Summary of Findings

Introduction

We will be predicting which team will win or lose a game based on stats at 15 and 10 minutes into the game, hence a classifier problem. The target variable we will be using is the 'result' column. Because every game has 12 rows attached to it, we can simply remove all rows but the team-wide data rows pertaining to each game.Because the each team-row has data for both the side(blue or red) the team is on and also the enemy side (or the data is simply a difference in a stat betweenthe two teams, such as xp diff at 10), we can simply pick a side we want to focus on; here we chose blue. The result column therefore represents a blue win with a 1, and a red win with a 0.

First, we clean the data. We remove all rows with the value datacompleteness as incomplete.

Baseline Model

The baseline model has one ordinal feature, 'patch', one nominal feature, 'league', and 18 quant feats regarding various stats at 10 and 15 minutes. Turretplates is included because turretplates fall at 14 minutes, therefore only being relevant in the first 15 minutes of the game. The R-squared, or accuracy is around 66%. This is not bad, as a lot of things can change between the game ending and the 15 minute mark of a game, such as champion scaling differences or throws of a lead.

Final Model

Here, we added a few features: we included the team name as nominal feature, as certain teams may be better at abusing leads in the earlygame compared to other teams. We also included the gametime, binarized between long and short with a threshold of 22 minutes. The shorter a game is, the more relevant the earlygame lead, which could therefore impact the relevance of the data shown. We also standardized the quantitative data by the groups of 'league', the league they play in. Different leagues have different playstyles and metas locally, which can impact the power of an earlygame lead. The model I chose is a decisiontreeclassifier. After running GridSearchCV, I discovered that max depth and min sample leaves of (2,2) gave the best results. With these values, we get a score of 73.73%, which much improved compared to the base model.

Fairness Evaluation

The interesting subset I used was game time. We defined a short game as less than 25 minutes, and a long game as longer than 25 minutes. Null: Long and Short game times will have similar acurracies, the model is fair. Alternative: Short games will have a higher accuracy than long games the model is unfair. Using the difference in accuracy as the test statistic, we ran 1000 trials, resulting in a pvalue of 0.066. With a significance level of 0.05, we fail to reject the null hypothesis.

Code

In [806…

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
import re
from lab import StdScalerByGroup
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import GridSearchCV

%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
league = pd.read_csv('2022_LoL.csv')
league.head()
```

```
C:\Users\dskon\anaconda3\envs\dsc80\lib\site-packages\IPython\core\interactiveshel
l.py:3433: DtypeWarning: Columns (2) have mixed types.Specify dtype option on impor
t or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[806]:

	gameid	datacompleteness	url	league	year	split	playoffs	date
0	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08
1	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08
2	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08
3	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08
4	ESPORTSTMNT01_2690210	complete	NaN	LCK CL	2022	Spring	0	2022-01-10 07:44:08

5 rows × 123 columns

In [962…

```
#dropping extraneous/unneeded columns
cleaned = league[league['datacompleteness'] == 'complete'] #only complete games
leaguecols = ' '.join(cleaned.columns)
cols = ['gameid','gamelength', 'teamname', 'league', 'patch', 'result', 'participan
cols = cols + re.findall(r'\w+at\d{2}', leaguecols) #get only relevant and realisti
cleaned = cleaned[cols]
games = cleaned[(cleaned['participantid'] == (100))] #choose one side
games = games.dropna().drop('gameid',axis = 1)
games
```

Out[962]:

	gamelength	teamname	league	patch	result	participantid	turretplates	opp_turretplat
10	1713	Fredit BRION Challengers	LCK CL	12.01	0	100	5.0	0
22	2114	T1 Challengers	LCK CL	12.01	0	100	2.0	3
46	1972	KT Rolster Challengers	LCK CL	12.01	1	100	1.0	4
70	2488	DWG KIA Challengers	LCK CL	12.01	0	100	4.0	0
94	2020	Kwangdong Freecs Challengers	LCK CL	12.01	1	100	6.0	4
...	...	...	...	...	...	...	...	
146278	1896	Córdoba Patrimonio eSports	NEXO	12.20	0	100	3.0	6
146290	1479	TDC Esports	NEXO	12.20	0	100	1.0	10
146302	1715	unknown team	LPLOL	12.21	1	100	2.0	7
146314	2155	unknown team	LPLOL	12.21	0	100	4.0	6
146326	2183	GTZ Esports	LPLOL	12.21	0	100	7.0	2

10400 rows × 38 columns

In [103…

```
```

Baseline Model

In [106…

```
ordinal_feats = ['patch']
nominal_feats = ['league']
quant_feats = ['golddiffat10', 'xpdiffat10', 'csdiffat10', 'killsat10', 'assistsat1
 'opp_deathsat10', 'golddiffat15','xpdiffat15', 'csdiffat15','killsat15','assistsat
```

In [103…

```
pl = Pipeline([
    ('preproc', ColumnTransformer(
        transformers = [
            ('quant', SimpleImputer(), quant_feats), #impute missing nrs.
            ('ohe', OneHotEncoder(), nominal_feats),
            ('ord', OrdinalEncoder(), ordinal_feats),
            ]
    )),
    ('dt', DecisionTreeClassifier())
])
```

In [104…

```
X_train, X_test, y_train, y_test = train_test_split(games.drop('result',axis = 1),
pl.fit(X_train, y_train)
pl.score(X_test,y_test)
```

Out[1040]:

```
0.6603846153846153
```

Final Model

In [104…

```
ordinal_feats = ['patch']
nominal_feats = ['league','teamname']
quant_feats = [ 'golddiffat10', 'xpdiffat10', 'csdiffat10', 'killsat10', 'assistsat
 'opp_deathsat10', 'golddiffat15','xpdiffat15', 'csdiffat15','killsat15','assistsat
#games_test = games[ordinal_feats+nominal_feats+quant_feats+ ['gamelength', 'result
#X_train, X_test, y_train, y_test = train_test_split(games_test.drop('result',axis




def helper1(x):
    x = x.copy()
    y = Binarizer(threshold=1320)
    x['gamelength'] = y.transform(x[['gamelength']])
    return x
bin_ft = FunctionTransformer(helper1)




# Feature pipeline for all categories
feature_pipeline = ColumnTransformer([
    ('ord', OrdinalEncoder(), ordinal_feats),
    ('ohe', OneHotEncoder(handle_unknown='ignore'), nominal_feats),
    ('std', StdScalerByGroup(), ['league'] + quant_feats)

])

pl1 = Pipeline([
    ('ft', bin_ft),
    ('ct', feature_pipeline),
    ('dt', DecisionTreeClassifier())
])
```

In [104…

```
hyperparameters = {
    'dt__max_depth': [2, 3, 4, 10, 15],
    'dt__min_samples_split': [2, 5, 7, 10, 15, 20],
}
grids = GridSearchCV(pl1, param_grid=hyperparameters, return_train_score=True)
grids.fit(X_train, y_train)
```

```
Out[1047]: GridSearchCV(estimator=Pipeline(steps=[('ft',
                                                   FunctionTransformer(func=<function helper
           1 at 0x000001EBCF1CD1F0>)),
                                                  ('ct',
                                                   ColumnTransformer(transformers=[('ord',
                                                                                    OrdinalE
           ncoder(),
                                                                                    ['patch
           ']),
                                                                                   ('ohe',
                                                                                    OneHotEn
           coder(handle_unknown='ignore'),
                                                                                    ['league
           ',
                                                                                     'teamna
           me']),
                                                                                   ('std',
                                                                                    StdScale
           rByGroup(),
                                                                                    ['league
           ',
                                                                                     'golddi
           ffat10',
                                                                                     'xpdiff
           at10',
                                                                                     'csdiff
           at10',
                                                                                     'killsa
           t10',
                                                                                     'assist
           sat10',
                                                                                     'deaths
           at10',
                                                                                     'opp_ki
           llsat10',
                                                                                     'opp_as
           sistsat10',
                                                                                     'opp_de
           athsat10',
                                                                                     'golddi
           ffat15',
                                                                                     'xpdiff
           at15',
                                                                                     'csdiff
           at15',
                                                                                     'killsa
           t15',
                                                                                     'assist
           sat15',
                                                                                     'deaths
           at15',
                                                                                     'opp_ki
           llsat15',
                                                                                     'opp_as
           sistsat15',
                                                                                     'opp_de
```

```
 athsat15'])])),
                                          ('dt', DecisionTreeClassifier())]),
             param_grid={'dt__max_depth': [2, 3, 4, 10, 15],
                         'dt__min_samples_split': [2, 5, 7, 10, 15, 20]},
             return_train_score=True)
```

In [104…

```
grids.best_params_
```

Out[1048]:

```
{'dt__max_depth': 2, 'dt__min_samples_split': 2}
```

In [105…

```
plfinal = Pipeline([
    ('ft', bin_ft),
    ('ct', feature_pipeline),
    ('dt', DecisionTreeClassifier(max_depth=2,min_samples_split=2))
])
```

In [105…

```
plfinal.fit(X_train, y_train)
plfinal.score(X_test,y_test)
```

Out[1053]:

```
0.7373076923076923
```

In [ ]:

```

```

Fairness Evaluation

In [106…

```
from sklearn.metrics import accuracy_score
```

In [106…

```
X = games.drop('result', axis = 1)
y = games['result']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
plfinal.fit(X_train,y_train)
test_pred = plfinal.predict(X_test)
res = X_test
res['prediction'] = test_pred
res['actual'] = y_test
res['short'] = res.gamelength <= 1500
```

In [106…

```
obs = (
        res.groupby('short')
        .apply(lambda x: accuracy_score(x['actual'], x['prediction']))
        .diff()
        .abs()
        .iloc[-1]
    )

differences = []
# Permutation test of prediction
for i in range(1000):
    mean_diff = (
        res
        .assign(short=res['short'].sample(frac = 1, replace = False).reset_index(dr
        .groupby('short')
        .apply(lambda x: accuracy_score(x['actual'], x['prediction']))
        .diff()
        .abs()
        .iloc[-1]
    )
    differences.append(mean_diff)

(obs < differences).mean()
```

Out[1067]:

```
0.066
```

In [ ]:

In [ ]: