Machine Learning Applications for Analyzing Sailboat Race Handicaps

TCSS 702 Design Project in Computing and Software Systems

John C. Reed Jr.
Committee Chair: Professor Isabelle Bichindaritz, Ph. D.
Committee Member: Professor Ankur M. Teredesai, Ph. D.

August 22, 2008
## Contents

1 **Introduction** .................................................. 1

2 **Background** .................................................... 2
   2.1 Performance Handicap Racing Fleet (PHRF) .................. 2
   2.2 Rapidminer .................................................. 2
   2.3 Outlier Detection ........................................... 3
      2.3.1 Density Based Cluster Analysis ....................... 4
      2.3.2 Distance Based Cluster Analysis ..................... 4
      2.3.3 Local Outlier Factors ............................... 4
      2.3.4 Support Vector Machines ............................ 5
   2.4 Related Work ............................................... 5

3 **Dataset** ....................................................... 7
   3.1 Data Description ........................................... 7
   3.2 Data Preprocessing ........................................ 9
      3.2.1 Data Integration ..................................... 10
      3.2.2 Data Cleaning ....................................... 10
      3.2.3 Data Transformation ................................ 10
      3.2.4 Data Reduction ...................................... 10

4 **Sailboat Handicap Analysis** ................................ 11
   4.1 Skill Detection ........................................... 11
   4.2 Outlier Detection .......................................... 14
   4.3 Custom Analysis Algorithm ................................ 14
   4.4 Validation ................................................ 17
      4.4.1 Actual Changes in the Database ..................... 17
      4.4.2 Expert Analysis ..................................... 17
      4.4.3 Global Constant .................................... 18
   4.5 Web Application .......................................... 18
      4.5.1 Data Visualization .................................. 19
      4.5.2 Security ............................................ 19
      4.5.3 Web Services ....................................... 20

5 **Results** ...................................................... 20

6 **Discussion** .................................................. 24

7 **Future Work** ................................................ 25

8 **Conclusion** .................................................. 26

References .......................................................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Data Description</td>
<td>30</td>
</tr>
<tr>
<td>A.1</td>
<td>Data Dictionary</td>
<td>30</td>
</tr>
<tr>
<td>A.2</td>
<td>Descriptive Data Summarization</td>
<td>37</td>
</tr>
<tr>
<td>B</td>
<td>Charts and Graphs</td>
<td>43</td>
</tr>
<tr>
<td>C</td>
<td>Use Cases</td>
<td>46</td>
</tr>
<tr>
<td>D</td>
<td>Source Code</td>
<td>61</td>
</tr>
<tr>
<td>D.1</td>
<td>Rapidminer process files</td>
<td>61</td>
</tr>
<tr>
<td>D.1.1</td>
<td>Main process file</td>
<td>61</td>
</tr>
<tr>
<td>D.1.2</td>
<td>SVM process file</td>
<td>64</td>
</tr>
<tr>
<td>D.2</td>
<td>SQL Server Assembly</td>
<td>66</td>
</tr>
<tr>
<td>D.3</td>
<td>MS Access VB Scripts</td>
<td>102</td>
</tr>
<tr>
<td>D.4</td>
<td>Java Data Parser and Preprocessor</td>
<td>118</td>
</tr>
<tr>
<td>List of Tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Local Outlier Factor Results</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>Distance Based Cluster Analysis Results</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>Support Vector Machine One Class Analysis Results</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Density Based Cluster Analysis Results</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Custom Race Comparison Algorithm Results</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>Expert Analysis Results</td>
<td>24</td>
</tr>
<tr>
<td>List of Figures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Rapidminer user interface</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Outlier detection algorithms</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Sailboat Racing Database</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>J Boat 24 class performance distribution</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Comparison of two boats races over time</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>Effect of performance before and after skill detection</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Sample PHRF-NW Protocol Change Notice (PCN)</td>
<td>44</td>
</tr>
<tr>
<td>8</td>
<td>Sailboat Dimensions</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>Effect of performance before skill detection</td>
<td>47</td>
</tr>
<tr>
<td>10</td>
<td>Two dimensional view of data before skill detection</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>Two dimensional view of data before skill detection</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>Effect of performance after skill detection</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>Effect of performance after skill detection</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>Two dimensional view of data after skill detection</td>
<td>53</td>
</tr>
<tr>
<td>15</td>
<td>Two dimensional view of data after skill detection</td>
<td>54</td>
</tr>
<tr>
<td>16</td>
<td>Custom Algorithm Ratio Distribution</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>Custom Algorithm Ratio Distribution</td>
<td>56</td>
</tr>
</tbody>
</table>
Abstract

Amateur sailboat racing is an exciting and challenging sport enjoyed by many people. However, to account for the large diversity of boats a handicap system needs to be used that theoretically gives every boat a chance to win. The most popular system, the Performance Handicap Racing Fleet (PHRF), only takes the sailboats speed potential, not the crews skill or weather conditions, into account and because it rates each boat individually questions sometimes arise as to the accuracy. The heart of the system is a rating in seconds per mile that allows for a corrected finish time to fairly compare results of boats that may have actual finish times very far apart. The current handicap system accepts that mistakes are possible and procedures are in place to rectify improper handicaps. However, it can be very difficult for handicappers to analyze all of the data in order to determine if a correction should be made. This paper details the progress made in developing a system that uses several machine learning techniques to analyze and detect potential outliers in a large real-world dataset, with the overall goal of assisting the handicappers.

After the data preprocessing a subset was chosen for outlier detection. However, each sailboats skill level had to be determined and factored out since it is not accounted for in the actual handicap. Then several different methods of outlier detection were used, including Support Vector Machines (SVMs), distance based cluster analysis, density based cluster analysis, and local outlier factor analysis. In the end, the performance of generic algorithms proved to be poor, so a custom algorithm tailored to the specific dataset was created. The algorithm, which tries to emulate how humans detect outlier boats, greatly improved the outlier prediction accuracy. Finally, to tie everything together, a web application was created to enable the handicappers to visualize the data and results of the outlier detection algorithms.
1 Introduction

Sailing is a pastime enjoyed by many people. It is prized for its serenity but at the same time for its complexity. The skill required has led to the long standing and exciting sport of racing. However, not everyone can afford a million dollar racing boat or likes the sometimes rough rides of smaller racing boats. To accommodate the everyday amateur racer a system of rules and handicaps has been set up to try to put everyone on an even playing field. This can be difficult as the sailboats can be just as different as the racers themselves. An amateur race between twenty teams may contain twenty different sailboats in various sizes and shapes. To make everything fair each boat is assigned a handicap based on how well it should perform under ideal conditions. This is in the form of a number representing seconds per mile to be subtracted from the boats finish time giving a corrected finish time that is compared to all other race participants corrected times to determine the winner. In the end though, the handicap system is not perfect, due to the massive amount of factors used to determine it, and sailors can become frustrated when they are unable to beat a boat whose handicap is too high. This paper details a system that will help experts better set handicaps using outlier detection and data visualization, creating an environment that is more enjoyable for everyone.

The main difficulty in this project is the complexity of the data. Not only is the amount of variables large, in the current database the sailboat table has 67 fields (see figure 3), but there is also an amount of randomness to it. That is, sometimes a chance event, such as a wind gust or equipment failure, can affect a race. Any algorithm created needs to be smart enough to detect when something is not right and not use that particular piece of data as there is too much data for that task to be done by hand. Even a much simpler field of horse race odds prediction with only 12 input variables requires sophisticated data mining algorithms. [11] Sailboat racing has many more determining factors to take into account.

Extensive research[15, 19, 22, 17] has been done and so far no project has been found that attempts to solve the same problem as this one. One possible explanation is the lack of commercial interest. Most amateur racing organizations are non-profit and are run by volunteers with few paid employees. In contrast, millions of dollars are pored into professional races such as the Americas Cup and extensive research is done on boat performance. However, most professional racing does not use handicaps, either all the boats are the same or the goal is to make the sailboat go as fast as it can go, so research funding is geared more towards design and simulation. Another possible reason this area has not been explored academically is the small size of the community. The PHRF Northwest currently has only 891 active sailboats, which is minuscule compared to the population of the region even considering many people may race on one boat. With such a small community one has to wonder how many are researchers in the computer science field.

The lack of prior computer science projects in this area has both good and bad aspects. It is good in the sense that the idea may be novel and thus garner a
lot of merit. However, at the same time there are no sources to gain insight from because even if another project has the same goal there are usually many ways to achieve it. Possible methods for approaching the problem in an intelligent way include case-based reasoning [2, 26] and support vector machines [6, 27] both of which offer extensive areas of new research.

2 Background

The following sections detail the background knowledge necessary for the project and related work that helps further the project.

2.1 Performance Handicap Racing Fleet (PHRF)

In the United States the most popular handicapping system is the Performance Handicap Racing Fleet\(^1\) (PHRF) \(^2\). It is an administered handicap system, meaning local committees subjectively decide on the ratings, as apposed to a strict measurement-based system so the same boat can have different ratings depending on what region inspects it. As a result, it can sometimes be controversial.\(^9\)

One of the benefits of the PHRF is that it is constantly adapting through observation. A handicap reflects an estimate of a sailboats speed potential based on knowledge of previous racing experience supported by a consensus of the effect of differential hull and rigging parameters. Handicaps are arrived at through an empirical process based upon observation and analysis of race results. The basic process when rating a sailboat is to start with a standard, base boat, rating and make adjustments based on set rules concerning the hull and rigging parameters.\(^1\) However, as technology advances and more information is gained the rules are changed accordingly, although be it with much debate. Once the committee agrees on a rule change, a protocol change notice is issued. A sample notice can be seen in appendix B. This projects main aim is to make the entire process along with conflict resolution easier.

2.2 Rapidminer

The primary tool chosen to perform the outlier detection on the data is Rapid Miner.\(^23\) It is an open source data mining tool written in Java that is able to perform various types of regressions, classifications, and other data mining tasks, including outlier detection. Additionally, many third party packages have been incorporated into Rapidminer such as LIBSVM, which uses support vector

\(^1\)http://www.ussailing.org/phrf/index.asp
\(^2\)There is some confusion as to what the P stands for in PHRF. When first formed, in Southern California, the organizations name was the Pacific Handicap Racing Fleet. As the system became more popular and spread to the east coast the system was renamed the Performance Handicap Racing Fleet.\(^1\) However, most clubs on the west coast have stayed with the term Pacific, so the exact meaning comes down to which coast you live on, although the system is the same.
machines to perform outlier detection. The interface requires some learning, but greatly facilitates experimentation, as components can be easily swapped out or even just disabled for a particular run.

![Rapidminer user interface](image)

**Figure 1: Rapidminer user interface**

### 2.3 Outlier Detection

According to Hodge and Austin[16]outliers are defined as an outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs or as an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data. Many terms have been used including novelty detection, anomaly detection, noise detection, deviation detection, or exception mining, but all involve a similar process or goal. In addition to the many different names, there are many more algorithms for solving them. However, only a few are supported by Rapidminer. These being density based cluster analysis, distance based cluster analysis, local outlier factors, and support vector machines.
2.3.1 Density Based Cluster Analysis

The first step in density based cluster analysis is to find clusters with a specified minimum density and radius. Then it is simply a matter of identifying points not contained in any cluster as outliers. For example, see Figure 22b. The blue points are in dense regions and thus belong to clusters, while the red points, which are outliers, are in much less dense regions. One of the primary advantages of this method is the ability to find unusually shaped clusters, like the donut shaped one in Figure 22b.

Rapidminer uses a distance based (DB) outlier detection algorithm which calculates the DB(p,D)-outliers for an example set passed to the algorithm. DB(p,D)-outliers are Distance based outliers according to Knorr and Ng. A DB(p,D)-outlier is an object to which at least a proportion of p of all objects are farther away than distance D. It essentially implements a global homogeneous outlier search.[20, 21] Even though the algorithm is labeled as distance based, it is effectively calculating the density and Rapidminer lists it as such.

2.3.2 Distance Based Cluster Analysis

The first step in distance based cluster analysis is to compute the optimal clusters using a distance based algorithm such as K-Means, which groups data points with the centroid they are closest too. After the clustering algorithm is executed, finding the outliers is as simple as finding the points farthest from the centroid. A graphical example of the end result can be seen in Figure 22a.

Rapidminers distance based cluster analysis is a little more complicated. It performs a $D^k(n)$ Outlier Search according to the outlier detection approach recommended by Ramaswamy, Rastogi and Shim in "Efficient Algorithms for Mining Outliers from Large Data Sets". It is primarily a statistical outlier search based on a distance measure similar to the DB(p,D)-Outlier Search from Knorr and Ng. However, it utilizes a distance search through the k-th nearest neighborhood, so it implements some sort of locality as well. Basically, the method states that those objects with the largest distance to their k-th nearest neighbors are likely to be outliers because it can be assumed that those objects have a sparser neighborhood than the average object. This effectively provides a simple ranking over all the objects in the data set according to the distance to their k-th nearest neighbors.[25] Even though this sounds very similar to the density based method, Rapidminer lists this algorithm as distance based.

2.3.3 Local Outlier Factors

The Local Outlier Factors (LOF) algorithm in Rapidminer is a type of density based cluster analysis. It measures the density of objects and their relation to each other, called the local reachability density. The LOF is computed based on the average ratio of the local reachability density of an object and its k-nearest neighbors.

The LOF algorithm in Rapidminer consists of two basic steps. In the first step, the objects are grouped into containers. For each object, using a radius...
screening of all other objects, all the available distances between that object and another object, or group of objects, on the same radius of a given distance are associated with a container. The container then has the distance information, the list of objects within that distance, and the number of objects recorded in it. In the second step, three things are done. First, the containers for each object are counted in ascending order, according to the cardinality of the object list within, to find the k-distances for each object and the objects in that k-distance, which are all of the objects in all of the subsequent containers that have a smaller distance. Next, using this information, the local reachability densities are computed by using the maximum of the actual distances and the k-distance for each object pair, averaging it by the cardinality of the k-neighborhood, and then taking the reciprocal value. Lastly, the LOF is computed for each object value in the specified range, up to the upper bound, by averaging the ratio between the local reachability density of all objects in the k-neighborhood and the object itself. The maximum LOF in the specified range is passed as the final LOF for each object.\[5\]

2.3.4 Support Vector Machines

SVM methodologies have their roots in Vladimir Vapniks 1979 work *Estimation of Dependencies Based on Empirical Data*.\[6\] However, using SVMs for regression was not proposed until 1997.\[10\] Once SVMs were developed for performing regression it was a natural step to use them for outlier detection.

Even more recent research on weighted margin SVMs\[27\] looks promising as it allows for the incorporation of prior knowledge. SVMs already work well when the size of the input variables is large relative to the amount of training data, but incorporating prior knowledge pushes this benefit even farther. However, one problem is that the large amount of coefficients outputted in the regression make it almost impossible for a human to interpret thus artificial intelligence-based knowledge extraction routines could also be helpful. Experiments done by Jordaan and Smits have shown that SVMs can work well finding outliers in high dimensionality industrial data.\[18\] The actual implementation used by Rapidminer is LibSVM created by Chang and Lin.\[8\] In the most basic sense, it finds the hyperplane that separates the data and then determines the outliers by finding the points farthest from the hyperplane, as can be seen in Figure 22c.

2.4 Related Work

It can be difficult to find projects directly related to setting handicaps. One can infer that this is due either to the simplicity of many handicapping algorithms, such as in golf, bowling, or tennis, that rely only on past performance or that possibly researchers have yet to explore certain areas due to a lack of interest or unfamiliarity of the underlying fields. One area of research that may provide ideas is that of skill-based systems for racing sailboats. One such system, Robosail, uses a combination of agent technology, machine learning, data mining,
Figure 2: Outlier detection algorithms
and rule-based reasoning to maneuver a sailboat in a race. [3] A more related project uses support vector machine (SVM) methods to predict horse racing odds based on twelve input variables. [11] Odds are very similar to handicaps in that the lower the handicap the more likely that participant is to win if the handicap was not used, so this research involving SVMs proved useful.

Research by Graves et al. analyzed NASCAR races using permutation models.[14] They used performance metrics for each driver and were able to predict the winner of one series with 88% accuracy, although not all of their results were as good. The study used mainly custom analysis tools and focused more on simulation, even so, it still provided some insight into methodologies for analyzing this type of data.

The hardest part in finding useful information is that the majority of research involving sailboats deals mainly with simulation. Such as Pjilipott et al.’s work that uses velocity prediction programs and other models to predict race outcomes.[24] Though not necessarily related to this project, the studies can give some insight into the nature of the data and could be potentially useful in the future in detecting a boat’s skill level by comparing predicted performance to actual performance. However, they do not tend to deal with any race handicap issues.

3 Dataset

The dataset for this project is fairly complex. The following sections describe how the data was collected, the structure of the data, and how the data was processed.

3.1 Data Description

The database was created in the summer of 2006 in an effort to accumulate all of the wide spread data into one place. It was designed to accommodate multiple sailboat clubs and races. As a result, there are many tables with various functions. See figure 3 for the structure of the database, including attribute types and entity relationships. A detailed explanation of each table is also presented below in figure 3. The main tables used for this project being the Boat, Race, and Result. The full dataset contains 14325 race results for 1060 races with 639 boats of 299 distinct types participating.

The Member table contains information relating to a member. The username and password are allowed to be blank only if the account is a non-login one for data storage purposes only. This would be the case if a member did not have computer access, but their information is needed to be in the database for analysis purposes. The Boat table contains all the information relating to a boat. The MemberID would be considered the owner or administrator of the boat though more than one member can be associated with a boat as seen below only this member can edit it. The Club table contains all the information relating to a club. The AdminID is the member who is not already an administrator who has
Figure 3: Sailboat Racing Database
the ability to edit the information. The Race table contains general information about a race such as a description and race conditions. Most race information is allowed to be null since it will not be known at race time. The Result table will contain the results for all of the races. Since a boats class can change per race the class a boat raced under for a particular race is stored here where as the boats default class is stored in the Boat table. The Sponsor table contains a relationship between a race and a club, since more than one club can sponsor a race or more the one clubs boats may be racing in a race. The ClubMembership contains the relationship between members, boats, and clubs. A boat can be null because a member can belong to a club without a boat, but the member can not be null. In this relationship you can also have many members associated with one boat. There is a flag to set for due paying and active members of the club for filtering and analysis purposes. The Course table contains basic information about a course and a link to its first node. The CourseNode table represents a course node and is basically a pointer to a marker. Since markers can be used repeatedly, the information is not duplicated. Also, with a pointer to the next node an effective linked list is created that supports an infinite course length. The Marker table contains the information about a marker used in race courses. The Admins table defines which members are administrators and what level of access they have, i.e. whether they are handicappers, race administrators, etc. Whenever an exception is caught it is saved in the ErrorLog table along with any user information. Stored procedures and triggers in the database will delete old errors and keep the log from growing too large. The StandardBoats table contains standard boat types with the same attributes as the boats table except for identifying information, since these boats do not exist. It can be used to fill in missing information in the Boat table. However, the same attributes need to be maintained in the Boat table because physical modifications can be made to individual boats that change them from the standard.

3.2 Data Preprocessing

The following sections detail various aspects of the data preprocessing, including data integration, cleaning, transformation, and reduction. The data was obtained from race results published on the web. As a result, much preprocessing was needed. The first step involved creating a parsing program in Java to extract the information and put it into comma separated value text files. The text files were cleaned and processed using regular expressions before importing them into Microsoft Access upon which more cycles of cleaning and processing were performed. Microsoft Access was chosen for the majority of the data processing for several reasons. The graphical interfaces allow easy manual manipulation of the data. The built in SQL engine allows powerful filtering and processing of the data. Lastly, the built in Visual Basic language allows scripts to be created in the database that are quick and easy to write, yet very powerful. After the preprocessing was completed the data was migrated to Microsoft SQL Server using the Upsizing Wizard in Access.
3.2.1 Data Integration

This was probably one of the most difficult parts of the data preprocessing. Since results from three separate clubs were obtained in different formats, one just in plain text, one in html tables, and one in html spans and divisions, processing took a lot of time. It had to be determined which fields were the same, although corrected time was one of the most universal. Also misspellings or different spellings had to be corrected along with differences in abbreviations. For instance, one club uses RANG to abbreviate Ranger and another club uses RANC.

3.2.2 Data Cleaning

The data cleaning involved removing records that had too many missing values or improperly formatted values. The second scenario was mostly dealt with by MS Access as improperly formatted fields were automatically identified upon importing the data. They could then either be fixed or removed. Finding records with missing information is also relatively easy with simple SQL queries in MS Access. However, many bad records were removed from the text file, prior to importing, using Textpads regular expression features. For example, if a record contained DNF, meaning did not finish, a replace operation could be run with the regular expression `\.*DNF\.*\n` to remove all those lines. This tool was used to remove many bad records that fit known patterns.

One problem that appeared when running visual basic scripts was the presence of a in the boat name, or any text field for that matter. The simplest solution was to remove this character from all of the data. Although MS Access usually does not make a distinction between upper and lowercase letters, this problem also needed to be fixed, though most of the attributes chosen for this project are numeric.

3.2.3 Data Transformation

The corrected time was transformed from a textual representation into numerical seconds. This allowed for easier mathematical calculations, such as determining the seconds behind or missing information such as the race distance. Other than that, no data was discretized or otherwise transformed. However, it is being debated whether it may be useful to discretize the race distance or even the seconds behind, which would actually be akin to the finishing place. Initial experiments with such transformation have provided no improvement.

3.2.4 Data Reduction

The overall dataset was significantly reduced for the purposes of outlier detection by selecting only the attributes that were the most relevant. Although there are many attributes describing a boat there are only a few, critical ones, that contribute most to its performance. These being the ones related to the sail area, I, J, P, and E, and water displacement, DISP, LWL, and DRFT. The
more sail area the faster a boat will go and the more water it displaces the slower it will go with the same sail area. See Figure 8 in Appendix B for a visual representation of these attributes.

In the Results table, these are the distance, corrected seconds behind, which determines place, and the PHRF rating, all of which are numeric. The PHRF rating was chosen because it encapsulates a boat’s performance, which should be an excellent predictor, even though the race times are already corrected based on this factor. Additionally, initial experiments on the full set of race results failed with out of memory exceptions. As a result, the race information was aggregated into averages that significantly reduced the size of the dataset.

4 Sailboat Handicap Analysis

The following sections detail the different aspects of the sailboat handicap analysis project.

4.1 Skill Detection

Skill detection was deemed a very important aspect of the project. It is clear from analysis on the original dataset that performance was a key contributor to the selection of outliers, as can be seen in figure 9 Appendix B. These diagrams show that initial experiments on the original data produce clear patterns.

Actual handicaps are determined by performance somewhat independent of skill. The national organization that assigns handicaps factors out skill by looking at only the top 25% of racers.[4] In other words, ignoring 75% of the data. Their dataset is so large this is not a problem for them. However, it is more desirable to be able to look at all racers, so skill needs to be factored out some other way. The simplest way to determine skill is to compare how different people perform on the same type of boat. If one person consistently performs better than another and they have the same boat, then the person who performs better can be considered to have a higher skill level. Figure 4 shows the performance of one of the largest classes of boats. The Y axis is the average seconds behind, or performance, and the X axis is each individual boat spaced evenly. As can be seen, there is quite a wide distribution.

However, the majority of boat types have only a few members, or even just one. In that case, similar handicap ratings can be used for comparison instead of exact boat types. The theory being that two boats with the same rating should perform similarly. The only problem is that two boats with the same rating might perform differently under different conditions. Therefore, comparisons based on ratings are less accurate, so they are given a lower weight in the overall formula. Since, intuitively, skill changes over time it was decided to consider skill as a linear function instead of as a static value. The slope of the function being the rate of improvement, positive or negative, and the Y Intercept being the performance difference from average at year zero, which is the first year data was collected. In the current dataset that year is 1996, but
Figure 4: J Boat 24 class performance distribution

the algorithm finds it every time it is run, so it will be updated if new data is added. Even though a boat’s average performance was calculated per year, each value was compared against the global average of the specific boat type or handicap rating. Taking the global average, as opposed to the average of the same year being calculated, should be more accurate, as it contains much more data than the latter.

Since the skill calculation requires most of the data from the database, it was decided to perform the calculations locally, as opposed to remotely. Even though the web application runs locally, on the same machine, it is designed to run remotely. Thus a different approach was needed for the skill detection. With Microsoft SQL Server there are several different methods of performing calculations within the database itself. The most basic way is to create a SQL function or stored procedure. The code is stored and ran in the database, which is desirable, but does not support some of the more advance features that full programming languages have making it less functional.

A second method to perform calculations in SQL Server is through the Data Transformation Services (DTS) component. Using DTS, code can be written in Visual Basic (VB) Script that manipulates the data in the database. Using VB Script is a plus, as much data manipulation code had already been written in it for MS Access during the data preprocessing stage. The biggest drawback of DTS, and the primary reason for not using it, is that it is not supported in free version of SQL Server, which is used by this project. A final way to manipulate the data in SQL Server, and ultimately the one chosen, is to use managed code compiled into a DLL assembly that can be loaded directly into SQL Server.

A major obstacle in creating the SQL Server assemblies was the lack of good documentation. The tutorials online and on MSDN were very general. Going so far as to say load some template, add some lines of code, and compile it. However, the templates were missing from the express and standard editions of visual studio, which were the versions used at home and school respectively. Once the assembly is compiled and loaded the methods in the DLL can be linked to custom SQL functions or stored procedures. These functions or stored
procedures can then be called from a client or from custom code such as the
web application. One of the major advantages of a SQL Server assembly is that
it can easily be modified to run externally because the only difference in the
code base is how the connection is made. This means the code is not explicitly
tied to SQL Server. If the database server is changed in the future, only a
few modifications would be necessary to enable the same code to run on it.
Pseudocode for the skill detection algorithm can be seen in Source 1.

**Source Code 1** Skill Detection Algorithm Pseudocode

```plaintext
create hashtable HC to store class average objects
create hashtable HR to store rating average objects
create hashtable HB to store boat data objects
for each boat B in the database
    get average seconds behind A
    get number of races R
    create boat data object and add to HB
    get B.class object BC from HC or create a new one if it does not exist
    add A,R to BC and add, or overwrite, BC to HC
    get B.rating object BR from HR or create a new one if it does not exist
    add A,R to BR and add, or overwrite, BR to HR
end for each
for each average performance A per boat B per year Y
    add A,Y to B in HB
end for each
for each boat B in HB
    get B.class average CA and count CC from HC and add to B
    get B.rating average RA and count RC from HR and add to B
    adjust each years average in B based on CA, CC, RA, and RC
    perform linear regression on adjusted year averages in B
end for each
update skill function of all boats in database
```

After the skill detection algorithm is executed and the corresponding func-
tion values are added to the database, the last step is to recalculate all of the
race results. Race results are looked at year by year and each boat’s results are
adjusted based on the value outputted from its skill function with the year as
input. To this end, extra attributes are added to the Results table to store the
new corrected times and the new corrected seconds behind. This ensures the
old values are preserved.
4.2 Outlier Detection

After the data preprocessing, the subset of data defined earlier was obtained by running the batch file seen in Appendix D source code 6. The output is a comma separated file that can be loaded directly into Rapidminer. Both the original data and the skill corrected data are present, so a filter is needed in Rapidminer to ignore one or the other depending on which tests are being run. Since all of the algorithms were very similar, except SVM, only two process files needed to be created in Rapidminer. One for SVM outlier detection and one for the others, both of which can be found in appendix D. The process files are in XML format which is text based and allows for easy sharing. There was a bug in the current version, 4.1, so in order to run the SVM outlier detection the HEAD of the cvs at Sourceforge needed to be checked out and compiled.

4.3 Custom Analysis Algorithm

As a result of poor performance of Rapidminer outlier detection algorithms, a new custom algorithm was designed and implemented. The approach of this algorithm is similar one of the methods a handicapper might use to decide if a boat is an outlier. The process starts by looking at one boat and then comparing it to each individual boat raced against. Like a handicapper, it looks for patterns between individual boats, such as if one consistently finishes before or after the other. For example, a handicapper would look at figure 5, graphs of two boats finishing times for the same races superimposed on each other, and see a clear pattern in the first but not in the second. A human is able to see this, but a computer needs to do some analysis. The simplest way to see how the boats compare to each other is to subtract their times for every race. If all the differences were added up, the end result could be used to determine if a pattern is present. The closer to zero the less likely there is a pattern because any pattern would continue to add in the same direction, either positive or negative. However, the dataset is prone to outliers, so it is likely to have an extreme difference between any two boats and all it takes is just one to cancel out a potentially large pattern in the opposite direction. For instance, if two boats had 10 races with a difference of 10, all it would take is one difference of -100 to bring the sum back to zero, even though there is a clear pattern in over 90% of the data. To smooth out the outliers the differences are discretized into two bins, either positive or negative with zeros being ignored. This essentially ignores the degree of difference, which could be an important factor, but requires future work to make a determination.

The next step is to divide the number of positive and negative differences creating a ratio. This could be done in the same order in which the boats are compared. However, dividing the smaller value by larger instead has several benefits. First, it ignores whether the overall difference was positive or negative. This allows for a better comparison between values, as all negative differences is just as significant as all positive differences. Second it normalizes values between 0 and 1. If one boat always wins over another, the numerator will be zero and
zero divided by anything is zero. It can never go below zero because there are no negative numbers. The largest possible value is when both boats have won an equal number of times and it is known the \( x/x \) is 1 no matter what \( x \) is. If either number is decreased it will go on top and the number will be less than 1, so the maximum is 1. Thus the closer the number is to zero the more of a pattern there is and the more likely that ratio is an outlier. A final benefit is that the same value can be added to both boats so redundant or extra calculations can be eliminated.

Along with the ratio a count of the number of races is also recorded, showing the magnitude of the ratio. These two values together constitute a score for both of the boats being compared. For instance figure 55a has a discernible pattern with 32 positive differences and 0 negative ones. This gives it a ratio of 0/32 and a magnitude of 32. Figure 55b, which has much less of a pattern, has 17 positive differences and 18 negative ones, giving it a ratio of 17/18 and a magnitude of 35.

![Figure 5: Comparison of two boats races over time](image)

After all of the scores are computed they are analyzed to find a threshold above which the score would be considered an outlier. In computing the threshold both the ratio and the magnitude are used. The magnitude threshold chosen was just the global average. So a score would have to have a magnitude greater than average. Similarly, the ratio threshold is based on boat averages, as opposed to all values. It would be more accurate if every ratio was used, but this would require more memory and processing time. Also, a boats average ratio is a good indicator of whether or not it is an outlier. A simple statistical analysis is done on the averages to find the bottom x percent of ratios, where x is a user settable value. From this the threshold is found, which is simply the highest value in the bottom x percent. Anything lower than this threshold can be considered an outlier. Finally, points are assigned based on the score threshold. A boat is awarded one point for every score whose ratio is below the
ratio threshold and magnitude is above the magnitude threshold.

One problem with assigning points is that boats who have participated in more races have a greater chance to get more points, so the values are not directly comparable to each other. To compensate for this the points must be normalized. It is usually best to normalize on the same or most similar attribute, so the first test divided the points by the number of boats raced against, which would be the theoretical maximum points. Other tests were done normalizing on the number of races, the number of years raced, and other variables. In the end, no matter what normalization method is used, the more points a boat has the more likely it is an outlier.

Psuedocode for the algorithm can be seen in Source Code 2. The theoretical worst case performance of the algorithm is $N^2 + K$. This would be the case is every boat had to be compared to every other boat for every race. However, on the actual dataset only 32664 comparisons were performed in 15 minutes vs. 408321 comparisons in roughly 3 hours if it was $n^2$. Out of 639 boats, 102 boats had an average count of 1. Since the average number of points for comparison was 5, none of these boats would ever receive any points and never get marked as outliers. This fact helps emphasize the need for more data. One last observation found that the distribution of ratios was much closer to 0 than expected, as can be seen in Appendix figure.

Source Code 2 Custom Analysis Algorithm Psuedocode

create statistics object stat
get list of boats and create boat objects
foreach boat object A
    foreach boat raced against B
        if A does not contain B
            forall races between A and B
                count Positive P and Negative N differences
                calculate ratio of P and N C
                add value C to A
                add value C to B
            end if
        end for each
    calculate outlier value D based on race comparisons
    add A,D to stat
end for each
have stat determine most probable outliers
add each boat outlier result to database
4.4 Validation

To test the accuracy of any of the outlier detection algorithms it must first be
known which boats are actually outliers. This can be difficult to determine on a
real world dataset such as the one used in this project. Some of the outliers may
not have even been identified yet. To overcome this difficulty several methods
of validation were employed, as described below.

4.4.1 Actual Changes in the Database

One of the simplest ways to determine which boats are outliers is to look at what
changes to their ratings have been made in the past. To this end, the database
was scanned using the SQL query in Source 3 to find actual adjustments which
are then used to test the accuracy of the outlier detection algorithms. In the
actual data, 174 of 639 boats have been adjusted at least once, which is over 27
percent. Of those, 48 have been adjusted by 20 seconds or more. Since so many
boats have been adjusted in the past only the top 10

Source Code 3 SQL query to find changes

```sql
SELECT b.Name, b.BoatType, q2.bid, q2.low, q2.high, q2.diff,
q2.num - 1 As changes FROM (SELECT q.bid, min(q.p) As low,
max(q.p) As high, max(q.p) - min(q.p) As diff, COUNT(*) As num
FROM (SELECT r.BOAT_ID as bid, r.PHRF As p FROM Result
As r GROUP BY r.BOAT_ID,r.PHRF) As q GROUP BY q.bid) As q2,
Boat as b WHERE b.BoatID = q2.bid AND q2.num ¡¿ 1 AND
q2.low ¡¿ 0 ORDER BY q2.diff DESC;
```

The major problem with these actual outliers is that they do not take time
into account, or in other words it is not recorded when a change was made.
Since none of the outlier detection algorithms take time into account anyway
the information is not necessary. However, in the future, better results will most
likely be obtainable by taking time into account as one of the variables.

In addition to actual changes the actual appeals can also be used to test
accuracy. Both contain the same type of information, but the appeals are more
formal. However, with 10 years of meeting minutes and many records only on
paper, it is an enormous undertaking to process all of the appeals.

4.4.2 Expert Analysis

To make up for some of the limitations of the primary validation method, experts
in the field of handicapping were asked to generate a list of outliers, both past
and current highly suspected ones. The list was quite long at first, but was
reduced by putting a 10 race limit on the boats. A boat had to have participated
in at least 10 races to be included. The reason being, no outlier detection
algorithm could be expected to detect a boat as an outlier if it has only done
one or two races. In fact, the custom algorithms magnitude threshold ended up
being 7, so it was impossible for any boat with less than 7 races to be marked as an outlier.

4.4.3 Global Constant

In the perfect system a global constant would arise across all races where the standard deviation between sailboats corrected finish times would be as close to zero as possible, that is all sailboats should finish at about the same time. In the web application the global constant is calculated by averaging the number of seconds behind first place over all races. The average for each boat’s average could be used, but many boats have been in few races, so a few boats that have not been in many races and have an extremely high value could skew the overall average. The average speed could also be used, but distance information is not present in all races. If the scoring was done based on distance, then it can be derived, but some clubs only score based on time not distance. However, no system is perfect, but handicaps are designed to equalize all boats, so the variance should be minimized.

The main benefit of the global constant is that the effect of changes can be calculated. For instance, if a boat is identified as an outlier, its handicap can be corrected and all of the results recalculated in the database. If the new global constant is lower after the recalculations, then changing the boat’s handicap had a positive effect.

4.5 Web Application

Finally, to tie everything together a web application was developed in Microsoft Visual Web Developer Express using the c# language. The current web application consists of 82 source files ranging in size from only a few lines to over 3400 lines for the main web service. The web structure was designed so that a member or administrator did not have to browse through a web of links to be able to perform a certain task. Almost all the necessary tasks can be reached from the main homepage and link back to the homepage so very little navigating is needed.

A use case diagram can be seen in appendix C. Most of the use cases deal with managing data. Things such as adding boats and editing race results.

Along with managing data, the web application provides several tools for handicappers. First, the web application calculates and displays statistics for handicappers. The simplest are just counts of the number of members, clubs, boats, boat classes, races, and results. More complex statistics include the global constant and the linear skill function, both of which were described in a prior section. Also, the average global constant is displayed, which is the average of each boat’s average. A unique statistic compiled by the national organization is the PHRF minimum, maximum, and average. This helps show if a boat’s current rating is within the range of the national average for the same type of boat.
In addition to displaying simple statistics the web application is also able to create charts and graphs. Charts and graphs include boats per races, races per boats, PHRF race distribution, PHRF boat distribution, boat class distribution, performance by race, performance by boat, overall boat performance, and boat performance by year. The boat in class performance and boat in PHRF performance graphs are helpful because they show where a boat falls within a class or within a PHRF rating range. Finally, one major, and very useful, feature is the ability to compare boats’ race results. On an individual boat’s page is a list of every boat it has ever raced against. Selecting a boat from that list will display a graph of both boats finishing times behind first for every race they were both in. From this graph it is easy to see if one boat always finishes in front of or behind another or if they are highly competitive.

One thing the web application does not do, which it was originally intended to, is perform the actual outlier tests in real time. This was deemed impractical due to long run times. However, running the test independently has some advantages. First, the presentation is kept separate from mining, so there is no need write code to integrate the separate components. Second, the web application is able to display the results of any test instantaneously. Finally, since all of the outlier test results are precomputed and stored in the database, the web application is able to use polling or voting techniques. The theory being that the more tests that predict that a boat is an outlier the more likely it is one. A future step would be to add weighting to the polling, so if one algorithm is deemed more accurate than another it is given a higher weight, or more votes.

4.5.1 Data Visualization

The primary tool used to visualize the data was ZedGraph[7], an open source charting class library. ZedGraphs compiled library was call directly from the web application to produce, on the fly, graphs and charts. Also, to aide in visualization, a relatively new technology to web applications called Ajax[13] was used. The basic concept is to load only the parts of the page that need updated. Since the entire page is not reloaded, the user benefits by having a more persistent interface that uses less bandwidth. Also, server load is decreased because complex calculations, like creating a graph, do not need to be performed on every page update.

4.5.2 Security

As with any internet based application security is a major concern. Outlined here are the security steps taken to help protect the application and its users. Input validation is probably one of the most crucial aspects of security. If your input is validated then you have mitigated a very large number of attacks from cross side scripting to denial of service. To help with the validation of input all the data from the user is put into objects prior to it being used for any purpose. Inside the objects the data must pass a validation in order to be saved otherwise it is discarded and the objects corresponding validity flag is set to false. This
way, if code somewhere forgets to check if the object is valid before using it, then the invalid data is simply not there to cause any problems.

The passwords are hashed using a 160 bit SHA1 algorithm prior to being stored in the database. A hashing algorithm is one way in that you can make a hash from a password, but if the user constructs their password effectively it is almost impossible to derive the password from the hash. This allows the application to validate passwords without actually knowing them so if the database is compromised no passwords will be revealed. One drawback of this is that you can not send the user their password via email if they have forgotten it. The solution to this is to email the user a reset link enabling them to input a new password.

To avoid sending the password back and forth once the user is authenticated they are given a token. The token contains encrypted information that helps the system validate the user. Every method in the application deemed sensitive requires a valid token in order to operate. Tokens are encrypted with the secure DES algorithm and contain time information so that a token will expire after a certain amount of time and also user information such as their IP so a token cannot be used from different addresses. Also, the token is checked against the one the user was issued to make sure it has not been tampered with. The reason for implementing our own system is to create more open and simple standards. If Microsoft technologies were used it is unclear whether they could be used on all platforms.

4.5.3 Web Services

Web Services provide a way for just about any application, no matter what language it was written in, to access the methods on the server. Using web service interfaces requires the implementation of additional security measures, but allows other applications to manipulate the database. In ASP.Net web services are implemented by simply inheriting the web services class and defining the public methods as web services. No additional programming was needed.

5 Results

After countless hours of running tests in Rapidminer, the top results were collected and are reported here. Even though numerous settings were tried, it is near impossible to iterate through every possible combination, so there may exist configurations that yield better results. To see if the results are significant, they can be compared to random chance. Statistics predict that randomly guessing should yield about 8.6% of the outliers. Any percentage significantly above this is highly unlikely to be due to random chance. However, after running numerous outlier tests with countless variations a common accuracy of 14% emerged, even for some tests that were expected to not perform well. So, to make sure there was not a miss calculation and the actual accuracy of guessing was 8.6%, a test was devised to repeatedly make guesses and score them. The
test, whose code can be seen in appendix D, made the same number of guesses
as there were actual outliers for each test so the precision and recall would be
the same. After a thousand iterations the average accuracy turned out to be
8.4%, which is very close to the calculated value. A distribution of the accuracy
of the thousand tests can be seen in appendix B figure 14.

As previously mentioned, skill detection is a very important aspect in outlier
detection. Every algorithm was executed on the data both before and after skill
detection. What emerged were clear patterns that skill was a factor and evidence
that the skill detection algorithm was working, as can be seen in Figure 6.
The chart shows that before skill detection the distance based cluster analysis
algorithm found the boats that either performed very well or very poorly as
outliers. Performance is charted on the X axis with the larger numbers having
worse performance. The Y axis is the boat’s rating and is just used to spread
the data out, as the distribution is fairly uniform. The boat ID could be used
instead, but that would make the Y axis too large, as one line would be need
for each boat. If performance is a factor, vertical patterns will be seen, as in the
figure. After skill detection no clear patterns can be seen, showing that the skill
detection algorithm was effective at reducing the effect of skill. An extensive
collection of charts for every algorithm can be seen in appendix B.

One of the first, and best performing, algorithms in Rapidminer was LOF.
The results are presented in Table 1 in the standard precision recall format.
After the skill detection algorithm was applied all of the LOF’s output values
changed, however large values remained large and small values small, so there
was not much change in the order and thus no change in the results.

<table>
<thead>
<tr>
<th>Predicted Outlier</th>
<th>Not Outlier</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>52</td>
<td>20%</td>
</tr>
<tr>
<td>42</td>
<td>532</td>
<td>92.68293%</td>
</tr>
</tbody>
</table>

Table 1: Local Outlier Factor Results

Like the LOF algorithm, the distance based cluster analysis algorithm had
little change in accuracy before and after skill detection, though the actual
outliers found was not exactly the same. The results can be seen in Table 2.

<table>
<thead>
<tr>
<th>Predicted Outlier</th>
<th>Not Outlier</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>52</td>
<td>13.33333%</td>
</tr>
<tr>
<td>47</td>
<td>532</td>
<td>91.88255%</td>
</tr>
</tbody>
</table>

Table 2: Distance Based Cluster Analysis Results

The SVM test results, as seen in table 3, seem good at first, but the output
of the algorithm was poor. The predictions either contained too many or too few outliers. In the end, too many outlier predictions yielded better results. So although the algorithm found 25% of the outliers, its precision was only about as good as the distance based cluster analysis. After the skill detection algorithm was applied to the data, the SVM algorithm predicted far too many outliers to be useful. Given ample time to fine tune the algorithm’s settings these shortcomings could potentially be overcome, as the SVM outlier detection was expected to be one of the best for this type of data.

Of particular note, the SVM outlier detection algorithm was the only one that was able to run on the full fourteen thousand plus dataset, but the results were very poor. Before skill detection it was only able to find 9% of the outliers with about the same precision. After skill detection it was only able to find 2% of the outliers. To try to help to algorithm, extra attributes were added, but that only decreased the performance further. Overall, the SVM outlier detection performed much better on the aggregated dataset.

The only algorithm to show a significant change after skill detection was the density base cluster analysis. During the course of experiments it was found that
removing the BoatType attribute, which was the only non numeric, increased the performance significantly. The results presented here, in Table 4, are the ones derived from the same set of data used for the other algorithms, in order to make a fair comparison. The other algorithms performed the same with or without this attribute.

Table 4: Density Based Cluster Analysis Results

Even though the custom algorithm was designed with the exact problem in mind, it is still unbiased. That is, it had no direct knowledge of the information from which the list of outliers was based, that being the ratings and their changes. The algorithm only looked at corrected times not the ratings. Keeping that fact in mind, the results presented below are very good, especially when compared to the generic algorithms.

As stated in the section describing the custom algorithm, several methods of normalization were tried. However, after normalization the accuracy dropped to 18% in the best case. So the results presented here are from the unnormalized results. It can be theorized that boats that have more data are more likely to have been identified as outliers and thus be on the list. Similarly, the unnormalized score has a better chance of being higher when there is more data for a boat, so the two have a better chance to be correlated. The normalized scores may be good at finding previously undetected outliers, but that is difficult to test. In other matters, skill detection only changed the results by one boat, though the order of boats was changed due to scores being different. The results can be seen in Table 5. The results after skill detection are not presented as they are essentially the same.
Outlier Test Results

<table>
<thead>
<tr>
<th></th>
<th>Outlier</th>
<th>Not Outlier</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Outlier</td>
<td>24</td>
<td>39</td>
<td>38.09524%</td>
</tr>
<tr>
<td>Predicted Not</td>
<td>31</td>
<td>545</td>
<td>94.61806%</td>
</tr>
<tr>
<td>Recall</td>
<td>43.63636%</td>
<td>93.32191%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Custom Race Comparison Algorithm Results

Finally, the expert generated list of outliers was used to score the top performing algorithms. To show the importance of the amount of data, the number of races is listed next to each boat’s name in Table 6. It can be seen that the more races a boat has been in, or in other words the more data, the more likely it will be detected as an outlier. Each column represents a specific test, whether that boat passes the test, and a ranking, if available. Even with limited data in the database, all of the algorithms performed well in detecting the expert selected outliers. In fact, the custom algorithm was able to find four out of seven. The odds of which are similar to drawing seven playing cards from a deck and getting four aces. The percentage of selections made is about the same, but the percentage of outliers is much less than the percentage of aces in a deck, so it is not an exact comparison.

<table>
<thead>
<tr>
<th></th>
<th>Comp(rank)</th>
<th>LOF(rank)</th>
<th>SVM</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss Conduct(228)</td>
<td>110(1)</td>
<td>3.92(2)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Koosah(212)</td>
<td>102(4)</td>
<td>0.51(51)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Flasher(78)</td>
<td>73(31)</td>
<td>0.17(No)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fjord(83)</td>
<td>68(35)</td>
<td>0.38(61)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UFO(28)</td>
<td>45(No)</td>
<td>0.1(No)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Aquarius(41)</td>
<td>39(No)</td>
<td>0.16(No)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Time Warp(10)</td>
<td>4(No)</td>
<td>0.12(No)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Accuracy</td>
<td>57%</td>
<td>43%</td>
<td>57%</td>
<td>14%</td>
</tr>
</tbody>
</table>

Table 6: Expert Analysis Results

6 Discussion

In the course of this project Rapidminer released several major bug fixes. Experiments done in the very beginning produced some very interesting results that couldn’t be reproduced exactly with later versions. Specifically, the distance based cluster analysis algorithm initially predicted almost all of the outliers with a very poor performance metric. However, the latest version with the same data and settings predicts outliers with both very poor and very good
performance metrics, but no where in between. This still shows that performance is the main factor before skill detection is applied and is actually more in line with what one would expect, the best and worst performers being detected as outliers.

The second set of data, with the skill detection, showed some markedly different results, even when running on the original data which was present, but initially ignored. The only observed difference was that the original dataset performance metric was defined as integer, whereas all numeric attributes were defined as real numbers in the second dataset. Otherwise, all the settings were the same.

One of the primary problems with the data is its variable nature over time. Though not necessarily time series data, as one race can be compared to another independent of when they occurred, some things such as skill theoretically change over time. In fact, the main reasoning for making skill detection a function of time was to try to factor out the effects of time and make the data more static. Also, some races, especially long ones have very large courses that allow boats to take dramatically different routes. Even though two boats taking completely different routes makes them seem incomparable, this could be considered a factor of skill. Boats with a higher skill level will take the better route. So factoring out skill may help mitigate this problem.

7 Future Work

Probably one of the best ways to improve upon this project would be the collection of more data. Clear trends were seen where outlier boats with more data than others were more likely to be detected, so it can be inferred that more data will help with the detection of previously unknown outliers. Also, for skill detection, many boats had nothing to compare against, so additional data would improve this area of the project too. In addition to more data, it may be beneficial to use more data transformations such as weighting and normalizing.

In the very begging just about everything, such as scoring and inputting results, was done by hand in excel, but as the project progressed tasks became more automated as can be seen in appendix Dsource code 4 which loads results from a text file and adds them to the database the runs a stored procedure to score them. However, the test information still needs to be added manually and the proper id number substituted. Ideally, the whole process, from downloading the dataset to uploading the result, would be completely automated, running on a defined schedule.

All of the planned data visualization uses cases were completed, however there are still some data manipulation use cases that need to be finished. Though not necessarily academic, they are essential to the functioning of the web application. Completing the web application is a simple task, it will just take time.

Since this project ’s main goal is to find improper handicaps, a model based outlier detection methodology could be employed. The simplest methodology
would be to use any algorithm that creates a model of the data. Once a model has been generated, it can be used on the same data to predict attributes, most notably the handicap. The further the actual value is away from the predicted one the more likely it is an outlier.

Even though more than one method was used to test algorithm results, there are still addition ways in which they can be tested. One such way would be to insert bad boats into the database or intentionally make boats already there outliers. These fake outliers could then be used to test the different algorithms accuracies. Another additional testing method could be to create an entirely synthetic set of data. In this way, it would be precisely known which boats are outliers and which are not. In addition to additional tests, Gao and Tan[12] proposed some interesting methods for converting outlier algorithm scores into probabilities that would be very useful for this project.

8 Conclusion

Overall, the project proved to be very challenging. The data preprocessing consumed a large amount of time in the beginning and put the project behind schedule. Also, a lot of time was spent on skill detection because it is important, but at present it has not helped in outlier detection. Although the skill detection algorithm did significantly change the data, it did not significantly improve most of the algorithms results. At the same time though, it did not make any of the results significantly worse and one algorithm, the density based, was actually improved enough to put it on par with the others.

The main difficulty in the project was the complexity of the data. Although the data is not exactly time series, some variables, such as skill, do change over time. Also, there are many unknown variables that affect each race. Some of those variables, such as wind speed and direction, can actually vary greatly over the course of a race.

In the end, the results are not fantastic, but very encouraging. Two of the generic algorithms, LOF and density based clustering with skill detection, were able to perform far enough above random chance to be considered significant. The custom algorithm was able to further improve on the ability to detect outliers by almost a factor of 2. Overall, significant progress has been made. Even though all of the project use cases have not been completed, a usable web application has been developed that allows handicappers to view the results of outlier tests and visually inspect the data in the database. These tools will greatly improve the handicappers ability to detect and correct improperly rated boats.

References

Date last modified: April 11, 2006.


A Data Description

A.1 Data Dictionary
### Result Table – used to store individual results

<table>
<thead>
<tr>
<th>Name</th>
<th>Max Length(bytes)</th>
<th>Type</th>
<th>Format</th>
<th>Sample</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RaceID</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>BoatID</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Skipper</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>2</td>
<td>Member ID</td>
</tr>
<tr>
<td>PHRF</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>128</td>
<td>Rating</td>
</tr>
<tr>
<td>Elapsed</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>2868</td>
<td>In seconds</td>
</tr>
<tr>
<td>Corrected</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>2600</td>
<td>In seconds</td>
</tr>
<tr>
<td>EndTime</td>
<td>Dec31,9999(8)</td>
<td>datetime</td>
<td>Date Time</td>
<td>2/16/2006 2:36:00 PM</td>
<td></td>
</tr>
<tr>
<td>Cor_Sec_Behind</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>1800</td>
<td>Seconds behind 1st</td>
</tr>
<tr>
<td>Skill_Sec</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>2300</td>
<td></td>
</tr>
<tr>
<td>Skill_Sec_Behind</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>Points</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Place</td>
<td>$2^{31}-1(4)$</td>
<td>int</td>
<td>#</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td>50c(100)</td>
<td>nvarchar(50)</td>
<td>C*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Race Table – used to store information about a race

<table>
<thead>
<tr>
<th>Name</th>
<th>Max Length(bytes)</th>
<th>Type</th>
<th>Format</th>
<th>Sample</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RaceID</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>ClubID</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CourseID</td>
<td>$2^{63}-1(8)$</td>
<td>bigint</td>
<td>#</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Dec31,9999(8)</td>
<td>datetime</td>
<td>Date Time</td>
<td>2/16/2006 12:00:00 PM</td>
<td>Start time</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Length/Size</td>
<td>Category</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>AvgWindSpeed</td>
<td>smallint</td>
<td>32,767(2)</td>
<td>#</td>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>WindDirection</td>
<td>char(3)</td>
<td>3c(3)</td>
<td>CCC</td>
<td>SSW</td>
<td>Not used</td>
</tr>
<tr>
<td>Boat Table – Information about individual boats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoatID</td>
<td>bigint</td>
<td>2^63-1(8)</td>
<td>#</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>MemberID</td>
<td>bigint</td>
<td>2^63-1(8)</td>
<td>#</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ClubID</td>
<td>bigint</td>
<td>2^63-1(8)</td>
<td>#</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>Speed Demon</td>
<td>Not Used</td>
</tr>
<tr>
<td>HullIDNumber</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>134720</td>
<td>Not Used</td>
</tr>
<tr>
<td>BoatType</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>C&amp;C 30</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>nchar(1)</td>
<td>1c(2)</td>
<td>C</td>
<td>B</td>
<td>Not Used, not consistent between clubs</td>
</tr>
<tr>
<td>SailNumber</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>1122</td>
<td>Not Used</td>
</tr>
<tr>
<td>BoatDesigner</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>abc</td>
<td></td>
</tr>
<tr>
<td>YearDesigned</td>
<td>smallint</td>
<td>32,767(2)</td>
<td>#</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>Rigging</td>
<td>nvarchar(50)</td>
<td>50c(100)</td>
<td>C*</td>
<td>Standard</td>
<td></td>
</tr>
<tr>
<td>HullSpeed</td>
<td>bit</td>
<td></td>
<td>0,1</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>MotorType</td>
<td>smallint</td>
<td>32,767(2)</td>
<td>#</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MotorFuel</td>
<td>smallint</td>
<td>32,767(2)</td>
<td>#</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Type</td>
<td>#.#</td>
<td>#</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>------</td>
<td>---</td>
<td>-----------</td>
</tr>
<tr>
<td>MotorHP</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ReductionGear</td>
<td>1(1)</td>
<td>bit</td>
<td>0,1</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>RedGearRatio</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PropDiameter</td>
<td>2^31-1(4)</td>
<td>int</td>
<td>#</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>PropBlades</td>
<td>32,767(2)</td>
<td>smallint</td>
<td>#</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>PropType</td>
<td>32,767(2)</td>
<td>smallint</td>
<td>#</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PropWater</td>
<td>1(1)</td>
<td>bit</td>
<td>0,1</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>PropCenter</td>
<td>1(1)</td>
<td>bit</td>
<td>0,1</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>PropDrive</td>
<td>32,767(2)</td>
<td>smallint</td>
<td>#</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PropOutboard</td>
<td>1(1)</td>
<td>bit</td>
<td>0,1</td>
<td>False</td>
<td>Not Used</td>
</tr>
<tr>
<td>KeelType</td>
<td>32,767(2)</td>
<td>smallint</td>
<td>#</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>RudderType</td>
<td>32,767(2)</td>
<td>smallint</td>
<td>#</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LengthOverall</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>25</td>
<td>In feet</td>
</tr>
<tr>
<td>LengthWaterline</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>21</td>
<td>In feet</td>
</tr>
<tr>
<td>Beam</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>10</td>
<td>In feet</td>
</tr>
<tr>
<td>Draft</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>6</td>
<td>In feet</td>
</tr>
<tr>
<td>HullMaterial</td>
<td>50c(100)</td>
<td>nvarchar(50)</td>
<td>C* fiber</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Displacement</td>
<td>2^31-1(4)</td>
<td>int</td>
<td>#</td>
<td>8000</td>
<td></td>
</tr>
<tr>
<td>Ballast</td>
<td>2^31-1(4)</td>
<td>int</td>
<td>#</td>
<td>1600</td>
<td></td>
</tr>
<tr>
<td>Standing</td>
<td>2^31-1(4)</td>
<td>int</td>
<td>#</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Races</td>
<td>2^31-1(4)</td>
<td>int</td>
<td>#</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Approved</td>
<td>1(1)</td>
<td>bit</td>
<td>0,1</td>
<td>true</td>
<td>Not Used</td>
</tr>
</tbody>
</table>
### StandardBoats Table

This table contains the default information for each boat type and is used to fill in missing information in the Boat Table.

<table>
<thead>
<tr>
<th>ID</th>
<th>2^{31}-1(4)</th>
<th>int</th>
<th>#</th>
<th>265</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BOAT TYPE]</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>MORG27MK</td>
</tr>
<tr>
<td>RTG</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>195</td>
</tr>
<tr>
<td>T</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>NULL</td>
</tr>
<tr>
<td>CODE</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>5555M5</td>
</tr>
<tr>
<td>CF</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>S</td>
</tr>
<tr>
<td>DESIGNER</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>Morgan</td>
</tr>
<tr>
<td>YR</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>71</td>
</tr>
<tr>
<td>RIG</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>MHS</td>
</tr>
<tr>
<td>PRP</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>IBE</td>
</tr>
<tr>
<td>KL</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>FNM</td>
</tr>
<tr>
<td>RDR</td>
<td>255c(510)</td>
<td>nvarchar(255)</td>
<td>C*</td>
<td>SSP</td>
</tr>
<tr>
<td>I</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>37</td>
</tr>
<tr>
<td>ISP</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>37</td>
</tr>
<tr>
<td>J</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>11.75</td>
</tr>
<tr>
<td>JC</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>11.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>---</td>
<td>---------</td>
<td>---</td>
</tr>
<tr>
<td>P</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>31</td>
</tr>
<tr>
<td>E</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>10</td>
</tr>
<tr>
<td>IY</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>NULL</td>
</tr>
<tr>
<td>JY</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>NULL</td>
</tr>
<tr>
<td>PY</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>NULL</td>
</tr>
<tr>
<td>EY</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>NULL</td>
</tr>
<tr>
<td>LOA</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>27.5</td>
</tr>
<tr>
<td>LWL</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>25</td>
</tr>
<tr>
<td>BEAM</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>9.8</td>
</tr>
<tr>
<td>DRFT</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>4.5</td>
</tr>
<tr>
<td>BAL</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>3.3</td>
</tr>
<tr>
<td>DISP</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>7.1</td>
</tr>
<tr>
<td>[SA/D_UP]</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>22.64952</td>
</tr>
<tr>
<td>[SA/D_Down]</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>36.30138</td>
</tr>
<tr>
<td>[D/L]</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>202.85714</td>
</tr>
<tr>
<td>SPD</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>6.65</td>
</tr>
<tr>
<td>[B/D]</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>0.46478</td>
</tr>
<tr>
<td>ADDED</td>
<td>Dec31,9999(8)</td>
<td>datetime</td>
<td>Date Time</td>
<td>NULL</td>
</tr>
<tr>
<td>CHANGED</td>
<td>Dec31,9999(8)</td>
<td>datetime</td>
<td>Date Time</td>
<td>3/13/1997 12:00:00 AM</td>
</tr>
<tr>
<td>JIB</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>336.93125</td>
</tr>
<tr>
<td>SPIN</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>652.125</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>----------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>MAIN</td>
<td>1.79E+308(8)</td>
<td>float</td>
<td>#.#</td>
<td>186</td>
</tr>
<tr>
<td>REMARKS</td>
<td>100c(200)</td>
<td>nvarchar</td>
<td>C*</td>
<td>IB Version</td>
</tr>
</tbody>
</table>

Descriptive Data Summarization
The following charts and graphs summarize the project dataset.

The chart above shows that a lot of boats have participated in few races. For instance, 77 boats have only participated in 1 race. The results from these boats may not be that useful, and in fact, looking at the data shows several of them have incredibly high finish times. As a result, it may be a good idea to filter out boats who have participated in fewer than a certain threshold of races. However, experimentation is needed to see if this has a positive effect and, if so, what the best threshold is.
The above chart shows that there is a very wide distribution of boat classes. Only a few classes have a lot of boats, as can be seen in the chart below where 196 classes only have 1 member.
Another interesting chart above shows that the handicap distribution is very wide. Note that this is charted against races not individual boats which is actually similar. Below is a chart showing the number of boats per race. In the extremes one race had 1 boat and one race had 52 boats. The highest was 81 races with 7 boats participating.
Lastly, present below are some distributions of key attributes chosen for this project. I, J, P, E, DISP, LWL, and DRFT are boat measurements. The last two boxplots show the corrected seconds behind, first over all races then averaged per boat.
B Charts and Graphs
Protocol Change Notice

Notice No: PCN-07-04
Date: May 4, 2007
Subject: Code 5 Spinnaker Rating Band Change
Author: Alan Grim, Chief Handicapper

Handicappers, Members,

This Protocol Change Notice (PCN) revises the rating band for code 5 spinnakers to remove some of the advantage enjoyed by boats with code 6 spinnakers. The upper limit on code 5 spinnaker SAFS has been increased from 1.015 to 1.045, the same for the lower limit on code 6, as follows:

I. RATING ADJUSTMENTS FOR NON-STANDARD SPINNAKERS: (*NS* Spinnakers)

4. RATING ADJUSTMENT TABLE FOR NS SPINNAKERS: (SAFS)

<table>
<thead>
<tr>
<th>NOMINAL SAFS</th>
<th>SAFS RANGE</th>
<th>RATING</th>
<th>RATING ADJUSTMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00 STD</td>
<td>0.955+ – 1.045</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1.00 STD</td>
<td>1.045+ – 1.075</td>
<td>-3</td>
<td>6</td>
</tr>
</tbody>
</table>

Handicappers,

Check all code 6 spinnakers to determine which may become code 5. Submit Change Forms as required to revise rating code and as-sailed ratings.

Alan Grim
Chief Handicapper, PHRF-NW

Figure 7: Sample PHRF-NW Protocol Change Notice (PCN)
For Standard Class Boats:
- SPL = JC (pole tacked)
- BPL = JC (Sprit-tacked)

For Masthead Rigs:
- ISP = I

For Fractional Rigs
- ISP = I (Fractional Spinnakers)
- ISP = ISP (Masthead Spinnakers)

For Altered Rigs:
- H = Altered Spinnaker Hoist

Figure 8: Sailboat Dimensions
C Use Cases

Figure 15 shows the individual stakeholders and their associated use cases.
Figure 9: Effect of performance before skill detection
Figure 9: Effect of performance before skill detection
Figure 10: Two dimensional view of data before skill detection
Figure 10: Two dimensional view of data before skill detection
Figure 11: Effect of performance after skill detection
Figure 11: Effect of performance after skill detection
Figure 12: Two dimensional view of data after skill detection

(a) Distance Based Cluster Analysis

(b) Density Based Cluster Analysis
Figure 12: Two dimensional view of data after skill detection
(a) Before Skill Detection

Figure 13: Custom Algorithm Ratio Distribution
(b) After Skill Detection

Figure 13: Custom Algorithm Ratio Distribution
Figure 14: Random Guessing Accuracy Distribution
Figure 15: Project Usecase Diagram
The following are the use cases for each individual stake holder.

**Member**

Add Member - A user can create an account on the system. In addition any administrator can create a non-login account for the purposes of storing a member's information.

Edit Member - A user can edit their own information. Also, an administrator can edit the information of any member they created.

Add Boat - A user can input all the data for their boat and it will be added and linked to their member account. An administrator can add a boat and link it to any member account they created.

Edit Boat - A user can edit their boat's information. An administrator can edit the information of any boat they added.

Join Club - A user can link their member account to one or more clubs in the system. An administrator can link a member they added to a club the administrator belongs to.

View Race Results - A user can view the results of a race or the overall results of the races their boats have been in.

**Race Administrator**

Add Race - Add a race to the system including which boats will be participating.

Edit Race - Change information about a race or which boats are participating.

Add Course - Make a new course consisting of markers linked together showing all the legs, i.e. A to B to C to A

Edit Course - Edit a course's markers.

Add Marker - Add a new marker to the database.

Edit Marker - Edit a marker's information.

Input Race Results - Enter results for a race for which the race administrator belongs to one of the sponsoring clubs. An administrator can input to any race.

Modify Race Results - Edit the results of a race for which the race administrator's associated club was a sponsor. An administrator can edit any race.

**Handicapper**

Approve Boat - When a user adds a boat initially its approved flag is set to false so that its data is not used in analysis. Once the handicapper has gone over the
boats information and assigned it a handicap the boat is considered approved.

Edit Member Boat - A handicapper can edit the information of any boat associated with any of the clubs the handicapper is an administrator. An administrator can edit any boat in the database.

Add Standard Boat - Add a new standard boat from which other boats base ratings are derived.

Edit Standard Boats - Edit a standard boats information.

Administrator
Add Club - Add a new club to the database.

Edit Club - Edit a clubs information including who the administrator of the club is.

View Error Log - View system errors that have been logged to the database.
D  Source Code

Source Code 4 SQL code to import test results

```sql
use sailwater
DELETE FROM tempt
BULK INSERT tempt
FROM 'c:\SRC\soab.csv'
WITH
(
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n'
)
--coptt is used to hold a blank set of test results
--must get proper TestID first
Update coptt set TestID=125
INSERT INTO TestResult(TestID, BoatID, Passed, Value)
SELECT TestID, BoatID, Passed, Value FROM Coptt
Update TestResult SET Value=
(SELECT Value From tempt
WHERE tempt.BoatID=TestResult.BoatID
AND TestResult.TestID=125)
WHERE Exists
(SELECT Value From tempt
WHERE tempt.BoatID=TestResult.BoatID
AND TestResult.TestID=125)
--Update TestResult set Value=1 Where TestID=125 AND Passed=1
exec scoreTest 125, 0.1
exec calcPrecRecall 125, 101
```

D.1  Rapidminer process files

D.1.1  Main process file

```xml
<?xml version="1.0" encoding="windows-1252"?>
<process version="4.1">
  <operator name="Root" class="Process" expanded="yes">
    <operator name="MemoryCleanUp" class="MemoryCleanUp" breakpoints="after"/>
  </operator>
  <operator name="ExampleSource (3)" class="ExampleSource">
    <parameter key="attributes" value="Z:\702\RacesAll.aml"/>
  </operator>
</process>
```
DECLARE @CNT INT  
SET @CNT = 1  
WHILE (@CNT < 100)  
BEGIN  
  Delete FROM tempt  
  INSERT INTO tempt(BoatID)  
  SELECT TOP 55 BoatID FROM copt ORDER BY NEWID()  
  Update tempt set Value='True'  
  Update TestResult SET Passed=0, Value=0 Where TestID=127  
  Update TestResult SET Passed=  
  (SELECT Value From tempt  
   WHERE tempt.BoatID=TestResult.BoatID  
   AND TestResult.TestID=127)  
   WHERE Exists  
   (SELECT Value From tempt  
   WHERE tempt.BoatID=TestResult.BoatID  
   AND TestResult.TestID=127)  
   Update TestResult set Value=1 Where TestID=127 AND Passed=1  
   exec calcPrecRecall 127, 101  
   insert into RandT(Accuracy)  
   SELECT CAST(predT_actT AS float) / CAST(predT_actT + predF_actT AS float) * 100  
   FROM Test Where TestID=127  
   SET @CNT = @CNT + 1  
END  
SELECT Count(*), AVG(Accuracy), MIN(Accuracy), Max(Accuracy) FROM RandT
<operator name="ExampleSource (2)" class="ExampleSource" activated="no">
  <parameter key="attributes" value="Z:\702\TCSSS702Boats.aml"/>
</operator>
<operator name="ExampleSource" class="ExampleSource" activated="no">
  <parameter key="attributes" value="Z:\702\BoatSkill.aml"/>
  <parameter key="column_separators" value="\s*;\s*;\s*t"/>
</operator>
<operator name="ExampleFilter" class="ExampleFilter">
  <parameter key="condition_class" value="no_missing_attributes"/>
</operator>
<operator name="FeatureNameFilter" class="FeatureNameFilter">
  <parameter key="skip_features_with_name" value="b.AVG_SKILL_SEC||b.BoatType||r.SKILL_SEC_BEHIND"/>
</operator>
<operator name="FeatureNameFilter (2)" class="FeatureNameFilter">
  <parameter key="skip_features_with_name" value="BoatType"/>
</operator>
<operator name="FrequencyDiscretization" class="FrequencyDiscretization" breakpoints="after" activated="no">
  <parameter key="number_of_bins" value="100"/>
</operator>
<operator name="LOFOutlierDetection" class="LOFOutlierDetection" breakpoints="after" activated="no">
  <parameter key="minimal_points_upper_bound" value="60"/>
</operator>
<operator name="SVDReduction" class="SVDReduction" breakpoints="after" activated="no">
  <parameter key="keep_example_set" value="true"/>
</operator>
<operator name="DistanceBasedOutlierDetection" class="DistanceBasedOutlierDetection" breakpoints="after">
  <parameter key="number_of_outliers" value="60"/>
</operator>
<operator name="DensityBasedOutlierDetection (2)" class="DensityBasedOutlierDetection" breakpoints="after" activated="no">
  <parameter key="distance" value="0.17"/>
  <parameter key="distance_function" value="cosine distance"/>
  <parameter key="proportion" value="0.95"/>
</operator>
<operator name="DensityBasedOutlierDetection" class="DensityBasedOutlierDetection" breakpoints="after" activated="no">
  <parameter key="distance" value="0.13225"/>
  <parameter key="distance_function" value="cosine distance"/>
  <parameter key="proportion" value="0.95"/>
</operator>
<operator>
<operator name="SOMDimensionalityReduction" class="SOMDimensionalityReduction">
</operator>
<operator name="PrincipalComponentsGenerator" class="PrincipalComponentsGenerator" activated="no">
</operator>
</operator>

</process>

D.1.2 SVM process file

<?xml version="1.0" encoding="windows-1252"?>
<process version="4.1">

<operator name="Root" class="Process" expanded="yes">

<operator name="ExampleSetGenerator" class="ExampleSetGenerator" activated="no">

<parameter key="number_examples" value="200"/>
<parameter key="number_of_attributes" value="2"/>
<parameter key="target_function" value="gaussian mixture clusters"/>
</operator>

<operator name="ExampleSource (2)" class="ExampleSource">

<parameter key="attributes" value="Z:\702\RacesAll.aml"/>
</operator>

<operator name="ExampleSource" class="ExampleSource" activated="no">

<parameter key="attributes" value="Z:\702\BoatSkillSVMb.aml"/>
</operator>

<operator name="ExampleFilter" class="ExampleFilter">

<parameter key="condition_class" value="no_missing_attributes"/>
</operator>

<operator name="AttributeFilter" class="AttributeFilter">

<parameter key="condition_class" value="is_numerical"/>
</operator>

<operator name="FeatureNameFilter (2)" class="FeatureNameFilter">

<parameter key="filter_special_features" value="true"/>
<parameter key="skip_features_with_name" value="b.BoatType||r.SKILL_SEC_BEHIND||b.AVG_SKILL_SEC"/>
</operator>

<operator name="FeatureNameFilter" class="FeatureNameFilter">

<parameter key="filter_special_features" value="true"/>
<parameter key="skip_features_with_name" value="label"/>
</operator>

<operator name="FeatureGeneration" class="FeatureGeneration">

<list key="functions">

<parameter key="class" value="const[0]()"/>
</list>

</process>
<parameter key="keep_all" value="true"/>
</operator>
<operator name="AttributeSubsetPreprocessing"
class="AttributeSubsetPreprocessing" expanded="yes">
<parameter key="attribute_name_regex" value="class"/>
<operator name="Numeric2Polynominal" class="Numeric2Polynominal">
</operator>
</operator>
<operator name="ChangeAttributeRole" class="ChangeAttributeRole">
<parameter key="name" value="class"/>
<parameter key="target_role" value="label"/>
</operator>
<operator name="LibSVMLearner" class="LibSVMLearner">
<parameter key="C" value="100.0"/>
<list key="class_weights">
</list>
<parameter key="gamma" value="1.0"/>
<parameter key="keep_example_set" value="true"/>
<parameter key="svm_type" value="one-class"/>
</operator>
<operator name="ModelApplier" class="ModelApplier" breakpoints="after">
<list key="application_parameters">
</list>
</operator>
<operator name="SOMDimensionalityReduction" class="SOMDimensionalityReduction">
</operator>
</operator>
</process>
D.2 SQL Server Assembly

using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Data.Sql;
using System.Data.SqlClient;
using System.Data.SqlServer;
using System.Data.SqlTypes;
using System.Data;
using Microsoft.SqlServer.Server;

namespace sailwater
{
    public class SQL_Funcs
    {
        const decimal myversion = 1.5M;
        const int ratings_range = 3;
        const string boattable = "Boat";
        const string resulttable = "Result";

        [SqlFunction(DataAccess = DataAccessKind.Read, TableDefinition = "boatID bigint, testCount int, results nvarchar(1000)", Name = "CountTests", FillRowMethodName = "FillTestCount")]
        public static IEnumerable InitTestCount()
        {
            ArrayList output = new ArrayList();
            SqlConnection conn = null;
            try
            {
                ArrayList boats = new ArrayList();
                conn = new SqlConnection("context connection=true");
                String res = "SELECT BoatID FROM Boat";
                conn.Open();
                SqlCommand commr = new SqlCommand(res, conn);
                SqlDataReader rr = commr.ExecuteReader();
                //determine all the boats to be analyzed
                if (rr.HasRows)
                {
                    while (rr.Read())
                    {
                        long BoatID = rr.GetInt64(0); //null not allowed
                    }
                }
            }
        }
    }
}
for ID

   boats.Add(BoatID);

}
}
rr.Close();
commr.Dispose();
conn.Close();
for (int i = 0; i < boats.Count; i++)
{
   long bid = (long)boats[i];
   String sql1 = "SELECT tr.Passed FROM TestResult
   AS tr, Test As t " +
   "WHERE t.TestID=tr.TestID AND t.actual<>1
   AND tr.BoatID=@BoatID " +
   "ORDER BY tr.TestID";
   SqlParameter pBoatID = new SqlParameter("@BoatID", bid);
   conn.Open();
   SqlCommand comm = new SqlCommand(sql1, conn);
   comm.Parameters.Add(pBoatID);
   SqlDataReader r = comm.ExecuteReader();
   bool result = r.HasRows;
   if (result)
   {
      String results = "";
      int count = 0;
      while (r.Read())
      {
         if (!r.IsDBNull(0))
         {
            bool passed = r.GetBoolean(0);
            if (passed)
            {
               count++;
               if (results.Length < 1000)
               {
                  results = results + "1";
               }
            }
            else
            {
               if (results.Length < 1000)
               {
                  results = results + "0";
               }
            }
      }
   }
   conn.Close();
   }
ATestCount atc = new ATestCount(bid, count, results);
output.Add(atc);
}
r.Close();
comm.Dispose();
conn.Close();
}
//ignore errors, just make sure everything is cleaned up
catch (Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    throw new Exception("An error has occurred", e);
    //output = "There was an error: \n" + e.Message + "\n"
    + e.StackTrace;
}
return output;
}
public static void FillTestCount(Object obj, out SqlInt64 boatID, out SqlInt32 testCount, out SqlChars results)
{
    ATestCount atc = (ATestCount)obj;
    boatID = new SqlInt64(atc.boatID);
    testCount = new SqlInt32(atc.testCount);
    results = new SqlChars(atc.results);
}
public class ATestCount
{
    private long _boatID;
    private int _count;
    private string _results;

    public ATestCount(long boatID, int testCount, string results)
    {
        _boatID = boatID;
        _count = testCount;
```csharp
    _results = results;
}

public long boatID
{
    get
    {
        return _boatID;
    }
    set
    {
        _boatID = value;
    }
}

public int testCount
{
    get
    {
        return _count;
    }
    set
    {
        _count = value;
    }
}

public string results
{
    get
    {
        return _results;
    }
    set
    {
        _results = value;
    }
}

}[SqlFunction(Name = "GetVersion")]
public static SqlString getVersion()
{
    return new SqlString(myversion.ToString());
}

public static void calcSkillSecBehind()
{
    SqlConnection conn = null;
```
String output = "Process completed sucessfully";
try
{
    String sqlQuery = "select ID, RACE_ID, SKILL_SEC, SKILL_SEC_BEHIND 
    FROM Result ORDER BY RACE_ID, SKILL_SEC";
    conn = new SqlConnection("context connection=true");
    DataSet ds = new DataSet();
    SqlDataAdapter daBoat = new SqlDataAdapter(sqlQuery, conn);
    //SqlCommandBuilder cmdBldr = new SqlCommandBuilder(da);
    daBoat.UpdateCommand = new SqlCommand( 
        "UPDATE Result SET SKILL_SEC_BEHIND = @SKILL_SEC_BEHIND 
        WHERE ID = @ID", conn);
    daBoat.UpdateCommand.Parameters.Add( 
        "@SKILL_SEC_BEHIND", SqlDbType.Int, 4, "SKILL_SEC_BEHIND");
    SqlParameter parameter = daBoat.UpdateCommand.Parameters.Add( 
        "@ID", SqlDbType.BigInt);
    parameter.SourceColumn = "ID";
    parameter.SourceVersion = DataRowVersion.Original;
    daBoat.Fill(ds, "Result");
    int oldskillsec = -1;
    String oldseason = " ";
    int boatcount = 0;
    //go through and add each boats average for the year
    foreach (DataRow dataRow in ds.Tables["Result"].Rows)
    {
        boatcount++;
        String season = null;
        if (!dataRow.IsNull("RACE_ID"))
        {
            season = dataRow["RACE_ID"].ToString();
            int skillsec = 0;
            if (!dataRow.IsNull("SKILL_SEC"))
            {
                skillsec = (int)dataRow["SKILL_SEC"]; 
            }
            if (!oldseason.Equals(season))
            {
                oldskillsec = skillsec;
                oldseason = season;
            }
        }
    
    
}
dataRow["SKILL_SEC_BEHIND"] = skillsec - oldskillsec;
//dataRow["SKILL_SEC"] = 0;

int updates = daBoat.Update(ds, "Result");
SqlContext.Pipe.Send("Updated " + updates.ToString() + " of " + boatcount.ToString() + " Records ");
}

catch (Exception e)
{
  if (conn != null)
  {
    conn.Close();
  }
  throw new Exception("An error has occurred", e);
  output = e.Message;
}
SqlContext.Pipe.Send(output);
}
public static void calcNewTimes()
{
  SqlConnection conn = null;
  String output = "Process completed sucessfully";
  try
  {
    String sqlQuery = "select re.ID, re.ELAPSED, re.SKILL_SEC, " +
    "bo.SkillSlope, bo.SkillIntercept, ra.Season from
    Result AS re, " +
    "Boat AS bo, Race as ra WHERE ra.RaceID=re.RACE_ID AND bo.BoatID=re.BOAT_ID " +
    "ORDER BY ra.Season";
    conn = new SqlConnection("context connection=true");
    DataSet ds = new DataSet();
    SqlDataAdapter daBoat = new SqlDataAdapter(sqlQuery, conn);
    //SqlCommandBuilder cmdBldr = new SqlCommandBuilder(da);
    daBoat.UpdateCommand = new SqlCommand("UPDATE Result SET SKILL_SEC = @SKILL_SEC " +
    "WHERE ID = @ID", conn);

    //...
SqlParameter parameter = daBoat.UpdateCommand.Parameters.Add("@ID", SqlDbType.BigInt);
parameter.SourceColumn = "ID";
parameter.SourceVersion = DataRowVersion.Original;

daBoat.Fill(ds, "Result");
int raceyear = -1; //used for the x coordinate
String oldseason = " ";
int boatcount = 0;
//go through and add each boats average for the year
foreach (DataRow dataRow in ds.Tables["Result"].Rows)
{
    boatcount++;
    String season = null;
    if (!dataRow.IsNull("Season"))
    {
        season = dataRow["Season"].ToString();
        double adjust = 0;

        if (!oldseason.Equals(season))
        {
            raceyear++;
            oldseason = season;
        }
    }
    else
    {
        String elapsed = dataRow["ELAPSED"].ToString();
        double ielapsed = convertTime(elapsed);
        if (!dataRow.IsDBNull( "SkillSlope" ) &&
        !dataRow.IsDBNull( "SkillIntercept" ))
        {
            double slope = double.Parse(dataRow["SkillSlope" ].ToString());
            double intercept = double.Parse(dataRow["SkillIntercept" ].ToString());
            adjust = slope * raceyear + intercept;
        }
        ielapsed = ielapsed + adjust;
        dataRow["SKILL_SEC"] = Math.Round(ielapsed, 0);
        //dataRow["SKILL_SEC"] = 0;
    }
}
int updates = daBoat.Update(ds, "Result");
SqlContext.Pipe.Send("Updated " + updates.ToString() + " of " + boatcount.ToString() + " Records ");
}
catch (Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    throw new Exception("An error has occured", e);
    output = e.Message;
}
SqlContext.Pipe.Send(output);
}
private static int convertTime(String strTime)
{
    int output = 0;
    int[] iadjust = new int[] {1,60,3600,86400};
    String[] fields = strTime.Split(new Char[] { ':' });
    int cadjust = 0;
    for (int i = fields.Length - 1; i >= 0; i--)
    {
        int ifield = 0;
        try
        {
            ifield = int.Parse(fields[i]);
        }
        catch (FormatException fe)
        {
        }
        catch (OverflowException oe)
        {
        }
        if (cadjust < 4)
        {
            output = output + ifield * iadjust[cadjust];
        }
        cadjust++;
    }
    return output;
}
private static int countBoats()
int output = 0;
SqlConnection conn = null;
try
{
    String strSQL = "SELECT Count(*) FROM Boat;";
    conn = new SqlConnection("context connection=true");
    conn.Open();
    SqlCommand ticomm = new SqlCommand(strSQL, conn);
    SqlDataReader rti = ticomm.ExecuteReader();
    if (rti.HasRows)
    {
        rti.Read();
        if (!rti.IsDBNull(0))
        {
            output = rti.GetInt32(0);
        }
    }
    rti.Close();
    ticomm.Dispose();
}
catch (Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    //throw new Exception("An error has occured", e);
    output = -1;
}
if (conn != null)
{
    conn.Dispose();
}
return output;

public static void calcPrecRecall(long TestID, long ActualID)
{
    String sqlQuery = 
        "SELECT a.Passed AS actual, p.Passed AS predicted FROM " + 
        "(SELECT BoatID, Passed FROM TestResult WHERE TestID = @ActualID) AS a, " + 
        "(SELECT BoatID, Passed FROM TestResult WHERE TestID = @TestID) AS p " + 
    SqlConnection conn = null;
String output = "Process completed successfully";
try {
    conn = new SqlConnection("context connection=true");
    DataSet ds = new DataSet();
    SqlDataAdapter daTest = new SqlDataAdapter(sqlQuery, conn);
    SqlParameter pActualID = new SqlParameter("@ActualID", ActualID);
    daTest.SelectCommand.Parameters.Add(pActualID);
    SqlParameter pTestID = new SqlParameter("@TestID", TestID);
    daTest.SelectCommand.Parameters.Add(pTestID);
    daTest.Fill(ds, "Result");
    int PredT_ActT=0, PredT_ActF=0, PredF_ActT=0, PredF_ActF=0;
    // go through and set count predicted vs actual
    foreach (DataRow dataRow in ds.Tables["Result"].Rows) {
        bool act = (bool)dataRow["actual"];
        bool pred = (bool)dataRow["predicted"];
        if (pred) {
            if (act) {
                PredT_ActT++;
            } else {
                PredT_ActF++;
            }
        } else {
            if (act) {
                PredF_ActT++;
            } else {
                PredF_ActF++;
            }
        }
    }
    conn.Open();
    String SQLUpdate = "Update Test Set predT_actT=@TT,
predT_actF=@TF, " +
"predF_actT=@FT, predF_actF=@FF WHERE TestID=@TestID";
SqlCommand tcomm = new SqlCommand(SQLUpdate, conn);
SqlParameter pTT = new SqlParameter("TT", PredT_ActT);
SqlParameter pTF = new SqlParameter("TF", PredT_ActF);
SqlParameter pFT = new SqlParameter("FT", PredF_ActT);
SqlParameter pFF = new SqlParameter("FF", PredF_ActF);
SqlParameter pTestID2 = new SqlParameter("@TestID",
TestID);
tcomm.Parameters.Add(pTT);
tcomm.Parameters.Add(pTF);
tcomm.Parameters.Add(pFT);
tcomm.Parameters.Add(pFF);
tcomm.Parameters.Add(pTestID2);
int tresult = tcomm.ExecuteNonQuery();
if (tresult > 0)
{
    SqlContext.Pipe.Send("Added values for test " +
TestID.ToString() + " to database");
}
tcomm.Dispose();
conn.Close();
}
catch (Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    //throw new Exception("An error has occurred", e);
    output = e.Message + e.StackTrace;
}
SqlContext.Pipe.Send(output);
if (conn != null)
{
    conn.Dispose();
}
}

public static void scoreTest(long testID, float percentage)
{
    SqlConnection conn = null;
    String output = "Process completed successfully";
    int lastIndex = (int)Math.Round(countBoats() * percentage);
    try
    {
        String sqlQuery = "SELECT ResultID, TestID, BoatID,
Passed, Value, Notes " +
"FROM TestResult WHERE TestID=@TestID " +
"ORDER BY Value DESC";
conn = new SqlConnection("context connection=true");
DataSet ds = new DataSet();
SqlDataAdapter daBoat = new SqlDataAdapter(sqlQuery,
conn);
SqlParameter pTestID = new SqlParameter("@TestID",
testID);
daBoat.SelectCommand.Parameters.Add(pTestID);
SqlCommandBuilder cmdBldr = new SqlCommandBuilder(da);
daBoat.UpdateCommand = new SqlCommand("UPDATE TestResult SET Passed = @Passed " +
"WHERE ResultID = @ResultID", conn);
SqlParameter parameter = daBoat.UpdateCommand.Parameters.Add("@ResultID", SqlDbType.BigInt);
parameter.SourceColumn = "ResultID";
parameter.SourceVersion = DataRowVersion.Original;
daBoat.Fill(ds, "Result");

int boatcount = 0;
//go through and set the top percentage to true
foreach (DataRow dataRow in ds.Tables["Result"].Rows)
{
    boatcount++;
    if (boatcount < lastindex)
    {
        dataRow["Passed"] = true;
    }
    else
    {
        dataRow["Passed"] = false;
    }
}

int updates = daBoat.Update(ds, "Result");
SqlContext.Pipe.Send("Scored " + updates.ToString() + " of " + boatcount.ToString() + " Records ");

} catch (Exception e)
if (conn != null)
{
    conn.Close();
}
//throw new Exception("An error has occured", e);
output = e.Message + e.StackTrace;
SqlContext.Pipe.Send(output);
}
if (conn != null)
{
    conn.Dispose();
}

public static void findOutliers()
{
    String output = "Process completed sucessfully";
    //String column = "COR_SEC_BEHIND";
    String column = "SKILL_SEC_BEHIND";
    SqlConnection conn = null;
    try
    {
        conn = new SqlConnection("context connection=true");
        DataSet ds = new DataSet();
        String res;
        res = "SELECT BoatID, Races FROM Boat";
        SqlDataAdapter daResult = new SqlDataAdapter(res, conn);
        daResult.Fill(ds, "Boats");
        Hashtable boats = new Hashtable(600);
        OutlierStat os = new OutlierStat();
        //determine all the boats to be analyzed
        foreach (DataRow dataRow in ds.Tables["Boats"].Rows)
        {
            long BoatID = (long)dataRow["BoatID"];
            int Races = (int)dataRow["Races"];
            OutlierBoat ob = new OutlierBoat(BoatID);
            ob.Races = Races;
            boats.Add(BoatID, ob);
        }
        IDictionaryEnumerator Enumerator;
        Enumerator = boats.GetEnumerator();
        
}
// go through every boat set averages by comparing to
boats raced against
long comparisons = 0;
while (Enumerator.MoveNext())
{
    OutlierBoat b = (OutlierBoat)Enumerator.Value;
    String sql1 = "SELECT re.BOAT_ID " +
        "FROM (Select RACE_ID FROM Result WHERE BOAT_ID=@BoatID) AS ra, Result AS re " +
        "WHERE re.RACE_ID=ra.RACE_ID AND re.BOAT_ID<>@BoatID GROUP BY re.BOAT_ID";
    SqlParameter pBoatID = new SqlParameter("@BoatID", b.longID);
    conn.Open();
    SqlCommand comm = new SqlCommand(sql1, conn);
    comm.Parameters.Add(pBoatID);
    SqlDataReader r = comm.ExecuteReader();
    bool result = r.HasRows;
    ArrayList rlist = new ArrayList();
    if (result)
    {
        while (r.Read())
        {
            if (!r.IsDBNull(0))
            {
                rlist.Add(r.GetInt64(0));
            }
        }
    }
    r.Close();
    comm.Dispose();
    conn.Close();
    //SqlConnection.Pipe.Send("Performed " + comparisons.ToString() + " comparisons");
    comparisons = comparisons + rlist.Count;
}
for each
for (int i = 0; i < rlist.Count; i++)
{
String sql2 = "SELECT b1." + column + ", b2." + column + " FROM (SELECT " + column + ", RACE_ID FROM Result WHERE BOAT_ID=@boatA) AS b1, (SELECT " + column + ", RACE_ID FROM Result WHERE BOAT_ID=@boatB) AS b2 " + "WHERE b1.RACE_ID=b2.RACE_ID;";
SqlParameter pboatA = new SqlParameter("@boatA", b.longID);
SqlParameter pboatB = new SqlParameter("@boatB", rlist[i]);
conn.Open();
SqlCommand recomm = new SqlCommand(sql2, conn);
recomm.Parameters.Add(pboatA);
recomm.Parameters.Add(pboatB);
SqlDataReader rer = recomm.ExecuteReader();
bool reresult = rer.HasRows;
int positive = 0;
int negative = 0;
int count = 0;
if(!b.containsValue((long)rlist[i]) && boats.Contains(rlist[i])){
    //ignore boat if not in list
    OutlierBoat b2 = (OutlierBoat)boats[rlist[i]];
    if (reresult)
    {
        while (rer.Read())
        {
            if (!rer.IsDBNull(0) && !rer.IsDBNull(1))
            {
                //we now have the finish times
                int ireresult = rer.GetInt32(0)-rer.GetInt32(1);
                count++;
                if (ireresult >= 0)
                {
                    positive++;
                } else
                {
                    negative++;
                }
            }
        }
    }
}
// since it doesn't matter if the difference is positive or negative
// divide the smaller by the bigger to normalize values between 0 and 1

float ratio = 0f;
if (positive > negative)
{
    ratio = negative / positive;
}
else
{
    ratio = positive / negative;
}
OutlierValue ov = new OutlierValue(ratio, count);

b.addValue((long)rlist[i], ov);
b2.addValue(b.longID, ov);

} } rer.Close();
recomm.Dispose();
conn.Close();

} // now we have all the boats raced against calculate average score
os.addAverage(b.getAverage());

} SqlContext.Pipe.Send("Performed " + comparisons.ToString() + " comparisons");

// now all the boats are done find the outlier score threshold
OutlierValue ovt = os.getThreshold(0.5f);
// create a test
String name = "Race Comparison: " + column;
DateTime now = DateTime.Now;
String desc = "Look for patterns by comparing boats over time";
String newtest = "INSERT INTO TEST(Name, Date, Description, actual) values(@Name, @Date, @Description, @actual)";
conn.Open();
SqlCommand tcomm = new SqlCommand(newtest, conn);
SqlParameter pName = new SqlParameter("@Name", name);
SqlParameter pDate = new SqlParameter("@Date", now);
SqlParameter pDescription = new SqlParameter("@Description", desc);
SqlParameter pActual = new SqlParameter("@actual", false);

tcomm.Parameters.Add(pName);
tcomm.Parameters.Add(pDate);
tcomm.Parameters.Add(pDescription);
tcomm.Parameters.Add(pActual);
int tresult = tcomm.ExecuteNonQuery();
if (tresult > 0)
{
    SqlContext.Pipe.Send("Added test to database");
}
conn.Close();
if (tresult > 0)
{
    //get test ID
    String testID = "SELECT TestID FROM Test WHERE Name=@Name AND Date=@Date;";
    conn.Open();
    SqlCommand ticomm = new SqlCommand(testID, conn);
    SqlParameter piName = new SqlParameter("@Name", name);
    SqlParameter piDate = new SqlParameter("@Date", now);
    ticomm.Parameters.Add(piName);
tcomm.Parameters.Add(piDate);
    long ltestID = 0;
    SqlDataReader rti = ticomm.ExecuteReader();
    if (rti.HasRows)
    {
        rti.Read();
        if (!rti.IsDBNull(0))
        {
            ltestID = rti.GetInt64(0);
        }
    }
    rti.Close();
tcomm.Dispose();
    SqlContext.Pipe.Send("Test ID is: " + ltestID.ToString());
}
IDictionaryEnumerator bEnumerator;
bEnumerator = boats.GetEnumerator();
int resadded = 0;
//compute score for each boat and update in database
while (bEnumerator.MoveNext())
{
    OutlierBoat b = (OutlierBoat)bEnumerator.Value;
    string insert = "insert into TestResult(TestID, BoatID, Passed, Value, Notes) " +
        "values (@TestID, @BoatID, @Passed, @Value, @Notes)";

    SqlCommand icomm = new SqlCommand(insert, conn);
    SqlParameter pTestID = new SqlParameter("@TestID", ltestID);
    SqlParameter pBoatID = new SqlParameter("@BoatID", b.longID);
    SqlParameter pPassed = new SqlParameter("@Passed", "");
    SqlParameter pValue = new SqlParameter("@Value", b.getScore(o));
    SqlParameter pNotes = new SqlParameter("@Notes", "");

    icomm.Parameters.Add(pTestID);
    icomm.Parameters.Add(pBoatID);
    icomm.Parameters.Add(pPassed);
    icomm.Parameters.Add(pValue);
    icomm.Parameters.Add(pNotes);

    int iresult = icomm.ExecuteNonQuery();
    if (iresult > 0)
    {
        resadded++;
    }
    icomm.Dispose();
}
SqlContext.Pipe.Send("Added " + resadded.ToString() + " results");
conn.Close();
if (resadded > 0)
{
    scoreTest(ltestID, 0.1f);
    calcPrecRecall(ltestID, 101);
}
}
} //ignore errors, just make sure everything is cleaned up
catch (Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    //throw new Exception("An error has occured", e);
    output = "There was an error: \n" + e.Message + "\n" + e.StackTrace;
}
SqlContext.Pipe.Send(output);
}
public static void adjustForSkill()
{
    String output = "Process completed sucessfully";
    SqlConnection conn = null;
    try
    {
        Hashtable Classes = new Hashtable(600); //used to get class averages
        Hashtable Ratings = new Hashtable(600);
        Hashtable Boats = new Hashtable(600);

        conn = new SqlConnection("context connection=true");
        DataSet ds = new DataSet();
        SqlDataAdapter daBoat = new SqlDataAdapter("select BoatID, AVG_SEC_BEHIND, STANDARD, Races, CUR_PHRF, SkillSlope, SkillIntercept, SkillRSquared from Boat", conn);
        //SqlCommandBuilder cmdBldr = new SqlCommandBuilder(da);
        daBoat.UpdateCommand = new SqlCommand("UPDATE Boat SET SkillSlope = @SkillSlope, SkillIntercept = @SkillIntercept, SkillRSquared = @SkillRSquared " +
            "WHERE BoatID = @BoatID", conn);

        SqlParameter parameter = daBoat.UpdateCommand.Parameters.Add("@BoatID", SqlDbType.BigInt);
        parameter.SourceColumn = "BoatID";
        parameter.SourceVersion = DataRowVersion.Original;
daBoat.Fill(ds, "Boat");
String res;
res = "SELECT re.BOAT_ID, ra.Season, AVG(re.COR_SEC_BEHIND)as AVG_SEC_BEHIND" + 
" FROM Result as re, Race as ra WHERE ra.RaceID=re.RACE_ID" + 
" GROUP BY re.BOAT_ID, ra.Season ORDER BY ra.Season";
SqlDataAdapter daResult = new SqlDataAdapter(res,
conn);
daResult.Fill(ds, "Result");

//go through each one, more efficient than searching
//since we have to look at all of them anyway
//find class and rating averages(depends on Boat table being up to date) and initialize boats
int boatcount = 0;
foreach (DataRow dataRow in ds.Tables["Boat"].Rows)
{
    boatcount++;
    String BoatID = dataRow["BoatID"].ToString();
    String standard = null;
    if (!dataRow.IsNull("STANDARD"))
    {
        standard = dataRow["STANDARD"].ToString();
    }
    String cur_phrf = null;
    if (!dataRow.IsNull("CUR_PHRF"))
    {
        cur_phrf = dataRow["CUR_PHRF"].ToString();
    }
    int avgsecbehind = 0;
    if (!dataRow.IsNull("AVG_SEC_BEHIND"))
    {
        avgsecbehind = int.Parse(dataRow["AVG_SEC_BEHIND"].ToString());
        //should catch parsing exceptions
    }
    int numraces = 0;
    if (!dataRow.IsNull("Races"))
    {
        numraces = int.Parse(dataRow["Races"].ToString());
    }
    //each boat should be unique so add each one
    if (standard != null && cur_phrf != null)
    {
        boat_data bd = new boat_data(int.Parse(BoatID),
unnecessary code...
```csharp
int.Parse(standard), int.Parse(cur_phrf));
    bd.numRaces = numraces;
    Boats.Add(BoatID, bd);
}

skill_data myskill = null;
//first see if this class exists in hashtable
if (standard != null)
{
    if (Classes.Contains(standard))
    {
        myskill = (skill_data)Classes[standard];
    }
    else//if not make a blank one
    {
        myskill = new skill_data(standard);
        Classes.Add(standard, myskill);
    }
    myskill.add(numraces, avgsecbehind);
    Classes[standard] = myskill;
}

skill_graph mygraph = null;
if (cur_phrf != null)
{
    if (Ratings.Contains(cur_phrf))
    {
        mygraph = (skill_graph)Ratings[cur_phrf];
    }
    else
    {
        mygraph = new skill_graph(cur_phrf);
        Ratings.Add(cur_phrf, mygraph);
    }
    mygraph.add(numraces, avgsecbehind);
    Ratings[cur_phrf] = mygraph;
}

} SqlContext.Pipe.Send("Loaded " + boatcount.ToString() + " Boats ");
//done gathering global average information
//now add points to boats
int raceyear = -1; //used for the x coordinate
String oldseason = " ";
```
boatcount = 0;
//go through and add each boat's average for the year
foreach (DataRow dataRow in ds.Tables["Result"].Rows)
{
    boatcount++;
    String BID = dataRow["BOAT_ID"].ToString();
    String season = null;
    if (!dataRow.IsNull("Season"))
    {
        season = dataRow["Season"].ToString();
    }
    int avgsec = 0;
    if (!dataRow.IsNull("AVG_SEC_BEHIND"))
    {
        avgsec = int.Parse(dataRow["AVG_SEC_BEHIND"].ToString());
    }
    if (!oldseason.Equals(season))
    {
        raceyear++;
        oldseason = season;
    }
    if (Boats.Contains(BID))
    {
        boat_data bd = (boat_data)Boats[BID];
        bd.add(raceyear, avgsec);
        Boats[BID] = bd;
    }
}
SqlContext.Pipe.Send("Processed " + boatcount.ToString() + " Points ");
IDictionaryEnumerator Enumerator;
Enumerator = Boats.GetEnumerator();
boatcount = 0;
while (Enumerator.MoveNext()) //go through every boat set averages and calculate points
{
    boatcount++;
    boat_data b = (boat_data)Enumerator.Value;
    //this assumes no boat had a null class or rating
    if(b.myClass != 0){
        if(Classes.Contains(b.myClass.ToString())){
            b.classAverage = ((skill_data)Classes[ b.myClass.ToString() ]).getAverage();
            b.classCount = ((skill_data)Classes[ b.myClass.ToString() ]...
getRaceCount();

if(b.myRating != 0)
{
    if(Ratings.Contains(b.myRating.ToString()))
    {
        b.ratingAverage = ((skill_graph)Ratings[b.myRating.ToString()]).getAllAvg(Ratings);
        b.ratingCount = ((skill_graph)Ratings[b.myRating.ToString()]).getRaceCount();
    }
    if(b.classCount != 0 || b.ratingCount != 0){ //then there is nothing to compare against
        b.calculatePoints();
        b.linearReg();
    }
}
SqlContext.Pipe.Send("Updated " + boatcount.ToString() + " Boats ");
boatcount = 0;
//finally go through and update the Boat table
foreach (DataRow dataRow in ds.Tables["Boat"].Rows)
{
    String BID = dataRow["BoatID"].ToString();
    if (Boats.Contains(BID))
    {
        boatcount++;
        boat_data bd = (boat_data)Boats[BID];
        double si = bd.getSkillIntercept();
        double ss = bd.getSkillSlope();
        double sr = bd.getSkillRSquared();

        if (!double.IsNaN(si) && !double.IsNaN(ss) && !double.IsNaN(sr))
        {
            //SqlContext.Pipe.Send("y = " + ss.ToString() + "x + " + si.ToString() + " (" + sr.ToString() + ")");
            dataRow["SkillIntercept"] = si;
            dataRow["SkillSlope"] = ss;
            dataRow["SkillRSquared"] = sr;
        }
    }
}
int updates = daBoat.Update(ds, "Boat");
SqlContext.Pipe.Send("Updated " + updates.ToString() + " of " + boatcount.ToString() + " Records ");
catch(Exception e)
{
    if (conn != null)
    {
        conn.Close();
    }
    throw new Exception("An error has occurred", e);
    output = e.Message;
}
SqlContext.Pipe.Send(output);

public class skill_graph
{
    private int min;
    private int max;
    private float average;
    private float all_average;
    private int count;
    private int race_count;
    private int myvalue;
    private String myRating;
    private Boolean average_notran;
    public skill_graph(String rating)
    {
        myRating = rating;
        try
        {
            myvalue = int.Parse(myRating);
        }
        catch (Exception e)
        {
            myvalue = 0;
        }
        min = int.MaxValue;
        max = int.MinValue;
        average = 0;
        all_average = 0;
        count = 0;
        average_notran = true;
    }
    public float calculateAllAvg(Hashtable Ratings)
    {
        return 0;
    }
}
float newavg = 0.0f;
all_average = average;int total = count;
for (int i = (ratings_range * -1); i < ratings_range;
i++)
{
    int rat = myvalue + i;
    if (Ratings.Contains(rat.ToString()) && i != 0)
    {
        skill_graph temp = (skill_graph)Ratings[rat.ToString()];
        float tempavg = temp.getAverage() * temp.getRaceCount();
        total = total + temp.getRaceCount();
        newavg = newavg + tempavg;
    }
}
if (newavg > 0)
{
    all_average = ((average * count) + newavg) / total;
    average_notran = false;
}
return all_average;
}
public float getAllAvg(Hashtable Ratings)
{
    if (average_notran) { return calulateAllAvg(Ratings);
}
    return all_average;
}
public void add(int races, int input)
{
    float total = average * race_count;
    total = total + input * races;
    race_count = race_count + races;
    count++;
    average = total / race_count;
    if (input > max) { max = input; }
    if (input < min) { min = input; }
}
public int getRaceCount()
{
    return race_count;
}
public int getMin()

90
public class skill_data {
    private float average;
    private int count;
    private int min;
    private int max;
    private int race_count;
    private String _myclass;
    public skill_data(String myclass) {
        average = 0.0f;
        count = 0;
        race_count = 0;
        min = int.MaxValue;
        max = int.MinValue;
        _myclass = myclass;
    }
    public void add(int races, int input) {
        float total = average * race_count;
        total = total + input * races;
        race_count = race_count + races;
        count++;
        average = total / race_count;
        if (input > max) { max = input; }
        if (input < min) { min = input; }
    }
    public String getMyClass() {
public int getRaceCount()
{
  return race_count;
}

public int getMin()
{
  return min;
}

public int getMax()
{
  return max;
}

public float getAverage()
{
  return average;
}

public class boat_data
{
  private double skillIntercep;
  private double skillSlope;
  private double skillRSquared;
  private float _classAverage;
  private float _ratingAverage;
  private int _classCount;
  private int _ratingCount;
  private int myID;
  private int _myClass;
  private int _myRating;
  private int _numRaces;
  private ArrayList points;
  private ArrayList avgpoints;

  public boat_data(int id, int classid, int rating)
  {
    myID = id;
    myClass = classid;
    myRating = rating;
    points = new ArrayList();
    avgpoints = new ArrayList();
    skillIntercep = 0;
    skillSlope = 0;
  }
}
skillRSquared = 0;
_classAverage = 0;
_ratingAverage = 0;
_classCount = 0;
_ratingCount = 0;
_numRaces = 0;

public double getSkillIntercept()
{
    return skillIntercep;
}
public double getSkillSlope()
{
    return skillSlope;
}
public double getSkillRSquared()
{
    return skillRSquared;
}
public int numRaces
{
    get
    {
        return _numRaces;
    }
    set
    {
        _numRaces = value;
    }
}
public int myClass
{
    get
    {
        return _myClass;
    }
    set
    {
        _myClass = value;
    }
}
public int myRating
{
    get
    {

93
return _myRating;
}
set
{

_myRating = value;
}

public void calculatePoints()
{
    points.Clear();
    for (int i = 0; i < avgpoints.Count; i++)
    {
        ReportPoint tp = (ReportPoint)avgpoints[i];
        double skill = 0;
        //code to calculate skill
        double avg = tp.Y_Coord;
        double year = tp.X_Coord;
        //we will either use the class average (preferred)
        or the ratings as the major factor to adjust
        int classraces = _classCount - _numRaces; //dont
        count our own races
        int ratingraces = _ratingCount - _numRaces;
        if (classraces * 10 > ratingraces)
        {
            skill = (_classAverage - avg) * .9 + (_ratingAverage - avg) * .1;
        }
        else
        {
            skill = (_classAverage - avg) * .1 + (_ratingAverage - avg) * .9;
        }
        ReportPoint np = new ReportPoint(year, skill);
        points.Add(np);
    }
}

public void add(int year, int average)
{
    ReportPoint np = new ReportPoint(year, average);
    avgpoints.Add(np);
}

public int getID()
{

94
return myID;
}

public float classAverage
{
    get
    {
        return _classAverage;
    }
    set
    {
        _classAverage = value;
    }
}

public float ratingAverage
{
    get
    {
        return _ratingAverage;
    }
    set
    {
        _ratingAverage = value;
    }
}

public int classCount
{
    get
    {
        return _classCount;
    }
    set
    {
        _classCount = value;
    }
}

public int ratingCount
{
    get
    {
        return _ratingCount;
    }
    set
    {
        _ratingCount = value;
    }
}
//linear regression obtained from
//http://codebetter.com/blogs/grant.killian/archive/2004/05/21/14161.aspx
public void linearReg()
{
    double sumOfX = 0;
    double sumOfY = 0;
    double sumOfXSq = 0;
    double sumOfYSq = 0;
    double ssX = 0;
    double ssY = 0;
    double sumCodeviates = 0;
    double sCo = 0;
    if (points.Count > 1)
    {
        for (int ctr = 0; ctr < points.Count; ctr++)
        {
            ReportPoint objPoint = (ReportPoint)points[ctr];
            double x = objPoint.X_Coord;
            double y = objPoint.Y_Coord;
            sumCodeviates += (x * y);
            sumOfX += x;
            sumOfY += y;
            sumOfXSq = sumOfXSq + (x * x);
            sumOfYSq = sumOfYSq + (y * y);
        }
        sumOfXSq = Math.Round(sumOfXSq, 2);
        sumOfYSq = Math.Round(sumOfYSq, 2);
        ssX = sumOfXSq - ((sumOfX * sumOfX) / points.Count);
        ssY = sumOfYSq - ((sumOfY * sumOfY) / points.Count);
        double RNumerator = (points.Count * sumCodeviates) - (sumOfX * sumOfY);
        double RDenom = (points.Count * sumOfXSq - (Math.Pow(sumOfX, 2))) * (points.Count * sumOfYSq - (Math.Pow(sumOfY, 2)));
        sCo = sumCodeviates - ((sumOfX * sumOfY) / points.Count);
        double dblSlope = sCo / ssX;
        double meanX = sumOfX / points.Count;
        double meanY = sumOfY / points.Count;
        double dblYintercept = meanY - (dblSlope * meanX);
        double dblRSquared = Math.Pow(db1R, 2);
        double skillIntercep = dblYintercept;
        double skillSlope = dblSlope;
else
// if there is just one point use y coordinate as the
intercept
    if (points.Count > 0)
    {
        ReportPoint objPoint = (ReportPoint)points[0];
        skillIntercep = objPoint.Y_Coord;
        skillSlope = 0;
        skillRSquared = 1.0;
    }
}

public class ReportPoint
{
    private double _dblX;
    private double _dblY;

    public ReportPoint(double X_Coordinate, double Y_Coordinate)
    {
        _dblX = X_Coordinate;
        _dblY = Y_Coordinate;
    }

    public double X_Coord
    {
        get { return _dblX; }
        set { _dblX = value; }
    }

    public double Y_Coord
    {
        get { return _dblY; }
        set { _dblY = value; }
    }

    public String toString()
    {
        return "(" + _dblX.ToString() + "," + _dblY.ToString() + ")";
    }
}

public class OutlierBoat
{
    private long lID;
}
private Hashtable values;
private float ov_ratio_avg, ov_count_avg;
private int ov_ratio, ov_count;
private int comparisons;
private int _races;
public OutlierBoat(long ID)
{
    lID = ID;
    values = new Hashtable(600);
    ov_count_avg = 0;
    ov_ratio_avg = 0;
}
public int Races
{
    get
    {
        return _races;
    }
    set
    {
        _races = value;
    }
}
public OutlierValue getAverage()
{
    int count = (int)Math.Round((double)ov_count_avg);
    OutlierValue ov_avg = new OutlierValue(ov_ratio_avg, count);
    return ov_avg;
}
public double getScore(OutlierValue threshold)
{
    double output = 0;
    IDictionaryEnumerator Enumerator;
    Enumerator = values.GetEnumerator();
    int count = 0;
    //compute score for each boat
    while (Enumerator.MoveNext())
    {
        OutlierValue val = (OutlierValue)Enumerator.Value;
        if (val.compareTo(threshold) < 0)
        {
            output++;
        }
        if (val.count >= threshold.count)
        {
            output++;
        }
    }
}
count++;
}
}
int dividend = 1;
if (comparisons > 0)
{
    dividend = comparisons;
}
//return (output / dividend) * (_races / dividend);
return output;
}
public void addValue(long boatID, OutlierValue ov)
{
    //ignore values with only 1 point
    if (ov.count > 1)
    {
        float ftemp = ov.ratio_avg * ov.ratio;
        ftemp = ftemp + ov.ratio;
        ov.ratio++;          
        ov.ratio_avg = ftemp / ov.ratio;

        float ctemp = ov.count_avg * ov.count;
        ctemp = ctemp + ov.count;
        ov.count++;          
        ov.count_avg = ctemp / ov.count;

comparisons++;
} 
    values.Add(boatID, ov);
}
public bool containsValue(long boatID)
{
    return values.Contains(boatID);
}
public long longID
{
    get
    {
        return lID;
    }
}
}
public class OutlierValue
{
    private float _ratio;
    private int _count;
}
public OutlierValue(float ratio, int count) {
    _ratio = ratio;
    _count = count;
}

public float ratio {
    get {
        return _ratio;
    }
}

public int count {
    get {
        return _count;
    }
}

//possibly return a more accurate number ie. myvalue - other
public int compareTo(OutlierValue other) {
    int output = 0;
    if (_ratio <= other.ratio && _count >= other.count) {
        output = -1;
    } else {
        output = 1;
    }
    return output;
}

public class OutlierStat {
    private ArrayList boats;
    public OutlierStat() {
        boats = new ArrayList();
    }
    public void addAverage(OutlierValue ov) {
        boats.Add(ov);
    }
}
public OutlierValue getThreshold(float percent) {
    long count = 0;
    ArrayList aratio = new ArrayList();
    for (int i = 0; i < boats.Count; i++)
    {
        count = count + ((OutlierValue)boats[i]).count;
        // cant be a pattern if there is only 1 value
        if (((OutlierValue)boats[i]).count > 1)
        {
            aratio.Add(((OutlierValue)boats[i]).ratio);
            // SqlContext.Pipe.Send(((OutlierValue)boats[i]).ratio.ToString());
        }
    }
    int icount = (int)Math.Round((double)count / boats.Count);
    aratio.Sort();
    int index = (int)Math.Round((double)aratio.Count*percent);
    SqlContext.Pipe.Send("index: " + index.ToString());
    float fratio = (float)aratio[index];
    OutlierValue ov = new OutlierValue(fratio, icount);
    return ov;
}
D.3 MS Access VB Scripts

CalculateAvgSecBehind
Option Compare Database

Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String
    strSQL = "SELECT AVG(COR_SEC_BEHIND) AS AVGSEC FROM Results WHERE BOAT_ID="

    'Copy To
    Set rstData = rstdb.OpenRecordset("SELECT * FROM Boats")
    rstData.MoveFirst

    With rstData
        Do While Not .EOF
            'Copy From
            Set rst = db.OpenRecordset(strSQL & !ID & ";")
            rst.MoveFirst
            .Edit
            'Attributes to copy
            !AVG_SEC_BEHIND = rst!AVGSEC
            .Update
            .MoveNext
        Loop
    End With

    MsgBox "Finished Updating Records"

    rstData.Close
    rst.Close
    Set rstData = Nothing
    Set rst = Nothing
End Sub

CalculatePlace
Option Compare Database

102
Public Sub Main()

    Dim strTime, strTemp As String
    Dim intCount, intPlace As Long
    Dim intNewRace, intCurRace As Long

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb

    'table to search
    Set rstData = rstdb.OpenRecordset("SELECT * FROM Results ORDER BY RACE_ID, COR_SEC")
    rstData.MoveFirst
    intCurRace = rstData!RACE_ID
    intPlace = 0
    With rstData
        Do While Not .EOF
            intNewRace = !RACE_ID
            If intNewRace = intCurRace Then
                intPlace = intPlace + 1
            Else
                intPlace = 1
                intCurRace = intNewRace
            End If
            .Edit
            !COR_PLACE = intPlace
            intCount = intCount + 1
            .Update
        .MoveNext
        Loop
    End With

    MsgBox "Finished Updating " & intCount & " Records"

    rstData.Close
    Set rstData = Nothing

End Sub

CalculateSecBehind
Option Compare Database

Public Sub Main()
Dim strTime, strTemp As String
Dim intCount, intTime As Long
Dim intNewRace, intNewTime, intCurRace, intCurTime As Long

Dim rstData As DAO.Recordset
Dim rstdb As DAO.Database
Set rstdb = CurrentDb

' table to search
Set rstData = rstdb.OpenRecordset("SELECT * FROM Results ORDER BY RACE_ID, COR_SEC")
rstData.MoveFirst
intCurRace = rstData!RACE_ID
intCurTime = rstData!COR_SEC
With rstData
    Do While Not .EOF
        intNewRace = !RACE_ID
        intNewTime = !COR_SEC
        If intNewRace = intCurRace Then
            intTime = intNewTime - intCurTime
        Else
            intTime = 0
            intCurRace = intNewRace
            intCurTime = intNewTime
        End If
        .Edit
        !COR_SEC_BEHIND = intTime
        intCount = intCount + 1
        .Update

        .MoveNext
    Loop
End With
MsgBox "Finished Updating " & intCount & " Records"

rstData.Close
Set rstData = Nothing

End Sub

ConvertTimeToDistance
Option Compare Database
Public Sub Main()

Dim strFinish, strCorFinish, strCorrected, strElapsed As String
Dim intCount As Integer
Dim intCorrected, intElapsed, intPHRF, intTemp As Long
Dim intFinish, intCorFinish, intDiff As Long
Dim dblDistance As Double

Dim rstData As DAO.Recordset
Dim rstdb As DAO.Database
Set rstdb = CurrentDb

'table to search
Set rstData = rstdb.OpenRecordset("SELECT * FROM Results WHERE RACE_ID < 300;")
rstData.MoveFirst

With rstData
    Do While Not .EOF
        strCorrected = !CORRECTED
        intCorrected = StringToSec(strCorrected)
        strFinish = !FINISH
        intFinish = StringToSec(strFinish)
        strCorFinish = !COR_FINISH
        intCorFinish = StringToSec(strCorFinish)
        intDiff = intFinish - intCorFinish
        intElapsed = intCorrected + intDiff
        strElapsed = SecToString(intElapsed)
        intPHRF = !phrf
        If intPHRF <> 0 Then
            intTemp = intDiff / intPHRF * 100 'keep two decimal places
            dblDistance = intTemp / 100
            .Edit
            !DISTANCE = dblDistance
            !ELAPSED = strElapsed
            intCount = intCount + 1
            .Update
        End If
        .MoveNext
    Loop
End With

MsgBox "Finished Updating " & intCount & " Records"
rstData.Close
Set rstData = Nothing
End Sub

Public Function StringToSec(ByVal strTime As String) As Long
    Dim intTime As Long
    Dim intA, intB As Integer

    intTime = 0
    intA = InStr(strTime, ":")

    If intA > 0 Then
        intB = InStr(intA + 1, strTime, ":")
        If intB > 0 Then
            strTemp = Mid(strTime, 1, intA - 1)
            intTemp = Val(strTemp)
            intTime = intTemp * 3600 'hours
            strTemp = Mid(strTime, intA + 1, intB - (intA + 1))
            intTemp = Val(strTemp)
            intTime = intTime + intTemp * 60 'minutes
            strTemp = Mid(strTime, intB + 1)
            intTemp = Val(strTemp)
            intTime = intTime + intTemp
        End If
    End If
    StringToSec = intTime
End Function

Public Function SecToString(ByVal intTime As Long) As String
    Dim strTemp As String
    Dim intHour, intMin As Long
    intHour = Fix(intTime / 3600)
    strTemp = intHour & ":
    intTime = intTime - intHour * 3600
    intMin = Fix(intTime / 60)
    If intMin < 10 Then
        strTemp = strTemp & "0"
    End If
    strTemp = strTemp & intMin & ":"
    intTime = intTime - intMin * 60
    If intTime < 10 Then
        strTemp = strTemp & "0"
    End If
    strTemp = strTemp & intTime
SecToString = strTemp

End Function

ConvertTimeToSecs

Option Compare Database

Public Sub Main()

Dim strTime, strTemp As String
Dim intCount As Long
Dim intTime As Long
Dim intA, intB, intTemp As Long

Dim rstData As DAO.Recordset
Dim rstdb As DAO.Database
Set rstdb = CurrentDb

' table to search
Set rstData = rstdb.OpenRecordset("SELECT * FROM Results")
rstData.MoveFirst

With rstData
    Do While Not .EOF
        strTime = !CORRECTED
        intTime = 0
        intA = InStr(strTime, ":")
        If intA > 0 Then
            intB = InStr(intA + 1, strTime, ":")
            If intB > 0 Then
                strTemp = Mid(strTime, 1, intA - 1)
                intTemp = Val(strTemp)
                intTime = intTemp * 3600 'hours
                strTemp = Mid(strTime, intA + 1, intB - (intA + 1))
                intTemp = Val(strTemp)
                intTime = intTime + intTemp * 60 'minutes
                strTemp = Mid(strTime, intB + 1)
                intTemp = Val(strTemp)
                intTime = intTime + intTemp
            End If
        End If
        .Edit
        !COR_SEC = intTime
        intCount = intCount + 1
        .Update
    Loop
End With

End If
End If
.r.MoveNext
Loop
End With

MsgBox "Finished Updating " & intCount & " Records"

rstData.Close
Set rstData = Nothing

End Sub

CopyBoatData
Option Compare Database

Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String

    'From Matching Attribute
    strCriteria = "[Boat_Name] =?"

    'Copy From
    Set rst = db.OpenRecordset("SELECT * From SYCResults")
rst.MoveFirst

    'Copy To
    Set rstData = rstdb.OpenRecordset("SELECT * FROM Boats WHERE CLUB='syc'")
    With rstData
    Do While Not .EOF
        'To Matching attribute
        rst.FindFirst (strCriteria & !Name & ":")
    If rst.NoMatch Then
        MsgBox "No entry found"
    Else
        .Edit
        'Attributes to copy

108
Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String

    'From Matching Attribute
    strCriteria = 

    'Copy From
    Set rst = db.OpenRecordset(“SELECT * From Boats WHERE CLUB=’syc’”)  
    rst.MoveFirst

    'Copy To
    Set rstData = rstdb.OpenRecordset(“SELECT * FROM SYCResults”)

    With rstData
        Do While Not .EOF
            'To Matching attribute
            rst.FindFirst (strCriteria & !Boat_Name & ””)
            If rst.NoMatch Then

            End If
            rst.MoveFirst
            .MoveNext
        Loop
    End With

End Sub
MsgBox "No entry found"
Else
    .Edit
    'Attributes to copy
    !BoatID = rst!ID
    .Update
End If
rst.MoveFirst
    .MoveNext
Loop
End With
MsgBox "Finished Updateing Records"

rstData.Close
rst.Close
Set rstData = Nothing
Set rst = Nothing
End Sub

CopyRaceID
Option Compare Database

Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String

    Dim intCount As Integer
    intCount = 0
    'From Matching Attribute
    strCriteria = "[ORIGINAL] ="

    'Copy From
    Set rst = db.OpenRecordset("SELECT * From Races")
    rst.MoveFirst

    'Copy To
    Set rstData = rstdb.OpenRecordset("SELECT * FROM Results WHERE RENAME_FLAG='w'")
With rstData
  Do While Not .EOF
    'To Matching attribute
    rst.FindFirst (strCriteria & !RACE_ID & "")
    If rst.NoMatch Then
      MsgBox "No entry found"
    Else
      .Edit
      'Attributes to copy
      !RACE_ID = rst!ID
      intCount = intCount + 1
      .Update
    End If
    rst.MoveFirst
  .MoveNext
  Loop
End With

MsgBox "Finished Updating " & intCount & " Records"

rstData.Close
rst.Close
Set rstData = Nothing
Set rst = Nothing
End Sub

DecimalScaleDISP
Option Compare Database

Public Sub Main()
  Dim strFinish, strCorFinish, strCorrected, strElapsed As String
  Dim intCount As Integer
  Dim dblDisp As Double
  Dim rstData As DAO.Recordset
  Dim rstdb As DAO.Database
  Set rstdb = CurrentDb

  'table to search
  Set rstData = rstdb.OpenRecordset("SELECT DISP FROM BoatTypes WHERE ID > 423 AND DISP IS NOT NULL;")
  rstData.MoveFirst
With rstData
    Do While Not .EOF

        If !DISP <> 0 Then
            dblDisp = !DISP
            dblDisp = dblDisp / 1000
            .Edit
            !DISP = dblDisp
            intCount = intCount + 1
            .Update
        End If
    .MoveNext
Loop
End With

MsgBox "Finished Updating " & intCount & " Records"

rstData.Close
Set rstData = Nothing

End Sub

FindDuplicates
Option Compare Database

Public Sub Main()

    Dim strCurRace As String
    Dim strNewRace As String
    Dim strNewBoat As String
    Dim strNewBoata As String
    Dim strCurBoat As String
    Dim intCount As Integer
    Dim intTotal As Integer
    Dim d As New Dictionary
    d.CompareMode = BinaryCompare

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String

    'On Error GoTo NoPart
    'table to search
Set rstData = rstdb.OpenRecordset("SELECT * FROM SYCResults ORDER BY Race, Start, Finish, Place")
rstData.MoveFirst
strCurRace = rstData!Race
intCount = 0
strNewBoata = rstData!Boat_Name
d.Add strNewBoata, "1"
rstData.MoveNext
intTotal = 1
With rstData
  Do While Not .EOF
    strNewRace = !Race
    strNewBoat = !Boat_Name
    If strNewRace = strCurRace Then
      'if not found just add it
      If Not d.Exists(strNewBoat) Then
        d.Add strNewBoat, "1"
      Else 'otherwise we have duplicate names so change race
        strCurBoat = d.Item(strNewBoat)
        Dim num As Integer
        num = Val(strCurBoat)
        num = num + 1
        d.Item(strNewBoat) = "" & num
        d.Remove (strNewBoat)
        d.Add strNewBoat, "" & num
        .Edit
        !Race = !Race & " " & num
        intCount = intCount + 1
        .Update
      End If
    Else 'start a new cycle
      strCurRace = strNewRace
      d.RemoveAll
      d.Add Key:=strNewBoat, Item:="1"
    End If
  End If
  .MoveNext
  intTotal = intTotal + 1
  Loop
End With

MsgBox "Finished Updating " & intCount & " Records"

113
End Sub

FindMostCurrentPHRF
Option Compare Database

Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String

    'From Matching Attribute
    strCriteria = "[BOAT_ID] ="

    'Copy From
    Set rst = db.OpenRecordset("SELECT * FROM Results, Boats, Races
WHERE Results.BOAT_ID=Boats.ID AND Results.RACE_ID=Races.ID ORDER BY
Races.YEAR DESC;")
rst.MoveFirst

    'Copy To
    Set rstData = rstdb.OpenRecordset("SELECT * FROM Boats")
rstData.MoveFirst

    With rstData
        Do While Not .EOF
            'To Matching attribute
            rst.FindFirst (strCriteria & !ID)
            If rst.NoMatch Then
                MsgBox "No entry found"
            End If
        End Do
    End With
End Sub
Else
  .Edit
  'Attributes to copy
  !CUR_PHRF = rst!phrf
  .Update
End If
rst.MoveFirst
  .MoveNext
Loop
End With

MsgBox "Finished Updating Records"

rstData.Close
rst.Close
Set rstData = Nothing
Set rst = Nothing

End Sub

FindNumberOfRaces
Option Compare Database

Public Sub Main()
  Dim rst As DAO.Recordset
  Dim db As DAO.Database
  Set db = CurrentDb
  Dim strCriteria As String
  Dim rstData As DAO.Recordset
  Dim rstdb As DAO.Database
  Set rstdb = CurrentDb
  Dim strSQL As String
  strSQL = "SELECT COUNT(*) AS NUM FROM Results WHERE BOAT_ID=
  'Copy To
  Set rstData = rstdb.OpenRecordset("SELECT * FROM Boats")
rstData.MoveNext

  With rstData
    Do While Not .EOF
      'Copy From
      Set rst = db.OpenRecordset(strSQL & !ID & ";")
rst.MoveFirst
      .Edit
      'Attributes to copy
      !NUM_OF_RACES = rst!num
  End With
End Sub
Public Sub Main()
    Dim rst As DAO.Recordset
    Dim db As DAO.Database
    Set db = CurrentDb
    Dim strCriteria As String

    Dim rstData As DAO.Recordset
    Dim rstdb As DAO.Database
    Set rstdb = CurrentDb
    Dim strSQL As String
    strSQL = "INSERT INTO Races (CLUB, YEAR, NAME, ORIGINAL) VALUES (";

    'Copy To
    Set rstData = rstdb.OpenRecordset("SELECT DISTINCT Results.RACE_ID, Races.NAME, Races.CLUB, Races.YEAR FROM Results, Races WHERE Results.RACE_ID=Races.ID AND RENAME_FLAG='w';")
    rstData.MoveFirst

    With rstData
        Do While Not .EOF
            'Copy From
            .Execute (strSQL & !CLUB & "," & !Year & "," & !Name & "," & !RACE_ID & ");")

            .MoveNext
        Loop
    End With
End Sub

MsgBox "Finished Updating Records"
rstData.Close
Set rstData = Nothing
End Sub
D.4 Java Data Parser and Preprocessor

package sailparser;

import com.sun.org.apache.bcel.internal.generic.ARRAYLENGTH;
import java.awt.Frame;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.regex.*;
import javax.swing.JFileChooser;

public class Parse {
  /*private String[] tokens = {
    "b[\s]{2}\b", "[0-9]\s[0-9]", "FUN USY", "LIA STW", "NTO CAT", "JOY CAT", "II OLS"};
  private int[] offsets = {0,1,3,3,3,3,3};
  private int[] endoffset = {0,-1,-3,-3,-3,-3,-3};
  private String[] match = {
    "<\[/DIV>"};
  private String[] replace = {"
  private int columns = 8;*/
  /*private String[] tokens = {
    "<\[/Tt][Dd]>"};
  private int[] offsets = {6};
  private int[] endoffset = {6};
  private String[] match = {
    "</DIV>$"};
  private String[] replace = {""};
  private int columns = 8;*/
  private String[] tokens = {
    "<\[/Tt][Dd]>"};
}
private int[] offsets = {0};
private int[] endoffset = {5};
private String[] match = {"<\[/Tt][Dd]>$","<\[/Tt][Rr]>"};
private String[] replace = {"",""};
private int columns = 12;
private String clusterdir;

/** Creates a new instance of Main */
public Parse() {
   clusterdir = "";
}

public void run() {
   Frame parent = new Frame();
   final JFileChooser fc = new JFileChooser();
   File[] files = null;
   FileInputStream fstream; DataInputStream in = null; BufferedReader br;
   FileWriter fw; BufferedWriter bw = null;
   //select directory
   fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
   int returnVal = fc.showOpenDialog(parent);
   if (returnVal == JFileChooser.APPROVE_OPTION) {
      File file = fc.getSelectedFile();
      files = file.listFiles();
   } else {
      files = new File[0];
   }
   try {
      fw = new FileWriter("output.txt");
   bw = new BufferedWriter(fw);
   } catch (IOException ioe) {
      System.out.println(ioe.getStackTrace());
   }
   for(int i = 0; i < files.length; i++) {
      if (files[i].isFile()) {
         try {
            FileInputStream fstream = new FileInputStream(files[i]);
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            String strLine;
            //Read the file line by line
while ((strLine = br.readLine()) != null) {
  //do any defined replaces
  for(int k = 0; k < match.length; k ++){
    strLine = strLine.replaceAll(match[k],replace[k]);
  }

  String outline = "";
  int count = 1;

  int lastend = 0;
  int[] starts = new int[columns + 1];
  for(int k = 0; k < columns; k++){
    starts[k]=-1;
  }
  starts[0] = 0;
  starts[columns] = strLine.length();
  //go through all of the tokens
  for(int j = 0; j < tokens.length; j++){
    Pattern pattern =
    Pattern.compile(tokens[j]);
    Matcher matcher =
    pattern.matcher(strLine);
    while (matcher.find()) {
      //System.out.print("Match: "+matcher.start()+"-"+matcher.end()+
      \"\" +matcher.group()+\"\"");
      if(count < columns){
        starts[count] = matcher.start()+offsets[j];
      }
      count++;
    }

    //System.out.println("\n");
  }
  Arrays.sort(starts);
  for(int k = 0; k < columns; k++){
    int pstart = starts[k];
    int pend = starts[k+1];
    if(pstart < pend &\n
    String extracted = strLine.substring(pstart,pend);
    extracted = extracted.trim();
    outLine += extracted;
    if(k < columns) outLine += ", ";
}
else{
    System.out.print("Error: " + pstart + "-" + pend + ")
}
}

outLine += "" + files[i].getAbsolutePath();
//System.out.println(outLine);
//write the output if it has the correct amount of columns
if(count == columns){
    bw.write(outLine);
    bw.newLine();
}

in.close();
}catch(IOException ioe){
    System.out.println(ioe.getStackTrace());
}
}

try{
    bw.close();
}catch(IOException ioe){
    System.out.println(ioe.getStackTrace());
}

parent.dispose();
System.out.println("Run Complete");
}/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    Parse p = new Parse();
    p.run();
}

/*@ Preprocess.java *
* Created on August 30, 2007, 12:10 PM *
* To change this template, choose Tools | Template Manager *
* and open the template in the editor."
package sailparser;

import java.awt.*;
import java.io.*;
import java.util.regex.*;
import javax.swing.*;

/**
 * @author Owner
 */
public class Preprocess {

    private String[] match = {
        "\n", "<DIV style="[^"\"]+">", "<span class="[^"\"]+">", "</span>", 
        "<font[^\"]*>", "<tr[^\"]*>", "</font>",
    };
    private String[] replace = {
        "", "", "", "", "",
    };
    private String tokenpattern = ";top:[0-9]+PX;";
    private boolean isunique = true; /*
    private String[] match = {
        "[^\n\f\r]", "<\[Tt\][Dd]\[^>]*\]">", "<font[^>]*>";
    };
    private String[] replace = {
        "", "", "", "", ""
    };
    private String tokenpattern = "<\[Tt\][Rr]\]">;"
    private boolean isunique = false;
    /**
     * Creates a new instance of Preprocess */
    public Preprocess() {
    }

    public void run() {
        Frame parent = new Frame();
        final JFileChooser fc = new JFileChooser();
        File[] files = null;
        FileInputStream fstream; DataInputStream in = null; BufferedReader br;
        FileWriter fw; BufferedWriter bw = null;
        //select directory
        fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int returnVal = fc.showOpenDialog(parent);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            files = file.listFiles();
        } else {
            files = new File[0];
        }
        for (int i = 0; i < files.length; i++) {
            if (files[i].isFile()) {
                try{


            }
```java
fw = new FileWriter("new_" + files[i].getName());
bw = new BufferedWriter(fw);
fstream = new FileInputStream(files[i]);
in = new DataInputStream(fstream);
br = new BufferedReader(new InputStreamReader(in));

String oldToken = "";
String strLine;
//Read the file line by line
while ((strLine = br.readLine()) != null) {
    //get the line token first
    Pattern pattern =
        Pattern.compile(tokenpattern);
    Matcher matcher =
        pattern.matcher(strLine);
    String newToken = "";
    if(matcher.find()) newToken = matcher.group();
    //System.out.println("\" + newToken + \" - \" + \
    oldToken + \\
    //do any defined replaces
    for(int k = 0; k < match.length; k++){
        strLine = strLine.replaceAll(match[k],replace[k]);
    }
    if(isunique){
        if(!oldToken.equals(newToken) && !oldToken.equals("")){
            strLine = \\
            //strLine = \\
            oldToken + "\n";
        }
    }else if(!newToken.equals("")){
        strLine = \\
        oldToken = newToken;
    }else if(oldToken.equals("")) & newToken.equals(""){
        //strLine = \\
    }in.close();
bw.close();
}catch(IOException ioe){
    System.out.println(ioe.getStackTrace());
}
}
parent.dispose();
System.out.println("Run Complete");
```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    Preprocess p = new Preprocess();
    p.run();
}
Source Code 6 Batch file to get data from database

REM get aggregate dataset
REM get full dataset
pause