Observing the Behavior of Regression on Man-Made Concepts

Ofri Shoshany
Department: CICS
UMass Amherst
Amherst, MA 01003
oshoshan@umass.edu

Joshua Espinosa
Department: CICS
UMass Amherst
Amherst, MA 01003
espinosa@umass.edu

Abstract

Poker is a game that's often used in machine learning frameworks due to the complexity and flexibility of the data set to model many different types of situations. Additionally, Findler says that poker is a "game of partial information" [2]. Therefore, this project will talk about evaluating a poker hand not only through different regression methods on full hands, but also on incomplete hands to predict its strength. Due to the verity of Domingos' interpretation of Occam's Razor as shown in [3] (discussed later), we made our selection of models through a limited filter in order to safeguard against overfitting. KNN Regression, random forests, and gradient boosted regression trees were used, respectively, due to the low dimensionality, flexible complexity and scalability, and robustness to overfitting and outliers. These three models offer a concise overview of the different types of models and their expected values. Our results indicate that Random Forest Regression performed the best, but due to our strict penalization to adhere to the spirit of the game of poker rather than theoretical efficiency, we did not observe RMSE error lower than about .6635. Additionally, our model almost always opted to remove the features corresponding to the "man-made" suit features, where man-made is defined by an artificial categorical label that has little to no mathematical meaning until explicitly defined. This was the case because for the most part the suit of the hands did not affect the hand classification since the suit is only relevant for determining "flush," "straight flush," and "royal flush," all of which are rare occurrences. Our results show that some of these artificial labels such as a "flush" are indeed relatively hard for machine learning algorithms to work with. We however found that these abnormalities are often predictable and can therefore be addressed through strategies such as feature selection.

1 Introduction

This project is designed to investigate the behavior of regression on "man-made" concepts, where man-made is defined by an artificial categorical label that has little to no mathematical meaning until explicitly defined. The dataset we chose to investigate this was a dataset designed to identify 5-card poker hands. Our goal was to correctly identify complete and incomplete poker hands as best as possible and observe the behavior exhibited by the regression. We choose to observe regression including artificial concepts instead of genuine mathematical statistics because we felt that this would provide a more interesting and complicated distribution to observe. For example, it is complicated to identify an out-of-order "straight" in a poker hand using machine learning because the concept of a "straight" is man-made. Therefore, it is relatively hard to use machine learning to figure out that a data instance containing 5,3,6,4,2 of random suits (which is a straight) is completely different than a data instance containing 5,3,6,4,8 of random suits (which is not a recognized hand).

We feel that our research in this project is important because it can be used to understand and solve other problems based on man-made concepts such as one of the many unsolved games. Other research has also tackled the poker dataset with the theoretical basis that this dataset is mathematically flexible to model many different scenarios [3]. However, we took a different approach to this and tried focused more on understanding the behaviors that man-made concepts have on machine learning. This was done with the basis that this research might be used past the

context of this class and perhaps be relevant for building programs with the intent of analyzing poker or a different man-made concept on the professional level. As such, we've constructed a harsher penalization scheme that maintains the original 0-9 scale of strength (with 0 being the weakest hand and 9 being the strongest hand) from the dataset documentation in mind. Therefore, if the machine fails to guess accurately for a stronger hand, we then simply calculate the RMSE without normalization of the feature-label space to 0s and 1s such as in [4]. This is also to account for the fact that stronger hands are much more important and rare, and therefore should be given stronger weights.

2 Related work

Much of the previous research done with the poker hand dataset seems to be concerned mostly with the theoretical implications it provides, given that poker has categorical and integer attribute types [3] and can model many different situations [2]. As a result, the poker hand is often deemed an important and convenient dataset in machine learning.

Findler goes in depth on this aspect and portrays poker as a "game of partial information" [2]. He mentions how poker is a "large scale complex problem" due to the layers of depth involved, including the mathematical as well as meta factors such as bluffing [2]. He classifies the poker problem into three main categories of decision making under this uncertain, partial problem of poker: (1) probability and payoff factors, (2) personality variables of the player as well as his or her opponents, and (3) situational variables such as environment and social pressure. He furthers to mention that poker is a game "with incomplete information about past and present situations and the consequences of actions" [2]. With all this uncertainty and justification for poker as an advanced science, Findler makes a good case for the ethos that reinforces the poker dataset as a valuable one to learn from, not only from a philosophical or psychological perspective, but also from a machine learning standpoint given its end-goal challenge of stimulating human cognitive interaction with others in a social environment [2]. From this dataset we decided to play on the concept of "partial information" and try to rank incomplete poker hands of 1-4 cards in addition to full hands of 5 cards.

In his research, we found Disken [4] to also be very interested in the strength of a poker hand. His experiments primarily relied on neural networks and the normalization of the feature-label space to binary 1s and 0s. This was in order to "understand the strength of the hand at a glance" [4]. This transformation would allow matrix multiplication with the labels to produce a more strict scoring scheme of correctness since "only the correctly predicted values will be one and all wrong predictions will be zero" [4]. Disken uses gradient descent with momentum to find the vector of steepest MSE (mean squared error) and also used a back-propagation neural network, placing special priority on the learning rate, validation limit, and max epoch parameters [4]. In his results, Disken finds that the back-propagation neural network worked better by almost double that of the gradient descent and that special attention to the aforementioned parameters played a big difference in the results, the most notable being the validation limit in correlation with the max epoch [4]. What ended up happening was that when the validation limit was too low, it didn't allow the model enough room to train with [4]. With this research in mind, we chose to make sure we allow ample space for hyperparameters to run their course through exhaustive search in order to focus slightly more on accuracy.

As far as model selection goes, research was done to see if the complexity of models had an indication on the degree of overfitting produced by a certain quantity of models. This research was mostly based on Zahálka et al. [3] and their experiment on verifying Domingos' interpretation to Occam's Razor. A widely known theoretical concept called "Occam's Razor" states that "entities should not be multiplied beyond necessity" [3]. While this concept is inherently relevant to machine learning, the interpretation of such has been in debate for quite some time [3]. Domingos' interpretation was that the complexity of models correlates with but does not directly cause more overfitting. In fact, he believes that the number of "tested candidate models" overfit more from a larger set of models than a smaller set, due to the fact that complex models seem to require more data to learn from, and the implicit human error in the resulting assumption that complexity and model size are then directly correlated [3]. Zahálka et

al. [3] work to verify this claim and test on 30 different UCI Machine Learning datasets. They use random model selection and gradient descent learning in addition to grid search across a set number of models and degrees of complexity, making sure that the tests between the number of models and degrees of complexity do not intertwine with dependence. What they found is that Domingos' claim held true and selecting with a larger set of models led to a more biased machine with more overfitting, but that complexity of the models affected the overfitting results very little. As a result, taking this experimental data into account, we chose to use only a few select models for our project to safeguard against this Occam's Razor overfitting that we witnessed in [3].

3 Data sets

We used a training and testing data set of poker hands from the data repository that can be accessed from [1]. From this we took our training set from a file named “poker-hand-training-true.data” which we renamed to “finaldata.txt” in our code. We then took our testing set from the file named “poker-hand-testing.data” which we renamed to “testdata.txt”. Before running the program, it is important to make sure the data sets are saved in the same location as the code itself or the path to the file is defined. Additionally, it would be a good idea to double check that the last line in the data sets is a new, empty line. Failure to do so may result in errors with end of line parsing.

The training set contains a 25,010 by 11 data matrix with the first 10 out of 11 columns corresponding to features with type integer, and the last column corresponding to the labels which are also integers. The testing set contains a 1,000,000 by 11 data matrix with the same setup. Each data instance is delimited by commas. The features are a list of playing card attributes such as suit(1-4) and value(1-13) card pairs, which are represented in the form: [suit of card1, value of card 1, suit of card2, value of card 2, suit of card3, value of card 3, suit of card4, value of card 4, suit of card5, value of card 5]. The label identifies the hand value (0-9) from worst (no poker hand) to best (royal flush). More information about the data set can be seen at the previously mentioned dataset link.

4 Proposed solution

The first step of our program is to parse and store the data. The training and testing data set are read in line by line through a loop. The loop removes the unnecessary formatting characters, and then splits the data by commas into an integer array. The loop then stores the integers in the first $n * 2$ indices of the array into an array of features for each hand of n cards(1-5), and stores the integer in the last index of the array as the label. After the data has been parsed, there is a loop that goes through the main components of the pipeline using the appropriate testing and training variables for each n number of cards in the hand.

Within this loop we then do feature selection via Scikit-learn's SelectKBest algorithm, with the K chosen through an 80%-20% cross validation split. We chose to use regression instead of classification in order to quantify more correct predictions. This is because, while classification will give us integers, regression gives us more interesting values such as 1.8 instead of mapping to 2. Then we can see that this prediction is better than a prediction of 1.6, which might also map to 2 in classification. We measure how good these predictions are by penalization via RMSE (root mean squared error). MSE (mean squared error) measures the variance and RMSE is simply the square root of that and instead of measuring the general distribution and distance it keeps the predicted values normalized with respect to the actual values to effectively give the standard deviation with the formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2}$$

After that we tuned the hyperparameters of each of our models (KNN Regression, Random Forests, and Gradient Boosted Regression Trees) through grid search over values we deemed were reasonable. We ended up sticking with exhaustive search in the range of 1 to 50 over all of the models because it offers a substantial cost-efficient means of analyzing behavioral patterns of the models. This number is also large enough to provide ample room for proper learning so that we didn't replicate the bad results of Table 10 in [4].

The following goes into detail on the models chosen and why they were chosen.

4.1 KNN Regression

This model operates on the concept of K Nearest Neighbors (KNN) that polls nearby data points in the same hyperplane and takes a majority vote. Since this is a regression model, it then outputs a value of the vote. This model is especially useful for the poker dataset since we are interested in similar hand combinations that map to a certain hand type on the hand strength scale from 0 to 9 (with 0 being the weakest and 9 being the strongest).
The regression model this operates on is:

$$f_{KNN}(x) = \frac{1}{K} \sum_{i \in N_k(x)} y_i$$

Where the number of neighbors, K, is our main hyperparameter found through grid search.
This model was chosen since the dimensionality of our dataset is very low (only 10 features excluding the column of labels) and KNN excels in instances where it can afford to store all the data points in memory to map out neighbor correlations between them. Of course this then requires more storage, but due to our focus on accuracy this is negligible.

4.2 Random Forests

Random Forests builds on the concept of Regression Trees and therefore it is necessary to explain that first. Regression Trees have the same regression function rules of a binary tree structure where each node has the form:

$$(x_d < t) or (x_d = t)$$

It then assigns the data case into the appropriate left or right sub-tree accordingly. Each leaf represents a single predicted output which is an average of the data cases in their respective leaf nodes. The main difference between this regressor and the Extra Trees regressor is that in this regressor, the sub sample size is always the same as the original input sample size but the samples are drawn with replacement. The hyperparameters for this regressor are also the number of trees used and maximum depth of each tree. However, for computational feasibility we focused on only the number of trees. This is because the randomized aspects of this regressor causes it to typically be bad with a low number of trees and good with a higher number of trees. We chose this model because random forests are very accurate and have robustness to overfitting due to the large amount of sampling with replacement. However, due to this their computational complexity can cause this to be relatively expensive.

4.3 Gradient Boosted Regression Trees (GBRT)

The general idea of GBRT is to build a sequence of simple regression trees that build off the residual negative effectiveness of the previous tree, with the goal to create a network of trees such that the error is minimized. This also builds off the regression tree formula discussed under the aforementioned "Random Forests," with the main difference being that chaining simple trees together provides a good generalization. Furthermore, GBRT is very robust to overfitting via robust loss functions. GBRT uses the following formula:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x, y_m)$$

"Where $\beta_m, m = 1,2,\ldots,M$ are the basis expansion coefficients, and $b(x, y)$ are simple functions of the multivariate argument$x$, with a set of parameters $y = (y_1, y_2, \ldots, y_M)$" [5].

We chose this model because through our research we saw many people using gradient descent in their pipelines and we thought we would also give it a try but instead use boosting instead of gradient descent with momentum in order to see if an alternative method might produce better results than the ones experimented on by Zahálka in [3]. It turns out that our results are still worse than his, but this may be partly due to the fact that we are using the RMSE error function whereas he is using MSE.

5 Experiments and results

We performed many different experiments in order to correctly identify poker hands and observe resulting behavior. The most interesting experiments that we performed involved observing the results that hands of different sizes had on our regression and our results. The first experiment we performed was to observe the results of feature selection for each hand size. We decided to show our results using tables instead of graphs because our graphs for all 5 cards were either too large for the page limit or too cluttered. Our results can be seen in Table 1.

Table 1: Table showing the results of feature selection for each hand size

Number of Cards:	**5**
Features available	0(suit),1(value),2(suit),3(value),4(suit),5(value),6(suit),7(value),8(suit),9(value)
Feature removed (KNN)	0(suit),2(suit),4(suit)
Feature removed (Ran. Forest)	0(suit),2(suit),4(suit)
Feature removed (GBRT)	0(suit),2(suit),4(suit)
Number of Cards:	**4**
Features available	0(suit),1(value),2(suit),3(value),4(suit),5(value),6(suit),7(value)
Feature removed (KNN)	0(suit),2(suit),4(suit)
Feature removed (Ran. Forest)	0(suit),2(suit),4(suit)
Feature removed (GBRT)	0(suit),2(suit),4(suit)
Number of Cards:	**3**
Features available	0(suit),1(value),2(suit),3(value),4(suit),5(value)
Feature removed (KNN)	0(suit),2(suit),4(suit)
Feature removed (Ran. Forest)	0(suit),2(suit),4(suit)
Feature removed (GBRT)	0(suit),2(suit),4(suit)
Number of Cards:	**2**
Features available	0(suit),1(value),2(suit),3(value)
Feature removed (KNN)	0(suit),2(suit),3(value)
Feature removed (Ran. Forest)	0(suit),2(suit)
Feature removed (GBRT)	0(suit),2(suit)
Number of Cards:	**1**
Features available	0(suit),1(value)
Feature removed (KNN)	0(suit)
Feature removed (Ran. Forest)	0(suit)
Feature removed (GBRT)	0(suit)

The results from Table 1 show us a few things, the first of which is that the card suit features are much noisier than the card value features. We can see this because only one card value was removed from finding the best features to keep using select-k-best feature selection, while at the same time 36 card suits were removed. The suits were a lot more noisy than the values because values are used a lot more often for determining the poker hand value. The only time that suits are considered is during a flush, straight flush, and royal flush with probabilities of 0.0019654, 0.0000138517, and 0.00000153908 (respectively) given the total number of each card and their possible

combinations. Since these are relatively uncommon, suits are frequently just noise, and are therefore frequently discarded from feature selection.

These results also show that the features that are removed (or kept) stay constant through different models and number of cards (with one outcast). The features that are consistently removed through different models show that the models are affected by noisy features in very similar ways. The fact that most of the features removed stay constant as card number varies shows that even when there are less cards in the hand, and therefore greater uncertainty, the existing noisy features are still recognized. Therefore, the models view the features as having a greater effect on the label independently than they do dependently, which we know is not always the case (such as in a flush).

The second experiment we performed was to observe the results of hyperparameter selection for each hand size. Our results can be seen in Table 2.

Table 2: Table showing the results of hyperparameter selection for each hand size

Number of Cards:	**5**	**4**
Optimal Number of Hyperparameters with KNN (n_neighbors)	13	33
Optimal Number of Hyperparameters with Random Forest (n_estimators)	48	48
Optimal Number of Hyperparameters with GBRT (n_estimators)	49	49
Number of Cards:	**3**	**2**
Optimal Number of Hyperparameters with KNN (n_neighbors)	12	43
Optimal Number of Hyperparameters with Random Forest (n_estimators)	30	37
Optimal Number of Hyperparameters with GBRT (n_estimators)	49	49
Number of Cards:	**1**	
Optimal Number of Hyperparameters with KNN (n_neighbors)	43	
Optimal Number of Hyperparameters with Random Forest (n_estimators)	18	
Optimal Number of Hyperparameters with GBRT (n_estimators)	1	

The results from Table 2 show the some of the major differences in the different model types. As a general trend, both random forest regression and gradient boosting regression use fewer hyperparameters (number of estimators) the fewer cards there are. Because fewer cards mean more uncertainty and also less results to train on, this suggests that the number of estimators required for these models is more dependent on fitting to the data than predicting on it.

The number of optimal parameters in KNN Regression do not have as predictable of a trend and are a lot more chaotic. Due to the fact that there is no obvious trend as the number of cards decreases, we can infer that the number of neighbors required for KNN regression is highly dependent on both fitting to the data *and* predicting on it.

The next experiment we performed was to actually observe the results of our RMSE score based on our cross validated data for each hand size. Our results can be seen in Table 3. Note that since we are using RMSE, a smaller value signifies a better score.

Table 3: Table showing the results of RMSE on cross validated training data

Number of Cards:	**5**	**4**
RMSE with KNN Regression	0.707747233422	0.724331970815
RMSE with Random Forest Regression	0.665501139672	0.714543189254
RMSE with Gradient Boosted Regression Trees	0.728557119447	0.73943840356
Number of Cards:	**3**	**2**
RMSE with KNN Regression	0.736280887346	0.791457333807
RMSE with Random Forest Regression	0.736533221459	0.750939042106
RMSE with Gradient Boosted Regression Trees	0.756135506183	0.76481894351
Number of Cards:	**1**	
RMSE with KNN Regression	0.791457333807	
RMSE with Random Forest Regression	0.774387677767	
RMSE with Gradient Boosted Regression Trees	0.774387217974	

The results from table 3 show us that as there are less cards and therefore greater uncertainty, our models get worse at predicting the card hands. This is trivial because predicting with more uncertainty should generally always produce worse results than predicting with less uncertainty. The one surprising thing about these results is that the score only gets slightly worse with significantly more uncertainty. This is because in poker, the higher value hands generally require a larger number of correct cards. This means that missing a few of the higher value hands due to greater uncertainty won't affect the RMSE too much a lot due to their rarity.

The final experiment we performed was to observe the results of our RMSE score based on our test data for each hand size. Our results can be seen in Table 4. Note that since we are using RMSE, a smaller value signifies a better score.

Table 4: Table showing the results of RMSE on the test data

Number of Cards:	**5**	**4**
RMSE with KNN Regression	0.712386401201	0.723396544192
RMSE with Random Forest Regression	0.663533037293	0.710179242587
RMSE with Gradient Boosted Regression Trees	0.727862463518	0.737644973623
Number of Cards:	**3**	**2**
RMSE with KNN Regression	0.730466783791	0.790285967711
RMSE with Random Forest Regression	0.727440875211	0.74878891721
RMSE with Gradient Boosted Regression Trees	0.75403553782	0.764842402386

Number of Cards:	1
RMSE with KNN Regression	0.790285967711
RMSE with Random Forest Regression	0.773549934875
RMSE with Gradient Boosted Regression Trees	0.77341172637

The results from Table 4 validate our results from Table 3. Since table 4 uses results based on the original testing set which contains 1,000,000 instances, we know that our model is not overfit since the results are adequately similar for every corresponding RMSE. We can also see that our random forest regression was our most successful one because it produces the best RMSE out of all our models for 2-5 cards, and produces very close to the best RMSE for 1 card. We can also see that our best RMSE is from random forest regression with 5 cards, which produces an RMSE of 0.663533037293.

6 Discussion and conclusions:

In conclusion, we found several interesting behaviors from our experiments. First, we found that during feature selection, the features that were removed were almost always the suits. This is because the only time that suits matter is during a flush, which is very rare and results in suits being relatively noisy. We also found that as we had fewer cards in the poker hand, the correctness of our regression model did go down, but not significantly. This is because even though fewer cards in the hand create much greater uncertainty, this only decreases the likelihood of getting a good hand in relatively few cases.

Due to these behaviors of the man-made concept of poker hands, we observed our best scores that resulted from measuring RMSE to stem from Random Forest Regression, which achieved the scores of 0.665501139672 on the training set and 0.663533037293 on the test set, which is not horrible but not perfect. Here we can see that the test set error is actually lower than the training set error. This is likely due to random chance allowing the test set to have easier cases than the training set. Additionally, this high value for the RMSE is also very indicative of our harsh penalization scheme of not normalizing our feature space before fitting the models. We could have done what some other researchers have done and map the data into a binary space, but this would produce less interesting results and would not have helped us as much in being able to understand other problems based on artificial concepts. As a result, we believe that in order to achieve a more ideal error score in identifying poker hands, we might need to use deep learning to train a regressor or classifier to be able to easily identify and deal with similar troubling man-made concepts. Since these artificial concepts we have so heavily talked about are indeed “man-made,” perhaps the obvious choice for a model that can understand these complex interactions and correlations is a neural network due to their goal of artificially replicating of the human brain.

7 References:

[1] Cattral, Robert, and Franz Oppacher. "Poker Hand Data Set." *UCI Machine Learning Repository*. Carleton University, Department of Computer Science, 01 Jan. 2007. Web. 30 Apr. 2016. <http://archive.ics.uci.edu/ml/datasets/Poker Hand>.

[2] Findler, Nicholas V. Studies in machine cognition using the game of poker. State Univ. of New York at Buffalo, Buffalo 1977, New York, NY, pp.230-245, 1977, <http://dx.doi.org/10.1145/359461.363617>

[3] Zahálka, Jan, and Filip Železný. An Experimental Test of Occam’s Razor in Classification. Czech Technical University, 2010. Web. <http://ida.felk.cvut.cz/zelezny/pubs/mlj.10b.pdf>.

[4] Dişken, Gökay. Predicting Poker Hand's Strength with Artificial Neural Networks. Adana Bilim Ve Teknoloji Üniversitesi, 2010. <https://eembdersler.files.wordpress.com/2010/09/2014913024-gokaydisken-project.pdf>.

[5] Ogutu, Joseph O, Hans-Peter Piepho, and Torben Schulz-Streeck. "A Comparison of Random Forests, Boosting and Support Vector Machines for Genomic Selection." *BMC Proceedings* 5.Suppl 3 (2011): S11. *PMC*. Web. 1 May 2016. <http://doi.org/10.1186/1753-6561-5-S3-S11>