Name: Thanh Duong

Math 353: Probability and Statistics

Final Project

Popular combos of Clash Royale

GitHub Link: https://github.com/thanhqduong/Math353Project (https://github.com/thanhqduong/Math353Project)

Disclaimer: As the project deals with a really big size data, it may takes a long time to run the whole thing. My super old laptop takes ~ 4 minutes to run :(

1. Introduction

My data is about matches played in the game Clash Royale. Each of the players has 8 cards of different troops, spells, or buildings, to protect their castles and attack the others. Instead of playing each card individually, players usually think of "combos" by playing some appropriate combinations of cards at the same time. My project is to explore those combos that players have developed. The question I want to answer is "What is the cards that are usually played with this card," or more specifically, "Given the card A, what cards are usually played with."

2. Method

The data is obtained from: https://www.kaggle.com/datasets/bwandowando/clash-royale-season-18-dec-0320-dataset (https://www.kaggle.com/datasets/bwandowando/clash-royale-season-18-dec-0320-dataset)

From the dataset, I would get the deck (the set of 8 cards) from each match. Due to its large size, I randomly choose a random day to work with. I also remove all the duplicate deck.

The data cleaning part above is done in Excel. The cleaned data can be found at https://github.com/thanhqduong/Math353Project/blob/main/data.xlsx (https://github.com/thanhqduong/Math353Project/blob/main/data.xlsx)

For the game, on each level, you have new cards unlocked. If I try to recommend cards to every level, I may recommend cards to players which have not been unlocked yet. Therefore, on the scale of the project, I would focus only on the last-level player. It is really suitable for the dataset, as most of the players are on the last level.

3. The Math

The Machine Learning - Data Mining algorithm we will use is the Apriori algorithm. That is an Association Rule algorithm, which is used a lot in recommendation system. The algoritm itself has a lot of metric to use from; in the project, I would use the "Confidence" metric. The metric bases on the conditional probability mentioned in the Bayes Theorem P(A|B).

4. Assumption

I assume that if the two cards appear in decks the same time a lot, it means that players usually play them together as a combo. Although there are chances that players just want to have them together without using both of them, I think it is very unlikely. In fact, for the same Apriori algorithm, the metric "Lift" could be used to tackle this problem; however, I think it is out of the scope of the project.

5. The Code, and how-to-use instruction

Notation: The Markdown comment is the intruction to follow when running the code, while the in code comment (with "#") is for those who are more interested in the code.

The Model

From the github link, download the 2 files "data.xlsx" and "cardMasterList.csv" and put them inside the same folder as this file.

In [1]:
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas_profiling as pp
import scipy
import matplotlib.ticker as ticker
from sklearn import preprocessing
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

In [2]:
```
# load data
df = pd.read_excel("data.xlsx", sep=',')
```

In [3]:
```
# since the data is still really big, I only take a small sample of it.
df = df.sample(frac=0.08, replace=True)
```

In [4]:
```
df.head()
```

Out[4]:

	winner.card1.id	winner.card2.id	winner.card3.id	winner.card4.id	winner.card5.id	winner
377637	26000037	28000008	27000010	26000003	26000007	
311928	26000037	26000008	26000011	28000001	26000056	
82172	26000055	26000064	28000004	26000049	26000056	
240944	26000030	27000006	26000041	28000003	28000011	
97474	26000067	26000030	26000048	28000012	26000064	

In [5]:
```
# transform the data into a sparse matrix
records = []
for i in range(0, df.shape[0]):
    records.append([str(df.values[i,j]) for j in range(0, df.shape[1])])
te = TransactionEncoder()
te_ary = te.fit(records).transform(records)
df = pd.DataFrame(te_ary, columns=te.columns_)
df.head()
```

Out[5]:

	26000000	26000001	26000002	26000003	26000004	26000005	26000006	26000007	260000
0	False	False	False	True	False	False	True	True	Fal
1	False	False	False	False	False	False	False	False	Tr
2	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	Fal
4	False	False	False	False	False	False	False	False	Fal

5 rows × 102 columns

In [6]:
```
# the size of the sparse matrix; that is why we can only take a small sample
df.shape
```

Out[6]: (30965, 102)

In [7]:
```
# use Apriori, choose only cards with the probability of appearing more than
frequent_itemsets = apriori(df, min_support=0.004, use_colnames=True)
```

In [8]:
```
# choose the "Confidence" metric, only calculate those with the confidence of
rules = association_rules(frequent_itemsets, metric="confidence", min_thresho
```

The Result

Choose the card you want to find the combo: follow the link https://github.com/thanhqduong/Math353Project/blob/e2625142b7240f92db208084a2f768de4c (https://github.com/thanhqduong/Math353Project/blob/e2625142b7240f92db208084a2f768de4c to find the correct ID

In [9]:
```
# load the Master List to convert card ID to the card name
ml = pd.read_csv("cardMasterList.csv", sep=',', header = 0)
def getName(string):
    for i in range(len(ml)):
        if str(ml.id[i]) == string:
            print(ml.name[i])
            break
```

In [10]:
```
string = str(input("Card ID: ")) # type the ID of the card
n = int(input("Number of cards to print at most (the result may have less): "
print("You choose the card:")
getName(string)
l = []
for row in rules.iterrows():
    if string in row[1][0]:
        if len(row[1][0]) == 1:
            l.append((row[1][5], row[1][1]))
l = sorted(l, reverse = True)
n = min(len(l), n)
print("The card above is usually played with: ")
for i in range(n):
    print(str(i + 1) + ":")
    ds = list(l[i][1])
    for x in ds:
        getName(x)
```

```
Card ID: 26000003
Number of cards to print at most (the result may have less): 5
You choose the card:
Giant
The card above is usually played with:
1:
Zap
2:
Fireball
3:
Witch
4:
Arrows
5:
Wizard
```

Note: If you want to find the combos of another card, simply copy the cell above to the next cell. Below is an example.

In [11]:

```
string = str(input("Card ID: ")) # type the ID of the card
n = int(input("Number of cards to print at most (the result may have less): "
print("You choose the card:")
getName(string)
l = []
for row in rules.iterrows():
    if string in row[1][0]:
        if len(row[1][0]) == 1:
            l.append((row[1][5], row[1][1]))
l = sorted(l, reverse = True)
n = min(len(l), n)
print("The card above is usually played with: ")
for i in range(n):
    print(str(i + 1) + ":")
    ds = list(l[i][1])
    for x in ds:
        getName(x)
```

```
Card ID: 26000048
Number of cards to print at most (the result may have less): 4
You choose the card:
Night Witch
The card above is usually played with:
1:
Golem
2:
Baby Dragon
3:
Golem
Baby Dragon
4:
Lightning
```

In [ ]: