Important
For the code to run you need to make a small change to line 64 (data_path=Path to “Card sets”) Make sure this is set correctly otherwise you will get an error.

Intro
I found that there were very limited amounts of Magic the Gathering card readers and that they were often unreliable and oftentimes did not help organize your collection.

With the help of a mechanical engineering friend, we wanted to build a machine that can automate taking pictures of cards and reading the set, name and condition and storing that into a database. You would put a stack of cards in the machine. The machine would read one card at a time and store that information into a CSV file. The machine is currently in development and this is more of a proof of concept.

The first step in reading cards would be set recognition; Magic the Gathering (Mtg) cards are broken down into different sets (there are ~100 sets) and these sets have very specific set symbols that are unique, I wanted to see how accurate I could get with a small sample of selected cards from 6 sets.

This is a theros set symbol (common)

This is a Dominaria set symbol (mythic)

Ideally, I would get 28 pictures for each set because that's all the states an Mtg card could be: This includes 7 colors - White, Blue, Green, Red, Black, Colorless, and Gold and 4 rarities - Common, Uncommon, Rare and Mythic.

Inputs/Outputs

The inputs are the pictures I took of the cards and the output would be the predicted set. Some issues encountered in making my own data set included the lengthy time required to take the pictures and the positioning of the cards (some cards were slightly off-centered so when I cropped the set symbol, it was clipped).

There are currently 112 active pictures in the data set.

Baseline
I split my data into four groups; x_test, x_train, y_test, y_train with 80% being used for train and 20% for testing.

In this experiment, I decided to use different classification methods.

I used:
SGD Classifier

K-nearest neighbor
Logistic Regression

for my different models. (Note these are all from sklearn)

SGD Classifier has many different parameters that you can tune - penalties, alpha, iterations, random seed and many more.
To be consistent with the other models, I went with the default parameters and only changed the iterations to 1000 and gave it a random state of 42 so it can be repeated.

For the K-nearest neighbor, the biggest parameter you can change is the “K-nearest neighbor” itself. Because I had a small data set, I changed the default K from 5 to 3.

For Logistic Regression I chose to have the same parameters as the SGD Classifier

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

These are the three links to the webpage if you want to see the full scope of parameters.

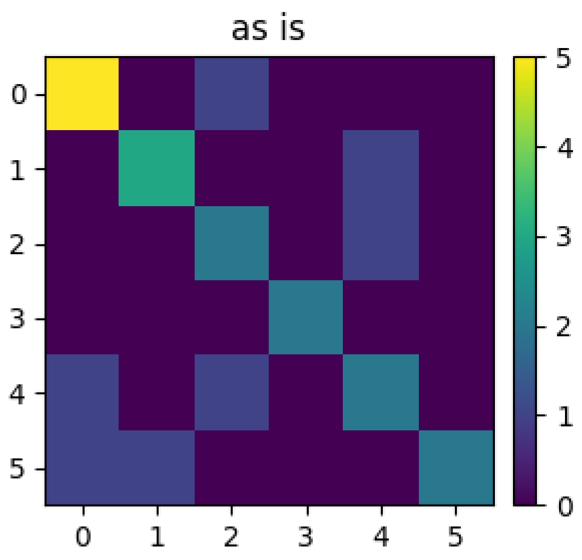
Evaluation
The evaluation is simple. Did we correctly predict the right set and did we do it in a reasonable time (ie., for 6 sets, it should run relatively fast because I want to expand this to contain every set.)

The goal would be to achieve 100% accuracy; because reading a set symbol should not be stochastic. A set symbol is unique and does not change other than the variations talked about above.

Note: If we tried to randomly guess what set a card belongs to, we would have an accuracy of ~17%. (⅙).

Results
Time taken for Get_croped_img_data: 8.912018775939941 ~1.5 secs per set therefore for 100 sets around 2 mins 30 seconds to get all the data which is reasonable.

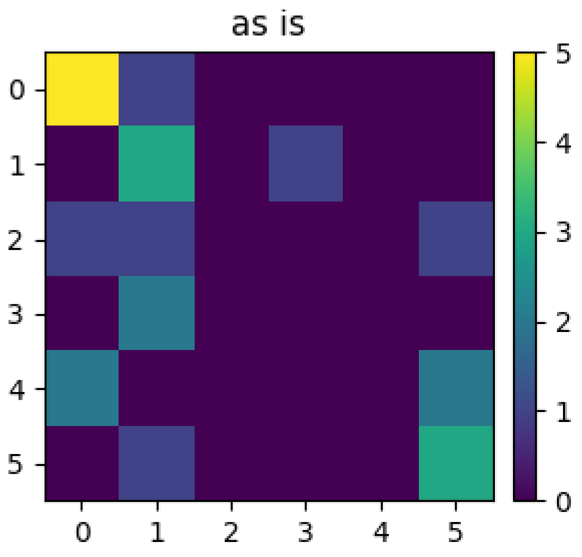
SGD



```
This is this data in a nice picture
[[5 0 1 0 0 0]
 [0 3 0 0 1 0]
 [0 0 2 0 1 0]
 [0 0 0 2 0 0]
 [1 0 1 0 2 0]
 [1 1 0 0 0 2]]
Percentage correct:  69.56%


Time taken for SGD Classifier
0.461600 Seconds
```

K-near



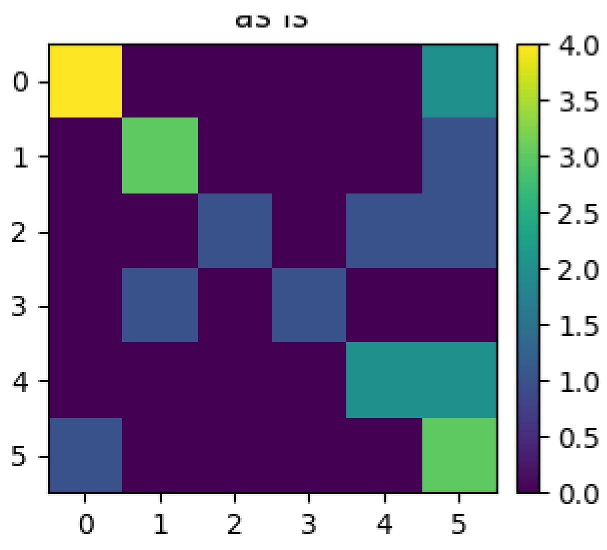
```
This is this data in a nice picture

[[5 1 0 0 0 0]
 [0 3 0 1 0 0]
 [1 1 0 0 0 1]
 [0 2 0 0 0 0]
 [2 0 0 0 0 2]
 [0 1 0 0 0 3]]


Percentage correct:  47.83

Time taken for K-Neighbors Classifier:
0.004986 Seconds
```

Lin



This is this data in a nice picture

```
[[4 0 0 0 0 2]
 [0 3 0 0 0 1]
 [0 0 1 0 1 1]
 [0 1 0 1 0 0]
 [0 0 0 0 2 2]
 [1 0 0 0 0 3]]
```

Percentage correct: 60.87

Time taken for LogisticRegression:
1.6041958 Seconds

```
['Dominaria', 'Ice Age', 'Innistrad', 'Mirage', 'Theros', 'Unstable']
0                  1          2           3          4             5
```

These pictures are the confusion matrix of my results. Ideally, we would only have different colors on the diagonal line. The different confusion matrix shows me that the SGD Classifier method works the best but still has a large error of 30%.

Also, we can see that logistic regression took the longest time to run while k-nearest neighbors was exceptionally fast to compute. Though it is my opinion that all these methods had acceptable run times but the accuracy difference that SGD achieved would have me believe it is strongest model for this data.

Some of the issues that would decrease accuracy are low-quality input pictures and a general lack of data. In the future, I will try and standardize my picture taking first so it's more consistent. Also adding more data to train on. I could add more data by doing some bootstrapping on the pictures. When making my data, for every image, I crop different areas and add them to the model therefore allowing me to double or triple the input.

References

I used Sklern for the three models links are above.
I also used this tutorial for image classification
https://kapernikov.com/tutorial-image-classification-with-scikit-learn/

I was heavily influenced by this code to help me figure out how to read my data and its suggestion to use greyscale and GSD classification method.