Final Project Script

June 6, 2025

1 Final Project: Soccer Machine Learning

1.0.1 DSC 140

1.0.2 Leo Sanchez

[ ]:
```
# basic imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from datetime import datetime
import scipy.stats as stats

# KNN imports
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay

# NN imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
# data files from local device
events_data = pd.read_csv('/Users/leo/Library/CloudStorage/
 ↪OneDrive-UniversityofMountUnion/DSC 140 Fall 2024/archive (1)/events.csv')
game_info = pd.read_csv('/Users/leo/Library/CloudStorage/
 ↪OneDrive-UniversityofMountUnion/DSC 140 Fall 2024/archive (1)/ginf.csv')
dictionary = open('/Users/leo/Library/CloudStorage/
 ↪OneDrive-UniversityofMountUnion/DSC 140 Fall 2024/archive (1)/dictionary.
 ↪txt')
```

[ ]:
```
# column identification for games_info file
game_info.columns
```

[ ]:
```
# merging events dataset with game_info based on
# id_odsp column, retaining all rows from events_data and
# adding the country and date columns from game_info
events = events_data.merge(game_info[['id_odsp', 'country', 'date']],␣
 ↪on='id_odsp', how='left')
```

[ ]:
```
# lambda function to extract the year from a date string in the format␣
 ↪'YYYY-MM-DD'.
# applied function to the date column of the events dataset and store the␣
 ↪results in a new year column
extract_year = lambda x: datetime.strptime(x, "%Y-%m-%d").year
events['year'] = [extract_year(x) for key, x in enumerate(events['date'])]
```

[ ]:
```
# filter the events file to create a new DataFrame shots
# that contains only the rows where the event_type is 1 which corresponds to␣
 ↪shot events
shots = events[events.event_type == 1]
```

[ ]:
```
# get column indices for player, player2, and country in shots DataFrame
player_idx = shots.columns.get_loc('player')
player2_idx = shots.columns.get_loc('player2')
country_idx = shots.columns.get_loc('country')
```

[ ]:
```
# Apply title-case formatting to the player, player2, and country columns in␣
 ↪the shots
# The function with a lambda ensures that the transformation is only applied to␣
 ↪string values
shots.iloc[:, player_idx] = shots.iloc[:, player_idx].apply(lambda x: x.title()␣
 ↪if isinstance(x, str) else x)
shots.iloc[:, player2_idx] = shots.iloc[:, player2_idx].apply(lambda x: x.
 ↪title() if isinstance(x, str) else x)
shots.iloc[:, country_idx] = shots.iloc[:, country_idx].apply(lambda x: x.
 ↪title() if isinstance(x, str) else x)
```

[ ]:
```
# group the shots by shot_outcome and count the number of occurrences of each␣
 ↪outcome
# by counting the id_event in each group
# rename the id_event column to count, then calculate the total count of
# all shot outcomes by summing the count column.
pie = shots[['shot_outcome', 'id_event']].groupby('shot_outcome').count().
 ↪reset_index().rename(columns={'id_event': 'count'})
count = pie['count'].sum()
count
```

[ ]:
```
# convert shot_outcome to integer
# replace the numeric values
pie.shot_outcome = pie.shot_outcome.astype(int)
```

```
pie.shot_outcome = pie.shot_outcome.replace({1: 'On Target', 2: 'Off Target', 3:
 ↪ 'Blocked', 4: 'Hit the Bar'})
```

[ ]:
```
# pie chart
fig, ax = plt.subplots(figsize=[12,8])
labels = pie['shot_outcome']
plt.pie(x=pie['count'], autopct="%.1f%%", labels=labels, explode=[0.02]*4,␣
 ↪pctdistance=0.12, \
       textprops=dict(fontsize=12))
plt.title("Shot Outcomes", fontsize=24)
plt.tight_layout()
plt.show()
```

[ ]:
```
# convert shot_place to integers
bar = shots[['shot_place', 'id_event']].groupby('shot_place').count().
 ↪reset_index().rename(columns={'id_event': 'count'})
bar.shot_place = bar.shot_place.astype(int)
bar.shot_place = bar.shot_place.replace({1: 'High', 2: 'Blocked', 3: 'Bottom␣
 ↪left corner', 4: 'Bottom right corner', \
                                         5: 'Centre of the goal', 6: 'High and␣
 ↪wide', 7: 'Hits the bar', 8: 'Misses to the left', \
                                         9: 'Misses to the right', 10: 'Too␣
 ↪high', 11: 'Top centre of the goal', \
                                         12: 'Top left corner', 13: 'Top right␣
 ↪corner'})
```

[ ]:
```
# replace numeric goal values with labels Goal and No Goal
goals  = shots[['is_goal', 'id_event', 'country']].groupby(['is_goal',␣
 ↪'country']).count().reset_index().rename(columns={'id_event': 'count'})
goals.is_goal = goals.is_goal.replace({1: 'Goal', 0: 'No Goal'})
```

[ ]:
```
# calculate the percentage of goals and no-goals for each country
goals['percentage']=0
for i in range(len(goals)):
    for country in goals.country.unique():
        if goals.iloc[i,goals.columns.get_loc("country")]==country:
            goals.iloc[i,goals.columns.get_loc("percentage")]=goals.
 ↪iloc[i,goals.columns.get_loc("count")] / \
                                                        goals[goals.
 ↪country==country]['count'].sum()
goals['percentage']=round(goals['percentage']*100,2)
```

[ ]:
```
# function to display the percentages on top of the bars in a bar plot
def show_values_on_bars(axs):
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
```

```
            _y = p.get_y() + p.get_height()
            value = '{:.2f}%'.format(p.get_height())
            ax.text(_x, _y+2, value, ha="center", fontsize=14)

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
    else:
        _show_on_single_plot(axs)
```

[ ]:
```
# bar plot
sns.set_style("whitegrid")
fig, ax = plt.subplots(figsize=[12,6])
ax = sns.barplot(data=goals, y='percentage', hue='is_goal', x='country')
ax.set_ylabel(ylabel='Percentage %')
ax.set_xlabel(xlabel='League')
ax.set_xticks(range(len(goals['country'].unique())))
ax.set_xticklabels(goals['country'].unique(), rotation=45)
plt.title("Goal/No-Goal per Country")
plt.tight_layout()
ax.grid(color='black', linestyle='-', axis='y')
plt.legend(fontsize=10)
show_values_on_bars(ax)
plt.show()
```

[ ]:
```
# group by shot_place and count events
bar = shots[['shot_place', 'id_event']].groupby('shot_place').count().
 ↪reset_index().rename(columns={'id_event': 'count'})
```

[ ]:
```
# contingency table between is_goal and shot_place
# shows the frequency count of goals (1/0) for each shot place
contingency_table = pd.crosstab(goals['is_goal'], bar['shot_place'])
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
```

[ ]:
```
# handle missing values
# replace nan with the mean
X = shots[['shot_place']].fillna(shots['shot_place'].mean())
y = shots['is_goal']
```

[ ]:
```
# split data into training and test sets for KNN
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=10)
```

[ ]:
```
# lists to store k values and corresponding accuracy scores
k_values = []
accuracy_scores = []
```

[ ]:
```
# loop over different k values
for k in range(1, 75, 2):
    classifier = KNeighborsClassifier(n_neighbors=k)
    classifier.fit(X_train, y_train)

    # predict the target on the test set
    prediction = classifier.predict(X_test)

    # calculate the accuracy and store the result
    accuracy = accuracy_score(y_test, prediction)
    k_values.append(k)
    accuracy_scores.append(accuracy * 100)
```

[ ]:
```
# plotting the results
plt.plot(k_values, accuracy_scores, linewidth=2, color="green")
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy (%)")
plt.grid(True)
plt.show()
```

[ ]:
```
# classifier using training data in test data
# predict and records an accuracy score
classifer = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print('Accuracy:', accuracy*100)
```

[ ]:
```
# generates a confusion matrix
# shows how many predictions were correct
ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test)
```

[ ]:
```
# get unique shot locations
unique_shot_places = shots['shot_place'].unique()
```

[ ]:
```
# predict goal probability for each shot place
shot_goal_probabilities = []
```

[ ]:
```
for shot in unique_shot_places:
    # creates a dataframe with a single shot_place
    shot_df = pd.DataFrame({'shot_place': [shot]})
    shot_df = shot_df.fillna(shots['shot_place'].mean())

    # predict using the trained model
    probability = classifier.predict_proba(shot_df)[0][1]
    shot_goal_probabilities.append((shot, probability))
```

[ ]:
```
# convert to DataFrame for visualization
shot_prob_df = pd.DataFrame(shot_goal_probabilities, columns=['shot_place',␣
 ↪'goal_probability'])
```

[ ]:
```
# plot the probabilities
sns.barplot(data=shot_prob_df, x='shot_place', y='goal_probability')
plt.xlabel("Shot Place")
plt.ylabel("Goal Probability")
plt.title("Goal Probability by Shot Place")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

[ ]:
```
# mapping shot_place numeric codes to labels
shot_place_labels = {
    1: 'High',
    2: 'Blocked',
    3: 'Bottom left corner',
    4: 'Bottom right corner',
    5: 'Centre of the goal',
    6: 'High and wide',
    7: 'Hits the bar',
    8: 'Misses to the left',
    9: 'Misses to the right',
    10: 'Too high',
    11: 'Top centre of the goal',
    12: 'Top left corner',
    13: 'Top right corner'
}
```

[ ]:
```
# convert the probabilities to a DataFrame
shot_goal_prob_df = pd.DataFrame(shot_goal_probabilities,␣
 ↪columns=['shot_place', 'goal_probability'])
```

[ ]:
```
# map the numeric shot_place codes to their labels
shot_goal_prob_df['shot_place_label'] = shot_goal_prob_df['shot_place'].
 ↪map(shot_place_labels)
```

[ ]:
```
# plotting the goal probabilities for each shot placement
sns.set_style("whitegrid")
fig, ax = plt.subplots(figsize=[13, 6])
sns.barplot(data=shot_goal_prob_df, y='shot_place_label', x='goal_probability',␣
 ↪ax=ax)
ax.set_xlabel("Goal Probability", fontsize=14)
ax.set_ylabel("")
ax.set_title("Goal Probability by Shot Placement", fontsize=18)
plt.tight_layout()
```

```
plt.show()
```

[ ]:
```
# summary of stats in odd_a
print(game_info['odd_a'].describe())
print(shots['shot_place'].value_counts())
print(shots['country'].unique())
print(shot_goal_probabilities)
```

[ ]:
```
# attempt of scatterplot for goal_probability and shot_place
# with correlation scores
plt.figure()
plt.scatter(shot_goal_prob_df['goal_probability'],␣
 ↪shot_goal_prob_df['shot_place'])
pearson_corr = shot_goal_prob_df['shot_place'].
 ↪corr(shot_goal_prob_df['goal_probability'])
spearman_corr = shot_goal_prob_df['shot_place'].
 ↪corr(shot_goal_prob_df['goal_probability'], method='spearman')
print("Pearson Correlation:", pearson_corr)
print("Spearman Correlation:", spearman_corr)
```

[ ]:
```
# extract the count for the group where is_goal is Goal/No Goal
# Kolmogorov-Smirnov test on the two groups
group1 = goals[goals['is_goal'] == 'Goal']['count']
group2 = goals[goals['is_goal'] == 'No Goal']['count']
ks_stat, ks_p_value = stats.ks_2samp(group1, group2)
print(f"KS Test Statistic: {ks_stat}, P-value: {ks_p_value}")
```

[ ]:
```
# preprocess the data fo NN
X = shots[['shot_place']].fillna(shots['shot_place'].mean())
y = shots['is_goal']
```

[ ]:
```
# split data into train and test sets
_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=10)
```

[ ]:
```
# NN model
model = Sequential()
model.add(Dense(12, input_dim=X.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[ ]:
```
# compiler
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# training
model.fit(X_train, y_train, epochs=10, batch_size=10)
```

[ ]:
```
# testing
model.fit(X_test, y_test, epochs=20, batch_size=10)
```

[ ]:
```
# accuracy scores
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```