Data Analysis on the Euros

By Kyle Shiroma

CIC Program May 24-July 11, 2024

Faculty Mentor: Dr.Doina Bein, CSUF

Purpose

To explore the effectiveness of using various machine learning models to predict the outcomes of the UEFA European Championship (tournament).

First Steps

1. Downloaded Jupyter Notebook
2. Went to Kaggle for a dataset
3. Kaggle provided dataset on Euros covering matches from 1960-2024
4. Downloaded data
5. Opened it in Jupyter notebook

Dataset Overview

Description: Data covers information from every year the Euros were played

Included:

- Home and Away Teams
- Home and Away Team Scores
- Match Attendance
- Stadium Capacity
- Weather Conditions
- Winner

id_match	home_team	away_team	home_team_code	away_team_code	home_score
52509	Russia	Germany	RUS	GER	0.0
52503	France	Spain	FRA	ESP	1.0
52485	Scotland	England	SCO	ENG	0.0
52508	Czechia	Italy	CZE	ITA	2.0
52514	Portugal	Türkiye	POR	TUR	1.0
52484	Switzerland	Netherlands	SUI	NED	0.0
52502	Bulgaria	Romania	BUL	ROU	1.0
52513	Türkiye	Croatia	TUR	CRO	0.0
52507	Italy	Russia	ITA	RUS	2.0
52501	Romania	France	ROU	FRA	0.0
52483	Netherlands	Scotland	NED	SCO	0.0
52512	Denmark	Portugal	DEN	POR	1.0
52506	Germany	Czechia	GER	CZE	2.0
52500	Spain	Bulgaria	ESP	BUL	1.0
52482	England	Switzerland	ENG	SUI	1.0

Data Preprocessing

Measures taken:

1. Loading and combining csv files using glob module and concat
2. Handling missing values by replacing NaNs with the mean/dropping
3. Changed categorical values to numerical values through one-hot encoding

*Utilized df.shape, df.head, and df.describe to get a better overview of the data

```python
# Step 1: Load and concatenate DataFrames from multiple CSV files
all_dfs = []
for one_filename in glob.glob('data_euros/data_euros*.csv'):
    print(f'Loading {one_filename}')
    new_df = pd.read_csv(one_filename)
    all_dfs.append(new_df)

# Concatenate all DataFrames into one
df = pd.concat(all_dfs, ignore_index=True)

# Step 2: Handle missing values
# Replace NaNs in numerical columns with the mean of the column
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns
for col in numerical_cols:
    df[col].fillna(df[col].mean(), inplace=True)

# Drop rows with NaNs in categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
df.dropna(subset=categorical_cols, inplace=True)
```

```python
# Step 3: Encode categorical variables
df = pd.get_dummies(df, columns=['home_team', 'away_team', 'condition_weat
```

Decision Trees Classifier

Description

- Nonlinear relationships
- Feature importance and quick
- Accuracy: 55%

```
Accuracy: 0.55
              precision    recall  f1-score   support

     Belgium       0.50      1.00      0.67         1
     Croatia       0.50      0.50      0.50         2
     Czechia       0.33      1.00      0.50         1
     Denmark       1.00      1.00      1.00         1
     England       1.00      0.25      0.40         4
      France       0.50      1.00      0.67         1
     Germany       0.67      1.00      0.80         2
     Iceland       0.00      0.00      0.00         0
       Italy       1.00      1.00      1.00         3
 Netherlands       0.40      0.67      0.50         3
      Poland       0.00      0.00      0.00         2
    Portugal       0.00      0.00      0.00         3
      Russia       0.00      0.00      0.00         1
    Slovakia       0.00      0.00      0.00         1
       Spain       1.00      1.00      1.00         3
      Sweden       0.50      0.50      0.50         2
 Switzerland       0.00      0.00      0.00         0
     Ukraine       0.00      0.00      0.00         1
       Wales       0.00      0.00      0.00         0

    accuracy                           0.55        31
   macro avg       0.39      0.47      0.40        31
weighted avg       0.54      0.55      0.50        31
```

```python
import pandas as pd
import glob
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

#decision tree classifier

# Step 1: Load and concatenate DataFrames from multiple CSV files
all_dfs = []
for one_filename in glob.glob('data_euros/data_euros*.csv'):
    print(f'Loading {one_filename}')
    new_df = pd.read_csv(one_filename)
    all_dfs.append(new_df)

# Concatenate all DataFrames into one
df = pd.concat(all_dfs, ignore_index=True)

# Step 2: Select relevant features and target variable
features = ['home_team', 'away_team', 'home_score', 'away_score', 'match_attendance', 'stadium_capacity',
target = 'winner'
```

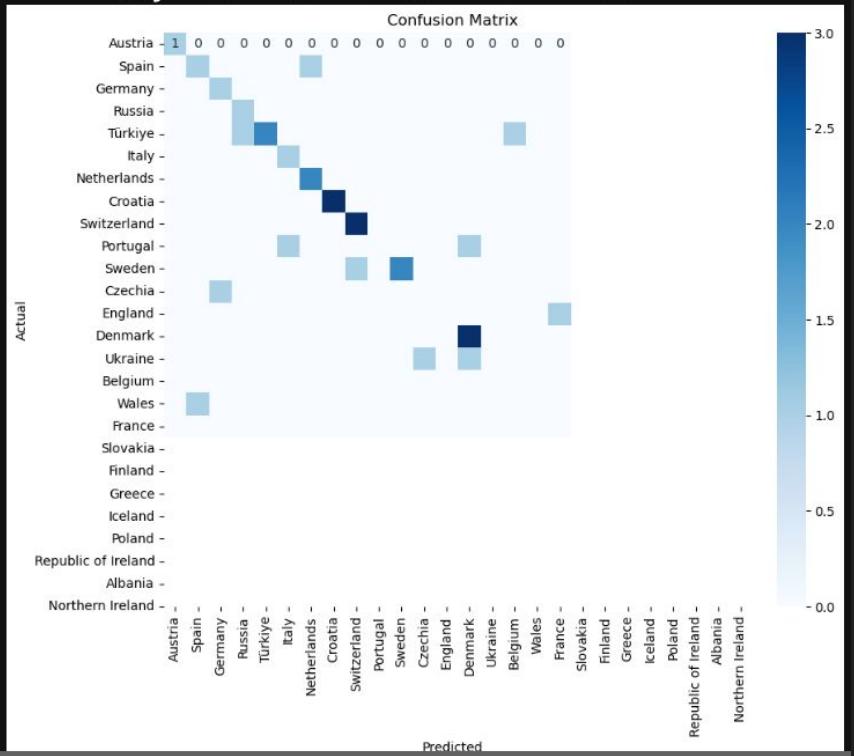
```python
df = df[features + [target]].dropna()  # Ensure only rows with complete data for selected features are used

# Step 3: Encode categorical variables
df = pd.get_dummies(df, columns=['home_team', 'away_team', 'condition_weather', 'stadium_name'])

# Step 4: Split data into training and testing sets
X = df.drop(columns=[target])
y = df[target]

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

# Step 5: Initialize and train Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Step 6: Predict on the test set and evaluate the model
y_pred = clf.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Random Forest Classifier

Description

- Accuracy 65%
- Relevant features
- Parameter tuning
- True Positives and Negatives
- False Positives and Negatives

```
Best parameters found:
{'max_depth': 20, 'min_samples_split': 2, 'n_
estimators': 100}
Accuracy with best estimator: 0.65
```



Code & Results

	precision	recall	f1-score	support
Belgium	1.00	1.00	1.00	1
Croatia	1.00	0.50	0.67	2
Czechia	0.50	1.00	0.67	1
Denmark	1.00	1.00	1.00	1
England	1.00	1.00	1.00	4
France	0.33	1.00	0.50	1
Germany	0.50	1.00	0.67	2
Italy	1.00	1.00	1.00	3
Netherlands	1.00	1.00	1.00	3
Poland	0.00	0.00	0.00	2
Portugal	0.00	0.00	0.00	3
Republic of Ireland	0.00	0.00	0.00	0
Russia	0.00	0.00	0.00	1
Slovakia	0.00	0.00	0.00	1
Spain	0.60	1.00	0.75	3
Sweden	0.00	0.00	0.00	2
Switzerland	0.00	0.00	0.00	0
Ukraine	0.00	0.00	0.00	1

```python
#Initialize and train RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

#Predict on the test set and evaluate the model
y_pred = clf.predict(X_test)

#Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred))

#Parameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy')

grid_search.fit(X_train, y_train)

best_estimator = grid_search.best_estimator_
print("Best parameters found:")
print(grid_search.best_params_)

#Use the best estimator for predictions
y_pred_grid = best_estimator.predict(X_test)
accuracy_grid = accuracy_score(y_test, y_pred_grid)
print(f'Accuracy with best estimator: {accuracy_grid:.2f}')

#Plot confusion matrix
conf_mat = confusion_matrix(y_test, y_pred_grid)
```

Tensorflow Neural Network Experiment

Description

- Large amounts of data
- Flexibility
- Accuracy 13.7%

```python
#Build the TensorFlow model
model = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),  #Define input shape
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(len(columns), activation='softmax')  #Output layer matches the number of unique classes
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Train model
model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_test_scaled, y_test))

#Evaluate model
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test Accuracy: {accuracy}")

#Predictions
predictions = model.predict(X_test_scaled)
```

```
an
Epoch 45/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.1711 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
Epoch 46/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.1953 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
Epoch 47/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.1796 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
Epoch 48/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1971 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
Epoch 49/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.2060 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
Epoch 50/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2174 - loss: nan - val_accuracy: 0.1333 - val_loss: n
an
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 896us/step - accuracy: 0.1331 - loss: nan
Test Accuracy: 0.13333334028720856
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

Model Evaluations/Reflection

Decision Trees & Random Forest

The use of more countries data with their players, specifically player rank, as well as countries participation needs to be more accurate

Tensorflow Improvement

The use of fouls have been indicated as a correlation in winning games, and would want to leverage that if I have more time in feature engineering and tuning for the tensorflow model.

Ie: more fouls had a higher chance of losing
Reason: fouls led to goal scoring opportunities like free kicks or penalties

Sources

Kaggle Dataset
https://www.kaggle.com/datasets/piterfm/football-soccer-uefa-euro-1960-2024

Dr.Doina Bein PPT Lecture and Slides

Kaggle Intermediate Machine Learning

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74