NSU Florida
NOVA SOUTHEASTERN UNIVERSITY

Nova Southeastern University
NSUWorks

CEC Theses and Dissertations

College of Engineering and Computing

2017

Pulsar Search Using Supervised Machine Learning

John M. Ford
Nova Southeastern University, jmford94@gmail.com

This document is a product of extensive research conducted at the Nova Southeastern University College of Engineering and Computing. For more information on research and degree programs at the NSU College of Engineering and Computing, please click here.

Follow this and additional works at: https://nsuworks.nova.edu/gscis_etd


Part of the Computer Sciences Commons

Share Feedback About This Item

NSUWorks Citation

John M. Ford. 2017. *Pulsar Search Using Supervised Machine Learning.* Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (1001)
https://nsuworks.nova.edu/gscis_etd/1001.

This Dissertation is brought to you by the College of Engineering and Computing at NSUWorks. It has been accepted for inclusion in CEC Theses and Dissertations by an authorized administrator of NSUWorks. For more information, please contact nsuworks@nova.edu.

Pulsar Search Using Supervised Machine Learning

by

John M. Ford

A Dissertation Report Submitted in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Computer Science

Graduate School of Computer and Information Sciences
Nova Southeastern University
May 1, 2017

We hereby certify that this dissertation, submitted by John Ford, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

Sumitra Mukherjee, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Michael J. Laszlo, Ph.D. Date
Dissertation Committee Member

_____ _____

Francisco J. Mitropoulos, Ph.D. Date
Dissertation Committee Member

Approved:

_____ _____

Yong X. Tao, Ph.D., P.E., FASME Date
Dean, College of Engineering and Computing

SIGNED ORIGINAL ON FILE AT NSU

College of Engineering and Computing
Nova Southeastern University

2017

An Abstract of a Dissertation Report Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Pulsar Search Using Supervised Machine Learning

by

John M. Ford

Pulsars are rapidly rotating neutron stars which emit a strong beam of energy through mechanisms that are not entirely clear to physicists. These very dense stars are used by astrophysicists to study many basic physical phenomena, such as the behavior of plasmas in extremely dense environments, behavior of pulsar-black hole pairs, and tests of general relativity. Many of these tasks require a large ensemble of pulsars to provide enough statistical information to answer the scientific questions posed by physicists. In order to provide more pulsars to study, there are several large-scale pulsar surveys underway, which are generating a huge backlog of unprocessed data. Searching for pulsars is a very labor-intensive process, currently requiring skilled people to examine and interpret plots of data output by analysis programs. An automated system for screening the plots will speed up the search for pulsars by a very large factor. Research to date on using machine learning and pattern recognition has not yielded a completely satisfactory system, as systems with the desired near 100% recall have false positive rates that are higher than desired, causing more manual labor in the classification of pulsars. This work proposed to research, identify, propose and develop methods to overcome the barriers to building an improved classification system with a false positive rate of less than 1% and a recall of near 100% that will be useful for the current and next generation of large pulsar surveys. The results show that it is possible to generate classifiers that perform as needed from the available training data. While a false positive rate of 1% was not reached, recall of over 99% was achieved with a false positive rate of less than 2%. Methods of mitigating the imbalanced training and test data were explored and found to be highly effective in enhancing classification accuracy.

ACKNOWLEDGEMENTS

This accomplishment would not have been possible without the support and encouragement of many people. First and foremost, my parents and grandparents encouraged my curiosity in the world, and encouraged me to learn about and explore it. I regret that my grandparents and my mother have already passed away, but I am glad that my father is here to see this finally accomplished.

I would like to thank my family, friends and colleagues for their understanding when I was absorbed in this research and the preliminary studies. It took a tremendous amount of time to complete this project, and that was time taken from the people around me.

I was inspired to pursue this dissertation by Dr. Nicole Radziwill, a colleague who pursued her own Ph.D research in mid-career, and encouraged me to do the same. Dr. Scott Ransom and Dr. Paul Demorest provided enthusiastic encouragement and shared their knowledge of pulsars and search techniques. I am indebted to the Parkes High Time Resolution Universe project for providing easy access to the data, and in particular to Dr. Rob Lyon and Dr. Vincent Morello for providing the data used in their research.

Finally, I would like to thank the faculty and staff of Nova Southeastern University College of Engineering and Computing and especially my dissertation committee, Dr. Francisco Mitropolis, Dr. Michael Laszlo, and the chair, Dr. Sumitra Mukherjee, for their support and guidance, and their patience with me during the times it seemed like no progress was being made. I truly appreciate your knowledge, experience, and help throughout my time at NSU.

Contents

List of Tables

List of Figures

Figure Page

Chapter 1
Introduction

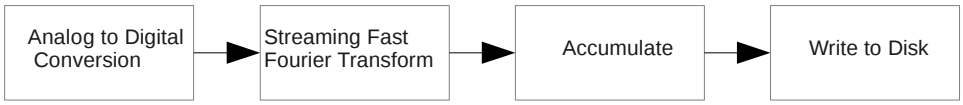
Background

Pulsars are rapidly rotating neutron stars which emit a strong beam of energy through mechanisms that are not entirely clear to physicists. These very dense neutron stars are used by astrophysicists to study many phenomena. Fundamental tests of general relativity can be made using them as tools (D. Lorimer and Kramer, 2005). Currently, an experiment is being performed to try to detect gravitational waves by the North American Nanohertz Observatory for Gravitational Waves (2012) by studying the timing variations of an array of pulsars scattered around the celestial sphere. These experiments in fundamental physics require a large set of pulsars for study, providing the impetus for systematically searching the sky for new pulsars. Pulsars discovered in ongoing pulsar surveys, as well as in the reprocessed data of several archival surveys, are continually being added to the census of pulsars in the nearby universe.

Problem Statement

Searching for pulsars is a very labor-intensive process, currently requiring skilled people to examine and interpret plots of data output by analysis programs. An automated system for screening the plots would speed up the search for pulsars by a very large factor. Research to date on using machine learning and pattern recognition has not yielded a satisfactory system. This work proposes to research, identify, and propose methods to overcome the barriers to such a system.

Figure 1.1: Pulsar data collection process



Searching for Pulsars

Searching for pulsars is a compute-intensive and human-intensive task. The raw data is collected from a large (40 to 100 meter diameter) radio telescope at a very high sample rate with a specialized radio telescope receiver system and custom hardware signal processor. The signal processor receives the signal from the telescope, digitizes it, and performs a Fourier transform on the time series, changing it into a power spectrum with many frequency channels. Each channel represents the instantaneous signal power in a small frequency band 1 to 4 megahertz (MHz) wide. The instantaneous power spectrum is sent to a computer where it is stored for later processing. Figure 1.1 shows a simplified schematic of the data collection process.

Once the time series of power spectral data is stored on disk, it can be processed to search for pulsars. Not only is the pulsar signal very faint and buried in random background noise, it is also often obscured with radio frequency interference (RFI). When many samples of a faint signal buried in random noise are averaged together, the signal-to-noise ratio is improved because the random noise cancels while the (non-random) signal builds up with the addition of each time sample (Hassan & Anwar, 2010). However, RFI is not a random process and does not average out over time, so the first step in the processing is to attempt to find and remove any RFI signals from the data to avoid confusing the RFI with an astronomical signal.

In order to use the previously described averaging process to build up the signal-

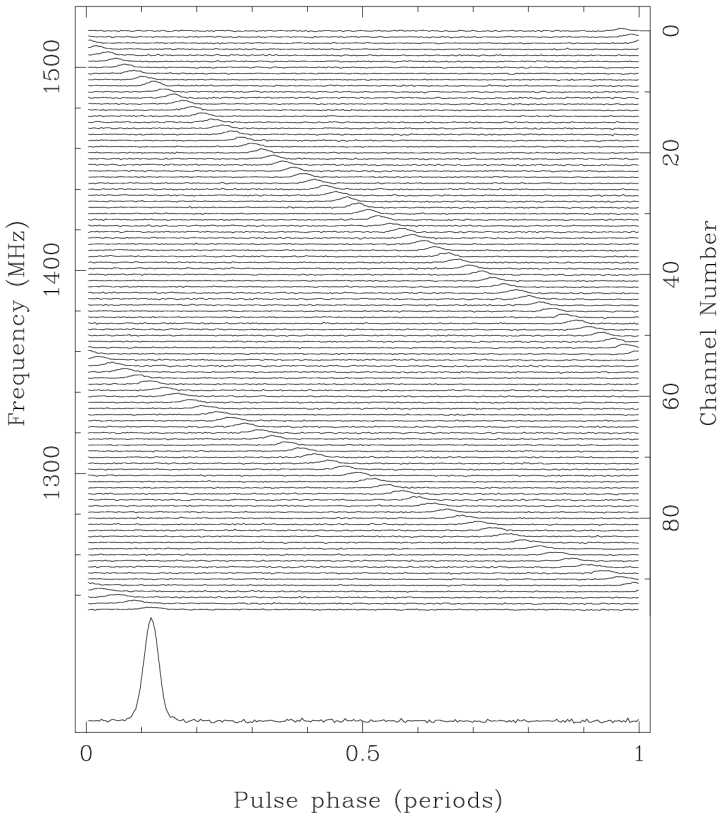
to-noise ratio, in the case of pulsar search data processing, an added complication is that the pulse period is unknown, and so the averaging process must be performed at many different trial pulse periods (known as *folding*) to search the pulse period parameter space and find the true pulsar period.

Another parameter space that must be searched is the *Dispersion Measure* (DM) space. Dispersion is caused by the interstellar medium, and is different for every pulsar, depending on its distance and the number of electrons in the interstellar medium in the direction of the pulsar. Dispersion causes the lower frequencies of the signal to arrive later than the higher frequencies. This smears out, or *disperses*, the pulse. This smearing will completely obliterate the pulse if the signal is not de-dispersed before folding. Figure 1.2 shows the effects of dispersion on the time of arrival of the pulse. Note that in the upper part of the figure, the signal in each frequency channel across the band is spread nearly evenly in time. The lower panel shows the results of de-dispersing the pulse and summing all of the frquency channels. The pulse is clearly visible with high signal to noise ratio.

The degree of dispersion given by the dispersion measure parameter must also be searched at the same time as the pulsar period, creating a combinatorial explosion in the number of output data sets created from each input data set. The output data sets are usually presented to the scientist graphically and these plots are called *diagnostic plots* (described below). A simplified diagram of this signal processing pipeline is described in Figure 1.3. Complete details of the signal processing pipeline in typical use may be found in D. Lorimer and Kramer (2005) or in McLaughlin (2011).

The final step in the classical analysis of pulsar search data is the manual examination of diagnostic plots like the one in Figure 1.4. The plots are examined

Figure 1.2: Dispersion from the Interstellar Medium. From D. Lorimer and Kramer (2005)



.

Figure 1.3: Pulsar signal processing pipeline

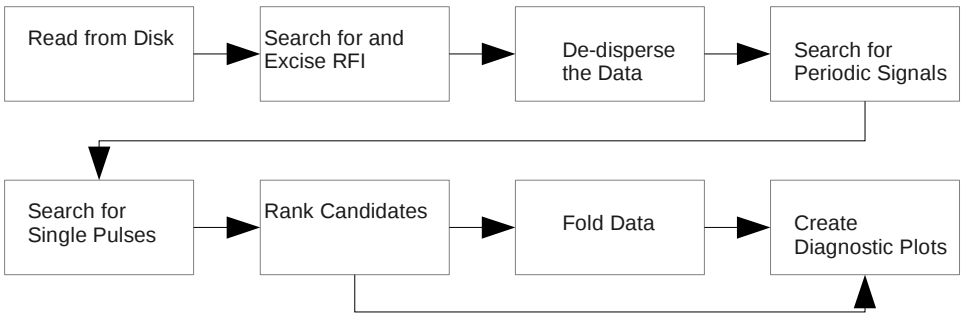
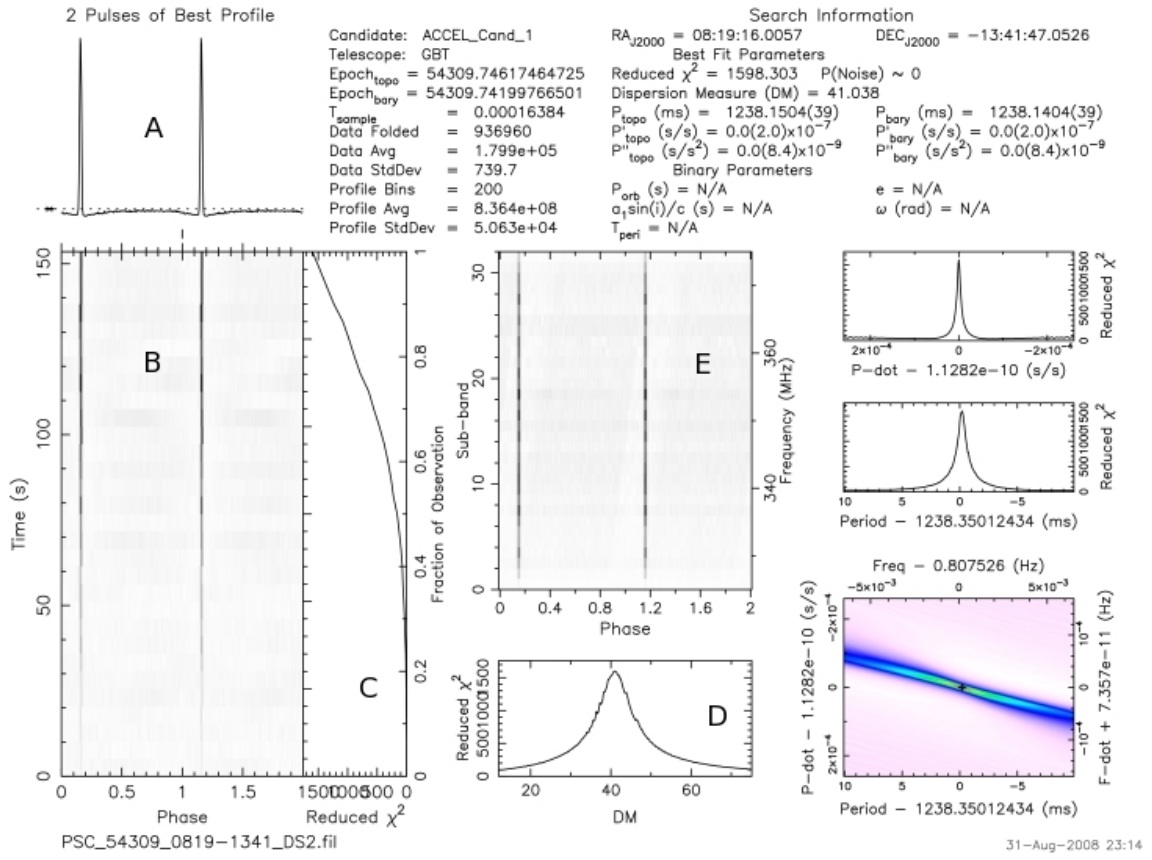


Figure 1.4: Diagnostic plot for Pulsar J0820-1350



by astronomers or trained volunteers (often interested high school or undergraduate students!) to determine if a pulsar signal is likely present in the data. Diagnostic plots that appear to contain a pulsar signal are saved, and that region of the sky is observed again to confirm whether a pulsar is present. For a particular pointing on the sky, a few hundred or a thousand diagnostic plots might be created, resulting in millions of plots being created from a large-scale survey.

Figures 1.4 – 1.6 show diagnostic plots derived from Green Bank Telescope 350 MHz data collected for the Pulsar Search Collaboratory (Heatherly, 2013) (Adapted with permission). Figure 1.4 shows the plot for known pulsar J0820-1350. Data shown in tabular form in the upper right corner of the plot give the summary of the

statistics of the processed data. An important measure is the value of Reduced χ^2, It is important to note that the Reduced χ^2 value builds over time as the data are processed and compared to the model. This can be seen in the Reduced χ^2 vs Time subplot (Labeled "C" in Figure 1.4).

There are four main features in the plots that astronomers use when deciding if a candidate could be a pulsar:

- First, in the *2 Pulses of Best Profile* subplot in the upper left-hand corner of the figure, labeled "A", the peaks should be significantly above the noise floor. Compare the error bars in the lower left corner of the same subplot in Figures 1.4 and 1.5.

- Second, in the *Phase vs Time* subplot, labeled "B", vertical lines in phase with the peaks should appear throughout the entire observation time, unless, as in this case, the telescope beam is drifting across the sky, in which case the pulsar should smoothly come into the beam and drift out later. This indicates that the signal is continuous in time, as pulsar signals usually are. In this plot, the pulsar drifted into the beam at the beginning of the pointing, and then was constant throughout the rest of the pointing. The data file that follows this one in time would show the pulsar strongly in the beginning of the scan, and show it drifting out of the beam at the end of the scan.

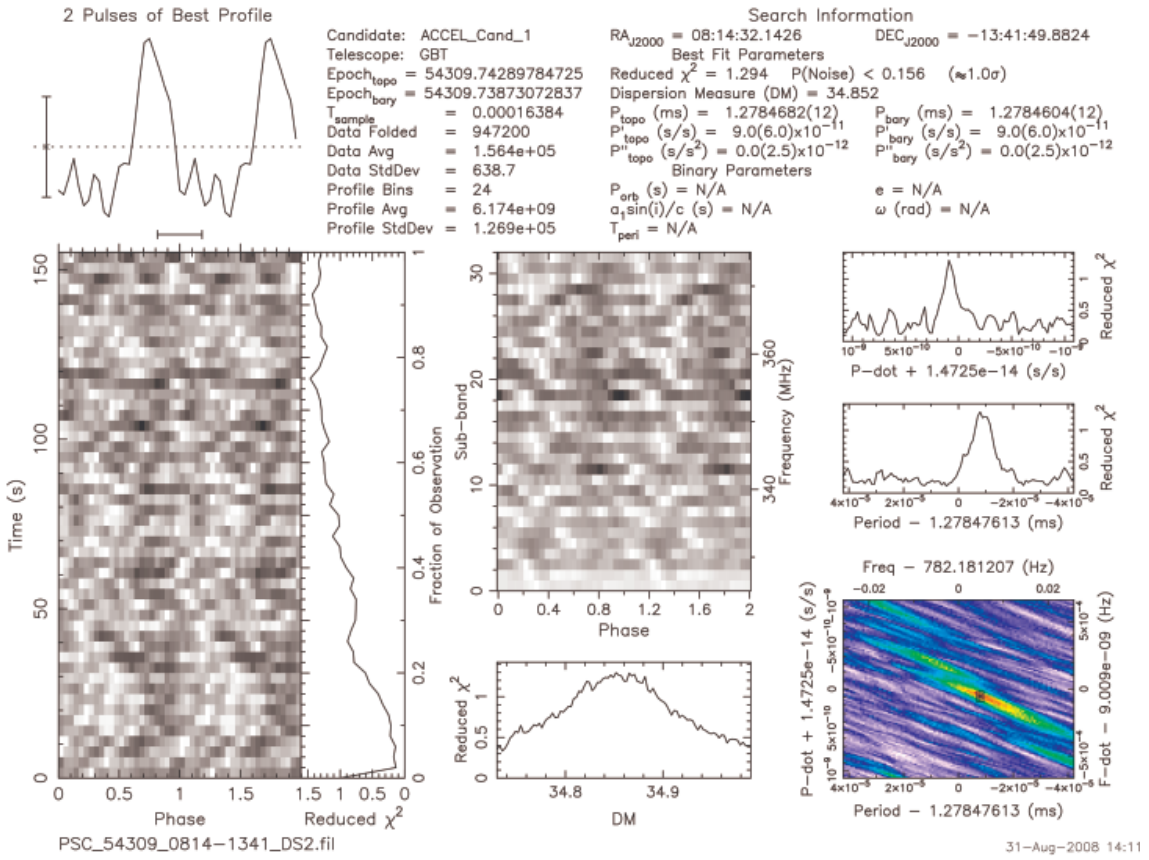
- Third, in the *Phase vs Frequency* subplot, labeled "E", the vertical lines should also span most of the frequency space, indicating the signal is a broadband signal, which is characteristic of a pulsar signal. Compare this with Figure 1.6, a plot of a man-made interference signal, where the signal is present only at a narrow band of frequencies.

- Fourth, a bell-shaped curve in the *DM vs Reduced* χ^2 subplot, labeled "D", shows that the signal's reduced χ^2 value depends strongly on DM, peaking at the trial DM, as it should. Compare this with the plot in Figures 1.5 and 1.6, where there is no strong dependence of Reduced χ^2 with DM.

Figure 1.5 shows a plot generated from a signal that is normal background noise. Note the lack of systematic signals that were present in Figure 1.4. The error bars on the *2 Pulses of Best Profile* subplot are nearly as large as the pulse peaks. The *Phase vs Time* and *Phase vs Frequency* subplots are disorganized and appear random. The *DM vs reduced* χ^2 subplot shows only a very weak dependency of χ^2 on DM.

Figure 1.6 shows a plot generated from a man-made interference signal. Some of the charateristics seem to be the same as the pulsar signal in Figure 1.4. However, other characteristics are clearly different. The *Phase vs Frequency* subplot shows a strong signal at only a small band of frequencies, rather than across the band as might be expected of a true pulsar signal. The *DM vs Reduced* χ^2 subplot shows no strong peak in the curve.

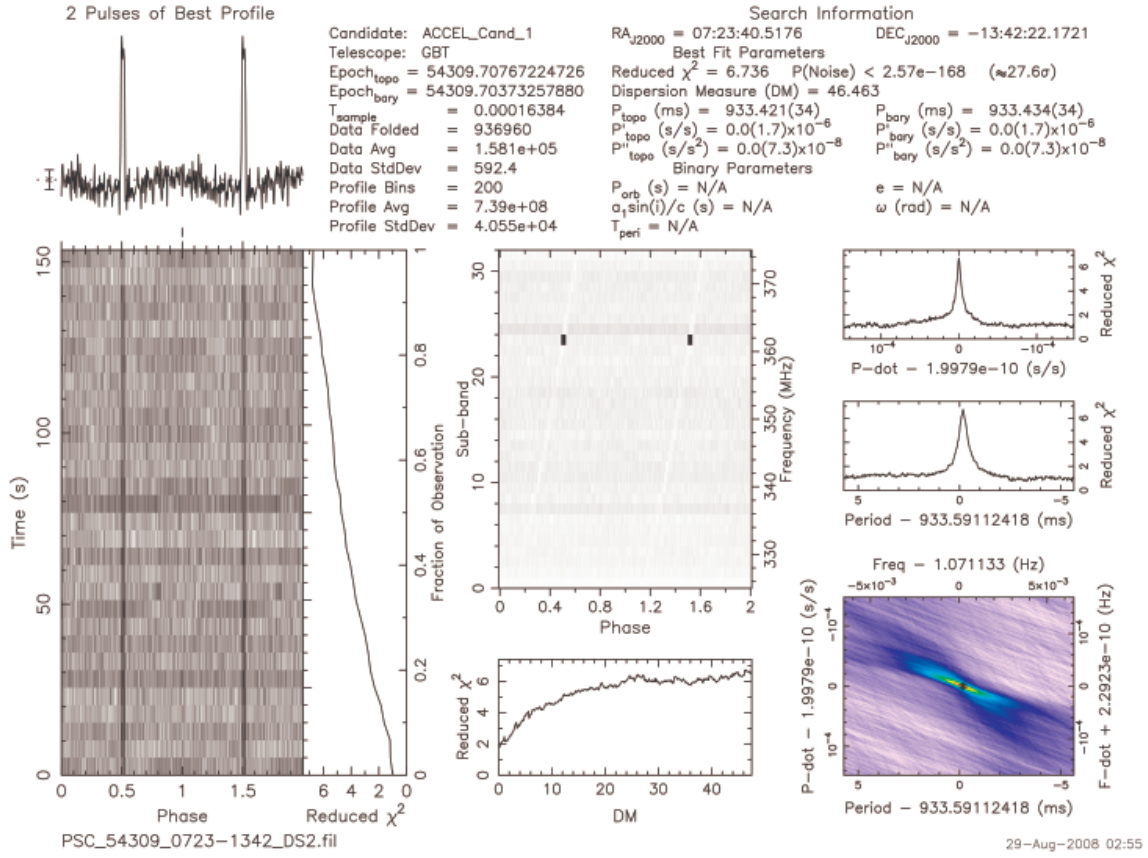
Figure 1.5: Diagnostic plot of background noise in the direction of 0814-1341



Research Goal

The first published attempt to use a machine learning approach to detect pulsars in diagnostic data was published by Eatough et al. (2010). Their work used 14,400 pointings out of the Parkes Multibeam Pulsar Survey (PMPS), one of the largest comprehensive searches undertaken to date (Manchester et al., 2001). The 14,400 beams were processed through their standard pulsar search pipeline, generating 2.5 million candidate plots containing possibly all types of pulsars: binary pulsars, slow pulsars, and millisecond pulsars. Out of these 2.5 million plots, 501 pulsars were found by manual means, yielding a very small $\frac{501}{2{,}500{,}000} = 0.02\%$ success ratio. Such a small success ratio makes the job of manually viewing the plots tedious and extremely

Figure 1.6: Diagnostic plot of radio frequency interference in the direction of 0723-1342



error-prone.

An automated method of screening the candidates is needed to reduce the human effort needed to examine the candidate plots. Eatough et al. (2010) used an Artificial Neural Network (ANN) as a binary classifier to screen the 2.5 million candidate plots.

The goal of the proposed research was to develop an improved method for pulsar identification using supervised machine learning techniques. Private communications with pulsar scientists (Demorest, 2013; Ransom, 2013) set the goals as

- The false positive rate should be less than 5%.
- Precision of the new algorithm should be greater than the 3.6% of the current

state of the art.

- Recall greater than 99% (Less than 1% of the pulsars are missed)

These are very difficult specifications to meet, but the relaxed false positive rate specification from that achieved by Eatough et al. (2010) provided a ray of hope. In fact, a paper (Morello et al., 2014) reports that they have acheived a 100 percent recall and just 0.64 percent false positive rate on a large data set. See Chapter 2 for more on this paper. Additionally, a recent paper by Lyon, Stappers, Cooper, Brooke, and Knowles (2016) provides more analysis and a mathematical background for choosing features.

Prior Research and Significance

As noted in the background section, there are more and larger pulsar surveys being planned for current and future telescopes. The Five Hundred Meter Spherical Telescope being built in China will be able to find as many as 5,000 pulsars in a short amount of time (Smits, Lorimer, et al., 2009). The Square Kilometer Array (SKA) is expected to find more than 20,000 pulsars (about 10 times the number currently known!) (Smits, Kramer, et al., 2009). As the ratio of candidates to confirmed pulsars is about 10,000:1, upwards of 200 million candidates will need to be examined to complete the SKA survey. Clearly, it is impractical to examine all 200 million candidates with human eyes.

Research in automated pulsar search is in its infancy. Research in machine learning for building classifiers is a mature area. One characteristic of the pulsar search problem that makes this research interesting is the imbalanced training data available. The very small number of known pulsars and the large amount of data available means that the training must be done with a very imbalanced data set, or else large

amounts of data without pulsars must be discarded. Other machine learning techniques have not been used in identifying pulsars thus far, and it is worthwhile to consider some of these techniques. One such technique is the *Support Vector Machine*. The SVM can perform well in cases where there is an imbalance in the classes available in the training data, since only training examples nearest the maximum-margin hyperplane separating the two classes are required. Seiffert, Khoshgoftaar, Van Hulse, and Napolitano (2007) discusses the issues involved with classifying very rare events using the SVM.

Experience with many large-scale pulsar surveys (Eatough et al., 2010; D. R. Lorimer, 2011) has shown the need for a more automated system for classifying pulsar candidates. Eatough et al. (2010) proposed an ANN utilizing eight input features derived from plots generated by the pulsar search software output. This ANN, when trained, reduced the number of candidates that had to be viewed from 2.5 million to 13,000. Unfortunately, it detected only about 92% of the pulsars in the test data set. Changing the scoring added to the number of false positives but did not materially increase the success rate. An attempt to improve on these results was recently published (Bates et al., 2012), but the effort failed in spite of using 14 more parameters in the ANN inputs. This lack of success with more features suggests that the features chosen for the studies by Eatough et al. (2010) and Bates et al. (2012) may have not been the best features to use for classification, leading to failure of the trained network to generalize well. Studies of optimizing the feature sets used in classification problems have been done (Van Hulse, Khoshgoftaar, Napolitano, & Wald, 2009), and algorithms have been developed to automate the feature optimization. Weston et al. (2000) outlines a procedure for choosing features to be used in support vector machine classifiers. Since an ANN is a special case of a support vector machine (Vapnik, 1999),

these techniques may be applied to the problem.

The work of both Eatough et al. (2010) and Bates et al. (2012) used simple statistics derived from the pulsar candidate plots as inputs to the ANN. These statistics may not include critical information that was lost when the plots were reduced to simple statistics. In addition, millisecond pulsars, a class of pulsars especially prized for their stable spin periods and emissions, are underrepresented in the training data they used in training their ANN. Both of these problems need to be studied further as part of this work.

Eatough comments that the CPU time to run the ANN is a very small fraction of the time spent creating the candidate plot in the first place (2 minutes vs. 3 hours). From that perspective, it is feasible to cascade several classifiers together to extract more pulsars from the stream and exclude more false positives. Again, for this application it is important to minimize false negative results.

In addition to the ANN research described above, work has been done on improving the algorithms for scoring the candidate plots. Using these improved techniques, Keith et al. (2009) found 28 more pulsars in a data set that had already been mined. Their method included performing statistical analyses on the data making up the diagnostic plots. They analyzed the *subband, DM curve, and pulse profile* plots, and combined the output scores from these analyses to form a score to decide whether a candidate was a strong candidate.

A very recent system called PEACE: Pulsar Evaluation Algorithm for Candidate Extraction (Lee et al., 2013) demonstrated the utility of careful feature selection in an algorithm similar to the one described above. This paper used six *quality factors* in the scoring of the pulsar candidate. They achieved good results using these six factors, with 100% of the known pulsars in the data set ranked in the top 3.7% of the

candidates. These experiences will be used to help define the candidate feature sets to be optimized.

An effort was made by Lyon et al. (2016) to rigorously derive a feature set from the data provided by (Morello et al., 2014). That feature set is used for the experiments in this research.

Other Machine Learning Methods

If one subscribes to the No Free Lunch theorems (D. Wolpert & Macready, 1997), then one should try to use the a priori knowledge of the problem to choose the best possible algorithm match to the problem. Support vector machines will be investigated to find out if they offer advantages over the ANN, and to provide a structure for investigating the feature set to be used.

Naive Bayes classifiers were investigated as part of the background work for this paper but do not seem to be applicable to this work due to the difficulty of establishing prior probabilities due to the rarity of the pulsars in the data.

Machine Learning in the face of unbalanced training data

Work by Seiffert et al. (2007) on very imbalanced data sets shows that even with the minority making up as little as 0.1% of the examples, effective classifiers can be built using techniques to mitigate the effects of the unbalanced data set. Their work used 11 different learning algorithms and built over 200,000 classifiers. Their conclusions show that data sampling, a technique for selecting a subset of data for learning purposes, can increase the performance of the classifiers.

Barriers and Issues

Automated pulsar searching is a difficult problem due to the relative scarcity of exemplars and the huge volumes of sometimes poor-quality data. Pulsar signals also

exhibit a great deal of variability from pulsar to pulsar. Some produce very narrow pulses, while others produce wide pulses. The pulsar signal, even in the best of cases, is weak and buried in noise. Combined with terrestrial interference, the signals are very difficult to find even for human eyes. In addition, machine learning is a very new topic in the pulsar search community. Only a few papers have been published on the subject (Bates et al., 2012; Eatough et al., 2010; Morello et al., 2014). Although there is other on-going research by astronomers, there are no computer science researchers involved in these studies.

Finally, the data sets themselves are many tens to hundreds of terabytes in size. Even with permission to use the data, it is unwieldy to copy it around the internet. Fortunately, astronomers are willing to help by physically copying the data onto media and shipping it to one another once it is public.

Unbalanced Training Data

One particularly difficult problem involves the data available to train a machine learning algorithm. For example, in Eatough et al. (2010), only 259 pulsars and 1625 non-pulsar signals were available for training. This was culled out of a total of 2,500,000 diagnostic plots. Particularly scarce in the training data are the millisecond pulsars. These have some characteristics that differ from normal pulsars that caused them to be missed in larger proportion than the normal pulsars in the ANN studies. Some ideas to counter this problem are to use some of the ideas of Hu, Liang, Ma, and He (2009) and Seiffert, Khoshgoftaar, Van Hulse, and Napolitano (2009) in synthesizing and augmenting the minority exemplars. This is an opportunity for research as much as it is a barrier!

New Area of Research

As the application of machine learning techniques is new to the field of pulsar astronomy, there has not been much research published to guide the way forward. There is enthusiasm in the pulsar astronomy community for these techniques, and a great deal of data is publicly or semi-publicly available for experimentation, but there are many data formats and differing data quality across the different data sets.

Definition of Terms

Binary pulsar A pulsar in orbit around a companion star, or vice versa.

Dispersion The effect on a broadband electromagnetic signal traveling through the interstellar medium that imparts a frequency dependent delay to the signal.

Dispersion Measure A measurement of the delay experienced by the pulsar signal as it transits the interstellar medium. It is affected by the electron density along the line of sight. The units of dispersion measure are parsecs per cubic centimeter.

Folding Averaging a time series signal using a particular repetition period

Interstellar Medium The gases and ions between stars. Space is not quite a vacuum.

Millisecond Pulsar A pulsar with a period measured in milliseconds.

Pulsar A rapidly rotating neutron star that emits a powerful beam of energy as it rotates

Pulse Period The period at which the pulsar signal repeats

Power Spectrum A measurement of the signal power as a function of frequency

Radio Frequency Interference Unwanted signals generated by humans or natural processes that interfere with reception of desired signals

Reduced χ^2 The measure of how a signal differs from an assumed model, which in the case of radio astronomy data is white gaussian noise.

Slow pulsar A pulsar with a period approaching or exceeding 1 second.

Summary

Pulsars are rapidly rotating neutron stars which emit a strong beam of energy through mechanisms that are not entirely clear to physicists. These very dense stars are used by astrophysicists to study many basic physical phenomena, such as the behavior of plasmas in extremely dense environments, behavior of pulsar-black hole pairs, and other extreme physics. Many of these tasks require a large ensemble of pulsars to provide enough information to complete the science.

In order to provide more pulsars to study, there are several large-scale pulsar surveys underway, which are generating a huge backlog of unprocessed data. Searching for pulsars is a very labor-intensive process, currently requiring skilled people to examine and interpret plots of data output by analysis programs. An automated system for screening the plots would speed up the search for pulsars by a very large factor. Eatough et al. (2010) recounts a private communication (Lee) describing an automated pulsar candidate ranking algorithm. A method of using scores that indicate the degree of similarity between the candidate and a typical pulsar is described in another paper (Keith et al., 2009). Neither of these last two methods used ANNs or other machine learning techniques to inspect plots, rather the algorithms were used as a filter to limit the number of candidate plots that needed to be viewed.

The approach of using artificial neural networks is significant since it may allow an automated detection and classification pipeline to be used to relieve the burgeoning backlog of pulsar search data and to allow economical reprocessing of archived search data. Reprocessing of data is desirable when advances in search algorithms raise the possibility of additional pulsars being found in the archived data. Some of the most sought-after and rare pulsars are those found in binary systems, where the pulsar is orbiting another object, or where two pulsars are orbiting each other. These binary pulsars require advanced search algorithms that take into account the acceleration of the pulsars due to the presence of its orbiting companion. Before recent increases in the available computing power available to researchers, these advanced algorithms were not typically run on all data due to the extra computational complexity. Older data sets may yield some of these exotic systems if the data are reprocessed with new algorithms including automated detection and classification methods.

Research to date on using machine learning and pattern recognition has not yielded a satisfactory system, with more than 7% of the pulsars in a test data set missed by the first automated system to attempt this problem. Later systems have claimed 100% recall, but this needs further research to confirm. This work proposes to research, identify, and propose methods to overcome the barriers to building an improved classification system with a false positive rate of less than 0.5% and a recall of near 100%.

Chapter 2

Review of the Literature

Machine Learning in Astronomy

Machine learning algorithms were not applied in pulsar searching prior to Eatough et al. (2010). They have been used in other branches of astronomy in such applications as the classification of galaxies (Lahav, Naim, Sodré, & Storrie-Lombardi, 1996; Zhang, Li, & Zhao, 2009), in estimation of redshifts of stars in the Sloan Digital Sky Survey (Firth, Lahav, & Somerville, 2003), and in the classification of microlensing events from large variability studies (Belokurov, Evans, & Du, 2003).

One of the new topics in astronomy is transient detection. Transient objects appear in astronomical images from many causes. Some are asteroids, comets, and other near-earth phenomena, while others are more exotic, such as Gamma-ray bursts (GRB), supernovae, and variable stars. New telescopes are being built to image the sky rapidly, so that transients can be detected quickly, giving other telescopes time to follow up on them before they fad away. A prime example of this is the work by Morgan et al. (2012) in applying machine learning to classify GRBs as coming from sources with a particularly interesting redshift. The purpose is to maximize use of the avaiable follow-up time to study these more interesting GRBs. This work uses a Random Forest set of classifiers to do the work. Another work using the Random Forest approach is from Brink et al. (2013) that is used to look for real transient events from the Palomar Transient Factory, a telescope dedicated to looking for transient events. This system correctly classifies 92% of real transients in the data, with a 1% false positive rate. For transient detection in the Pan-STARRS1 Medium Deep Survey, a machine learning system has been applied to the problem, yielding a 90%

recall rate with a 1% false positive rate (Wright et al., 2015).

Machine Learning in Pulsar Search

Eatough et al. (2010) used an Artificial Neural Network (ANN) as a binary classifier to screen the 2.5 million candidate plots. The effort used a set of 8 input features derived from candidate plots as the input vector to the ANN. In addition to these 8 features, an additional small trial was done with a set of 12 features with minimal effect on the success rates. The following are the features extracted from the data forming the diagnostic plots and used in the feature vector (the last 4 features listed were not used in the full experiment):

- Pulse profile signal-to-noise ratio (SNR)
- Pulse profile width
- χ^2 of the fit to the theoretical dispersion measure (DM) - SNR curve
- Number of DM trials with SNR > 10
- χ^2 of the fit to the optimized theoretical dispersion measure (DM) - SNR curve
- χ^2 of the fit to the theoretical acceleration - SNR curve
- number of acceleration trials with SNR > 10
- χ^2 of the fit to the optimized theoretical acceleration - SNR curve
- RMS scatter in subband maxima
- Linear correlation across subbands
- RMS scatter in subintegration maxima
- Linear correlation across subintegrations

From the data, which contained 501 pulsars, training data consisting of 259 input vectors from known pulsars and 1625 input vectors from non-pulsar signals were used to train the ANN. An additional validation data set was reserved from the data set with 28 pulsar signals and 899 non-pulsar signals. The remainder of the data, which contained the rest of the pulsars, was used as a test sample. Unfortunately, some of the test sample contained pulsars that were used in training, which makes some of the statistics a bit optimistic.

This ANN was very effective in reducing the number of plots to be examined by eye from 2.5 million to 13,000, a reduction of a factor of almost 200. However, the ANN recovered only about 92% of the known pulsars in the data set, which is not acceptable to scientists (Ransom, 2013), who would demand a false negative rate of at worst a few percent before entrusting the search to the machine.

Bates et al. (2012) also attempted to use this method to find pulsars, but were not as successful as Eatough et al. (2010). They used 22 features in the input vector, including all of the features used by Eatough et al. (2010). Their success rate was no better with more features. This lack of success with more features suggests that Eatough et al. (2010) and Bates et al. (2012) may have not chosen the features to use for classification in an optimal way, leading to failure of the trained network to generalize well.

Recently, a system called SPINN (Morello et al., 2014) was developed that can detect 100 percent of the known pulsars in the High Time Resolution Universe survey (Keith, 2013). This system uses a custom neural network software implementation that allows finer control of the learning process than that used by Eatough et al. (2010). To improve learning and generalization the system employed recommendations from earlier work on efficient backpropagation algorithms by LeCun, Bottou,

Orr, and Muller (1998). Specifically, the following recommendations were employed:

- Feature scaling to to ensure each feature has zero mean and unit standard deviation over the traning set.
- The hyperbolic tangent function as an activation function
- Training of the system in batches containing different mixes of training data.

In addition to these techniques, the class imbalance was mitigated by oversampling the minority class pulsars so that the ratio of pulsars to non-pulsars was 4:1. *This may potentially be a problem! It's not clear that in the 5-fold cross validation that they took care to not have the oversampled pulsar candidates in the test and training sets...*

The feature set used in this research was smaller than that used by Eatough et al. (2010) and Bates et al. (2012), lending credence to the speculation that better feature selection could have improved performance of the efforts. The specific features used here are:

- Signal to Noise of the folded pulse profile
- Intrinsic equivalent duty cycle of the pulse profile
- Ratio between the period and dispersion measure
- Validity of the optimized dispersion measure
- Time domain persistence of the signal
- Root-mean-square distance between the folded profila and the sub-integrations that make up the profile

It should be noted that one of the criteria mentioned in the description of Figure 1.6 as being critical to human classification is not used in this feature set. The authors state that using continuity of the *Phase vs Frequency* plots hurt the performance. However, In other surveys with strong narrowband interference, this may be a very good feature to use to identify RFI.

Studies on optimizing the feature sets used in classification problems have been done (Van Hulse et al., 2009), and algorithms have been developed to automate the feature optimization. Weston et al. (2000) outlines a procedure for choosing features to be used in support vector machine classifiers. Since an ANN is a special case of a support vector machine (Vapnik, 1999), these techniques may be applied to the problem. Many of the papers cited in the section above on transient science machine learning algorithms spend a great deal of time on feature selection.

Lyon et al. (2016) have released work on optimal feature selection for the pulsar search problem. This work condenses the data in the pulsar data files into eight features:

- Mean of the integrated profile P.
- Standard deviation of the integrated profile P.
- Excess kurtosis of the integrated profile P.
- Skewness of the integrated profile P.
- Mean of the DM-SNR curve D.
- Standard deviation of the DM-SNR curve D.
- Excess kurtosis of the DM-SNR curve D.

- Skewness of the DM-SNR curve D.

The work of all of these authors(Lyon et al. (2016), Eatough et al. (2010) and Bates et al. (2012)) used simple statistics derived from the pulsar candidate plots as inputs to the ANN. These statistics may not include critical information that was lost when the plots were reduced to simple statistics. In addition, millisecond pulsars, a class of pulsars especially prized for their stable spin periods and emissions, are underrepresented in the training data they used in training their ANN. Both of these problems need to be studied further as part of this work.

Eatough comments that the CPU time to run the ANN is a very small fraction of the time spent creating the candidate plot in the first place (2 minutes vs. 3 hours). From that perspective, it is feasible to cascade several classifiers together to extract more pulsars from the stream and exclude more false positives. Again, for this application it is important to minimize false negative results.

Zhu et al. (2014) have worked on the pulsar classification problem in a different way, by training a system to directly read the pulsar diagnostic plots. This system combines many techniques together to optimize the classification problem. The subplots identified in Figure 1.4 are fed to classifiers that classify the subplot as containing a pulsar or non-pulsar, and then in turn the output of these classifiers are fed into a second classifier that determines, based on the scores from the first classifier, if the sample is a pulsar or non-pulsar. This system has been tested on Green Bank Telescope data, and has given good results, with 100% of the pulsars scoring in the top 1% of the candidates. This yields about a 100 fold reduction in the number of candidates to be examined.

In addition to the ANN research described above, work has been done on improving the algorithms for scoring the candidate plots. Using these improved techniques,

Keith et al. (2009) found 28 more pulsars in a data set that had already been mined. Their method included performing statistical analyses on the data making up the diagnostic plots. They analyzed the *subband, DM curve, and pulse profile* plots, and combined the output scores from these analyses to form a score to decide whether a candidate was a strong candidate.

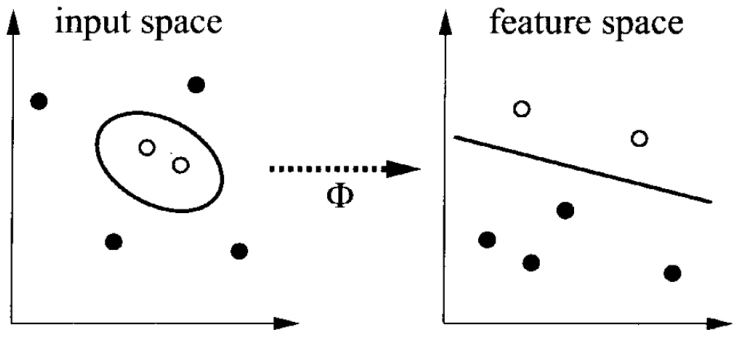
A recent system called PEACE: Pulsar Evaluation Algorithm for Candidate Extraction (Lee et al., 2013) demonstrated the utility of careful feature selection in an algorithm similar to the one described above. This paper used six *quality factors* in the scoring of the pulsar candidate. They achieved good results using these six factors, with 100% of the known pulsars in the data set ranked in the top 3.7% of the candidates. The features used by PEACE are:

- Signal to noise ratio of the folded pulse profile
- Topocentric period
- Width of the pulse profile
- Persistence of signal in the time domain
- Persistence of signal in the frequency domain
- Ratio between puslse witdth and Dispersion Measure smearing time

Support Vector Machines

Support vector machines (SVMs) are finding greater applications in data mining of large data sets, and in particular with pattern matching applications (Burges, 1998). Although support vector machines were developed starting in the 1970s (Han, Kamber, & Pei, 2011), they began to be studied in earnest in the 1990s (Cortes & Vapnik, 1995).

Figure 2.1: Nonlinear transformation Adapted from Scholkopf et al. (1997)



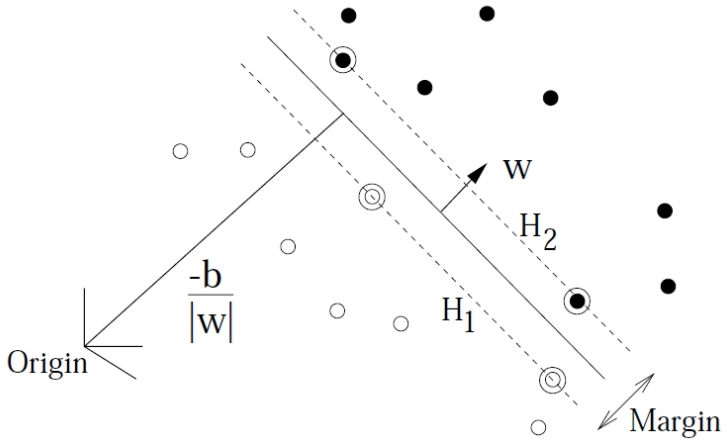
Support vector machines are machine learning algorithms that can extend linear modeling to nonlinear class boundaries. This is accomplished by transforming the inputs to the machine using a nonlinear function mapping (Witten, Frank, & Hall, 2011). For instance, a circular region could be transformed using a polar to rectangular mapping, thereby linearizing the problem, as shown in Figure 2.1 adapted from Scholkopf et al. (1997).

The basic idea of the support vector machine is to create a maximum-margin hyperplane between the classes, as shown in Figure 2.2. The maximum-margin hyperplane is that hyperplane that creates the greatest distance between the classes. The class instances that are closest to the maximum-margin hyperplane are called the *support vectors* (circled in Figure 2.2). The set of support vectors (tuples of data) define the hyperplane. None of the tuples further from the hyperplane matter to the SVM. This is important in the case of large data sets with a small number of minority classes, as only those tuples along the maximum-margin hyperplane are needed to define the classification.

Vapnik (1999) describes some of the useful theoretical properties of the SVM:

- The optimization problem used to construct the SVM has a unique solution

Figure 2.2: Maximum Margin Hyperplane and Support Vectors Adapted from Burges (1998)



- The learning process for constructing an SVM is faster than for a neural network
- While learning the decision rules, the support vectors are determined
- Changing the decision function is possible by changing only the kernel function used to define the feature space

The SVM is inherently a binary classifier. In the case of multiple possible classifications, an SVM for each possible decision must be created. An ensemble of SVMs and other classifiers may be profitably used to increase the classification accuracy of a given data set (Witten et al., 2011).

Statistical learning theory

The definition of the learning problem solved using the SVM comes from the statistical learning theory described by Vapnik (1999). The statisitical learning problem is defined in terms of minimizing the loss from misclassified observed data. The problem of pattern recognition (classification) is one of three problems described by Vapnik (1999). In the interest of brevity, the other two problems, regression estimation and density estimation, are not included in this treatment.

The model learning problem consists of three parts:

- A data generator that returns vectors taken from a fixed distribution $P(x)$
- A "supervisor" that returns an output vector for each of the input vectors.
- A learning machine capable of implementing a set of functions $f(x, \alpha), \alpha \in \Lambda$.

Choosing the best $f(x, \alpha), \alpha \in \Lambda$ that maps the input to the output is the learning problem. The training data is a set of l independent identically distributed observations taken from $P(x, y) = P(x)P(y|x)$:

$$(x_1, y_1), \ldots, (x_l, y_l) \tag{2.1}$$

Given these definitions, the learning problem is defined as minimizing the loss due to improper classification of the observations by the learned response of the machine. The expected value of the risk is given by the following equation, known as the risk functional:

$$R(\alpha) = \int L(y, f(x, \alpha)) dP(x, y) \tag{2.2}$$

The goal of the learning process is to find the function $f(x, \alpha)$ that minimizes equation 2.2. Since the problem set addressed in this paper is one of pattern recognition rather than regression or density estimation, the supervisor function simply outputs either a 0 or a 1, based on its evaluation of the indicator functions $f(x, \alpha), \alpha \in \Lambda$. If the loss function is

$$L(y, f(x, \alpha)) = \left\{ \begin{array}{ll} 0 & : y = f(x, \alpha) \\ 1 & : y \neq f(x, \alpha) \end{array} \right\} \tag{2.3}$$

then evaluating the the risk functional given by equation 2.2 returns the probability of classification error. The learning problem in the pattern recognition case consists of

minimizing the probability of classification errors, based on minimizing equation 2.2 with the given training data as inputs.

Feature Selection in SVMs

Selecting which features of a data set to use in the classification process is an important consideration. Selection of features is done to eliminate redundant or irrelevant features from inclusion in the SVM. Careful selection of a subset of features can improve the classification capability of the SVM while reducing computational load. (Weston et al., 2000)

Importance of feature selection Feature selection in SVMs is important to improve the performance of the SVM. The performance of the SVM can be considered in two different ways. First, the speed of running the algorithm on training data and input data is affected by the number of features used in the SVM. Second, the classification and generalization performance is negatively affected by the inclusion of irrelevant or redundant features.

Feature selection algorithms and methods The feature selection problem may be formulated in two different ways:

- Given a set of features of a cardinality n, and fixed subset of features of cardinality m, where $m << n$, find the m features that give the smallest expected errors.
- Given a maximum allowable expected error, find the smallest m that gives this expected error.

The expected error in either case is based on equation 2.3, with the input data modified to exclude some of the features by multiplying the input data by a vector

σ consisting of 0 where the feature is excluded, and 1 where the feature is included. Equation 2.4 shows the formulation of the problem. The task is to find σ and α that minimize the modified loss functional.

$$\tau(\rho, \alpha) = \int V(y, f((x * \sigma), \alpha))dP(x, y) \tag{2.4}$$

subject to the conditions $\|\sigma\| = m$, where $P(x, y)$ is fixed, but unknown, $x * \sigma$ is an element wise product of the input vector and the feature selection vector, $V(...)$ is the loss functional, and $\|\sigma\|_0$ is the 0 norm of the vector σ.

Two ways to attack this problem are known as the filter method and the wrapper method. The filter method is implemented by preprocessing the data to remove features before the SVM is trained, while the wrapper method provides the subset of features by repeatedly training the SVM with different feature sets and estimating the accuracy. This is a more computationally expensive procedure, but can give better results, since the power of the SVM is used to help sort out the feature set. This is still an active area of research, and much has been written on the subject of feature selection. Three promising approaches are given next.

Feature Selection in SVMs: R^2W^2 method Weston et al. (2000) introduce a method that combines the filter method and the wrapper method in a way that eliminates the computational complexity of the wrapper method while preserving the superior results usually obtained. The R^2W^2 algorithm consists of defining the SVM problem in terms of the dual formulation expressed as the Lagrangian of the problem (as explained in chapter 7 of Hamel (2011)).

The paper defines the radius of the hypersphere, R containing the support vectors in feature space, and the margin, M, and from Theorem 1 in Weston et al. (2000),

the expectation of the error probability is given by

$$E\{P_{err}\} \leq \frac{1}{l} E\left\{\frac{R^2}{M^2}\right\} = \frac{1}{l} E\left\{R^2 W^2(\alpha^0)\right\} \tag{2.5}$$

where the expectation is taken over a training set of size l.

Minimizing equation 2.5 requires a search over a space as large as the number of features, which is a combinatorially difficult problem when a large number of features are included. To make this problem more tractable, Weston et al. (2000) suggest substituting real-valued vector $\sigma \in \mathbb{R}^n$ for the the binary-valued vector $\sigma \in \{0,1\}^n$. This allows the use of a gradient descent algorithm to find the optimum value using the derivatives of the criterion, which are given below from Weston et al. (2000):

$$\frac{\partial R^2 W^2(\sigma)}{\partial \sigma_k} = R^2(\sigma)\frac{\partial W^2(\alpha^0, \sigma)}{\partial \sigma_k} + W^2(\alpha^0, \sigma)\frac{\partial R^2(\sigma)}{\partial \sigma_k} \tag{2.6}$$

and

$$\frac{\partial R^2(\sigma)}{\partial \sigma_k} = \sum_i \beta_i^0 \frac{\partial K_\sigma(X_i, X_i)}{\partial \sigma_k} - \sum_{i,j} \beta_i^0 \beta_j^0 \frac{\partial K_\sigma(x_i, x_j)}{\partial \sigma_k} \tag{2.7}$$

and

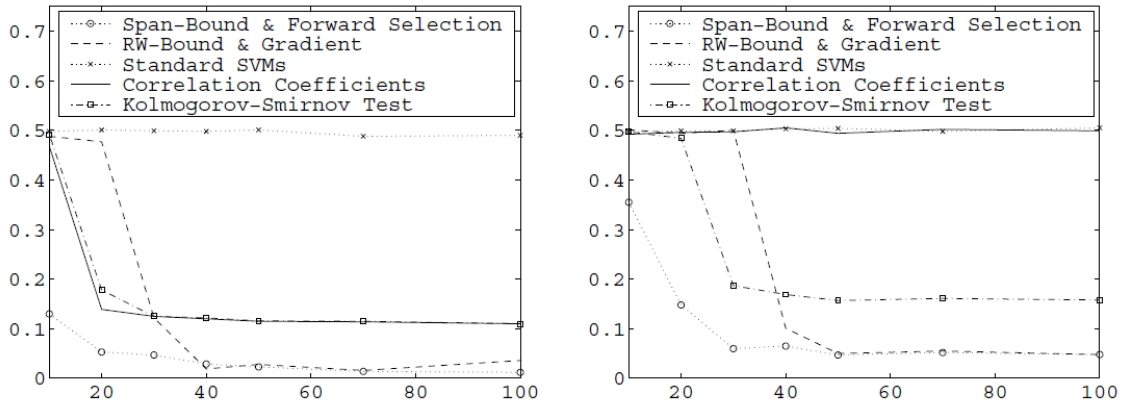
$$\frac{\partial W^2(\alpha^0, \sigma)}{\partial \sigma_k} = -\sum_{i,j} \alpha_i^0 \alpha_j^0 y_i y_j \frac{\partial K_\sigma(x_i, x_j)}{\partial \sigma_k} \tag{2.8}$$

In order to test the utility of this method, Weston et al. (2000) derived SVMs using this method, and also using three standard filter algorithms:

- Pearson Correlation Coefficients
- Fisher Criterion Score
- Kolmogorov-Smirnov test

The data for this test was two synthetically derived data sets consisting of a linear problem and a nonlinear problem. The linear problem had six out of 202 dimensions

Figure 2.3: Results for linear and nonlinear models Adapted from Weston et al. (2000)



that were relevant, and even these were partially redundant. In the nonlinear problem, only two out of 52 dimensions were relevant.

The standard SVM algorithms performed poorly on this extremely redundant and irrelevant data set, even with a large number of training points. The best standard methods yielded an error rate of 13%, compared to 3% for this new method. Figure 2.3 shows the compilation of results for this experiment.

The algorithm was also tested on real-world data sets including a face recognition application and a cancer-screening application. In the case of the face recognition application, it outperformed the other methods of feature reduction, but the reduced feature set classifier did not perform any better than the full-rank classifier. For the cancer screening application, the algorithm did much better than the other feature reduction methods, and outperformed the full-rank classifier. Using all 7129 genes, the linear SVM made 1 error out of 34 test examples. The reduced feature set classifier of 20 genes built with the R^2W^2 classifier made no errors, while 3 errors were made by a classifier built using the Fisher score. Classifiers using only 5 genes were also produced with the R^2W^2 method and the Fisher score method, and they made 1 and 5 errors, respectively.

While the toy problems were an extreme case of irrelevant data, the experiments showed that the introduction of irrelevant features into an SVM negatively affects the classification ability of the machine, and so features should be screened to be sure that they are contributing to the information needed to properly classify the instances.

Genetic Algorithms for Feature Selection As mentioned previously, there are two different formulations of the feature selection problem. Either the number of features can be specified and the generalization error estimated, or the allowable generalization error can be specified and the number of features needed can be estimated from the allowable error.

In the filter approach to the feature selection problem, a preprocessing step is used to separate the features to be used in the classification process from the features that are not used. This can be misleading to the classification algorithm, since the classification algorithm does not see all of the features, but only the subset that the preprocessing step did not remove.

In this study of feature selection, instead of using the standard k-fold validation routine, the genetic algorithm is used to estimate the error, based on including or excluding features in the calculation. This approach saves much of the computation that is required for a k-fold validation approach. This method can provide a lower variance, but a higher bias than cross-validation (Frohlich, Chapelle, & Scholkopf, 2003) The CHC genetic algorithm (Whitley, 1994) was used in this work. The defining feature of the CHC algorithm is the use of the population elitist strategy, where the best individuals of each generation replace the worst individuals of the parent generation. This allows faster mutations and hence a faster search strategy.

Several results were obtained from this study. The genetic algorithm shows a

significant overfitting problem. The study also found that the leave-one-out error bounds can be used as an alternative performance measure. It usually provides a better generalization performance, but leads to more features being selected. Using the same R^2W^2 bound described earlier, along with the genetic algorithm, shows a good generalization performance, comparable to the recursive feature elimination (RFE) method. The genetic algorithm is less computationally intensive, taking only about $\frac{1}{3}$ the run time of the RFE algorithm for equivalent classification performance. If the number of significant features is not known before beginning the analysis, the genetic algorithm method provides a more efficient method of determining the feature set to be used in the problem.

Optimal Feature Selection: Simultaneous training and feature selection Nguyen and De la Torre (2010) provide a view on the problem that is different from the preceding treatments. This view is that there is value in working on both training and feature selection in one process. This is because, as noted above, doing the procedure in 2 steps can cause a loss of information. Other researchers have proposed this approach before, but their methods led to non-convex optimization problems. The approach of Nguyen and De la Torre (2010) propose a convex framework for jointly learning optimal feature weights and SVM parameters. The method provides a set of weights that are sparse, and therefore are useful for selecting features, with the missing weights corresponding to features that are trimmed.

The error function is modified for jointly learning the kernel and SVM parameters. Parameterizing the kernel and SVM parameters provides the mechanism to handle this learning method. The input space to feature space mapping is provided by a parameter vector, p, where $\psi(x_i) = \psi(x_i, p)$. Different values of p provide different feature spaces, and these feature spaces are not directly comparable. Instead, nor-

malized margins, describing the margin of the respective feature spaces in a way that can be compared between the two implementations are employed.

Class Imbalance

Work by Seiffert et al. (2007) on very imbalanced data sets shows that even with the minority making up as little as 0.1% of the examples, effective classifiers can be built using techniques to mitigate the effects of the unbalanced data set. Several sampling techniques are explored that have been used in the past to try to mitigate the effects of the unbalanced class membership. The two most common techniques are random minority oversampling, (ROS) and random majority undersampling (RUS). In the former, the minority classes are duplicated randomly in the training data, while in the latter, some majority samples are randomly excluded from the training set.

Rather than randomly selecting majority members, techniques have been developed to more systematically choose majority members to exclude, such as one-sided selection (Kubat and Matwin, 1997), where the majority samples to be discarded are determined to be redundant, or noisy in some way. Wilson's Editing (Wilson, 1972) uses a kNN classification technique to evaluate which majority members get misclassified when classified against the remaining examples in the training set. The work of Seiffert et al. (2007) discussed above used 11 different learning algorithms and built over 200,000 classifiers. Their conclusions show that various methods of data sampling, a technique for selecting a subset of data for learning purposes, can increase the performance of the classifiers.

On the other side of the equation, another method of incrreasing the proportion of minority members is to manufacture synthetic samples by perturbing some of the features of the real members in a systematic way. This technique is known as

Synthetic Minority Oversampling Technique (SMOTE) (Chawla, Bowyer, Hall, and Kegelmeyer, 2002). The authors acknowledge that in classification problems, it is common for the number of examples of the normal case (the "uninteresting" case) to predominate by a large margin over the unusual or more interesting case, such as in our case of the pulsar vs non-pulsar. The paper shows that the combination of oversampling the minority class and undersampling the majority class can improve classifier performance of several different classification algorithms.

The methods of generating synthetic minority class members may work for the pulsar search problem since the physics that generate many of the signal properties are known, and new members of the pulsar class may be created by perturbing or modifying the existing class members within the physics from currently accepted pulsar models.

Ensemble Classifiers

Ensemble classifiers (Witten et al., 2011) have properties which may be surprising on first glance. Weak classifiers can be combined to produce strong classification results, in some cases stronger than atraining a model to a high degree of specialization. This is due to the face that an ensemble of weak classifiers has more resilience than the highly trained single model. It has been suggested (Witten et al., 2011) that in many cases, experts are really quite ignorant! The process is similar to the appointment of a diverse committee of humans to help make a decision. Many times the committee members will bring a different perspective to the table and help the overall competence of the committee, even if they are not an expert in the field being discussed.

Three types of ensemble classifiers are common. *Bagging* is a technique where the output of several models is used in a non-weighted voting scheme to determine the

output of the group of models. *Boosting* (Schapire, 1990) is a similar technique, but the outputs of the various base classifiers are weighted in some way related to the strength of their classification of the sample. It is similar to the way a human will give more weight to a person's opinion who is more learned in a subject.

A third way of combining multiple models is *stacking*, introduced by D. H. Wolpert (1992). Stacking is a way of combining models in a more intelligent way, using a meta-learner to combine the results of other learners, instead of using a simple voting mechanism, either weighted (boosting) or not weighted (bagging).

It is possible to combine several of the algorithms described in the previous sections to accomplish the classification task. An example of combining techniques is given by Seiffert et al. (2009). In this work, different types of resampling are combined with boosting to create classifiers that outperform more complex classifiers.

Chapter 3

Methodology

Methodology

The study concentrated on improving the classification performance of the work by Eatough et al. (2010) and Morello et al. (2014) by adding support vector machines and in experimenting with ensemble classifiers. Work in feature selection and in mitigating the imbalanced training data available for this problem was also shown to be very important. In particular, improving the false positive responses of the systems was of great interest to the pulsar community. The data set developed by Morello et al. (2014) as part of the SPINN work is from the Parkes Multibeam Pulsar Survey and is a superset of the data used by Eatough et al. (2010). This same data was used in the work by Lyon et al. (2016), to create their optimal feature set, and all of the data has been made public in the form used for the research by Lyon et al. (2016). This research continues use of this data set. The following general process was used as a guide for this research:

- Study the characteristics of normal and millisecond pulsars
- Develop a validation approach
- Study the information available for each pointing in the data
- Reproduce the results of the study by Lyon et al. (2016)
- Develop Support Vector Machine classifiers operating on the HTRU-1 data
- Experiment with ensemble classifiers and cascade or stacked classifiers

- Design, prototype, train, test, and evaluate a system that can handle the HTRU-1 data set and improve upon the performance using the techniques described above

The remainder of this chapter details the methodology used to accomplish the above processes.

Study the characteristics of normal pulsars and millisecond pulsars

As a good hunter knows his quarry, learning the characteristics of different types of pulsars helps decide which algorithms are appropriate. It was theorized that different types of pulsars might require different algorithms for best classification results. Results from previous research (Bates et al., 2012) show that millisecond pulsars have some characteristics, such as pulse width, different from common (normal) pulsars.

Develop a validation approach

The data sets available for research in pulsar search are necessarily sparse in the fraction of positive class examples. Given the imbalance in the data, it is difficult to develop a comprehensive validation and test data set. One way that this problem can be mitigated is to use multi-fold cross validation. This allows the training data to be used for testing during training by randomly selecting portions of the training data to use for training models, and holding out the rest of the training data for testing that particular model, The process is repeated using a different random selection of training data to train with, and different test data. Once the suite of models trained by the cross-validation technique were trained, they were tested with the pristine data that was held out from the training process, which in this case was 25% of the total HTRU-1 data set.

The requirements for pulsar classification were laid out in chapter 1. The most important criteria is that the pulsars should not be missed when candidates are run

through the classifier. But this must be balanced by the number of false positives generated. Obviously, a system that simply classified all samples as pulsars would meet the first criterion, but would be of no benefit. A secondary criterion is needed.

The primary measure of the efficacy of the models generated in this reseach is *recall*. The recall of the system is a measure of the number of positive samples that are lost by the system. It is defined as:

$$recall = \frac{TP}{TP + FN} \tag{3.1}$$

In the field of diagnostic tests, recall is also called *sensitivity*. This measure was the primary selector for the training process. Other statistics were also calculated and used in the evaluation process. The *specificity*, which measures the proportion of negative examples correctly classified, is a marker for the false positive rate that was the secondary criterion. A low value of specificity will mean an excess number of false positives will be returned, diminishing the utility of the classifier. Specificity is defined as:

$$S = \frac{TN}{TN + FP} \tag{3.2}$$

The FPR is the secondary measure of the effectiveness of the classifier. It is defined as

$$FPR = \frac{FP}{FP + TN} \tag{3.3}$$

The FPR can also be calculated from Specificity and Prevalence, where Prevalence is defined as:

$$P = \frac{(TP + FN)}{TP + TN + FP + FN)} \tag{3.4}$$

So FPR may be calculated as

$$FPR = (1 - S) * (1 - P) \tag{3.5}$$

Since for the extremely imbalanced data studied in this research $P << 1$, FPR may be approximated as $FPR \approx (1 - S)$.

Study the information available for each pointing in the HTRU-1 data set

The data used in this study was extracted from the SPINN data using the PulsarFeatureLab (Lyon et al., 2016) software. The data extracted for each pointing consists of 4 simple statistics derived from the folded pulse profile, and 4 statistics from the DM-SNR curve. These statistics were extracted from the Pulsar Hunter Candidate XML files distributed by the SPINN project, combined with the pulsar, non-pulsar label, and written to a comma separated text file for further processing.

Statistics of the folded pulse profile

The arithmetic mean of the folded and integrated pulse profile was calculated over the sampled pulse profile, and is given by equation 3.6

$$Prof_{\mu} = \frac{1}{n}\sum_{i=1}^{n} p_i. \tag{3.6}$$

The standard deviation of the profile forms another statistic used in the predictions. It was calculated from the samples of the pulse profile as shown in equation 3.7

$$Prof_{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(p_i - Prof_{\mu})^2}{n-1}} \tag{3.7}$$

The kurtosis is another measure of the central tendency of the distribution. It essentially measures the number of samples in the tails of the distribution. The kurtosis, k for a normal distribution is 3, and the excess kurtosis of a distribution is defined as $k - 3$. The sample kurtosis was calculated by equation 3.8.

$$Prof_k = \frac{\frac{1}{n}\sum_{i=1}^{n}(p_i - Prof_{\mu})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(p_i - Prof_{\mu})^2\right)^2} - 3 \tag{3.8}$$

The skewness of the pulse profile measures the symmetry of the profile about some mean or mode of the data. For this work, the sample skewness of the profile was calculated by equation 3.9.

$$Prof_s = \frac{\frac{1}{n}\sum_{i=1}^{n}(p_i - Prof_\mu)^3}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(p_i - Prof_\mu)^2}^3} \tag{3.9}$$

Statistics based on the DM-SNR curve, D

The same four statistics were calculated from the DM-SNR curve. The equations for these statistics are given here:

$$DM_\mu = \frac{1}{n}\sum_{i=1}^{n} d_i. \tag{3.10}$$

$$DM_\sigma = \sqrt{\frac{\sum_{i=1}^{n}(d_i - DM_\mu)^2}{n-1}} \tag{3.11}$$

$$DM_k = \frac{\frac{1}{n}\sum_{i=1}^{n}(d_i - DM_\mu)^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(d_i - DM_\mu)^2\right)^2} - 3 \tag{3.12}$$

$$DM_s = \frac{\frac{1}{n}\sum_{i=1}^{n}(d_i - DM_\mu)^3}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(d_i - DM_\mu)^2}^3} \tag{3.13}$$

To form these statistics, the PulsarFeatureLab Python script was run against the known non-pulsar data, and the output dumped to a CSV text file. A label was then appended to each line of the file (in this case, a 0, for "non-pulsar".) Once the nonpulsar data was processed, the script was run against the known pulsar data, and output to another file. Again, the label (a 1, for "pulsar") was added to the end of the line of the CSV file, denoting that these examples are pulsars. The 2 files were then merged together to form the data set with labeled examples of pulsars and non-pulsars.

Reproduce the results of the study by Lyon et al. (2016)

The SPINN study provided the data used by Lyon et al. (2016) in developing the features important to the classifier. The first step in this new work was to reproduce the classification results of that study.

The first step in accomplishing this task was to select a computational platform. Several options were considerd, including Python, R, and C++ based systems. Python and R have the advantage of being script languages, which are ideal for exploratory development and testing. Most of the machine learning algorithms are available for either platform. Most of these are implemented as native code linked into the script engine, and are therefore fairly fast and efficient. In the end, the R platform was chosen in part due to the very good support for managing the training and evaluation of the models that was provided by the Classification and Regression Training (caret) package (from Jed Wing et al., 2016). This package provides a common consistent interface to a large number of (more than 230) machine learning models. In addition to the caret package, the R system provides simple tools that allow the training to proceed in parallel if the machine has multiple cores. The doParallel package (Revolution Analytics & Weston, 2015) was used to allow the training to use all four of the cores available on the machines used to run the models. The R system is cross-platform, and Linux and Mac OS versions (Version 3.3.2) were used in the research with equal utility. A complete listing of the packages used for the research is included in Appendix A.

In order to assure a common operating environment for all of the work on different models using the same data set, a script was written to read in the data from disk and processes it for use by the machine learning algorithms. The script also preloads some universally used utility libraries.

The script consists of the following steps "CommonNoPreproc.R" A to prepare the data and environments uniformly for experiments:

- The common libraries needed for the experiments were loaded: doParallel for parallel processing, and the caret library for managing the training and testing of the models.

- The data file used in the experiments was defined in this script allowing repeated experiments to easily be run on the same data set. A new data set can be simply defined and used for another series of experiments without changing other parameters or code by editing this file.

- Feature names were defined corresponding to the columns in the data set.

```
featureNames  <-  c("Prof-mu", "Prof-sigma","Prof-kurtosis",
                    "Prof-skew", "DM-mu", "DM-sigma",
                    "DM-kurtosis","DM-skew","Class")
```

- The seed for the random number generator used for assembling subsamples of data was defined here, so that each run of an experiment can be reproduced by setting the random seed to this globally defined constant before using the random number generator.

- Data was read in and divided into the data proper, and the label. This allowed the labels to be manipulated and put into a form that the learning algorithms will handle. The numeric label value was changed into a factor of 2 levels, "pulsar", and "nonpulsar". After these manipulations, the label was put back into the data structure, readying the data for experiments.

- The data was divided into a training set, and a test set. The training set consists of 75% of the data, and is formed from the full data set by sampling from the

classes in the same proportion as the original data set. That is, 75% of the non-pulsar data and 75% of the pulsars are selected for the training set. 25% of each class is then reserved for the testing set.

- The parallel processing system was set up to use all available cores on the machine. This allows most algorithms to be trained in parallel, saving clock time.

- The random number generator is seeded with the previously defined value.

Running this script set up the environment, reads the data into a data frame that was also used for the rest of the experiments. The experiments by Lyon et al. (2016) on this data set used a neural network to classify the data. Several neural network algorithms were studied for this research, including the RSNNS package implementing the Stuttgart Neural Network Simulator (Bergmeir & Benítez, 2012), and the "nnet" R package (Venables & Ripley, 2002). After some experimenting, the "nnet" package was selected for use with the "caret" package to generate the models trained and evaluated in this research. The preliminary experiments were all accomplished using the command shell of the 'R' system. As the scripts for running the full experiment were being written, pieces of the script were pasted into the shell and executed, validating the code as it was written. Once all of the preliminary experiments were completed and the final configurations and processing steps were completed, the finished script, consisting of the steps described below, was used to execute the experiment and collect data and plots. The script is given in Appendix A.

The neural network was trained using the caret package's "train" function. To use this function, the user sets up the conditions for training, and then runs the "train" method to construct a family of models that are evaluated according the the metric

the user wants to use to evaluate the models. The metric used in this study was "Sensitivity", since we are most interested in a low false-negative characteristic for the predictions, and it is equivalent to recall, our primary measurement. A grid of training parameters is defined internally to the train function using the caret package's "expand.grid()" function. This function allows the user to specify the parameters that can be varied by the model training software and guides the training of the model, but in this case the system was allowed to generate its own grid, and the input to the training was the number of parameter values for the system to try for each parameter. The variable "tuneLength" is set to 6 in the script, directing the system to try 6 different values for each parameter. The function expands this specification into a grid containing (in this case) 36 cells, each of which holds a unique combination of the parameters. This causes the system to train 36 different neural networks for each iteration.

The "trainControl" command sets up the conditions for training. The following trainControl parameter settings were used for training the neural network.

```
baseTrainCtrl <- trainControl(method = "repeatedcv", number = numFolds,
                     repeats = numRepeats,
                     preProcOptions=myPreProcOptions,
     classProbs = TRUE, seeds = seeds,
                     verboseIter = TRUE,
                     summaryFunction = twoClassSummary)
```

The method argument controls how the training will proceed. Repeated cross-validation was specified for the reasons described above. By default, the cross-validation consists of 10-fold cross validation. Arguments are available to change the number of folds if desired, however 10-fold cross validation was used. The cross-validation is repeated five times, providing 50 models in all that are trained. The preProcOptions argument tells the system to preprocess the training and test data

to center the data around zero, and scale it to one standard deviation. The summaryFunction is a built-in function that calculates the ROC, the Sensitivity, and the Specificity of the model. These metrics are used by the algorthm to select the best model out of all the models generated in the repeated cross-validation scheme. Once the trainControl object was created, it was passed to the train function, which returns the optimized model, which in this case is the model based on the original unmodified training data:

```
orig <- caret::train(Class ~ ., data = dataset$original,
                             method = myMethod,
                             preProcess = c("scale","center"),
                             verbose= verbosity,
                             metric = myMetric,
                             tuneLength = myTuneLength,
                             trControl = ctrl)
```

In order to explore the effects of changing the sampling strategy, additional training data sets were created by using functions provided by the caret package (up and down sampling), the "ROSE" package (Lunardon, Menardi, & Torelli, 2014) and the "DMwR" package (Torgo, 2010). The same input test data set was retained and used for all tests of all models and sampling strategies. The downsampled data was created using:

```
set.seed(mySeed)
dsPulsarTrain<- downSample(x = pulsarTrain[, -ncol(pulsarTrain)],
                           y = pulsarTrain$Class)
```

The random number generator seed was reset before each random sampling, so that repeatable results can be constructed. The downsample function removes majority class members to create a balanced training data set. In this case, it created a data set with 897 pulsars and 897 non-pulsars.

Upsampled data was created using:

```
set.seed(mySeed)
usPulsarTrain <- upSample(x = pulsarTrain[, -ncol(pulsarTrain)],
                          y = pulsarTrain$Class)
```

returning a training set with the pulsars upsampled to the level of the non-pulsars, or 67497 pulsars and non-pulsars.

Random Over Sampling Examples (ROSE) data was created by the ROSE function, giving approximately 34000 samples of each class:

```
set.seed(mySeed)
rosePulsarTrain <- ROSE(Class ~ ., data  = pulsarTrain)$data
```

Finally, the SMOTE data was created using the SMOTE function to create synthetic minority samples, based on the nearest neighbors to the minority class members.

```
set.seed(mySeed)
smotePulsarTrain <- SMOTE(Class ~ ., data  = pulsarTrain)
```

In this case, SMOTE returned 2691 pulsars and 3588 non-pulsars.

With each of these four new distinct training data sets in hand, the models were trained as described above, holding all other variables constant.

The five models, including the original model built with non-resampled data, were applied to the test data, and evaluated on their sensitivity using the following function:

```
test_sensitivity<- function(model,data) {
    ct = confusionMatrix(predict(model,data),data$Class,
                         positive = 'pulsar')
}
```

This function calls the predict() function on each model with the test data and uses the results to generate a confusion matrix. It was applied to all of the models and the test data using the "lapply" function, and the results of the confusionMatrix() call inside the function are returned and saved for later analysis. Results for this process are given in chapter 4 in the neural network section.

Develop support vector machine classifiers working on HTRU-1 data

Support vector machines are suited to the task of classifying pulsar data, since they work well with limited and unbalanced training data. There are several implementations of SVMs available in the caret package, including a wrapper for the popular high-performance libSVM package included in the e1071 (Meyer, Dimitriadou, Hornik, Weingessel, & Leisch, 2015) package, which is what was used in this work. Additional SVM packages, such as that in the "WEKA" package (Witten et al., 2011), and the "kernlab" (Karatzoglou, Smola, Hornik, & Zeileis, 2004) package are also included in the caret suite and were used in initial preliminary experiments. The final script used to train and evaluate the models is given in appendix A, which is the same script and procedure used to train the neural network as described in section 3 above. The grid of training control variables was modified by the train function to be compatible with the SVM model by defining a grid consisting of *cost* and *weight* parameters, each with 6 distinct values. The trained models were evaluated with the same test data as used for the neural network experiments. The results of this experiment are presented in chapter 4 in the SVM section.

C5.0 classifier

A C5.0 classifier was also trained and tested along with the neural network and SVM classifiers. This classifier uses an improved version of the well-known C4.5 algorithm (Quinlan, 1986; Witten et al., 2011). This algorithm is a top-down decision tree learner which uses information gain to build the tree. The "C50" R package (Kuhn, Weston, Coulter, & code for C5.0 by R. Quinlan, 2015) provides the C5.0 algorithm and a caret training interface. The training of the C5.0 algorithm was done with the same parameters as for the neural network and SVM. Results of this experiment are presented in chapter 4 in the C5.0 classifier section.

Bootstrap Aggregation Ensemble classifiers

Training several classifiers with different randomly selected sub-sampled sets of the training data can yield classifiers that, together, perform better than a single classifier. It is interesting to note that an ensemble classifier almost never performs worse than any of its constituent classifiers (Witten et al., 2011). In this research, an ensemble classifier was generated using the bootstrap aggregation technique, using a decision tree model as the base. This is accomplished using the "treebag" caret method which is an implementation of bagged decision trees based on the "ipred" (Peters & Hothorn, 2015) library. In the preliminary experiments undertaken during this work, an aggregation of 50 trees was specified in the training function, giving a large number of trees working in the ensemble. More trees, up to a point of diminishing returns, reduce the variance of the model outputs. After analysis of the preliminary results, the final runs of the training script used an aggregation of only 25 trees, which provided equivalent performance at a 50% reduction in computing.

Other Ensembles and Stacked classifiers

In addition to the bootstrap aggregation decision tree classifier, ensembles consisting first of a simple arithmetic combination of the base models, and also a more complex ensemble of the four models (neural net, SVM, C5.0, and bagged decision trees) was built that combines the output of each of the models by stacking a model on top of the base models. Development of a stacked classifier involves first training and evaluating base classifiers using different models, and combining the resulting models with another model that takes the outputs of the first tier of models and uses them as additional input for learning the final decision function. Using the facilities of caret again, results from the models generated in the previous experiments were evaluated, and another model set was trained and then stacked with an additional

model and integrated into a final predictor system.

Base Models

The base models to be stacked were defined by by creating a list of models to use for the base of the stack. A neural net, a support vector machine with a linear kernel, a set of bagged tree classifiers, and a C5.0 model comprised the set of models. The models were trained using the five data sets, and evaluated using the same techniques as previously reported.

Simple Ensemble Classifer

Recalling that the primary metric for this work is to be sensitivity, or recall, it makes sense to think about a way to combine the base classifiers together to make a classifier that maximizes the sensitivity. All of the base models did well when trained on modified training data, and had both good sensitivity and low false positive rates. This allowed implementation of a simple method of combining the results such that if any of the base models predicted a sample was a pulsar, then that sample was assigned the pulsar label(the logical "or" function.) Code was generated that provided this function as shown below:

```
data <- ensembleTestData[[sampling]]
theSum = as.numeric(data$V10) + as.numeric(data$V11) +
         as.numeric(data$V12) + as.numeric(data$V13)
numResults <- theSum < 8
```

The data variable is a matrix of the original test data with columns added to it corresponding to results from the base models. The vector numResults contains logical (True or False) values corresponding to the output of the simple ensemble classifier's (theSum ¡ 8) comparison. The vector of results is fed to the confusionMatrix() function and the results are given in chapter 4.

Meta-learning Models

The meta-learner models or top models for the stack were defined as a neural net model and a C5.0 model. The models were trained using the data from the original training set, modified with the addition of columns of output values from the base models. The following describes the procedure used to form the stacked classifier.

The training procedure was repeated for each of the top models defined in the list of possible top models. A trainControl object was defined to manage the training of the top models:

```
topCtrl <- trainControl(method = "repeatedcv", number = numFolds,
                        repeats = numRepeats,
                        preProcOptions=c("scale","center"),
                        classProbs = TRUE, seeds = seeds,
                        verboseIter = TRUE,
                        summaryFunction = twoClassSummary)
```

This trainControl object is passed to the caret::train() function when training the top models. Next, the list of predictors in the data set is created. This list contains all of the columns of the data set, except for the Class label field, which is defined by 'outcomeName'.

```
predictors <- names(training)[!names(training) %in% outcomeName]
```

Each of the base models were evaluated and the predictions they produced were saved as new columns in the training and testing data:

```
testing[,(ncol(testing)+1)] <-
            caret::predict.train(object=model, testing[,predictors])
training[,(ncol(training)+1)] <-
            caret::predict.train(object=model, training[,predictors])
```

The new columns represent additional knowledge gained through the use of the base models. This new information was then fed to the top model training procedure by

extending the list of predictors to include the new columns, but again not the column of outcome values:

```
predictors <- names(training)[!names(training) %in% outcomeName]
```

The final model for each top model option is then trained with the following:

```
modelFinal <- caret::train(training[,predictors],
                           training[,outcomeName],
                           method=ensembleModel,
                           preProcess = c("scale","center"),
                           metric=myMetric,
                           tuneLength = myTuneLength,
                           verbose=verbosity,
                           trControl= topCtrl)
```

As the final models are of the same form as the base models, the procedure developed to evaluate the base models can be used to test and evaluate the stacked classifier models. The results of the testing and evaluation are presented in chapter 4.

Develop a working on-line classifier system

The results from the work described in the previous sections is a set of models that can be used to predict the class of any single data sample, as well as being able to produce predictions in a batch mode as was done in these experiments. As shown in Figure 1.3, in normal use the pulsar processing pipeline writes candidate files to the file system on the computer when it finds a possible pulsar in the data. Using the PulsarFeatureLab Python script (Lyon et al., 2016), a new candidate file is processed to extract the features needed for the classifier. The features are written to an output file, which is then read by the R system. The new candidate data is preprocessed using the same algorithms as used in the training process, and then it is run through the classification models generated previously. Applying the classification models takes less than a second, which is orders of magnitude faster than the generation of candidates in the first place, so this process can be integrated into

the processing pipeline as the candidates are generated without slowing down the processing. Another method, which may be simpler in practice, would be to run the classifier against the candidate files in batch mode once per day or once per hour, depending on the speed of candidate generation.

Figure 3.1: Classifier System

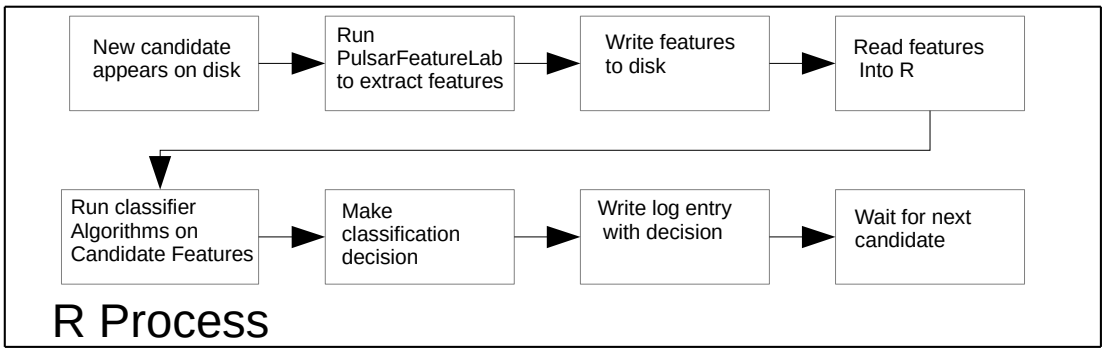


Figure 3.1 shows the block diagram of the proposed classifier. It connects to the end of the processing chain in figure 1.3 by watching for new candidate files to appear in the pipeline output directory. The on-line classifier would be written in R, with the PulsarFeatureLab Python script being called from inside the R process using the "system" call to invoke the Python script and wait for the output. The classifier system can remain loaded and running on the computer between candidates, waiting for new files to appear in its working directory. This would reduce the overhead of starting and stopping the rather heavyweight R application for each new candidate file that is produced.

Chapter 4

Results

Introduction

This dissertation developed and evaluated improved supervised learning methods for pulsar detection based on available search data. Experimenting with many machine learning algorithms and data processing techniques provided a great deal of insight into the methods by which machine learning can be applied to various problems. The number of possibilities for models, data handling and evaluation is staggering. The results of this research is a study of the efficacy of different machine learning algorithms and data processing methods for the problem of imbalanced noisy training data. Evaluations of several models including neural networks, support vector machines, bagged trees, a C5.0 model are included. A stacked ensemble using these as base models with a simple combiner, a neural network and C5.0 decision tree classifier as the meta-learners are presented. The remainder of this section outlines the common data sets produced for training the models and the model parameters used for training. Following that, results of training and evaluating each of the models are presented.

Data Sets

The pulsar data used for this research was extracted from the Parkes Multibeam Pulsar Survey archive and collated by Morello et al. (2014), and subsequently made available on the Web. Lyon et al. (2016) describes the formation of a set of eight features from the data using rigorous mathematical techniques to ensure that the features are useful in the detection of pulsars. In Figure 4.2, one of the features

Table 4.1: Pulsar data sets

Data Set	Pulsar	Nonpulsar
Original	897	67497
Downsampled	897	897
Upsampled	67497	67497
SMOTE sampled	2691	3588
ROSE sampled	34534	33860

was assigned a zero "importance" value, meaning that the model didn't use it in its training and classification process. Some of the models had a different view of the importance of the features. The importance values change with different resampling methods and models, indicating that the features are correlated in some way.

The data was preprocessed before the machine learning modeling effort by replacing the numerical values "1" and "0" denoting a pulsar or non-pulsar class, respectively, with a factor variable labeled "pulsar" and nonpulsar". This is necessary for the modeling infrastructure to properly interpret the class labels. Since pulsars are rare in the search data, and part of the purpose of this research was to look at the effect of imbalance on the training process, the data was then passed to sampling functions to create four more data sets from the raw data, each of which contained a more balanced set of data. Table 4.1 shows the distribution of pulsar and nonpulsar classes in the five data sets.

The down-sampled version was created by discarding majority (non-pulsar) samples to balance with the pulsar samples. This created a data set of only 1794 samples out of the original 68394, half pulsar and half nonpulsar. An up-sampled data set was created by reusing the minority (pulsar) samples to create a data set with 67,497 pulsar and nonpulsar samples for a total of 134,994 samples. A data set using the SMOTE technique (Chawla et al., 2002) was created using the default values for the

percent under/over sampling, and the number of nearest neighbors to use for the synthetic sample construction. The default value for the number of nearest neighbors is five, and for the percentage of oversampling and undersampling is 200, meaning that for each minority sample, two new samples are created from its nearest neighbors. With these parameters. two majority samples are selected for each new sample, yielding a data set with 6279 samples, of which 2691 samples were pulsars, and 3588 were non-pulsars. Finally, a data set using the ROSE algorithm (Lunardon et al., 2014) was created using the ROSE function with its default probability parameter of 0.5, meaning that a nearly balanced data set is produced with 34534 non-pulsars and 33860 pulsar examples. The five data sets were used to train each of the base models, which were tested with the data held out from the training process for testing the final models.

Common model parameters

The models, both the base models and the stacked ensembles) were all trained using a common *trainControl* object and training parameters. Table 4.2 gives the parameters used in training the base models. The trainControl object was created using the following code:

```
baseTrainCtrl <- trainControl(method = "repeatedcv", number = numFolds,
                     repeats = numRepeats,
                     preProcOptions=myPreProcOptions,
                     classProbs = TRUE, seeds = seeds,
                     verboseIter = FALSE,
                     summaryFunction = twoClassSummary)
```

The method argument controlled how the training was accomplished, namely repeated cross validation. 10-fold cross validation was used. The cross-validation is repeated five times, providing 50 models that are trained. The preProcOptions argument tells the system to preprocess the training and test data to center the data

Table 4.2: Model Training Parameters

trainControl()		
Parameter	Description	Value
number	Cross validation folds	10
repeats	Cross validation repeats	5
preProcOptions	Data preprocessing applied	Center and Scale
classProbs	Return the class probabilities	True
seeds	List of seeds for fold creation	Random
verboseIter	Verbose flag	True
summaryFunction	Evaluation function	twoClassSummary
train()		
Parameter	Description	Value
tuneLength	Number of different parameter values for each parameter	6
metric	Selection metric	"Sens"

around zero, and scale it to one standard deviation. The summaryFunction is a built-in function that calculates the area under the ROC curve, the Sensitivity, and the Specificity of each model. These metrics were used by the algorthm to select the best model out of all the models generated in the repeated cross-validation scheme. Once the trainControl object was created, it was passed to the train function, which returns the optimized model, which in this case is the model based on the original unmodified training data.

Several test runs of training the models were undertaken to find the best options for these parameters. It is imperative that many combinations of subsets of examples are chosen to minimize the variation in the output due merely to the random distribution of samples. The optimal choice is to use the leave-one-out procedure of taking all of the samples except one and modeling them, repeating for all samples. This is, however, computationally intensive and infeasible except for small data sets.

A compromise can be made by dividing the data into groups (folds) to train models using the different folds and validating the model produced by each fold with the data in the other folds. The number of folds (denoted by k) and repeats are chosen to minimize the variance in the final models. An empirical procedure was followed in this research to choose the number of folds and repeats at values that appeared to be past the point of diminishing returns. The experimental procedure followed to arrive at these values was to allow the seeds for randomly selecting the cross-validation folds and repeats to be random, and to examine the training results to determine the stability of the training process. Using 10 folds and five repeats was chosen.

The preprocessing options given to the models ensured that each of the features in the feature set was properly scaled and centered about zero, since the native values of these features varied by two orders of magnitude, from a minimum of about -7 to a maximum of 237.

The twoClassSummary function that is built in to caret was specified as it incorporates the sensistivity and specificity measurements which are essential to evaluate the models for this application.

Each of the stacked ensemble models are trained and evaluated with the same model parameters and the same data sets as the base models.

Organization and presentation of results

For each of the models studied, the results of training and evaluation of the models is presented. The results are organized by model type, with all of the model types trained and evaluated with a common data set. Statistics on the sensitivity, specificity, accuracy, area under the ROC curve, and the false positive rate for each model and data set are presented and summarized in tables. Data on the variable importance for each of the trials is plotted. Finally, the performance data for the

Table 4.3: Neural Network Models

Data Set	Hidden Neurons	Decay Rate
original	11	0.0178
downsample	1	0.0001
upsample	11	0.0
SMOTE	11	0.0178
ROSE	11	0.1

stacked ensembles of the base models is presented. Raw results and more plots are provided in the appendices.

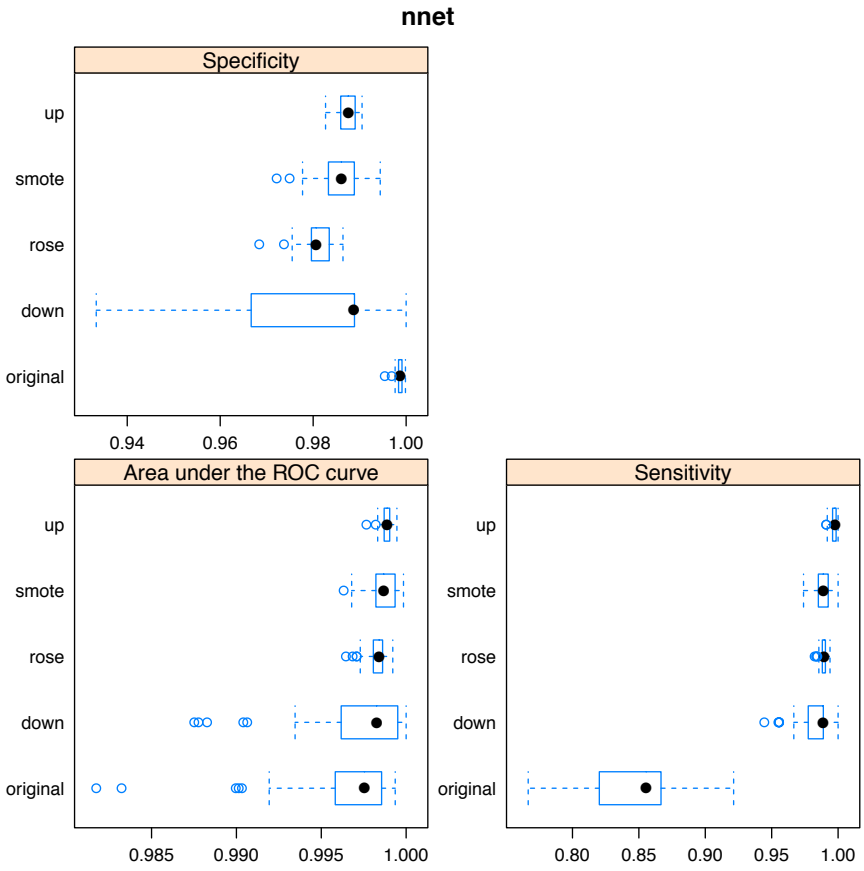
Neural Network Classifier Results

As described in chapter 3, the neural network was trained with five distinct training data sets, holding all other variables constant. Each of the models was tested with 25% of the data, that is, the test data set that was held out at the beginning. The models were evaluated based on their sensitivity to the training data set.

The network architecture consisted of eight input neurons, a single hidden layer of neurons, and one output neuron. In the various trials, a hidden layer with 1, 3, 5, 7, 9, and 11 neurons was trained. Each of these neurons was fully connected to the input and output layers. The weight decay tunable parameter, which controls how fast the weights change while the training takes place, helps to avoid overfitting. The number of hidden neurons and decay rate parameters for the final models chosen through training iterations for the neural network are given in Table 4.3.

Figure 4.1 shows a box and whiskers plot of the results of training the five models. The box and whiskers plot is a compact way of expressing the statistics of the training process. The center of the box is the median value of the data set. The ends of the box show the upper and lower quartile values (the median of the data above the median and below the median), and the whiskers show the upper and lower extrema. The

Figure 4.1: Box and Whiskers plot with all three metrics returned from training the neural network models



complete data set used to create the graph is given in the appendix for the related model. An abbreviated version is shown in Table 4.4, which omits the 1st and 3rd quartiles.

The box and whiskers plot graphically shows that the sensitivity, area under the ROC curve, and specificity (and by extension false positive rate) are all improved by resampling the data. Clearly, all of the resampling methods have improved sensitivity over the original model built with the highly imbalanced training data. Table 4.4 shows the statistics of the model generation results for all of the models across the available metrics returned by the twoClassSummary function. Note that these statis-

Table 4.4: Neural Network Model Training Summary Statistics

Data Set~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	0.996	0.004	0.982	0.999
original~Sens	50	0.848	0.035	0.767	0.921
original~Spec	50	0.999	0.001	0.995	1.000
down~ROC	50	0.997	0.003	0.988	1.000
down~Sens	50	0.983	0.015	0.944	1.000
down~Spec	50	0.980	0.017	0.933	1.000
up~ROC	50	0.999	0.0003	0.998	0.999
up~Sens	50	0.997	0.002	0.991	1.000
up~Spec	50	0.987	0.002	0.983	0.991
smote~ROC	50	0.999	0.001	0.996	1.000
smote~Sens	50	0.989	0.006	0.974	1.000
smote~Spec	50	0.986	0.005	0.972	0.994
rose~ROC	50	0.998	0.001	0.996	0.999
rose~Sens	50	0.989	0.002	0.982	0.994
rose~Spec	50	0.981	0.003	0.968	0.986

tics are generated by the training process, and are not the results of testing with pristine data. The numbers given below in the text are the results of testing the system with pristine data.

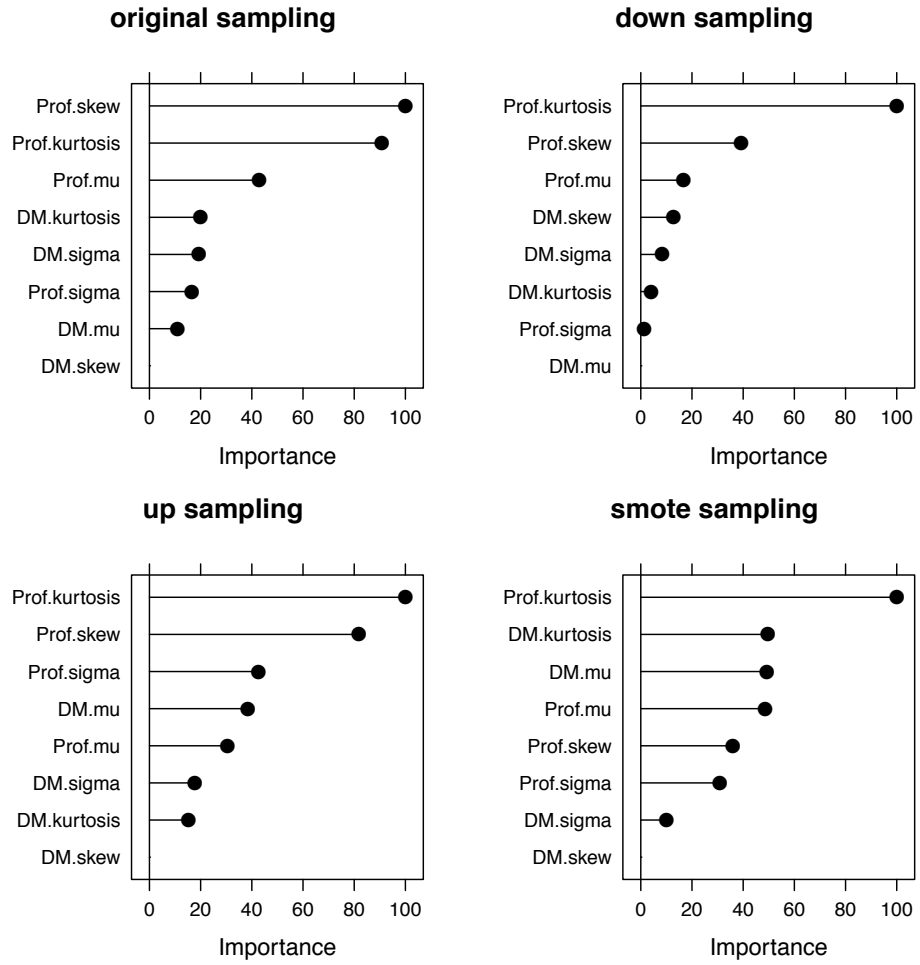
The best sensitivity on the test data achieved with final model fit with the original unresampled data set was 83.28%. The false positive rate, which is important for this application, was approximated from the specificity as $FPR \approx (1 - Spec)$, or $1 - 0.99871 = 0.129\%$. The accuracy of this model is 99.65%. While the FPR is excellent, the sensitivity doesn't meet the requirements of the users. The accuracy is also good, but the raw accuracy is not a good metric for an extremely unbalanced training and test data set such as this. A summary of the statistics are displayed in the box and whiskers plot, and the full statistics are given in appendix B.

Models built with modified data sets gave better results for the primary metric. Using the downsampled data set for training, the system acheived a sensitivity of

99.997%, and a FPR of 1.60%, both of which meet the requirements set by pulsar scientists. The accuracy figure for this network is slightly worse than that using the original training data, at 98.32%, due to the worse FPR. Using the upsampled data set, the best sensitivity was 97.993%, slightly worse than the downsampled data set, but still much better than the original unmodified data set results. The FPR was 1.44%, and the accuracy was 98.55% The SMOTE-modified data set provided a sensitivity of 98.328%, an FPR of 1.40%, and an accuracy of 98.60%. Using the ROSE technique to modify the data set gave mediocre results but still improved on the original data set. This technique gave a sensitivity of 89.30%, an FPR of 0.49%, and an accuracy of 99.37%. Considering sensitivity as the primary criterion, the downsampled data set gave the best performance in training the neural network for the pulsar/non-pulsar classification problem.

While training the networks, in some cases different variables were chosen by the network as the most important. Some of the variables ended up not having any predictive value for some of the networks. Figure 4.2 is a variable importance plot. This plot shows all of the features used in training the models, and the relative importance that the model places on each. Greater importance is shown by a longer line on the plot. In the ROSE data set, the "DM.kurtosis" feature had no importance in the model. In other models, the "DM.skew" ends up with little importance. This indicates that these two variables may be correlated across the samples, and one or the other ends up not being needed for predicting the outcome when both are presented to the model.

Figure 4.2: Neural network model variable importance plots for four different sampling methods



Support Vector Machine Classifiers

A support vector machine with a linear kernel was trained in the same way that the neural network was trained as described earlier in this chapter. The SVM consisted of a linear kernel, and the training parameters were cost and weight. Cost refers to the cost of a constraints violation during training. Higher values of cost will make the algorithm work harder to avoid constraint violations. The class weight parameter can be used to help mitigate against the effects of different class sizes. The parameters

Figure 4.3: Neural network model variable importance plots for the ROSE sampling method

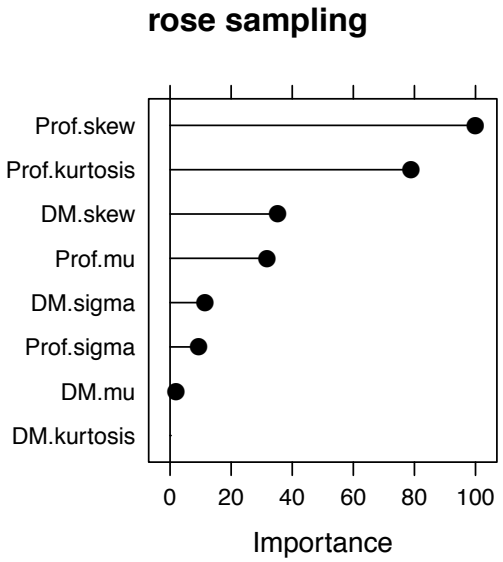


Table 4.5: Support Vector Machine Models

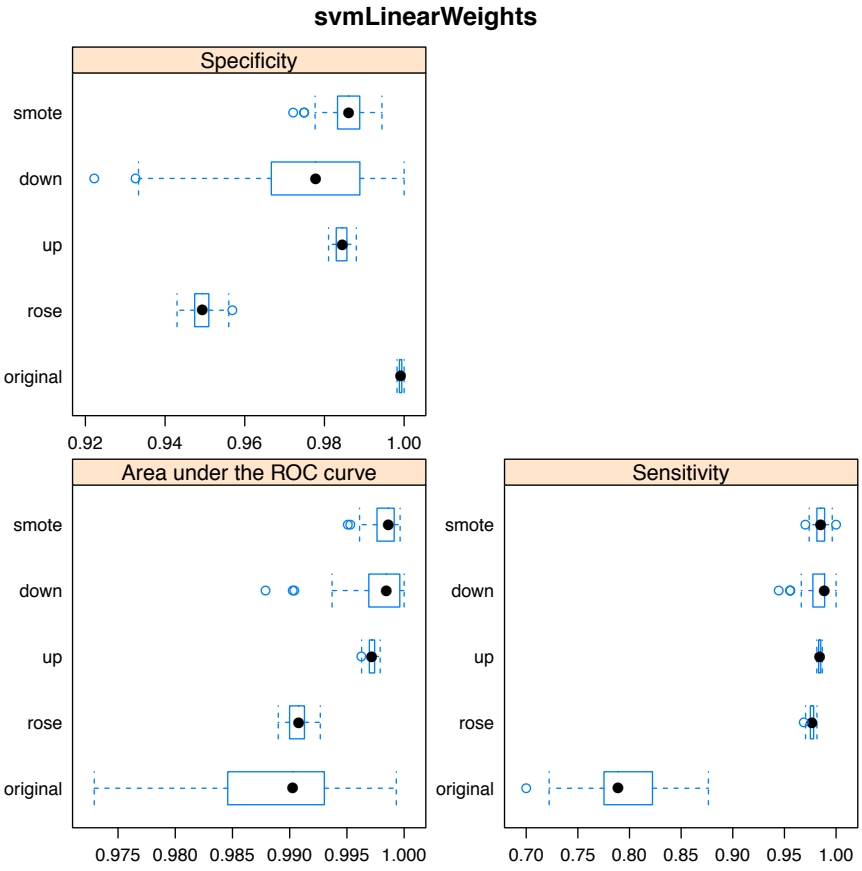
Data Set	Cost	Weight
original	1	1
downsample	8	1
upsample	1	1
SMOTE	1	1
ROSE	0.25	1

for these training parameters that produced the final models are given in Table 4.5.

Figure 4.4 shows a box and whiskers plot of the results of training the five SVM models with the training data. This plot shows that for the SVM the sensitivity, area under the ROC curve, and specificity are all improved by resampling the data. Table 4.6 shows the statistics of the model generation results.

Testing with the test data set, the sensitivity achieved with final model fit with the original unresampled data set was 76.590%. The false positive rate was 0.12%. The accuracy of this model is 99.57%. As in the case of the neural network models,

Figure 4.4: Box and Whiskers plot with all three metrics returned from training the SVM models



modified data sets gave better results for the primary metric. Using the downsampled data set for training, the system acheived a sensitivity of 98.328%, and a FPR of 1.57%. The accuracy figure for this machine is also slightly worse than that using the original training data, at 98.43%, due to the worse FPR. Using the upsampled data set, the best sensitivity was 98.662%, slightly better than the downsampled data set. The FPR was 1.57%, and the accuracy was 98.43% The SMOTE-modified data set provided a sensitivity of 97.993%, an FPR of 1.33%, and an accuracy of 98.66%. Using the ROSE technique to modify the data set gave the least improvement in sensitivity over the unmodified training data, yielding a sensitivity of 97.324%, an

Table 4.6: Support Vector Machine Training Summary Statistics

data Set~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	0.989	0.006	0.973	0.999
original~Sens	50	0.795	0.036	0.700	0.876
original~Spec	50	0.999	0.0004	0.998	1.000
down~ROC	50	0.998	0.003	0.988	1.000
down~Sens	50	0.983	0.014	0.944	1.000
down~Spec	50	0.978	0.020	0.922	1.000
up~ROC	50	0.997	0.0004	0.996	0.998
up~Sens	50	0.984	0.001	0.981	0.987
up~Spec	50	0.984	0.002	0.981	0.988
smote~ROC	50	0.998	0.001	0.995	1.000
smote~Sens	50	0.985	0.007	0.970	1.000
smote~Spec	50	0.985	0.005	0.972	0.994
rose~ROC	50	0.991	0.001	0.989	0.993
rose~Sens	50	0.977	0.003	0.969	0.981
rose~Spec	50	0.949	0.003	0.943	0.957

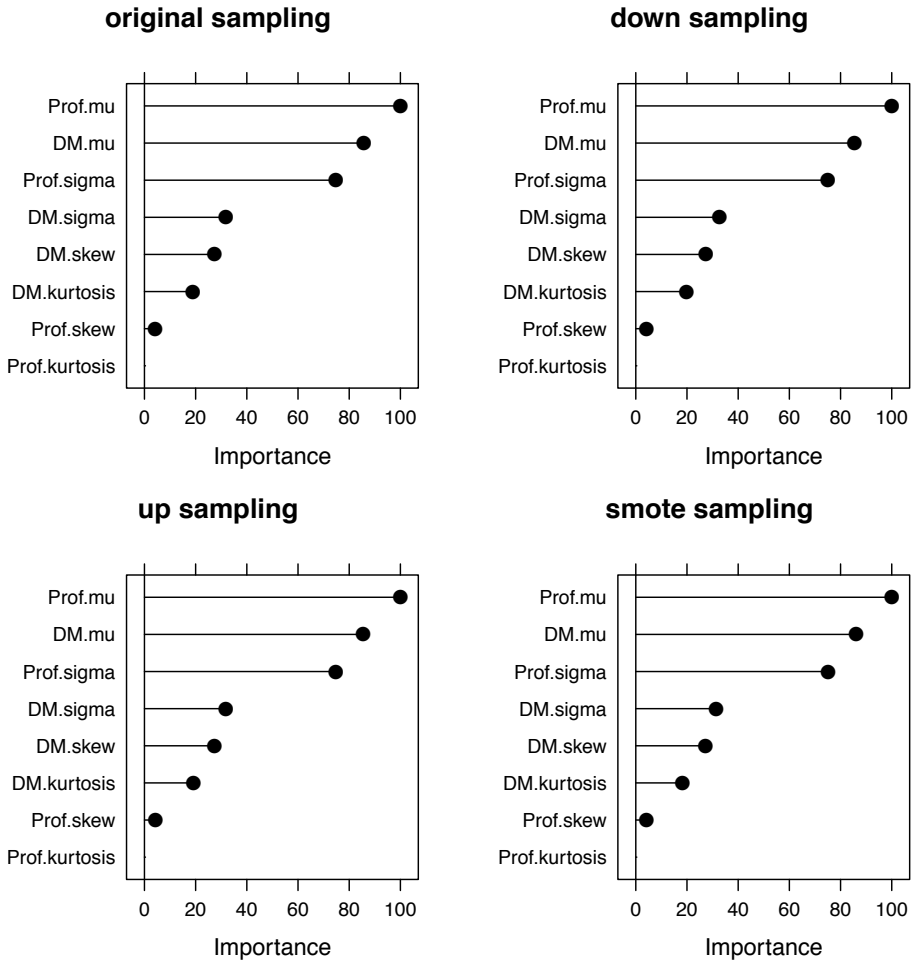
FPR of 1.53%, and an accuracy of 98.45%. The downsampled data set yielded the best performance for training the linear kernel SVM on this problem, if sensitivity is the primary metric. A summary of the training results are shown in the box and whiskers plot, and the full raw results are given in appendix C.

As in the case of the neural network models, some of the variables ended up not having any predictive value for some of the models. Curiously, the profile mean was an all-or-nothing case. For all but one of the models, it had the highest variable importance. But for the other, it was at the very bottom of the importance list, as can be seen in Figure 4.5, where the ROSE data set showed the "profile.mu" feature had no importance in the model, while in the other models, this was of top importance.

C5.0 Classifier

A C5.0 classifier was trained in the same way that the neural network and SVM were trained. The C5.0 classifier used three training parameters whose values for the

Figure 4.5: Support Vector Machine variable importance plots for four different sampling methods



final best model generated from each training data set are given in Table 4.7. The "model" parameter describes the type of model used in the classifier, either rule sets or decision trees. The "winnow" parameter controls whether to winnow out features, performing dynamic feature selection. The "trials" parameter controls the number of boosting iterations.

Figure 4.7 shows a Box and Whiskers plot of the results of training the five C5.0 models with the training data. This plot shows that the sensitivity, area under the

Figure 4.6: Support Vector Machine variable importance plots for ROSE sampling methods

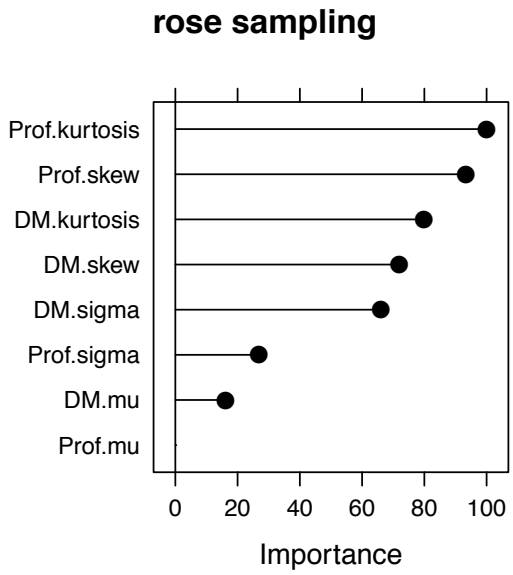
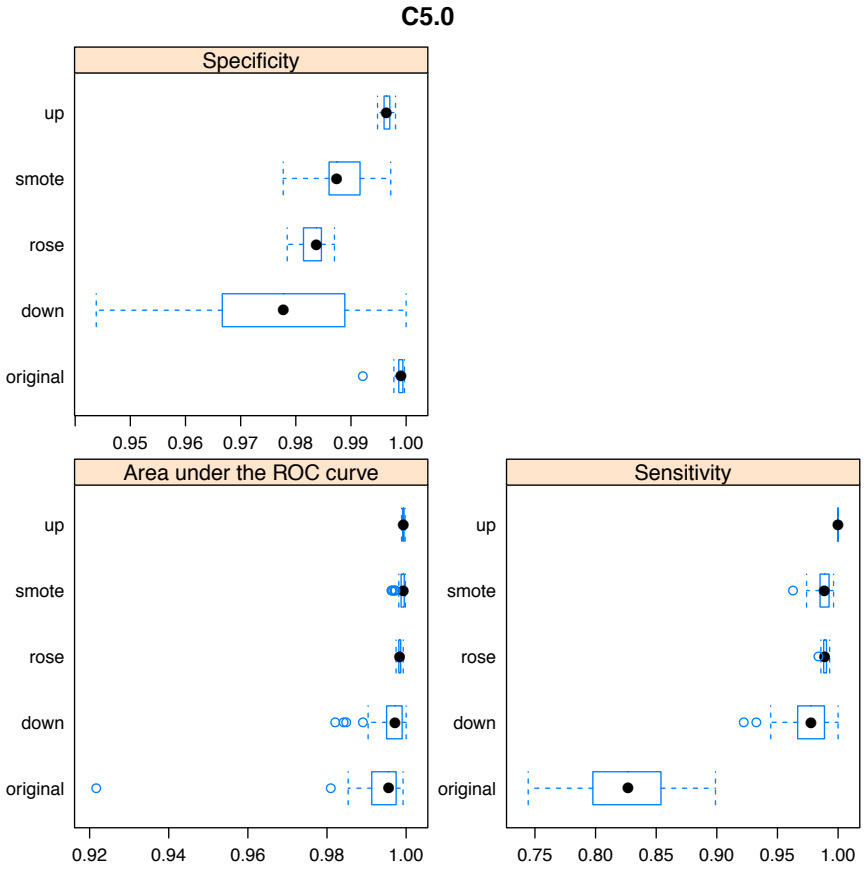


Table 4.7: C5.0 Models

data	model	winnow	trials
original	tree	false	30
down	rules	false	50
up	tree	true	1
SMOTE	rules	false	50
ROSE	tree	false	50

ROC curve, specificity (and by extension, false positive rate) are again all improved by resampling the data. Table 4.8 shows the statistics of the model generation results for all of the models across the metrics returned by the twoClassSummary function. Note that the data from the upsampled data model indicate that this model will perform the best on the test data, however the numbers from the testing do not bear this out. The models using the upsampled data are likely overfitted, causing poorer performance on the test data set. Note, however, that in spite of the poorer performance relative to the training data, the performance is still quite a bit better

Figure 4.7: Box and Whiskers plot of the three metrics returned from training the models



than the performance of models trained on the unmodified training data set.

The sensitivity achieved with final model fit with the original unresampled data set was 79.930%. The false positive rate was 0.12%. The accuracy of this model is 99.62%. Again, modified data sets gave better results for the primary metric. Using the downsampled data set for training, the system achieved a sensitivity of 97.659%, and a FPR of 1.72%. The accuracy figure for this machine is also slightly worse than that using the original training data, at 98.27%. Using the upsampled data set, the best sensitivity was 89.298%, far worse than the downsampled data set. The FPR was 0.41%, and the accuracy was 99.45%. The SMOTE-modified data set provided

Table 4.8: C5.0 Model Training Summary Statistics

Data Set~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	0.993	0.011	0.922	0.999
original~Sens	50	0.822	0.037	0.744	0.899
original~Spec	50	0.999	0.001	0.992	1.000
down~ROC	50	0.996	0.004	0.982	1.000
down~Sens	50	0.975	0.018	0.922	1.000
down~Spec	50	0.980	0.015	0.944	1.000
up~ROC	50	0.999	0.0002	0.999	1.000
up~Sens	50	1.000	0.000	1	1
up~Spec	50	0.996	0.001	0.995	0.998
smote~ROC	50	0.999	0.001	0.996	1.000
smote~Sens	50	0.987	0.007	0.963	0.996
smote~Spec	50	0.988	0.005	0.978	0.997
rose~ROC	50	0.998	0.0004	0.997	0.999
rose~Sens	50	0.989	0.002	0.984	0.993
rose~Spec	50	0.983	0.002	0.978	0.987

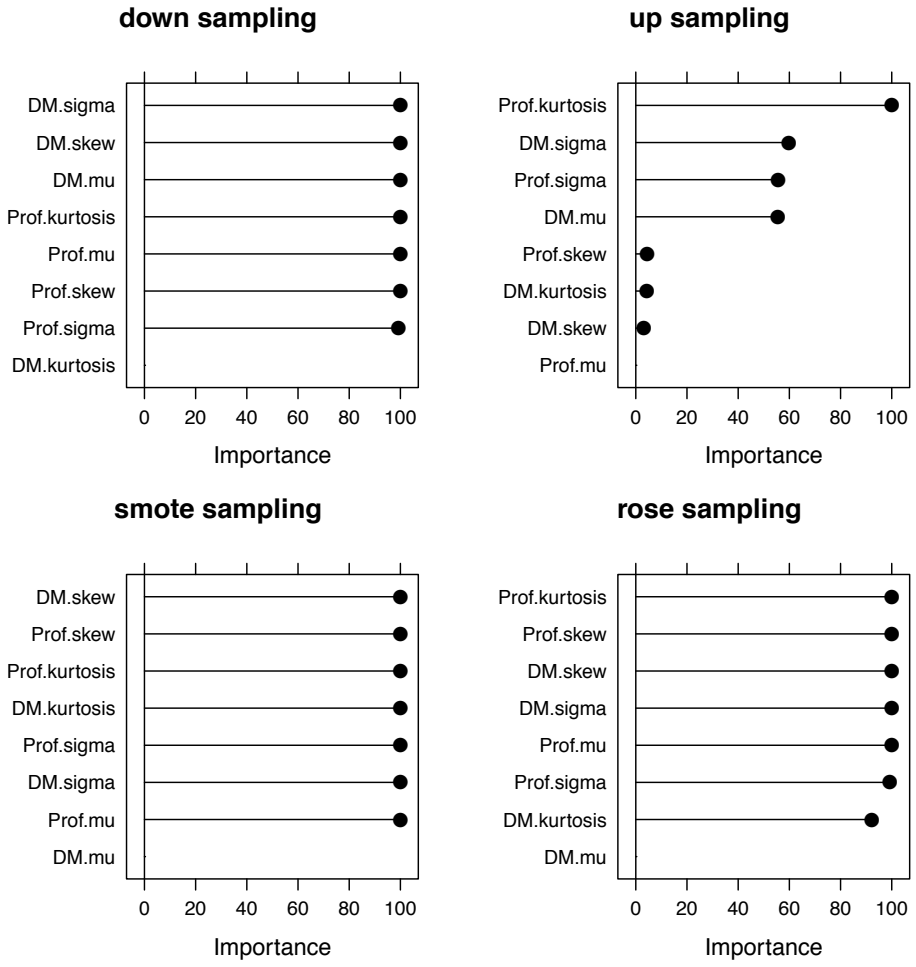
a sensitivity of 96.656%, an FPR of 1.18%, and an accuracy of 98.79%. Using the ROSE technique gave a sensitivity of 92.310%, an FPR of 0.51%, and an accuracy of 99.39%. These results show that training with the downsampled data set provided a model with the best performance for sensitivity. The full statistics are given in appendix D.

As in the previous cases, some of the variables ended up not having any predictive value for some of the models. The model importance plot for the original data set model showed 100% importance for all, so no plot was generated.

Ensemble Classifiers

Ensembles of bagged tree classifiers were also extensively studied. The effects of resampling on the classifiers produced has been explored, and the results are presented in this section. Using the caret treebag interface to the R Recursive PARTitioning (rpart) library (Therneau, Atkinson, & Ripley, 2015), a freely available version of the

Figure 4.8: C5.0 Model Variable Importance plots for four different sampling methods



Classification and Regression Trees (CART) algorithms of Breiman (Breiman, 1996).

Using the five training data sets, the bagged trees were trained and evaluated. The models trained with the original data set provided sensitivity of 81.610% and a FPR of 0.01%. Accuracy of this model was 99.63%. Training with downsampled data provided a model with a sensitivity of 98.328%, a FPR of 1.87%, and an accuracy of 98.13. Models trained with upsampled data provided a mediocre sensitivity of 85.953%, a FPR of 0.27%, and an accuracy of 99.55%. SMOTE-trained models provided better results, with a sensitivity of 96.321%, a FPR of 1.29%, and an accu-

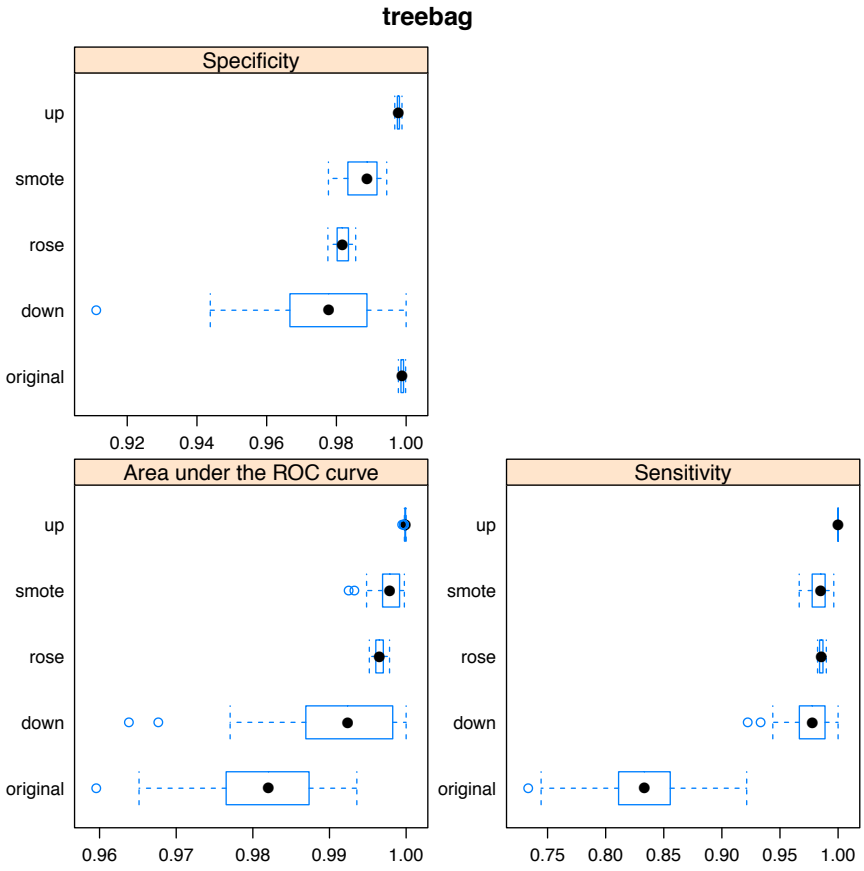
Table 4.9: Recursive Partitioned Tree Model Training Summary Statistics

Data Set~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	0.981	0.007	0.960	0.994
original~Sens	50	0.827	0.039	0.733	0.921
original~Spec	50	0.999	0.0004	0.998	1.000
down~ROC	50	0.992	0.008	0.964	1.000
down~Sens	50	0.974	0.017	0.922	1.000
down~Spec	50	0.975	0.018	0.911	1.000
up~ROC	50	1.000	0.0001	0.999	1.000
up~Sens	50	1.000	0.000	1	1
up~Spec	50	0.998	0.0005	0.997	0.999
smote~ROC	50	0.998	0.002	0.992	1.000
smote~Sens	50	0.984	0.008	0.967	0.996
smote~Spec	50	0.987	0.005	0.978	0.994
rose~ROC	50	0.996	0.001	0.995	0.998
rose~Sens	50	0.986	0.002	0.982	0.990
rose~Spec	50	0.982	0.002	0.978	0.986

racy of 98.68%. ROSE-trained models fared better than the upsample-trained models with a sensitivity of 93.980%, a FPR of 0.62%, and an accuracy of 99.31%. Figure 4.9 shows a box and whiskers plot of the results of testing the five models with the full set of training data. This plot shows that the sensitivity, area under the ROC curve , and specificity are all negatively affected by downsampling the data in this case, although all of the resampling methods improved the sensitivity over the original model built with the highly imbalanced training data. Table 4.9 shows the training results for these models. The complete statistics and training details are given in Appendix F.

As in the sections on the other models, the variable importance varied greatly. Figure 4.10 shows the variable importance results from the training sessions with the bagged tree algorithm.

Figure 4.9: Box and Whiskers plot of the three metrics returned from training the models

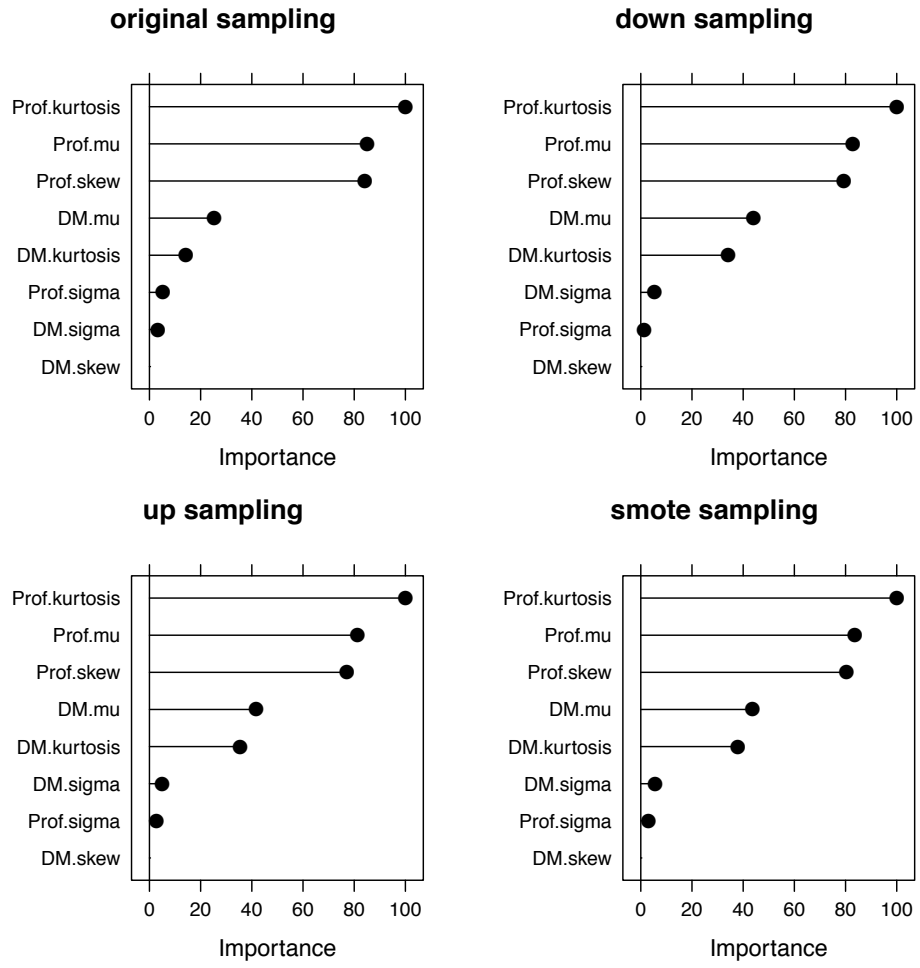


Stacked Classifiers

In addition to the bagged trees, a stacked ensemble classifer was made using the classifiers from the group above. A classifier was created that uses the output from each of these models and combines it with another classifier that decides which outcome to assign to each example. A neural network and a C5.0 classifier were used for this work for the top models, along with an ad-hoc simple ensembling method.

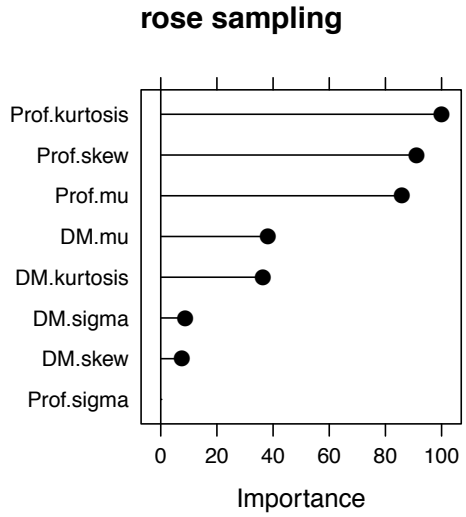
The same training sets were used for these models, and the models were trained in a similar fashion to the base models, but there was one additional step in the

Figure 4.10: Bagged Tree Model Variable Importance plots for four different sampling methods



process that ties together the base model predictions and the stacked ensemble. Before training the stacked classifiers, the prediction results of each of the base models was added to the training data as additional features in the data set. Then the same training algorithms were used as for training the base models. The models were trained and evaluated using the same methods as previously used for the base models. All of the stacked ensemble models performed extremely well in the cross-validation and resampling evaluations as shown in Tables 4.11 and 4.13, but they didn't perform as well in testing against new data.

Figure 4.11: Bagged Tree Model Variable Importance plots for the ROSE sampling method



Simple Ensemble

A simple ensemble was constructed by simply taking the results of the base models, summing the predicted value for each example, and using a threshold to predict the class. This produced remarkably good results for such a simple technique, suggesting that there is not much more information left in the training data that can be extracted during the training process for the more complex ensembled models. The sensitivity of this simple ensemble using the downsampled data (the best case) was 99.310%, with a false positive rate of 2.76%. The accuracy for this system was 97.27%.

Neural Network Top Model

The number of hidden neurons and decay rate parameters for the final top models chosen through training iterations for the neural network are given in Table 4.10. The results for this classifier followed very closely the results seen previously for classifiers trained on a specific training data set, that is, better base models produced better

Table 4.10: Neural Network Models

Data Set	Hidden Neurons	Decay Rate
original	5	0.1000
downsample	1	0.1000
upsample	1	0.1000
SMOTE	7	0.0006
ROSE	1	0.1000

stacked classifiers. All except the downsampled training sets produced classifiers that were inferior to the base models that they were built from, suggesting the models are being overfitted. Table 4.11 shows the results of training the neural network on the training data including the base model predictions. The trained networks were nearly perfect in both sensitivity and in specificity in all cases except when trained on the original data. In practice, they didn't perform as well. The sensitivity achieved with the classifiers using base models built with downsampling (the best performers) was 97.659%, and the false positive rate was 1.51%. The accuracy value was 98.43%.

C5.0 Ensemble Classifier

Using the same techniques as described above for the neural network ensemble classifier, a C5.0 meta-classifier was trained with the ensemble training data. The most effective classifier for each training data set was produced with the parameters shown in Table 4.12. The best sensitivity was acheived with both up and downsampling models. The sensitivity was 98.328%, with a false positive rate of 1.87%, and an accuracy of 98.13%. Table 4.13 shows the training statistics, which show that the classifier performs nearly perfectly in most trials on the training data, suggesting that this classifier is also overfitting on the training data.

Table 4.11: Neural Network Training Results, Base Models Built With Downsampling

Data Set ~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	1.000	0.001	0.997	1.000
original~Sens	50	0.993	0.009	0.966	1.000
original~Spec	50	1.000	0.0001	1.000	1.000
down~ROC	50	1.000	0.000	1	1
down~Sens	50	1.000	0.000	1	1
down~Spec	50	1.000	0.000	1	1
up~ROC	50	1.000	0.00002	1.000	1.000
up~Sens	50	1.000	0.000	1	1
up~Spec	50	1.000	0.0001	1.000	1.000
smote~ROC	50	1.000	0.000	1	1
smote~Sens	50	1.000	0.001	0.996	1.000
smote~Spec	50	1.000	0.000	1	1
rose~ROC	50	1.000	0.0001	0.999	1.000
rose~Sens	50	1.000	0.0002	0.999	1.000
rose~Spec	50	1.000	0.0002	0.999	1.000

Table 4.12: C5.0 Ensemble Models

data	model	winnow	trials
original	rules	true	1
down	rules	true	1
up	rules	true	1
SMOTE	rules	false	1
ROSE	rules	true	1

Table 4.13: C5.0 Classifier Training Statistics

Data Set~Metric	Iterations	Mean	St. Dev.	Min	Max
original~ROC	50	0.996	0.005	0.983	1.000
original~Sens	50	0.992	0.009	0.966	1.000
original~Spec	50	1.000	0.0001	1.000	1.000
down~ROC	50	0.999	0.002	0.994	1.000
down~Sens	50	1.000	0.000	1	1
down~Spec	50	0.999	0.003	0.989	1.000
up~ROC	50	1.000	0.00003	1.000	1.000
up~Sens	50	1.000	0.000	1	1
up~Spec	50	1.000	0.0001	1.000	1.000
smote~ROC	50	1.000	0.001	0.996	1.000
smote~Sens	50	1.000	0.001	0.993	1.000
smote~Spec	50	1.000	0.000	1	1
rose~ROC	50	1.000	0.0002	0.999	1.000
rose~Sens	50	1.000	0.0001	0.999	1.000
rose~Spec	50	1.000	0.0003	0.999	1.000

Chapter 5

Conclusions

Conclusions

The use of machine learning in the field of pulsar search is quite feasible, and necessary for future generations of telescopes. The experiments undertaken as part of this research have shown that very high sensitivity can be achieved while maintaining good specificity. The combination of the two metrics is important, but sensitivity is the primary optimized variable. This is because there are very few pulsars, and it is not acceptable to let many of them slip away undetected.

The particular learning model was much less important than the processing of the training data. Effective classifiers were built using a neural network, a support vector machine, a C5.0 algorithm, and a bagged tree ensemble. Due to the highly imbalanced nature of the pulsar data, it was found that changing the mix of samples in the training set by over or undersampling the training data has a profound effect on the training results. Training with a standard unmodified training set produced mediocre results, but when the training set was modified, the sensitivity improved greatly. Using the ROSE or SMOTE technique, or even by simply under and oversampling the data to produce more balanced data was key to getting acceptable performance from the all of the models. Ensembling classifiers to produce better results was not effective in this case, because the base models used in the ensemble were highly correlated, so together they did not add value to the ensemble. In addition, the ensembles did not generalize well, doing very well on the training data, but not on the test data. A simple ensemble assembled from the base models using simple arithmetic combination (essentially a logical "or" function) worked very well, but gave more false positives

than might be desired, and it was still within the goal specified in the propsal for this work.

The experience of using the R programming language for machine learning development has been quite smooth and productive. The tools for visualizing the results (such as the box and whiskers plots) are superb, and the nature of the scripting language and the remarkably complete ecosystem around it brings immense power to bear on the problems. As the system is cross-platform, and the problems are easily parallelizable, it is easy to scale up as problem size increases. The final program used to train,test, and evaluate the models reported on here took more than 750 CPU-hours on a MacBook Pro with 16 GB of memory and a solid state disk, so the ability to run the system on larger machines is imperative.

Implications

The implications of the results of this research are that even with a small number of training examples in a sea of negative examples, a system can be produced that can discriminate between the cases. The algorithms studied (neural networks, support vector machines, bagged trees, C5.0 tree algorithms) all were improved by pre-processing the data to improve the balance of positive and negative examples in various ways. Additionally, the feature selection performed by Lyon et al. (2016) must be revisited, as the features seem to be correlated, and so may not be the optimal feature set to use in this application. Starting off with the 22 features of Bates et al. (2012) and using feature extraction techniques to find the best ones may yield fruit.

Recommendations for Future Work

There are several activities that can be continued in this line of research. One of these is to examine how the pulsar data sets differ from obseratory to observatory, and pulsar machine to pulsar machine, and how to best generalize the models that

are produced by one group so that they can be used by other groups. To this end, the Pulsar Feature Lab Lyon et al., 2016 should be encouraged and nurtured. The Pulsar Feature Lab is a software suite for generating different feature sets from a common data source, thereby providing a reliable and repeatable source of data for experiments and training. Adopting this suite as a common tool for pulsar classification data would do much to make sure that all of the research is undertaken with similar tools and data.

Another study that could be undertaken to help further the field is more research in feature sets. Even though the eight features in the Lyon et al. (2016) feature set were selected in a rigorous and deliberate fashion, some of them turned out to have no predictive value in the models in this research. The 22 features used by Bates et al. (2012) and the 8 features used by Lyon et al. (2016) can be combined into one data set, and then automated feature extraction tools can be brought to bear to attempt to sort out the important features from the unimportant features in practical model generation.

A third topic for future exploration is to build a production pipeline that can look at every DM trial and fold period output to look for low signal to noise ratio pulsars that are filtered out now, since there are too many plots to look at by hand.

Extending this research by a foray into the fields of deep learning and image processing may also pay dividends. The signals in the pulsar search area are complex and nuanced, and problem seems like it would be a good fit to the emerging deep learning technology. Image processing techniques such as Hough transforms, edge enhancement, or other techniques to sharpen the features visible to human eyes in the pulsar candidate plots such as figure 1.4 may make it easier for machines as well as humans to pick out pulsars from noise. Finding pulsars in low SNR environments

is also a challenge, and this challenge becomes more important as time goes on and the "low-hanging fruit" of bright sources are discovered, leaving the weaker signals to be pulled up out of the noise.

Summary

Pulsars are rapidly rotating neutron stars which emit a strong beam of energy through mechanisms that are not entirely clear to physicists. These very dense neutron stars are used by astrophysicists to study many phenomena. Fundamental tests of general relativity can be made using them as tools (D. Lorimer and Kramer, 2005). Currently, an experiment is being performed by the North American Nanohertz Observatory for Gravitational Waves (2012) to try to detect gravitational waves by studying the timing variations of an array of pulsars scattered around the celestial sphere. These experiments in fundamental physics require a large set of pulsars for study, providing the impetus for systematically searching the sky for new pulsars. Pulsars discovered in ongoing pulsar surveys, as well as in the reprocessed data of several archival surveys, are continually being added to the census of pulsars in the nearby universe.

There are more and larger pulsar surveys being planned for current and future telescopes. The Five Hundred Meter Spherical Telescope being built in China will be able to find as many as 5,000 pulsars in a short amount of time (Smits, Lorimer, et al., 2009). The Square Kilometer Array (SKA) is expected to find more than 20,000 pulsars (about 10 times the number currently known!) (Smits, Kramer, et al., 2009). As the ratio of candidates to confirmed pulsars is about 10,000:1, upwards of 200 million candidates will need to be examined to complete the SKA survey. Clearly, it is impractical to examine all 200 million candidates with human eyes. Searching for pulsars is a very labor-intensive process, currently requiring skilled people to examine

and interpret plots of data output by analysis programs. An automated system for screening the plots would speed up the search for pulsars by a very large factor.

Research in automated pulsar search is in its infancy. Research in machine learning for building classifiers is a mature area. One characteristic of the pulsar search problem that makes this research interesting is the imbalanced training data available. The very small number of known pulsars and the large amount of data available means that the training must be done with a very imbalanced data set, or else large amounts of data without pulsars must be discarded. Other machine learning techniques have not been used in identifying pulsars thus far, and it is worthwhile to consider some of these techniques.

Research to date on using machine learning and pattern recognition has not yielded a satisfactory system. The first published attempt to use a machine learning approach to detect pulsars in diagnostic data was published by Eatough et al. (2010). Their work used 14,400 pointings out of the Parkes Multibeam Pulsar Survey (PMPS), one of the largest comprehensive searches undertaken to date (Manchester et al., 2001). The 14,400 beams were processed through their standard pulsar search pipeline, generating 2.5 million candidate plots containing possibly all types of pulsars: binary pulsars, slow pulsars, and millisecond pulsars. Out of these 2.5 million plots, 501 pulsars were found by manual means, yielding a very small $\frac{501}{2{,}500{,}000} = 0.02\%$ success ratio. Such a small success ratio makes the job of manually viewing the plots tedious and extremely error-prone. An automated method of screening the candidates is needed to reduce the human effort needed to examine the candidate plots. Eatough et al. (2010) used an Artificial Neural Network (ANN) as a binary classifier to screen the 2.5 million candidate plots, yielding some 13,000 candidates to manually screen. Unfortunately, only around 92% of the pulsars were recovered.

The goal of this research was to develop an improved method for pulsar identification using supervised machine learning techniques. Private communications with pulsar scientists (Demorest, 2013; Ransom, 2013) set the goals as

- The false positive rate should be less than 5%.
- Precision of the new algorithm should be greater than the 3.6% of the current state of the art.
- Recall greater than 99% (Less than 1% of the pulsars are missed)

This work proposed to research, identify, and propose methods to overcome the barriers to such a system. The results reported in the previous chapter show that it is possible to generate classifiers that perform as needed from the available training data. While a false positive rate of 1% was not reached, recall of over 99% was achieved with a false positive rate of less than 2%, meeting the requirements of pulsar scientists as noted above.

Methods of mitigating the imbalanced training and test data were explored and found to be highly effective in enhancing classification accuracy. Experiments used majority undersampling (downsampling), minority oversampling (upsampling), ROSE sampling, and SMOTE sampling to modify the data sets used for training in an effort to improve the mediocre classification accuracy obtained using the original training data. This research showed that all of the sampling methods improved the recall of the models dramatically, allowing up to 99% recall of the test data pulsars.

Ensembles of classifiers were built and tested in an effort to maximize the recall of pulsars from the test data. Mixed results were obtained from stacked ensembles, with a simple logical "or" of the classification results of the base models increasing the recall without impacting the false positive rate dramatically. More sophisticated stacked

ensembles suffered from overfitting, leaving room for more research into combining the algorithms with limited training data. Bagged trees performed well, however.

Automated pulsar search appears to now be feasible, and is absolutely required to deal with the volume of data expected from instruments under construction.

Appendix A

R Packages and scripts

The packages and scripts used to train and evaluate the models are given in this appendix. Just two scripts were finally produced after much development and testing of alternatives.

Common Script with No Preprocessing

This script reads in the data from the data file, and formats it and creates labels for the pulsar and non-pulsar cases. It loads in common libraries and sets up the environment for further processing by the actual training and testing script. This script made it easy to experiment with alternative models and steps from the R command line by loading in and setting up the training and testing data.

```
##
##
## This script is what is needed to create the data set that can be
## passed to a classifier.  The data is the HTRU-[1,2] data from from
## Lyon et al feature set
## It:
## Reads in the data file
## Normalizes the numerical values
## Relabels the target features to be non-numeric, as some code is
## confused by numerical labels on the features.
## Creates a set of training and test data, using 75% for training.
## Sets up a parallel execution environment to take advantage of
## multiple cores on the machine
##
##
## Load in the necessary libraries
##

library(doParallel)
library(caret)
```

```
normalize <- function(x) {
    return( (x -min(x))/(max(x) - min(x)))
}

## Point to the CSV version of whatever data you want to analyse.
## This is for the Lyon, et al. feature set.

theData <-'~/path/to/the/data/file'

##
## Names for convenience of humans. Code doesn't care...
##
featureNames  <-  c("Prof-mu", "Prof-sigma","Prof-kurtosis","Prof-skew",
                    "DM-mu", "DM-sigma","DM-kurtosis","DM-skew","Class")

## for reproduceability
mySeed <- 123

pd <-  read.table(theData, header=FALSE,sep = ",", col.names = featureNames)
pdd <- pd[,-9]
pdl <- pd[,9]    ## Chop off the class
pdn <-pdd

## Make the pulsar class the lowest numbered factor so it will default
## to the positive value.  This is needed to train for "sensitivity".

pdl<-replace(pdl, pdl==0, 2)
pdn[9] <- factor(pdl)
## Stick the class column back on after converting to factor type.
names(pdn)[names(pdn) == 'V9'] <- 'Class'    ## label the class column

##
## Some of the libraries object to using "0" and "1" as the class
## names!
##

levels(pdn$Class) <- c('pulsar','nonpulsar')

## Now we start munging data!

inTrain <- createDataPartition(pdn$Class, p = 0.75,list = FALSE)
## Create a test and train data set
```

```
pulsarTrain <- pdn[inTrain, ]
pulsarTest <- pdn[-inTrain, ]

## Parallelize!
cl <- makeCluster(4,outfile="")
registerDoParallel(cl)

##For reproduceability
set.seed(mySeed)
```

Script to train, test, and evaluate all of the models

.

The following script was the result of much experimentation and trials of small sections of the script on small parts of the problems. The entire script takes over 750 CPU-hours to run on a MacBook Pro with a 2.5 GHz Intel Core I7 processor, 16 GB of memory, and a solid-state disk. The script trains, selects, tests, generates plots and text output for all of the models.

```
##
##
## This script reads in a data set, creates modified training data
## from it, trains a list of models on these data sets.  It then
## collects the results of testing, and creates an ensemble of the
## best models from the set of base models.  It tests the ensemble,
## capturing the data as before.

## This reads in the data and puts it in a form the learning models
## like
##
##
source('CommonNoPreproc.R')

##
## Load in the necessary libraries
##
library(kernlab)
library(gmodels)
library(ROSE)
library(DMwR)
```

```
#
#Function to test the models using sensitivity as the metric
                                        #

test_sensitivity<- function(model,data, file) {
    ct = confusionMatrix(predict(model,data),data$Class,
                         positive = 'pulsar')
    capture.output( print(ct), file = file, append=TRUE)
    return(ct)
}

test_roc <- function(model, data) {
    library(pROC)
    roc_obj <- roc(data$Class,
                   predict(model, data, type = "prob")[, "nonpulsar"],
                   levels = c("pulsar", "nonpulsar"))
    ci(roc_obj)
}

print_model_text<- function(model,theFile) {
    capture.output( print(model),
                   file=theFile,append = TRUE)

}

date.time.append <- function(str, sep = '_',
                             date.format ="%Y_%m_%d_%H_%M_%S") {
    stopifnot(is.character(str))
return(paste(str, format(Sys.time(), date.format), sep = sep))
}

createDataSets <- function( data, seed )
{
    ## Downsampling the majority Class
    set.seed(seed)
    dsPulsarTrain<- downSample(x = data[, -ncol(data)],
                               y = data$Class)

    table(dsPulsarTrain$Class)
    ## Upsampling the majority Class
    set.seed(seed)
    usPulsarTrain <- upSample(x = data[, -ncol(data)],
                              y = data$Class)
```

```
    table(usPulsarTrain$Class)

    ## Synthetic Minority Oversampling the data

    set.seed(seed)
    smotePulsarTrain <- SMOTE(Class ~ ., data  = data)
    table(smotePulsarTrain$Class)

    ## ROSE sampling the data
    set.seed(seed)
    rosePulsarTrain <- ROSE(Class ~ ., data  = data)$data
    table(rosePulsarTrain$Class)

    ##
    ## Create a list of training data
    ##

    trainingData = list(original = data,
                        down = dsPulsarTrain,
                        up = usPulsarTrain,
                        smote = smotePulsarTrain,
                        rose = rosePulsarTrain)
    return (trainingData)
}

##
## Fit the Models with different data sets, save them to a list
##
##  This data structure holds ALL of the data for the processing run

testfunc <- function( m,d)
{
    md <- vector("list",length(m))
    md <- setNames(md,m)
    for ( i in m )
    {
        md[[i]] <- d
    }

    return( md )
}

trainModels <- function(mdls,dataset,seed,
```

```
                        ctrl,verbosity,myTuneLength,myMetric)
{
    ## The vector of results.  Each vector element is a list of length
    ## `length(dataset)`

    models <- vector("list",length(mdls))
    models <- setNames(models,mdls)

    for( name in mdls)
    {
        myMethod = name

        set.seed(seed)

        orig <- caret::train(Class ~ ., data = dataset$original,
                             method = myMethod,
                             preProcess = c("scale","center"),
                             verbose= verbosity,
                             metric = myMetric,
                             tuneLength = myTuneLength,
                             trControl = ctrl)
        print(orig)

        set.seed(seed)

        down <- caret::train(Class ~ ., data = dataset$down,
                             method = myMethod,
                             metric = myMetric,
                             preProcess = c("scale","center"),
                             verbose= verbosity,
                             tuneLength = myTuneLength,
                             trControl = ctrl)
        print(down)

        set.seed(seed)
        up <- caret::train(Class ~ ., data = dataset$up,
                           method = myMethod,
                           metric = myMetric,
                           preProcess = c("scale","center"),
                           verbose= verbosity,
                           tuneLength = myTuneLength,
                           trControl = ctrl)
        print(up)
```

```
        set.seed(seed)
        smote <- caret::train(Class ~ ., data = dataset$smote,
                              method = myMethod,
                              preProcess = c("scale","center"),
                              verbose= verbosity,
                              tuneLength = myTuneLength,
                              metric = myMetric,
                              trControl = ctrl)
        print(smote)

        set.seed(seed)

        rose <- caret::train(Class ~ ., data = dataset$rose,
                             method = myMethod,
                             preProcess = c("scale","center"),
                             verbose= verbosity,
                             tuneLength = myTuneLength,
                             metric = myMetric,
                             trControl = ctrl)
        print(rose)

        models[[name]] <- list(original = orig,
                       down = down,
                       up = up,
                       smote = smote,
                       rose = rose)
    }

    return(models)
}

trainEnsembleModels <- function(topMdls,mdls,trainingData,testData,
                                seeds,verbosity,tunel,metric)
{
    ## The vector of results.  Each vector element is a list of
    ## length `length(dataset)`

    models <- vector("list",length(topMdls))
    models <- setNames(models,topMdls)
    seed = mySeed
    for( name in topMdls)
    {
        myMethod = name
```

```
        set.seed(seed)
        orig <- ensemble(name,mdls,'original',dataSet,testData,
                         "Class", seeds,verbosity,tunel,metric)
        print(orig)

        set.seed(seed)

        down <- ensemble(name,mdls,'down',dataSet,testData,"Class",
                         seeds, verbosity,tunel,metric)
        print(down)

        set.seed(seed)
        up <- ensemble(name,mdls,'up',dataSet,testData,"Class",seeds,
                       verbosity ,tunel,metric)
        print(up)

        set.seed(seed)
        smote <- ensemble(name,mdls,'smote',dataSet,testData,"Class",
                          seeds, verbosity,tunel,metric)
        print(smote)

        set.seed(seed)
        rose <- ensemble(name,mdls,'rose',dataSet,testData,"Class",
                         seeds, verbosity,tunel,metric)
        print(rose)

        models[[name]] <- list(original = orig,
                       down = down,
                       up = up,
                       smote = smote,
                       rose = rose)
    }

    return(models)
}

## Pass in the list of models.  This is a 2d list of lists, organized
## by method first, then dataset.

testAndOutput2 <- function( label, mdls, testdata )
{
    results <- list()
    graph <- list()
```

```
## Create a common base name for the run
dirName <-date.time.append(label)
dir.create(dirName)
## for each method, we summarize the results
for ( name in names(mdls))
{
    myFileBase <- paste(dirName,name,sep="/")
    myTxtFile<-paste(myFileBase,".txt",sep="")
    cat(sprintf("Summary Text Output for the %s model\n\n",name),
        file = myTxtFile)
    ## name is one of the "'label'Models"
    print(sprintf("Applying model %s",name))
    results$preds[[name]] <- lapply(mdls[[name]],test_sensitivity,
                                    data = testdata,file=myTxtFile)
    results$preds[[name]] <- lapply(results$preds[[name]], as.vector)
    results$preds[[name]] <- do.call("rbind", results$preds[[name]])
    results$preds[[name]] <- as.data.frame(results$preds[[name]])

    print("Resampling models")
    results$models_resamples[[name]] <- resamples(mdls[[name]])
    ## Generate plots and output text ##

    ## Make some plots
    print(paste("Making a dotplot for",name))
    pdf(paste(myFileBase,"dotplot.pdf",sep="_"))
    print(dotplot(results$models_resamples[[name]],main=name, scales = list(relation
    Sys.sleep(0.2)
    dev.off()
    print(paste("Making a bwplot for",name))
    pdf(paste(myFileBase,"BWplot.pdf",sep="_"))
    print(bwplot(results$models_resamples[[name]],main=name, scales = list(relation
                strip=strip.custom(var.name='metric',strip.names= c(FALSE,TRUE),
                                   factor.levels=c("Area under the ROC curve", "Sen
                                                   "Specificity"), par.strip.text=l
    Sys.sleep(0.2)
    dev.off()
    print("Calculating model summary")

    results$modelSummary[[name]] <-
        summary(results$models_resamples[[name]])
    print("Calculating model differences")
    results$modelDiff[[name]] <-
        diff(results$models_resamples[[name]])
    print("Calculating model variable importance")
```

```
results$modelVarImp[[name]] <- lapply(mdls[[name]],varImp)
print("Creating variable importance plot")

results$modelImportance <- results$modelVarImp[[name]]
pdf(paste(myFileBase, "varImpplot.pdf",sep="_"))
row = 1
column = 1
pg=TRUE;
for( miName in names(results$modelImportance))
{
    if(name == 'C5.0' && miName == 'original')
    {
        next
    }
    graph[[miName]] = plot(results$modelImportance[[miName]],
                           main=paste(miName,"sampling"),
                           col="black",cex=1.25)
    print(plot(graph[[miName]],split= c(row,column,2,2),newpage=pg))
    pg =FALSE
    row = row + 1
    if( row > 2)
    {
        row = 1
        column = column + 1
        if(column >2)
        {
            column=1
            pg=TRUE
        }
    }
}
Sys.sleep(0.2)
dev.off()

##
## Create the file and dump the output stats...
##

result = lapply(mdls[[name]],print_model_text,myTxtFile)
capture.output(print(summary(results$modelDiff[[name]])),
               file=myTxtFile,append=TRUE)
capture.output(print(results$modelVarImp[[name]]),
               file=myTxtFile,append = TRUE)
capture.output( print(results$modelSummary[[name]]),
```

```
                        file=myTxtFile,append = TRUE)

    } ## end of for loop

    return(results)

}     ## End of evaluate and print function



ensemble <- function(ensembleModel,mdls, sampleMethod, trainingData,
                     testing,outcomeName,seed,verbosity,myTuneLength,
                     myMetric)
{
    numFolds = 10
    numRepeats = 5
    training <- trainingData[[sampleMethod]]
    print(str(training))

    topCtrl <- trainControl(method = "repeatedcv", number = numFolds,
                            repeats = numRepeats,
                            preProcOptions=c("scale","center"),
                            classProbs = TRUE, seeds = seed,
                            verboseIter = TRUE,
                            summaryFunction = twoClassSummary)
    ##
    ## this loop runs the given ensembler over
    ## One of the training data sets available.

    ## record the original predictor varlables.
    predictors <- names(training)[!names(training) %in% outcomeName]

    ## for each of the models in the baseModels, evaluate the models
    ## on the data and save the results in the original data
    ## structures

    for( mName in names(mdls))
    {
        print(paste("evaluating",mName))
        model <- mdls[[mName]][[sampleMethod]]
        testing[,(ncol(testing)+1)] <-
            caret::predict.train(object=model, testing[,predictors])
        training[,(ncol(training)+1)] <-
            caret::predict.train(object=model, training[,predictors])
```

```
    }
    ## add the predictors for the top model that are the output of the
    ## base models

    ePredictors <- names(training)[!names(training)
                                   %in% outcomeName]

    modelFinal <- caret::train(training[,ePredictors],
                               training[,outcomeName],
                               method=ensembleModel,
                               preProcess = c("scale","center"),
                               metric=myMetric,
                               tuneLength = myTuneLength,
                               verbose=verbosity,
                               trControl= topCtrl)

    return(modelFinal);
}

getEnsembleTestData <- function(topMdls,mdls,testData)
{
        orig <- addColumns(mdls,'original', testData,"Class")
        down <- addColumns(mdls,'down',testData,"Class")
        up <- addColumns(mdls,'up',testData,"Class")
        smote <- addColumns(mdls,'smote',testData,"Class")
        rose <- addColumns(mdls,'rose',testData,"Class")
        testData <- list(original = orig,
                         down = down,
                         up = up,
                         smote = smote,
                         rose = rose)
    return(testData)
}

addColumns <- function(mdls,sampleMethod,testing,outcomeName)
{
    predictors <- names(testing)[!names(testing) %in% outcomeName]
    print(sampleMethod)
    for( mName in names(mdls))
    {
        print(paste("evaluating",mName))
        model <- mdls[[mName]][[sampleMethod]]
        testing[,(ncol(testing)+1)] <-
            caret::predict.train(object=model, testing[,predictors])
```

```
    }
    return(testing)
}

testEnsemble2  <- function()
{
    methods <-c('original','down','up','smote','rose')
    ensembleResults <- vector("list",length(methods))
    ensembleResults <- setNames(ensembleResults,methods)
    for(method in methods)
    {
        ensembleResults[[method]] <-
            testAndOutput2(paste('EnsembleTest',method,sep=''),
                           ensembleModels,
                           ensembleTestData[[method]])
    }
    return(ensembleResults)
}


## Try a simple ensemble using calculations
simpleEnsemble <-function(label)
{
    dirName <-date.time.append(label)
    dir.create(dirName)
    myFileBase <- paste(dirName,"simple",sep="/")
    myTxtFile<-paste(myFileBase,".txt",sep="")
    cat(sprintf("Summary Text Output for a simple model\n\n"),
        file = myTxtFile)
    for(sampling in c('original','down','up','smote','rose'))
    {
        capture.output(print(paste("Ensembling with",sampling)),
                       file=myTxtFile,append=TRUE)
        data <- ensembleTestData[[sampling]]
        theSum = as.numeric(data$V10) + as.numeric(data$V11) +
            as.numeric(data$V12) + as.numeric(data$V13)
        numResults <- theSum < 8
        dat <- numResults
        dat <- replace(dat,dat==0,2)
        dat <-factor(dat)
        levels(dat)  <- c('pulsar','nonpulsar')
        capture.output(print(confusionMatrix(dat,pulsarTest$Class)),
                       file=myTxtFile,append = TRUE)
    }
```

```
}

print(paste("Starting to train at", Sys.time()))

## Here we start defining the particulars for the run

##
## Choose sensitivity, as that is a synonym for recall
##
myMetric = 'Sens'

##
## Define the base models here and set the list element names
## for easy access
##

baseModels = c('nnet', 'svmLinearWeights','treebag','C5.0')
#baseModels = c('nnet', 'C5.0')

##
## define the Top models here
##

topModels = c('nnet','C5.0')

##
## Tell caret::train to try N random values for each tuneable
## parameter in the model
##
myTuneLength = 6

##
## Tell caret::train to use 5 repeats of 10 fold cross validation
##
numRepeats = 5
numFolds = 10

##
## Center the data around 0, with a width of 1 std deviation
##

myPreProcOptions=c("scale","center")
                                        #
##
```

```
##For caret training in parallel, we need to
##create a list of seeds, and change the seed for each resampling,
## but keep the list consistent for the whole training process
##

seeds <- vector(mode = "list", length = 51)


## fill in the first vector of seeds
for(i in 1:50) seeds[[i]]<- sample.int(n=1000, 130)

## The last one is only one long
seeds[[51]]<-sample.int(n=1000, 1)

baseTrainCtrl <- trainControl(method = "repeatedcv", number = numFolds,
                     repeats = numRepeats,
                     preProcOptions=myPreProcOptions,
     classProbs = TRUE, seeds = seeds,
                     verboseIter = TRUE,
                     summaryFunction = twoClassSummary)

##
## Set up the derived data sets (5 right now)
##

dataSet = createDataSets(pulsarTrain,mySeed)

##
## Fit the Models with different data sets, save them to a list
##
##  This data structure holds ALL of the data for the processing run
##  models is a 2 d data structure, basemodels rows, dataset columns

models = trainModels(baseModels,dataSet,mySeed, baseTrainCtrl,TRUE,
                     myTuneLength,myMetric)

##
## Models built.  Now test and output data to files for later.
##

baseResults <- testAndOutput2("BaseModels", models,pulsarTest)

##
## Now build the stacked ensembles
```

```
##

ensembleModels <- trainEnsembleModels(topModels,models,dataSet,pulsarTest,
                                      seeds,TRUE, myTuneLength,myMetric)

##
## Test, plot, and output data to files
##

ensembleTestData <- getEnsembleTestData(topModels,models,pulsarTest)

ensembleResults <- testEnsemble2()

print(paste("Finished at", Sys.time()))

##
## Now we have our array of models trained with all of the different
## sample sets, and the summary stats and plots are made.  Let's
## save the models, optionally, as they are huge...
##
#saveFile = 'PulsarEnsembleBaseModels.rds'
#saveRDS(models,saveFile)
##


saveFileBase <- date.time.append('savedModels/PulsarEnsembleModels', sep='_')
saveFileBaseModels <- paste(saveFileBase,'BaseModels',sep="_")
saveBaseModelsFile <- paste(saveFileBase,".rds",sep="")
saveRDS(models,saveBaseModelsFile)
saveFileEnsembleModels <- paste(saveFileBase,'EnsembleModels',sep="_")
saveEnsembleModelsModelsFile <- paste(saveFileEnsembleModels,".rds",sep="")
saveRDS(ensembleModels,saveEnsembleModelsModelsFile)

saveFileEnsembleModels <- paste(saveFileBase,'EnsembleModels',sep="_")
saveEnsembleModelsModelsFile <- paste(saveFileEnsembleModels,".rds",sep="")
saveRDS(ensembleModels,saveEnsembleModelsModelsFile)
```

Script to train, test, and evaluate all of the models

. The platform and all the versions of the modules used in this research program are given in this appendix.

```
R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.11.6 (El Capitan)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] grid      parallel  stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] nnet_7.3-12       DMwR_0.4.1        ROSE_0.0-3        gmodels_2.16.2
 [5] kernlab_0.9-25    caret_6.0-71      ggplot2_2.1.0     lattice_0.20-33
 [9] doParallel_1.0.10 iterators_1.0.8   foreach_1.4.3

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.7         compiler_3.3.1      nloptr_1.0.4        plyr_1.8.4
 [5] bitops_1.0-6        class_7.3-14        tools_3.3.1         xts_0.9-7
 [9] rpart_4.1-10        lme4_1.1-12         nlme_3.1-128        gtable_0.2.0
[13] mgcv_1.8-16         Matrix_1.2-6        SparseM_1.72        stringr_1.1.0
[17] pROC_1.8            caTools_1.17.1      MatrixModels_0.4-1 gtools_3.5.0
[21] stats4_3.3.1        gdata_2.17.0        minqa_1.2.4         ROCR_1.0-7
[25] TTR_0.23-1          reshape2_1.4.2      car_2.1-3           magrittr_1.5
[29] gplots_3.0.1        scales_0.4.0        codetools_0.2-14    MASS_7.3-45
[33] splines_3.3.1       quantmod_0.4-7      abind_1.4-5         pbkrtest_0.4-6
[37] colorspace_1.2-7    quantreg_5.29       KernSmooth_2.23-15 stringi_1.1.2
[41] munsell_0.4.3       zoo_1.7-13
```

Appendix B

Neural Network Plots, Raw Results and scripts

Plots

This section contains the larger versions of the plots included in chapter 4. An additional dot-plot is also included.

Figure B.1: Neural network model variable dot plot

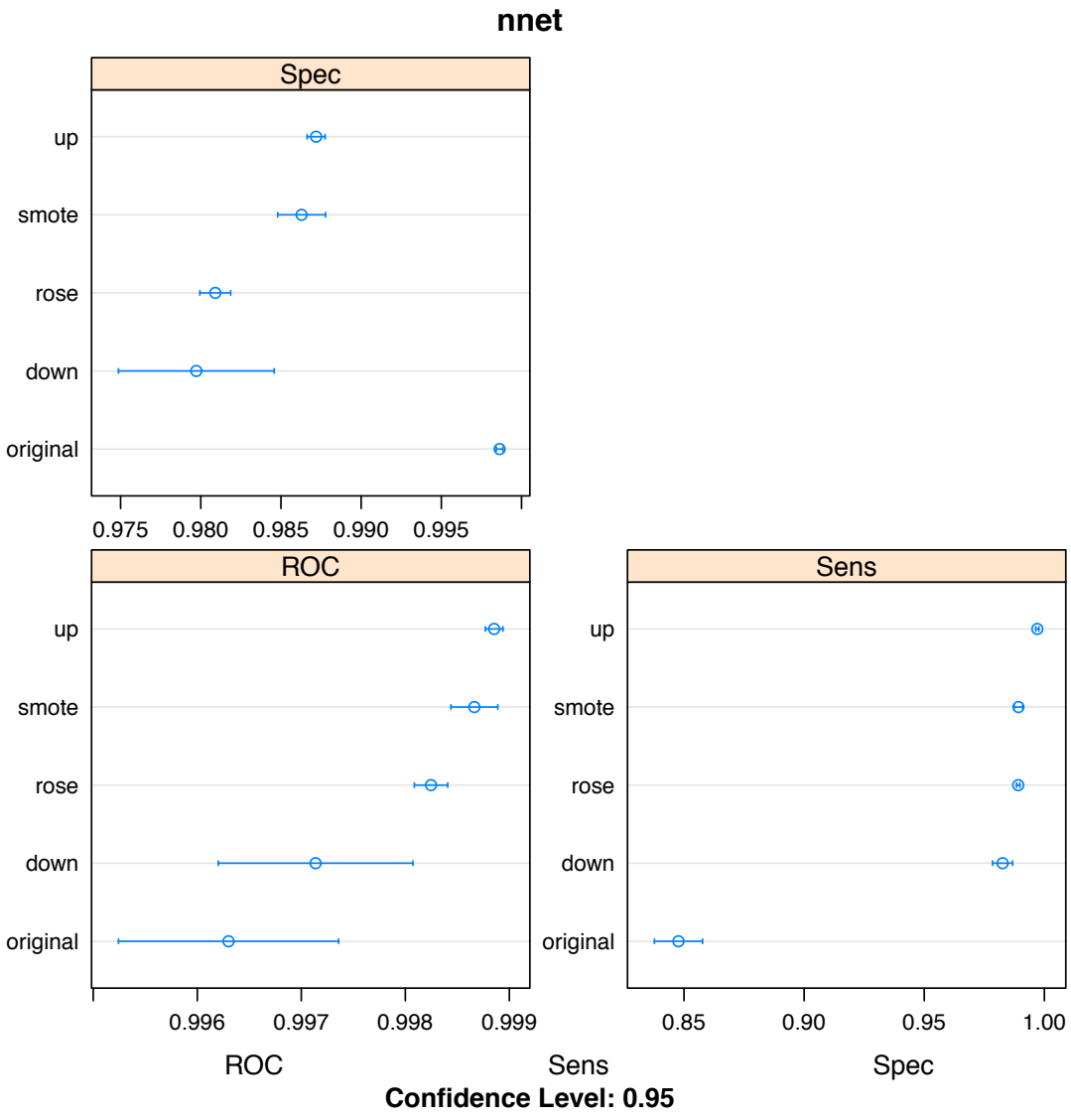


Figure B.2: Neural network model variable box and whiskers plot

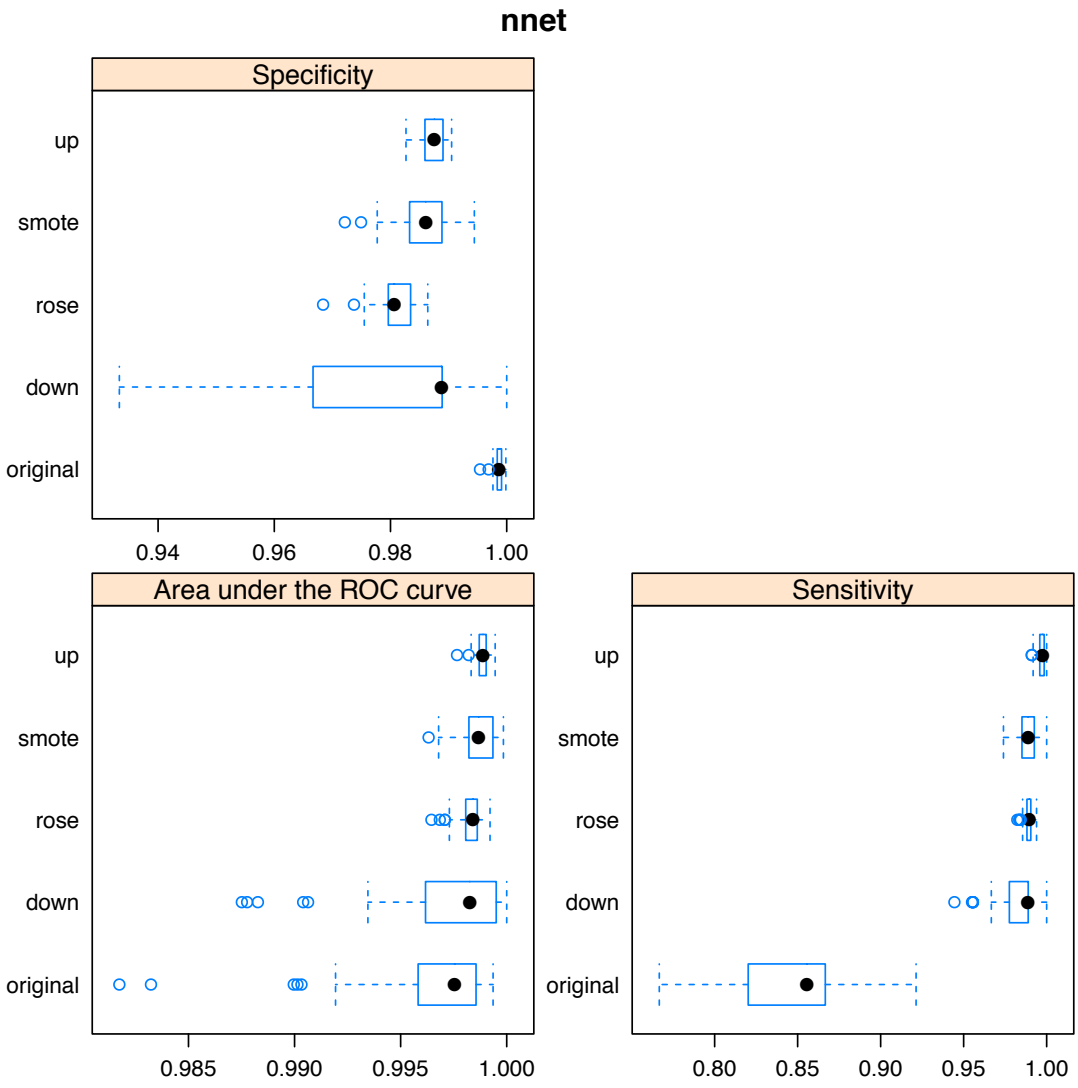


Figure B.3: Neural network model variable importance plot for four sampling methods

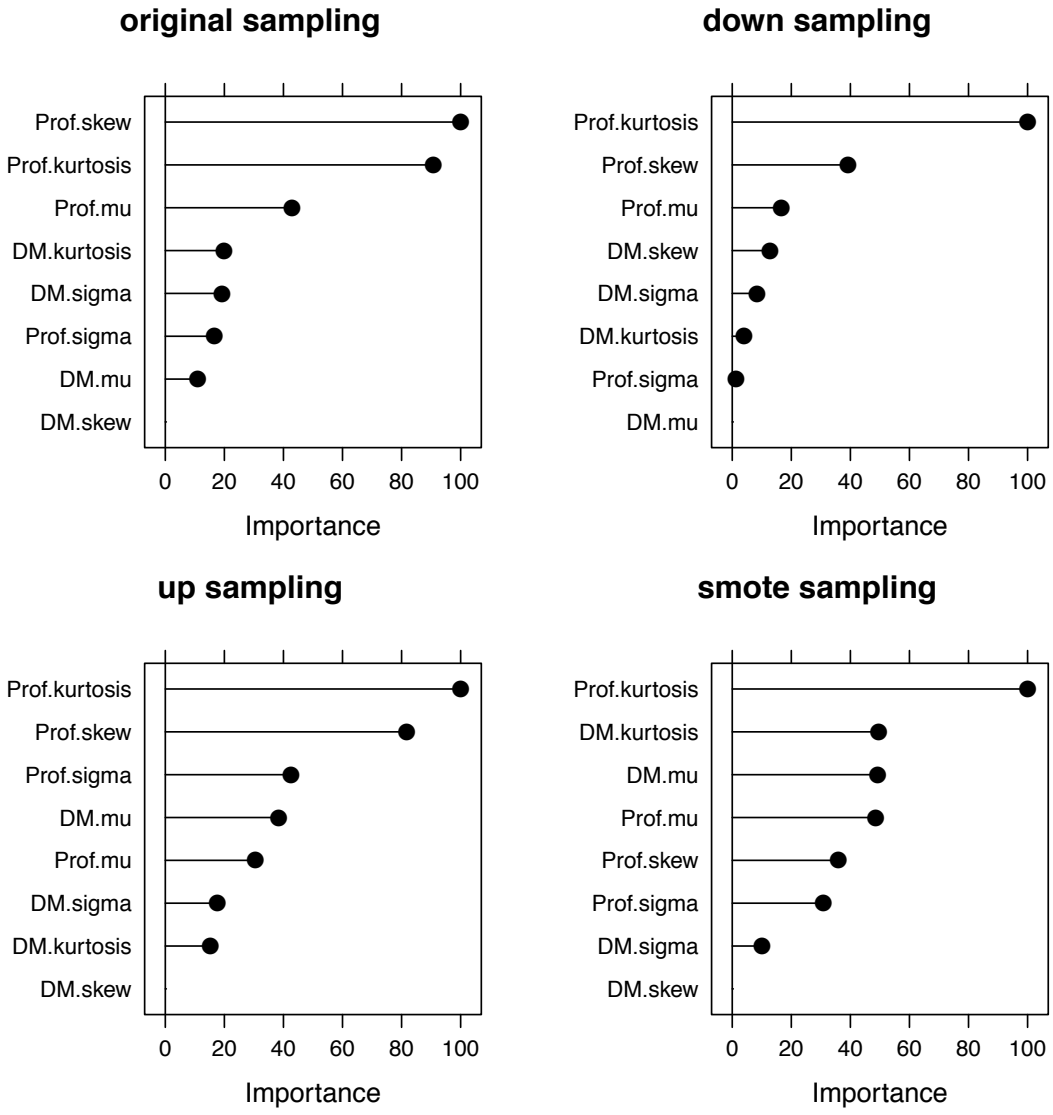
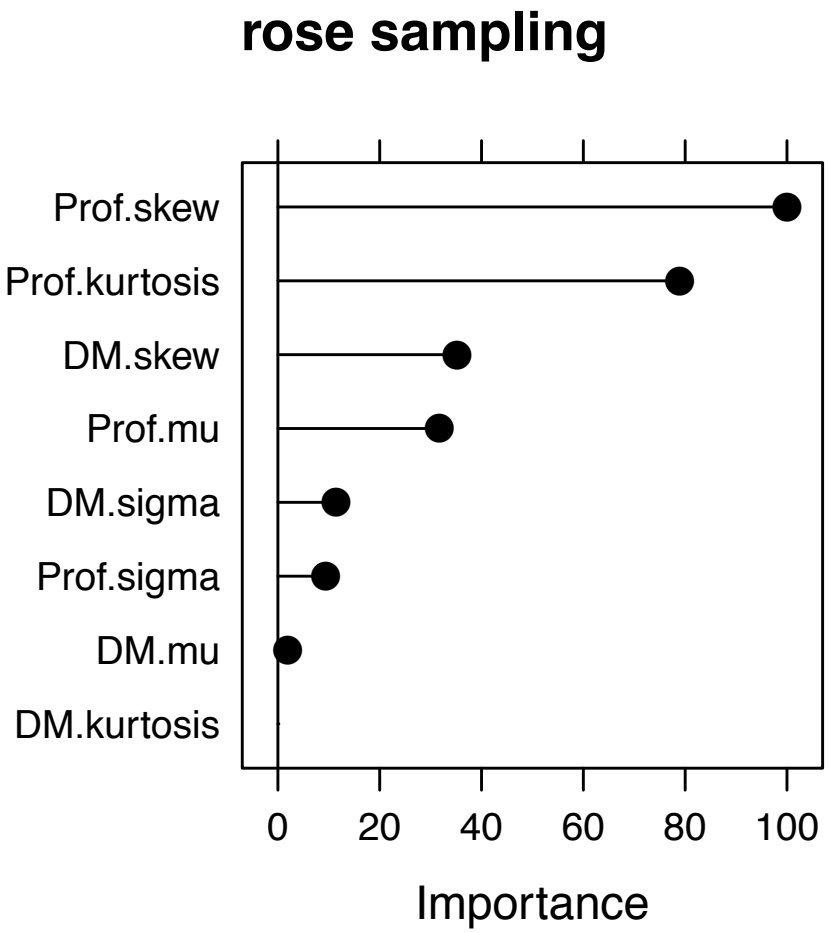


Figure B.4: Neural network model variable importance plot for ROSE sampling



Data

This section contains the raw ascii data output from the training and evaluation process for the neural network model.

```
Summary Text Output for the nnet model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       249        29
  nonpulsar     50     22470

               Accuracy : 0.9965
                 95% CI : (0.9957, 0.9973)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8613
 Mcnemar's Test P-Value : 0.02444

            Sensitivity : 0.83278
            Specificity : 0.99871
         Pos Pred Value : 0.89568
         Neg Pred Value : 0.99778
             Prevalence : 0.01312
         Detection Rate : 0.01092
   Detection Prevalence : 0.01219
      Balanced Accuracy : 0.91574

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       296       379
  nonpulsar      3     22120

               Accuracy : 0.9832
                 95% CI : (0.9815, 0.9849)
    No Information Rate : 0.9869
```

```
    P-Value [Acc > NIR] : 1

                  Kappa : 0.6005
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98997
            Specificity : 0.98315
         Pos Pred Value : 0.43852
         Neg Pred Value : 0.99986
             Prevalence : 0.01312
         Detection Rate : 0.01298
   Detection Prevalence : 0.02961
      Balanced Accuracy : 0.98656

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       293       324
  nonpulsar      6     22175

               Accuracy : 0.9855
                 95% CI : (0.9839, 0.987)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.9651

                  Kappa : 0.6333
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97993
            Specificity : 0.98560
         Pos Pred Value : 0.47488
         Neg Pred Value : 0.99973
             Prevalence : 0.01312
         Detection Rate : 0.01285
   Detection Prevalence : 0.02706
      Balanced Accuracy : 0.98277

       'Positive' Class : pulsar

Confusion Matrix and Statistics
```

```
          Reference
Prediction  pulsar nonpulsar
  pulsar       294       314
  nonpulsar      5     22185

               Accuracy : 0.986
                 95% CI : (0.9844, 0.9875)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.8829

                  Kappa : 0.642
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98328
            Specificity : 0.98604
         Pos Pred Value : 0.48355
         Neg Pred Value : 0.99977
             Prevalence : 0.01312
         Detection Rate : 0.01290
   Detection Prevalence : 0.02667
      Balanced Accuracy : 0.98466

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       267       112
  nonpulsar     32     22387

               Accuracy : 0.9937
                 95% CI : (0.9926, 0.9947)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7845
 Mcnemar's Test P-Value : 4.6e-11

            Sensitivity : 0.89298
            Specificity : 0.99502
         Pos Pred Value : 0.70449
         Neg Pred Value : 0.99857
             Prevalence : 0.01312
```

```
         Detection Rate : 0.01171
   Detection Prevalence : 0.01662
      Balanced Accuracy : 0.94400

       'Positive' Class : pulsar

Neural Network

68394 samples
    8 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9849507  0.7325144  0.9972621
   1    0.0001000000  0.9887778  0.6772260  0.9976532
   1    0.0005623413  0.9893283  0.7591436  0.9976236
   1    0.0031622777  0.9944330  0.6483396  0.9984947
   1    0.0177827941  0.9946464  0.6571735  0.9986518
   1    0.1000000000  0.9944034  0.6452434  0.9986933
   3    0.0000000000  0.9963658  0.8359051  0.9986162
   3    0.0001000000  0.9964783  0.8385393  0.9984651
   3    0.0005623413  0.9972422  0.8372584  0.9985036
   3    0.0031622777  0.9967240  0.8052534  0.9987140
   3    0.0177827941  0.9975215  0.8307091  0.9987288
   3    0.1000000000  0.9976731  0.8372409  0.9987199
   5    0.0000000000  0.9953990  0.8298752  0.9987525
   5    0.0001000000  0.9972356  0.8336554  0.9988177
   5    0.0005623413  0.9974458  0.8345518  0.9988088
   5    0.0031622777  0.9962904  0.8194981  0.9988385
   5    0.0177827941  0.9977834  0.8394432  0.9988444
   5    0.1000000000  0.9978517  0.8329813  0.9988681
   7    0.0000000000  0.9968150  0.8325618  0.9987496
   7    0.0001000000  0.9959820  0.8425618  0.9987081
   7    0.0005623413  0.9967516  0.8332035  0.9987614
   7    0.0031622777  0.9975900  0.8381248  0.9987585
   7    0.0177827941  0.9975961  0.8376579  0.9987970
   7    0.1000000000  0.9977670  0.8354207  0.9989451
   9    0.0000000000  0.9958720  0.8356155  0.9986518
   9    0.0001000000  0.9951516  0.8403171  0.9987081
```

```
   9    0.0005623413  0.9859218  0.8217203  0.9987377
   9    0.0031622777  0.9970459  0.8376629  0.9987940
   9    0.0177827941  0.9976048  0.8378777  0.9988296
   9    0.1000000000  0.9976213  0.8396554  0.9989007
  11    0.0000000000  0.9955295  0.8401373  0.9985333
  11    0.0001000000  0.9960746  0.8400924  0.9985836
  11    0.0005623413  0.9864551  0.8241199  0.9986251
  11    0.0031622777  0.9864928  0.8211885  0.9987555
  11    0.0177827941  0.9963005  0.8477004  0.9986340
  11    0.1000000000  0.9977285  0.8423271  0.9988118

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 11 and decay = 0.01778279.
Neural Network

1794 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9953244  0.9785868  0.9795006
   1    0.0001000000  0.9971373  0.9826267  0.9797228
   1    0.0005623413  0.9976868  0.9824045  0.9797228
   1    0.0031622777  0.9976842  0.9821823  0.9810587
   1    0.0177827941  0.9976866  0.9812809  0.9819501
   1    0.1000000000  0.9977552  0.9821848  0.9817278
   3    0.0000000000  0.9935004  0.9750137  0.9737054
   3    0.0001000000  0.9948891  0.9768240  0.9754856
   3    0.0005623413  0.9956403  0.9768390  0.9757054
   3    0.0031622777  0.9967554  0.9790612  0.9790537
   3    0.0177827941  0.9975523  0.9783895  0.9792684
   3    0.1000000000  0.9980225  0.9783895  0.9817203
   5    0.0000000000  0.9878821  0.9719276  0.9761323
   5    0.0001000000  0.9919250  0.9732584  0.9725793
   5    0.0005623413  0.9946511  0.9743720  0.9748040
   5    0.0031622777  0.9968259  0.9741598  0.9763695
   5    0.0177827941  0.9973622  0.9770587  0.9772634
   5    0.1000000000  0.9979873  0.9772784  0.9819451
   7    0.0000000000  0.9848043  0.9710337  0.9670137
```

```
  7    0.0001000000  0.9906205  0.9723770  0.9681273
  7    0.0005623413  0.9937524  0.9712459  0.9739076
  7    0.0031622777  0.9955286  0.9737054  0.9732484
  7    0.0177827941  0.9963513  0.9730462  0.9797104
  7    0.1000000000  0.9981014  0.9783920  0.9821648
  9    0.0000000000  0.9835047  0.9737104  0.9719276
  9    0.0001000000  0.9893416  0.9701398  0.9687890
  9    0.0005623413  0.9935607  0.9721548  0.9708015
  9    0.0031622777  0.9956499  0.9748240  0.9719176
  9    0.0177827941  0.9972196  0.9750337  0.9737079
  9    0.1000000000  0.9980643  0.9781698  0.9808290
 11    0.0000000000  0.9784493  0.9725893  0.9650112
 11    0.0001000000  0.9904679  0.9703446  0.9725893
 11    0.0005623413  0.9934579  0.9725918  0.9701448
 11    0.0031622777  0.9952358  0.9728140  0.9690212
 11    0.0177827941  0.9969409  0.9741448  0.9750462
 11    0.1000000000  0.9980763  0.9779426  0.9814956

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 1 and decay = 1e-04.
Neural Network

134994 samples
     8 predictor
     2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494,
121496, 121495, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9973286  0.9838986  0.9822718
   1    0.0001000000  0.9959512  0.9786123  0.9827281
   1    0.0005623413  0.9963462  0.9767401  0.9830304
   1    0.0031622777  0.9968458  0.9841090  0.9813059
   1    0.0177827941  0.9975218  0.9844527  0.9823074
   1    0.1000000000  0.9974284  0.9844527  0.9822096
   3    0.0000000000  0.9973618  0.9858752  0.9824378
   3    0.0001000000  0.9969780  0.9852439  0.9831163
   3    0.0005623413  0.9974761  0.9852381  0.9824556
   3    0.0031622777  0.9974950  0.9834688  0.9832082
   3    0.0177827941  0.9976143  0.9857711  0.9825770
```

```
   3    0.1000000000  0.9973107  0.9829297  0.9829533
   5    0.0000000000  0.9985113  0.9914545  0.9840082
   5    0.0001000000  0.9985327  0.9925182  0.9837060
   5    0.0005623413  0.9985773  0.9929152  0.9839282
   5    0.0031622777  0.9984207  0.9907434  0.9842778
   5    0.0177827941  0.9986000  0.9912441  0.9837445
   5    0.1000000000  0.9985744  0.9906870  0.9835460
   7    0.0000000000  0.9986799  0.9947198  0.9849860
   7    0.0001000000  0.9987279  0.9943849  0.9850838
   7    0.0005623413  0.9987026  0.9941686  0.9853534
   7    0.0031622777  0.9986479  0.9937212  0.9851934
   7    0.0177827941  0.9987277  0.9937716  0.9848230
   7    0.1000000000  0.9987659  0.9931049  0.9848349
   9    0.0000000000  0.9988033  0.9958872  0.9867549
   9    0.0001000000  0.9987738  0.9961124  0.9866068
   9    0.0005623413  0.9987480  0.9960176  0.9863135
   9    0.0031622777  0.9987921  0.9956798  0.9863638
   9    0.0177827941  0.9987828  0.9956799  0.9859757
   9    0.1000000000  0.9988357  0.9955968  0.9859461
  11    0.0000000000  0.9988539  0.9970784  0.9871935
  11    0.0001000000  0.9987660  0.9968176  0.9871313
  11    0.0005623413  0.9988218  0.9966991  0.9871461
  11    0.0031622777  0.9988057  0.9970665  0.9869001
  11    0.0177827941  0.9987803  0.9967050  0.9867105
  11    0.1000000000  0.9988395  0.9962724  0.9862809

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 11 and decay = 0.
Neural Network

6279 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9980146  0.9842454  0.9862318
   1    0.0001000000  0.9979519  0.9839480  0.9855631
   1    0.0005623413  0.9977004  0.9819405  0.9863432
   1    0.0031622777  0.9978955  0.9801561  0.9866778
```

```
  1    0.0177827941  0.9978620  0.9808996  0.9866775
  1    0.1000000000  0.9982412  0.9822376  0.9868446
  3    0.0000000000  0.9980971  0.9843186  0.9858420
  3    0.0001000000  0.9975861  0.9829062  0.9850609
  3    0.0005623413  0.9985908  0.9853603  0.9856737
  3    0.0031622777  0.9980366  0.9821710  0.9869001
  3    0.0177827941  0.9986738  0.9863258  0.9860639
  3    0.1000000000  0.9987627  0.9845419  0.9853957
  5    0.0000000000  0.9975236  0.9859548  0.9831632
  5    0.0001000000  0.9967023  0.9797874  0.9853949
  5    0.0005623413  0.9982421  0.9862514  0.9861765
  5    0.0031622777  0.9985010  0.9849131  0.9869014
  5    0.0177827941  0.9986325  0.9848393  0.9865654
  5    0.1000000000  0.9989006  0.9849886  0.9873466
  7    0.0000000000  0.9972894  0.9856566  0.9841691
  7    0.0001000000  0.9979450  0.9863274  0.9838359
  7    0.0005623413  0.9982315  0.9877367  0.9852288
  7    0.0031622777  0.9981785  0.9869184  0.9858402
  7    0.0177827941  0.9986145  0.9865472  0.9861764
  7    0.1000000000  0.9988978  0.9858053  0.9875137
  9    0.0000000000  0.9966438  0.9855082  0.9848371
  9    0.0001000000  0.9974503  0.9871436  0.9838910
  9    0.0005623413  0.9977871  0.9853589  0.9852276
  9    0.0031622777  0.9982688  0.9887038  0.9862322
  9    0.0177827941  0.9986754  0.9880347  0.9866221
  9    0.1000000000  0.9989057  0.9851364  0.9871796
 11    0.0000000000  0.9955767  0.9855803  0.9836114
 11    0.0001000000  0.9973202  0.9860273  0.9843358
 11    0.0005623413  0.9978515  0.9875892  0.9856737
 11    0.0031622777  0.9981099  0.9876621  0.9856193
 11    0.0177827941  0.9986639  0.9892240  0.9862875
 11    0.1000000000  0.9988778  0.9870684  0.9868449

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 11 and decay = 0.01778279.
Neural Network

68394 samples
    8 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
```

```
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9904749  0.9748767  0.9495157
   1    0.0001000000  0.9902034  0.9755024  0.9485529
   1    0.0005623413  0.9904813  0.9747552  0.9511223
   1    0.0031622777  0.9894712  0.9751143  0.9485233
   1    0.0177827941  0.9907480  0.9747262  0.9511636
   1    0.1000000000  0.9907447  0.9747262  0.9512050
   3    0.0000000000  0.9955556  0.9840272  0.9669817
   3    0.0001000000  0.9959192  0.9838535  0.9696043
   3    0.0005623413  0.9951544  0.9834192  0.9664737
   3    0.0031622777  0.9958688  0.9837551  0.9704548
   3    0.0177827941  0.9962386  0.9842879  0.9711223
   3    0.1000000000  0.9963469  0.9845427  0.9719728
   5    0.0000000000  0.9971229  0.9857184  0.9751093
   5    0.0001000000  0.9971332  0.9857763  0.9751979
   5    0.0005623413  0.9971671  0.9856952  0.9757177
   5    0.0031622777  0.9966583  0.9853882  0.9727525
   5    0.0177827941  0.9972022  0.9860833  0.9754932
   5    0.1000000000  0.9969997  0.9867376  0.9740756
   7    0.0000000000  0.9975053  0.9867724  0.9777614
   7    0.0001000000  0.9975968  0.9870041  0.9776669
   7    0.0005623413  0.9976199  0.9869520  0.9775428
   7    0.0031622777  0.9976210  0.9866450  0.9776669
   7    0.0177827941  0.9977525  0.9875659  0.9780154
   7    0.1000000000  0.9978220  0.9882087  0.9780331
   9    0.0000000000  0.9980015  0.9878670  0.9799232
   9    0.0001000000  0.9980357  0.9883419  0.9795629
   9    0.0005623413  0.9980801  0.9884403  0.9802304
   9    0.0031622777  0.9980819  0.9883882  0.9804962
   9    0.0177827941  0.9980624  0.9884346  0.9800886
   9    0.1000000000  0.9981362  0.9889327  0.9805316
  11    0.0000000000  0.9981339  0.9883534  0.9804903
  11    0.0001000000  0.9983520  0.9889326  0.9812168
  11    0.0005623413  0.9981649  0.9885156  0.9808742
  11    0.0031622777  0.9982720  0.9890369  0.9813290
  11    0.0177827941  0.9982396  0.9889326  0.9811991
  11    0.1000000000  0.9982471  0.9891295  0.9809037

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 11 and decay = 0.1.

Call:
```

```
summary.diff.resamples(object = results$modelDiff[[name]])

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down       up         smote      rose
original           -0.0008368 -0.0025534 -0.0023634 -0.0019466
down     1.0000000            -0.0017165 -0.0015266 -0.0011098
up       0.0001536 0.0054660              0.0001900  0.0006068
smote    0.0007936 0.0208424  1.0000000              0.0004168
rose     0.0053153 0.1651019  1.544e-08  0.0246424

Sens
         original  down       up         smote      rose
original           -1.349e-01 -1.494e-01 -1.415e-01 -1.414e-01
down     < 2.2e-16            -1.445e-02 -6.597e-03 -6.503e-03
up       < 2.2e-16 8.476e-08              7.854e-03  7.949e-03
smote    < 2.2e-16 0.05108    1.135e-10              9.448e-05
rose     < 2.2e-16 0.02710    < 2.2e-16  1.00000

Spec
         original  down      up        smote     rose
original            0.018911  0.011440  0.012347  0.017730
down     2.489e-09           -0.007471 -0.006565 -0.001181
up       < 2.2e-16 0.03355              0.000906  0.006290
smote    < 2.2e-16 0.15026   1.00000              0.005384
rose     < 2.2e-16 1.00000   6.095e-14 1.157e-07

$original
nnet variable importance

              Overall
Prof.skew      100.00
Prof.kurtosis   90.79
Prof.mu         42.91
DM.kurtosis     19.92
DM.sigma        19.26
Prof.sigma      16.57
DM.mu           11.02
DM.skew          0.00

$down
```

```
nnet variable importance

             Overall
Prof.kurtosis 100.000
Prof.skew      39.273
Prof.mu        16.632
DM.skew        12.818
DM.sigma        8.411
DM.kurtosis     4.016
Prof.sigma      1.282
DM.mu           0.000

$up
nnet variable importance

             Overall
Prof.kurtosis  100.00
Prof.skew       81.77
Prof.sigma      42.57
DM.mu           38.47
Prof.mu         30.46
DM.sigma        17.70
DM.kurtosis     15.25
DM.skew          0.00

$smote
nnet variable importance

             Overall
Prof.kurtosis  100.00
DM.kurtosis     49.66
DM.mu           49.27
Prof.mu         48.60
Prof.skew       36.00
Prof.sigma      30.92
DM.sigma        10.05
DM.skew          0.00

$rose
nnet variable importance

             Overall
Prof.skew     100.000
Prof.kurtosis  78.917
```

```
DM.skew        35.220
Prof.mu        31.730
DM.sigma       11.414
Prof.sigma      9.408
DM.mu           1.930
DM.kurtosis     0.000


Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50

ROC
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9817  0.9959 0.9975 0.9963  0.9985 0.9994    0
down     0.9875  0.9962 0.9983 0.9971  0.9995 1.0000    0
up       0.9977  0.9987 0.9989 0.9989  0.9990 0.9995    0
smote    0.9963  0.9982 0.9987 0.9987  0.9993 0.9998    0
rose     0.9965  0.9981 0.9984 0.9982  0.9986 0.9992    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.7667  0.8207 0.8556 0.8477  0.8667 0.9213    0
down     0.9444  0.9775 0.9888 0.9826  0.9889 1.0000    0
up       0.9910  0.9959 0.9976 0.9971  0.9985 1.0000    0
smote    0.9740  0.9851 0.9888 0.9892  0.9926 1.0000    0
rose     0.9823  0.9881 0.9896 0.9891  0.9904 0.9939    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9954  0.9984 0.9987 0.9986  0.9991 0.9999    0
down     0.9333  0.9667 0.9888 0.9797  0.9889 1.0000    0
up       0.9827  0.9860 0.9876 0.9872  0.9890 0.9905    0
smote    0.9721  0.9833 0.9861 0.9863  0.9889 0.9944    0
rose     0.9684  0.9796 0.9807 0.9809  0.9835 0.9864    0
```

Appendix C

Support Vector Machine Results

Plots

This section contains the plots for the support vector machine training and evaluation process. An extra dotplot is included.

Figure C.1: Support vector machine model variable dot plot

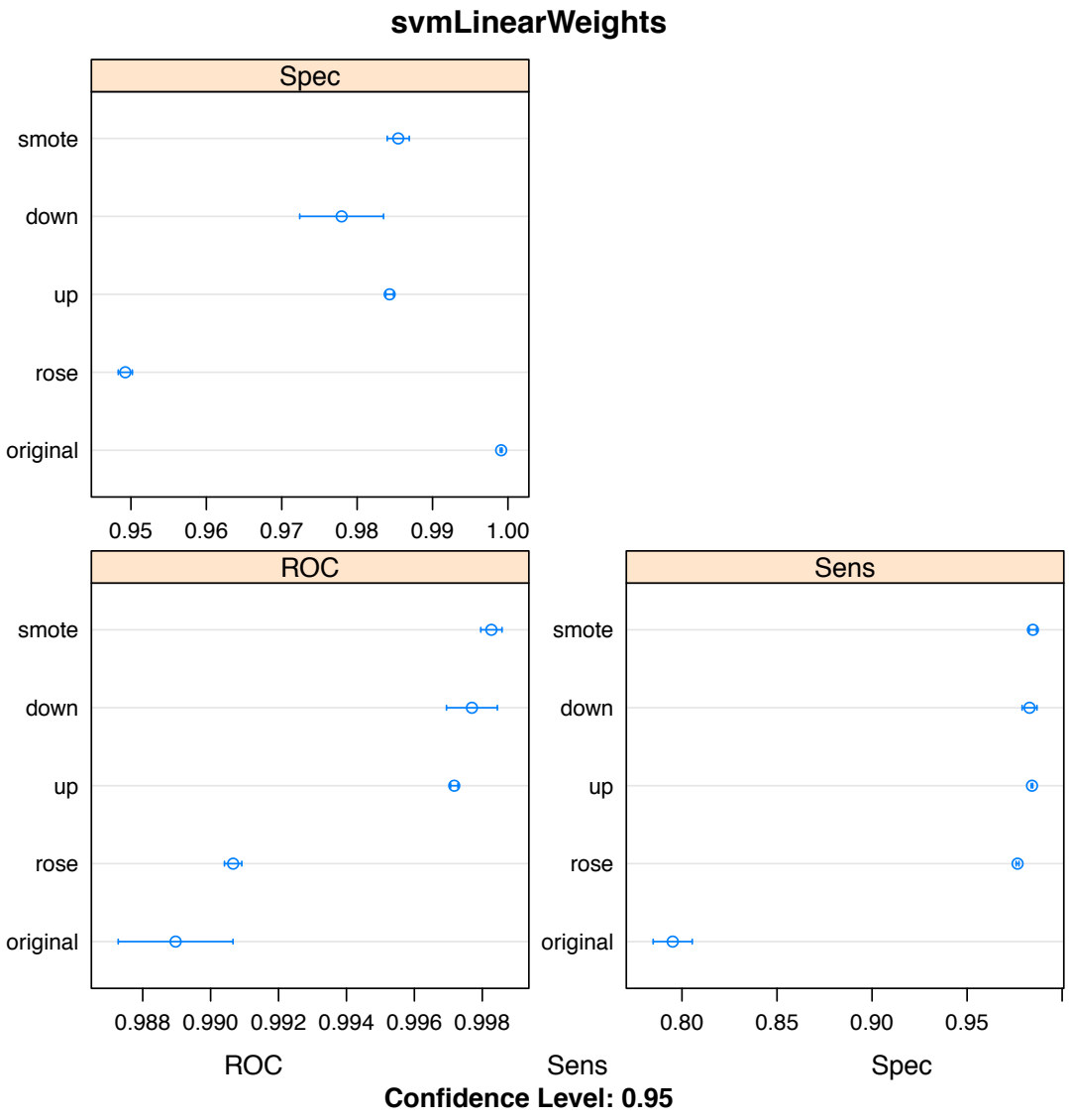


Figure C.2: Support vector machine model variable box and whiskers plot

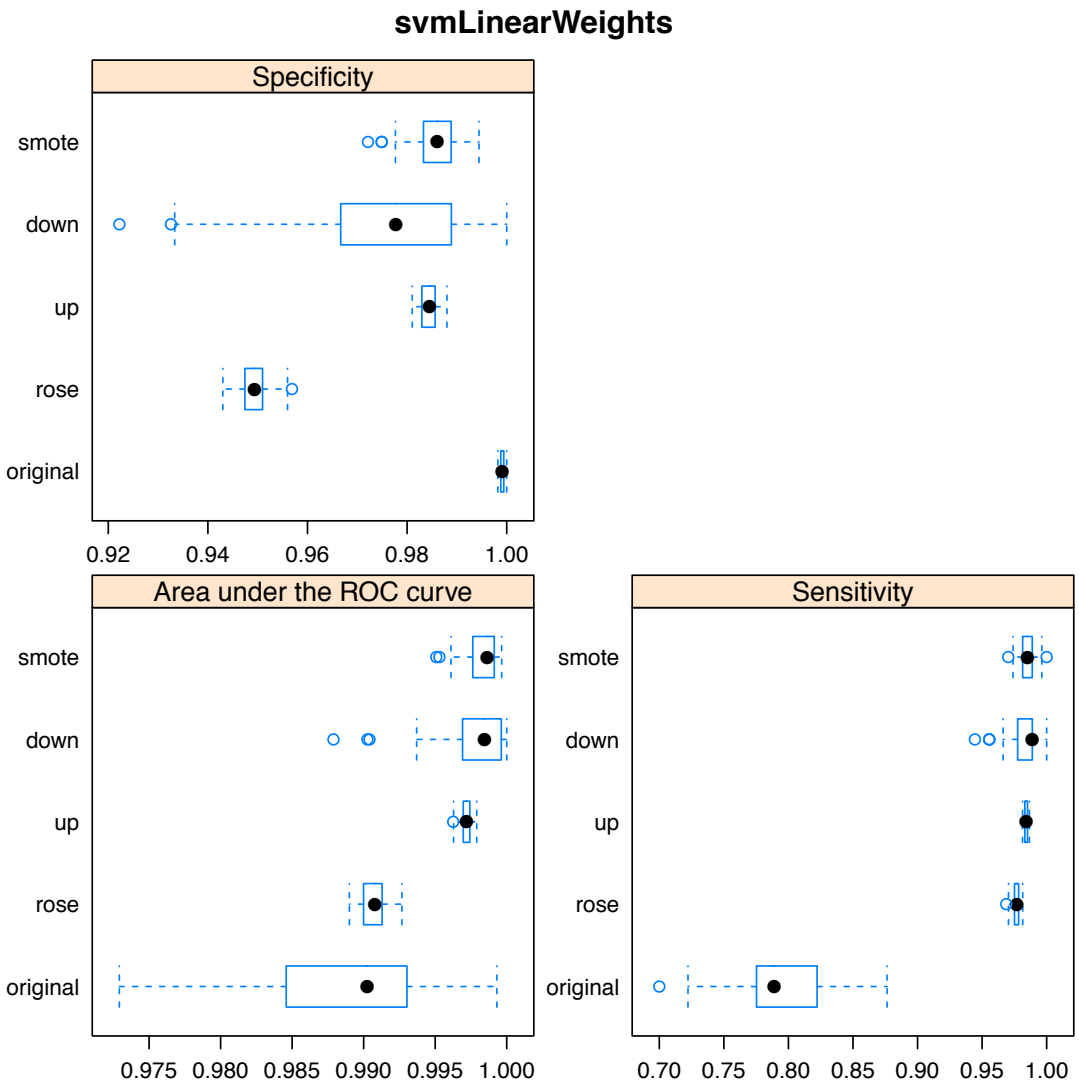


Figure C.3: Support vector machine model variable importance plots for four sample sets

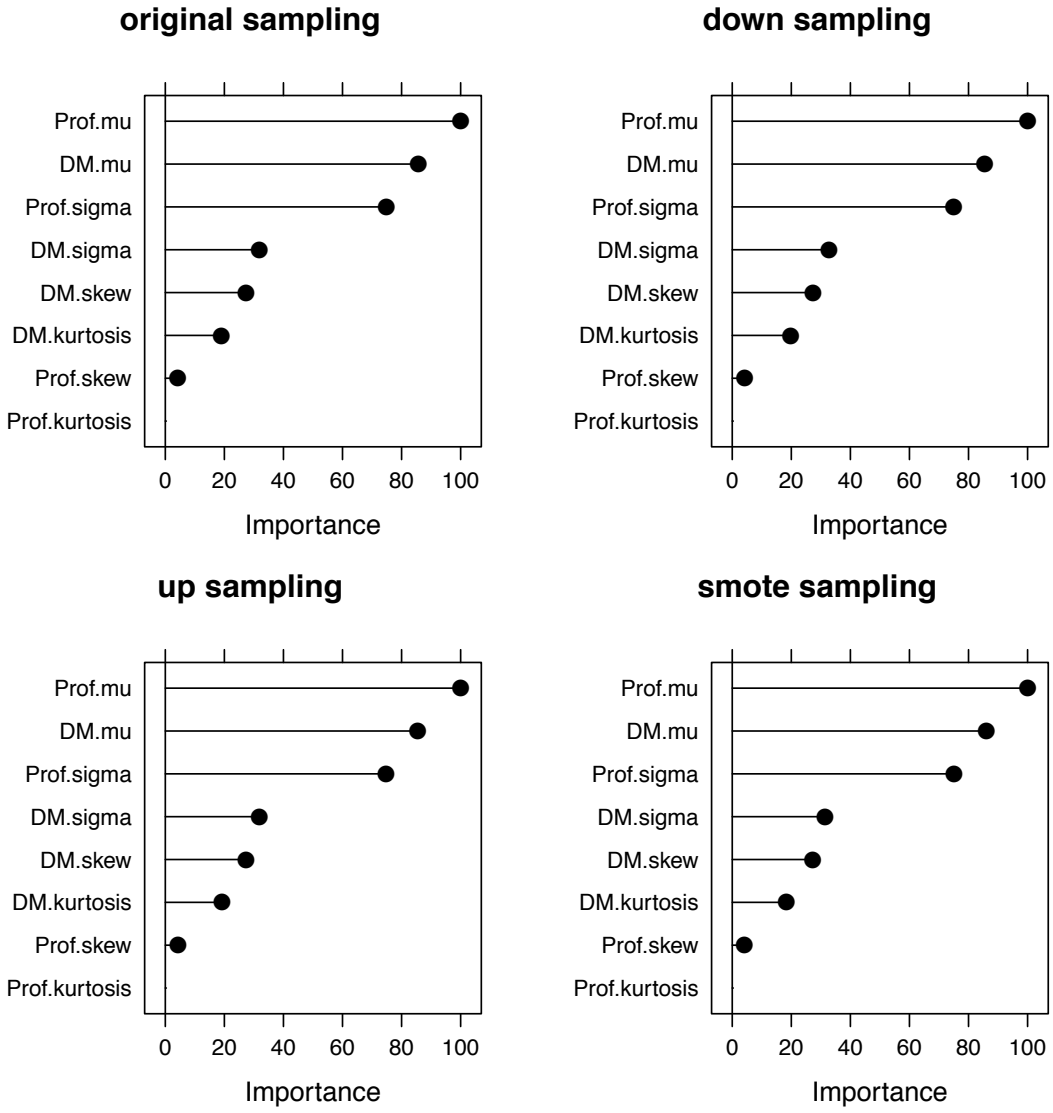
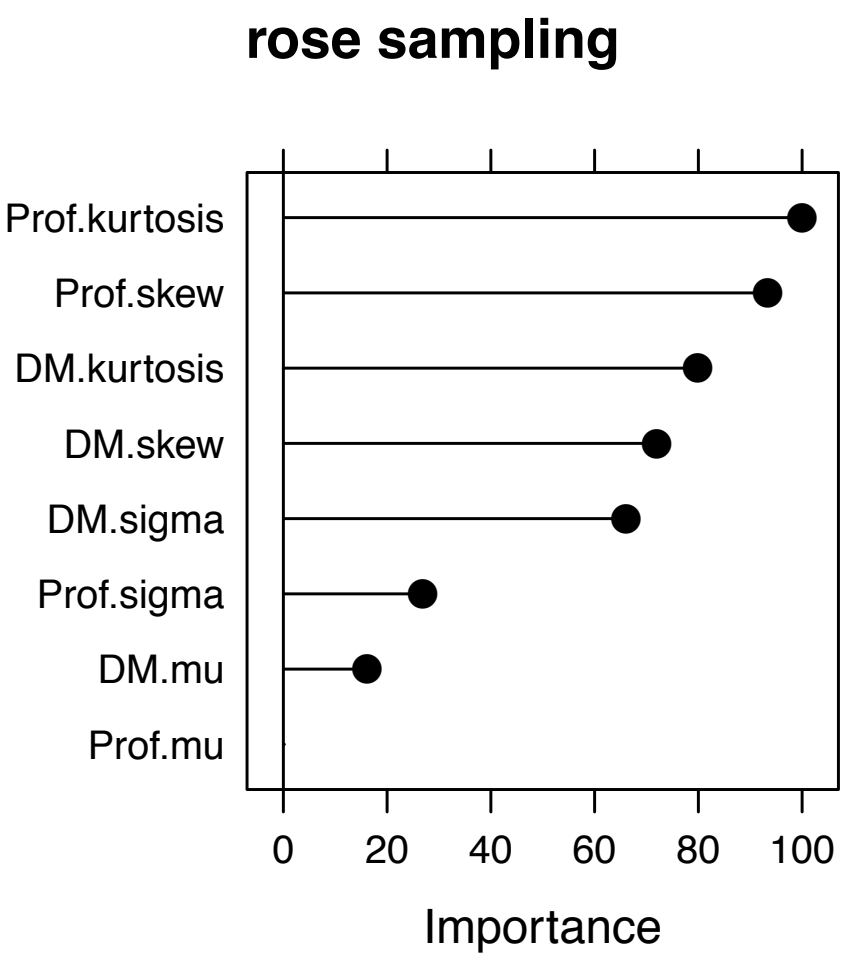


Figure C.4: Support vector machine model variable importance plots for ROSE sample sets



Data

This section contains the text data from the training and evaluation process.

```
Summary Text Output for the svmLinearWeights model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       229        27
  nonpulsar     70     22472

               Accuracy : 0.9957
                 95% CI : (0.9948, 0.9965)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8231
 Mcnemar's Test P-Value : 2.004e-05

            Sensitivity : 0.76589
            Specificity : 0.99880
         Pos Pred Value : 0.89453
         Neg Pred Value : 0.99689
             Prevalence : 0.01312
         Detection Rate : 0.01004
   Detection Prevalence : 0.01123
      Balanced Accuracy : 0.88234

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       294       353
  nonpulsar      5     22146

               Accuracy : 0.9843
                 95% CI : (0.9826, 0.9859)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.9996
```

```
                 Kappa : 0.6147
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98328
            Specificity : 0.98431
         Pos Pred Value : 0.45440
         Neg Pred Value : 0.99977
             Prevalence : 0.01312
         Detection Rate : 0.01290
   Detection Prevalence : 0.02838
      Balanced Accuracy : 0.98379

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       295       353
  nonpulsar      4     22146

               Accuracy : 0.9843
                 95% CI : (0.9826, 0.9859)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.9995

                 Kappa : 0.6161
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98662
            Specificity : 0.98431
         Pos Pred Value : 0.45525
         Neg Pred Value : 0.99982
             Prevalence : 0.01312
         Detection Rate : 0.01294
   Detection Prevalence : 0.02842
      Balanced Accuracy : 0.98547

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
```

```
  pulsar       293       299
  nonpulsar      6     22200

               Accuracy : 0.9866
                 95% CI : (0.985, 0.9881)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.6504

                  Kappa : 0.6516
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97993
            Specificity : 0.98671
         Pos Pred Value : 0.49493
         Neg Pred Value : 0.99973
             Prevalence : 0.01312
         Detection Rate : 0.01285
   Detection Prevalence : 0.02597
      Balanced Accuracy : 0.98332

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       291       345
  nonpulsar      8     22154

               Accuracy : 0.9845
                 95% CI : (0.9828, 0.9861)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.999

                  Kappa : 0.6156
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97324
            Specificity : 0.98467
         Pos Pred Value : 0.45755
         Neg Pred Value : 0.99964
             Prevalence : 0.01312
         Detection Rate : 0.01276
   Detection Prevalence : 0.02790
```

```
      Balanced Accuracy : 0.97896

       'Positive' Class : pulsar

Linear Support Vector Machines with Class Weights

68394 samples
    8 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results across tuning parameters:

  cost  weight  ROC        Sens       Spec
  0.25  1       0.9897180  0.7908889  0.9991051
  0.25  2       0.9856033  0.7440474  0.9994844
  0.25  3       0.9840866  0.7175131  0.9995911
  0.25  4       0.9829856  0.7021298  0.9996326
  0.25  5       0.9823095  0.6867366  0.9996592
  0.25  6       0.9825290  0.6675456  0.9996711
  0.50  1       0.9891847  0.7937878  0.9991140
  0.50  2       0.9853890  0.7467266  0.9994637
  0.50  3       0.9839190  0.7172909  0.9995911
  0.50  4       0.9826644  0.7045868  0.9996266
  0.50  5       0.9822453  0.6883021  0.9996474
  0.50  6       0.9825875  0.6700000  0.9996711
  1.00  1       0.9889691  0.7951211  0.9991022
  1.00  2       0.9852546  0.7478377  0.9994518
  1.00  3       0.9837811  0.7172859  0.9995941
  1.00  4       0.9825299  0.7048090  0.9996237
  1.00  5       0.9821851  0.6898602  0.9996504
  1.00  6       0.9825105  0.6706667  0.9996681
  2.00  1       0.9888567  0.7948964  0.9990933
  2.00  2       0.9851758  0.7491810  0.9994429
  2.00  3       0.9837164  0.7170637  0.9995941
  2.00  4       0.9824935  0.7050312  0.9996266
  2.00  5       0.9821422  0.6898602  0.9996474
  2.00  6       0.9825446  0.6715581  0.9996711
  4.00  1       0.9888267  0.7951161  0.9990992
  4.00  2       0.9851493  0.7498502  0.9994459
  4.00  3       0.9836973  0.7175081  0.9995911
  4.00  4       0.9824689  0.7054806  0.9996237
```

```
  4.00  5       0.9821228  0.6900824  0.9996474
  4.00  6       0.9825624  0.6713358  0.9996681
  8.00  1       0.9888036  0.7948964  0.9990903
  8.00  2       0.9851255  0.7500724  0.9994489
  8.00  3       0.9836881  0.7179526  0.9995911
  8.00  4       0.9824321  0.7059251  0.9996207
  8.00  5       0.9821203  0.6905293  0.9996474
  8.00  6       0.9825977  0.6706692  0.9996652

Sens was used to select the optimal model using  the largest value.
The final values used for the model were cost = 1 and weight = 1.
Linear Support Vector Machines with Class Weights

1794 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results across tuning parameters:

  cost  weight  ROC        Sens       Spec
  0.25  1       0.9968272  0.9775081  0.9761648
  0.25  2       0.9970529  0.9527441  0.9906367
  0.25  3       0.9968987  0.9382497  0.9930911
  0.25  4       0.9969361  0.9288814  0.9930911
  0.25  5       0.9970273  0.9230811  0.9946492
  0.25  6       0.9972036  0.9188439  0.9957628
  0.50  1       0.9971289  0.9792859  0.9739301
  0.50  2       0.9972495  0.9554257  0.9897503
  0.50  3       0.9971966  0.9418202  0.9930911
  0.50  4       0.9972730  0.9340125  0.9946492
  0.50  5       0.9974396  0.9275556  0.9946492
  0.50  6       0.9976434  0.9250936  0.9964295
  1.00  1       0.9974221  0.9799551  0.9737079
  1.00  2       0.9974389  0.9592135  0.9890836
  1.00  3       0.9974343  0.9447191  0.9939825
  1.00  4       0.9977496  0.9360250  0.9946492
  1.00  5       0.9978693  0.9311161  0.9957628
  1.00  6       0.9978483  0.9288889  0.9959850
  2.00  1       0.9975898  0.9808414  0.9743795
  2.00  2       0.9975525  0.9614382  0.9908664
  2.00  3       0.9975820  0.9465194  0.9937578
```

```
  2.00  4       0.9978399  0.9387116  0.9950936
  2.00  5       0.9979366  0.9355830  0.9953159
  2.00  6       0.9979187  0.9326866  0.9953159
  4.00  1       0.9976483  0.9821823  0.9763820
  4.00  2       0.9975185  0.9654557  0.9904145
  4.00  3       0.9977063  0.9509713  0.9939800
  4.00  4       0.9978242  0.9431710  0.9948714
  4.00  5       0.9978563  0.9378152  0.9948714
  4.00  6       0.9979062  0.9353608  0.9950936
  8.00  1       0.9976947  0.9828539  0.9779426
  8.00  2       0.9975748  0.9676854  0.9899625
  8.00  3       0.9977240  0.9543121  0.9937553
  8.00  4       0.9978135  0.9469613  0.9944245
  8.00  5       0.9978035  0.9404919  0.9944270
  8.00  6       0.9978336  0.9380400  0.9948714

Sens was used to select the optimal model using  the largest value.
The final values used for the model were cost = 8 and weight = 1.
Linear Support Vector Machines with Class Weights

134994 samples
     8 predictor
     2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494, 121496,
121495, ...
Resampling results across tuning parameters:

  cost  weight  ROC        Sens       Spec
  0.25  1       0.9972061  0.9828466  0.9846838
  0.25  2       0.9972163  0.9719247  0.9889684
  0.25  3       0.9972834  0.9645317  0.9908944
  0.25  4       0.9973258  0.9568158  0.9920648
  0.25  5       0.9973204  0.9503770  0.9925774
  0.25  6       0.9972798  0.9480302  0.9927552
  0.50  1       0.9971926  0.9841119  0.9843193
  0.50  2       0.9972007  0.9719484  0.9890069
  0.50  3       0.9972727  0.9645317  0.9907788
  0.50  4       0.9973201  0.9555654  0.9919819
  0.50  5       0.9973211  0.9506200  0.9925360
  0.50  6       0.9972796  0.9492570  0.9927374
  1.00  1       0.9971744  0.9841475  0.9843134
```

```
  1.00  2       0.9971959  0.9720076  0.9890128
  1.00  3       0.9972701  0.9646591  0.9907462
  1.00  4       0.9973105  0.9562913  0.9918604
  1.00  5       0.9973210  0.9509459  0.9925448
  1.00  6       0.9972808  0.9495088  0.9927434
  2.00  1       0.9971658  0.9841475  0.9843549
  2.00  2       0.9972007  0.9719069  0.9890188
  2.00  3       0.9972707  0.9648369  0.9906988
  2.00  4       0.9973057  0.9565106  0.9918278
  2.00  5       0.9973209  0.9511504  0.9925478
  2.00  6       0.9972821  0.9496422  0.9927523
  4.00  1       0.9971615  0.9841475  0.9843697
  4.00  2       0.9972051  0.9718536  0.9890039
  4.00  3       0.9972710  0.9650266  0.9906633
  4.00  4       0.9973034  0.9565817  0.9918070
  4.00  5       0.9973209  0.9511771  0.9925360
  4.00  6       0.9972829  0.9497044  0.9927523
  8.00  1       0.9971594  0.9841475  0.9843667
  8.00  2       0.9972074  0.9718358  0.9890010
  8.00  3       0.9972712  0.9652192  0.9906455
  8.00  4       0.9973020  0.9566114  0.9918041
  8.00  5       0.9973205  0.9512334  0.9925389
  8.00  6       0.9972834  0.9497429  0.9927493

Sens was used to select the optimal model using  the largest value.
The final values used for the model were cost = 1 and weight = 1.
Linear Support Vector Machines with Class Weights

6279 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
Resampling results across tuning parameters:

  cost  weight  ROC        Sens       Spec
  0.25  1       0.9980287  0.9812719  0.9866219
  0.25  2       0.9980143  0.9644742  0.9906349
  0.25  3       0.9979293  0.9497574  0.9930876
  0.25  4       0.9978364  0.9427743  0.9935891
  0.25  5       0.9978707  0.9360859  0.9942025
  0.25  6       0.9978719  0.9332623  0.9947041
```

```
  0.50  1       0.9981744  0.9845425  0.9860086
  0.50  2       0.9982134  0.9672248  0.9908022
  0.50  3       0.9981031  0.9595693  0.9929206
  0.50  4       0.9980144  0.9473801  0.9938681
  0.50  5       0.9979766  0.9418075  0.9942025
  0.50  6       0.9979813  0.9369021  0.9948714
  1.00  1       0.9982680  0.9846165  0.9854506
  1.00  2       0.9982459  0.9698254  0.9910814
  1.00  3       0.9981808  0.9633592  0.9931993
  1.00  4       0.9981415  0.9536985  0.9938683
  1.00  5       0.9980927  0.9458943  0.9942584
  1.00  6       0.9980835  0.9421046  0.9945927
  2.00  1       0.9983034  0.9840220  0.9860639
  2.00  2       0.9982542  0.9704197  0.9915271
  2.00  3       0.9982281  0.9650690  0.9931434
  2.00  4       0.9982369  0.9580091  0.9937567
  2.00  5       0.9981548  0.9479006  0.9939240
  2.00  6       0.9981032  0.9437398  0.9943698
  4.00  1       0.9982958  0.9832785  0.9868444
  4.00  2       0.9982848  0.9711629  0.9914713
  4.00  3       0.9982398  0.9655151  0.9929206
  4.00  4       0.9982410  0.9600151  0.9935337
  4.00  5       0.9981453  0.9492389  0.9938126
  4.00  6       0.9980964  0.9448545  0.9943141
  8.00  1       0.9982739  0.9833529  0.9867329
  8.00  2       0.9982954  0.9712375  0.9914158
  8.00  3       0.9982427  0.9655154  0.9928092
  8.00  4       0.9982527  0.9609814  0.9933664
  8.00  5       0.9981316  0.9500565  0.9938126
  8.00  6       0.9981022  0.9449285  0.9940913

Sens was used to select the optimal model using  the largest value.
The final values used for the model were cost = 1 and weight = 1.
Linear Support Vector Machines with Class Weights

68394 samples
    8 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
Resampling results across tuning parameters:
```

```
  cost  weight  ROC        Sens       Spec
  0.25  1       0.9906624  0.9765505  0.9492499
  0.25  2       0.9908383  0.9596106  0.9638748
  0.25  3       0.9908743  0.9443677  0.9700709
  0.25  4       0.9909043  0.9285398  0.9739870
  0.25  5       0.9909108  0.9156307  0.9771648
  0.25  6       0.9909092  0.9036657  0.9792794
  0.50  1       0.9906587  0.9765100  0.9492853
  0.50  2       0.9908375  0.9595295  0.9639161
  0.50  3       0.9908739  0.9443214  0.9700945
  0.50  4       0.9909043  0.9284818  0.9740224
  0.50  5       0.9909110  0.9156423  0.9771943
  0.50  6       0.9909093  0.9037062  0.9792794
  1.00  1       0.9906564  0.9764463  0.9493207
  1.00  2       0.9908372  0.9595237  0.9639516
  1.00  3       0.9908738  0.9442982  0.9701063
  1.00  4       0.9909044  0.9284876  0.9740461
  1.00  5       0.9909109  0.9156712  0.9771943
  1.00  6       0.9909092  0.9037294  0.9792853
  2.00  1       0.9906558  0.9764289  0.9493030
  2.00  2       0.9908371  0.9595064  0.9639634
  2.00  3       0.9908739  0.9442924  0.9701181
  2.00  4       0.9909044  0.9284818  0.9740343
  2.00  5       0.9909112  0.9156423  0.9771884
  2.00  6       0.9909093  0.9037178  0.9792735
  4.00  1       0.9906552  0.9764347  0.9492853
  4.00  2       0.9908372  0.9594774  0.9639693
  4.00  3       0.9908738  0.9443040  0.9701122
  4.00  4       0.9909045  0.9284818  0.9740461
  4.00  5       0.9909110  0.9156423  0.9771884
  4.00  6       0.9909092  0.9037236  0.9792794
  8.00  1       0.9906552  0.9764289  0.9492971
  8.00  2       0.9908369  0.9594890  0.9639693
  8.00  3       0.9908740  0.9442924  0.9701122
  8.00  4       0.9909042  0.9284760  0.9740461
  8.00  5       0.9909108  0.9156481  0.9771825
  8.00  6       0.9909094  0.9037352  0.9792794

Sens was used to select the optimal model using  the largest value.
The final values used for the model were cost = 0.25 and weight = 1.

Call:
summary.diff.resamples(object = results$modelDiff[[name]])
```

```
p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down       up         smote      rose
original           -0.0087257 -0.0082053 -0.0092990 -0.0016934
down     7.230e-12              0.0005204 -0.0005733  0.0070323
up       5.960e-12 1.00                   -0.0010936  0.0065119
smote    1.615e-13 1.00       3.610e-07               0.0076056
rose     0.51      < 2.2e-16  < 2.2e-16  < 2.2e-16

Sens
         original  down       up         smote      rose
original           -0.1877328 -0.1890264 -0.1894955 -0.1814294
down     < 2.2e-16             -0.0012935 -0.0017626  0.0063034
up       < 2.2e-16 1.00000                -0.0004691  0.0075969
smote    < 2.2e-16 1.00000    1.00000                 0.0080660
rose     < 2.2e-16 0.01682    < 2.2e-16  2.171e-09

Spec
         original  down      up        smote     rose
original            0.021160  0.014789  0.013652  0.049852
down     6.235e-09           -0.006371 -0.007508  0.028693
up       < 2.2e-16 0.2523              -0.001137  0.035064
smote    < 2.2e-16 0.1862    1.0000                0.036201
rose     < 2.2e-16 6.178e-13 < 2.2e-16 < 2.2e-16

$original
ROC curve variable importance

              Importance
Prof.mu          100.000
DM.mu             85.714
Prof.sigma        74.819
DM.sigma          31.867
DM.skew           27.353
DM.kurtosis       18.953
Prof.skew          4.232
Prof.kurtosis      0.000

$down
ROC curve variable importance
```

```
              Importance
Prof.mu          100.000
DM.mu             85.460
Prof.sigma        75.019
DM.sigma          32.777
DM.skew           27.321
DM.kurtosis       19.819
Prof.skew          4.238
Prof.kurtosis      0.000

$up
ROC curve variable importance

              Importance
Prof.mu          100.000
DM.mu             85.496
Prof.sigma        74.717
DM.sigma          31.850
DM.skew           27.378
DM.kurtosis       19.181
Prof.skew          4.338
Prof.kurtosis      0.000

$smote
ROC curve variable importance

              Importance
Prof.mu          100.000
DM.mu             86.057
Prof.sigma        75.168
DM.sigma          31.437
DM.skew           27.267
DM.kurtosis       18.309
Prof.skew          4.163
Prof.kurtosis      0.000

$rose
ROC curve variable importance

              Importance
Prof.kurtosis     100.00
Prof.skew          93.38
DM.kurtosis        79.88
DM.skew            71.96
```

```
DM.sigma           66.09
Prof.sigma         26.90
nDM.mu              16.12
Prof.mu             0.00


Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50

ROC
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9729  0.9847 0.9903 0.9890  0.9929 0.9993    0
down     0.9879  0.9970 0.9985 0.9977  0.9996 1.0000    0
up       0.9963  0.9970 0.9972 0.9972  0.9974 0.9979    0
smote    0.9951  0.9976 0.9986 0.9983  0.9991 0.9996    0
rose     0.9890  0.9900 0.9908 0.9907  0.9913 0.9927    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.7000  0.7759 0.7889 0.7951  0.8217 0.8764    0
down     0.9444  0.9776 0.9888 0.9829  0.9889 1.0000    0
up       0.9813  0.9831 0.9841 0.9841  0.9852 0.9867    0
smote    0.9703  0.9814 0.9851 0.9846  0.9888 1.0000    0
rose     0.9687  0.9752 0.9768 0.9766  0.9783 0.9815    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9982  0.9988 0.9991 0.9991  0.9994 1.0000    0
down     0.9222  0.9694 0.9778 0.9779  0.9889 1.0000    0
up       0.9810  0.9830 0.9845 0.9843  0.9856 0.9880    0
smote    0.9721  0.9833 0.9861 0.9855  0.9889 0.9944    0
rose     0.9430  0.9474 0.9494 0.9492  0.9509 0.9569    0
```

Appendix D

C5.0 Results

Plots

This section contains the plots for the C5.0 models' training and evaluation process. An extra dotplot is included.

Figure D.1: C5.0 model dot plot

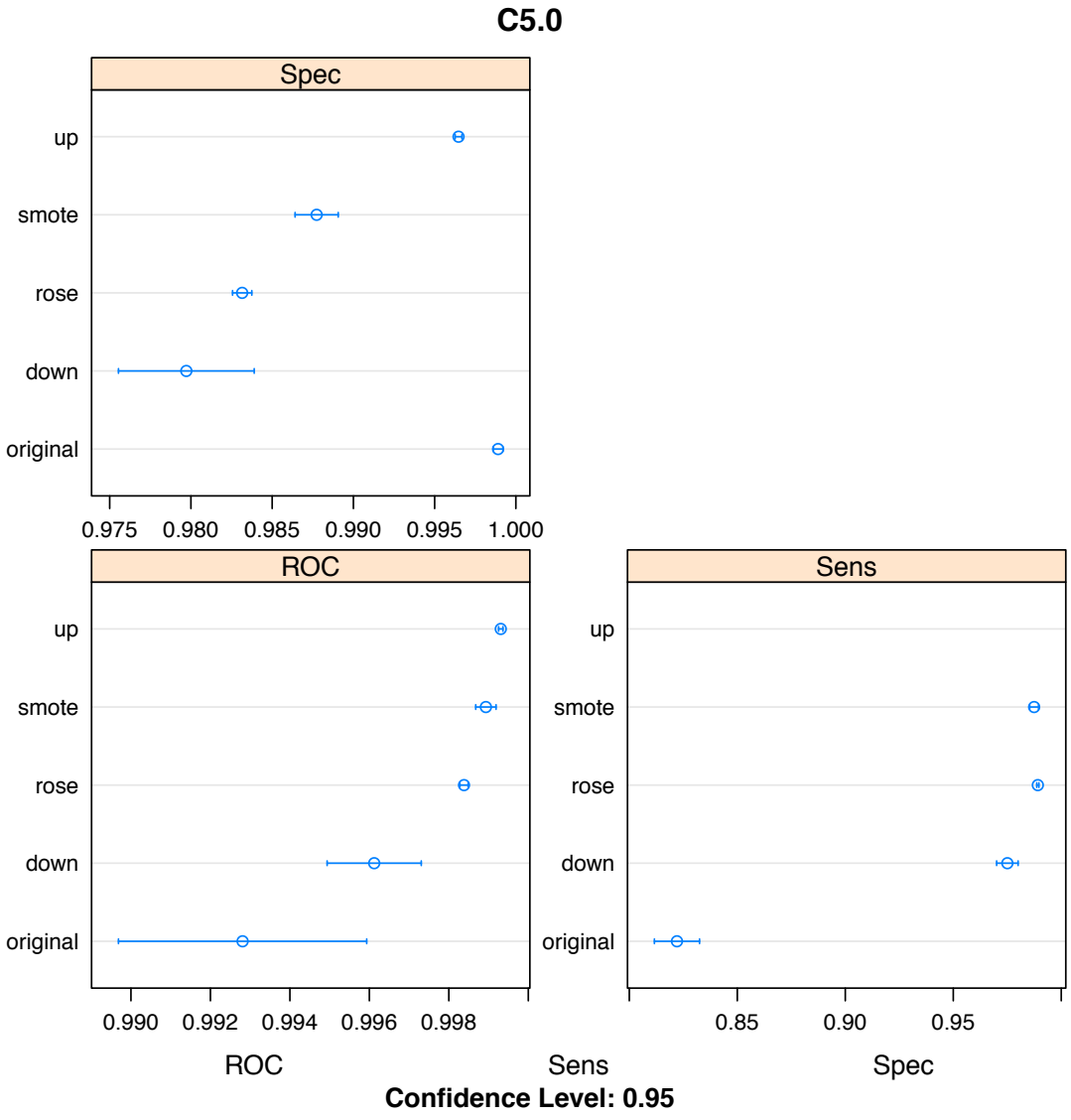


Figure D.2: C5.0 model box and whiskers plot

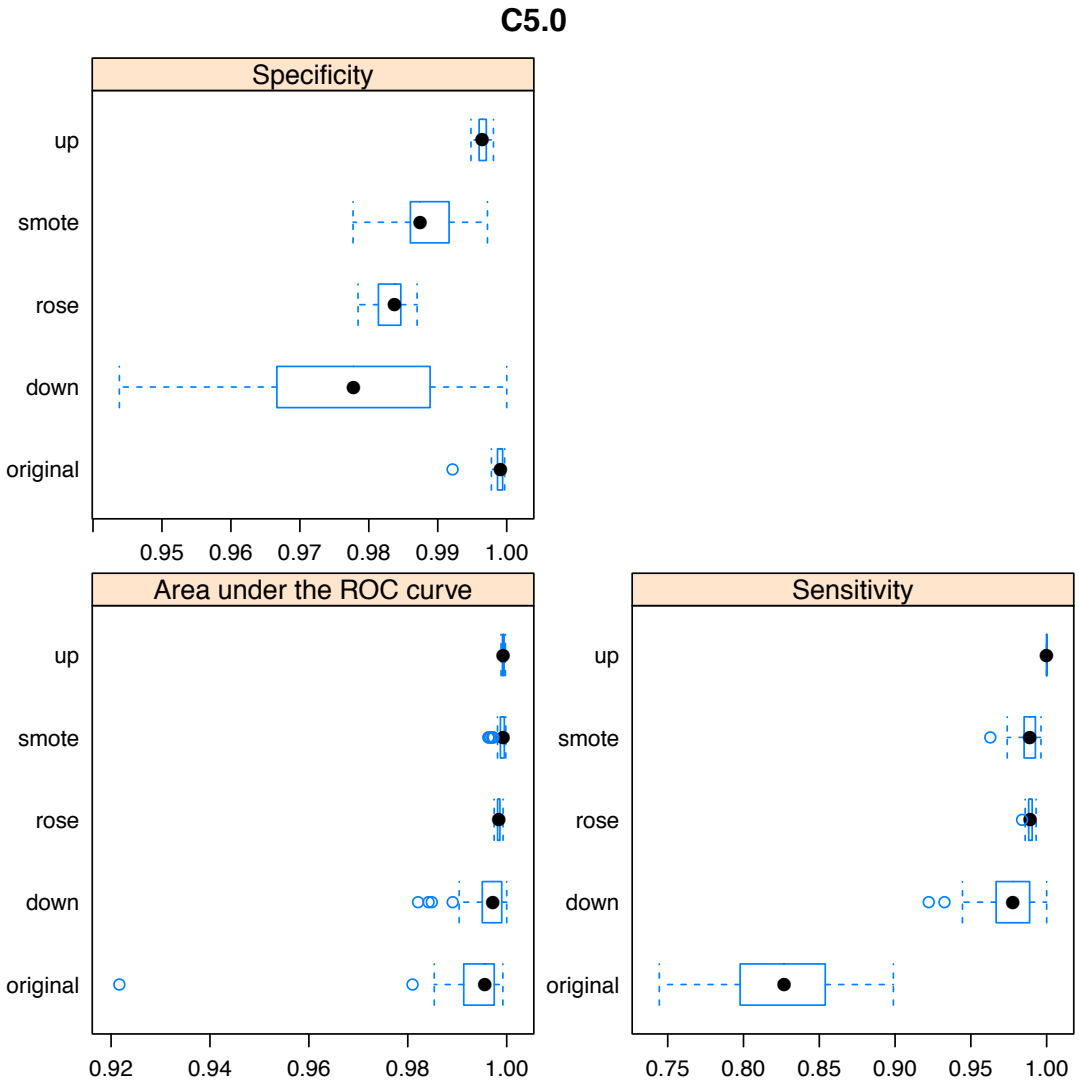
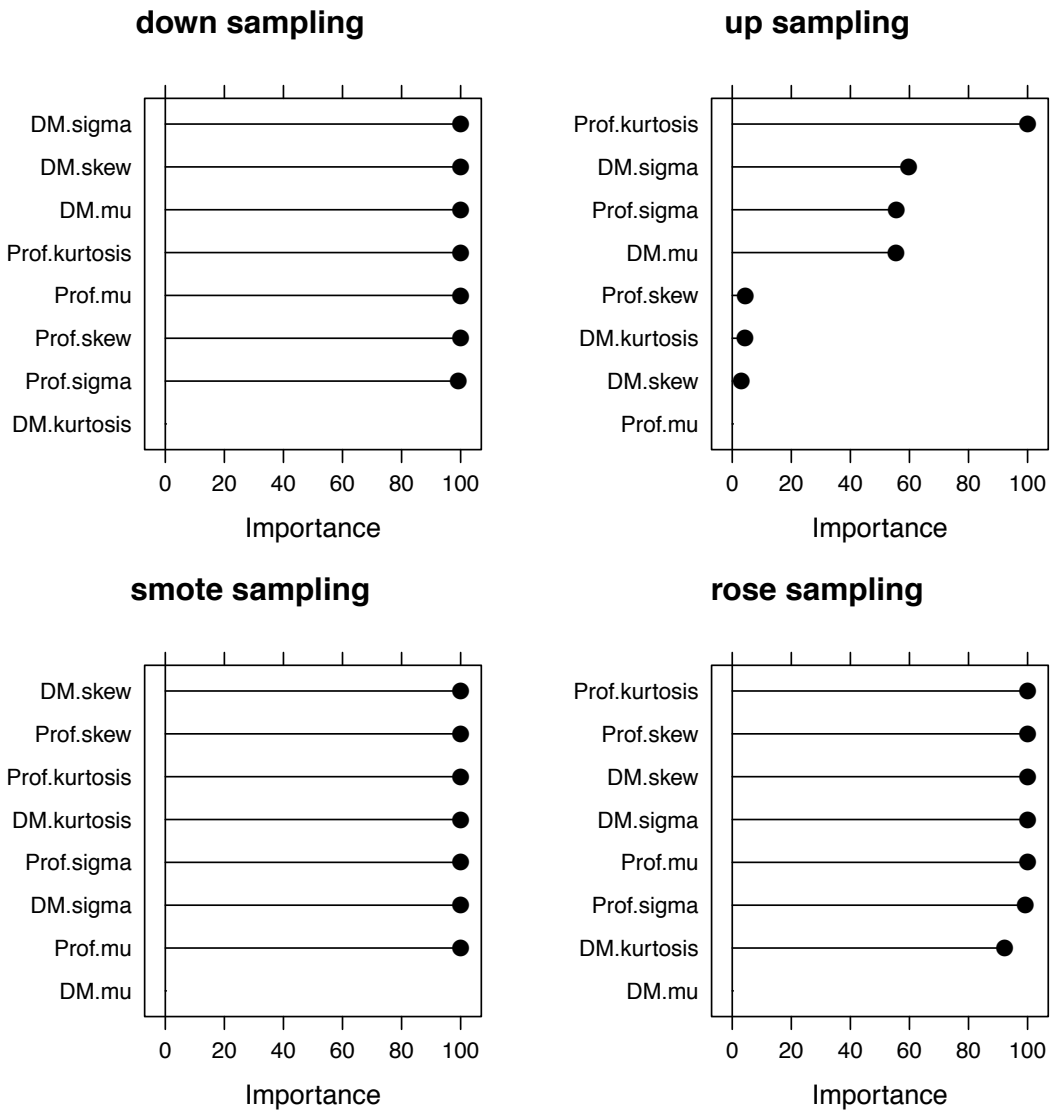


Figure D.3: C5.0 model variable importance plots for four sample sets



Data

```
Summary Text Output for the C5.0 model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       239        27
  nonpulsar     60     22472

               Accuracy : 0.9962
                 95% CI : (0.9953, 0.9969)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8441
 Mcnemar's Test P-Value : 0.0006019

            Sensitivity : 0.79933
            Specificity : 0.99880
         Pos Pred Value : 0.89850
         Neg Pred Value : 0.99734
             Prevalence : 0.01312
         Detection Rate : 0.01048
   Detection Prevalence : 0.01167
      Balanced Accuracy : 0.89907

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       292       387
  nonpulsar      7     22112

               Accuracy : 0.9827
                 95% CI : (0.9809, 0.9844)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5897
```

```
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98280
         Pos Pred Value : 0.43004
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02978
      Balanced Accuracy : 0.97969

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       267        93
  nonpulsar     32     22406

               Accuracy : 0.9945
                 95% CI : (0.9935, 0.9954)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8076
 Mcnemar's Test P-Value : 8.025e-08

            Sensitivity : 0.89298
            Specificity : 0.99587
         Pos Pred Value : 0.74167
         Neg Pred Value : 0.99857
             Prevalence : 0.01312
         Detection Rate : 0.01171
   Detection Prevalence : 0.01579
      Balanced Accuracy : 0.94442

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       289       266
```

```
  nonpulsar     10     22233

               Accuracy : 0.9879
                 95% CI : (0.9864, 0.9893)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.09392

                  Kappa : 0.6712
 Mcnemar's Test P-Value : < 2e-16

            Sensitivity : 0.96656
            Specificity : 0.98818
         Pos Pred Value : 0.52072
         Neg Pred Value : 0.99955
             Prevalence : 0.01312
         Detection Rate : 0.01268
   Detection Prevalence : 0.02434
      Balanced Accuracy : 0.97737

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       276       115
  nonpulsar     23     22384

               Accuracy : 0.9939
                 95% CI : (0.9929, 0.9949)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.797
 Mcnemar's Test P-Value : 9.451e-15

            Sensitivity : 0.92308
            Specificity : 0.99489
         Pos Pred Value : 0.70588
         Neg Pred Value : 0.99897
             Prevalence : 0.01312
         Detection Rate : 0.01211
   Detection Prevalence : 0.01715
      Balanced Accuracy : 0.95898
```

```
       'Positive' Class : pulsar

C5.0

68394 samples
    8 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9149366  0.8001873  0.9988651
  rules  FALSE   10      0.9894412  0.7991361  0.9986281
  rules  FALSE   20      0.9913611  0.8087091  0.9988444
  rules  FALSE   30      0.9920961  0.8097978  0.9989481
  rules  FALSE   40      0.9927754  0.8100325  0.9989896
  rules  FALSE   50      0.9927938  0.8136005  0.9989985
  rules   TRUE    1      0.9076484  0.7931086  0.9989244
  rules   TRUE   10      0.9834187  0.7975106  0.9977747
  rules   TRUE   20      0.9852197  0.8066592  0.9980148
  rules   TRUE   30      0.9855601  0.8053208  0.9981007
  rules   TRUE   40      0.9859264  0.8095506  0.9981955
  rules   TRUE   50      0.9859242  0.8088914  0.9981985
  tree   FALSE    1      0.9342529  0.8066841  0.9987407
  tree   FALSE   10      0.9905206  0.8107241  0.9984207
  tree   FALSE   20      0.9916918  0.8167091  0.9986814
  tree   FALSE   30      0.9922092  0.8202821  0.9987940
  tree   FALSE   40      0.9925919  0.8209488  0.9988592
  tree   FALSE   50      0.9928094  0.8220699  0.9989037
  tree    TRUE    1      0.9256448  0.7973184  0.9988355
  tree    TRUE   10      0.9843043  0.7957528  0.9977392
  tree    TRUE   20      0.9865233  0.8077828  0.9979170
  tree    TRUE   30      0.9870331  0.8102397  0.9979851
  tree    TRUE   40      0.9871373  0.8109288  0.9980711
  tree    TRUE   50      0.9871577  0.8147116  0.9981392

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 50, model = tree
and winnow = FALSE.
C5.0
```

```
1794 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9726353  0.9685818  0.9694407
  rules  FALSE   10      0.9949695  0.9741573  0.9777079
  rules  FALSE   20      0.9959791  0.9743745  0.9790512
  rules  FALSE   30      0.9962489  0.9730337  0.9801648
  rules  FALSE   40      0.9963708  0.9739226  0.9792734
  rules  FALSE   50      0.9962974  0.9737129  0.9781573
  rules   TRUE    1      0.9709091  0.9652160  0.9710187
  rules   TRUE   10      0.9939597  0.9717029  0.9745893
  rules   TRUE   20      0.9942475  0.9703695  0.9765943
  rules   TRUE   30      0.9944744  0.9717029  0.9750337
  rules   TRUE   40      0.9946312  0.9714831  0.9757029
  rules   TRUE   50      0.9948213  0.9719251  0.9750387
  tree   FALSE    1      0.9877305  0.9676904  0.9703271
  tree   FALSE   10      0.9945011  0.9705943  0.9801598
  tree   FALSE   20      0.9952930  0.9723770  0.9785943
  tree   FALSE   30      0.9958601  0.9723645  0.9794906
  tree   FALSE   40      0.9960401  0.9737129  0.9788265
  tree   FALSE   50      0.9961217  0.9750487  0.9797179
  tree    TRUE    1      0.9872562  0.9643246  0.9721298
  tree    TRUE   10      0.9941530  0.9712534  0.9734782
  tree    TRUE   20      0.9947184  0.9714782  0.9750362
  tree    TRUE   30      0.9950667  0.9712534  0.9750337
  tree    TRUE   40      0.9951060  0.9705818  0.9752559
  tree    TRUE   50      0.9952673  0.9696904  0.9752609

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 50, model = tree
and winnow = FALSE.
C5.0

134994 samples
     8 predictor
     2 classes: 'pulsar', 'nonpulsar'
```

```
Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494, 121496,
121495, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens      Spec
  rules  FALSE    1      0.9985671  0.999923  0.9955435
  rules  FALSE   10      0.9998175  1.000000  0.9981658
  rules  FALSE   20      0.9998631  1.000000  0.9982429
  rules  FALSE   30      0.9998840  1.000000  0.9982933
  rules  FALSE   40      0.9998898  1.000000  0.9983318
  rules  FALSE   50      0.9998890  1.000000  0.9983881
  rules   TRUE    1      0.9985671  0.999923  0.9955435
  rules   TRUE   10      0.9998175  1.000000  0.9981658
  rules   TRUE   20      0.9998631  1.000000  0.9982429
  rules   TRUE   30      0.9998840  1.000000  0.9982933
  rules   TRUE   40      0.9998898  1.000000  0.9983318
  rules   TRUE   50      0.9998890  1.000000  0.9983881
  tree   FALSE    1      0.9993072  1.000000  0.9964709
  tree   FALSE   10      0.9998449  1.000000  0.9972591
  tree   FALSE   20      0.9998810  1.000000  0.9977362
  tree   FALSE   30      0.9998903  1.000000  0.9977836
  tree   FALSE   40      0.9998959  1.000000  0.9978666
  tree   FALSE   50      0.9998965  1.000000  0.9978932
  tree    TRUE    1      0.9993072  1.000000  0.9964709
  tree    TRUE   10      0.9998449  1.000000  0.9972591
  tree    TRUE   20      0.9998810  1.000000  0.9977362
  tree    TRUE   30      0.9998903  1.000000  0.9977836
  tree    TRUE   40      0.9998959  1.000000  0.9978666
  tree    TRUE   50      0.9998965  1.000000  0.9978932

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = tree
and winnow = TRUE.
C5.0

6279 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
```

```
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9873101  0.9827586  0.9848936
  rules  FALSE   10      0.9983097  0.9852113  0.9872367
  rules  FALSE   20      0.9987659  0.9866983  0.9876270
  rules  FALSE   30      0.9989323  0.9874407  0.9877384
  rules  FALSE   40      0.9989641  0.9872177  0.9876266
  rules  FALSE   50      0.9990047  0.9872926  0.9877378
  rules   TRUE    1      0.9866896  0.9822390  0.9852287
  rules   TRUE   10      0.9983671  0.9848401  0.9869011
  rules   TRUE   20      0.9987949  0.9864009  0.9872358
  rules   TRUE   30      0.9988849  0.9860278  0.9876259
  rules   TRUE   40      0.9989059  0.9860289  0.9876818
  rules   TRUE   50      0.9989101  0.9860284  0.9877374
  tree   FALSE    1      0.9937762  0.9814955  0.9828327
  tree   FALSE   10      0.9983693  0.9835757  0.9887966
  tree   FALSE   20      0.9988289  0.9862520  0.9884068
  tree   FALSE   30      0.9989611  0.9859543  0.9886295
  tree   FALSE   40      0.9990209  0.9861030  0.9882391
  tree   FALSE   50      0.9990067  0.9857310  0.9879045
  tree    TRUE    1      0.9937936  0.9807523  0.9830560
  tree    TRUE   10      0.9983382  0.9832788  0.9874031
  tree    TRUE   20      0.9987582  0.9846174  0.9877372
  tree    TRUE   30      0.9988208  0.9852862  0.9874586
  tree    TRUE   40      0.9989381  0.9852113  0.9875142
  tree    TRUE   50      0.9989419  0.9851373  0.9875698

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 30, model = rules
and winnow = FALSE.
C5.0

68394 samples
    8 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
```

```
  rules  FALSE    1      0.9870904  0.9840852  0.9789604
  rules  FALSE   10      0.9975464  0.9846875  0.9860366
  rules  FALSE   20      0.9981284  0.9861991  0.9861193
  rules  FALSE   30      0.9982878  0.9866624  0.9863083
  rules  FALSE   40      0.9984233  0.9867261  0.9863201
  rules  FALSE   50      0.9984664  0.9866856  0.9863201
  rules   TRUE    1      0.9869682  0.9840273  0.9787773
  rules   TRUE   10      0.9974469  0.9848091  0.9858181
  rules   TRUE   20      0.9980553  0.9862107  0.9859657
  rules   TRUE   30      0.9982031  0.9865639  0.9861193
  rules   TRUE   40      0.9983442  0.9867087  0.9860543
  rules   TRUE   50      0.9983862  0.9866277  0.9861607
  tree   FALSE    1      0.9949878  0.9825968  0.9758653
  tree   FALSE   10      0.9974477  0.9877454  0.9819965
  tree   FALSE   20      0.9980185  0.9886778  0.9828529
  tree   FALSE   30      0.9981983  0.9887415  0.9831719
  tree   FALSE   40      0.9982767  0.9890079  0.9832310
  tree   FALSE   50      0.9983821  0.9891527  0.9831542
  tree    TRUE    1      0.9949895  0.9827010  0.9760189
  tree    TRUE   10      0.9973908  0.9876701  0.9819846
  tree    TRUE   20      0.9979748  0.9884462  0.9828529
  tree    TRUE   30      0.9981666  0.9886372  0.9831601
  tree    TRUE   40      0.9982371  0.9888573  0.9832900
  tree    TRUE   50      0.9983362  0.9889905  0.9831896

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 50, model = tree
and winnow = FALSE.

Call:
summary.diff.resamples(object = results$modelDiff[[name]])

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down       up         smote      rose
original           -0.0033123 -0.0064979 -0.0061229 -0.0055727
down     0.5336316            -0.0031855 -0.0028106 -0.0022604
up       0.0012658 1.631e-05              0.0003749  0.0009251
smote    0.0024843 0.0002608  0.0703213              0.0005502
rose     0.0081408 0.0041212  < 2.2e-16  0.0046965
```

```
Sens
         original  down      up        smote     rose
original           -0.152979 -0.177930 -0.165371 -0.167083
down     < 2.2e-16           -0.024951 -0.012392 -0.014104
up       < 2.2e-16 1.555e-12            0.012559  0.010847
smote    < 2.2e-16 8.500e-05 1.194e-15           -0.001712
rose     < 2.2e-16 6.535e-06 < 2.2e-16 1

Spec
         original  down      up        smote     rose
original            0.019186  0.002433  0.011165  0.015749
down     1.694e-11           -0.016753 -0.008021 -0.003436
up       2.886e-15 1.886e-09            0.008733  0.013317
smote    < 2.2e-16 0.01191   < 2.2e-16            0.004584
rose     < 2.2e-16 1.00000   < 2.2e-16 1.883e-07

$original
C5.0 variable importance

              Overall
Prof.skew         NaN
Prof.sigma        NaN
Prof.mu           NaN
DM.skew           NaN
DM.mu             NaN
DM.kurtosis       NaN
DM.sigma          NaN
Prof.kurtosis     NaN

$down
C5.0 variable importance

              Overall
Prof.skew      100.00
DM.skew        100.00
DM.sigma       100.00
DM.mu          100.00
Prof.kurtosis  100.00
Prof.mu        100.00
Prof.sigma      99.25
DM.kurtosis      0.00

$up
C5.0 variable importance
```

```
              Overall
Prof.kurtosis 100.000
DM.sigma       59.813
Prof.sigma     55.595
DM.mu          55.511
Prof.skew       4.480
DM.kurtosis     4.291
DM.skew         3.103
Prof.mu         0.000

$smote
C5.0 variable importance

              Overall
Prof.skew         100
Prof.sigma        100
DM.sigma          100
DM.kurtosis       100
Prof.mu           100
DM.skew           100
Prof.kurtosis     100
DM.mu               0

$rose
C5.0 variable importance

              Overall
Prof.kurtosis  100.00
DM.sigma       100.00
Prof.skew      100.00
DM.skew        100.00
Prof.mu        100.00
Prof.sigma      99.26
DM.kurtosis     92.28
DM.mu            0.00


Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50
```

```
ROC
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9217  0.9914 0.9956 0.9928  0.9975 0.9992    0
down     0.9821  0.9951 0.9972 0.9961  0.9990 1.0000    0
up       0.9988  0.9991 0.9993 0.9993  0.9995 0.9997    0
smote    0.9963  0.9987 0.9993 0.9989  0.9995 0.9998    0
rose     0.9975  0.9982 0.9984 0.9984  0.9986 0.9993    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.7444  0.7983 0.8268 0.8221  0.8539 0.8989    0
down     0.9222  0.9667 0.9778 0.9750  0.9888 1.0000    0
up       1.0000  1.0000 1.0000 1.0000  1.0000 1.0000    0
smote    0.9628  0.9851 0.9888 0.9874  0.9926 0.9963    0
rose     0.9838  0.9882 0.9891 0.9892  0.9903 0.9930    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9921  0.9987 0.9991 0.9989  0.9994 0.9997    0
down     0.9438  0.9694 0.9778 0.9797  0.9889 1.0000    0
up       0.9948  0.9960 0.9964 0.9965  0.9970 0.9981    0
smote    0.9777  0.9860 0.9874 0.9877  0.9916 0.9972    0
rose     0.9784  0.9815 0.9838 0.9832  0.9846 0.9870    0
```

Appendix E

Bagged Trees Results

Plots

This section contains the plots for the bagged tree training and evaluation process. An extra dotplot is included.

Figure E.1: Bagged tree model dot plot

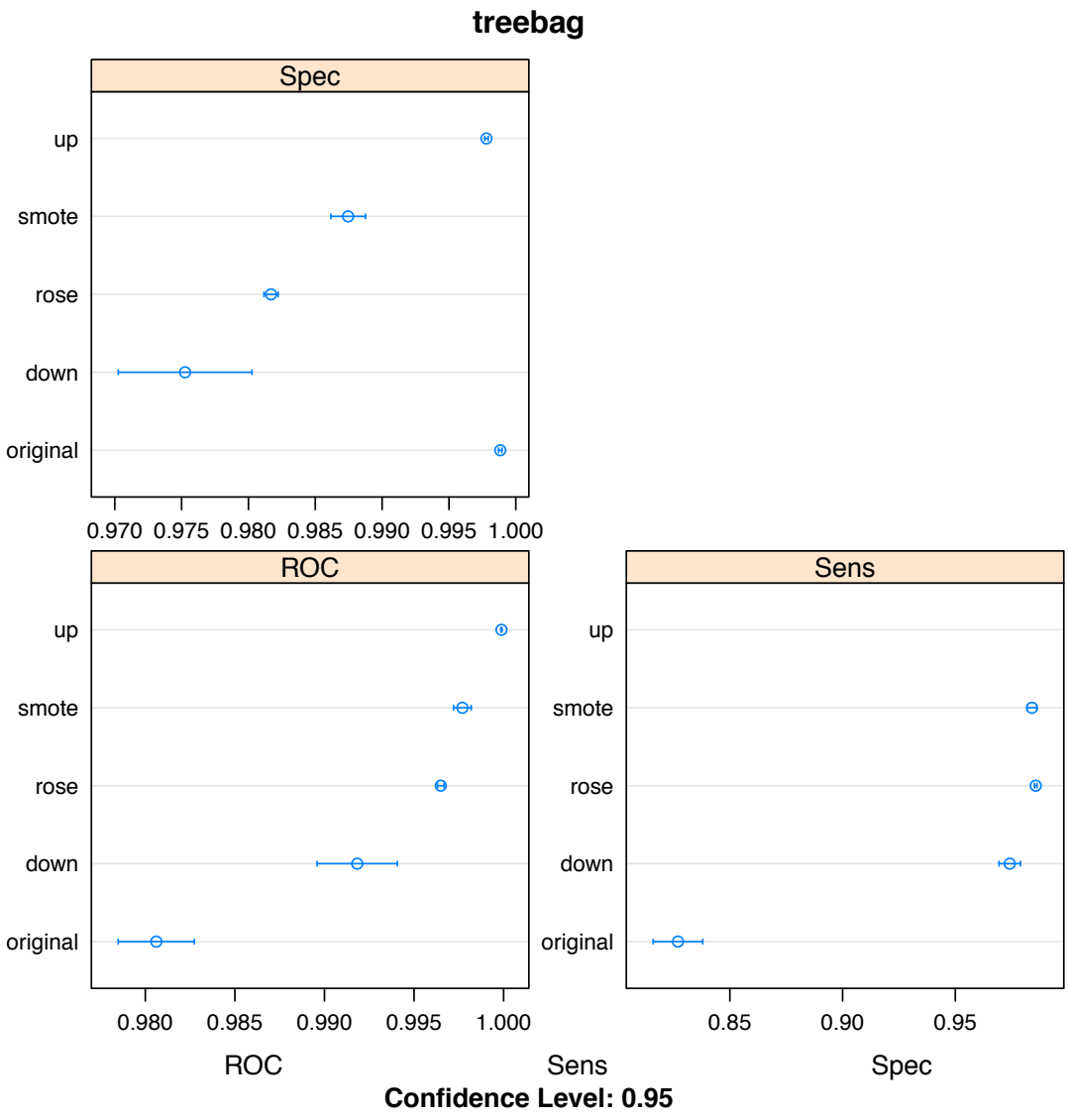


Figure E.2: Bagged tree model box and whiskers plot

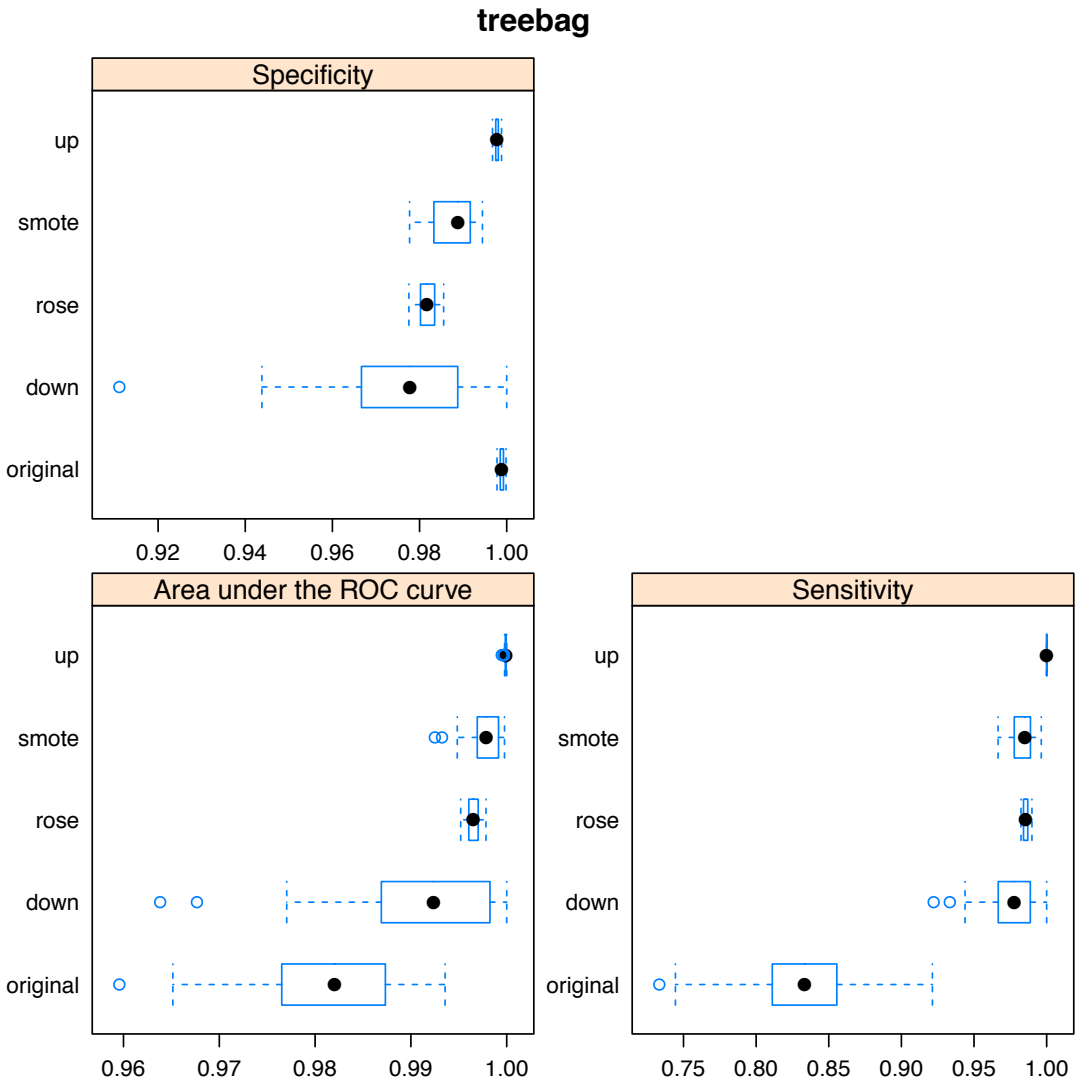


Figure E.3: Bagged tree model variable importance plots for four sample sets

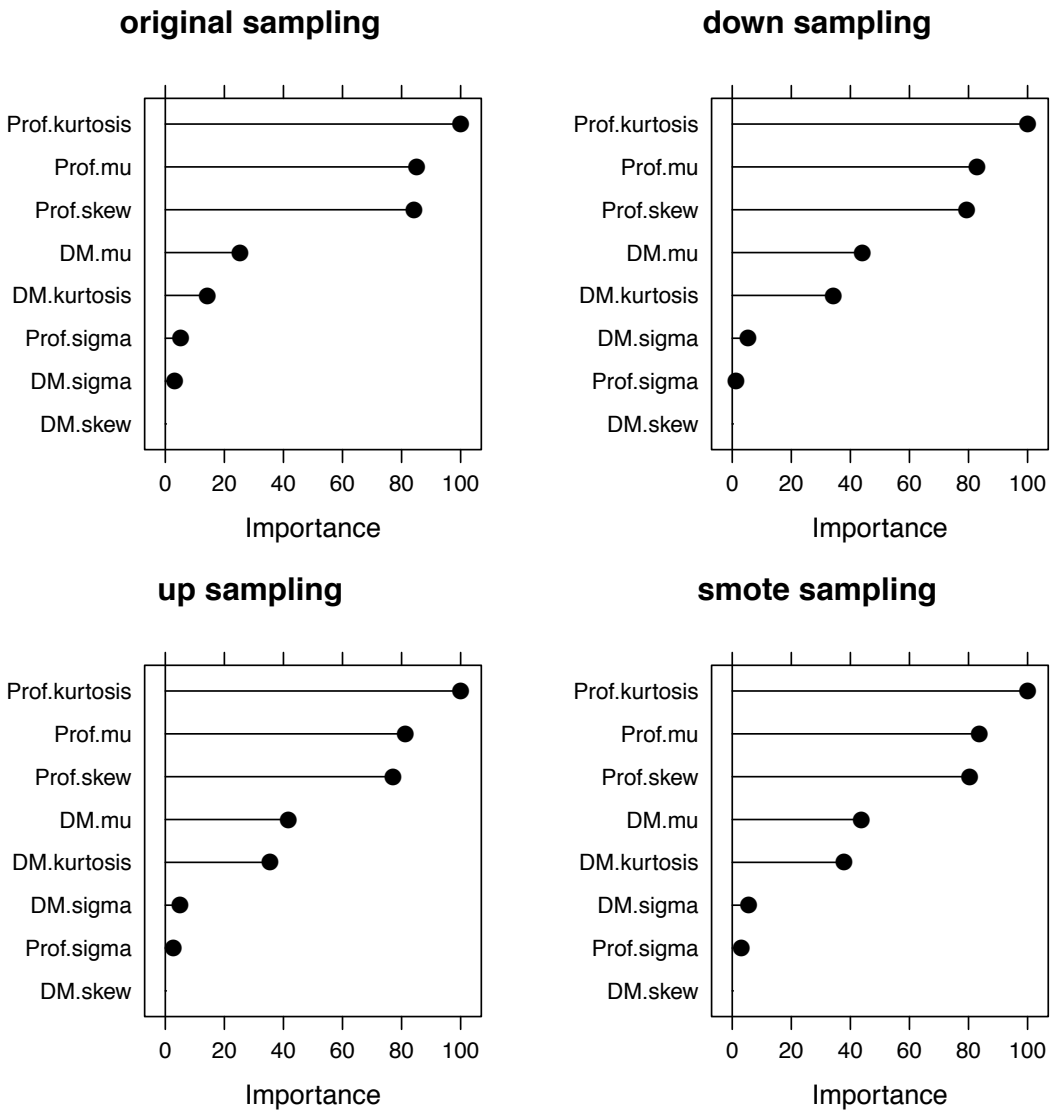
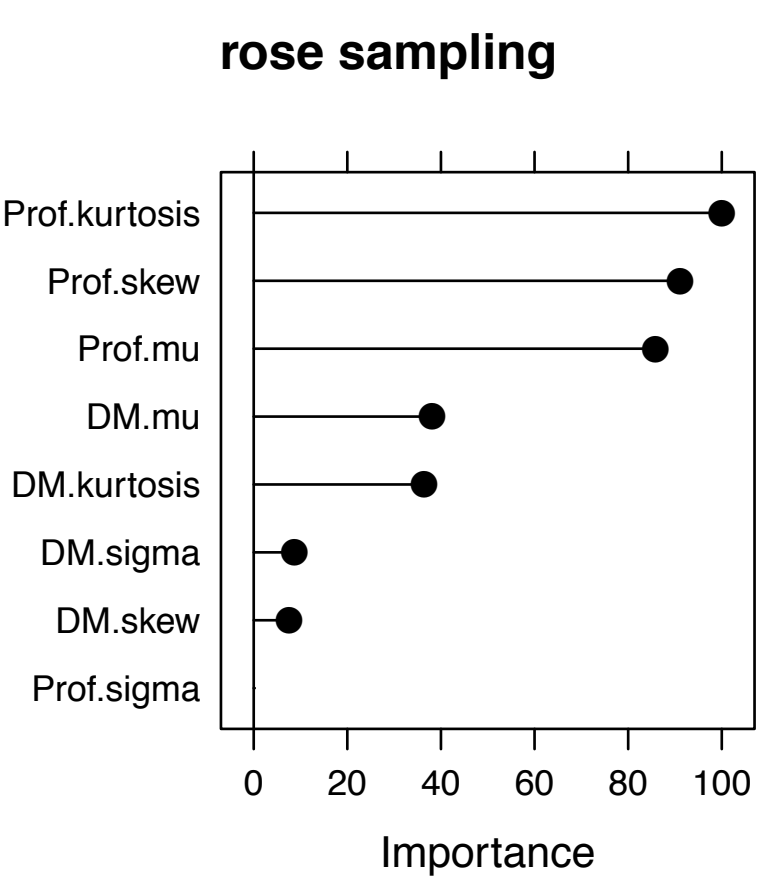


Figure E.4: Bagged tree model variable importance plots for ROSE sample sets



Data

```
Summary Text Output for the treebag model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       244        30
  nonpulsar     55     22469

               Accuracy : 0.9963
                 95% CI : (0.9954, 0.997)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8498
 Mcnemar's Test P-Value : 0.009237

            Sensitivity : 0.81605
            Specificity : 0.99867
         Pos Pred Value : 0.89051
         Neg Pred Value : 0.99756
             Prevalence : 0.01312
         Detection Rate : 0.01070
   Detection Prevalence : 0.01202
      Balanced Accuracy : 0.90736

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       294       421
  nonpulsar      5     22078

               Accuracy : 0.9813
                 95% CI : (0.9795, 0.983)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.572
 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.98328
            Specificity : 0.98129
         Pos Pred Value : 0.41119
         Neg Pred Value : 0.99977
             Prevalence : 0.01312
         Detection Rate : 0.01290
   Detection Prevalence : 0.03136
      Balanced Accuracy : 0.98228

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       257        61
  nonpulsar     42     22438

               Accuracy : 0.9955
                 95% CI : (0.9945, 0.9963)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8308
 Mcnemar's Test P-Value : 0.07613

            Sensitivity : 0.85953
            Specificity : 0.99729
         Pos Pred Value : 0.80818
         Neg Pred Value : 0.99813
             Prevalence : 0.01312
         Detection Rate : 0.01127
   Detection Prevalence : 0.01395
      Balanced Accuracy : 0.92841

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       288       291
  nonpulsar     11     22208
```

```
               Accuracy : 0.9868
                 95% CI : (0.9852, 0.9882)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.5843

                  Kappa : 0.65
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.96321
            Specificity : 0.98707
         Pos Pred Value : 0.49741
         Neg Pred Value : 0.99950
             Prevalence : 0.01312
         Detection Rate : 0.01263
   Detection Prevalence : 0.02540
      Balanced Accuracy : 0.97514

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       281       139
  nonpulsar     18     22360

               Accuracy : 0.9931
                 95% CI : (0.992, 0.9941)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7782
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.93980
            Specificity : 0.99382
         Pos Pred Value : 0.66905
         Neg Pred Value : 0.99920
             Prevalence : 0.01312
         Detection Rate : 0.01233
   Detection Prevalence : 0.01842
      Balanced Accuracy : 0.96681
```

```
       'Positive' Class : pulsar

Bagged CART

68394 samples
    8 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results:

  ROC        Sens       Spec
  0.9806127  0.8269738  0.9988414


Bagged CART

1794 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results:

  ROC        Sens       Spec
  0.9918295  0.9741523  0.9752634


Bagged CART

134994 samples
     8 predictor
     2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494, 121496,
121495, ...
Resampling results:
```

```
  ROC      Sens  Spec
  0.99988  1     0.9978043


Bagged CART

6279 samples
   8 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
Resampling results:

  ROC        Sens       Spec
  0.9976991  0.9840226  0.9874577


Bagged CART

68394 samples
    8 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
Resampling results:

  ROC        Sens       Spec
  0.9964938  0.9856778  0.9816952



Call:
summary.diff.resamples(object = results$modelDiff[[name]])

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down      up        smote     rose
```

```
original           -0.011217 -0.019267 -0.017086 -0.015881
down     7.298e-08           -0.008050 -0.005870 -0.004664
up       < 2.2e-16 2.961e-08            0.002181  0.003386
smote    < 2.2e-16 5.509e-05 1.228e-10             0.001205
rose     < 2.2e-16 0.001410  < 2.2e-16 0.001142

Sens
         original  down      up        smote     rose
original           -0.147179 -0.173026 -0.157049 -0.158704
down     < 2.2e-16           -0.025848 -0.009870 -0.011526
up       < 2.2e-16 1.007e-13            0.015977  0.014322
smote    < 2.2e-16 0.0006623 < 2.2e-16           -0.001655
rose     < 2.2e-16 9.142e-05 < 2.2e-16 1.0000000

Spec
         original  down      up        smote     rose
original            0.023578  0.001037  0.011384  0.017146
down     9.153e-12           -0.022541 -0.012194 -0.006432
up       2.874e-14 4.770e-11            0.010347  0.016109
smote    < 2.2e-16 0.0003383 < 2.2e-16             0.005762
rose     < 2.2e-16 0.1258708 < 2.2e-16 3.506e-09

$original
treebag variable importance

              Overall
Prof.kurtosis 100.000
Prof.mu        85.129
Prof.skew      84.185
DM.mu          25.319
DM.kurtosis    14.218
Prof.sigma      5.197
DM.sigma        3.252
DM.skew         0.000

$down
treebag variable importance

              Overall
Prof.kurtosis 100.000
Prof.mu        82.894
Prof.skew      79.360
DM.mu          44.076
DM.kurtosis    34.198
```

```
DM.sigma        5.327
Prof.sigma      1.290
DM.skew         0.000

$up
treebag variable importance

              Overall
Prof.kurtosis 100.000
Prof.mu        81.298
Prof.skew      77.119
DM.mu          41.663
DM.kurtosis    35.445
DM.sigma        4.948
Prof.sigma      2.756
DM.skew         0.000

$smote
treebag variable importance

              Overall
Prof.kurtosis 100.000
Prof.mu        83.675
Prof.skew      80.434
DM.mu          43.691
DM.kurtosis    37.907
DM.sigma        5.579
Prof.sigma      3.067
DM.skew         0.000

$rose
treebag variable importance

              Overall
Prof.kurtosis 100.000
Prof.skew      91.092
Prof.mu        85.854
DM.mu          38.151
DM.kurtosis    36.435
DM.sigma        8.725
DM.skew         7.568
Prof.sigma      0.000
```

```
Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50

ROC
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9596  0.9765 0.9820 0.9806  0.9873 0.9936    0
down     0.9638  0.9875 0.9924 0.9918  0.9982 1.0000    0
up       0.9995  0.9999 0.9999 0.9999  0.9999 1.0000    0
smote    0.9925  0.9970 0.9979 0.9977  0.9991 0.9998    0
rose     0.9952  0.9960 0.9965 0.9965  0.9970 0.9978    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.7333  0.8111 0.8333 0.8270  0.8551 0.9213    0
down     0.9222  0.9667 0.9778 0.9742  0.9888 1.0000    0
up       1.0000  1.0000 1.0000 1.0000  1.0000 1.0000    0
smote    0.9665  0.9777 0.9851 0.9840  0.9888 0.9963    0
rose     0.9823  0.9841 0.9855 0.9857  0.9870 0.9899    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
original 0.9978  0.9985 0.9988 0.9988  0.9992 0.9999    0
down     0.9111  0.9667 0.9778 0.9753  0.9888 1.0000    0
up       0.9967  0.9975 0.9978 0.9978  0.9981 0.9988    0
smote    0.9777  0.9840 0.9888 0.9875  0.9916 0.9944    0
rose     0.9776  0.9803 0.9817 0.9817  0.9835 0.9855    0
```

Appendix F

Stacked Classifiers

A set of stacked classifiers was produced to explore the efficacy of stacking. This appendix contains the detailed results, plots and text data from those experiments.

Neural Network Ensemble Plots

This section contains the plots for the neural network ensemble training and evaluation process.

Figure F.1: Neural network ensemble model dot plot

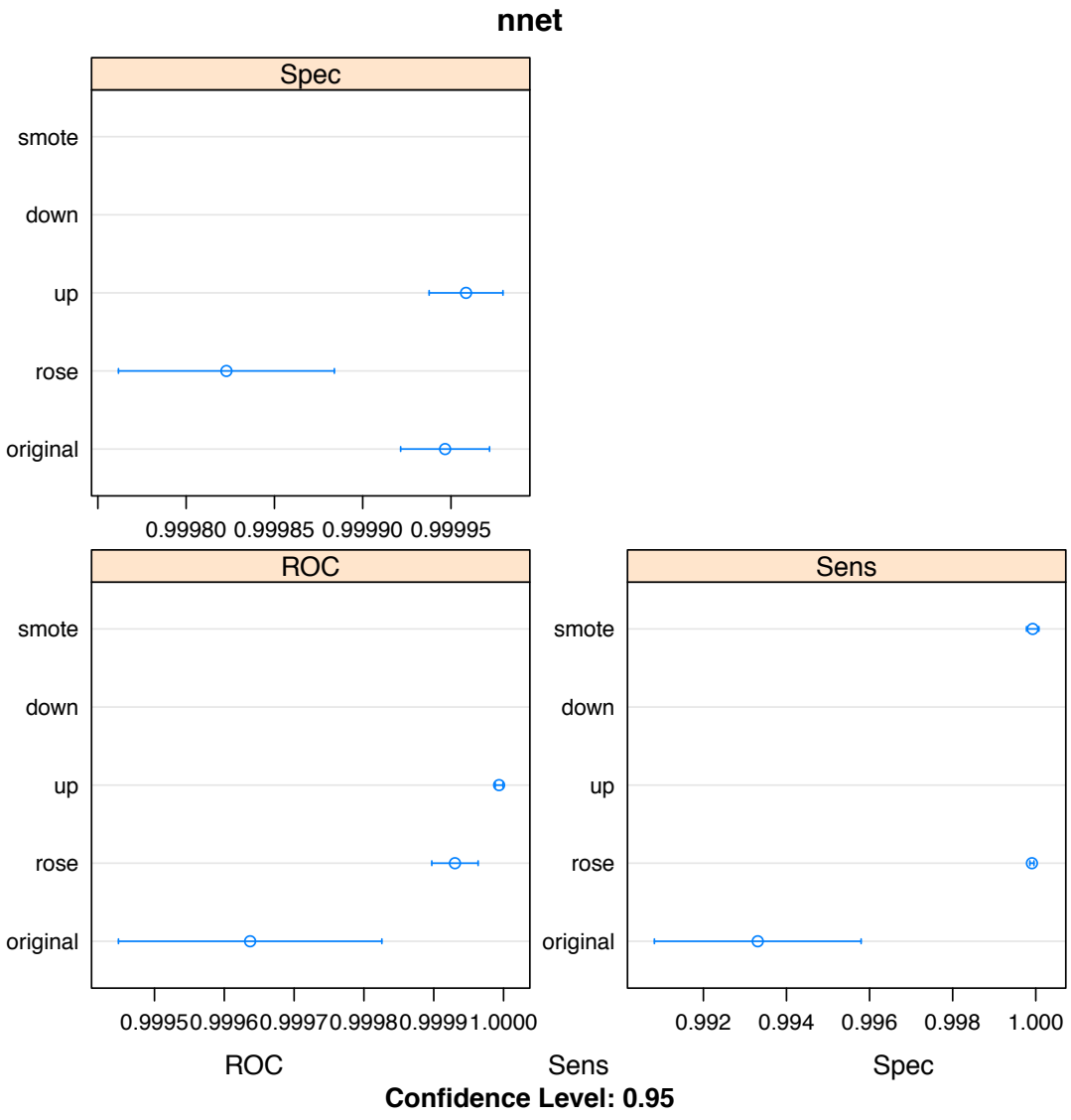


Figure F.2: Neural network ensemble model box and whiskers plot

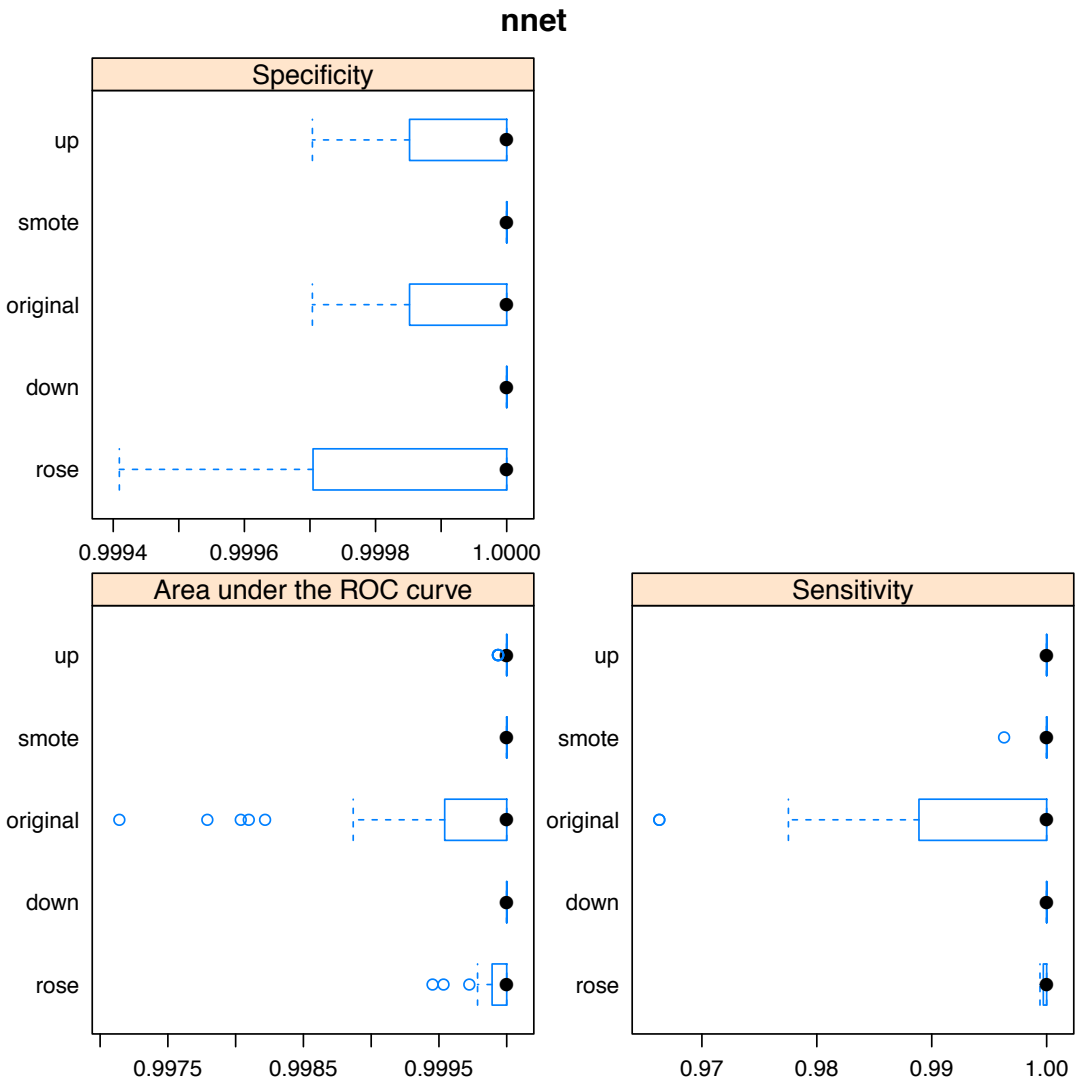


Figure F.3: Neural network ensemble model variable importance plots for original sample sets, 1 of 2

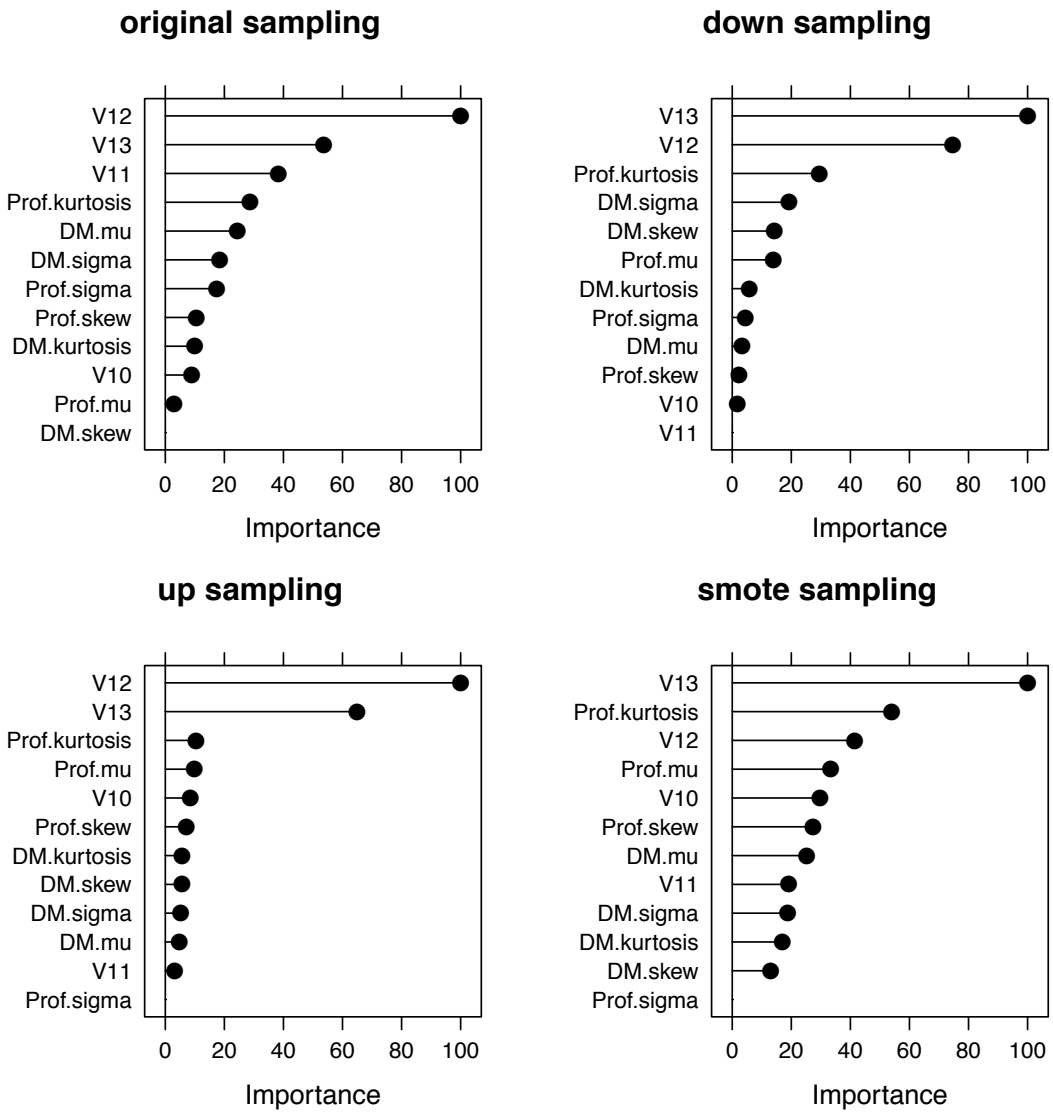


Figure F.4: Neural network ensemble model variable importance plots for original sample sets, 2 of 2

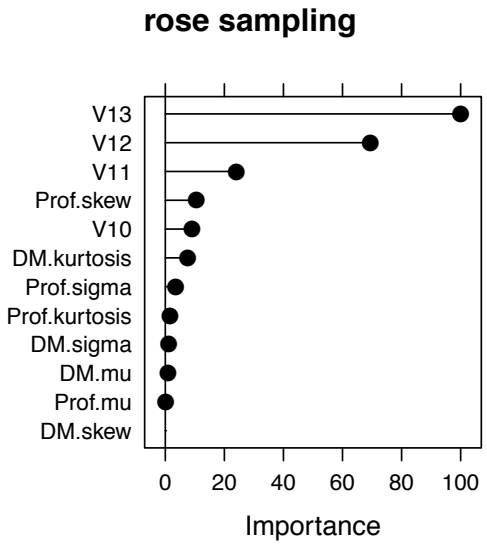


Figure F.5: Neural network ensemble model variable importance plots for downsampled data, 1 of 2

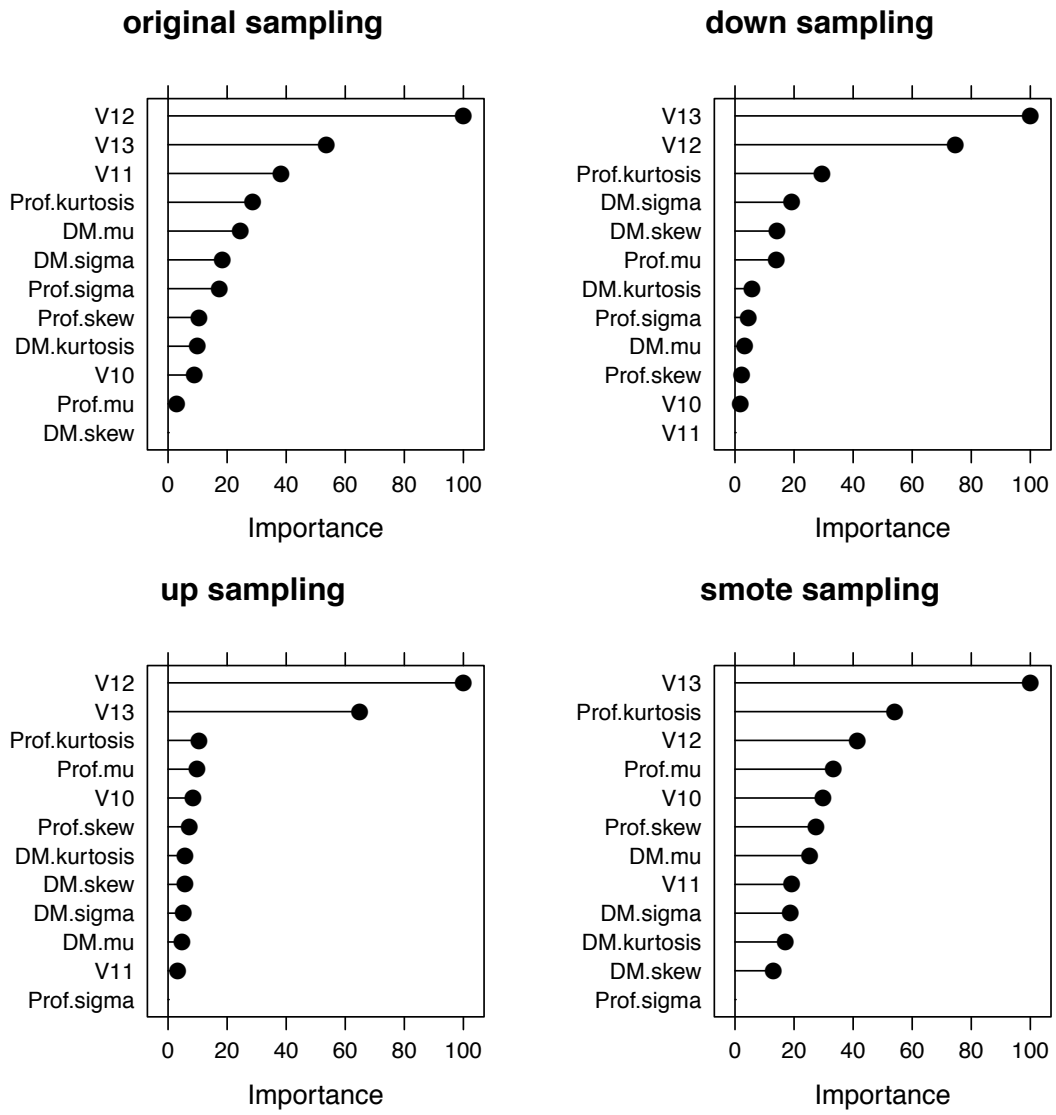


Figure F.6: Neural network ensemble model variable importance plots for downsampled data, 2 of 2

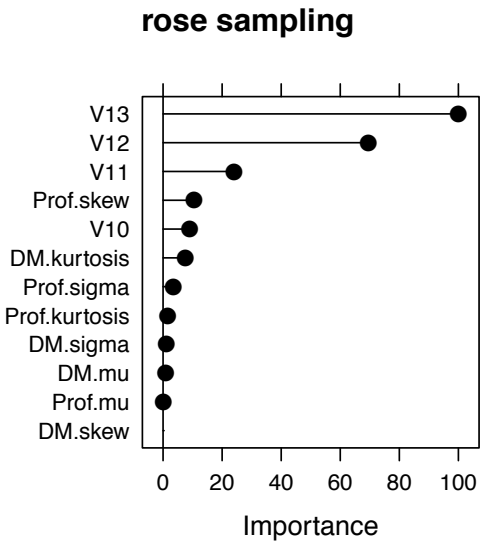


Figure F.7: Neural network ensemble model variable importance plots for upsampled data, 1 of 2

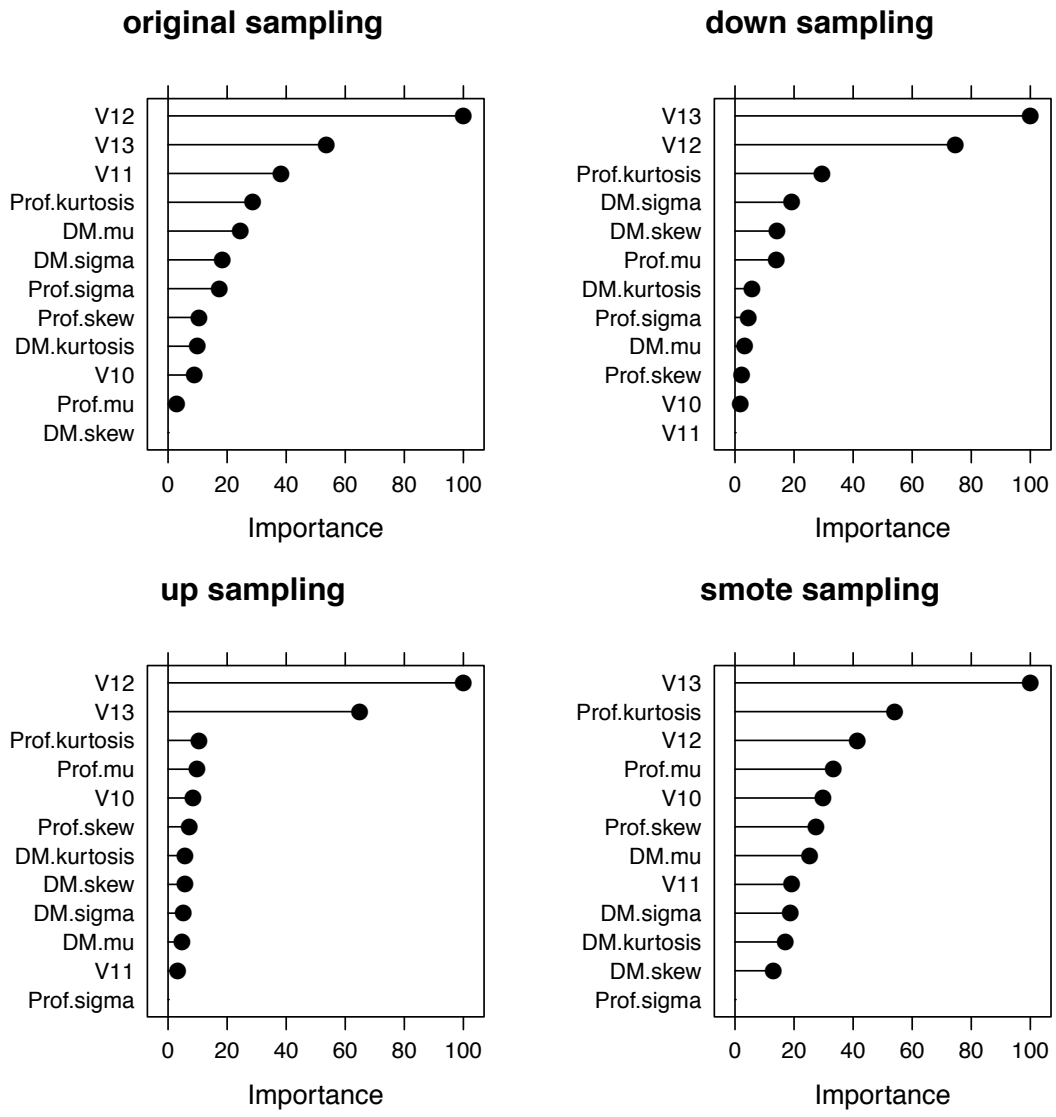


Figure F.8: Neural network ensemble model variable importance plots for upsampled data, 2 of 2

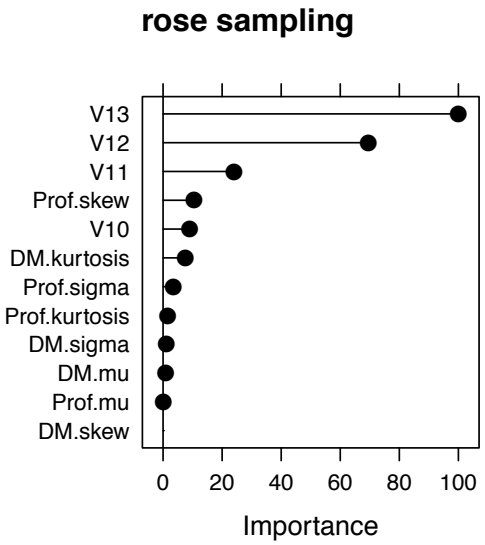


Figure F.9: Neural network ensemble model variable importance plots for SMOTE sampling, 1 of 2

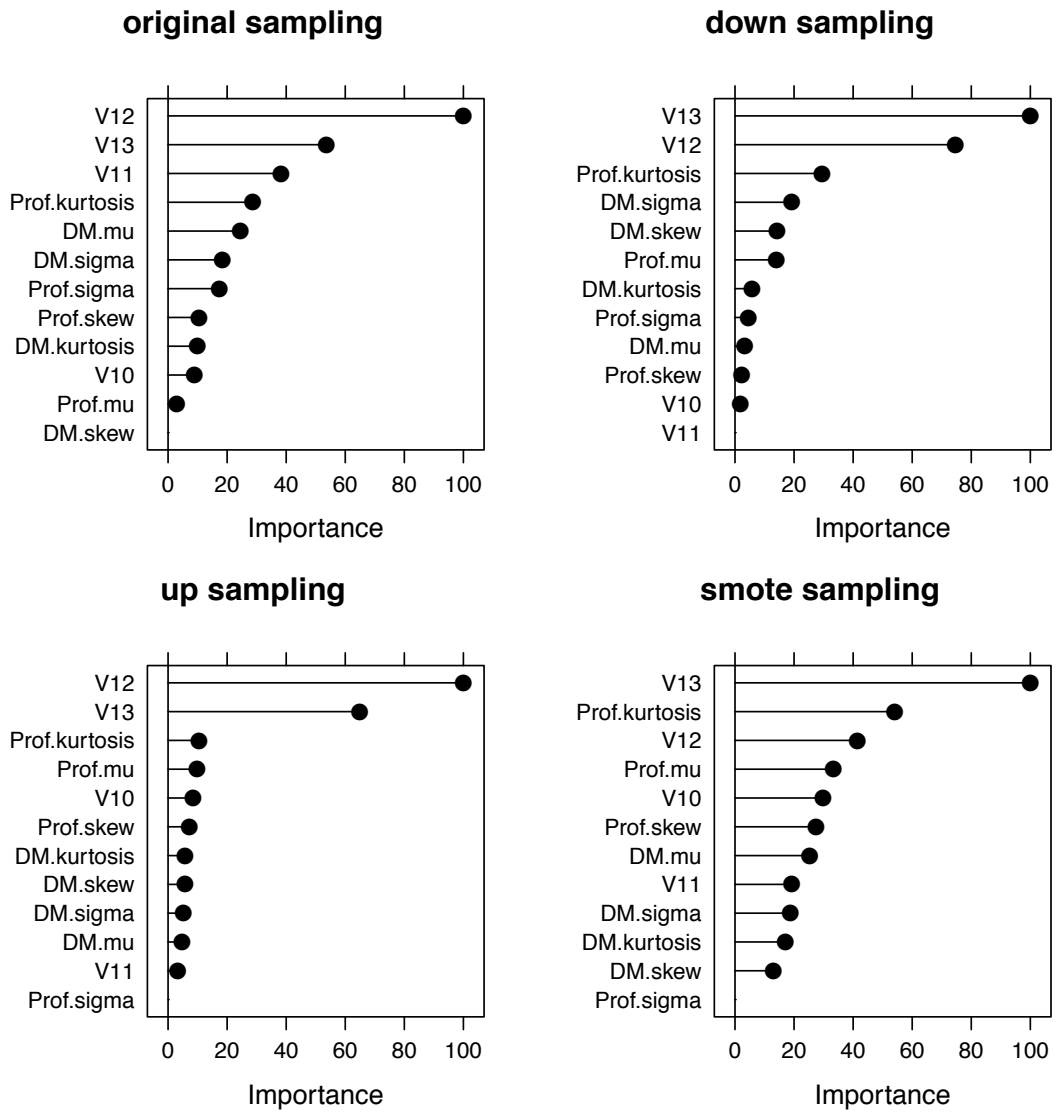


Figure F.10: Neural network ensemble model variable importance plots for SMOTE sampling, 2 of 2

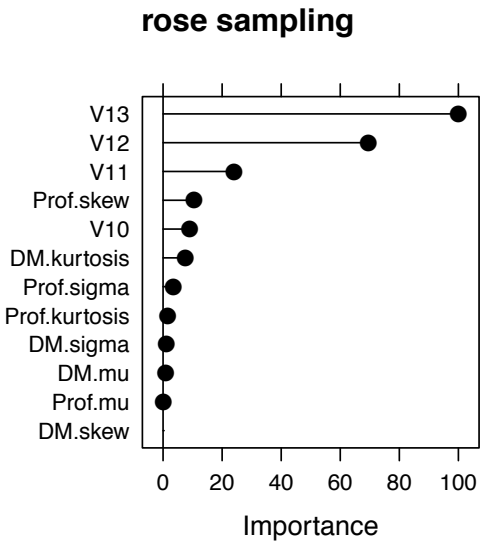


Figure F.11: Neural network ensemble model variable importance plots for ROSE sampling, 1 of 2

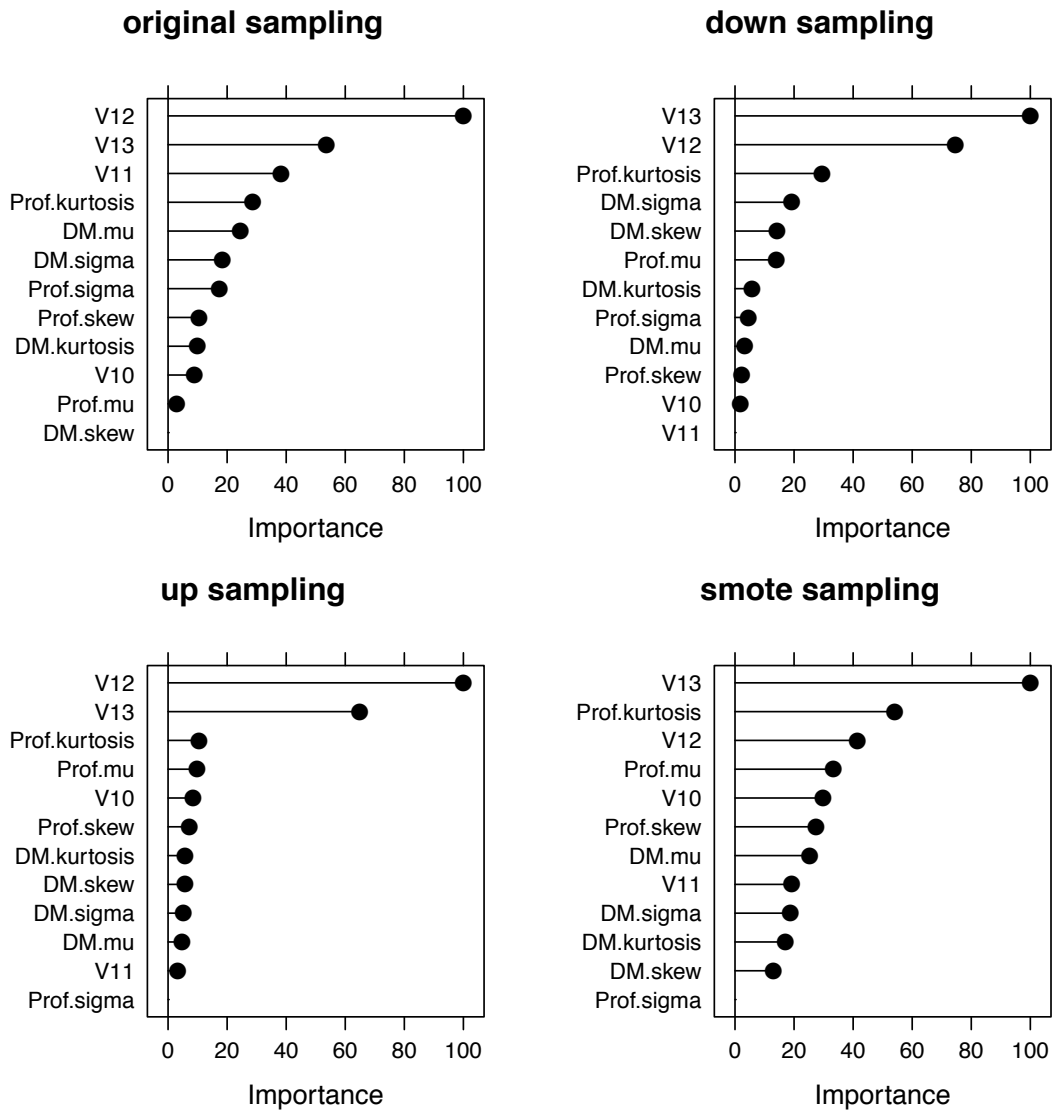
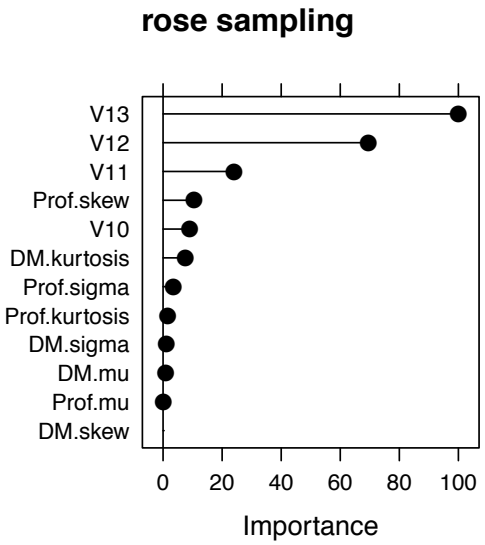


Figure F.12: Neural network ensemble model variable importance plots for ROSE sampling, 2 of 2



Neural Network Ensemble Data

```
Summary Text Output for the nnet model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       292       418
  nonpulsar      7     22081

               Accuracy : 0.9814
                 95% CI : (0.9795, 0.9831)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5709
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98142
         Pos Pred Value : 0.41127
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.03114
      Balanced Accuracy : 0.97901

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       292       340
  nonpulsar      7     22159

               Accuracy : 0.9848
                 95% CI : (0.9831, 0.9863)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.9971

                  Kappa : 0.6205
 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.97659
            Specificity : 0.98489
         Pos Pred Value : 0.46203
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02772
      Balanced Accuracy : 0.98074

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       292       305
  nonpulsar      7     22194

               Accuracy : 0.9863
                 95% CI : (0.9847, 0.9878)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 0.7851

                  Kappa : 0.6456
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98644
         Pos Pred Value : 0.48911
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02619
      Balanced Accuracy : 0.98152

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       292       393
  nonpulsar      7     22106
```

```
               Accuracy : 0.9825
                 95% CI : (0.9807, 0.9841)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5859
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98253
         Pos Pred Value : 0.42628
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.03005
      Balanced Accuracy : 0.97956

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       292       477
  nonpulsar      7     22022

               Accuracy : 0.9788
                 95% CI : (0.9768, 0.9806)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5381
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.97880
         Pos Pred Value : 0.37971
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.03373
      Balanced Accuracy : 0.97769
```

```
       'Positive' Class : pulsar

Neural Network

68394 samples
   12 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9533508  0.8668489  0.9998726
   1    0.0001000000  0.9784764  0.9277378  0.9997748
   1    0.0005623413  0.9948240  0.9868639  0.9995526
   1    0.0031622777  0.9979683  0.9928614  0.9998311
   1    0.0177827941  0.9965567  0.9544245  0.9999585
   1    0.1000000000  0.9971821  0.9733084  0.9999348
   3    0.0000000000  0.9971839  0.9844045  0.9999348
   3    0.0001000000  0.9983955  0.9910836  0.9999200
   3    0.0005623413  0.9982849  0.9888489  0.9998430
   3    0.0031622777  0.9979947  0.9792434  0.9998548
   3    0.0177827941  0.9981400  0.9888539  0.9998252
   3    0.1000000000  0.9991529  0.9897503  0.9999348
   5    0.0000000000  0.9971579  0.9886142  0.9998607
   5    0.0001000000  0.9969093  0.9872859  0.9999111
   5    0.0005623413  0.9977404  0.9888489  0.9998459
   5    0.0031622777  0.9973182  0.9897403  0.9999319
   5    0.0177827941  0.9990308  0.9910737  0.9999378
   5    0.1000000000  0.9996370  0.9933084  0.9999467
   7    0.0000000000  0.9970079  0.9866217  0.9998133
   7    0.0001000000  0.9969141  0.9874931  0.9998667
   7    0.0005623413  0.9973496  0.9832759  0.9998607
   7    0.0031622777  0.9980553  0.9881748  0.9998963
   7    0.0177827941  0.9984868  0.9899576  0.9999437
   7    0.1000000000  0.9995697  0.9921873  0.9999496
   9    0.0000000000  0.9963549  0.9870712  0.9998252
   9    0.0001000000  0.9970601  0.9845918  0.9997600
   9    0.0005623413  0.9971670  0.9863870  0.9998844
   9    0.0031622777  0.9983960  0.9890687  0.9998904
   9    0.0177827941  0.9986397  0.9897478  0.9999467
   9    0.1000000000  0.9996160  0.9924145  0.9999496
```

```
  11    0.0000000000  0.9961867  0.9863970  0.9998281
  11    0.0001000000  0.9970453  0.9854906  0.9998193
  11    0.0005623413  0.9972543  0.9879501  0.9998459
  11    0.0031622777  0.9982085  0.9875081  0.9999052
  11    0.0177827941  0.9988937  0.9892934  0.9999496
  11    0.1000000000  0.9997008  0.9912959  0.9999556

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 5 and decay = 0.1.
Neural Network

1794 samples
  12 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9987679  0.9973333  0.9988814
   1    0.0001000000  1.0000000  1.0000000  1.0000000
   1    0.0005623413  1.0000000  1.0000000  1.0000000
   1    0.0031622777  1.0000000  1.0000000  1.0000000
   1    0.0177827941  1.0000000  1.0000000  1.0000000
   1    0.1000000000  1.0000000  1.0000000  1.0000000
   3    0.0000000000  0.9996617  1.0000000  0.9988814
   3    0.0001000000  0.9993333  0.9988889  0.9997778
   3    0.0005623413  1.0000000  1.0000000  0.9995556
   3    0.0031622777  1.0000000  1.0000000  1.0000000
   3    0.0177827941  1.0000000  1.0000000  1.0000000
   3    0.1000000000  1.0000000  1.0000000  1.0000000
   5    0.0000000000  0.9998642  0.9997753  0.9995556
   5    0.0001000000  1.0000000  1.0000000  0.9997778
   5    0.0005623413  1.0000000  1.0000000  1.0000000
   5    0.0031622777  1.0000000  1.0000000  1.0000000
   5    0.0177827941  1.0000000  1.0000000  1.0000000
   5    0.1000000000  1.0000000  1.0000000  1.0000000
   7    0.0000000000  1.0000000  0.9997778  0.9995556
   7    0.0001000000  1.0000000  1.0000000  0.9997778
   7    0.0005623413  1.0000000  0.9997778  1.0000000
   7    0.0031622777  1.0000000  1.0000000  1.0000000
   7    0.0177827941  1.0000000  1.0000000  1.0000000
```

```
   7    0.1000000000  1.0000000  1.0000000  1.0000000
   9    0.0000000000  0.9999975  1.0000000  0.9995556
   9    0.0001000000  1.0000000  1.0000000  1.0000000
   9    0.0005623413  1.0000000  1.0000000  1.0000000
   9    0.0031622777  1.0000000  1.0000000  1.0000000
   9    0.0177827941  1.0000000  1.0000000  1.0000000
   9    0.1000000000  1.0000000  1.0000000  1.0000000
  11    0.0000000000  1.0000000  1.0000000  1.0000000
  11    0.0001000000  1.0000000  1.0000000  0.9997778
  11    0.0005623413  1.0000000  1.0000000  1.0000000
  11    0.0031622777  1.0000000  1.0000000  1.0000000
  11    0.0177827941  1.0000000  1.0000000  1.0000000
  11    0.1000000000  1.0000000  1.0000000  1.0000000

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 1 and decay = 0.1.
Neural Network

134994 samples
    12 predictor
     2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494, 121496,
121495, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9990076  0.9961688  0.9992918
   1    0.0001000000  0.9992293  0.9990074  0.9993363
   1    0.0005623413  0.9998931  0.9989481  0.9998430
   1    0.0031622777  0.9999789  1.0000000  0.9999674
   1    0.0177827941  0.9999091  0.9992296  0.9997363
   1    0.1000000000  0.9999936  1.0000000  0.9999585
   3    0.0000000000  0.9996558  0.9991881  0.9999348
   3    0.0001000000  0.9999844  1.0000000  0.9999644
   3    0.0005623413  0.9981963  0.9964592  0.9999170
   3    0.0031622777  0.9999916  1.0000000  0.9999467
   3    0.0177827941  0.9999725  0.9999615  0.9999556
   3    0.1000000000  0.9999889  0.9998015  0.9999170
   5    0.0000000000  0.9999777  1.0000000  0.9999437
   5    0.0001000000  0.9999873  1.0000000  0.9999467
   5    0.0005623413  0.9999902  1.0000000  0.9999644
```

```
   5    0.0031622777  0.9999879  1.0000000  0.9999556
   5    0.0177827941  0.9999918  1.0000000  0.9999526
   5    0.1000000000  0.9999910  1.0000000  0.9999674
   7    0.0000000000  0.9999851  1.0000000  0.9999615
   7    0.0001000000  0.9999862  1.0000000  0.9999437
   7    0.0005623413  0.9999829  1.0000000  0.9999378
   7    0.0031622777  0.9999905  1.0000000  0.9999496
   7    0.0177827941  0.9999905  1.0000000  0.9999556
   7    0.1000000000  0.9999924  1.0000000  0.9999674
   9    0.0000000000  0.9999851  1.0000000  0.9999615
   9    0.0001000000  0.9999919  1.0000000  0.9999585
   9    0.0005623413  0.9999903  0.9999733  0.9999467
   9    0.0031622777  0.9999902  1.0000000  0.9999378
   9    0.0177827941  0.9999893  1.0000000  0.9999556
   9    0.1000000000  0.9999937  1.0000000  0.9999644
  11    0.0000000000  0.9999837  1.0000000  0.9999496
  11    0.0001000000  0.9999855  1.0000000  0.9999467
  11    0.0005623413  0.9999913  1.0000000  0.9999615
  11    0.0031622777  0.9999909  1.0000000  0.9999289
  11    0.0177827941  0.9999883  1.0000000  0.9999496
  11    0.1000000000  0.9999925  1.0000000  0.9999674

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 1 and decay = 0.1.
Neural Network

6279 samples
  12 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9977740  0.9953997  0.9999443
   1    0.0001000000  1.0000000  0.9996288  1.0000000
   1    0.0005623413  1.0000000  0.9994804  1.0000000
   1    0.0031622777  1.0000000  0.9995547  1.0000000
   1    0.0177827941  1.0000000  0.9995547  1.0000000
   1    0.1000000000  1.0000000  0.9995547  1.0000000
   3    0.0000000000  0.9999718  0.9998519  0.9998327
   3    0.0001000000  0.9999360  0.9995547  0.9997772
```

```
   3    0.0005623413  0.9997314  0.9977695  0.9997772
   3    0.0031622777  0.9999998  0.9996288  1.0000000
   3    0.0177827941  1.0000000  0.9995547  1.0000000
   3    0.1000000000  1.0000000  0.9995547  1.0000000
   5    0.0000000000  1.0000000  0.9996291  0.9998884
   5    0.0001000000  1.0000000  0.9997775  1.0000000
   5    0.0005623413  0.9999998  0.9996288  0.9999443
   5    0.0031622777  1.0000000  0.9996291  1.0000000
   5    0.0177827941  1.0000000  0.9995547  1.0000000
   5    0.1000000000  1.0000000  0.9995547  1.0000000
   7    0.0000000000  1.0000000  0.9997772  0.9999443
   7    0.0001000000  0.9999959  0.9997772  0.9998884
   7    0.0005623413  1.0000000  0.9999259  1.0000000
   7    0.0031622777  0.9999998  0.9996288  0.9999443
   7    0.0177827941  1.0000000  0.9995547  1.0000000
   7    0.1000000000  1.0000000  0.9995547  1.0000000
   9    0.0000000000  0.9999994  0.9998516  0.9999443
   9    0.0001000000  0.9999996  0.9996291  1.0000000
   9    0.0005623413  0.9999994  0.9996291  0.9999443
   9    0.0031622777  1.0000000  0.9996288  1.0000000
   9    0.0177827941  1.0000000  0.9995547  1.0000000
   9    0.1000000000  0.9998939  0.9990343  0.9998329
  11    0.0000000000  0.9999996  0.9998519  0.9998886
  11    0.0001000000  0.9999557  0.9984387  0.9997772
  11    0.0005623413  1.0000000  0.9997772  1.0000000
  11    0.0031622777  1.0000000  0.9997034  0.9999441
  11    0.0177827941  1.0000000  0.9995547  1.0000000
  11    0.1000000000  1.0000000  0.9995547  1.0000000

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 7 and decay = 0.0005623413.
Neural Network

68394 samples
   12 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
Resampling results across tuning parameters:

  size  decay         ROC        Sens       Spec
   1    0.0000000000  0.9998592  0.9998610  0.9997460
```

```
  1    0.0001000000  0.9988801  0.9990676  0.9973833
  1    0.0005623413  0.9999174  0.9998436  0.9998228
  1    0.0031622777  0.9999119  0.9998784  0.9998287
  1    0.0177827941  0.9999105  0.9998900  0.9998228
  1    0.1000000000  0.9999304  0.9999073  0.9998228
  3    0.0000000000  0.9998821  0.9998552  0.9997992
  3    0.0001000000  0.9998930  0.9998900  0.9998110
  3    0.0005623413  0.9999166  0.9998378  0.9998287
  3    0.0031622777  0.9999243  0.9999073  0.9998110
  3    0.0177827941  0.9999284  0.9998958  0.9998110
  3    0.1000000000  0.9999351  0.9998958  0.9998169
  5    0.0000000000  0.9998598  0.9997857  0.9997933
  5    0.0001000000  0.9998983  0.9997626  0.9997637
  5    0.0005623413  0.9999084  0.9998089  0.9997815
  5    0.0031622777  0.9999094  0.9998842  0.9997815
  5    0.0177827941  0.9999296  0.9998668  0.9998169
  5    0.1000000000  0.9999357  0.9998900  0.9998169
  7    0.0000000000  0.9998203  0.9996757  0.9997401
  7    0.0001000000  0.9998943  0.9997162  0.9997165
  7    0.0005623413  0.9998993  0.9997394  0.9997106
  7    0.0031622777  0.9999013  0.9998031  0.9997283
  7    0.0177827941  0.9999313  0.9998784  0.9998228
  7    0.1000000000  0.9999449  0.9998784  0.9998287
  9    0.0000000000  0.9998213  0.9996467  0.9996456
  9    0.0001000000  0.9998874  0.9996931  0.9997460
  9    0.0005623413  0.9999074  0.9997568  0.9996751
  9    0.0031622777  0.9999059  0.9998089  0.9997460
  9    0.0177827941  0.9999233  0.9998900  0.9998169
  9    0.1000000000  0.9999377  0.9999015  0.9998169
 11    0.0000000000  0.9998261  0.9995599  0.9996751
 11    0.0001000000  0.9998909  0.9996931  0.9996338
 11    0.0005623413  0.9999028  0.9997683  0.9996929
 11    0.0031622777  0.9999030  0.9998089  0.9997637
 11    0.0177827941  0.9999400  0.9998900  0.9998169
 11    0.1000000000  0.9999487  0.9998900  0.9998228

Sens was used to select the optimal model using  the largest value.
The final values used for the model were size = 1 and decay = 0.1.

Call:
summary.diff.resamples(object = results$modelDiff[[name]])

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
```

```
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down       up         smote      rose
original           -3.630e-04 -3.566e-04 -3.630e-04 -2.933e-04
down     0.0029272              6.395e-06  0.000e+00  6.963e-05
up       0.0038669 0.2140743              -6.395e-06  6.323e-05
smote    0.0029272 NA         0.2140743               6.963e-05
rose     0.0272250 0.0009573  0.0046140  0.0009573

Sens
         original  down       up         smote      rose
original           -6.692e-03 -6.692e-03 -6.618e-03 -6.599e-03
down     1.707e-05              0.000e+00  7.407e-05  9.266e-05
up       1.707e-05 NA                      7.407e-05  9.266e-05
smote    1.299e-05 1.000000   1.000000                1.859e-05
rose     2.210e-05 0.003035   0.003035   1.000000

Spec
         original  down       up         smote      rose
original           -5.334e-05 -1.185e-05 -5.334e-05  1.239e-04
down     0.0008393              4.148e-05  0.000e+00  1.772e-04
up       1.0000000 0.0019986              -4.148e-05  1.357e-04
smote    0.0008393 NA         0.0019986               1.772e-04
rose     0.0090729 3.923e-06  0.0010369  3.923e-06

$original
nnet variable importance

              Overall
V12           100.000
V13            53.647
V11            38.276
Prof.kurtosis  28.714
DM.mu          24.468
DM.sigma       18.378
Prof.sigma     17.381
Prof.skew      10.524
DM.kurtosis     9.935
V10             8.948
Prof.mu         2.989
DM.skew         0.000

$down
```

```
nnet variable importance

              Overall
V13           100.000
V12            74.612
Prof.kurtosis  29.488
DM.sigma       19.267
DM.skew        14.278
Prof.mu        13.966
DM.kurtosis     5.811
Prof.sigma      4.518
DM.mu           3.308
Prof.skew       2.292
V10             1.798
V11             0.000

$up
nnet variable importance

              Overall
V12           100.000
V13            64.940
Prof.kurtosis  10.459
Prof.mu         9.831
V10             8.528
Prof.skew       7.188
DM.kurtosis     5.734
DM.skew         5.727
DM.sigma        5.190
DM.mu           4.776
V11             3.250
Prof.sigma      0.000

$smote
nnet variable importance

              Overall
V13            100.00
Prof.kurtosis   54.04
V12             41.43
Prof.mu         33.31
V10             29.78
Prof.skew       27.40
DM.mu           25.26
```

```
V11             19.17
DM.sigma        18.81
DM.kurtosis     17.03
DM.skew         13.01
Prof.sigma       0.00

$rose
nnet variable importance

               Overall
V13           100.0000
V12            69.4941
V11            24.0271
Prof.skew      10.5322
V10             9.0635
DM.kurtosis     7.5855
Prof.sigma      3.5201
Prof.kurtosis   1.6055
DM.sigma        1.1557
DM.mu           0.8963
Prof.mu         0.1537
DM.skew         0.0000


Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50

ROC
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9971  0.9995      1 0.9996       1    1    0
down     1.0000  1.0000      1 1.0000       1    1    0
up       0.9999  1.0000      1 1.0000       1    1    0
smote    1.0000  1.0000      1 1.0000       1    1    0
rose     0.9995  0.9999      1 0.9999       1    1    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9663  0.9889      1 0.9933       1    1    0
down     1.0000  1.0000      1 1.0000       1    1    0
up       1.0000  1.0000      1 1.0000       1    1    0
smote    0.9963  1.0000      1 0.9999       1    1    0
```

```
rose     0.9994  0.9998      1 0.9999       1    1    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9997  0.9999      1 0.9999       1    1    0
down     1.0000  1.0000      1 1.0000       1    1    0
up       0.9997  0.9999      1 1.0000       1    1    0
smote    1.0000  1.0000      1 1.0000       1    1    0
rose     0.9994  0.9997      1 0.9998       1    1    0
```

C5.0 Ensemble Plots

This section contains the plots for the C5.0 ensemble training and evaluation process.

Figure F.13: C5.0 ensemble model dot plot

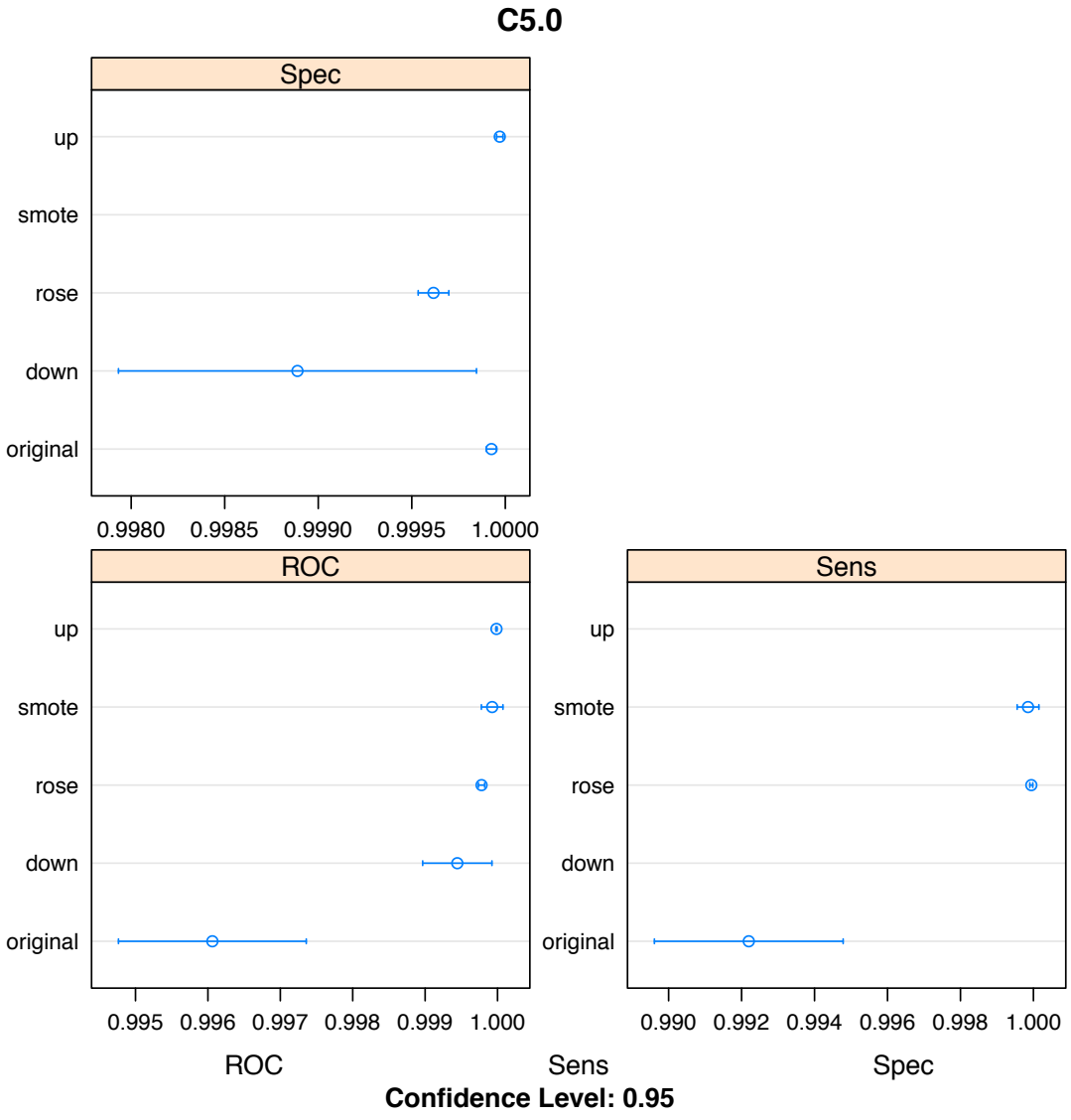


Figure F.14: C5.0 ensemble model box and whiskers plot

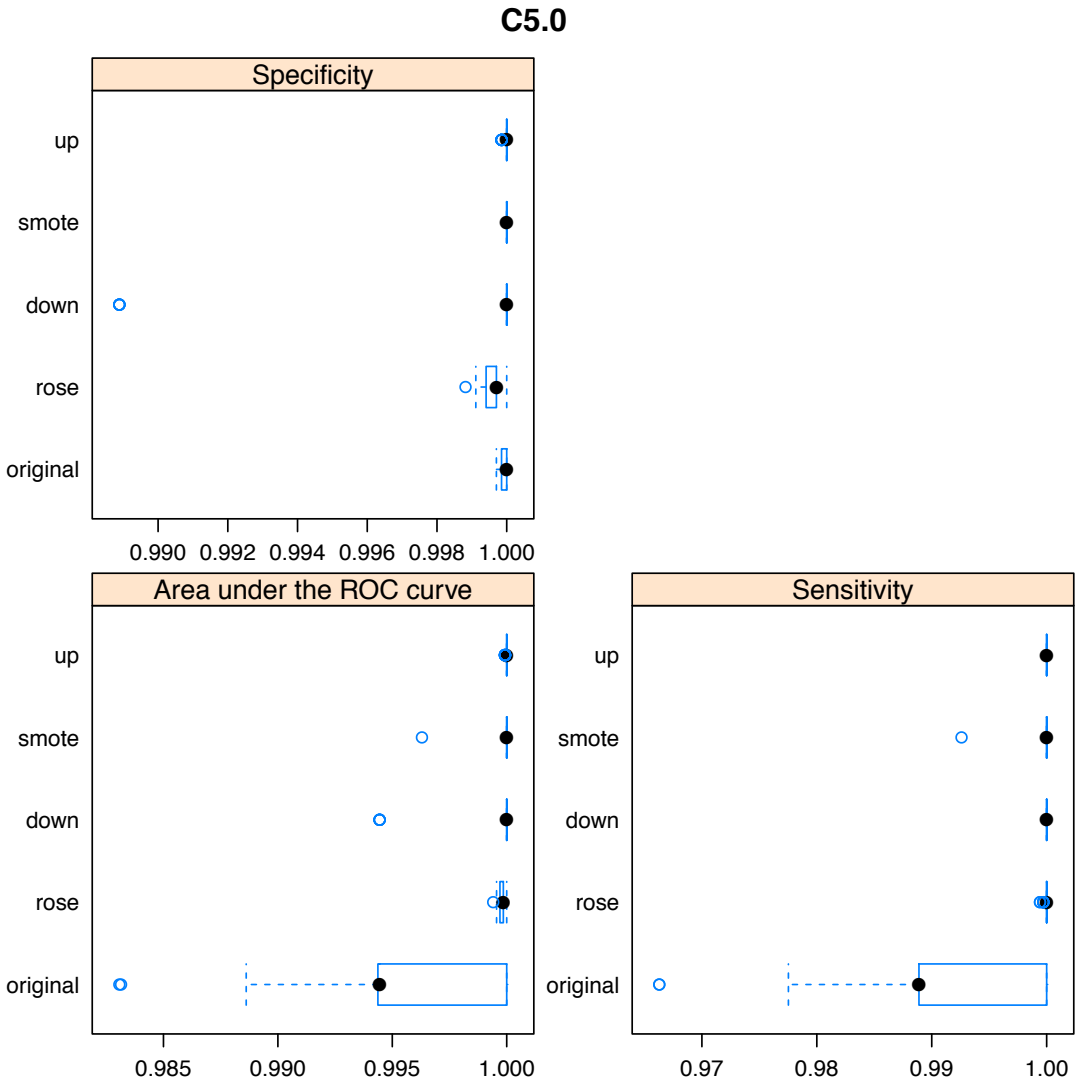


Figure F.15: C5.0 ensemble model variable importance plots for original sample set

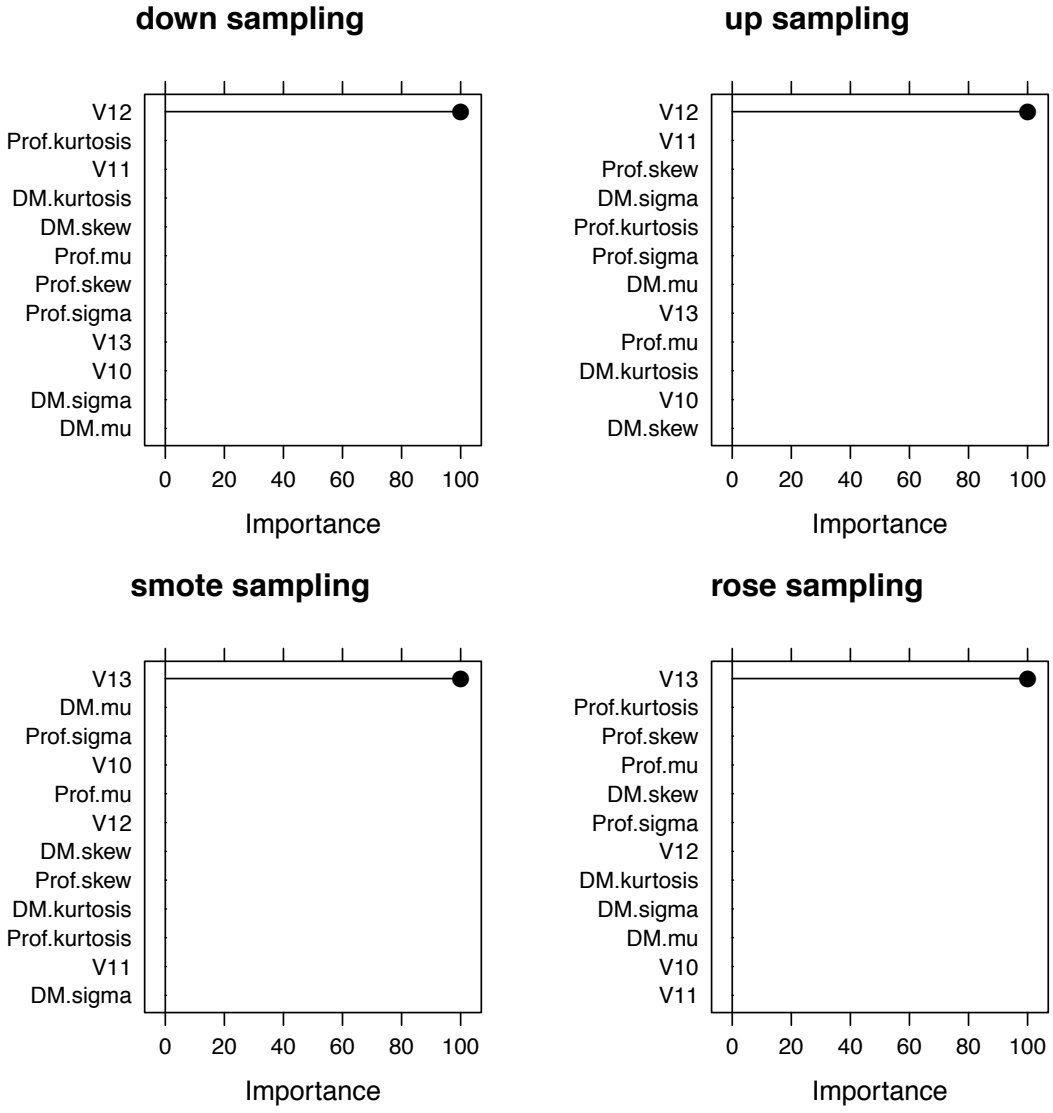


Figure F.16: C5.0 ensemble model variable importance plots for downsampled data

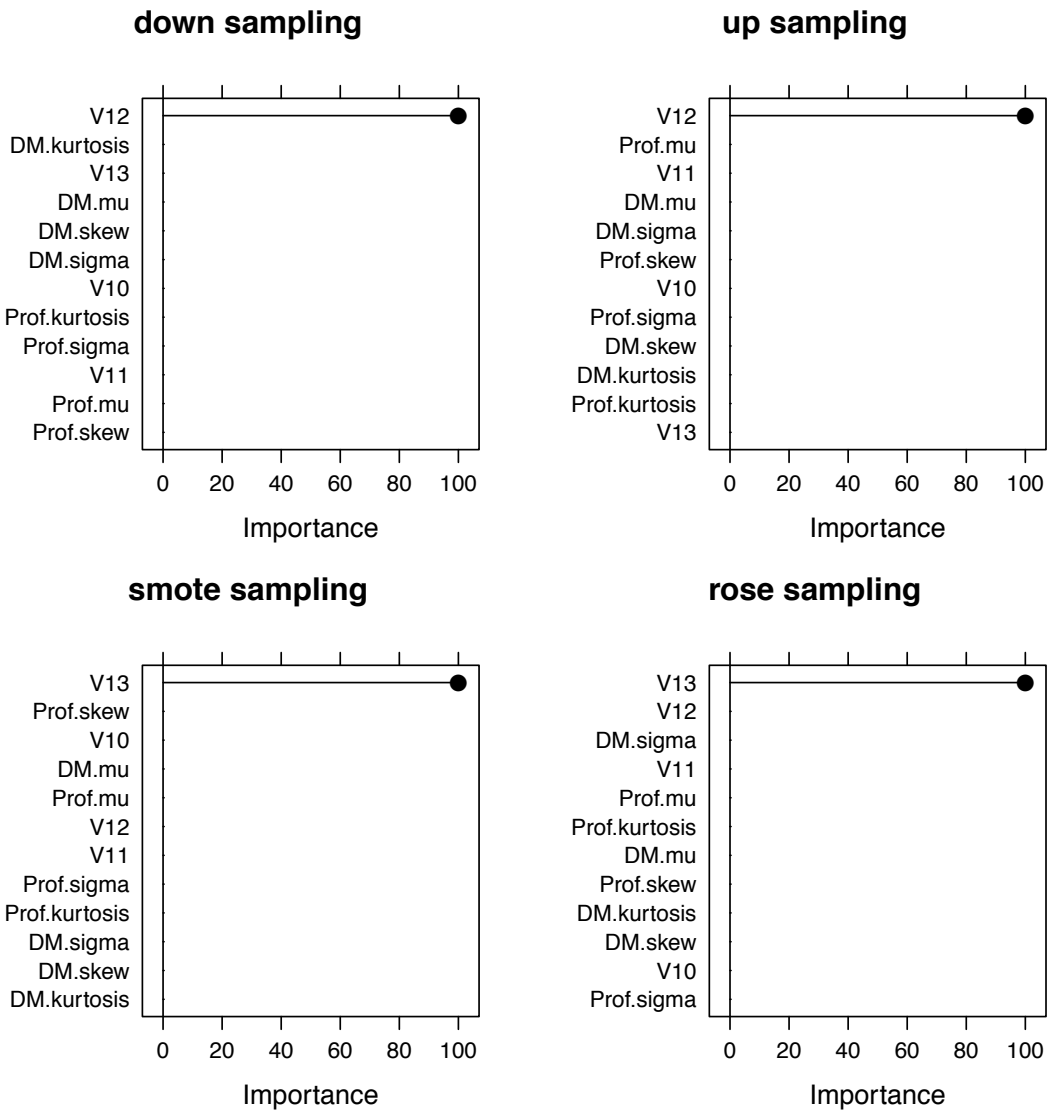


Figure F.17: C5.0 ensemble model variable importance plots for upsampled data

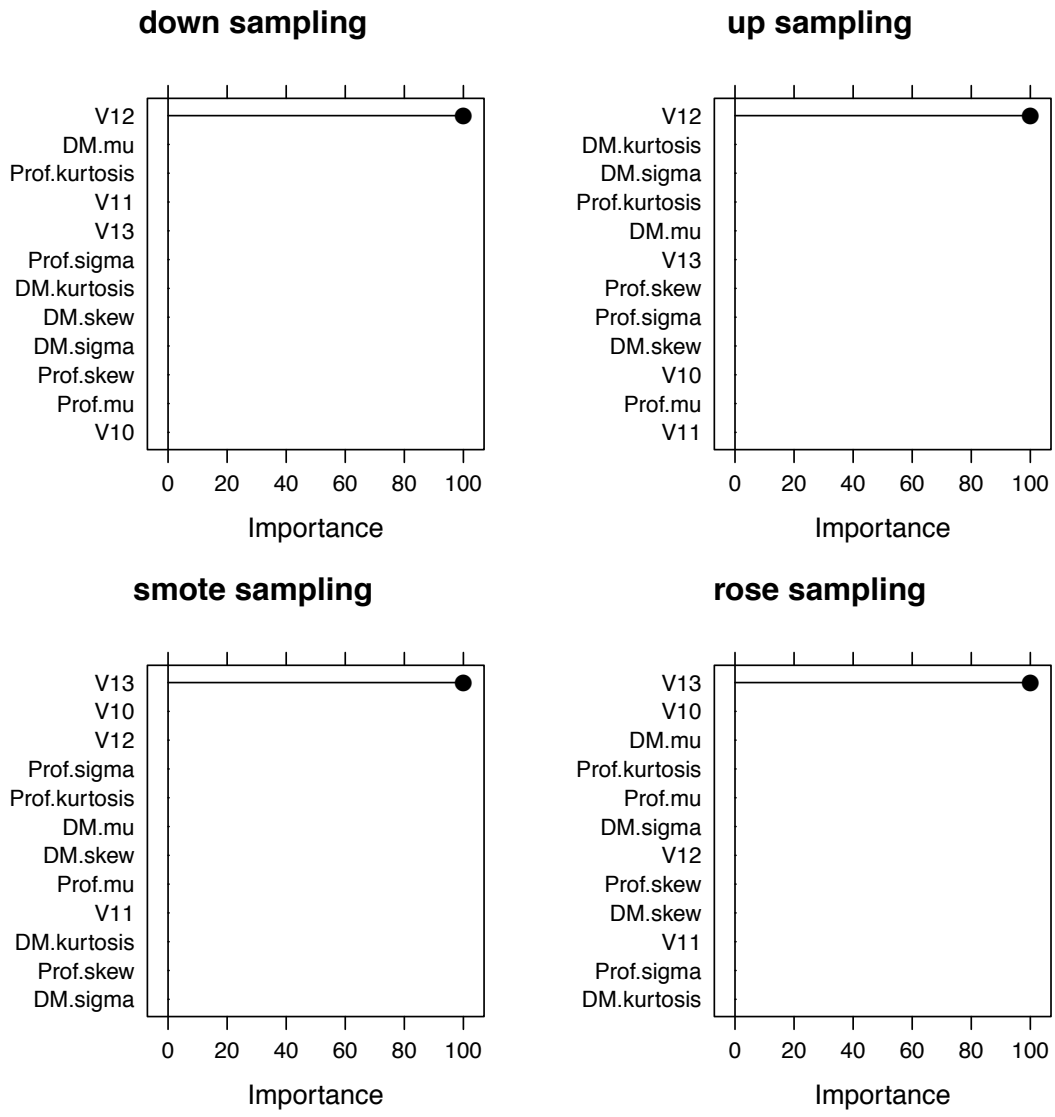


Figure F.18: C5.0 ensemble model variable importance plots for SMOTE sampling

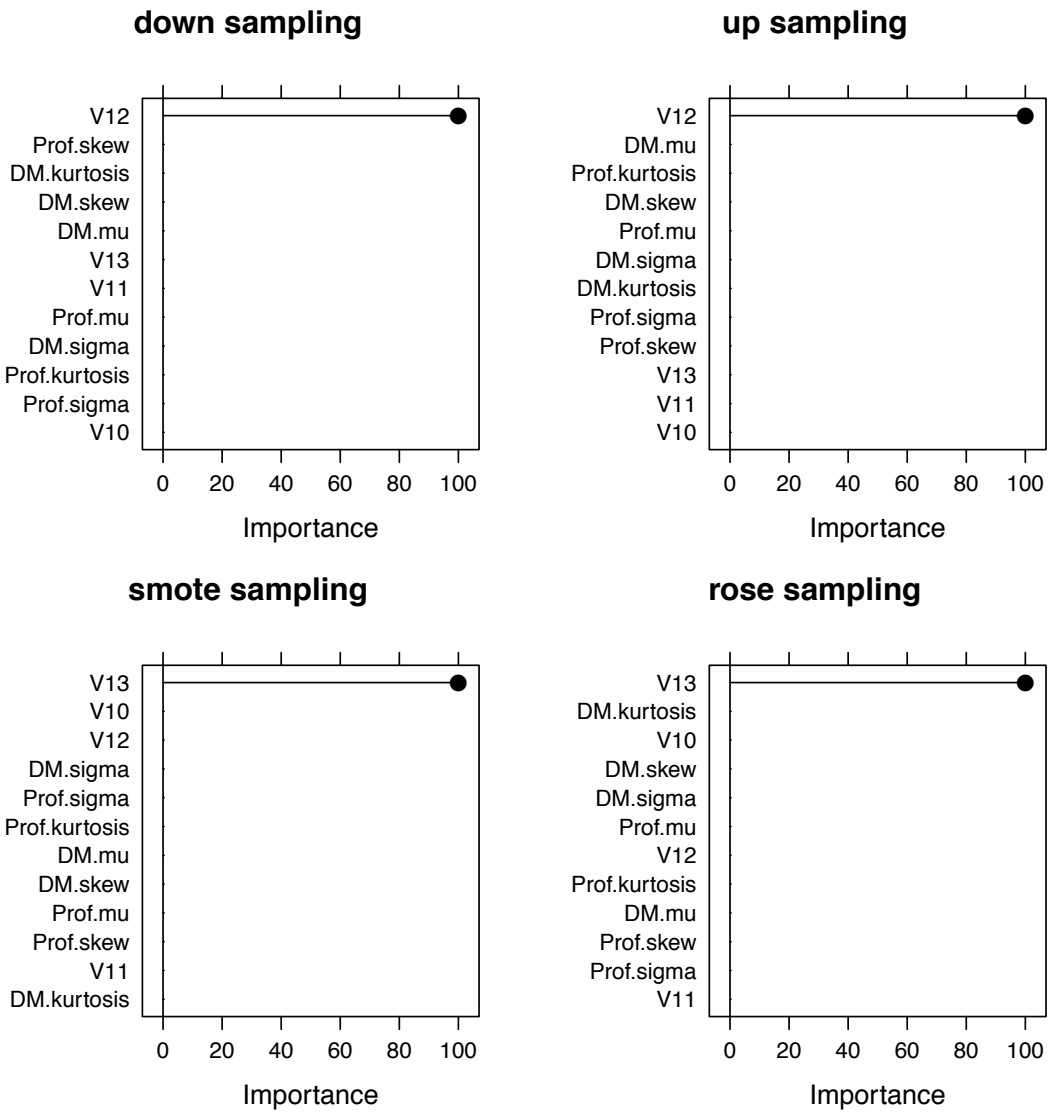
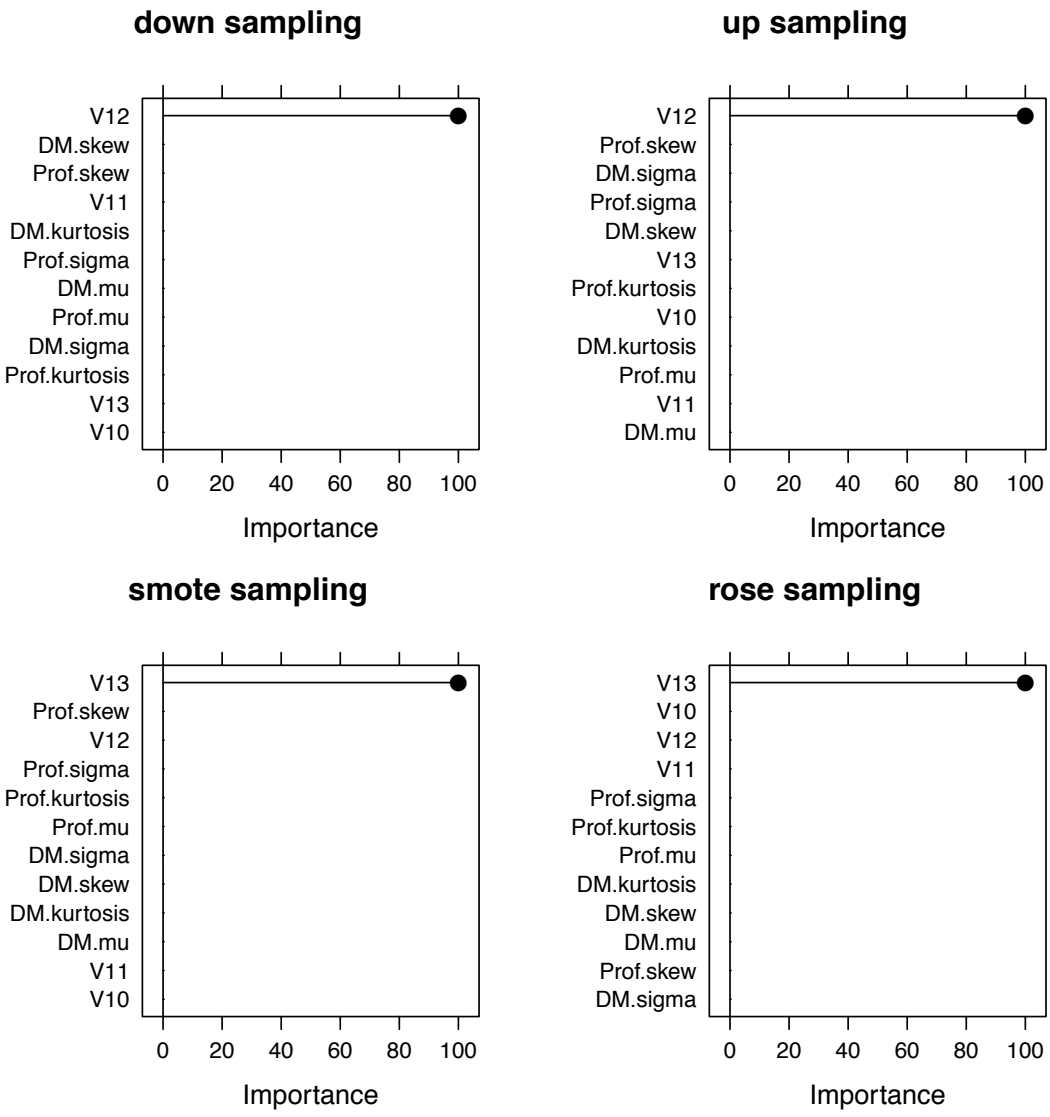


Figure F.19: C5.0 ensemble model variable importance plots for ROSE sampling



C5.0 Ensemble Data

```
Summary Text Output for the C5.0 model

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       292       385
  nonpulsar      7     22114

               Accuracy : 0.9828
                 95% CI : (0.981, 0.9845)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5909
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98289
         Pos Pred Value : 0.43131
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02970
      Balanced Accuracy : 0.97974

       'Positive' Class : pulsar

Confusion Matrix and Statistics

          Reference
Prediction  pulsar nonpulsar
  pulsar       294       421
  nonpulsar      5     22078

               Accuracy : 0.9813
                 95% CI : (0.9795, 0.983)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.572
 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.98328
            Specificity : 0.98129
         Pos Pred Value : 0.41119
         Neg Pred Value : 0.99977
             Prevalence : 0.01312
         Detection Rate : 0.01290
   Detection Prevalence : 0.03136
      Balanced Accuracy : 0.98228

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       294       421
  nonpulsar      5     22078

               Accuracy : 0.9813
                 95% CI : (0.9795, 0.983)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.572
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98328
            Specificity : 0.98129
         Pos Pred Value : 0.41119
         Neg Pred Value : 0.99977
             Prevalence : 0.01312
         Detection Rate : 0.01290
   Detection Prevalence : 0.03136
      Balanced Accuracy : 0.98228

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       292       387
  nonpulsar      7     22112
```

```
               Accuracy : 0.9827
                 95% CI : (0.9809, 0.9844)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5897
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98280
         Pos Pred Value : 0.43004
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02978
      Balanced Accuracy : 0.97969

       'Positive' Class : pulsar

Confusion Matrix and Statistics

           Reference
Prediction  pulsar nonpulsar
  pulsar       292       387
  nonpulsar      7     22112

               Accuracy : 0.9827
                 95% CI : (0.9809, 0.9844)
    No Information Rate : 0.9869
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5897
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97659
            Specificity : 0.98280
         Pos Pred Value : 0.43004
         Neg Pred Value : 0.99968
             Prevalence : 0.01312
         Detection Rate : 0.01281
   Detection Prevalence : 0.02978
      Balanced Accuracy : 0.97969
```

```
       'Positive' Class : pulsar

C5.0

68394 samples
   12 predictor
    2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61556, 61555, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9960614  0.9921973  0.9999259
  rules  FALSE   10      0.9989588  0.9750262  0.9999081
  rules  FALSE   20      0.9991855  0.9801448  0.9999081
  rules  FALSE   30      0.9992255  0.9826167  0.9999111
  rules  FALSE   40      0.9992660  0.9850537  0.9999111
  rules  FALSE   50      0.9992953  0.9875181  0.9999141
  rules   TRUE    1      0.9960615  0.9921973  0.9999259
  rules   TRUE   10      0.9960646  0.9921973  0.9999289
  rules   TRUE   20      0.9960646  0.9921973  0.9999289
  rules   TRUE   30      0.9960646  0.9921973  0.9999289
  rules   TRUE   40      0.9960646  0.9921973  0.9999289
  rules   TRUE   50      0.9960646  0.9921973  0.9999289
  tree   FALSE    1      0.9960612  0.9921973  0.9999259
  tree   FALSE   10      0.9988073  0.9817428  0.9998963
  tree   FALSE   20      0.9992187  0.9868564  0.9999170
  tree   FALSE   30      0.9992556  0.9872959  0.9999170
  tree   FALSE   40      0.9992888  0.9870762  0.9999141
  tree   FALSE   50      0.9993256  0.9886367  0.9999170
  tree    TRUE    1      0.9960615  0.9921973  0.9999259
  tree    TRUE   10      0.9960646  0.9921973  0.9999289
  tree    TRUE   20      0.9960646  0.9921973  0.9999289
  tree    TRUE   30      0.9960646  0.9921973  0.9999289
  tree    TRUE   40      0.9960646  0.9921973  0.9999289
  tree    TRUE   50      0.9960646  0.9921973  0.9999289

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules
and winnow = TRUE.
C5.0
```

```
1794 samples
  12 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1614, 1616, 1615, 1615, 1615, 1614, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens  Spec
  rules  FALSE    1      0.9994444  1     0.9988889
  rules  FALSE   10      0.9994444  1     0.9988889
  rules  FALSE   20      0.9994444  1     0.9988889
  rules  FALSE   30      0.9994444  1     0.9988889
  rules  FALSE   40      0.9994444  1     0.9988889
  rules  FALSE   50      0.9994444  1     0.9988889
  rules   TRUE    1      0.9994444  1     0.9988889
  rules   TRUE   10      0.9994444  1     0.9988889
  rules   TRUE   20      0.9994444  1     0.9988889
  rules   TRUE   30      0.9994444  1     0.9988889
  rules   TRUE   40      0.9994444  1     0.9988889
  rules   TRUE   50      0.9994444  1     0.9988889
  tree   FALSE    1      0.9994444  1     0.9988889
  tree   FALSE   10      0.9994444  1     0.9988889
  tree   FALSE   20      0.9994444  1     0.9988889
  tree   FALSE   30      0.9994444  1     0.9988889
  tree   FALSE   40      0.9994444  1     0.9988889
  tree   FALSE   50      0.9994444  1     0.9988889
  tree    TRUE    1      0.9994444  1     0.9988889
  tree    TRUE   10      0.9994444  1     0.9988889
  tree    TRUE   20      0.9994444  1     0.9988889
  tree    TRUE   30      0.9994444  1     0.9988889
  tree    TRUE   40      0.9994444  1     0.9988889
  tree    TRUE   50      0.9994444  1     0.9988889

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules
and winnow = TRUE.
C5.0

134994 samples
    12 predictor
     2 classes: 'pulsar', 'nonpulsar'
```

```
Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 121495, 121495, 121494, 121494, 121496,
121495, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens  Spec
  rules  FALSE    1      0.9999852  1     0.9999704
  rules  FALSE   10      0.9999899  1     0.9998222
  rules  FALSE   20      0.9999895  1     0.9998222
  rules  FALSE   30      0.9999896  1     0.9998222
  rules  FALSE   40      0.9999895  1     0.9998222
  rules  FALSE   50      0.9999899  1     0.9998222
  rules   TRUE    1      0.9999852  1     0.9999704
  rules   TRUE   10      0.9999852  1     0.9999704
  rules   TRUE   20      0.9999852  1     0.9999704
  rules   TRUE   30      0.9999852  1     0.9999704
  rules   TRUE   40      0.9999852  1     0.9999704
  rules   TRUE   50      0.9999852  1     0.9999704
  tree   FALSE    1      0.9999852  1     0.9999704
  tree   FALSE   10      0.9999900  1     0.9998222
  tree   FALSE   20      0.9999895  1     0.9998222
  tree   FALSE   30      0.9999896  1     0.9998222
  tree   FALSE   40      0.9999895  1     0.9998222
  tree   FALSE   50      0.9999899  1     0.9998222
  tree    TRUE    1      0.9999852  1     0.9999704
  tree    TRUE   10      0.9999852  1     0.9999704
  tree    TRUE   20      0.9999852  1     0.9999704
  tree    TRUE   30      0.9999852  1     0.9999704
  tree    TRUE   40      0.9999852  1     0.9999704
  tree    TRUE   50      0.9999852  1     0.9999704

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules
and winnow = TRUE.
C5.0

6279 samples
  12 predictor
   2 classes: 'pulsar', 'nonpulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 5651, 5652, 5651, 5651, 5652, 5651, ...
```

```
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9999259  0.9998519  1
  rules  FALSE   10      0.9999259  0.9998519  1
  rules  FALSE   20      0.9999259  0.9998519  1
  rules  FALSE   30      0.9999259  0.9998519  1
  rules  FALSE   40      0.9999259  0.9998519  1
  rules  FALSE   50      0.9999259  0.9998519  1
  rules   TRUE    1      0.9998516  0.9997032  1
  rules   TRUE   10      0.9998516  0.9997032  1
  rules   TRUE   20      0.9998516  0.9997032  1
  rules   TRUE   30      0.9998516  0.9997032  1
  rules   TRUE   40      0.9998516  0.9997032  1
  rules   TRUE   50      0.9998516  0.9997032  1
  tree   FALSE    1      0.9999259  0.9998519  1
  tree   FALSE   10      0.9999259  0.9998519  1
  tree   FALSE   20      0.9999259  0.9998519  1
  tree   FALSE   30      0.9999259  0.9998519  1
  tree   FALSE   40      0.9999259  0.9998519  1
  tree   FALSE   50      0.9999259  0.9998519  1
  tree    TRUE    1      0.9998516  0.9997032  1
  tree    TRUE   10      0.9998516  0.9997032  1
  tree    TRUE   20      0.9998516  0.9997032  1
  tree    TRUE   30      0.9998516  0.9997032  1
  tree    TRUE   40      0.9998516  0.9997032  1
  tree    TRUE   50      0.9998516  0.9997032  1

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules
and winnow = FALSE.
C5.0

68394 samples
   12 predictor
    2 classes: 'nonpulsar', 'pulsar'

Pre-processing: scaled (8), centered (8), ignore (4)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 61555, 61555, 61554, 61554, 61555, 61555, ...
Resampling results across tuning parameters:

  model  winnow  trials  ROC        Sens       Spec
  rules  FALSE    1      0.9997791  0.9999421  0.9996161
```

```
  rules  FALSE   10      0.9999175  0.9997162  0.9997224
  rules  FALSE   20      0.9999191  0.9997915  0.9997342
  rules  FALSE   30      0.9999288  0.9998205  0.9997401
  rules  FALSE   40      0.9999283  0.9998378  0.9997224
  rules  FALSE   50      0.9999338  0.9998552  0.9997283
  rules   TRUE    1      0.9997791  0.9999421  0.9996161
  rules   TRUE   10      0.9997791  0.9999421  0.9996161
  rules   TRUE   20      0.9997791  0.9999421  0.9996161
  rules   TRUE   30      0.9997791  0.9999421  0.9996161
  rules   TRUE   40      0.9997791  0.9999421  0.9996161
  rules   TRUE   50      0.9997791  0.9999421  0.9996161
  tree   FALSE    1      0.9997791  0.9999421  0.9996161
  tree   FALSE   10      0.9999173  0.9997162  0.9997224
  tree   FALSE   20      0.9999231  0.9998147  0.9997342
  tree   FALSE   30      0.9999275  0.9998320  0.9997342
  tree   FALSE   40      0.9999273  0.9998494  0.9997165
  tree   FALSE   50      0.9999311  0.9998494  0.9997224
  tree    TRUE    1      0.9997791  0.9999421  0.9996161
  tree    TRUE   10      0.9997791  0.9999421  0.9996161
  tree    TRUE   20      0.9997791  0.9999421  0.9996161
  tree    TRUE   30      0.9997791  0.9999421  0.9996161
  tree    TRUE   40      0.9997791  0.9999421  0.9996161
  tree    TRUE   50      0.9997791  0.9999421  0.9996161

Sens was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules
and winnow = TRUE.

Call:
summary.diff.resamples(object = results$modelDiff[[name]])

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

ROC
         original  down       up         smote      rose
original           -3.383e-03 -3.924e-03 -3.864e-03 -3.718e-03
down     2.850e-05             -5.407e-04 -4.815e-04 -3.346e-04
up       1.749e-06 0.2716                  5.926e-05  2.061e-04
smote    9.743e-07 0.6261     1.0000                  1.469e-04
rose     5.669e-06 1.0000     6.305e-11  0.5807

Sens
```

```
         original  down       up         smote      rose
original           -7.803e-03 -7.803e-03 -7.655e-03 -7.745e-03
down     1.694e-06              0.000e+00  1.481e-04  5.792e-05
up       1.694e-06 NA                     1.481e-04  5.792e-05
smote    1.021e-06 1.00000    1.00000                -9.023e-05
rose     2.026e-06 0.05623    0.05623    1.00000

Spec
         original  down       up         smote      rose
original            1.037e-03 -4.445e-05 -7.408e-05  3.099e-04
down     0.33070              -1.081e-03 -1.111e-03 -7.272e-04
up       0.01917   0.27157                -2.963e-05  3.543e-04
smote    1.538e-05 0.23778    0.01001                3.839e-04
rose     8.550e-08 1.00000    1.270e-09  1.333e-11

$original
C5.0 variable importance

              Overall
V12            100.00
V13             98.73
V11              0.26
DM.sigma         0.00
DM.kurtosis      0.00
DM.skew          0.00
Prof.sigma       0.00
V10              0.00
Prof.mu          0.00
Prof.kurtosis    0.00
DM.mu            0.00
Prof.skew        0.00

$down
C5.0 variable importance

              Overall
V12               100
Prof.skew           0
DM.sigma            0
Prof.mu             0
Prof.sigma          0
V10                 0
V13                 0
V11                 0
```

```
DM.kurtosis         0
DM.skew             0
DM.mu               0
Prof.kurtosis       0

$up
C5.0 variable importance

             Overall
V12              100
V13                0
DM.sigma           0
DM.mu              0
Prof.skew          0
Prof.sigma         0
V11                0
V10                0
Prof.mu            0
DM.kurtosis        0
DM.skew            0
Prof.kurtosis      0

$smote
C5.0 variable importance

             Overall
V13              100
Prof.kurtosis      0
DM.kurtosis        0
Prof.skew          0
V11                0
Prof.sigma         0
Prof.mu            0
DM.skew            0
V10                0
V12                0
DM.sigma           0
DM.mu              0

$rose
C5.0 variable importance

             Overall
V13              100
```

```
Prof.kurtosis       0
DM.skew             0
Prof.sigma          0
Prof.skew           0
V10                 0
V12                 0
V11                 0
DM.sigma            0
DM.mu               0
Prof.mu             0
DM.kurtosis         0


Call:
summary.resamples(object = results$models_resamples[[name]])

Models: original, down, up, smote, rose
Number of resamples: 50

ROC
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9831  0.9944 0.9944 0.9961  1.0000    1    0
down     0.9944  1.0000 1.0000 0.9994  1.0000    1    0
up       0.9999  1.0000 1.0000 1.0000  1.0000    1    0
smote    0.9963  1.0000 1.0000 0.9999  1.0000    1    0
rose     0.9994  0.9997 0.9999 0.9998  0.9999    1    0

Sens
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9663  0.9889 0.9889 0.9922       1    1    0
down     1.0000  1.0000 1.0000 1.0000       1    1    0
up       1.0000  1.0000 1.0000 1.0000       1    1    0
smote    0.9926  1.0000 1.0000 0.9999       1    1    0
rose     0.9994  1.0000 1.0000 0.9999       1    1    0

Spec
           Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
original 0.9997  0.9999 1.0000 0.9999  1.0000    1    0
down     0.9889  1.0000 1.0000 0.9989  1.0000    1    0
up       0.9999  1.0000 1.0000 1.0000  1.0000    1    0
smote    1.0000  1.0000 1.0000 1.0000  1.0000    1    0
rose     0.9988  0.9994 0.9997 0.9996  0.9997    1    0
```

Appendix G

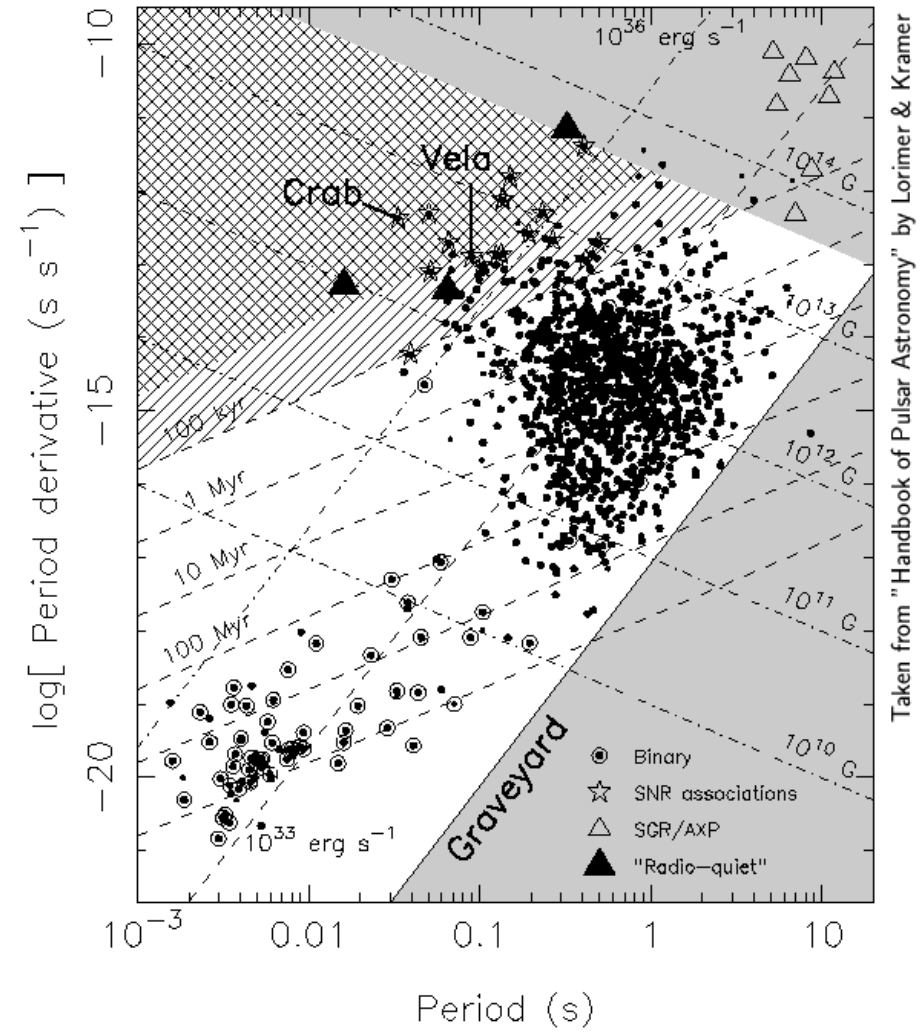
Pulsar Characteristics

Pulsars are still very mysterious objects. The mechanism by which they emit their powerful pulses of energy are still not yet fully understood. Enough pulsars have been discovered and studied that there is sufficient data to characterize them into a few different categories. Each category holds different challenges for the detection and timing of the pulsars. The classic $P/\dot{P}$ diagram shown in Figure G.1 shows distinct populations of pulsars. The normal pulsars inhabit the middle of the diagram, and the millisecond pulsars inhabit the lower left corner of the diagram. The exotic pulsars are found throughout the diagram.

Normal Pulsars

Normal pulsars are those that have a regular spin period, P, which is longer than about 0.5 seconds, and which has a relativly high rate of change of period, or $\dot{P}$. The normal pulsars are thought to be born when a star reaches the end of its main-sequence star lifetime and explodes in a supernova. The core that remains after the event is a neutron star that forms the pulsar. The $\dot{P}$ results from the need to supply the energy that is emitted by the pulsar. The required energy comes from the loss of angular momentum from the pulsar spinning down. The normal pulsar will eventually spin down and go to the pulsar graveyard in the lower right of the $P/\dot{P}$ diagram , unless it is lucky enough to have a companion to spin it up until it evolves into a millisecond pulsar.

Figure G.1: The $P/\dot{P}$ diagram (From D. Lorimer and Kramer (2005)



.

Millisecond Pulsars

Millisecond pulsars, as the name implies, spin at a more rapid rate than do the normal pulsars. Pulsars have been detected that have a period of as fast as about 1.3 milliseconds (Hessels et al., 2006). How they came about is still a matter for debate, but most physicists believe that the millisecond pulsars are formed from normal pulsars when a companion star gives up its mass and momentum to the millisecond pulsar, spinning it up. Some 80% of all millisecond pulsars occur in

a binary system with another star (D. Lorimer & Kramer, 2005), which tends to support this view of the formation of the millisecond pulsar. The pulse duty cycle of the millisecond pulsar tends to be higher than that of the normal pulsar. The spin period is also more regular.

More Exotic Pulsars

More exotic pulsars include pulsars in binary or terniary orbits with other pulsars or normal stars, or pulsars orbiting a black hole. Some pulsars do not exhibit a regular periodic pulse emission, rather they emit sporadically, often emitting pulses in a regular train, and then switching off for some time. These pulsars are difficult to find, since the method of folding the data to increase the SNR will actually obliterate the signal from these pulsars. Some pulsars are bright enough to detect with a single pulse with large sensitive antennas. These pulsars are typically young highly energetic pulsars, such as the Crab pulsar. These exotic systems are sought after simply because they are exotic, and provide new tools for probing the physics of pulsars.

Bibliography

Bates, S. D., Bailes, M., Barsdell, B. R., Bhat, N. D. R., Burgay, M., Burke-Spolaor, S., ... van Straten, W. (2012). The High Time Resolution Universe Survey VI: An Artificial Neural Network and Timing of 75 Pulsars. *ArXiv e-prints*. arXiv: 1209.0793 [astro-ph.SR]

Belokurov, V., Evans, N. W., & Du, Y. L. (2003). Light-curve classification in massive variability surveys - I. Microlensing. *Monthly Notices of the Royal Astronomical Society*, *341*(4), 1373–1384. doi:10.1046/j.1365-8711.2003.06512.x

Bergmeir, C. & Benítez, J. M. (2012). Neural networks in R using the stuttgart neural network simulator: RSNNS. *Journal of Statistical Software*, *46*(7), 1–26. Retrieved from http://www.jstatsoft.org/v46/i07/

Breiman, L. (1996). Bagging predictors. *Mach. Learn. 24*(2), 123–140. doi:10.1023/A:1018054314350

Brink, H., Richards, J. W., Poznanski, D., Bloom, J. S., Rice, J., Negahban, S., & Wainwright, M. (2013). Using machine learning for discovery in synoptic survey imaging data. *MNRAS*, *435*, 1047–1060. doi:10.1093/mnras/stt1306. arXiv: 1209.3775 [astro-ph.IM]

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. doi:10.1007/BF00994018

Demorest, P. (2013, January 7). personal communication.

Eatough, R. P., Molkenthin, N., Kramer, M., Noutsos, A., Keith, M. J., Stappers, B. W., & Lyne, A. G. (2010). Selection of radio pulsar candidates using artificial neural networks. *MNRAS*, *407*, 2443–2450. doi:10.1111/j.1365-2966.2010.17082.x. arXiv: 1005.5068 [astro-ph.IM]

Firth, A. E., Lahav, O., & Somerville, R. S. (2003). Estimating photometric redshifts with artificial neural networks. *MNRAS*, *339*, 1195–1202. doi:10.1046/j.1365-8711.2003.06271.x. eprint: arXiv:astro-ph/0203250

Frohlich, H., Chapelle, O., & Scholkopf, B. (2003). Feature selection for support vector machines by means of genetic algorithm. In *Tools with artificial intelligence, 2003. proceedings. 15th ieee international conference on* (pp. 142–148). doi:10.1109/TAI.2003.1250182

from Jed Wing, M. K. C., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., ... Candan., C. (2016). *Caret: Classification and regression training*. R package version 6.0-71. Retrieved from https://CRAN.R-project.org/package=caret

Hamel, L. H. (2011). *Knowledge discovery with support vector machines*. John Wiley & Sons.

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Hassan, U. & Anwar, M. S. (2010). Reducing noise by repetition: Introduction to signal averaging. *European Journal of Physics*, *31*(3), 453. Retrieved from http://stacks.iop.org/0143-0807/31/i=3/a=003

Heatherly, S. A. (2013). Pulsar search collaboratory home page. Retrieved from https://sites.google.com/a/pulsarsearchcollaboratory.com/pulsar-search-collaboratory/Home

Hessels, J. W. T., Ransom, S. M., Stairs, I. H., Freire, P. C. C., Kaspi, V. M., & Camilo, F. (2006). A Radio Pulsar Spinning at 716 Hz. *Science*, *311*, 1901–1904. doi:10.1126/science.1123430. eprint: astro-ph/0601337

Hu, S., Liang, Y., Ma, L., & He, Y. (2009). MSMOTE: Improving classification performance when training data is imbalanced. In *Computer science and engineering, 2009. wcse '09. second international workshop on* (Vol. 2, pp. 13–17). doi:10.1109/WCSE.2009.756

Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). Kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, *11*(9), 1–20. Retrieved from http://www.jstatsoft.org/v11/i09/

Keith, M. J. (2013). The High Time Resolution Universe surveys for pulsars and fast transients. In J. van Leeuwen (Ed.), *Iau symposium* (Vol. 291, pp. 29–34). IAU Symposium. doi:10.1017/S1743921312023095. arXiv: 1210.7868 [astro-ph.SR]

Keith, M. J., Eatough, R. P., Lyne, A. G., Kramer, M., Possenti, A., Camilo, F., & Manchester, R. N. (2009). Discovery of 28 pulsars using new techniques for sorting pulsar candidates. *MNRAS*, *395*, 837–846. doi:10.1111/j.1365-2966.2009.14543.x. arXiv: 0901.3570 [astro-ph.SR]

Kubat, M. & Matwin. (1997). Addressing the Curse of Imbalanced Training Sets : One-Sided Selection. In *Proceedings of the 14th international conference on machine learning* (pp. 179–186).

Kuhn, M., Weston, S., Coulter, N., & code for C5.0 by R. Quinlan, M. C. C. (2015). *C50: C5.0 decision trees and rule-based models*. R package version 0.1.0-24. Retrieved from https://CRAN.R-project.org/package=C50

Lahav, O., Naim, A., Sodré, L., Jr., & Storrie-Lombardi, M. C. (1996). Neural computation as a tool for galaxy classification: methods and examples. *MNRAS*, *283*, 207. eprint: arXiv:astro-ph/9508012

LeCun, Y., Bottou, L., Orr, G., & Muller, K. (1998). Efficient backprop. In G. Orr & M. K. (Eds.), *Neural networks: Tricks of the trade*. Springer.

Lee, K. J., Stovall, K., Jenet, F. A., Martinez, J., Dartez, L. P., Mata, A., ... Zhu, W. W. (2013). PEACE: pulsar evaluation algorithm for candidate extraction - a software package for post-analysis processing of pulsar survey candidates. *MNRAS*. doi:10.1093/mnras/stt758. arXiv: 1305.0447 [astro-ph.IM]

Lorimer, D. R. (2011). Blind surveys for radio pulsars and transients. In M. Burgay, N. D'Amico, P. Esposito, A. Pellizzoni, & A. Possenti (Eds.), *American institute of physics conference series* (Vol. 1357, pp. 11–18). doi:10.1063/1.3615066. arXiv: 1012.4695 [astro-ph.GA]

Lorimer, D. & Kramer, M. [Michael]. (2005). *Handbook of pulsar astronomy* (First). Cambridge, UK: Cambridge University Press.

Lunardon, N., Menardi, G., & Torelli, N. (2014). ROSE: A Package for Binary Imbalanced Learning. *R Journal*, *6*(1), 82–92.

Lyon, R. J., Stappers, B. W., Cooper, S., Brooke, J. M., & Knowles, J. D. (2016). Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*. doi:10.1093/mnras/stw656. eprint: http://mnras.oxfordjournals.org/content/early/2016/04/17/mnras.stw656.full.pdf+html

Manchester, R., Lyne, A., Camilo, F., Bell, J., Kaspi, V., D'Amico, N., ... Sheppard, D. (2001). The parkes multi-beam pulsar survey - i. observing and data analysis systems, discovery and timing of 100 pulsars. *Monthly Notices of the Royal Astronomical Society*, *328*(1), 17–35. doi:10.1046/j.1365-8711.2001.04751.x

McLaughlin, M. (2011). Radio Searches for Pulsars and Short-Duration Transients. In E. Göğüş, T. Belloni, & Ü. Ertan (Eds.), *American institute of physics conference series* (Vol. 1379, pp. 48–55). American Institute of Physics Conference Series. doi:10.1063/1.3629484. arXiv: 1103.1278 [astro-ph.SR]

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2015). *E1071: Misc functions of the department of statistics, probability theory group (formerly: E1071), tu wien*. R package version 1.6-7. Retrieved from https://CRAN.R-project.org/package=e1071

Morello, V., Barr, E. D., Bailes, M., Flynn, C. M., Keane, E. F., & van Straten, W. (2014). SPINN: a straightforward machine learning solution to the pulsar candidate selection problem. *MNRAS*, *443*, 1651–1662. doi:10.1093/mnras/stu1188. arXiv: 1406.3627 [astro-ph.IM]

Morgan, A. N., Long, J., Richards, J. W., Broderick, T., Butler, N. R., & Bloom, J. S. (2012). Rapid, Machine-learned Resource Allocation: Application to High-redshift Gamma-Ray Burst Follow-up. *APJ*, *746*, 170. doi:10.1088/0004-637X/746/2/170. arXiv: 1112.3654 [astro-ph.IM]

Nguyen, M. H. & De la Torre, F. (2010). Optimal feature selection for support vector machines. *Pattern Recognition*, *43*(3), 584–591.

North American Nanohertz Observatory for Gravitational Waves. (2012). Home page. Retrieved from http://nanograv.org/

Peters, A. & Hothorn, T. (2015). *Ipred: Improved predictors*. R package version 0.9-5. Retrieved from https://CRAN.R-project.org/package=ipred

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. doi:10.1007/BF00116251

Ransom, S. (2013, March 7). personal communication.

Revolution Analytics & Weston, S. (2015). *Doparallel: Foreach parallel adaptor for the 'parallel' package*. R package version 1.0.10. Retrieved from https://CRAN.R-project.org/package=doParallel

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227. doi:10.1007/BF00116037

Scholkopf, B., Sung, K.-K., Burges, C., Girosi, F., Niyogi, P., Poggio, T., & Vapnik, V. (1997). Comparing support vector machines with gaussian kernels to radial basis function classifiers. *Signal Processing, IEEE Transactions on*, *45*(11), 2758–2765. doi:10.1109/78.650102

Seiffert, C., Khoshgoftaar, T., Van Hulse, J., & Napolitano, A. (2007). Mining data with rare events: A case study. In *Tools with artificial intelligence, 2007. ictai 2007. 19th IEEE international conference on* (Vol. 2, pp. 132–139). doi:10.1109/ICTAI.2007.71

Seiffert, C., Khoshgoftaar, T., Van Hulse, J., & Napolitano, A. (2009). RUSBoost: A hybrid approach to alleviating class imbalance. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *40*(1), 185–197. doi:10.1109/TSMCA.2009.2029559

Smits, R., Kramer, M. [M.], Stappers, B., Lorimer, D. R., Cordes, J., & Faulkner, A. (2009). Pulsar searches and timing with the square kilometre array. *AAP*, *493*, 1161–1170. doi:10.1051/0004-6361:200810383. arXiv: 0811.0211

Smits, R., Lorimer, D. R., Kramer, M., Manchester, R., Stappers, B., Jin, C. J., ... Li, D. (2009). Pulsar science with the Five hundred metre Aperture Spherical Telescope. *AAP*, *505*, 919–926. doi:10.1051/0004-6361/200911939. arXiv: 0908.1689 [astro-ph.IM]

Therneau, T., Atkinson, B., & Ripley, B. (2015). *Rpart: Recursive partitioning and regression trees*. R package version 4.1-10. Retrieved from https://CRAN.R-project.org/package=rpart

Torgo, L. (2010). *Data mining with r, learning with case studies*. Chapman and Hall/CRC. Retrieved from http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR

Van Hulse, J., Khoshgoftaar, T., Napolitano, A., & Wald, R. (2009). Feature selection with high-dimensional imbalanced data. In *Data mining workshops, 2009. icdmw '09. IEEE international conference on* (pp. 507–514). doi:10.1109/ICDMW.2009.35

Vapnik, V. N. (1999). An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, *10*(5), 988–999.

Venables, W. N. & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth). ISBN 0-387-95457-0. New York: Springer. Retrieved from http://www.stats.ox.ac.uk/pub/MASS4

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000). Feature selection for SVMs. *NIPS*, *12*, 668–674.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, *4*, 65–85.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, *SMC-2*(3), 408–421. doi:10.1109/TSMC.1972.4309137

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241–259.

Wolpert, D. & Macready, W. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, *1*(1), 67–82. doi:10.1109/4235.585893

Wright, D. E., Smartt, S. J., Smith, K. W., Miller, P., Kotak, R., Rest, A., ... Waters, C. (2015). Machine learning for transient discovery in Pan-STARRS1 difference imaging. *MNRAS*, *449*, 451–466. doi:10.1093/mnras/stv292. arXiv: 1501.05470 [astro-ph.IM]

Zhang, Y., Li, L., & Zhao, Y. (2009). Morphology classification and photometric redshift measurement of galaxies. *Monthly Notices of the Royal Astronomical Society*, *392*(1), 233–239. doi:10.1111/j.1365-2966.2008.14022.x

Zhu, W. W., Berndsen, A., Madsen, E. C., Tan, M., Stairs, I. H., Brazier, A., ... Venkataraman, A. (2014). Searching for Pulsars Using Image Pattern Recognition. *APJ*, *781*, 117. doi:10.1088/0004-637X/781/2/117. arXiv: 1309.0776 [astro-ph.IM]