Abstract

It has been observed over time that football teams with a huge increase in performance will cause an increase in the amount of applications received in the following year. We would will use classifiers to see if we can predict whether or not a school will see an increase in the number of applications received based on the win-loss record of their football team for the 3 years following that season. In this study we make use of the ADTree, J48, LADTree and Random Forest decision tree methods as well as conjunctive Rule and Decision Table rule sets to perform our analysis. To do this we first gathered data from the IPEDS data set for characteristics about the schools and sports-reference for data on the Division I schools. Python is used to clean the data which is then analyzed through use of WEKA 3.6.13. We have found that with classifiers we could predict with up to 74% accuracy whether or not a school would see an increase in applications received. However, we could not differentiate if the increase was observable from the win-loss record or the included characteristics of the school.

1. Introduction

In 1985 Boston College saw a rise of 30% in the number of applications that it had received compared to the previous years. What changed? In their 1984 football season, their quarterback Doug Flutie threw a “Hail Mary”1 pass in the final seconds of their Thanksgiving Day game to win. Since then the increase in the number of applications a school receives due to athletic success has been known as the “Flutie Effect” (Chung 2013). This has suggested the idea that by having a successful football team a college can increase the number of applicants it will receive.

Research into the Flutie Effect has gone beyond just studying the increase of the number of applications that a college will receive. Research done by Pope & Pope in 2008 investigates the quality of the incoming applications as well. While their findings found that the increase in number of applications did not mean a decrease in the quality of applications as a whole other research has found that the success will cause a longer increase in applications from students with SAT scores below 900 (Chung 2013).

This research however has been limited by the methods used to study the impacts of athletic success. Most research to date has relied on the use of statistical regressions to analyze the data. Creating statistical regressions relies on the idea that there is already some linear equation that fits the data (Crutzen&Giabbanelli 2014). Thus the regressions will limit the connections that you are able to observe. Second, the regression methods used to study the change in applications due to athletic success are mostly similar due to the limited regressions used to study changes in higher education. Third, in previous studies the researchers had to specifically decide what it meant for a team to achieve athletic success. This has caused studies to measure success by different methods, where some have measured straight win-loss records, others have viewed it as a change in win percentage, along with some studies that have only measured who was winning championships. This has caused a limiting factor on how to study what it means for a colleges team to have athletic success. Due to the need for linear data in regression studies athletic success had to be defined this way. By observing the first recorded instance of the Flutie Effect, when Boston College made a final play for the win in the last

---

1 A desperate pass with only a small chance of success
seconds is what most studies attribute to the raise in applications. This suggests that we are not looking for a linear equation that is already fit in the data but events that are different from the norm. Thus in this study by using machine learning we are able to expand the type of data we use beyond win-loss statistics but also include factors such as whether or not a team was able to make a bowl game appearance. Finally, current studies use a one year time lag to observe changes, in this study however we will use a 3 year window instead.

In this paper we will analyze athletic success through the use of machine learning instead of statistical methods. By using machine learning we are able to examine much more varied data sets that possible using only statistical methods. While previous methods would limit their analysis only able to properly examine a few attributes that are part of a university in relation to athletic success by using machine learning we can examine any amount of attributes much more easily. Along with that, through use of machine learning we can compare not only the record of given teams, but also apply whether or not, championships or bowl appearances that can cause an increase in publicity for a given university.

In section 2 we will review the literature on how the Flutie effect impacts the quantity and the quality of students to universities. In section 3 we will explain how the data was obtained and cleaned for use in our analysis. Section 4 will introduce our analytical methods through the use of WEKA 3.6.13 and present the results of our findings in section 5. Throughout section 6 we will give discussion along with implications this work can mean for universities. Finally, in section 7 we will conclude

2. **Background**

Research into the Flutie effect has generally looked for two correlations. First, has there been a change in the number of applications following a successful athletic season. The second being is there a change in the quality of the applications after this surge. While less so than the two main topics, there has been research into the finances of universities in relation to their athletic programs. In this section we will discuss how past research has addressed these aspects.

2.1 **Quantity of Students**

In multiple studies it has been found that having a successful athletic season causes an increase in the number of applications that a university receives. The original study done by *Murphy and Trandel (1994)*, found that a winning season has a significant impact on the amount of applications. This study, of which many other studies have been based, was performed on division 1 universities, and the impact was assessed through the use of a regression method (i.e., the ordinary least squares method). Since then many studies have observed the Flutie Effect, though it should be noted that these studies have differed in how they measured athletic success. In the work done by *McEvoy (2005)*, the focus was on application amounts done by changes in win percentage. It was found that a record increase by .250 in a single season caused an increase in the number of applications. Note however, that a decrease in the team’s record by .250 did not cause a decrease in the amount of applications. In concurrence with that result, it was found that if a team were to change from a bad team to a great there can be an increase in the amount of applications by up to 18.6% (Chung 2013). These studies however focus on Division I university seasons. In a study done on Division II universities a winning football season was seen to have no real effect on the number of applications for the university (Castle &Kostelnik 2011).

2.2. **Quality of Students**
While there is agreement that a successful football season causes an increase in applications there is currently little consensus on how the quality of these applications is impacted by the success. Studies that look into the potential change of quality in students focus on Division I universities. In these studies, the quality of the applicants is determined by their SAT score. Scores are considered poor if they are under 900. At the time of these studies the SAT was scored out of 2400 with three 800 point sections. It was found that after a successful season, twice as many students who scored poorly on the SAT responded than those who did not. The study deduced though that since there was an increase in the number of applicants it allowed the schools to be more selective and thus retain their previous quality, though not improve it (Pope and Pope 2008). However, it has also been observed that the influence caused by a winning season will last longer on students with poor scores than those with higher scores (Chung 2013). From this it can be concluded that while the school may be able to maintain quality for the immediate following year from the conclusions by Pope and Pope the years after would only see the increase in poor scores.

2.3 Methodology

The majority of studies apply regression analysis methods to study the impact of a successful athletic season. A common method that was first used in the ordinary least squares (OLS) regression. Studies by Murphy and Trandel (1994), Chung (2013), Pope and Pope (2008) and Castle and Kostelnik (2011) applied such a regression. The difference in the studies would by while murphy and Trandel worked with Division I schools, Castle and Kostelnik studied Division II schools. Alternatively, Chung treated a winning season as a stock of good will that would decay over time, thus adding an attribute to the OLS regression. Pope and Pope used an Economic specification model of the OLS to fit a linear trend for each year. These methods lagged the data on the football team one year to get accurate results. Thus those using OLS were only able to apply the success to a single year other than Chung. An alternative method applied by McEvoy (2005) was Analysis of Variation. The final method observed compared percent change in the amount of applicants before and after a championship season (Tom & Cross 1996). In these studies how they measured athletic success. Studies either use win-loss records or a team’s results ina championship series. Our study will use machine learning and decision trees to observe possible connections in the data. Furthermore, we will look at not only the win loss records of the teams but whether or not they made it into a bowl game that season, something which could not be considered previously by regression tests. Finally, our study will use a three year time frame in our analysis to observe changes not just year by year, but of the course of three years.

3. Data Sources and Data Cleaning

3.1 Overview

In this study the focus has been to gather large amounts of data and use machine learning to find the connections between a successful football season and enrollment trends. Two criteria were used to decide the time span for data collection. First, we wanted to observe not only how success in one year affects enrollment in the next one, but whether this affects lasts up to 2 or 3 years. This is one of our contributions compared to past research which limited the investigation to a one year effect. Second, we used machine learning as an exploratory tool to generate patterns: we did not hypothesize that any
specific year would have patterns of interest, but rather swept a large number of years. Consequently, we gathered data for a span of 11 years (2004-2014).

The data on universities was gathered from the Integrated Postsecondary Education Data System (IPEDS). All athletic information involving their win-loss record for a season and whether or not they were in a bowl game for Division I schools was gathered from sports-reference (http://www.sports-reference.com/cfb/). The win-loss records for Division III schools was gathered from the d3football (http://www.d3football.com/teams/index). In this section we will focus first on how we collected the data, and then how we cleaned it for our final format.

3.2 Data Collection

The academic data from the IPEDS data set consists of 524 files (1.5 Gb) of schools that are ranked as division I or division III in football. The data set consists of survey data (provided by IPEDS as a CSV file) and accompanying dictionary (provided by IPEDS as an Excel spreadsheet from 2009 onward, and as HTML file before). All files manually downloaded. Given the number of files, this was considered a more time-efficient option than developing Selenium scripts for data collection. The Surveys are given at (http://nces.ed.gov/ipeds/datacenter/DataFiles.aspx) and the user must manually select the year of which surveys are desired. There is also the option to limit which surveys are visible by different sections, (i.e. Institutional Characteristics, Admissions and Test Scores...). Surveys answered by each school included but not limited to, data on institution characteristics, fall enrollment, graduation rates, and staffing information. The schools that are included in the data set before cleaning are any schools that answer the surveys for the IPEDS which includes and extends beyond the NCAA schools we are looking for.

The data for the Division I football teams was gathered by manually downloading the CSV files provided by sports-reference. A total of 132 files (each corresponding to one school) was obtained, totaling 676 Kb. By accessing the list of schools found at (http://www.sports-reference.com/cfb/schools/) we took only the schools that were still active in the present year which is determined by the “To” column on the webpage given. We clicked on each university, which opened the page displaying the win-loss record, the conference they are in, the difficulty rating for their season, who the coach was that year as well as whether or not the team participated in a bowl game (and whether they won or lost it). On that page we select the download CSV option to retrieve a CSV formatted file of the team’s records.

The data for Division III schools was collected through use of a script (Appendix 3). A total of 245 files (each corresponding to one school) was obtained, totaling 13.3 Mb. The list of Division III schools was accessed through http://www.d3football.com/teams/index and saved as text file. Each line of the file corresponded to a school name, and was transformed into a valid URL to retrieve the corresponding record from the d3football.com website. For example, the school entry ‘Adrian’ generated the URL http://www.d3football.com/teams/Adrian/2015/index, while schools with multiple words such as Buffalo State generated the URL http://www.d3football.com/teams/Buffalo_State/2015/index. Each webpage was downloaded by the script as HTML file.

In summary, the process outlined above resulted in data on the win-loss records of 377 schools, and administrative data (e.g., institutional characteristics, enrollment scores) for the larger set of American schools.
3.3 Data Cleaning

The previous section outlined the process of collecting data, which consequently has to be cleaned. The cleaning process is detailed in this section, including transformation of individual datasets and linking data across datasets. The first problem to be addressed is finding out which schools in the IPEDS data set are members of the NCAA for football (i.e. having a football team that is either Division I or III). IPEDS uses a 6 digit ID (e.g. Vanderbilt University is 221999) to identify each school. Our first course of action is to get these IDs for the NCAA schools only. Within IPEDS, the survey name ‘University Affiliations’ (part of Institutional Characteristics) has a column labeled “sport1”. A ‘1’ in this column indicates membership of the NCAA for football (i.e. Division I/II/III). For year 2014 of the IPEDS dataset, we extracted the list of IDs for schools with NCAA membership. We assumed that membership was static, that is, schools remained in NCAA Division I or III throughout the period under consideration (2004-2014). This assumption was made due to drastic difference in requirements to be either a Division I or Division III school. To be a Division III school according to the NCAA a school needs to have 5 teams for men and women along with offering no financial aid to the student athletes due to being athletes. Contrary to that, to be a Division I school the NCAA requires that a school sponsor seven sports for men and seven sports for women. Beyond that each school in Division I must play a minimum number of games against other Division I schools and 50% of the games above that minimum must also be against Division I schools. Also, teams must meet a minimum financial award system for their student athletes and under the maximum financial award amount. They must also have at least an attendance (or paid for tickets) of 15,000 per home game (NCAA website). With these facts we can safely assume that while it is possible schools may join into different Divisions it is very unlikely that there is such a jump during our time period.

Once we have all the schools we need to distinguish which ones are members of the NCAA which means it is either a Division I, II, or III school. To do this we identify a column in the data to determine which schools are members of the NCAA and run the scripts given in Appendix 4 on the data to retrieve the school names and their unique IDs for the IPEDS data set.

At this point we have the IDs for all schools that answered the surveys and are part of the NCAA. The IPEDS data does not include game records (e.g., win, loss) hence we have to link it with the records separately obtained for Division I (from sports-reference) and Division III teams (from D3football). This linking is non-straightforward because names are not consistent across datasets. Indeed, game records include partial school names (e.g. ‘Wake Forest’ instead of ‘Wake Forest University’), which need to be disambiguated in order to find the corresponding entry in the IPEDS data. A script was designed to assist with the process (Appendix 1) and cases that could not be automated 54 for Division I and 79 for Division III were resolved manually. After matching manually we had to remove 9 schools from the data set for due to being unable to automatically or manually match them. More detail on those is in Appendix 1. After matching we had 122 Division I schools and 237 Division III schools. This gives us a total data set of 359 schools.

Following this we next need to clean the athletic data for Division I and III schools and link them to the IPEDS data. In Appendix 1 we clean the downloaded webpage from D3football.com and extract the win loss records from where they appear on the page. Next we clean the records for the Division I
schools. Since sports reference gives us these in CSV format you can see in Appendix 1 that we just take the needed data of win, loss, and been in a bowl.

Now that we have all the sports data gathered and cleaned we need to merge all the separate IPEDS data and sports data into one file. For this we first find the files in IPEDS that can be merged and try to avoid replicating IDs. After we figure out which information we want from IPEDS we merge the files for each individual year. Then for each year we add the football statistics to that year. With this we have our complete files for each year with college data and their team data. The final step we take in cleaning the data is we merge each year into one large CSV file of size 11,099 Kb.

3.4 Preparing Data for WEKA

Figure 1) Process of cleaning data

3.4.1 Removing Data with Missing Attribute Values

In this thesis, we use WEKA to perform data mining. Consequently, the data that we collected and cleaned needs to be prepared for analysis in WEKA. This section emphasizes what had to be done specifically for the purpose of data analysis, rather than for the general-purpose data cleaning (section 3.3). WEKA takes in several file formats, and we use CSV as it was the format used so far (sections 3.2-3.3). A CSV file is only accepted by WEKA if all columns had the same length. However, the IPEDS survey changes yearly. For example in 2014 the headcount survey has 63 columns and the 2010 survey 123 columns. Such differences exist in each survey between years with the total number of columns increasing and decreasing haphazardly. Figure 1 demonstrates the process we followed for preparing the data for WEKA. We will remove schools missing data, balance the classes created as much as possible and then make use of the SMOTE filter to equalize the data.

Part of the difference is driven by the fact that the IPEDS is optional and some schools may fill it in one year, but not another. We first assessed which schools filled the part of the survey about the number of applications received in a given year (Appendix 2.1). We found 60 schools that had missing data for at least one year and discarded them, thus reducing our dataset to 299 schools. No further schools we deleted for missing data to any other questionnaire that we used in this thesis, such as the total number of students already enrolled in a school (i.e., the “school size”).

3.4.2 Creating Classes

Once we have taken care of the missing data, it means that the attributes of our dataset are ready for processing. Since the problem of interest is classification, it means that we need to prepare both the attributes and the classes for analysis. We used a binary class representing whether or not
there was a statistically significant increase in the amount of applications received. That is, if a university's enrollment changes in line with past years, then there was no 'particular' change. However, if the number of applications deviates from the expected trajectory, then there is a particular change. In considering how much deviation was significant, we also had to ensure that the resulting classes would be balanced. In other words, the threshold to be categorized as 'increasing in applications’ was chosen to avoid categorizing too many schools as ‘increasing’ compared to ‘non-increasing’.

Consider the following example based on University of Alabama at Birmingham in the year 2007. The number of applications it receives from 2004 to 2007 is as follows: 4,020, 4,255, 4,221, and 4,221. Over this 4-year period we compute the standard deviation of the amount of applications received though the following equation:

\[ \text{std} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 \]

with \( N \) being the number of years, \( N = 4 \) for this work, \( x_i \) representing the amount of applications received in a year, and \( \mu \) being the average amount of applications received in the time period.

If the number of applications in larger than \( \mu + \delta \times \text{std} \) then we consider the applications to be increasing. Otherwise, they are not increasing. The value of the threshold \( \delta \) was determined experimentally by computing the class imbalance for the values of \( \delta = 1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5 \).

Imbalance was measured as \( 1 - \frac{x_1}{y_1} + 1 - \frac{x_2}{y_2} + 1 - \frac{x_3}{y_3} \) with \( x \) representing the number of “SAME” defined classes and \( y \) representing the number of “MORE” defined classes. The subscript represents which year that count is for. For example if we were measuring the imbalance for 2007 then \( x_i \) is year plus one meaning 2008. This is demonstrated in Appendix 2.6. Below you can see a table of the imbalances for each year and the total deviation of the imbalances. We chose to measure at .7 times the standard deviation to minimize the imbalances deviation.

Table 1) Class balance at different deviations

<table>
<thead>
<tr>
<th>Deviation</th>
<th>2007 Imbalance</th>
<th>2008 Imbalance</th>
<th>2009 Imbalance</th>
<th>2010 Imbalance</th>
<th>2011 Imbalance</th>
<th>Imbalance Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0.734</td>
<td>1.66</td>
<td>1.417</td>
<td>2.517</td>
<td>4.122</td>
<td>1.302</td>
</tr>
<tr>
<td>1</td>
<td>0.896</td>
<td>1.498</td>
<td>1.227</td>
<td>2.003</td>
<td>2.916</td>
<td>0.787</td>
</tr>
<tr>
<td>0.9</td>
<td>1.067</td>
<td>1.465</td>
<td>1.156</td>
<td>1.656</td>
<td>2.278</td>
<td>0.483</td>
</tr>
<tr>
<td>0.8</td>
<td>1.224</td>
<td>1.465</td>
<td>1.089</td>
<td>0.983</td>
<td>1.552</td>
<td>0.242</td>
</tr>
<tr>
<td>0.7</td>
<td>1.304</td>
<td>1.387</td>
<td>1.065</td>
<td>1.137</td>
<td>1.203</td>
<td><strong>0.128</strong></td>
</tr>
<tr>
<td>0.6</td>
<td>1.516</td>
<td>1.379</td>
<td>0.964</td>
<td>0.943</td>
<td>1.02</td>
<td>0.264</td>
</tr>
<tr>
<td>0.5</td>
<td>1.626</td>
<td>1.348</td>
<td>1.073</td>
<td>0.808</td>
<td>0.927</td>
<td>0.330</td>
</tr>
</tbody>
</table>

Classes were not balanced: one class had up to 2.5 times as many instances as another in the case of 2007 distribution on application amounts for 2010, and the most balanced distribution still had .5 times as many of one instance than another in the case of 2007 distribution on application amounts for 2008. As noted by Poolsawad and colleagues in the case of medical data, it is common to have an
"imbalanced class distribution the imbalance at 0.7 deviation. On such data, learning classification methods generally perform poorly because the classifier often learns better the majority class" (Poolsawad et al., 2014). Indeed, in an unbalanced distribution, classifiers can simply label all individuals as belonging to one class and still achieve a low error rate when that class is sufficiently prevalent.

This problem arises in part because classifiers assume that the training data is balanced and that errors have the same cost (Poolsawad et al., 2014; Rahmand& Davis 2013). Thus, getting all minority classes wrong but the one majority class right is 'as good' from the viewpoint of a classifier as recognizing most minority classes but less of the majority class. However, from a clinical decision-making viewpoint, the objective is to find patterns in the data such that we can understand why individuals end up in a certain class, rather than blindly assign them label without finding mechanisms. Note that other issues than imbalance can affect classification errors, such as class overlaps; these issues are beyond the scope of this manuscript and we refer the reader to (Japkowicz 2003) for a more in-depth discussion.

There are mostly three ways to address the problem of class imbalances: eliminating cases from the majority class (under-sampling), creating new cases for the minority class (over-sampling), or biasing the classifier's algorithm (e.g., using non-uniform error costs depending on class imbalance). These techniques were reviewed in a white paper by (Garcia et al., 2007). In this paper, we use the J48 classifier from the Weka, as it is one of the most commonly used classifiers (Rahmand& Davis 2013). Thus, rather than modifying the algorithm, we used sampling techniques to address the problem of class imbalances. A comparison of sampling methods in (Batista et al., 2005) concluded that "over-sampling methods in general, and Smote-based methods in particular" were very efficient. Thus, we used the Synthetic Minority Over-sampling TECnique (SMOTE), which creates new cases for the minority class by interpolating between existing cases that lie together (Chawla et al., 2002). We used SMOTE 1.0.3 for Weka 3.6.13.

Section 4. Methods

The problem with most methods currently used to study the Flutie Effect is that by using regression methods the base assumption is there is some linear equation that fits the entire data. In this study we will make use of classifiers instead.

4.1 Overview of Classification Methods

At its most basic level a classifier is a function that assigns a label of whether or not the school would see an increase in the amount of applications based on different features. For this the computer is first passed a training set which contains whether or not there was an increase as well as the different variables for each school. Then the system determines how variables are connected to the increase in applications. This will result in a classifier (Crutzen et. al. 2015). We will make use of classifiers to identify which schools could see an increase in the number of applications received.

4.2 Methods Selected for this Study

In this study we apply 2 types of classifiers: decision trees and rule sets. A third option would have been Support Vector Machines(SVM), as these three are the most commonly used types of classifiers (Hamel 2009, Han &Kamber 2006). The possibility of using SVMs is discussed as part of future work while decision trees and rule sets are now introduced in turn.
4.2.1 Decision Tree

Decision trees use a “Divide and conquer” approach to solving problems. Each node of a tree will have a test involving a particular attribute. A test at a node will generally compare the attribute with a constant. However, a tree can compare an attribute with another attribute at a node or even apply a function to an attribute at a node for a test. Once a school has navigates through the nodes it will reach a leaf node which will classify the school as one of our two classes (Witten & Frank pg. 62). By applying this to each instance we can classify the entire set.

Figure 2) J48 Decision tree 2008 study 2009

Figure 3) ADTree decision tree 2008 study 2009

Figure 2 above is the visual representation of the tree derived from the J48 classifier and Figure 3 is a representation of the ADTree classifier. Both of these trees were created analyzing the 2008 data for if there would be an increase in the number of applications in 2009. As one can observe while the data is the same the trees themselves are generated different with different accuracy.
As stated earlier each node of the decision tree that is not a leaf will apply some test to an attribute to determine how it will continue to traverse the tree. With categorical data it can be as simple as, if “A” branch here, if “B” branch there. However we must also consider numeric attributes. For numeric attributes we attempt to restrict the possibilities to a binary split. With this split we try and place the threshold halfway between the values to split as evenly as possible (Witten & Frank pg. 189).

Over time it has been observed that a simple decision tree performs better than complex one. After generating rule the method to reduce the complexity of the tree is called pruning. There are two types of pruning, postpruning and prepruning. Prepruning involves trying to decide when to end a subtree while generating the tree. This method while seems helpful can throw away trees that should remain as it only looks at an individual level. The other method, postpruning will deal with subtrees after the full tree has been created (Witten & Frank pg. 192).

For postpruning there are two different operations considered. These are subtree replacement and subtree raising. In subtree replacement a subtree is replaced with a single leaf node instead. While this may decrease accuracy on the training set, it could make you tree more reliable on the testing data (Witten & Frank pg 192).

The other method that can be applied other than taking no action is subtree raising. Subtree raising will raise a node to replace the node prior to it. The branches from this raised node can be either more subtrees or leaf nodes. This method can be observed in figure 1 where subtree “C” is raised to replace subtree “B”. However since raising can be a potentially time consuming operation it is generally used in raising the subtree of the most popular branch (Witten & Frank pg. 193).

4.2.2 Rule Sets

Rule sets follow a similar pattern to decision trees. Through a series of preconditions similar to the tests conducted at the nodes in a decision tree. These preconditions are then logically ANDed together (Witten & Frank pg 65). Such rules can be taken from decision trees with the precondition being generated at each node to create a rule and one does this from root to leaf. Redundant rules are ten removed from the rule set (Witten and Frank 65). This can include rules that have the same output but rely on different attributes. For example one can get the rules of if a and b then x along with if c and d then x.

4.3 Computing Accuracy

4.3.1 10 fold cross-validation

To study the accuracy of the model we make use of the 10 fold cross-validation. For this the data is divided into 10 approximately equal parts and each part is used for testing of a classifier built on the remaining 10 parts. This gives the advantage of the performance of the classifier being tested on different instances than those used to form it (Crutzen&Giabbanelli 2014).

4.3.2 Confusion Matrix

A confusion matrix is a matrix with a row and column for each class. The elements of the matrix show the number of test examples which the actual class is the row and the predicted class is the...
column. A good result corresponds to larger numbers along the main diagonal and smaller numbers in non-diagonal elements. The accuracy of the model is then determined by the sum of the diagonal over the total number of instances (Witten & Frank pg. 163). For example looking at our test data in WEKA for a tree for 2007 data we get the confusion matrix of Table 2) Confusion Matrix

<table>
<thead>
<tr>
<th>Identified as A</th>
<th>Identified as B</th>
</tr>
</thead>
<tbody>
<tr>
<td>116</td>
<td>56</td>
</tr>
<tr>
<td>100</td>
<td>71</td>
</tr>
</tbody>
</table>

It can be observed in Table 1 we have a total of 343 elements with 116+71=187 correctly classified. This gives the total accuracy as 187/343 = 54.519% accurate for the model. The trees ability to correctly identify the schools that would see an increase in applications is sensitivity and specificity is identifying those that would not see an increase (Giabbanelli & Adams 2016).

4.4 Confounding Factors

When doing classifiers it is important to have enough data for the tree to have a general grasp of your data set when making rules. There are however confounding factors, or factors that matter in your analysis but cannot be accounted for in the data. In our set we can easily identify two confounding factors. Our data for the change in applications does not account for how large or popular a school was beforehand. While our data set does take into account the size of the student body for the year we are studying that is only a small portion of what determines the attractiveness of a school to potential students. Thus there needs to be a solid way to quantify a school for the model with key attributes that will define a baseline for what a school it. The other is whether or not the team is already a winning team. Most studies account for the change being observed more evidently in a team that changed from a losing team to a winning team. It is hard to account for if a team is already a winning or losing team from the data gathered and a separate metric would have to be applied.

Section 5 Results

Table 3) Classifiers used and their Parameters

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Parameters (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADTree</td>
<td>boosts = 10</td>
</tr>
<tr>
<td>J48</td>
<td>confidence factor = .25, min number per leaf=2, num-folds for pruning=3</td>
</tr>
<tr>
<td>LADTree</td>
<td>number of boosting iterations=10</td>
</tr>
<tr>
<td>Random Forest</td>
<td>number of trees=100</td>
</tr>
<tr>
<td>Conjunctive</td>
<td>Folds = 3, min weight = 2</td>
</tr>
<tr>
<td>Decision Table</td>
<td>cross-validation = 1</td>
</tr>
</tbody>
</table>
Table 4) Results for 2008

<table>
<thead>
<tr>
<th>Year of Season</th>
<th>Year of effect</th>
<th>Classifier</th>
<th>Correctly classified instances</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>2009</td>
<td>ADTree</td>
<td>54.5190</td>
<td>0.674</td>
<td>0.415</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>58.0175</td>
<td>0.820</td>
<td>0.399</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>52.7696</td>
<td>0.651</td>
<td>0.404</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Forest</td>
<td>56.8513</td>
<td>0.570</td>
<td>0.567</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conjunctive</td>
<td>55.9767</td>
<td>0.924</td>
<td>0.193</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table</td>
<td>57.1429</td>
<td>0.802</td>
<td>0.339</td>
</tr>
<tr>
<td>2010</td>
<td>ADTree</td>
<td>62.2047</td>
<td>0.589</td>
<td>0.656</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>67.9790</td>
<td>0.589</td>
<td>0.772</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LADTree</td>
<td>63.7795</td>
<td>0.615</td>
<td>0.661</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Forest</td>
<td>64.5669</td>
<td>0.656</td>
<td>0.635</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conjunctive</td>
<td>64.5669</td>
<td>0.672</td>
<td>0.619</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decision Table</td>
<td>60.3675</td>
<td>0.578</td>
<td>0.630</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>ADTree</td>
<td>66.0592</td>
<td>0.714</td>
<td>0.607</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>66.9704</td>
<td>0.791</td>
<td>0.548</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LADTree</td>
<td>66.7426</td>
<td>0.732</td>
<td>0.603</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Forest</td>
<td>74.2597</td>
<td>0.764</td>
<td>0.721</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conjunctive</td>
<td>61.7312</td>
<td>0.927</td>
<td>0.306</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decision Table</td>
<td>60.8200</td>
<td>0.709</td>
<td>0.507</td>
<td></td>
</tr>
</tbody>
</table>

Table 5) Results for 2009

<table>
<thead>
<tr>
<th>Year of Season</th>
<th>Year of effect</th>
<th>Classifier</th>
<th>Correctly classified instances</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>2010</td>
<td>ADTree</td>
<td>62.2995</td>
<td>0.683</td>
<td>0.562</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>54.8128</td>
<td>0.561</td>
<td>0.535</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>60.6952</td>
<td>0.635</td>
<td>0.578</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Forest</td>
<td>65.7754</td>
<td>0.640</td>
<td>0.676</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conjunctive</td>
<td>56.1497</td>
<td>0.386</td>
<td>0.741</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table</td>
<td>57.4866</td>
<td>0.317</td>
<td>0.838</td>
</tr>
<tr>
<td>2011</td>
<td>ADTree</td>
<td>56.9697</td>
<td>0.623</td>
<td>0.518</td>
<td></td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>54.2424</td>
<td>0.772</td>
<td>0.321</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LADTree</td>
<td>53.3333</td>
<td>0.562</td>
<td>0.506</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Forest</td>
<td>53.3333</td>
<td>0.512</td>
<td>0.554</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conjunctive</td>
<td>49.0909</td>
<td>0.556</td>
<td>0.429</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decision Table</td>
<td>54.2424</td>
<td>0.741</td>
<td>0.351</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>ADTree</td>
<td>55.5556</td>
<td>0.529</td>
<td>0.583</td>
<td></td>
</tr>
</tbody>
</table>
Table 6) Results for 2010

<table>
<thead>
<tr>
<th>Year of game</th>
<th>Year of effect</th>
<th>Classifier</th>
<th>Correctly classified instances</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>2011</td>
<td>ADTree</td>
<td>53.5385</td>
<td>0.679</td>
<td>0.382</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>52.0000</td>
<td>0.702</td>
<td>0.325</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>51.3846</td>
<td>0.679</td>
<td>0.338</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Forest</td>
<td>51.0574</td>
<td>0.536</td>
<td>0.485</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conjunctive</td>
<td>51.6616</td>
<td>0.667</td>
<td>0.362</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table</td>
<td>49.8489</td>
<td>0.917</td>
<td>0.067</td>
</tr>
<tr>
<td>2012</td>
<td>2011</td>
<td>ADTree</td>
<td>59.8916</td>
<td>0.567</td>
<td>0.632</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>56.6396</td>
<td>0.439</td>
<td>0.698</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>56.9106</td>
<td>0.588</td>
<td>0.549</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Forest</td>
<td>59.3838</td>
<td>0.566</td>
<td>0.621</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conjunctive</td>
<td>52.9412</td>
<td>0.777</td>
<td>0.291</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table</td>
<td>54.9020</td>
<td>0.806</td>
<td>0.302</td>
</tr>
<tr>
<td>2013</td>
<td>2011</td>
<td>ADTree</td>
<td>55.0000</td>
<td>0.663</td>
<td>0.437</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>58.4211</td>
<td>0.932</td>
<td>0.237</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>55.5263</td>
<td>0.684</td>
<td>0.426</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Random Forest</td>
<td>62.9333</td>
<td>0.670</td>
<td>0.589</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conjunctive</td>
<td>57.0667</td>
<td>0.881</td>
<td>0.268</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision Table</td>
<td>59.2000</td>
<td>0.762</td>
<td>0.426</td>
</tr>
</tbody>
</table>

Table 7) Results for 2011

<table>
<thead>
<tr>
<th>Year of game</th>
<th>Year of effect</th>
<th>Classifier</th>
<th>Correctly classified instances</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>2012</td>
<td>ADTree</td>
<td>60.6860</td>
<td>0.599</td>
<td>0.615</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J48</td>
<td>62.5330</td>
<td>0.672</td>
<td>0.578</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LADTree</td>
<td>59.8945</td>
<td>0.531</td>
<td>0.668</td>
</tr>
</tbody>
</table>
Section 6. Discussion

We made use of classifiers to examine the impact of the success of a school’s football team to changes in the amount of applications. Previous studies of the “Flutie effect” have found that when a team goes from bad to great there can be an increase up to 18.6% in the amount of applications received by the school (Chung 2013). The important part of this study is not to further reinforce the research on the Flutie effect but to observe if we can predict if a school will have an increase in the amount of applications received based on the win-loss record of the school’s football team.

Through the use of a random forest classifier we were able to identify which school would have an increase in applications with 74% accuracy. This indicates that through use of a classifier we were able to predict which schools would have an increase in the number of applications received. This means there could a relationship between the football team’s win-loss record and the number of applications received by the school. However, by observing the results you will notice that the 74% is a clear outlier with many having an accuracy below 60%. Since we know that the Flutie effect exists already to cause an increase in applications received by a school. Our work does not seem to support that athletic performance of a school’s football team alone is enough to indicate if there will be an increase in the number of applications. Thus we suggest that the Flutie effect can be considered an extreme event, which is an abnormally low or high value in some data distribution (Roberts 1999).

Past research has taken a statistical approach to this question and focused on the Flutie effect. Statistical methods however are reliant on assuming there is a linear equation that fits the data and trying to find it (Crutzen & Giabbanelli 2014). This research instead focused on using classifiers which aim
to fit the model to the data instead of the data to the model. This means that once we have the data formatted tests can be run on it, and adding more data is as simple as adding a new column to the data set. Wherein statistics need to modify the equation they are currently working with to attempt to add more data into the set. In this research we looked at the effect of the team’s success over 3 years instead of the usual one-year lag done by most regression methods.

One such limitation is being unable to quantify the individual schools themselves. For each school we only account for the number of students already enrolled in a school to give them some uniqueness. This limited the study to only a very general idea of each school making it hard to see how the athletic success could interplay.

Furthermore, in the use of a decision tree we do not know whether or not the increase came from the win-loss record itself or from the characteristics of the school. We may know that it can predict an increase accurately 74% of the time but it is impossible to tell if this is related to a teams win, losses, or possibly just the number of students enrolled in the school is the key factor.

More so the IPEDS dataset is rather unorganized. The survey changes what information is gathered on a yearly basis along with the format of the data itself. While the data itself is all in the same general area it may be labeled under a different column making it hard to generalize data over multiple years. This limits the ability to analyze the data as a whole, instead forcing the use of handpicking items for analysis which can make it difficult to find the connections that you were not looking for specifically.

To take this work further it could go in several directions. One work would be to observe the effect of removing the number of enrolled students from the data set. This would leave the model to depend entirely on the win-loss record and see if the accuracy is dependent upon the size of the university instead of the win-loss record. Furthermore, finding a fair way to quantify the schools beyond the number of students currently enrolled would allow for further insights that if a good football team increases applications what characteristics of schools lead tend to indicate the good team will cause an increase.

Section 7. Conclusion

We have made use of classifiers to study the effect that a football team has on the number of applications received by a school for the 3 years following a season. We have created a model that can predict with 74% accuracy whether or not a school will have an increase in applications in the 3 years following a football season. Furthermore we have shown that classifiers are a viable option to study the Flutie effect and the change in number of applications a school receives instead of traditional regression methods.
Appendix I Combine Data

May 13, 2016

1 1. Listing all schools and their IDs

The IPEDS data uses a 6 digit ID for each school (e.g., vanderbilt is 221999). This dataset provides information about enrollment trends, diversity of graduating class, etc. What it does not provide is whether that institution’s sport team had a win. This information is found in the SportsReference.com data. Since that website uses an abbreviated version of a school’s name (e.g., wake_forest for Wake Forest University) we need to link the abbreviated version to the school IPEDS ID. This was done as follows: 1. For the 2014 institution characteristics / university information, take first two columns UNITID INSTNM 2. For university athletics the teams school is identified as a shorthand version of the university name of either one or two parts of the full name for ease of saying and writing. This has become such common place that the databases that store the records use these shorthand titles to refer to the given university. 3. The scripts in this section were used to create matchings for D1 and D3.

The following 9 universities were removed: * oklahoma_state could not be matched because within the IPEDS data set there are three schools labeled as Oklahoma State. One being Southwestern Oklahoma State University, Southwestern Oklahoma State University, and Northwestern Oklahoma State University. Being given only the name of oklahoma state it would be impossible to correctly differentiate which entry is the correct Oklahoma State to link to the athletic data. * Augustana, Bethany, Bethel, Westminster(Mo.), and Westminster(Pa.) could not be matched since within the IPEDS data set these names have multiple matches with the same school name (same idea as for oklahoma). * Alfred, State, and Finlandia could not be matched due to the fact that they could not be found in the IPEDS data mined earlier. It is possible that they do not return surveys to IPEDS or that they were not found in the initial mining process. * Northwestern(Minn.) can not be matched because there is no clean identifier of locations for Minnesota and Northwestern.

In [1]: #returns True if the partial college name (e.g. Albion) is all included in the full one (e.g. Alfred)
def match(partial,full):
    full=full.lower()
    partial=partial.lower()
    words=partial.split('_')
    for word in words:
        if word not in full:
            return False
    return True

In [2]: #returns the list of all IDs that match on the partial name
def findMatches(partial):
    f=open("5schoolsID.txt")
    matches=[]
    for line in f:
        content=line.split(',
        if match(partial,content[1]):
            matches.append(content[0])
    f.close()
    return matches
In [3]: # generates a file having matches for each partial name
    # also shows the list of names having either multiple matches or no matches, as they need to be found magically
    def generateMatchesD3():
        f=open("5schools.txt")
        result=open("5combined.txt","w")
        result.write("School,ID\n")
        multipleMatches=[]
        noMatches=[]
        for line in f:
            content=line.strip('\n')
            matches=findMatches(content)
            if len(matches)>1:
                multipleMatches.append(content)
            elif len(matches)==0:
                noMatches.append(content)
            else:
                result.write(content="","+matches[0]+"\n")
        f.close()

        print "The following",len(multipleMatches),"schools had multiple matches and need to be found magically"
        print multipleMatches
        result.write("\n")
        for school in multipleMatches:
            result.write(school="\n")

        print "The following",len(noMatches),"schools had no matches and need to be found magically"
        print noMatches
        result.write("\n")
        for school in noMatches:
            result.write(school="\n")
        result.close()

In [4]: def generateMatchesforD1():
    f=open("1Schools.txt")
    result=open("1combined.txt","w")
    result.write("School,ID\n")
    multipleMatches=[]
    noMatches=[]
    for line in f:
        content=line.strip('\n')
        matches=findMatches(content)
        if len(matches)>1:
            multipleMatches.append(content)
        elif len(matches)==0:
            noMatches.append(content)
        else:
            result.write(content="","+matches[0]+"\n")
    f.close()

    print "The following",len(multipleMatches),"schools had multiple matches and need to be found magically"
    print multipleMatches
result.write("\n")
for school in multipleMatches:
    result.write(school+"\n")
print "The following",len(noMatches),"schools had no matches and need to be found magically"
print noMatches
result.write("\n")
for school in noMatches:
    result.write(school+"\n")
result.close()

We now must merge the separate school files

In [5]: def mergeSchools(merge1,merge2,outputName):
    inputFile1 = open(merge1,'r')
    inputFile2 = open(merge2,'r')
    outputFile = open(outputName,'w')

    outputFile.write(inputFile1.readline())
    outputFile.write(inputFile2.readline())

    for line in inputFile1:
        outputFile.write(line)

    for line in inputFile2:
        outputFile.write(line)

    inputFile1.close()
    inputFile2.close()
    outputFile.close()

In [6]: #this function will add the division column to the data set for differentiation during analysis
    def addDivisionCol():
        f = open("Scombined_original.txt")
        cleaned=open("ScombinedDiv.txt","w")

        cleaned.write(f.readline().strip('
')","Division"")
        #copy the header line

        for line in f:
            cleaned.write(line.strip('
')+","+3n"

        f.close()
        cleaned.close()
2. Take in the records used to D1 and D3 football teams and clean them

2.1 Clean the webpages from D3 schools

For D3 schools only we went to http://www.d3football.com/teams/index to get their win/loss records. For example http://www.d3football.com/teams/Adrian/2015/index provides the record for Adrian Bulldogs. There is a panel that gives past records. The shortname Adrian serves to link with the full university name.

We want data for years 2004-2014. Records on each page are organized in descending order from the latest year (=2015 for now). So we skip 2015 and take the next 10 lines from the win/loss records.

The cleanD3Records function does not have the years: these are added by running the function addYearsToD3 consecutively. For division 1 there is another column named bowl (for bowl games) but this does not apply to any of the D3 schools.

```python
In [7]: #clean D3 school files to get their win loss records
def cleanD3Records(outputFile):
    i = 0
    output = open(outputFile,'w')
    file1 = open("School Lists/Combined_original.txt","r") #open file containing school names

    output.write("School,win,loss\n")
    file1.readline()
    for line in file1:
        school = "Division 3 schools/"+line.split("",")[0].strip('\n')+ "\n" 
        record = open(school,'r')

        for lines in record:
            if lines.find('<a href="/conf/">' > 0 and i <= 11:  #find lines that have the wins
                record = lines.split("","")
                wins = record[0].split('-')[0]
                losses = record[0].split('-')[1]
                if i > 0: #exclude the first one since it is 2015
                    output.write(line.split("",")[0].strip('\n')+","+wins","+losses+\n')
                i = i +1

    output.close()

In [8]: cleanD3Records("D3 Win-Loss-noYear.csv")
```

```python
In [9]: def addYearsToD3(inFile,outputFile):
    inputFile = open(inFile,'r')
    output = open(outputFile,'w')
    i = 0
    output.write("year,"+inputFile.readline())
    school = ""
    for line in inputFile:
        if i > 10:  #reset counter
            i = 0
        if line.split('','')[0].find(school)<0:  #confirm that we have not gotten to a different s
            i = 0
```
year = str(2014-i) + ',' + line  
# creating the year column from 2014 (i=0) down to 2004 (i=10)
output.write(year)
school = line.split(',')[0]  
# save name of school just done
i = i+1

In [10]: addYearsToD3("D3 win-Loss-noYear.csv", "D3 Win-Loss.csv")

2.2 Clean the D1 csv files for the master file

Using the fact that all the schools we have came from sports reference we only need to adjust the name a bit to fit it into the file name and can use that to open all files that contain the win/loss record for a given year and if they got into a bowl for that year. You will notice that there are three separate if checks for special names. The name used in our list was a slightly longer and easier to match version, thus for those cases we had to open using the modified name of the sports reference given csv file name. From here we take the year (2004-2015). If the school does not have all of these years then we can eliminate it from the data set. We keep the shorthand version of the name from the list to match our master file. (We could potentially also save the id as that is also in the combined text file.) The amount of wins and losses is taken as a general way to see if they had a winning season which a threshold can later be determined through mining. The bowl column is to say whether or not the team entered a bowl; we have excluded if the team wins or loses the bowl since being in one can be assumed the more important fact. If the team did not make a bowl game in a given season then the column is left blank.

In [11]: def cleanD1Records(outputFile):
    
    output = open(outputFile,'w')
    file1 = open("School Lists/lcombined_original.txt","r")

    output.write("year,school,win,loss,bowl\n")
    file1.readline()
    for line in file1:
        i = 0
        fullSchool = line.split(',')[0]
        shortSchool = fullSchool
        # for some schools, the file name isn't the same as the shorthand
        if fullSchool.find("miami_florida")>= 0:
            shortSchool = "miami_fl"
        elif fullSchool.find("miami_ohio")>= 0:
            shortSchool = "miami_oh"
        elif fullSchool.find("texas_A&M")>= 0:
            shortSchool = "texas_am"

        fileName = "sport Reference xls/cfb_schools_"+shortSchool.replace("_-","_"+"_"+shortSchool.replace("","_")
        record = open(fileName,"r")

        record.readline()
        record.readline()

        for lines in record:
            linesList = lines.split(','
            year = linesList[1]
            wins = linesList[3]
            loses = linesList[4]
            if linesList[13] != "":
                bowl = 'Y'
            else:
                bowl = ""
            if year.find("2015") < 0:
                if i < 12:
output.write(year","+fullSchool","+wins","+loses","+bowl+
"
)
i = i + 1

In [12]: cleanDiRecords("D1 win-loss.csv")

3 3. Creating a master file by combining dataset FOR A SINGLE
YEAR (2014)

We have a schoolsListComplete that contains only NCAA schools, currently divisions 1 and 3 but not 2. It has: ID, School, Division. To this file, we append a SUBSET of other files on only matching school IDs. In all other files, the first column is the school ID on which to match.

In [13]: #for each school in the additional file with matching ID, add it to the data
# !!! we assume that the school IDs are SORTED for efficiency sake
# The exclusionList serves to DROP schools from that point onward because they did not return
# For example this applies to Tulane in year 2006 because they did not return student
# So the script INCLUDES Tulane in merge files UNTIL this university's ID is given in
def combine(original, addition, nameOfOutput, exclusionList=[]):
counting=0 # display progress of data processing every 30 schools (~10% of data)
inputFile1=open(original, 'r') # makes the left columns
inputFile2=open(addition, 'r') # makes the right columns
outputFile=open(nameOfOutput, 'w') # dumps results there
headersOriginal=inputFile1.readline()
headersAddition=inputFile2.readline()
headersAddition=headersAddition[headersAddition.index(',') + 1:] # removes the UNITID from the
outputFile.write(headersOriginal.strip('
') + "," + headersAddition + "
"
for line in inputFile1:
    # this is the SMALL file of select NCAA schools
    inputLine1=line.split(',')
    NCAAschoolCode=int(inputLine1[0])
    if NCAAschoolCode in exclusionList:
        continue # reads the next line and skips that school
    matchingCode=-1
    inputLine2=""
    # print NCAAschoolCode, 'attempted matching with...
    while matchingCode != NCAAschoolCode: # read extra lines from the big school list until
        inputLine2=inputFile2.readline()
        inputList2=inputLine2.split(',
"
        if len(inputList2[0]) == 0: # WE HAVE A BUG IN THE MATRIX
            print "Bug coming with school ", NCAAschoolCode, "and ethnicity file has line"
        if len(inputList2[0]) == 0:
            print "\t\t", inputList2
            matchingCode=int(inputList2[0])
# when we reach this line, we have the same school in both datasets
outputFile.write(line.strip('
') + "," + inputLine2[inputLine2.index(',')] + 1:]]
counting+=1
if counting%30==0:
    # print ",",
    inputFile1.close()
    inputFile2.close()
    outputFile.close()
    print ""

We create a separate script to deal with the ethnicity files for fall enrollment. Since each school has multiple records (see below) we add in a check to only get data from the line stating the total amount of
students enrolled. The line that we are interested in has the code ‘1’ to indicate total enrollment (see data
dictionary below).

In [14]: #When a file has a school on MULTIPLE rows, then the 2nd column indicates what property is being
#For example in ethnicity the same school is listed on multiple lines, and a ‘1’ in 2nd column

def shrinkData(original, value, nameOfOutput):
    inputFile = open(original, 'r')
    dataLines = [] #lines may NOT be ordered so we collect the ones we want when reading the file
    outputFile = open(nameOfOutput, 'w')
    outputFile.write(inputFile.readline())
    for line in inputFile:
        content = line.split(',')
        if int(content[1]) == value:
            dataLines[int(content[0])] = line #collect line
    #then we sort them
    import collections
    orderedData = collections.OrderedDict(sorted(dataLines.items()))
    for key in orderedData:
        outputFile.write(orderedData[key])
    outputFile.close()
    inputFile.close()

In [15]: #Shrinking all ethnicity files

shrinkData('IPEDS/2014/Fall Enrollment/Demographics/Ethnicities.csv', 1, 'IPEDS/2014/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2013/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2013/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2012/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2012/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2011/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2011/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2010/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2010/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2009/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2009/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2008/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2008/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2007/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2007/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2006/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2006/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2005/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2005/Fall Enrollment/Ethnicities1.csv')
shrinkData('IPEDS/2004/Fall Enrollment/Ethnicity/Ethnicity.csv', 1, 'IPEDS/2004/Fall Enrollment/Ethnicities1.csv')

In [18]: #shrinking headcounts for total number of students already in the school

shrinkData('IPEDS/2014/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2014/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2013/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2013/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2012/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2012/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2011/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2011/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2010/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2010/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2009/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2009/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2008/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2008/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2007/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2007/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2006/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2006/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2005/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2005/12-Month Enrollment/Headcount1.csv')
shrinkData('IPEDS/2004/12-Month Enrollment/Headcount/Headcount.csv', 1, 'IPEDS/2004/12-Month Enrollment/Headcount1.csv')

In [67]: #for each school in the additional file with matching ID, add it to the data
# !! we assume that the school IDs are SORTED for efficiency sake

def combineEthnicityTotal(original, addition, nameOfOutput):
    counting = 0 #display progress of data processing every 30 schools (=10% of data)
    inputFile1 = open(original, 'r') #makes the left columns
    inputFile2 = open(addition, 'r') #makes the right columns
    outputFile = open(nameOfOutput, 'w') #dumps results there
headersOriginal=inputFile1.readline()
headersAddition=inputFile2.readline()
headersAddition=headersAddition[headersAddition.index(',')+1:]
# removes the UNITID from the headers
outputFile.write(headersOriginal.strip('
')+'"'+headersAddition)+'
# headers created for the line in inputFile1:
# this is the SMALL file of select NCAA schools

for line in inputFile1:
    lineList=line.strip(','),
    NCASchoolCode=int(lineList[0])
    matchingCode=1
    lineList2=""
    # print NCASchoolCode, 'attempted matching with...'
    while matchingCode != NCASchoolCode:
        lineList2=lineList2.split(','),
        print "\t\t", lineList2
        if len(lineList2[0])==0:
            print 'WE HAVE A BUG IN THE MATRIX', NCASchoolCode, "and ethnicity file has line",
            matchingCode=int(lineList2[0])
        # when we reach this line, we have the same school in both datasets
        elif int(lineList2[1]) == 1:
            outputFile.write(lineList2[1]+'"
            counting=0
            # print ",",
    if counting==0:
        print "",
    inputFile1.close()
    inputFile2.close()
# print ""

In [19]:

In [20]:

In [21]:
In [22]: # get data for 2011
   combine('schoolListComplete.csv', 'IPEDS/2011/Institution Characteristics/University Affiliation.csv',
   '2011 merged/merge1.csv', 'IPEDS/2011/Institution Characteristics/University Information.csv',
   '2011 merged/merge2.csv', 'IPEDS/2011/Admissions Container/Admissions/admissions.csv',
   '2011 merged/merge3.csv', 'IPEDS/2011/Completions/Graduation demographics/Grad_Demographics.csv',
   '2011 merged/merge4.csv', 'IPEDS/2011/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2011 merged/merge6.csv', 'IPEDS/2011/Student Aid/Students/finances.csv',
   '2011 merged/merge7.csv', 'IPEDS/2011/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [23]: # get data for 2010
   combine('schoolListComplete.csv', 'IPEDS/2010/Institution Characteristics/University Affiliation.csv',
   '2010 merged/merge1.csv', 'IPEDS/2010/Institution Characteristics/University Information.csv',
   '2010 merged/merge2.csv', 'IPEDS/2010/Admissions Container/Admissions/admissions.csv',
   '2010 merged/merge3.csv', 'IPEDS/2010/Completions/Graduation demographics/Grad_Demographics.csv',
   '2010 merged/merge4.csv', 'IPEDS/2010/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2010 merged/merge6.csv', 'IPEDS/2010/Student Aid/Students/finances.csv',
   '2010 merged/merge7.csv', 'IPEDS/2010/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [24]: # get data for 2009
   combine('schoolListComplete.csv', 'IPEDS/2009/Institution Characteristics/University Affiliation.csv',
   '2009 merged/merge1.csv', 'IPEDS/2009/Institution Characteristics/University Information.csv',
   '2009 merged/merge2.csv', 'IPEDS/2009/Admissions Container/Admissions/admissions.csv',
   '2009 merged/merge3.csv', 'IPEDS/2009/Completions/Graduation demographics/Grad_Demographics.csv',
   '2009 merged/merge4.csv', 'IPEDS/2009/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2009 merged/merge6.csv', 'IPEDS/2009/Student Aid/Students/finances.csv',
   '2009 merged/merge7.csv', 'IPEDS/2009/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [25]: # get data for 2008
   combine('schoolListComplete.csv', 'IPEDS/2008/Institution Characteristics/University Affiliation.csv',
   '2008 merged/merge1.csv', 'IPEDS/2008/Institution Characteristics/University Information.csv',
   '2008 merged/merge2.csv', 'IPEDS/2008/Admissions Container/Admissions/admissions.csv',
   '2008 merged/merge3.csv', 'IPEDS/2008/Completions/Graduation demographics/Grad_Demographics.csv',
   '2008 merged/merge4.csv', 'IPEDS/2008/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2008 merged/merge6.csv', 'IPEDS/2008/Student Aid/Students/finances.csv',
   '2008 merged/merge7.csv', 'IPEDS/2008/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [26]: # get data for 2007
   combine('schoolListComplete.csv', 'IPEDS/2007/Institution Characteristics/University Affiliation.csv',
   '2007 merged/merge1.csv', 'IPEDS/2007/Institution Characteristics/University Information.csv',
   '2007 merged/merge2.csv', 'IPEDS/2007/Admissions Container/Admissions/admissions.csv',
   '2007 merged/merge3.csv', 'IPEDS/2007/Completions/Graduation demographics/Grad_Demographics.csv',
   '2007 merged/merge4.csv', 'IPEDS/2007/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2007 merged/merge6.csv', 'IPEDS/2007/Student Aid/Students/finances.csv',
   '2007 merged/merge7.csv', 'IPEDS/2007/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [27]: # get data for 2006
   combine('schoolListComplete.csv', 'IPEDS/2006/Institution Characteristics/University Affiliation.csv',
   '2006 merged/merge1.csv', 'IPEDS/2006/Institution Characteristics/University Information.csv',
   '2006 merged/merge2.csv', 'IPEDS/2006/Admissions Container/Admissions/admissions.csv',
   '2006 merged/merge3.csv', 'IPEDS/2006/Completions/Graduation demographics/Grad_Demographics.csv',
   '2006 merged/merge4.csv', 'IPEDS/2006/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
   '2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
   '2006 merged/merge6.csv', 'IPEDS/2006/Student Aid/Students/finances.csv',
   '2006 merged/merge7.csv', 'IPEDS/2006/12-Month Enrollment/Headcount/HeadcountReduced.csv',

9
combine('2006 merged/merge4.csv', 'IPEDS/2006/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
    '#WE EXCLUDE school 160755 (Tulane) and 212805 (Grove City) because they didn’t fill in student aid data
    combine('2006 merged/merge5.csv', 'IPEDS/2006/Student Aid/Students/finances.csv', '2006 merged/mea
    combine('2006 merged/merge6.csv', 'IPEDS/2006/12-Month Enrollment/Headcount/HeadcountReduced.csv',
    
In [28]: # get data for 2005
    combine('schoolListComplete.csv', 'IPEDS/2005/Institution Characteristics/University Affiliation
    combine('2005 merged/merge1.csv', 'IPEDS/2005/Institution Characteristics/University Information
    combine('2005 merged/merge2.csv', 'IPEDS/2005/Admissions Container/Admissions/admissions.csv',
    combine('2005 merged/merge3.csv', 'IPEDS/2005/Completions/Graduation demographics/Grad_Demographi
    #WE EXCLUDE school 160755 (Tulane) because they didn’t fill in student aid data
    combine('2005 merged/merge4.csv', 'IPEDS/2005/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
    #combine('2012 merged/merge5.csv', '2012/Fall Enrollment/retention_stud_to_fac/retention.csv',
    combine('2005 merged/merge5.csv', 'IPEDS/2005/Student Aid/Students/finances.csv', '2005 merged/mea
    combine('2005 merged/merge6.csv', 'IPEDS/2005/12-Month Enrollment/Headcount/HeadcountReduced.csv',

In [29]: # get data for 2004
    combine('schoolListComplete.csv', 'IPEDS/2004/Institution Characteristics/University Affiliation
    combine('2004 merged/merge1.csv', 'IPEDS/2004/Institution Characteristics/University Information
    combine('2004 merged/merge2.csv', 'IPEDS/2004/Admissions Container/Admissions/admissions.csv',
    combine('2004 merged/merge3.csv', 'IPEDS/2004/Completions/Graduation demographics/Grad_Demographi
    combine('2004 merged/merge4.csv', 'IPEDS/2004/Fall Enrollment/Ethnicity/EthnicitiesReduced.csv',
    #WE EXCLUDE school 212805 (Grove City) because they didn’t fill in student aid data
    combine('2004 merged/merge5.csv', 'IPEDS/2004/Student Aid/Students/finances.csv', '2004 merged/mea
    combine('2004 merged/merge6.csv', 'IPEDS/2004/12-Month Enrollment/Headcount/HeadcountReduced.csv',

Add year to first column of data

In [30]: def addYearInFront(filename, nameOfOutput, year):
    f = open(filename, 'r')
    output = open(nameOfOutput, 'w')
    output.write('Year,' + f.readline())
    for line in f:
        output.write(year + "", + line)
    f.close()
    output.close()

In [31]: addYearInFront('2014 merged/merge7.csv', '2014 merged/complete-2014.csv', '2014')
    addYearInFront('2013 merged/merge7.csv', '2013 merged/complete-2013.csv', '2013')
    addYearInFront('2012 merged/merge7.csv', '2012 merged/complete-2012.csv', '2012')
    addYearInFront('2011 merged/merge7.csv', '2011 merged/complete-2011.csv', '2011')
    addYearInFront('2010 merged/merge7.csv', '2010 merged/complete-2010.csv', '2010')
    addYearInFront('2009 merged/merge7.csv', '2009 merged/complete-2009.csv', '2009')
    addYearInFront('2008 merged/merge7.csv', '2008 merged/complete-2008.csv', '2008')
    addYearInFront('2007 merged/merge7.csv', '2007 merged/complete-2007.csv', '2007')
    addYearInFront('2006 merged/merge7.csv', '2006 merged/complete-2006.csv', '2006')
    addYearInFront('2005 merged/merge7.csv', '2005 merged/complete-2005.csv', '2005')
    addYearInFront('2004 merged/merge7.csv', '2004 merged/complete-2004.csv', '2004')

3.1 Combine cleaned football records with sets

We have the merged files gathering all relevant IPEDS data, and we seek to combine that with the win/loss records of the teams.

In [32]: # we have ‘D1 win-loss.csv’ and ‘D3 win-loss.csv’, we seek to combine them
    # no need to add the division column because the IPEDS data with which we’ll combine it already

10
def mergeAthletics(division1, division3, outputName):
    import collections
    f1 = open(division1, 'r')
    f2 = open(division3, 'r')
    output = open(outputName, 'w')
    datalines = {}  
    output.write(f1.readline())  # it has one extra column for Bawls
    f2.readline()  
    for line in f1:
        content = line.split(',', )
        datalines[content[1].lower()] + content[0] = line.lower()  # collect line
    for line in f2:
        content = line.split(',', )
        datalines[content[1].lower()] + content[0] = line.lower()  # collect line
    orderedData = collections.OrderedDict(sorted(datalines.items()))
    for key in orderedData:
        output.write(orderedData[key])
    output.close()
    f1.close()
    f2.close()

In [33]: mergeAthletics('D1 win-loss.csv', 'D3 win-loss.csv', 'winLoss.csv')

In [34]: # the athletics file is in the same folder as winLoss.csv
    # the IPEDS file are within their respective folders as 2004 merged/complete-2004 etc.
    def mergeIPEDsWithAthletics(athletics, IPEDRecord, outputFile, exclusionList=[]):
        schoolsUsed = {}
        year = int(IPEDRecord(IPEDRecord.index('-' )) + 1:IPEDRecord.index('.' ))
        winLoss = open(athletics, 'r')
        IPED = open(IPEDRecord, 'r')
        output = open(outputFile, 'w')
        output.write(IPED.readline().strip('
')) + win, loss, bowl
        IPED.close()
        winLoss.readline()  # don't care about the headers because we manually added them to the results
        for line in winLoss:
            content = line.split(',', )
            if int(content[0]) != year:  # only add the win loss record for the year we care about
                continue
            schoolName = content[1]
            if schoolName in exclusionList:  # schools that don't exist in the merged files (but have
                continue
            matchingLine = getLineForName(IPEDRecord, schoolName)
            schoolsUsed[schoolName] = ''
            win = content[2]
            loss = content[3].strip('
')
            if len(content[4]) == 4 or content[4] == "":
                bowl = 'n'
            else:
                bowl = content[4].strip('
')
            if bowl != 'y':
                bowl = 'n'
            else:
                winLoss.write(matchingLine.strip('
')) + win, loss, bowl
        winLoss.close()
        output.close()
        excluded = findUnusedNames(IPEDRecord, schoolsUsed)

11
import os, sys
filename=outputFile[:outputFile.index('.')]+(exclude '+str(excluded)+').csv
os.rename(outputFile, filename)

def findUnusedNames(ProbRecord, schoolsUsed):
    f=open(ProbRecord)
    f.readline()
    missing={}
    for line in f:
        content=line.split(',
        schoolName=content[2].lower()
        if schoolName not in schoolsUsed:
            missing[schoolName]=''
    f.close()
    #print 'The following ',len(missing),'schools have no record for this year:'
    #for key in missing:
    #    print key
    return len(missing)

#For a given school name (e.g. adrian), fetch the IPED line that matches it (based on
#column 2)
def getLineByName(ProbRecord,name):
    f=open(ProbRecord,'r')
    f.readline()
    for line in f:
        content=line.split(',
        schoolName=content[2].lower()
        if schoolName==name:
            return line
    f.close()
    print 'No match found for ',name
    return None

In [38]: mergeIPEDwithAthletics('winLoss.csv','2005 merged/complete-2005.csv','2005 merged/2005-FINAL.csv'
In [39]: mergeIPEDwithAthletics('winLoss.csv','2006 merged/complete-2006.csv','2006 merged/2006-FINAL.csv'
mergeIPEDwithAthletics('winLoss.csv','2008 merged/complete-2008.csv','2008 merged/2008-FINAL.csv'
mergeIPEDwithAthletics('winLoss.csv','2010 merged/complete-2010.csv','2010 merged/2010-FINAL.csv'
mergeIPEDwithAthletics('winLoss.csv','2012 merged/complete-2012.csv','2012 merged/2012-FINAL.csv'
In [41]: mergeIPEDwithAthletics('winLoss.csv','2013 merged/complete-2013.csv','2013 merged/2013-FINAL.csv'
mergeIPEDwithAthletics('winLoss.csv','2014 merged/complete-2014.csv','2014 merged/2014-FINAL.csv'

3.2 Merge Year files

In [42]: def addDifferentYears(file1,file2,nameOfOutput):
    f1=open(file1,'r')
    f2=open(file2,'r')
    output=open(nameOfOutput,'w')
    output.write(f1.readline())
for line in f1:
    output.write(line)
f2.readline()
for line in f2:
    output.write(line)
f1.close()
f2.close()
output.close()
Appendix 2 Prepping Data for Weka

May 13, 2016

1 1) Get Standard Deviation for Applications

For us to give WEKA a CSV file to analyze it must have equal column length among all data sets. The problem with our data is that the IPEDS survey changes how they format their data each year. Thus the amount of columns changes each years along with the header names. Therefore by comparing master data with the year data we can locate the admissions data within the master file and pull the data into dictionaries. However some years the schools do not answer the survey each year. Therefore we track which schools did not answer in which year.

In [3]: #get total amount of applications

```python
def applicationAmounts(appDict, IDList):
    from csv import reader
    missingDict = {}
    missingList1 = []
    missingList2 = []
    missingList3 = []
    missingList4 = []
    missingList5 = []
    missingList6 = []
    missingList7 = []
    missingList8 = []
    missingList9 = []
    missingList10 = []
    numMissing = 0
    firstLine = True
    f = open("Master CSV Final/merge-final.csv")

    for line in reader(f):
        if firstLine:
            firstLine = False
            continue

        elif int(line[0]) == 2004:
            if line[262] == "":
                #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
                numMissing += 1
                missingList1.append(line[1])
            else:
                appDict[line[1]].append(int(line[262])+int(line[264]))

        if int(line[0]) == 2005:
```
if line[254] in ".":
    #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
    numMissing += 1
    missingList2.append(line[1])
else:
    #print line[254], line[256], line[2]
    if line[256] == "":
        appDict[line[1]].append(int(line[254]))
    else:
        appDict[line[1]].append(int(line[254])+int(line[256]))

if int(line[0]) == 2006:
    if line[259] in ".":
        #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList3.append(line[1])
    else:
        #print line[257], line[259], line[2]
        appDict[line[1]].append(int(line[257])+int(line[259]))

if int(line[0]) == 2007:
    if line[261] in ".":
        #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList4.append(line[1])
    else:
        #print line[261], line[263], line[2]
        appDict[line[1]].append(int(line[261])+int(line[263]))

if int(line[0]) == 2008:
    if line[275] in ".":
        #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList5.append(line[1])
    else:
        #print line[275], line[277], line[2]
        appDict[line[1]].append(int(line[275])+int(line[277]))

if int(line[0]) == 2009:
    if line[273] in ".":
        #print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList6.append(line[1])
    else:
```python
# print line[273], line[275], line[2]
appDict[line[1]].append(int(line[273])+int(line[275]))

if int(line[0]) == 2010:
    if line[275] in ".":
        # print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList7.append(line[1])

else:
    # print line[275], line[277], line[2]
    appDict[line[1]].append(int(line[275])+int(line[277]))

if int(line[0]) == 2011:
    if line[276] in ".":
        # print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList8.append(line[1])

else:
    # print line[276], line[278], line[2]
    appDict[line[1]].append(int(line[276])+int(line[278]))

if int(line[0]) == 2012:
    if line[279] in "A" or line[279] in ".":
        # print "No data for: " + line[2] + " Year: " + line[0] + line[3]
        numMissing += 1
        missingList9.append(line[1])

else:
    # print line[279], line[281], line[2]
    appDict[line[1]].append(int(line[279])+int(line[281]))

if int(line[0]) == 2013:
    if line[282] in "A" or line[282] in ".":
        # print "No data for: " + line[2] + "YEAR: " + line[0] + line[3]
        numMissing += 1
        missingList10.append(line[1])

else:
    # print line[282], line[284], line[2]
    appDict[line[1]].append(int(line[282])+int(line[284]))

if int(line[0]) == 2014:
    appDict[line[1]] = [int(line[188])]
    IDList.append(line[1])
```

3
```python
print numMissing
missingDict[‘2004’] = missingList1
missingDict[‘2005’] = missingList2
missingDict[‘2006’] = missingList3
missingDict[‘2007’] = missingList4
missingDict[‘2008’] = missingList5
missingDict[‘2009’] = missingList6
missingDict[‘2010’] = missingList7
missingDict[‘2011’] = missingList8
missingDict[‘2012’] = missingList9
missingDict[‘2013’] = missingList10
return missingDict

In [4]: appTotalsDict = {}
listIDs = []
missingDataDict = {}
missingDataDict = applicationAmounts(appTotalsDict, listIDs)

60

In [5]: import pandas as pd
import numpy as np

In [10]: # Create a dictionary that can be plotted in a histogram
plotDict = {}
for key in missingDataDict:
    plotDict[key] = len(missingDataDict[key])


Here we show how many schools are missing data per year

In [11]: %matplotlib inline
import matplotlib.pyplot as plt
X = np.arange(len(missingDataDict))
plt.bar(X, plotDict.values(), align=’center’, width=0.5)
plt.xticks(X, plotDict.keys())
ymax = max(plotDict.values()) + 1
plt.ylim(0, ymax)
plt.show()
```
2 2) Total Headcount of a University

When considering a school's standing there are other factors that also much be taken into account. One such factor is the size of the school. We will take this into account by taking the total number of students currently enrolled in a given school. By this we mean to include undergraduates and graduates into our count.

In [3]: def EnrolledCount(totalDict):
    from csv import reader

    firstLine = True
    f = open("Master CSV Final/merge-final.csv")

    for line in reader(f):
        if firstLine:
            firstLine = False
            continue

        if int(line[0]) == 2014:
            totalDict[line[1]] = []
            totalDict[line[1]].append(line[999])

        if int(line[0]) == 2013:
            totalDict[line[1]].append(line[1198])

        if int(line[0]) == 2012:
            totalDict[line[1]].append(line[1196])

        if int(line[0]) == 2011:
            totalDict[line[1]].append(line[1191])
if int(line[0]) == 2010:
    totalDict[line[1]].append(line[100])

if int(line[0]) == 2009:
    totalDict[line[1]].append(line[1046])

if int(line[0]) == 2008:
    totalDict[line[1]].append(line[807])

if int(line[0]) == 2007:
    totalDict[line[1]].append(line[600])

if int(line[0]) == 2006:
    totalDict[line[1]].append(line[597])

if int(line[0]) == 2005:
    totalDict[line[1]].append(line[590])

if int(line[0]) == 2004:
    totalDict[line[1]].append(line[600])

f.close()

In [4]: totalDict = {}
   #totalDict will have 2014 in index 0
   EnrolledCount(totalDict)

3 3) Generate Standard deviation of amount of Applications for each School

Next we want to determine the standard deviation for each time period. We want to find the impact on the three years following the given year. Thus if we are studying 2004 we go up to 2007. We compute the standard deviation of the number of applications for the time period. The deviation is then stored in a dictionary for that year. For example we calculate the deviation in the number of applications from 2004 to 2007 we only look at those years.

In [5]: def computeDeviadEngt(idList, appDict, sDDict, num):
    for UID in idList:
        summation = 0
        summ = 0
        count = 0
        if len(appDict[UID]) == 11:
            #mean = sum(appDict[UID])/len(appDict[UID])
            for x in range(len(appDict[UID])-num-1,len(appDict[UID]) - 1):
                # for x in range(len(appDict[UID])-num - 1,len(appDict[UID]) ):
                count += 1
                summ += appDict[UID][x]
            print count
            mean = summ/count
            for x in range(len(appDict[UID])-num-1,len(appDict[UID]) -1):#len(appDict[UID]) - num
                summation += (appDict[UID][x] - mean) ** 2
summation = summation/count

standardDeviation = summation ** .5

sdDict[UID] = standardDeviation

In [8]:
#deviationDict04 = {}
#computeDeviation(listIDs, appTotalsDict, deviationDict04, 0)

#deviationDict05 = {}
#computeDeviation(listIDs, appTotalsDict, deviationDict05, 1)

#deviationDict06 = {}
#computeDeviation(listIDs, appTotalsDict, deviationDict06, 2)

In [9]:
deviationDict07 = {}
computeDeviation(listIDs, appTotalsDict, deviationDict07, 3)

deviationDict08 = {}
computeDeviation(listIDs, appTotalsDict, deviationDict08, 4)

deviationDict09 = {}
computeDeviation(listIDs, appTotalsDict, deviationDict09, 5)

deviationDict10 = {}
computeDeviation(listIDs, appTotalsDict, deviationDict10, 6)

deviationDict11 = {}
computeDeviation(listIDs, appTotalsDict, deviationDict11, 7)

Here we compare the change in the number of applications over a period of 3 years. We do this by finding the difference between the number of applications in the year we are interested in and the three following years. We then compare that difference with the standard deviation. We then will state whether that difference is, more than the standard deviation, less than the negative deviation, or between the two in which it is considered the same.

In [288]:
#This code works for the years from 2004 to 2011
def measureDeviation(idList, appDict, sdDict, diffDict, num, missingDict, year):
    num1 = 0

    for UID in idList:
        diffDict[UID] = []
        #print len(appDict[UID]), len(appDict[UID]) - 4

        if len(appDict[UID]) == 11:
            for x in range(len(appDict[UID]) - num - 1, len(appDict[UID]) - num - 4, -1):
                #print appDict[UID][len(appDict[UID])]
                    if (appDict[UID][x] - appDict[UID][len(appDict[UID]) - num - 1]) > sdDict[UID]:
                        diffDict[UID].append("MORE")

                        #elif (appDict[UID][x] - appDict[UID][len(appDict[UID]) - num - 1]) < -sdDict
                        #    diffDict[UID].append("LESS")
else:
    diffDict[UID].append("SAME")

else:
    #print "Lacking data for " + UID
    num1 += 1

print num1

In [289]: Dict2007SD = {}

measureDeviation(listIDs, appTotalsDict, deviationDict07, Dict2007SD, 3, missingDataDict, 2007)

Dict2008SD = {}

measureDeviation(listIDs, appTotalsDict, deviationDict08, Dict2008SD, 4, missingDataDict, 2008)

Dict2009SD = {}

measureDeviation(listIDs, appTotalsDict, deviationDict09, Dict2009SD, 5, missingDataDict, 2009)

Dict2010SD = {}

measureDeviation(listIDs, appTotalsDict, deviationDict10, Dict2010SD, 6, missingDataDict, 2010)

Dict2011SD = {}

measureDeviation(listIDs, appTotalsDict, deviationDict11, Dict2011SD, 7, missingDataDict, 2011)

47
47
47
47
47

4) Get data on the win loss records of schools as well as if they played in a bowl

Here we get the athletic record for a team for a given year. We take the data and put it into a dictionary for later use.

In [290]: def getRecords(year, fileName):
    from csv import reader
    firstLine = True
    f = open(fileName)
    recDict = {}
    for line in reader(f):
        if firstLine:
            firstLine = False
            continue

        recDict[line[1]] = []

        if int(line[0]) == year:
if 'y' in line:
    bowlCol = line.index('y')

elif 'n' in line:
    bowlCol = line.index('n')

recDict[line[i]].append(line[bowlCol]) #append bowl
recDict[line[i]].append(line[bowlCol - 1])#append loss
recDict[line[i]].append(line[bowlCol - 2])#append win

f.close()
return recDict

In [291]: recordDict11 = getRecords(2011,"Master CSV Final/2011readable.csv")
In [292]: recordDict10 = getRecords( 2010,"Master CSV Final/2010readable.csv")
In [293]: recordDict09 = getRecords( 2009,"Master CSV Final/2009readable.csv")
In [294]: recordDict08 = getRecords( 2008,"Master CSV Final/2008readable.csv")
In [296]: recordDict06 = getRecords( 2006,"Master CSV Final/2006readable.csv")
In [297]: recordDict05 = getRecords( 2005,"Master CSV Final/2005readable.csv")
In [298]: recordDict04 = getRecords( 2004,"Master CSV Final/merge-FINAL.csv")

5 Get division for file

In [299]: def getDiv(divDict):
    from csv import reader
    firstline = True

    f = open("Master CSV Final/merge-final.csv")

    for line in reader(f):
        if firstline:
            firstline = False
            continue

        divDict[line[i]] = line[3]
        if int(line[0]) < 2014:
            break

In [300]: divisionDict = {}
    getDiv(divisionDict)

In [301]: def removeEmptyDicts(dictionary, listIDs):
    for UID in listIDs:
        if UID in dictionary:
            if len(dictionary[UID]) == 0:
                del dictionary[UID]
6 5) Construct CSV file for WEKA to analyze

```python
In [304]: # recDict is in format of bowl,loss,win as a dictionary of a list
   # sdDict has format of year+1,year+2,year+3 category
   def makeCSV(IDList, sdDict, recDict, outputName, year, divDict, totalDict):
       CSV = open(outputName,'w')

       CSV.write("Division,NumStudents,wins,losses,bowl,SD" + str(year+1) + ",SD" + str(year+2)+"

       for UID in IDList:
           if UID in recDict and UID in sdDict:
               if len(recDict[UID]) == 3 and len(sdDict[UID]) == 3:

       CSV.close()
```

```python
In [306]: makeCSV(listIDs, Dict2007SD, recordDict07, "WEKA Test Files/Number of applications received/2007")
   makeCSV(listIDs, Dict2008SD, recordDict08, "WEKA Test Files/Number of applications received/2008")
   makeCSV(listIDs, Dict2009SD, recordDict09, "WEKA Test Files/Number of applications received/2009")
   makeCSV(listIDs, Dict2010SD, recordDict10, "WEKA Test Files/Number of applications received/2010")
   makeCSV(listIDs, Dict2011SD, recordDict11, "WEKA Test Files/Number of applications received/2011")
```

7 6) Equalize Class amounts

```python
In [307]: # 0 represents same, 1 represents more
   def makeGraphable(inFile, outFile):
       inputFile = open(inFile)
       output = open(outFile,'w')
       codeY1 = 'Q'
       codeY2 = 'Q'
       codeY3 = 'Q'
       output.write(inputFile.readline().strip('n')+ ',' + "SDYear+1"+','+'SDYear+2"+','+'SDYear+3"

       for line in inputFile:
           row = line.split(',
           if row[5] == "MORE":
               codeY1 = '1'
           else:
               codeY1 = '0'

           if row[6] == "MORE":
               codeY2 = '1'
           else:
               codeY2 = '0'

           row[7] = row[7].strip('n')
```

10
if row[7] == "MORE":
    codeY3 = '1'
else:
    codeY3 = '0'

output.write(line.strip(\'
')+',','+codeY1+',','+codeY2+',','+codeY3+\'
')

In [309]: makeGraphable("WEKA Test Files/Number of applications received/2007.csv","WEKA Test Files/2007.csv")
makeGraphable("WEKA Test Files/Number of applications received/2008.csv","WEKA Test Files/2008.csv")
makeGraphable("WEKA Test Files/Number of applications received/2009.csv","WEKA Test Files/2009.csv")
makeGraphable("WEKA Test Files/Number of applications received/2010.csv","WEKA Test Files/2010.csv")
makeGraphable("WEKA Test Files/Number of applications received/2011.csv","WEKA Test Files/2011.csv")

In [311]: import pandas as pd
import numpy as np

In [38]: f = pd.read_csv("WEKA Test Files/2007Graphable.csv")
g = f
%matplotlib inline
f['SDYear+1'].plot(kind='hist',bins=3)
f['SDYear+2'].plot(kind='hist',bins = 5)
g['SDYear+3'].plot(kind='hist')

#use a map to turn your categorical data into numerical (1, 2, 3...) and plot 3 bins
print abs(1-112.0/187)+abs(1-97.0/202)+abs(1-68.0/231)

1.62649920454

In [318]: f = pd.read_csv("WEKA Test Files/2008Graphable.csv")
g = f
def f.describe()
%matplotlib inline
f['SDYear+1'].plot(kind='hist',bins=3)
f['SDYear+2'].plot(kind='hist',bins = 5)
g['SDYear+3'].plot(kind='hist')

print abs(1-177.0/121)+abs(1-118.0/180)+abs(1-94.0/205)

1.34871777643

In [319]: f = pd.read_csv("WEKA Test Files/2009Graphable.csv")
g = f
f.describe()

%matplotlib inline
f['SDYear+1'].plot(kind='hist',bins=3)
f['SDYear+2'].plot(kind='hist',bins = 5)
g['SDYear+3'].plot(kind='hist')
print abs(1-144.0/155)+abs(1-102.0/197)+abs(1-97.0/202)

1.07300322467
In [320]: f = pd.read_csv("WEKA Test Files/2010Graphable.csv")
    g = f
    f.describe()

%matplotlib inline
f['SDYear+1'].plot(kind='hist', bins=3)
f['SDYear+2'].plot(kind='hist', bins = 5)
g['SDYear+3'].plot(kind='hist')

print abs(1-160.0/139)+abs(1-130.0/169)+abs(1-109.0/190)

0.808164156934
In [321]: f = pd.read_csv("WEKA Test Files/2011Graphable.csv")
    g = f
    f.describe()

%matplotlib inline
f['SDYear+1'].plot(kind='hist',bins=3)
f['SDYear+2'].plot(kind='hist',bins = 5)
g['SDYear+3'].plot(kind='hist')

print abs(1-183.0/116)+abs(1-140.0/159)+abs(1-130.0/169)

0.927852293012
Appendix 3) Get win-loss record for D3 schools

```python
# coding: utf-8
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium.common.exceptions import NoAlertPresentException
import unittest, time, re
import lxml.html.clean as clean

class Div3Webpages(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "http://www.d3football.com/
        self.verificationErrors = []
        self.accept_next_alert = True

    def getPages(self, school):
        driver = self.driver
        driver.get(self.base_url + "teams/" + school + "/2015/index")
        content = driver.page_source
        cleaner = clean.Cleaner()
        content = cleaner.clean_html(content)
        with open(school + ".txt", "w") as f:
            f.write(content.encode("utf-8"))

    def test_div3_webpages(self):
        f = open("Schools.txt", 'r')
        for school in f:
            school = school.strip('"
        self.getPage(school)

    def is_element_present(self, how, what):
        try:
            self.driver.find_element(by=how, value=what)
        except NoSuchElementException as e:
            return False
        return True

    def is_alert_present(self):
        try:
            self.driver.switch_to_alert()
        except NoAlertPresentException as e:
            return False
        return True

    def close_alert_and_get_its_text(self):
        try:
            alert = self.driver.switch_to_alert()
            alert_text = alert.text
            if self.accept_next_alert:
                alert.accept()
            else:
                alert.dismiss()
            return alert_text
        finally:
            self.accept_next_alert = True

    def tearDown(self):
        self.driver.quit()
        self.assertEqual([], self.verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

Appendix 4) Link IPED ID’s to school names

```python
# -*- coding: utf-8 -*-

Created on Wed Feb 24 11:25:06 2016
Get ids from the IPED that have universites with football
team part of the NCAA. This includes the schools for division 1 2 and 3
Author: eic

import csv

with open('Affiliations.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        content = list(row)
        if content[54] == '1': # content[54] holds the value of sport1. if 1 then NCAA football team.
            with open('ids.txt', 'a') as g:
                print(content[0])
                g.write(content[0] + '
')
```
References


Freshmen Application Pool at NCAA Division II Universities.” *Journal of Issues in Intercollegiate Athletics* 4 (2011) 411-427


