CRICTO-PAUL

Progress Report

Group Members:

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

1. GATHERING MATCH DATA

Cricket data is available in multiple formats throughout the internet. Being one of the most followed sport in world, there enough data available online on the sports websites providing ball-by-ball analysis. The data available on websites such as https://www.icc-cricket.com (International Cricket Council) and http://www.espncricinfo.com are in the form of a scorecard. The data mentioned can be obtained by scrapping them. However, it might result in obtaining inconsistent and redundant data which will require extensive cleaning.

One important source of data that we found on the go is the website https://cricsheet.org/ which provides the data of each tournament and match in a YAML and CSV formats. The website has itself extracted data from the official cricket websites and validated them. Cricsheet is cricket equivalent of Retrosheet – which is the impressive source of baseball event data.

2. DATA PROCESSING

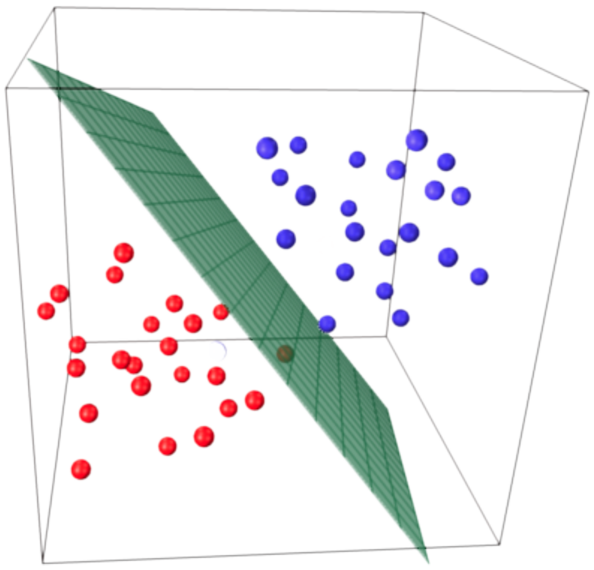
Based on the match information, we need to find the relevant features that can help us to build our model to predict the matches. This includes computing external features such as Net Rune rate and Required run rate which are calculated at every instant of the match.

We got the data from cricsheet.org and populated a csv file containing the ball by ball stats and the metadata of each match. Then we converted player dismissed into binary format as. Then, we calculated total runs and total wickets per over. We add 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.

3. BUILDING BASELINE MODEL

We are using logistic regression to build our baseline model. Logistic regression is a statistical method to analyze dataset with multiple independent variables. The central premise of Logistic Regression is that the input space is separable into two nice regions by a linear boundary. For 2 dimensions, the boundary is a straight line and for 3 dimensions it is a plane. The boundary is decided by input data and learning algorithm. This divined boundary is called 'linear discriminant'.



In this regression, the outcome is binary. The goal is to find best fitting model that relates the dichotomous variable with the set of independent variables. Logistic regression generates coefficients to predict a logit transformation of probability:

The logit function $logit(p)$ is defined as

$$logit(p) = b_o + b_1 X_1 + b_2 X_2 + ... + b_k X_k$$

$$logit(p) = ln\ (p/1 - p) \qquad \text{where } p = e^t/(1 + e^t)$$

We use logistic regression for training on all the matches except one which we want to use for testing.

4. FINDING PERCENTAGE TIMES A TEAM WINS FOR A GIVEN PROBABILTY

As suggested by the professor, we are trying to plot the event of the probability that team would win being at a certain threshold vs number of occurrences in which the team won with the predicted probability. We see that the number of occurrences increase with the increase in threshold, showing that if model predicted higher probability than there are more chances of team to win.

Next, we also tried to plot the graph 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. The 2nd 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 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%

This indicates better prediction of our model using logistic regression.

5. NEXT STEPS

Our current model takes the dynamics of the game like run-rate, remaining target, required run rate, and makes a prediction with assumption that the features are independent variables. But, in machine learning, model is as good as the features. So, we would be including features like player statistics, team rankings, venue details, match played at home/away and explore more complex models.

Player statistics:
It has been observed that players who have won titles like man of the match (this award shows that the player was most effective in that particular match compared to other members in making team win), more number of hundreds and fifties, if they are still batting, the chances of team's win was higher. Their presence also boosted team's confidence levels to win. So, we will be coming up with score function that scores player based on number of man of match awards (shows effective quotient), number of hundreds, fifties (experience), number of matches played (experience). There will be separate scoring function for bowler and batsman and captain of the team

Team stats:
Team's strength is usually aggregation of the skill and experience of various players. So we can use the score of players to compute team's score. Team formation also depends on combination of players for a given match. e.g. if the venue is a batting pitch i.e. where the outfield is fast and team needs to score more runs to stay in competition, than team goes with more batsman than bowlers, where as if the pitch of venue is bowling pitch, then team has to go with more bowlers than batsman as there are more chances of taking wickets than scoring runs. Thus, scoring function of team besides players and its rankings should also consider venue stats as team's strength varies based on combination of players which in turn varies based on venue. This team's scoring function output will be used as feature.

Venue details:

Match is also decided based on the venue. Different venues have different pitch conditions, outfields which are usually affected by the maintenance of board, weather conditions in past few days like rain, dampness which can become major factor in losing of wickets or runs conceded. We will be coming up with scoring function that includes pitch conditions, outfield stats.

Match played home/away:

Host team usually has advantage over the visiting team as the host team will be aware of the venue conditions more than the visitors. Also, there would be huge support from the spectators for the host team. Thus , the scoring function of this feature should reflect the support from crowd- percentage of filling of seats, %age of wins of the team at that venue.

Model:

We would be trying models like ensemble, decision trees, gradient boosting and come up with a better prediction model for this project.

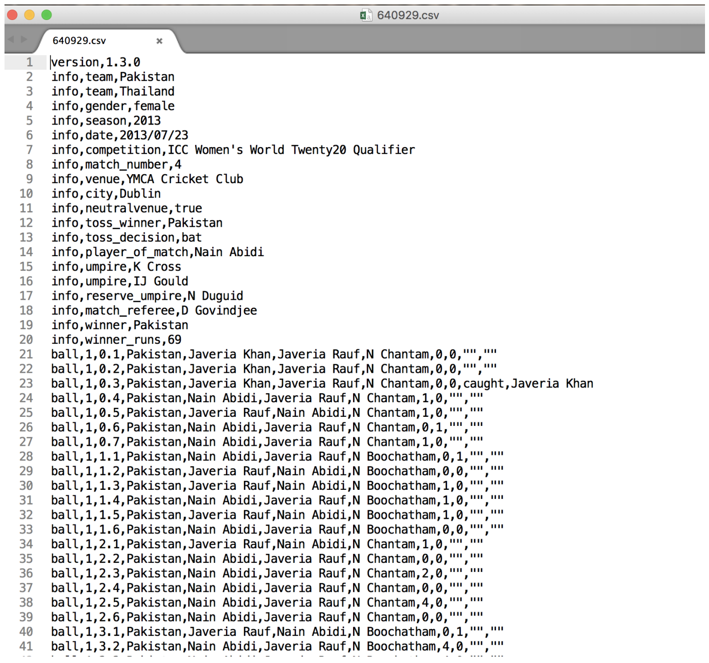
Plotting:

We would be plotting graphs that would compare different models and importance of various features on prediction of probability.

Data Gathering

The data obtained from Cricsheet is in the form of seperate csv files for each match. Data includes csv files of all International T20 format cricket matches. Each csv file has the following information.

Match Info: Teams played,Toss winner and decision to ball or bat, Winner of the match etc.
Score Info: Each ball information like innings, which ball of the over, runs scored


640929.csv

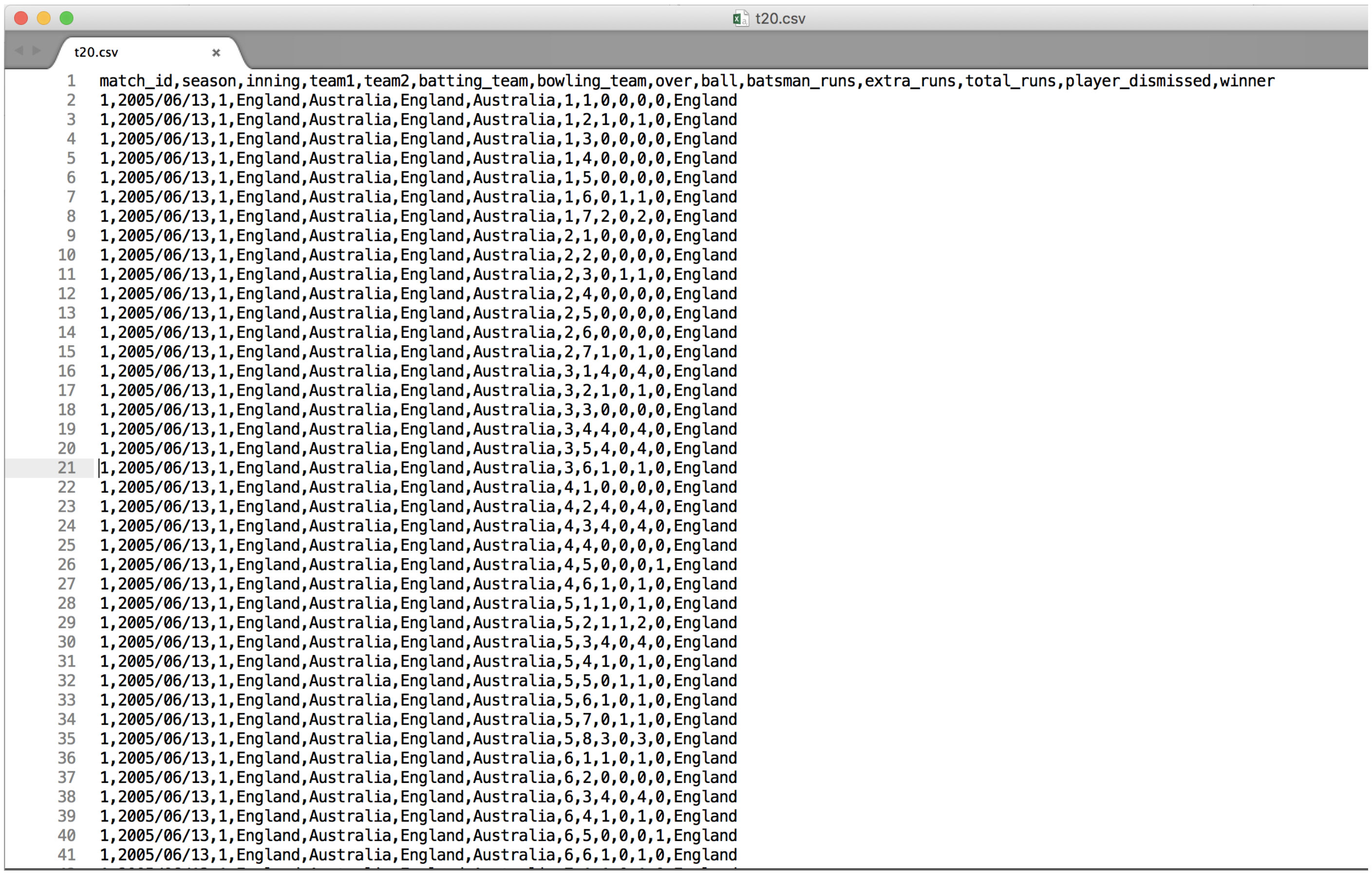
```
version,1.3.0
info,team,Pakistan
info,team,Thailand
info,gender,female
info,season,2013
info,date,2013/07/23
info,competition,ICC Women's World Twenty20 Qualifier
info,match_number,4
info,venue,YMCA Cricket Club
info,city,Dublin
info,neutralvenue,true
info,toss_winner,Pakistan
info,toss_decision,bat
info,player_of_match,Nain Abidi
info,umpire,K Cross
info,umpire,IJ Gould
info,reserve_umpire,N Duguid
info,match_referee,D Govindjee
info,winner,Pakistan
info,winner_runs,69
ball,1,0.1,Pakistan,Javeria Khan,Javeria Rauf,N Chantam,0,0,"",""
ball,1,0.2,Pakistan,Javeria Khan,Javeria Rauf,N Chantam,0,0,"",""
ball,1,0.3,Pakistan,Javeria Khan,Javeria Rauf,N Chantam,0,0,caught,Javeria Khan
ball,1,0.4,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,1,0,"",""
ball,1,0.5,Pakistan,Javeria Rauf,Nain Abidi,N Chantam,1,0,"",""
ball,1,0.6,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,0,1,"",""
ball,1,0.7,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,1,0,"",""
ball,1,1.1,Pakistan,Nain Abidi,Javeria Rauf,N Boochatham,0,1,"",""
ball,1,1.2,Pakistan,Javeria Rauf,Nain Abidi,N Boochatham,0,0,"",""
ball,1,1.3,Pakistan,Javeria Rauf,Nain Abidi,N Boochatham,1,0,"",""
ball,1,1.4,Pakistan,Nain Abidi,Javeria Rauf,N Boochatham,1,0,"",""
ball,1,1.5,Pakistan,Javeria Rauf,Nain Abidi,N Boochatham,1,0,"",""
ball,1,1.6,Pakistan,Nain Abidi,Javeria Rauf,N Boochatham,0,0,"",""
ball,1,2.1,Pakistan,Javeria Rauf,Nain Abidi,N Chantam,1,0,"",""
ball,1,2.2,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,0,0,"",""
ball,1,2.3,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,2,0,"",""
ball,1,2.4,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,0,0,"",""
ball,1,2.5,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,4,0,"",""
ball,1,2.6,Pakistan,Nain Abidi,Javeria Rauf,N Chantam,0,0,"",""
ball,1,3.1,Pakistan,Javeria Rauf,Nain Abidi,N Boochatham,0,1,"",""
ball,1,3.2,Pakistan,Nain Abidi,Javeria Rauf,N Boochatham,4,0,"",""
```

We consolidate the csv files of all the matches to create a single csv file with all the relevant features. This can be done using the code below.

In [ ]:

```
import csv
import sys
import os

csvFiles = os.listdir(sys.argv[1])
#csvFiles.sort()
csvFileN = []
for fileN in csvFiles:
    csvFileN.append(int(fileN.split(".")[0]))
csvFileN.sort()
csvFiles = []
for fileN in csvFileN:
    csvFiles.append(str(fileN)+".csv")

#print "after sorted\n",csvFiles
path = sys.argv[1]
f = open(sys.argv[2], 'a')
writer = csv.writer(f)
writer.writerow( ('match_id','season','inning','team1','team2','batting_team','bowling_team','over','ball','ba
tsman_runs','extra_runs','total_runs','player_dismissed','winner') )
id = 0
for fileN in csvFiles:
    id = id+1
    with open(path+fileN) as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')

        i=0
        team1 = ""
        team2 = ""
        winner = ""
        batting_team = ""
        bowling_team = ""
        count = 0
        season = ""
        count1 = 0
        for row in readCSV:
            if len(row)>2 and row[0] == "info":
                if count == 0 and row[1] == "team":
                    team1 = row[2]
                    count = 1
                elif count == 1 and row[1] == "team":
                    team2 = row[2]
                elif row[1] == "winner":
                    winner = row[2]
                elif row[1] == "date":
                    season = row[2]
            elif len(row)>0 and row[0] == "ball" and row[1] == "1":
                if row[3]==team1:
                    batting_team = team1
                    bowling_team = team2
                else:
                    batting_team = team2
                    bowling_team = team1
                if count1 == 0:
                    team1 = batting_team
                    team2 = bowling_team
                    count1=1
                #print id, row[1], batting_team, bowling_team, row[2].split('.')[0], row[2].split('.')[1], row
[-4],row[-3], int(row[-4])+int(row[-3]), "Nan" if row[-1]=="" else row[-1], winner
                writer.writerow( (id, season, row[1], team1, team2, batting_team, bowling_team, int(row[2].spl
it('.')[0])+1, row[2].split('.')[1], row[-4],row[-3], int(row[-4])+int(row[-3]), 0 if row[-1]=="" else 1, winn
er) )
            elif len(row)>0 and row[0] == "ball" and row[1] == "2":
                #print id, row[1], bowling_team, batting_team,  row[2].split('.')[0], row[2].split('.')[1], ro
w[-4],row[-3], int(row[-4])+int(row[-3]), "Nan" if row[-1]=="" else row[-1], winner
                writer.writerow((id, season, row[1], team1, team2, bowling_team, batting_team,  int(row[2].spl
it('.')[0])+1, row[2].split('.')[1], row[-4],row[-3], int(row[-4])+int(row[-3]), 0 if row[-1]=="" else 1, winn
er))
f.close()
```

After all the csv files are combined using the above code, we get a single csv file which has all the important features. The file look like this



```
match_id,season,inning,team1,team2,batting_team,bowling_team,over,ball,batsman_runs,extra_runs,total_runs,player_dismissed,winner
1,2005/06/13,1,England,Australia,England,Australia,1,1,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,2,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,3,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,4,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,5,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,6,0,1,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,1,7,2,0,2,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,1,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,2,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,3,0,1,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,4,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,5,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,6,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,2,7,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,1,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,2,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,3,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,4,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,5,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,3,6,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,4,1,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,4,2,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,4,3,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,4,4,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,4,5,0,0,0,1,England
1,2005/06/13,1,England,Australia,England,Australia,4,6,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,1,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,2,1,1,2,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,3,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,4,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,5,0,1,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,6,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,7,0,1,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,5,8,3,0,3,0,England
1,2005/06/13,1,England,Australia,England,Australia,6,1,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,6,2,0,0,0,0,England
1,2005/06/13,1,England,Australia,England,Australia,6,3,4,0,4,0,England
1,2005/06/13,1,England,Australia,England,Australia,6,4,1,0,1,0,England
1,2005/06/13,1,England,Australia,England,Australia,6,5,0,0,0,1,England
1,2005/06/13,1,England,Australia,England,Australia,6,6,1,0,1,0,England
```

```
In [ ]:
```

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 [333]:
```
#Now, we can use this information to compute new features which can be used for training the model

#Here the total cummulative runs in each innings and wickets till a given point in the match is found
train['innings_wickets'] = train.groupby(['match_id', 'inning'])['player_dismissed'].cumsum()
train['innings_score'] = train.groupby(['match_id', 'inning'])['total_runs'].cumsum()
# train[train.match_id == 1]
train.sample(n=10)
```

Out[333]:

	match_id	inning	over	team1	team2	batting_team	winner	total_runs	player_dismissed	innings_wickets	innings_s
2025	58	1	13	South Africa	Australia	South Africa	South Africa	2	0	3	83
21040	574	2	9	West Indies	Afghanistan	Afghanistan	West Indies	11	0	6	52
13103	361	1	20	Bangladesh	Australia	Bangladesh	Australia	9	1	5	153
3391	94	2	3	Pakistan	Sri Lanka	Sri Lanka	Sri Lanka	14	0	0	29
9806	273	2	4	Australia	Sri Lanka	Sri Lanka	Sri Lanka	5	0	0	32
10089	281	2	13	Kenya	Canada	Canada	Canada	2	0	3	81
14082	388	1	12	Nepal	Hong Kong	Nepal	Hong Kong	2	1	8	25
8899	247	2	5	Afghanistan	Ireland	Ireland	Ireland	17	0	2	54
14948	411	1	3	Hong Kong	Scotland	Hong Kong	Scotland	0	1	2	21
9575	266	1	14	Pakistan	South Africa	Pakistan	Pakistan	19	0	2	154

In [334]:

```
#To find the the current situation of the match, we need to know the total runs of target
temp = train.groupby(['match_id', 'inning'])['total_runs'].sum().reset_index()
temp = temp.loc[temp['inning']==1,:]
temp['inning'] = 2 #Since target is after the first innings is over
temp.columns = ['match_id', 'inning', 'score_target']
train = train.merge(temp, how='left', on = ['match_id', 'inning'])
train['score_target'].fillna(-1, inplace=True)

#The second team is judged on the basis of numbers of REMAINING runs required to win

def get_remaining_target(row):
    if row['score_target'] == -1.:
        return -1
    else:
        return row['score_target'] - row['innings_score']

train['remaining_target'] = train.apply(lambda row: get_remaining_target(row),axis=1)

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

Out[334]:

er	team1	team2	batting_team	winner	total_runs	player_dismissed	innings_wickets	innings_score	score_target	remainii
	West Indies	Afghanistan	Afghanistan	West Indies	12	1	8	65	112.0	47.0
	Hong Kong	United Arab Emirates	Hong Kong	United Arab Emirates	7	0	1	11	-1.0	-1.0
	Australia	New Zealand	New Zealand	Australia	13	0	0	49	214.0	165.0
	South Africa	England	England	South Africa	6	0	2	133	174.0	41.0
	New Zealand	Bangladesh	New Zealand	New Zealand	4	0	3	129	-1.0	-1.0
	Pakistan	Zimbabwe	Zimbabwe	Pakistan	11	0	1	14	136.0	122.0
	Sri Lanka	Australia	Sri Lanka	Australia	5	1	4	26	-1.0	-1.0
	New Zealand	Pakistan	Pakistan	Pakistan	9	1	3	70	99.0	29.0
	Afghanistan	Scotland	Afghanistan	Afghanistan	24	0	1	31	-1.0	-1.0
	India	Sri Lanka	India	India	4	0	2	124	-1.0	-1.0

In [335]:
```python
'''
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]:

ıs	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

Making Baseline Model

Now selecting some features that can be used to train our Baseline model. One of the most relevant feature is "Remaining Target" where the total number of runs required to win the match is computed at every instant. This helps the model to understand the cases where teams with previous such target requirements had performed.

In [374]:
```
x_cols = ['inning', 'over', 'remaining_target','is_batting_team']

#Taking all the matches except the last one for training
validation = train.loc[train.match_id == 576,:] #Last match of the file is kept as testing
training = train.loc[train.match_id != 576,:]

# creating the input and target variables
training_X = np.array(training[x_cols[:]])
training_y = np.array(training['target'])

validation_X = np.array(validation[x_cols[:]])
validation_y = np.array(validation['target'])

print(training_X.shape, training_y.shape)
print(validation_X.shape, validation_y.shape)
```

```
((21086, 4), (21086,))
((39, 4), (39,))
```

In [375]:
```
#Predicting the probability of the team winning in each over

logistic = LogisticRegression()
logistic.fit(training_X,training_y)
prediction= logistic.predict_proba(validation_X)
print(prediction)
```

```
[[ 0.66267992  0.33732008]
 [ 0.64508046  0.35491954]
 [ 0.6270795   0.3729205 ]
 [ 0.60871934  0.39128066]
 [ 0.59004619  0.40995381]
 [ 0.57110977  0.42889023]
 [ 0.55196284  0.44803716]
 [ 0.5326606   0.4673394 ]
 [ 0.51326009  0.48673991]
 [ 0.49381955  0.50618045]
 [ 0.47439767  0.52560233]
 [ 0.45505296  0.54494704]
 [ 0.43584295  0.56415705]
 [ 0.41682361  0.58317639]
 [ 0.39804864  0.60195136]
 [ 0.37956892  0.62043108]
 [ 0.36143201  0.63856799]
 [ 0.34368167  0.65631833]
 [ 0.32635754  0.67364246]
 [ 0.30949488  0.69050512]
 [ 0.15559803  0.84440197]
 [ 0.16893807  0.83106193]
 [ 0.1703753   0.8296247 ]
 [ 0.22394045  0.77605955]
 [ 0.27513729  0.72486271]
 [ 0.3233182   0.6766818 ]
 [ 0.35014793  0.64985207]
 [ 0.36767525  0.63232475]
 [ 0.37005038  0.62994962]
 [ 0.39847447  0.60152553]
 [ 0.43839921  0.56160079]
 [ 0.47913785  0.52086215]
 [ 0.4707126   0.5292874 ]
 [ 0.53913309  0.46086691]
 [ 0.59556056  0.40443944]
 [ 0.65455983  0.34544017]
 [ 0.70459824  0.29540176]
 [ 0.71125526  0.28874474]
 [ 0.74376927  0.25623073]]
```

The result obtained above is in the form of a two dimentional matrix. We have selected the 'Target' with respect to Team1,hence we get all these probabilities with respect to Team1. So, the first column of this two dimensional matrix represents the probability of Team1 Losing and the second column is the probability of Team1 Winning. Consequently, as the probability values are complementary to each other, the first column represnts the probability of Team2 Winning while the second column represents the probbility of Team2 Losing.

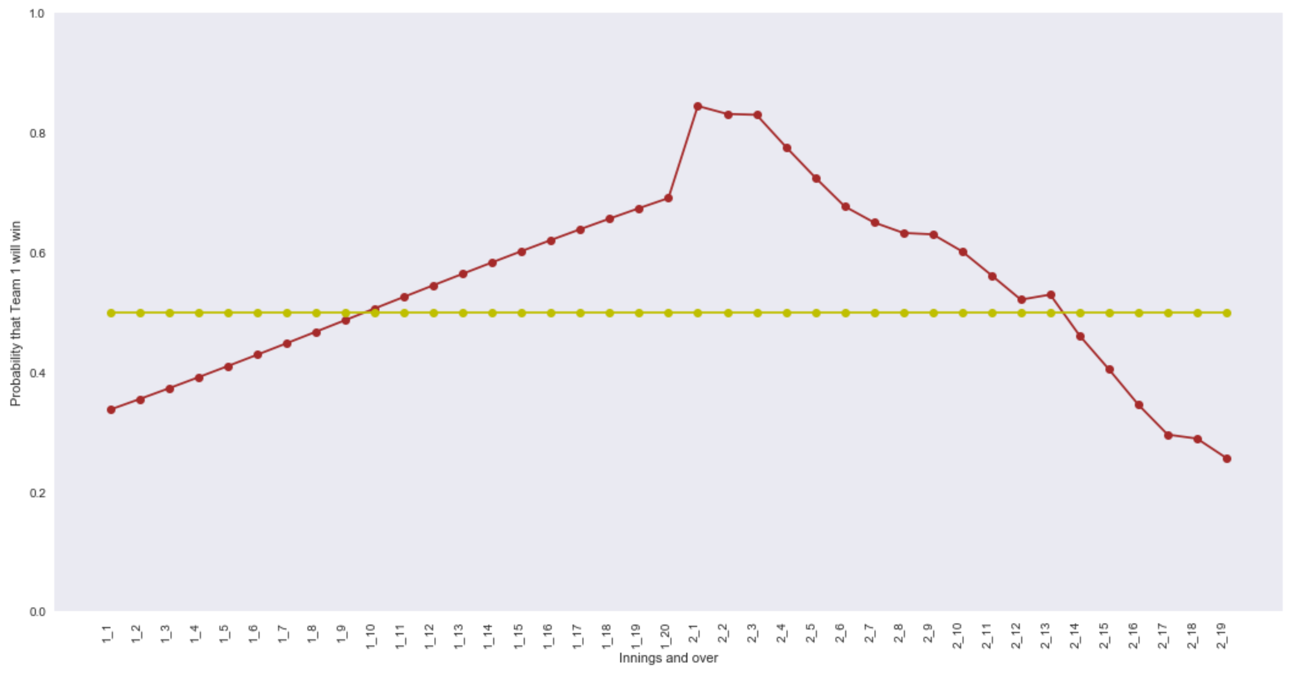
All this classification is due to the Logistic regression as helps us predict which datapoint belongs to the 0th class or 1th class where 0 represents losing for us while 1 represents winning. Logistic regression is a statistical method to analyze datset with multiple independent variables. In this regression, the outcome is binary. The goal is to find best fitting model that relates the dichotomoous variable with the set of independent variables. Logistic regression generates coefficients to predict a logit transformation of probability :

logit(p) = $b_o+b_1X_1+b_2X_2+...+b_kX_k$
logit(p) = ln(p/1-p)

$$\left[\begin{array}{cc} \text{P(Team1Loses)} & \text{P(Team1Wins)} \\ n1 & 1\text{-}n1 \\ n2 & 1\text{-}n2 \\ n3 & 1\text{-}n3 \\ . & . \\ n4 & 1\text{-}n4 \\ \text{P(Team2 Wins)} & \text{P(Team2 Loses)} \end{array}\right]$$

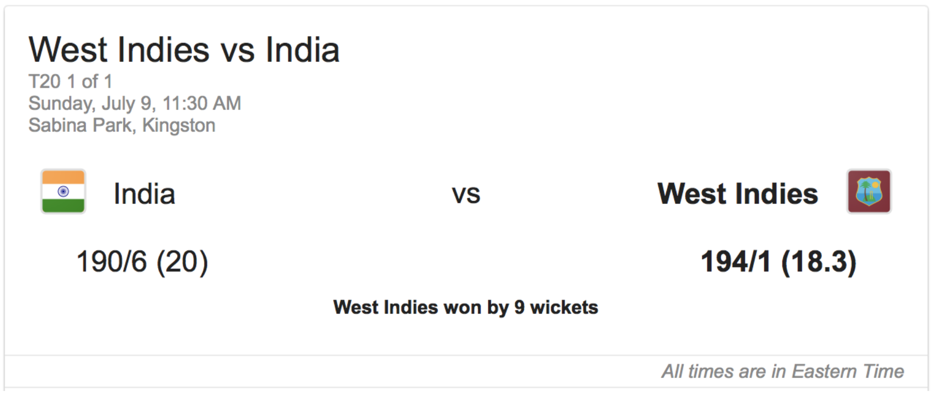
Reference e.g : Predicted class [2], real class 2
Probabilities for each class from 0 to 2
[[0.00168787 0.28720074 0.71111138]]
www.dummies.com/programming/big-data/data-science/using-logistic-regression-in-python-for-data-science/

```
In [376]: #Now as we have the probabilites of winning at each over, let us draw a plot to show this over-by-over

          fig, ax1 = plt.subplots(figsize=(18,9))
          labels = np.array(validation.apply(lambda row: str(row['inning']) + "_" + str(row['over']), axis=1))
          ind = np.arange(len(labels))
          width = 0.7
          ax1.set_xticks(ind+((width)/2.))
          ax1.set_ylim([0,1])

          ax1.set_xticklabels(labels, rotation='vertical')
          ax1.set_ylabel("Probability that Team 1 will win")
          ax1.set_xlabel("Innings and over")
          probs=np.array(prediction)[:,1]
          #probs=np.append(probs,1)
          ax1.plot(ind+0.45, probs, color='brown', marker='o')
          ax1.plot(ind+0.45, np.array([0.5]*probs.shape[0]), color='y', marker='o')
          ax1.grid(b=False)
          plt.show()
```



This chart shown above gives us the probability of the First team winning in every over of the match.

It is the analysis of the latest T20 match played internationally in 2017. The game is played between India(Team1) and West Indies(Team2). The probabilities of India winning in each of the over is computed and shown in the given diagram. The end result of the game is that "West Indies" wins the match. Hence, in the last few overs, the probability that India wins plummets.



West Indies vs India

T20 1 of 1
Sunday, July 9, 11:30 AM
Sabina Park, Kingston

India vs **West Indies**

190/6 (20) **194/1 (18.3)**

West Indies won by 9 wickets

All times are in Eastern Time

Building a sophisticated model

The current features are not significant enough to develop an accurate prediction. Also, the graph obtained is too general to get any insights. It does not react to sudden changes that have happened in the match such as Hitting of six or Falling of Wickets. Hence, we try to build a more sophisticated model based on some more relevant and insightful featues.

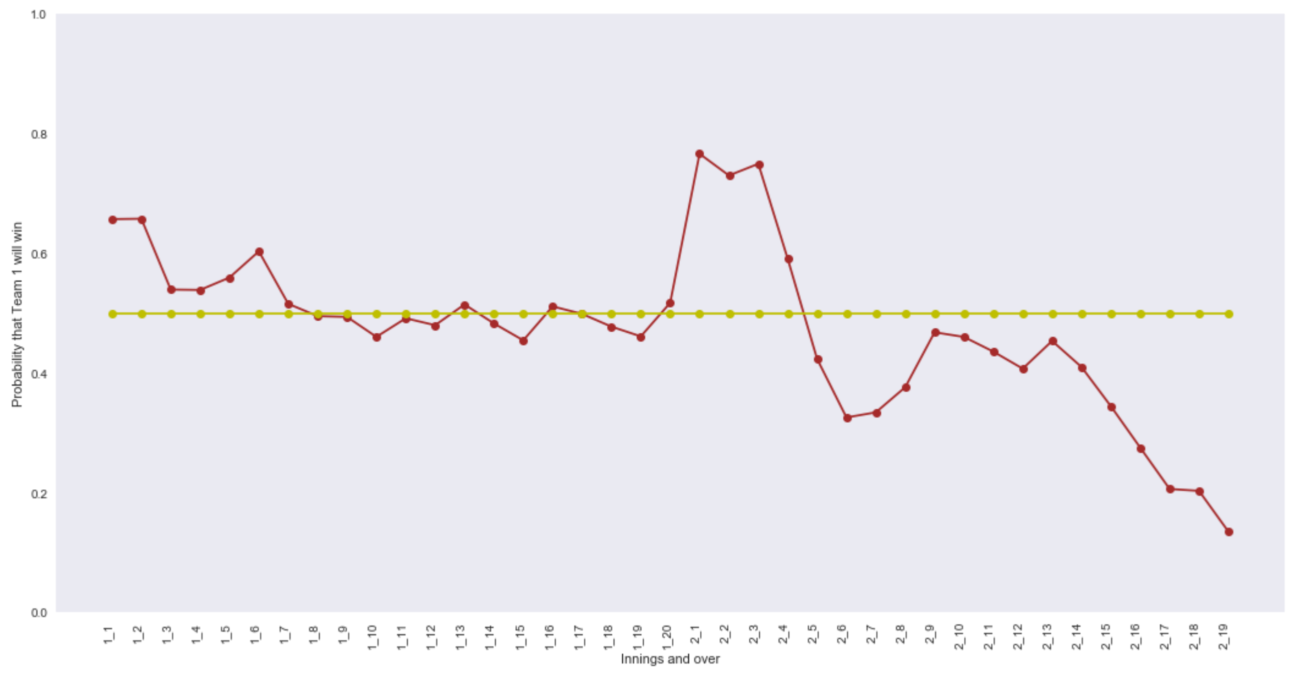
In [377]:
```
x_cols = ['inning', 'over', 'total_runs', 'player_dismissed', 'innings_wickets', 'innings_score', 'score_targe
t', 'remaining_target', 'run_rate', 'required_run_rate', 'runrate_diff', 'is_batting_team']

# let us take all the matches but for the final as development sample and final as val sample #
validation = train.loc[train.match_id == 576,:]
training = train.loc[train.match_id != 576,:]

# create the input and target variables #
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]
print(training_X.shape, training_y.shape)
print(validation_X.shape, validation_y.shape)
```

```
((21086, 12), (21086,))
((39, 12), (39,))
```

In [378]:
```
#Implementing the Logistic Regression Model on these additional features

logistic = LogisticRegression()
logistic.fit(training_X,training_y)
prediction= logistic.predict_proba(validation_X)
print(prediction)
```

```
[[ 0.14296584  0.85703416]
 [ 0.14229777  0.85770223]
 [ 0.26064129  0.73935871]
 [ 0.26127545  0.73872455]
 [ 0.24050922  0.75949078]
 [ 0.19673915  0.80326085]
 [ 0.28445559  0.71554441]
 [ 0.30500208  0.69499792]
 [ 0.30627194  0.69372806]
 [ 0.33963983  0.66036017]
 [ 0.30857846  0.69142154]
 [ 0.32015015  0.67984985]
 [ 0.28550493  0.71449507]
 [ 0.31717711  0.68282289]
 [ 0.34546773  0.65453227]
 [ 0.28875442  0.71124558]
 [ 0.30128156  0.69871844]
 [ 0.32255321  0.67744679]
 [ 0.33887148  0.66112852]
 [ 0.2827653   0.7172347 ]
 [ 0.03377128  0.96622872]
 [ 0.07006171  0.92993829]
 [ 0.05055255  0.94944745]
 [ 0.20918371  0.79081629]
 [ 0.3764304   0.6235696 ]
 [ 0.47428581  0.52571419]
 [ 0.46574425  0.53425575]
 [ 0.42373576  0.57626424]
 [ 0.33197054  0.66802946]
 [ 0.3397752   0.6602248 ]
 [ 0.36429578  0.63570422]
 [ 0.39274644  0.60725356]
 [ 0.3464275   0.6535725 ]
 [ 0.38997924  0.61002076]
 [ 0.45554982  0.54445018]
 [ 0.52476395  0.47523605]
 [ 0.59376442  0.40623558]
 [ 0.59688261  0.40311739]
 [ 0.66490822  0.33509178]]
```

```
In [380]: #Lets plot this Probabilities to see if it has performed better than our Baseline model



          fig, ax1 = plt.subplots(figsize=(18,9))
          labels = np.array(validation.apply(lambda row: str(row['inning']) + "_" + str(row['over']), axis=1))
          ind = np.arange(len(labels))
          width = 0.7
          ax1.set_xticks(ind+((width)/2.))
          ax1.set_ylim([0,1])

          ax1.set_xticklabels(labels, rotation='vertical')
          ax1.set_ylabel("Probability that Team 1 will win")
          ax1.set_xlabel("Innings and over")
          probs=np.array(prediction)[:,1]

          ax1.plot(ind+0.45, probs, color='brown', marker='o')
          ax1.plot(ind+0.45, np.array([0.5]*probs.shape[0]), color='y', marker='o')
          ax1.grid(b=False)
          plt.show()
```



This graph is for the same match as discussed above(T20 between India and West Indies). However, this is a more accurate representation of the actual gameplay that had occured. It is changing drastically taking account of the wickets fallen and the runs scored in that over.We have used logistic regression for training on all the matches except this one to obtain the probabilities of winning per over.

For e.g. At Innings 2_1 given above, the probability of India winnning has increased significantly. This is due the massive 13 runs scored in that over itself by Virat Kohli and Shikhar Dhawan. This gave India an amazing headstart as the score for that over was way above the required run rate for that over and also since it was the first over, there were no wickets fallen as well. The scorecard for the given match can be seen with the given link:

www.espncricinfo.com/series/11124/game/1098211/West-Indies-vs-India-Only-T20I-wi-v-ind-2017

Finding Percentage Times a Team Wins for a Given Probability

As suggested by the professor, we can find the number of times a team ends up winning a match when it has a specific probability during a match

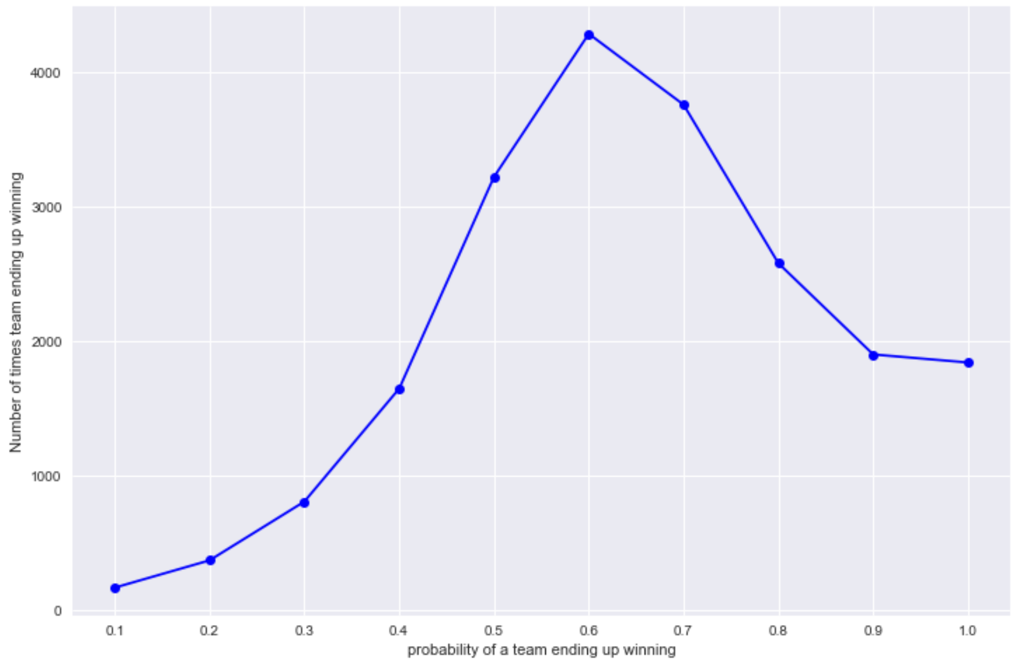
This can be computed by testing the Model on each and every match by training on all other remaing matches. For our case, the model is trained and tested on 577 different times using each match as testing in each iteration.

In [342]:
```
ids=train.match_id.unique()
logistic = LogisticRegression()
cp=[]
ap=[]

train['is_team1_won']=train['team1']==train['winner']


for id in ids:
    validation = train.loc[train.match_id == id,:]
    training = train.loc[train.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]
    logistic.fit(training_X,training_y)
    prediction=logistic.predict_proba(validation_X)
    np_prediction=np.array(prediction)

    ap.extend(np_prediction[:,1])
    ap.extend(np_prediction[:,0])
    cp.extend(np_prediction[:,1 if train[train.match_id==id].is_team1_won.unique() else 0])
```

```
In [343]: # print(len(cp))

          dict={}
          for i in [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]:
              dict[i]=0
          #print(dict)

          for i in cp:
              j=math.ceil(i*10)/10
              dict[j]=dict[j]+1



          print('id=',id)

          dict = collections.OrderedDict(sorted(dict.items()))
          print(dict)

          x=list(dict.keys())
          y=list(dict.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_xlabel('probability of a team ending up winning')
          ax.set_ylabel('Number of times team ending up winning')
```

```
          ('id=', 576)
          OrderedDict([(0.1, 166), (0.2, 369), (0.3, 806), (0.4, 1646), (0.5, 3220), (0.6, 4284), (0.7, 3757), (0.8, 25
          78), (0.9, 1900), (1.0, 1840)])
          [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
          [166, 369, 806, 1646, 3220, 4284, 3757, 2578, 1900, 1840]
```

```
Out[343]: <matplotlib.text.Text at 0x144cdca50>
```



In the above graph, we are trying to plot the event in which probability that team would win being at a certain threshold vs number of occurences in which the team won with the predicted probability. To get this stats, in each iteration we selected a match and trained on all other matches and made the prediction on the selected match. In the graph we see that , the number of occurences increase with the increase in threshold, showing that if model predicted higher probability than there are more chances of team to win.

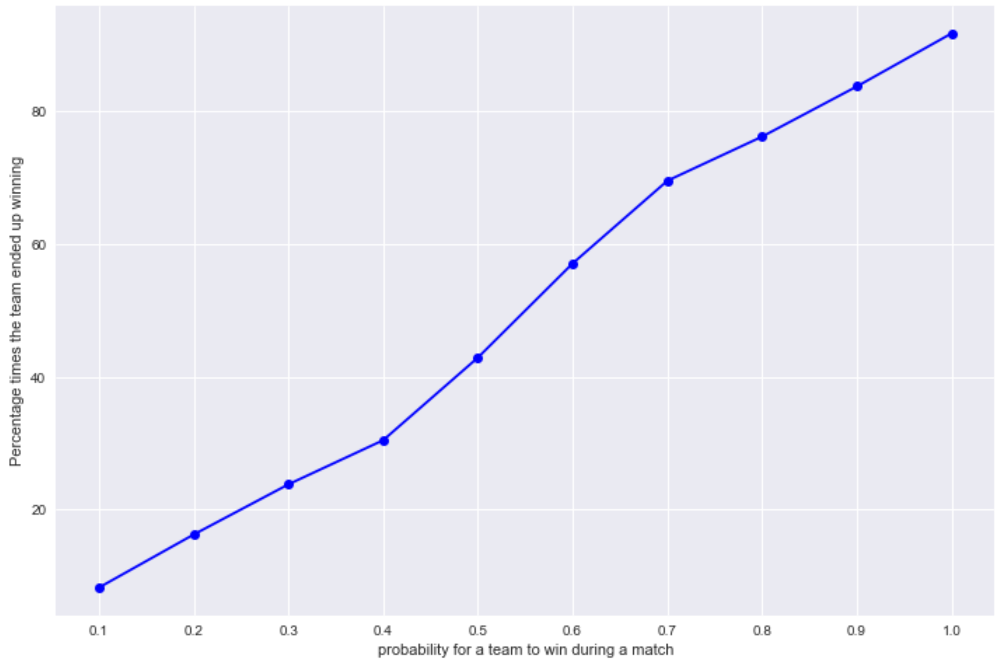
However, we also see an anomality here. We see that the number of occurences decrease above the probability threshold of 0.6. This occurs because there are fewer instances where the model has predicted the Team to win with very High confidence and with high probability.

For e.g: It is certain that the Team would have won when it has observed 1.0 probability of winning. However, the model has never been able to predict a winning of the team with such high confidence. Hence, the number of occurances of 1.0 is very low.

In [398]:
```python
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]:
    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]
[0.08275174476570289, 0.1626267078007933, 0.23817966903073287, 0.30464556727743847, 0.4291044776119403, 0.570
8955223880597, 0.6953544327225616, 0.7618203309692672, 0.8373732921992068, 0.9172482552342971]
```

Out[398]:
```
<matplotlib.text.Text at 0x166133050>
```



To rectify the issue observed in the graph before, 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.