CMSC726 – Final Project Report
NCAA Football Win/Loss Predicition

Cody Buntain
Department of Computer Science
University of Maryland

December 19, 2012

1 Introduction

Sports team performance prediction has been a topic of interest for as long as sports has been around and has been tackled in as many different ways. Today, large organizations have grown up around this problem to provide higher quality rankings, predictions, betting lines, etc. These organizations coupled with the desire to know who will win a game and how a team can improve has resulted in the collection of vast amounts of readily-accessible data specific to this problem. One organization, the NCAA, maintains an electronic and publicly available copy of a significant number of statistics for every team in the two primary divisions, the Football Bowl Subdivision (FBS) a the Football Championship Subdivision (FCS) (approximately 240 teams) going back 10 years. Even with all of this information, ranking mechanisms and the Vegas line (who has a vested interested in top-quality prediction) still only get between 80-85% of winners correct. This project seeks to build a collection of classifiers using the NCAA data as input that rival the picks predicted by published team rankings.

2 Methods

Given the data available via the NCAA's website [1], which is replete with final game scores, this problem is clearly one of supervised learning. That is, the data at hand provides easily computable labels of win/loss per game. In addition, the NCAA stores complete statistics on all teams in the FBS and FCS divisions from 2008 to the present (prior to 2008 is incomplete), so the project has a wealth of data on which to learn (approximately 100 games per week over 14 weeks per season over 4 seasons for a total of around 6000 samples plus the current 2012 season for testing). Furthermore, for each team, the NCAA provides a selection of features to use (e.g., per-week passing yards, rushing yards, touchdowns, sacks, etc.).

For learning algorithms, this project will start with the development of an averaged perceptron as a baseline for performance. Following, the project will explore various configurations of support vector machines (SVMs), both linear and kernelized, to determine whether non-linear systems perform better than the averaged perceptron's linear classifier. Lastly, the project will include the development of a neural network. Once these three learners have been developed, the project will conclude with a brief exploration of ensemble methods for combining these learners in an interesting way.

For evaluation, each learner and the final ensemble method will be compared with the prediction results of using only the team rankings in the 2012 season to determine which solution provides the best accuracy.

[1] http://www.ncaa.org/wps/wcm/connect/public/NCAA/Resources/Stats/Football/index.html

2.1 Data and Code

This project contains a divided code base and a SQLite database for storing team schedules and game statistics. The code base consists of two segments: the first is a set of Python scripts for downloading and parsing team lists, schedules, and game statistics from the NCAA website and storing this data in the database. The second, more interesting segment is the Java-based connector that pulls data from the database, transforms it to a learnable dataset, and performs the training, tuning, and testing. As part of the data extraction process, the Java code also provides a middle layer for creating object-oriented structures around play schedules, teams, and games, which will provide the foundation for the learning code to be developed.

Using this Python acquisition code, the project's data store currently contains per-game statistics for every NCAA FBS and FCS team for all regular-season and bowl games from 2008 to 2011 and regular season games from 2012 for a total of approximately 15,000 data points.

2.1.1 Sample Form

Information stored in the database contain per-game team match-ups as well as each team's statistics from its previous game. To create a single sample based for each game, the project had to combine the teams' statistics in a reasonable manner. As such, each sample used for learning is the difference between the first team's statistics and the second team's statistics with an additional field for home/away (note that team order is taken from the NCAA website). Therefore, each sample has 18 fields and contains the following difference information:

- Rushing offensive yards
- Passing offensive yards
- Total offensive yards
- Offensive points scored
- Defensive rushing yards
- Defensive passing yards
- Total defensive yards
- Defensive points scored
- Net yards from punts
- Punt return yards
- Kick-off return yards
- Turnover vount
- Passing defense
- Pass efficiency
- Number of sacks
- Number of tackles
- Number of sacks allowed
- Home/away

2.1.2 Third-Party Libraries

While this project includes a custom averaged perceptron implementation, the project makes heavy use of existing libraries to support SVM and neural network development. For the SVM library, this project leverages David Soergel's jlibsvm library[2], a Java port of the venerable LibSVM[3]. To avoid reimplementing a neural network framework, this project makes use of the Java-based Encog machine learning framework[4]. Though Encog contains mechanisms for SVMs, hidden Markov models, and many other learning algorithms, only those APIs relating to neural networking were used.

2.2 Algorithms

For completeness, implementations details and parameters for each learning algorithm are included.

2.2.1 Averaged Perceptron

The implementation of the averaged perceptron contained herein follows the algorithm provided in Algorithm 3.6 on page 48 of the CIML textbook.[5] No tuning was performed on this algorithm as it does not use regularization and there are no other hyper parameters to set.

2.2.2 SVM Learner

Jlibsvm provides a significant number of configurable parameters for learning, but the documentation is rather lacking, so early stages of development were fraught with frustration in trying to identify a workable configuration that did not result in algorithm failure. Of particular interest to the problem herein are the various types of kernels and their parameters; this project primarily explored the differences between a linear kernel (i.e., no kernel) and the Guassian radial basis function (RBF) kernel. For parameters, the chief ones were minimum error ϵ, cost of misclassification C, and training example "influence" *gamma* (specific to the RBF kernel).

Finding a good combination of these parameters was not an easy task as early attempts resulted in jlibsvm's encountering its iteration threshold when attempting to minimize error. Fortunately, the library contains a parameter grid capability that allows the developer to specific a range of C values and a collection of kernels (each with its own parameter). In training, these ranges were set for $C \in \{2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$ and a set of kernels that included the linear kernel and the RBF kernel with ten different *gamma* values: $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000\}$. Jlibsvm trains across all of the combinations of parameters and provides the model with the best cross validation accuracy as the returned model.

2.2.3 Neural Network Learner

As mentioned, Encog provides a great deal of flexibility in the algorithms and parameters used with several different activation function implementations (hyperbolic tangent, sigmoid, linear, bipolar, and several others) and many different types and forms of networks (e.g., self-organizing maps to Elman networks). Owing to time constraints, this project was constrained to the standard feedforward network architecture with resilient backpropagation as the training algorithm. Structurally, the network was comprised of an input layer, two hidden layers (each layer with 18 nodes), and an output layer of one single node. Furthermore, each layer supported bias, and each node used the hyperbolic tangent as its activation function (necessary to support negative output values). For cutoff, the resilient backpropagation algorithm was allowed to run up to 60 minutes before halting.

[2] Available at http://dev.davidsoergel.com/trac/jlibsvm/

[3] Also available at http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[4] Available at http://www.heatonresearch.com/encog

[5] Hal Daume, III. 2012. A Course in Machine Learning (Pre-Release ed.). http://ciml.info

2.2.4 Ensemble Learner

Given that this project already contained three different implementations for predicting wins, it made sense to try and combine these learners in such a way so as to increase performance. To avoid long training times, however, the ensemble method used was a simple weighted average. That is, each learner was trained on 80% of the training data and then tested against the held-out tuning data (the other 20% of the training data). Accuracy across the learners was summed and weights were calculated for each learner according to Equation 1:

$$w_i = \frac{acc_i}{\sum_j acc_j} \tag{1}$$

During testing, each of the three learners would then classify a sample as -1 or $+1$, weight this result, and then sum the results together. The final classification would be made based on the sign of this sum.

3 Results

Overall, the results were rather unimpressive given the initial goals. Each learning algorithm's test set accuracy, precision, and recall score are presented in Table 1.

Algorithm	**Accuracy**	**Precision**	**Recall**
Average Perceptron	0.711803	0.710611	0.714632
SVM*	0.699272	0.686319	0.734034
Neural Network	0.659660	0.659660	0.659660
Ensemble	0.695635	0.694222	0.699272
ESPN-Ranking**	0.804511		

Table 1: This table shows some data

*The best SVM implementation used a C value of 8 and *gamma* of 0.00001.

** Predictions from the 2012 regular season using rankings from ESPN.com (only the top 25 teams each week).

4 Conclusion

While the results are not as high as one would like when compared to selecting a winner based solely on who has the highest rank from ESPN, the results are interesting in that they all seem to approach an asymptote. That is, there appears to be a limit very near 70% that a learning algorithm cannot surpass *given the data.* What this intuition suggests is that the limiting factor that prevents these algorithms from performing better is not in the algorithms or their applicability to the problem but rather in the data being used (which seems commensurate with the genetic algorithm results from a previous experiment on this data, results that also could not surpass around 70%). It may be the case that the variables governing approximately one-third of wins simply are not captured by the statistics being used.

Being limited by the statistics gathered seems perfectly reasonable in that one could use a large number of additional data. For instance, this project leverages only each team's previous game's statistics; why not use each team's average statistics prior to that game? Similarly, one could also include specific fields in the sample vector described above for conference (whether the team is FBS or FCS), historical win/loss ratio, number of injured first string players, and even go so far as to model specific player statistics.

Despite these limitations from data, there does exist additional latitude for more advanced learning procedures. Since each learning algorithm performs above 50%, one could leverage a boosting algorithm like AdaBoost to increase performance even more, though much of the infrastructure would need to be expanded to allow for weighted examples. Regardless of whether one wanted to pursue enhancing the learning algorithms *or* increasing the dimensionality of the data with more features, it is clear that this work provides at least a solid foundation for future improvement in addressing this problem.