Kobe_notebook

January 6, 2018

1 Kobe Bryant Shot Selection

Aviv Dotan and Ben Serota

1.1 Pre-proccessing

Imports

```
In [33]: import os
         import pandas as pd
         import numpy as np
         from sklearn.feature_selection import RFE
         from sklearn.model_selection import cross_val_score
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeRegressor, export_graphviz
         import matplotlib.pyplot as plt
         import graphviz
         from matplotlib.patches import Circle, Rectangle, Arc

         # Suppress warnings
         pd.options.mode.chained_assignment = None  # default='warn'
```

Define the input and output files' names and locations

```
In [34]: # Files definitions
         datadir = 'Data'
         datafilename = os.path.join(datadir, 'data.csv')
         resultsfilename = os.path.join(datadir, 'results.csv')
         fullresultsfilename = os.path.join(datadir, 'results_full.csv')
```

Load the data and sort it chronologically

```
In [35]: # Load data
         rawdata = pd.read_csv(datafilename)

         # Sort chronologically and redefine the index (for leakage handling)
         rawdata['date'] = pd.to_datetime(rawdata['game_date'])
```

```
rawdata.sort_values(['date', 'game_event_id'],
                    ascending = True, inplace = True)
rawdata.drop('date', axis = 1, inplace = True)
rawdata['index'] = range(len(rawdata))
rawdata.set_index('index', drop = True, inplace = True)
```

Define variables' types

In [36]:
```
# Define variables definitions

# Categorical variables
cat_vars = ['action_type', 'combined_shot_type', 'period', 'season',
            'shot_type', 'shot_zone_area', 'shot_zone_basic',
            'shot_zone_range', 'opponent']

# Irrelevant variables
drop_vars = ['game_id', 'lat', 'lon', 'team_id',
             'team_name', 'matchup']

# Date variables
date_vars = ['game_date']

# Output variable
pred_var = 'shot_made_flag'

# Prediction's columns
pred_cols = ['shot_id', 'shot_made_flag']
```

Generate new variables

In [37]:
```
# new variables

# home or not?
rawdata['home'] = rawdata['matchup'].apply(
                lambda x: 1 if (x.find('@') < 0) else 0)

rawdata['seconds_from_period_end'] = 60*rawdata['minutes_remaining'] +\
                                     rawdata['seconds_remaining']
rawdata['seconds_from_period_start'] = 60*(11 - rawdata['minutes_remaining']) +\
                                       (60 - rawdata['seconds_remaining'])
rawdata['seconds_from_game_start'] = (rawdata['period'] <= 4).astype(int) *\
                                     (rawdata['period']-1)*12*60 +\
                                     (rawdata['period'] > 4).astype(int) *\
                                     ((rawdata['period']-4)*5*60 + 3*12*60) +\
                                     rawdata['seconds_from_period_start']
rawdata['period_last_5_seconds'] = (rawdata['seconds_from_period_end'] < 6).\
        astype(int)
rawdata['game_last_5_seconds'] = rawdata['period_last_5_seconds'] *\
```

```
                                        (rawdata['period'] > 3).astype(int)
        rawdata['angle'] = np.arctan2(rawdata['loc_x'], rawdata['loc_y'])
```

Handle date and categorical variables

```
In [38]: for var in date_vars:
                # Convert the date variable to month, week no. and week day, and treat
                # them as categorical features
                datevar = pd.to_datetime(rawdata[var], format = "%Y-%m-%d")
                rawdata[var + '_month'] = datevar.dt.month
                rawdata[var + '_month_x'] = np.sin(2 * np.pi * rawdata[var + '_month'] / 12)
                rawdata[var + '_month_y'] = np.cos(2 * np.pi * rawdata[var + '_month'] / 12)
                cat_vars.append(var + '_month')
                rawdata[var + '_weekday'] = datevar.dt.dayofweek
                rawdata[var + '_weekday_x'] = np.sin(
                                2 * np.pi * rawdata[var + '_weekday'] / 7)
                rawdata[var + '_weekday_y'] = np.cos(
                        2 * np.pi * rawdata[var + '_weekday'] / 7)
                cat_vars.append(var + '_weekday')

        for var in cat_vars:
                # Replace the categorical feature with N binary features
                cat_list = pd.get_dummies(rawdata[var], prefix = var, prefix_sep = ': ')
                data1 = rawdata.join(cat_list)  # adds data columns to data
                rawdata = data1
```

Filter only relevant variables and split data columns

```
In [39]: data_vars = rawdata.columns.values.tolist()
         to_keep = [i for i in data_vars if (i not in cat_vars and
                                             i not in drop_vars and
                                             i not in date_vars)]
         rawdata = rawdata[to_keep]  # generating a new dataset, overriding old one.

         X_cols = [i for i in rawdata.columns if i not in pred_var]  # predict by cols
         Y_cols = [pred_var]
```

1.2 Prepare the training and evaluation data

Split the data into training and evaluation sets

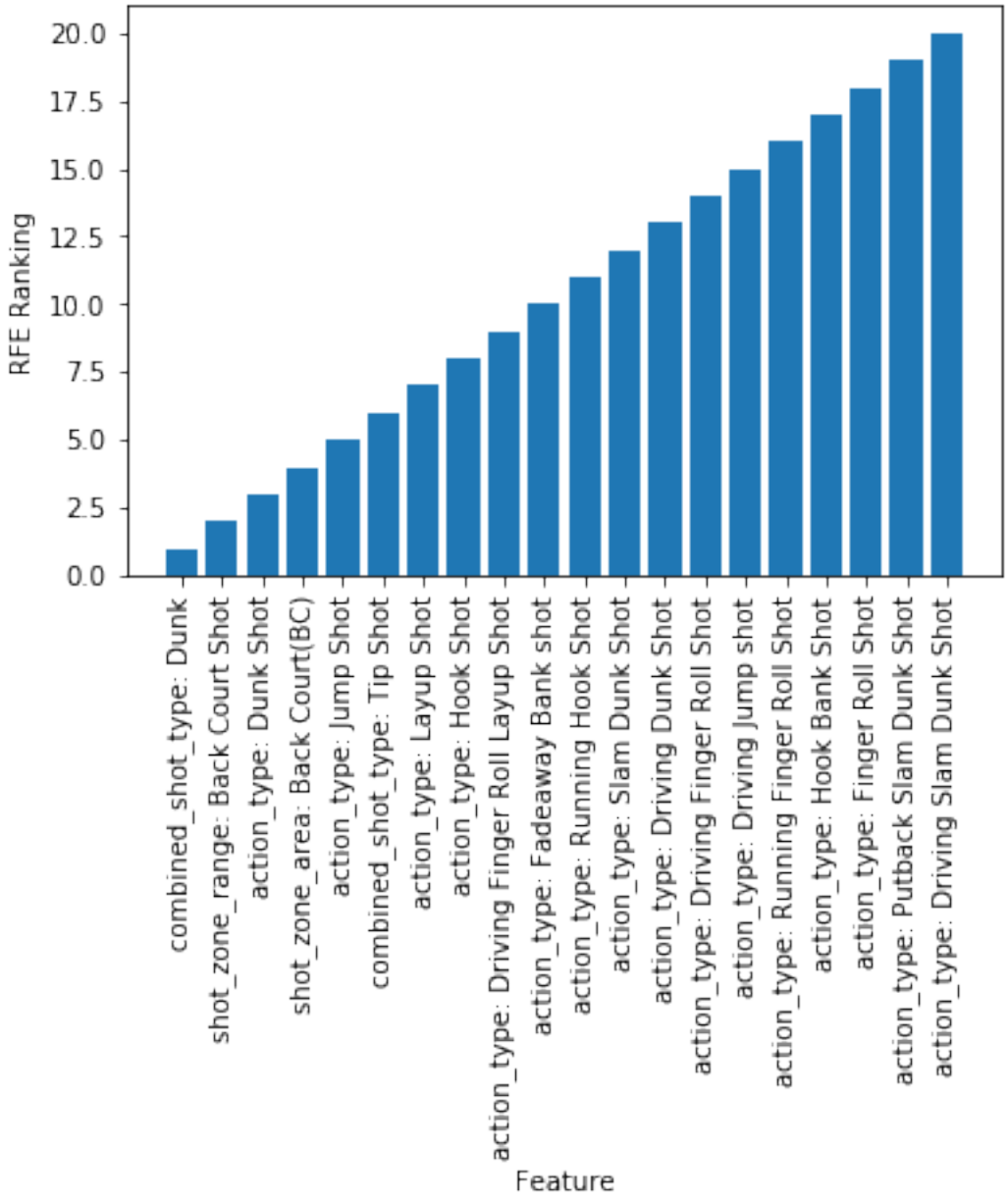
```
In [40]: # Split data
         test_rows = pd.isnull(rawdata[pred_var])    # gives indices of rows to predict
         traindata = rawdata[~test_rows]
         evaldata = rawdata[test_rows]

         # Prepare the training data
         X0 = traindata[X_cols]
         Y0 = np.ravel(traindata[Y_cols])
```

1.3 Features selection

Plot the features' rankings according to Recursive Feature Elimination

```
In [41]: # Plot RFE rankings
         lr = LogisticRegression()
         rfe = RFE(lr, 1)
         rfe = rfe.fit(X0, Y0)
         feature_df = pd.DataFrame(rfe.ranking_, index = X_cols, columns = ["ranking"])
         feature_df.sort_values("ranking", inplace = True)
         feature_df = feature_df.head(20)
         plt.figure()
         x_axis = np.arange(len(feature_df))
         plt.bar(x_axis, feature_df["ranking"])
         plt.xticks(x_axis, feature_df.index, rotation = 'vertical')
         plt.xlabel("Feature")
         plt.ylabel("RFE Ranking")
         plt.show()
```



Use Recursive Feature Elimination to find the most important 10 features

```
In [42]: # Choose a subset of features by recursive features elimination
         lr = LogisticRegression()
         n_features = 10
         rfe = RFE(lr, n_features)  # chooses the best N features from which to do LR
         rfe.fit(X0, Y0)
         X_cols_rfe = [X_cols[i] for i in range(len(X_cols)) if rfe.get_support()[i]]
         print('RFE chosen features: ', X_cols_rfe)
         print(len(X_cols_rfe))
```

```
RFE chosen features:  ['action_type: Driving Finger Roll Layup Shot', 'action_type: Dunk Shot'
10
```

Use LDA to find the optimal projection of the data to 1D

```
In [43]: # LDA
         lda = LinearDiscriminantAnalysis()
         lda.fit(X0, Y0)
         X0_lda = lda.transform(X0)
```

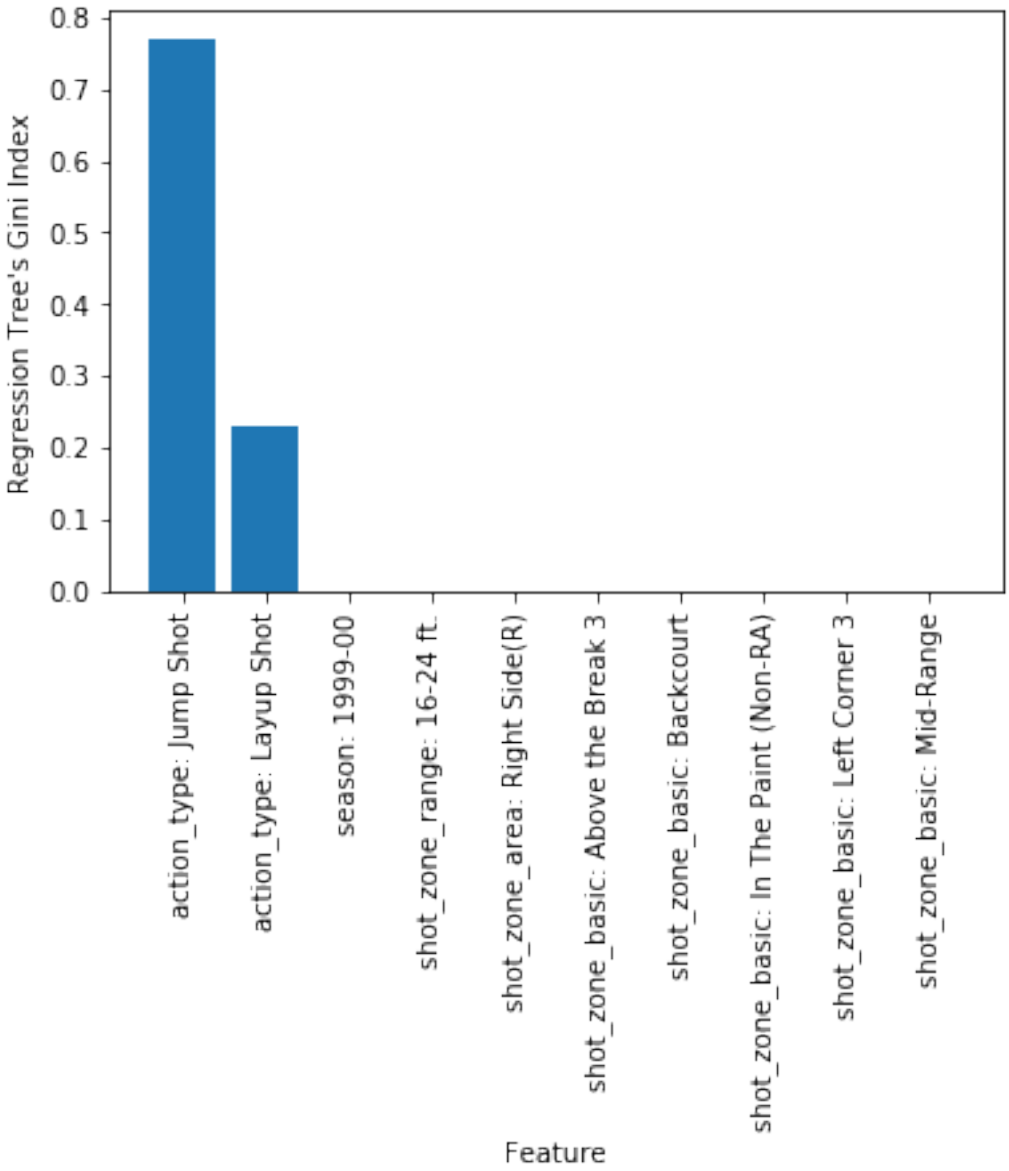
```
C:\Users\Aviv\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:388: UserWarning: Va
  warnings.warn("Variables are collinear.")
```

Build a regression tree

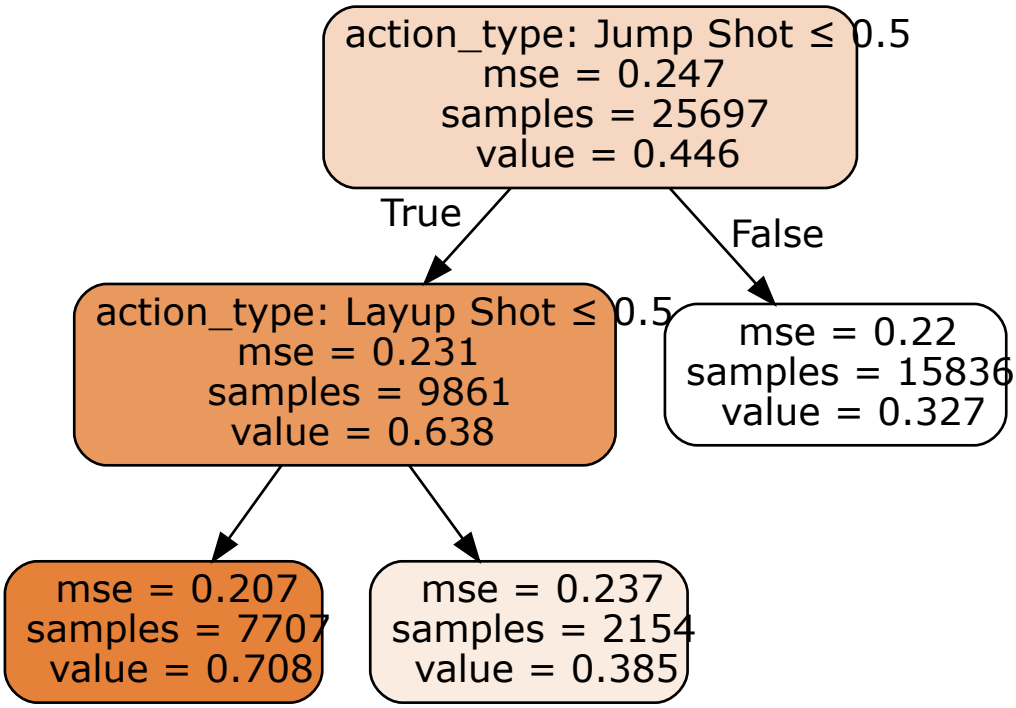
```
In [44]: # Regression tree
         dtr_mid = 0.005
         dtr = DecisionTreeRegressor(min_impurity_decrease = dtr_mid)
         dtr.fit(X0, Y0)
         X0_dtr = dtr.predict(X0)
```

Visualize the regression tree

```
In [45]: # Plot Regression Tree features' importances
         feature_df = pd.DataFrame(dtr.feature_importances_,
                                   index = X_cols,
                                   columns = ["importance"])
         feature_df.sort_values("importance", ascending = False, inplace = True)
         feature_df = feature_df.head(10)
         plt.figure()
         x_axis = np.arange(len(feature_df))
         plt.bar(x_axis, feature_df["importance"])
         plt.xticks(x_axis, feature_df.index, rotation = 'vertical')
         plt.xlabel("Feature")
         plt.ylabel("Regression Tree's Gini Index")
         plt.show()
         dot_data = export_graphviz(dtr, out_file = None, feature_names = X_cols,
                                    filled = True, rounded = True,
                                    special_characters = True)
         graph = graphviz.Source(dot_data, format = 'svg')
         graph
```


Regression Tree's Gini Index
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
action_type: Jump Shot
action_type: Layup Shot
season: 1999-00
shot_zone_range: 16-24 ft.
shot_zone_area: Right Side(R)
shot_zone_basic: Above the Break 3
shot_zone_basic: Backcourt
shot_zone_basic: In The Paint (Non-RA)
shot_zone_basic: Left Corner 3
shot_zone_basic: Mid-Range
Feature

Out[45]:



Filter the features found by RFE, and add the LDA projection and the regression tree's prediction as additional features

```
In [46]: # Filter columns
         # unique is just to not calculate one param multiple times
         X_cols_f = X_cols_rfe
         print('Final chosen features: ', X_cols_f)
         print(len(X_cols_f))
         X0 = X0[X_cols_f]
         X0.loc[:, 'LDA'] = X0_lda                # adding value of a single LDA dim
         X0.loc[:, 'DTR'] = X0_dtr       # adding value of DTR prediction
```

```
Final chosen features:  ['action_type: Driving Finger Roll Layup Shot', 'action_type: Dunk Shot
10
```

1.4 Build the model and test it

Use logistic regression for prediction, and display the cross-validation score

```
In [47]: # Test the model
         lr = LogisticRegression()
         scores = -cross_val_score(lr, X0, Y0, cv = 10, scoring = 'neg_log_loss')
         print('CV log-loss: ', np.mean(scores), '+/-', np.std(scores))
```

```
CV log-loss:  0.603837656249 +/- 0.0100247304665
```

1.5 Generate predictions

Use quick_test = False to handle leakage correctly

```
In [48]: # Set this to True to ignore leakage and get a faster code
         quick_test = True

         # For test only - Leakage problem
         if quick_test:          # (with leakage, quick and dirty)

                 # Fit the model
                 lr = LogisticRegression()
                 lr.fit(X0, Y0)  # trains on training data. Here the flesh lies.

                 # Generate the model's prediction
                 Xlda = lda.transform(evaldata[X_cols])
                 Xdtr = dtr.predict(evaldata[X_cols])
                 X = evaldata[X_cols_f]
                 X.loc[:, 'LDA'] = Xlda
                 X.loc[:, 'DTR'] = Xdtr
                 Y = lr.predict_proba(X)  # without _proba , this would have given 0/1
                 evaldata.loc[:, pred_var] = Y[:, 1]

         # Leakage handle
         else:                 # without leakage

                 c = 0
                 C = len(evaldata)
                 for t in evaldata.index:

                         # Filter only past data
                         ind = traindata.index
                         ind_t = ind[ind < t]
                         if len(ind_t) == 0:                 # if row picked is first
                                 evaldata.loc[t, pred_var] = 0.5     # For the first shot, jus
                                 continue
                         Xt = traindata.loc[ind_t, X_cols_f]
                         Yt = np.ravel(traindata.loc[ind_t, Y_cols])

                         # Fit the model
                         lda = LinearDiscriminantAnalysis()
                         lda.fit(traindata.loc[ind_t, X_cols], Yt)
                         Xt.loc[:, 'LDA'] = lda.transform(traindata.loc[ind_t, X_cols])
                         dtr = DecisionTreeRegressor(min_impurity_decrease = dtr_mid)
                         dtr.fit(traindata.loc[ind_t, X_cols], Yt)
                         Xt.loc[:, 'DTR'] = dtr.predict(traindata.loc[ind_t, X_cols])
                         lr = LogisticRegression()
                         lr.fit(Xt, Yt)  # trains on training data. Here the flesh lies.
```

```
                # Generate the model's prediction
                X = evaldata.loc[[t], X_cols_f]
                # using the LDA axis found, generate and add "LDA value"
                X.loc[:, 'LDA'] = lda.transform(evaldata.loc[[t], X_cols])
                X.loc[:, 'DTR'] = dtr.predict(evaldata.loc[[t], X_cols])
                Y = lr.predict_proba(X)     # without _proba , this would have given
                evaldata.loc[t, pred_var] = Y[0, 1]

                # Display progress
                c = c + 1
                print(t, ': ', c, '/', C)
```

Save the results

```
In [49]: # For future plots
         evaldata.to_csv(fullresultsfilename, header = True, index = False)

         # what we hand-in as output
         preddata = evaldata[pred_cols]
         preddata.to_csv(resultsfilename, header = True, index = False)
```

1.6 Plot results

```
In [55]: # defining court drawings:
         def draw_court(ax = None, color = 'black', lw = 2, outer_lines = False):
                 # If an axes object isn't provided to plot onto, just get current one
                 if ax is None:
                         ax = plt.gca()

                 # Create the various parts of an NBA basketball court

                 # Create the basketball hoop
                 # Diameter of a hoop is 18" so it has a radius of 9", which is a value
                 # 7.5 in our coordinate system
                 hoop = Circle((0, 0), radius = 7.5, linewidth = lw, color = color,
                               fill = False)

                 # Create backboard
                 backboard = Rectangle((-30, -7.5), 60, -1, linewidth = lw, color = color)

                 # The paint
                 # Create the outer box 0f the paint, width=16ft, height=19ft
                 outer_box = Rectangle((-80, -47.5), 160, 190, linewidth = lw, color =
                 color,
                                       fill = False)
                 # Create the inner box of the paint, widt=12ft, height=19ft
                 inner_box = Rectangle((-60, -47.5), 120, 190, linewidth = lw, color =
                 color,
```

```
                        fill = False)

    # Create free throw top arc
    top_free_throw = Arc((0, 142.5), 120, 120, theta1 = 0, theta2 = 180,
                         linewidth = lw, color = color, fill = False)
    # Create free throw bottom arc
    bottom_free_throw = Arc((0, 142.5), 120, 120, theta1 = 180, theta2 = 0,
                            linewidth = lw, color = color, linestyle =
                            'dashed')
    # Restricted Zone, it is an arc with 4ft radius from center of the hoop
    restricted = Arc((0, 0), 80, 80, theta1 = 0, theta2 = 180, linewidth = lw,
                     color = color)

    # Three point line
    # Create the side 3pt lines, they are 14ft long before they begin to arc
    corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth = lw,
                               color = color)
    corner_three_b = Rectangle((220, -47.5), 0, 140, linewidth = lw,
                               color = color)
    # 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
    # I just played around with the theta values until they lined up with the
    # threes
    three_arc = Arc((0, 0), 475, 475, theta1 = 22, theta2 = 158, linewidth =
    lw,
                    color = color)

    # Center Court
    center_outer_arc = Arc((0, 422.5), 120, 120, theta1 = 180, theta2 = 0,
                           linewidth = lw, color = color)
    center_inner_arc = Arc((0, 422.5), 40, 40, theta1 = 180, theta2 = 0,
                           linewidth = lw, color = color)

    # List of the court elements to be plotted onto the axes
    court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                      bottom_free_throw, restricted, corner_three_a,
                      corner_three_b, three_arc, center_outer_arc,
                      center_inner_arc]

    if outer_lines:
            # Draw the half court line, baseline and side out bound lines
            outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth = lw,
                                    color = color, fill = False)
            court_elements.append(outer_lines)

    # Add the court elements onto the axes
    for element in court_elements:
            ax.add_patch(element)
```

```
        return ax

# scatter shots
plt.figure()
draw_court(outer_lines = True)
plt.ylim(-60, 440)
plt.xlim(270, -270)
plt.title('Prediction for test-shots by location')
cm = plt.cm.get_cmap('RdYlBu')
sc = plt.scatter(x = evaldata['loc_x'], y = evaldata['loc_y'],
                 c = evaldata['shot_made_flag'], s = 5, cmap = cm, alpha = 1)
plt.colorbar(sc)

plt.show()
```