College Football Bowl Game Predictions

Spring 2015
15.077 Final Project
Geoffrey Gunow

1 Introduction

College Football has grown to be one of the most popular competitive sports over the last few decades. Indeed, sports betting on college football is a multi-billion dollar industry. In addition, many organizations such as ESPN offer cash prize competitions for predicting outcomes of games. In particular, ESPN is renowned for it's College Football Bowl Pick 'em Challenge. In this game, each player attempts to predict the winner of all N college football bowl games. In addition, the player assigns confidence points ranging from 1 to N on each game with no repeated numbers. If the player correctly predicts the outcome, they receive the confidence points assigned to the game. If they incorrectly predict the outcome, they receive no points. The player with the highest point total at the end of the games wins. Therefore to maximize the point total, the player should assign the highest confidence points to games in which the player is very certain of the outcome, and the lowest points to uncertain outcomes.

In this project, I concentrate on being able to accurately predict the outcomes of bowl games and also form a measure of confidence for each prediction. Therefore, multiple models are developed for predicting the outcomes of bowl games and methods for deriving confidence measures are discussed. Throughout this project, I make heavy use of the python package Scikit-learn [1]. All methods and analysis used in this report are based on Scikit-learn or self-implemented software.

2 Data Scraping and Feature Formation

First data needs to be gathered on each team participating in a bowl game in order to form features. This data is formed from regular season statistics. *Sunshine Forecast Downloadable Data Files* [2] provides data for each Division I FBS football game played from 2002 through 2013 in CSV format, a total of 383 bowl games. This data is scraped to produce easily-accessible dictionaries for the formation of features. Attributes provided for each team in this data are listed in Fig. 1.

In addition to these 21 attributes, the number of takeaways, giveaways, and turnover margin are calculated from these statistics leading to a total of 24 attributes. For each attribute, a feature for the game averaged value is created when applicable (games played, wins, and losses are not normalized by the number of games). An additional 24 features are created for games just against teams playing in bowl games. The idea behind creating these

- Offensive Pass Completions
- Offensive Pass Attempts
- Offensive Pass Interceptions
- Offensive Pass Yards
- Offensive Rush Attempts
- Offensive Rush Yards
- Offensive Fumbles

- Defensive Pass Completions
- Defensive Pass Attempts
- Defensive Pass Interceptions
- Defensive Pass Yards
- Defensive Rush Attempts
- Defensive Rush Yards
- Defensive Fumbles

- Points For
- Points Against
- Games Played
- Wins
- Losses
- Winning Percentage

Figure 1: Attributes for each team provided by *Sunshine Forecast Downloadable Data Files.*

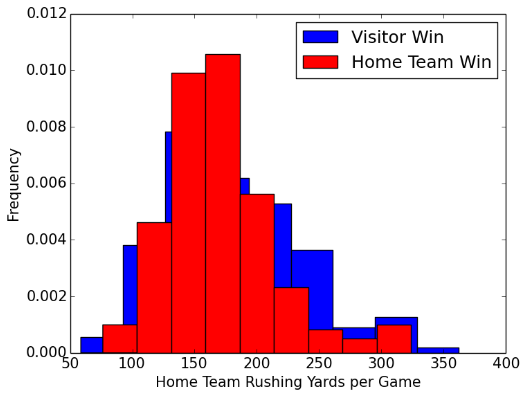
features is to factor in how teams perform against strong teams. With these features added, information about the strength of schedule is implicitly added. These 48 features are formed for both teams. Finally, the Vegas betting line is added, leading to a total of 97 features. Each game is categorized in terms of whether the home team won the game. With such a large number of features relative to the size of the dataset, regularization is very important. Data is partitioned into train, validate, and test sets based on a 50%, 30%, and 20% split respectively.

3 Feature Analysis

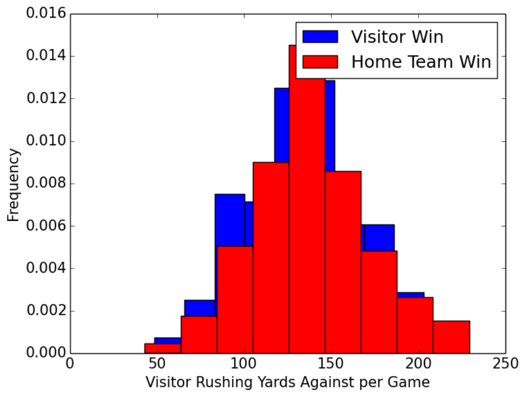
Now that the feature set is formed, it is often instructive to look at the data to get a sense of the underlying trends. First, we look at the normality of the features. Some algorithms rely on the assumption that the data is normally distributed. For instance, the Naive Bayes approach assumes that within each class the underlying features are normally distributed. To test this assumption, histograms are first created for every feature. For the purpose of illustration, the distribution of two features: the home team offensive rush yards per game and the visiting team defensive rush yards per game are plotted in Fig. 2.

We see for home team wins, the average visitor rushing yards against per game seems to follow a normal distribution well, but the home team rushing yards per game seems to have a skew. Normality can be better viewed using a quantile-quantile plot. For a truly normal distribution, the quantile-quantile plot should show a straight line. Deviations from a straight line might indicate a skew in the data. The quantile-quantile plot for these two features is shown in Fig. 3.

Note that the visitor rushing defense is quite linear on the quantile-quantile plot, suggesting that the normality assumption might be reasonable. However for the home team rushing offense, we see that at large values the quantile-quantile curve is highly non-linear, which is a manifestation of the skew observed from simply looking at the histogram. It might be possible to apply a transform to the data to eliminate the non-normality. Indeed, when a

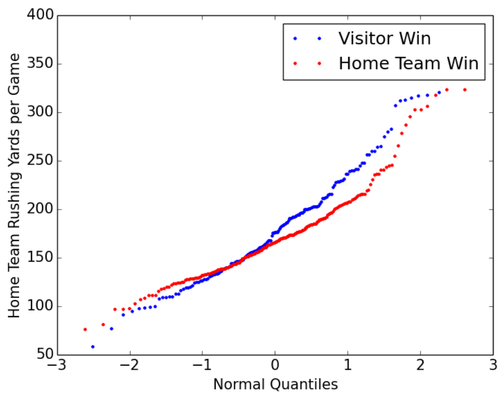


(a) Home Team Rushing Offense

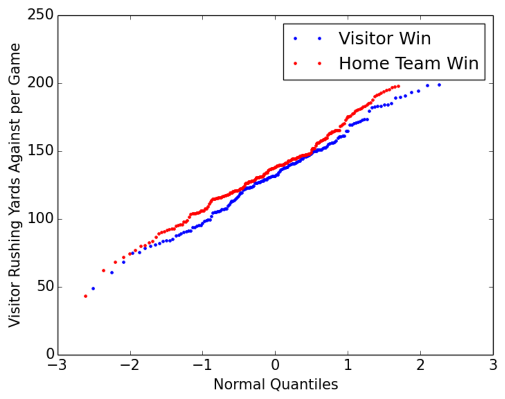


(b) Visitor Rushing Defense

Figure 2: Observed distributions of home team offensive rushing yards per game and visitor defensive rushing yards against per game.



(a) Home Team Rushing Offense



(b) Visitor Rushing Defense

Figure 3: Normal quantiles for home team offensive rushing yards per game and visitor defensive rushing yards against per game.

logarithmic transformation is applied to the home team rushing offense, the quantile-quantile plot becomes far more linear. However, transformations did not seem to impact the results greatly as many of the methods do not require the normality assumption. Therefore, the remaining results presented here will use non-transformed data.

Finally, to understand the interaction between these two chosen features, we plot them together in Fig. 4. Notice that the data does not seem to be very separable. Similar results are observed for all pairs of features in the data set. This hints that correctly classifying bowl game results might be a difficult task.

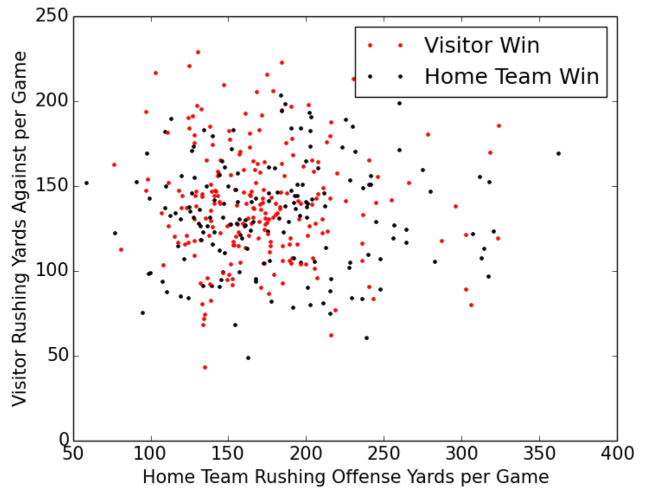


Figure 4: Bowl game results plotted against home team rushing offense and visitor rushing defense.

Perhaps a more accurate view of this data can be determined using principal component analysis (PCA) which applies a singular value decomposition to the feature data and returns the most significant singular vectors. These singular vectors can be a composition of any number of features. The two most significant features are shown in Fig. 5. We see once again that there is no obvious separation in the data.

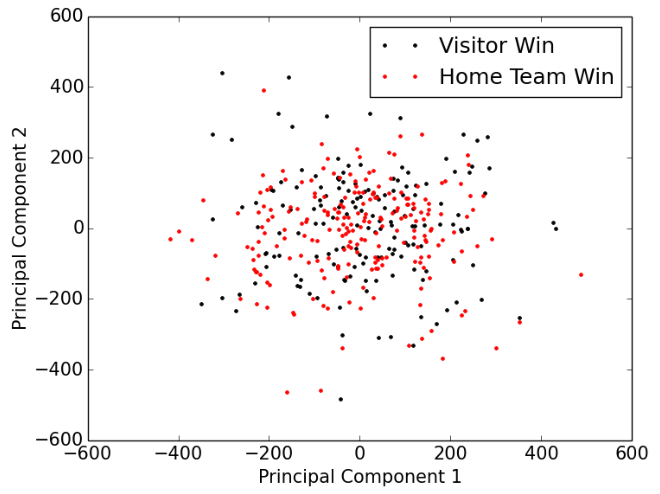


Figure 5: Game outcomes against the two most significant principal components of the feature data for prediction home team wins.

4 Model Development

4.1 Predicting Home Team Wins

Several different models are developed using a variety of algorithms. These include decision trees with pruning, random forests, logistic regression, Naive Bayes, and Support Vector Machines (with both linear and Gaussian kernels). All of these models utilize validation data to tune hyper-parameters. For all models except the decision tree, the data is first transformed so that the training data has mean zero and standard deviation one for all features. This allows many of the algorithms to run more efficiently and converge to a reasonable solution. This transformation is not applied to the data given to decision trees since it should not make any difference. In the case of Support Vector Machines (SVM), the Gaussian kernel utilizes an L-2 regularization whereas the linear kernel uses an L-1 regularization to promote sparsity. The L-2 regularization is also applied for logistic regression.

The results of these models on the data are shown in Tab. 1. Notice that all models have similar misclassification errors of near 40%. This error seems to be very high, but perhaps 60% accuracy on game predictions is as good as we can expect. However, notice that some models have very high validation misclassification errors. Particularly, the random forest model has low training error (15%) but high validation error (43%). The best model in terms of test error is the logistic regression model, but this model once again observes poor validation accuracy.

Table 1: Classification accuracy of several models applied to college football bowl data for predicting home team victories.

	Misclassification Error		
Model	Training	Validation	Test
Decision Tree w/ Pruning	0.330	0.374	0.377
Random Forest	0.147	0.426	0.377
Logistic Regression	0.173	0.435	0.325
Naive Bayes	0.304	0.409	0.364
SVM (Gaussian Kernel)	0.304	0.383	0.390
SVM (Linear Kernel)	0.288	0.417	0.416

The linear SVM model with L-1 regularization shows the worst test accuracy but does provide insight into the data through sparsity. After the tuned regularization penalty on validation data is applied, the linear SVM only uses 16 of the 97 available features. This motivates using only this reduced set of features in the other models. These 16 features are listed in Fig. 6. Of these 16 features, the line had by far the greatest weight.

All the models except the linear SVM model are trained on this reduced dataset. The results are shown in Table 2. With this reduced dataset we see an improvement in the random forest model, the Naive Bayes model, and the SVM with a Gaussian kernel. However, the logistic regression model increased its misclassification errors and the decision tree remained

- Home Team Offensive Rush Yards
- Visitor Offensive Pass Yards
- Home Team Offensive Pass Yards vs. Bowl Teams
- Home Team Offensive Pass Attempts vs. Bowl Teams
- Home Team Defensive Pass Interceptions
- Home Team Defensive Rush Attempts
- Visitor Defensive Pass Interceptions
- Visitor Defensive Pass Completions
- Home Team Defensive Pass Interceptions vs. Bowl Teams
- Home Team Defensive Pass Completions vs. Bowl Teams
- Visitor Defensive Pass Interceptions vs. Bowl Teams
- Visitor Defensive Pass Completions vs. Bowl Teams
- Visitor Defensive Pass Yards vs. Bowl Teams
- Visitor Points Against vs. Bowl Teams
- Visitor Takeaways vs. Bowl Teams
- Line

Figure 6: Features with nonzero coefficients in the linear SVM model with L-1 regularization.

Table 2: Classification accuracy of several models applied to college football bowl data for predicting home team victories on a trimmed dataset.

	Misclassification Error		
Model	Training	Validation	Test
Decision Tree w/ Pruning	0.330	0.374	0.377
Random Forest	0.073	0.426	0.364
Logistic Regression	0.283	0.452	0.351
Naive Bayes	0.267	0.417	0.299
SVM (Gaussian Kernel)	0.293	0.383	0.351

exactly the same. Digging further into the structure of the decision tree, we see that it is simply a decision stump based on the line, as shown in Fig. 7.

This result is quite unsatisfying since a model based purely on the Vegas betting line does not provide any interesting information. This indicates that any trends found further down the tree were pruned because they over-fitted the data and did not perform well in validation. The pruning is conducted by first fully forming the decision tree with no regularization penalty. Then a regularized error ϵ_R is calculated as

$$\epsilon_R = \sum_{i=1}^{N} |y_i - f(x_i)| + \alpha |T| \tag{1}$$

where $\sum_{i=1}^{N} |y_i - f(x_i)|$ represents the training misclassification error and $|T|$ represents the number of leaves in the tree. The parameter α is determined from validation data yielding the smallest misclassification error. Starting at the leaves of the tree, decision nodes are turned into leaves if the regularized error does not increase with its removal. This result implies that only the decision based on the Vegas betting line generalized well. However, it is

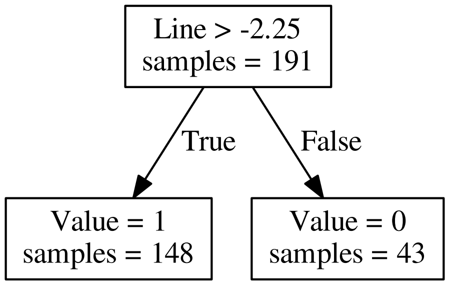


Figure 7: Decision stump formed from the decision tree algorithm.

interesting to note that the decision boundary is not taken at zero. Intuitively, if a decision stump were created with the line it should be about zero (meaning even odds). This is a pure manifestation of statistical fluctuation in the training data. It seems that the partitioned training data has more home team wins than losses. Since bowl games are played on neutral fields, there should be no large preference for one side or the other. The home and away titles are nearly randomly assigned.

4.2 Predicting Upsets

This motivates a re-casting of our problem. Instead of assigning labels based on whether the home team wins or loses, the problem is recast in terms of whether an upset occurs. More specifically, the label 0 is assigned to games in which the favored team wins and the label 1 is assigned to games in which the underdog wins. This requires a slight reworking of the features to be in terms of favored and underdog teams. In addition, the absolute value of the line is presented since the line is given in terms of how much the home team is favored. After taking the absolute value, it represents the general confidence that the favored team will win. A similar casting has been attempted by some in the past with limited success [3].

Despite the change of labels, many of the properties observed before are still true. For instance, the data is still highly mixed and no clear separation can be visible with respect to a few features by the human eye. This can be observed by again performing a principal component analysis and looking at how the labels correlated with the two largest principal components. This is shown in Fig. 8 which shows little difference from the previous analysis shown in Fig. 5.

The results of the new formulation are presented in Table 3 using the same train, validate, and test partitions as before. We see that most models have comparable misclassification errors with the predictions based on the home team winning. However, for the best performing models such as SVM, logistic regression, and random forest, the validation errors and

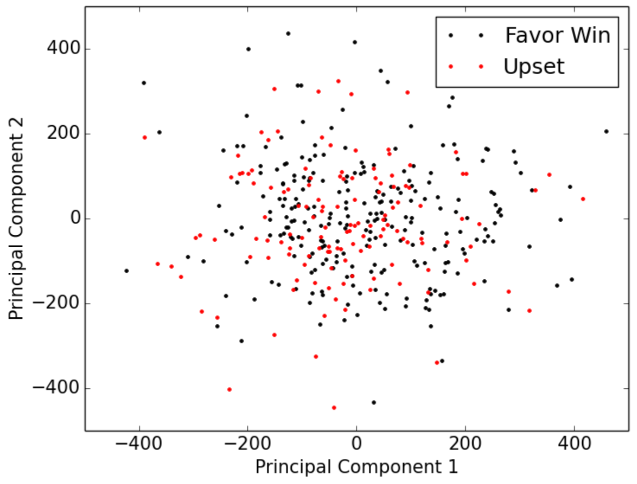


Figure 8: Game outcomes against the two most significant principal components of the feature data for predicting upsets.

Table 3: Classification accuracy of several models applied to college football bowl data for predicting bowl game upsets.

	Misclassification Error		
Model	Training	Validation	Test
Decision Tree w/ Pruning	0.335	0.383	0.364
Random Forest	0.099	0.365	0.377
Logistic Regression	0.168	0.348	0.338
Naive Bayes	0.330	0.504	0.429
SVM (Gaussian Kernel)	0.283	0.330	0.364
SVM (Linear Kernel)	0.293	0.374	0.364

test errors are now both reasonably low. Naive Bayes seems to have great difficulty on this dataset. This is likely due to two factors. First, as we observed in section 3, some features have significantly non-normal distributions, violating the assumptions of Naive Bayes. Second, Naive Bayes does not have the same straightforward tuning parameters as other models so generalizing to new data can be difficult.

Just as before, we can use the nonzero weighted features from the linear SVM to form a trimmed dataset on which to train. This time, there are just 13 features and no single feature has incredibly high weight – unlike our previous problem where the line received a very large weight. The features for the trimmed dataset are given in Figure 9.

Again, all models aside from the linear SVM model are re-trained on the reduced dataset. The results are displayed in Table 4. Notice that the reduced set helps the Random Forest

- Favored Team Giveaways vs. Bowl Teams
- Favored Team Defensive Completions vs. Bowl Teams
- Favored Team Defensive Fumbles vs. Bowl Teams
- Underdog Defensive Pass Interceptions
- Underdog Defensive Pass Completions
- Underdog Offensive Rush Yards
- Underdog Average Points
- Underdog Average Points Against vs. Bowl Teams
- Underdog Losses vs. Bowl Teams
- Underdog Fumbles vs. Bowl Teams
- Underdog Defensive Pass Yards vs. Bowl Teams
- Difference in Offensive Pass Yards vs. Bowl Teams
- Magnitude of the Line

Figure 9: Features with nonzero coefficients in the linear SVM model on upset predictions with L-1 regularization.

model and the Naive Bayes model. This is probably because these two models do not have a regularization penalty as straightforward as SVMs and logistic regression which can directly penalize complexity. Therefore, the reduced complexity of the trimmed feature set aids those models.

Table 4: Classification accuracy of several models applied to college football bowl data for predicting upsets on a trimmed dataset.

	Misclassification Error		
Model	Training	Validation	Test
Decision Tree w/ Pruning	0.335	0.383	0.364
Random Forest	0.251	0.383	0.338
Logistic Regression	0.298	0.409	0.416
Naive Bayes	0.309	0.435	0.416
SVM (Gaussian Kernel)	0.325	0.348	0.364

Also, notice that the decision tree again is unchanged. Upon further investigation the pruned decision tree is less than a stump – it is just a leaf. The decision tree always predicts that the favored team will win. This is again unsettling, but does provide a basis with which we can compare the effectiveness of the models. It is clear that for any model to be legitimate, it must be more accurate than the decision tree model. Therefore, we can disregard all models aside from the random forest on the trimmed dataset and all models aside from the logistic regression model on the full dataset.

If one model must be selected, it seems that the logistic regression model on the full data set would be preferred. It has low error on all train, validate, and test datasets and seems to be relatively robust. The classification matrix for this model on test data is presented in Table 5.

We see that the classification matrix seems to be relatively well behaved in the sense

Table 5: Classification matrix based on test data for the logistic regression model in predicting upsets.

	Actual	
Predicted	Favored Team Wins	Upset
Favored Team Wins	38	15
Upset	11	13

that we are making approximately equal type 1 and type 2 errors. Still, our accuracy is not as good as I had hoped entering this project. This has been observed in the past by other attempts [4, 5] at predicting college football outcomes. One reason college football bowl games are particularly hard to predict is that competing teams are ideally chosen of similar strength. Therefore an advantage one team has in a few features might be compensated by a disadvantage in other features. Adding in large statistical noise associated with sports games, it can be particularly hard for algorithms to decipher trends. Even if the true underlying trends are recovered from the data, the statistical variation might be so large that misclassification error remains large.

Finally, with the logistic regression model chosen as the best model, we can use it to easily extract probabilities of upsets. This should give us information about the confidence level on each game. The strategy would be to pick the team with the highest probability of winning and then place the highest amount of points on the games 1 through N where the probability of winning is the highest and place the lowest points on the games with lowest calculated winning probability. If we try this on our test set taking N to be the number of data points in the test set (77), we calculate a **final score of 2043**.

This seems quite good as the highest number of possible points is 3003 so we are receiving 68% of the possible point total. However, is this success due to the point placement strategy? To answer this question, we should test against a null distribution where point values are randomly assigned. Since we wish to test only the point placement strategy, we should hold the number of correct predictions constant. Through repeated trials, the point placements are randomly shuffled to form a distribution of point totals in which 49/77 games are correctly predicted, the accuracy of our logistic regression model. The resulting distribution is shown in Fig. 10. From this distribution we can see that our value is higher than the mean, which is 1989. However, we our point total is not very significant as 2043 lies within one standard deviation of the Gaussian distribution. Altogether, while our model performs better than average, it is difficult to prove that it is significantly better than simply betting with the Vegas line and randomly assigning confidence points.

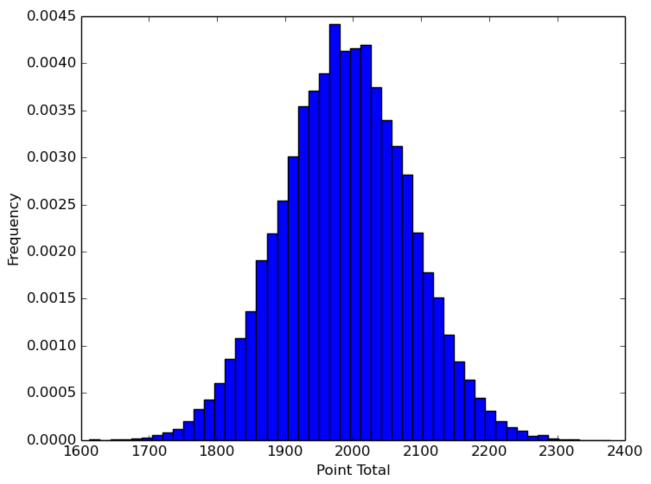


Figure 10: Distribution of point totals by randomly selecting confidence points assignments assuming 49/77 games were correctly predicted.

5 Conclusion

In this investigation, I attempted to form college football bowl predictions based on the statistics of the competing teams. This proved difficult as a clear separator in the data was not visible. The conditional distributions of the statistics for each class (wins and losses) largely overlapped. I tried re-casting the problem in terms of upset prediction which seemed a better posed problem. However, success was still marginal. The problem of predicting college football games is notoriously difficult due to the selection of teams to be evenly matched. Some of the final models developed did score higher than simply predicting the favored team, but the difference is not very significant. These models also allow for an estimation of the probabilities. Using these criteria, it is possible to form a ranking of the highest confidence games which resulted in a point total greater than that expected from a random assignment of confidence points. However, the difference once again is not very significant.

The models presented here could possibly be improved through the formation of better features. For instance, more intricate models for season performance could be determined based on the opponents. In the models developed here, a whole set of features is dedicated to performance against teams that finished their season with a bowl game. These features seem to be useful as many models prefer them over the season averaged features across all games. This could be taken a step further to include more comprehensive information regarding the team's opponents. There are a large number of other ways in which features could be added to the models to try to improve them, but the models and feature set presented here form a good starting point in which to understand the underlying behavior of college football bowl games.

Appendix A

This appendix

```
import numpy as np

# set random number seed
np.random.seed(0)

# load raw data
raw = dict()
for typ in ['stats', 'bowls']:
    raw[typ] = dict()
    for year in range(2002,2015):
        raw[typ][year] = dict()

        # direct to locations of stats and bowl results files
        if typ == 'stats':
            fname = 'Data/stats/cfb' + str(year) + 'stats.csv'
        if typ == 'bowls':
            fname = 'Data/bowls/bowl' + str(year) + 'lines.csv'

        with open(fname,'r') as fh:

            lines = fh.readlines()
            header = lines[0].split('\r')
            header = header[0].split('\n')
            header = header[0].split(',')

            # populate keys of dictionary
            for col in header:
                raw[typ][year][col] = list()

            # fill dictionary with values
            for line in lines[1:]:
                words = line.split(',')
                for i, word in enumerate(words):
                    try:
                        val = float(word)
                    except ValueError:
                        val = str(word)

                    raw[typ][year][header[i]].append(val)
```

```
# define alternate team names
alt_name = dict()
alt_name['UTEP'] = 'Texas-El Paso'
alt_name['U-A-B'] = 'Alabama-Birmingham'
alt_name['Troy'] = 'Troy State'

# delete games with unknown scores
for year in range(2002,2015):
    delete_ind = set()
    for i, val in enumerate(raw['stats'][year]['ScoreOff']):
        if val == ' ':
            delete_ind.add(i)
    if len(delete_ind) > 0:
        for col in raw['stats'][year].keys():
            new_list = list()
            for i, val in enumerate(raw['stats'][year][col]):
                if i not in delete_ind:
                    new_list.append(val)
            raw['stats'][year][col] = new_list

# create a set of bowl teams for each year
bowl_teams = dict()
for year in range(2002,2014):
    bowl_teams[year] = set()
    for team_status in ['Home Team', 'Visitor']:
        teams = raw['bowls'][year][team_status]
        for team in teams:
            bowl_teams[year].add(team)

# get list of game indexes for each team
team_games = dict()
opp_games = dict()
for year in range(2002,2014):
    team_games[year] = dict()
    for i, team in enumerate(raw['stats'][year]['TeamName']):
        if team not in team_games[year]:
            team_games[year][team] = [i]
        else:
            team_games[year][team].append(i)
    opp_games[year] = dict()
    for i, team in enumerate(raw['stats'][year]['Opponent']):
        if team not in opp_games[year]:
            opp_games[year][team] = [i]
        else:
            opp_games[year][team].append(i)

# define which entries not to record for stats
pass_attr = set(['TeamName', 'Opponent', 'Line', 'Site', 'Date'])

# form team data for each year
team_data = dict()
for year in range(2002,2014):
    team_data[year] = dict()
    for team in bowl_teams[year]:

        # check that team name is in the dictionary
        if team not in team_games[year]:
            team = alt_name[team]
        if team not in team_games[year]:
            print "Error: failed to find team named ", team
            exit()

        team_data[year][team] = dict()

        # populate dictionary
        for typ in ['all','bowl']:
            team_data[year][team][typ] = dict()
            for stat in raw['stats'][year].keys():
                if stat in pass_attr:
                    continue
                else:
                    team_data[year][team][typ][stat] = 0
            team_data[year][team][typ]['Matches'] = 0
            team_data[year][team][typ]['Wins'] = 0
            team_data[year][team][typ]['Losses'] = 0

        # get summed attributes for each requested attribute
        for game_ind in team_games[year][team]:

            # check if opponent is bowl-bound
            bowl_bound_opp = raw['stats'][year]['Opponent'][game_ind] \
                    in bowl_teams[year]
            # add stats for each category from the game
            for stat in raw['stats'][year].keys():
                if stat in pass_attr:
                    continue
                else:
                    stat_val = raw['stats'][year][stat][game_ind]
                    stat_val = float(stat_val)

                    team_data[year][team]['all'][stat] += stat_val
                    if bowl_bound_opp:
                        team_data[year][team]['bowl'][stat] += stat_val

            # determine win or loss
            if raw['stats'][year]['ScoreOff'][game_ind] \
                    > raw['stats'][year]['ScoreDef'][game_ind]:
                result = 'Wins'
            else:
                result = 'Losses'

            # increment number of games, record result
            team_data[year][team]['all']['Matches'] += 1
            team_data[year][team]['all'][result] += 1
            if bowl_bound_opp:
                team_data[year][team]['bowl']['Matches'] += 1
                team_data[year][team]['bowl'][result] += 1

# make game-averaged stats and record winning %
for year in team_data:
    for team in team_data[year]:
        for typ in ['all', 'bowl']:

            # record winning percentage
            season_stats = team_data[year][team][typ]
            N = season_stats['Matches']
            if N == 0:
                season_stats['WinPct'] = 0.0
            else:
                season_stats['WinPct'] = float(season_stats['Wins']) / N

            # normalize stats by the number of games
            for stat in raw['stats'][year].keys():
                if stat in pass_attr:
                    continue
                else:
                    if N != 0:
                        season_stats[stat] /= N

# Add derived features
for year in team_data:
    for team in team_data[year]:
        for typ in ['all', 'bowl']:

            season_stats = team_data[year][team][typ]
            season_stats['Takeaways'] = season_stats['PassIntDef'] \
                    + season_stats['FumblesDef']
            season_stats['Giveaways'] = season_stats['PassIntOff'] \
                    + season_stats['FumblesOff']
            season_stats['TOMargin'] = season_stats['Takeaways'] \
                    - season_stats['Giveaways']
'''
# print team stats for a year
for stat in team_data[2013]['Michigan']['all'].items():
    print stat
'''

# recast data in terms of underdogs and favorites
for year in range(2002,2014):
    N = len(raw['bowls'][year]['Home Team'])

    for team_status in ['Underdog', 'Favored']:
        if team_status not in raw['bowls'][year]:
            raw['bowls'][year][team_status] = list()
            raw['bowls'][year][team_status + ' Score'] = list()

    # go through all games
    for i in range(N):

        # extract the line
        line = raw['bowls'][year]['Line'][i]

        # determine the index of the favorite
        if line > 0:
            index = 0
        elif line < 0:
            index = 1
        else:
            # flip a coin
            if np.random.random() > 0.5:
                index = 0
            else:
                index = 1

        # extract data
        home = raw['bowls'][year]['Home Team'][i]
        home_score = raw['bowls'][year]['Home Score'][i]
        visit = raw['bowls'][year]['Visitor'][i]
        visit_score = raw['bowls'][year]['Visitor Score'][i]

        # create lists for iterationd
        teams = [home, visit]
        scores = [home_score, visit_score]
        favor = ['Favored', 'Underdog']

        # add data to dictionary
        for j in range(2):
            idx = abs(index - j)
            raw['bowls'][year][ favor[j] ].append(teams[idx])
            raw['bowls'][year][favor[j]+' Score'].append(scores[idx])


# form X-data
X = list()
Y = list()
for year in range(2002,2014):
    N = len(raw['bowls'][year]['Favored'])
    for i in range(N):
        xvector = list()
        xheader = list()
        for team_status in ['Favored', 'Underdog']:

            # get team name
            team = raw['bowls'][year][team_status][i]
            if team not in team_data[year]:
                team = alt_name[team]

            # add stats
            for typ in ['all', 'bowl']:
                for stat in team_data[year][team][typ].items():

                    # determine name for stat
                    stat_name = team_status + '␣' + typ + stat[0]
                    xheader.append(stat_name)

                    # add stat value
                    xvector.append(stat[1])

        # get teams involved in matchup
        bowl_year = raw['bowls'][year]
        teams = [bowl_year['Favored'][i], bowl_year['Underdog'][i]]
        for k, team in enumerate(teams):
            if team not in team_data[year]:
                teams[k] = alt_name[team]

        # get matchup statistics
        for k in [0, 1]:
            offense = teams[k]
            defense = teams[1-k]
            for typ in ['all', 'bowl']:
                for attack in ['Pass', 'Rush']:

                    # determine name for stat
                    stat_name = ''
                    if k == 0:
                        stat_name += 'Favored'
                    else:
                        stat_name += 'Underdog'
                    stat_name += '␣Matchup␣' + attack + '␣' + typ
                    xheader.append(stat_name)

                    # get stat value
                    off_stats = team_data[year][offense][typ]
                    def_stats = team_data[year][defense][typ]
                    val = off_stats[attack + 'YdsOff'] \
                            + def_stats[attack + 'YdsDef']
                    xvector.append(val)

        # add line
        line = raw['bowls'][year]['Line'][i]
```

```
        xheader.append('Line')
        xvector.append(abs(line))

        # add result
        favored_score = raw['bowls'][year]['Favored␣Score'][i]
        underdog_score = raw['bowls'][year]['Underdog␣Score'][i]
        if underdog_score > favored_score:
            Y.append(1)
        else:
            Y.append(0)

        # add feature vector
        X.append(xvector)

N = len(Y)
with open('formed_data.csv', 'w') as fh:
    for header in xheader:
        fh.write(header + ',')
    fh.write('Result\n')
    for i in range(N):
        for j in range(len(X[i])):
            fh.write(str(X[i][j]) + ',')

        fh.write(str(Y[i]))
        if i != N-1:
            fh.write('\n')
```

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Sunshine Forecast Downloadable Data Files. College football cumulative statistics file. http://www.repole.com/sun4cast/data.html.

[3] A.S. Padron and J.D. Sinsay. Upset prediction in college football. *CS229, Machine Learning, Stanford University*, Autumn 2013.

[4] B. Liu and P. Lai. Beating the ncaa football point spread. *CS229, Machine Learning, Stanford University*, December 2010.

[5] John Hamann. What it takes to win: A machine learning analysis of the college football box score. *CS229, Machine Learning, Stanford University*, December 2011.