Progress Report: {European Soccer} Week 9

Brian Pascente
Carson Holscher
Jacob Richardson
Justin O'Donnell

Dataset Overview:

- **Final_Data_Wins.csv (csv)**: This is the same dataset as last weeks, but now has data of how many wins and ties each player has and the number of games played.
    - **Number of examples**: 10754 players with 25 columns

What We Are Predicting:

- **Target Variable**: Player performance (Number of wins).
- **Problem Type**: This is a regression problem, as the goal is to predict continuous values (wins).

Summary:

Brian Pascente:

Since the gradient boosting worked well with the dataset last week, I decided to continue using that with the new dataset.

Target variable: Goals

https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/Gradient_goals_ver2.ipynb

I ran the same gradient boost as last time but with this dataset to see the results of assessing players by the amount of goals. The results show that the R^2 actually went down from last week (0.97->0.59), which was interesting because I thought the R^2 would change, but I did not expect it to change by around half the amount just from adding the win and amount of games played columns.

```
Gradient Boosting Regression Performance (After Hyperparameter Tuning):
Best Parameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 200}
R²: 0.59
MSE: 0.01
MAE: 0.05
Best GBR Prediction: 0.20, Actual: 0.20
Worst GBR Prediction: 0.01, Actual: 0.43
```

Target variable: Wins

https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/Gradient_win.ipynb

I ran gradient boost, but changed the target to wins since that is also an important factor when assessing how good a player is. Overall, the results for this were extremely good and cross validation tests us that it's not overfitting. I think this is the case because there is not much noise in the data, and many teams did not get any wins, which made it easier to predict the data.

```
Gradient Boosting Regression Performance (After Hyperparameter Tuning):
Best Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
R²: 1.00
MSE: 0.00
MAE: 0.00
Best GBR Prediction: 0.00, Actual: 0.00
Worst GBR Prediction: 0.00, Actual: 0.00


Cross-validated R² scores: [1. 1. 1. 1. 1.]
Mean CV R²: 1.00
```

Target variable: Win rate

https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/Gradient_win_rate%20(2).ipynb

I also ran a gradient boost on the win rate, since just the number of wins may be too simple to determine the performance of a player. I did this by dividing the number of wins by the total games played, and dropping the two columns plus the number of games tied. The results for this were also really good, and it looked more realistic than the wins result. The results states its not overfitting, and the most important feature is the highest value of the player, which makes sense.

```
3]: df['win_rate'] = df['wins'] / df['games_played']
```

```
4]: df = df.drop('name', axis=1)
    df = df.drop('player', axis=1)
    df = df.drop('wins', axis=1)
    df = df.drop('ties', axis=1)
    df = df.drop('games_played', axis=1)
    X = pd.get_dummies(df.drop('win_rate', axis=1), drop_first=True)
    X = X.dropna()

    y = df['win_rate']
    y = y[X.index]


    # Remove outliers from y using IQR method
    Q1 = y.quantile(0.25)
    Q3 = y.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter out outliers
    non_outliers = (y >= lower_bound) & (y <= upper_bound)
    X = X[non_outliers]
    y = y[non_outliers]


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
Gradient Boosting Regression Performance (After Hyperparameter Tuning):
Best Parameters: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 200}
R²: 0.99
MSE: 0.00
MAE: 0.01
Best GBR Prediction: 0.30, Actual: 0.30
Worst GBR Prediction: 0.25, Actual: 0.21

Cross-validated R² scores: [0.99190421 0.98359328 0.99338693 0.99248849 0.99674541]
Mean CV R²: 0.99
```

```
]: importances = best_gbr_model.feature_importances_
feature_names = X_train.columns

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})


importance_df_sorted = importance_df.sort_values(by='Importance', ascending=False)


most_important_feature = importance_df_sorted.iloc[0]
print("Most Important Feature:")
print(most_important_feature)
```

```
Most Important Feature:
Feature       highest_value
Importance          0.13364
Name: 15, dtype: object
```

Target variable: Current worth

https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/Gradient_value.ipynb

Since how much a player is currently worth is important to determine how good a player is, I ran gradient boost on that to see how accurate it is with it. Overall, the results weren't as good as the win rate due to the high MSE/MAE since there's a lot of variance between the players.

```
Gradient Boosting Regression Performance (After Hyperparameter Tuning):
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
R²: 0.82
MSE: 428164209931.84
MAE: 358480.11
Best GBR Prediction: 249897.58, Actual: 250000.00
Worst GBR Prediction: 1789490.16, Actual: 6000000.00
```

Carson Holscher:

I had to trim things pretty aggressively, because using the largest amount of data that doesn't crash my computer from using too much ram still overfits badly & takes over 8 hours to train Given the abysmal results & the amount of time it takes to do a good job, I do not recommend proceeding with this model

Target variable: Goals
(https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/random forest goals v2.ipynb):
Not much has changed from last week, other than the fact that I'm using PCA.

```
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

grid = {
    'n_estimators': [10, 100],
    'max_depth': [3, 4, 5]
}

rf = GridSearchCV(RandomForestRegressor(random_state=42),param_grid=grid,return_train_score=True,n_jobs=-1)
rf.fit(X_train,y_train)

cv_results = cross_validate(rf,X_train,y_train,return_train_score=True)
R2_trainCV = cv_results['train_score'].mean()
R2_valid   = cv_results['test_score'].mean()
predictions = rf.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print('train R2 (CV) =',R2_trainCV,'  valid R2 =',R2_valid)
print()
R2_train = rf.score(X_train,y_train)
R2_test  = rf.score(X_test,y_test)
print('     train R2 =',R2_train,'    test R2 =',R2_test)
print('mse = ' + str(mse))
print('mae = ' + str(mae))
```

```
train R2 (CV) = 0.6433722642365198   valid R2 = 0.24962806895973158

     train R2 = 0.605279005072545     test R2 = 0.11252117586298704
mse = 0.46545741966236004
mae = 0.4861546882449755
```

The random forest regression model achieved a very bad testing R^2 of 0.11, indicating it explains 11% of the variance in the target variable, with highaverage errors (MSE: 0.46, MAE: 0.48).

Target variable: Current Worth
(https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/random forest value.ipynb)
Used a random forest regressor & last week's dataset in the interest of time

```
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

grid = {
    'n_estimators': [10, 100],
    'max_depth': [3, 4, 5]
}

rf = GridSearchCV(RandomForestRegressor(random_state=42),param_grid=grid,return_train_score=True,n_jobs=-1)
rf.fit(X_train,y_train)

cv_results = cross_validate(rf,X_train,y_train,return_train_score=True)
R2_trainCV = cv_results['train_score'].mean()
R2_valid   = cv_results['test_score'].mean()
predictions = rf.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print('train R2 (CV) =',R2_trainCV,'  valid R2 =',R2_valid)
print()
R2_train = rf.score(X_train,y_train)
R2_test  = rf.score(X_test,y_test)
print('     train R2 =',R2_train,'    test R2 =',R2_test)
print('mse = ' + str(mse))
print('mae = ' + str(mae))
```

```
train R2 (CV) = 0.47557558944554507   valid R2 = 0.05115947361424382

     train R2 = 0.5059262807809011     test R2 = 0.03457244817831062
mse = 0.08287297084520451
mae = 0.1970545104382699
```

The results severely overfit, & given the shenanigans I pulled to make the model train in a timely manner, this is not a big surprise

Target variable: Wins

Used the current dataset, aggressively trimmed down with PCA in the interest of time.

```
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

grid = {
    'n_estimators': [10, 100],
    'max_depth': [3, 4, 5]
}

rf = GridSearchCV(RandomForestRegressor(random_state=42),param_grid=grid,return_train_score=True,n_jobs=-1)
rf.fit(X_train,y_train)

cv_results = cross_validate(rf,X_train,y_train,return_train_score=True)
R2_trainCV = cv_results['train_score'].mean()
R2_valid   = cv_results['test_score'].mean()
predictions = rf.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print('train R2 (CV) =',R2_trainCV,'  valid R2 =',R2_valid)
print()
R2_train = rf.score(X_train,y_train)
R2_test  = rf.score(X_test,y_test)
print('     train R2 =',R2_train,'    test R2 =',R2_test)
print('mse = ' + str(mse))
print('mae = ' + str(mae))
```

```
train R2 (CV) = -144263.392144877   valid R2 = 0.0

     train R2 = -253961.80631848585     test R2 = 0.0
mse = 8.852991508842805e-28
mae = 2.9753977059954195e-14
```

Once again, the results are severely overfit.

Target variable: Win Percentage

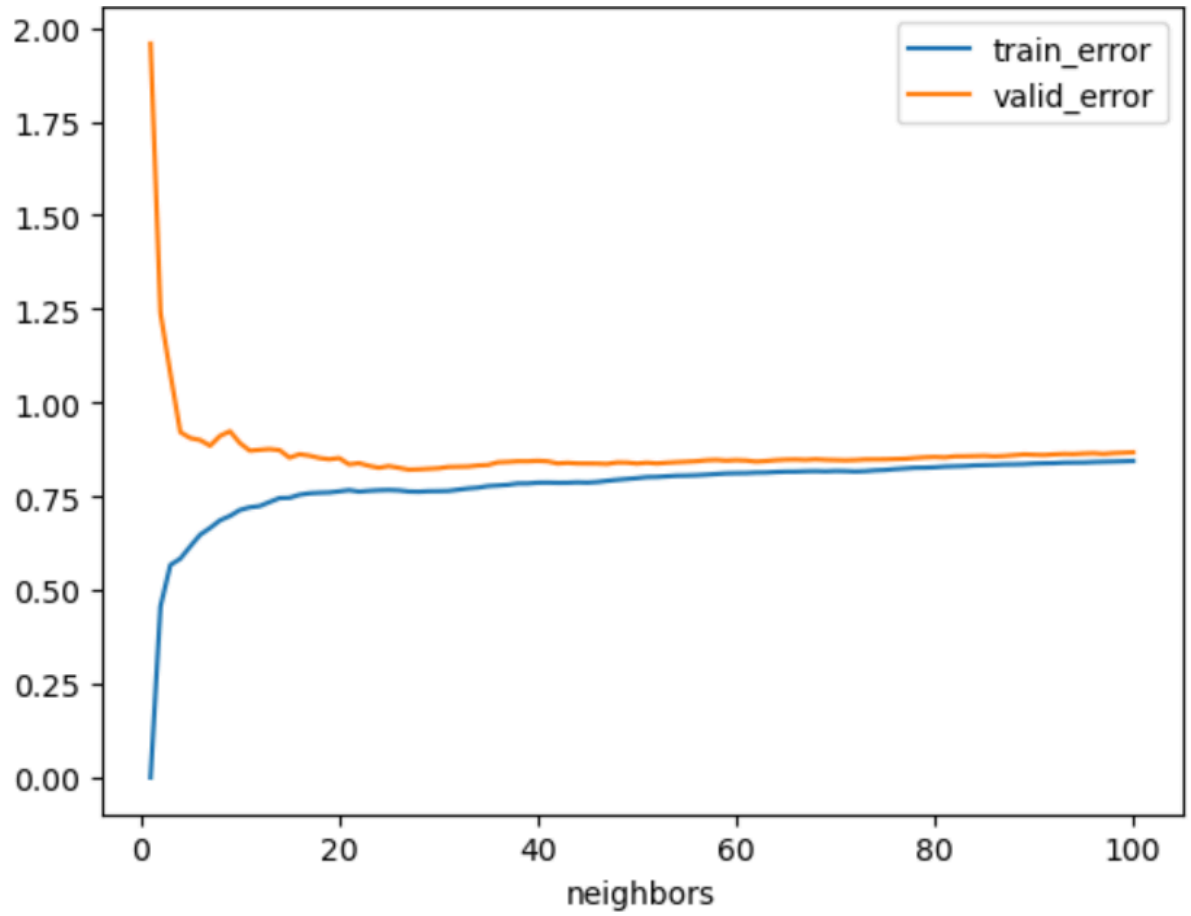
```
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

grid = {
    'n_estimators': [10, 100],
    'max_depth': [3, 4, 5]
}

rf = GridSearchCV(RandomForestRegressor(random_state=42),param_grid=grid,return_train_score=True,n_jobs=-1)
rf.fit(X_train,y_train)

cv_results = cross_validate(rf,X_train,y_train,return_train_score=True)
R2_trainCV = cv_results['train_score'].mean()
R2_valid   = cv_results['test_score'].mean()
predictions = rf.predict(X_test)

mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)

print('train R2 (CV) =',R2_trainCV,'  valid R2 =',R2_valid)
print()
R2_train = rf.score(X_train,y_train)
R2_test  = rf.score(X_test,y_test)
print('     train R2 =',R2_train,'    test R2 =',R2_test)
print('mse = ' + str(mse))
print('mae = ' + str(mae))
```

Jupyter has elected to not save my results, & I am about out of time, but they were pretty similar to wins (severely overfit).

Jacob Richardson:

I created a KNN regression model for our dataset using grid search and cross validation was used to find the best KNN hyperparameter.
(Github:https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/PR1-Copy2.ipynb)
In order to properly predict the data, I made each row into a team by averaging all the player's stats:

team	height	age	appearance	assists	goals conceded	games_injured	award	current_value	highest_value	wins
1.FC Köln	186.647059	24.823529	49.882353	0.098022	0.184745	26.352941	1.323529	2.996324e+06	5.617647e+06	0
1.FC Union Berlin	184.888889	27.592593	59.407407	0.105756	0.098266	24.555556	2.333333	4.670370e+06	6.770370e+06	0
1.FSV Mainz 05	186.666667	25.259259	53.259259	0.099091	0.159545	22.851852	1.481481	4.211111e+06	6.525926e+06	0
AA Argentinos Juniors	177.774678	25.866667	9.900000	0.050369	0.054579	4.233333	0.300000	8.150000e+05	1.288333e+06	0
AC Ajaccio	179.208012	28.433333	43.200000	0.043312	0.105117	17.200000	0.533333	9.050000e+05	1.875000e+06	22
...	...	...	...	...	...	...	...	...	...	...
Wolverhampton Wanderers	182.107143	26.250000	57.035714	0.101310	0.143143	29.214286	3.892857	1.423929e+07	2.326429e+07	25
Yokohama F. Marinos	176.216216	25.351351	11.270270	0.102654	0.081990	16.054054	2.000000	4.844595e+05	7.891892e+05	0
Yokohama FC	176.513514	25.621622	8.864865	0.074818	0.157258	11.432432	0.675676	3.560811e+05	5.540541e+05	0
Zenit St. Petersburg	182.217391	26.565217	52.739130	0.155672	0.123928	13.304348	6.869565	7.391304e+06	9.200000e+06	0
Ümraniyespor	183.227841	28.391304	45.739130	0.071753	0.300376	18.086957	1.565217	4.293478e+05	1.331522e+06	0



The R2 value for the optimal neighbors (K=27) was 0.198, which indicates KNN may not be the best predictor of team wins from the features used:

```
features = ['height', 'appearance', 'assists', 'goals conceded', 'games_injured', 'award', 'current_value', 'highest_value']
target = 'wins'
```

```
K = np.arange(100)+1 #Grid Search
grid = {'n_neighbors':K}
```

```
knnCV.fit(X_train, y_train)
```

```
#Pull information regarding best parameter
ix = results['valid_error'].idxmin()
results.iloc[ix]
```

```
neighbors      27.000000
train_error     0.762969
valid_error     0.821184
Name: 26, dtype: float64
```

```
# Compute Test Results - R2
knn = KNeighborsRegressor(n_neighbors=27)
knn.fit(X_train, y_train)
print(knn.score(X_test, y_test))
```

```
0.19764590979868768
```

Here were the best and worst predictors:

✅ Best Non-Zero Predictions:

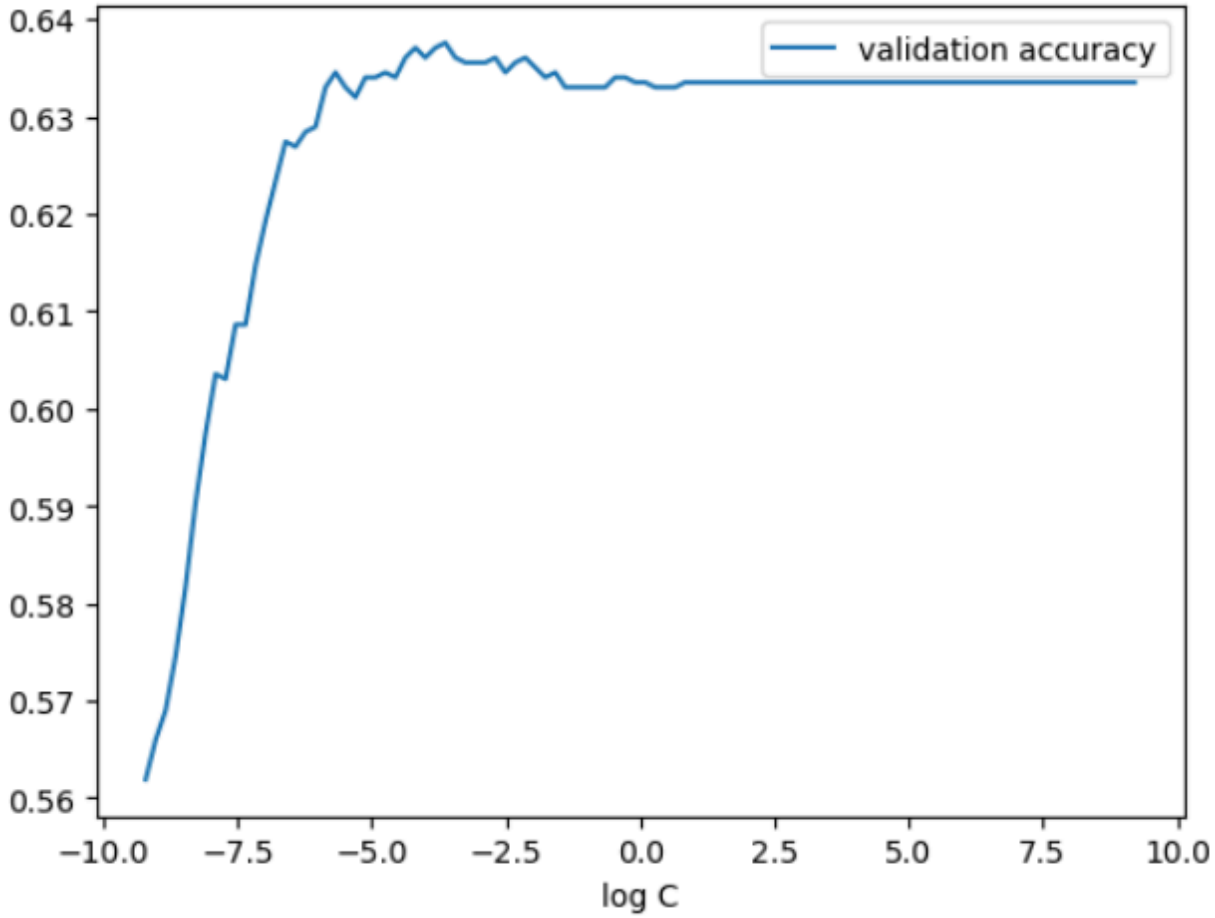
team	Actual	Predicted	Error	Absolute Error
Sparta Rotterdam	15	15.703704	-0.703704	0.703704
Boavista FC	17	15.814815	1.185185	1.185185
Gil Vicente FC	26	18.037037	7.962963	7.962963
Real Betis Balompié	56	38.481481	17.518519	17.518519
VfL Bochum	13	31.259259	-18.259259	18.259259

X Worst Non-Zero Predictions:

team	Actual	Predicted	Error	Absolute Error
CA Osasuna	67	12.518519	54.481481	54.481481
AS Monaco	95	34.777778	60.222222	60.222222
OGC Nice	106	34.296296	71.703704	71.703704
Borussia Mönchengladbach	109	32.888889	76.111111	76.111111
FC Barcelona	234	66.555556	167.444444	167.444444

Justin O'Donnell:

Coefficient Factor:
(https://github.com/rhit-pascenby/CSSE415-EuropeanSoccer/blob/main/CoefficientFactorsWinPercent.html)
In this code, I used the same algorithm as goals but changed the target value to win percent. I also had to manipulate the data greatly to get the final_data_wins.csv file that I shared with the team. As it was a combination of three dataset with a common api_id for each team and had to use that to cross determine the wins, ties, and games played of each player. In the coefficient factor code, I took the wins and divided them by games_played in order to get a win percentage. I deemed 0.33% win rate to be the divide between good and bad. This is because that is about the average win rate of a decent Premier League team. As it equates to a little over 1 point per game.



The validation accuracy was very high for this coefficient factor algorithm.

The chart for coefficient factor definitely had the most interesting data. The highest coefficient factor for success in a player/team is highest_value, which is very agreeable. As there is a reason higher spending teams do better. What surprised me the most was injuries being negligible. Often teams underperform due to an excess in injuries, for example Arsenal this recent Premier League season. But this is not a proven fact and more due to fan speculation.

So I believe this graph shows the surprising reality that injuries do not make a major difference in the success of a team. Appearances being a negative factor is also surprising but makes sense after further inspection. The reason it is surprising is higher performing teams recently have been pushing for more substitutions as their bench will out perform the other team's bench due to the huge pay difference. But more likely higher performing teams have more fit players, so they need less substitutions. One thing I should clarify is that appearances has a strong correlation with substitutes because both teams have 11 players starting with one appearance for the game and they only have more players with an appearance if they make a substitution. This should not be read into too much as there are many other factors such as how many players are on the roster.

The most surprising is that second_yellow_cards have a positive impact. I think this is just due to the lower amount of data that we acquired due to the number of teams we had to cut out from the data, due to there being no match data. This number is about around 2,000 players that remained with around 8,000 players being dropped due to a lack of data.