# Financial Magic

Boris A. Perkhounkov, Cooper Gates Frye, Emily Margaret Franklin

## Introduction

We used machine learning techniques to attempt to predict the prices of cards for the Magic: The Gathering trading card game. Each card is fundamentally fairly similar to a stock, since they have a market price which changes on a day to day based on supply, demand, and perception of likely future value. Therefore, our project is an application of machine learning techniques to a finance-like domain which is considerably less explored at present.

There are currently 15440 unique magic cards in existence. Based on our knowledge of the game's microeconomy, and the factors which influence the price of traded commodities more generally, we know the factors which affect card prices are a combination of speculation (people buying for the express purpose of selling later) and and direct demand. The direct demand comes primarily from people wanting access to the card for tournaments, and the degree to which a card is desired for tournament play comes from what it does (hereafter referred to as the card's *attributes*) and whether it has done well in tournaments previously.

Therefore the input to our algorithm is a set of Magic cards, and historical data on the price and tournament use of those cards. We then use a support vector machine to output a prediction of the price of other cards in the future (exactly how far in the future is a parameter).

## Related Work

We apply the methods of Support Vector Regression, which is given a detailed exposition in "A tutorial on support vector regression" (Smola). The Python Scikit learn package's implementation of SVR is based on this paper. More generally, our project is fundamentally a time series prediction task. "Predicting Time Series with Support Vector Machines" (Muller) performs a similar task, but treats it as a more general problem.

## Dataset Acquisition And Feature Choice

Our first data component to collect was historical price data. We scraped historical price data from aggregator site http://www.mtgstocks.com/ and saved data on 28375 cards (many cards have more than one printing, and the printing can have different price patterns) over a period of 1246 days. As expected with real world data there were a variety of minor issues: 116 cards do not have price data, around 1000 of the expected cards do not exist, and many cards are missing price data in the middle of filled-in price data. In Magic there is a notion of cards being "bulk" -- enough supply relative to demand that you can't find anyone who will buy them, and the "official" prices are essentially useless. We therefore examine only cards that have a current price of over 1 dollar, which was 5049 of them.

Next, we downloaded JSON files containing card data describing gameplay qualities (name of card, usage, description, etc.) from http://mtgjson.com/. To create vectors with which we could train a

machine, we mapped from the price data to card name. We had a list of keywords that indicate card abilities, and we checked the description to see if the card had such an ability.

Unfortunately, we were unable to map the name of the card from pricing data to the JSON data in the following situations: a card had more than one card printed on it, as we treated price data entities as singular cards; the character Æ appeared in a card name (Æther), as our JSON parsing software did not allow us to look up cards with UTF-8 characters. Thankfully, all cards that fall in these categories are under $1 at present, so we can ignore them. Otherwise, mapping the data went well.

Additionally, we downloaded tournament winners' decklists from tournament websites listed at http://magic.wizards.com/en/events/coverage. There were 12 Pro Tour tournaments hosted by Wizards of the Coast over the time period from which we have price data, and so we collected the lists of cards of the decks of each of the top 8 players and saved them locally. Mapping this data to the JSON files revealed 3 typos, which we corrected, and a handful of Æ-containing cards.

Summarily, our raw feature vectors looked roughly like this:

```
================PRICE DATA==================
Price on 2012-06-09 17:00:00            2133
Price on 2012-06-10 17:00:00            2146
...
Price on 2015-11-05 16:00:00            6172
Price on 2015-11-06 16:00:00            6068

=========== CARD ATTRIBUTE DATA ============
Resource type 1: amount needed to play  1
...
Total resource cost                     2
...
Is card type: creature                  1
Is card type: land                      0
...
10th Edition                            0
...
Ice Age                                 0
Innistrad                               1
Invasion                                0
...
Zendikar                                0

============TOURNAMENT DATA================
Tournament 1, Winner's main deck        0
...
Tournament 4, Winner's main deck        3
Tournament 4, Winner's side deck        0
Tournament 4, other main decks          4
Tournament 4, other side decks          0
Tournament 4, all decks                 7
```

```
Tournament 5, Winner's main deck        0
...
```

We then standardized our features to have mean 0 and variance 1, so they would work with our algorithm.

# Methodology

We are interested in predicting the value of a continuous variable -- a classic regression problem. We wanted the freedom to experiment with kernels, so we decided to use ε-insensitive Support Vector Regression. Specifically, the algorithm attempts to solve the following optimization problem:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*)$$

$$\text{subject to} \quad \begin{cases} y_i - \langle w, x_i\rangle - b \le \varepsilon + \xi_i \\ \langle w, x_i\rangle + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

where (basically) each $x_i$, $y_i$ pair is one data point, ε gives how far away from the true value our prediction is allowed to be, and the ξ's are "slack variable" which allow for misses by more than ε, but incur a cost (these are important to have because of outliers and messy data).

However, one of the most important parts of asking "what will this cost in 'the future,'" is having a clear idea of exactly *when* in the future you mean. Predicting the price for tomorrow is too easy -- by just predicting today's price you do fairly well already. Attempting to predict the price three years into the future is also of little use -- the price is influenced by many chaotic factors, and even if you could do it perfectly it is slower to leverage into profit. We considered fixing the amount ahead we were predicting at some value, but ultimately decided to simply make it a parameter. So we train different SVMs for predicting ten days ahead, thirty, a hundred, three-sixty-five, and so on.

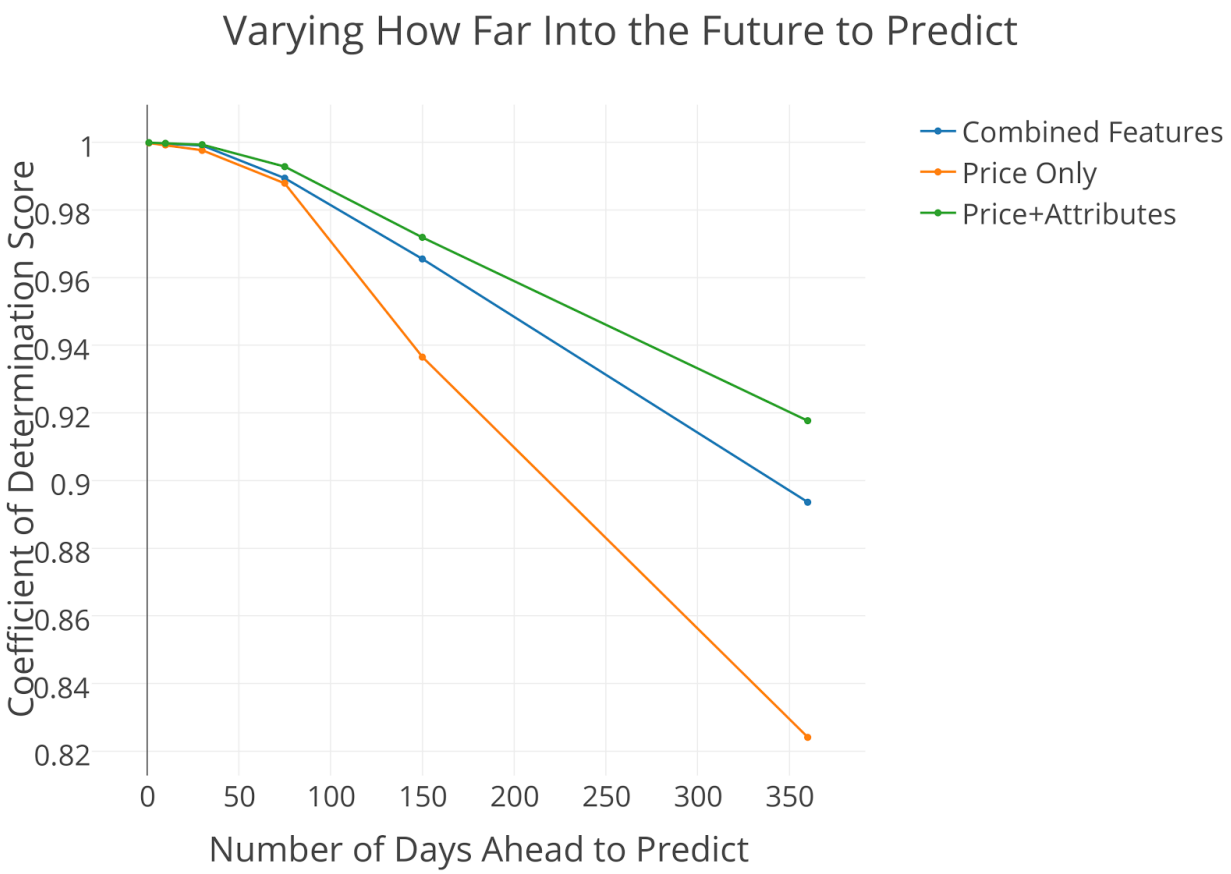
Finally, as important as the choice of learning algorithm itself is the choice of hyperparameters, which we decided on via of grid search (although the type of kernel is not formally a hyperparameter, we decided to treat it as one in that we tried linear, polynomial, and radial basis ones as part of the same model selection process). We decided to use 70% of our data points for training+validation, and evaluated the models by how well they compared on 7-fold cross validation. After the model was chosen this way, we apply it to the remaining 30% of the data to evaluate how well the algorithm performs.

# Results and Analysis

We found that the linear kernel actually ended up giving the best results, with a C of 100. The SVR library we used provides a built in score function, the coefficient of determination ($R^2$). Specifically it calculates (1 - u/v), where u is the sum of $(y - y_{pred})^2$ and v is the sum of $(y - y_{mean})^2$. Although it can be hard to get an intuitive sense of what these values mean, there was no particular metric which we thought would work better, so we focused our time elsewhere. We were curious about the relative importance of the past prices, the card attributes, and the tournament results, so we trained our algorithm on all the combinations of one and two, as well as all three combined. We found that just card attributes and just tournament data alone gave terrible results -- near a score of 0. This was

somewhat unsurprising, since our card attributes features weren't designed to be used independently, and just cover very basic qualities. Additionally, many cards, even those above a dollar, simply see no tournament play. However, both did provide an advantage over only using prices when training at the maximum size.

We the following relationship between how far ahead we predicted and our score:




As expected, accuracy for all methods decreases the greater the prediction time span. Furthermore, the methods which rely more heavily on past prices are more affected. However, we were quite surprised by just how high the accuracy was. Upon further inspection we discovered that this score was being boosted considerably by what were essentially freebies -- cards which cost over $1, but were very stable in price from day to day and month to month. In some contexts this would simply have been a stroke of good luck. However, the ultimate motivation is to make money by taking advantage of these predictions, so getting a bunch of cards which were staying steady correct is not actually helpful.

# Conclusion/Future Work

The most surprising result was the lack of impact from the tournament data. We suspect there are two main effects currently at fault. First, the number of cards (esp. expensive cards) that see tournament play is very small. There are many smaller tournaments beyond the ones we analyzed that

gradually affect card prices. Data for these tournaments could help, but the data is not readily available: we would have to search for everything by hand.

There are still a few issues with the data. The historical price datas' inconsistencies were mentioned above; to remedy those issues, we plan to smooth the data by replacing the intermediate missing data with an average of surrounding data and by removing bogus data when we notice it. Additionally, we could construct more features to improve performance. For example, day to day prices are highly correlated to one another, so taking averages over certain time periods could be beneficial.

The system as it currently exists predicts accurately in the 20 to 50 day range. We will likely complete the final tweaks described in this section in the near future and see if it improves accuracy, but even as it functions right now, it should be profitable to buy magic cards that were predicted to increase in price. As for whether or not this is more profitable than investing in something else will be the subject of a separate study.

# References

Müller, K-R., et al. "Predicting time series with support vector machines."*Artificial Neural Networks—ICANN'97*. Springer Berlin Heidelberg, 1997. 999-1004.

Smola, Alex J., and Bernhard Schölkopf. "A tutorial on support vector regression." *Statistics and computing* 14.3 (2004): 199-222.

Additionally we cite the Python Scikit Learn libraries (Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.), specifically the libraries sklearn.svm, sklearn.cv, sklearn.preprocessing, sklearn.decomposition, sklearn.pipeline, and sklearn.grid_search.