Submitted 16 12 2024

Multi-class Classification of Magic the Gathering Card Color

Carter Conboy

ABSTRACT

This paper reports a ML model which classifies the color of Magic the Gathering cards based on their rules text. Using text embedding and Softmax Regression the model performed 3x better than random guessing and marginally better than One-Versus-All Multi-Class Classification.

INTRODUCTION

Magic The Gathering (MTG) is a popular trading card game which contains over 26,000 unique cards. Players select subsets of these cards to build decks which they use to play against each other. Each card is labeled as belonging to 1 of 5 colors: Blue, Black, Red, White or Green. The game's design philosophy attempts to maintain a "color pie" where each color of card owns a unique style and set of game mechanics. The mechanics of any specific card are defined by it's rules text, therefore the "color pie" enables the classification of a card's color through this text. An effective solution to this problem could have broader applications, including the detection of when a card's mechanics do not match it's color (known as a "color pie break") which can be useful for deck development. Additionally, this model can be applied as a benchmark for future MTG card vector representations, which are integral to the development of MTG card search engines or Game Bots.

Data Acquisition and Prepossessing

The rules text and color for all cards in "core sets" (sets of cards which represent the games fundamental design direction) were sourced from web scraping the popular card database ScryFall. In order to convert the card's rules text into numerical representations which could be used for training each was processed through the text embedding pipeline in Figure 2. Gensim's Doc2Vec model was used, which leverages Continuous bag of words and Skip gram techniques to generate semantically meaningful feature extractions. To avoid data leaks, the model is trained using the corpus of documents from the train partition of the data, then the trained Doc2Vec model generates embeddings for both partitions. Finally we are left with data in the following form:

$$\mathrm{D} = \{(x_p, y_p) \,|\, p = 1,2,\ldots,P\}, \tag{1}$$

Figure 1. Basic Card Structure with Rules Text and Color Highlighted

where P is the number of cards, K is the number of classes (5), $y_p \in \{1,2,\ldots,K\}$ representing class labels, and $x_p \in \mathbb{R}^N$ is an N-dimensional feature vector.

Card Color Classification as an Optimization Problem

The objective of binary classification is to design a conditional probability model what will push $P(y_p|x_p) = 1$ resulting in a linear decision boundary in the form of $\hat{w}^{*T}\hat{x_p}$ where the sign of the result will predict label $\hat{y_p}$ for given sample $\hat{x_p}$. This can be expanded to multi-class classification by finding K binary classifiers which distinguish between the kth class and non kth classes. Resulting in K parameter vectors:

$$\mathrm{W} = [\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_K] \tag{2}$$

such that each $\hat{w}_k$ represents the parameters for linear boundary $\hat{w}_k^{*T}\hat{x} = 0$, where k = 1,2, … , K. Then by normalizing each parameter set by their feature touching weights we can interpret the output of the model as the distance between the sample and the decision boundary, with the maximum distance between each model's output as the

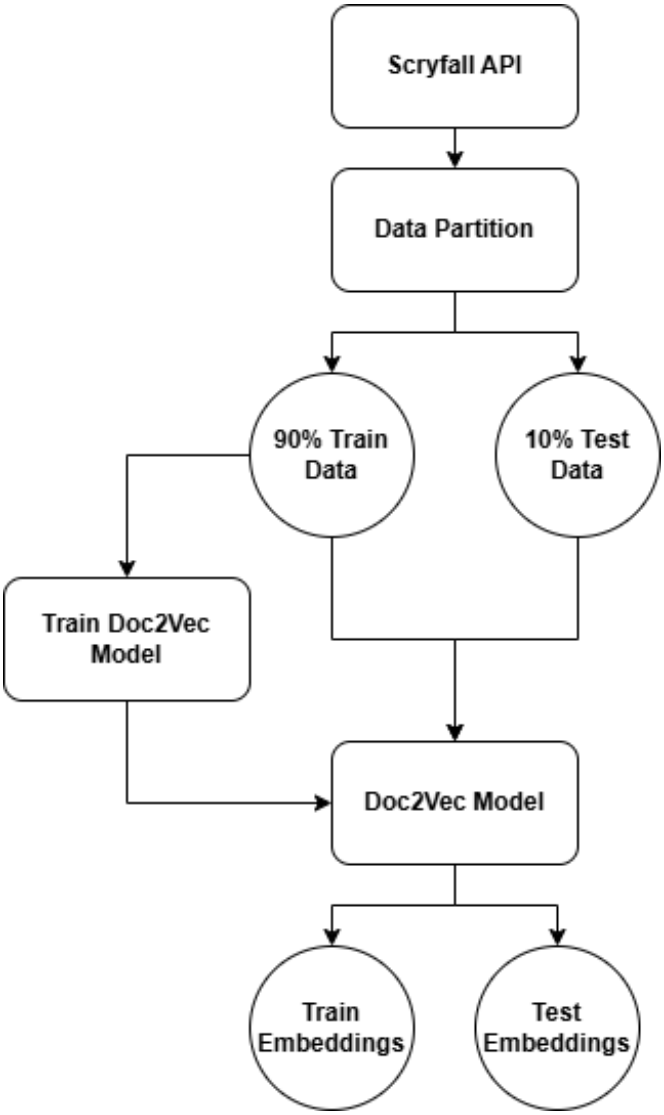

Figure 2. Text Embedding Pipeline. The card information is sourced from ScryFall, then the embeddings are generated using a Doc2Vec Model trained on the train partition to avoid data leaks.

predicted label. This solution takes the form:

$$y = \underset{k=1,2,\ldots,K}{\text{argmax}} \; \hat{w}_k^T \hat{x}_p \tag{3}$$

Given this general solution to the multi-class classification problem we can formulate the current card color classification as an optimization problem. Because the model with the largest output will be $\hat{w}_{y_p}$the For any given sample (x_p, y_p) then

$$\max_{1 \le j \le k} (\hat{w}_j^T \hat{x}_p) = \hat{w}_{y_p}^T \hat{x}_p \tag{4}$$

By minimizing the average difference between these terms across all training samples we are able to calculate the optimal parameter sets for each class, thus the training objective is to minimize the following function:

$$f(\hat{w}_1, \hat{w}_1, \ldots, \hat{w}_k) = \frac{1}{P} \sum_{p=1}^{P} \left[\max_{1 \le j \le k} (\hat{w}_j^T \hat{x}_p - \hat{w}_{y_p}^T \hat{x}_p) \right] \tag{5}$$

Using softmax to convert to a smooth function and adding a regularization term we obtain the final objective function (Softmax Regression):

$$f(\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_k) = \frac{1}{P} \sum_{p=1}^{P} \left[\log \left(\sum_{j=1}^{k} e^{\hat{w}_j^T \hat{x}_p} \right) - \hat{w}_{y_p}^T \hat{x}_p \right] + \frac{\mu}{2} \sum_{j=1}^{k} \|\hat{w}_j\|_2^2 \tag{6}$$

Additional Background Information

Feature Extraction: Feature extraction is a machine learning process which attempts to identify and extract important features from raw data. The goal is to provide the model with more informative data which will improve learning. In this project, the feature extraction technique used is text embedding, which converts raw text into a vector representation. The text embedding method used is Doc2Vec, which utilizes unsupervised learning to map sequences of text into a vector space such that semantically similar text samples are spatially clustered. The model is trained on a corpus of documents taking document ids and tokenized content as input, as well as user defined output dimensions and training epochs.

Dimensionality Reduction: Dimensionality reduction is used to convert high-dimensional data to lower dimensions in such a way that preserves relevant information about the data. In this project Principle Component Analysis (PCA) is used to project the data into 2-Dimensional space so that it can be visualized. PCA works by projecting the data onto orthogonal hyper-planes (principle components) which contain the highest variation. The principle components are the eigenvectors of the data's covariance matrix and thus can be calculated through a singular value decomposition (SVD) of the data.

Weight Regularization: Weight regularization is used to prevent over-fitting and increase training efficiency. In this project the regularization technique used is ridge regularization, which adds a penalty to large model weights. The regularization term as defined in the objective function is:

$$\frac{\mu}{2} \sum_{j=1}^{k} ||\hat{w}_j||_2^2 \tag{7}$$

Adding this term as a penalty to the loss function optimizes for low energy in the model parameters, which results in a lower condition number making gradient descent methods more efficient. Additionally the regularization term is independent of the dataset, which allows for better model generalization. The magnitude of this penalty is controlled by hyper-parameter μ.

Iterative Optimization Method: Minimization of the Softmax Regression cost function does not have a closed-form solution solution, thus iterative optimization techniques must be utilized. This project will use the memory-less Broyden-Fletcher-Goldfarb-Shanno Algorithm (ML-BFGS) which supplements gradient calculations with curvature information through gradual approximation of the Hessian. The algorithm is described below:

1) Compute the following where k is the current iteration and x is the optimization parameter:

$$\delta_k = x_{x+1} - x_k$$

$$\gamma_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

$$\rho_k = \frac{1}{\gamma_k^T \delta_k}$$

$$t_k = \delta_k^T \nabla f(x_{k+1})$$

$$q_k = \nabla f(x_{k+1}) - \rho_k t_k \gamma_k$$

2) Compute the search direction at $(k+1)$th iteration:

$$d_{k+1} = \rho_k \left(\gamma_k^T q_k - t_k\right) \delta_k - q_k$$

3) Repeat until end condition is met, in this case simply repeat if $k < iter$ where $iter$ is a number of specified iterations

MATERIALS AND METHODS

Dataset

The dataset contains cards from all core sets accessed from the ScryFall database. The training partition contained 3740 samples of 600 x 1 vectors for a total of 8.55 MB and the testing partition contained 935 samples of 600 x 1 vectors for a total of 2.13 MB.

The Model

The Softmax regression model has K sub-models, each with N weights and 1 bias term, where K is the number of classes and N is the number of features. Thus in this project the model had a total of 3005 parameters, each occupying 8 bytes in memory for a total modal size of 23.48 KB.

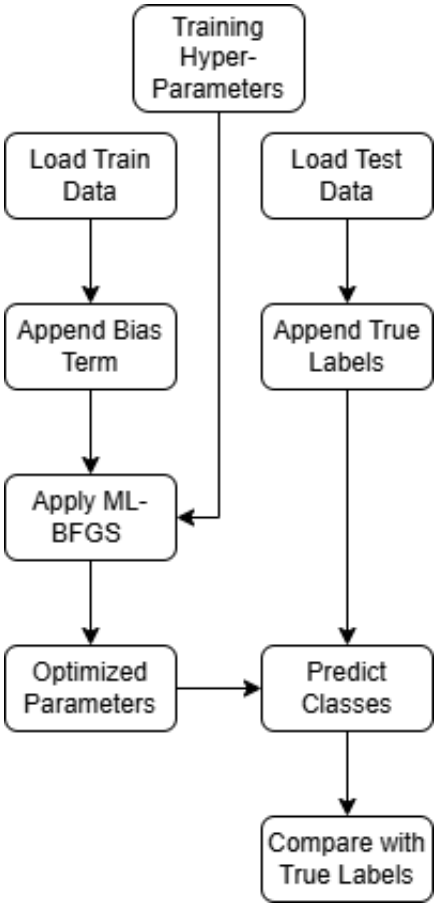

Figure 3. Training and Testing program Flow

Training

As described in the introduction the primary method used is formulating the data as a Softmax regression problem, then using the ML-BFGS iterative optimization method to solve for parameters to minimize it. The minimized Softmax cost function provides trained model parameters $(\hat{w}_1^*, \hat{w}_2^*, \ldots, \hat{w}_k^*)$ which are used to classify the test partition with the following formula:

$$\mathrm{j}^* = arg\left(\max_{1 \le j \le k} \hat{w}_j^{*T} \hat{x}\right) \tag{8}$$

where $\hat{x} = [x, 1]^T$, $j \in 1, 2, \ldots, K$ and j^* is the predicted class. ML-BFGS utilizes a predefined gradient of the Softmax function which is calculated to be:

$$\nabla_{\hat{w}_k} f(\hat{w}) = \frac{1}{P} \sum_{p=1}^{P} \left(\frac{e^{\hat{w}_k^T \hat{x}_p}}{\sum_{j=1}^{K} e^{\hat{w}_j^T \hat{x}_p}} \right) \hat{x}_p - \overline{x}_k + \mu \hat{w}_k \tag{9}$$

The training and testing process are implemented in Matlab and follow the outline shown in Figure 3. All code is made available in the appendix.

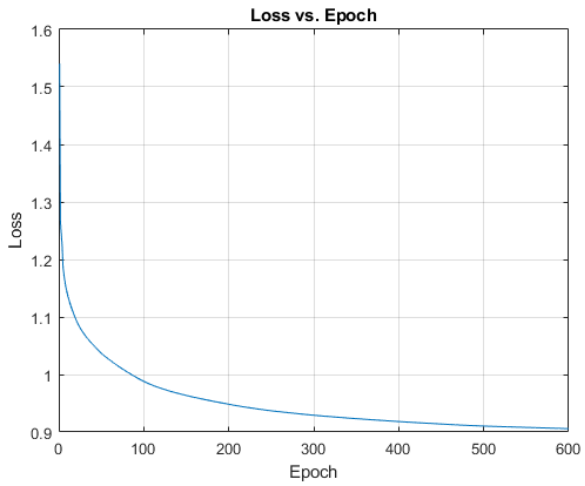

Figure 4. Training Loss per Epoch

Training parameters were selected heuristically. Higher dimensional text embeddings required more training epochs before converging which is expected with ML-BFGS, which will converge in n iterations for strictly convex functions, where n the number of model dimensions. Training loss per epoch is shown in Figure 4, and follows the expected trend of steep loss reduction flattening out as the optimization converges. Training was done on an Intel i7-8565U cpu and took approximately 0.2592 seconds per epoch.

RESULTS

This section reports test results for class prediction with the trained model on the test data partition. It also also reports results from a One-Versus-All classification with logistic regression for comparison. The following formulas were used to calculate the metrics:

$$\textbf{Accuracy} = \frac{\textit{Total Correct Samples}}{\textit{Total Samples}} \tag{10}$$

$$\textbf{Accuracy}_C = \frac{\textit{Total Correct Samples in } C}{\textit{Total Samples in } C} \tag{11}$$

Table 1. Training Hyper-Parameters

Parameter	Value
Epochs	600
μ	$1e-4$
Embedding Dimensions	600
Embedding Epochs	300

$$\textbf{Precision}_C = \frac{TP_c}{TP_c + FP_c} \tag{12}$$

$$\textbf{Recall}_C = \frac{TP_c}{TP_c + FN_c} \tag{13}$$

$$\textbf{F1 Score}_C = \frac{2 \cdot precision_c \cdot recall_c}{precision_c + recall_c} \tag{14}$$

where C is a class from $1,2\ldots,K$

DISCUSSION

The results demonstrate that softmax regression multi-class classification is a viable solution for MTG card color classification. An accuracy of 60.2% shows that the model is learning the dataset (3X better than random guessing). In this case False Positives are non-critical, therefore accuracy is a more important metric than F1 Score, even so, F1, precision and recall all score similarly, indicating that the results are not skewed by class imbalances. Class by class metrics reveal similar results for each class, with Green cards being the easiest to classify and White cards being the hardest. The confusion matrix in Figure 5 displays that White and Black cards were the most difficult to differentiate, this is expected as they share many common themes. Additionally, the model out performs One-Versus-All classification with logistic regression across all metrics indicating that the joint

Table 2. Average Model Performance

Accuracy	F1 Score	Precision	Recall
60.2%	0.6020	0.6033	0.6024

Table 3. Model Performance by Class

Color	Class Accuracy	F1 Score	Precision	Recall
Blue	60.1%	0.6339	0.6705	0.6010
Black	60.4%	0.5698	0.5385	0.6049
White	59.2%	0.5908	0.5894	0.5922
Red	60.5%	0.6137	0.6222	0.6054
Green	60.8%	0.6021	0.5959	0.6085

Table 4. One-Versus-All Model Performance

Accuracy	F1 Score	Precision	Recall
57.6%	0.5764	0.5770	0.5778

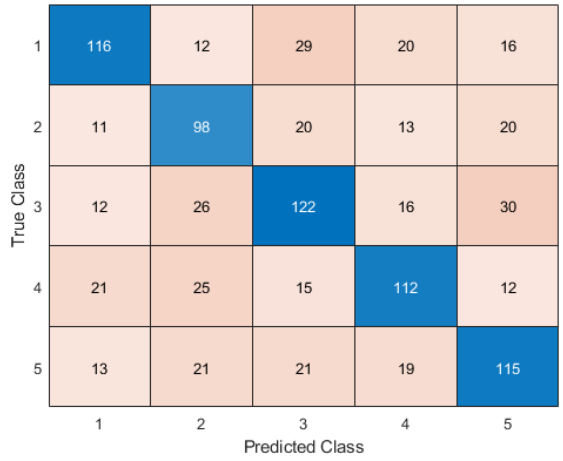

Figure 5. Confusion Matrix. Class Legend (Blue: 1, Black: 2, White: 3, Red: 4, Green: 5)

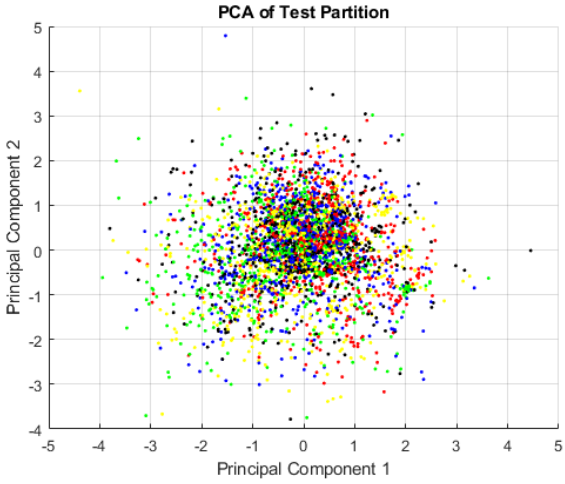

Figure 6. Test Partition Data Visualized in 2-D with PCA. Sample colors are correlated to class.

probability distribution and lower imbalance sensitivity of the Softmax regression model are beneficial to this problem.

The accuracy scores are somewhat low, but observing the dataset visualized in Figure 6 shows that this is clearly a difficult task, with no obvious clustering or boundaries for classes. The differentiability of the classes is controlled by two main constraints: the quality of the text embeddings and the strength of the ”color pie” within MTG card design. The former will always be a hard constraint on the problem, as there are no concrete limits to which cards can belong to which color. The latter is a clear frontier for improvement, as it is likely that the feature extraction in this project did not reach the limit of useful information retrievable from the card’s rules text. Finally, more complex models may be able to better learn the patterns in the data, but again, this is constrained by the information contained in the data itself.

CONCLUSION

The project presents a viable solution to the prediction of MTG card color based on rules text. Optimizing a Softmax Regression Cost function with rules text embeddings we are able to generate a set of optimal parameters which can be used to effectivly predict a cards color at over 60% accuracy.

ACKNOWLEDGEMENTS

I gratefully acknowledge the author of the external ML Functions used in the project, W.-S. Lu, as well as Professor Lei Zhao for his teaching and mentorship.

APPENDIX

Softmax Regression

```
load('x_test.mat');
load('x_train.mat');
load('y_test.mat');
load('y_train.mat');

Dtr = x_train;
Dte = x_test;
yte = y_test;
ytr = y_train;

tr_length = size(y_train, 2)
te_length = size(y_test, 2)
num_dimentions = size(x_train, 1);
size(Dtr)
size(ytr)
Dtr = [Dtr;double(ytr)];
Dte = [Dte; ones(1,te_length)];

K = 5;
mu = 0.00001;
iter = 600;

[Ws,f]=
    SRMCC_bfgsML(Dtr,'f_SRMCC','g_SRMCC',mu,K,iter)

%Generate Confusion Matrix
[~,ind_pre] = max((Dte'*Ws)');
C = zeros(K,K);
for j = 1:K
 ind_j = find(yte == j);
 for i = 1:K
 ind_pre_i = find(ind_pre == i);
 C(i,j) = length(intersect(ind_j,ind_pre_i));
 end
end

C
```

```
%Calculate Accuracy
accuracy = trace(C)/sum(C,"all")*100
precision = zeros(1, K);
recall = zeros(1, K);
f1_score = zeros(1, K);

%Calculate F1 Score
for i = 1:K
   TP = C(i, i);
   FP = sum(C(:, i)) - TP;
   FN = sum(C(i, :)) - TP;

   precision(i) = TP / (TP + FP);
   recall(i) = TP / (TP + FN);

   f1_score(i) = 2 * (precision(i) *
       recall(i)) / (precision(i) +
       recall(i));
end

f1_score
precision
recall

mean(f1_score)
mean(precision)
mean(recall)

confusionchart(C)
```

One-Versus-All Logistic Regression

```
load('x_test.mat');
load('x_train.mat');
load('y_test.mat');
load('y_train.mat');

Dtr = x_train;
Dte = x_test;
yte = y_test;
ytr = y_train;

tr_length = size(y_train, 2)
te_length = size(y_test, 2)
num_dimentions = size(x_train, 1);

% Normalize Training Data
Xtr = zeros(num_dimentions,tr_length);
m = zeros(1,num_dimentions);
v = zeros(1,num_dimentions);

for i = 1:num_dimentions
 xi = Dtr(i,:);
 m(i) = mean(xi);
 v(i) = sqrt(var(xi));
 Xtr(i,:) = (xi - m(i))/v(i);
end

Xte = zeros(num_dimentions,te_length);
for i = 1:num_dimentions
 xi = Dte(i,:);
 Xte(i,:) = (xi - m(i))/v(i);
end

% Done
iterations = 600;
mu = 0.0001;
models = zeros(num_dimentions+1,5);
% Train Models
for model_number = 1:5
   w0 = zeros(1,num_dimentions+1);

   ytr_model = ones(size(y_train));
   ytr_model(y_train ~= model_number) = -1;

   yte_model = ones(size(y_test));
   yte_model(y_test ~= model_number) = -1;

   D = [Xtr; ones(1,tr_length); ytr_model];
   [xs,fs,k] =
       grad_desc('f_wdbc','g_wdbc',w0',D,mu,iterations
   models(:,model_number) = xs;
end

Dte = [Xte; ones(1,te_length);];
correct_guesses = 0;
% Make Predictions
for p = 1:te_length
   [M,I] = max([Dte(:, p)'*models(:,1),
       Dte(:, p)'*models(:,2), Dte(:,
       p)'*models(:,3), Dte(:,
       p)'*models(:,4), Dte(:,
       p)'*models(:,5)]);
   if I == y_test(p)
      correct_guesses = correct_guesses + 1;
   end
end

accuracy = correct_guesses/te_length*100
```

Principle Component Analysis

```
load('x_test.mat');
load('x_train.mat');
load('y_test.mat');
load('y_train.mat');

size(x_test)
size(x_train)
size(y_test)
size(y_train)

[coeff, score, ~] = pca(x_train');

pc1 = score(:, 1);
pc2 = score(:, 2);
```

```
colors = [
    0, 0, 1;
    0, 0, 0;
    1, 1, 0;
    1, 0, 0;
    0, 1, 0
];

label_colors = colors(y_train, :);

figure;
scatter(pc1, pc2, 5, label_colors, 'filled');
xlabel('Principal Component 1');
ylabel('Principal Component 2');
title('PCA of Test Partition');
grid on;
```

Web Scraping

```
import requests
import pandas as pd
from sklearn.model_selection import
    train_test_split

# Define the sets
sets = ['M21', 'foundations', 'M20', 'M19',
    '8ED', '9ED', '7ED', '10E', '6ED', '5ED',
    '4ED',
    'M15','M14','M13','M12','M11','M10','ORI']

# Function to check if a card is a single
    color and not colorless
def is_single_color(card):
    # Check if the card has one color and is
        not colorless
    if 'colors' in card:
        return len(card['colors']) == 1 and
            'Colorless' not in card['colors']
    else:
        return False


def get_cards_from_set(set_code):
    url = f'https://api.scryfall.com
    /cards/search?q=set:{set_code}'
    cards = []

    while url:
        response = requests.get(url)
        data = response.json()

        for card in data['data']:
            if is_single_color(card) and
                card['layout'] != 'token': #
                Exclude tokens
               # Ensure no multiple printings
                if not any(c['name'] ==
                    card['name'] for c in cards):
                    cards.append({
                        'name': card['name'],
                        'color': '
                            '.join(card['colors'])
                            if 'colors' in card
                            else 'Colorless',
                        'oracle_text':
                            card.get('oracle_text',
                            '')
                    })
        url = data.get('next_page', None)
    return cards

all_cards = []
for set_code in sets:
    all_cards.extend(get_cards_from_set(set_code))

# Create DataFrame
df = pd.DataFrame(all_cards)

# Train Test Partitions
train_df, test_df = train_test_split(df,
    test_size=0.2, random_state=42)

# Save the DataFrame
test_df.to_pickle("df_text_test.pkl")
train_df.to_pickle("df_text_train.pkl")

print(len(test_df))
print(len(train_df))
```

Doc2Vec Model Training

```
import pandas as pd
import gensim
# Load DataFrame
df = pd.read_pickle("./df_text_train.pkl")
df_test =
    pd.read_pickle("./df_text_test.pkl")

df = pd.DataFrame(df['oracle_text'])
df_test =
    pd.DataFrame(df_test['oracle_text'])

# Preprocess the rules text
df['processed_text'] =
    df['oracle_text'].apply(
gensim.utils.simple_preprocess)
df_test['processed_text'] =
    df_test['oracle_text'].apply(
gensim.utils.simple_preprocess)

train_corpus = []
test_corpus = []

for index, row in df.iterrows():
```

```
    train_corpus.append(gensim.models.doc2vec.
    TaggedDocument(row['processed_text'],
        [index]))

for index, row in df_test.iterrows():
    test_corpus.append(row['processed_text'])

model =
    gensim.models.doc2vec.Doc2Vec(vector_size=50,
    min_count=2, epochs=40)
model.build_vocab(train_corpus)
model.train(train_corpus,
    total_examples=model.corpus_count,
    epochs=model.epochs)
```

Text Embedding

```
import pandas as pd
import numpy as np
from scipy.io import savemat

df_train =
    pd.read_pickle("./df_embedding_train.pkl")
df_test =
    pd.read_pickle("./df_embedding_test.pkl")

color_map = {'U': 1, 'B': 2, 'W': 3, 'R': 4,
    'G': 5}

def prepare_data(df, color_map):
    embeddings =
        np.array(df['embedding'].tolist()).T #
        Transpose so each column is a sample

    labels = df['color'].map(color_map).values

    return embeddings, labels

# Prepare train and test data
x_train, y_train = prepare_data(df_train,
    color_map)
x_test, y_test = prepare_data(df_test,
    color_map)

# Save to mat files
savemat('x_train.mat', {'x_train': x_train})
savemat('y_train.mat', {'y_train': y_train})
savemat('x_test.mat', {'x_test': x_test})
savemat('y_test.mat', {'y_test': y_test})
```

REFERENCES

1. W.-S. Lu, "LABORATORY MANUAL ECE 403/503 OPTIMIZATION for MACHINE LEARNING," 2021. Available: https://web.uvic.ca/ leizhao/2024-Fall/LabManual2021.pdf
2. "Gensim: topic modelling for humans," radimrehurek.com. https:radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html
3. "Lei Zhao," Uvic.ca, 2024. https://web.uvic.ca/leizhao/2024-Fall.php (accessed Dec. 15, 2024).