CRICTO-PAUL

www.crictopaul.com

Final Report

Group Members:

1) Soham Urmish Mehta	111496015
2) Sai Prasanth Gumpalli	111480980
3) Venkata Kedarnath Pakala	111014006

1. INTRODUCTION

This project aims to whether it is possible to find the outcome of a cricket match and predict a winner based on past data and given features. It aims to predict the result of a future match similar to Octopus Paul but in a much informed and sophisticated manner. Not only is prediction made at the start of the match, but it is also made at the passing of each and every over of the match. This gives a more accurate prediction taking into consideration the events in a given match.

A vast amount of ball-by-ball data of every match played since 1800's is available online for analysis. This data can be combined to find the batting, balling, fielding averages of each team in a given year. Also, the ranking of players in each team can help in identifying the team's performance during that match. Other features include knowing whether match is being played at home ground or away, the past performance of a team in that particular ground and the net run rate of a team against the competitor. These all parameters can be combined to identify the current strength of a given team and its odds to win in a given scenario.

2. FINDING RELEVNT FEATUES IN DATA

After the data is obtained in the csv format, it is read inside the python notebook. Some of the features can be directly used for analysis and model training while some more features have to be computed to as to make sense of the data, Firstly, we convert the player dismissed into binary format so that the model can use it for training. Then, we calculate total runs and total wickets per over. We added new features of innings_wickets and inning_score which can be used to compute number of remaining score and wickets. Next, we computed remaining target which has impact on the outcome of match. More remaining target could decrease chances of winning. Finally, we computed the required run rate, run rate difference which indicate how far is the batting team from the winning. Higher the required run rate or run rate difference lower the chances of winning.

Some of the major features used for our final prediction model are:

1) Total runs
2) Batting team
3) Player dismissed
4) Innings wickets
5) Innings Score
6) Score target
7) Remaining Target
8) Run rate
9) Required run rate

Data Processing

The data obtaind from cricsheet is computed into a single csv. Now, we need to find the relevant features on which we can train our model. Also, the irrelevant features are droppped or else not considered into the train dataset

In [ ]:
```
import operator
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import math
import collections
import seaborn as sns
%matplotlib inline

pd.set_option('display.max_columns', 50)
```

In [331]:
```
data= pd.read_csv("/Users/soham/Desktop/Courses/DSF/Project/t20.csv")
data.drop("season", axis=1, inplace=True)
data.shape
```

Out[331]: (132837, 13)

In [332]:
```
#Converting the Player Dismissed into a binary format (categorical into Integer)

data.player_dismissed.fillna(0, inplace=True)
data['player_dismissed'].loc[data['player_dismissed'] != 0] = 1


#Storing the relevant coloumns into a different dataset and aggegating the terms in an over format so that we
 can have
#the relevant features such as Total Runs in the over, Run rate and required Run rate to win the match.

train = data.groupby(['match_id', 'inning', 'over', 'team1', 'team2', 'batting_team', 'winner'])[['total_runs'
, 'player_dismissed']].agg(['sum']).reset_index()
train.columns = train.columns.get_level_values(0)

train.shape
train.sample(n=10)
# train[train.match_id == 1 ]

#This is the basic information which we require to make an analysis for any match
#It has information about both teams, runs and wickets in each over
```

Out[332]:

	match_id	inning	over	team1	team2	batting_team	winner	total_runs	player_dismissed
3981	111	1	1	Sri Lanka	New Zealand	Sri Lanka	New Zealand	2	0
6554	180	2	18	Australia	South Africa	South Africa	South Africa	18	0
1167	33	2	10	New Zealand	South Africa	South Africa	South Africa	5	0
9694	269	1	20	New Zealand	England	New Zealand	England	10	2
6470	178	2	14	India	West Indies	West Indies	India	4	0
2963	83	1	14	England	India	England	England	4	0
14570	400	2	17	United Arab Emirates	Afghanistan	Afghanistan	Afghanistan	16	0
15123	415	2	9	Netherlands	Nepal	Nepal	Netherlands	2	1
12682	350	2	7	West Indies	Bangladesh	Bangladesh	West Indies	6	0
18281	500	2	14	India	Sri Lanka	Sri Lanka	Sri Lanka	7	0

In [335]:
```
'''
One of the most important feature in cricket is to identify the Net Run Rate and Required Run Rate.
These parameters give the batsman understanding of how many runs are required to be scored per over to win
The bowlers on the other hand try to defend and prevent the batting team to hit that many runs in an over

Net Run Rate= Total RUNS scored / Total OVERS played

Required Run Rate= Total RUNS Remaining/ Total OVERS remaining

'''

train['run_rate'] = train['innings_score'] / train['over']

def get_required_rr(row):
    if row['remaining_target'] == -1:
        return -1.
    elif row['over'] == 20:
        return 99
    else:
        return row['remaining_target'] / (20-row['over'])

train['required_run_rate'] = train.apply(lambda row: get_required_rr(row), axis=1)

def get_rr_diff(row):
    if row['inning'] == 1:
        return -1
    else:
        return row['run_rate'] - row['required_run_rate']

#For training the model, we also compute the difference between the net run rate and the required run rate at
 instant
train['runrate_diff'] = train.apply(lambda row: get_rr_diff(row), axis=1)
train['is_batting_team'] = (train['team1'] == train['batting_team']).astype('int')

#To train our model, we require a target. For that, we keep the Winning and Losing with reference to Team1
# This means that if Team1 is winning then the column 'Target' is 1 (Indicating Win) and
# if the Team1 is losing, we keep the 'Target' as 0 (Indicating Loss)
train['target'] = (train['team1'] == train['winner']).astype('int')

train.sample(n=10)
```

Out[335]:

ns	player_dismissed	innings_wickets	innings_score	score_target	remaining_target	run_rate	required_run_rate	runrate_diff	is_batting
	0	1	36	84.0	48.0	4.000000	4.363636	-0.363636	0
	0	1	102	109.0	7.0	11.333333	0.636364	10.696970	0
	0	1	13	145.0	132.0	6.500000	7.333333	-0.833333	0
	0	0	13	149.0	136.0	4.333333	8.000000	-3.666667	0
	0	3	40	-1.0	-1.0	4.444444	-1.000000	-1.000000	1
	0	3	123	141.0	18.0	8.200000	3.600000	4.600000	0
	1	2	13	-1.0	-1.0	3.250000	-1.000000	-1.000000	1
	0	0	50	-1.0	-1.0	10.000000	-1.000000	-1.000000	1
	0	6	156	189.0	33.0	8.210526	33.000000	-24.789474	0
	0	1	20	191.0	171.0	6.666667	10.058824	-3.392157	0

Improving Features

In [154]:
```
x_cols = ['inning', 'over', 'total_runs', 'player_dismissed', 'innings_wickets', 'innings_score', 'sco
re_target', 'remaining_target', 'run_rate', 'required_run_rate', 'runrate_diff', 'is_batting_team']

# let us take all the matches but for the final as development sample and final as val sample #
val_df = train_df.ix[train_df.match_id == 411,:]
dev_df = train_df.ix[train_df.match_id != 411,:]

# create the input and target variables #
dev_X = np.array(dev_df[x_cols[:]])
dev_y = np.array(dev_df['target'])
val_X = np.array(val_df[x_cols[:]])#[:-1,:]
val_y = np.array(val_df['target'])#[:-1]
print(dev_X.shape, dev_y.shape)
print(val_X.shape, val_y.shape)
```

```
(21088, 12) (21088,)
(37, 12) (37,)
```

3. OBSERVATIONS

Since we now have a cleaned dataset of all the matches played in the cricket history, we can surely find some important insights. This is itself a part of being able to understand what all features have a high weight while considering them for training our model. One such observation we found is that the team that wins the toss has much higher chance of winning the match despite the fact that it has no significant impact on overall performance. More such relevant insights can be found in the further pages.

OBSERVATIONS

In [72]:
```
import operator
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, explained_variance_score, r2_score
import math
%matplotlib inline

pd.set_option('display.max_columns', 50)
```

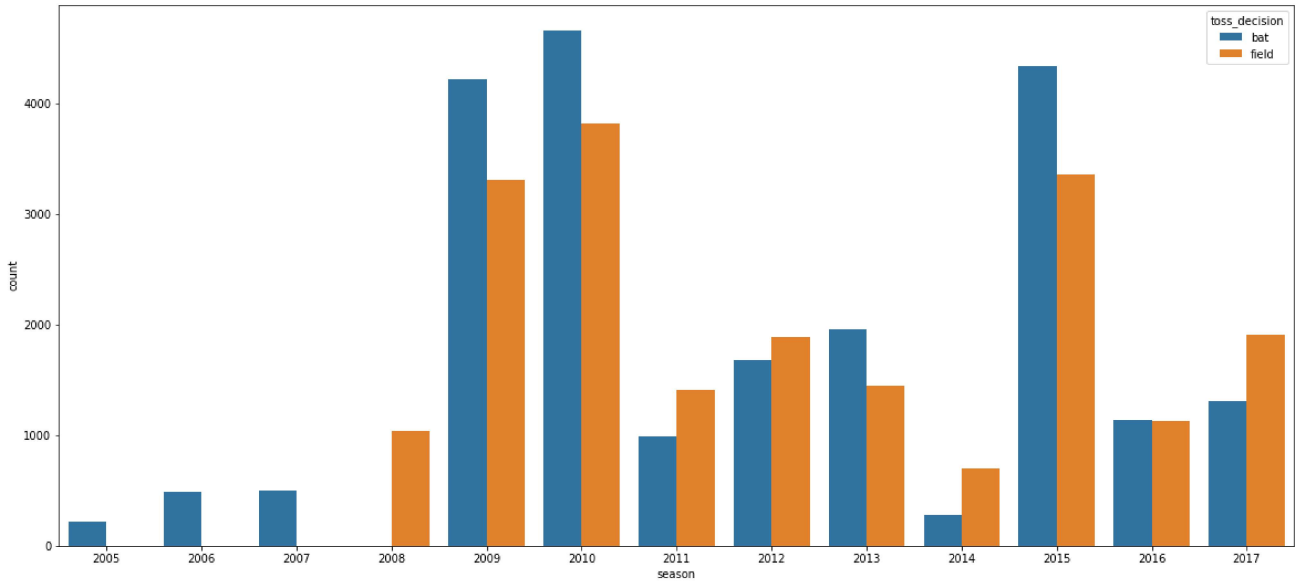
In [4]:
```
balls_df=pd.read_csv('output_with_toss.csv')
```

In [77]:
```
balls_df1=balls_df[~balls_df.season.str.contains('/')]
#balls_df1= balls_df1[ balls_df1.season.str not in ['2008','2005','2007']]
```

The Result of Toss Decision Taken every year

The following graph shows the toss decisions per year. The blue bars show the decision is toss winner has chosen to bat first and orange bar shows the toss decision is fielding. The graph indicates that teams mostly select batting first when they win the toss.

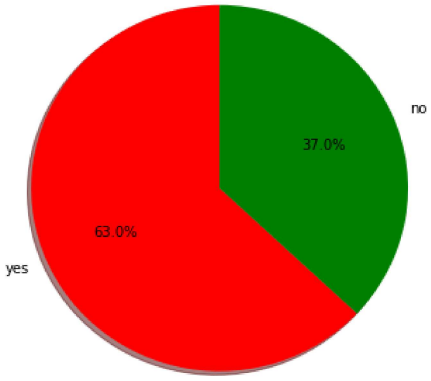
In [19]:
```
plt.subplots(figsize=(20,9))
sns.countplot(x='season',hue='toss_decision',data=balls_df1)
plt.show()
```



In [23]:
```
matches_played_byteams=pd.concat([balls_df['team1'],balls_df['team2']])
matches_played_byteams=matches_played_byteams.value_counts().reset_index()
matches_played_byteams.columns=['Team','Total Matches']
matches_played_byteams['wins']=balls_df['winner'].value_counts().reset_index()['winner']
matches_played_byteams.set_index('Team',inplace=True)
```

Toss Winner could be Match Winner

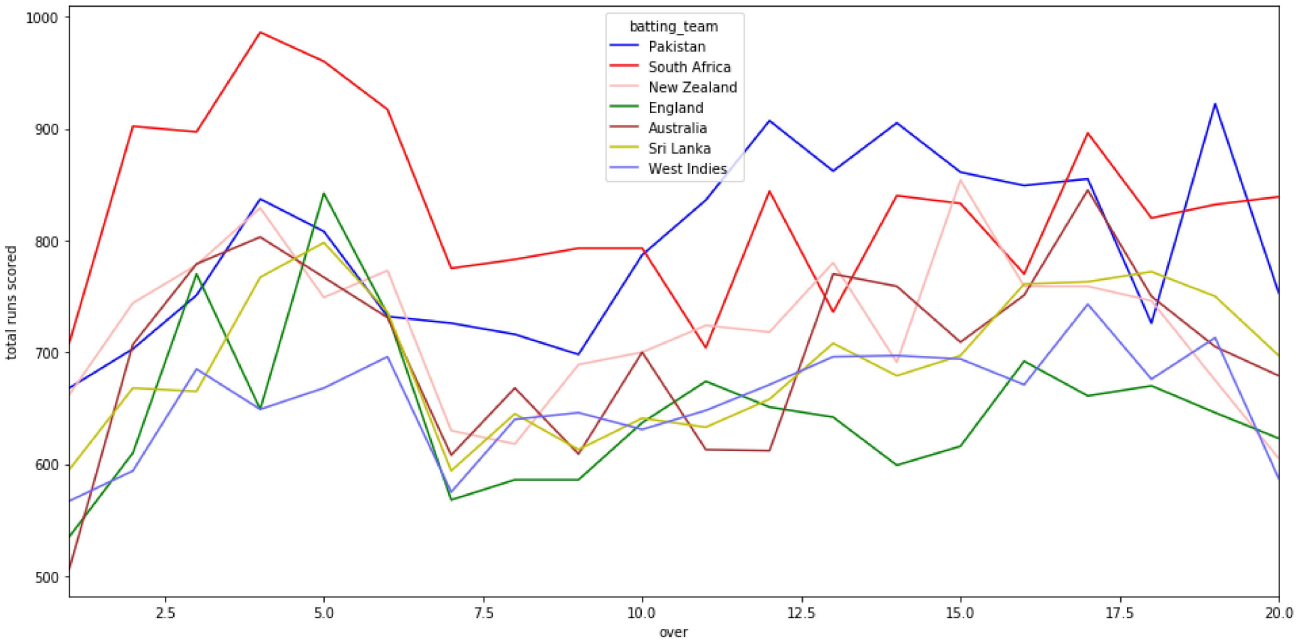
In [85]:
```
df=balls_df[balls_df['toss_winner']==balls_df['winner']]
slices=[len(df),(576-len(df))]
labels=['yes','no']
plt.pie(slices,labels=labels,startangle=90,shadow=True,explode=(0,0),autopct='%1.1f%%',colors=['r','g'])
fig = plt.gcf()
fig.set_size_inches(6,6)
plt.show()
```



Runs per over by each team

Below graph shows the number of runs per over by each team across years which played more than 1000 matches.

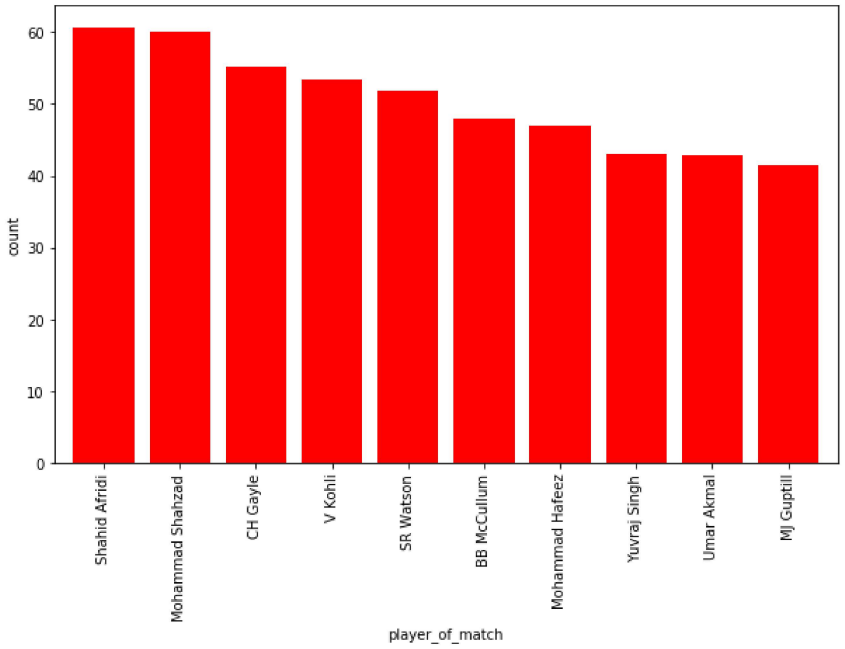
In [75]:
```
runs_per_over = balls_df.pivot_table(index=['over'],columns='batting_team',values='total_runs',aggfunc=sum)
runs_per_over[(matches_played_byteams[matches_played_byteams['Total Matches']>1000].index)].plot(color=["b", "r", "#Ffb6b
2", "g","brown','y','#6666ff','black','#FFA500']) #plotting graphs for teams that have played more than 100 matches
#x=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
#plt.xticks(x)
plt.ylabel('total runs scored')
fig=plt.gcf()
fig.set_size_inches(16,8)
plt.show()
```



Player of the match

Below graph shows the top 10 players with maximum number of player of the match awards.Shaid Afridi of Pakistan has maximum number of player of the match awards as indicated by the graph. Players with maximum number of awards are asset and can be deciding factor in the match's outcome

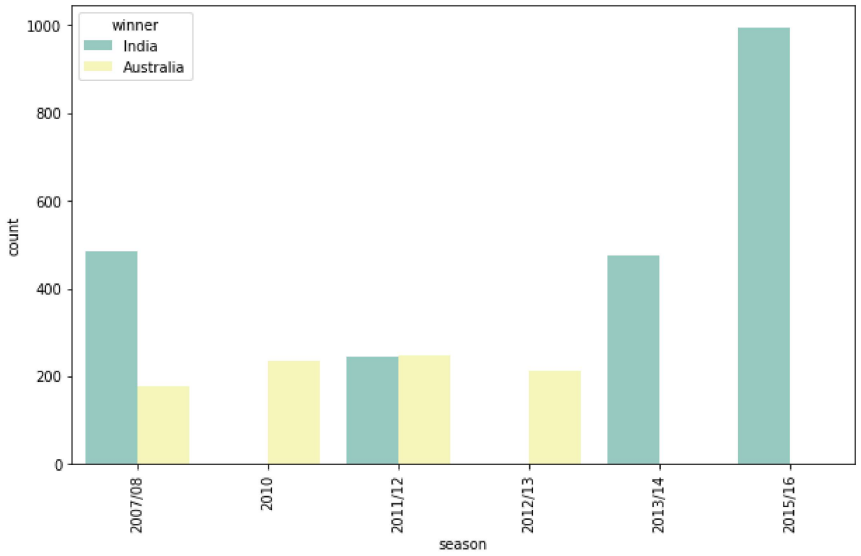
In [58]:
```
plt.subplots(figsize=(10,6))
#the code used is very basic but gets the job done easily

counts=balls_df['player_of_match'].value_counts()/40
ax=counts.head(10).plot.bar(width=.8, color='R')  #counts the values corresponding
# to each batsman and then filters out the top 10 batsman and then plots a bargraph
ax.set_xlabel('player_of_match')
ax.set_ylabel('count')
#for p in ax.patches:
#    ax.annotate(format(p.get_height()), (p.get_x()+0.15, p.get_height()+0.25))
plt.show()
```



Team Pair Stats

Below graph shows the dominance of a team over other per year in the matches they play against each other. The graph indicates that India started dominating the India-Australia matches from 2013

In [76]:
```
def team1_vs_team2(team1,team2):
    mt1=balls_df[((balls_df['team1']==team1)|(balls_df['team2']==team1))&((balls_df['team1']==team2)|(balls_df['team2']==
team2))]
    sns.countplot(x='season', hue='winner',data=mt1,palette='Set3')
    plt.xticks(rotation='vertical')
    #leg = plt.legend( loc = 'upper center')
    fig=plt.gcf()
    fig.set_size_inches(10,6)
    plt.show()
team1_vs_team2('India','Australia')
```



4. MAKING ADVANCED MODELS

4.1 XG Boost Classifier

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way.

For our problem we have selected the Objective function as Binary logistic. The maximum depth is selected as 8 and the evaluation metric is AUC(Area under Cover). The subsample is selected to be 0.7

4.2 Neural Network

An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. We have taken 5 hidden layers and random state as 1. The initial learning rate is selected to be $1e^{-9}$ and maximum iterations as 100.

4.3 Decision Tree Classifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The hyper parameters of our decision tree are selected in such a way where the Maximum depth is 4. The criterion is selected to be Entropy.

4.4 AdaBoost

Abstract Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. In our case, the maximum depth is 4 and number of estimators is selected to be 300.

5. EVALUATING THE MODELS

All the models that are implemented are evaluated on the basis of the parameters given in the table below. Apart from these generic measure, we have also devised our own evaluation function. Each of the model is evaluated with respect to the number of times in a given match it provides a correct output in consistency with the final result. The code and implementation of this evaluation function can be found in the further pages.

Model	Root Mean Squared Error	Mean Absolute Error	Expected Variance Score	R2 Score
Logistic Regression	0.242631	0.492562	0.423984	0.185362
XGBoost	0.148337	0.346834	0.293783	0.119835
Neural Network	0.234976	0.478323	0.420367	0.167847
Decision Tree	0.142209	0.330522	0.385243	0.198237
AdaBoost	0.203145	0.399333	0.337838	0.258734

Logistic Regression

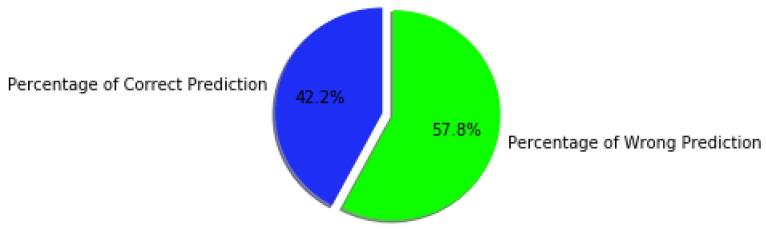
In [233]:
```
logistic = LogisticRegression()
logistic.fit(dev_X,dev_y)
preds= logistic.predict_proba(val_X)
#print(preds)
```

Finding the evaluation function for each model.

This is done by running by training the model for each and every match individually by using k-fold method In each iteration, the accuracy of the prediction is calculated by counting the number of times the model predicts the winning team to win

This value is consolidated for all the matches to find the total evaluation percentage

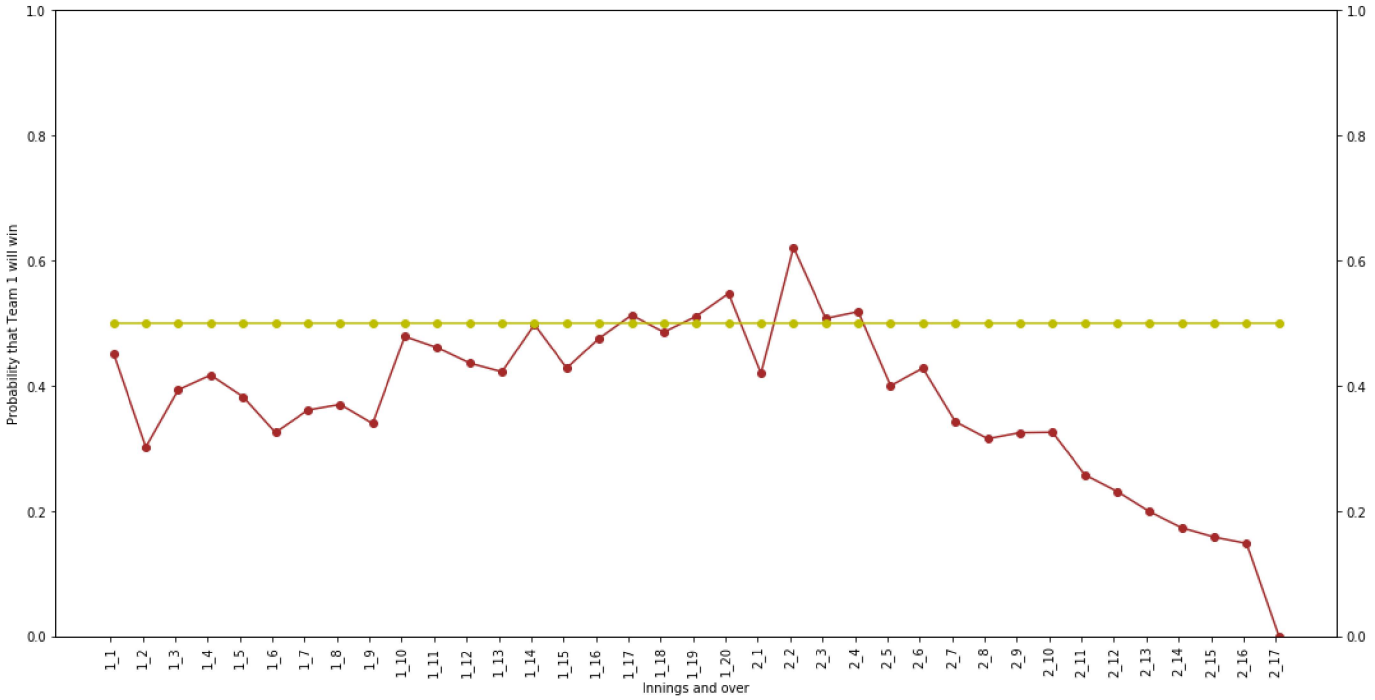
In [187]:
```
ids=train_df.match_id.unique()
logistic = LogisticRegression()
cp=[]
ap=[]
c=0
#ids=np.array([411])
for id in ids:
    val_df = train_df.ix[train_df.match_id == id,:]
    dev_df = train_df.ix[train_df.match_id != id,:]
    dev_X = np.array(dev_df[x_cols[:]])
    dev_y = np.array(dev_df['target'])
    val_X = np.array(val_df[x_cols[:]])#[:-1,:]
    val_y = np.array(val_df['target'])#[:-1]
    logistic.fit(dev_X,dev_y)
    pred1=logistic.predict_proba(val_X)
    if(val_y[0]==0):
        c=c+list(pred1[:,0]>=0.5).count(True)
    else:
        c=c+list(pred1[:,0]<0.5).count(True)
    np_pred1=np.array(pred1)

ans = c/(ids.shape[0]*40)*100


#Plotting the Evaluation Function Result
slices=[ans,100-ans]
list(slices)
labels=['Percentage of Correct Prediction','Percentage of Wrong Prediction']
plt.pie(slices,labels=labels,colors=['#1f2ff3', '#0fff00'],startangle=90,shadow=True,explode=(0,0.1),autopct='%
1.1f%%')
fig = plt.gcf()
fig.set_size_inches(3,3)
plt.show()


print('Evaluation score for logistic regression:',round(ans,2),' %')
```




```
Evaluation score for logistic regression: 42.21  %
```

In [186]:
```
drawgraphNoBar(preds[:,1],val_df['total_runs'],val_y[0])
```

```
(37,)

(37,)
```

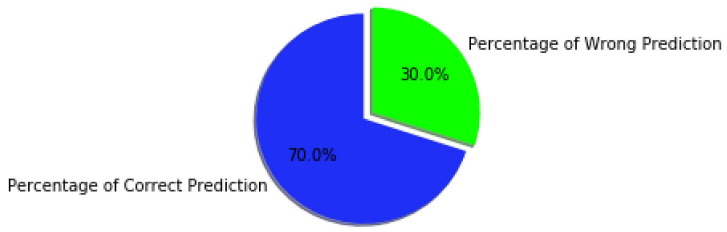


XG Boost

In [194]:
```
def runXGB(train_X, train_y, seed_val=0):
    param = {}
    param['objective'] = 'binary:logistic'
    param['eta'] = 0.05
    param['max_depth'] = 8
    param['silent'] = 1
    param['eval_metric'] = "auc"
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    num_rounds = 100

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)
    model = xgb.train(plst, xgtrain, num_rounds)
    return model
```

In [195]:
```
model = runXGB(dev_X, dev_y)
xgtest = xgb.DMatrix(val_X)
predsXG = model.predict(xgtest)
```

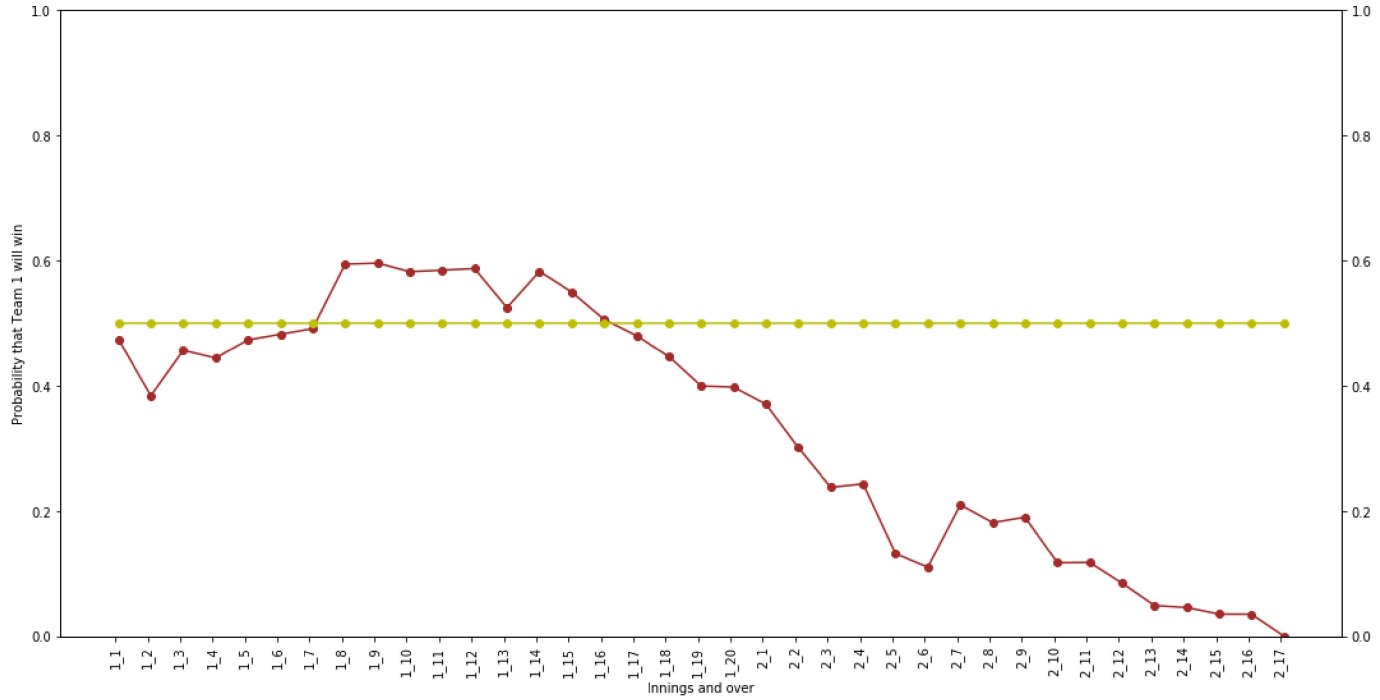
In [234]:
```
#Finding the Evaluation function

ids=train_df.match_id.unique()

cp=[]
ap=[]
c=0
ids=np.array([411])#ids[-2:-1]
for id in ids:
    val_df = train_df.ix[train_df.match_id == id,:]
    dev_df = train_df.ix[train_df.match_id != id,:]
    dev_X = np.array(dev_df[x_cols[:]])
    dev_y = np.array(dev_df['target'])
    val_X = np.array(val_df[x_cols[:]])#[:-1,:]
    val_y = np.array(val_df['target'])#[:-1]
    model = runXGB(dev_X, dev_y)
    xgtest = xgb.DMatrix(val_X)
    predsXG = model.predict(xgtest)

    if(val_y[0]==1):
        c=c+list(predsXG>=0.5).count(True)
    else:
        c=c+list(predsXG<0.5).count(True)
    #np_pred1=np.array(pred1)


ans = c/(ids.shape[0]*40)*100


#Plotting the Evaluation Function Result
slices=[ans,100-ans]
list(slices)
labels=['Percentage of Correct Prediction','Percentage of Wrong Prediction']
plt.pie(slices,labels=labels,colors=['#1f2ff3', '#0fff00'],startangle=90,shadow=True,explode=(0,0.1),autopct='%
1.1f%%')
fig = plt.gcf()
fig.set_size_inches(3,3)
plt.show()


print('Evaluation score for XG Boost:',round(ans,2),' %')
```




```
Evaluation score for XG Boost: 70.0  %
```

In [198]:
```
drawgraphNoBar(predsXG,val_df['total_runs'],val_y[0])
```

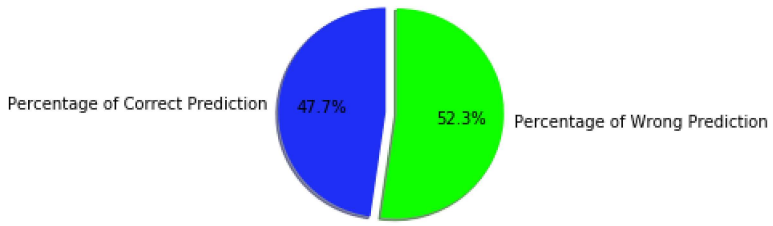
```
(37,)

(37,)
```



Neural Network

In [162]:
```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(solver='lbfgs', alpha=1e-5, activation='logistic',
                   hidden_layer_sizes=(2, 1), random_state=1,learning_rate_init=1e-9,max_iter=10)
mlp.fit(dev_X,dev_y)
predsNN=mlp.predict(val_X)
```

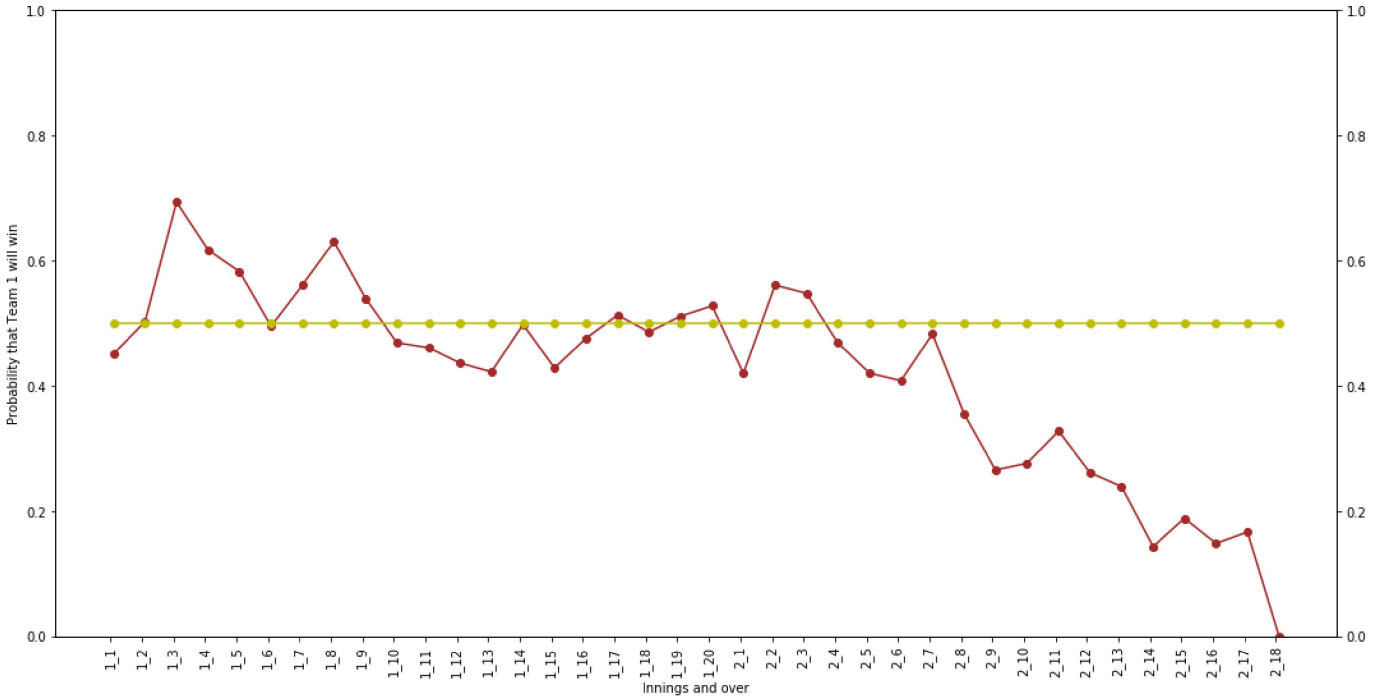
In [235]:
```
#Finding the Evaluation function

ids=train_df.match_id.unique()

cp=[]
ap=[]
c=0
#ids=np.array([411])#ids[-2:-1]
for id in ids:
    val_df = train_df.ix[train_df.match_id == id,:]
    dev_df = train_df.ix[train_df.match_id != id,:]
    dev_X = np.array(dev_df[x_cols[:]])
    dev_y = np.array(dev_df['target'])
    val_X = np.array(val_df[x_cols[:]])#[:-1,:]
    val_y = np.array(val_df['target'])#[:-1]
    mlp = MLPRegressor(solver='lbfgs', alpha=1e-5, activation='logistic',
                       hidden_layer_sizes=(5, 2), random_state=1,learning_rate_init=1e-9,max_iter=1000)
    mlp.fit(dev_X,dev_y)
    predsNN=mlp.predict(val_X)
    if(val_y[0]==1):
        c=c+list(predsNN>=0.5).count(True)
    else:
        c=c+list(predsNN<0.5).count(True)
    #np_pred1=np.array(pred1)


ans = c/(ids.shape[0]*40)*100


#Plotting the Evaluation Function Result
slices=[ans,100-ans]
list(slices)
labels=['Percentage of Correct Prediction','Percentage of Wrong Prediction']
plt.pie(slices,labels=labels,colors=['#1f2ff3', '#0fff00'],startangle=90,shadow=True,explode=(0,0.1),autopct='%
1.1f%%')
fig = plt.gcf()
fig.set_size_inches(3,3)
plt.show()


print('Evaluation score for Neural Network:',round(ans,2),' %')
```




```
Evaluation score for Neural Network: 47.71  %
```

In [215]:
```
drawgraphNoBar(predsNN,val_df['total_runs'],val_y[0])
```

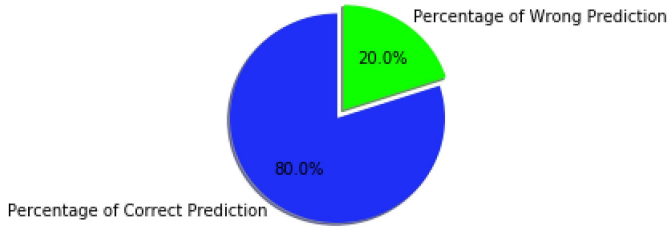
```
(38,)

(38,)
```



Decision Trees

In [236]:
```
from sklearn.tree import DecisionTreeRegressor
regr_1 = DecisionTreeRegressor(max_depth=4)
regr_1.fit(dev_X, dev_y)
# Predict
y_1 = regr_1.predict(val_X)
```

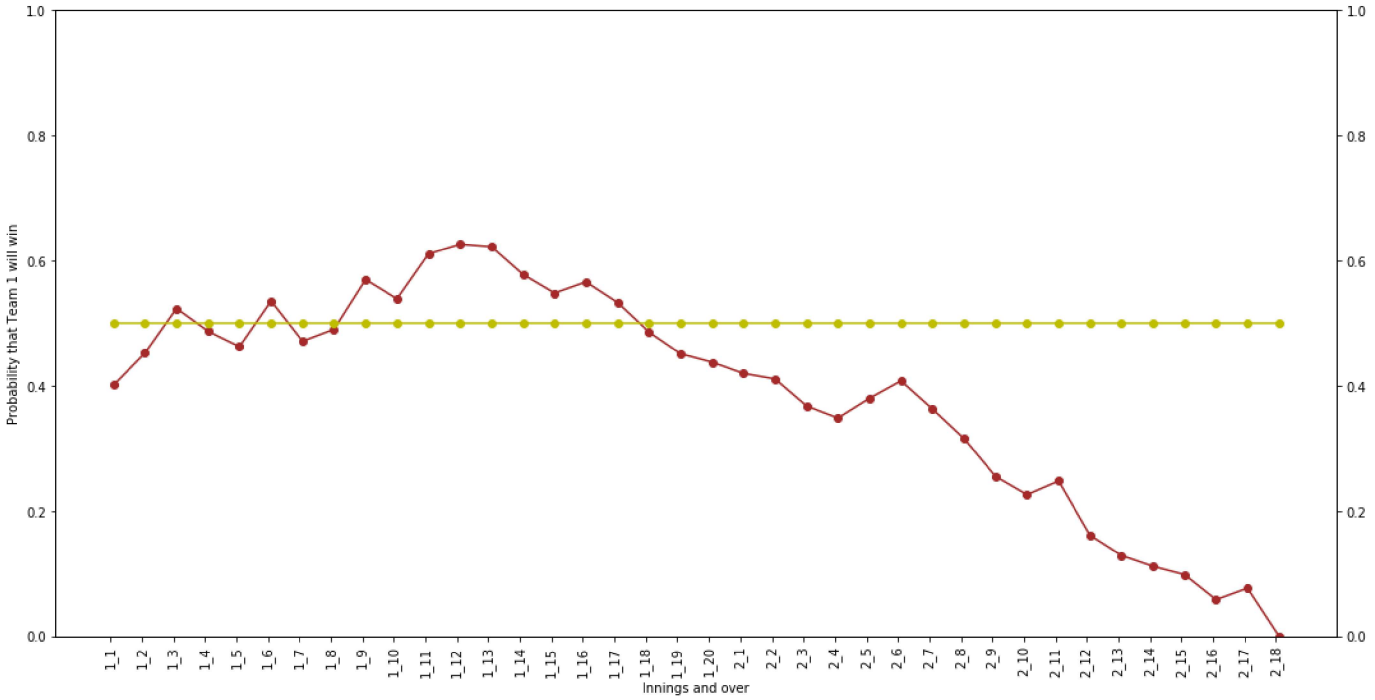
In [168]:
```
#Finding the Evaluation function

ids=train_df.match_id.unique()

cp=[]
ap=[]
c=0
ids=np.array([411])#ids[-2:-1]
for id in ids:
    val_df = train_df.ix[train_df.match_id == id,:]
    dev_df = train_df.ix[train_df.match_id != id,:]
    dev_X = np.array(dev_df[x_cols[:]])
    dev_y = np.array(dev_df['target'])
    val_X = np.array(val_df[x_cols[:]])#[:-1,:]
    val_y = np.array(val_df['target'])#[:-1]

    regr_1 = DecisionTreeRegressor(max_depth=4)
    regr_1.fit(dev_X, dev_y)
    # Predict
    y_1 = regr_1.predict(val_X)


    if(val_y[0]==1):
        c=c+list(y_1>=0.5).count(True)
    else:
        c=c+list(y_1<0.5).count(True)
    #np_pred1=np.array(pred1)


ans = c/(ids.shape[0]*40)*100


#Plotting the Evaluation Function Result
slices=[ans,100-ans]
list(slices)
labels=['Percentage of Correct Prediction','Percentage of Wrong Prediction']
plt.pie(slices,labels=labels,colors=['#1f2ff3', '#0fff00'],startangle=90,shadow=True,explode=(0,0.1),autopct='%
1.1f%%')
fig = plt.gcf()
fig.set_size_inches(3,3)
plt.show()


print('Evaluation score for Deciaion tree is:',round(ans,2),' %')
```




```
Evaluation score for Deciaion tree is: 80.0  %
```

In [226]:
```
drawgraphNoBar(y_1,val_df['total_runs'],val_y[0])
```

```
(38,)

(38,)
```



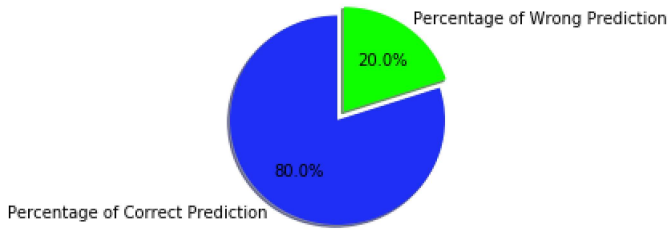
AdaBoost

In [170]:
```
from sklearn.ensemble import AdaBoostRegressor
regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),n_estimators=300, random_state=np.random.RandomSta
te(1))
regr_2.fit(dev_X, dev_y)
#Predict
y_2 = regr_2.predict(val_X)
```

In [171]:
```
y_2
```

Out[171]:
```
array([ 0.38985555,  0.38985555,  0.38985555,  0.38985555,  0.38985555,
        0.38985555,  0.38985555,  0.38985555,  0.65167053,  0.65167053,
        0.65167053,  0.65167053,  0.65167053,  0.38985555,  0.38985555,
        0.38985555,  0.38985555,  0.38985555,  0.38985555,  0.38985555,
        0.36176935,  0.36176935,  0.24816299,  0.24816299,  0.16369664,
        0.16369664,  0.19579988,  0.16369664,  0.16369664,  0.16369664,
        0.16369664,  0.16369664,  0.16369664,  0.16369664,  0.16369664,
        0.16369664,  0.16369664])
```

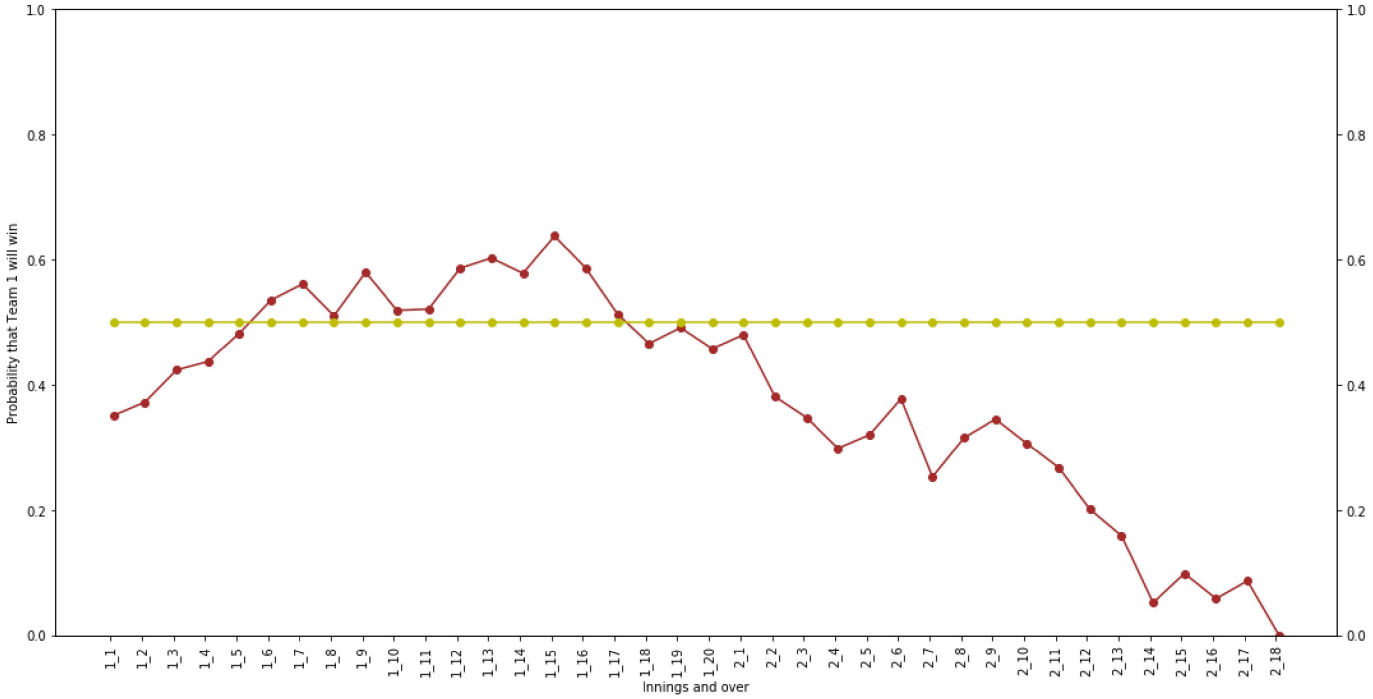
In [237]:
```
#Finding the Evaluation function

ids=train_df.match_id.unique()

cp=[]
ap=[]
c=0
ids=np.array([411])#ids[-2:-1]
for id in ids:
    val_df = train_df.ix[train_df.match_id == id,:]
    dev_df = train_df.ix[train_df.match_id != id,:]
    dev_X = np.array(dev_df[x_cols[:]])
    dev_y = np.array(dev_df['target'])
    val_X = np.array(val_df[x_cols[:]])#[:-1,:]
    val_y = np.array(val_df['target'])#[:-1]

    regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),n_estimators=300, random_state=np.random.Rando
mState(1))
    regr_2.fit(dev_X, dev_y)
    #Predict
    y_2 = regr_2.predict(val_X)


    if(val_y[0]==1):
        c=c+list(y_2>=0.5).count(True)
    else:
        c=c+list(y_2<0.5).count(True)
    #np_pred1=np.array(pred1)


ans = c/(ids.shape[0]*40)*100


#Plotting the Evaluation Function Result
slices=[ans,100-ans]
list(slices)
labels=['Percentage of Correct Prediction','Percentage of Wrong Prediction']
plt.pie(slices,labels=labels,colors=['#1f2ff3', '#0fff00'],startangle=90,shadow=True,explode=(0,0.1),autopct='%
1.1f%%')
fig = plt.gcf()
fig.set_size_inches(3,3)
plt.show()


print('Evaluation score for Adaboost is:',round(ans,2),' %')
```




```
Evaluation score for Adaboost is: 80.0  %
```

In [232]:
```
drawgraphNoBar(y_2,val_df['total_runs'],val_y[0])
```

```
(38,)

(38,)
```



In [121]:
```
#
```

Evaluating the Models

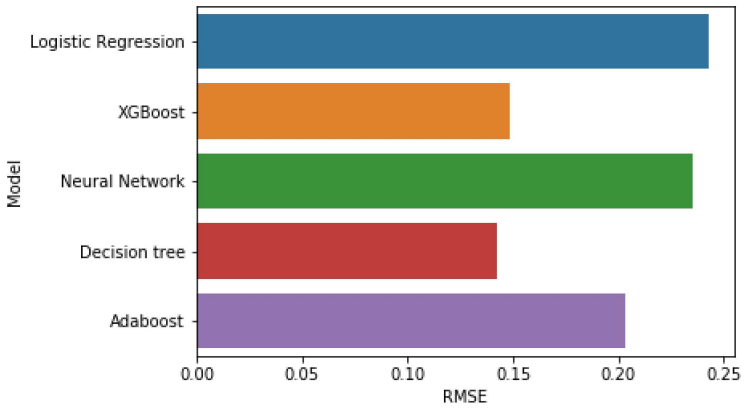
In [111]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, explained_variance_score, r2_scor
e
rmseXG=mean_squared_error(predsXG,val_y)
compdf=pd.DataFrame(columns=['Model','RMSE','Abs Error','Expected Variance score','R2'])
#compdf.columns=['Model','RMSE','Abs Error','R2','Expected Variance score']
compdf.loc[0]=['Logistic Regression',mean_squared_error(preds,val_y),mean_absolute_error(preds,val_y),
explained_variance_score(preds,val_y),r2_score(preds,val_y)]
compdf.loc[1]=['XGBoost',mean_squared_error(predsXG,val_y),mean_absolute_error(predsXG,val_y),explaine
d_variance_score(predsXG,val_y),r2_score(predsXG,val_y)]
compdf.loc[2]=['Neural Network',mean_squared_error(predsNN,val_y),mean_absolute_error(predsNN,val_y),e
xplained_variance_score(predsNN,val_y),r2_score(predsNN,val_y)]
compdf.loc[3]=['Decision tree',mean_squared_error(y_1,val_y),mean_absolute_error(y_1,val_y),explained_
variance_score(y_1,val_y),r2_score(y_1,val_y)]
compdf.loc[4]=['Adaboost',mean_squared_error(y_2,val_y),mean_absolute_error(y_2,val_y),explained_varia
nce_score(y_2,val_y),r2_score(y_2,val_y)]
compdf
```

Out[111]:

	Model	RMSE	Abs Error	Expected Variance score	R2
0	Logistic Regression	0.242631	0.492562	0.423984	0.185362
1	XGBoost	0.148337	0.346834	0.293783	0.119835
2	Neural Network	0.234976	0.478323	0.420367	0.167847
3	Decision tree	0.142209	0.330522	0.385243	0.198237
4	Adaboost	0.203145	0.399333	0.337838	0.258734

In [113]:
```
sns.barplot(compdf['RMSE'],compdf['Model'])
```

Out[113]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4e1a50208>
```



6. FINAL MATCH PREDICTION

In this part, we try to actually verify the results of our model. We test the model on all the matches apart from a selected one. This trained set is then used to finally test the remaining match and provide an over-by-over probability analysis. For our final prediction we have selected match number 411 which is match between India and Australia played in 2014 at Shere Bangla Stadium. Following page shows the comparisons of our prediction outcome and actual results.

As suggested by the professor, we also tried to plot the event of the probability that team would win looking at the odds before the match. This is done by giving the models features such as toss results, venue and past performances of each of the teams etc.

Lastly, we have tried to plot that a graph which reflects how the chances of team to win increase if the model predicts team's victory with higher probability . We see that the number of occurrences increase with the increase in threshold probability, showing that if model predicted higher probability than there are more chances of team to win. It shows near linear relationship of the model's prediction with the team's victory. In this graph , we have plotted the probability thresholds against the percentage of occurrences of that threshold. This resulted in near linear curve.

For e.g. if the model predicted lower probability of team's win say 0.2, than percentage of occurrences of lower probability with end result of team's win is ~15%. Similarly , if the probability threshold is 1.0,the number of occurrences of 1.0 is low but the percentage of occurrences of 1.0 causing team's win with such a predicted probability is ~90-95%

FINAL MATCH PREDICTION

For verfifying our prediction model, let us select a match from the T20 International World cup. It is the 28th Match, of Group 2 (N) World T20 played at the Shere Bangla National Stadium, Mirpur, Dhaka on Mar 30 2014. India top the group ahead of the semi-finals with Ashwin being the Man of the Match

Selecting the Festures for our prediction

In [18]:
```
x_cols = ['inning', 'over', 'total_runs', 'player_dismissed', 'innings_wickets', 'innings_score', 'score_t
arget', 'remaining_target', 'run_rate', 'required_run_rate', 'runrate_diff', 'is_batting_team']

# let us take all the matches but for the final as development sample and final as val sample #
val_df = train_df.loc[train_df.match_id == 411,:]
dev_df = train_df.loc[train_df.match_id != 411,:]

# create the input and target variables #
dev_X = np.array(dev_df[x_cols[:]])
dev_y = np.array(dev_df['target'])
val_X = np.array(val_df[x_cols[:]])#[:-1,:]
val_y = np.array(val_df['target'])#[:-1]
print(dev_X.shape, dev_y.shape)
print(val_X.shape, val_y.shape)
```

```
((21088, 12), (21088,))
((37, 12), (37,))
```

Training the Model using XGBoost Classifier

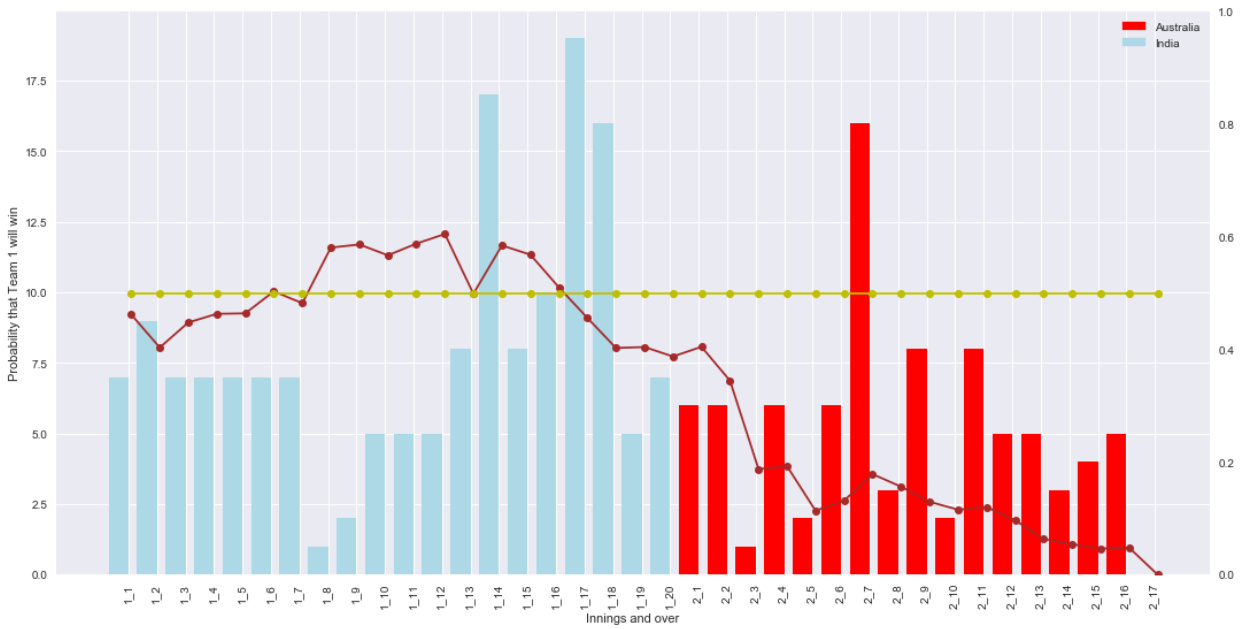
In [12]:
```
def runXGB(train_X, train_y, seed_val=0):
    param = {}
    param['objective'] = 'binary:logistic'
    param['eta'] = 0.05
    param['max_depth'] = 8
    param['silent'] = 1
    param['eval_metric'] = "auc"
    param['min_child_weight'] = 1
    param['subsample'] = 0.7
    param['colsample_bytree'] = 0.7
    param['seed'] = seed_val
    num_rounds = 100

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X, label=train_y)
    model = xgb.train(plst, xgtrain, num_rounds)
    return model
```

In [13]:
```
model = runXGB(dev_X, dev_y)
xgtest = xgb.DMatrix(val_X)
predsXG = model.predict(xgtest)
```

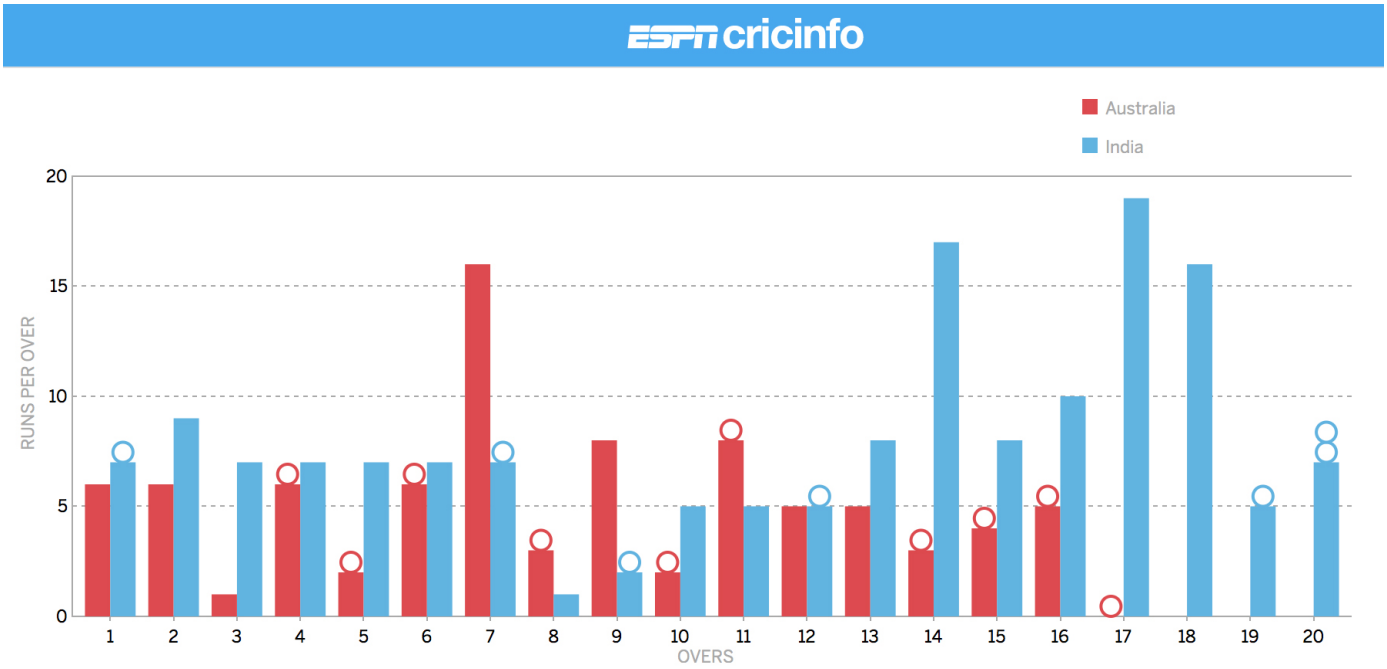
Plotting the Probabilities of winning at each over for Team 1

In [21]:
```
#Function to draw the Graph

def drawgraph(preds,scores,yexp):
    fig, ax1 = plt.subplots(figsize=(18,9))
    labels = np.array(val_df.apply(lambda row: str(row['inning']) + "_" + str(row['over']), axis=1))
    ind = np.arange(len(labels))
    #print(ind.shape)
    #print()
    width = 0.7
    ax2 = ax1.twinx()
    ax1.set_xticks(ind+((width)/2.))
    ax2.set_ylim([0,1])
    ax1.bar(ind,scores , width=width, color=['lightblue']*20 + ['red']*20)

    red_patch = mpatches.Patch(color='red', label='Australia')
    blue_patch = mpatches.Patch(color='lightblue', label='India')
    plt.legend(handles=[red_patch,blue_patch])
    ax1.set_xticklabels(labels, rotation='vertical')
    ax1.set_ylabel("Probability that Team 1 will win")
    ax1.set_xlabel("Innings and over")
    #probs=np.array(preds)
    #probs=[x for x in list(probs)]
    #probs=np.array(probs)[:,1]
    probs=preds
    #print(probs.shape)
    #probs=np.append(probs,1)
    probs[probs.shape[0]-1]=yexp
    ax2.plot(ind+0.45, probs, color='brown', marker='o')
    ax2.plot(ind+0.45, np.array([0.5]*probs.shape[0]), color='y', marker='o')
    ax2.grid(b=False)
    plt.show()
```

In [22]:
```
drawgraph(predsXG,val_df['total_runs'],val_y[0])
```



Comparing our results of probabilities with the results obtained by ESPN



Source : ESPN

Analysis

We observe the runs scored by each team in each Over is same in both of the graphs.

However, the prediction of probabilities keep varying throughout the match. It sometimes favours the batting team while sometimes favouring the bowling team. At each over, it is accounting the most important moments such as the NUMBER OF RUNS scored in that over(Advantage for Batting team) and the WICKETS falled in the over(Advantage for the bowling team). In this way, the model predicts the winning probability at each over accounting all the important events

Example: Consider the 7th Over in the Indian Innings(Blue Lines). Up unitl this point India has been playing really well, however there is a Fall of wicket in the 7th Over (Shown by a round circle in ESPN graph). This counts as an advantage to the Australian team and the probability of them winning goes above 0.5 in our graph (See 1_8).

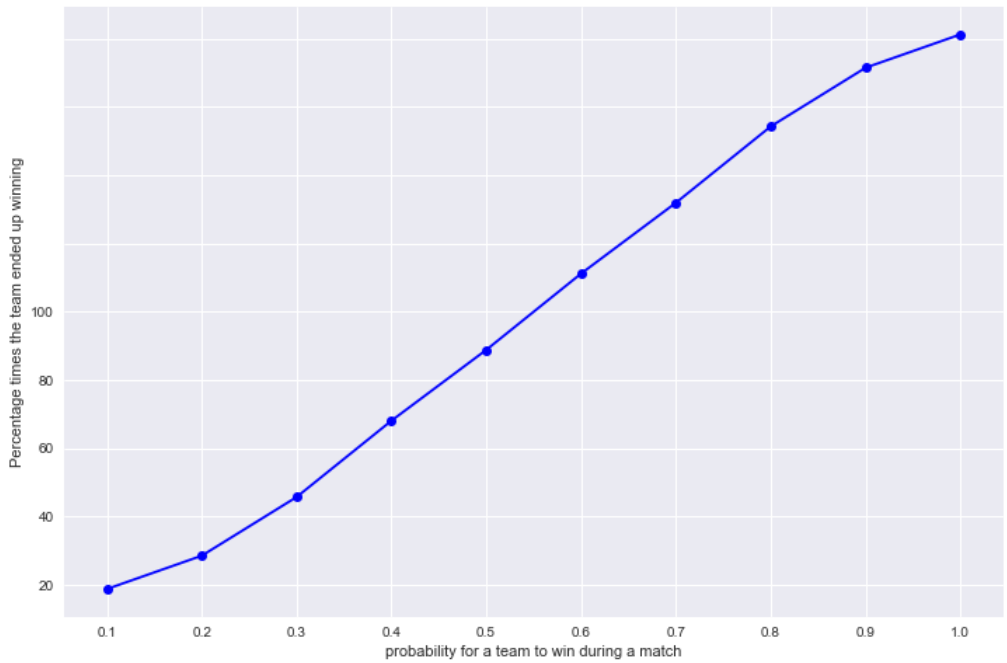
Our model is also considering the number of runs scored at each over. Hence, when there is a fall of wicket in the first over itself of the Indian Innings, the probability of Australia still falls down in the consequent over(1_2) as India manages to score really high runs in that over.

In [40]:
```
ids=train_df.match_id.unique()
logistic = LogisticRegression()
cp=[]
ap=[]
train_df['is_team1_won']=train_df['team1']==train_df['winner']
for id in ids:
    validation = train_df.loc[train_df.match_id == id,:]
    training = train_df.loc[train_df.match_id != id,:]
    training_X = np.array(training[x_cols[:]])
    training_y = np.array(training['target'])
    validation_X = np.array(validation[x_cols[:]])[:-1,:]
    validation_y = np.array(validation['target'])[:-1]
    model = runXGB(dev_X, dev_y)
    xgtest = xgb.DMatrix(val_X)
    predsXG = model.predict(xgtest)

    ap.extend(predsXG)
    cp.extend(predsXG)
```

In [43]:
```
dict1={}
dictAll={}
dictFinal={}
for i in [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]:
    dict1[i]=0
    dictAll[i]=0
    dictFinal[i]=0
for i in cp:
    j=math.ceil(i*10)/10
    dict1[j]=dict1[j]+1
for i in ap:
    j=math.ceil(i*10)/10
    dictAll[j]=dictAll[j]+1
for i in [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]:
    if(dictAll[i]==0):
        dictAll[i]=0.01
    dictFinal[i]=dict1[i]/float(dictAll[i])

print('The value of the final dictionary are')
x=list(sorted(dictFinal.keys()))
y=list(sorted(dictFinal.values()))
print(x)
print(y)
fig, ax = plt.subplots(figsize=[12,8])
plt.plot(x,y,color='b',marker='o')
ax.set_xticks(x)
ax.set_yticklabels([0,20,40,60,80,100])
ax.set_xlabel('probability for a team to win during a match')
ax.set_ylabel('Percentage times the team ended up winning')
```

```
The value of the final dictionary are
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
[0.09368191721132897, 0.14248592464270246, 0.2284632376143894, 0.3403031887088343, 0.4441055718475073, 0.
5558944281524927, 0.6596968112911658, 0.7715367623856106, 0.8575140753572975, 0.906318082788671]
```

Out[43]: <matplotlib.text.Text at 0x11a21acd0>



We now consider the Number of times a threshold probabilty ends up winning match with respect to all the times that threshold is ever seen. The above graph shows near linear relationship of the model's prediction with the team's victory. In this graph , we have plotted the probability thresholds against the percentage of occurences of that threshold. This resulted in near linear curve.

Fore.g. if the model predicted lower probability of team's win say 0.2, than percentage of occurences of lower probability with end result of team's win is ~15%. Similarly , if the probability threshold is 1.0,the number of occurances of 1.0 is low but the percentage of occurences of 1.0 causing team's win with such a predicted probability is ~90-95%

This graphs gives insights in a more meaningful way that reflects how the chances of team to win increase if the model predicts team's victory with higher probability.

In [ ]:

PREDICTING MATCH RESULTS BASED ON INITIAL ODDS

In [342]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [261]:
```
#load the matches data
matches=pd.read_csv('./output.csv')
matches.head()
```

Out[261]:

	team1	team2	toss_decision	toss_winner	venue	winner
0	Australia	Sri Lanka	field	Sri Lanka	Melbourne Cricket Ground	Sri Lanka
1	Australia	Sri Lanka	field	Sri Lanka	Simonds Stadium, South Geelong	Sri Lanka
2	Australia	Sri Lanka	field	Sri Lanka	Adelaide Oval	Australia
3	Ireland	Hong Kong	bat	Hong Kong	Bready Cricket Club, Magheramason	Hong Kong
4	Zimbabwe	India	field	India	Harare Sports Club	Zimbabwe

In [262]:
```
#drop the matches whose outcomes are either tie or DL or stopped due to rain
matches = matches.dropna(subset=['winner'])

#To find unique team names
result = matches.sort_values(['team1'], ascending=[1])
```

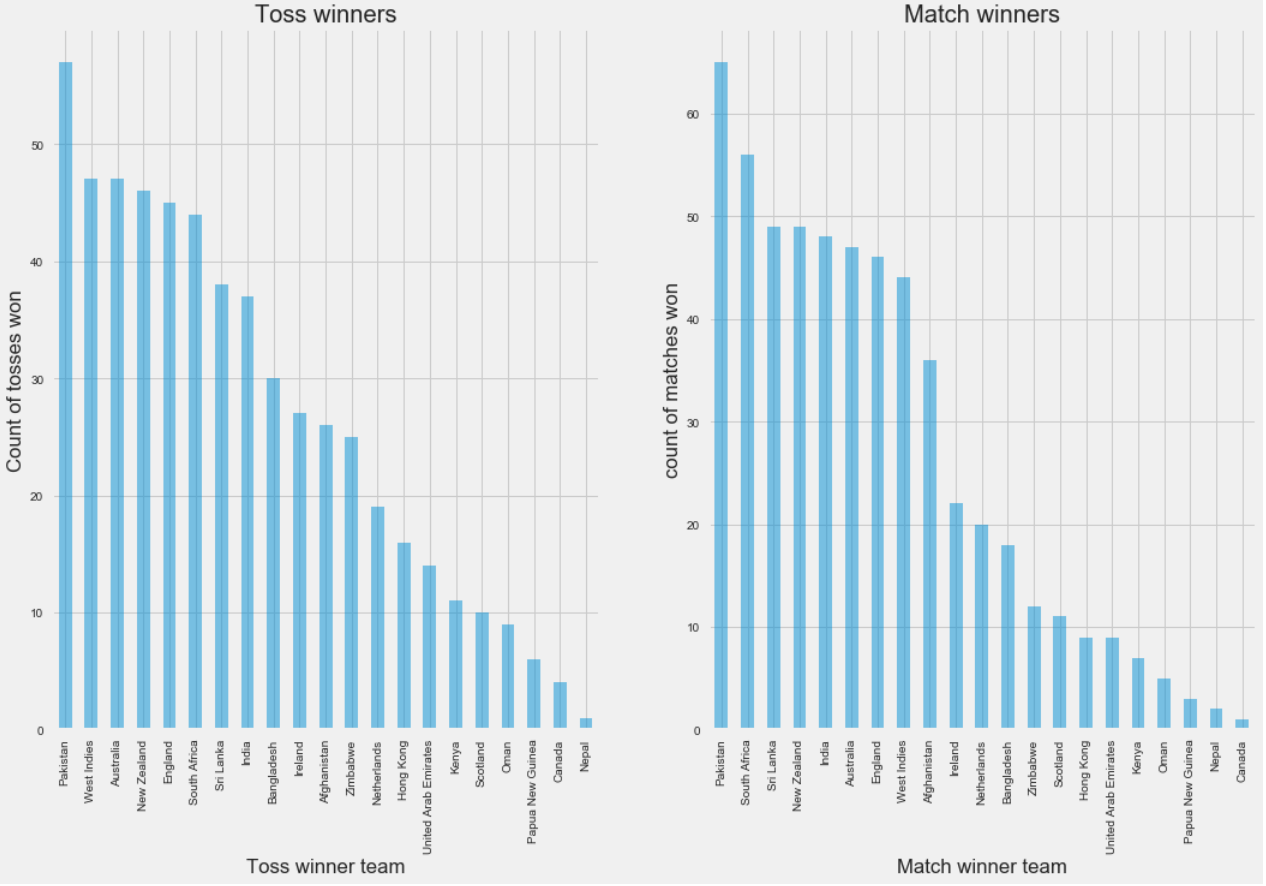
In [263]:
```
#countries names to number encoding
naToNumDict = {'Afghanistan':1, 'Australia':2, 'Bangladesh':3, 'Bermuda':4, 'Canada':5, 'Engla
 'Hong Kong':7, 'India':8, 'Ireland':9, 'Kenya':10, 'Nepal':11, 'Netherlands':12, 'New Zealand
 'Oman':14, 'Pakistan':15, 'Scotland':16, 'South Africa':17, 'Sri Lanka':18,
 'United Arab Emirates':19, 'West Indies':20, 'Zimbabwe':21, 'Papua New Guinea':22}

encode = {'team1': naToNumDict,
          'team2': naToNumDict,
          'toss_winner': naToNumDict,
          'winner': naToNumDict}

#dictionary of teams mapped to numbers for future reference
dicVal = encode['winner']

matches.replace(encode, inplace=True)
#Matches after teams got encoded to numericals
matches.head()
```

Out[263]:

	team1	team2	toss_decision	toss_winner	venue	winner
0	2	18	field	18	Melbourne Cricket Ground	18
1	2	18	field	18	Simonds Stadium, South Geelong	18
2	2	18	field	18	Adelaide Oval	2
3	9	7	bat	7	Bready Cricket Club, Magheramason	7
4	21	8	field	8	Harare Sports Club	21

In [264]:
```
#Find some stats on the match winners and toss winners
temp1=df['toss_winner'].value_counts(sort=True)
temp2=df['winner'].value_counts(sort=True)
toss = {}
tossIdx = []
winner = {}
winIdx = []

# No of tosses won by each team
for idx, val in temp1.iteritems():
    toss[list(dicVal.keys())[list(dicVal.values()).index(idx)]] = val
    tossIdx.append(list(dicVal.keys())[list(dicVal.values()).index(idx)])

# No of matches won by each team
for idx, val in temp2.iteritems():
    winner[list(dicVal.keys())[list(dicVal.values()).index(idx)]] = val
    winIdx.append(list(dicVal.keys())[list(dicVal.values()).index(idx)])
```

In [284]:
```
fig = plt.figure(figsize=(16,10))
axis1 = fig.add_subplot(121)
axis1.set_xlabel('Toss winner team')
axis1.set_ylabel('Count of tosses won')
axis1.set_title("Toss winners")
tempx = pd.Series(toss, index = tossIdx)
tempx.plot(kind='bar',alpha = 0.5)

axis2 = fig.add_subplot(122)
tempy = pd.Series(winner, index = winIdx)
tempy.plot(kind = 'bar', alpha = 0.5)
axis2.set_xlabel('Match winner team')
axis2.set_ylabel('count of matches won')
axis2.set_title("Match winners")
mlt.show()
```



The above plots shows that toss winner team vs counts of tosses won and match winner tem vs count of matches won. It is clearly seen that pakistan has won max number of tosses and also won maximum number of matches but this trend didn't

follow down the 2nd, 3rd teams...

In [285]:
```
#converting categorical columns toss_decision and venue to numericals
from sklearn.preprocessing import LabelEncoder
cat_var = ['toss_decision','venue']
le = LabelEncoder()
for i in cat_var:
    df[i] = le.fit_transform(df[i])
df.head()
```

Out[285]:

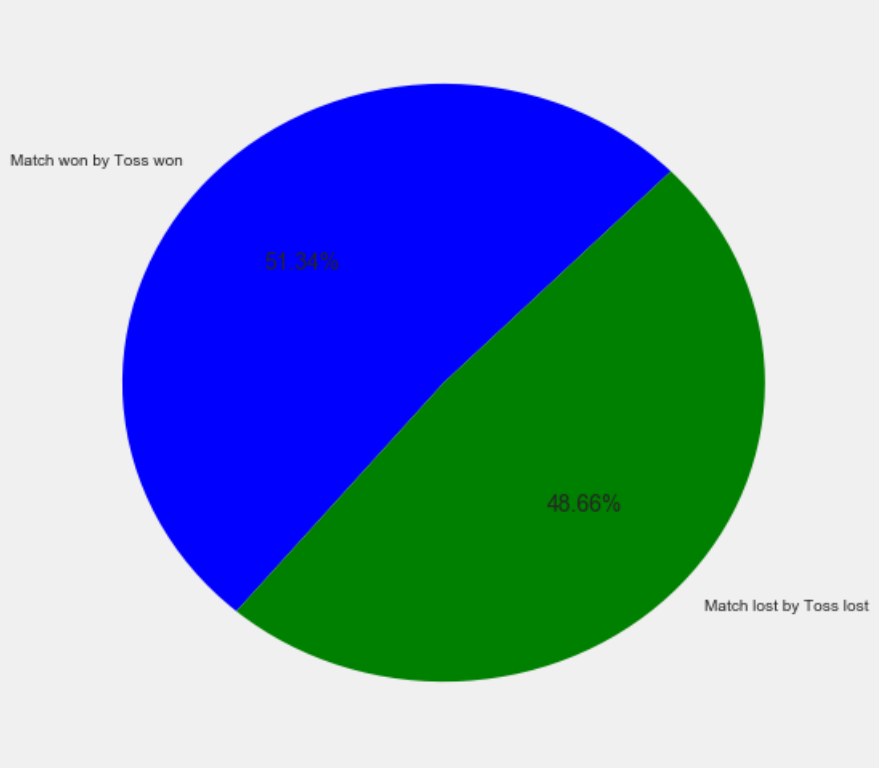
	team1	team2	toss_decision	toss_winner	venue	winner
0	2	18	1	18	45	18
1	2	18	1	18	71	18
2	2	18	1	18	1	2
3	9	7	0	7	8	7
4	21	8	1	8	27	21

In [327]:
```
#Baseline model considering only toss winner
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

model_bl = RandomForestClassifier(n_estimators=100)
outcome_var = ['winner']
predictor_var = ['team1', 'team2', 'toss_winner']
model_bl.fit(df[predictor_var], df[outcome_var])
predictions = model_bl.predict(df[predictor_var])
accuracy = metrics.accuracy_score(predictions, df[outcome_var])
print 'Accuracy : %s' % '{0:.2%}'.format(accuracy)
```

```
/home/prasasnth/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:8: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the shape o
f y to (n_samples,), for example using ravel().


Accuracy : 78.89%
```

Baseline model just considers the team1, team2 and toss_winner as its features and uses random forest to predict. The model performs decent enough as it has insufficient number of features.

In [328]:
```
#plotting the pie chart
mlt.style.use('fivethirtyeight')
df_filter=df[df['toss_winner']==df['winner']]
#559 number of matches
split=[len(df_filter),(559-len(df_filter))]
mlt.pie(split,labels=['Match won by Toss won','Match lost by Toss lost'],startangle=45,autopct
fig = mlt.gcf()
fig.set_size_inches(8,8)
mlt.show()
```



From the above pie it is clearly evident that though approximately 50% of the team toss_winners are the match winners but still it is not that sufficient enough to predict the output of the match.

In [329]:
```
#Advanced model with venue, toss_decision as the additional features.
model = RandomForestClassifier(n_estimators=100)
outcome_var = np.array(['winner'])
predictor_var = np.array(['team1', 'team2', 'venue', 'toss_winner','toss_decision'])
model.fit(df[predictor_var], df[outcome_var])
predictions = model.predict(df[predictor_var])
accuracy = metrics.accuracy_score(predictions, df[outcome_var])
print 'Accuracy : %s' % '{0:.2%}'.format(accuracy)
```

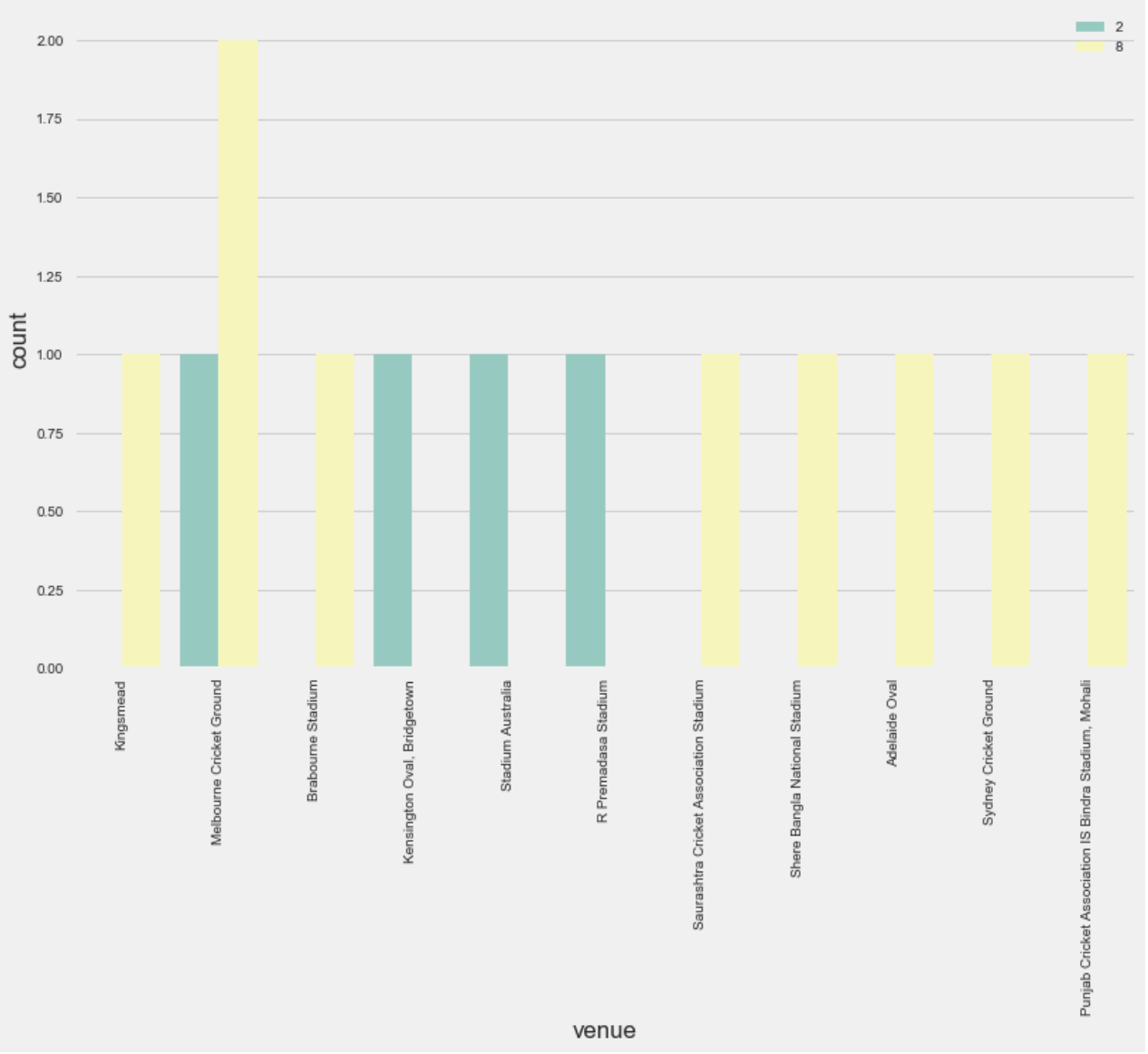
```
/home/prasasnth/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:5: DataConversion
Warning: A column-vector y was passed when a 1d array was expected. Please change the shape o
f y to (n_samples,), for example using ravel().
  """

Accuracy : 95.17%
```

In [330]:
```
#Coefficients of the features
coeff = pd.Series(model.feature_importances_, index=predictor_var).sort_values(ascending=False
print(coeff)
```

```
team2            0.260978
venue            0.249608
team1            0.246903
toss_winner      0.183502
toss_decision    0.059009
dtype: float64
```

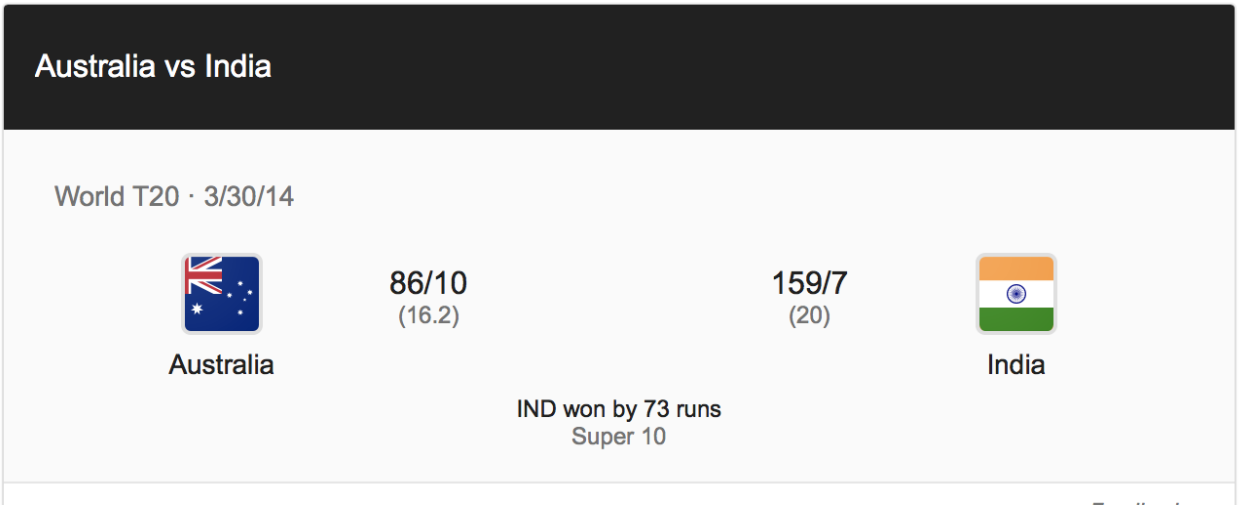
#Importance of features: Ignoring teams, venue seems to be one of important factors in determining winners followed by toss winning and toss_decision

In [341]:
```
for showing the importance of venue in determining the output of a match
orn as sns
g the teams
l['Australia']
l['India']
ll the matches played by those two teams
=matches[((matches['team1']==team1)|(matches['team2']==team1))&((matches['team1']==team2)|(matc
ot(x='venue', hue='winner',data=tempMatches,palette='Set3')
rotation='vertical')
egend( loc = 'upper right')
()
e_inches(12,8)
```



Prediction for our Selected Match

Prediction for our Selected Match



In [343]:
```
# Sample test
# 'team1', 'team2', 'venue', 'toss_winner', 'toss_decision'
team1='India'
team2='Australia'
toss_winner='Australia'
#Venue: sheri bangla national stadium corresponds to 70, field corresponds to label 1
sample=[dicVal[team1],dicVal[team2],'70',dicVal[toss_winner],'1']
sample = np.array(sample).reshape((1, -1))
output=model.predict(sample)
print "Actual Winning Team: India\n",
print "Predicted Winning Team: ",list(dicVal.keys())[list(dicVal.values()).index(output)]
```

```
Actual Winning Team: India
Predicted Winning Team:  India
```

Thus, our model correctly predicts the chance of a team winning based on just inital match conditions

In [ ]:

7. SUMMARY AND FUTURE SCOPE

The machine learning models are very handy tools when it comes to predict the outcome of a match. Match prediction helps in understanding the dynamics of the game. It is also a way to place bets at right points of time to get maximum profit. But any ML model is limited by the quality of data on which it predicts. We were limited by the amount of data present in dataset. Comparing various ML models we get to know how various models predict, what hyper parameters to tune. As future work, we would like to extend the functionality of the application to predict the outcomes of matches that could result in draw, interrupted by rain, player injury and match results affected by D/L scores. This can make our prediction system capable of handling unexpected circumstances.

8. REFRENCES

1] XG Gradient Boosting : https://github.com/dmlc/xgboost

2] Decision Trees : scikit-learn.org/stable/modules/tree.html

3] Finding correlation: https://www.datascience.com/blog/introduction-to-correlation-learn-data-science-tutorials

4] ESPN Cricinfo : http://www.espncricinfo.com/series/8604/game/682951/Australia-vs-India-28th-Match,-Group-2-world-t20

5] Plots and Figures : https://seaborn.pydata.org/generated/seaborn.heatmap.htm, http://www.kaggle.com, https://matplotlib.org/users/pyplot_tutorial.html

6] Cricket data : https://cricsheet.org/downloads, http://www.howstat.com/cricket/home.asp, https://www.icc-cricket.com

7] Adaboosting : https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/

8] Manipulating CSV files using Python : https://www.protechtraining.com/blog/post/737