Magic: the Gathering & NLP

Audrey Gilbreath

Magic: the Gathering (MTG) was first released in 1993 and is right now more popular than ever among all demographics and continuing to grow after thirty years. Game design and development is centered around Magic's "color pie" and the color's gameplay themes and archetypes. A balanced color pie creates an environment for healthy formats[1] and diverse metagame[2]. Both of which are critical factors in player engagement, satisfaction, and growth. With more new cards being released on a regular basis than ever before, there has been an increase colors bleeding, and has become a focal point in the Magic community and is creating contention and concern among players and content creators. As a means to maintain and improve player satisfaction, can a card's color be predicted based on its text using machine learning?

[1] Different variations or modes to play MTG.
[2] The themes/archetypes/cards/decks players *choose* to play

The Dataset & EDA

MTGJSON.com have collected all card data and related data from the major MTG websites and put it into a json and csv open to the public. The dataframe had data on every card ever printed, with each row representing one card and containing its data. The dataframe had 89 columns and I performed an extensive EDA to determine which columns are necessary. I kept the `colors` and `text` for target and feature variables, and the `keywords`, and `name` columns for data preprocessing.

Figure 1

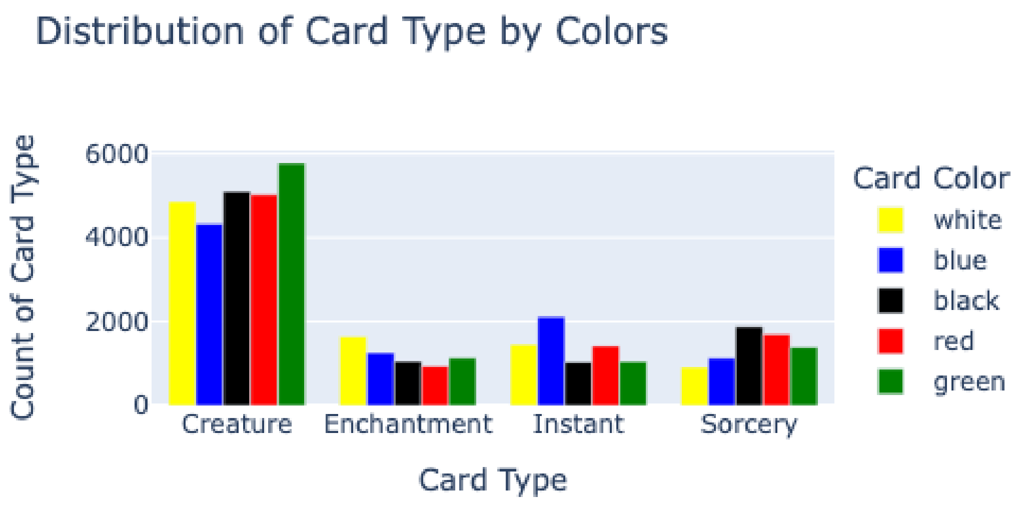


figure 1 shows the card type distribution by color. Green has the most creature cards, white has the most enchantments, blue has the most instants, and black has the most sorceries. So far, in terms of card type, the colors are aligning with their theme.

Of the four columns kept, only one column had zero missing values, the name column. The other columns' missing values are due to game design. The text column had close to one thousand nans, but that's because those cards didn't have text, like the card in Figure 2.

Figure 2

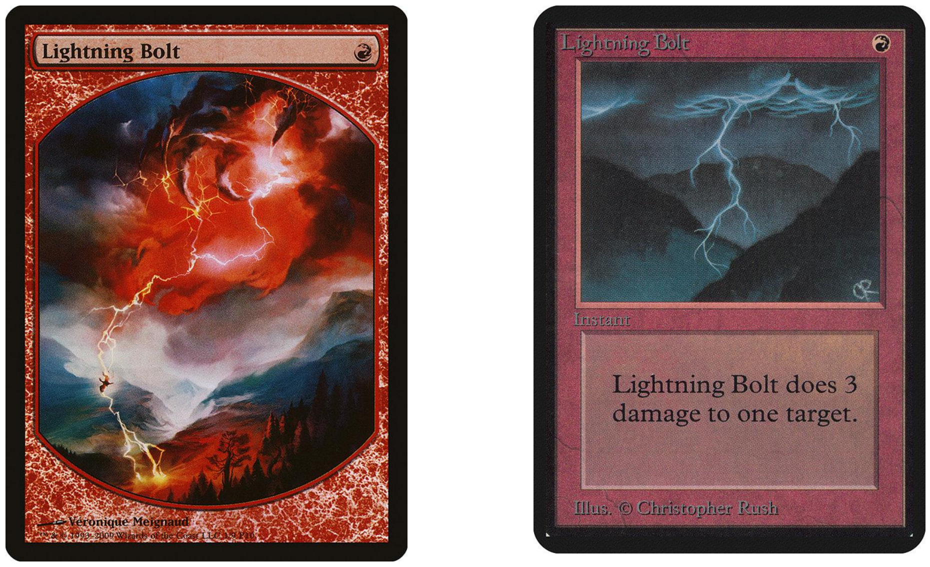

Lightning Bolt
Véronique Meignaud
Lightning Bolt
Instant
Lightning Bolt does 3 damage to one target.
Illus. © Christopher Rush

Figure 2 shows two different printings of the same card. One with text and one without. Of the duplicate name cards, the ones that were printed first had text and were kept. Next, all columns were dropped except for name, text, keywords, and colors. The colors column was encoded using sklearn's LabelEncoder, the name column was used to find any card names in the card's text and remove the name from the card text, same with colors.

Data Preprocessing

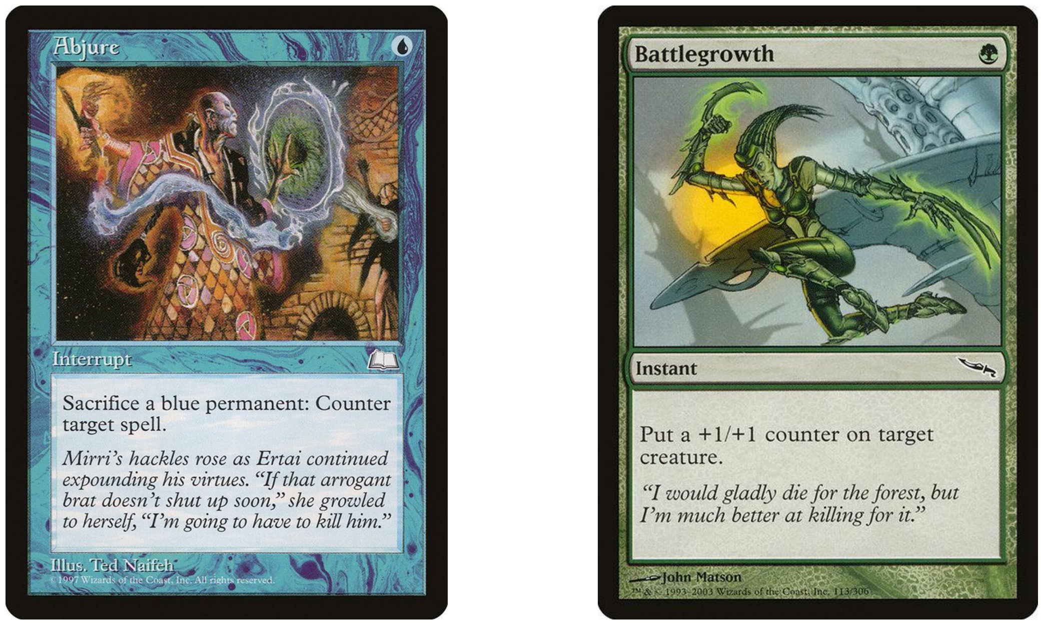
I dropped the rows with card type ' land' as well as any other rows with colorless card type. From there, the colors column' classes needed to be remapped so that the values are a permutation of the mono-colors, instead of a combination. Some of the options available to me for this kind of problem was to treat it as a multi-class problem or a multi-label one. For now, this will be a multi-class problem, and so the multi-color cards were dropped. Card names that appeared in the text were removed, as well, to preemptively decrease the number of tokens in the final document-term matrix. The colors were also removed from the card names so that the machine wouldn't learn to make predictions on that (which I think it would since the machine is only seeing mono-color cards and if any color is mentioned on the card it would be the card's color).

I only wanted mono-color cards, including colorless cards that had a mana value, such as artifacts, but not lands. Unsets were dropped since they are outside of usual play; duplicate cards were dropped–the first print was kept since the first print always has text, but a later print might be full art. The MTGJSON file only had oracle text (text with updated rules language) so there was no extra noise with old rules language, such as "interrupt" being updated to "instant". Cards with no text were dropped, as well.

Text Preprocessing

After splitting the data into train, test ,validation sets, I used TF-IDF vectorizer along with a custom tokenizer to finish cleaning the text and to convert the text into numerical data. The custom tokenizer removed punctuation and splitlines before tokenizing the text with word_tokenize, and then filtered out noise such as stop words and root words using WordNetLemmatizer. For the TF-IDF vectorizer, I utilized its built in n-grams_range parameter to make uni- and bi-grams, and as a simple way to begin dimension reduction I set min_df to 30 so that only words that appear thirty or more times in the corpus are vectorized. I included bi-grams into the corpus because bi grams show up in the card text as a part of game play, as seen in figure 3.

Figure 3


Abjure
Interrupt
Sacrifice a blue permanent: Counter target spell.
Mirri's hackles rose as Ertai continued expounding his virtues. "If that arrogant brat doesn't shut up soon," she growled to herself, "I'm going to have to kill him."
Illus. Ted Naifeh
Battlegrowth
Instant
Put a +1/+1 counter on target creature.
"I would gladly die for the forest, but I'm much better at killing for it."
John Matson

Both cards in Figure 3 have the word "counter". However the blue card's "counter target spell" prevents another player from taking a turn, and the green card's "+1/+1 counter" increases creature power and toughness.

I chose TF-IDF over CountVectorizer because some keywords are only used in a couple of sets (like *Evolve*) and some are constant in every set (like *flying*).

Modeling

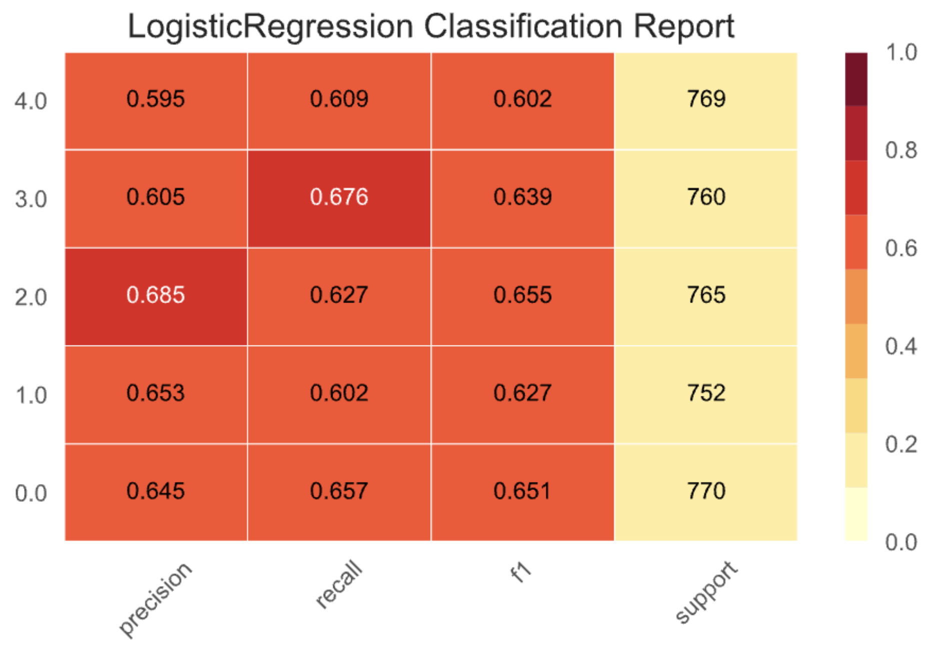
I used Logistic Regression and Random Forest to model, and precision to measure model success. I chose precision since this project topic does not have business sensitivity or repercussions to be concerned about with false positives.

Both baseline models had a ~63% validation accuracy, but the Logistic Regression was overfitting on the train dataset by ten percentage points, and the Random Forest was overfitting by more than thirty percentage points.

I used GridSearchCV to optimize, however, parameter and hyperparameter tuning did not do much to improve model performance. The best RandomForest model was still overfitting by thirty percentage points. The Logistic Regression accuracy did not improve much but it was only overfitting by four percentage points.

The best model was the Logistic Regression, C=0.1, max_iter=500, multi_class='ovr', solver='lbfgs'.

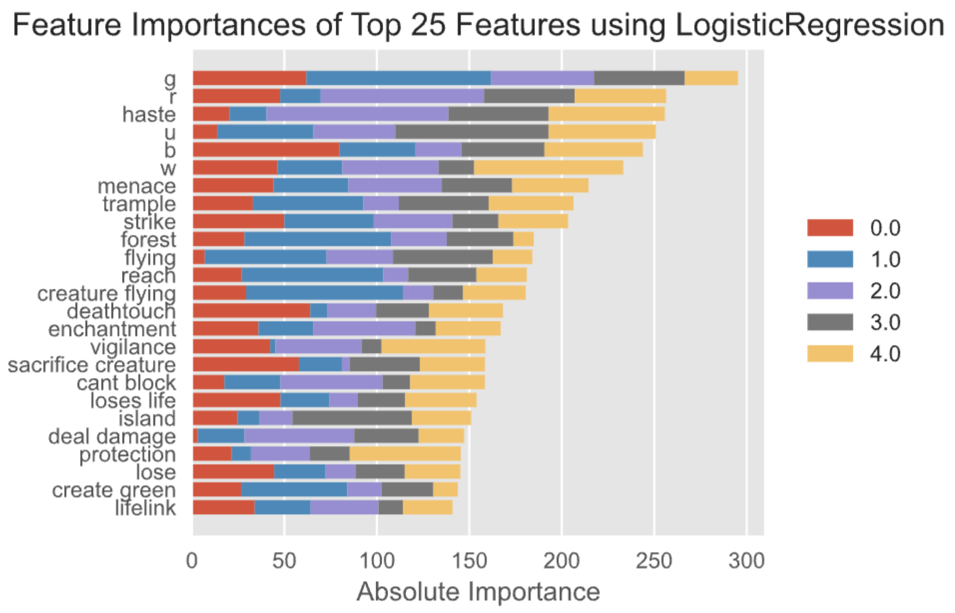
Figure 4



Classification report for best Logistic Regression model

In figure 4, the classification report for the Logistic Regression the precision is higher than the recall, but not by much. The confusion matrix showed that the model was particularly having trouble classifying white and blue.

Figure 5



Shows the top 25 import features by class.

Looking at the top features by class, I found a major issue. The color abbreviations did not get removed when I removed the colors from the text during preprocessing (I explain more on this in the Final Thoughts section). Sometimes there's a mana symbol in the card's text. Mono-color cards can only have their color mana symbol in the text. So the models were fitting to whether or not the cards told them their color or not. And it seems about 63% of cards have a mana symbol on them which is why the baseline and optimized models performed similarly.

Final Thoughts

After further investigation into why the color abbreviations weren’t removed, I learned it was because the abbreviations were surrounded by curly brackets {}, so they didn’t get picked up from my line of code during preprocessing. I need to add their removal to the custom tokenizer after punctuation and tokenizing have been complete, but before stemming. This should make major improvements and I expect the Random Forest model would then perform quite well considering it was able to classify the other 30% of cards precisely.