# -\*- coding: utf-8 -\*-

"""

Created on Tue Jun 12 22:26:18 2018

@author: James

Predict Player Position from Stats

"""

import pandas as pd

import os

import numpy as np

import matplotlib.pyplot as plt

import datetime

from sklearn.model\_selection import cross\_validate, train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

def load\_nflcombine\_data():

"""

# Load basic player statistics (souce: https://www.kaggle.com/kendallgillies/nflstatistics)

basic\_stats = pd.read\_csv('input\_data/nflstatistics/Basic\_Stats.csv',sep= ',')

basic\_stats = basic\_stats[basic\_stats['Age']<120]

basic\_stats = basic\_stats[basic\_stats['Position'].notnull()]

"""

# Load combine statistics (source: https://www.kaggle.com/kbanta11/nfl-combine)

combine\_stats = pd.DataFrame()

for filename in os.listdir('input\_data/nfl-combine/'):

temp\_df = pd.read\_csv('input\_data/nfl-combine/'+filename,sep= ',')

temp\_df['Side'] = filename[4:len(filename)-4]

combine\_stats = pd.concat([combine\_stats, temp\_df])

# Convert height feature to numeric

combine\_stats['Height']=combine\_stats['Height'].str[0].astype('int64')\*12 + combine\_stats['Height'].str[2:].astype('int64')

combine\_stats = combine\_stats.rename(index=str, columns={"Wt": "Weight"})

return combine\_stats

def split\_test\_training\_sets(df,one\_hot=False):

# Replace missing fields with column averages

df=df.fillna(df.mean())

for col in df.columns[1:]:

if df[col].dtype == "object":

if one\_hot:

# Convert categorical fields into multiple binary fields

temp\_ar = pd.get\_dummies(df[col])

df[list(temp\_ar)] = temp\_ar

else:

# Drop categorical columns

df = df.drop([col],axis=1)

# Spit the data into a training set and test set

return train\_test\_split(df,test\_size=0.33,random\_state=1)

def summary\_stats(df,col):

# Dispaly summary positions by position

grouped = df.groupby(col)

print(grouped.aggregate(np.mean))

print(grouped.size())

def display\_histogram(df):

# Show histograms

for col,typ in df.dtypes.iteritems():

if typ in ['float64','int64']:

plt.figure()

df[col].plot.hist(title=col)

def display\_scatter(df,x,y):

# Show scatter for inputted fields

plt.figure()

plt.scatter(df[x],df[y],color='black')

plt.title(x+" vs "+y)

plt.xlabel(x)

plt.ylabel(y)

# Show Cam Newton's stats

cam = df[df['Player'].str.contains('Cam Newton')]

if len(cam) != 0:

plt.scatter(cam[x],cam[y],color='#0083C9',marker='^',s=500)

def display\_scatter\_pairs(df,features,player):

df = df[features]

axes = pd.tools.plotting.scatter\_matrix(df, alpha=0.2)

for f1 in range(0,len(features)):

for f2 in range(0,len(features)):

if f1 != f2:

plt.sca(axes[f1, f2])

plt.scatter(player[features[f2]],player[features[f1]],color='#0083C9',marker='^',s=150)

plt.tight\_layout()

plt.savefig('scatter\_matrix.png')

def display\_parameter\_curve(df,x,y,x\_title=0,y\_title=0,

grp\_name='', log=False):

if x\_title == 0: x\_title = x

if y\_title == 0: y\_title = y

# Show line plot for inputted fields

if grp\_name == '':

plt.plot(df[x],df[y],color='black')

else:

for grp in df[grp\_name].unique():

df\_grp = df[df[grp\_name]==grp]

plt.plot(df\_grp[x],df\_grp[y],label=grp)

plt.title(x\_title+" vs "+y\_title)

plt.xlabel(x\_title)

plt.ylabel(y\_title)

if log:

plt.xscale('log')

if grp\_name != '':

plt.legend()

plt.show()

def decision\_tree(df,label):

# Split dataset into features and inputted label

y = df[label]

X = df.drop([label],axis=1)

# Train Decison Tree on training data with cross validation

DT = DecisionTreeClassifier()

scores = cross\_validate(DT,X,y,return\_train\_score=True)

print(scores)

return DT

def svm(df,label):

# Split dataset into features and inputted label

y = df[label]

X = df.drop([label],axis=1)

# Train Support Vector Machine on training data with cross validation

SVM = SVC()

scores = cross\_validate(SVM,X,y,return\_train\_score=True)

print(scores)

return SVM

def rf(df,label):

# Split dataset into features and inputted label

y = df[label]

X = df.drop([label],axis=1)

# Train Random Forest on training data with cross validation

RF = RandomForestClassifier()

scores = cross\_validate(RF,X,y,return\_train\_score=True)

print(scores)

return RF

def grid\_search(df,clf,label,params,save=True,verbose=True):

# Split dataset into features and inputted label

y = df[label]

X = df.drop([label],axis=1)

clf\_gs = GridSearchCV(clf, params, return\_train\_score=True)

clf\_gs.fit(X,y)

if verbose:

print(clf\_gs.best\_score\_)

print(clf\_gs.best\_params\_)

df\_cv = pd.DataFrame.from\_dict(clf\_gs.cv\_results\_)

if save:

df\_cv.to\_csv('output\_data/grid\_result\_'+datetime.datetime.now().strftime("%Y%m%d%H%M%S")+'.csv')

return df\_cv

def save\_confusion\_matrix(cm,label,name):

cm = np.concatenate((np.resize(label,(len(label),1)),cm),axis=1)

label = np.insert(label,0,'')

cm = np.concatenate((np.resize(label,(1,len(label))),cm),axis=0)

cm = cm.astype('str')

np.savetxt(name+".csv", cm, delimiter=",", fmt="%s")

def main():

return

if \_\_name\_\_ == "\_\_main\_\_":

main()