League of Legends Competitive Match Data

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Predict if a team will win or lose a game.
  - Predict which role (top-lane, jungle, support, etc.) a player played given their post-game data.
  - Predict how long a game will take before it happens.
  - Predict which team will get the first Baron.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

Summary of Findings

Introduction

Many people loves betting online which team will win in a LOL mathcing. In this project, we will predict a team's result in a match given early game datas. It is a classification problem, since the result is either win or lose. And we will pick 'teamname', 'firstblood', 'firstdragon', 'firstherald', 'golddiffat15', 'xpdiffat15', 'csdiffat15', 'killsat15', 'assistsat15' and 'deathsat15' as features.

Baseline Model

In baseline model we have Nomimal feature:'teamname' Quantitative features:'firstblood', 'firstdragon', 'firstherald', 'golddiffat15', 'xpdiffat15', 'csdiffat15', 'killsat15', 'assistsat15' and 'deathsat15' We leave all quantitavie features as 'is', and do Onehotcoder transform on 'teamname' The accuracy of our model on test set is 67.56%, which is better than just randomly pick win or lose(50% accuracy), but still not good enough.

Final Model

In final Model, I apply more transformer. -I apply StdScalerByGroup() on 'golddiffat15' groupby 'league', since different league has different style of playing this game. -I transform boolean variable 'False' into -1, so now losing 'firstblood' and losing a resource in jungle will have negative effect. -I apply binarizer() on 'csdiffat15' with threshold 15; in another word, if the 'cs' difference is over the threhold now means they have lane advantage.

The accuracy now increase to 74.12%.

Fairness Evaluation

Null Hypothesis: My model is fair;the precision for 'LCK' league is same as other leagues. Alternative hypothesis: My model is unfair; the precision for 'LCK' league is different than other leagues.

Result: The p-value is 0.225, so I can't reject my null hypothesis

Code

In [ ]:

```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import itertools
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import Binarizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.base import BaseEstimator, TransformerMixin

class StdScalerByGroup(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self, X, y=None):
        """
        :Example:
        >>> cols = {'g': ['A', 'A', 'B', 'B'], 'c1': [1, 2, 2, 2], 'c2': [3, 1, 2, 0]}
        >>> X = pd.DataFrame(cols)
        >>> std = StdScalerByGroup().fit(X)
        >>> std.grps_ is not None
        True
        """
        # X might not be a pandas DataFrame (e.g. a np.array)
        df = pd.DataFrame(X)

        # Compute and store the means/standard-deviations for each column (e.g. 'c1' and 'c2'),
        # for each group (e.g. 'A', 'B', 'C').
        # (Our solution uses a dictionary)
        group = df.columns[0]
        self.grps_ = df.groupby(group).agg(['mean', 'std']).to_dict()

        return self

    def transform(self, X, y=None):
        """
        :Example:
        >>> cols = {'g': ['A', 'A', 'B', 'B'], 'c1': [1, 2, 3, 4], 'c2': [1, 2, 3, 4]}
        >>> X = pd.DataFrame(cols)
        >>> std = StdScalerByGroup().fit(X)
        >>> out = std.transform(X)
        >>> out.shape == (4, 2)
        True
        >>> np.isclose(out.abs(), 0.707107, atol=0.001).all().all()
        True
        """

        try:
            getattr(self, "grps_")
        except AttributeError:
            raise RuntimeError(
                "You must fit the transformer before tranforming the data!")

        # Hint: Define a helper function here!

        df = pd.DataFrame(X)
        def helper(x, col):
            return (x[0]-self.grps_[(col, 'mean')][x[1]])/self.grps_[(col, 'std')][x[1]]

        group = df.columns[0]
        f = df.columns[1:]

        for col in f:
            df[col] = list(zip(df[col], df[group]))
            df[col] = df[col].apply(lambda x: helper(x, col))

        return df[f]
```

In [ ]:

```
league = pd.read_csv(os.path.join('data','2022_LoL_esports_match_data_from_OraclesElixir_20221207.csv'))  #read dateset


# data cleaning
def clean_league(league):
    df = league.copy()
    df['datacompleteness'] = df['datacompleteness'].apply(
        lambda x: True if x == 'complete' else False)
    df[['playerid', 'teamid']].astype(
        str).applymap(lambda x: x.split(':')[-1])  # convert id into 31 digits string
    return df


# takes in dataframe like league_cleaned and return two dataframe describe teams and players seperately
def seperate_team_player(league_cleaned):
    return league_cleaned[league_cleaned['position'] != 'team'].reset_index(drop=True), league_cleaned[league_cleaned['position']
     == 'team'].reset_index(drop=True)

team_rows = seperate_team_player(clean_league(league))[1] # focus on team data of matches
team_rows.head()

features_and_result_list = [
    'league', 'teamname', 'firstblood', 'firstdragon', 'firstherald', 'golddiffat15', 'xpdiffat15', 'csdiffat15', 'killsat15', '

df = (
    team_rows[features_and_result_list].dropna().reset_index(drop=True)
)

df.head()
```

```
c:\Users\gl121\anaconda3\envs\dsc80\lib\site-packages\IPython\core\interactiveshell.py:3398: DtypeWarning: Columns (2) have mixe
d types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[ ]:

	league	teamname	firstblood	firstdragon	firstherald	golddiffat15	xpdiffat15	csdiffat15	killsat15	assistsat15	deathsat15	result
0	LCK CL	Fredit BRION Challengers	1.0	0.0	1.0	107.0	-1617.0	-23.0	5.0	10.0	6.0	0
1	LCK CL	Nongshim RedForce Challengers	0.0	1.0	0.0	-107.0	1617.0	23.0	6.0	18.0	5.0	1
2	LCK CL	T1 Challengers	0.0	0.0	1.0	-1763.0	-906.0	-22.0	1.0	1.0	3.0	0
3	LCK CL	Liiv SANDBOX Challengers	1.0	1.0	0.0	1763.0	906.0	22.0	3.0	3.0	1.0	1
4	LCK CL	KT Rolster Challengers	0.0	1.0	0.0	1191.0	2298.0	15.0	3.0	8.0	1.0	1

Baseline Model

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(df[['teamname', 'firstblood', 'firstdragon', 'golddiffat15', 'xpdiffat15', '
                                                    df['result'],
                                                    random_state=1,test_size=0.1)
#90% train set and 10% test set


preproc = ColumnTransformer(
    transformers=[
        ('keep', FunctionTransformer(lambda x:x), [
         'firstblood', 'firstdragon', 'golddiffat15', 'xpdiffat15', 'csdiffat15', 'killsat15', 'assistsat15', 'deathsat15']),
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['teamname'])
    ]
)

pl = Pipeline([
    ('preprocessor', preproc),
    ('DecisionTreeClassifier', DecisionTreeClassifier())
])

pl.fit(x_train,y_train)
y_pred = pl.predict(x_test)
```

```
print('r^2 on train:',pl.score(x_train,y_train))
print('accuracy on test:',(y_pred == y_test).mean())
```

```
r^2 on train: 1.0
accuracy on test: 0.6756247053276756
```

Final Model

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(df[['league','teamname', 'firstblood', 'firstdragon','golddiffat15', 'xpdiffa
                                                    df['result'],
                                                    random_state=1, test_size=0.1)

preproc = ColumnTransformer(  #now apply more transformer on features
    transformers=[
        ('keep', FunctionTransformer(lambda x:x), ['xpdiffat15', 'killsat15', 'assistsat15', 'deathsat15']),
        ('zero_to_negative', FunctionTransformer(lambda x: 2*x-1), ['firstblood', 'firstdragon']),
        ('StdScalerByGroup', StdScalerByGroup(), ['league', 'golddiffat15']),
        ('Binarizer',Binarizer(threshold=15),['csdiffat15']),
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['teamname'])
    ]
)

hyperparameters = {   #values we will try in gridsearch
    'DecisionTreeClassifier__max_depth': [2,4,6,8,10,15,20],
    'DecisionTreeClassifier__min_samples_split': [2, 4, 6, 8, 10, 15, 20],
}

pl = Pipeline([
    ('preprocessor', preproc),
    ('DecisionTreeClassifier', DecisionTreeClassifier())
])

searcher = GridSearchCV(pl, hyperparameters, cv=5)

searcher.fit(x_train, y_train)
y_pred = searcher.predict(x_test)

(y_pred == y_test).mean()  #accuracy
```

Out[ ]:

```
0.7411598302687411
```

Fairness Evaluation

In [ ]:

```
fairness_df = x_test[['league']].copy()
fairness_df['actual'] = y_test
fairness_df['predict'] = y_pred
fairness_df = fairness_df.reset_index(drop=True)
fairness_df.head()
```

Out[ ]:

	league	actual	predict
0	LCO	1	1
1	SL	1	1
2	CT	0	0
3	VCS	0	1
4	LCK CL	0	0

In [ ]:

```
lck = fairness_df[fairness_df['league']=='LCK']
non_lck = fairness_df[fairness_df['league'] !='lck']
lck_precision = metrics.precision_score(lck['actual'], lck['predict'])
non_lck_precision = metrics.precision_score(non_lck['actual'], non_lck['predict'])
obs_diff = abs(lck_precision-non_lck_precision)

n=1000
result=[]
for _ in range(n):
    df = fairness_df.copy()
    df['league'] = df['league'].sample(frac=1).reset_index(drop=True)

    lck = fairness_df[df['league']=='LCK']
    non_lck = fairness_df[df['league'] !='lck']

    lck_precision = metrics.precision_score(lck['actual'], lck['predict'])
    non_lck_precision = metrics.precision_score(non_lck['actual'], non_lck['predict'])

    result.append(abs(lck_precision-non_lck_precision))

(result>=obs_diff).mean() #p-value
```

Out[ ]:

```
0.224
```