final

April 22, 2025

1 League of Legends Analysis

Name: Daniel

Website Link: https://helkd27.github.io/league-of-legends-analysis/

[1]:
```python
import pandas as pd
import numpy as np
import re

import plotly.express as px
pd.options.plotting.backend = 'plotly'

from lec_utils import * # Feel free to uncomment and use this. It'll make your plotly graphs look like ours in lecture!
```

1.1 Step 1: Introduction

[2]:
```python
# Load the data
df = pd.read_csv('2022_LoL_esports_match_data_from_OraclesElixir.csv')
df.head()
```

[2]:
```
                   gameid datacompleteness  url league  …  deathsat25  \
0  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         1.0
1  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         2.0
2  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         0.0
3  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         2.0
4  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         2.0

   opp_killsat25  opp_assistsat25 opp_deathsat25
0            0.0              2.0            0.0
1            1.0              5.0            1.0
2            3.0              4.0            3.0
3            3.0              4.0            0.0
4            0.0              7.0            2.0

[5 rows x 161 columns]
```

1.2 Step 2: Data Cleaning and Exploratory Data Analysis

Selecting only the rows which contain team data and are complete

[3]:
```python
team = df[(df['position'] == 'team') & (df['datacompleteness'] == 'complete')]
team.head()
```

[3]:
```
                     gameid datacompleteness  url league  …  deathsat25  \
10  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         7.0
11  ESPORTSTMNT01_2690210         complete  NaN   LCKC  …         6.0
22  ESPORTSTMNT01_2690219         complete  NaN   LCKC  …         8.0
23  ESPORTSTMNT01_2690219         complete  NaN   LCKC  …         1.0
46  ESPORTSTMNT01_2690227         complete  NaN   LCKC  …         2.0

   opp_killsat25  opp_assistsat25 opp_deathsat25
10           7.0             22.0            6.0
11           6.0             12.0            7.0
22           8.0             13.0            1.0
23           1.0              1.0            8.0
46           2.0              1.0            5.0

[5 rows x 161 columns]
```

This step simply removes the beginning of the teamid and all games in which the team does not have an idea count as a singular team. For `firstmidtower` fill unknown values with 0, as if it unknown, it is most likely that neither team took the mid tower

There is also 310 teams with no teamid, so filled each team with an ID that I knew they didnt have, 1. Removing `oe:team:` from the header was a style choice to make future data columns cleaner to read

[4]:
```python
team['teamid'] = (
  team
  ['teamid']
  .fillna('oe:team:1')
  .apply(lambda x: x.replace('oe:team:', '') if isinstance(x, str) else x)
)
team['firstmidtower'].fillna(0, inplace=True)
```

[160]:
```python
total_cols = [
  'teamid', 'gameid', 'result',
  'firstblood', 'firstbaron', 'firstdragon', 'firstherald', 'firstmidtower',
  'firsttothreetowers', 'firsttower', 'goldat10', 'xpat10', 'csat10',
  'golddiffat10', 'golddiffat15', 'xpdiffat10', 'csdiffat10', 'killsat10',
  'assistsat10', 'deathsat10'
]

# print(team[total_cols].head().to_markdown(index=False)
team[total_cols].head()
```

[160]:
```
                              teamid                 gameid  result  firstblood
firstbaron  firstdragon  firstherald  firstmidtower  firsttothreetowers
firsttower  goldat10   xpat10  csat10  golddiffat10  golddiffat15  xpdiffat10
csdiffat10  killsat10  assistsat10  deathsat10
10  733ebb9dbf22a401c0127a0c80193ca  ESPORTSTMNT01_2690210       0         1.0
0.0          0.0          1.0            1.0                 1.0          1.0
16218.0  18213.0   322.0         1523.0          107.0        137.0         -8.0
3.0          5.0          0.0
11  7c64febcd5ccff13dcd035dc6867a00  ESPORTSTMNT01_2690210       1         0.0
0.0          1.0          0.0            0.0                 0.0          0.0
14695.0  18076.0   330.0        -1523.0         -107.0       -137.0          8.0
0.0          0.0          3.0
22  731b7a9fd004cdbe2bcb3da795bce47  ESPORTSTMNT01_2690219       0         0.0
0.0          0.0          1.0            0.0                 0.0          0.0
14939.0  17462.0   317.0        -1619.0        -1763.0      -1586.0        -27.0
1.0          1.0          3.0
23  e7a7c6bf58eb268ed3f13aac4158aa8  ESPORTSTMNT01_2690219       1         1.0
1.0          1.0          0.0            1.0                 1.0          1.0
16558.0  19048.0   344.0         1619.0         1763.0       1586.0         27.0
3.0          3.0          1.0
46  b9733b8e8aa341319bbaf1035198a28  ESPORTSTMNT01_2690227       1         0.0
1.0          1.0          0.0            1.0                 1.0          1.0
15466.0  19600.0   368.0         -103.0         1191.0        813.0         13.0
0.0          0.0          1.0
```

[163]:
```python
rotated = (
  team
  [['gameid', 'result', 'golddiffat10', 'golddiffat15'
    # , 'golddiffat20', 'golddiffat25'
    ]]
  [team['golddiffat10'].notna()]
  .set_index(['gameid', 'result'])
  .melt(ignore_index=False)
  .reset_index()
  .rename(
    columns={
      'gameid': 'gameid',
      'result': 'result',
      'variable': 'time',
      'value': 'golddiff'
    }
  )
  .assign(time=lambda df: df['time'].str.replace('golddiffat', '').astype(str))
  .assign(result = lambda df: df['result'].apply(lambda x: 'Win' if x == 1 else␣
 ↪'Loss'))
)
# print(rotated.head(5).to_markdown(index=False))
```

```
rotated.head()
```

[163]:

```
                 gameid result  time  golddiff
0  ESPORTSTMNT01_2690210   Loss    10    1523.0
1  ESPORTSTMNT01_2690210    Win    10   -1523.0
2  ESPORTSTMNT01_2690219   Loss    10   -1619.0
3  ESPORTSTMNT01_2690219    Win    10    1619.0
4  ESPORTSTMNT01_2690227    Win    10    -103.0
```

[164]:

```
# Filter data for positive gold difference at 10 minutes
positive_gold_diff = team[team['golddiffat10'] > 0]

# Calculate win rates
win_rate_data = positive_gold_diff['result'].value_counts()

# Create a pie plot
fig = px.pie(
  values=win_rate_data.values,
  names=['Win', 'Loss'],
  title='Win Rates with Positive Gold Difference at 10 Minutes',
  labels={'value': 'Percentage', 'names': 'Result'}
)
fig.write_html('assets/win_rate_pie_chart.html')
fig.show()
```

[165]:

```
positive_gold_diff = team[team['xpdiffat10'] > 0]

win_rate_data = positive_gold_diff['result'].value_counts(normalize=True) * 100

# Create a pie plot
fig = px.pie(
  values=win_rate_data.values,
  names=['Win', 'Loss'],
  title='Win Rates with Positive XP Difference at 10 Minutes',
  labels={'value': 'Percentage', 'names': 'Result'}
)
fig.show()
```

[167]:

```
golddiffplot = px.box(
  rotated,
  y='time',
  x='golddiff',
  color='result',
  title='Gold Difference at 10 and 15 Minutes',
  labels={
    'variable': 'Game Time',
    'value': 'Gold Difference',
```

```
    'result': 'Game Result',
    'time': 'Time (min)',
    'golddiff': 'Gold Difference'
  }
)
# width, height = golddiffplot.layout.width, golddiffplot.layout.height
# print(f"Plot size: {width}x{height} pixels")
# print(golddiffplot.layout)
golddiffplot.write_html('assets/golddiff.html', include_plotlyjs='cdn', )
# Get the size of the plot in pixels

golddiffplot.show()
```

[170]:

```
(
  team
  [['teamid', 'result', 'firstblood', 'firsttower', 'firstdragon',␣
↪'firstherald', 'firstbaron']]
  .groupby('teamid')
  .mean()
  .sort_values('result', ascending=False)
  .head()
  # .to_markdown(index=False)
)
```

[170]:

```
                                  result  firstblood  firsttower  firstdragon  \
firstherald  firstbaron
teamid
5f51496531ff55ed2b01327a33b81c7     1.0        0.67        0.33         0.67
0.67         1.0
b30ea1fcb1eafc1cf974fcc3988ae78     1.0        0.50        0.50         0.50
1.00         1.0
785bf7c68fff8e64a5b17ba5972f10a     1.0        0.50        0.50         0.25
0.50         1.0
3e3ca0895ded506fd637e723d6ae2a1     1.0        1.00        0.67         0.33
0.67         1.0
600b76b49b52ddab1ef5f9f9ca4a657     1.0        0.75        1.00         0.75
1.00         0.5
```

[171]:

```
pivot = (team.pivot_table(
  columns='firsttower',
  index='teamid',
  values='result',
  aggfunc='mean'
))
pivot.sort_values(by='teamid', ascending=False).head()
```

[171]:
```
firsttower                        0.0   1.0
teamid
ff07fcf769a41fa17ded4746368d6c7  0.50  0.76
fe59c993d0bda004e54eaecdd957f54  0.17  0.33
fe409cbd7c72eb621d9c4e7eac75936  0.32  0.67
fcec508e780bbd1ad493852640f5b36  0.33  0.67
fca935f82fd01de843aa2799eb575ea  0.21  0.27
```

[174]:
```python
# print(pivot.sort_values(by='teamid', ascending=False).head().
 ↪to_markdown(index=True))
(pivot.sort_values(by='teamid', ascending=False).head())
```

[174]:
```
firsttower                        0.0   1.0
teamid
ff07fcf769a41fa17ded4746368d6c7  0.50  0.76
fe59c993d0bda004e54eaecdd957f54  0.17  0.33
fe409cbd7c72eb621d9c4e7eac75936  0.32  0.67
fcec508e780bbd1ad493852640f5b36  0.33  0.67
fca935f82fd01de843aa2799eb575ea  0.21  0.27
```

1.3 Step 3: Framing a Prediction Problem

Looking at the primlimiary analysis from the above section, we can see that early game advantage shown by positive Gold and XP difference more often then not causes the team with these advantages to win.

Going back to the origional question stated in the introduction, we can solve this problem using a **binary classification model**. This model will be predicting the `result` column, indicating the whether the team won the game or not. The baseline model will be trained on all columns with distinct data points from 10 minutes into the game. These include: - `golddiffat10` - `xpdiffat10` - `csdiffat10` - `killsat10` - `deathsat10` - `assistsat10`

To evaluate the baseline and eventually the final model, both *F1-score* and *accuracy* will be used. The *F1-score* is particularly useful for handling imbalanced data, as it balances precision and recall, providing a more nuanced view of model performance. On the other hand, *accuracy* offers a straightforward measure of the proportion of correct predictions, which can still provide valuable insights when the dataset is not heavily imbalanced. By considering both metrics, we can gain a comprehensive understanding of the model's performance.

1.4 Step 4: Baseline Model

[74]:
```python
# Importing necessary libraries

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
# from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

[106]:

```python
# List of Columns for Baseline Model
cols = ['golddiffat10', 'xpdiffat10', 'csdiffat10', 'killsat10', 'deathsat10',␣
↪'assistsat10']
X = team[cols]
y = team['result']
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y)

# Create a pipeline with a scaler and a classifier
model = Pipeline([
  ('scaler', StandardScaler()),
  ('classifier', KNeighborsClassifier(n_neighbors=5))
])
```

[107]:

```python
from sklearn.metrics import accuracy_score

model.fit(x_train, y_train)
y_pred = model.predict(x_test)
# Calculate F1 score and accuracy for training data
y_train_pred = model.predict(x_train)
f1_train = f1_score(y_train, y_train_pred)
accuracy_train = accuracy_score(y_train, y_train_pred)

# Calculate F1 score and accuracy for test data
f1_test = f1_score(y_test, y_pred)
accuracy_test = accuracy_score(y_test, y_pred)

# Create a DataFrame to store the results
results_df = pd.DataFrame({
  'Dataset': ['Train', 'Test'],
  'F1 Score': [f1_train, f1_test],
  'Accuracy': [accuracy_train, accuracy_test]
})

print(results_df.to_markdown(index=False))
```

```
| Dataset   |   F1 Score |   Accuracy |
|:----------|-----------:|-----------:|
| Train     |   0.761578 |   0.760698 |
| Test      |   0.65492  |   0.654467 |
```

1.5 Step 5: Final Model

[ ]:
```
# List of Columns for Baseline Model
cols = ['golddiffat10', 'xpdiffat10', 'csdiffat10', 'killsat10', 'deathsat10',␣
 ↪'assistsat10',
        'firstblood', 'firsttower', 'firstdragon', 'firstherald', 'firstbaron',␣
 ↪'firstmidtower',
        'firsttothreetowers']

X = team[cols]
y = team['result']
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y)

knn_best = Pipeline([
  ('scaler', StandardScaler()),
  ('GridSearch', GridSearchCV(
    KNeighborsClassifier(),
    param_grid={
      'n_neighbors': range(5, 20),
    }
  ))
])
forest_best = Pipeline([
  ('scaler', StandardScaler()),
  ('GridSearch', GridSearchCV(
    RandomForestClassifier(),
    param_grid={
      # 'n_estimators': [10, 50, 100],
      'max_depth': range(1, 10),
    }
  ))
])
```

[136]:
```
knn_best.fit(x_train, y_train)
forest_best.fit(x_train, y_train)
```

[136]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('GridSearch',
                 GridSearchCV(estimator=RandomForestClassifier(),
                              param_grid={'max_depth': range(1, 10)}))])
```

[176]:
```
forest_y_pred = forest_best.predict(x_test)
knn_y_pred = knn_best.predict(x_test)

print(f"F1 Score for Randomforest: {f1_score(y_test, y_pred)}")
print(f"Accuracy for Randomforest: {accuracy_score(y_test, forest_y_pred)}")
```

```
print(f"F1 Score for KNN: {f1_score(y_test, y_pred)}")
print(f"Accuracy for KNN: {accuracy_score(y_test, knn_y_pred)}")
# Create a DataFrame to store the results
results_df = pd.DataFrame({
  'Model': ['Random Forest', 'KNN'],
  'F1 Score': [f1_score(y_test, forest_y_pred), f1_score(y_test, knn_y_pred)],
  'Accuracy': [accuracy_score(y_test, forest_y_pred), accuracy_score(y_test,␣
 ↪knn_y_pred)]
})
print(results_df.to_markdown(index=False))
```

```
F1 Score for Randomforest: 0.8406232400976159
Accuracy for Randomforest: 0.8477852852852853
F1 Score for KNN: 0.8406232400976159
Accuracy for KNN: 0.8406531531531531
| Model          |   F1 Score |   Accuracy |
|:---------------|-----------:|-----------:|
| Random Forest  |   0.847125 |   0.847785 |
| KNN            |   0.840623 |   0.840653 |
```

[198]:
```
# pipeline.named_steps['classifier']
importance = forest_best.named_steps['GridSearch'].best_estimator_.
 ↪feature_importances_

df_importance = pd.DataFrame({
  'feature': cols,
  'importance': importance
}).sort_values(by='importance', ascending=False)

# print(df_importance.head().to_markdown(index=False))
df_importance.head()
```

[198]:

	feature	importance
10	firstbaron	0.46
12	firsttothreetowers	0.17
11	firstmidtower	0.11
0	golddiffat10	0.07
1	xpdiffat10	0.06

As we can see from the importance feature dataframe from above we can see that the most important indicator to winning the game is getting the first baron