Clash Royale Match Analysis and Machine Learning Prediction Using Player Deck Data

Eylül Pelin Kılıç

Data Preparation

I used Clash Royale API to fetch data. My raw data(matches_50_players.xlsx) consisted of the last 25-30 of the match data from 50 players that use the most popular 5 decks (approximately 1500 matches). These decks were:

```
deck0: ['Zap', 'Goblin Giant', 'Dark Prince', 'Heal Spirit', 'Elite Barbarians', 'Electro Wizard', 'Rage', 'Sparky']
```

```
deck1: ['Executioner', 'Giant Snowball', 'Ram Rider', 'Boss Bandit', 'Lightning', 'Royal Ghost', 'Guards', 'Bandit']
```

```
deck2: ['Goblin Giant', 'P.E.K.K.A', 'Goblin Machine', 'Mega Minion', 'Goblin Curse', 'Bomber', 'Arrows', 'Rage']
```

```
deck3: ['Executioner', 'Giant Snowball', 'Ram Rider', 'Boss Bandit', 'Lightning', 'Royal Ghost', 'Guards', 'Barbarian Barrel']
```

```
deck4: ['P.E.K.K.A', 'Goblin Giant', 'Mega Minion', 'Goblin Machine', 'Goblin Curse', 'Bomber', 'Arrows', 'Rage']
```

Although, I listed the players who frequently used these decks from RoyaleAPI and fetched their data, there were also matches that used other decks because every player can change their deck from match to match. So, I added a new column named team_deck_name that labeled the most frequent decks and named not frequent decks as ‘other’. (Categorization)

The match data consisted of team_deck, opponent_deck, team_avg_elixir, opponent_avg_elixir, and win. At first, I checked if there were any NaN values, to handle missing data bu there was not.

Then, I encoded the match data with the cards that were used in these matches by adding columns like “team_has_*card_name*” and “opponent_has_*card_name*”(Multilabel Binarizer).

Also, to get a better understanding of the matches, I had to determine which cards were of which types(eg: Electro Wizard, ['ranged', 'air', 'stun', 'support']). Thus, I fetched another data (card_types_full_multi_labeled.xlsx) that contained the attributes: card_name, card_type and elixir.

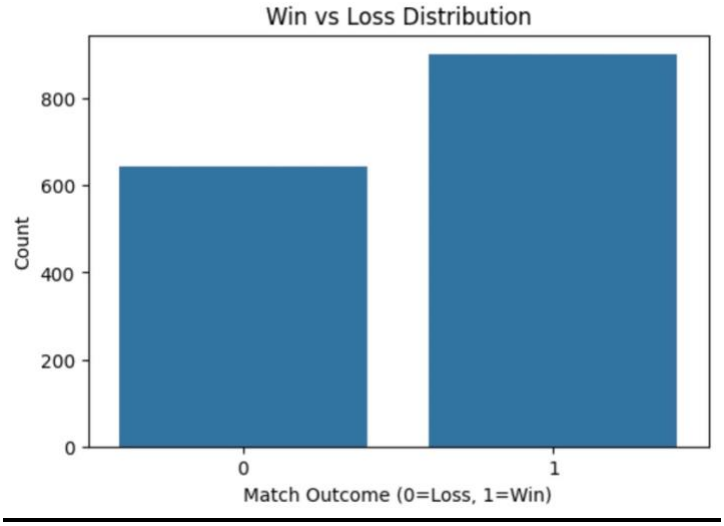
By using this new data I also encoded the matches with the card types that were used in this match and added new columns as “team_n_*card_type*” and “opponent_n_*card_type*”. These

columns showed how many cards of this type were used in the deck.(Custom Frequency Encoding)

At the end I had the data with the merged information from these two tables and finalized as “matches_50_players_prep.xlsx”. This data consisted of approximately 350 columns and 1500 match instances. This would further lead to “curse of dimentionality“(as there were so much columns), so I handled the data accordingly by using the appropriate methods such as (????)

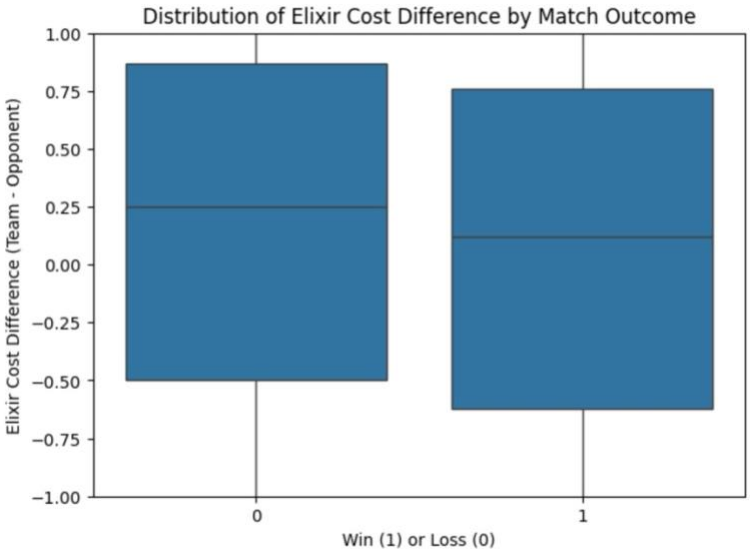
EDA

Insight 1:



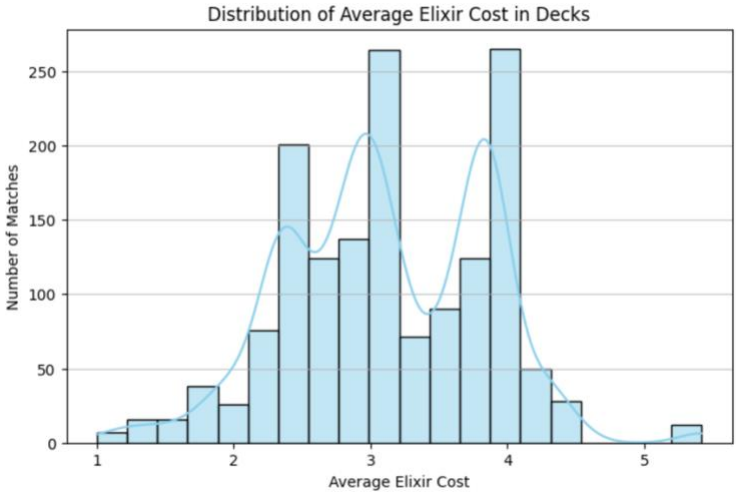
This bar chart visualizes the distribution of match outcomes in the dataset, categorizing matches as wins (1) or losses (0). This imbalance suggests that players or decks in this sample generally have a higher chance of winning, which might influence model training and evaluation. It will be important to consider this distribution when developing predictive models to ensure balanced performance and to potentially apply techniques such as resampling or class weighting if necessary.

Insight 2:



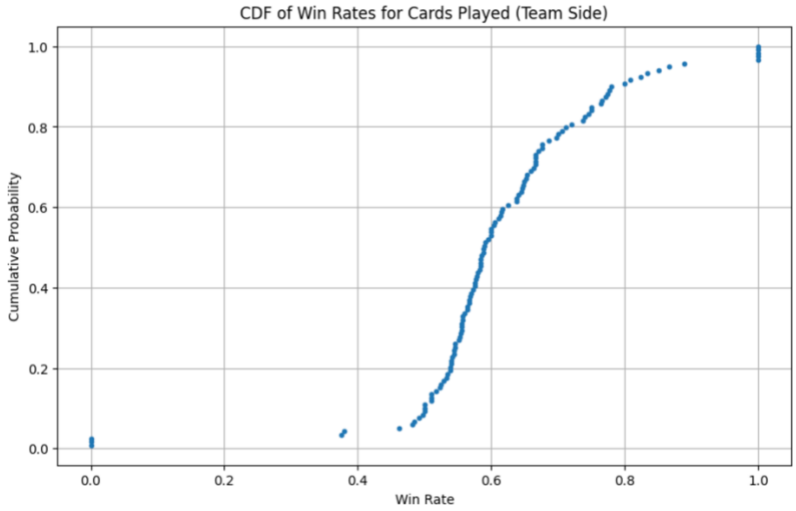
This boxplot visualizes the distribution of the elixir cost difference (team's average elixir cost minus opponent's average elixir cost) grouped by match outcome (win or loss). The median elixir cost difference for winning matches is slightly lower than for losing matches, suggesting that teams tend to win more often when their average elixir cost is closer to or slightly less than their opponent's.

Insight 3:



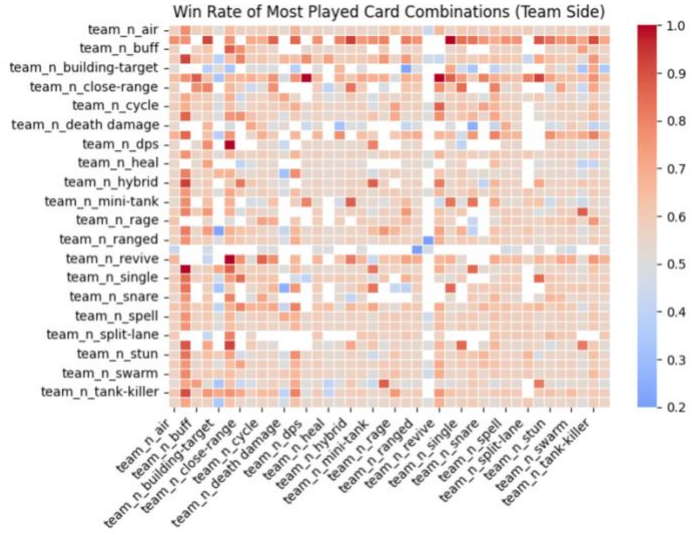
This histogram shows that the average elixir cost in decks mostly falls between 2 and 4. The distribution is bimodal, with two clear peaks around 3 and 4 elixir. This suggests that decks tend to cluster around these elixir costs. This insight supports further analysis on how elixir cost categories might influence match outcomes or deck performance.

Insight 4:



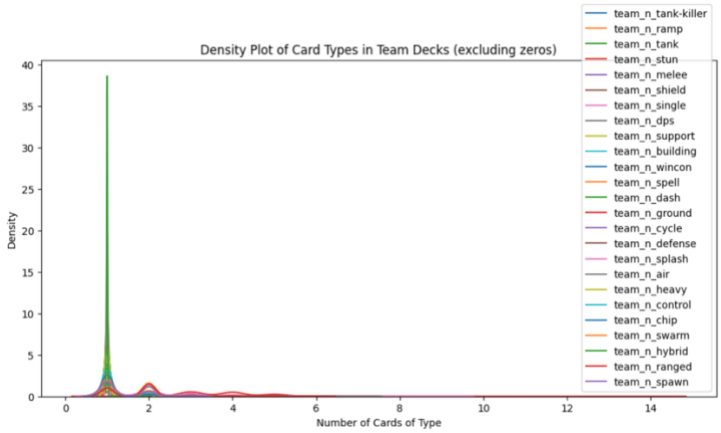
This CDF plot illustrates the cumulative distribution of win rates for cards played on the team side. Most cards have a win rate clustered between approximately 0.4 and 0.8, indicating a moderate level of effectiveness for the majority of cards. This distribution helps identify which cards consistently contribute to winning outcomes and which have more variable success, guiding strategic deck-building decisions.

Insight 5:



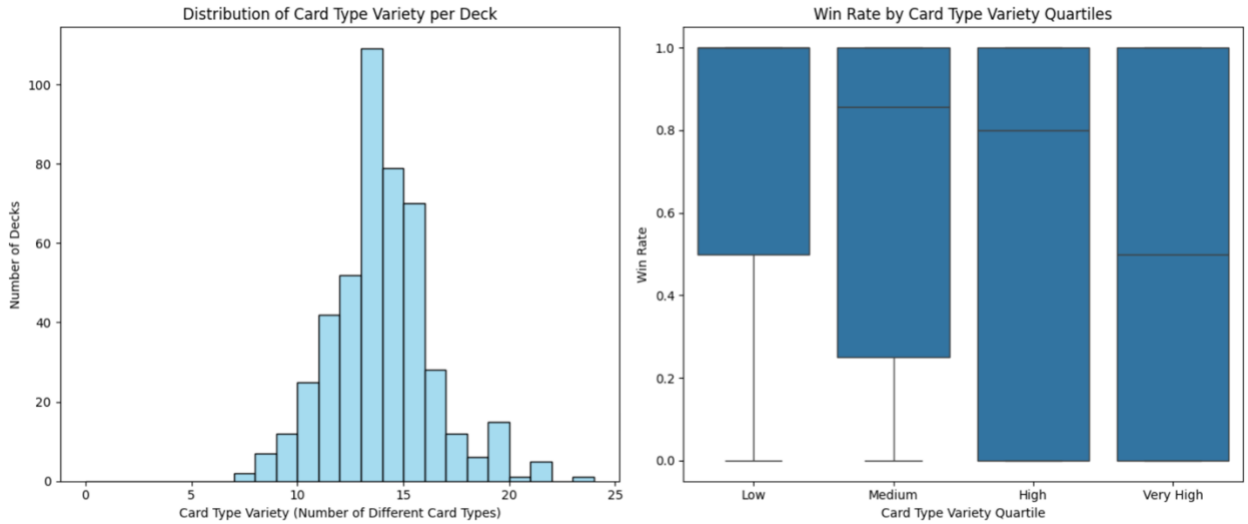
The heatmap reveals that certain pairs of cards types significantly contribute to higher win rates when played together in matches.

Insight 6:



This density plot reveals the distribution of card type counts per type in team decks, excluding zero counts. Most card types cluster around 1 or 2 cards per deck, indicating players typically include a limited number of cards from each type, which reflects strategic diversity without overloading decks. The peak at 1 card for many types suggests a standard approach where players prefer including one key card of a certain type rather than multiples.

Insight 7:



Most decks have 10–20 different card types, with a peak around 13–15. I thought it seemed likely to a normal distribution but I was not sure. In the right part, we see while low variety decks show consistent win rates, very high variety decks have more varied success.

Hypothesis Testing

HT1:

Null Hypothesis (H0): The average elixir cost is the same for winning and losing matches.

Alternative Hypothesis (H1): The average elixir cost is lower in winning matches than in losing matches.

***: I used Mann-Whitney U compared to t-test because I was sure my average elixir cost was not normally distributed.(With the help of Insight 3 -> The distribution is bimodal)

Results:

Mann-Whitney U statistic: 256306.50000

P-value (one-tailed): 0.00004

Reject H0: Winning matches have significantly lower average elixir cost than losing matches.

HT2:

Null Hypothesis (H0): The variable team_cardtype_variety is normally distributed.
Alternative Hypothesis (H1): The variable team_cardtype_variety is not normally distributed.

***: I had this hypothesis because of Insight 7, (the distribution seemed like normal at first.) But to choose whether I should use ANOVA or Kruskal-Wallis in my next hypothesis, as ANOVA assumes that distribution is normal while Kruskal-Wallis does not. I used Shapiro-Wilk to check if it was normal.

Results:

Shapiro-Wilk Test Statistic: 0.9027,

p-value: 0.0000

Data is likely NOT normally distributed (reject H0) -> So, I will use Kruskal-Wallis in my next hypothesis.

HT3:

Null Hypothesis (H0): There is no significant difference in win rates among decks with low, medium, and high card type variety.

Alternative Hypothesis (H1): At least one group (low, medium, or high card type variety) has a significantly different win rate compared to the others.

Kruskal-Wallis statistic: 14.1977,

p-value: 0.0008

Reject H0: Significant difference in win rates among groups.

HT4:

Null Hypothesis (H0): The combination of "tank" and "support" card types does not affect the win rate.

Alternative Hypothesis (H1): The combination of "tank" and "support" card types increases the win rate.

***: Via Insight 5, I thought that some card type combinations could influence the win rate so I constructed this hypothesis. I chose tank and support because in the heatmap it seemed like they had a great synergy.

Results:

Chi-square statistic: 7.0264,

p-value: 0.0080

Reject H0: There is an association between Tank+Support combo and match outcome.

***: I used Chi Square because I am using 2 categorical types. (has combo / don't have)(win/lose)

Machine Learning

Objective: The goal of this analysis is to build predictive models to estimate the outcome of Clash Royale matches (win or lose) based on various game-related features, including the cards used in the team and opponent decks, average elixir cost, card type distributions, and strategic combinations. By evaluating multiple classification algorithms, we aim to identify which model best predicts match results and to understand the impact of different cards and deck compositions on winning probability.

Feature Engineering:

What I did so far?

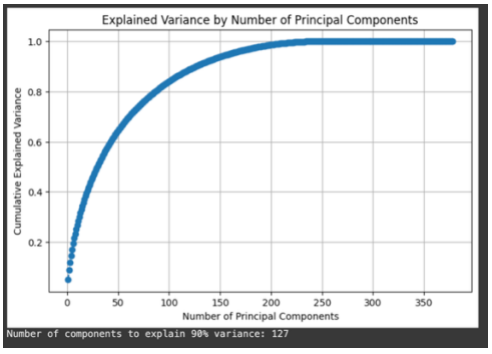
To improve model performance and address data complexity, several feature engineering steps were applied in the data preparation steps:

- Multilabel Binarizer was used to represent the presence of individual cards in both team and opponent decks (e.g., team_has_Zap, opponent_has_MegaMinion), converting categorical card data into machine-readable format.
- Card type frequencies were computed for each deck (e.g., team_n_air, team_n_spell) to capture strategic composition beyond individual cards. (Custom Frequency Encoding).
- Additional features like diff_avg_elixir and cardtype_variety were included to reflect deck balance and diversity.
- To manage the **curse of dimensionality** caused by hundreds of features, regularization methods and tree-based models are planned to reduce overfitting and improve generalization.

Handling Curse of Dimensionality:

Given the large number of features in the dataset, there is a risk of the **curse of dimensionality** as I wrote before, which can lead to overfitting and poor generalization in machine learning models. To mitigate this, I considered and applied the following strategies:

Dimensionality reduction: The Principal Component Analysis (PCA) method was applied to the very high-dimensional feature matrix in the model for dimension reduction. First, after scaling all features, enough components were selected with PCA to preserve 90% of the variance in the data. Thus, the original high-dimensional data was reduced to a lower-dimensional space with minimal information loss. This reduced feature matrix (X_reduced) was used to create more efficient and faster-working models. Thanks to PCA, both the computational cost was reduced and the risk of overfitting the model was reduced.



Model choice: Selecting models that are robust to high-dimensional data: Regularized logistic regression, Random Forests, and gradient boosting machines.

Regularized Logistic Regression

Without PCA but with L1 and L2:

```
L2 Regularized Logistic Regression
Accuracy: 0.5258620689655172
              precision    recall  f1-score   support

           0       0.36      0.22      0.27       187
           1       0.58      0.73      0.65       277

    accuracy                           0.53       464
   macro avg       0.47      0.48      0.46       464
weighted avg       0.49      0.53      0.50       464

L1 Regularized Logistic Regression
Accuracy: 0.5280172413793104
              precision    recall  f1-score   support

           0       0.36      0.22      0.27       187
           1       0.58      0.74      0.65       277

    accuracy                           0.53       464
   macro avg       0.47      0.48      0.46       464
weighted avg       0.49      0.53      0.50       464
```

With PCA:

```
Accuracy: 0.9870689655172413
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       187
           1       0.98      1.00      0.99       277

    accuracy                           0.99       464
   macro avg       0.99      0.98      0.99       464
weighted avg       0.99      0.99      0.99       464
```

***:My insight on why PCA really outperformed L1 and L2:

PCA reduces dimensionality by transforming the original features into a smaller set of uncorrelated principal components that capture most of the variance in the data. This transformation can help the model focus on the most informative aspects of the data while discarding noise and redundancy.

In contrast, L1 and L2 regularization work by shrinking or selecting features but do not explicitly address feature correlations or reduce dimensionality. When features are highly correlated like in my case, regularization struggles to identify the optimal subset(retaining redundant information) and leading to less efficient models.

Random Forests

Without PCA:

```
Random Forest Classifier Performance (Without PCA)
Accuracy: 0.5969827586206896
              precision    recall  f1-score   support

           0       0.50      0.33      0.39       187
           1       0.63      0.78      0.70       277

    accuracy                           0.60       464
   macro avg       0.57      0.55      0.55       464
weighted avg       0.58      0.60      0.58       464
```

With PCA:

```
Random Forest Classifier Performance
Accuracy: 0.8232758620689655
              precision    recall  f1-score   support

           0       0.87      0.66      0.75       187
           1       0.80      0.94      0.86       277

    accuracy                           0.82       464
   macro avg       0.84      0.80      0.81       464
weighted avg       0.83      0.82      0.82       464
```

Random Forest was chosen because it is a robust ensemble method that handles high-dimensional data well, reduces overfitting through averaging multiple decision trees, and provides feature importance for interpretability.

PCA is effective because it reduces dimensionality by capturing the most important variance in the data, simplifying the feature space while preserving key information.

Gradient Boosting Machines

Without PCA:

```
Gradient Boosting Classifier Performance (Without PCA)
Accuracy: 0.5905172413793104
              precision    recall  f1-score   support

           0       0.49      0.28      0.35       187
           1       0.62      0.80      0.70       277

    accuracy                           0.59       464
   macro avg       0.55      0.54      0.53       464
weighted avg       0.57      0.59      0.56       464
```

With PCA:

```
Gradient Boosting Classifier Performance (With PCA)
Accuracy: 0.5818965517241379
              precision    recall  f1-score   support

           0       0.47      0.26      0.33       187
           1       0.61      0.80      0.70       277

    accuracy                           0.58       464
   macro avg       0.54      0.53      0.51       464
weighted avg       0.55      0.58      0.55       464
```

***:My insight on why PCA might reduce performance in Gradient Boosting Classifier:

When applying PCA, the feature space is transformed into a lower-dimensional space composed of principal components, which are linear combinations of the original features.

This transformation can cause some information loss, especially information contained in lower-variance components that might still be important for classification. Since PCA focuses on maximizing variance preservation, it does not guarantee that the transformed components retain all the predictive signals relevant to the target.

In the case of GBC, the model can effectively utilize the original high-dimensional features, including subtle, non-linear interactions. However, after PCA, these non-linear and feature-specific patterns may be harder to capture because the principal components are linear combinations and may obscure the original structure.

Therefore, the reduction in performance after PCA for GBC could be due to:

- Loss of important predictive information in the discarded principal components**.**
- Inability of PCA to preserve non-linear feature interactions critical for GBC**.**
- The model's reliance on original feature space nuances which PCA smooths ou**t.**
- Potential suboptimal number of components selected**, causing underfitting.**

Conclusion

Regularized Logistic Regression with PCA is best!