“Do I Hear 3NT?”: Learning a Bridge Bidder

Greg White (SUID: gregw)

I. Introduction

This is an application project, applying machine learning methods to the problem of optimally bidding in Contract Bridge. The motivation is to advance progress on an imperfect information game-playing problem on which AI techniques apparently have yet to overtake expert human players, and to attempt to apply machine learning methods to an application that has been the focus of relatively more research in other AI fields [1]. Bidding in Bridge is a nuanced problem that takes human players extensive experience to conquer: players have imperfect information (they don’t know which cards are held by each other player), and bids both represent possible contracts that would dictate the play phase of the game and convey information to one’s partner. That makes it both interesting and challenging. To bound the scope of the problem and find problems that should be well-suited to machine learning algorithms, this project breaks the problem of Bridge bidding into two separate sub-problems: (1) knowing only the 13 cards in one’s hand before anyone else has made a non-Pass bid, with what should one start the bidding?, and (2) knowing the 13 cards in one’s hand and the 13 cards in one’s partner’s hand, in what final contract should one’s team end up playing the hand? These sub-problems are important parts of the complexity of the overall bidding problem. If one could perform both of these sub-problems well, the overall bidding problem becomes one of how to sequentially bid to share enough information about one’s hand with one’s partner to determine the features that would allow one to classify the optimal final contract.

Specifically, this project attempted to learn one multi-class classifiers for each of those sub-problems. The input for the opening bid problem algorithm is a string describing the 13 cards held in one bridge hand. We then use a softmax regression to output a predicted bid within the set of 36 valid opening bids (Pass, and bids of the form [1-7][C|D|H|S|NT]). The input for the final contract problem algorithm is a string describing the 2 13 card hands held by one team. We then attempt to use a softmax regression to output a predicted optimal final contract within the same set of 36 valid final contracts (ignoring the doubling and redoubling bids).

II. Related Work

Prior research has proposed a variety of methods to solve all or part of Bridge bidding [1]. Two notable papers use Monte Carlo simulation methods and perfect information game playing search algorithms to bid based on the expected value of the resulting contract played through the play phase for the probability-weighted outcomes of the bidding, we describe those in more detail below. Other methods that have been proposed include hypothetical interface functions with constraint logic programming to attempt to maximize gain by cooperating with the partner and minimize loss by competing with the opponents [2], and langauge processing methods and a logic-based model [3].

Matthew Ginsberg proposed an approach to bidding based on matching a database of prior deals with the bidding thusfar to generate a set of example deals consistent with the bidding, simulate the remainder of the bidding according to that past deal, and then solving the play phase as if all players had perfect information and played optimally. It makes the bid that has the highest expected value based on how the remainder of the bid phase is expected to play out [4]. This method is interesting, though does not appear to apply learning methods to generalize a likely bid pattern effectively beyond the deals in the database. It also does not take into account that the agent is playing with and attempting to communicate with a partner.

Asaf Amit and Shaul Markovitch proposed an approach that builds on Ginsberg’s, but leverages the fact that the agent plays with a partner and can use bids to communicate information to his/her partner [5]. In particular, they simulate the decision process of each agent, including allowing a bid that conveys information to a partner to limit the set of possible states the partner agent matches against the database considers and searches to only those consistent with the information portrayed by that bid. Their approach uses a co-learning algorithm to have agents cooperatively learn a decision network.

While we believe the factors we mentioned above make bidding in Bridge an interesting domain for machine learning techniques, there are also a few reasons the overall problem may not be suited to a pure machine learning approach. The characteristics of the bid phase (sequential bidding, bids convey information as well as or instead of being proposed final contracts, cooperative and adversary agents) may better lend themselves to a state-based, variable-based, and/or logic-based AI approach, with machine learning as part of the training

of the model. Also, in competitive tournament play, the bidder's partner must explain exactly what a bid means in a set of allowable bidding conventions to the opposing team if asked: this could be difficult or impossible with a model that classifies bids based on a set of features and weights.

A. Datset and Processing

The data that is being used comes from the annual World Computer Bridge Championships tournament, and a collection of tournaments and online non-tournament games played by human players and computer programs and made available by the game hosts in a standardized raw text file format to describe bridge games called Portable Bridge Notation (PBN). The data has been collected and aggregated from many files across a variety of websites, there was not one central repository. The dataset includes World Computer Bridge Championship historical results [6] back to their first availability in 2008, as well as many other tournaments tracked by those who oversee the PBN standards [7] [8]. Each hand played was parsed to generate up to 4 training/testing examples for our opening bid problem (each of the players up to and including the first non-Pass bid can be used), and each hand was analyzed to generate at least one training/testing example for our final bid problem, for the team with the higher optimal score.

There are a few challenges associated with generating ground truth labels for a supervised learning algorithm with each of the opening bid and the final contract problems.

First and primarily related to the opening bid problem, Bridge players (human and the current state-of-the-art computer programs) use multiple "bidding conventions", agreed upon with one's partner in advance of playing and known to the opposing team, that provide guidelines for what certain specific bids mean. Some of these are called "artificial conventions", because a bid may mean something other than a novice's intuition would suggest. For example, some players bid "1 Club" meaning an especially strong hand for an opening bid, others bid it meaning the weakest possible opening bid. Some players bid 2 of a suit to mean a stronger hand than 1 of that suit would indicate, others bit it to mean a much weaker hand than 1 of that suit would indicate, but with more cards in that suit. There are also differences beyond just these counterintuitive conventions in how different pairs bid, including how strong a player's hand should be to start with a specific strong opening bid.

Second, the notion of a ground truth label for the optimal final contract is less easily observed in the data than the opening bid. The way the players bid when the hand was recorded may reflect specifics of the game situation that we do not want our model to capture (e.g., score though the previous hand, whether one side is "vulnerable"), and the players bidding are not able to see their partners' cards.

Because of these challenges, in addition to parsing the raw data about how the hand was played from the text files for our training and testing sets into usable data structures, we've processed the raw data about bids made by computer and human players in a few ways.

1) Where possible, we use data from tournaments as opposed to casual play. Bridge tournaments are played with pairs of partners compared on their performance on the same exact card deal, which for our purposes means we are likely to get multiple pairs of partners making opening bids on the same hand, and can be thoughtful about attempting to not capture idiosyncracies in how specific players bid differently than others in our model. For this same reason we are only training and testing the model on hands for which the dataset has only one observed bid, and are reporting error for both all hands and for only hands for which we have more than one instance of the ground truth label.
2) To generate ground truth labels for our examples for the final contract problem, for each unique deal, we use a third party C++ implementation of a perfect-information Minimax solver for the play phase of the game [9], which is able to compute the optimal contract for a hand assuming that during the play phase all players knew which cards are held by the others and all played optimally. This makes each parsing each example take a relatively long time (finding the optimal contract requires solving many large state space search problems and comparing the results).

B. Features

We've used two different feature templates to represent the 13-card raw input hand of the player for the opening bid problem. We've also attempted to use essentially the same feature templates- summing across both hands- to represent the 2 13-card raw input hands of the team for the final contract problem.

1) Raw cards indicator feature: This feature vector has $n = 53$ features (52 cards plus a bias term), and is 1 for each of the 13 cards the player has in his/her hand, and 0 otherwise. This feature vector should require a large amount of data to train, and it should not generalize well because it does not account for the interactions between cards, but it is a relatively simple feature vector that requires no domain knowledge.
2) Indicator features using limited domain knowledge: This feature vector has $n = 1+4*14+4*11+41$

features currently. The first 56 features after the bias term are indicator features for the number of cards in each suit. Each player has between 0 and 13, inclusive, cards in each suit (as does each team). The next 44 features are indicator features for the number of "high card points" in each suit. High card points are counted as 4 for an ace, 3 for a king, 2 for a queen, and 1 for a jack, and are a simple way that human players assess the strength of their hands and decide what to bid. In each suit, there are between 0 and 10, inclusive, high card points. The next 41 features are indicator features for the total number of high card points: total high card points is between 0 and 40 inclusive (and is actually a maximum of 37 in the opening bid problem because the player has only 13 cards, 40 is only possible for the final contract problem when considering both hands of the team).

III. Methods

There are a finite, fixed number of possible valid opening bids (36), so we attempted to use a softmax regression to model each problem as if the classification is distributed according to a multinomial distribution. This also has the advantage of modeling the probability of each class: a benefit in our problem because there are often multiple bids that would be reasonable. This model attempts to maximize the log-likelihood of the training dataset $\{x^{(i)}, y^{(i)}\}$ where $x^{(i)}$ is a feature vector representing the features of the raw input hand of 13 cards of the player for the opening bid problem or the raw input of 2 13 card hands of the team for the final contract problem:

$$l(\theta) = \sum_{i=1}^{m} \log \prod_{l=1}^{36} \left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{36} e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}} \quad (1)$$

We're using a vectorized stochastic gradient ascent update algorithm which we have implemented ourselves, derived by taking the partial derivative with respect to each paramater. There are 36 possible classes, but the model is explicitly paramaterized by 35 parameter vectors, with the probability of the final class being 1 minus the sum of the probability of the other 35 classes. Let n represent the number of features for each input hand, including the bias term. Then our vectorized parameter matrix is:

$$\theta = \begin{bmatrix} \dots & \theta_1^T & \dots \\ \dots & \theta_2^T & \dots \\ & \vdots & \\ \dots & \theta_{35}^T & \dots \end{bmatrix} \text{ where each } \theta_j^T \in \mathbb{R}^n, \quad (2)$$

and let $x^{(i)} \in \mathbb{R}^n$ be the feature vector for the ith example in the training set.

The gradient ascent update rule for a single training example is:

$$\theta := \theta + \alpha \left(\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \frac{1}{1 + \sum_{j=1}^{35} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ \vdots \\ e^{\theta_{y^{(i)}-1}^T x^{(i)}} \\ e^{\theta_{y^{(i)}}^T x^{(i)}} \\ e^{\theta_{y^{(i)}+1}^T x^{(i)}} \\ \vdots \\ e^{\theta_{35}^T x^{(i)}} \end{bmatrix} \right) (x^{(i)})^T \quad (3)$$

where the 1 in the vector of otherwise all 0s is at the position $y^{(i)}$.

For hyperparameters, the models use: $\alpha = 0.05$, and the model is trained until either the change in the norm of the parameter matrix θ is less than 0.01 or 1,000 iterations through the entire training set are complete. The opening bid classifiers reach 1,000 iterations before reaching the convergence test. These were tested anecdotally on early iterations of the model and chosen because they offer an attractive trade-off between run-time and accuracy.

IV. Experiments and Discussion

A. Opening Bid Problem

The primary metric we measured was accuracy: the percentage of examples on which our classifier predicts the same label as the actual or optimal label. We would note that this might understate the effectiveness of our classifier, as many hands have multiple possible reasonable bids, so evaluating only the accuracy metric does not capture that if the classifier predicts a reasonable but not the actual optimal label with the highest probability and the optimal label with slightly lower probability, this is in a sense less ineffective than represented by the accuracy metric, which classifies that performance as inacurate. For this reason, we've also attempted to contextualize the accuracy metric with a qualitative analysis of the confusion matrix of the opening bid classifier. Also, as noted above because there can be multiple reasonable labels that different players will assign to a given hand for the opening bid problem, we report both the accuracy on the entire training and dev datasets, and the accuracy on only the subset of the training or dev dataset for which we have more than one instance of the same label in the dataset.

Table IV-A presents the accuracy of our opening bid problem classifiers using each of the feature templates on as much data as we've been able to collect: training on approximately 17K examples and testing on approximately 5K examples.

TABLE I
OPENING BID PROBLEM ACCURACY RATES

	Training dataset (70% of data) accuracy		Dev dataset (20% of data) accuracy	
	All hands	All labels agreed	All hands	All labels agreed
m	16,602	9,552	4,744	2,733
Raw cards features	84.0%	89.6%	83.1%	88.9%
Domain-knowledge features	89.7%	94.7%	87.5%	93.4%

Overall, we are pleased with the accuracy rates the opening bid classifiers achieve, particularly the 93% accuracy of the domain knowledge feature template on the dev dataset where we have higher confidence in the correctness of the label.

As expected, both feature templates have significnatly (approximately 600 basis points) higher accuracy when only tested on the dev datasets for which we have multiple agreeing labels. This suggests that our concerns about the potential for singular labels in the dataset to capture idiosyncracies of a team's specific bidding conventions are reasonable.

Both models also generalize from the train to the dev dataset well, with only a small decrease in accuracy. We are surprised that the raw card feature template generalizes as well as it does, suggesting that the opening bid problem may be a simpler one than it seems.

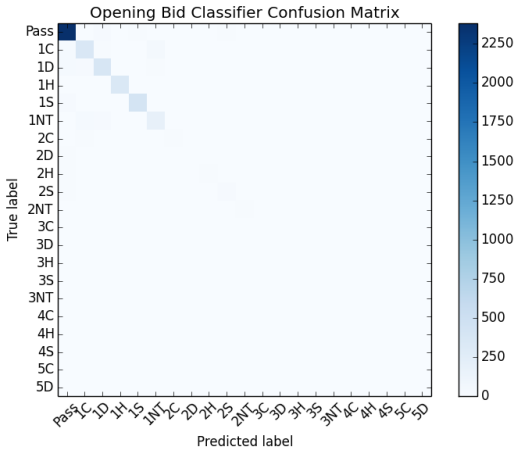
On a smaller dataset of only the computer bridge tournaments (which are more predictable than all data because the computer agents are more likely to adhere strictly to a set of rules than human players, and because tournaments let us only use hands that have at least 2 different players bid the same way), the domain-knowledge features model was able to achieve no error on the training dataset (approximately 1,300 hands), but still approximately 10% error on the 365 hand dev dataset. With fewer training iterations (100 iterations through all training examples vs. 1,000), the errors achieved were similar.

To contextualize the accuracy rates and perform error analysis, we've also analyzed the confusion matrices for the opening bid classifiers. We used the `confusion_matrix` method from the `Scikit-learn` library [10] to efficiently calculate the confusion matrices and adapted the code example given in the `Scikit-learn` documentation [11] to plot a confusion matrix using `matplotlib`. This is the only place we've used `Scikit-learn` in the project.

First, consider the non-normalized (simply count of frequency) confusion matrix for the domain knowledge feature template on the dev dataset. Note, the other permutations of non-normalized confusion matrices (by feature template, by dataset) look very similar so we are only showing one for the sake of brevity.

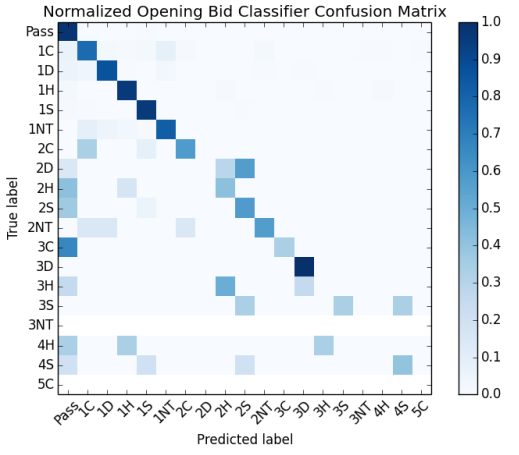
We would note that there are many fewer than 36 frequent labels in this classification problem, and Pass is a much more frequent label than any other. This makes intuitive sense (bidding is sequential, so players typically open bidding low before they know if their partner has a good hand), and it likely helps to simplify the classification problem and improve our model's accuracy.

Fig. 1. Non-normalized confusion matrix: domain features, dev set



To better understand the misclassifications our model makes, we also present the normalized (normalized across the max of each true label row) confusion matrix for the most accurate model we have for the opening bid problem: the domain knowledge features on the dev dataset with only more than one agreed label.

Fig. 2. Normalized confusion matrix: domain features, dev set



Based on this confusion matrix, we would call out that even the misclassifications are on types of situations that human bidders also consider difficult decisions, and that the model classifies even many of those correctly. For example, bidding Pass or 2 or 3 of a suit with a weak hand and many cards in one suit, or bidding 1 No Trump vs. bidding 1 of a strong suit with a particularly strong hand.

B. Final Contract Problem

Our attempt at modeling the final contract problem with a softmax regression and these feature templates has not been successful thusfar. The models have accuracy rates lower than 25%, and they devolve to predicting only a small number of classes. We believe there are two reasons the final contract problem performance is so much worse than the opening bid problem.

The final contract problem is a more complicated multi-class classification problem than the opening bid problem is. In particular, there are actually 36 valid bids that are all reasonably possible, vs. the opening bid problem which actually has a relatively small number of classes within the 36 that are much more likely than the others and some that are almost never the correct label.

We also believe that a softmax regression to predict one of 36 classes does not capture some of the structure of the problem that we could use to guide a better model choice. Human players approach the sequential bidding process as more of a two-stage process: first find a suit fit, then maximize the denomination of the bid. If continuing this project, we would attempt to implement a different learning model to incorporate this idea and the structure of the problem that there's an optimal suit and within that suit an optimal denomination: our hypothesis is that either a neural network with a single hidden layer and softmax activation on both the hidden layer and the output layer, or a model of sequential softmax regressions, where the first yields the optimal suit and the second (maybe with varied parameters and/or features for each of the 5 suits) yields the optimal denomination within that suit.

V. Conclusion and Future Work

In summary, we implemented a softmax regression model to attempt to classify the optimal opening bid given a player's hand and optimal final contract given a team's hand, as sub-problems of optimally bidding in Bridge. The opening bid problem classifier with only simple domain knowledge incorporated in the features achieves 93% accuracy out of 36 possible classes on the subset of our dev set for which we have more than one instance of our ground truth label. Even the misclassifications that the model makes are in instances that many human players would consider borderline, and the model classifies even many of these situations correctly. We are pleased with this result on the opening bid problem. Our attempt to solve the problem of the optimal final contract for a team using similar methods did not have nearly as much success: we believe the model we attempted to use does not take into account some of the structure of the problem that could be helpful in informing a better model to use, and think that (1) sequential softmax regressions to first classify the suit and then to classify the denomination within the suit or (2) a neural network might be more accurate approaches.

References

[1] P. Bethe, "The state of automated bridge play," *New York University, New York, NY, USA, Tech. Rep*, 2010.

[2] T. Ando, N. Kobayashi, and T. Uehara, "Cooperation and competition of agents in the auction of computer bridge," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 86, no. 12, pp. 76–86, 2003.

[3] W. Jamroga, "Modelling artificial intelligence on a case of bridge card play bidding," in *Proceedings of the 8th International Workshop on Intelligent Information Systems*, pp. 267–277, 1999.

[4] M. L. Ginsberg, "Gib: Steps toward an expert-level bridge-playing program," in *IJCAI*, pp. 584–593, 1999.

[5] A. Amit and S. Markovitch, "Learning to bid in bridge," *Machine Learning*, vol. 63, no. 3, pp. 287–327, 2006.

[6] "2016 world computer bridge championship results." https://allevybridge.com/allevy/computerbridge/2016scores.html, 2017.

[7] "Pbn archive." http://www.angelfire.com/games2/pbnarchive/pbn/, 2017.

[8] "Pbn databases." http://home.claranet.nl/users/veugent/pbn/pbn_databases.htm, 2017.

[9] B. Haglund and S. Hein, "Double dummy solver written in c++ for the bridge card game." https://github.com/dds-bridge/dds, 2017.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] "sklearn - plot confusion matrix." http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html, 2017.