SkatBot: Teaching a Computer to Play Cards

Ivan Leung, Pedro Milani, Ben-han Sung

Abstract

This project taught a computer to play a popular German card game Skat*. Gameplay data was collected using a computer gameplay simulation and reduced to a set of appropriate feature vectors. We applied Softmax regression and support vector machines (SVM) to predict the best card to play each turn. Cross-validation predicted human card outputs with around 48% (suit) and 64% (rank) accuracy. For additional validation, we created an AI to play the game using these algorithms. In a 75-game experiment, our* SkatBot *outperformed the baseline random algorithm by 7.44 points on average, but scored lower than the human team by 6.44-points. However, when the intrinsic variance of* Skat *scores is considered, the t-test of difference of means (at α = 0.05) concludes that there is a significant difference between the ability of* SkatBot *and the random algorithm, but not between* SkatBot *and the human team.*

Introduction

Skat is a 3-player card game where two players team up against one. The cards are ordered by suit (♦, ♥, ♠, ♣) and by rank (7, 8, 9, Q, K, 10, A, J), and each card rank has a point value. The standalone player chooses one suit as trump suit, which beats all cards of other suits. Each player starts with 10 cards and plays one card per round. In each round, players must play a card of the same suit as the starting card unless they do not have any. The winner of a round wins the cards played and starts the next round. The team with the most points after 10 rounds wins the game

One unique aspect of *Skat* is that no communication is allowed between players. This means no one knows what cards the other players have on their hands. However, knowing how the cards are probabilistically distributed is crucial to winning the game. There are some complex card-counting and non-verbal communication strategies that we expect the computer to learn by analyzing human playing patterns.

Objective

We want to train the computer to predict the best card to play per turn. The prediction algorithm can be applied repeatedly to play an entire game of *Skat*. A simplifying assumption we make is that the computer always plays as a member of the two-person team. Our card prediction strategy is a two-step process: 1) predict the suit of the best card to play, and 2) predict the rank, conditioned on the chosen suit. From a dataset of 41 games between humans, we extracted 282 training examples for suit prediction and 497 for rank prediction. Each example represents one instance when the player had a choice and is labeled with the actual human decision.

Features

Suit examples have 29 features and rank examples have 41, intuitively chosen to provide essential information such as number of cards on hand for each suit, the order of play, and whether cards on hand are winning (Appendix 1).

Models

Choosing the suit (k = 4) and the rank (k = 11) are two multiclass classification problems. We used two algorithms:

1. Softmax Regression with L2 regularization. We used gradient ascent to maximize:

$$l(\theta) = \sum_{i=1}^{m}\left\{\sum_{p=1}^{k}\left[1\{y^{(i)} = p\} * \left(\theta_p^T x^{(i)} - \log\left(\sum_{j=1}^{k} exp\left(\theta_j^T x^{(i)}\right)\right)\right)\right]\right\} - C\sum_{j=1}^{k}\theta_j^T\theta_j$$

2. Multi-class C-SVM. We used LibSVM's implementation with three different kernels
   a) Linear: $K(x,z) = x^T z$
   b) Polynomial: $K(x,z) = (x^T z + a)^d$

c) Gaussian: $K(x,z) = \mathrm{e}^{-\gamma*\|x-z\|^2}$

We did not consider using generative models. Naive Bayes is not appropriate because not every feature we have is binary. On the other hand, since most features are binary, we did not think they would follow a strong multivariate Gaussian distribution, so we ruled out Gaussian Discriminant Analysis.

Results

We used 10-fold cross-validation to determine the optimal model parameters. We used test accuracy and test illegal (the times when the algorithm chose an illegal suit/rank) to evaluate the results, with the goal of maximizing accuracy and minimizing illegal predictions. Fig. 1 and 2 show sample results (left contour for accuracy, right contour for illegal) for SVM and softmax regression respectively. Table 1 summarizes the training and testing results for the optimal model parameters of the respective algorithms.

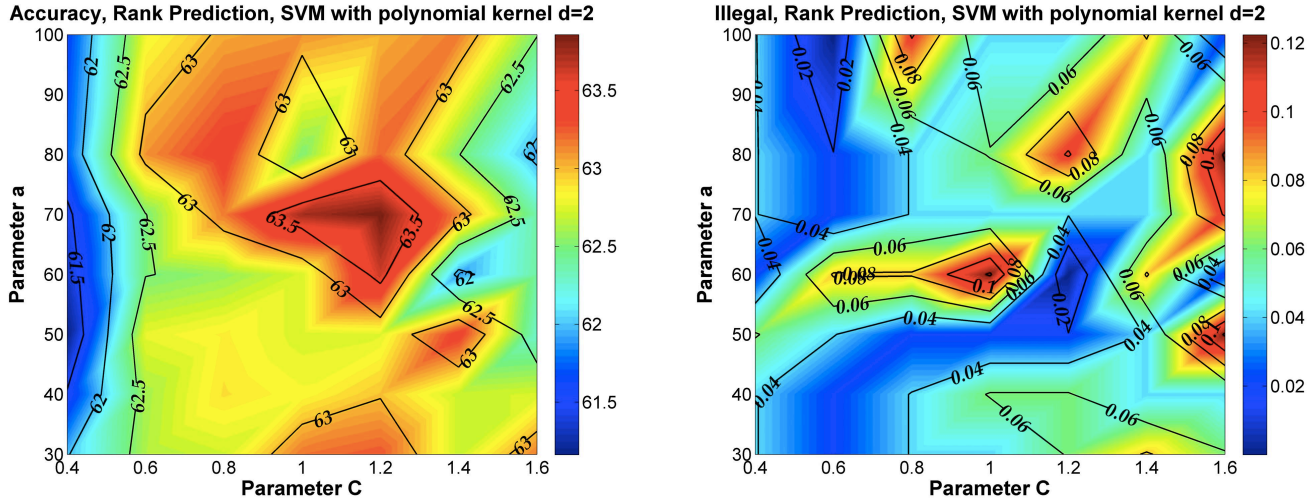

Fig. 1. Contour map for % accuracy (left) and % illegal (right) for SVM

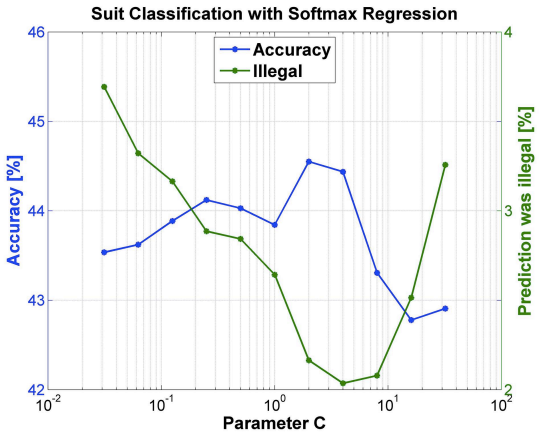

Fig. 2. % Accuracy (blue) and % Illegal (green)

While this cross-validation is commonly used to assess the performance machine learning applications, we reckon that the MSE might not be a good metric in our case. During games of *Skat*, there are times when more than one card can be a good play. Because our data set only marks the card that was actually played as the "correct" output, the calculated error ignores the possibility of multiple valid outputs. Furthermore, there were some training examples where the human player chose a poor card; if the computer chose a better card, this would be considered a "wrong" choice by the above metric.

Model		**Suit Prediction**				**Rank Prediction**			
		Test Accuracy	**Training Accuracy**	**Test Illegal**	**Training Illegal**	**Test Accuracy**	**Training Accuracy**	**Test Illegal**	**Training Illegal**
Softmax		45.5%	57.1%	2.0%	1.1%	64.7%	77.3%	0.4%	0.4%
SVM	**Linear**	43.7%	60.6%	1.8%	0.3%	63.5%	83.5%	0.0%	0.0%
	Polynomial	48.6%	74.8%	0.7%	0.0%	63.9%	87.1%	0.0%	0.0%
	Gaussian	48.3%	68.1%	0.3%	0.4%	63.6%	84.5%	0.3%	0.0%

Table 1. Summary of test and train MSE of suit and rank predictions using different algorithms.

Discussion

As another way to measure the performance of each prediction algorithm, we played against a computer that used each of the four algorithms and counted the number of moves we unanimously agreed were suboptimal. From 18 suit decisions and 46 rank decisions observed in 3 games, we found that Softmax regression made the fewest obvious mistakes (Table 2). So, we chose the Softmax algorithm for our last round of experimental testing.

	% suboptimal moves	
	Suit	Rank
Softmax	0.06	0.02
SVM-linear	0.11	0.13
SVM-poly	0.11	0.17
SVM-Gaussian	0.11	0.15

Table 2. % sub-optimized moves of algorithms

This last round consisted of validating the *SkatBot*'s performance in real gameplay. We compared the scores earned by a baseline random algorithm, SkatBot (Softmax regression), and a human team against the same human player in 25 games each. The following is the setup for our experimental validation:

Null hypothesis: Opponent 1 and Opponent Skatbot have the same mean *Skat* score. h_0: $\mu_{O1} = \mu_S$ (1)

Alternate hypothesis: h_a: $\mu_{O2} \neq \mu_S$ (2)

To facilitate statistical power calculation, we also assumed a fixed effect model. In other words, we assumed that each algorithm imposes a fixed effect δ on the final *Skat* score *S*:

$S_{O1} = \mu + \delta_{O1} + \varepsilon$, where $E[S_{O1}] = \mu_{O1}$, and $\varepsilon \sim N(0, \sigma^2)$ (3)

$S_S = \mu + \delta_S + \varepsilon$, where $E[S_S] = \mu_S$, and $\varepsilon \sim N(0, \sigma^2)$ (4)

$\delta_{O1} + \delta_S = 0$ (5)

Furthermore, we expressed the expected difference between two *Skat* scores as the absolute difference between the opponents' effects. Thus we define:

$\delta \equiv |E[S_{O1}] - E[S_S]| = |\delta_{O1} - \delta_S|$ (6)

Table 3 summarizes the key findings of our experimental validation. In one set of test, we took the random algorithm to be Opponent 1, and in the second set of test, the human team played as Opponent 1. The mean score of SkatBot was 7.48 points higher than that of the random, and the mean score of human players was 6.44 points higher than SkatBot. When we compared SkatBot and random, we rejected h_0 at the $\alpha = 0.05$ level. However, when we compared humans and SkatBot, we failed to reject h_0 at the $\alpha = 0.05$ level, because of the high variance of the 25 games with human players.

Metric	***Random***	***SkatBot***	Human
Mean score (points)	36.24	43.72	50.16
Standard deviation	20.88	17.55	23.88
p-value	0.022	-	0.141
Statistical power (δ = 15)	0.980	-	0.697

Table 3: summary of validation experiment testing for h_0: $\mu_{O1} = \mu_{O2}$

We note that failing to reject h_0 is not sufficient to conclude that Softmax and human players yield comparable performance in *Skat*. We also need to show that under our experimental conditions, we have a high probability of rejecting h_0 for some acceptable margin δ. The statistical power given an effect difference of δ is expressed as:

$$power = P\left(\frac{\left|\frac{1}{n_{O1}}\left(\sum_{i=1}^{n_{O1}} s_{O1i}\right) - \frac{1}{n_S}\left(\sum_{j=1}^{n_S} s_{Sj}\right)\right| - \delta}{s\sqrt{1/n_{O1} + 1/n_S}} > t_{n_{O2}+n_S-2}^{1-\frac{\alpha}{2}} - \frac{\delta}{s\sqrt{1/n_{O1} + 1/n_S}} \middle| \delta\right) \quad (7)$$

From experience playing *Skat*, we would like SkatBot to be comparable to human players within a 5-point margin. However, due to the high intrinsic variance of *Skat* scores, as seen from the high standard deviation of games played against the same opponent, statistical power is only 37.4% at $\delta = 5$. At $\delta = 15$, we have approximately 70% power, which leads us to believe that the true score difference between SkatBot and human players is less than 15 points. Though not ideal, a 15-point margin is still well within the score variance for the same opponent (which ranges from 17.55 to 23.88 points).

Conclusions and Future Work

We applied supervised learning to the problem of creating an AI to play a card game. Our results were encouraging (since we significantly outperform the baseline), but further work is needed to reach the theoretical bound (which is getting as many points as the humans who trained the algorithm).

We used cross-validation on shrinkage parameters to select the most relevant features for our learning algorithms; we did not have time to do a detailed feature analysis, so this could be an area of improvement. Indeed, during our experimental validation, we observed instances where our prediction algorithm made questionable suit predictions (all rank predictions, though, seemed to be acceptable, so the suit features probably need some work). We could also try using reinforcement learning in the future, instead of supervised learning. The difficulty of this approach is representing all the possible game states and actions; however, it would be interesting, from a theoretical standpoint, to find whether a computer learning by itself can outperform a computer being taught by humans. A final possibility for future work would be to relax some simplifying gameplay assumptions, so our *SkatBot* could play in any situations.

Appendix 1: features

Suit features: (i = {♦, ♥, ♠, ♣})	Rank features: (j = {7, 8, 9, Q, K, 10, A, J})
1-4: # cards in suit i on hand	1: Am I playing first?
5-8: # cards in suit i remaining & not on hand	2: # points on the table
9-12: Have winning card in suit i?	3: Has opponent played?
13-16: Has opponent played a different suit on top of suit i?	4: Has teammate played?
17-20: Has teammate played a different suit on top of suit i?	5: Is my team winning?
21: Am I playing first?	6: Has opponent played a different suit on top of this suit?
22: # points on the table	7: Has teammate played a different suit on top of this suit?
23: Has opponent played?	8: How many cards of suit are remaining in game and not on hand?
24: Has teammate played?	9-19: Do I have this card j
25: Is my team winning?	20-30: Is card j a winning card?
26-29: Do I have a card with more than 10 points in suit i?	31-41: If I have card j, does card j beat opponent's card on the table?

Scripts

Our datasets, MATLAB scripts, and code for the game are publicly available online here: https://github.com/bhnascar/SkatBot

References

Ng, Andrew. Course notes for CS229, Fall 2014, Stanford CA.

Owen, Art. "One categorical predictor at k levels". Course notes for Stats 305, Fall 2013, Stanford, CA

McLeod, John. “Skat”. August 2014. http://www.pagat.com/schafk/skat.html