Math 462 Final Report: NBA Game Prediction

Yucheng Shi

April 16, 2018

1 Introduction

NBA Basketball Association is the men's professional basketball league in North America. According to `www.statista.com`, the total NBA league revenue of 2015-2016 season is around 5.87 billion. As a multi-billion dollar industry, there is no surprise a lot of studies have been developed on NBA games.

Though a lot of data and statistics have been collected, it is still very difficult to analyze and predict a game. In order to achieve good prediction accuracy, some machine learning methods are used over NBA box score data.

The main objective of this project is to achieve a good prediction accuracy using machine learning methods. In addition, I would also want to what are the important features to be a winner. This would be meaningful as coaches and players would be able to know what are the key factors to win a game.

2 Background

NBA has 30 teams (29 in the U.S. and 1 in Canada). The 30 teams are divided into two conferences of three divisions with five teams each.

Eastern Conference			Western Conference		
Atlantic	Central	Southeast	Northwest	Pacific	Southwest

Table 1: NBA divisions

During the regular season, each team plays 82 games, half home and half away. Each team plays the other teams in its own division four times and plays six of the teams from the other two divisions in its own conference four times, and the remaining four teams three times. Also, each team plays all the teams from the other conference two times.

NBA Playoffs involve only eight teams in each conference. The three division winners and the team with the next best record from the fonference are given the top four seeds. The next four teams of best record are given the lower four seeds. The playoffs are then of tournament format.

There are many statistics for a game, such as `teamRslt`, `teamPTS`, `teamAST`, `teamTO` and so on. The meaning of these statistics are explained in the appendix.

3 Methodology

Problem Definition

For each regular season NBA game in 2017-2018 season, there is a list of data related to this game, as explained in the Background. Given all these data from previous seasons, we want to predict the winner of a game without looking directly at `teamRslt`.

Modeling Assumptions

For each team, we are going to assume that the lineups are consistent throughout a season. This is a simplization of the whole model in the sense that "player-level" data are used as features of classification. The assumption is also why I want to train the classifier only on one-season data.

Model

The problem is a classical classification problem. We experiment with four machine learning binary classification methods: i)*k-nearest Neighbor*, ii) *SVM (Support Vector Machines)*, iii) *Random Forest*, iv) *Gradient Tree Boosting.* After choosing the best classification method (with highest accuracy in predicting 2016-2017 season games), the classification method is used for the prediction of 2017-2018 season games. When it is possible to do cross validation, 10-fold cross validation is used.

k-nearest Neighbor. k-nearest neighbor classification is one of the most fundamental classification methods. It is a classification method based on the Euclidean distance between test case and training cases. Say we have n input cases $x_i(i = 1, 2, \ldots, n)$ characterized by p features $(x_{i1}, x_{i2}, \ldots, x_{ip}$. The Euclidean distance between case x_i and x_j is

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{p}(x_{ik} - x_{jk})}.$$

For a test case x, let ω be its true class and $\hat{\omega}$ be the predicted class for x. The predicted class of x, *i.e.,* $\hat{\omega}$ is set to be the most frequent true class among k nearest training samples.

Support Vector Machine. Support Vector Machine is a non-probabilistic binary linear classifier. While a linear classifier considers all points while minimizing the mean square error, the classifier would not be good if we are not provided sufficient training data. A solution to this is that instead of minimizing the MSE, maximizing the margin of classification. SVM does just that. The advantage of SVM is that SVM focuses on training points that define the decision boundary (such points are call support vectors). Also, "kernel trick" provides us a method of mapping original space to a higher dimensional space such that the points are linearly separable. Linear kernel, polynomial kernel are tested, but for the data we are using, linear kernel has the higher accuracy.

Random Forests. Random Forests work by constructing many decision trees when we are training and output the mode of the classes of the individual trees. The advantage of decision tree is that it is simple to understand and implement. But "deep" trees tend to "overfit" their training sets. Random forests is a way to average multiple deep decision trees that are trained on different parts of the same training set so that the variance is reduced.

Gradient Tree Boosting. Gradient tree boosting uses decision trees of fixed size as weak learners. A loss function is developed according to the problem we are looking at (in this project logarithmic loss function is used). Also, an additive model is used to add weak learners to minimize the loss function. In particular, a gradient descent procedure is used to minimize the loss function. To perform the gradient descent procedure, a tree is added to the model to reduce the loss (follow the gradient). The output is then added to the current forests to impove the final output.

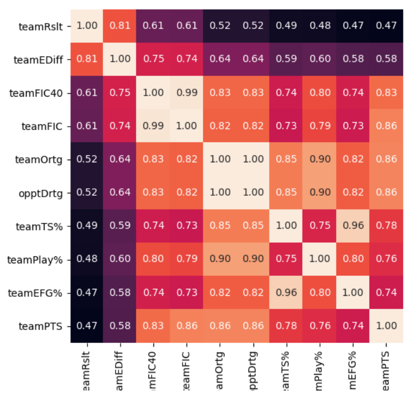
Codes that do the classification are enclosed in appendix.

Data Preparation

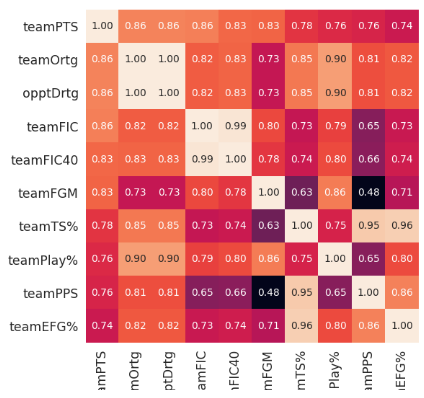
Dataset for this project is downloaded from `https://www.kaggle.com/pablote/nba-enhanced-stats`. Also, a lot of time is devoted to validating these data using data scraped from other source (ESPN). The scraper can be found in appendix.

Feature Design

Feature Design is probably the most important part of Machine Learning classification based model. To take a first glance of what could be good features, a 10-largest correlation matrix with respect to `teamRslt` and `teamPTS` is plotted as heatmap. Other correlation matrices with



(a) Correlation matrix of parameters correlated to `teamRslt`



(b) Correlation matrix of parameters correlated to `teamPTS`

respect to suspiciously ”good” features in hope to find other less correlated suspiciously ”good” features. After examining parameters (*garbage in, garbage out*), I pick the following features

- `teamDrtg` (advanced statistics): defensive rating (An estimate of points allowed per 100 possessions).
- `teamDREB` (advanced statistics): team defensive rebounds.
- `teamFGA`: field goals attempted.
- `opptPTS`: opponent points.
- `opptAST`: opponent total assists.
- `teamBLK`: team total blocks

4 Results & Analysis

Results

The Python script used to model can be found in appendix. The result for 2016-2017 regular season is as follows. When running the code, please do not change the random state by default. `save-model.py` saves the classifier for future use and analysis.

Classifier	**Accuracy**	**2017-2018**
KNN	72%	60%
Random Forests	94%	63%
SVM	58%	52%
GTB	73%	66%

Table 2: Accuracy of each classifiers on 2016-2017 season compared to 2017-2018 season (Default random state)

Analysis

1. As we can see from the table, random forest has a "significant" high accuracy in predicting 2016-2017 season but a relatively low accuracy in predicting 2017-2018 season. This is probably the double effects of the overfitting problem of decision trees and insufficient data. To solve this issue, we can choose a threshold to accept random forest as the classifier. Or we can simply train these classifiers on much more data (more seasons). This is a little bit contradiction with our assumption because when more seasons data involve, our assumption that the roster would not change doesn't make sense anymore.
2. The accuracy of support vector machine is low compared to other 3 classifications method. This is unexpected after testing multiple kernel. Possible reason is that NBA games could be really really close.

5 Conclusion

After examining the mutual information of features, the several dominant features are as follows. The most dominant feature is `opptPTS` (opponent total points). This is expected because there is a strong correlation between the number of points that a team's opponent gets and the number of games that a team will win in the future. The next several dominant features are `teamDrtg`, `opptDREB` and `opptAST`. This is surprising because these features (including `opptPTS`) are all defensive statistics that characterize how well a team defenses their opponent. This says in order to win a game, it is more important to prevent the opposing team from playing well than simply playing well. This conclusion coincides with the point of legendary football coach Paul "Bear" Bryant, "Offense sells tickets. Defense wins championships."

6 Future Works

1. Since most of the work is done by scripts, it won't be difficult to modify the scripts for future use. A script that automatically simulates a full season would be great.

2. To predict the outcome of a game, only statistics are involved. That is to say, we can group arbitrary players as a team, input this team to the model and see if this team works out.

3. To increase accuracy in the middle of a season, up-to-date data can be used in training.

A Basketball Statistics Abbreviations

This explains some of crucial abbreviations in raw data.

- **PTS**: points
- **FGM, FGA, FG%**: field goals made, attempted and percentage
- **FTM, FTA, FT%**: free throws made, attempted and percentage
- **REB, OREB, DREB**: rebounds, offensive rebounds, defensive rebounds
- **AST**: assists
- **STL**: steals
- **BLK**: blocks
- **Pace**: Possessions per game
- **PPP**: Points per possession
- **TO%**: Turnover percentage

B Codes

B.1 Scrapers

B.1.1 team data

```
import pandas
from pandas import *
import collections
import numpy
import requests
from bs4 import BeautifulSoup
import csv

url = 'http://espn.go.com/nba/teams'
r = requests.get(url)

soup = BeautifulSoup(r.text)
tables = soup.find_all('ul', class_='medium-logos')

teams = []
prefix_1 = []
prefix_2 = []
teams_urls = []
for table in tables:
    lis = table.find_all('li')
    for li in lis:
        info = li.h5.a
        teams.append(info.text)
        url = info['href']
        teams_urls.append(url)
        prefix_1.append(url.split('/')[-2])
        prefix_2.append(url.split('/')[-1])


dic = {'url': teams_urls, 'prefix_2': prefix_2, 'prefix_1': prefix_1}
teams = pandas.DataFrame(dic, index=teams)
teams.index.name = 'team'
print(teams)
teams.to_csv('team.csv')
```

B.1.2 raw team stat

```
import numpy as np
import pandas as pd
import requests
from bs4 import BeautifulSoup
from datetime import datetime, date

teams = pd.read_csv('team.csv')
BASE_URL = 'http://espn.go.com/nba/team/schedule/_/name/{0}/year/{1}/seasontype/2'


log = open('errorlog.log','w')
match_id = []
dates = []
home_team = []
home_team_score = []
visit_team = []
visit_team_score = []
for year in [2016,2017]:
    for index, row in teams.iterrows():
        _team = row['prefix_1']
        print year, _team
        url = BASE_URL.format(_team, str(year))
        r = requests.get(url)
        table = BeautifulSoup(r.text).table
        for row in table.find_all('tr')[1:]: # Remove header
            columns = row.find_all('td')
            try:
                _home = True if columns[1].li.text == 'vs' else False
                _other_team = columns[1].find_all('a')[1]['href'].split('/')[7]
                _score = columns[2].a.text.split(' ')[0].split('-')
                _won = True if columns[2].span.text == 'W' else False
                home_team.append(_team if _home else _other_team)
                visit_team.append(_team if not _home else _other_team)
                try:
                    d = datetime.strptime(columns[0].text, '%a, %b %d')
                    dates.append(date(year, d.month, d.day))
                except:
                    dates.append(date(year, 2, 29))
                if _home:
                    if _won:
                        home_team_score.append(_score[0])
                        visit_team_score.append(_score[1])
                    else:
                        home_team_score.append(_score[1])
                        visit_team_score.append(_score[0])
                else:
                    if _won:
                        home_team_score.append(_score[1])
                        visit_team_score.append(_score[0])
                    else:
                        home_team_score.append(_score[0])
                        visit_team_score.append(_score[1])
                match_id.append(columns[2].a['href'].split('?id=')[1])
            except Exception as e:
                print >> log, e, columns
                pass # Not all columns row are a match, is OK


log.close()
dic = {'match_id': match_id, 'date': dates, 'home_team': home_team, 'visit_team': visit_
        'home_team_score': home_team_score, 'visit_team_score': visit_team_score}


games = pd.DataFrame(dic)
games.set_index('match_id')
games = games.drop_duplicates(cols='match_id').set_index('match_id')
#print(games)
games.to_csv('games_2016_2017.csv')
```

B.1.3 Processed data

```
import numpy as np
import pandas as pd
import requests
from bs4 import BeautifulSoup
from datetime import datetime, date

games = pd.read_csv('games_2016_2017.csv')
games.set_index('match_id')
BASE_URL = 'http://espn.go.com/nba/boxscore?gameId={0}'

request = requests.get(BASE_URL.format(games['match_id'][0]))

table = BeautifulSoup(request.text).find('table', class_='mod-data')
heads = table.find_all('thead')
headers = heads[0].find_all('tr')[1].find_all('th')[1:]
headers = [th.text for th in headers]
columns = ['match_id', 'team', 'playerId'] + headers
players = pd.DataFrame(columns=columns)

def get_players(players, team_name, matchId):
    array = np.zeros((len(players), len(headers)+1), dtype=object)
    array[:] = np.nan
    for i, player in enumerate(players):
        cols = player.find_all('td')
        if cols[1].text.startswith('DNP') or cols[1].text.startswith('NWT'):
            continue
        array[i, 0] = cols[0].a['href'].split('/')[7]
        for j in range(1, len(headers) + 1):
            array[i, j] = cols[j].text
    frame = pd.DataFrame(columns=columns)
    for x in array:
        line = np.concatenate(([matchId, team_name], x)).reshape(1,len(columns))
        new = pd.DataFrame(line, columns=frame.columns)
        frame = frame.append(new)
    return frame

for index, row in games.iterrows():
    print(index), row['match_id']
    request = requests.get(BASE_URL.format(row['match_id']))
    bodies = BeautifulSoup(request.text).find_all('tbody')
    team_1 = 'visit'
    team_1_players = bodies[0].find_all('tr') + bodies[1].find_all('tr')
    team_1_players = get_players(team_1_players, team_1, row['match_id'])
    players = players.append(team_1_players)
    team_2 = 'home'
    team_2_players = bodies[3].find_all('tr') + bodies[4].find_all('tr')
    team_2_players = get_players(team_2_players, team_2, row['match_id'])
    players = players.append(team_2_players)


players = players.set_index(['match_id'])
players.to_csv('gameDetail_2016_2017.csv')
```

B.1.4 Training Script

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import pickle

from sklearn import svm
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics

def NBA_Game_Prediction(filename = '../Data/2016-17_officialBoxScore.csv', testsize = 0.
        """The funciton takes inputs:
        'filename' is the name of the file containing data.
        'testsize' specifies the protion of data that would be used for training.
        'randomstate' is the random seed used to split data.
```

```
    'feature_cols' is the feature plan to be used.
    'option' specifies classifier that will be used. It can be 1 to 5, with 1 for k
    'save' is the indicator of saving results.
    """
    print('')
    data = pd.read_csv(filename)
    x = data[feature_cols]
    y = data['teamRslt']
    x_train, x_test, y_train, y_test = train_test_split(x, y , test_size=testsize,
    if (option == 1):
        print("n-nearest␣neighbor␣classification␣is␣used,␣score␣is␣%.4f" % KNei
    elif (option == 2):
        print('support␣vector␣machine␣classification␣is␣used,␣score␣is␣%.4f' % (
    elif (option == 3):
        print('random␣forest␣classification␣is␣used,␣score␣is␣%.4f' % RandomFor
    elif (option == 4):
        print('gradient␣tree␣boosting␣classification␣is␣used,␣score␣is␣%.4f' %
    elif (option == 5):
        print('n-nearest␣neighbor␣classification␣is␣used,␣score␣is␣%.4f'% KNeig
        print('support␣vector␣machine␣classification␣is␣used,␣score␣is␣%.4f' %
        print('random␣forest␣classification␣is␣used,␣score␣is␣%.4f' % RandomFor
        print('gradient␣tree␣boosting␣classification␣is␣used,␣score␣is␣%.4f' %


def KNeighbors_(x_train, y_train , x_test, y_test, save = False):
    clf = KNeighborsClassifier(n_neighbors = 4)
    clf.fit(x_train, y_train)
    if (save):
        pickle.dump(clf, open('KNN-model.s', 'wb'))
    return metrics.accuracy_score(y_test, clf.predict(x_test))

def SVM_(x_train, y_train, x_test, y_test, save = False):
    clf = LinearSVC(random_state = 3)
    clf.fit(x_train, y_train)
    if (save):
        pickle.dump(clf, open('SVM-model.s', 'wb'))
    return metrics.accuracy_score(y_test, clf.predict(x_test))

def RandomForest_(x_train, y_train, x_test, y_test, save = False):
    clf = RandomForestClassifier()
    clf.fit(x_train, y_train)
    if (save):
        pickle.dump(clf, open('RandomForest-model.s', 'wb'))
    return metrics.accuracy_score(y_test, clf.predict(x_test))

def GradientBoosting_(x_train, y_train, x_test, y_test, save = False):
    clf = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0, max_d
    clf.fit(x_train, y_train)
    if (save):
        pickle.dump(clf, open('GTB-model.s', 'wb'))
    return clf.score(x_test, y_test)
```

```
if __name__ == "__main__":
        NBA_Game_Prediction()
```

References

[1] Grant Avalon; Batuhan Balci; Jesus Guzman Various Machine Learning Approaches to Predicting NBA Margins

[2] Renato Amorim Torres Prediction of NBA games based on Machine Learning Methods

[3] Wikipedia. Basketball statistics

[4] Basketball reference `www.basketball-reference.com`

[5] Piush Vaish `https://github.com/piushvaish/pythonTutorials/blob/master/pythonDataAnalysis/Projects/Predicting%20NBA%202017/NBA_2017.ipynb`

[6] Richardson Lee `https://github.com/leerichardson/game_simulation/blob/master/final_paper.pdf`

[7] Matthew Beckler `https://www.mbeckler.org/coursework/2008-2009/10701_report.pdf`