UNIVERSITÀ DI PISA

COMPUTER ENGINEERING

Data Mining and Machine Learning

CLASH ROYALE - DECK BUILDER

Fabio Cognata, Luca Di Giacomo

Academic year: 2021/2022

INDEX

DESIGN

- **INTRODUCTION**

The application is a tool for clash-royale's game community that allows decks building with the support of data analysis and machine learning algorithms to give an idea of the quality of the final deck analysing cards' parameters such as Health Points, Damage, and other statistics concerning the game.

The application also gives suggestions to build a competitive deck proposing specific combinations adapting to already selected cards.

It is thought of as a completely free to use application, without any user registration to make it as open as possible. It also offers information about each card retrieved by official sources such as the game's wiki.

- **REQUIREMENTS**

Functional Requirements

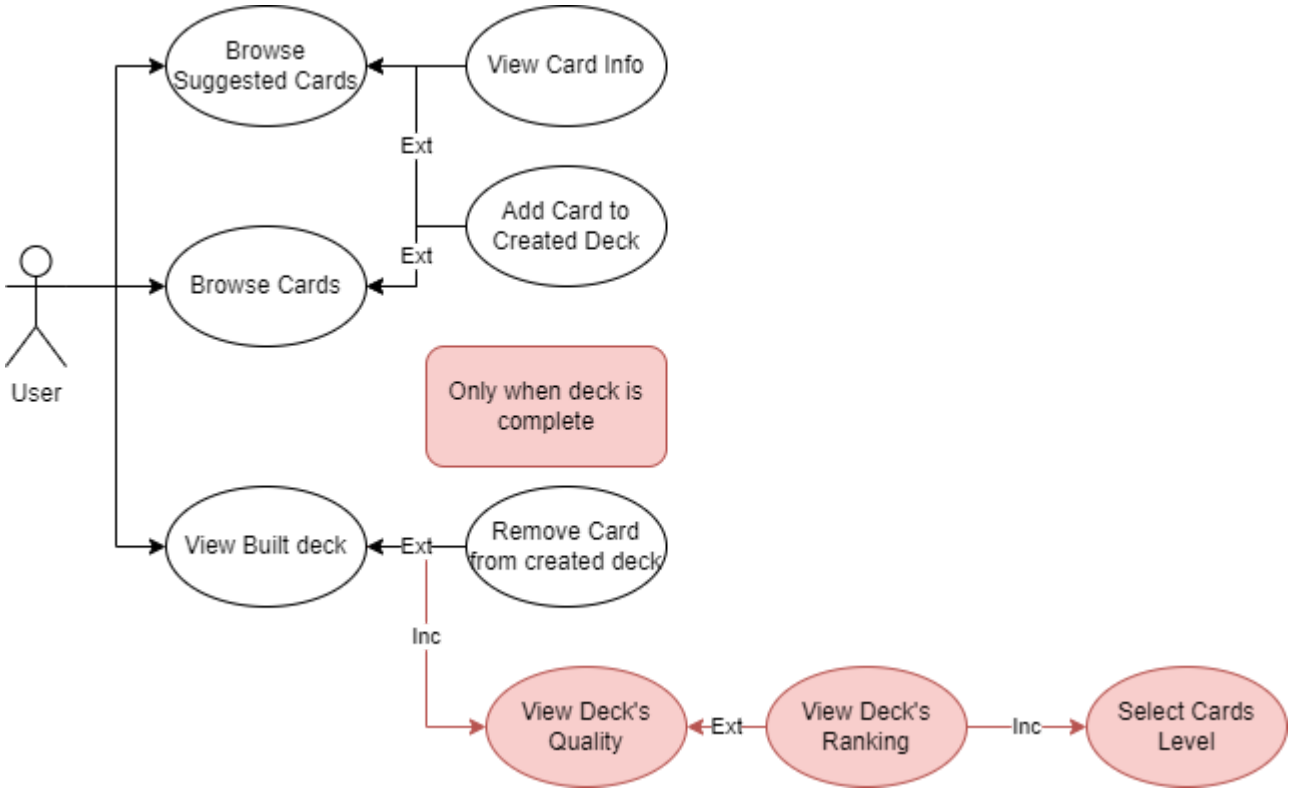
A user can:

- Explore the cards viewing informations about them
- Build decks by adding/removing specific cards.
- View suggestions based on the actual state of the deck during the building phase
- Receive a preview of the quality of the deck, once it is completely built
- Receive a ranking of the deck, once levels for each card are selected

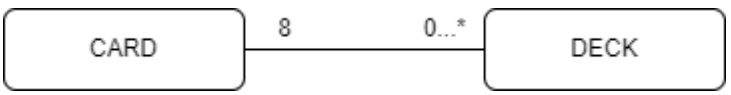
Non-Functional Requirements

- The application should be easy to use.
- The GUI should be intuitive and easy to use
- The application should be responsive to changes

- **USE CASES**



- **CLASS ANALYSIS DIAGRAM**



Card model


```
_id: ObjectId("61ec1d7733caab2faf2b39cd")
CID: 26000000
name: "Knight"
elixir: 3
type: "Troop"
rarity: "Common"
speed: 60
› hitpoints: Array
› damage: Array
count: 1
hps: 1.2
ability: false
range: 1200
attacks_ground: true
attacks_air: false
target_only_buildin… : false
target_only_troops: false
target_only_towers: false
imageUrl: "https://api-assets.clashroyale.com/cards/300/jAj1Q5rclXxU9kVImGqSJxa4w..."
```

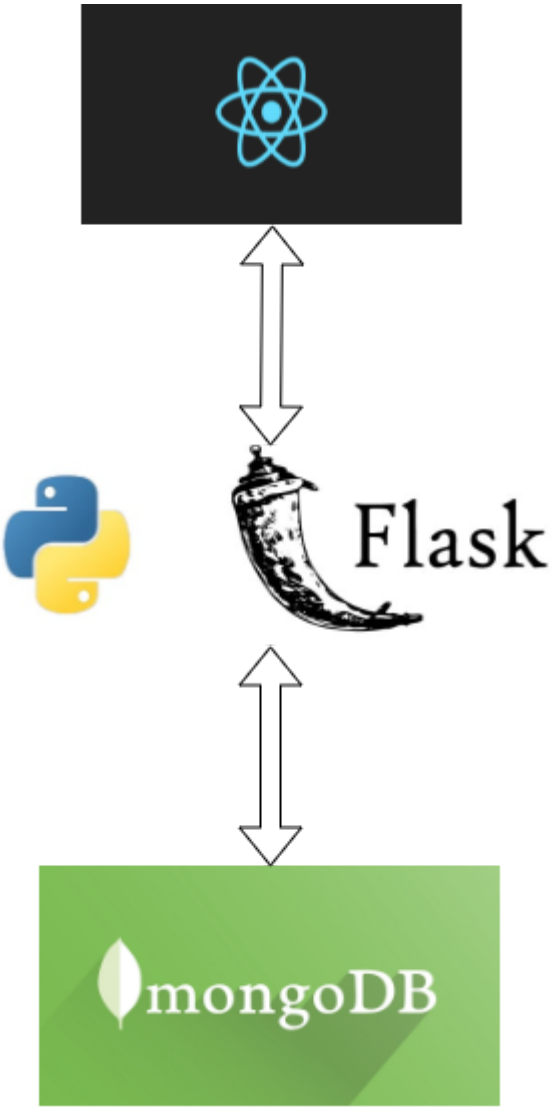

Deck model


```
_id: ObjectId("6204ef4c0ed73f2ecf5cd3ca")
tag: "#YY0PQ0UGY"
trophies: 3969
crowns: 3
› card1: Object
› card2: Object
› card3: Object
› card4: Object
› card5: Object
› card6: Object
› card7: Object
› card8: Object
troop: 6
building: 0
spell: 2
common: 0
rare: 1
epic: 4
legendary: 3
elixir: 3.25
rank: "Fun Deck"
conf_synergy: 1117.6393115803144
lift_synergy: 31655.633070878168
```

- **APPLICATION ARCHITECTURE**

The application has been developed using ReactJS Frontend based on Node environment for building a usable and user friendly GUI, interacting with an Anaconda's environment with Flask framework to build APIs.

As database, has been chosen as a Document-based DB given the high flexibility required by the cards that can be of various types and rarities involving different characteristics for each of them.



DATASET

- **DESCRIPTION**

The cards dataset has been built by official clash royale APIs at:
https://developer.clashroyale.com/#/
Retrieving 106 cards, then integrated with informations retrieved from both the official wiki at:
https://clashroyale.fandom.com/it/wiki/Clash_Royale_Wiki
And the statistics of the updated APIs from github:
https://royaleapi.github.io/cr-api-data/

For the decks dataset has been used a .csv file from kaggle, containing the full list of matches played during season 18 (2021) of Clash Royale:
https://www.kaggle.com/bwandowando/clash-royale-season-18-dec-0320-dataset

The final dataset resulted in a 1.05GB database containing both the cards and decks collection formatted as descripted above.

Volume: ~1GB

Variety: Multiple sources combined together in order to obtain the most possible informations about each deck and its performances in games

- **PRE-PROCESSING**

Data cleaning

Since the necessity of integrating multiple sources has been necessary to clean information from various sources that did not concern the application, and others that could be easily obtained by already existing data provided from official sources.

Some data also lacked crucial information such as card's damage for some newly added spells and buildings that had to be manually integrated from the game's wiki-page.

Numerosity reduction

The initial kaggle's dataset contained a huge amount of outdated information about cards and matches, thus, useless for the analysis such as deleted cards for matches that were played at the beginning of the season.
To handle that issue only the most recent games have been selected, up to the most recent relevant patch.

Data validation

An ulterior step has been added to finally update the cards with most recent information retrieved from: https://royaleapi.github.io/cr-api-data/.

Decks extraction

For decks building has been used the kaggle's dataset combined with the cleaned and validated information about cards, using the Card ID parameter as identifier for the updates.

Synergy extraction through FPA
Frequent pattern analysis has been used among the winning decks from matches collection to obtain a synergy score based on the rule's confidence and lift score to recognize frequent and good combinations of cards. The final score is a sum of all

the rules matched by the deck.

- **MODELS ANALYSIS AND EVALUATION**

Frequent pattern analysis for suggestions - FPA.ipynb

The pre-processing for that analysis consisted in an extraction of a list of card IDs formatted as strings retrieved from the winning matches list

```
cardList = []

for match in matchesList:
  localList = [str(match['card1']['id']),str(match['card2']['id']),str(match['card3']['id']),str(match['card4']['id']),
               str(match['card5']['id']),str(match['card6']['id']),str(match['card7']['id']),str(match['card8']['id'])]
  cardList.append(localList)
```

```
['28000001', '28000011', '27000000', '26000001', '26000005', '26000011', '28000000', '26000021']
['26000014', '26000012', '26000035', '26000042', '28000012', '26000018', '26000043', '26000045']
```

The script then saves a .csv for future uses and starts analysing the lists for finding frequent patterns.

For that analysis the library mlxtend has been used for using Apriori and FP-Growth algorithms of pattern finding, requiring a Dataframe with all the instances in the lists and a row for each list with True/False if the card ID is present in the row.

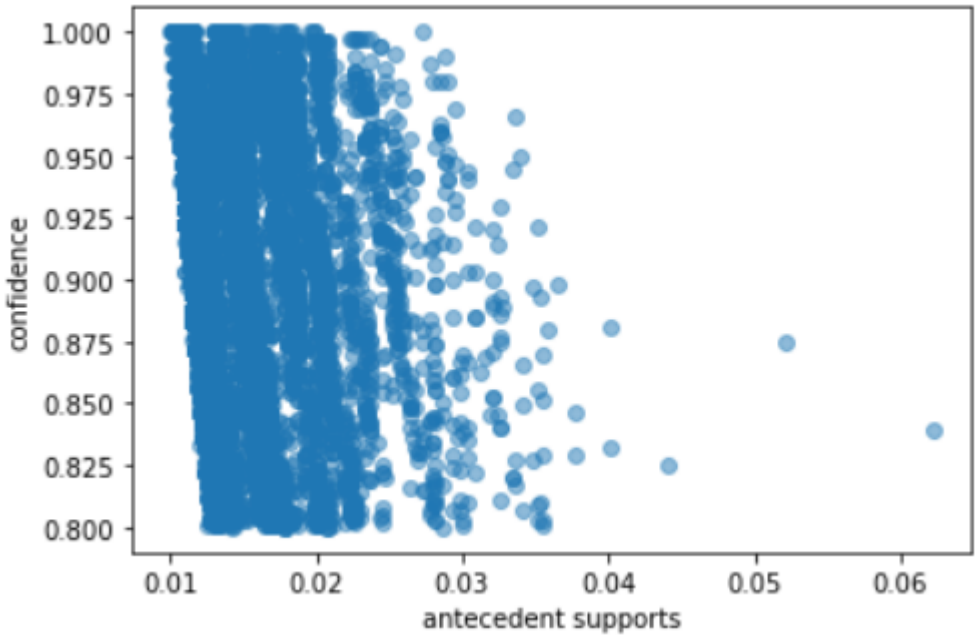
```
te = TransactionEncoder()
encoded_dataset = te.fit(cardList).transform(cardList)
df = pd.DataFrame(encoded_dataset, columns=te.columns_)
df.head(10)
```

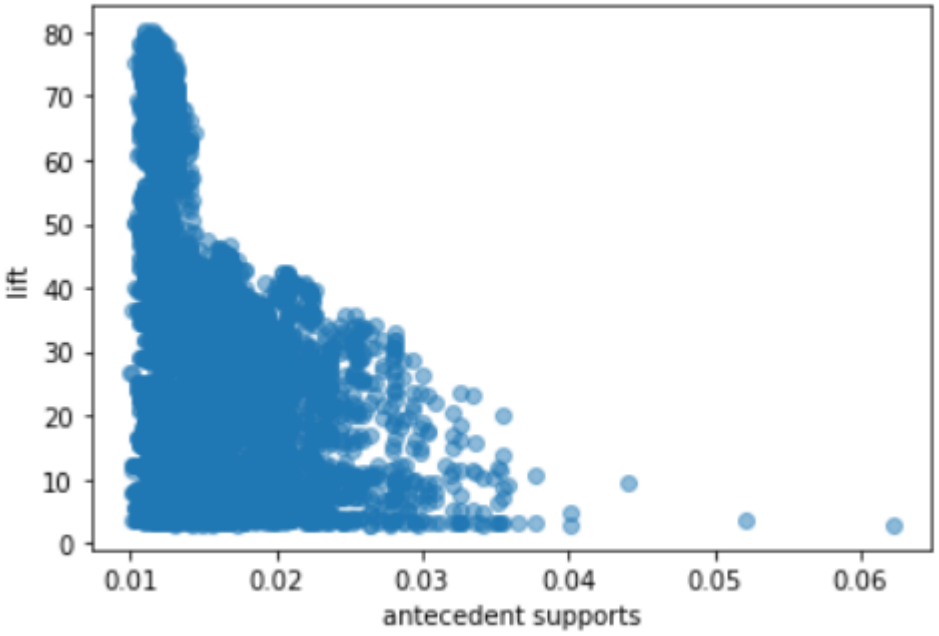
Out[3]:

	26000000	26000001	26000002	26000003	26000004	26000005	26000006	26000007	26000008	26000009	...	28000009	28000010	28000011	28000012	2800
0	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	F
1	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	F
2	False	False	False	False	False	False	False	False	False	True	...	False	False	False	True	F
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	F
4	True	False	False	False	False	False	False	False	True	False	...	False	False	False	False	F
5	False	False	False	False	False	False	False	False	False	True	...	False	False	False	True	F
6	False	True	False	False	True	True	False	False	False	False	...	False	False	False	False	F
7	False	False	False	False	False	False	False	False	False	False	...	False	False	True	False	F
8	False	True	False	False	False	True	False	False	False	False	...	False	False	True	False	F
9	False	False	False	False	False	False	False	False	False	False	...	False	False	False	True	F

10 rows × 102 columns

The metric chosen for the analysis has been the “confidence” measure because of the interest of finding the same combination of cards with a min_support of 0.01(1%).
The plotting below shows how the points are dense in the region between 0.01 and 0.04 without going under the 0.8 of confidence, while the lift is varying





FP-Growth performances were practically the same in terms of patterns discovered, but improving a lot in terms of speed.

For that reason, it has been chosen as the FPA algorithm in the application.

- **CLASSIFICATION MODELS**

INTRODUCTION

The classification has been carried out analysing the performances of various classification models combined with different attribute selection algorithms found in the sklearn library for python.

The core problems of the analysis were to differentiate between the various ranks grouped into three main ones (https://clash.world/news/clash-royale-leagues) by only analysing cards in the deck differentiating between players behaviours in the various leagues.

A first not-level-based analysis has been carried out to only recognize between decks that could be ranked into these three main categories labelled as "Competitive", and others not belonging to any of those, labelled as: "Fun Deck", then Competitive decks could receive a deeper level-based classification where the application tries to guess the maximum level at which the deck could be played with those statistics.

For both a pre-processing phase was required to extract main features of the decks, performed into the "CSV for Classifier.ipynb" notebook where cards of all the decks in the collection are analysed to obtain information and statistics of the deck, and extracting the rank label from the trophies amount of the player playing the deck.

Another issue was the unbalance of the dataset, since most of the records gathered into the "middle" region ("Challenger, Master").

In order to not create too many instances with the SMOTE algorithm, it has been decided to under-sample the initial dataset with most recent records obtained from the `battleTime` feature extracting 3.5k instances of each class sorted by decreasing battle-time.

The models have been, thus, evaluated by using shuffled 10-Fold cross validation with sklearn pipelines.

UNLEVELED ANALYSIS

For this analysis a pre-processing step for re-elaborate ranks was necessary, infact all the matches recorded below the 4000 trophies (minimal league) have been re-labeled as “Fun Deck” and all the others as “Competitive”. All cards of the matches also have been brought to the maximum level with all their statistics in order to transpose all the statistics and make the analysis coherent for all recorded matches.

The first table shows the result without any attribute selection, using all 23 attributes of the dataset.

NO ATTRIBUTE SELECTION						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	79,62%	77,59%	77,20%	77,36%	0,09s	0,01s
RF	85,87%	84,92%	82,90%	83,73%	1,19s	0,04s
KNN	70,95%	67,80%	67,11%	67,37%	0s	0,49s
NB	51,93%	65,84%	61,79%	51,07%	0,02s	0,01s

In the second table, a k-best selection has been performed setting the better performing parameter of k=15, selecting the following features with the relative scores:

troop	[246.201292868721]
building	[278.96481449969025]
spell	[388.78841535763866]
common	[738.7918549280175]
epic	[1221.3164203726903]
legendary	[256.00679070290647]
conf_synergy	[13019517.496775448]
lift_synergy	[365556911.39465773]
avgSpeed	[287.8639620945912]
avgSpellDuration	[912.3669917208254]
totalHP	[283438.2331835143]
totalDMG	[16881.910918981455]
totalUnitsCount	[4488.421084104702]
avgRange	[9001.664319770227]
totalBuffCount	[662.2977576257209]

SELECT K-BEST + Chi^2 Test(K=15)						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	79,40%	77,26%	77,20%	77,22%	0,13s	0,02s
RF	85,09%	84,17%	82,38%	83,12%	1,13s	0,05s
KNN	71,35%	68,22%	67,61%	67,86%	0,07s	0,08s
NB	51,81%	65,86%	71,73%	50,93%	0,03s	0,02s

As Final analysis a test with variance threshold selector has been run with a threshold of 0.7 giving the best results, reported in the table below:

VARIANCE THRESHOLD (T=0.7)						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	78,77%	76,58%	76,53%	76,55%	0,09s	0,01s
RF	79,27%	77,14%	76,93%	77,02%	0,08s	0,01s
KNN	71,06%	77,86%	67,25%	67,49%	0,02s	0,4s
NB	51,95%	65,96%	61,85%	51,07%	0,03s	0,02s

Selecting the following 17 attributes:

Troop-count, spell-count, common-count, rare-count, epic-count, legendary-count, conf_synergy, lift_synergy, avgSpeed, avgSpellDuration, totalHP, totalDMG, totalUnitsCount, abilityCount, avgRange, totalAirT, totalProjectiles

LEVELLED ANALYSIS

For this analysis the level has been used as a total of all card levels with statistics matching it. In order to increase the quality of results, the levels have been splitted as the average of the various cards' rarity.

The main problem that occured in this analysis was the decreasing differences of the various classes, infact as more the rank of players increases the more the play-style became similar which made skill and luck the principal factors of influence during games.

The following tables show the results of the various analysis performed:

NO ATTRIBUTE SELECTION						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	70,18%	69,74%	70,18%	69,56%	0,1s	0,01s
RF	75,93%	76,99%	75,93%	76,21%	3,12s	0,2s
KNN(N=4)	55,35%	54,10%	55,35%	53,74%	0,01s	1,4s
NB	49,32%	49,40%	49,32%	48,09%	0,05s	0,06s

K-BEST with Chi^2 (K=20)						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	69,88%	69,31%	69,88%	69,16%	0,12s	0,01s
RF	74,31%	75,18%	74,31%	74,54%	3,07s	0,32s
KNN(N=4)	55,27%	53,96%	55,27%	53,66%	0,07s	0,09s
NB	49,07%	49,09%	49,07%	47,86%	0,17s	0,07s

SELECTED ATTRIBUTES:

troop	[182.09327448019513]
building	[98.74779860013547]
spell	[309.262796536353]
rare	[207.91883647456152]
epic	[73.49730700179533]
legendary	[110.59477738117602]
conf_synergy	[9027406.667200817]
lift_synergy	[237147162.38026124]
avgSpeed	[513.7494146228144]
avgSpellDuration	[91426.95679346568]
totalHP	[21375.76727481534]
avgAS	[1980.2577645839751]
totalUnitsCount	[126.93418333699715]
abilityCount	[18234.55077614098]
totalBuildingT	[166.07904626150062]
totalBuffCount	[95.59092804135746]
totalProjectiles	[336.18342207623937]
commonLevel	[415.4951361221885]
rareLevel	[372.5667253924158]
epicLevel	[1207.7050332101444]
levelSum	NaN

K-BEST with Mutual Info Gain (K=20)						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	69,38%	68,87%	69,38%	68,72%	0,41s	0,01s
RF	75,35%	76,39%	75,35%	75,64%	3,90s	0,15s
KNN(N=4)	55,30%	54,00%	55,30%	53,68%	3,05s	0,1s
NB	48,87%	49,01%	48,87%	47,51%	4,72s	0,03s

SELECTED ATTRIBUTES:

troop	[0.06104933324304729]	[0.05452604087106794]	[0.05377208460959482]	[0.06986774526032935]
spell	[0.0625394403876045]	[0.06260217589945372]	[0.06494543552275656]	[0.05570396508043984]
rare	[0.021065408319623558]	[0.02346274861713149]	[0.02200782626607456]	NaN
legendary	[0.023727319749998532]	[0.02669141830592414]	[0.018447530081513897]	[0.024660625079550025]
elixir	[0.037685326596771374]	[0.0466905672391007]	[0.05844554967532889]	[0.041014937962012166]
conf_synergy	[0.17808920971462916]	[0.1816916379803868]	[0.17316333092202973]	[0.17979716750734043]
lift_synergy	[0.18471094044727132]	[0.1758109630235487]	[0.17113136824151676]	[0.18048176039659825]
levelSum	[0.07464715620516871]	[0.07385930640699567]	[0.07180818294344848]	[0.07484545061364223]
avgSpeed	[0.11988435619150928]	[0.10695008834944764]	[0.11448761564433285]	[0.11752239325244918]
avgSpellDuration	[0.19868962771244458]	[0.19629674093966432]	[0.19651172490646074]	[0.19469028597552773]
totalHP	[0.17273266245891716]	[0.18107027002514053]	[0.17858610823820587]	[0.18478258675871628]
totalDMG	[0.10753911874276456]	[0.12002177361787347]	[0.10291570710662312]	[0.10930445883584072]
avgAS	[0.048285060818054726]	[0.040458624844885493]	[0.04122462131717919]	[0.041658267990717146]
abilityCount	[0.08426028413900699]	[0.09661452937467896]	[0.08624598995468236]	[0.09055330608953849]
totalAirT	[0.014051446295923142]	[0.01697781614823146]	NaN	NaN
totalBuildingT	[0.017833707125327614]	NaN	[0.017973940065784344]	NaN
totalProjectiles	[0.2010088489269075]	[0.1875355453573595]	[0.19362488601030736]	[0.19849681155852616]
commonLevel	[0.2330814787329074]	[0.23006980311746617]	[0.23225638438187346]	[0.22908628687894939]
rareLevel	[0.25264882050411797]	[0.25265155338016876]	[0.2521237357351842]	[0.23800730202304754]
epicLevel	[0.32848335704790665]	[0.3192800976141805]	[0.3147538904842411]	[0.3216168434281501]
building	NaN	[0.019300308527368415]	NaN	[0.017375958008571946]
epic	NaN	NaN	[0.029875160910072918]	NaN
totalGroundT	NaN	NaN	NaN	[0.014661301737639576]
totalBuffCount	NaN	NaN	NaN	[0.019689047952134775]

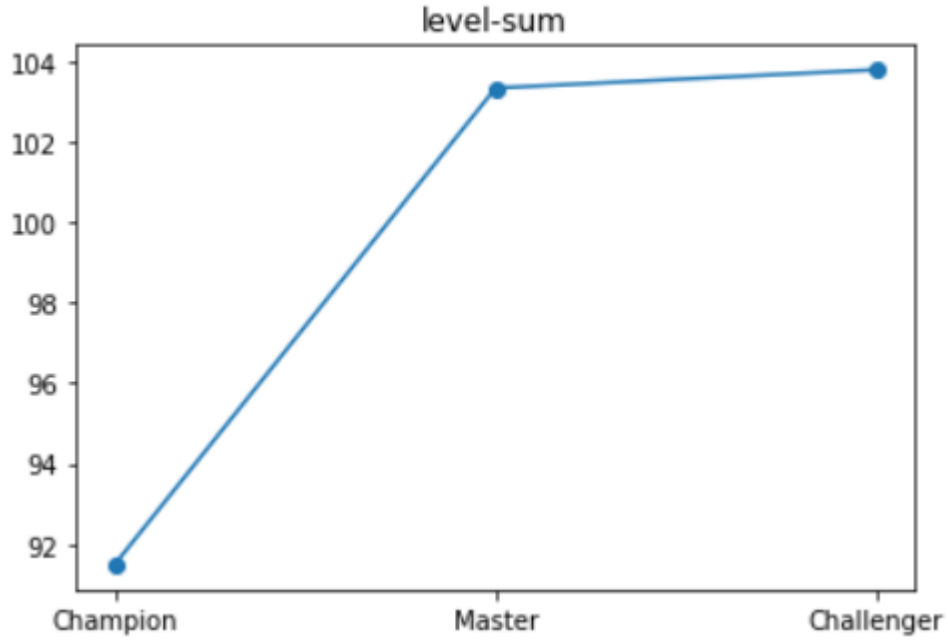
VARIANCE THRESHOLD (T=0.7)						
Algorithm	Accuracy	Precision	Recall	F-Measure	fit-time	score-time
J48	69,88%	69,48%	69,88%	69,24%	0,08s	0,01s
RF	75,22%	76,30%	75,22%	75,51%	2,57s	0,14s
KNN(N=4)	55,19%	53,92%	55,19%	53,63%	0,03s	1,1s
NB	49,31%	49,41%	49,31%	48,07%	0,05s	0,02s

SELECTED ATTRIBUTES:

troop
spell
common
rare
epic
legendary
conf_synergy
lift_synergy
levelSum
avgSpeed
avgSpellDuration
totalHP
totalDMG
totalUnitsCount
abilityCount
avgRange
totalAirT
totalProjectiles
commonLevel
rareLevel
epicLevel
legendaryLevel

- **ANALYSIS FINAL CONSIDERATIONS**

It’s noticeable that, from the analysis, all results are highly influenced by players’ skill, specially with the growth of the rank.
In Fact in most of the cases players tend to follow the trend of higher level players copying their deck, resulting in few visible differences such as cards level and rarity of the owned cards indicating also the amount of time spent on the game with cards also having minimal properties’ differences for each level resulting in very near classes.



Synergy helped to distantiate classes tracing the behaviour of the various ranks, increasing almost linearly with the level of the players, but with huge variations: