A MACHINE LEARNING MODEL FOR TENNIS GAME PREDICTIONS

Group 10

Team Members: Godfred Somua-Gyimah, Bharath Chowdary Ballamudi

Introduction

According to a recent study (Pempus, 2015), sports betting is currently a $1.3 Trillion industry globally. In 2014, almost $4 billion was spent on sports betting in Nevada alone and these figures continue to rise. Generally, most seasoned gamblers rely on their deep domain knowledge to place bets. On the other hand, most low-risk gamblers and bettors with limited domain knowledge generally depend on the betting odds of one or a few bookmakers. Nonetheless, betting losses continue to increase in both cases. It is estimated that American gamblers recorded the highest gambling loss ($119 billion) worldwide in 2013 (Aziz, 2014).

Tennis is undoubtedly one of the most popular sports in the world. The increase in popularity of the sport, coupled with the expansion of the online sports betting market, has led to a significant rise in the volume of tennis bets recently. The potential profit, as well as academic interest, has fuelled the search for accurate tennis match prediction algorithms (Sipko, 2015).

Current prediction algorithms vary in the type of features used in building the model as well as in their prediction methods. Barnett and Clarke (2005) and O'Malley (2008) have developed stochastic tennis prediction models using Markov chains. Both models considered the probability of a player winning individual points or complete sets, based on previous serve statistics of both players. The eventual outcome of a match is then determined by the probability of each player, winning a point during their serve(Sipko, 2015). Knottenbelt (2012) built on this original approach and suggested a model, which only uses both players' performances against common opponents, rather than their average statistics across all opponents. Madurska (2012) later modified Knottenbelt's model to accommodate the changes in player performance from one set to the next.

While several models exist for varied tennis prediction purposes, very few use Machine Learning approaches. Of all the ML models, Logistic Regression and Artificial Neural Network (ANN), are the most widely found in published literature on tennis match predictions. The Association of Tennis Professionals (ATP) have a rating system which ranks all professional tennis players, based on their accummulated points from games. Based on the outcome of a new game and the current ranking / points rating of the two players, the point addition and deduction are determined for the winner and loser respectively. Clarke and Dyte (2009) developed a logistic regression model using this feature. Their model based game outcome predictions on the difference in rating points between the two players. However, their model was limited, since the use of the single feature, rank_difference, implied inherent bias in favor of the higher-ranked player. Ma et al.(2013) also built a logistic regression model based on 16 features comprising player characteristics, performance and match statistics. However, all the models mentioned above, did not provide any specific figures on prediction accuracy.
Somboonphokkaphan (2009) presented a 3-layer feed-forward Artificial Neural Network using 27 features which comprised of player and match statistics. Whilst the author reported a 75% prediction accuracy, the model was built using data from only two tournaments (2007 and 2008 Grand Slam). It is therefore difficult to predict the model's performance for the general case or for other major tournaments.

The objective of this study is to build a tennis model, that can be generalized over all major tournaments with acceptable prediction accuracy.

Data Collection

In this project, we will attempt to build a model for predicting the outcome of sports competitions but the specific case of Tennis betting will be used as an illustration. A dataset containing the tournament statistics of all major Tennis competitions from 2010 to 2016 was compiled using information obtained from http://www.tennis-data.co.uk/alldata.php. The original data set for each year had forty (40) features which included the following:

ATP = Tournament number
Location = Venue of tournament
Tournament = Name of tounament (including sponsor if relevant)
Data = Date of match
Series = Name of ATP tennis series (Grand Slam, Masters, International or International Gold)
Court = Type of court (outdoors or indoors)
Surface = Type of surface (clay, hard, carpet or grass)
Round = Round of match
Best of = Maximum number of sets playable in match
Winner = Match winner
Loser = Match loser
WRank = ATP Entry ranking of the match winner as of the start of the tournament
LRank = ATP Entry ranking of the match loser as of the start of the tournament
WPts = ATP Entry points of the match winner as of the start of the tournament
LPts = ATP Entry points of the match loser as of the start of the tournament
W1 = Number of games won in 1st set by match winner
L1 = Number of games won in 1st set by match loser
W2 = Number of games won in 2nd set by match winner
L2 = Number of games won in 2nd set by match loser
W3 = Number of games won in 3rd set by match winner
L3 = Number of games won in 3rd set by match loser
W4 = Number of games won in 4th set by match winner
L4 = Number of games won in 4th set by match loser
W5 = Number of games won in 5th set by match winner
L5 = Number of games won in 5th set by match loser
Wsets = Number of sets won by match winner
Lsets = Number of sets won by match loser
Comment = Comment on the match (Completed, won through retirement of loser, or via Walkover)

```
B365W = Bet365 odds of match winner
B365L = Bet365 odds of match loser
B&WW = Bet&Win odds of match winner
B&WL = Bet&Win odds of match loser
EXW = Expekt odds of match winner
EXL = Expekt odds of match loser
LBW = Ladbrokes odds of match winner
LBL = Ladbrokes odds of match loser
PSW = Pinnacles Sports odds of match winner
PSL = Pinnacles Sports odds of match loser
MaxW = The highest odd of match winner
MaxL = The highest odd of match loser
AvgW = The average odd of match winner
AvgL = The average odd of match loser
```

In [1]:
```
import pandas as pd
Tennisdata = pd.read_csv("F:/Data_Files/TennisData/Best_data/Men_Tennis.csv")
```

In [2]:
```
Tennisdata.head()
```

Out[2]:

	ATP	Location	Tournament	Date	Series	Court	Surface	Round	Best of	Winner	...	EXW	EXL	LBW	LBL	PS
0	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Dimitrov G.	...	1.68	2.10	1.62	2.25	1.6
1	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Kudla D.	...	1.58	2.35	1.53	2.50	1.6
2	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Kamke T.	...	1.82	1.90	1.80	2.00	1.9
3	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Chung H.	...	1.82	1.90	1.73	2.10	1.9
4	1	Brisbane	Brisbane International	1/5/2016	ATP250	1	1	1	3	Goffin D.	...	1.30	3.40	1.29	3.75	1.3

5 rows × 40 columns

Data Cleaning & Feature Engineering

By studing the dataset, we realized that some of our features had to be changed to numeric values so that they can be used in the model. The following changes were made in excel to represent certain feature values with numbers:

Court : Indoor = 0 and Outdoor = 1

Surface: Hard = 1, Clay = 2, Grass = 3, Carpet = 4

Round: 0th Round = 0, 1st Round = 1, 2nd Round = 2, 3rd Round = 3, 4th Round=4, Quarterfinals=5, Round Robin=6, Semifinals=7, The Final=8

Also, we realized that match results could be determined by retirement or walkovers due to injury. Since these two are difficult to factor into the model, we decided to use only data where the match was actually completed.

In [3]:
```
Tennisdata = Tennisdata[Tennisdata['Comment'] == 'Completed']
```

The dataset contained games with both 3 sets and 5 sets. Since over 90% of the games were 3 set games, we decided to leave out the 5 set games, in order to improve data uniformity.

In [4]:
```
Tennisdata = Tennisdata[Tennisdata['Best of'] == 3]
```

In [5]:
```
len(Tennisdata)
```

Out[5]: 14276

In [6]:
```
Tennisdata.head()
```

Out[6]:

	ATP	Location	Tournament	Date	Series	Court	Surface	Round	Best of	Winner	...	EXW	EXL	LBW	LBL	PS
0	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Dimitrov G.	...	1.68	2.10	1.62	2.25	1.6
1	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Kudla D.	...	1.58	2.35	1.53	2.50	1.6
2	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Kamke T.	...	1.82	1.90	1.80	2.00	1.9
3	1	Brisbane	Brisbane International	1/4/2016	ATP250	1	1	1	3	Chung H.	...	1.82	1.90	1.73	2.10	1.9
4	1	Brisbane	Brisbane International	1/5/2016	ATP250	1	1	1	3	Goffin D.	...	1.30	3.40	1.29	3.75	1.3

5 rows × 40 columns

Since our model will use both the pre-match statistics and betting odds, we will remove instances with some missing data.

In [7]:
```
Tennisdata = Tennisdata[Tennisdata['Court'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['Surface'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['Round'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['WRank'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['LRank'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['WPts'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['LPts'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['B365W'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['B365L'].notnull()== True]
```

```
Tennisdata = Tennisdata[Tennisdata['EXW'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['EXL'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['LBW'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['LBL'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['PSW'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['PSL'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['MaxW'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['MaxL'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['AvgW'].notnull()== True]
Tennisdata = Tennisdata[Tennisdata['AvgL'].notnull()== True]
```

In [8]:

```
import numpy as np

Tennisdata = Tennisdata[np.isfinite(Tennisdata['Court'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['Surface'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['Round'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['WRank'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['LRank'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['WPts'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['LPts'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['B365W'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['B365L'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['EXW'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['EXL'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['LBW'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['LBL'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['PSW'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['PSL'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['MaxW'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['MaxL'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['AvgW'])]
Tennisdata = Tennisdata[np.isfinite(Tennisdata['AvgL'])]
```

In [9]:

```
Tennisdata.dropna(how='any', subset=['Court', 'Surface', 'Round', 'WRank', 'LRank', 'WPts','LPts

Tennis = Tennisdata[['Court', 'Surface', 'Round', 'WRank', 'LRank', 'WPts','LPts','B365W', 'B365
```

We will check the data type for all columns to make sure that our data is clean.

In [10]:
```
Tennis.dtypes
```

Out[10]:
```
Court         int64
Surface       int64
Round         int64
WRank       float64
LRank       float64
WPts        float64
LPts        float64
B365W       float64
B365L       float64
EXW         float64
EXL         float64
LBW         float64
LBL         float64
PSW         float64
PSL         float64
MaxW        float64
MaxL        float64
AvgW        float64
AvgL        float64
dtype: object
```

Since the goal is to build a model for pre-match predictions, we ignored the in-game statistics and made use of only the pre-game statistics (e.g. Court, Surface, WRank, LRank, WPts, LPts) as well as the betting odds from bookmakers.

Also, the dataset presents statistics for winners and losers. So, it is impossible to build a model for predicting game outcomes, using the data as is. So, we transformed the data by creating the following new features: PlayerA, PlayerB, PlayerAPts and PlayerBPts. For every game, the player with the higher ranking is set as PlayerA, with the rank number used instead of player name. Similarly, the lower-ranked player is set as PlayerB. In order to make the model generic, we used player ranks, instead of specific player names for each instance of the data. In this case, each of the two players (PlayerA, PlayerB) in a game, is represented by their ATP Ranking at the start of the tournament, instead of their individual names.

We also suspected that the difference between player ranks and player points for the two players (as at the start of the tournament) could be important for our model. In football, baseball, snooker and other sports games, a similar principle known as the Elos rating, has been used. Therefore, we created two extra features, Rank_Diff:
Rank_Diff = 'PlayerA Rank' - 'PlayerB Rank'
Pt_Diff = 'PlayerA Pts' - 'PlayerB Pts'

We also created a binary feature for the outcome of every match such that: win=1 and lose=0. This will be the (Y) for our binary classifier model.

In [11]:
```
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PlayerA'] = Tennis['WRank']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PlayerA_Pts'] = Tennis['WPts']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_B365'] = Tennis['B365W']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_EX'] = Tennis['EXW']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_LB'] = Tennis['LBW']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_PS'] = Tennis['PSW']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_MaxBet'] = Tennis['MaxW']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PA_AvgBet'] = Tennis['AvgW']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PlayerB'] = Tennis['LRank']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PlayerB_Pts'] = Tennis['LPts']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_B365'] = Tennis['B365L']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_EX'] = Tennis['EXL']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_LB'] = Tennis['LBL']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_PS'] = Tennis['PSL']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_MaxBet'] = Tennis['MaxL']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'PB_AvgBet'] = Tennis['AvgL']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'Rank_Diff'] = Tennis['PlayerA'] - Tennis['PlayerB']
Tennis.ix[Tennis.WRank > Tennis.LRank, 'Pt_Diff'] = Tennis['PlayerA_Pts'] - Tennis['PlayerB_Pts'
Tennis.ix[Tennis.WRank > Tennis.LRank, 'Outcome'] = 1

Tennis.ix[Tennis.WRank < Tennis.LRank, 'PlayerA'] = Tennis['LRank']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PlayerA_Pts'] = Tennis['LPts']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_B365'] = Tennis['B365L']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_EX'] = Tennis['EXL']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_LB'] = Tennis['LBL']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_PS'] = Tennis['PSL']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_MaxBet'] = Tennis['MaxL']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PA_AvgBet'] = Tennis['AvgL']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PlayerB'] = Tennis['WRank']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PlayerB_Pts'] = Tennis['WPts']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_B365'] = Tennis['B365W']
```

```
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_EX'] = Tennis['EXW']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_LB'] = Tennis['LBW']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_PS'] = Tennis['PSW']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_MaxBet'] = Tennis['MaxW']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'PB_AvgBet'] = Tennis['AvgW']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'Rank_Diff'] = Tennis['PlayerA'] - Tennis['PlayerB']
Tennis.ix[Tennis.WRank < Tennis.LRank, 'Pt_Diff'] = Tennis['PlayerA_Pts'] - Tennis['PlayerB_Pts'
Tennis.ix[Tennis.WRank < Tennis.LRank, 'Outcome'] = 0
```

```
C:\Anaconda2\lib\site-packages\pandas\core\indexing.py:288: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.ht
ml#indexing-view-versus-copy
  self.obj[key] = _infer_fill_value(value)
C:\Anaconda2\lib\site-packages\pandas\core\indexing.py:465: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.ht
ml#indexing-view-versus-copy
  self.obj[item] = s
```

In [12]:
```
Tennis.head()
```

Out[12]:

	Court	Surface	Round	WRank	LRank	WPts	LPts	B365W	B365L	EXW	...	PlayerB_Pts	PB_B365	PB_EX	PE
0	1	1	1	28.0	15.0	1360.0	2145.0	1.66	2.10	1.68	...	2145.0	2.10	2.10	2.1
1	1	1	1	69.0	129.0	719.0	459.0	1.53	2.37	1.58	...	719.0	1.53	1.58	1.5
2	1	1	1	277.0	231.0	185.0	225.0	1.72	2.00	1.82	...	225.0	2.00	1.90	2.(
3	1	1	1	51.0	60.0	817.0	782.0	1.83	1.83	1.82	...	817.0	1.83	1.82	1.7
4	1	1	1	16.0	37.0	1880.0	1105.0	1.28	3.50	1.30	...	1880.0	1.28	1.30	1.2

5 rows × 38 columns

There are too many features. We will reduce the data to only features that will be used in building the model.

In [13]:
```
Tennis = Tennis[['Court', 'Surface', 'Round','PlayerA','PlayerA_Pts','PA_B365','PA_EX','PA_LB','
                 'PA_MaxBet','PA_AvgBet','PlayerB','PlayerB_Pts','PB_B365','PB_EX','PB_LB','PB_P
                 'PB_MaxBet','PB_AvgBet','Rank_Diff','Pt_Diff','Outcome']]
```

In [14]:
```
Tennis.head()
```

Out[14]:

	Court	Surface	Round	PlayerA	PlayerA_Pts	PA_B365	PA_EX	PA_LB	PA_PS	PA_MaxBet	...	PlayerB_Pts	PB_
0	1	1	1	28.0	1360.0	1.66	1.68	1.62	1.68	1.68	...	2145.0	2.10
1	1	1	1	129.0	459.0	2.37	2.35	2.50	2.40	2.50	...	719.0	1.53
2	1	1	1	277.0	185.0	1.72	1.82	1.80	1.90	1.90	...	225.0	2.00
3	1	1	1	60.0	782.0	1.83	1.90	2.10	1.96	2.10	...	817.0	1.83
4	1	1	1	37.0	1105.0	3.50	3.40	3.75	3.74	3.75	...	1880.0	1.28

5 rows × 22 columns

We want to change the 'Outcome' column to integers

We will find and delete all rows with NA or NaN data

In [15]:
```
Tennis = Tennis.reset_index(drop=True)
Tennis[Tennis.isnull().any(axis=1)]
```

Out[15]:

	Court	Surface	Round	PlayerA	PlayerA_Pts	PA_B365	PA_EX	PA_LB	PA_PS	PA_MaxBet	...	PlayerB_Pts
10705	1	2	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

1 rows × 22 columns

In [16]:
```
Tennis = Tennis.drop(Tennis.index[[10705]])
```

In [17]:
```
Tennis[Tennis.isnull().any(axis=1)]
```

Out[17]:

	Court	Surface	Round	PlayerA	PlayerA_Pts	PA_B365	PA_EX	PA_LB	PA_PS	PA_MaxBet	...	PlayerB_Pts	PB_E

0 rows × 22 columns

In [18]:
```
Tennis.Outcome.astype(int);
```

```
The original data is sorted by tournament and this can lead to skewed model results. Therefore,
we will sort the data by 'randomising' the index to reduce bias when splitting the data into
train and test sets.
```

In [19]:
```
Tennis = Tennis.sort_values(['PA_B365'], ascending=True)
Tennis = Tennis.reset_index(drop=True)
```

Model Selection

In [20]:
```
import numpy as np
import scipy as sp
import sklearn as sk
```

In [21]:
```
len(Tennis)
```

Out[21]: 14004

```
Data Splitting: 80% of the data will be used for training and cross-validation.
The remaining 20% will be used for testing.
We will also separate the data into X and Y for both train and test sets.
```

In [22]:
```
trainset = Tennis[:(int(0.8*len(Tennis)))]
testset = Tennis.tail(int(0.2*len(Tennis)))
```

In [23]:
```
X_trainset = trainset[['Court', 'Surface', 'Round','PlayerA','PlayerA_Pts','PA_B365','PA_EX','PA
                  'PA_MaxBet','PA_AvgBet','PlayerB','PlayerB_Pts','PB_B365','PB_EX','PB_LB','PB_P
                  'PB_MaxBet','PB_AvgBet','Rank_Diff','Pt_Diff']]
X_testset = testset[['Court', 'Surface', 'Round','PlayerA','PlayerA_Pts','PA_B365','PA_EX','PA_L
                  'PA_MaxBet','PA_AvgBet','PlayerB','PlayerB_Pts','PB_B365','PB_EX','PB_LB','PB_P
                  'PB_MaxBet','PB_AvgBet','Rank_Diff','Pt_Diff']]
Y_trainset = trainset['Outcome']
Y_testset = testset['Outcome']
```

Trying Different Models

As part of the model selection process, we will carry out a 10-fold cross validation on the train data using different ML techniques. For the model selection stage, we will use the ML package, Scikit Learn and evaluate the models based on their cross-validation accuracy.
We will then select one of the models with good prediction accuracy and build it from scratch, using knowledge gained in the Comp Sci 5402 Machine Learning class.

In [25]:
```
# Try Linear Regression
from sklearn import linear_model
from sklearn.cross_validation import cross_val_score
model = linear_model.LinearRegression()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: -0.01 (+/- 0.05)
```

In [26]:
```
# Try Logistic Regression
from sklearn import linear_model
from sklearn.cross_validation import cross_val_score
model = linear_model.LogisticRegression()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.64 (+/- 0.31)
```

In [27]:
```
# Try Linear Support Vector Classifier(SVC)
from sklearn import svm
from sklearn.cross_validation import cross_val_score
model = svm.LinearSVC()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.54 (+/- 0.15)
```

In [28]:
```
# Try NuSVC
from sklearn import svm
model = svm.NuSVC()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.60 (+/- 0.01)
```

In [29]:
```
# Try RBF Kernel SVM
from sklearn import svm
model = svm.SVC(kernel = 'rbf')
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.60 (+/- 0.00)
```

In [30]:
```
# Try Gaussian NB
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.64 (+/- 0.34)
```

In [31]:
```
# Try Bernoulli NB
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.60 (+/- 0.00)
```

In [31]:
```
# Try Bernoulli NB
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.60 (+/- 0.00)
```

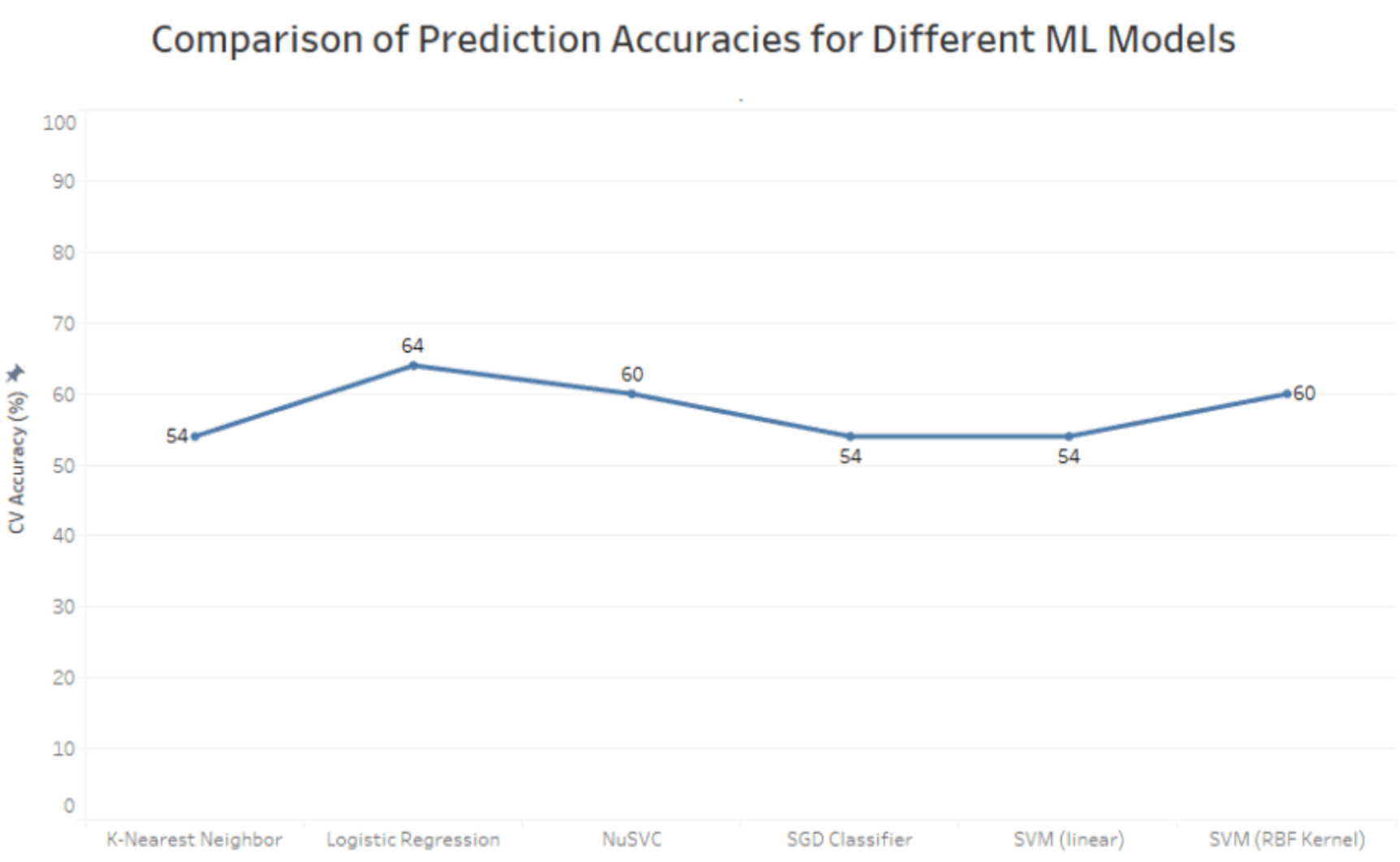
In [32]:
```
# Try SGDClassifier
import numpy as np
from sklearn.linear_model import SGDClassifier
model = SGDClassifier(loss="hinge", penalty="l2")
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.54 (+/- 0.18)
```

In [33]:
```
# Try K Nearest Neighbors
import numpy as np
from sklearn import neighbors
model = neighbors.KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(model, X_trainset, Y_trainset, cv=10)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.54 (+/- 0.03)
```

Results


Comparison of Prediction Accuracies for Different ML Models
CV Accuracy (%)
100
90
80
70
60
50
40
30
20
10
0
54
64
60
54
54
60
K-Nearest Neighbor
Logistic Regression
NuSVC
SGD Classifier
SVM (linear)
SVM (RBF Kernel)

Building our own Logistic Regression Model in Matlab

We will export the train and test data sets for use in Matlab

In [37]:
```
trainset.to_csv("G:/Dropboxx/PhD Mining Engineering/Coursework/07FS2016_local/COMP SCI 5402 - Ma
testset.to_csv("G:/Dropboxx/PhD Mining Engineering/Coursework/07FS2016_local/COMP SCI 5402 - Mac
```

Matlab Code

```
% Logistic Regression

clc;
clear all;

% load the data sets
X = csvread('X_trainset - Copy.csv',1);
Y = csvread('Y_trainset - Copy.csv');

%sample out a subset from the input data
[X, idx] = datasample(X,round(0.02*size(X,1)),'replace',false);
Y = Y(idx,:);
Y = 2*(Y-0.5); % change values from (0,1) to (-1,1)
X = X(:,[4,5,12,13,20,21]);% select a few features out of the total
X = [ones(size(X,1),1),X];
[N,D] = size(X);
w = zeros(D,1); %initate w to zeros
sum =0;
alpha =3;
```

```
for t=1:1000

    for i=1:N
        sum = sum + (Y(i)*X(i,:))/(1 + exp(Y(i)*w.'*X(i,:).'));
        Delta_E_in = -sum/N;
    end
    sum =0;
    if(Delta_E_in == 0)
        break;
    else
        w = w - (alpha*Delta_E_in).';
    end
end

errors=0;
y_cap = X*w; %y_cap is an estimate of Y
Y_prediction = sign(y_cap);

for i=1:length(Y_prediction)
    if sign(Y(i)) ~= sign(Y_prediction(i))
        errors = errors+1;
    end
end

error_rate = errors/N
```

The full matlab code is in the file "logReg.m"

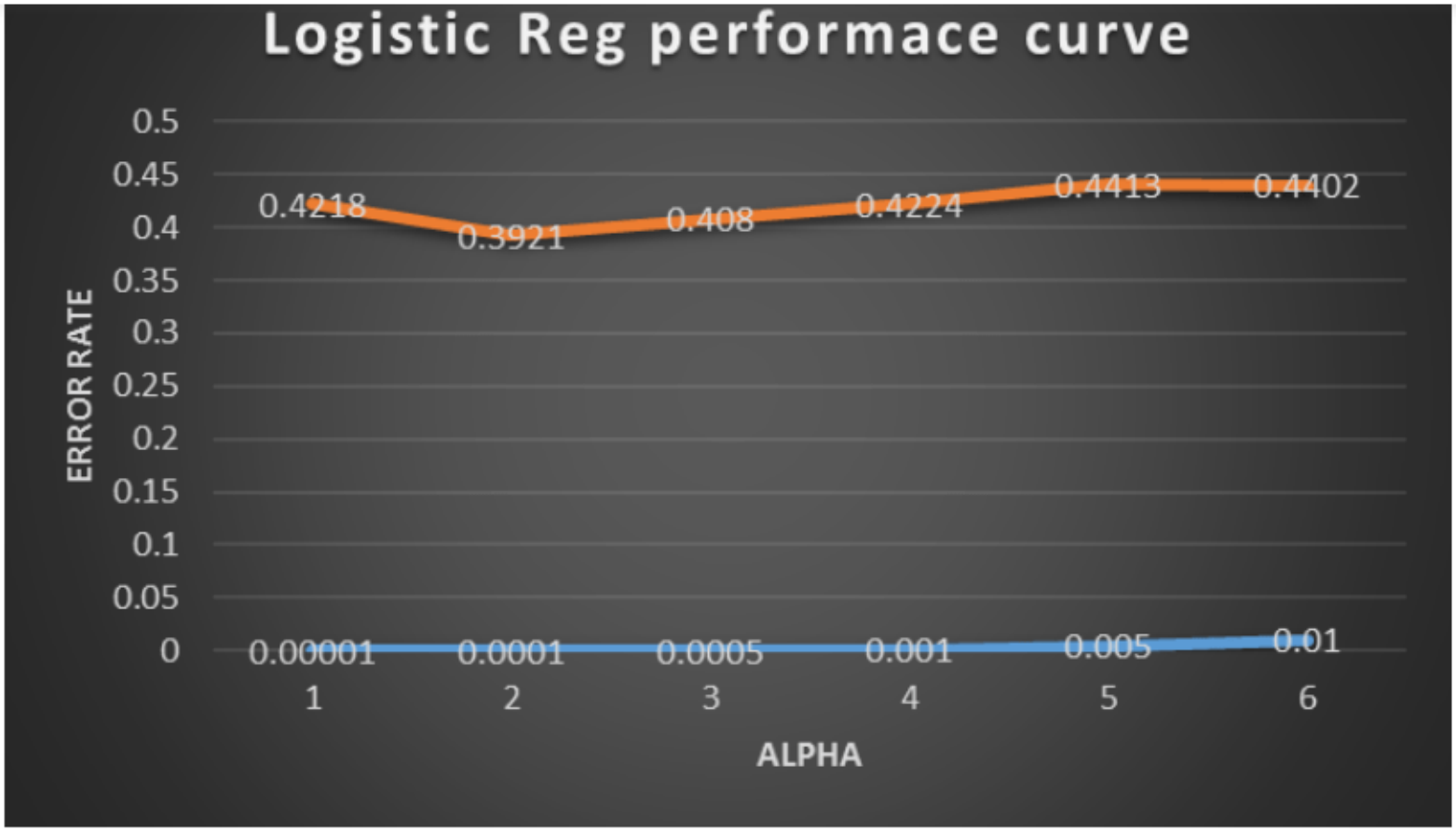
The Logistic Regression is employed to binary classify the explanatory variable representing the winning result. Further, 10 fold cross validation method is used to validate the generated model on the test data. The winning prediction varies every time the model is trained and evaluated as the training and test data are selected on random basis each time for 10 fold. Roughly, the accuracy of the model is observed to hover around the range 60%-65%. The step size (alpha) defines the performance of a model. Below are the results obtained with alpha=0.001. The accuracy is less than 58% in this case.

```
>> error_train     >> error_test

error_train =      error_test =          average_error =

    0.3979             0.3875                0.4224
    0.3989             0.3741
    0.5060             0.4920
    0.3959             0.4054
    0.4042             0.4205
    0.3993             0.4009
    0.3979             0.3866
    0.5402             0.5509
    0.4055             0.4143
    0.3974             0.3920
```

As can be deduced from the above set of results, the 10-fold cross validation performance is impacted primarily by the two components (3rd and 8th) of the 10-fold system, which tilted the scales from the average error otherwise tending to be less than 40 to -what it ended up to be- 42. Arguably, the training data must have been badly scattered in these two cases that the learning must have been bad resulting in bad fit and hence the bad validation results.
Moving on, however, running the model over a range of alpha values can help us decide on the alpha value that consistently delivers best results. The below attached graph (Fig 1) is one such attempt where the error rate is plotted against its corresponding alpha value. The best performance is observed at alpha value of 0.001. The model takes about 2 hours for a single run. And so due to this computational expense as well as the limited high computing resources at our disposal, we were unable to try more alpha values for better parameter tuning. However, in future we intend to experiment more alpha values to arrive at the best accuracy possible.



CONCLUSION

In this project, we explored the ability of Machine Learning techniques to predict the outcome of tennis games. Unlike previous tennis models, our model incorporates all the following features along with others: the betting odds from four major bookmakers, player point difference and player rank difference. In all, our model had 22 features. The Linear Regression, Support Vector Machine(linear kernel), Support Vector Machine(polynomial kernel), RBF Kernel SVM, K-Nearest Neighbor and the Stochastic Gradient Descent (SGD) classifiers were tested using the train data. The best model was obtained with Logistic Regression, which gave 64% cross-validation accuracy and 60% accuracy on the test data. By comparison with previous tennis models, it can be concluded that our model performs fairly well and gives predictions which are in the ballpark of current good tennis models. For the RBF Kernel SVM, better results may have been obtained by tuning gamma and C. This will be explored in future works. Also, future works will focuss on engineering more features and training the data on other ML classifier models such as DecisionTrees, Gradient Boosting methods, Neural Networks and Random Forest in an attempt to obtain better predictive performance.

REFERENCES

[1] Pempus, B. (2015). Global Sports Betting At Minimum A $1.3 Trillion Industry, Expert Tells United Nations. Retrieved from http://www.cardplayer.com/poker-news/18683-global-sports-betting-at-minimum-a-1-3-trillion-industry-expert-tells-united-nations

[2] Aziz, J. (2014). How did Americans manage to lose $119 billion gambling last year? Retrieved from http://theweek.com/articles/451623/how-did-americans-manage-lose-119-billion-gambling-last-year

[3] Sipko, Michal, and William Knottenbelt. (2015). "Machine Learning for the Prediction of Professional Tennis Matches.", Final Project, Imperial College London, London, UK.

[4] Mathis, Simon. (2012). "Prediction of Tennis Results."

[5] T. Barnett and S. R. Clarke. (2005). Combining player statistics to predict outcomes of tennis matches.IMA Journal of Management Mathematics, 16:113–120.

[6] J. A. O'Malley. (2008). Probability Formulas and Statistical Analysis in Tennis. Journal of Quantitative Analysis in Sports.

[7] W. J. Knottenbelt, D. Spanias, and A. M. Madurska. (2012). A common-opponent stochastic model for predicting the outcome of professional tennis matches. Computers and Mathematics with Applications,64:3820–3827.

[8] A. M. Madurska. (2012). A Set-By-Set Analysis Method for Predicting the Outcome of Professional Singles Tennis Matches. Technical report, Imperial College London, London.

[9] A. Somboonphokkaphan, S. Phimoltares, and C. Lursinsap. (2009) Tennis Winner Prediction based on Time-Series History with Neural Modeling. IMECS 2009: International Multi-Conference of Engineers and Computer Scientists, Vols I and II, I:127–132.