

An Artificial Neural Network Approach to College Football Prediction and Ranking

Michael Pardee
ECE 539 Final Project
12/23/1999

Table of Contents

(click on a page number to the right of a section to automatically jump to that section)

INTRODUCTION.....	3
WORK PERFORMED.....	4
DATA COLLECTION	4
DATA SETUP FOR THE NEURAL NETWORK	5
NEURAL NETWORK DESIGN	5
SELECTION OF NETWORK PARAMETERS	6
NETWORK USAGE	7
RESULTS	8
PREDICTION RATES	8
POST-SEASON WINNERS.....	9
BASELINE STUDY	10
CONCLUSION.....	11
REFERENCES.....	12
APPENDIX A: CODE.....	12
PP.M.....	12
TEAMIFYV.M	18
PLAYOFF.M	19
MAKE_CUMULATIVE_DATA.M	21
LOAD_ALL_TEAMS_DATA.M.....	22
LOADFILES.M	23
CONVDIR.PL	24
PARSESTAT.PL	24
TXT2MAT.PL	26
APPENDIX B: ADDITIONAL FILES	27
GAME11	27
GAME11.TXT	27
GAME11.HTML	28
MASSEY INDEX WEEKLY RANKINGS FOR 1999.....	30
1999 SEASON SCHEDULE/ MIS-PREDICTIONS OF MASSEY INDEX.....	30

Introduction

At the end of each college football season, fans always argue over who is the best team in the country because Division I-A college football is the only major college sport that does not determine a champion by a post-season playoff. The bowl games that some of the better teams play in at the end of the year do not always match up the best teams (like Wisconsin vs. Stanford this year in the Rose Bowl), and fans are always left wondering who should be the national champion.

In recent years, the TV network ABC has tried to remedy the situation by scheming up a complicated formula that they use to determine who the best two teams in the country are, and then letting those two teams play to determine a national champion. This formula, which they call the [BCS](#) (Bowl Championship Series) has four components: **Poll** results, **Computer Rankings**, **Strength of Schedule**, and how many **Losses** a team has. The **Polls** are conducted by sportswriters from around the nation, but humans hold certain biases and don't always look at the information subjectively. The **Computer Rankings** are determined by 8 separate computer ranking schemes devised by numerous private entities across the country. To my knowledge, none of those 8 make the source code publicly available, and only a handful of them make their algorithms known. A least squares model is common, but the weights on the inputs are often determined by what the originator of that method deems "fair." The **Strength of Schedule** component is weighted two-thirds (66 2/3%) for the opponents record and one-third (33 1/3%) for the opponent's opponents record. While the number of **Losses** a team might seem like an unbiased statistic, this component is weighted in the overall formula with an arbitrary weight.

The main problem with the [BCS](#) is the fact that the weights of the input are determined arbitrarily by humans who may hold biases towards certain teams or style of play. To remedy the situation, I propose a ranking system that uses an artificial neural network to determine the weights to be applied to input based solely on past game results. To determine which teams are the best, I develop a predictor model based on the data from past seasons, and then I am determine who is better than whom by predicting who would win if the teams played playoff games after the regular season.

To determine the effectiveness of my algorithm, I can compare it's prediction capability to the prediction capability of a ranking system assuming that the higher ranked team will always win. One might say that a system that is solely meant to predict game outcomes cannot be compared to a ranking system, because in certain instances it is impossible for the ranking system's "predictions" to be consistent. Most web sites that post computer rankings explicitly state that the rankings are not meant to predict who will win if the teams played each other. On the other hand, I say that rankings SHOULD be able to predict the winner with a *reasonable* degree of certainty, and here's why—If team A is ranked #1 and team B is ranked #7, and team B wins when they play each other, I would say that that ranking system was faulty. Of course we must consider average performance, given the extreme randomness of football games. But if a certain prediction system could predict team B beating team A on a consistent basis, and it predicts that no other team beats team B on a consistent basis, then team A should not be ranked #1. Instead, team B should be declared the national champion. A ranking system should be reasonably good at predicting the winner, otherwise its rankings would never correlate with the results of a post-season playoff if one occurred. Ultimately, determining the national champion without an actual playoff becomes a sequence of "what if team x beat team y?" scenarios, and

the prediction part of my method is capable of determining what *should* happen in those situations.

Due to limited resources, limited data, and limited time (albeit a lot of time) being allocated to this project, I decided to tackle the problem of prediction and ranking considering only the Big Ten football conference, composed of 11 Midwestern football teams (including Wisconsin of course). I did this because I was able to get detailed game statistics for all 121+ games played by big ten teams for the 1998 and 1999 seasons from the [Big Ten web-site](#). To develop my neural network model, I needed a substantial amount of data. While most ranking methods only use a few inputs like score, strength of schedule, etc., they have historical data going back over 100 years to look at. I could not find data (without paying \$\$ for it) that included a sufficient number of years for all college football teams. Instead I chose to compensate for only having 2 years of Big Ten data to work with by including 22 different statistical categories for each game. I was then able to train my network on the 1998 season, and test on the 1999 season with reasonable results (or train on 1999 and test on 1998).

Work Performed

Data Collection

Once I found enough data to use on the Big Ten web-site, I had to download an html file for each of the 242+ games. I realized that in the Big Ten portion of the season, most of the teams play each other, so there are really more like 140+ distinct games. However, I decided that the only easy way to keep games for each team in the correct order would be keep all 11 games for each team in its own directory. In addition, by keeping two accounts, I can train my neural network with the two “different” scenarios—Team A plays Team B, and Team B plays Team A. There is a possible difference between those two scenarios in my approach, because the order of the teams input changes the inputs to the neural network. But ideally, my neural network should predict the exact same outcome for both scenarios. So, pretending there are 242+ games instead of 140+ helps to reduce the dependence on the outcome on the order the teams are listed in.

To simplify things, I only used 11 games for each team, even if a team played more (some teams played 4 non-conference games for a total of 12 games). If 12 games were played instead of 11, I did not include the first game of the season (always non-conference). I also did not include bowl game results from the end of the 1998 season since they were all against non-conference opponents and probably could not be used effectively.

I was unable to automate the process of downloading the html files for each game, because when you download the games from the [Big Ten web-site](#) with an automated web-downloader (like the scheduler in IE5 or third party programs like *Web Stripper*) the Java scripting on the web-site confuses the programs. So instead I had to download each file individually by hand. For each team, I put all of its 11 html files in its own directory, with standardized names like “Wisconsin” and “Michigan St.”

Then, I was able to write a perl script, [parsestat.pl](#), to convert the html file containing the entire description of the game into just a list of the relevant statistics. I have included an example of one of the html files ([game11.html](#)), and the output created from it ([game11.txt](#)). After doing that, I found that Matlab had difficulty reading in that format, so I made another script, [txt2mat.pl](#), to do some more processing on the text. After running `txt2mat.pl` on

game11.txt, it became [game11](#). Another problem I had was that when I downloaded the html files, there was not a consistent naming scheme to them. So I created another perl script, [convdir.pl](#), that would take an entire directory of html files for a team, sort it by file creation time (I had saved all teams game files in order of first to last), rename the files to a standard naming convention (game1.html, game2.html, etc...), run parsestat.pl on each file, and then run txt2mat.pl on each file.

To import the data from each team's directory into Matlab, I created [load_all_teams_data.m](#) to go through the directory of each team. Load_all_teams_data runs convdir.pl and [loadfiles.m](#) (which creates a big matrix of all the data for one team) for each team, and finally combines all of the teams data into one big matrix. Because I could not use 3-dimensional arrays in Matlab, I had to use 2 dimensional arrays, and use the reshape command extensively to handle to 3-D nature of my data (11 teams by 11 games by 22 stats).

Data Setup for the Neural Network

After running that set of programs to create suitable matrices for the 98 and 99 seasons, I saved them into the files [all_teams_data_98](#) and [all_teams_data_99](#). Then I ran a program I called [make_cumulative_data.m](#) to change the week by week statistics into a cumulative format, where each row for a team corresponds to all of the team's cumulative data up until that week. (cumulative data is **average stats** through that week) Each row of the cumulative data file has a teams cumulative statistics, the cumulative statistics of their opponents, their opponent's cumulative statistics, and their opponent's cumulative opponents statistics. (Note that the cumulative statistics of their opponents is not the same as their opponent's cumulative statistics.) After compiling all of the cumulative data, I only saved the cumulative data starting after the first Big Ten games (cumulative data includes all data from all prior weeks, so non-conference game stats ARE included), so there are a total of 88 columns in each cumulative data file. Each row of the cumulative data file specifies cumulative stats **up to, but not including the current game**. The last entry in each row of the cumulative data file specifies which team won the current game. So this cumulative file can be used to train a neural network given past game statistics and current game outcome. I saved the cumulative files [bt_98](#) and [bt_99](#) representing the 98 and 99 Big Ten cumulative data.

Later in the project, I needed to conduct a round robin tournament for all teams, so I made the data setup file [playoff.m](#) which takes a given week and sets up the data for a tournament if it were played that week (i.e. specifying week 12 sets up a tournament using all 11 games played during the regular season; specifying week 6 would conduct the tournament based on cumulative data through 5 games)

Neural Network Design

Over the course of the ECE 539 semester, I had had exposure to many artificial neural network learning algorithms, including simple perceptron learning, back-propagation perceptron learning, auto-regressive models, and time-delayed neural networks. Based on the results I saw during the semester for the various learning experiments conducted, I determined that back-propagation perceptron learning would work the best for my application. I used a multi-layer perceptron neural network back-propagation algorithm with one hidden layer implemented in my Matlab file [pp.m](#) (Pardee Predictor). It is based on Professor Yu Hen Hu's bp.m and uses bpfun.m and bptest.m also written by Professor Hu. It uses a training file and a testing file with

each row specifying game data and the last column specifying game results. I have hard coded pp.m to use 94 inputs and one output because all of my data is in that format.

Pp.m lets the user specify the name of the training, the name of the testing file, the number of hidden neurons to use, the learning rate, the momentum constant, the maximum number of epochs to run, the game up to which data is used to predict a champion, and the data file from the season you want to run a playoff on. It outputs the percentage of games correctly predicted by my Neural Network, and a list of games that were incorrectly predicted.

While developing the network, I ran many experiments. The first experiment that I tried was to see if I could make my neural network accurately predict a game outcome based on the **stats from that game**. While this may seem trivial, It was hard to achieve 100% testing and training accuracy. At first, I had horrible results, and I found at that by scaling all of the data to the same range, the statistics with inherently larger values drowned out the smaller (but sometimes more important) values like the number of fumbles. After scaling each statistical column to a range determined by its min and max values, I was able to achieve greater than 98% accuracy training on the 1998 data and testing on the 1999 data, and also with the training and testing samples switched.

Using what I had learned in that experiment, I began running predictions of game outcomes **based only on past data**. At first, I was discouraged by the fact that I could only achieve a maximum correct percentage testing rate of about 75%, until I realized that the Massey Index Ratings (a computer ranking used in the BCS formula) had a correct percentage of prediction of 68.2% for the identical games. A full baseline study is included in the Results section of this paper.

Selection of Network Parameters

To select the network parameters, I first ran experiments to get a rough idea of what parameters worked well. Then, I held each parameter constant, and varied only one of the parameters. I selected the number of hidden neurons, the learning rate, the momentum constant, the number of samples to include in each epoch, and the maximum number of epochs to run in this fashion. The following results are an average of ten trials. For each run, these are the network parameters unless otherwise specified: 12 hidden neurons, learning rate of 0.9, momentum constant of 0.9, 22 samples per epoch, and 400 maximum epochs. The results are the correct testing percentage obtained using the 98 data for training and the 99 data for testing. For configurations that converged some times but not others, I averaged results by counting non-converging trials as a 50% correct rate. Here are some representative samples:

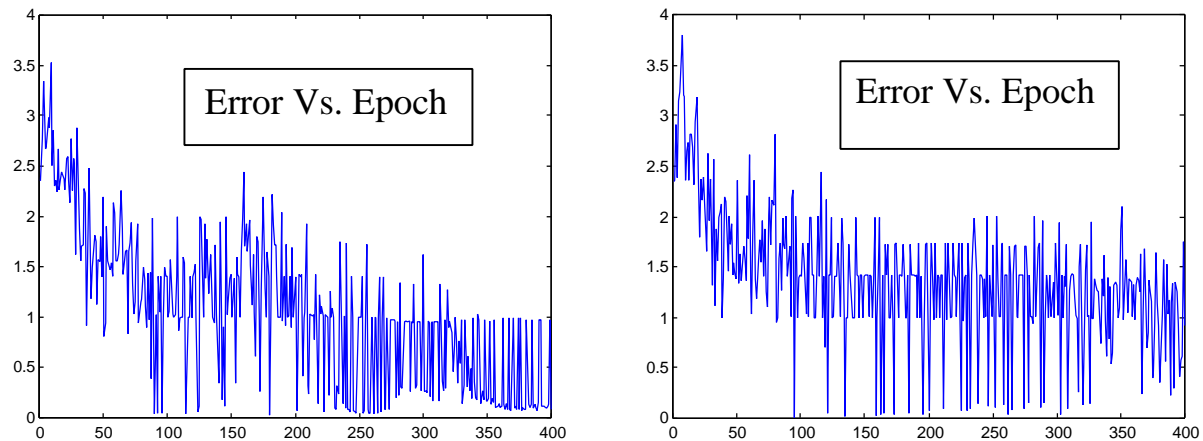
Variable	Too low value	Too low rate	Good value	Good rate	Too high value	Too high rate
hidden neurons	5	72.3%	12	76.2%	20	60.7%
learning rate	.5	73.4%	.9	76.2%	1	74%
momentum	.5	69.1%	.9	76.2%	1	67.1%
samples per epoch	11	72.8%	22	76.2%	44	51.9%
maximum epochs	100	67.3%	400	76.2%	2000	70.5%

I noticed that the momentum constant and the number of epochs run were the most sensitive variables. I also noticed that no matter what the network parameters, sometimes the training set did not converge (randomness of network initialization and re-ordering of rows while training prevents results from being the same each time) no matter what network parameters

were chosen. To eliminate this undesirable behavior, I added code to pp.m that would check to make sure the training correct classification rate was above 60% after 1/3 of the maximum number of epochs. If it was not, it was not likely to converge, so I started training over again from epoch 0 with new initialization. This way we are almost guaranteed convergence (and reasonable results) each time pp.m is run.

I also noticed an inter-dependence of variables. Having a low learning rate and low momentum constant was OK if enough epochs were run. However, I wanted to be able to run many trials of pp.m so I wanted relatively fast runs (running 10000 iterations is slow). The high learning rate and momentum I decided on gave good results after only a few hundred epochs. If more epochs are run, the training rate goes to 99 or 100%, but the network is over-trained, and then we see diminishing testing rates around 68%. Adding more hidden neurons only ruined the convergence of the network, and didn't seem to have improved results when it did converge. The same rule of diminishing returns applied for increasing the number of samples per epoch.

Here are two graphical examples showing the error decreasing as more epochs are run, until a steady-state noisy lower level is reached. By stopping it after about 400 epochs, we can catch the training right at the critical point so that it does not over-train:



Network Usage

Once I had a neural network developed and trained, I used it to predict winners in the testing set. To run pp.m, one can type something like: `[crate, upsets]=pp('bt_all','bt_98',12,.9,.9,22,400)` at the Matlab prompt to get a rate of correct prediction and a list of mis-predictions. I could reliably predict the winner of a game over 76% of the time (sometimes up to 80%). I put my network to use by using it to conduct a “virtual” post-season tournament. First, I had each team “play” every other team, and predicted who would win. Then the 8 teams with the most number of wins were put into a 3-round single elimination tournament, seeded by the number of wins they had during the round robin tournament. Two or more teams could end up with the same number of wins during the round robin phase, but I do not implement a tie break system. In practice there are rarely more than three teams with the same number of wins, and while testing I noticed that the order of the tied teams plays little role in determining the champion at the end of the 8 team playoff.

The flexibility of pp.m enabled me to make predictions for the 1999 season based on 1998 data, or make predictions for the 1998 season based on 1999 data. I could also make

tournament predictions using a neural network trained on 1998,1999, or both years combined, for a tournament held at any point during or after the 1998 or 1999 seasons. Using game data from 1998 and 1999 together to predict post-season tournament winners does not violate the assumption of separate training and testing sets because those tournaments would use cumulative data for all 11 games, which was never used at any point in the training of the neural network.

Results

Prediction Rates

By training on the 1998 data and testing on the 1999 data, I saw an average prediction rate of 76.2% over 100 trials with a range of 65% to 84%. Reversing the training and testing sets, using 1999 data for training and 1998 data for testing, I saw an average of 74% correct over 20 trials with a range of 69% to 81%. This suggests that my method should work for any arbitrary year, and not just 1999. Here is a sample of the **actual Matlab output** of my program which lists the games that my method mis-predicted for a the 1999 season:

```
Game results my NN mis-predicted:

Michigan      defeated Wisconsin
Michigan      defeated Purdue
Illinois      lost to Indiana
Purdue        lost to Michigan
Ohio St.      defeated Purdue
Indiana       defeated Illinois
Minnesota     defeated Illinois
Purdue        lost to Ohio St.
Ohio St.      lost to Penn St.
Iowa          lost to Northwestern
Michigan      lost to Illinois
Minnesota     lost to Ohio St.
Penn St.      defeated Purdue
Ohio St.      defeated Minnesota
Iowa          lost to Indiana
Purdue        lost to Penn St.
Ohio St.      lost to Michigan St.
Wisconsin     defeated Purdue
Ohio St.      lost to Illinois
```

The percentage of correct prediction for that particular run of pp.m was 77.2727%. Note that it lists most upsets twice, i.e. *Michigan defeated Purdue*, and *Purdue lost to Michigan*. I left duplicate output of this form in to check that both scenarios have the same result (remember that the teams that play can be listed in order 1 2 or order 2 1). Also note that *Michigan defeated Wisconsin* was a mis-prediction, but my neural network correctly predicted the Wisconsin would lose to Michigan. This means that the position of the teams matters, despite the fact that it shouldn't in an ideal model. To my credit, I saw that most "un-symmetric" anomalies of this sort occurred on very close match-ups, like Wisconsin vs. Michigan. In one way, this type of output can be interpreted as a game being "too close to call".

Here is another typical output of bp.m, but this is for the 1998 season, for which I used the 1999 season to train. The percentage of correct prediction for this run was 78.4091%

Game results my NN mis-predicted:

```
Iowa          defeated Illinois
Michigan St.  defeated Indiana
Indiana       lost to Michigan St.
Michigan St.  lost to Minnesota
Michigan      defeated Indiana
Purdue        lost to Penn St.
Northwestern lost to Iowa
Minnesota     defeated Michigan St.
Michigan St.  defeated Ohio St.
Michigan      defeated Penn St.
Ohio St.      defeated Indiana
Michigan      defeated Wisconsin
Ohio St.      lost to Michigan St.
Indiana       lost to Illinois
Wisconsin     lost to Michigan
Penn St.      defeated Michigan St.
Illinois      defeated Indiana
Ohio St.      defeated Michigan
```

Post-Season Winners

Pp.m can also conduct a Big Ten tournament. Since there is no real Big Ten tournament, there is no data to compare these results to. I could compare them to other peoples rankings, but correlation to those would say little about correctness. However, one can look for near-impossible results (like Iowa beating Wisconsin) to make sure the results make sense. Just like a real sports tournament, you could run the tournament 3 times and get 3 different champions. But, most of the time the same team will win. For 1999, Wisconsin wins the Big Ten Tournament almost all of time. For 1998, Ohio St. wins the Big Ten Tournament most of the time, but Michigan, Purdue, and Wisconsin all win it once in a while. While the winner may change, there is almost never more than four teams out of the eight that win the tournament.

To run a Big Ten Post-season Tournament for 1999 using 1998 as training data, here is what you would type: `[crate, upsets]=pp('bt_98','bt_99',12,.9,.9,22,400,12,'all_teams_data_99');`
To run a Big Ten Post-season Tournament for 1998 using 1999 as training data, you would type: `[crate, upsets]=pp('bt_99','bt_98',12,.9,.9,22,400,12,'all_teams_data_98');`

Note that we can also conduct a Big Ten Tournament for 1998 or 1999 by typing:

```
[crate, upsets]=pp('bt_all','bt_98',12,.9,.9,22,400,12,'all_teams_data_98');
```

```
[crate, upsets]=pp('bt_all','bt_99',12,.9,.9,22,400,12,'all_teams_data_99');
```

where `bt_all` is a data file with all of the 1998 AND 1999 Big Ten data. As I mentioned in the Network Usage section, using game data from 1998 and 1999 together to predict post-season tournament winners does not violate the assumption of separate training and testing sets. Those tournaments would use cumulative data for all 11 games, which was never used at any point in the training of the neural network..

Here is an example of actual Matlab output from `pp.m` :

```
round_robin_seeding =
 1 Wisconsin
 2 Michigan
 3 Minnesota
 4 Michigan St.
```

5 Purdue
6 Penn St.
7 Illinois
8 Ohio St.
9 Indiana
10 Northwestern
11 Iowa

Here are the match-ups in the single elimination tournament for each round:

```
round_1 =  
Wisconsin      vs. Ohio St.  
Michigan       vs. Illinois  
Minnesota      vs. Penn St.  
Michigan St.   vs. Purdue  
  
round_2 =  
Wisconsin      vs. Purdue  
Michigan       vs. Minnesota  
  
round_3 =  
Wisconsin      vs. Michigan  
  
Runner_up =  
Michigan  
  
Big_ten_champion =  
Wisconsin
```

You can see who won each match-up by noticing who played in the following round.

Baseline Study

In all of my research, I could not find ANY public prediction systems for college football that gave predictions for all teams on a weekly basis. (Without paying for them) Because of the possible money to be made in wagering on the games, it costs at least \$30 to get prediction information, and I'm sure the algorithms involved are kept very secret.

However, as I described in the introduction, a ranking system should be able to predict the outcome of games reasonably well, so I can compare my prediction system to other computer ranking systems. Despite trying very hard, I was only able to find one computer ranking system that is used in the [BCS](#) that listed rankings on a week-by-week basis so that I could compare its prediction accuracy to my model's predictions. The ranking system that I used as a comparison is called the [Massey Index](#). Looking at the [Massey Index weekly rankings](#) for Big Ten Teams (see Appendix B) I found that in 14 out of 44 games played (28 out of 88) the Massey Index had the losing team ranked higher than the winning team. The 1999 Big Ten football schedule can be found in Appendix B, and I have highlighted all of the game outcomes that were mis-predicted by Massey's Index in Green.

Therefore, the Massey Index percentage of correct predictions for Big Ten vs. Big Ten games was $(44-14)/44 = 68.2\%$. My prediction scheme had an average correct prediction rate of over 76%. Additionally, I was able to get prediction rates over 80% some of the time, and if I

had more years of data to train on (I only trained on one season) I suspect I could be correct up to 90% of the time. I had no problems getting above 97% of training predictions correct, even if I trained on the data from both seasons combined. This might suggest that there is not a large difference in year to year data vs. outcomes.

Most importantly, the main advantage that a neural network approach has over conventional methods is that the weights are determined objectively in a neural network model. There is little opportunity for certain individuals to knowingly or unknowingly bias the weights in any way. This particular advantage of the neural network model is hard to quantify, but I think it shows up when comparing the prediction percentages. If a lot of people are fond of Notre Dame, for example, they are more likely to be ranked higher in the polls. However, if they are not truly as good as everyone thinks they are, the rankings are likely to be a poor predictor and Notre Dame will lose to teams ranked lower than them.

I was able to find very little information on other people's attempts at the neural network approach. I did find some information on a neural network model developed by Rick L. Wilson (rlwilsn@okway.okstate.edu), an associate professor of Management Science and Information Systems at Oklahoma State University. He does not publish his current rankings to my knowledge and I was unable to contact him to find out if he has conducted more neural network rankings since the 1995 season. To that end, I was unable to compare my results with another neural network approach. A good future project might be to implement Rick L. Wilson's neural network scheme and compare the results to mine.

Conclusion

My neural network predictor can predict with about 76% accuracy which team will win a Big Ten college football game based solely on past data and game outcomes. If subjective computer rankings like Massey's Index are used to predict game outcomes, they are likely to be about 68% accurate. (Assuming Massey's Index is representative of other computer ranking systems) I assume Massey's Index is representative of other ranking systems because it is very popular, and it has been deemed "worthy" of determining a national champion by the folks at ABC. Therefore, I have constructed a more accurate approach to predicting college football games using a neural network model.

My prediction system could also be used to generate weekly rankings. Weekly rankings could be based on round-robin results, and tie breakers could be done with a single elimination tournament. However, I do not believe that ranking teams is necessarily the best way to determine who the best teams in the country are.

To resolve the debate over a national champion, my neural network predictor could be used to narrow down the field of eligible teams from all of the 114 Division I-A schools. If a "virtual" post-season tournament is conducted as I did for the Big Ten, it is unlikely that the same team will win the tournament each time the tournament is run. However, like I saw in the Big Ten Tournament, if 64 teams are put into a "virtual" tournament (NCAA basketball uses a 64 team tournament) maybe only 8 or so could ever win. Those eight teams might not be the 8 highest ranked teams in the country, but they are the ones that deserve to play for the national championship. Ideally, those eight teams could be involved in a REAL college football playoff that would only take 3 weeks to play, and a REAL national champion could be determined.

References

Wilson, R.L. (1995), "Ranking College Football Teams: A Neural Network Approach,"
Interfaces, Vol. 25, No. 4, 44-59.

ABC's Bowl Championship Series web-site: <http://abccfb.go.com/road/bcsrank.html>

1994 Nation Champion determined by a Neural Network: <http://lionhrtpub.com/orms/10-95text/fb.html>

Massey Index College Football Homepage: <http://www.mratings.com/rate/cf-m.htm>

Massey Index Theory Description: <http://www.mratings.com/theory/index.htm>

Wilson Power Rankings Page with many links to other good rankings sites:

<http://www.cae.wisc.edu/~dwilson/>

Big Ten Football web-site: <http://www.bigten.org/sports/football/>

Appendix A: Code

pp.m

```
%file pp.m
%Written by Mike Pardee
%Pardee Predictor for sporting events
%this was designed specifically for College Football, but
%could be used for any organized sport
%There are some specifics in this file hardcoded to be used
% with the data I have for the 1998 and 1999 Big Ten Conference games
%
%OUTPUTS
%trate is the percentage of games correctly predicted by my Neural Network
(NN)
%upsets is the list of games that were incorrectly predicted
%
%INPUTS
%file_name is the name of the training file (no extensions please)
%test_name is the name of the testing file (no extensions please)
%hidden is the number of hidden neurons to use
%alpha is the learning rate (between 0 and 1)
%mom is the momentum constant (between 0 and 1)
%epoch is the maximum number of epochs to run
%rgame specifies the game up to which data is used
% to predict a champion (giving rgame=12) will use all data,
% and therefor predict a standard postseason tournament
%if rgame is not included, no tournament is run
%if rgame is specified, you must include data file of data from
%the season you want to run the playoff on, ex. all_teams_data_99
%
% general structure and some lines of code taken from:
```

```

% bp.m - backpropagation driver program
% copyright (c) 1996 by Yu Hen Hu
% Created: 9/19/96
%
% I use the following files written By professor Hu:
% bpfun.m to perform backpropagation training
% bptest.m to test training data results.
% bptestp.m Professor Hu's file with the output z accessible
% randomize.m to reorder training data
%
%My files I call from this program:
%teamifyv.m -- turns each number in the vector into its corresponding team
name
%playoff.m -- construct data for a round_robin playoff -- this uses
all_teams_data_98 or 99
%
%I use these additional files to set up the data:
%load_all_teams_data.m -- creates stat files for entire seasons from a
season's directory by calling:
% convdir.pl which converts a all of the html files in a directory into
% suitable data for use with matlab. convdir.pl calls:
% parsestat.pl which takes an html account of a game from the Big Ten
website
% and converts it into a format suitable for matlab with the help of
% txt2mat.pl which puts text comments in the file instead of text headings
% loadfiles.m (called by load_all_teams) takes the data from singular data
files for each individual game
% I used load_all_teams_data to make the two files: all_teams_data_98 and
all_teams_data_99
% make_cum_data.m -- makes a file with cumulative game data up to the current
game
% I used this to make cumulative files from all_teams_data_98 and 99,
% then lopped off the first three non-conference games and saved those
files
% as bt_98 and bt_99 which are my final files used for training and
testing

% Good results: [crate,
upsets]=pp('bt_98','bt_99',12,.9,.9,22,400,12,'all_teams_data_99');

function
[trate,upsets]=b PPP(file_name,test_name,hidden,alpha,mom,e_size,nepoch,rgame,
season_file)

if nargin==7
    rgame=1; %do not run tournament!
end

eval(['load ' file_name]); %load training file
eval(['load ' test_name]); %load testing file
% it is assumed that the training file is an ascii array
% of numbers in a K by MN matrix. There are K lines, each
% corresponding to a 1xM feature vector and an 1xN target vecto

data=eval(file_name); %training data
Testdata=eval(test_name); %testing data

```

```

[Ntrain,MN]=size(data); %get sizes of training and testing data
N=1;% MN-M; %N = 1 = output dimension, win or loss, 0 or 1
M=MN-N;%94; %%94 inputs in training file
H=hidden;% input parameter hidden is the number of hidden neurons
[feature,xmin,xmax]=scalecol(data(:,1:M),-0.5,0.5); % scale input to range of
-.5 to .5
% but unlike the normal bp file by Hu, I scale each variable (column)
independantly,
% so that statistics with inherently large values do not overwhelm the others

for i=1:M %now scale the testing input
    % testing input should subject to the same linear scaling as
    % training input. Thus, testing input may not be limited to -.5 to .5!
    tfeature(:,i)=(Testdata(:,i)-xmin(i))*(0.5-(-0.5))/(xmax(i)-xmin(i))+(-
0.5);
end

target=data(:,M+1:MN);% non-scaled testing output
ttarget=Testdata(:,M+1:MN);% non-scaled testing output

K=e_size; %e_size is the # of samples to include in each epoch
ne=floor(Ntrain/K);

% initialize weight matrices
randn('seed',sum(100*clock));
Wh=randn(H,M+1)*0.005;    Wo=randn(N,H+1)*0.005;
dWh=zeros(H,M+1);    dWo=zeros(N,H+1);

% Prepare training data for an epoch
error=zeros(1,nepoch); check=0; converged=0;
train=randomize([feature target]);

done=0;
while (done==0), %we are not 'done' until we have a realistic classification
rate
    for nn=1:nepoch,
        % since there are Ntrain samples, with K samples per epoch, it takes
        % ne+1 epochs to go throught entire training samples once. In fact,
        % only ne epochs are executed. After that, the order of the Ntrain
        % samples are re-suffled (randomized), and the next K*ne samples will
        % be used in the next round. Each time K*ne samples have been trained,
        % we will perform convergence test.
        ns=rem(nn,ne)*K; %nn modulo ne: 1, 2, ..., ne-1, 0 1, 2, ..., ne-1, 0,
        if ns == 0,
            ns=ne*K;
            check=1; % to check for convergence once later
            train=randomize([feature target]); % randomize once each run of
entire training file
        end
        train_ep=train(ns-K+1:ns,:);
        % call bpfun.m
        % Usage: [err,Wh,Wo,dWh,dWo]=bpfun(M,H,N,Wh,Wo,dWh,dWo,train,alpha,mom)
        [err,Wh,Wo,dWh,dWo]=bpfun(M,H,N,Wh,Wo,dWh,dWo,train_ep,alpha,mom);
        error(nn)=err;

        % don't need to plot since we have automated convergence
        %if rem(nn,20)==0,

```

```

% plot([max(1,nn-1000):nn],error(max(1,nn-1000):nn)); drawnow
%end

if check == 1,
    [Cmat,crate]=bptest(Wh,Wo,[feature data(:,M+1:MN)]) ; % training
classification rate

    if ((crate<=60)&(nn>nepoch/3)) %If it looks like we are not going to
converge, start over!
        train=randomize([feature target]);
        Wh=randn(H,M+1)*0.005;    Wo=randn(N,H+1)*0.005;
        error=0;
        nn=1;
        %clf; %if plotting, clear the figure before starting over
        done=0;
        break
    end

    if crate>=99, %if we get 99 or 100% training correct, we can stop
        converged=1;
        disp([file_name ' converged at ' num2str(nn) ' epoch']);
        disp(['with classification rate = ' num2str(crate)]);
        %plot([1:nn],error(1:nn)); drawnow
        done=1;
        break
    else
        check=0;
    end
end

done=1; %we have either 99% or better correct, or we have trained for the
max number of epochs
end
end

% Now compute testing data set results
%disp('Here is the confusion matrix and the % correct classification for
TRAINING');
[Confusion_matrix_for_training,Percent_correct_for_training]= bptest(Wh,Wo,[fe
ature target])
%disp('Here is the confusion matrix and the % correct classification for
TESTING');
%use bptest.m written by Hu with output of z accessible
[Cmat,crate,z]=bptestp(Wh,Wo,[tfeature ttarget]);
Confusion_matrix_for_Testing =Cmat
Percent_Correct_for_Testing=crate
trate=crate;

%Save mis-predictions in nice format to use later if desired:
num_upsets=0;
for j=2:length(tttarget)+1
    i=j-1;
    if ((ttarget(i)-z(i))~=0) %must be both 1 or both 0, or it is a
misprediction
        num_upsets=num_upsets+1;
        team1=teamifyv(Testdata(i,1));
        team2=teamifyv(Testdata(i,48));
        if (ttarget(i)==1)

```

```

        result=' defeated ';
    else
        result=' lost to ';
    end
    upsets(num_upsets,:)= [team1 result team2];
end
end
upsets(1,:)=('Game results my NN mis-predicted: ');

if rgame==1 %don't run tournament
    break
end

%% NOW, in addition to training and testing, use the NN we have built
% to run a 'VIRTUAL PLAYOFF' between all of the teams, with the following
format:
%First Conduct a round robin tournament between all the teams, with each team
playing
%every other team.
%Then, based on how many games each team won in the round-robin phase,
% seed the top eight teams into a single elimination tournament,
% the winner of which should be the Big-Ten champion

playoff %load rr tournament data
%load rr_data; %contains setup for round robin tournament

%scale the round robin tournament data
for i=1:size(rr_data,2)-1
    rr_data(:,i)=(rr_data(:,i)-xmin(i))*(0.5-(-0.5))/(xmax(i)-xmin(i))+(-
0.5);
    % testing input should subject to the same linear scaling as
    % training input. Thus, testing input may not be limited to -.5 to .5!
end

randomize(rr_data); %run bptestp on the round robin data to determine
%how many games each team would win
[Cmat,crate,z]=bptestp(Wh,Wo,[rr_data]);

% Count wins in round-robin phase:
wins=zeros(1,11);
for team=1:11
    wins(team)=sum(z((team-1)*10+1:team*10,1),1);
end
[wins,teams]=sort(wins);
teams=fliplr(teams);
round_robin_seeding=[num2str((1:11)') repmat(' ',11,1) teamifyv(teams)] %
repmat(' ',11,1) num2str(fliplr(wins)')]

disp('Here are the matchups in the single elimination tournament for each
round:');
%now we have teams in round robin win order, now we need to play them off
%Round 1
% 1 plays 8, 2 plays 7, 3 plays 6, 4 plays 5
round1 = ...
    [teams(1) teams(2) teams(3) teams(4); ...%team
    teams(8) teams(7) teams(6) teams(5)]; %opponent
round_1=[teamifyv(round1(1,:)) repmat(' vs. ',4,1) teamifyv(round1(2,:))]

```

```

%use our NN to see who should win playoff game
playoff_data=zeros(1,95);
for pgame=1:4
    team=round1(1,pgame);
    opponent=round1(2,pgame);
    pos = game-1+11*(team-1);
    opp_pos = game-1+11*(opponent-1);
    playoff_data(pgame,:)=[cdata(pos,:) cdata(opp_pos,:) -1];%result];
end

%scale round 1 data
for i=1:size(playoff_data,2)-1
    playoff_data(:,i)=(playoff_data(:,i)-xmin(i))*(0.5-(-0.5))/(xmax(i)-
xmin(i))+(-0.5);
    % testing input should subject to the same linear scaling as training
input.
end

%use bptest.m written by Hu with output of z accessible to determine first
round winners
[Cmat,crate,z]=bptestp(Wh,Wo,[playoff_data]);

%determine list of winners to use in round 2
for i=1:4
    if z(i,1)==1;
        winner(i)=round1(1,i);
    else
        winner(i)=round1(2,i);
    end
end

% Now conduct round 2 of the playoffs, now with 4 teams left
% see round 1 for more detailed comments
round2 = ...
[winner(1) winner(2) ;...%team
winner(4) winner(3) ]; %opponent
round_2=[teamifyv(round2(1,:)) repmat(' vs. ',2,1) teamifyv(round2(2,:))]

playoff_data=zeros(1,size(playoff_data,2));
for pgame=1:2
    team=round2(1,pgame);
    opponent=round2(2,pgame);
    pos = game-1+11*(team-1);
    opp_pos = game-1+11*(opponent-1);
    playoff_data(pgame,:)=[cdata(pos,:) cdata(opp_pos,:) -1];%result];
end

for i=1:size(playoff_data,2)-1
    playoff_data(:,i)=(playoff_data(:,i)-xmin(i))*(0.5-(-0.5))/(xmax(i)-
xmin(i))+(-0.5);
    % testing input should subject to the same linear scaling as
    % training input. Thus, testing input may not be limited to -.5 to .5!
end

%use bptest.m written by Hu with output of z accessible
[Cmat,crate,z]=bptestp(Wh,Wo,[playoff_data]);

```

```

for i=1:2
    if z(i,1)==1;
        winner(i)=round2(1,i);
    else
        winner(i)=round2(2,i);
    end
end

%Round 3 -- the Final Game
round3 = ...
    [winner(1) ;...%team
    winner(2) ]; %opponent
round_3=[teamifyv(round3(1,:)) repmat(' vs. ',1,1) teamifyv(round3(2,:))]

playoff_data=zeros(1,size(playoff_data,2));
for pgame=1:1
    team=round3(1,pgame);
    opponent=round3(2,pgame);
    pos = game-1+11*(team-1);
    opp_pos = game-1+11*(opponent-1);
    playoff_data(pgame,:)=[cdata(pos,:) cdata(opp_pos,:) -1];%result];
end

for i=1:size(playoff_data,2)-1
    playoff_data(:,i)=(playoff_data(:,i)-xmin(i))*(0.5-(-0.5))/(xmax(i)-
xmin(i))+(-0.5);
    % testing input should subject to the same linear scaling as
    % training input. Thus, testing input may not be limited to -.5 to .5!
end

%use bptest.m written by Hu with output of z accessible
[Cmat,crate,z]=bptestp(Wh,Wo,[playoff_data]);

for i=1:1
    if z(i,1)==1;
        winner(i)=round3(1,i);
        loser=round3(2,i);
    else
        winner(i)=round3(2,i);
        loser=round3(1,i);
    end
end

Runner_up=teamifyv(loser)
Big_ten_champion=teamifyv(winner(1))

```

teamifyv.m

```

%Mike Pardee
%teamifyv.m
%Takes a vector of numbers and outputs a text string representing the
corresponding teams

function strout = teamifyv(teamnum);
strout='          ';
for i=1:(length(teamnum))

```

```

if (teamnum(i)==1)
    strout(i,:) = 'Wisconsin  ' ;
    %return
end
if (teamnum(i)==2)
    strout(i,:) = 'Michigan St.' ;
    %return
end
if (teamnum(i)==3)
    strout(i,:) = 'Michigan  ' ;
    %return
end
if (teamnum(i)==4)
    strout(i,:) = 'Minnesota  ' ;
    %return
end
if (teamnum(i)==5)
    strout(i,:) = 'Penn St.  ' ;
    %return
end
if (teamnum(i)==6)
    strout(i,:) = 'Illinois  ' ;
    %return
end
if (teamnum(i)==7)
    strout(i,:) = 'Purdue  ' ;
    %return
end
if (teamnum(i)==8)
    strout(i,:) = 'Ohio St.  ' ;
    %return
end
if (teamnum(i)==9)
    strout(i,:) = 'Indiana  ' ;
    %return
end
if (teamnum(i)==10)
    strout(i,:) = 'Northwestern' ;
    %return
end
if (teamnum(i)==11)
    strout(i,:) = 'Iowa  ' ;
    %return
end

if (teamnum(i)==99)
strout(i,:) = 'Non-Big Ten ' ;
    %return
end
end

```

playoff.m

```

%Mike Pardee
%playoff.m
%called by pp.m
%takes data from a season, and constructs data to be used in a round-robin

```

```

%playoff conducted in pp.m

% season_file must be defined in pp.m
eval(['load ' season_file]); %load all teams data file
data=eval(season_file); %call it data

cumdata=zeros(11*11,23*2+1);
cdata=zeros(11*11,23*2+1);

%construct historical data from sequential game data up through current game
for team=1:11
    cumulative=zeros(11,size(data,2));

    for game=2:12 %cumulative(12) has historical data from games 1-11
        %take average stats over number of games played
        cumulative(game,:)=(cumulative(game-1,:).*(game-2)+data(team+(game-
2)*11,:))./(game-1);
        %each games cumulative data is cumulative data from previous game +last
game
    end
    %cumulative now holds historical data for previous games for games 2-12
for team and teams opponents
    cumdata((team-1)*11+1+1:(team-1)*11+1+11,:)=cumulative(2:12,:);

end
cdata = cumdata(2:11*11+1,:);
%save cdata cdata -ascii
% cumdata has all historical data for team vs. opponent
% now we need to get the data of the opponents in the right rows for each
game

train_data=zeros(10*11,23*4+2+1);

game=rgame; %rgame is input from pp.m

for team=1:11
    skip_yourself=0;
    for opponent=1:11
        pos = game-1+11*(team-1);
        td_pos = opponent+10*(team-1); %training data only has 10 per team
        %td_pos = team + (opponent-1)*11;

        %take teams data and opponents data from prior games and result of
present game

        if (opponent==team) %can't play yourself
            skip_yourself=1;
            %opp_pos=1; %not used for non-conference opponents
            %train_data(td_pos,:)= [cdata(pos,:) zeros(1,47) result];
            %since we don't have data for non-conference opponents, model
them as horrible (=0)
        else
            opp_pos = game-1+11*(opponent-1);
            rr_data(td_pos-skip_yourself,:)= [cdata(pos,:) cdata(opp_pos,:) -
1];%result];
            [team opponent td_pos-skip_yourself*11];

```

```

        end

    end

end

%save rr_data rr_data -ascii

```

make_cumulative_data.m

```

%Mike Pardee
%make_cumulative_data.m
%Makes a cumulative data file for use with pp.m

%uncomment the one you don't want to generate data for
%load 'all_teams_data_98'; data = all_teams_data_98;
load 'all_teams_data_99'; data = all_teams_data_99;

cumdata=zeros(11*11,23*2+1);
cdata=zeros(11*11,23*2+1);

%construct historical data from sequential game data up through current game
for team=1:11
    cumulative=zeros(11,size(data,2));

    for game=2:12 %cumulative(12) has historical data from games 1-11
        %take average stats over number of games played
        cumulative(game,:)=(cumulative(game-1,:).*(game-2)+data( team+(game-
2)*11,:))./(game-1);
        %each games cumulative data is cumulative data from previous game +last
game
    end
    %cumulative now holds historical data for previous games for games 2-12
for team and teams opponents
    cumdata((team-1)*11+1+1:(team-1)*11+1+11,:)=cumulative(2:12,:);

end
cdata = cumdata(2:11*11+1,:);
% cumdata has all historical data for team vs. opponent
% now we need to get the data of the opponents in the right rows for each
game

train_data=zeros(10*11,23*4+2+1);

for team=1:11
    for game=2:11
        opponent=data(team+(game-1)*11,24);
        pos = game-1+11*(team-1);
        td_pos = game-1+10*(team-1); %training data only has 10 per team
        td_pos = game-1+10*(team-1); %training data only has 10 per team
        td_pos = team + (game-2)*11;

        %take teams data and opponents data from prior games and result of
present game
        result = data(team+(game-1)*11,47);
        [opponent result cdata(pos,47)]

        if (opponent==99)

```

```

        opp_pos=1; %not used for non-conference opponents
        train_data(td_pos,:)= [cdata(pos,:) zeros(1,47) result];
        %since we don't have data for non-conference opponents, model
them as horrible (=0)
    else
        opp_pos = game-1+11*(opponent-1);
        train_data(td_pos,:)= [cdata(pos,:) cdata(opp_pos,:) result];
    end

end

end
save train_data train_data -ascii

```

load_all_teams_data.m

```

%Mike Pardee
%load_all_teams_data.m
%loads all data from a directory structure with html files
%for each game in a separate directory for each team
clear all

%matlab does not do three dimensional arrays, so
%each team will get a row in a big array, knowing
%there are game_size columns per game will let separate the data later.
game_size = 23*2+1;%23 stats per team + win\loss
all_teams_data = zeros(11,11*(game_size));

team=1
cd wisconsin; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
% wisconsin identifier # is 1
all_teams_data(1,:)=reshape(bigmat,1,11*game_size);

team=2
cd michigan_st; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 2
all_teams_data(2,:)=reshape(bigmat,1,11*game_size);

team=3
cd michigan; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 3
all_teams_data(3,:)=reshape(bigmat,1,11*game_size);

team=4
cd minnesota; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 4
all_teams_data(4,:)=reshape(bigmat,1,11*game_size);

team=5
cd penn_st; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 5
all_teams_data(5,:)=reshape(bigmat,1,11*game_size);

```

```

team=6
cd illinois; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
% illinois identifier # is 6
all_teams_data(6,:)=reshape(bigmat,1,11*game_size);

team=7
cd purdue; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 7
all_teams_data(7,:)=reshape(bigmat,1,11*game_size);

team=8
cd ohio_st; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 8
all_teams_data(8,:)=reshape(bigmat,1,11*game_size);

team=9
cd indiana; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 9
all_teams_data(9,:)=reshape(bigmat,1,11*game_size);

team=10
cd northwestern; dos('perl c:\pl\convdir.pl');
loadfiles; cd ..
%identifier # is 10
all_teams_data(10,:)=reshape(bigmat,1,11*game_size);

team=11
cd iowa; dos('perl c:\pl\convdir.pl');
loadfiles; %cd .. %identifier # is 11
all_teams_data(11,:)=reshape(bigmat,1,11*game_size);

all_teams_data=reshape(all_teams_data,121,47);
save all_teams_data all_teams_data -ascii

cd ..

```

loadfiles.m

```

%Mike Pardee
%loadfiles.m
%called by load_all_data.m
%executes a perl script which changes html
%files for each game into matlab usable files

game_size = 23*2+1;%23 stats per team + win\loss
bigmat = zeros(11,game_size);
for i=1:11
    fname = ['game' mat2str(i)];
    eval(['load ' fname]);
    eval(['current = game' mat2str(i) ';'']);
    if (team ~= current(1))
        %make sure team is always listed first
        current=fliplr(current);
    end
end

```

```

end

%second column of stats is final score
if (current(2,1)>current(2,2))
    %win for team
    win=1;
elseif (current(1,1)==current(1,2))
    %tie (no ties in 98 or 99 seasons!)
    win=.5;
else %team lost
    win=0;
end

current=reshape(current,1,size(current,1)*2);
current(47)=win;
bigmat(i,:)=current;
end
save bigmat bigmat -ascii

```

convdir.pl

```

#! c:\perl\bin\perl -w
#Mike Pardee
#file: convdir.pl
#ECE 539 Final Project
#this converts a directory full of html files
#into properly named data files suitable for use with Matlab

#NOTE: WILL ONLY WORK ON DOS SYSTEMS!
# could work on unix with slight modification
# to the dir command

@afiles = `dir /od /b *.html`;
print "@afiles\n";

$i = 1;
foreach $tfile (@afiles){
    chomp $tfile;
    $tfile =~ s/\.html//;
    print "$tfile.html \n";
    `perl c:\\pl\\parsestat.pl $tfile.html game$i.txt`;
    `perl c:\\pl\\txt2mat.pl game$i.txt game$i`;
    $i++;
}

```

parsestat.pl

```

#! c:\perl\bin\perl -w
#Mike Pardee
#ECE 539 Final Project
# parses an html file and extracts relevant game statistics

$old = shift(@ARGV);
$new = shift(@ARGV);

open SF, $old;

```

```

open DF, "> $new";

$_="";

#look for beginning of box score
while(!(/NCAAF/)){$_=<SF>;}
s/\<pre\>//; # format ncaaf line
#print DF; #print DF ncaaf line

$_ = <SF>;$_ = <SF>;
s/\(.*\)/ /; #strip out ranking info
#print DF; # print DF first set of scores
s/([a-zA-Z]+) ([a-zA-Z]+)/$1_$2/;
s/([a-zA-Z]+) ([a-zA-Z]+)/$1_$2/;
@ts1 = split /\s{1,}/, $_;
#print DF @ts1;
#print DF "\n";

$_=<SF>;
s/ FINAL//;
s/\(.*\)/ /;
s/([a-zA-Z]+) ([a-zA-Z]+)/$1_$2/;
s/([a-zA-Z]+) ([a-zA-Z]+)/$1_$2/;
@ts2=split /\s{1,}/, $_;
#print DF @ts2;
#print DF "\n";
#print DF; #print DF second set of scores

$teams = "Team_numbers \t\t $ts1[0] \t\t $ts2[0]\n";
#print DF $teams;
#$teams =~ s/Iowa St/99/; #non conference
$teams =~ s/Wisconsin/1/;
$teams =~ s/Michigan_St/2/;
$teams =~ s/Michigan/3/;
$teams =~ s/Minnesota/4/;
$teams =~ s/Penn_St/5/;
$teams =~ s/Illinois/6/;
$teams =~ s/Purdue/7/;
$teams =~ s/Ohio_St/8/;
$teams =~ s/Indiana/9/;
$teams =~ s/Northwestern/10/;
$teams =~ s/Iowa/11/;
$teams =~ s/([a-zA-Z]+) ([a-zA-Z]+)/$1_$2/;
$teams =~ s/\s[a-zA-Z_]+/ 99/; #non conference

#print DF "Team_names $teams"; # team names
print DF $teams;
print DF "final_score \t\t $ts1[5] \t\t $ts2[5]\n";
print DF "1_q_score \t\t $ts1[1] \t\t $ts2[1]\n";
print DF "2_q_score \t\t $ts1[2] \t\t $ts2[2]\n";
print DF "3_q_score \t\t $ts1[3] \t\t $ts2[3]\n";
print DF "4_q_score \t\t $ts1[4] \t\t $ts2[4]\n";

# NOW LOOK FOR REST OF BOX SCORE
while(!(/^First downs/)){ $teams=$_; $_=<SF>;}

while(!(/^Time of possession/)){

```

```

##print DF DF $_;
s/Punts/Punts-Ave/;
s/First downs/First_downs/;
s/ yards/_yards/;
s/yards lost/yards_lost/;

if(/Passes/){
s/Passes/P_Completions-P_Attempts-P_Int/;
@fields = split /-| {2,}/, $_;
print DF "$fields[0] \t\t $fields[3] \t\t $fields[6]\n";
print DF "$fields[1] \t\t $fields[4] \t\t $fields[7]\n";
print DF "$fields[2] \t\t $fields[5] \t\t $fields[8]";
$_ = "";
}
if (/-/){
@fields = split /-| {2,}/, $_;
print DF "$fields[0] \t\t $fields[2] \t\t $fields[4]\n";
print DF "$fields[1] \t\t $fields[3] \t\t $fields[5]";
}
else {print DF}
#print DF "\n";
#print DF;

$_=<SF>;
}
s/Time of possession/Possession_time/;
s/(\d+):(\d+)/$1/g; #seconds not crucial
print DF; # time of possession

close SF;
close DF;

```

txt2mat.pl

```

#Mike Pardee
#ECE 539 Final Project
#used to convert easy to read .txt file
#into Matlab readable format

#! c:\perl\bin\perl -w

$old = shift(@ARGV);
$new = shift(@ARGV);

open SF, $old;
open DF, "> $new";

while(<SF>){
s/M/-/;
#print;
@twovalus = split /\s+/, $_;
print DF "$twovalus[1]\t\t$twovalus[2]\t\t%$twovalus[0]\n";
}

close SF;
close DF;

```

Appendix B: Additional Files

game11

this is a sample of one of the files that is input to Matlab (Wisconsin vs. Iowa 1999):

```
11    1    %Team_numbers
3     41   %final_score
0     13   %1_q_score
3     14   %2_q_score
0     7    %3_q_score
0     7    %4_q_score
14    29   %First_downs
28    60   %Rushed
116   420  %yards
131   184  %Passing_yards
1     1    %Sacked
0     2    %yards_lost
21    20   %Return_yards
14    11   %P_Completions
30    16   %P_Attempts
0     0    %P_Int
6     2    %Punts
50.0  43.0  %Ave
3     0    %Fumbles
1     0    %lost
4     4    %Penalties
18    44   %yards
22    37   %Possession_time
```

game11.txt

this is a sample of one of the files that is output from parsestat.pl (Wisconsin vs. Iowa 1999):

```
Team_numbers      11      1
final_score       3      41
1_q_score         0      13
2_q_score         3      14
3_q_score         0      7
4_q_score         0      7
First_downs      14      29
Rushed           28      60
yards            116     420
Passing_yards    131     184
Sacked           1      1
yards_lost       0      2
Return_yards     21      20
P_Completions    14      11
P_Attempts       30      16
P_Int            0      0
Punts           6      2
Ave              50.0    43.0
Fumbles          3      0
lost             1      0
Penalties        4      4
```

yards	18	44
Possession_time	22	37

game11.html

this is a sample of one of the files that is input to Matlab (Wisconsin vs. Iowa 1999):

Wisconsin 41, Iowa 3

RECAP

MADISON, Wisconsin (Ticker) --- Ron Dayne became the all-time leading rusher in NCAA Division I-A history and ninth-ranked Wisconsin made it all the more special by earning a return trip to the Rose Bowl.

Dayne rushed for 216 yards and a touchdown on 27 carries to break Ricky Williams' record in the Badgers' 41-3 Big Ten Conference victory over last-place Iowa.

Needing just 99 yards to set the mark, Dayne surpassed Williams on a 31-yard run with 4:23 remaining in the second quarter. He rushed for 127 yards on 17 carries in the first half alone.

"I am happy and grateful," Dayne said. "I don't know whether to laugh or cry."

Dayne, currently second in the nation in rushing, closed the regular season with 6,397 yards and eclipsed the 100-yard mark for the 31st time in 41 career starts. The NCAA does not include bowl games in its records.

It was the 11th time Dayne cleared 200 yards, tying the record shared by Williams and Marcus Allen.

The Wisconsin fans marked the occasion by hanging Dayne's uniform No. 33 from the stadium facade.

"I just want to say to all the fans, 'Thanks, I love you all,'" he said. "I was amazed and dazed."

Last season, Williams broke the 22-year-old rushing record held by Tony Dorsett, finishing his career at Texas with 6,279 yards before the New Orleans Saints made him the fifth pick in the draft.

After being held to 80 yards at Minnesota on October 16, Dayne came on strong, averaging 196.2 yards over the final five contests, including three 200-yard games. He rushed for 222 yards in last week's 28-21 victory over Purdue.

"With our offense, he was going to get enough carries," Wisconsin coach Barry Alvarez said.

"The important thing was that he had to stay healthy. We felt if he stayed healthy for 11 games, he would break the record and thank goodness it worked out."

Dayne presented the game ball to Bernie Wyatt, the assistant coach who recruited him to Wisconsin.

"It feels good," Dayne added. "We got one more game to play and I just want to go out and play hard and enjoy it."

Earlier today, Wisconsin (9-2, 7-1) got some help from Michigan, which upset No. 8 Penn State, 31-27. The Badgers' victory gave them the outright Big Ten championship and clinched their third trip to the Rose Bowl in the 1990s.

Knowing Penn State had lost, the Badgers put away Iowa (1-9, 0-7) early, grabbing a 27-3 halftime lead and cruising to their seventh straight victory.

"I tried to do everything like I usually do," Dayne said. "It seemed like a normal game to me." With the outcome well in hand, the capacity crowd of 79,404 turned Camp Randall Stadium into a party-like atmosphere in the second half.

Alvarez, on crutches while awaiting knee surgery, came down from the press box early in the fourth quarter to hug his players, drawing a huge ovation from the Wisconsin fans.

"Ron's been a special player to me and I think we have a good bond," Alvarez said. "The first time I met him, I hugged him, so I think it was only appropriate that I hugged him when he broke the record. I'll never forget that big smile of his." Freshman Brooks Bollinger completed 9-of-12 passes for 144 yards and three touchdowns for Wisconsin, which was just 2-2 following consecutive losses to Cincinnati and Michigan in September. Bollinger, who improved to 7-0 as a starter, also rushed for 113 yards on 11 carries.

Wisconsin, which gained 604 yards of total offense, becomes the first Big Ten team to make back-to-back appearances in Pasadena since Michigan in 1992 and 1993. The Badgers have won consecutive titles for the first time in more than 100 years. Wisconsin captured back-to-back Western Conference crowns in 1896 and 1897, its first two seasons of play.

Iowa was limited to just 247 yards. Scott Mullen completed 14-of-30 passes for 131 yards while Ladell Betts rushed for 73 yards on 16 carries. The Badgers took a 13-0 lead in the first quarter on Bollinger's four-yard touchdown pass to Chad Kuhns and Dayne's one-yard TD plunge.

After Iowa got on the board on Jason Baker's 22-yard field goal 4:07 into the second period, Bollinger tossed TD passes of 24 and 16 yards to Chris Chambers, giving the Badgers a 24-point halftime margin.

Bollinger added a two-yard TD run in the fourth quarter before Scott Kavanaugh capped the

scoring with a one-yard TD plunge.

Wisconsin posted its third straight win over Iowa and grabbed a 37-36-2 lead in the all-time series.

The Hawkeyes close their first season under coach Kirk Ferentz next week against Minnesota.

BOXSCORE

NCAAF	1	2	3	4	F	
	-	-	-	-	--	
Iowa	0	3	0	0	3	
Wisconsin (9)	13	14	7	7	41	FINAL

Wisconsin-Kuhns 4 pass from Bollinger (Pisetsky kick)
Wisconsin-Dayne 1 run (kick failed)
Iowa-FG Mclaughlin 22
Wisconsin-Chambers 24 pass from Bollinger (Pisetsky kick)
Wisconsin-Chambers 16 pass from Bollinger (Pisetsky kick)
Wisconsin-Bollinger 2 run (Pisetsky kick)
Wisconsin-Kavanagh 1 run (Pisetsky kick)

	Iowa	Wisconsin
First downs	14	29
Rushed-yards	28-116	60-420
Passing yards	131	184
Sacked-yards lost	1-0	1-2
Return yards	21	20
Passes	14-30-0	11-16-0
Punts	6-50.0	2-43.0
Fumbles-lost	3-1	0-0
Penalties-yards	4-18	4-44
Time of possession	22:14	37:46

Individual Statistics

RUSHING: Iowa-Betts 16-73, Mullen 12-43. Wisconsin-Dayne 27-216, Bollinger 11-113, Unertl 7-37, Faulkner 5-36, Bennett 6-14, Carpenter 1-5, Kavanagh 1-1, Team 2-minus 2.

PASSING: Iowa-Mullen 14-30-0-131. Wisconsin-Bollinger 9-12-0-144, Kavanagh 2-4-0-40.

RECEIVING: Iowa-Kasper 8-74, Oliver 2-27, Yamini 2-12, Betts 1-16, Thein 1-2. Wisconsin-Chambers 3-73, D Brown 3-58, Kuhns 3-39, Carpenter 1-11, N Davis 1-3.

Att: 79,404

Massey Index weekly rankings for 1999

Week	12/5	11/28	11/21	11/14	11/7	10/31	10/24	10/17	10/10	10/3	9/26	9/19	9/12	9/5	8/29	Pre
Wisconsin	7	6	6	7	9	11	10	16	15	26	54	59	18	17	12	11
Michigan	8	9	10	9	13	13	11	5	6	4	7	6	8	9	9	7
Michigan St	9	10	11	12	12	16	12	9	3	5	4	3	26	25	21	23
Penn St	10	12	12	11	7	4	4	6	7	13	16	15	22	3	3	8
Minnesota	17	17	18	17	21	29	20	15	20	11	23	29	37	39	52	53
Purdue	19	19	23	24	22	17	26	18	23	18	11	11	6	8	17	15
Illinois	22	23	24	28	35	41	32	48	37	39	21	14	35	53	76	76
Ohio St	36	36	37	31	25	19	22	30	30	41	25	26	9	22	16	2
Indiana	69	67	66	63	63	62	56	69	66	77	91	92	84	69	68	69
Northwestern	85	85	84	83	79	76	71	67	78	76	71	73	79	85	50	51
Iowa	97	98	95	99	99	96	92	80	68	61	56	53	74	63	46	45

1999 Season Schedule/ Mis-predictions of Massey Index

Mis-predictions using Massey Index are highlighted in green.

Note: There were no mispredictions involving Indiana, so their schedule is not included explicitly here.

Wisconsin (9-2, 7-1 BIG TEN)	Result
Murray State	W, 49-10
Ball State	W, 50-10
at Cincinnati	L, 12-17
MICHIGAN	L, 16-21
at OHIO STATE	W, 42-17
at MINNESOTA	W, 20-17 (OT)
INDIANA	W, 59-0
MICHIGAN STATE	W, 40-10
at NORTHWESTERN	W, 35-19
at PURDUE	W, 28-21
IOWA	W, 41-3

Michigan St. (9-2, 6-2 BIG TEN)	Result
Oregon	W, 27-20
Eastern Michigan	W, 51-7
at Notre Dame	W, 23-13
at ILLINOIS	W, 27-10
IOWA	W, 49-3
MICHIGAN	W, 34-31
at PURDUE	L, 28-52
at WISCONSIN	L, 10-40
OHIO STATE	W, 23-7
at NORTHWESTERN	W, 34-0
PENN STATE	W, 35-28

Michigan (9-2, 6-2 BIG TEN)	Result
Notre Dame	W, 26-22
Rice	W, 37-3
at Syracuse	W, 18-13
at WISCONSIN	W, 21-16
PURDUE	W, 38-12
at MICHIGAN STATE	L, 31-34
ILLINOIS	L, 29-35
at INDIANA	W, 34-31
NORTHWESTERN	W, 37-3
at PENN STATE	W, 31-27
OHIO STATE	W, 24-17

Minnesota (8-3, 5-3 BIG TEN)	Result
Ohio	W, 33-7
Northeast Louisiana	W, 35-0
Illinois State	W, 55-7
at NORTHWESTERN	W, 33-14
WISCONSIN	L, 17-20 (OT)
at ILLINOIS	W, 37-7
OHIO STATE	L, 17-20
PURDUE	L, 28-33
at PENN STATE	W, 24-23
INDIANA	W, 44-20
at IOWA	W, 25-21

Penn St. (9-3, 5-3 BIG TEN)	Result
Arizona	W, 41-7
Akron	W, 70-24
Pittsburgh	W, 20-17
at Miami	W, 27-23
INDIANA	W, 45-24
at IOWA	W, 31-7
OHIO STATE	W, 23-10
at PURDUE	W, 31-25
at ILLINOIS	W, 27-7
MINNESOTA	L, 23-24
MICHIGAN	L, 27-31
at MICHIGAN STATE	L, 28-35

Illinois (7-4, 4-4 BIG TEN)	Result
Arkansas State	W, 41-3
San Diego State	W, 38-10
at Louisville	W, 41-36
MICHIGAN STATE	L, 10-27
at INDIANA	L, 31-34 (OT)
MINNESOTA	L, 7-37
at MICHIGAN	W, 35-29
PENN STATE	L, 7-27
at IOWA	W, 40-24
at OHIO STATE	W, 46-20
NORTHWESTERN	W, 29-7

Purdue (7-4, 4-4 BIG TEN)	Result
at Central Florida	W, 47-13
Notre Dame	W, 28-23
Central Michigan	W, 58-16
NORTHWESTERN	W, 31-23
at MICHIGAN	L, 12-38
at OHIO STATE	L, 22-25
MICHIGAN STATE	W, 52-28
PENN STATE	L, 25-31
at MINNESOTA	W, 33-28
WISCONSIN	L, 21-28
at INDIANA	W, 30-24

Ohio St. (6-6, 3-5 BIG TEN)	Result
Miami	L, 12-23
UCLA	W, 42-20
Ohio	W, 40-16
Cincinnati	W, 34-20
WISCONSIN	L, 17-42
PURDUE	W, 25-22
at PENN STATE	L, 10-23
at MINNESOTA	W, 20-17
IOWA	W, 41-11
at MICHIGAN STATE	L, 7-23
ILLINOIS	L, 20-46
at MICHIGAN	L, 17-24

Northwestern (3-8, 1-7 BIG TEN)	Result
Miami (Ohio)	L, 3-28
TCU	W, 17-7
at Duke	W, 15-12 (OT)
at PURDUE	L, 23-31
MINNESOTA	L, 14-33
at INDIANA	L, 17-34
IOWA	W, 23-21
WISCONSIN	L, 19-35
at MICHIGAN	L, 3-37
MICHIGAN STATE	L, 0-34
at ILLINOIS	L, 7-29

Iowa (1-10, 0-8 BIG TEN)	Result
Nebraska	L, 7-42
at Iowa State	L, 10-17
Northern Illinois	W, 24-0
at MICHIGAN STATE	L, 3-49
PENN STATE	L, 7-31
at NORTHWESTERN	L, 21-23
INDIANA	L, 31-38
at OHIO STATE	L, 11-41
ILLINOIS	L, 24-40
at WISCONSIN	L, 3-41
MINNESOTA	L, 21-25