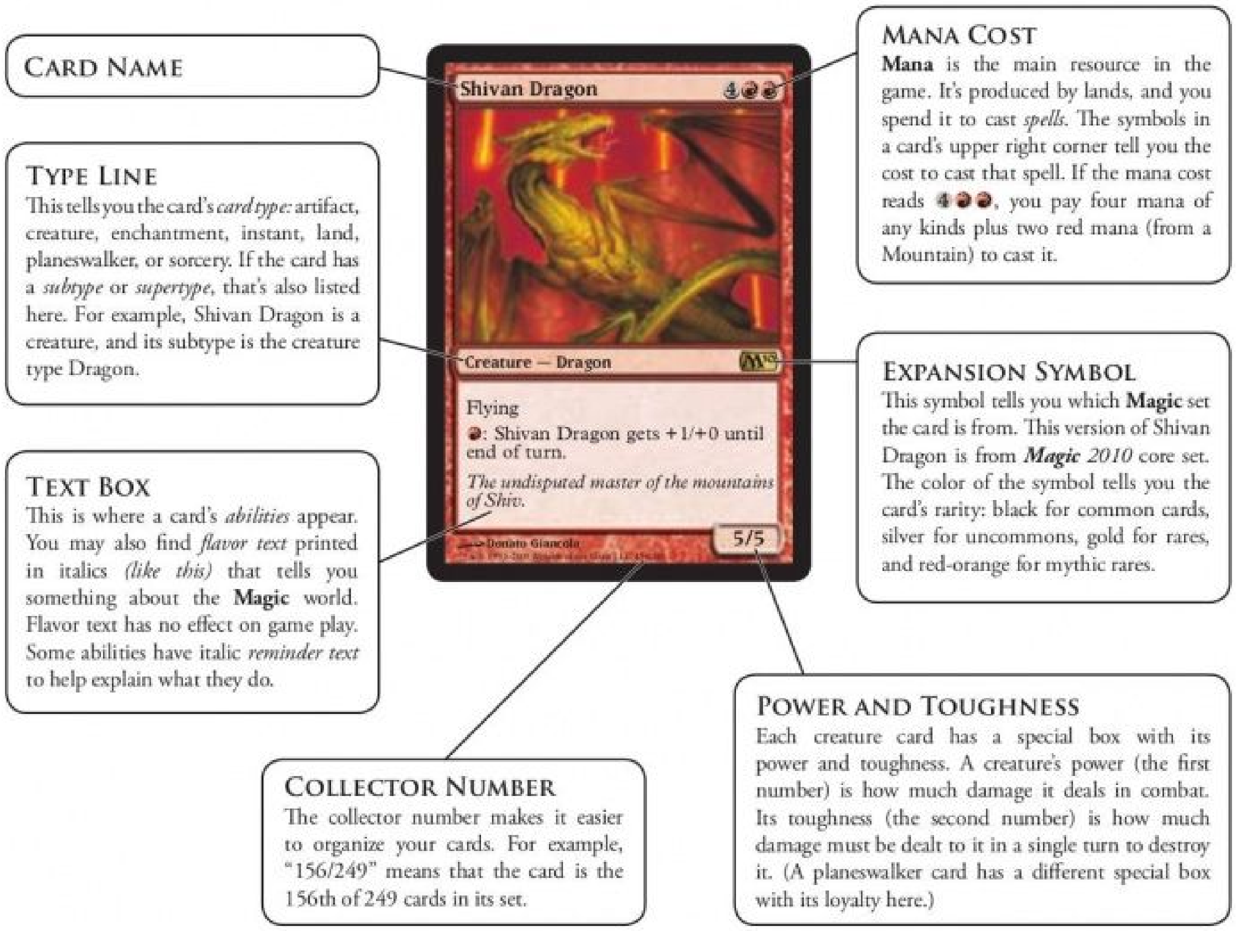
Matan Diamond
mdiamond8

Magic: The Gathering Creature Color Classification

Magic: The Gathering is a collectible trading card game created in 1993 by Wizards of the Coast. The game involves assembling a deck of cards (which Wizards of the Coast releases in booster packs) and then playing out a game with the goal of eliminating your opponent using the cards in your deck. The game is deep, with many mechanics spread across 20,000 cards released over the past decades. The dataset I will be using is a detailed list of every card ever printed, which is public information that has been fortunately compiled into a verbose dataset on Kaggle here.


Shivan Dragon
4●●
Creature — Dragon
Flying
●: Shivan Dragon gets +1/+0 until end of turn.
The undisputed master of the mountains of Shiv.
Donato Giancola
5/5

CARD NAME

TYPE LINE
This tells you the card's *card type:* artifact, creature, enchantment, instant, land, planeswalker, or sorcery. If the card has a *subtype* or *supertype*, that's also listed here. For example, Shivan Dragon is a creature, and its subtype is the creature type Dragon.

TEXT BOX
This is where a card's *abilities* appear. You may also find *flavor text* printed in italics *(like this)* that tells you something about the **Magic** world. Flavor text has no effect on game play. Some abilities have italic *reminder text* to help explain what they do.

MANA COST
Mana is the main resource in the game. It's produced by lands, and you spend it to cast *spells*. The symbols in a card's upper right corner tell you the cost to cast that spell. If the mana cost reads 4●●, you pay four mana of any kinds plus two red mana (from a Mountain) to cast it.

EXPANSION SYMBOL
This symbol tells you which **Magic** set the card is from. This version of Shivan Dragon is from ***Magic*** *2010* core set. The color of the symbol tells you the card's rarity: black for common cards, silver for uncommons, gold for rares, and red-orange for mythic rares.

COLLECTOR NUMBER
The collector number makes it easier to organize your cards. For example, "156/249" means that the card is the 156th of 249 cards in its set.

POWER AND TOUGHNESS
Each creature card has a special box with its power and toughness. A creature's power (the first number) is how much damage it deals in combat. Its toughness (the second number) is how much damage must be dealt to it in a single turn to destroy it. (A planeswalker card has a different special box with its loyalty here.)

A *Magic: The Gathering* card contains a great deal of information. I will be predicting the *color* of *creature* cards based on:

- The *types* the creature has (e.g. an elf, goblin, warrior, etc.)
- The *power* and *toughness* of the creature
- The *converted mana cost* which is the total numeric cost to play the card
- The *text box* which holds a great deal of information hidden in natural language, including the creature's abilities and other keywords.

Matan Diamond
mdiamond8

Data: the Gathering

Lava Axe
Sorcery

Lava Axe deals 5 damage to target player or planeswalker.

"Catch!"

Shoe Tree
Creature — Treefolk
3/3

Shoe Tree comes into play with up to two shoe counters on it. Use your shoes as counters.

Shoe Tree gets +1/+1 for each shoe counter on it.

It grows several feet a year.

Daybreak Ranger
Creature — Human Archer Werewolf
2/2

: Daybreak Ranger deals 2 damage to target creature with flying.

At the beginning of each upkeep, if no spells were cast last turn, transform Daybreak Ranger.

The full dataset is large, which is good for our data-hungry classification algorithms. However, much of the data has to be thrown away for a variety of reasons:

- Non-creature cards (the goal is to classify creatures specifically)
- Cards from tournament-prohibited sets (generally joke cards with grey borders)
- Double-faced cards (unique cards with another card on their backside)
- Cards with a printing before July 21, 2003 (the color identities changed)
- Cards from sets that focus on breaking the color identities

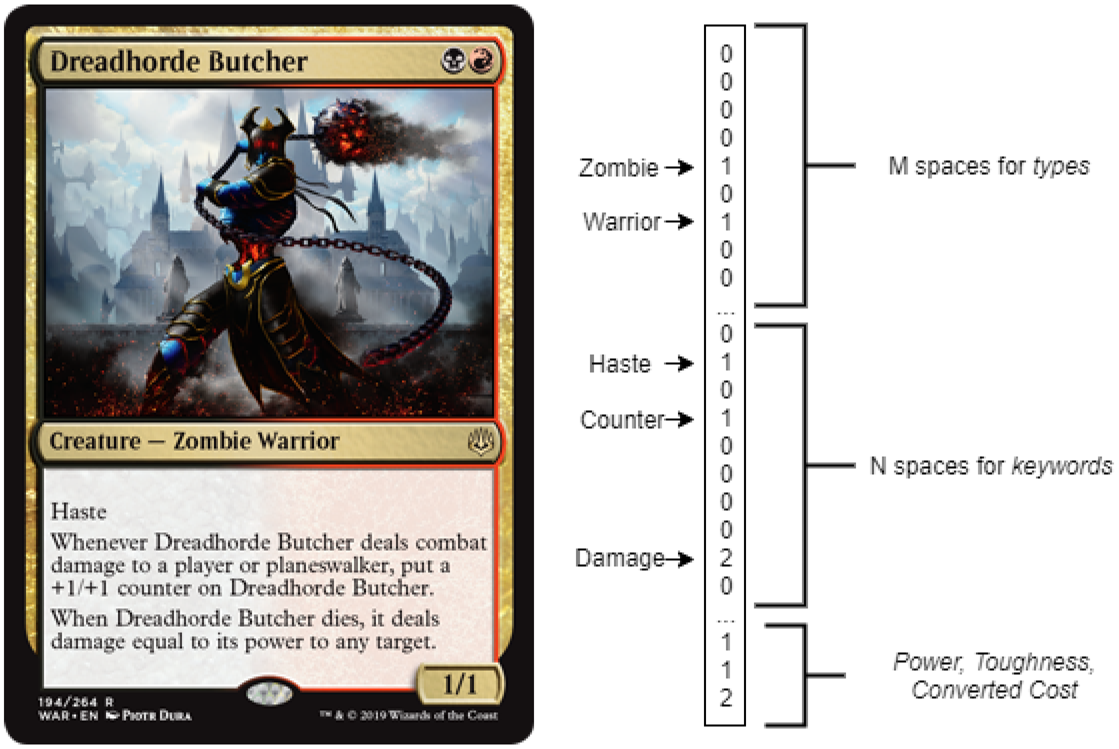
In addition, we have to do some significant preprocessing to turn our dataset into something we can use. Specifically, our categorical data (*types*) and the natural language data (*text box*) both must be converted into numeric features in a clever way so that the scikit learn algorithms can understand them.

I spent a good deal of time looking into algorithms to vectorize the *text box*. I ended up using Doc2vec (similar to Word2Vec but for vectorizing paragraphs or other larger groupings of text) to vectorize the text bodies but I found after experimentation that my classifiers actually performed worse with the addition of the Doc2vec vectors, so I opted for a more traditional approach for parsing the *text box* that I believe better fits the domain of *Magic: The Gathering* Cards. The dataset includes a list of keywords that can be found on the cards, so I opted to do a modified bag-of-words approach where I selected the N most common keywords and devoted one slot in the full feature vector of each creature for each common keyword.

This one-hot encoding is effective because it allows us to keep track of how many instances of each keyword appear on each creature and because it keeps the algorithm from learning colors based on the numeric distance between IDs in a single feature approach.

Matan Diamond
mdiamond8

The *types* also had to be processed because scikit learn cannot handle categorical data. I opted to use a one-hot approach again for *types*, this time taking all *types* that appeared above a predetermined threshold number of times in the training data. I devoted a slot in the feature vector for each of these M *types*[1]. This is important again because creatures can have multiple *types* and we will misrepresent the data with a scalar representation.


Dreadhorde Butcher
Creature — Zombie Warrior
Haste
Whenever Dreadhorde Butcher deals combat damage to a player or planeswalker, put a +1/+1 counter on Dreadhorde Butcher.
When Dreadhorde Butcher dies, it deals damage equal to its power to any target.
1/1
Zombie
Warrior
Haste
Counter
Damage
0 0 0 0 1 0 1 0 0 … 0 1 0 1 0 0 0 0 2 0 … 1 1 2
M spaces for *types*
N spaces for *keywords*
Power, Toughness, Converted Cost

The output also has to be vectorized. Because there can be overlap in colors, I opted for a one-hot approach again for the colors of each creature. The predictions of the model are vectors of length five. For *Dreadhorde Butcher* this would be [0, 0, 1, 1, 0] indicating “red” and “black” but not “white,” “blue” or “green”.

To evaluate the performance of a multi-label classification algorithm, we need to adjust our metrics slightly because of the notion of “partial correctness.” We will seek to maximize the “exact match ratio” which is the proportion of samples that are fully correct, alongside slightly differing versions of accuracy, precision, recall and F1 measure to better fit the muli-label problem. “Hamming Loss” is a concept that reports how many times on average, the relevance of an example to a class label is incorrectly predicted.

[1] I also added two more spaces for *artifact* and *enchantment*, which are *types* but not *subtypes*.

Matan Diamond
mdiamond8

Algorithms

Random Forests

A random forest is an ensemble learning method in which a multitude of decision trees are constructed with random splits on random subsets of the dataset. Bootstrap aggregation is the process of training this forest of random trees, and subsequently querying them all, averaging their predictions to get each desired prediction.

The hyperparameters for our random forests are:

- *n_estimators*: the number of trees in the forest
- *max_depth*: the maximum number of levels in each tree

Support Vector Machines

A support vector machine is a supervised learning method where a decision boundary is calculated that maximizes the space between classes of differing labels. To use support vector machines for multiclass output data, one has to fit a support vector machine to each combination of classes. Querying involves querying each decision boundary and then combining the outputs into the prediction vector. This is called the 'one vs one' classification technique, which is necessary for multiclass non-linear SVMs.

The hyperparameter for our support vector machines are:

- *kernel*: specifically 'poly', 'rbf' or 'sigmoid'.
- *C*: penalty parameter of the error term.

Deep Neural Networks

Deep learning involves using a multi-layer neural network to extract higher level features from raw data. By using multiple layers, we can model complex concepts based on the combinations of the concepts that make them up. Training a multilayer perceptron network involves feeding data through the network and then backpropagating the loss to tune the weights of the connections.

The hyperparameters for our multilayer perceptron network are:

- *activation*: the activation function to use, either 'identity', 'logistic', 'tanh' or 'relu'
- *solver*: the weight optimizer to use, either 'lbfgs', 'sgd' or 'adam'

Matan Diamond
mdiamond8

Tuning Hyperparameters

Preprocessing Tuning

I anticipated the size of the feature vector produced from preprocessing to be an important hyperparameter to tune. Specifically, I assumed the space devoted to *types* and *keywords* would have a major impact on algorithm performance. I found a 0.48% standard deviation of accuracy doing a simple grid search over the range I assumed to be reasonable values, with no notable trend as to which values were better. The range indicated to me that these values would be less important than I first anticipated and opted to go with 25 for the minimum amount of total creatures *types* for a given type to be included (about 0.5% of the training data) and 100 for the amount of total space devoted to *keywords*[2]. With these settings, the full feature vector ends up around a length of 175, varying slightly depending on the training data fed into it.

Random Forest Tuning

To tune the random forests, I used the scikit learn GridSearchCV model selector which exhaustively searches over a list of parameters while performing k-fold cross-validation. I found that the number of *n_estimators* stopped mattering at values greater than 100 and that having no max depth was always most effective. I ended up using no max depth and a value of 370 for *n_estimators*.

Below is the data for different metrics for random forests[3].

accuracy_score	precision_score	recall_score	f1_score	hamming_loss
0.464627	0.516512	0.479218	0.484780	0.165522

Support Vector Machine Tuning

To tune the support vector classifier, I tried all available kernels with differing values of C. I ended up finding the linear kernel was significantly more accurate than every nonlinear kernel (by at least 30%) and that a *C* value of 4 gave the highest performance for the linear classifier.

accuracy_score	precision_score	recall_score	f1_score	hamming_loss
0.448941	0.491869	0.457312	0.462894	0.147325

[2] Technically, the *keyword* space was divided in half, 50 slots for keywordAbilities and 50 for keywordActions
[3] Calculated with *k*-fold cross validation, *k*=5 for each table

Matan Diamond
mdiamond8

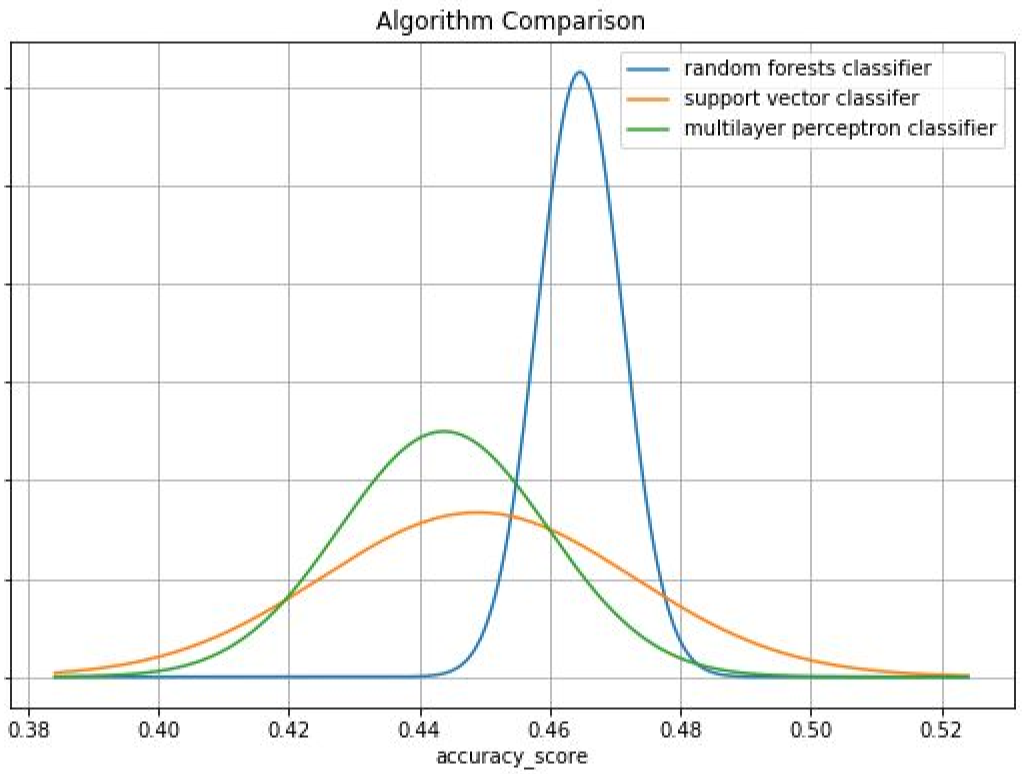
Multilayer Perceptron Classifier Tuning

I again used GridSearchCV to search over a list of parameters with k-fold cross-validation. I found that the default alpha of 0.0001 outperformed nearby values above and below it and that hidden layer size of 150 performed better than smaller sizes as well.

accuracy_score	precision_score	recall_score	f1_score	hamming_loss
0.443765	0.523739	0.518322	0.504187	0.170698

Comparing Algorithm Performance

Of the metrics used, I am convinced that we care most about the *accuracy* score for measuring performance. Some notion of "partial correctness" is valid since predicting a red-green card to be red (and only red) is better than predicting it to be blue, we are still interested in finding a model that best represents the color identities as well as the identities of the combinations. Thus, we want to maximize the number of samples we get completely right which the true *accuracy* score describes well.

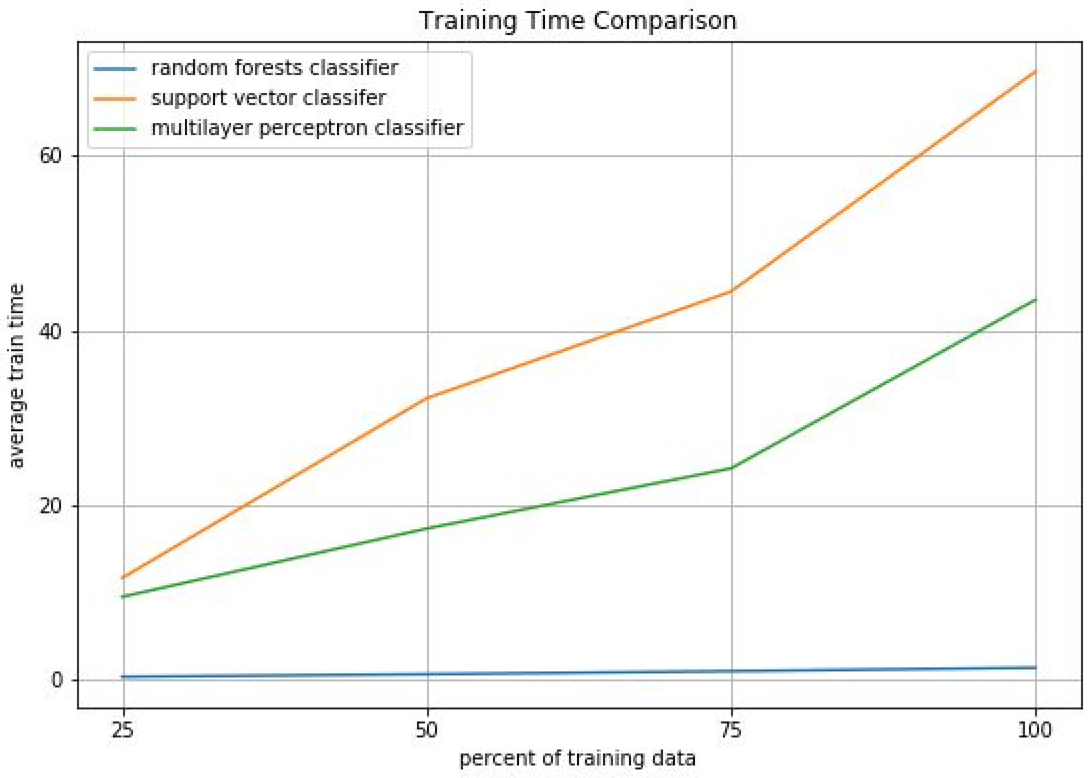


These confidence intervals are derived from the *accuracy* scores under *k*-fold cross validation where *k*=5. Random forests are clearly the best algorithm for our dataset based solely on performance, because they score higher on average and with lower variance.

Matan Diamond
mdiamond8

Conclusion

In addition to performance, we want to make sure that training times are reasonable for an algorithm we select. To test this, I sampled portions of the training data and timed the training for each algorithm to get an idea of how adding more training data might impact training time.



Random forests not only train in considerably less time than the other algorithms, but they also remain fairly constant in their time to process the data. Thus, based on their better and more consistent performance and their ability to take on more data (i.e. should this be applied to all cards instead of just creatures) I would select random forests with bootstrap aggregation to be the best learning algorithm for the domain of *Magic: the Gathering* cards.

Acknowledgements

- Python libraries such as pandas, numpy, matplotlib, time, json, regex
- Scikit-learn was used for the majority of the problems in this assignment, including model selection, preprocessing, metrics and the learning algorithms themselves.
- I learned a lot about working with multiple input classes and multi-label output data, including this page on metrics and this stackexchange answer on ways to combine features.
- All *Magic* card images can be found on Gatherer.