://lightgbm.readthedocs.io/en/latest/Experiments.html.

[2] https://archive.ics.uci.edu/ml/datasets/Poker+Hand.

[3] Texas hold ’em. https://en.wikipedia.org/wiki/Texas_hold_%27em.

[4] Xgboost parameters. https://xgboost.readthedocs.io/en/latest/parameter.html.

[5] D. Dua and C. Graff. Uci machine learning repository. 2017.

[6] J. Heinrich and D. Silver. Deep reinforcement learning from self-play in imperfect-information games. 03 2016.

[7] S. Jabin. Poker hand classification. *2016 International Conference on Computing, Communication and Automation*, page 269, 04 2016.

[8] F. O. Robert Cattral and D. Deugo. Evolutionary data mining with automatic rule generalization. 2016.