League of League of Legends Role Prediction from Post-Game Data

Name(s): Brandon Dioneda, Samuel Zhang

Website Link: https://apm-denizens.github.io/dsc80-project5/

Code

In [ ]:
```
import os
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import QuantileTransformer
from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier

import plotly.express as px
import plotly.graph_objs as go

from IPython.display import display

import black
```

Framing the Problem

Predict which role (top-lane, jungle, support, etc.) a player played given their post-game data.

In [ ]:
```
# Load Dataset
league_fp = os.path.join(
    'data',
    '2022_LoL_esports_match_data_from_OraclesElixir.csv'
)
league_raw = pd.read_csv(league_fp)

league_raw.dtypes
```

```
/Users/elim-mbp-01/.pyenv/versions/3.8.16/envs/dsc80_39/lib/python3.8/site
-packages/IPython/core/interactiveshell.py:3505: DtypeWarning: Columns (2)
have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[ ]:
```
gameid               object
datacompleteness     object
url                  object
league               object
year                  int64
                     ...
assistsat15        float64
deathsat15         float64
opp_killsat15      float64
opp_assistsat15    float64
opp_deathsat15     float64
Length: 123, dtype: object
```

In [ ]:
```python
# ======================
# CLEANING
# ======================

league = league_raw.copy()

# ========================================================
# Some rows are marked with datacompleteness = "partial"/"ignore".
# According to Costin, this label just means that there's a NaN
# value in the row (excluding NaN's from team/player related columns)
# Will not ignore.
# league = league[league["datacompleteness"] == "complete"]
# ========================================================

# ========================================================
# Each 'gameid' corresponds to up to 12 rows — one for each of the 5
# players on both teams and 2 containing summary data for the two teams
# (try to find out what distinguishes those rows). After selecting your
# line of inquiry, make sure to remove either the player rows or the
# team rows so as not to have issues later in your analysis.

# 24900 rows without a champion
# (league_raw["champion"].isna()).sum()
# 149400 rows total. 12 rows per match. 149400 / 12 = 12450 matches.
# 12450 * 2 = 24900 team related rows
# (149400 / 12) = 12450 matches. 12450 * 2 = 24900
# Filtering by whether there's a champion listed or not, should get
# only the player related rows
league = league[league["champion"].notna()]

# Also drop the team related columns. Every single value in those
# columns should be null for the player related rows
team_columns_mask = league.isnull().sum(axis=0) == league.shape[0]
league = league.drop(columns=league.columns[team_columns_mask])
print(f"{team_columns_mask.sum()} team-related columns removed")
# ========================================================

# ========================================================
# NaN values for players are represented as "unkown player"
# Mark these as NaN
league["playername"] = league["playername"].apply(
    lambda val: np.nan if val == "unknown player" else val
)
# ========================================================


# ========================================================
```

```python
# Many columns should be of type bool but are not.
# Checks each column to see if the set of unique values is {0, 1}
# If so, converts the column to bool
columns_to_bool = []
for column in league.columns:
    value_counts = league[column].value_counts()
    if set(value_counts.index) == {0, 1}:
        columns_to_bool.append(column)
        league[column] = league[column].astype(bool)
print(f"CONVERTED {len(columns_to_bool)} COLUMNS TO BOOL")
# ============================================================

# ============================================================
league = league.convert_dtypes()
league.info()
# ============================================================
```

```
26 team-related columns removed
CONVERTED 6 COLUMNS TO BOOL
<class 'pandas.core.frame.DataFrame'>
Int64Index: 124500 entries, 0 to 149397
Data columns (total 97 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   gameid                     124500 non-null  string
 1   datacompleteness           124500 non-null  string
 2   url                        18680 non-null   string
 3   league                     124500 non-null  string
 4   year                       124500 non-null  Int64
 5   split                      90070 non-null   string
 6   playoffs                   124500 non-null  boolean
 7   date                       124500 non-null  string
 8   game                       124500 non-null  Int64
 9   patch                      124410 non-null  Float64
 10  participantid              124500 non-null  Int64
 11  side                       124500 non-null  string
 12  position                   124500 non-null  string
 13  playername                 122334 non-null  string
 14  playerid                   122331 non-null  string
 15  teamname                   124455 non-null  string
 16  teamid                     122730 non-null  string
 17  champion                   124500 non-null  string
 18  ban1                       122555 non-null  string
 19  ban2                       122665 non-null  string
 20  ban3                       122400 non-null  string
 21  ban4                       122510 non-null  string
 22  ban5                       122240 non-null  string
 23  gamelength                 124500 non-null  Int64
 24  result                     124500 non-null  boolean
 25  kills                      124500 non-null  Int64
 26  deaths                     124500 non-null  Int64
 27  assists                    124500 non-null  Int64
 28  teamkills                  124500 non-null  Int64
 29  teamdeaths                 124500 non-null  Int64
 30  doublekills                106310 non-null  Int64
 31  triplekills                106310 non-null  Int64
 32  quadrakills                106310 non-null  Int64
 33  pentakills                 106310 non-null  Int64
 34  firstblood                 124500 non-null  boolean
 35  firstbloodkill             124500 non-null  boolean
 36  firstbloodassist           124500 non-null  boolean
 37  firstbloodvictim           124500 non-null  boolean
 38  team kpm                   124500 non-null  Float64
 39  ckpm                       124500 non-null  Float64
 40  barons                     102600 non-null  Int64
 41  opp_barons                 102600 non-null  Int64
 42  inhibitors                 103040 non-null  Int64
 43  opp_inhibitors             103040 non-null  Int64
 44  damagetochampions          124490 non-null  Int64
 45  dpm                        124490 non-null  Float64
 46  damageshare                124490 non-null  Float64
 47  damagetakenperminute       124490 non-null  Float64
 48  damagemitigatedperminute   106310 non-null  Float64
 49  wardsplaced                124490 non-null  Int64
 50  wpm                        124490 non-null  Float64
 51  wardskilled                124490 non-null  Int64
 52  wcpm                       124490 non-null  Float64
```

```
 53  controlwardsbought        124490 non-null  Int64
 54  visionscore               124490 non-null  Int64
 55  vspm                      124490 non-null  Float64
 56  totalgold                 124500 non-null  Int64
 57  earnedgold                124500 non-null  Int64
 58  earned gpm                124500 non-null  Float64
 59  earnedgoldshare           124500 non-null  Float64
 60  goldspent                 124490 non-null  Int64
 61  total cs                  124500 non-null  Int64
 62  minionkills               124490 non-null  Int64
 63  monsterkills              124490 non-null  Int64
 64  monsterkillsownjungle     18620 non-null   Int64
 65  monsterkillsenemyjungle   18620 non-null   Int64
 66  cspm                      124500 non-null  Float64
 67  goldat10                  106310 non-null  Int64
 68  xpat10                    106310 non-null  Int64
 69  csat10                    106310 non-null  Int64
 70  opp_goldat10              106310 non-null  Int64
 71  opp_xpat10                106310 non-null  Int64
 72  opp_csat10                106310 non-null  Int64
 73  golddiffat10              106310 non-null  Int64
 74  xpdiffat10                106310 non-null  Int64
 75  csdiffat10                106310 non-null  Int64
 76  killsat10                 106310 non-null  Int64
 77  assistsat10               106310 non-null  Int64
 78  deathsat10                106310 non-null  Int64
 79  opp_killsat10             106310 non-null  Int64
 80  opp_assistsat10           106310 non-null  Int64
 81  opp_deathsat10            106310 non-null  Int64
 82  goldat15                  106310 non-null  Int64
 83  xpat15                    106310 non-null  Int64
 84  csat15                    106310 non-null  Int64
 85  opp_goldat15              106310 non-null  Int64
 86  opp_xpat15                106310 non-null  Int64
 87  opp_csat15                106310 non-null  Int64
 88  golddiffat15              106310 non-null  Int64
 89  xpdiffat15                106310 non-null  Int64
 90  csdiffat15                106310 non-null  Int64
 91  killsat15                 106310 non-null  Int64
 92  assistsat15               106310 non-null  Int64
 93  deathsat15                106310 non-null  Int64
 94  opp_killsat15             106310 non-null  Int64
 95  opp_assistsat15           106310 non-null  Int64
 96  opp_deathsat15            106310 non-null  Int64
dtypes: Float64(13), Int64(60), boolean(6), string(18)
memory usage: 97.5 MB
```

In [ ]:
```
league.head()
```

Out[ ]:

	gameid	datacompleteness	url	league	year	split	playo
0	ESPORTSTMNT01_2690210	complete	<NA>	LCK CL	2022	Spring	Fa
1	ESPORTSTMNT01_2690210	complete	<NA>	LCK CL	2022	Spring	Fa
2	ESPORTSTMNT01_2690210	complete	<NA>	LCK CL	2022	Spring	Fa
3	ESPORTSTMNT01_2690210	complete	<NA>	LCK CL	2022	Spring	Fa
4	ESPORTSTMNT01_2690210	complete	<NA>	LCK CL	2022	Spring	Fa

5 rows × 97 columns

Generating Train/Val/Test Splits

In [ ]:
```
# Note that class labels are uniform.
league["position"].value_counts()
```

Out[ ]:
```
top    24900
jng    24900
mid    24900
bot    24900
sup    24900
Name: position, dtype: Int64
```

In [ ]:
```
league.shape
```

Out[ ]:
```
(124500, 97)
```

In [ ]:
```
X = league.drop(columns=["position"])
y = league["position"]

X_2, X_test, y_2, y_test = train_test_split(
    X, y,
    test_size=0.2, random_state=16
)
X_train, X_val, y_train, y_val = train_test_split(
    X_2, y_2,
    test_size=0.25, random_state=16
)
# 0.8 * 0.25 = 0.2
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(74700, 96)
```

Baseline Model

In [ ]:
```
league["kills"].isna().sum(), league["deaths"].isna().sum()
```

Out[ ]:
```
(0, 0)
```

In [ ]:
```
((league["kills"] / (league["deaths"]+1)) == float("inf")).sum()
```

Out[ ]:
```
0
```

In [ ]:
```
preproc_base = ColumnTransformer(
    transformers=[
        (
            "champion",
            OneHotEncoder(handle_unknown="ignore"),
            ["champion"]
        ),
        (
            "kill-death ratio",
            FunctionTransformer(
                lambda df: (df["kills"] / (df["deaths"] + 1))
                .astype("float32")
                .to_frame()
            ),
            ["kills", "deaths"],
        ),
    ],
    remainder="drop",
)

league_pl_base = Pipeline(
    [
        ("preprocessor", preproc_base),
        ("random-forest", RandomForestClassifier(random_state=16)),
    ]
)
```

In [ ]:
```
# EVALUATION OF BASELINE MODEL
league_pl_base.fit(X_train, y_train)
print(
    league_pl_base.score(X_train, y_train),
    league_pl_base.score(X_val, y_val)
)
```

Out[ ]:
```
(0.9421552878179384, 0.930843373493976)
```

In [ ]:
```
pd.Series(league_pl_base.predict(X_train)).value_counts()
```

Out[ ]:
```
bot    15346
top    15135
sup    14938
mid    14666
jng    14615
dtype: int64
```

In [ ]:
```
y_train.value_counts()
```

Out[ ]:
```
jng    15023
sup    15019
top    14971
bot    14889
mid    14798
Name: position, dtype: Int64
```

Final Model

In [ ]:
```
# GRIDSEARCH

hyperparameters = {
    'random-forestn_estimators': [25, 50, 100, 150],
    'random-forestcriterion': ['gini', 'entropy'],
    'random-forestmax_depth' : [5, 10, 15, None],
    'random-forestmax_features': ['sqrt', 'log2']
}

grid_search = GridSearchCV(league_pl_base, hyperparameters, cv=5)
grid_search.fit(X_train, y_train)
# print(grid_search.best_params_)

# BEST PARAMS:
# {
#     'random-forestn_estimators': 150
#     'random-forestcriterion': 'gini',
#     'random-forestmax_depth': None,
#     'random-forestmax_features': 'log2',
# }
```

In [ ]:
```
preproc_final = ColumnTransformer(
    transformers=[
        (
            "champion",
            OneHotEncoder(handle_unknown="ignore"),
            ["champion"]
        ),
        (
            "kill-death ratio",
            FunctionTransformer(
                lambda df: (df["kills"] / df["deaths"])
                .astype("float32")
                .replace(np.inf, np.NaN)
                .fillna(0)
                .to_frame()
            ),
            ["kills", "deaths"],
        ),
        (
            "standardization",
            Pipeline(
                [
                    (
                        "impute",
                        SimpleImputer(strategy="constant",
                                      fill_value=0)
                    ),
                    ("scale", StandardScaler()),
```

```
                ]
            ),
            [
                "cspm",
                "earnedgold",
                "earned gpm",
            ],
        ),
        (
            "quantile-transform",
            Pipeline(
                [
                    (
                        "impute",
                        SimpleImputer(strategy="constant",
                                      fill_value=0)
                    ),
                    (
                        "quantile",
                        QuantileTransformer(
                            n_quantiles=100,
                            output_distribution="normal"
                        ),
                    ),
                ]
            ),
            ["visionscore", "vspm", "wardsplaced", "wpm",],
        ),
    ],
    remainder="drop",
)

league_pl_final = Pipeline(
    [
        ("preprocessor", preproc_final),
        (
            "random-forest",
            RandomForestClassifier(
                # BEST COMBINATION OF HYPERPARAMETERS
                # ACCORDING TO GRIDSEARCH
                random_state=16,
                criterion="gini",
                max_depth=None,
                max_features="log2",
                n_estimators=150,
            ),
        ),
    ]
)
```

In [ ]:

```
hist_trace = go.Histogram(
    x=league["visionscore"],
    nbinsx=25
)
layout = go.Layout(
    title="Visionscore Histogram",
    xaxis_title="Visionscore",
    yaxis_title="Frequency"
)
fig = go.Figure(data=[hist_trace], layout=layout)
```

```
# fig.write_html("visionscore_histogram.html")
fig.show()
```

In [ ]:
```
hist_trace = go.Histogram(
    x=league["earnedgold"],
    nbinsx=25
)
layout = go.Layout(
    title="Earned Gold Histogram",
    xaxis_title="Earned Gold",
    yaxis_title="Frequency"
)
fig = go.Figure(data=[hist_trace], layout=layout)
# fig.write_html("earnedgold_histogram.html")
fig.show()
```

In [ ]:
```
for colname in [
                "visionscore",
                "earned gpm",
                "cspm",
                "vspm",
                "earnedgold",
                "wardsplaced",
                "wpm",
            ]:
    print(f"{colname}: {league[colname].isna().sum()}")
```

```
visionscore: 10
earned gpm: 0
cspm: 0
vspm: 10
earnedgold: 0
wardsplaced: 10
wpm: 10
```

In [ ]:
```
# EVALUATION OF FINAL MODEL TRAIN & VALIDATION SETS
league_pl_final.fit(X_train, y_train)
print(
    league_pl_final.score(X_train, y_train),
    league_pl_final.score(X_val, y_val)
)
```

Out[ ]:
```
(1.0, 0.9572690763052208)
```

In [ ]:
```
# EVALUATION OF FINAL MODEL TEST SET
league_pl_final.score(X_test, y_test)
```

Out[ ]:
```
0.9568674698795181
```

Fairness Analysis

Null Hypothesis: Our model is fair. It's accuracy for the Red & Blue sides are roughly the same, and any differences are due to random chance.
Alt Hypothesis: Our model is unfair. It's accuracy for the Blue team is higher.

In [ ]:
```
blue_wr = league.loc[league["side"] == "Blue", "result"].mean()
red_wr = league.loc[league["side"] == "Red", "result"].mean()
```

```
print(blue_wr, red_wr, blue_wr + red_wr)
```

```
0.5242570281124498 0.47558232931726907 0.9998393574297189
```

In [ ]:
```
def get_test_stat(league_df: pd.DataFrame):
    league_df_red = league_df.loc[league_df["side"] == "Red"]
    red_score = league_pl_final.score(
        league_df_red.drop(columns=["position"]),
        league_df_red["position"]
    )

    league_df_blue = league_df.loc[league_df["side"] == "Blue"]
    blue_score = league_pl_final.score(
        league_df_blue.drop(columns=["position"]),
        league_df_blue["position"]
    )

    return blue_score - red_score

observed_test_stat = get_test_stat(league)
observed_test_stat
```

Out[ ]:
```
0.0008353413654619279
```

In [ ]:
```
league_cp = league.copy()

test_stats = []
for _ in range(100):
    league_cp["side"] = np.random.permutation(league_cp["side"])
    test_stats.append(get_test_stat(league_cp))
```

In [ ]:
```
test_stats_arr = np.array(test_stats)
```

In [ ]:
```
(test_stats_arr >= observed_test_stat).mean()
```

Out[ ]:
```
0.13
```

In [ ]:
```
hist_trace = go.Histogram(x=test_stats, nbinsx=10)
layout = go.Layout(
    title="Histogram of Test Statistics",
    xaxis_title="Test Statistic (Difference in Accuracy)",
    yaxis_title="Frequency"
)

fig = go.Figure(data=[hist_trace], layout=layout)
fig.add_vline(
    x=observed_test_stat,
    line_width=2,
    line_dash="solid",
    line_color="red"
)

fig.write_html("teststats_histogram.html")
fig.show()
```