Dominion Machine Learning Project

Tyler Gill, Krista Purser, Wayne Robison
CS 478, Winter 2013
Department of Computer Science
Brigham Young University

Abstract

Dominion is a card game in which players choose cards to buy in order to improve their deck. In this paper we use machine learning techniques to attempt to solve the problem of which cards to buy in Dominion. We discuss the primary challenges in solving this problem, namely automating collection of data and deciding on appropriate input and output features. Though in the end we were not able to create an accurate predictor, we obtained promising results for learning when to begin buying victory points. In the end, we were able to come to conclusions about particular cards, come up with a hypothesis for how an accurate predictor might be made, and tackle a subset of this problem successfully.

1 Introduction

Dominion is a card game created by Donald X. Vaccarino. During the course of the game, players work to improve their deck by buying cards that are then added to it. Many cards come with some ability that enables the player to play or buy more or better cards on future turns. At the end of the game, whichever player has the most Victory points in their deck is the winner. An important aspect of the game is simply deciding what to buy each turn because the cards in each deck determine the outcome of the game. We want to be able to input a set of game features and output the probability of winning for buying any particular card.

The following terms are used in the game and will be used in this paper to describe our decision making process.

- **The Supply:** cards that are a part of this game and can be bought by the players.
- **Victory Cards:** cards that have a value of winning points
- **Standard Victory Cards:** victory cards that are included in the supply in every game
- **Attack Cards:** cards that allow you to attack other players and give them disadvantages
- **End Game:** the game ends when either a certain number of piles are gone or when certain high value victory cards are gone. End game is reached when one of these conditions is fulfilled or nearly fulfilled.

1.1 Motivation

There are several attributes of this task that make it a good candidate for machine learning:

- The game has a wealth of winning strategies but the search space for these strategies is much too wide for any one person to exhaust. We think the task is worthwhile because it would be quite hard for humans to explore in its entirety on their own through intuition. Certainly a computer would be able to find more information than any human being and tend to win more often.
- There is a clear definition of good performance for this task. Higher end scores mean better performance for any given card.
- There is a clear definition what experience means for this task. Experience involves detailed descriptions of the current game state, what card was bought, and the outcome of the game.
- It is clear that this task definitely fits into the category of learning. Given more and more data about buying cards and the outcome of a game, accuracy will certainly improve.
- There is a wealth of data we could use to feed as experience to the learner from an online version of the game so that hopefully we will be able to obtain good accuracy.

2 Feature Selection

2.1 About the Data

Our data comes from the game logs of the website dominion.isotropic.org. This site provided an online implementation of Dominion, though it was shut down on March 15, 2013 in order to move players to the official online implementation. Even though this was not the 'official' implementation, it was supported by the creator of Dominion, and had a large fan base who played millions of games through the course of its nearly two years of operation.

A log of each of these games was stored and made available for download. These logs show the final results of each game, as well as the turn by turn log of what each player

did. Out training instances come from these games. Each instance indicates a single card that was bought by a player.

Additionally, if no cards were bought in a turn, an instance is created with the special output "None".

2.2 Collecting the Data

To collect the data for our training, we built a parser that processes each game log and tracks the game state. It automatically extracts instances and all features we defined, including our output features.

We found that while this made collecting large amounts of data much simpler than having a human classify instances, it also made a proper definition of the goodness of an instance incredibly important. In order to automatically label each instance, we had to find a mapping from our knowledge of the current and final state of the game to some measure of goodness. As will be explored later in this report, much of our work revolved around trying to find one such reasonable mapping.

In order to verify that instance features were being computed correctly, the cards believed to be in each player's deck, the supply, and the trash were compared to the expected values provided by the log file. Games where these did not match were ignored.

Additionally, games with only a single player, games that ended in a tie, and games where a player resigned were also ignored.

The actual instances we collected came from the games played between March 1 and March 15, 2013. Excluding games that we ignored, this dataset included 178,741 games. From these games, we extracted 8,416,915 instances. Of these, 4,247,997 came from winning players, and this smaller subset of 'winning' instances was used as our dataset for some of our experiments.

2.3 Pruning the Features

We started with a rather large set of features initially. Most noticeably, it included a feature for each and every card, indicating which cards are actually in the game being played. But as we ran our learner with this full set of features, it soon became apparent that the sheer number of features had overwhelmed the learner. So we chose to compress this to a more concise description of what is in the supply and bucketed the cards into groups with similar attributes.

We also played with our description of the game state itself. The following is a discussion of features we picked initially and why we found them important.

- The number of players was included because some cards (such as those that allow you to steal cards from all players) work better with more players.
- We also include a set of features indicating whether any of a particular group of cards was in the supply. We included these because many cards have similar effects, and the presence of any of them may affect the usefulness of a card. We thought this provided an accurate description of the supply without indicating specifically which cards were there. For example, some cards give you the ability to play multiple cards in a turn, and some cards allow you to buy multiple cards in a turn.
- We also included a number of features that indicated the progression of the game, such as the turn number, the number of empty piles, and the number of various victory cards left. We felt this was important as nearing the end of the game can influence the player to emphasize buying victory cards rather than more useful cards. Cards that are worthwhile at the beginning of the game may not be worthwhile at the end of the game.
- We also included a few attributes to describe the state of the current player, including the number of abilities they had and the contents of their deck. The number of various abilities the current player had is important because it specifies what a player is able to do. For example, a player may have been able to play more cards than they did. In this case, it might be beneficial to buy more action cards that can be played on future turns. We describe the player's deck by its size and ratios of certain types of cards. This gives a fairly accurate view of the composition of the deck without specifying each and every card, much as we described the supply.

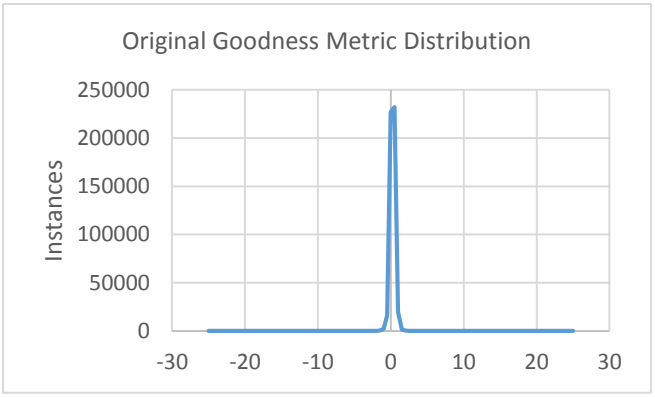
In addition to choosing these features, we also analyzed the data and normalized it to try to exclude outliers. We often lowered the maximum value of a feature from the actual maximum for that feature. For example, in final scores there were a number of outrageous scores that were considerably larger than the majority of the scores. We capped our maximum value at a smaller value that was still larger but closer to the majority. We thought doing this kind of preprocessing we allow us to do better in the general case instead of overtraining on outlying border cases.

2.4 Classifying Instances

As discussed earlier, we spent a lot of time deciding how to convey that an instance was good or bad. We felt that decisions that led to losing scores were bad and decisions that led to winning scores were good. Our first attempt to convey this information used this formula:

$$\frac{(player\_final\_score - average\_final\_score)}{average\_final\_score}$$

Figure 1: Distribution of original goodness metric values in data.



We then modified this formula to be

$$\frac{(player\_final\_score - average\_final\_score)^2}{average\_final\_score}$$

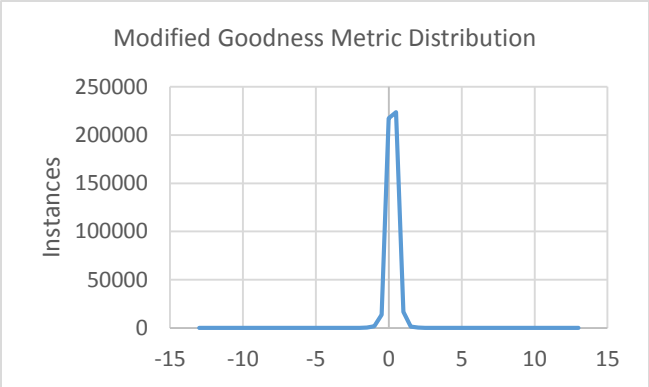


Figure 3: Distribution of modified goodness metric values in data.

Both of these were intended to encode whether or not a player won and by how much. Unfortunately both were so closely clustered around zero that our model could not differentiate between good and bad instances, even when we tried to boost the first piece of the equation to emphasize scores above and below the average. After running through a few practice and partial runs, it was clear that most cards were being predicted with a single weight, no matter what inputs were given.

Thus we decided to try a slightly different approach. We used the raw final score as our indication of goodness. The results using this metric can be seen in our initial results. We felt this might perform better than the others because there was a much wider variation in scores and also because generally high scores are also winning scores.

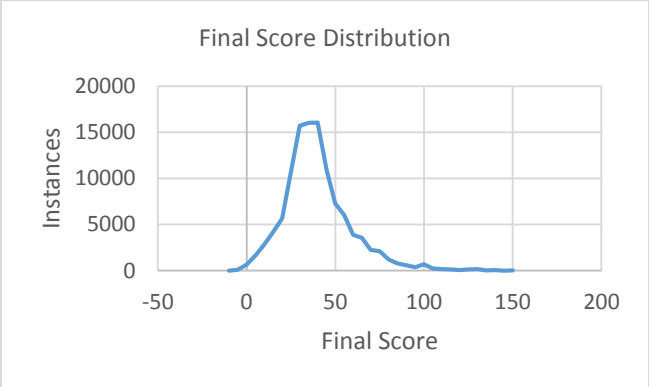


Figure 4: Distribution of final score values in data.

In the end we would have liked to have tried one more variation on the original idea where there is still some encoding of whether someone won or lost by using the following metric, but due to time constraints we were only able to try this metric on a small subset of the data.

Possible future goodness measure distribution:

$$player\_final\_score - average\_final\_score$$

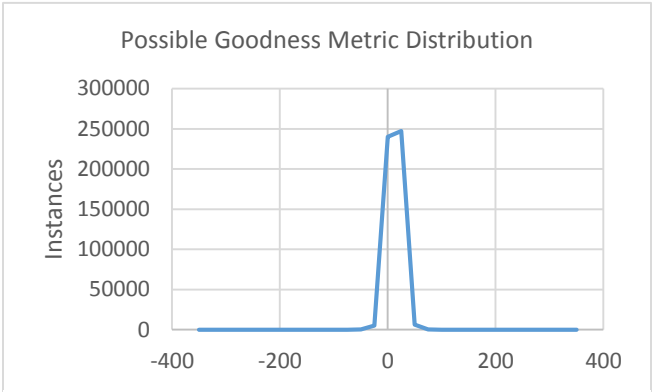


Figure 2: Distribution of possible goodness metric values in data.

3 Experiments and Results

Over the course of this project, we tried learning two different tasks—which card should be bought, and whether it is time to buy victory cards.

3.1 Predicting Which Card to Buy

To predict which card to buy, we used the final score of the game to rank instances. We assumed that over time, it would learn that buying a particular card now would lead it to finish with a certain amount of points. Essentially, we believed that the player's final score, or some variation of it, would directly correlate to how good their actions that game were.

Learning Models

For our initial model, each card is assigned its own learner. Each card was then trained independently on its own set of data, which consisted of all instances where that particular card was bought. The individual learners used backpropagation and perceptron learning models. We chose backpropagation and perceptron because they work well with real input values and give real valued results, not just classifications. A real output value allows us to rank available cards by how good they would be to buy.

As we created the learners, it was clear that training time would be our primary problem. The sheer amount of data slowed training to a crawl and forced us to spend much of our time trying to speed up training. The learners that we used for our experiments draw from a database, which was more efficient than using the text-based ARFF files. The database holds all instances and can more efficiently get the relevant instances during training. It also allows us to more easily pick and choose relevant columns and randomize our data. The model also trains our multiple card learners in parallel in order to speed up time.

Results

After deciding that the models should try to predict the final score, we ran both backpropagation and perceptron on the complete data set. These were the results:

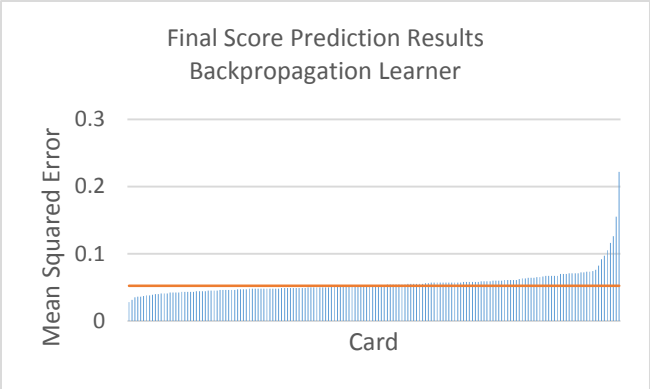


Figure 6: Results of predicting the final score if a specific card is bought on the testing set. The orange line is the total mean squared error over all cards.

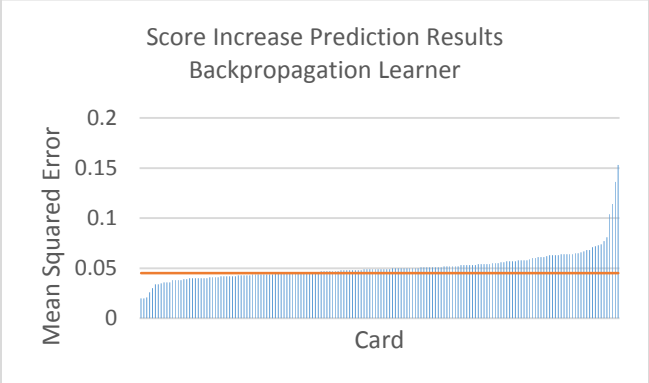
As these results indicate, the backpropagation learner was able to get considerably better results, though they came at the cost of much larger runtimes (16 hours for backprop, 30 minutes for perceptron).

The results from the perceptron do have a slight error. We realized that instead of outputting continuous predictions, it was hard thresholded to either zero or one. It is likely that fixing this would increase the mean squared error of its predictions, but when we realized this, it was too late to rerun our experiments.

In an effort to evaluate just how good these learned models were in a real setting, we attempted to play a game against a basic Dominion bot using our model to choose which card to buy. We did not collect any hard results, but sadly it seemed that most of the recommendations were not very helpful. It consistently suggested one of the cards with the highest MSE. It appeared that this card received very high weights and produced a very high output, even though this card would only be helpful in certain cases. It appeared that our data contained a large number of the border case and took the border case as the regular case. In hindsight, since it is well known that this card is not worthwhile in most cases, there would be few instances to help the learner know that purchasing it would be bad. This is just one example of why our data may not be truly representative of the full range of decisions.

We also were able to try one other metric on a subset of the data. We took the final score minus the average score in

Figure 7: Results of predicting the total change in score over the rest of the game if a specific card is bought.



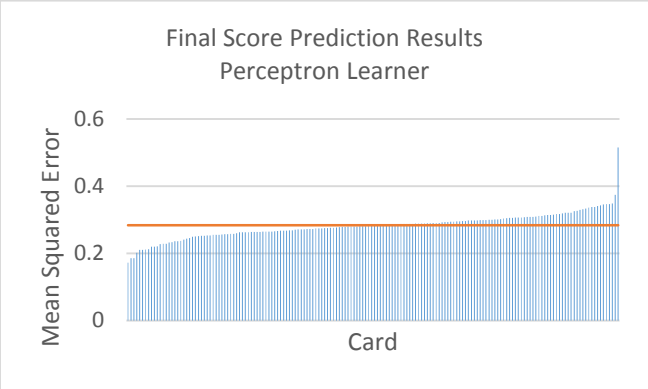


Figure 5: Results of predicting the final score if a specific card is bought on the testing set.

an attempt to encode some sense of who won or lost and also to normalize the data slightly so that games with overall higher scores were not considered better than those with overall lower scores, since the goal is to win in either case. We trained using only the backprop model because we already knew it got much better results.

As can be seen from the graph of the MSE error of each backprop learner, the accuracy was comparable to those with the final score being predicted with no significant differences. In both cases, most learners had a MSE error between .04 and .06.

This can be seen even more clearly in this comparison graph in which we see all three results together. Both backprop results are very similar, while the perceptron is considerably worse.

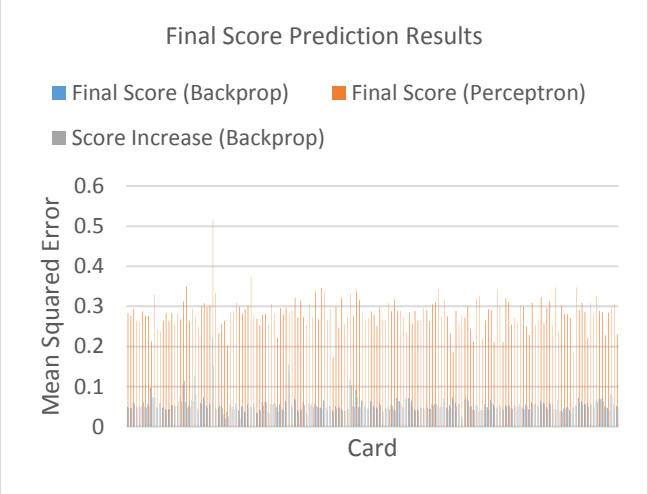


Figure 8: Comparison of all prediction results.

After viewing these initial results, several things became clear about the problem we had chosen. First, it did not perform well when put to use in the intended domain. Reasons for this might include that the data is not representative of a full range of decisions that could be made. Second, it became clear that the majority of cards are not significantly better or worse to buy than others. In our graph, the majority

of cards have a similar MSE and also similar output values which means it was equally approximately equally likely that any one of these cards would be picked by the predictor. This may indicate that most strategies are approximately equal which certainly validates why the game itself is often a close call in the end, which is what makes it so much fun to play to begin with. Third, we could do a good job of predicting certain kinds of cards, even if we did poorly in the general case. The cards with high and lowest MSE tended to be the ones with the highest output values. The models were overconfident on a few card strategies that were almost always used correctly (those with high MSE). But they were also confident on a particular subset of cards: the victory cards. These cards had a low MSE and high output. Our predictor occasionally output these cards at times that made logical sense to us. It became clear that while the overall problem of predicting which specific card to buy might be overwhelmingly complex, predicting when to buy victory cards was a simpler but still interesting subset of the problem that we might be able to obtain reasonable results.

Further Directions

In our model, we built in methods for adjusting the model. Unfortunately actually running experiments, which took up to 18 hours, limited us greatly to the kind of adjustments we could make. We would have liked to have played with the stopping criteria, the number of layers in backprop, and which target feature we are attempting to predict. We also think it would have been useful to do further analysis of our data and pruning of the features, both to decrease the time it takes to reach and to limit features to those that are truly relevant. We also considered creating an ensemble of predictors which we thought might improve our poor accuracies.

We spent a significant amount of time considering how this problem should actually be solved. Though we had a large amount of data, it did not represent the full number of decisions that could be made. Those playing the game are making their own decisions and despite our best attempts, we could only approximate whether or not their decisions were truly good. We figured that an entirely different kind of data would be needed to create an accurate model, and we did not have the ability to produce this type of data. We think an approach where a model actually played games and made its initial decisions at random would be better able to predict which decisions are good and bad. Unfortunately creating such a model at this point was well beyond what we could do with the time allotted.

Beyond just trying to solve this problem, we started to recognize subsets of the buying problem that might be solvable with our data. We tackled one of these problems below and got good results. But we also considered a few others. We thought grouping our predictors to buy certain kinds of abilities and not particular cards might be a simpler and more learnable task. This means that we'd have fewer models and each model would have a larger subset of the data to train on. We also thought we could focus on learning when to buy particular kinds of cards, such as when to buy actions vs money vs victory points as again this would simplify the decision surface and perhaps make it better able to divide the data.

3.2 Predicting When to Buy Victory Cards

The results discussed above let us to tackle a simpler task-deciding when was the time to start buying victory cards.

In most games, there are two general stages of the game. In the first, each player focuses on building up his deck with powerful cards. In the second, each focuses instead on using the deck he's built to acquire as many victory cards as possible before the game ends. Detecting this point in the game is important. If a player starts buying victory cards too early, his deck becomes cluttered by them and he cannot continue to buy them. If he starts buying them too late, the game ends before he has enough points to win.

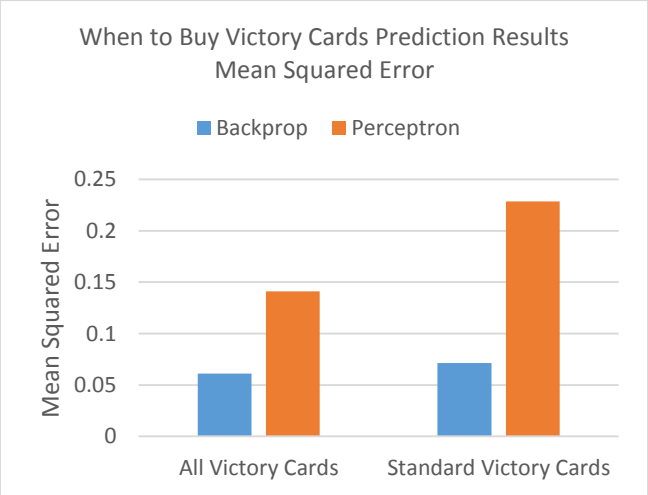
We detected this change-over point in the game by looking only at the instances of cards bought by winning players (which was approximately half of our data). For each of these instances, the target output was set to zero if the player had not yet bought a victory card in the current game. Otherwise, it was set to one.

We ran two different variations on this experiment. In the first, the purchase of any victory card indicated that the game had entered the victory card stage of the game. In the second, only purchasing the standard victory cards (those present in the supply of every game) triggered the change. We thought that there would likely be a difference between these two tasks, as some victory cards also have further abilities that might make them somewhat valuable earlier in the game as opposed to the standard victory cards which simply have a point value and offer no further advantage.

The following graphs show our accuracy on the test set with both MSE and classification accuracy. Once again backprop performed much better than perceptron. Notice that for classification accuracy, accuracies were 75% or above.

These results appear to show that we were right in our assumption that this was a more solvable task. Furthermore, looking at the weights learned by the model showed trends

Figure 9: Results of when to buy predictions on testing set.



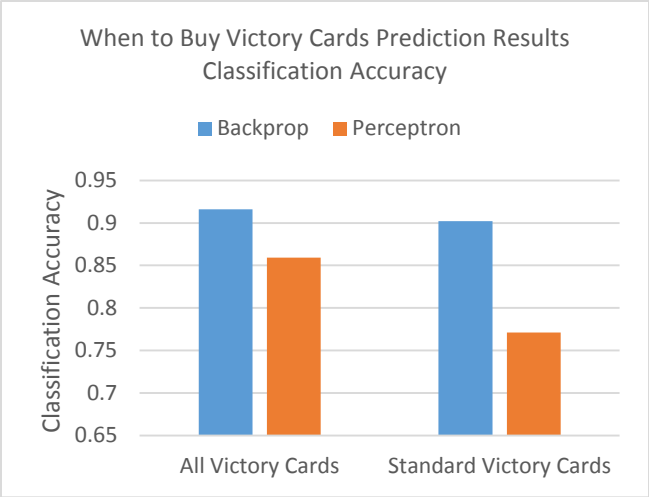


Figure 10: Classification results of predicting when to buy victory cards on the testing set.

similar to what intuition would lead to. Additionally, it did not appear to be overfitting the data, as accuracies on the training, validation, and testing sets were all within a fraction of a percent.

Further Directions

If we had more time, we would have liked to explore this task further. First, our definition of when the game had switched to the buy victory cards stage was quite basic. Rather than simply toggling a flag the first time a victory card was purchased, we would have liked to try more sensitive measures. For example, detecting this state change when the majority of the most recently bought cards were victory cards or by applying some statistics over the course of the game to detect the point when victory cards started being bought more than action or treasure cards. More advanced measures like this would likely help especially in games where a stray victory card was bought early in the game.