In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val
from sklearn import svm, grid_search, preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

```
C:\Users\Adhithya\Anaconda3\lib\site-packages\sklearn\cross_validation.py:4
1: DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and fun
ctions are moved. Also note that the interface of the new CV iterators are
different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Adhithya\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: De
precationWarning: This module was deprecated in version 0.18 in favor of th
e model_selection module into which all the refactored classes and function
s are moved. This module will be removed in 0.20.
  DeprecationWarning)
C:\Users\Adhithya\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boost
ing.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy
module and should not be imported. It will be removed in a future NumPy rel
ease.
  from numpy.core.umath_tests import inner1d
```

In [2]:
```
reg = pd.read_csv("Prelim2019_RegularSeasonDetailedResults.csv")
reg.columns
```

Out[2]:
```
Index(['Season', 'DayNum', 'WTeamID', 'WScore', 'LTeamID', 'LScore', 'WLo
c',
       'NumOT', 'WFGM', 'WFGA', 'WFGM3', 'WFGA3', 'WFTM', 'WFTA', 'WOR', 'W
DR',
       'WAst', 'WTO', 'WStl', 'WBlk', 'WPF', 'LFGM', 'LFGA', 'LFGM3', 'LFGA
3',
       'LFTM', 'LFTA', 'LOR', 'LDR', 'LAst', 'LTO', 'LStl', 'LBlk', 'LPF'],
      dtype='object')
```

In [3]:
```
tourney = pd.read_csv("NCAATourneyDetailedResults.csv")
y_tourney = tourney.loc[tourney['Season']==2018]
y_tourney
```

Out[3]:

	Season	DayNum	WTeamID	WScore	LTeamID	LScore	WLoc	NumOT	WFGM	WFGA
981	2018	134	1347	71	1254	61	N	0	28	59
982	2018	134	1382	65	1417	58	N	0	23	60
983	2018	135	1393	60	1113	56	N	0	20	49
984	2018	135	1411	64	1300	46	N	0	21	57
985	2018	136	1104	86	1439	83	N	0	30	50
986	2018	136	1138	89	1112	68	N	0	34	62
987	2018	136	1181	89	1233	67	N	0	36	67
988	2018	136	1196	77	1382	62	N	0	27	65
989	2018	136	1211	68	1422	64	N	0	25	59
990	2018	136	1222	67	1361	65	N	0	22	59
991	2018	136	1242	76	1335	60	N	0	27	61
992	2018	136	1246	78	1172	73	N	0	26	51
993	2018	136	1260	64	1274	62	N	0	26	55
994	2018	136	1276	61	1285	47	N	0	21	47
995	2018	136	1326	81	1355	73	N	0	27	72
996	2018	136	1348	83	1328	78	N	1	30	77
997	2018	136	1371	94	1301	83	N	0	28	58
998	2018	136	1397	73	1460	47	N	0	27	60
999	2018	136	1403	70	1372	60	N	0	25	54
1000	2018	136	1437	87	1347	61	N	0	31	52
1001	2018	137	1120	62	1158	58	N	0	21	59
1002	2018	137	1139	79	1116	62	N	0	29	59
1003	2018	137	1153	68	1209	53	N	0	24	62
1004	2018	137	1155	79	1308	68	N	0	33	59
1005	2018	137	1199	67	1281	54	N	0	21	49
1006	2018	137	1243	69	1166	59	N	0	23	50
1007	2018	137	1267	81	1455	75	N	0	28	60
1008	2018	137	1277	82	1137	78	N	0	31	58
1009	2018	137	1305	87	1400	83	N	1	32	63
1010	2018	137	1314	84	1252	66	N	0	31	60
...	...	...	...	...	...	...	...	...	...	...
1018	2018	138	1211	90	1326	84	N	0	31	58

	Season	DayNum	WTeamID	WScore	LTeamID	LScore	WLoc	NumOT	WFGM	WFGA
1019	2018	138	1242	83	1371	79	N	0	28	56
1020	2018	138	1246	95	1138	75	N	0	36	64
1021	2018	138	1260	63	1397	62	N	0	22	44
1022	2018	138	1276	64	1222	63	N	0	21	59
1023	2018	138	1403	69	1196	66	N	0	28	63
1024	2018	138	1437	81	1104	58	N	0	25	63
1025	2018	139	1155	84	1120	53	N	0	29	61
1026	2018	139	1199	75	1462	70	N	0	24	55
1027	2018	139	1243	50	1420	43	N	0	18	44
1028	2018	139	1305	75	1153	73	N	0	30	61
1029	2018	139	1345	76	1139	73	N	0	26	52
1030	2018	139	1393	55	1277	53	N	0	15	42
1031	2018	139	1401	86	1314	65	N	0	31	60
1032	2018	139	1452	94	1267	71	N	0	33	66
1033	2018	143	1199	75	1211	60	N	0	27	58
1034	2018	143	1243	61	1246	58	N	0	19	54
1035	2018	143	1260	69	1305	68	N	0	29	52
1036	2018	143	1276	99	1401	72	N	0	39	63
1037	2018	144	1181	69	1393	65	N	0	22	56
1038	2018	144	1242	80	1155	76	N	0	28	60
1039	2018	144	1403	78	1345	65	N	0	28	59
1040	2018	144	1437	90	1452	78	N	0	27	54
1041	2018	145	1260	78	1243	62	N	0	27	47
1042	2018	145	1276	58	1199	54	N	0	19	49
1043	2018	146	1242	85	1181	81	N	1	30	69
1044	2018	146	1437	71	1403	59	N	0	19	57
1045	2018	152	1276	69	1260	57	N	0	25	59
1046	2018	152	1437	95	1242	79	N	0	36	65
1047	2018	154	1437	79	1276	62	N	0	27	57

67 rows × 34 columns

In [4]:
```
teams = pd.read_csv('Teams.csv')
```

In [5]:

```
def getTeamData(dataset,team):
    team_name = teams[teams['TeamID']==team].iloc[0,1]
    games = dataset[(dataset['WTeamID']==team) | (dataset['LTeamID']==team)]
    length = len(games)
    if len(games) > 0:
        wins = games[games['WTeamID']==team]
        losses = games[games['LTeamID']==team]
        total_wins = len(wins)
        tot_wins = sum(wins['WScore'])
        tot_loss = sum(losses['LScore'])
        opp_wins = sum(wins['LScore'])
        opp_loss = sum(losses['WScore'])
        ppg = (tot_wins + tot_loss)/length
        points_given = (opp_wins + opp_loss)/length
        FGPerc = (sum(wins['WFGM']) + sum(losses['LFGM']))/ (sum(wins['WFGA']
        FG3Perc = (sum(wins['WFGM3']) + sum(losses['LFGM3']))/ (sum(wins['WFG
        ast = (sum(wins['WAst']) + sum(losses['LAst']))
        rbs = (sum(wins['WOR']) + sum(wins['WOR']) + sum(losses['LOR'])) + su
        to = (sum(wins['WTO']) + sum(losses['LTO']))
        return [total_wins,ppg,points_given,FGPerc,FG3Perc,ast,rbs,to]
    return None
```

In [6]:

```
team_data = pd.DataFrame(columns=['WTeamID','LTeamID','FTeam','STeam','TotWir
tourney_data = pd.DataFrame(columns=['WTeamID','LTeamID','FTeam','STeam','Tot

def makeDataset(w_team,l_team, reg_year, tourney_year, reg_tourney, win):
    regular = reg[reg['Season']==reg_year]
    tourney = reg[reg['Season']==tourney_year]

    if reg_tourney:
        dataset = regular
        pop = team_data
    else:
        dataset = tourney
        pop = tourney_data

    w_name = teams[teams['TeamID']==w_team].iloc[0,1]
    l_name = teams[teams['TeamID']==l_team].iloc[0,1]
    w_team_data = getTeamData(dataset, w_team)
    l_team_data = getTeamData(dataset, l_team)

    if win:
        w_data = np.array(w_team_data) - np.array(l_team_data)
        w_data = np.append(w_data,1)
        w_data = np.append(np.array([w_team,l_team,w_name,l_name]),w_data)
        pop.loc[-1] = w_data
        pop.index += 1
    else:
        l_data = np.array(l_team_data) - np.array(w_team_data)
        l_data = np.append(l_data,0)
        l_data = np.append(np.array([w_team,l_team,l_name,w_name]),l_data)
        pop.loc[-1] = l_data
        pop.index += 1
    return
```

In [7]:
```
def populate(reg_year, tourney_year):
    reg = pd.read_csv("Prelim2019_RegularSeasonDetailedResults.csv")
    y_reg = reg[reg['Season']==reg_year]

    tourney = pd.read_csv("NCAATourneyDetailedResults.csv")
    y_tourney = tourney[tourney['Season']==tourney_year]

    reg_win = True
    tourney_win = True
    for i,row in y_reg.iterrows():
        w_team = row['WTeamID']
        l_team = row['LTeamID']
        makeDataset(w_team,l_team, reg_year, tourney_year, True, reg_win)
        reg_win = not(reg_win)

    for i,row in y_tourney.iterrows():
        w_team = row['WTeamID']
        l_team = row['LTeamID']
        makeDataset(w_team, l_team, reg_year, tourney_year, False, tourney_wi
        tourney_win = not(tourney_win)

populate(2018,2018)
```

In [8]:
```
tourney_data
```

Out[8]:

	WTeamID	LTeamID	FTeam	STeam	TotWins	PPG	
66	1347	1254	Radford	Long Island	3.0	-9.759469696969703	-11.9
65	1382	1417	UCLA	St Bonaventure	-4.0	4.0	
64	1393	1113	Syracuse	Arizona St	0.0	-15.970674486803517	-10.7
63	1411	1300	NC Central	TX Southern	1.0	-7.259962049335854	-9.0
62	1104	1439	Alabama	Virginia Tech	-2.0	-7.365808823529406	-1.7
61	1138	1112	Arizona	Buffalo	2.0	-3.8146167557932387	-5.3
60	1181	1233	Duke	Iona	6.0	4.848484848484858	-6.5
59	1196	1382	St Bonaventure	Florida	5.0	1.84375	
58	1211	1422	Gonzaga	UNC Greensboro	6.0	12.370967741935488	3.7

In [9]:
```
# print(len(team_data))
# small = team_data.iloc[0:9000, :]
small=team_data
small
```

Out[9]:

	WTeamID	LTeamID	FTeam	STeam	TotWins	PPG	
5404	1104	1272	Alabama	Memphis	-2.0	1.529411764705884	-1.20
5403	1107	1233	Iona	Albany NY	-1.0	6.525904203323549	8.2
5402	1112	1319	Arizona	Northern Arizona	23.0	15.849019607843132	-7.490
5401	1113	1226	Idaho St	Arizona St	-8.0	-10.444700460829495	-0.1797
5400	1116	1359	Arkansas	Samford	15.0	4.888235294117649	-8.40
5399	1120	1313	Norfolk St	Auburn	-12.0	-12.041666666666671	0.8
5398	1124	1146	Baylor	Cent Arkansas	1.0	-5.797379032258064	-10.17
5397	1127	1288	Morgan St	Binghamton	2.0	6.275862068965523	5.51
5396	1130	1263	Boston College	Maine	15.0	11.427450980392166	-2.292

In [10]:
```
data = small.iloc[:,4:12]
outcomes = small['Outcome']
X_train, X_test, y_train, y_test = train_test_split(data, outcomes, test_size
```

In [11]:
```
Cs = [0.001, 0.01, 0.1, 1, 10]
gammas = [0.01,0.1, 1]
param_grid = {'C': Cs, 'gamma' : gammas}
grid_search = GridSearchCV(svm.SVC(kernel='rbf'), param_grid)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

```
{'C': 1, 'gamma': 0.01}
```

In [12]:
```
# svc.fit(X_train, y_train)
# print(svc.score(X_test, y_test))

svc = svm.SVC(C=1, kernel='rbf', degree=0.01)
scores = cross_val_score(svc,data, outcomes,cv=7)
print(scores)
```

```
[0.5084088  0.50777202 0.52202073 0.5738342  0.54015544 0.5388601
 0.56088083]
```

In [13]:
```
log = LogisticRegression()
log.fit(X_train,y_train)
log.score(X_test,y_test)
```

Out[13]: 0.7392108508014796

In [14]:
```
rf = RandomForestClassifier(n_estimators=1000, max_depth=10)
rf.fit(X_train,y_train)
rf.score(X_test,y_test)
```

Out[14]: 0.7244143033292232

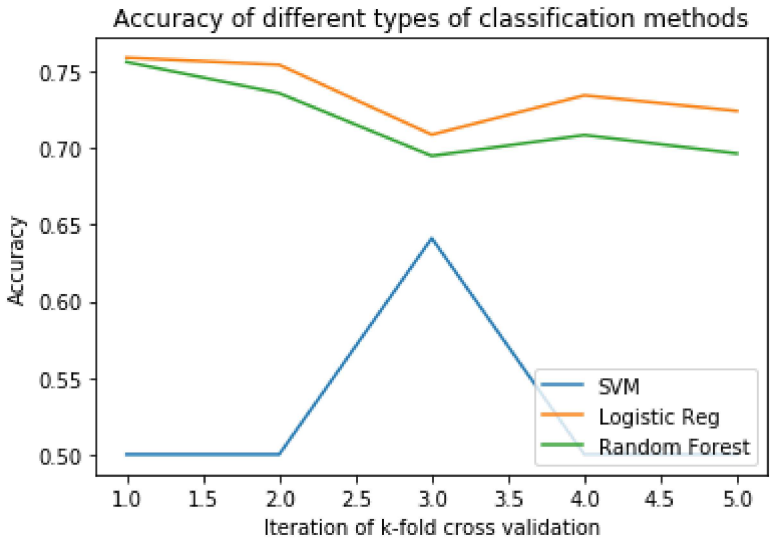
In [15]:
```
kfolds = StratifiedKFold(n_splits=5)
```

In [16]:
```
svm_scores = []
log_scores = []
rf_scores = []
for train_index,test_index in kfolds.split(data,outcomes):
    X_train, X_test, y_train, y_test = data.iloc[train_index], data.iloc[test
    svc = svm.SVC(C=0.1, kernel='rbf', degree=0.1)
    svc.fit(X_train,y_train)
    svm_scores.append(svc.score(X_test,y_test))
    log = LogisticRegression()
    log.fit(X_train,y_train)
    log_scores.append(log.score(X_test,y_test))
    rf = RandomForestClassifier(n_estimators=1000,max_depth=10)
    rf.fit(X_train,y_train)
    rf_scores.append(rf.score(X_test,y_test))
print(svm_scores)
print(log_scores)
print(rf_scores)
```

```
[0.5, 0.5, 0.6410730804810361, 0.5, 0.5]
[0.7587800369685767, 0.7541589648798521, 0.7086031452358927, 0.734259259259
2593, 0.7240740740740741]
[0.756007393715342, 0.7356746765249538, 0.6947271045328399, 0.7083333333333
334, 0.6962962962962963]
```

In [17]:
```
plt.plot(range(1,6),svm_scores,label='SVM')
plt.plot(range(1,6),log_scores,label='Logistic Reg')
plt.plot(range(1,6),rf_scores,label='Random Forest')
plt.title('Accuracy of different types of classification methods')
plt.xlabel('Iteration of k-fold cross validation')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

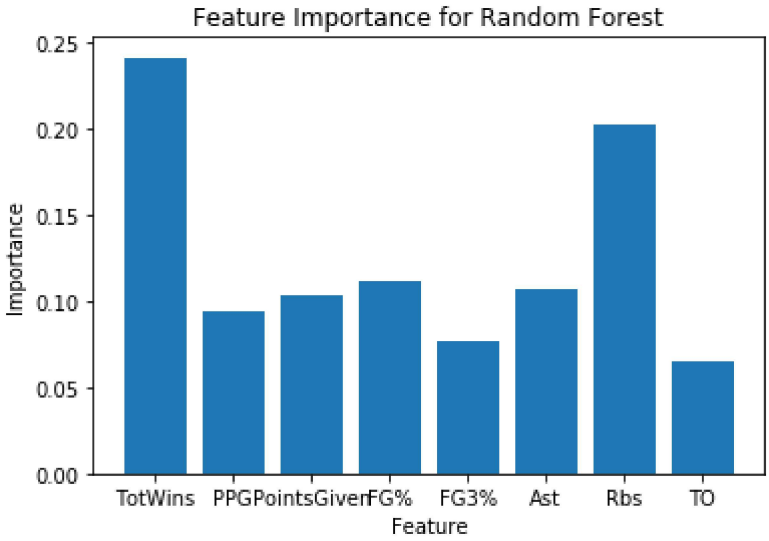
Out[17]: <matplotlib.legend.Legend at 0x1f5804bc080>



In [18]:
```
print(rf.feature_importances_)
print(rf.classes_)
plt.bar(data.columns,rf.feature_importances_)
plt.title('Feature Importance for Random Forest')
plt.xlabel('Feature')
plt.ylabel('Importance')
```

```
[0.24116866 0.09361379 0.10302133 0.11158397 0.07713035 0.10653974
 0.20228778 0.06465439]
['0.0' '1.0']
```

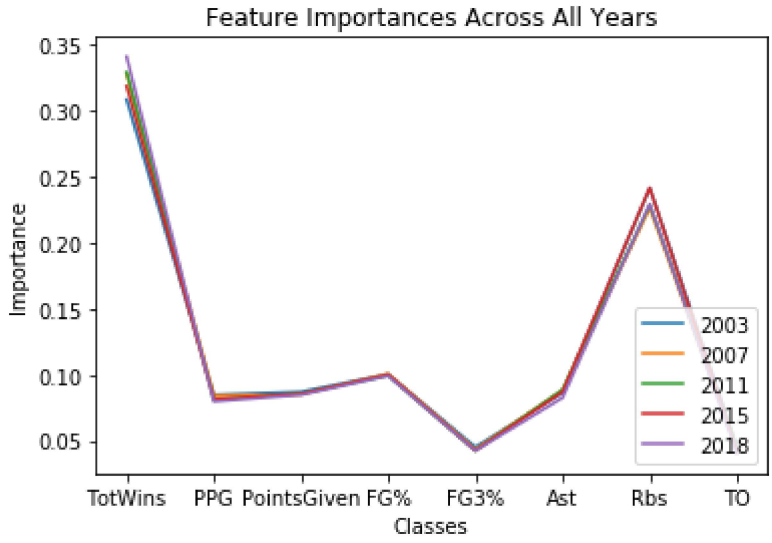
Out[18]: Text(0,0.5,'Importance')



In [24]:
```
feature_data = []
def feature_importances(reg_year,tourney_year):
    populate(reg_year,tourney_year)
    data = team_data.iloc[:,4:12]
    outcomes = team_data['Outcome']
    X_train, X_test, y_train, y_test = train_test_split(data, outcomes, test_
    rf = RandomForestClassifier(n_estimators=1000, max_depth=10)
    rf.fit(X_train,y_train)
    rf.score(X_test,y_test)
    features = rf.feature_importances_
    feature_data.append(features)
    return features
for i in [2003,2007,2011,2015,2018]:
    plt.plot(team_data.iloc[:,4:12].columns,feature_importances(i,i),label=st

plt.title('Feature Importances Across All Years')
plt.xlabel('Classes')
plt.ylabel('Importance')
plt.legend(loc='lower right')
```

Out[24]:
```
<matplotlib.legend.Legend at 0x1f58d56eeb8>
```



In [26]:
```
feature_data
```

Out[26]:
```
[array([0.30855997, 0.08488928, 0.08717956, 0.10055448, 0.0452646 ,
        0.08644862, 0.24157837, 0.04552512]),
 array([0.32804531, 0.08436316, 0.08509224, 0.10085157, 0.0436503 ,
        0.08827515, 0.22629429, 0.04342799]),
 array([0.32918581, 0.08112411, 0.08589497, 0.0994287 , 0.04244137,
        0.08885846, 0.22860069, 0.04446591]),
 array([0.31871235, 0.08156201, 0.08579163, 0.10028802, 0.04265662,
        0.08719159, 0.2416876 , 0.04211018]),
 array([0.34097741, 0.07983656, 0.08489189, 0.09927898, 0.04254565,
        0.08265555, 0.22937384, 0.04044012])]
```

In [32]:
```
predicted = log.predict(tourney_data.iloc[:,4:12])
actual = np.array(tourney_data['Outcome'])

count=0
total=0
for i in range(0,len(actual)):
    if actual[i] == predicted[i]:
        count+=1
    total+=1
count/total

# probs = rf.predict_proba(tourney_data.iloc[:,4:12])
# print(probs)
pd_conf = pd.DataFrame(columns=['Predicted Loss', 'Predicted Win'])
pd_conf.loc[0]=confusion_matrix(predicted,actual)[0]
pd_conf.loc[1]=confusion_matrix(predicted,actual)[1]
```

Out[32]: 5

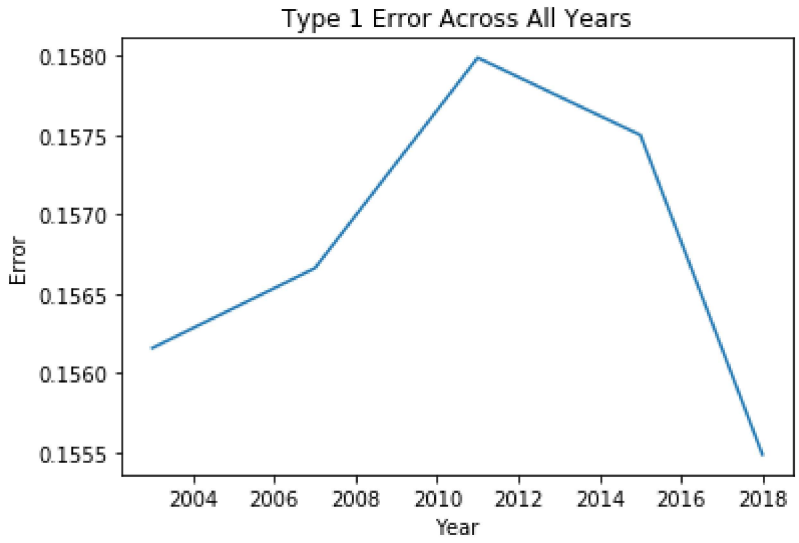
In [34]:
```
feature_data = []
def accuracy(reg_year,tourney_year):
    populate(reg_year,tourney_year)
    data = team_data.iloc[:,4:12]
    outcomes = team_data['Outcome']
    X_train, X_test, y_train, y_test = train_test_split(data, outcomes, test_
    log = LogisticRegression()
    log.fit(X_train,y_train)
    log.score(X_test,y_test)
    predicted = log.predict(tourney_data.iloc[:,4:12])
    actual = np.array(tourney_data['Outcome'])
    matrix = confusion_matrix(predicted,actual)
    type1 = matrix[0][1] / sum([sum(matrix[0]),sum(matrix[1])])
    return type1

all_errors = []
years = [2003,2007,2011,2015,2018]
for i in years:
    all_errors.append(accuracy(i,2018))

plt.plot(years,all_errors)

plt.title('Type 1 Error Across All Years')
plt.xlabel('Year')
plt.ylabel('Error')
```

Out[34]: Text(0,0.5,'Error')



In [20]:
```
predicted = rf.predict(tourney_data.iloc[:,4:12])
actual = np.array(tourney_data['Outcome'])
pd_conf = pd.DataFrame(columns=['Predicted Loss', 'Predicted Win'])
pd_conf.loc[0]=confusion_matrix(predicted,actual)[0]
pd_conf.loc[1]=confusion_matrix(predicted,actual)[1]
pd_conf
```

Out[20]:

	Predicted Loss	Predicted Win
0	20	11
1	13	23

In [21]:
```
predicted = svc.predict(tourney_data.iloc[:,4:12])
actual = np.array(tourney_data['Outcome'])
pd_conf = pd.DataFrame(columns=['Predicted Loss', 'Predicted Win'])
pd_conf.loc[0]=confusion_matrix(predicted,actual)[0]
pd_conf.loc[1]=confusion_matrix(predicted,actual)[1]
pd_conf
```

Out[21]:

	Predicted Loss	Predicted Win
0	0	0
1	33	34

In [102]:
```
pbp = pd.read_csv('Events_2018.csv')
short = pbp.iloc[0:100000,:]
```

In [77]:
```
reg18 = reg[reg['Season']==2018]
impt = short[['WTeamID','LTeamID','WPoints','LPoints','ElapsedSeconds']]
play_data = pd.DataFrame(columns=['WPoints','LPoints','ElapsedSeconds','TotWi
def train_data(df):
    win = True
    index = 0
    for i,row in df.iterrows():
        w_team_stats = np.array(getTeamData(reg18,row['WTeamID']))
        l_team_stats = np.array(getTeamData(reg18,row['LTeamID']))
        team_diff = l_team_stats - w_team_stats
        if win:
            team_diff = w_team_stats - l_team_stats
            team_diff = list(team_diff)
            team_diff.append(1)
        else:
            team_diff = list(team_diff)
            team_diff.append(0)
        win = not(win)
        row_data = list(row[['WPoints','LPoints','ElapsedSeconds']])
        row_data.extend(team_diff)
        play_data.loc[index] = row_data
        index += 1
    return play_data
full = train_data(impt)
condensed = full[(full['WPoints']!=0) | (full['LPoints']!=0)]
```

In [103]:
```
reg18 = reg[reg['Season']==2018]
impt = short[['WTeamID','LTeamID','WPoints','LPoints','ElapsedSeconds']]
play_data = pd.DataFrame(columns=['WPoints','LPoints','ElapsedSeconds','Outcc
def train_data(df,season, populate):
    win = True
    index = 0
    for i,row in df.iterrows():
        w_team_stats = np.array(getTeamData(season,row['WTeamID']))[0]
        l_team_stats = np.array(getTeamData(season,row['LTeamID']))[0]
#         team_diff = [l_team_stats - w_team_stats]
#         if win:
#             team_diff = [w_team_stats - l_team_stats]
#             team_diff.append(1)
#         else:
#             team_diff.append(0)
        if win:
            team_diff = [0]
        else:
            team_diff = [1]
        win = not(win)
        row_data = list(row[['WPoints','LPoints','ElapsedSeconds']])
        row_data.extend(team_diff)
        populate.loc[index] = row_data
        index += 1
    return populate
full = train_data(impt,reg18,play_data)
condensed = full[(full['WPoints']!=0) | (full['LPoints']!=0)]
```

In [112]:
```
condensed
```

Out[112]:

	WPoints	LPoints	ElapsedSeconds	Outcome
1	3	0	15	1
5	5	0	47	1
7	5	2	86	1
13	6	2	120	1
14	7	2	120	0
20	7	4	158	0
27	7	6	171	1
42	7	8	238	0
56	8	8	272	0
71	10	8	334	1
94	10	10	381	0
104	10	12	413	0
105	13	12	440	1
109	13	13	454	1
110	13	14	454	0
111	13	15	454	1
119	13	17	511	1
129	13	19	552	1
150	13	20	617	0
151	13	21	617	1
165	16	21	701	1
177	18	21	754	1
197	18	22	808	1
198	20	22	815	0
210	23	22	884	0
217	25	22	897	1
231	26	22	951	1
232	27	22	951	0
233	27	24	975	1
240	27	25	1001	0
...	...	...	...	...
99797	3	5	174	1
99801	3	8	210	1
99806	3	11	259	0

	WPoints	LPoints	ElapsedSeconds	Outcome
99807	5	11	288	1
99820	7	11	369	0
99823	7	13	392	1
99837	10	13	442	1
99851	12	13	525	1
99868	12	16	568	0
99877	12	19	655	1
99895	13	19	707	1
99896	14	19	707	0
99897	14	22	733	1
99902	17	22	757	0
99910	18	22	803	0
99918	19	22	825	0
99919	20	22	825	1
99921	20	24	841	1
99933	20	25	897	1
99934	20	26	897	0
99940	20	28	921	0
99942	22	28	953	0
99943	22	30	971	1
99945	25	30	980	1
99956	27	30	1040	0
99972	27	31	1095	0
99976	28	31	1118	0
99977	29	31	1118	1
99981	32	31	1151	1
99998	34	31	1295	0

16415 rows × 4 columns

In [119]:
```
data = condensed.iloc[:,0:2]
outcomes = condensed['Outcome'].astype('int')
X_train, X_test, y_train, y_test = train_test_split(data, outcomes, test_size
```

In [120]:
```
prob_log = LogisticRegression()
prob_log.fit(X_train, y_train)
prob_log.score(X_test, y_test)
```

Out[120]: 0.48913705583756345

In [98]:
```
pbp17 = pd.read_csv('Events_2017.csv')
```

In [123]:
```
reg17 = reg[reg['Season']==2017]
short17 = pbp17[['WTeamID','LTeamID','WPoints','LPoints','ElapsedSeconds']]
test_data = pd.DataFrame(columns=['WPoints','LPoints','ElapsedSeconds','Outco
testing = train_data(short17.iloc[0:466,:],reg17,test_data)
prob_log.predict_proba(testing.iloc[:,0:2])
```

Out[123]:
```
array([[0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48891683, 0.51108317],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
       [0.48778139, 0.51221861],
```

In [69]:
```
model_set = pd.DataFrame(columns=['WPoints','LPoints','ElapsedSeconds','TotWi
reg17 = reg[reg['Season']==2017]
data = pbp17.iloc[0:466,:]
impt = data[['WTeamID','LTeamID','WPoints','LPoints','ElapsedSeconds']]
def train_data(df):
    index = 0
    win=True
    for i,row in df.iterrows():
        w_team_stats = np.array(getTeamData(reg17,row['WTeamID']))
        l_team_stats = np.array(getTeamData(reg17,row['LTeamID']))
        team_diff = l_team_stats - w_team_stats
        if win:
            team_diff = w_team_stats - l_team_stats
            team_diff = list(team_diff)
            team_diff.append(1)
        else:
            team_diff = list(team_diff)
            team_diff.append(0)
        win = not(win)
        row_data = list(row[['WPoints','LPoints','ElapsedSeconds']])
        row_data.extend(team_diff)
        model_set.loc[index] = row_data
        index += 1
    return model_set
full = train_data(data)
condensed = full[(full['WPoints']!=0) | (full['LPoints']!=0)]
condensed
```

Out[69]:

	WPoints	LPoints	ElapsedSeconds	TotWins	PPG	PointsGiven	FG%	FG3%
4	3.0	0.0	26.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
26	4.0	0.0	177.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
27	5.0	0.0	177.0	-5.0	2.107527	9.13001	-0.00566	0.016821
34	5.0	3.0	223.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
38	5.0	6.0	262.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
40	8.0	6.0	288.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
71	8.0	9.0	429.0	-5.0	2.107527	9.13001	-0.00566	0.016821
73	11.0	9.0	442.0	-5.0	2.107527	9.13001	-0.00566	0.016821
106	11.0	11.0	615.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
118	11.0	13.0	633.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
119	11.0	14.0	633.0	-5.0	2.107527	9.13001	-0.00566	0.016821
130	11.0	17.0	683.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
150	12.0	17.0	736.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
160	13.0	17.0	767.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
161	14.0	17.0	767.0	-5.0	2.107527	9.13001	-0.00566	0.016821
163	14.0	20.0	784.0	-5.0	2.107527	9.13001	-0.00566	0.016821

	WPoints	LPoints	ElapsedSeconds	TotWins	PPG	PointsGiven	FG%	FG3%
168	16.0	20.0	814.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
176	16.0	21.0	847.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
177	16.0	22.0	847.0	-5.0	2.107527	9.13001	-0.00566	0.016821
180	16.0	24.0	881.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
181	18.0	24.0	901.0	-5.0	2.107527	9.13001	-0.00566	0.016821
205	18.0	25.0	971.0	-5.0	2.107527	9.13001	-0.00566	0.016821
218	18.0	26.0	1020.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
219	20.0	26.0	1035.0	-5.0	2.107527	9.13001	-0.00566	0.016821
229	20.0	27.0	1071.0	-5.0	2.107527	9.13001	-0.00566	0.016821
235	22.0	27.0	1114.0	-5.0	2.107527	9.13001	-0.00566	0.016821
237	22.0	30.0	1130.0	-5.0	2.107527	9.13001	-0.00566	0.016821
238	25.0	30.0	1142.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
239	25.0	33.0	1159.0	-5.0	2.107527	9.13001	-0.00566	0.016821
248	26.0	33.0	1180.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
...	...	...	...	...	...	...	...	...
268	33.0	38.0	1274.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
274	34.0	38.0	1295.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
280	36.0	38.0	1323.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
281	37.0	38.0	1323.0	-5.0	2.107527	9.13001	-0.00566	0.016821
295	39.0	38.0	1406.0	-5.0	2.107527	9.13001	-0.00566	0.016821
305	42.0	38.0	1444.0	-5.0	2.107527	9.13001	-0.00566	0.016821
311	44.0	38.0	1509.0	-5.0	2.107527	9.13001	-0.00566	0.016821
318	45.0	38.0	1543.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
319	46.0	38.0	1543.0	-5.0	2.107527	9.13001	-0.00566	0.016821
327	47.0	38.0	1566.0	-5.0	2.107527	9.13001	-0.00566	0.016821
333	47.0	39.0	1590.0	-5.0	2.107527	9.13001	-0.00566	0.016821
334	47.0	40.0	1590.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
336	50.0	40.0	1611.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
370	50.0	42.0	1772.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
372	51.0	42.0	1786.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
373	52.0	42.0	1786.0	-5.0	2.107527	9.13001	-0.00566	0.016821
378	52.0	44.0	1861.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
384	52.0	46.0	1905.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
397	52.0	48.0	1956.0	-5.0	2.107527	9.13001	-0.00566	0.016821
410	55.0	48.0	2047.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
414	57.0	48.0	2091.0	5.0	-2.107527	-9.13001	0.00566	-0.016821

	WPoints	LPoints	ElapsedSeconds	TotWins	PPG	PointsGiven	FG%	FG3%
418	60.0	48.0	2117.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
420	60.0	51.0	2133.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
422	63.0	51.0	2164.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
428	65.0	51.0	2205.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
434	66.0	51.0	2230.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
441	67.0	51.0	2275.0	-5.0	2.107527	9.13001	-0.00566	0.016821
442	68.0	51.0	2275.0	5.0	-2.107527	-9.13001	0.00566	-0.016821
455	70.0	51.0	2345.0	-5.0	2.107527	9.13001	-0.00566	0.016821
456	70.0	53.0	2354.0	5.0	-2.107527	-9.13001	0.00566	-0.016821

66 rows × 12 columns

In [70]:
```
condensed.loc[condensed.index[231]]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-70-69c1cfd805d3> in <module>()
----> 1 condensed.loc[condensed.index[231]]

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in __getitem__(se
lf, key)
   2082
   2083         if is_scalar(key):
-> 2084             return getitem(key)
   2085
   2086         if isinstance(key, slice):

IndexError: index 231 is out of bounds for axis 0 with size 66
```

In [71]:
```
probs = prob_log.predict_proba(condensed.iloc[:,0:10])
for i in range(0,len(probs)):
    print(str(i) + ':' + str((probs[i])))
```

```
0:[0.12928681 0.87071319]
1:[0.12572592 0.87427408]
2:[0.86988385 0.13011615]
3:[0.12575631 0.87424369]
4:[0.12533332 0.87466668]
5:[0.12655893 0.87344107]
6:[0.86670697 0.13329303]
7:[0.86836134 0.13163866]
8:[0.12065419 0.87934581]
9:[0.12059469 0.87940531]
10:[0.86395127 0.13604873]
11:[0.12010397 0.87989603]
12:[0.11932255 0.88067745]
13:[0.11912368 0.88087632]
14:[0.86276728 0.13723272]
15:[0.8629647 0.1370353]
16:[0.12036726 0.87963274]
17:[0.11970142 0.88029858]
18:[0.862955 0.137045]
19:[0.11942792 0.88057208]
20:[0.86321222 0.13678778]
21:[0.861361 0.138639]
22:[0.11742805 0.88257195]
23:[0.86108476 0.13891524]
24:[0.8602385 0.1397615]
25:[0.86035638 0.13964362]
26:[0.86058682 0.13941318]
27:[0.11935882 0.88064118]
28:[0.86253282 0.13746718]
29:[0.11959914 0.88040086]
30:[0.86330196 0.13669804]
31:[0.863536 0.136464]
32:[0.1210553 0.8789447]
33:[0.1210726 0.8789274]
34:[0.86583998 0.13416002]
35:[0.86560315 0.13439685]
36:[0.12252561 0.87747439]
37:[0.12259184 0.87740816]
38:[0.12310318 0.87689682]
39:[0.86713434 0.13286566]
40:[0.86608726 0.13391274]
41:[0.86702644 0.13297356]
42:[0.86650208 0.13349792]
43:[0.12286188 0.87713812]
44:[0.86687637 0.13312363]
45:[0.86688951 0.13311049]
46:[0.86642206 0.13357794]
47:[0.12328641 0.87671359]
48:[0.12463169 0.87536831]
49:[0.12070918 0.87929082]
50:[0.12096124 0.87903876]
51:[0.86481401 0.13518599]
```

```
52:[0.12000965 0.87999035]
53:[0.11926332 0.88073668]
54:[0.86117311 0.13882689]
55:[0.11779825 0.88220175]
56:[0.11787411 0.88212589]
57:[0.11903671 0.88096329]
58:[0.11923814 0.88076186]
59:[0.12027951 0.87972049]
60:[0.12043648 0.87956352]
61:[0.12039539 0.87960461]
62:[0.86285843 0.13714157]
63:[0.12044611 0.87955389]
64:[0.86286917 0.13713083]
65:[0.12001269 0.87998731]
```

In [ ]: