Baseball Prediction
Using Ensemble Learning

by

Arlo Lyle

(Under the direction of Dr. Khaled Rasheed)

Abstract

As the salaries of baseball players continue to skyrocket and with the ever-increasing popularity of fantasy baseball, the desire for more accurate predictions of players' future performances is building both for baseball executives and baseball fans. While most existing work in performance prediction uses purely statistical methods, this thesis showcases research in combining multiple machine learning techniques to improve on current prediction systems by increasing the accuracy of projections in several key offensive statistical categories. By using the statistics of players from the past thirty years, the goal of this research is to more accurately learn from this data how a player's performance changes over time and apply this knowledge to predicting future performance. Results have shown that using machine learning techniques to predict a player's performance is comparable to the accuracy seen by some of the best prediction systems currently available.

Index words: Machine Learning, Ensemble Learning, Baseball Prediction, Model Trees, Artificial Neural Networks, Support Vector Machines, Bagging, Boosting, Stacking

Baseball Prediction
Using Ensemble Learning

by

Arlo Lyle

B.S., The University of Tulsa, 2005

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2007

Baseball Prediction
Using Ensemble Learning

by

Arlo Lyle

Approved:

Major Professor: Dr. Khaled Rasheed

Committee: Dr. Walter D. Potter
Dr. Donald Nute

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2007

Acknowledgments

I would like to thank my advisor Dr. Khaled Rasheed for his guidance throughout the duration of my research and thesis writing. I would like to thank Dr. Walter D. Potter and Dr. Donald Nute for being members of my committee and for their help throughout my two years at the University of Georgia. I would also like to thank Andy Walz. His vast knowledge of baseball helped to shape my rough idea into a thesis topic. He also directed me toward several freely available baseball statistic databases, without which this thesis could not have been completed. Lastly, I would like to thank my parents for their constant support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

During the mid 1970s to early 1980s two events occurred which sparked the current interest in baseball prediction: the advent of free agency in Major League Baseball and the invention of fantasy baseball. These two events made the advanced statistical analysis being done by a small group of people interesting for a much larger cross section of the general public. In the process, these events transformed the work being done by that group from a hobby into a profitable venture.

Until the repeal of the reserve clause in 1976, Major League baseball teams were able to hold control over their players paying them and doing with them as they pleased even after a player's contract had expired. Free agency created a market for players where teams now had to compete for the privilege of having a player on their team[3]. As a consequence salaries began to increase dramatically. In 1975, the year before free agency, the average salary of a professional baseball player was $44,676 and five years later in 1980 the average salary was $143,756, an increase of almost $100,000. By comparison, in the previous five year period from 1970 to 1975 the average salary increased by only about $15,000[10]. With such an increase in such a short period of time both the consequence of overpaying for a player and the reward of finding future stars were magnified.

A few years later in 1980 Dan Okrent and a group of his friends met at a restaurant called La Rotisserie Française. From the discussions between those friends at this restaurant evolved what is now called Rotisserie baseball, the most popular form of fantasy baseball. Briefly, Rotisserie baseball leagues consist of around ten teams comprised of actual Major

League Baseball players. Prior to the start of the season, a draft is held where team owners select the players who will be on their team. Throughout the season teams gain statistics such as home runs, runs batted in, wins, and so on based on the performances of their players in their real life games. At the end of the season the teams within the league are ranked based on how they performed in these various statistical categories. Often the winner of a fantasy baseball league receives a monetary prize, which provides extra incentive to win[23]. Whereas the drastic increase in player salaries only affected the small group of people in baseball front offices, fantasy baseball had the opportunity to reach the millions of baseball fans around the world. As more and more fantasy baseball leagues began popping up, the participants of those leagues wanted any advantage they could get when it came to determining which players would perform the best for their team.

Though the 1980s saw an explosion of popularity for the statistical analysis of baseball, it had been percolating for about a decade prior. In 1971, The Society for American Baseball Research (SABR) was founded. While SABR focuses on all forms of baseball research, the group has also fostered a new way of looking at baseball. This new way of looking at baseball can be epitomized by a single man: Bill James. James felt that existing statistics and convention led people to misjudge players and mismanage their teams. In turn he thought there was room for vast improvement by using statistical analysis and at times creating entirely new statistics, which more accurately captured what was occurring on the field of play. James called this new field of study sabermetrics. In 1977, he published his first of many *Baseball Abstracts*, tackling such issues as the inaccuracy of fielding statistics and how the speed of base runners changes with age[9].

Throughout the 1980s there were several occurrences that provided assistance to those interested in sabermetrics. First, companies like STATS Incorporated arose which collected extensive statistics on baseball games, including those developed by Bill James and others. With such a boon of statistical data the amount of research that could be done multiplied. Also around this time computers became more readily available to the average consumer,

which provided yet another advantage to the sabermetric researcher. Finally with the advent of the Internet, this statistical data became available worldwide, much of it free of charge. Through organizations like Retrosheet it is possible to download player statistics from 1871 to today and box scores from every game played since 1957 from anywhere in the world without spending a cent[23].

Today this has all come to culmination. In 2005 the average salary of a Major League Baseball player was $2,632,655[10]. On top of that outrageous figure, the disparity between the payrolls of the richest teams and poorest teams in baseball has become astronomical. In 2002 the New York Yankees had a payroll of $126 million, while the Oakland Athletics' payroll was a mere $40 million. Yet teams like the Athletics are still able to compete due to their ability to discover unknown and undervalued players. The documentation of this process in Michael Lewis' 2003 book *Moneyball*, sparked a revolution in the way owners and general managers did their jobs. Soon they were hiring sabermetricians as fast as they could in order to get a leg up on other teams[9].

On the same token, fantasy baseball has also been witnessing an explosion. The Internet has transformed fantasy baseball from a pastime played among friends to a worldwide phenomenon. Through services provided by Yahoo!, ESPN, and a multitude of others, participants can compete against strangers in any place reached by the Internet. In addition, fantasy baseball has sparked fantasy versions of other sports. A recent poll shows that some fifteen million Americans have played some form of fantasy sports. Along with this growth in popularity have come fantasy baseball websites like RotoWire and RotoWorld, and fantasy baseball magazines and books like *Baseball Prospectus* and *Ron Shandler's Baseball Forecaster*, both of which provide customers with detailed predictions of players' performances for the upcoming season[23].

1.2 Previous Work

While much work has been done over the past ten years in the area of baseball prediction, due to competition between companies and the importance of providing the best predictions, very little information is provided about how those companies actually calculate their predictions. Currently the most popular, and similarly accurate[23], baseball prediction systems are the Player Empirical Comparison and Optimization Test Algorithm (PECOTA), Diamond Mind Baseball, and Ron Shandler's Baseball Forecaster. Of the three only Baseball Prospectus, the creators of PECOTA, provides any information about how their predictions are calculated. Other slightly less accurate, and coincidentally noncommercial, systems include the Szymborski Projection System (ZiPS), Chone, and Marcels. In this thesis I chose to use PECOTA, ZiPS, and Marcels to compare against my prediction systems.

In 1998, Baseball Prospectus debuted their first projection system called Vladimir, which they used until the 2003 introduction of PECOTA. Though little information was provided about how Vladimir worked, it utilized artificial neural networks with various statistical categories as well as age and something called career path as inputs to predict the offensive performance of hitters. The statistics used were adjusted to account for the level of play of the player (minor leagues or major leagues) and various other factors such as park effects. The career path input was an attempt to manually group similar players into categories based on how they had performed in the past and would likely perform in the future[6][27]. At the time the standard projection system was developed by STATS Incorporated. In comparisons between the two systems using data from the 1998 season, Vladimir was unable to match the performance of the STATS system[2].

In 2003 Baseball Prospectus introduced PECOTA. Unlike Vladimir, PECOTA uses purely statistically methods to calculate predictions. As well, PECOTA generates predictions for both hitters and pitchers. To generate its predictions for a given player PECOTA uses the career paths of similar players from the past to determine how that player will perform in the future. To find players from the past that are comparable to the player in

question, PECOTA uses what is called a Similarity Score. The Similarity Score for two players is calculated by first starting with a perfect score of one hundred and subtracting based on differences between the players in several statistical categories. Then with a group of the maximally similar players, a player's future performance can be predicted by assuming that the player in question will continue to be similar to those other players as his career continues. How this is actually performed is not provided by Baseball Prospectus[7]. Baseball Prospectus continues to use PECOTA to this day and in a comparison with all other major current baseball prediction systems using data from the 2006 season, it performed the best when predicting the on base percentage plus slugging percentage of players with five hundred at bats or more[21].

Somewhat less accurate than PECOTA, the ZiPS system uses adapted versions of other prediction systems in its calculations. To predict the future performance of hitters, it uses a modified version of Bill James' Brock5 system[20]. Brock is a series of prediction systems developed by James for STATS Incorporated. One of the early versions of the system, Brock2, is available free of charge on the Internet. Brock2 predicts performance based on previous season statistics, age, and batting sustenance level using formulae developed by James[5].

Similar to the Brock prediction systems is Marcels. Meant as somewhat of a baseline for comparisons of other prediction systems, Marcels uses rather simplistic formulae to predict hitters' future performances. To make its predictions, Marcels uses the previous three seasons of a player's career, the players age, and various league averages[11].

In this thesis, I explored the use of machine learning techniques for baseball predictions as opposed to the purely statistical methods that currently dominate the landscape. While the idea of using machine learning to predict players' statistics is not a new one, as seen by the Vladimir system, the field of machine learning has advanced greatly since 1998. Along with neural networks, model trees and support vector machines, all of which are used in this thesis, have proven to work well for continuous prediction. In addition methods for combining multiple predictors such as bagging, boosting, and stacking have been shown to

increase performance over single predictors. Lastly I have drastically rethought the way the inputs to baseball prediction systems should be constructed. Instead of looking at a player's season totals for a given number of years in the past, I use totals from the player's previous one hundred sixty two games (the number of games in one season) along with a measure of how long it took the player to compete in those one hundred sixty two games. By using this method I believe I am able to more accurately capture a player's previous performance level in relation to other players.

1.3 Baseball Statistics

In this thesis I chose to predict the season totals for six offensive baseball statistics: runs, hits, doubles, triples, home runs, and runs batted in. For those unfamiliar with baseball, it is probably unclear what these statistics refer to. A player scores a run when he progresses along the base path to home plate either through scoring a home run or by being batted in by another player. A player earns a hit whenever he is able to hit the ball into play and reach base safely without the opposing team committing an error or by hitting a home run. A double is a hit where the player reaches second base and a triple is when the player reaches third base. A player earns a home run either by hitting the ball out of the ballpark in fair territory or by hitting the ball into play and reaching home plate safely. Lastly a player earns a run batted in when another player scores as a result of him putting the ball into play. If a player hits a home run, he also receives a run batted in for batting himself in. More in depth descriptions of the rules of baseball can be found on Major League Baseball's official website[13].

1.4 Goals Of This Thesis

This thesis aims to apply machine learning techniques to predict baseball player performance in six major offensive statistical categories: runs, hits, doubles, triples, home runs, and runs batted in. Towards that goal, I strive to create a baseball prediction system which performs

similarly well or better than existing systems, while using a novel approach in the creation of the inputs for my prediction system. More specifically the objectives of this thesis are:

1. Creating a simple, yet novel approach to extracting input data from available statistical databases.

2. Finding machine learning techniques appropriate for the domain of baseball prediction.

3. Implementing the proposed machine learning and ensemble learning techniques.

4. Evaluating the predictive accuracy of the models on the basis of their ability to predict unseen data not used in training.

5. Comparing the predictive accuracy of the models with existing baseball prediction systems.

Chapter 2

Machine Learning Techniques Used

Within this thesis I used several machine learning techniques to address the issue of baseball prediction. Since the domain being used involves continuous prediction as opposed to discrete classification, the number of eligible machine learning techniques was decreased. I decided to use three popular techniques that have been shown affective for continuous prediction problems: model trees, artificial neural networks, and support vector machines. In addition I employed three ensemble learning techniques, which allow for multiple machine learning models to be combined, often resulting in increased performance. The three ensemble learning techniques used were bagging, boosting, and stacking.

2.1 Base Predictors

2.1.1 Model Trees

Model trees are an extension of decision trees proposed by J. R. Quinlan which allow for continuous prediction. Decision trees are a popular machine learning technique which perform discrete classification using discrete and/or continuous inputs. As their name suggests, decision tree algorithms such as ID3 and C5 generate tree based models that can be transformed into human readable rules[12]. Using model trees, leaf nodes, which in decision trees would represent discrete classifications, represent linear regression functions, thus allowing for continuous values.

Quinlan introduced model trees by proposing the M5 algorithm[16]. As with decision trees, model trees are built using a divide-and-conquer scheme as follows. The set of training examples T is associated with a leaf in the tree or T is split into subsets using some test on

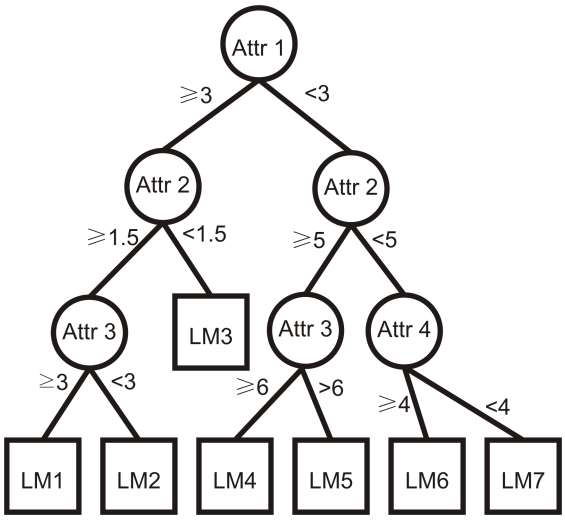

Figure 2.1: Model Tree

one of the attributes of the training examples. A split of the set T represents an inner node in the tree. This process is continued recursively until no more splits occur. The decision at any given time as whether to split the set of training examples or not is based on the standard deviation of the target values of the examples. Unless the set's size or standard deviation is low, a split is performed. Once it is determined that a split is necessary, how the split is performed must be decided. This is done by calculating the expected reduction of error of every potential attribute test. Where T_i is the ith subset of the training examples created by a given attribute test and $sd(T_i)$ is the standard deviation of the target values of the examples in T_i, the expected reduction in error of a given test can be written as:

$$\Delta error = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \ . \tag{2.1}$$

After calculating the expected reduction of error of every potential attribute test, the test which maximizes the expected reduction of error is chosen. Figure 2.1 is a sample model tree. The circular inner nodes represent attributes that are used in an attribute test, with the branches from those nodes representing the tests themselves. The square leaf nodes

represent the linear model used to predict instances which follow the given path through the tree.

Once the model tree has been constructed on the set of training examples, the target values of unseen instances can be classified by traversing the tree from the root node until a leaf node is reached. Once a leaf node is reached its associated linear function is used on the given instance to create a prediction. Though instead of using this value directly, a smoothing process is performed. This process is done by adjusting the predicted value based on intermediate predictions along the path from the root node to the given leaf node. Where N_i is the successor of N in the path from the root to the leaf, n_i is the number of training examples at N_i, $SP(N_i)$ is the smoothed prediction at N_i, $P(N)$ is the prediction at N, and k is a smoothing constant, the formula for smoothed prediction can be written as:

$$SP(N) = \frac{n_i \times SP(N_i) + k \times P(N)}{n_i + k} \ . \tag{2.2}$$

This formula is calculated recursively from the root to the leaf. Experiments have shown that by smoothing predictions, accuracy can be substantially increased.

In this thesis I used the M5' algorithm to generate model trees. The M5' algorithm was created by Yong Wang and Ian H. Witten[24] as an improvement over Quinlan's M5 algorithm. The main difference between M5' and M5 is the way M5' handles enumerated attributes and missing values. In the M5' algorithm, enumerated attributes are transformed into binary variables. This means that if a enumerated attribute has k possible values it will be transformed into k - *1* binary attributes. As a consequence all model trees generated by the M5' algorithm are binary. In order to take care of missing values, the formula for the expected reduction in error is modified in such a way as to favor attributes with fewer missing values. Once the attribute to split on has been determined, and if that attribute is numeric, only those training examples with the given attribute are used in determining the threshold for the attribute test. Training examples that are missing the given attribute are assigned to one of the two resulting sets based on how closely their target values match those of the resulting sets. During testing if an instance reaches a node in the tree corresponding

to an attribute that instance is missing the value of the attribute is determined either by averaging the values of that attribute in the training examples at that node if the attribute is numeric or choosing the subnode with the most training examples if the attribute is binary.

2.1.2 Neural Networks

Artificial neural networks (ANNs) are another popular method for continuous prediction. ANNs are a product of early artificial intelligence work aimed at modeling the inner workings of the human brain as a way of creating intelligent systems. Though many artificial intelligence researchers have steered in different directions, ANNs remain useful in many domains, including those which contain noise.

ANNs are made up of a collection of connected nodes which mimic the neurons within the brain. Each node in an ANN has a collection of associated input links which correspond to inputs to the ANN or outputs from another node in the network. Each link to a node has a corresponding weight which determines the importance of that link. In addition each node has another link with a fixed input of -1 which corresponds to a bias weight. All of these weights are initially set randomly and then appropriate values are learned based on some learning algorithm, as I will describe later. The node then uses some input function, generally summation, to calculate the input value to that node. Where n is the number of input links of a node, a_i is the input on link i, and W_i is the weight on link i the input function can be written as:

$$input = \sum_{i=0}^{n} W_i \times a_i \ . \tag{2.3}$$

After calculating the input to the node, this value is then passed to the activation function. One quite common activation function used in ANNs is the sigmoid function. Unlike some other activation functions such as the threshold function, the sigmoid function is continuous, which is important during the training of the ANN. When using the sigmoid function, the activation function can be written as:

$$activation = \frac{1}{1 + e^{-input}} \ . \tag{2.4}$$

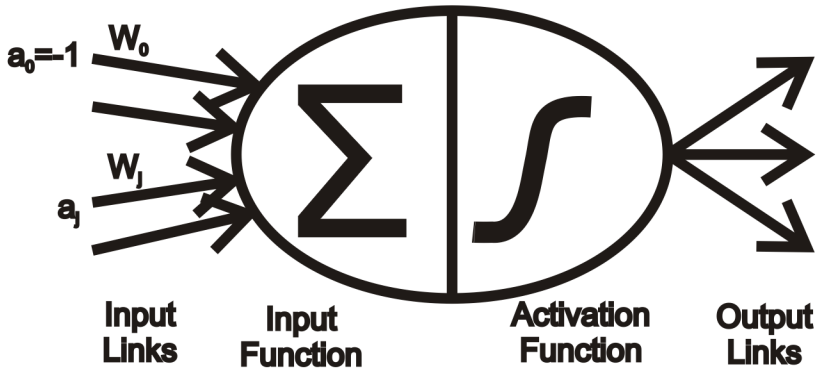

Figure 2.2: Artfificial Neural Network Node

Once the activation for a node has been calculated, it is sent via the node's output links to one or more other nodes in the network or if the node is an output node the activation is outputted from the ANN. Figure 2.2 shows a sample node within an ANN. As discussed previously the input links are passed to the input function, the summation function, the result of the input function is passed to the activation function, the sigmoid function, and lastly the result of the activation function is passed along the output links to other nodes in the ANN.

Now these nodes must be put together. There are two general types of ANNs which reflect the way nodes are connected to one another: acyclic and recurrent. Acyclic, or feed-forward, networks are very popular and get their name due to the fact that the resulting ANN does not contain any loops and values are fed forward from the input nodes to the output nodes. Recurrent networks on the other hand do contain loops within their structure which allows for a sort of short term memory of previous inputs to the network. While recurrent networks can quickly become complex to understand, they can be used as an interesting way to model the brain. Additionally ANNs can be fully connected. This means that every node in every layer of an ANN is linked to every node in the next layer of the ANN. While fully connected

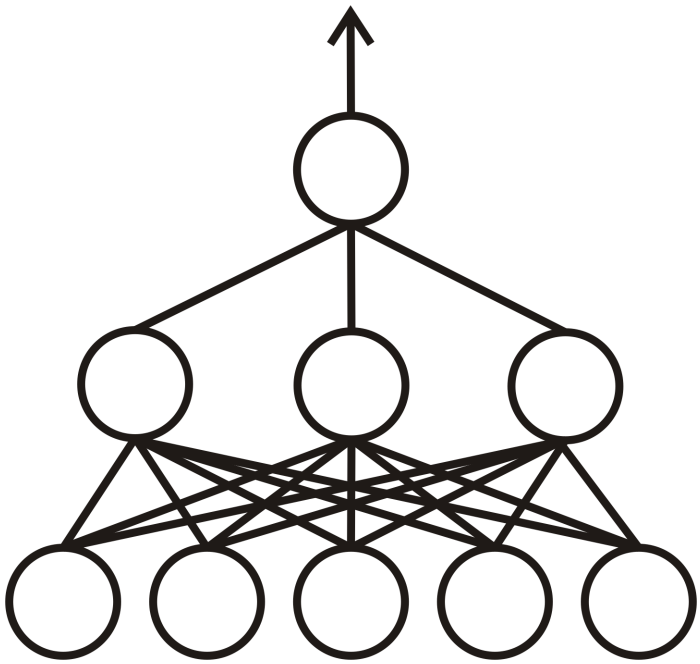

Figure 2.3: Fully Connected Neural Network

ANNs are quite popular, there are other useful designs which incorporate slabs within layers. Figure 2.3 shows a sample fully connected ANN with five input nodes, three hidden nodes, and a single output node. Lastly, ANNs can be characterized by the number of layers they have. Each ANN must have at least two layers, one for the inputs and one for the outputs. Additionally the network may contain one or more hidden layers. While the number of input and output nodes is generally fixed based on the domain at hand, the number of hidden layers and hidden nodes within those layers is up to the user, though generally a single hidden layer is the most popular choice. The number of hidden nodes to be used can vary greatly and depends on the problem at hand[18].

It still remains to learn appropriate values for the weights of the links within the ANN such that the ANN is able to provide reasonable outputs given inputs from the domain at hand. One of the most popular methods for training an ANN is trough the back-propagation

of error. The back-propagation algorithm works by using a set of training examples from the domain for which the correct output values are known. Then over a number of epochs the weights within the network are slowly tuned based on how incorrect the ANN is at predicting the output for each of the given training examples. Here an epoch refers to a single iteration of training on the set of training examples. For a given training example, first the output o_n generated by the network for every node n in the ANN is calculated. Then for each output node k its error on the given training example is calculated. Where t_k is the correct output, the error for an output node can be written as:

$$\delta_k = o_k \times (1 - o_k) \times (t_k - o_k). \tag{2.5}$$

Next, for each hidden node h its error on the given training example is calculated. Where w_{hk} is the weight on the link between the hidden node h and the output node k, the error for a hidden node can be written as:

$$\delta_h = o_h \times (1 - o_h) \times \sum_{k \in outputs} w_{hk} \times \delta_k \ . \tag{2.6}$$

Lastly each link within the network is updated. Where η is a value for learning rate, w_{ji} is the weight on the link from node j to node i, and x_{ji} is the input from node j to node i given the current training example, the formula for updating the weight for a link can be written as:

$$w_{ji} = w_{ji} + \eta \delta_j x_{ji}. \tag{2.7}$$

This process is repeated for every member of the set of training examples and then again for each epoch. In order to help prevent over-fitting on the training data, a validation set may be used in order to stop training before the specified number of epochs has elapsed. This is done by measuring performance during training on a validation set that is separate from the training and testing sets. If performance on the validation set begins to repeatedly decline, training will be halted[12].

2.1.3 Support Vector Machines

Support vector machines are a relatively new machine learning method for both discrete classification and continuous prediction developed by Vladimir Vapnik[22]. Support vector machines use the idea that concepts that are linearly separable are easy to learn. Yet not all domains are quite so simple. Support vector machines operate on the idea that by expanding the feature space of the domain to be learned the concepts involved may become linearly separable[25].

The first step in generating a support vector machine is determining the mapping to be used from the training examples to the new space. For example if a training set has two attributes, a_1 and a_2, examples can be mapped from two dimensional space to four dimensional space using the following polynomial of degree three:

$$w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3. \tag{2.8}$$

In this example w_1 through w_4 are weights that need to be learned. The number of weights involved can quickly grow out of hand. In this case with two attributes and a polynomial of degree three there are only four weights to learn, but in a case that has ten attributes and uses a polynomial of degree five the number of weights quickly jumps to over two thousand. Another problem arises in cases where the number of weights to be learned greatly exceeds the number of original attributes. In cases such as these it becomes very easy for over-fitting to occur due to the large number of parameters in the model.

Support vector machines address these two problems by incorporating what is called the maximum margin hyperplane. The maximum margin hyperplane is a hyperplane in the new space that provides the greatest separation between the classes involved. The maximum margin hyperplane can be found by finding the convex hulls of the classes involved. If the classes are linearly separable, then the convex hulls will not overlap. The maximum margin hyperplane is that which is orthogonal to the shortest line between the convex hulls and intersects it at its midpoint. The maximum margin hyperplane can be represented using

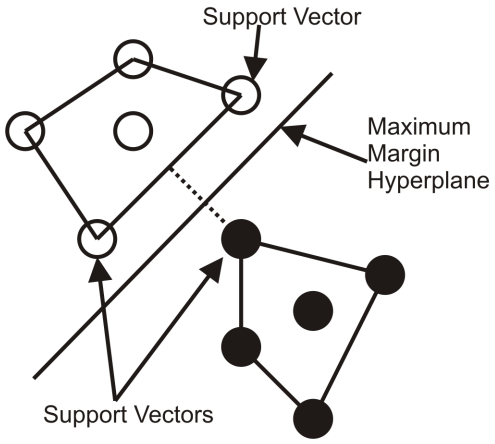

Figure 2.4: Maximum Margin Hyperplane

the training examples from the different classes that are closest to the maximum margin hyperplane. These particular training examples are called support vectors. Figure 2.4 shows how this would look in a two dimensional case. The empty circles and filled circles represent two different classes. Convex hulls have been drawn around the two classes and a dotted line has been drawn to represent the shortest line between the two convex hulls. As can be seen, the maximum margin hyperplane intersects the dotted line perpendicularly at its midpoint. In this case there are three support vectors that can be used to represent the maximum margin hyperplane. Using the support vectors in a given case the maximum margin hyperplane can be written very simply as:

$$x = b + \sum_{i \in support\ vectors} \alpha_i y_i (\mathbf{a_i} \cdot \mathbf{a})^n. \tag{2.9}$$

In this equation x is the classification given by the support vector machine, b and α_i are weights to be learned, y_i is the class of the given support vector, which for binary classification is either 1 or -1, $\mathbf{a_i}$ is the given support vector, $\mathbf{a}$ is a member of the test set in vector form,

and n is an exponent which represents new space onto which the original attributes are mapped, this value can be found experimentally[25].

The problem of finding the support vectors and learning the weight b and α_i belongs to a class of optimization problems called constrained quadratic optimization for which there exist many methods in order to solve this type of problem. In my research, I used a method designed specifically for training support vector machines called sequential minimal optimization. Sequential minimal optimization improves on existing methods for training support vector machines by simplifying the computations involved and thus speeding up the process a great deal[15].

2.2 Ensemble Learning

In every day life it is often found that by combining the opinions, judgments, or conjectures of a group of experts a more accurate picture of a situation can be outlined than if only a single expert were used. The same has been shown to hold true for machine learning. If multiple base classifiers are combined, it has often been found that the group is more accurate than the individual, though improvement is not guaranteed. For this thesis I have decided to use three general ensemble learning techniques in hopes of generating more accurate predictions as compared to the predictions generated by the base classifiers.

2.2.1 Bagging

The most simplistic of the three ensemble learning methods I used was bagging. Bagging was developed by Leo Breiman[1] as a way to aggregate predictors in order to increase predictive accuracy. The key to bagging is to create many significantly different versions of the same classifier all trained on the same training data. In order to accomplish this goal, separate training sets are generated for each classifier that is to be constructed. Each training set is generating through a process of sampling from the entire set of training examples. Examples are chosen at random with replacement. This means that examples may be repeated or left

out of a resulting training set. By completing this process, classifiers are constructed which all were trained on information about the same domain, but which usually have slightly different models and give different results given the same input.

The next step in the process of bagging is to combine all the predictions provided by the individual classifiers into a single prediction. In many cases the ultimate prediction is generated by a simple vote. In the case of discrete classification a simple plurality may be used to determine the result to be outputted, whereas in the case of continuous prediction the average of all the individual predictions may be used. Another method often used with continuous prediction, which is a bit more complicated, is a weighted average. In the weighted average, the weight which is assigned to an individual classifier corresponds to its accuracy on the training set. In order to calculate a classifier's weight, the classifier is used to predict the target output of each member of the entire training set, even though it may not have been trained on some of the members of the training set. This is done in order to get a measurement on which the accuracy of the classifiers can be compared. Once a classifier's total error has been determined, a weight is assigned to the classifier which is inversely proportionate to its error. Thus, classifiers with high error will get low weight, and vice versa[25].

Once all the weights have been calculated, predictions can be made on unseen examples using the following equation for a weighted average:

$$prediction = \frac{\sum_{i}^{n} w_i p_i}{\sum_{i}^{n} w_i} \ . \tag{2.10}$$

In this formula n is the number of classifiers, w_i is the weight of the ith classifier, and p_i is the prediction provided by the ith classifier. While this process of bagging has been shown to improve performance in many cases, especially those cases were the individual classifiers generated are quite different, improvement is not guaranteed. In fact it has been proven that situations exists were bagging decreases performance. How well bagging improves performance has to do with many factors, including the type of classifier being used in

bagging. Some classifiers, though using different samplings of the training set, may not create models which are sufficiently different for bagging to perform at its best[25].

2.2.2 Boosting

As mentioned in the previous section, bagging only works well when the classifier used is able to transform small differences in training sets into significant differences in the models generated. Ideally the goal is to generate models which compliment the predictive power of one another. Boosting was developed by Yoav Freund and Robert E. Schapire[4] as a way of iteratively creating models which compliment those that have been created previously. Boosting is similar to bagging in that it uses only one type of base classifier, but instead of relying on a uniform random selection to set the training sets of classifiers, the training sets are based on the strengths and weaknesses of previously constructed classifiers.

The key difference between boosting and bagging is the addition of a weight to each of the examples in the training set. Initially all the weights are set to one, thus each training example is given equal importance to begin with. At this point it is time to generate the first classifier. If the type of classifier being used is able to accept weighted training examples, as some are, then all the training examples are used during training. If the weights cannot be used directly, then a probabilistic sampling is taken where a higher weight corresponds to a higher probability of being selected. Generally training sets are chosen to be the same size as the number of training examples, thus examples may be left out or repeated.

Once the model has been generated its accuracy is calculated on the entire training set. The classifier's total error on the training set is used to adjust the weights of the training examples. Once the total error on the training set is calculated, all the weights of correctly classified training examples are updated using the following equation:

$$w' = w \times \frac{e}{1-e} \ . \tag{2.11}$$

In this formula w' is the new weight, w is the current weight, and e is the total error. It should also be noted that e must be less than one half or weights will will be increased

instead of decreased as is intended. Once all the weights of correctly classified examples have been updated, all the weights are normalized. Normalizing the weights makes sure that not only is the importance of correctly classified examples decreased but also the importance of incorrectly classified examples is increased. This is the key to the idea of making complimentary models. By decreasing the importance of examples which are already correctly classified, harder examples which have not been correctly classified can get more of the focus[25].

Generally in classification problems, the process of boosting is terminated when the total error exceeds one half, though in classification domains with many classes a loss function is used since it may be hard to create classifiers that have error less than one half. Once the boosting process has been completed all the models that have been created must be combined to generate a single prediction. This process is done much the same way as it is in bagging, with a weighted vote.

In many cases boosting is not used in continuous prediction problems due to issues involving how to update the weights of training examples and how to alter the termination conditions, because the total error is no longer a measure of how many training examples were incorrectly classified and cannot be used in the same way as before. In my research, I used a methodology for applying the process of boosting to continuous prediction that involves discretizing the results on continuous prediction. This in done by introducing a threshold value. If the error for a given prediction is within the threshold, the prediction is considered correct, otherwise it is considered an incorrect prediction. Through this process the total error again represents the number of incorrectly predicted instances and thus can be used in the weight update function mentioned previously and for the termination conditions.

2.2.3 Stacking

Stacking differs from both bagging and boosting in that bagging and boosting deal with only one type of classifier, whereas stacking combines multiple types of classifiers into a single output. Stacking, also referred to as stacked generalization, was developed by David

H. Wolpert[26] as a way to combine base classifiers of different types in order to improve performance over the individual classifiers. Instead of combining the classifiers' outputs the same way each time using some sort of vote, the goal of stacking is to learn how the base classifiers should be combined given a certain input.

The first step in stacking is to train the individual base classifiers or level-0 classifiers. Unlike bagging and boosting where differences between the models generated are manufactured by using different training sets for each classifier, the differences between the models generated is taken care of based on the different types of classifiers being used. For this reason all the classifiers can be trained using the same training set. Next a different set of examples is used to generate a training set of the level-1 classifier. The goal of the level-1 classifier is to take the outputs given by the level-0 classifiers based on some example and classify the original example based on the level-0 outputs.

In many cases not enough examples exist in a domain such that separate training sets can be used for level-0 and level-1 classifiers. For this reason cross-validation is often used on a solitary training set. By using cross-validation it is possible to split the training set in such a way that over a number of folds a training set for the level-1 classifier can be generated which is the same size as the training set for the level-0 classifiers. In each fold a fraction of the training set is chosen to be set aside. Each level-0 classifier is training on the remaining section of the training set. When the level-0 classifiers have been trained, they are used to classify the examples that were set aside. The outputs for all the level-0 classifiers for a given example represent a single training example for the level-1 classifier. In subsequent folds, other parts of the training set are set aside until all members of the training set have been used to generate training examples for the level-1 classifier.

Now the level-1 classifier, generally a relatively simple classifier, is trained on its examples. At this point, the level-0 classifiers are often retrained on the complete training set in order to make them as accurate as possible. Now unseen instances may be classified. The process occurs in two steps. First the unseen instance is classified by each of the level-0 classifiers,

then the outputs from level-0 classifiers are sent to the level-1 classifier which provides the ultimate classification based on those outputs. Though only discrete classification has been discussed to this point, stacking also allows for continuous prediction depending on the classifiers used. In any case the inputs of the problem must match the acceptable inputs of every level-0 classifier, the outputs of the level-0 classifiers much match the inputs of the level-1 classifier, and lastly the outputs of the level-1 classifier must match the outputs of the problem[25].

Chapter 3

Implementation

3.1 Inputs

3.1.1 Format

The first step in the implementation of the baseball prediction systems used in my research was to determine what inputs were to be used and how those inputs would be constructed. One major influence in my selection of the inputs to be used was the availability of data. In constructing the inputs, I wanted to use the lowest possible level of abstraction. Most existing baseball prediction systems use season totals as inputs, but I felt that this approach led to inconsistent instances. Season totals can represent a player who had seven hundred plate appearances in one hundred sixty games or it could represent a player who had fifty plate appearances in thirty games. Ideally I would have liked to use totals over a set number of plate appearances prior to the season to be predicted, but this data was not readily available. Instead I decided to use a player's totals over his one hundred sixty two games prior to the season being predicted. This game by game data was freely available on the Internet through both Retrosheet[17] and the Day By Day Database at Baseball Musings[14].

After determining how I would construct the inputs I would use for my baseball prediction systems, I needed to determine which statistical categories would be used. Again I decided to depart from standard convention. Ultimately, I decided to use only seven statistics in my set of inputs: at bats, runs, hits, doubles, triples, home runs, and runs batted in. Instead of using derived statistics such as batting average and on base percentage, I used the more basic statistics from which most of the common derived statistics can be calculated.

In addition to the seven statistical totals used, I also used two other inputs. The first of these additional inputs was the player's age at the start of the season to be predicted. Since a player's performance generally increases up to a certain age and declines afterward, I felt that age would play an important role in predicting future performance. The second additional input used was the number of days that elapsed during the time it took the player to play the one hundred sixty two games used to calculate the statistical totals mentioned earlier. I felt that by including this figure I could capture some element of how injury prone a player is. In a scheme that relies on season totals, a player who is often injured may be indistinguishable from a bad player, for which there is no straight forward solution. Whereas in my scheme an injury prone player and another player of similar ability who played all one hundred sixty two games of the previous season may have similar statistical totals, but my new input takes care of distinguishing between the two.

The nine inputs used in my research and the method in which they were constructed represents a rather novel approach to baseball prediction. Besides the differences already mentioned, some existing baseball prediction systems make adjustments on their data. Common adjustments include modifying statistics to take into account effects on performance attributed to differences in ballparks and changes in the level of league wide performance over time. Much of the data used to make these types of adjustments is not readily available, thus no adjustments were made to the statistics used in my research. Additionally some prediction systems have used inputs which represent an attempt to manually cluster players into groups based on their previous performance. One of the goals of the sabermetric movement is to remove subjectivity and misconception from baseball conversation. Towards this aim, one of the objectives of my research was to use as little human input as possible and let the statistics speak for themselves.

3.1.2 Instances

As mentioned earlier the game by game data needed to construct instances of the form previously described were freely available on the Internet trough Retrosheet and Baseball Musings. In addition I retrieved the season totals to be used as target attributes from Sean Lahman's Baseball Archive[8] and MLB.com[19]. The first step in constructing the instances used in my research was determining what years to use statistics from. I wanted to select a time period which would provide a large number of instances, but represented a relatively small change in average league wide performance. The time period I settled on was 1973 to 2006, with the totals from the 2006 season used as the targets for my testing set. I chose to start with the 1973 season, because it marks the first year designated hitters were used in the American League and is considered by some as the start of the modern era of baseball.

Instances were constructed by first creating a list of all the player and season combinations which represented an instance of a player having played one hundred sixty two games prior to the given season. For a given player and season combination a single instance with fifteen values was constructed with the nine values representing the inputs described earlier and the last six representing the target season totals to be predicted. The statistical categories used for the target season totals as mentioned before were runs, hits, doubles, triples, home runs, and runs batted in.

While the number of player and season combinations between the 1973 and 2006 seasons where the player played one hundred sixty two games prior to the given season was near ten thousand, due to time constraints and the time involved in manually inputting much of the data used in my research only two thousand one hundred fifty one training instances were used. The number of testing instances used was three hundred thirty, which is the total number of players who played in one hundred sixty two games prior to the 2006 season and whose 2006 season totals were predicted by the three existing prediction systems used as comparisons.

3.2 Approach

3.2.1 Weka

All of the various machine learning techniques used in my research were implemented in Java using the Weka 3.4.7 package[25]. Weka is a collection of machine learning algorithms developed at the University of Waikato in New Zealand. The machine learning algorithms provided in Weka can be accessed through a graphical user interface (GUI), command line operations, or a Java application programming interface (API). In my research I decided to use Weka via the API. Since the GUI itself is built using the API, the API is capable of performing all the functions of the GUI with the exception of the visualization features which allow users to view their results in various manners. The major advantage of using the API versus the GUI is increased control and the ability to use the machine learning algorithms implemented by Weka in other applications. As well the ability to automate parameter tuning by using the API was a key advantage.

3.2.2 Implementation Details

The basic implementation of all the machine learning techniques used in my research involved using the nine inputs previously discussed and one output. Though artificial neural networks allow for multiple outputs, model trees and support vector machines do not. For this reason, I made the decision to create separate predictors for each of the six outputs to be predicted. In order to create a baseball prediction system which provides predictions for all six of the outputs, the individual predictors of a given machine learning algorithm can be combined into a single program with relative ease.

Model Trees

The implementation for my baseball predictors using model trees was quite simple due to the small number of settings available through Weka. The two major settings provided by

Weka for implementing model trees are options to turn off pruning and smoothing. Based on trials using the various combinations of these two settings, I found that performance was increased by using both pruning and smoothing in all instances.

Artificial Neural Networks

Artificial neural networks provided by far the largest number of parameters to be set among the machine learning techniques used in this thesis. To begin with I decided to use a fully connected neural network with a single hidden layer. I chose this configuration based on it being the standard configuration used in neural network research. Beyond this the three key settings for my neural networks were the number of hidden units, learning rate, and momentum. The first thing I noticed was that the optimal values for these settings varied over the six statistical categories I was predicting. To determine the optimal values for these parameters, I created a program which looped through a range of momentum values for each learning rate value in a given range for a given range of hidden units. For each set of values thirty neural networks were created using random seeds for the the initialization of weights. The errors of these thirty neural networks were then averaged. Through this process I found that the optimal number of hidden units for the six statistical categories ranged from one for home runs to eleven for doubles. Whereas the range of variation in values for learning rate and momentum was quite small with the optimal learning rate being five hundredths and the optimal momentum being zero (momentum turned off) for most of the six statistical categories.

One result of this process of parameter tuning was discovering the wide range of errors possible due to changes in the seed used to initialize the weights of the neural network. Weka does not allow for the ability to set the range of initial weights. For this reason the starting state of the neural network is quite variable. Due to this issue, I decided to treat the random seed value as a parameter. In order to determine the optimal random seed, I used the optimal parameters already discovered and created one hundred neural networks with different seeds

for each of the six statistical categories. I used the random seed that resulted in the lowest error.

Additional settings used in all my neural network implementations included normalizing the inputs and outputs of the neural networks as well as using part of the training set as a validation set. I decided to use Weka's default validation set size of ten percent of the training set size (two hundred fifteen instances) and a validation threshold of ten, meaning that if error on the validation set increased ten straight epochs, training was terminated.

Support Vector Machines

For my implementations of support vector machines there was only a single key parameter to be tuned. This parameter was the exponent of the maximum margin hyperplane equation discussed in the previous chapter. As with my neural network implementations, I found that the optimal value for this parameter varied over the six statistical categories being predicted, though since only one parameter was involved tuning proved to be much simpler. I simply created a number of support vector machines over a range of exponent values for each of the six statistical categories and used the value which corresponded to the smallest error. The scope of optimal exponents was rather small, ranging from two to two and four tenths.

Bagging

While Weka provides an implementation of bagging, I decided to implement the process myself in Java while still using Weka to create the base predictors used in the bagging process. I decided to implement bagging myself because the process is relatively simple and in order to have complete control of the process. In my research I created three different baggers corresponding to the three different base predictors used. For all of the base predictors used for bagging I used the parameters previously discussed. To determine the number of predictors to be bagged I performed trials with various numbers of predictors. I settled on using a value of one hundred base predictors for each of the three baggers created. While

error was decreased by using more than one hundred base predictors, the decrease was quite small in relation to the large increase in runtime. Each of the base predictors was trained using a uniformly random sampling of the training set with replacement. The sampled set size was set to the same as the entire training set size (two thousand one hundred fifty one instances). This insured that the predictors generated would represent different models of the domain and provide different opinions for unseen instances. Lastly, since the process of bagging is nondeterministic, I constructed five baggers for each base predictor and statistical category combination and used an average of the resulting evaluation metrics.

Boosting

Weka provides several implementations of boosting, though none of them are for continuous prediction, thus I had to develop my own implementation using the discretization process mentioned earlier. As with bagging I implemented boosting in Java, using Weka to create the base classifiers. I created three different boosters corresponding to the three different base predictors used. For all of the base predictors used for boosting I used the parameters previously discussed. Since the process of boosting allows for termination conditions, the number of base classifiers used did not need to be determined as was the case for bagging. Though I did need to determine the threshold values to be used in the discretization process. To determine what thresholds to use I performed trials using various threshold values. I began with the lowest error generated by the existing baseball prediction systems for each of the six predicted baseball statistics and went from there. I found that error was minimized when using threshold values that where slightly higher than the lowest error of the existing prediction systems. Lastly, as with bagging, the process of boosting is nondeterministic. For this reason I constructed five boosters for each base predictor and statistical category combination and used an average of the resulting evaluation metrics.

Stacking

As is the case for bagging, Weka provides an implementation of stacking, but I decided to implement the process myself in Java, while still using Weka to create the level-0 and level-1 predictors. I made this decision for much the same reasons I did for bagging. In my research I created three different implementations of stacking. For each implementation the level-0 predictors corresponded to the three different base predictors used. The three different implementations correspond to the three different level-1 predictors used. It has been shown that in general relatively simple machine learning algorithms can be used as level-1 predictors while still generating a high level of performance. For this reason I made the decision not to use artificial neural networks as level-1 predictor. Instead I used linear regression, model trees, and support vector machines as my three different level-1 predictors. As mentioned in the previous chapter, a process of cross-validation is often used to generate a full set of instances for the level-1 predictors. In my implementation of stacking I decided upon using ten fold cross-validation for this process.

For all of the level-0 predictors used for stacking I used the parameters previously discussed, though since the level-1 predictors were being trained on different data and linear regression had not been previously used, the parameters for the level-1 predictors needed to be tuned. The implementation of linear regression provided by Weka does not have parameters, thus no tuning was performed. For model trees I again settled on using both pruning and smoothing. For support vector machines the scope of optimal exponents for the maximum margin hyperplane equation ranged from one and one tenth to one and a half. This in and of itself shows that simpler predictors are needed as level-1 predictors.

Chapter 4

Evaluation and Results

4.1 Evaluation Metrics

In order to provide comparisons both between the individual baseball prediction systems I generated as well as between my prediction systems and existing prediction systems, I had to decide on a set of evaluation metrics to use. I decided on mean absolute error, root mean squared error, and correlation coefficient. While correlation coefficient is one of the most commonly used metrics when comparing baseball prediction systems, I felt that mean absolute error and root mean squared error could provide extra insight into the performance of the various systems.

Mean absolute error is one of the easiest performance measures to calculate as well the most straight forward to understand. This measure simply reports on average how wrong a given predictor is. Where n is the number of test instances, p_i is the predicted target value for a given instance, and a_i is the actual target value for the instance, the equation for mean absolute error can be written as:

$$MAE = \frac{\sum_{i=0}^{n} |p_i - a_i|}{n}. \tag{4.1}$$

One advantage of using mean absolute error is that it treats each error equally, whereas other measures can be greatly affected by only a few outliers with large error.

Root mean squared error is a bit more complicated performance measure both in calculation and in interpretation. As its name suggests the root mean squared error formula involves squaring the error involved in predicting the target values of test instances. This process of squaring error makes large individual errors quite costly. Using the same variables

as in the mean absolute error equation, the root mean squared error equation can be written as:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n} (p_i - a_i)}{n}}. \tag{4.2}$$

Used in tandem, mean absolute error and root mean squared error can tell a lot about predictors. While two predictors may have similar mean absolute errors, if one has a much higher root mean squared error, this means it is less consistent in its prediction accuracy.

Correlation coefficient is by far the most odd of the commonly used evaluation metrics. For one, unlike most metrics where the goal is to decrease the value of the metric, higher values for correlation coefficient are better. Additionally correlation coefficient is independent of scale, meaning that correlation coefficient is not affected if a given set of predictions are multiplied by a constant factor. Using the same variables as the previous two equations and where $\bar{p}$ is the average of the n predictions, $\bar{a}$ is the average of the n actual target values, s_p is the standard deviation of the n preditions, and s_a is the standard deviation of the n actual target values the formula for correlation coefficient can be written as:

$$r = \frac{\sum_{i=0}^{n} (p_i - \bar{p})(a_i - \bar{a})}{(n-1)s_p s_a}. \tag{4.3}$$

As its name suggests the correlation coefficient measures the statistical correlation between two sets of numbers. The values for correlation coefficient range from -1 to 1 with 1 corresponding to perfect positive correlation, 0 corresponding to no correlation, and -1 corresponding to perfect negative correlation[25].

4.2 Evaluation Conditions

As mentioned previously I was able to generate two thousand one hundred fifty one training instances based on actual statistics from the 1973 through 2005 seasons and three hundred thirty testing instances from the 2006 season. Each of the baseball prediction systems I created was trained using the same set of training instances and tested used the same set

of testing instances. Additionally the evaluation results for the three existing prediction systems used for comparison were calculated using the same three hundred thirty players' season totals for the 2006 season as the set of testing instances. All of my prediction systems accepted the same nine input attributes discussed earlier. For the three base predictors and the three implementations of stacking the systems generated were deterministic, thus the results provided represent a single run. For all the implementations of bagging and boosting the systems generated were nondeterministic. For this reason the results provided represent the average of five runs.

4.3 Prediction Results

As mentioned previously I decided to use three different existing baseball prediction systems for comparison with my prediction systems. Those systems are PECOTA developed by Baseball Prospectus, ZiPS, and Marcels. These three prediction systems provide a good sampling of the scope of existing systems, thus providing a way of placing the performance of my prediction systems in context.

The first of the six statistical categories that I predicted was runs. Table 4.1 shows a comparison between all of the predictors I constructed. In bold I have highlighted the values which represent the best performing predictor for the given evaluation metric. As can be seen artificial neural networks generated the best mean absolute error, stacking using a support vector machine as the level-1 predictor generated the best root mean square error, and bagging using model trees generated the best correlation coefficient. Table 4.2 shows a comparison between these best values and the performance of the three existing baseball prediction systems. As can be seen Baseball Prospectus' PECOTA not only outperforms Marcels and ZiPS, but also the best values of my predictors. While PECOTA outperforms my predictors, the margin is relatively small. In addition the best performances of my predictors outperform ZiPS and Marcels for mean absolute error and root mean squared error and correlation coefficient for ZiPS.

Table 4.1: Runs prediction results.

	MT	**NN**	**SVM**	**Bag MT**	**Bag NN**	**Bag SVM**
MAE	19.8	**19.1**	19.3	19.5	19.5	19.3
RMSE	25.2	24.7	24.7	24.9	24.8	24.8
r	.716	.716	.721	**.725**	.722	.720
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	19.5	19.6	19.7	19.8	19.5	19.2
RMSE	25.0	25.0	25.6	25.2	24.8	**24.5**
r	.715	.718	.701	.716	.722	.723

Table 4.2: Runs comparison to existing systems.

	My Best	**BP**	**Z**	**M**
MAE	19.1	**18.7**	20.7	21.6
RMSE	24.5	**24.2**	27.0	26.2
r	.725	**.738**	.700	.736

Table 4.3: Hits prediction results.

	MT	**NN**	**SVM**	**Bag MT**	**Bag NN**	**Bag SVM**
MAE	35.4	35.5	35.3	**34.7**	35.2	35.3
RMSE	45.5	**44.6**	46.1	44.8	45.2	46.0
r	.683	.690	.676	**.696**	.690	.677
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	35.3	34.9	35.9	35.9	35.4	34.8
RMSE	46.3	45.7	47.4	45.7	45.4	45.2
r	.683	.686	.669	.690	.689	.692

Table 4.4: Hits comparison to existing systems.

	My Best	**BP**	**Z**	**M**
MAE	**34.7**	NA	37.7	38.3
RMSE	**44.6**	NA	50.2	47.6
r	.696	NA	.677	**.717**

Table 4.5: Doubles prediction results.

	MT	**NN**	**SVM**	**Bag MT**	**Bag NN**	**Bag SVM**
MAE	8.40	8.45	8.43	8.40	8.46	8.43
RMSE	10.5	10.4	10.5	10.5	10.6	10.5
r	.620	.626	.618	**.630**	.619	.620
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	8.47	8.45	8.49	8.52	8.48	**8.39**
RMSE	10.6	10.7	10.6	10.6	10.6	**10.4**
r	.615	.614	.609	.616	.620	.625

Table 4.6: Doubles comparison to existing systems.

	My Best	**BP**	**Z**	**M**
MAE	8.39	**8.01**	9.39	9.05
RMSE	10.4	**10.0**	11.7	11.0
r	.630	**.664**	.584	.637

The second statistic predicted was hits. Table 4.3 shows a comparison between all of the predictors I constructed. As can be seen bagging model trees generated both the best mean absolute error and correlation coefficient and artificial neural networks generated the best root mean squared error. PECOTA does not perform predictions for hits, thus Table 4.4 only shows a comparison between these best values and the performances of ZiPS and Marcels and the averages for those two systems. As with runs, my predictors outperform both ZiPS and Marcels in mean absolute error and root mean squared error, but only ZiPS in correlation coefficient.

Table 4.7: Triples prediction results.

	MT	**NN**	**SVM**	**Bag MT**	**Bag NN**	**Bag SVM**
MAE	1.41	1.30	1.30	1.41	1.36	1.31
RMSE	1.96	1.99	1.89	1.95	1.94	1.89
r	.661	.687	.698	.666	.679	.695
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	1.37	1.35	1.30	1.41	1.40	**1.27**
RMSE	2.07	2.03	1.98	1.89	1.90	**1.86**
r	.626	.648	.677	.697	.688	**.706**

Table 4.8: Triples comparison to existing systems.

	My Best	**BP**	**Z**	**M**
MAE	**1.27**	1.39	1.36	1.47
RMSE	**1.86**	1.97	1.93	1.95
r	**.706**	.695	.693	.682

The next statistic predicted was doubles. Table 4.5 shows a comparison between all of the predictors I constructed. As can be seen stacking using a support vector machine as the level-1 predictor generated the best mean absolute error and root mean squared error and bagging model trees provided the best correlation coefficient. Table 4.6 shows a comparison between these best values and the performance of the existing prediction systems. As can be seen PECOTA outperforms Marcels and ZiPS as well as my predictors, but as with runs and hits my predictors outperform the latter two in mean absolute error and root mean squared error, but only ZiPS in correlation coefficient.

Table 4.9: Home runs prediction results.

	MT	NN	SVM	Bag MT	Bag NN	Bag SVM
MAE	5.85	**5.56**	5.57	5.73	5.69	5.60
RMSE	7.76	7.53	**7.51**	7.66	7.62	7.54
r	.744	.755	**.756**	.752	.753	.754
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	**5.56**	5.64	5.59	5.85	5.84	5.62
RMSE	7.53	7.58	7.60	7.76	7.76	7.58
r	.755	.753	.751	.744	.744	.752

Table 4.10: Home runs comparison to existing systems.

	My Best	BP	Z	M
MAE	5.56	**5.50**	5.98	6.26
RMSE	7.51	**7.50**	7.87	7.90
r	.756	**.761**	.757	.749

The next statistic predicted was triples. Table 4.7 shows a comparison between all of the predictors I constructed. As can be seen stacking using a support vector machine as the level-1 predictor generated the best mean absolute error, root mean squared error and correlation coefficient. Table 4.8 shows a comparison between these best values and the performance of the existing prediction systems. As can be seen ZiPS outperforms Marcels and PECOTA, but all three are outperformed by my predictor using stacking with a support vector machine as the level-1 predictor.

The next statistic predicted was home runs. Table 4.9 shows a comparison between all of the predictors I constructed. As can be seen both artificial neural networks and boosting

Table 4.11: Runs batted in prediction results.

	MT	**NN**	**SVM**	**Bag MT**	**Bag NN**	**Bag SVM**
MAE	20.0	**19.5**	**19.5**	19.8	19.8	**19.5**
RMSE	25.6	25.0	25.1	25.4	25.4	25.1
r	.703	.709	.710	**.711**	**.711**	.710
	Boost MT	**Boost NN**	**Boost SVM**	**Stack LR**	**Stack MT**	**Stack SVM**
MAE	19.6	**19.5**	**19.5**	19.7	**19.5**	**19.5**
RMSE	25.1	25.1	25.1	25.1	25.0	**24.8**
r	.710	.710	.708	.703	.708	**.711**

Table 4.12: Runs batted in comparison to existing systems.

	My Best	**BP**	**Z**	**M**
MAE	19.5	**19.0**	21.4	21.4
RMSE	24.8	**24.4**	27.8	26.5
r	.711	**.727**	.709	.709

model trees generated the best mean absolute error and support vector machines generated the best root mean squared error and correlation coefficient. Table 4.10 shows a comparison between these best values and the performance of the existing prediction systems. As can be seen PECOTA outperforms Marcels and ZiPS, as well as my predictors, but my best values for the evaluation metrics outperform the latter two in mean absolute error and root mean squared error, but only ZiPS in correlation coefficient.

The last statistic predicted was runs batted in. Table 4.11 shows a comparison between all of the predictors I constructed. As can be seen artificial neural networks, support vector machines, bagging support vector machines, boosting both artificial neural networks and

Table 4.13: Percent differences of my best predictors and best existing predictors.

	runs	**hits**	**doubles**	**triples**	**home runs**	**runs batted in**
MAE	2.14%	**-7.96%**	4.74%	**-6.62%**	1.09%	2.63%
RMSE	1.24%	**-6.30%**	4.00%	**-3.63%**	0.13%	1.64%
r	-1.76%	-2.93%	-5.12%	**1.88%**	-0.66%	-2.20%

support vector machines and stacking using both a model tree and a support vector machine all generated the best mean absolute error, artificial neural networks generated the best root mean squared error and bagging with both model trees and artificial neural networks as well as stacking using support vector machines generated the best correlation coefficient. Table 4.12 shows a comparison between these best values and the performance of the existing prediction systems. As can be seen PECOTA outperforms Marcels and ZiPS, as well as my predictors, but my best values for the evaluation metrics outperform the latter two for all three evaluation metrics.

Table 4.13 shows the percent differences between the best values generated by my predictors for the three evaluation metrics and the best values generated by the existing baseball prediction systems. For mean absolute error and root mean squared error smaller numbers are better with negative percentages representing cases where my best predictor outperformed the best of the existing prediction systems. These instances are highlighted in bold. For correlation coefficient larger numbers are better with positive percentages representing cases where my best predictor outperformed the best of the existing prediction systems. Again these instances are highlighted in bold.

As can be seen in Table 4.13 my best predictors outperform the best of existing prediction systems for mean absolute error and root mean squared error when predicting hits. As a reminder PECOTA, which outperformed Marcels and ZiPS for all of the statistics predicted except triples, did not provide predictions for hits, thus this comparison is between the

Table 4.14: Percent differences of my best predictors and best free predictors.

	runs	**hits**	**doubles**	**triples**	**home runs**	**runs batted in**
MAE	**-7.73%**	**-7.96%**	**-7.29%**	**-6.62%**	**-7.02%**	**-8.88%**
RMSE	**-9.26%**	**-6.30%**	**-5.45%**	**-3.63%**	**-4.57%**	**-6.42%**
r	-1.49%	-2.93%	-1.10%	**1.88%**	-0.13%	**0.28%**

best values of my predictors for the three evaluation metrics and the best values of only Marcels and ZiPS. Additionally it can be seen that my best predictors outperformed the existing prediction systems for triples for all three evaluation metrics. In the cases where my predictors did not outperform the existing systems, it can be seen that difference between the metrics remained below about five percent and in most cases below three percent.

Table 4.14 is quite similar to Table 4.13, except in this case my best predictors are compared only to the two noncommercial prediction systems, Marcels and ZiPS. For hits and triples the values in both these tables are the same because PECOTA does not predict hits and the noncommercial predictors outperformed PECOTA when predicting triples. As can be seen my best predictors outperform the noncommercial predictors across the board for mean absolute error and root mean squared error. Additionally with the exception of hits, the percent difference in the performance of my best predictors and the noncommercial predictors is less than two percent when using correlation coefficient.

The following three tables are quite similar to the previous two, but in these cases I use stacking with a support vector machine as the level-1 predictor for comparisons instead of the best value provided by my predictors for the given statistic and evaluation metric combination. I chose stacking using a support vector machine because it provided consistently good results over the six statistics predicted with a maximum percent different of less than one between the correlation coefficient it generated and the best correlation coefficient generated by my other predictors. As mentioned earlier, correlation coefficient is the standard

Table 4.15: Percent differences of stacked SVM and best existing predictors.

	runs	**hits**	**doubles**	**triples**	**home runs**	**runs batted in**
MAE	2.67%	**-7.69%**	4.74%	**-6.62%**	2.18%	2.63%
RMSE	1.24%	**-5.04%**	4.00%	**-3.63%**	1.07%	1.64%
r	-2.03%	-3.49%	-5.87%	**1.58%**	-1.18%	-2.20%

Table 4.16: Percent differences of stacked SVM and free predictors.

	runs	**hits**	**doubles**	**triples**	**home runs**	**runs batted in**
MAE	**-7.25%**	**-7.69%**	**-7.29%**	**-6.62%**	**-6.02%**	**-8.88%**
RMSE	**-6.49%**	**-5.04%**	**-5.45%**	**-3.63%**	**-3.68%**	**-6.42%**
r	-1.77%	-3.49%	-1.88%	**1.88%**	-0.13%	**0.28%**

evaluation metric used when comparing baseball prediction systems, thus this is the reason I used it to determine which of my prediction systems performed more consistently well compared to the others.

Table 4.15 shows a comparison between stacking using a support vector machine and the best of the three existing prediction systems. The results in this table are quite similar to the results in Table 4.13 due to the fact stacking using a support vector machine often provided the best value for a given statistic and evaluation metric combination and in the other cases provided values very close to the best of my other predictors.

Again, as with Table 4.15, the results in Table 4.16 are quite similar to the results in Table 4.14 for the reason stated above. In this comparison stacking using a support vector machine outperforms the competition in mean absolute error and root mean squared error. For correlation coefficient stacking using a support vector machine outperforms the noncommercial predictors for triples and runs batted in.

Table 4.17: Percent differences of stacked SVM and average of existing predictors.

	runs	**hits**	**doubles**	**triples**	**home runs**	**runs batted in**
MAE	**-5.42%**	**-8.42%**	**-4.77%**	**-9.93%**	**-4.91%**	**-5.34%**
RMSE	**-5.04%**	**-7.57%**	**-4.59%**	**-4.62%**	**-2.32%**	**-5.34%**
r	-0.28%	-0.72%	-0.48%	**2.32%**	-0.53%	-0.56%

Table 4.17 shows a comparison between stacking using a support vector machine as the level one predictor and the average of the three existing prediction system. As with the comparison to the noncommercial predictors my best predictors outperform the average of the three existing predictors across the board for mean absolute error and root mean squared error. As for correlation coefficient, stacking using a support vector machine performs above average only when predicting triples.

Chapter 5

Conclusions

In this thesis I aimed to construct a baseball prediction system using machine learning techniques for six major offensive statistics which performed comparably with existing prediction systems. Accurate baseball predictions have become more and more important in recent years for baseball executives as well as fantasy baseball players. Towards this goal of generating more accurate predictions, I employed model trees, artificial neural networks, and support vector machines as well as the ensemble learning techniques of bagging, boosting, and stacking. The performance of these various techniques were compared amongst one another as well as with three existing baseball prediction systems.

While the prediction systems I created did not prove to outperform the existing prediction systems, they did perform quite well. None of individual systems implemented in this thesis outshone the others in all of the six statistical categories predicted. Instead different machine learning techniques used seemed tailored for different statistics. This led to the results for the individual statistics predicted looking quite different. Even among the three existing prediction systems performance was not consistent across the six statistics.

The results showed that the three base predictors chosen for this thesis can perform well in the baseball prediction domain. By themselves each was able to provide predictions that were on par with the three prediction systems used for comparisons. On top of the performance of the base predictors the use of ensemble learning was shown in many cases to improve performance. While there were a few cases in which one or more of the base predictors outperformed the best of the ensemble predictors, they often provided the best values for the three evaluation metrics.

Individually each type of ensemble learning had varying results depending on the component predictors involved and the statistical category being predicted. For bagging the base predictor that saw the best improvement across the six statistical categories was model trees. In each instance bagging model trees provided better or equal results for the three evaluation metrics compared to a single model tree. For artificial neural networks and support vector machines the effect of bagging was less successful with the results often being slightly worse than the base predictor alone. For boosting the results were quite similar, though bagging seemed to work slightly better than boosting overall. In particular there were instances where boosted model trees performed worse than a single model tree. This could be explained by the fact that I was using somewhat of an ad-hoc formalization of boosting for continuous prediction. For stacking using a support vector machine as the level-1 predictor proved to be the best configuration with both stacking using linear regression and stacking using a model tree performing slightly worse. In multiple instances stacking using a support vector machine provided the best value for one or more of the evaluation metrics among all of my prediction systems. In particular when predicting both triples and runs batted in, it provided the best values for all three metrics, the only one of my predictions systems to do that for even one of the statistical categories.

The only one of the six statistical categories predicted in which one of my prediction systems outperformed all of the three existing prediction systems for all of the evaluation metrics was triples. I believe that this is the case because triples more than any of the other six statistics relies heavily on a player's speed and hitting ability, which remain relatively constant over a player's career. Statistics like home runs that rely on a player's power and to some degree his home ballpark are more variable. Additionally statistics like runs and runs batted in have elements that are outside of a player's control. Unless the player hits a home run, he must rely on other players to bat him in to score a run or for other players to get on base to have opportunities to earn runs batted in. If the player is on a bad team, it is often the case that these opportunities occur less than if he was on a good team. Lastly the average

number of triples in a season as well as the standard deviation are small in comparison to the other statistics, which leads to fewer outliers.

Additionally the results showed that the approach used for constructing the instances on which the various prediction systems were trained and tested was a good choice for the domain. The approach I decided upon was quite different than that of existing baseball prediction systems, including those used in this thesis for comparison. The major departure was the use of data from only the one hundred sixty two games prior to the season being predicted. While most prediction systems use more data than this, my prediction systems were able to perform comparably. Additionally the selection of attributes to be used as inputs proved to work well for the domain.

Chapter 6

Future Work

While I believe that the research done for this thesis was relatively successful there are several areas in which work can be done that could possibly increase the accuracy of the prediction systems created as well as increase the predictive capabilities of the systems. The first and most obvious area in which work can be done is that of the training instances. As mentioned earlier out of the near ten thousand instances available from the 1973 to 2005 seasons I was only able to generate two thousand one hundred fifty one. Using more instances during the training of my prediction systems would provide more information about how players' performance changes over time and would provide a more accurate model.

Next I think it would be interesting to study the effect of using more than one hundred sixty two games to construct the inputs to my prediction systems. While the process of creating instances used proved to work well and led to relatively accurate predictions, it is possible that by extending the number of games used a more accurate picture of a player's past performance could be derived, thus leading to more accurate predictions.

As discussed previously, many existing baseball prediction systems make adjustments to actual game data in order to account for effects relating to a player's home ballpark, the time period in which he played, and the league in which he played, among others. By performing these and similar adjustments it is possible to more accurately compare players without taking into account differing circumstances that affect a player's performance. Thus it is possible to more accurately use one player's past performance to predict another player's future performance. Additionally performing such adjustments could increase the number of

players whose future performance could be predicted by my prediction systems, including players in the minor leagues and even college.

Given the current implementation of my prediction systems, I am unable perform predictions for players who have not played one hundred sixty two games. While this does not represent a large number of players, it is a gap in the capabilities of my prediction systems. In order to fully compete with existing baseball prediction systems, it is necessary to address this issue. While I do not know of a straight forward approach to including these players, this is an important area to be tackled in future work.

Lastly, many of the existing baseball prediction systems do not just predict the performance of hitters, as mine do, but also pitchers. It would be interesting to apply the techniques discussed in this thesis to the domain of pitching statistics. It has been shown that predicting statistics for pitchers is harder than predicting statistics for hitters[21], but given the performance of my prediction systems for hitters, I would expect to perform comparably to existing systems using a similar approach to the one discussed in this thesis.

Bibliography

[1] Breiman, L. (1996) *Bagging Predictors.* Machine Learning. 26: 123-140.

[2] Cameron, D. and G. Spira (1999) "The Prospectus Projection Project." *Baseball Prospectus.* http://www.baseballprospectus.com/article.php?articleid=229.

[3] Depken, A. (1999) "Free-Agency and the Competitiveness of Major League Baseball." *Review of Industrial Organization.* 14: 205-217.

[4] Freund, Y. and R. E. Schapire (1996) "Experiments With a New Boosting Algorithm." In *Proceedings of the Thirteenth International Conference on Machine Learning.* 148-156.

[5] Grabiner, D. "Brock2 Program Information." *The Baseball Archive.* http://baseball1.com/bb-data/grabiner/brock2.html.

[6] Huckabay, G. (2002) "6-4-3: Reasonable Person Standard." *Baseball Prospectus.* http://www.baseballprospectus.com/article.php?articleid=1581.

[7] Keri, J. ed. (2006) *Baseball Between the Numbers: Why Everything You Know About the Game Is Wrong.* New York City, NY: Basic Books.

[8] Lahman, S. "Lahman's Baseball Database." *Sean Lahman's Baseball Archive.* http://baseball1.com/content/category/6/36/82/.

[9] Lewis, M. (2003) *Moneyball: The Art of Winning an Unfair Game.* New York City, NY: W. W. Norton & Company.

[10] "Major League Baseball Salaries." *Baseball Almanac.* http://www.baseball-almanac.com/charts/salary/major_league_salaries.shtml.

[11] "Marcel 2007." *Tango On Baseball.* http://www.tangotiger.net/marcel/.

[12] Mitchell, T. (1997) *Machine Learning.* New York City, NY: McGraw-Hill.

[13] "Official Rules." *Major League Baseball.* http://mlb.mlb.com/mlb/official_info/official_rules/foreword.jsp.

[14] Pinto, D. "Baseball Musings Day By Day Database." *Baseball Musings.* http://www.baseballmusings.com/cgi-bin/ChoosePlayer.py.

[15] Platt, J. C. (1999) "Fast Training Of Support Vector Machines Using Sequential Minimal Optimization". In *Advances In Kernel Methods: Support Vector Learning.* B. Schölkopf, C. J. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press. 12: 185-208.

[16] Quinlan, J. R. (1992) "Learning With Continuous Classes. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence.* 343-348.

[17] *Retrosheet.* http://www.retrosheet.org.

[18] Russell, S. and P. Norvig (2003) *Artificial Intelligence: A Modern Approach.* Upper Saddle River, NJ: Prentice Hall.

[19] "Statistics." *Major League Baseball.* http://mlb.mlb.com/mlb/stats/index.jsp.

[20] Szymborski, D. (2003) "About ZiPS." *Baseball Think Factory.* http://www.baseballthinkfactory.org/files/primate_studies/discussion/dszymborski_2003-03-04_0/.

[21] Szymborski, D. (2006) "2006 Projection Results." *Baseball Think Factory.* http://www.baseballthinkfactory.org/files/newsstand/discussion/2006_projection_results/.

[22] Vapnik, V. N. (1995) *The Nature of Statistical Learning Theory.* New York City, NY: Springer.

[23] Walker, S. (2006) *Fantasyland: A Season on Baseball's Lunatic Fringe.* New York City, NY: Viking.

[24] Wang, Y. and I. H. Witten (1997) "Inducing Model Trees for Continuous Classes." In *Proceedings of the Ninth European Conference on Machine Learning.* 128-137.

[25] Witten, I. H. and E. Frank (2005) *Data Mining: Practical Machine Learning Tools and Techniques.* San Francisco, CA: Morgan Kaufmann.

[26] Wolpert, D. H. (1992) "Stacked Generalization." *Neural Networks.* Oxford, UK: Pergamon Press. 5: 241-259.

[27] Woolner, K. "Stats Glossary." *Stathead.* http://www.stathead.com/bbeng/woolner/statglossary.htm.