Predicting tennis match results with machine learning

Introduction

Tennis is a well-known sport that has over 85 million players around the world (1). Thus, it follows naturally that the competitive scene of tennis is also of high quality. Likewise, the betting companies offer a way to place bets on the winners of the matches. This shows that being able to accurately predict the outcome of a tennis match may prove to be financially beneficial.

With plenty of motivation to predict the winner of the match, it's time to determine a way to achieve this with machine learning (ML). Fortunately, with a sport that prevalent, there is plenty of open-source data available on the Internet. In this project, I will be using the dataset put together by Jeff Sackmann, who has gathered the results of the Association of Tennis Professionals (ATP) Tour, a worldwide top-tier tennis competition series, for over half a century, starting from the year 1968. The dataset contains information about the winner of the matches, the surface on which the match was played, the age, whether the player was left- or right-handed, and many other features. The data points are the tennis matches. Another source that is possibly useful for either complementing the algorithm or verifying the results is the website https://www.atptour.com which has the all-time stats for every single professional tennis player.

I will mainly focus on players that are still active today, as analyzing the players from the last millennium would not be of much use since the majority of them have retired long ago (and thus would not prove to be useful from the future games prediction standpoint). The training and testing set are obtained from the same source.

The core aim

The ultimate question is: which of the two players will win? This is what we will be trying to figure out (by estimating the respective probabilities of each player winning), as this, in theory, should yield a hefty return on bookmakers' sites. Fortunately, in tennis, it is impossible to have the match result in a draw. Thus, we're dealing with a classification problem.

Here's some background about the game, to add context to those not familiar with the sport. The scoring system is divided into sets, games, and points. A set is won if the player wins 6 games with at least a 2-game lead. A game is won if the player has won 4 points (15, 30, 40, game) with at least a 2-point lead. A match is won after two or three sets, depending on the layout of the tournament. The game is played on three different types of courts: hard, clay, and grass. These affect the bounce and speed of the ball, and thus suit some players more than others.

When a player serves an "ace", it denotes a service that went unreturned (grants the server a point right away).

Features and models

The features that I intend to use in the project are the following: the court type (hard, clay, grass), player's age, world ranking, player's height, and whether the player is left- or right-handed. For player-related features, I will take into account both players' attributes.

As can be noted, I will not analyze the probability of winning an individual point, as this would require extra methods for converting the probability of winning a given point to the probability of winning the whole match (and would require an approach that goes out of scope of the course at hand).

The methods that I will implement are, in no particular order: Logistic Regression, Decision Tree, K-nearest neighbors, and Support Vector Machines. These should be the most suitable for a classification problem as this (i.e. win/loss, yes/no problems).

Methods (logistic regression)

In this report, I will give an overview of the results gotten from the Logistic Regression model. It was one of the models listed previously and as we had covered it in classes, too, I figured it would be a suitable starting point.

I used the data from the years 2000-2020 (2 decades' worth of ATP matches) to predict the outcome of the next game. This resulted in a total of slightly over half a million data points. However, after cleaning up the data (particularly removing the rows which had a large proportion of null-values), I was left with 446,331 data points. It is worth noting here that for player heights, I replaced null-values with the mean height of all players.

The data was given similarly to this layout: "winner_name", "winner_rank", etc… It meant that there was no label, per se, and thus I created one. The way I implemented this was to replace the winner and loser column names with player1 and player2, respectively, after which I picked half of the dataset and swapped the two columns. This way, I had half of the dataset with labels 1 (for player 1's win) and the other half with 0's (player 2's win). This move was necessary for the model to work (it did not accept labels with only 1's in it, as initially all the player1's were unswapped and thus had 1 as their label).

The reason why I chose to include players' height, ranking, hand, and the surface in the feature list is that as an enthusiastic tennis player, these attributes seem to play the largest role in determining the winner of the game.

I used Sci-kit (4) learn package for fitting the models and the Pandas (5) package for analyzing and processing the data. Additionally, I used Matplotlib (6) to visualize some of the aspects. For cross-validation, I used the KFold cross-validation method with k = 3. This value for k was more than enough, as with a dataset of nearly half a million data points, the variation is minimal – one could have gotten a surprisingly accurate picture of the model by splitting the data only once, even. Nonetheless, to render the model as robust as possible, I kept the KFold method.

For the loss function, I used logistic loss, as this seemed to be the most suitable loss function for a classification problem that uses logistic regression as its model. Furthermore, the logistic loss is widely documented and has proven to be the optimal loss function for such a problem.

In the appendix, you can see the actual code and the results obtained with the aforementioned approaches.

References

1) Anon, (2021). *How Many People Play Tennis In The World? - Tennis Creative*. [online] Available at: https://tenniscreative.com/how-many-people-play-tennis/#:~:text=How%20many%20people%20play%20tennis%20in%20the%20US%3F [Accessed 9 Feb. 2022].
2) GitHub. (n.d.). *GitHub - JeffSackmann/tennis_atp: ATP Tennis Rankings, Results, and Stats*. [online] Available at: https://github.com/JeffSackmann/tennis_atp.
3) The Official website of the ATP Tour: https://www.atptour.com (09.02.2022)
4) scikit-learn (2019). *scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/.
5) Pandas (2018). *Python Data Analysis Library — pandas: Python Data Analysis Library*. [online] Pydata.org. Available at: https://pandas.pydata.org/.
6) Matplotlib (2012). *Matplotlib: Python plotting — Matplotlib 3.1.1 documentation*. [online] Matplotlib.org. Available at: https://matplotlib.org/.

tennis_predictor

March 9, 2022

[246]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, log_loss
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import svm
import seaborn as sns
```

[209]:
```
seed = 42

## improve speed by limiting max columns shown at once
pd.set_option('display.max_columns', 50)
```

0.0.1 Merge the match data from 2000-2020

[194]:
```
frames = []

# Add games from years 2000-2020
for year in range(2000, 2021):
    frames.append(pd.read_csv(f"./data/matches_{year}.csv"))

df = pd.concat(frames)
```

0.0.2 Clean up the data

[195]:
```
## irrelevant columns
irr_cols = []
player_cols = ["ace", "df", "svpt", "1stIn", "1stWon", "2ndWon", "SvGms",␣
 ↪"bpSaved", "bpFaced"]
general_cols = ["winner_name", "tourney_name", "loser_name", "winner_ioc",␣
 ↪"loser_ioc", "minutes", "score", "draw_size", "round", "loser_seed",␣
 ↪"loser_entry", "loser_rank_points", "winner_rank_points", "winner_seed",␣
 ↪"winner_entry", "match_num", "tourney_id", "tourney_level", "best_of"]
## add winner/loser cols
```

```
for el in player_cols:
    irr_cols.append(f"w_{el}")
    irr_cols.append(f"l_{el}")

## add the general cols
irr_cols.extend(general_cols)

## drop irrelevant cols
df.drop(columns=irr_cols, inplace=True)

## calculate mean height of the players and replace null values with it
mean_ht = (df["winner_ht"].mean() + df["loser_ht"].mean()) / 2
df.loc[df["winner_ht"].isnull(), "winner_ht"] = mean_ht
df.loc[df["loser_ht"].isnull(), "loser_ht"] = mean_ht

## drop empty values
df.dropna(axis=0, inplace=True)
```

0.0.3 Convert numerical values to floats

[196]:
```
numeric_cols = ["winner_rank", "loser_rank", "winner_age", "loser_age",␣
↪"winner_ht", "loser_ht"]
df[numeric_cols] = df[numeric_cols].astype(float)
```

0.0.4 Adjust the dataset for performing predictions

[197]:
```
## replace winner/loser with player1 & 2
for (colName, colData) in df.iteritems():
    if "winner" in colName:
        endingVal = colName.split("_")[1]
        df.rename(columns={colName: f"player1_{endingVal}"}, inplace=True)
    elif "loser" in colName:
        endingVal = colName.split("_")[1]
        df.rename(columns={colName: f"player2_{endingVal}"}, inplace=True)


## swap the labels for half of the values

first_half, second_half = df[:int(len(df) / 2)], df[int(len(df) / 2):]
first_half.insert(loc=0, column="label", value=1)

## swap columns
scols = list(second_half.columns)

np.warnings.filterwarnings('ignore')
```

```
## swap the columns for the second half
for attr in ["id", "hand", "ht", "age", "rank"]:
    scols[scols.index(f"player1_{attr}")], scols[scols.
 ↪index(f"player2_{attr}")] = scols[scols.index(f"player2_{attr}")],␣
 ↪scols[scols.index(f"player1_{attr}")]
    # second_half.insert(loc=0, column="label", value=0)
    second_half["label"] = 0

halves = []

df = pd.concat([first_half, second_half])
```

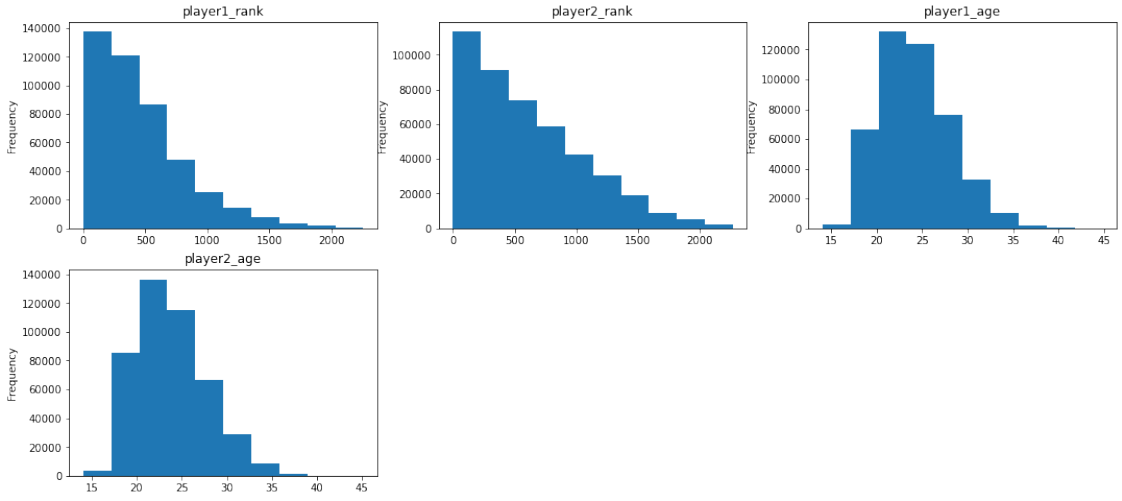
0.0.5 Visualize data

[198]:
```
cols = ["player1_rank", "player2_rank", "player1_age", "player2_age"]

plt.figure(1, figsize=(18, 8))

for i in range(len(cols)):
    plt.subplot(2, 3, i + 1)
    df[cols[i]].plot(kind="hist", title=cols[i])

plt.show()
```



0.0.6 Prepare data for ML models

[239]:
```
## convert categorical values (player hand, surface) to numbers
le = LabelEncoder()
df["surface"] = le.fit_transform(df["surface"])
```

```
df["player1_hand"] = le.fit_transform(df["player1_hand"])
df["player2_hand"] = le.fit_transform(df["player2_hand"])

## features that will be used in the model
features = ["player1_ht", "player2_ht", "player1_age", "player2_age",␣
 ↪"player1_rank", "player2_rank", "surface", "player1_hand", "player2_hand"]
X = df[features]
y = df["label"]

kf = KFold(n_splits=3, shuffle=True, random_state=seed)
```

0.0.7 Logistic Regression

[253]:
```
train_err_sum, val_err_sum, train_acc_sum, val_acc_sum, precision_sum = 0, 0,␣
 ↪0, 0, 0

for train_indices, val_indices in kf.split(X):
    X_train, X_val, y_train, y_val = X.iloc[train_indices], X.
 ↪iloc[val_indices], y.iloc[train_indices], y.iloc[val_indices]

    clf_1 = LogisticRegression()

    ## training the model
    clf_1.fit(X_train, y_train)
    y_pred_train_1 = clf_1.predict(X_train)
    acc_train_1 = clf_1.score(X_train, y_train)
    accuracy_1 = clf_1.score(X, y)

    ## validation
    y_pred_val_1 = clf_1.predict(X_val)
    acc_val_1 = clf_1.score(X_val, y_val)


    ## confusion matrix
    conf_mat = confusion_matrix(y, np.concatenate([y_pred_train_1,␣
 ↪y_pred_val_1]))

    ## calculate the precision by dividing TP by (TP + FP)
    ## it denotes the proportion of true positives in total positives
    true_pos = conf_mat[1][1]
    false_pos = conf_mat[0][1]
    precision = true_pos / (true_pos + false_pos)

    ## errors
    err_train = log_loss(y_train, y_pred_train_1)
    err_val = log_loss(y_val, y_pred_val_1)
```

```
    train_err_sum += err_train
    val_err_sum += err_val
    train_acc_sum += acc_train_1
    val_acc_sum += acc_val_1
    precision_sum += precision

    print(f"training accuracy: {acc_train_1}")
    print(f"validation accuracy: {acc_val_1}")
    print(f"precision: {precision}")
    print(f"training error: {err_train}")
    print(f"validation error: {err_val}")
    print("--------------")


n = kf.get_n_splits()

print("total results:")
print(f"\taverage training accuracy: {train_acc_sum/n}")
print(f"\taverage validation accuracy: {val_acc_sum/n}")
print(f"\taverage precision: {precision_sum/n}")
print(f"\taverage training error: {train_err_sum/n}")
print(f"\taverage validation error: {val_err_sum/n}")
```

```
training accuracy: 0.6059975668282059
validation accuracy: 0.6066529100600228
precision: 0.54646239908461
training error: 13.608530672524596
validation error: 13.585896416317375
--------------
training accuracy: 0.6061420784126579
validation accuracy: 0.6059673202175068
precision: 0.5476622806640103
training error: 13.603536457890108
validation error: 13.609572367930502
--------------
training accuracy: 0.6054497671010977
validation accuracy: 0.6055304247296289
precision: 0.5481662760534687
training error: 13.627447922550193
validation error: 13.624661241464239
--------------
total results:
        average training accuracy: 0.6058631374473206
        average validation accuracy: 0.6060502183357195
        average precision: 0.5474303186006964
        average training error: 13.613171684321633
```

```
average validation error: 13.606710008570706
```