Bleecker Coyne: AI Baseball Card Appraisal

Introduction:

Since kindergarten, I have been a die-hard baseball fan. In third grade, this fandom turned into an obsession over baseball cards and by high school I ran a 4 figure ebay business. In this project, I envision the baseball card and collectibles industry (I'm convinced this idea is scalable to all collectibles) as a securities market where prices and value are predictable based on present features. I built a dataset for Shohei Ohtani's rookie card sales from September to November– this covers the regular season, the post season with arguably the greatest single game sports performance of all time, and the offset. These mark 3 key fluctuations in a baseball card's market and I believe Ohtani's card serves as a prime example how to predict any collectible's values based on past sales. I then tested it on 3 modified classifiers we build in class, ran predictions, and used it to inform a $380 investment in a Jackie Robinson card (unfortunately not covered in this writeup).

Experimental Setup:

I built the database by hand pulling sales data from https://www.cardladder.com/ for 2018 Topps Chrome Shohei Ohtani Rookie Card. THIS TOOK SEVERAL HOURS. The dataset contains 60 examples.

During the first week, the project was a total nightmare. I couldn't get any of the models from class to work and realized that I had to adjust my database. So, I 1-hot encoded my base, parallel, refractor, and grade features and normalized my sales feature by price from 0-1 and my date feature by the number of days since the sale took place.

This worked wonders. Now that my database was in a good place, I modified the classifiers I'd written throughout the term to make sure they were calculating a regression. Finally, I flushed out my ClassifierComparator.java file to train my classifiers.

Since my sales data was normalized (in the root file), I built a simple de-normalizing function that would retrieve the actual sales price. I ran a 10-fold cross validation, performed an 80/20 training testing split, and my classifiers worked (for now)!!

Now that my classifiers were functional, I began optimizing them. After a discussion with CLAUDE (AI use), I decided to use Mean Absolute Error (MAE) to determine each model's accuracy. I chose this metric since it was very intuitive not only to me as a programmer but to anyone who might use this tool in the future. Each model's MAE represents, on average, the dollar difference between the prediction and actual price; when using an AI price predictor, the average error in dollars is the number one question any collector would want to know. I tested KNN and NN based on the experiments I'd written to optimize the model's parameters earlier in the term:

Neural Net:

Configuration: 10 hidden nodes, learning rate of 0.01, 5000 iterations

KNN:

Configuration: K=1 -> MAE: $23.09, **K=3 -> MAE: $14.83**, K=5 -> MAE: $19.80, K=7 -> MAE: $17.27

Perceptron:

Configuration: 100 iterations (I didn't test this)

Note: during the progress reports, I talked about implementing decision trees as well. I couldn't get my decision tree code to work (and I had too much PTSD from the first assignment) so decided to only use Perceptron, KNN, and NN.

Results:

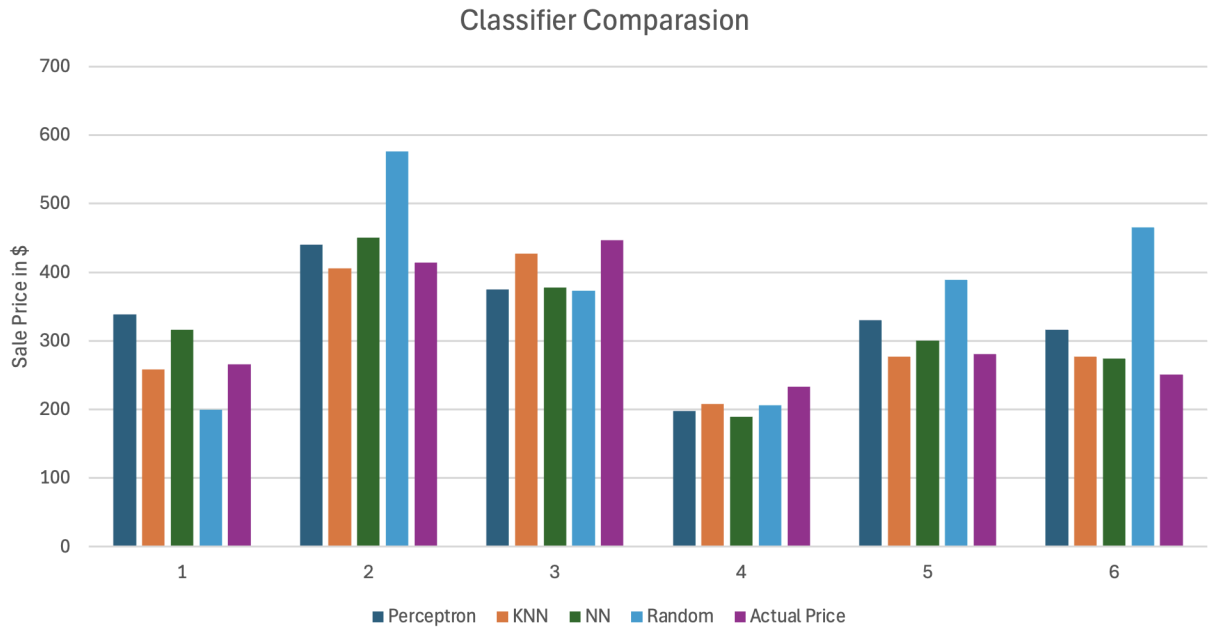
I was very very happy with my outcomes. Building a ML model of baseball card price prediction has always been an idea of mine and now that I'm getting caught up to speed with the industry, I think there's potential to build a real product out of this project.

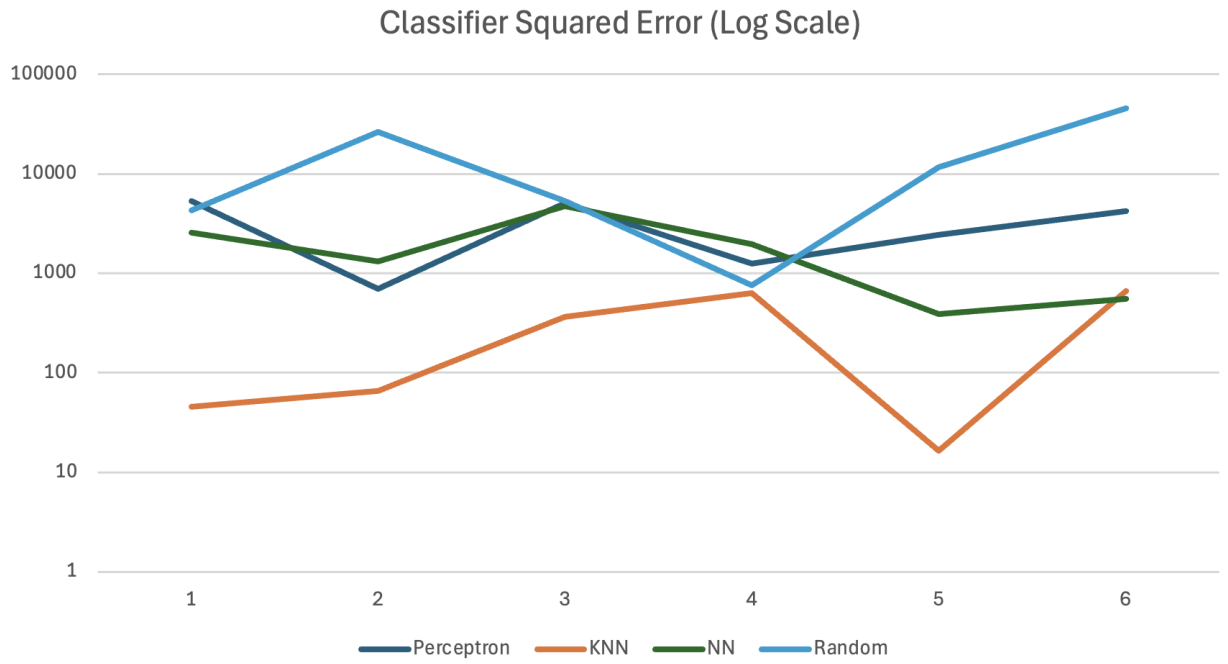
When I started analyzing my results, I decided to move to a 90/10 training testing split. Given the scarcity of data, I want to be sure that my model is training on as much information as possible. Additionally, buying decisions are almost always made with a small number of card sales– the way a collector might use this product would be to input the date of a recent sale and compare the model's prediction to the sale value itself. My 90/10 split left me with 6 test cases (60 examples) but this is how I'd envision a user determining their trust in the mode (based on few or a single test case).

Additionally, I added more metrics at this stage. Prof Kauchak recommended I consider Squared Error (as it is sensitive to outliers) so calculated the following stats for each model.

Metric	Perceptron	KNN	NN	Random
Mean Squared Error (MSE):	$ 3,156.07	$ 298.43	$ 1,915.08	$ 11,177.17
Root Mean Squared Error (RMSE):	$ 56.18	$ 17.28	$ 43.76	$ 105.72
Mean Absolute Error (MAE):	$ 53.29	$ 14.83	$ 40.50	$ 87.66
Mean Absolute Percent Error (MAPE):	18.04%	5.21%	13.10%	33.49%
Max Absolute Error:	$ 73.08	$ 25.72	$ 68.62	$ 179.09

I also added a Random classifier at this stage to serve as a control. I can get all excited by my results, but if the sales prices are too similar (i.e. the market value hasn't changed over the course of the dataset) the low Mean Absolute Percent Errors could be misleading. So, like a well trained CS 158 students, I hoped into excel to graph my results:





The X axis represents each example and the Y axis represents the value in dollars $. The first chart shows each classifier's prediction alongside the actual sales price. Though we can see that random is not doing very well, it's hard to discern which model is most accurate. The second chart demonstrates their relative accuracy by plotting the square error from each of the predictions. Random was performing so poorly compared to the rest, I had to put the graph in log scale just to visualize their performance. Here, we can see that KNN is outperforming the other classifiers by a very hearty margin.

Across all metrics, KNN outperformed the neural network, perceptron, and random

baseline. With K=3, KNN achieved an MAE of **$14.83**, far lower than the neural network (**$40.50**) and perceptron (**$53.29**). MSE and RMSE followed the same pattern: KNN had an RMSE of **$17.28**, compared to **$43.76** (NN) and **$56.18** (Perceptron). The random model performed dramatically worse across all metrics (RMSE **$105.72**).

Conclusion:

Initially, I concluded that my KNN algorithm works best for baseball card sales. All my metrics clearly point to its relative success compared to other metrics. However, I think there's more nuance and that overfitting is playing a role. Looking at the data more closely, KNN consistently underpredicted the sale price. While NN and Perceptron both have predictions below the actual value, they tend to make overpredictions (see first chart). This makes theoretical sense:

- KNN is a local averaging algorithm that predicts around a local cluster
- NN & Perceptron predicts round global trends

This explains why NN and Perceptron are a) less accurate and b) predict higher on average: they are swayed by high sales and outliers as our square error chart demonstrates.

However, this isn't an inherent problem. In fact, it's very useful. Baseball card prices are closely tied to a player's performance and popularity– if they have a big game, their card prices will immediately skyrocket, if a scandal comes out, their card princess will plummet (ask the collector how lost 1.1 million on Wander Franco cards). Perceptron and NN are very useful in this way. If there is a change in the market, they will adjust and predict that value more quickly than KNN. Therefore, I propose this very simple equation incorporating ensemble learning:

If there's no significant change to the players market,

Bleecker_Prediction = 0.6(KNN_Pred)+0.2(NN_Pred)+0.2(Perceptron_Pred)

If there is a significant change to the player's market,

Bleecker_Prediction = 0.33(KNN_Pred)+0.33(NN_Pred)+0.33(Perceptron_Pred)

I started evaluating these equations but I believe they'd only lead to overfitting on the Shohei Ohtani Market. I've evaluated 3 different model's performance on baseball card sales data, and have determined 2 equations I will definitely be using while investing in the future!