Data Collection, Analysis, and Classification for the Development of a Sailing Performance Evaluation System

Ryan Sammon

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfilment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

in Mechanical Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Ontario, Canada

May 2013

Abstract

The work described in this thesis contributes to the development of a system to evaluate sailing performance. This work was motivated by the lack of tools available to evaluate sailing performance. The goal of the work presented is to detect and classify the turns of a sailing yacht. Data was collected using a BlackBerry PlayBook affixed to a J/24 sailing yacht. This data was manually annotated with three types of turn: tack, gybe, and mark rounding. This manually annotated data was used to train classification methods. Classification methods tested were multi-layer perceptrons (MLPs) of two sizes in various committees and nearest-neighbour search. Pre-processing algorithms tested were Kalman filtering, categorization using quantiles, and residual normalization. The best solution was found to be an averaged answer committee of small MLPs, with Kalman filtering and residual normalization performed on the input as pre-processing.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

The performance of teams in vehicle racing scenarios is heavily dependant on the actions of the vehicle crew. In automotive racing, the tracks are fixed, allowing the driver to evaluate their performance based upon ideal paths around turns and lap times. In sailing, the courses are not as rigidly defined, and the ideal course of action for the crew at any time is completely dependant on the environmental conditions at that time. A simple method for evaluating sailing performance is true speed towards the next mark. This method can only be used when sailing in a straight line. Sailing performance depends heavily on how the crew executes turns. Therefore, a system to evaluate the quality of turn execution is desired.

1.2 Goal of the Thesis

The goal of the thesis is to contribute to the development of system that can evaluate sailing performance. The system must be able to estimate the state of the vessel in order to compute performance metrics. The most reasonable way to do this is to affix a collection of sensors to the vessel. Each sensor in the collection measures some aspect of the state of the vessel. These measurements can then be combined to obtain an estimate for the complete state of the vessel.

The system must also be able to detect the start and end of each turn in order to compute the performance metrics for the turn. Furthermore, each turn must be classified as the performance metrics vary between the different types of turn. The two primary types of turn in sailing are the tack and the gybe. During a tack the bow of the vessel turns across the wind, while during a gybe the stern of the vessel turns across the wind. In general, tacks are done while sailing upwind, and gybes are done while sailing downwind. A third type of turn, the mark rounding, is considered due to its prevalence in racing scenarios. The marks that define a racing course must be rounded. The turn required to accomplish this differs significantly from a typical tack or gybe, although it may technically be as such. Therefore, the system must use some form of classifier to differentiate between the three possible types of turn considered.

1.3 Outline of the Thesis

Chapter 2 contains a description of the methodology employed and its justification within the literature. Chapter 3 contains a description the hardware and software used to collect the required data. Chapter 4 contains a description of the algo-

rithms applied to the collected data. Chapter 5 contains descriptions and results of the tests performed, as well as a discussion of these results. Chapter 6 contains a description of the future work and the conclusion of the thesis.

2 Methodology

The methodology chosen to meet the requirements is to classify each data point collected by the sensors affixed to the vessel into one of four classes: tack, gybe, mark rounding, straight sailing. The rationale behind this choice is that it addresses the requirement of turn classification and the requirement of turn start and end detection at the same time. The most reasonable way to construct such a classifier is via supervised learning, with training data provided via manually classified data points.

2.1 Literature Review

The first step in the literature review was to consult an overview of statistical classification methods [1]. This was followed by searching for literature which described the application of these methods to the classification of measurements. Unfortunately, this search did not yield any evidence of previous efforts to classify measurements from sailing vessels. However, this search did yield several results that involved either measurements from sailing vessels, or classification of measurements in general. These results were reviewed in order to determine what type of supervised learning classifier to use and what pre-processing to perform on the measurements.

Several results made use of a support vector machine (SVM) to classify measurements. In [2], a SVM was used to differentiate between slipping and non-slipping wheels on a wheeled robot. In [3], a SVM was used to differentiate between the gait of young people and the gait of elderly people. These results show that SVMs can be used to classify measurements with satisfactory results. However, SVMs are binary classifiers in isolation. In order to perform classification into three or more classes, several SVMs must be combined, which increases the complexity of the system [4].

Another supervised learning classifier used by several results was a neural network. In [5] and [6], a neural network was used to classify electroencephalography (EEG) measurements. In [7], a neural network was used to classify gait patterns. These results show that neural networks can also obtain satisfactory results when classifying measurements. The performance of a neural network is dependant on both the number of neurons employed [8], and the number of neuron layers [9]. Neural networks are commonly combined into committees to increase their performance [10].

The development of an unmanned catamaran utilizing a wing-sail is presented in [11]. This project involved taking measurements from a sailing vessel using a global positioning system (GPS) and processing those measurements with a

variant of the Kalman filter. Other unmanned projects took measurements using accelerometers, gyroscopes, and magnetometers, in addition to GPS measurements [12] [13] [14]. All of these projects utilized the Kalman filter or some variant of it to process or fuse the measurements. Other pre-processing algorithms such as categorization using quantiles [15] and residual normalization [16] were found in the literature. These techniques are described in detail in Chapter 4.

2.2 Summary of Proposed Methodology

The sensors to be used are a GPS receiver, a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer [12] [13] [14]. The sensor measurements will be pre-processed by Kalman filtering, categorization using quantiles [15], and residual normalization [16].

A neural network is chosen as the classifier [5] [6] [7]. In particular, the multilayer perceptron (MLP) is chosen. Due to the lack of previous efforts to classify measurements from sailing vessels, it is reasonable to apply simple and classic approaches. Because it is difficult to estimate the effect of the network dimensions on performance [8] [9], a MLP with a large number of neurons and layers and a MLP with a small number of neurons and layers will be tested. Furthermore, each size of MLP will be tested in both individual and committee form [10].

It is useful to have another classifier to compare the MLP results with. Nearest-neighbour search is chosen as this classifier due to its simplicity and common use as baseline for comparison. It does not have an involved training process, but does require more memory than a MLP [17].

3 Data Collection

Measurements from a sailing vessel must be collected and manually classified as a first step towards training the classification methods. The required measurements were collected over a number of days between July 26, 2012 and Sept. 29, 2012. In addition to summarizing the data collected and its manual classification, this section describes the sensors that were used, the hardware platform which housed them, and the vessel to which they were affixed.

3.1 Sensors

The sensors required for state estimation are a GPS receiver, a three-axis accelerometer, a three-axis gyroscope, and a three-axis magnetometer. While the ability to perform state estimation is not utilized in this work, it is a requirement of the system that will be utilized in future work. The first step in performing state estimation is to remove the hard and soft-iron effects from the magnetometer measurements [18]. After this is done, the magnetometer and accelerometer measurements can be used to calculate the orientation of the vessel, assuming that it is not accelerating [19]. Obviously, the vessel will be accelerating at certain times, and this assumption will not always hold. When this assumption is not valid, the gyroscope measurements can provide an estimate of orientation, either through integration or fusion via the Kalman filter. The GPS receiver provides measurements of position and velocity. However, these velocity measurements are in the plane normal to gravity and must be transformed to the body-frame of the vessel using its orientation. These body-frame velocity measurements can then be fused with the accelerometer measurements via the Kalman filter to provide an optimal state estimate. It is important to note that if the GPS signal is lost, the accelerometer measurements can be integrated to provide an estimate of velocity. However, errors will accumulate, eventually making the estimate useless.

3.2 Hardware Platforms

The hardware platform on which the sensors were installed was changed partway through the data collection phase. This section describes the original hardware platform, the reasons for which it was abandoned, and the final hardware platform.

3.2.1 Custom Hardware Platform

The development of the original hardware platform, which was custom built for the purposes of this work, is presented in [20]. The core of the platform is the Axon II micro-controller, which is an Amtel ATmega640 micro-controller with additional

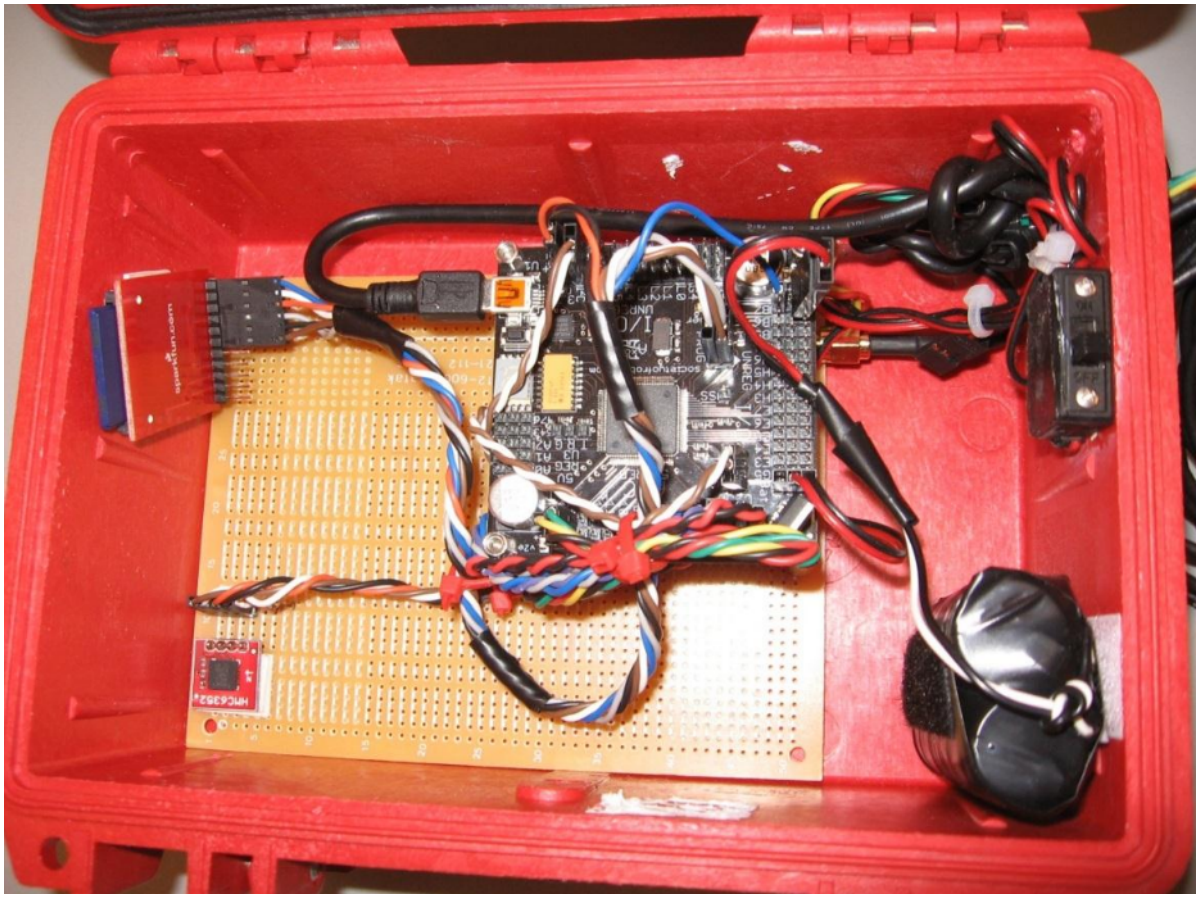

Figure 3.1: Photograph of the Custom Hardware Platform [20]

features to make interfacing with external devices easier. The identities of the sensor components are shown in Table 3.1, while their specifications are shown in Table 3.2. A photograph of the assembled platform is shown in Figure 3.1. The values in Table 3.2 for the GPS heading and speed measurements are based on the observed minimum changes. The sensors record to an on-board Secure Digital (SD) card. The magnetometer was changed from a two-axis component to the three-axis component shown in the tables after the documentation of the work. The platform has the capability to include wind speed and direction sensors. However, their resolution is very coarse and they are vulnerable to damage during data collection.

Table 3.1: Custom Hardware Sensor Components [20]

Sensor	Manufacturer	Product Identifier
GPS	Venus	634LPx
Accelerometer	Analog Devices	ADXL335
Gyroscope	STMicroelectronics	LY530ALH and LPR530AL
Magnetometer	Honeywell	HMC5883L

Table 3.2: Custom Hardware Sensor Specifications

Sensor	Range	Resolution
GPS Heading	0 [°] to 360 [°]	0.1 [°]
GPS Speed [20]	0 [m/s] to 515 [m/s]	0.1 [m/s]
Accelerometer [20]	$\pm$29.43 [m/s^2]	0.1451 [m/s^2]
Gyroscope [20]	$\pm$300 [°/s]	1.466 [°/s]
Magnetometer [21]	$\pm$8.1 [G]	4.35 [mG]

The major issue with this platform was reliability. Three complete copies were constructed, yet a copy with all sensors in perfect working order was not obtained. The primary reasons for this poor reliability are the quality of the components, and the quality of the workmanship. The components are hobby-grade and have a reliability to match, and all of the soldering was done by hand by workers who were relatively inexperienced. It is unfortunate that the platform had to be abandoned, but it was much more difficult to remedy its issues than switch to a new platform.

3.2.2 BlackBerry PlayBook

The new platform chosen, and the platform with which all data presented in this work was collected, was the BlackBerry PlayBook. Compared to the previous platform, this platform is highly reliable as it is manufactured with proper quality control. This platform also has the advantage that it is easier to code for as a robust application programming interface (API) is provided. Finally, any eventual commercialization of this work will benefit from the prevalence of the platform. The only disadvantage of this platform is that it was not possible to include the wind speed and direction sensors from the previous platform due to the restrictions the API puts on communication with external devices. This was seen as an acceptable drawback.

A BlackBerry PlayBook was taken apart to identify its internal components in [22]. The identities of the internal components are presented in Table 3.3. The identity of the magnetometer was not provided in [22], however it was discerned from the images of the component boards that were provided. The specifications of each component are presented in Table 3.4. The specifications of the exact GPS component used in the BlackBerry PlayBook were not available, as they are subject to a non-disclosure agreement [23]. However, the specifications of a different GPS component that is also distributed by Texas Instruments, and is likely very similar, were located in [24]. A diagram of the BlackBerry Playbook sensor axes is shown in Figure 3.2.

The sensors installed in the BlackBerry PlayBook have several different settings

Table 3.3: BlackBerry PlayBook Sensor Components [22]

Sensor	Manufacturer	Product Identifier
GPS	Texas Instruments	WL1283
Accelerometer	Bosch Sensortec	BMA150
Gyroscope	InvenSense	MPU-3050
Magnetometer	Honeywell	HMC5883L

Table 3.4: BlackBerry PlayBook Sensor Specifications

Sensor	Range	Resolution
GPS Heading	0 [°] to 360 [°]	0.01 [°]
GPS Speed [24]	0 [m/s] to 514 [m/s]	0.001 [m/s]
Accelerometer [25]	± 19.62 [m/s^2]	0.03832 [m/s^2]
Gyroscope [26]	± 1000 [°/s]	0.03049 [°/s]
Magnetometer [21]	± 8.1 [G]	4.35 [mG]

with regards to range, resolution, and frequency. These settings are configured when the BlackBerry PlayBook is manufactured and cannot be changed through the software API provided. Additionally, these settings are not available to the public. The values in Table 3.4 for the accelerometer, gyroscope, and magnetometer were determined by analysing the output of each sensor and matching the resolution observed to a range and resolution pair in the relevant data sheet. The values in Table 3.4 for the GPS heading and speed measurements are based on the observed minimum changes and the velocity limit set in the International Traffic in Arms Regulations (ITAR) as quoted in [24]. The GPS reports once per second, as reported in [24], and experimentally observed. The frequencies at which the other sensors report depend on their initial configuration, which includes enabling or disabling optional filtering. Efforts to determine these frequencies experimentally indicate they are greater than or equal to 50 [Hz].

The code for the application that collected the sensor measurements is located in Appendix C. This application functioned by requesting that all of the sensors report their measurements at 200 [Hz], but only recorded the measurements at 50 [Hz]. At the end of each recording period, the most recently reported measurements from each sensor were recorded to a file. The recording frequency of 50 [Hz] was chosen as a balance between accurate measurements and file size. As the application is typically run for hours at a time, the files can become quite large. It is important to note that the application requires permission to use the GPS receiver and write to files.

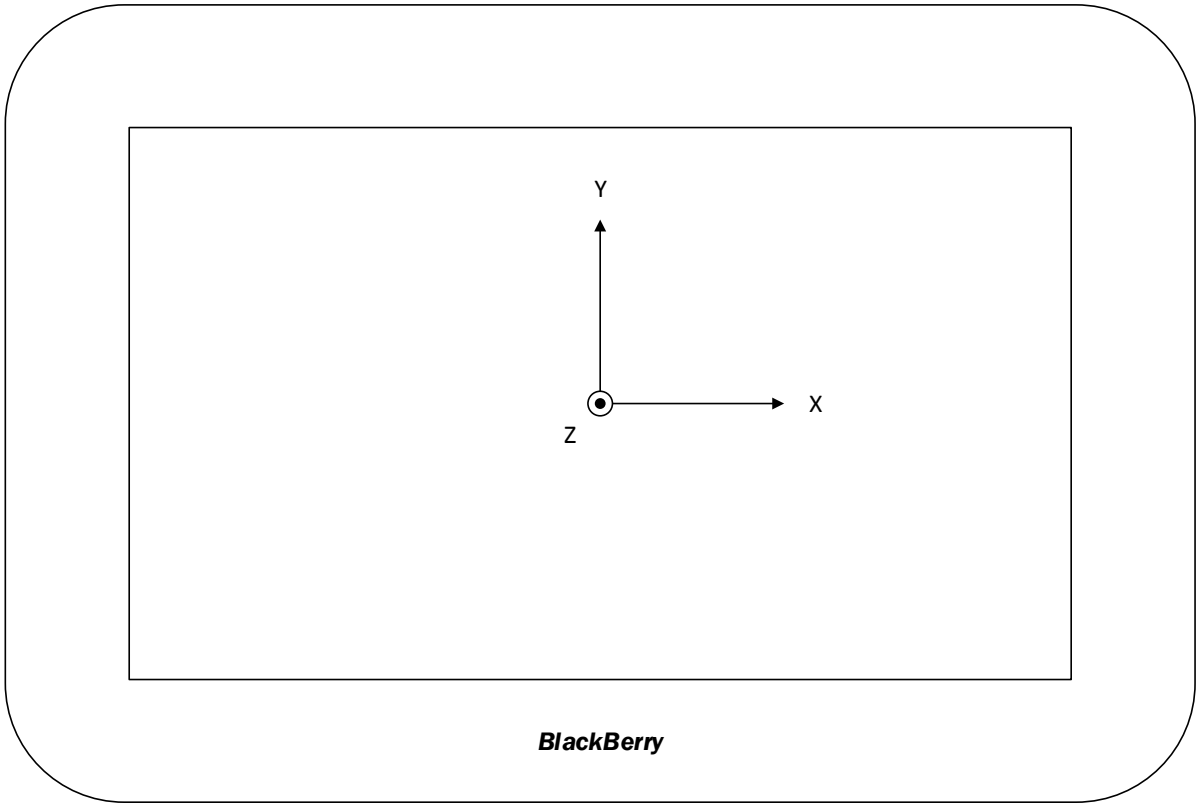

Figure 3.2: Diagram of BlackBerry PlayBook Axes

3.3 Vessel

The BlackBerry PlayBook was affixed to a J/24 sailing yacht. The J/24 is recognized by the International Sailing Federation (ISAF) as an international one-design class of sailing yacht [27]. This means that all vessels in a race must adhere to the class rules for the J/24 design. These rules define the weight, dimensions, and tolerances of every part of the vessel that could significantly alter performance [28]. The purpose of these rules is to create races where skill and luck alone decide the victor.

The J/24 sailing yacht used has a small cabin below deck which includes kitchen facilities and sleeping bunks. The BlackBerry PlayBook was affixed flat on its back to the floor of a sleeping bunk. As a result, it was not affixed at the centre of mass of the vessel. Ideally, this would be true, but it is impractical to achieve. The location of the centre of mass would have to be calculated, which is difficult as the dimensions of the yacht are irregular and it is constructed of several different materials. Additionally, the location of the centre of mass is likely inside part of the structure, making it impossible to reach. Finally, the location of the centre of mass is modified by the position of the crew. Therefore, the offset from the centre of mass was deemed to be acceptable. The axes defined for the vessel are shown in Figure 3.3. To be clear, X is forward, Y is left, and Z is the cross-product of

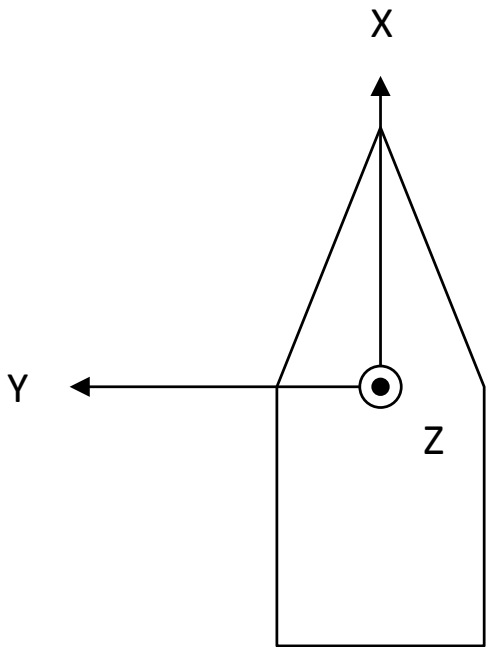

Figure 3.3: Diagram of Vessel Axes

these unit vectors. The BlackBerry PlayBook was affixed with X backward and Y right, resulting in the need to invert the values along these axes.

3.4 Data Collected

Data was collected on seven days between July 26, 2012 and Sept. 29, 2012 during races on the Ottawa River and based at the Nepean Sailing Club. The section of the Ottawa River where the races took place is referred to as Lac Deschênes. Prior to the start of each race, the BlackBerry PlayBook was affixed to the vessel and the data collection application started. The BlackBerry PlayBook was affixed at the same location each time. A summary of the data collected is shown in Table 3.5. It should be noted that the mean recording frequency is slightly less than the nominal recording frequency of 50 [Hz] defined in the code for the BlackBerry PlayBook application. The reason for this difference is that the application records the data only after the period has elapsed, leading to a maximum delay of the time it takes to execute one iteration of the main loop.

Each turn in the collected data was manually classified as either a tack, a gybe, or a mark rounding. This classification was based on knowledge of what occurred during each race. The number of each type of turn is shown in Table 3.6. In order to determine the start and end point of each turn, the GPS heading measurements were analysed. Each start point was manually identified as the point when the GPS heading begins to change, and each end point was manually identified as the point when the GPS heading stops changing. Due to variations in the GPS

Table 3.5: Data Collected

Date	Duration [s]	Data Points	Mean Frequency [Hz]
July 26, 2012	4060	182 207	44.88
Aug. 7, 2012	3168	141 200	44.57
Aug. 9, 2012	5702	256 142	44.93
Aug. 14, 2012	2976	132 360	44.48
Sept. 11, 2012	3520	157 223	44.67
Sept. 25, 2012	2211	98 417	44.52
Sept. 29, 2012	1390	62 028	44.64
Total	23 027	1 029 577	44.71

Table 3.6: Turns Recorded

Date	Tacks	Gybes	Mark Roundings
July 26, 2012	11	4	3
Aug. 7, 2012	4	3	3
Aug. 9, 2012	15	5	6
Aug. 14, 2012	7	3	3
Sept. 11, 2012	10	4	4
Sept. 25, 2012	6	1	3
Sept. 29, 2012	4	0	0
Total	57	20	22

heading measurements and the manual nature of each decision, these points are approximate. The expected error of these approximations is estimated to have an order of magnitude of 10^{-1} seconds. The total number of data points of each type of turn is shown in Table 3.7. This table also shows the total number of straight sailing data points, which dwarfs the number of data points in turns.

Table 3.7: Class Data Points Recorded

Date	Tack	Gybe	Mark Rounding	Straight Sailing
July 26, 2012	10 354	8825	4331	158 697
Aug. 7, 2012	4119	2555	7004	127 522
Aug. 9, 2012	12 506	6442	6930	230 264
Aug. 14, 2012	5471	3259	3925	119 705
Sept. 11, 2012	6690	3473	5826	141 234
Sept. 25, 2012	4394	970	4700	88 353
Sept. 29, 2012	5107	0	0	56 921
Total	48 641	25 524	32 716	922 696

4 Data Analysis

This section describes the pre-processing algorithms, classification algorithms, and confidence algorithms that are applied to the collected data to obtain the results. All of the algorithms described are implemented in MATLAB. The full MATLAB code of all algorithms is located in Appendix B.

4.1 Pre-Processing Algorithms

The pre-processing performed on the input data can have a significant effect on the results of classification methods. In this work, several pre-processing algorithms are employed. Some of these algorithms are certainly beneficial for the results of the classifications methods, while the benefits or drawbacks of others are uncertain without testing them. The flowchart shown in Figure 4.1 and the information contained in Table 4.1 identify these two groups of algorithms. The algorithms that have uncertain benefits and drawbacks are tested in every possible combination in order to completely understand their effects and their interaction with each other. The numbers in Figure 4.1 correspond to the pre-processing combinations described in Table 4.1. The initial state of the input is shown in Table 4.2. The following sections describe each algorithm in detail.

4.1.1 Sensor Correction

The sensors in the BlackBerry PlayBook used to collect the data are not perfect. Specifically, the GPS is prone to large noise spikes in both its heading and speed measurements. Additionally, the GPS heading is measured on the interval $[0°, 360°]$, resulting in discontinuities when crossing the interval bounds. In order to be differentiated, these discontinuities must be removed by transforming the GPS heading measurements to be continuous.

Table 4.1: Pre-Processing Combinations

Number	Optional Pre-Processing Algorithms Used
1	None
2	Kalman Filtering
3	Categorization using Quantiles
4	Residual Normalization
5	Kalman Filtering and Categorization using Quantiles
6	Kalman Filtering and Residual Normalization
7	Categorization using Quantiles and Residual Normalization
8	All

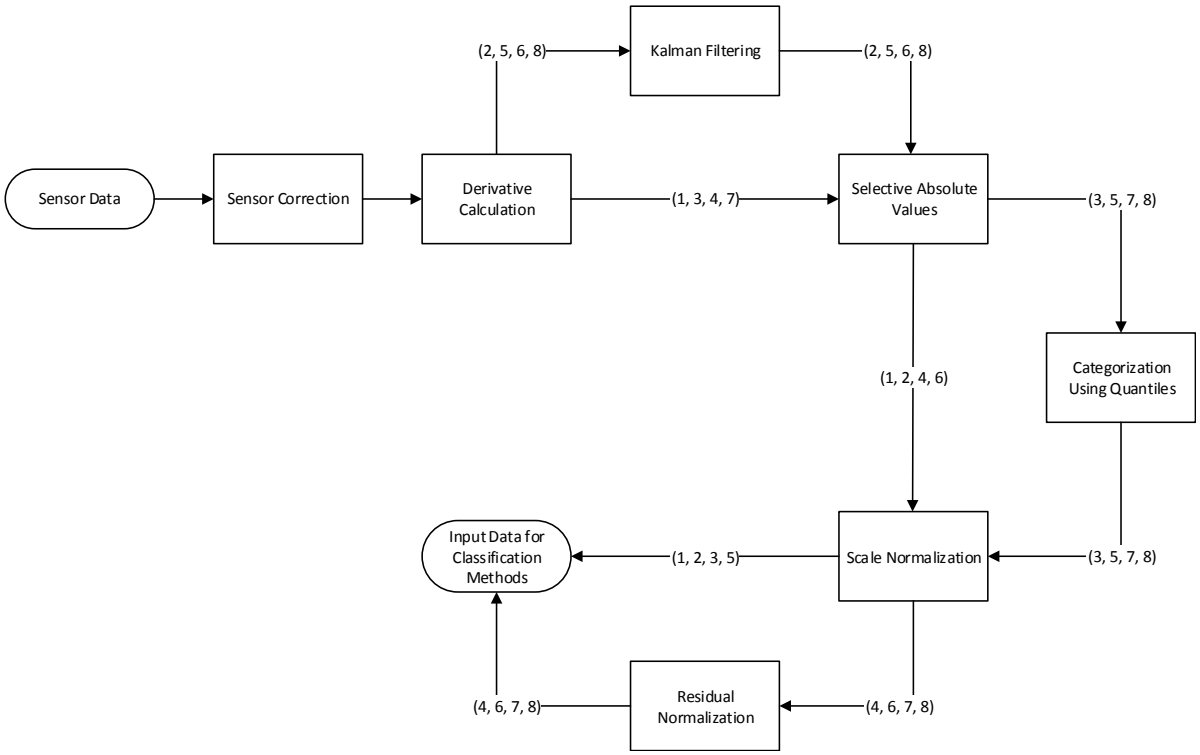

Figure 4.1: Pre-Processing Algorithm Flowchart

Table 4.2: Initial Input State

Input Dimension, i	Description
1	GPS Heading
2	GPS Speed
3	X Accelerometer
4	Y Accelerometer
5	Z Accelerometer
6	X Gyroscope
7	Y Gyroscope
8	Z Gyroscope

The first step to resolving the issues with the GPS heading measurements is to take its first differences. It is assumed first differences with an absolute value of more than 180° result from crossing the measurement interval bounds. To make the measurements continuous, 360° is subtracted from these differences which are positive, and 360° is added to these differences which are negative. The noise spikes are removed by taking a 95% normal confidence interval on the process mean of the first differences. Any first difference that has a value outside of this confidence interval is set to zero. This results in the removal of outliers caused by noise from the measurements. The GPS heading is then reconstructed by computing the cumulative sum of the first differences resulting from the above modifications.

The issue with the GPS speed measurements is simpler to resolve as they describe a rate of change, rather than a position. Therefore, first differences do not have to be taken. The noise spikes are removed by taking a 95% normal confidence interval on the process mean of the GPS speed measurements. Any GPS speed measurement that has a value outside of this confidence interval is set to the previous GPS speed measurement in the data. If no previous measurement exists, it is set to zero. The MATLAB code for the processing described in this section is located in Appendix B.3.

4.1.2 Derivative Calculation

The sensors used measure different aspects of the state of the vessel. The GPS partially measures the orientation as heading, and partially measures the first derivative of position as speed. The accelerometer measures the second derivative of position. The gyroscope measures the first derivative of orientation. It is desirable to supply the classification methods with as much useful information as possible. Therefore, it is desirable to differentiate measurements where possible, to a reasonable order. As the accelerometer provides second derivative measurements, all other measurements are differentiated up to the second derivative. This results in calculating the first and second derivatives of the GPS heading measurements, the first derivative of the GPS speed measurements, and the first derivatives of the gyroscope measurements. The derivatives are calculated using forward difference for the first value measured, backwards difference for the last value measured, and central difference for all other values measured. The changes made to the input are summarized in Table 4.3. The MATLAB code for the processing described in this section is located in Appendix B.4.

Table 4.3: Modifications to Input by Derivative Calculation

Input Dimension, i	Description	Modification
1	GPS Heading	Unaltered
2	First Derivative of GPS Heading	Added
3	Second Derivative of GPS Heading	Added
4	GPS Speed	Unaltered
5	First Derivative of GPS Speed	Added
6	X Accelerometer	Unaltered
7	Y Accelerometer	Unaltered
8	Z Accelerometer	Unaltered
9	X Gyroscope	Unaltered
10	Y Gyroscope	Unaltered
11	Z Gyroscope	Unaltered
12	First Derivative of X Gyroscope	Added
13	First Derivative of Y Gyroscope	Added
14	First Derivative of Z Gyroscope	Added

4.1.3 Kalman Filtering

Kalman filtering is a common technique for state estimation as it can combine multiple state indicators together to form a statistically optimal state estimate. The filter is applied recursively in two steps, the first being a prediction step, and the second being an update step. Equations 4.1 to 4.5 describe a Kalman filter with no control input, and have been adapted from [29].

Before the first iteration of the Kalman filter can be performed, initial values for the state estimate, $\mathbf{z}_0^u$, and its covariance matrix, $\mathbf{P}_0^u$, must be chosen. These values depend on the application and prior knowledge of the state. If the prior state is completely unknown, comprising the initial state estimate covariance matrix of large values will cause the state observations to quickly supersede the initial state estimate. The prediction stage of step k of the Kalman filter consists of predicting the state estimate,

$$\mathbf{z}_k^p = \mathbf{F}\mathbf{z}_{k-1}^u, \tag{4.1}$$

and predicting the state estimate covariance matrix,

$$\mathbf{P}_k^p = \mathbf{F}\mathbf{P}_{k-1}^u\mathbf{F}^T + \mathbf{Q}_{k-1}, \tag{4.2}$$

where $\mathbf{z}_{k-1}^u$ is the updated state estimate from step $k-1$, $\mathbf{F}$ is the state transition model, $\mathbf{z}_k^p$ is the predicted state estimate for step k, $\mathbf{Q}_{k-1}$ is the state transition model noise covariance matrix from step $k-1$, $\mathbf{P}_{k-1}^u$ is the updated state estimate covariance matrix from step $k-1$, and $\mathbf{P}_k^p$ is the predicted state estimate covariance matrix for step k.

The first step of the update stage is to calculate the optimal Kalman gain, $\mathbf{K}$. The equation for this calculation is

$$\mathbf{K} = \mathbf{P}_k^p \mathbf{H}^T \left(\mathbf{H} \mathbf{P}_k^p \mathbf{H}^T + \mathbf{R}_k \right)^{-1}, \tag{4.3}$$

where $\mathbf{H}$ is the observation model and $\mathbf{R}_k$ is the observation model noise covariance matrix for step k. The remaining steps of the update stage are to update the state estimate,

$$\mathbf{z}_k^u = \mathbf{z}_k^p + \mathbf{K} \left(\mathbf{x}_k - \mathbf{H} \mathbf{z}_k^p \right), \tag{4.4}$$

and update the state estimate covariance matrix,

$$\mathbf{P}_k^u = \left(\mathbf{I} - \mathbf{K}\mathbf{H} \right) \mathbf{P}_k^p, \tag{4.5}$$

where $\mathbf{x}_k$ is the state observation for step k, $\mathbf{z}_k^u$ is the updated state estimate for step k, $\mathbf{I}$ is the identity matrix of required size, and $\mathbf{P}_k^u$ is the updated state estimate covariance matrix for step k. It should be noted that there need not be a one to one relationship between prediction and update stages. For example, an application demanding high frequency state estimates but dependent on a low frequency state observer can predict as often as required, and update whenever possible.

The Kalman filter as used in this work is greatly simplified from the above description. In this work, the Kalman filter is utilized as a filter only, and not for state estimation. Furthermore, it is used on only one input dimension at a time, resulting in all vectors and matrices in the above description becoming scalars and all covariances becoming variances. The state transition model, F, and the observation model, H, become unity and can be omitted. Finally, the state transition noise variance, Q_i, and the observation noise variance, R_i, are assumed to be constant for each input dimension i. The values of these noise variances are shown in Table 4.4. These values were chosen by visually estimating the noise in each input dimension and its reduction through Kalman filtering. The initial value of the state estimate, z_0^u, is taken as zero. The initial value of the state estimate variance, P_0^u, is taken as 10^6.

The simplified prediction stage equations are

$$z_j^p = z_{j-1}^u, \tag{4.6}$$

and

$$P_j^p = P_{j-1}^u + Q_i, \tag{4.7}$$

where j is the data point in input dimension i being considered. The simplified

Table 4.4: Noise Variances for Kalman Filtering

Input Dimension, i	Description	Q_i	R_i
1	GPS Heading	10^{-5}	1
2	First Derivative of GPS Heading	10^{-4}	1
3	Second Derivative of GPS Heading	10^{-4}	1
4	GPS Speed	10^{-6}	1
5	First Derivative of GPS Speed	10^{-5}	1
6	X Accelerometer	10^{-3}	1
7	Y Accelerometer	10^{-3}	1
8	Z Accelerometer	10^{-3}	1
9	X Gyroscope	10^{-3}	1
10	Y Gyroscope	10^{-3}	1
11	Z Gyroscope	10^{-3}	1
12	First Derivative of X Gyroscope	10^{-2}	1
13	First Derivative of Y Gyroscope	10^{-2}	1
14	First Derivative of Z Gyroscope	10^{-2}	1

update stage equations are

$$K = P_j^p \left(P_j^p + R_i\right)^{-1}, \tag{4.8}$$

$$z_j^u = z_j^p + K\left(x_{ij} - z_j^p\right), \tag{4.9}$$

and

$$P_j^u = (1 - K)\, P_j^p, \tag{4.10}$$

where x_{ij} is the value of data point j in input dimension i. Finally, the equation

$$x'_{ij} = z_j^u \tag{4.11}$$

records the value of the Kalman filtered data point, x'_{ij}. After an input dimension has been Kalman filtered, its derivative is recalculated, if required, before being Kalman filtered itself. Algorithm 4.1 summarizes the process of Kalman filtering an input dimension. The MATLAB code for the processing described in this section is located in Appendix B.5.

4.1.4 Selective Absolute Values

The goal of the classification methods is to determine what type of turn, if any, is taking place. The direction of the turn is not required from the classification methods. Several input dimensions contain values that are positive or negative depending on the direction of the turn. Therefore, the values in these input dimensions are converted to absolute values. These input dimensions are the first

Algorithm 4.1: Kalman Filtering for Input Dimension i

Require: $Q_i \geq 0 \wedge R_i \geq 0$
  $z \leftarrow 0$
  $P \leftarrow 10^6$
  for all j **do**
    $P \leftarrow P + Q_i$
    $K \leftarrow P/\left(P + R_i\right)$
    $z \leftarrow z + K\left(x_{ij} - z\right)$
    $P \leftarrow \left(1 - K\right)P$
    $x_{ij} \leftarrow z$
  end for

Table 4.5: Modifications to Input by Selective Absolute Values

Input Dimension, i	Description	Modification
1	First Derivative of GPS Heading	Absolute Value
2	Second Derivative of GPS Heading	Absolute Value
3	GPS Speed	Unaltered
4	First Derivative of GPS Speed	Unaltered
5	X Accelerometer	Unaltered
6	Y Accelerometer	Absolute Value
7	Z Accelerometer	Unaltered
8	X Gyroscope	Absolute Value
9	Y Gyroscope	Unaltered
10	Z Gyroscope	Absolute Value
11	First Derivative of X Gyroscope	Absolute Value
12	First Derivative of Y Gyroscope	Unaltered
13	First Derivative of Z Gyroscope	Absolute Value

and second derivatives of GPS heading, the Y accelerometer, the X gyroscope and its derivative, and the Z gyroscope and its derivative. Furthermore, the GPS heading measurements are no longer useful to the pre-processing algorithms at this stage, and are removed from the input. The changes made to the input are summarized in Table 4.5. The MATLAB code for the processing described in this section is located in Appendix B.6.

4.1.5 Categorization using Quantiles

The presence of outliers in the data can be detrimental to the performance of classification methods. At this point in the pre-processing, steps have been taken to reduce the outliers generated by noisy GPS heading and speed measurements. However, it is possible that outliers were missed. It is also possible that outliers

Table 4.6: Example of Categorization using Two Quantiles

Input Values	$[1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 10]$
Quantiles	$[2.0, 3.5]$
Mapping	$x' = \begin{cases} 1, & \text{if } -\infty \leq x \leq 2.0 \\ 2, & \text{if } 2.0 < x \leq 3.5 \\ 3, & \text{if } 3.5 < x \leq \infty \end{cases}$
Output Values	$[1, 1, 1, 2, 2, 2, 3, 3, 3]$

exist in other input dimensions. Categorization using quantiles eliminates the issue of outliers in the data by replacing each value with the quantile that it belongs to. Quantiles were used in a similar manner in [15]. The number of quantiles used to categorize the data is a free parameter of the algorithm. However, it has an upper limit equal to the number of data points. The number of quantiles selected for this work is ten. The mapping equation is

$$x'_{ij} = \begin{cases} 1, & \text{if } -\infty \leq x_{ij} \leq q_{i,1} \\ 2, & \text{if } q_{i,1} < x_{ij} \leq q_{i,2} \\ \vdots & \\ n, & \text{if } q_{i,n-1} < x_{ij} \leq q_{i,n} \\ n+1, & \text{if } q_{i,n} < x_{ij} \leq \infty \end{cases}, \tag{4.12}$$

where x_{ij} is the value of data point j in input dimension i, n is the number of quantiles used for categorization, $q_{i,1} \ldots q_{i,n}$ are the values of the quantiles for input dimension i, and x'_{ij} is the value of the categorized data point. To illustrate this algorithm, a numerical example is presented in Table 4.6. The input data in this example contains an obvious outlier which is eliminated by the algorithm. The MATLAB code for the processing described in this section is located in Appendix B.7.

4.1.6 Scale Normalization

The units of measurement associated with each input dimension vary. Converting quantities to be non-dimensional is a common technique to aid in determining which quantities have the most influence on a system. Additionally, the maximum and minimum values of each input dimension vary. In order to easily calculate Euclidean distances between data points, as required by a classification method, all input dimensions must have the same minimum and maximum values. This is

achieved, along with making the quantity non-dimensional, by using the equation

$$x'_{ij} = \frac{x_{ij} - \min(\mathbf{x}_i)}{\max(\mathbf{x}_i) - \min(\mathbf{x}_i)}, \tag{4.13}$$

where x_{ij} is the value of data point j of input dimension i, $\mathbf{x}_i$ is the set of all data points in input dimension i, and x'_{ij} is the value of the scale normalized data point. After this equation has been applied to all data points, the minimum value in every input dimension is zero, and the maximum value in every input dimension is unity. Such scaling was suggested in [30]. The MATLAB code for the processing described in this section is located in Appendix B.8.

4.1.7 Residual Normalization

The equation to normalize residuals is,

$$x'_{ij} = \frac{x_{ij} - \overline{X}_i}{s_i}, \tag{4.14}$$

where $\overline{X_i}$ is the sample mean of input dimension i, s_i is the sample standard deviation of input dimension i, x_{ij} the value of data point j in input dimension i, and x'_{ij} is the value of the residual normalized data point. After this equation has been applied to all data points, the sample mean of every input dimension is zero, and the sample standard deviation of every input dimension is unity. This algorithm seeks to make each input dimension resemble the standard normal distribution. In [16], this was beneficial under certain conditions. The MATLAB code for the processing described in this section is located in Appendix B.9.

4.2 Classification Algorithms

This section describes the classification algorithms to be tested.

4.2.1 Multilayer Perceptron

The multilayer perceptron (MLP) is a form of feedforward neural network that has one or more hidden layers of neurons. Due to the performance of a MLP being dependent on the number of neurons used [8] and the number of hidden layers [9], two different sizes of MLP are tested. The details of these sizes are shown in Table 4.7. It is important to note that the values for the total weight and bias elements presented in the table include a bias element for each hidden and output neuron.

A committee of five MLPs is trained for each test. It is hoped that this will increase the total accuracy and address the problem of individual MLPs training

Table 4.7: Multilayer Perceptron Dimensions

Description	Neurons					Total Weight and Bias Elements
	Input Layer	Hidden Layer			Output Layer	
		1	2	3		
Large MLP	13	50	25	10	4	2279
Small MLP	13	10	5	–	4	219

poorly as a result of their initial conditions [10]. A total of four different ways to combine the results of the committee members are tested. Probability committees utilize the full result of each committee member, while answer committees utilize only the winning class result. Averaged committees simply average the results of the committee members, while networked committees utilize another MLP with twenty input neurons, four hidden neurons, and four output neurons to combine the results of the committee members. This results in four combinations and four types of committee.

Each MLP, including the MLP for a networked committee, is trained using the MATLAB implementation of the scaled conjugate gradient backpropagation algorithm presented in [31] for 250 epochs. The performance function to be minimized is the mean squared error. All neurons use the hyperbolic tangent sigmoid transfer function [32].

The identification of these classification methods is shown in Table 4.8. The MATLAB code for the large MLP variants is located in Appendix B.13. The MATLAB code for the small MLP variants is located in Appendix B.14.

4.2.2 Nearest-Neighbour Search

Nearest-neighbour search is a simple classification method that classifies an unknown data point based upon which training data point it is closest to, using some distance metric. The MATLAB implementation of the k–d tree algorithm presented in [33] is used to find the nearest neighbours. The distance metric used is the Euclidean distance. The number of data points in each leaf node of the k–d tree is set to 50. The identification of this classification method is shown in Table 4.8. The MATLAB code for this classification method is located in Appendix B.15.

4.3 Confidence Algorithms

Cross-validation is a common method for increasing the statistical significance of results, and is used in this work. The use of cross-validation enables the calculation

Table 4.8: Classification Method Identification

Number	Classification Method
1	Individual Large MLP
2	Averaged Probability Committee of Large MLPs
3	Averaged Answer Committee of Large MLPs
4	Networked Probability Committee of Large MLPs
5	Networked Answer Committee of Large MLPs
6	Nearest Neighbour Search
7	Individual Small MLP
8	Averaged Probability Committee of Small MLPs
9	Averaged Answer Committee of Small MLPs
10	Networked Probability Committee of Small MLPs
11	Networked Answer Committee of Small MLPs

of confidence metrics for the results. The following sections describe an algorithm to calculate bounded confidence intervals for the process mean of each result, and an algorithm that uses these intervals to determine the confidence in the ranking of each result.

4.3.1 Bounded Confidence Interval for Process Mean

The results of the classification algorithms for each cross-validation fold provide a set of values with which to construct confidence intervals. Due to the small number of cross-validation folds, the most reasonable probability density function to use is Student's t-distribution. This probability density function is

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{4.15}$$

where Γ is the gamma function and ν is the number of degrees of freedom. This function is useful on its own, but the quantities considered in this thesis are bounded by limits. For example, accuracies have a lower bound of zero and an upper bound of unity. Therefore, in order to make the confidence intervals more useful, these bounds must be enforced. The first step in enforcing these bounds is to calculate the Student's t-statistic of each of the bounds. The equations for these calculations are

$$t_{l,b} = (l_b - \overline{X})\frac{\sqrt{n}}{s} \tag{4.16a}$$

and

$$t_{u,b} = (u_b - \overline{X})\frac{\sqrt{n}}{s}, \tag{4.16b}$$

where $\overline{X}$ is the sample mean, s is the sample standard deviation, n is the number of data points in the sample, l_b is the lower bound, u_b is the upper bound, $t_{l,b}$ is the Student's t-statistic of the lower bound, and $t_{u,b}$ is the Student's t-statistic of the upper bound. With the Student's t-statistics of the upper and lower bound known, the bounds can be enforced by scaling up the probability density function so that unity probability lies between the bounds. The equation to be solved is

$$y \int_{t_{l,b}}^{t_{u,b}} f(t)\, dt = 1, \tag{4.17}$$

where y is the scaling factor to be solved for. This equation was inspired by the work presented in [34] and [35]. With the scaling factor known, the Student's t-statistics of the upper and lower limits of the confidence interval can be found. The equations to be solved are

$$y \int_{0}^{t_u} f(t)\, dt = \frac{p}{2} \tag{4.18a}$$

and

$$y \int_{t_l}^{0} f(t)\, dt = \frac{p}{2}, \tag{4.18b}$$

where p is the confidence interval probability (i.e. $p = 0.95$ produces a 95% confidence interval), t_u is the Student's t-statistic of the upper limit of the confidence interval, and t_l is the Student's t-statistic of the lower limit of the confidence interval. These equations put equal probability on each side of the sample mean are to be solved for t_u and t_l respectively. The above equations do not guarantee that $t_u \leq t_{u,b}$ or $t_l \geq t_{l,b}$, therefore the cases where one of these inequalities is not true must be handled. In both cases, a non-symmetric confidence interval will be produced. In the case where $t_u > t_{u,b}$, the equations

$$t_u = t_{u,b} \tag{4.19a}$$

and

$$y \int_{t_l}^{t_{u,b}} f(t)\, dt = p \tag{4.19b}$$

are applied. These equations produce a confidence interval bounded by the upper bound by solving for t_l again. In the case where $t_l < t_{l,b}$, the equations

$$t_l = t_{l,b} \tag{4.20a}$$

and

$$y \int_{t_{l,b}}^{t_u} f(t)\, dt = p \tag{4.20b}$$

are applied. These equations produce a confidence interval bounded by the lower bound by solving for t_u again. With the Student's t-statistics of the upper and lower bounds of the confidence interval known, the last step is to calculate their real values. These equations for these calculations are

$$l = \overline{X} + t_l \frac{s}{\sqrt{n}} \tag{4.21a}$$

and

$$u = \overline{X} + t_u \frac{s}{\sqrt{n}}, \tag{4.21b}$$

where l is the lower bound of the confidence interval, and u is the upper bound of the confidence interval. Algorithm 4.2 summarizes the calculation of these bounds.

4.3.2 Ranking Confidence

It is useful is rank accuracy results in terms of their sample mean. However, it is also useful to know with what confidence each rank is assigned to each result. Therefore, bounded confidence intervals for the process mean are used to obtain the value of this confidence. This value is obtained by reducing the probability contained within each confidence interval until the confidence interval for the result being considered does not intersect the confidence intervals of any lower-ranked results. The probability contained within each confidence interval when this condition is satisfied is taken as the confidence that the process mean of the result being considered is greater than the process mean of any lower ranked results. The value by which the probability is reduced each iteration is set to one percent. Table 4.9 contains fictional accuracy results, which will be used to illustrate the algorithm.

The values in Table 4.9 can be used to calculate the upper and lower confidence bounds of each result for all values of probability contained within the bounds. Figure 4.2 shows the values of these bounds, starting from 99% probability contained between them, decrementing by one percent per step, and ending at zero. The confidence in each rank can be easily determined by visual examination. The

Algorithm 4.2: Bounded Confidence Interval for Process Mean

Require: $n \geq 2 \wedge s \geq 0 \wedge 1 \geq p \geq 0$

$t_{l,b} \leftarrow (l_b - \overline{X})\frac{\sqrt{n}}{s}$

$t_{u,b} \leftarrow (u_b - \overline{X})\frac{\sqrt{n}}{s}$

$y \leftarrow$ solution of $y \int_{t_{l,b}}^{t_{u,b}} f(t)\, dt = 1$ for y

$t_u \leftarrow$ solution of $y \int_{0}^{t_u} f(t)\, dt = \frac{p}{2}$ for t_u

$t_l \leftarrow$ solution of $y \int_{t_l}^{0} f(t)\, dt = \frac{p}{2}$ for t_l

if $t_u > t_{u,b}$ **then**

  $t_u \leftarrow t_{u,b}$

  $t_l \leftarrow$ solution of $y \int_{t_l}^{t_{u,b}} f(t)\, dt = p$ for t_l

else if $t_l < t_{l,b}$ **then**

  $t_l \leftarrow t_{l,b}$

  $t_u \leftarrow$ solution of $y \int_{t_{l,b}}^{t_u} f(t)\, dt = p$ for t_u

end if

$l \leftarrow \overline{X} + t_l \frac{s}{\sqrt{n}}$

$u \leftarrow \overline{X} + t_u \frac{s}{\sqrt{n}}$

Table 4.9: Rank and Description of Fictional Accuracy Results

Rank	Sample Mean	Sample Standard Deviation	Sample Size
1	0.8960	0.056 83	5
2	0.7300	0.015 81	5
3	0.6000	0.5477	5

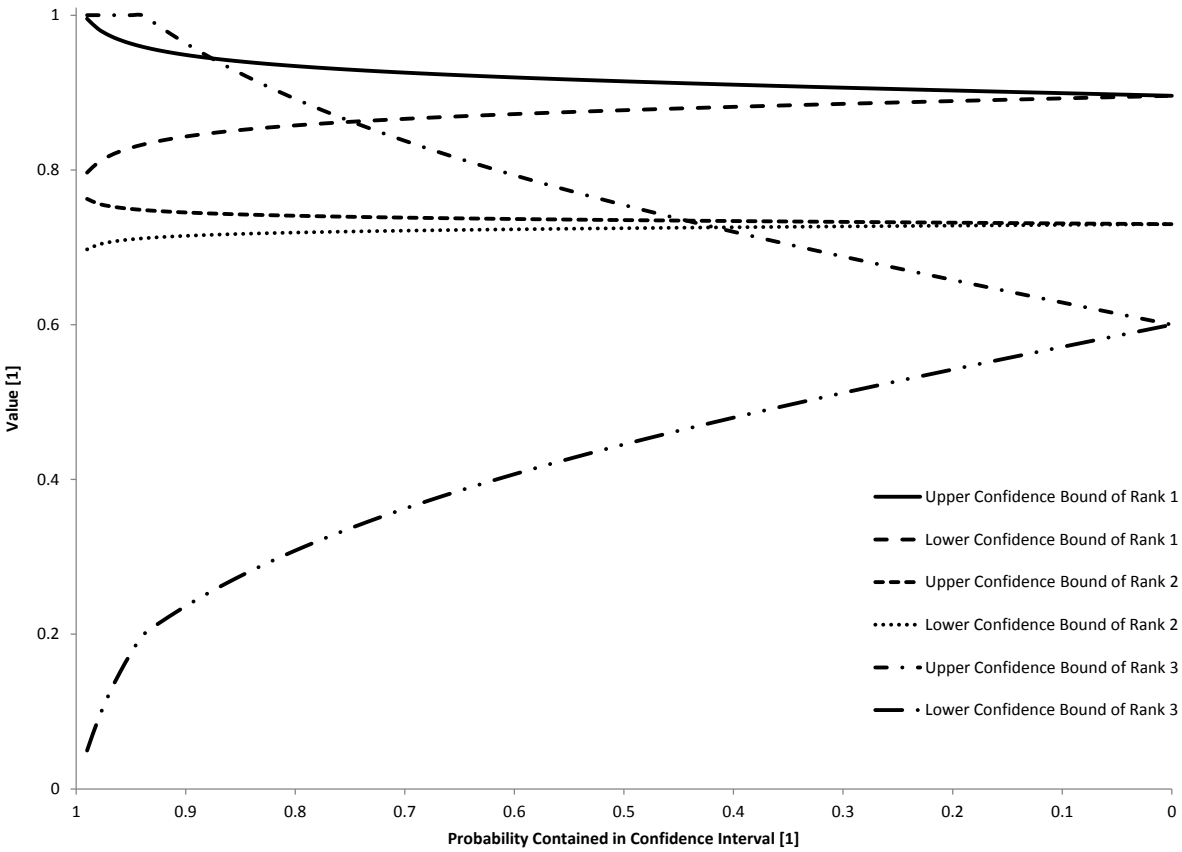

Figure 4.2: Bounded Confidence Intervals for Process Mean of Fictional Accuracy Results

confidence intervals of the first and second ranked results do not intersect below 99% probability contained between the confidence bounds, however the confidence intervals of the third ranked result intersect them both. At 74% probability contained between the confidence bounds, the upper bound of the third ranked result becomes less than the lower bound of the first ranked result. Therefore, the confidence that the first ranked result has the greatest process mean is 74%. At 41% probability contained between the confidence bounds, the upper bound of the third ranked result becomes less than the lower bound of the second ranked result. Therefore, the confidence that the second ranked result has a greater process mean than the third ranked result is 41%. As the third ranked result is the lowest ranked result, such a confidence cannot be defined. The confidence values for real results appear in brackets beside their rank.

5 Test Descriptions and Results

The classification methods identified in Table 4.8 were tested on the pre-processing algorithm combinations described in Table 4.1. This section presents the detailed description of each test, the results of each test, and a discussion of the results of each test. Each test is differentiated by what is considered as eligible training data, or what data is considered in general. The tests are done using five-fold cross-validation. The eligible training data for each test is split into five approximately equal sets. For each fold of the cross-validation, a different set is omitted from the training data presented to the classification methods. The purpose of this cross-validation is to enable estimation of the significance of the differences observed between results.

The eligible training data is split into sets uniformly and evenly. The first is made by taking approximately every fifth data point, starting with the first data point. The second set is made by taking approximately every fifth data point, starting with the second data point. This pattern was repeated to make the remaining three sets, and is equivalent to down-sampling by a factor of five with a different phase shift for each set. The reason why the eligible training data is split into sets in this manner is that the data has a temporal dependence. Data points located near the end of a turn may be very different than data points located near the start of a turn. If the eligible training data is split into sets randomly, the training of the classification methods may become temporally biased. Splitting the eligible training data into sets uniformly and evenly ensures that such bias does not manifest. It is important to note that the eligible training data itself must be selected in the same manner for this to be true.

The large proportion of data collected during straight sailing leads to a situation of class imbalance, with straight sailing as the dominant class by a large margin. There are several strategies for dealing with class imbalance [36]. In this case, there are both computational cost concerns and the desire to avoid inducing temporal bias as outlined above. Trial attempts which ignored the class imbalance failed to successfully complete due to the computational memory requirement exceeding the memory available on the test computer system. Therefore, the strategy employed is to under-sample the dominant class. This strategy addresses the computational cost concerns, and can be done in such a way that it does not induce temporal bias. The number of eligible training data points to be taken from the dominant class is set to the mean of the number of data points for each other class. Calculating this value from the values in Table 3.7, approximately 35,627 out of 922,696 straight sailing data points are to be considered as eligible training data. The selection of which data points are considered eligible training

data is done uniformly and evenly, by taking approximately every 26^{th} data point as eligible. The data points that are not selected are added to the testing data for all cross-validation folds.

All of the tests were run on a personal computer with 16 gigabytes of random access memory, and an Intel i7-2600 processor that has four physical cores running at 3.4 [GHz]. Three of these cores were allocated to MATLAB, allowing parallel processing of certain tasks. The increase in performance gained from the parallel processing can be estimated by comparing the real time results in Appendix A with the central processing unit (CPU) time results in this section. The CPU time results approximate the real time required for the task if only a single processor core was used. All time results presented are totals for the entire computation performed. Therefore, they are not normalized by the number of data points considered. The MATLAB code used to set up each test is located in Appendix B.12.

5.1 Test 1

The eligible training data for Test 1 is all data from all turns, along with the eligible training data selected by under-sampling the straight sailing data as previously described. This is the simplest way to select eligible training data, and is expected to give a baseline for subsequent tests.

5.1.1 Results

The accuracy results of Test 1 are presented in Table 5.1 to Table 5.6. The CPU time results of Test 1 are presented in Table 5.7 and Table 5.8.

Table 5.1: Mean Accuracy for Training Data, Test 1

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.5380	0.5421	0.5409	0.5387	0.5486	1.000
2	0.7255	0.7431	0.7412	0.7661	0.7557	1.000
3	0.6226	0.6433	0.6403	0.6510	0.6317	0.9980
4	0.5371	0.5425	0.5422	0.5630	0.5532	1.000
5	0.8677	0.9002	0.8987	0.8284	0.8333	1.000
6	0.7000	0.7439	0.7414	0.7693	0.7612	1.000
7	0.6224	0.6438	0.6418	0.6518	0.6449	0.9980
8	0.8515	0.8995	0.8971	0.9092	0.9047	1.000

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.4999	0.5210	0.5207	0.5529	0.5361
2	0.6572	0.6828	0.6790	0.6745	0.6955
3	0.5603	0.5731	0.5713	0.5630	0.5767
4	0.4958	0.5165	0.5165	0.5555	0.5304
5	0.6797	0.7286	0.7234	0.6939	0.6748
6	0.6300	0.6826	0.6796	0.6915	0.7004
7	0.5532	0.5728	0.5719	0.5854	0.5470
8	0.6781	0.7327	0.7276	0.7497	0.7204

Table 5.2: Standard Deviation of Accuracy for Training Data, Test 1 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	6.011	3.449	3.255	72.31	4.126	0
2	27.85	7.948	8.150	6.241	4.876	0
3	20.25	4.207	5.103	8.475	33.77	0.040 23
4	15.48	4.295	3.675	27.06	3.928	0
5	16.89	6.962	7.383	185.6	160.4	0
6	93.85	11.93	10.78	5.597	8.567	0
7	23.65	5.590	5.036	5.434	5.950	0.040 23
8	72.50	8.740	9.875	6.488	6.390	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	50.78	6.649	5.373	12.65	7.872
2	39.72	8.499	14.24	89.94	5.060
3	12.32	3.104	3.145	50.49	4.183
4	52.72	8.019	10.42	2.119	12.94
5	73.72	11.59	12.53	127.6	147.9
6	92.72	5.593	6.031	51.18	6.681
7	35.34	2.120	2.879	3.219	67.40
8	73.57	8.472	4.717	4.202	37.47

Table 5.3: Accuracy Ranking for Training Data, Test 1 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	73 (0.16)	70 (0.07)	71 (0.04)	72 (0.01)	66 (0.03)	1 (0.94)
2	25 (0.14)	19 (0.15)	21 (0.52)	14 (0.50)	16 (0.64)	1 (0.94)
3	51 (0.01)	46 (0.26)	48 (0.35)	43 (0.60)	49 (0.03)	2 (0.94)
4	74 (0.10)	68 (0.05)	69 (0.02)	59 (0.00)	64 (0.02)	1 (0.94)
5	9 (0.34)	5 (0.07)	7 (0.15)	12 (0.49)	11 (0.02)	1 (0.94)
6	29 (0.06)	18 (0.06)	20 (0.01)	13 (0.41)	15 (0.59)	1 (0.94)
7	52 (0.90)	45 (0.08)	47 (0.24)	42 (0.09)	44 (0.16)	2 (0.94)
8	10 (0.16)	6 (0.08)	8 (0.55)	3 (0.51)	4 (0.50)	1 (0.94)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	81 (0.15)	77 (0.04)	78 (0.41)	65 (0.12)	75 (0.42)
2	41 (0.38)	33 (0.01)	37 (0.03)	40 (0.26)	30 (0.01)
3	61 (0.29)	55 (0.09)	58 (0.25)	60 (0.07)	54 (0.39)
4	82 (1.00)	80 (0.74)	79 (0.00)	62 (0.20)	76 (0.65)
5	35 (0.00)	23 (0.10)	26 (0.09)	31 (0.02)	39 (0.00)
6	50 (0.25)	34 (0.08)	36 (0.05)	32 (0.14)	28 (0.01)
7	63 (0.01)	56 (0.30)	57 (0.15)	53 (0.59)	67 (0.09)
8	38 (0.03)	22 (0.32)	24 (0.21)	17 (0.53)	27 (0.26)

Table 5.4: Mean Accuracy for Testing Data, Test 1

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.7306	0.7377	0.7370	0.5656	0.7260	0.8363
2	0.7760	0.7895	0.7852	0.7712	0.7771	0.9898
3	0.6828	0.7039	0.7004	0.6728	0.6903	0.5597
4	0.7332	0.7414	0.7368	0.7066	0.7190	0.6952
5	0.8062	0.8387	0.8383	0.6844	0.8716	0.9275
6	0.7276	0.7867	0.7815	0.7761	0.7800	0.9793
7	0.6791	0.7012	0.6974	0.6772	0.6830	0.5756
8	0.7781	0.8376	0.8344	0.8499	0.8488	0.9308

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.7222	0.7625	0.7601	0.7137	0.7273
2	0.7963	0.8085	0.8055	0.7831	0.7835
3	0.6919	0.7093	0.7062	0.6905	0.6948
4	0.6978	0.7691	0.7627	0.7157	0.7456
5	0.7519	0.7817	0.7772	0.6167	0.6160
6	0.7662	0.8035	0.8041	0.7647	0.7884
7	0.6745	0.7109	0.7142	0.6888	0.7048
8	0.7239	0.7733	0.7688	0.7695	0.7653

Table 5.5: Standard Deviation of Accuracy for Testing Data, Test 1 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	12.81	5.150	5.063	309.1	10.62	0.5755
2	20.05	8.388	4.428	5.152	10.28	0.5224
3	16.07	8.859	6.955	3.850	31.66	0.7165
4	25.77	17.65	12.77	18.72	17.79	0.5812
5	11.81	3.589	3.672	373.9	40.00	0.4628
6	186.9	8.594	13.73	5.772	11.24	0.6205
7	12.68	11.45	7.876	3.514	10.31	0.7368
8	158.9	10.06	15.60	6.691	8.249	0.4892

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	150.9	16.44	13.30	6.991	16.23
2	27.49	18.22	17.46	39.95	12.11
3	10.43	6.971	7.400	11.67	12.72
4	208.1	16.36	11.43	10.80	15.56
5	17.28	13.30	8.274	336.7	337.3
6	164.1	20.04	16.13	39.06	11.10
7	138.3	22.54	10.36	10.84	15.52
8	151.2	10.65	10.94	5.107	5.497

Table 5.6: Accuracy Ranking for Testing Data, Test 1 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	51 (0.05)	47 (0.08)	48 (0.01)	87 (0.03)	54 (0.04)	11 (0.20)
2	32 (0.20)	19 (0.09)	22 (0.08)	34 (0.11)	30 (0.10)	1 (0.92)
3	79 (0.20)	66 (0.09)	68 (0.04)	83 (0.27)	75 (0.02)	88 (1.00)
4	50 (0.10)	46 (0.23)	49 (0.16)	63 (0.02)	57 (0.15)	71 (0.04)
5	14 (0.05)	8 (0.08)	9 (0.08)	77 (0.00)	5 (0.64)	4 (0.84)
6	52 (0.00)	21 (0.12)	26 (0.06)	31 (0.01)	27 (0.03)	2 (0.91)
7	80 (0.12)	67 (0.05)	70 (0.06)	81 (0.07)	78 (0.02)	86 (0.05)
8	28 (0.02)	10 (0.20)	12 (0.60)	6 (0.11)	7 (0.65)	3 (0.84)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	56 (0.06)	42 (0.13)	43 (0.33)	60 (0.13)	53 (0.07)
2	18 (0.36)	13 (0.13)	15 (0.07)	24 (0.04)	23 (0.01)
3	73 (0.03)	62 (0.11)	64 (0.10)	74 (0.00)	72 (0.04)
4	69 (0.00)	36 (0.01)	41 (0.01)	58 (0.12)	45 (0.20)
5	44 (0.31)	25 (0.01)	29 (0.00)	84 (0.00)	85 (0.13)
6	38 (0.01)	17 (0.36)	16 (0.03)	40 (0.06)	20 (0.14)
7	82 (0.04)	61 (0.09)	59 (0.04)	76 (0.02)	65 (0.05)
8	55 (0.02)	33 (0.14)	37 (0.05)	35 (0.03)	39 (0.02)

Table 5.7: Test 1, Mean Training CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	778.0	3890	3890	4323	4335	0
2	791.5	3957	3957	4401	4411	0
3	742.1	3710	3710	4152	4163	0
4	691.9	3459	3459	3882	3896	0
5	695.3	3476	3476	3899	3914	0
6	707.0	3535	3535	3958	3974	0
7	734.2	3671	3671	4099	4111	0
8	767.8	3839	3839	4276	4290	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	223.0	1115	1115	1516	1526
2	221.2	1106	1106	1509	1518
3	222.7	1114	1114	1517	1524
4	215.9	1079	1079	1484	1493
5	229.1	1145	1145	1554	1566
6	236.3	1181	1181	1595	1609
7	234.6	1173	1173	1587	1596
8	253.6	1268	1268	1695	1705

Table 5.8: Test 1, Mean Output CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	9.915	36.60	54.29	35.24	50.05	520.5
2	10.11	37.63	55.27	36.25	51.02	76.34
3	9.923	37.04	54.50	35.59	50.10	1545
4	9.374	34.91	51.30	33.57	47.31	2373
5	9.506	35.59	52.16	34.29	47.92	320.3
6	9.506	35.33	52.20	34.02	48.05	281.4
7	9.884	36.84	54.35	35.53	49.93	1838
8	10.22	38.29	56.13	36.95	51.63	386.0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	4.814	12.72	28.64	12.98	26.01
2	4.792	12.77	28.45	12.90	25.89
3	4.865	13.06	28.80	13.12	25.88
4	4.948	13.48	29.30	13.63	26.80
5	4.993	13.49	29.77	13.70	26.71
6	5.009	13.51	29.70	13.76	27.26
7	4.933	13.04	29.31	13.28	26.51
8	5.214	14.04	31.06	14.25	28.11

5.1.2 Discussion

The results in Table 5.1 reveal several interesting facts. Nearest-neighbour search is expected to have perfect training accuracy, however it did not have perfect training accuracy when categorization using quantiles was performed on the input. This indicates that the increased uniformity of distances between data points, resulting from categorization using quantiles, interfered with the performance of the nearest-neighbour search due to ties. When categorization using quantiles is not performed on the input, the values for each data point are real numbers rather than integers, greatly reducing the likelihood of ties. Nearest-neighbour search performed as expected in these cases. The best mean training accuracy, excluding nearest-neighbour search, was 90.92%, achieved by the networked probability committee of large MLPs with all additional pre-processing performed on the input. Some of the other large MLP variants are not far behind, but all small MLP variants performed considerably worse. The best mean training accuracy within the small MLP variants was 74.97%, achieved by the networked probability committee with all additional pre-processing performed on the input. This indicates that the reduced number of weight and bias elements present in the small MLPs, compared to the large MLPs, impairs their ability to fit the training data.

The results in Table 5.4 also reveal several interesting facts. The best mean testing accuracy was 98.98%, achieved by nearest-neighbour search with Kalman filtering. The best mean testing accuracy within the large MLP variants was 87.16%, achieved by the networked answer committee with Kalman filtering and categorization using quantiles. The best mean testing accuracy within the small MLP variants was 80.85%, achieved by the averaged probability committee with Kalman filtering. It is interesting to note that the top performing MLP variants for the training data are different than those for the testing data. This indicates that training accuracy does not necessarily translate into testing accuracy for this data. Additionally, the difference between the top performing large MLP variant and the top performing small MLP variant is smaller for the testing data than for the training data (6.31% versus 15.95%). This indicates that the increased training accuracy afforded by more weight and bias elements does not translate perfectly into testing accuracy.

The results in Table 5.7 indicate that it takes approximately three to four times as long to train large MLP variants than small MLP variants. The results in Table 5.8 indicate that the output time for the k–d algorithm used to perform nearest-neighbour search is heavily dependent on the pre-processing algorithms used. It took the minimum amount of time, 76.34 [s], with Kalman filtering alone. It took the maximum amount of time, 2373 [s], with residual normalization alone.

Therefore, it is likely far easier to partition the Kalman filtered input than the residual normalized input.

The testing accuracy results of this test make the MLP variants appear useless as nearest-neighbour search provides greatly superior accuracy. However, this was suspected to be an artefact of the way this test was constructed. In order to confirm this suspicion, the following test omits entire turns from the training data.

5.2 Test 2

The eligible training data for Test 2 is changed based upon the cross-validation fold. Every fifth turn of each class is omitted from the eligible training data. For the first cross-validation fold these omissions start with the first turn of each class. For the second cross-validation fold, these omissions start with the second turn of each class. This pattern is repeated for the remaining three cross-validation folds, and is equivalent to omitting the results of down-sampling by a factor of five with a different phase shift for each cross-validation fold. Data omitted in this way is described as hidden testing data. The remaining testing data is described as visible testing data.

5.2.1 Results

The accuracy results of Test 2 are presented in Table 5.9 to Table 5.17. The CPU time results of Test 2 are presented in Table 5.18 and Table 5.19.

Table 5.9: Mean Accuracy for Training Data, Test 2

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.5443	0.5589	0.5599	0.5872	0.5716	1.000
2	0.7291	0.7655	0.7637	0.7853	0.7752	1.000
3	0.6226	0.6656	0.6627	0.6453	0.6531	0.9983
4	0.5544	0.5592	0.5582	0.5876	0.5703	1.000
5	0.8698	0.9156	0.9132	0.8028	0.9261	1.000
6	0.7391	0.7640	0.7643	0.7850	0.7790	1.000
7	0.6464	0.6696	0.6670	0.6371	0.6720	0.9983
8	0.8484	0.9181	0.9166	0.9286	0.9253	1.000

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.5176	0.5379	0.5392	0.5547	0.5204
2	0.6466	0.6860	0.6805	0.7274	0.7117
3	0.5690	0.5895	0.5871	0.6019	0.5826
4	0.5321	0.5390	0.5372	0.5528	0.5189
5	0.6768	0.7539	0.7490	0.7500	0.7593
6	0.6550	0.6953	0.6937	0.7311	0.7147
7	0.5507	0.5885	0.5875	0.6044	0.5856
8	0.6951	0.7493	0.7426	0.7676	0.7563

Table 5.10: Standard Deviation of Accuracy for Training Data, Test 2 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	55.05	12.60	14.03	13.17	15.52	0
2	81.66	11.02	12.23	12.30	11.81	0
3	84.23	9.414	10.50	69.79	37.45	0.4692
4	16.01	14.33	14.74	14.15	16.16	0
5	82.58	9.724	11.18	198.3	5.576	0
6	50.43	14.95	11.91	10.76	9.671	0
7	30.95	12.89	12.13	95.91	11.01	0.4692
8	151.4	5.765	6.750	6.574	7.280	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	52.09	15.63	17.10	48.23	78.59
2	70.65	31.09	34.31	27.22	25.36
3	48.16	15.57	14.52	13.86	17.68
4	18.57	15.33	13.98	38.44	93.91
5	132.0	13.99	11.48	51.72	13.77
6	60.30	19.47	17.76	15.86	18.01
7	82.19	9.701	11.62	8.452	29.55
8	86.34	17.42	15.56	11.54	11.50

Table 5.11: Accuracy Ranking for Training Data, Test 2 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	74 (0.20)	68 (0.04)	66 (0.04)	59 (0.00)	63 (0.07)	1 (0.95)
2	31 (0.04)	18 (0.08)	21 (0.27)	13 (0.02)	16 (0.49)	1 (0.95)
3	52 (0.60)	44 (0.23)	45 (0.32)	50 (0.08)	47 (0.12)	2 (0.95)
4	71 (0.06)	67 (0.01)	69 (0.09)	57 (0.00)	64 (0.05)	1 (0.95)
5	10 (0.34)	8 (0.19)	9 (0.72)	12 (0.14)	4 (0.10)	1 (0.95)
6	29 (0.29)	20 (0.01)	19 (0.01)	14 (0.44)	15 (0.28)	1 (0.95)
7	49 (0.02)	42 (0.17)	43 (0.10)	51 (0.18)	41 (0.17)	2 (0.95)
8	11 (0.29)	6 (0.19)	7 (0.10)	3 (0.32)	5 (0.71)	1 (0.95)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	82 (1.00)	77 (0.03)	75 (0.01)	70 (0.00)	80 (0.01)
2	48 (0.00)	38 (0.13)	39 (0.06)	32 (0.43)	34 (0.42)
3	65 (0.33)	55 (0.06)	60 (0.05)	54 (0.55)	62 (0.50)
4	79 (0.23)	76 (0.05)	78 (0.27)	72 (0.04)	81 (0.01)
5	40 (0.11)	24 (0.10)	27 (0.37)	25 (0.01)	22 (0.19)
6	46 (0.04)	35 (0.00)	37 (0.25)	30 (0.06)	33 (0.11)
7	73 (0.18)	56 (0.06)	58 (0.01)	53 (0.18)	61 (0.10)
8	36 (0.04)	26 (0.01)	28 (0.15)	17 (0.15)	23 (0.15)

Table 5.12: Mean Accuracy for Visible Testing Data, Test 2

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.7902	0.7984	0.7898	0.7441	0.7712	0.8593
2	0.8144	0.8276	0.8248	0.8135	0.8187	0.9916
3	0.7023	0.7488	0.7452	0.7400	0.7509	0.6032
4	0.7838	0.7915	0.7874	0.7476	0.7775	0.7313
5	0.8099	0.8694	0.8672	0.9051	0.8896	0.9373
6	0.8162	0.8290	0.8257	0.8102	0.8205	0.9826
7	0.7242	0.7457	0.7422	0.5863	0.7225	0.6185
8	0.7803	0.8697	0.8686	0.8879	0.8857	0.9401

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.8144	0.8305	0.8210	0.7818	0.6307
2	0.7915	0.8360	0.8332	0.7991	0.8162
3	0.7448	0.7650	0.7612	0.7347	0.7441
4	0.8087	0.8197	0.8180	0.7774	0.6316
5	0.7220	0.8033	0.7995	0.7948	0.7860
6	0.8002	0.8393	0.8374	0.8092	0.8211
7	0.7242	0.7661	0.7587	0.7288	0.7238
8	0.7497	0.8062	0.7991	0.7922	0.7964

Table 5.13: Standard Deviation of Accuracy for Visible Testing Data, Test 2 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	41.75	17.83	9.422	17.94	15.70	4.633
2	21.70	19.78	19.46	21.35	27.61	0.2697
3	147.5	20.97	19.47	38.83	29.17	13.46
4	20.61	16.15	14.30	16.09	9.881	10.03
5	167.7	19.51	22.90	20.61	9.733	2.495
6	25.84	21.09	19.82	23.30	25.42	0.8449
7	16.29	12.38	11.64	320.8	13.40	12.81
8	232.0	15.57	20.27	8.907	11.22	2.368

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	41.03	14.13	8.637	33.49	347.1
2	166.1	22.49	23.27	30.26	23.45
3	20.67	12.04	8.009	8.677	21.57
4	20.85	16.29	14.75	34.83	348.0
5	219.3	18.46	10.24	21.38	22.30
6	166.9	22.35	21.39	28.76	29.65
7	157.8	17.67	10.86	13.29	35.83
8	152.5	21.44	14.95	28.00	39.54

Table 5.14: Accuracy Ranking for Visible Testing Data, Test 2 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	50 (0.02)	44 (0.05)	51 (0.14)	71 (0.00)	59 (0.25)	13 (0.81)
2	32 (0.04)	20 (0.08)	22 (0.12)	33 (0.06)	27 (0.02)	1 (0.96)
3	83 (0.29)	66 (0.05)	69 (0.02)	74 (0.18)	64 (0.02)	87 (0.08)
4	54 (0.05)	49 (0.06)	52 (0.06)	67 (0.10)	57 (0.00)	76 (0.14)
5	35 (0.01)	10 (0.03)	12 (0.44)	5 (0.68)	6 (0.14)	4 (0.91)
6	30 (0.10)	19 (0.05)	21 (0.03)	34 (0.00)	25 (0.03)	2 (0.95)
7	78 (0.00)	68 (0.02)	73 (0.07)	88 (1.00)	81 (0.00)	86 (0.16)
8	56 (0.03)	9 (0.01)	11 (0.05)	7 (0.18)	8 (0.70)	3 (0.73)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	31 (0.00)	18 (0.07)	24 (0.02)	55 (0.01)	85 (0.05)
2	48 (0.00)	16 (0.10)	17 (0.11)	43 (0.02)	29 (0.00)
3	70 (0.03)	61 (0.31)	62 (0.21)	75 (0.20)	72 (0.09)
4	37 (0.14)	26 (0.04)	28 (0.07)	58 (0.20)	84 (0.00)
5	82 (0.20)	39 (0.05)	41 (0.01)	46 (0.05)	53 (0.07)
6	40 (0.01)	14 (0.07)	15 (0.05)	36 (0.02)	23 (0.00)
7	79 (0.00)	60 (0.06)	63 (0.19)	77 (0.09)	80 (0.02)
8	65 (0.01)	38 (0.10)	42 (0.00)	47 (0.01)	45 (0.04)

Table 5.15: Mean Accuracy for Hidden Testing Data, Test 2

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.3645	0.3793	0.3786	0.4289	0.3981	0.2887
2	0.5136	0.5415	0.5387	0.5418	0.5410	0.3395
3	0.4009	0.4206	0.4200	0.3919	0.3903	0.3581
4	0.3849	0.3785	0.3806	0.4305	0.4001	0.3612
5	0.4586	0.4836	0.4826	0.3706	0.4376	0.3584
6	0.5309	0.5485	0.5458	0.5439	0.5378	0.3987
7	0.4136	0.4224	0.4207	0.4258	0.4186	0.3517
8	0.4442	0.4759	0.4710	0.4408	0.4330	0.3512

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.3408	0.3641	0.3609	0.3988	0.3926
2	0.4917	0.5090	0.5080	0.5293	0.5238
3	0.4088	0.4277	0.4267	0.4272	0.4236
4	0.3634	0.3683	0.3667	0.3891	0.3900
5	0.4876	0.5322	0.5315	0.5075	0.5241
6	0.5086	0.5337	0.5362	0.5489	0.5245
7	0.3987	0.4321	0.4349	0.4357	0.4201
8	0.4958	0.5218	0.5194	0.5089	0.5056

Table 5.16: Standard Deviation of Accuracy for Hidden Testing Data, Test 2 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	87.30	45.66	48.40	42.53	45.70	34.17
2	101.6	80.93	84.45	69.61	78.81	54.96
3	69.06	45.37	46.93	57.39	61.74	33.99
4	46.84	48.67	52.82	48.51	45.20	36.27
5	44.47	48.53	46.28	117.3	27.69	42.98
6	78.66	76.14	74.36	83.10	80.23	49.61
7	44.91	47.65	47.04	55.89	48.78	34.11
8	69.89	41.82	43.20	25.09	26.39	43.49

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	82.83	36.89	35.76	37.30	72.14
2	78.16	83.19	81.72	74.43	71.89
3	71.10	50.06	48.22	51.23	56.38
4	35.43	30.04	26.81	49.15	61.95
5	114.1	55.22	49.92	45.19	53.38
6	85.34	86.56	90.34	74.72	77.66
7	100.3	47.52	48.70	49.05	57.15
8	52.59	59.73	53.14	52.23	65.43

Table 5.17: Accuracy Ranking for Hidden Testing Data, Test 2 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	77 (0.00)	71 (0.01)	72 (0.00)	44 (0.02)	63 (0.07)	88 (1.00)
2	21 (0.06)	6 (0.00)	8 (0.00)	5 (0.00)	7 (0.02)	87 (0.72)
3	58 (0.01)	52 (0.00)	54 (0.02)	65 (0.02)	66 (0.00)	83 (0.14)
4	69 (0.09)	73 (0.07)	70 (0.02)	43 (0.02)	59 (0.02)	80 (0.00)
5	35 (0.46)	31 (0.01)	32 (0.12)	74 (0.02)	38 (0.04)	82 (0.00)
6	14 (0.02)	2 (0.02)	3 (0.01)	4 (0.02)	9 (0.01)	61 (0.00)
7	56 (0.16)	50 (0.02)	51 (0.00)	48 (0.03)	55 (0.12)	84 (0.01)
8	36 (0.10)	33 (0.09)	34 (0.32)	37 (0.10)	41 (0.02)	85 (0.19)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	86 (0.02)	78 (0.02)	81 (0.05)	60 (0.00)	64 (0.00)
2	29 (0.08)	22 (0.00)	25 (0.00)	15 (0.05)	18 (0.02)
3	57 (0.19)	45 (0.00)	47 (0.01)	46 (0.00)	49 (0.02)
4	79 (0.07)	75 (0.04)	76 (0.05)	68 (0.10)	67 (0.01)
5	30 (0.07)	12 (0.01)	13 (0.01)	26 (0.02)	17 (0.00)
6	24 (0.00)	11 (0.01)	10 (0.02)	1 (0.00)	16 (0.00)
7	62 (0.01)	42 (0.02)	40 (0.04)	39 (0.01)	53 (0.00)
8	28 (0.12)	19 (0.03)	20 (0.10)	23 (0.00)	27 (0.18)

Table 5.18: Test 2, Mean Training CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	683.7	3418	3418	3815	3845	0
2	712.5	3563	3563	3992	4001	0
3	682.8	3414	3414	3816	3830	0
4	596.1	2980	2980	3351	3368	0
5	599.0	2995	2995	3368	3381	0
6	707.4	3537	3537	3958	3970	0
7	639.9	3200	3200	3585	3600	0
8	622.8	3114	3114	3498	3509	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	221.5	1107	1107	1497	1510
2	206.3	1032	1032	1397	1404
3	194.4	972.2	972.2	1329	1342
4	187.8	938.9	938.9	1291	1300
5	182.2	911.1	911.1	1266	1277
6	188.4	942.2	942.2	1296	1307
7	188.6	943.2	943.2	1297	1306
8	188.9	944.3	944.3	1299	1310

Table 5.19: Test 2, Mean Output CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	10.11	37.64	55.29	37.63	52.47	555.9
2	10.67	39.82	58.34	39.91	55.59	103.8
3	10.28	38.58	56.71	38.57	53.35	1769
4	9.474	35.47	51.72	35.42	49.08	2572
5	9.519	35.72	52.37	35.71	49.51	371.2
6	10.70	39.95	58.75	39.88	55.69	336.9
7	10.06	37.92	55.14	37.97	52.11	1679
8	9.959	37.63	54.78	37.51	51.76	367.7
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	5.327	14.30	31.37	14.42	28.56	
2	5.099	13.72	30.23	13.92	27.57	
3	4.898	13.11	29.09	13.11	26.07	
4	4.807	12.84	28.28	13.23	26.02	
5	4.861	13.08	29.17	12.99	26.23	
6	4.813	12.85	28.56	12.97	25.89	
7	4.875	13.13	29.05	13.15	26.26	
8	4.927	13.42	29.36	13.52	26.34	

5.2.2 Discussion

When comparing results, it should be noted that all testing data in Test 1 is described as visible testing data. The training accuracy results in Table 5.9 show the same top performers and patterns as in Test 1. The visible testing accuracy results in Table 5.12 show the same patterns as in Test 1, but the top performing MLP variants have changed. The best mean visible testing accuracy within the large MLP variants was 90.51%, achieved by the networked probability committee with Kalman filtering and categorization using quantiles. The best mean visible testing accuracy within the small MLP variants was 83.93%, achieved by the averaged answer committee with Kalman filtering and residual normalization.

The results in Table 5.15 confirm the suspicions that Test 1 was constructed in a way that was biased towards nearest-neighbour search. The classification of hidden testing data is the most difficult task for the classifiers, and the hidden testing accuracy values support this claim. The best mean hidden testing accuracy was 54.89%, achieved by the networked probability committee of small MLPs with Kalman filtering and residual normalization. The best mean hidden testing accuracy within the large MLP variants was 54.85%, achieved by the averaged probability committee with Kalman filtering and residual normalization. The best mean hidden testing accuracy using nearest-neighbour search was 39.87%, achieved with Kalman filtering and residual normalization. It is clear from these results that the MLP variants are very useful, and nearest-neighbour search encounters difficulty classifying hidden testing data.

Subsequent tests focus on improving hidden testing accuracy and deciphering exactly why it is so poor compared to visible testing accuracy.

5.3 Test 3

The eligible training data for each cross-validation fold of Test 3 starts as the eligible training data from each cross-validation fold of Test 2. However, it is modified so that an approximately equal number of data points of each class are presented as training data in each cross-validation fold. The rationale behind this is that presenting a class in too large of a proportion may impair the learning process. The values in Table 3.7 show that the least represented class is gybes, with 25,524 data points. Therefore, the training data of the other classes must be under-sampled in a similar manner as the straight sailing class in Test 1 and Test 2. The exact details of this under-sampling depend on the cross-validation fold as different turns are omitted in each fold. Training data omitted by this under-sampling is taken as visible testing data.

Table 5.20: Mean Accuracy for Training Data, Test 3

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.5436	0.5516	0.5477	0.5862	0.5158	1.000
2	0.7456	0.7647	0.7616	0.7844	0.7786	1.000
3	0.6466	0.6738	0.6705	0.6363	0.6399	0.9987
4	0.5321	0.5463	0.5473	0.5769	0.5641	1.000
5	0.8478	0.9226	0.9212	0.9343	0.8827	1.000
6	0.7475	0.7660	0.7637	0.7888	0.7791	1.000
7	0.6507	0.6731	0.6705	0.6577	0.6742	0.9987
8	0.8835	0.9277	0.9266	0.9361	0.9334	1.000
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	0.5066	0.5232	0.5225	0.5473	0.5414	
2	0.6396	0.6862	0.6757	0.6659	0.7087	
3	0.5567	0.5745	0.5724	0.5914	0.5809	
4	0.5123	0.5285	0.5272	0.5549	0.5406	
5	0.7047	0.7592	0.7548	0.7781	0.7680	
6	0.6587	0.6815	0.6755	0.6903	0.7012	
7	0.5461	0.5742	0.5705	0.5909	0.5812	
8	0.6724	0.7615	0.7570	0.7772	0.7375	

5.3.1 Results

The accuracy results of Test 3 are presented in Table 5.20 to Table 5.28. The CPU time results of Test 3 are presented in Table 5.29 and Table 5.30.

Table 5.21: Standard Deviation of Accuracy for Training Data, Test 3 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	15.39	8.391	8.132	10.57	71.40	0
2	21.26	11.69	13.87	17.85	9.228	0
3	29.63	10.64	10.45	103.9	81.34	0.3915
4	39.42	14.60	11.13	16.76	14.76	0
5	145.1	6.556	7.630	4.291	107.9	0
6	24.59	17.99	17.48	20.26	20.77	0
7	15.47	9.806	9.085	46.73	6.746	0.3915
8	68.28	3.688	3.738	2.170	2.462	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	27.35	14.71	11.65	41.79	14.42
2	75.40	23.55	28.30	142.7	16.97
3	18.40	15.61	15.79	16.05	13.52
4	14.87	9.748	8.129	14.95	8.847
5	69.85	15.30	17.81	17.22	18.32
6	32.52	18.93	19.67	70.73	16.64
7	34.98	9.059	10.35	10.67	10.14
8	121.5	9.977	12.66	7.794	70.23

Table 5.22: Accuracy Ranking for Training Data, Test 3 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	72 (0.17)	66 (0.14)	67 (0.01)	55 (0.35)	80 (0.07)	1 (0.96)
2	29 (0.16)	21 (0.05)	23 (0.00)	14 (0.22)	16 (0.03)	1 (0.96)
3	49 (0.11)	40 (0.03)	43 (0.00)	52 (0.55)	50 (0.00)	2 (0.95)
4	75 (0.22)	70 (0.01)	69 (0.06)	58 (0.11)	63 (0.45)	1 (0.96)
5	12 (0.84)	8 (0.15)	9 (0.52)	4 (0.23)	11 (0.34)	1 (0.96)
6	28 (0.16)	20 (0.07)	22 (0.11)	13 (0.18)	15 (0.02)	1 (0.96)
7	48 (0.20)	41 (0.01)	44 (0.05)	47 (0.21)	39 (0.04)	2 (0.95)
8	10 (0.00)	6 (0.25)	7 (0.56)	3 (0.42)	5 (0.66)	1 (0.96)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	82 (1.00)	78 (0.04)	79 (0.13)	68 (0.00)	73 (0.05)
2	51 (0.03)	35 (0.18)	37 (0.00)	45 (0.07)	31 (0.14)
3	64 (0.13)	59 (0.02)	61 (0.11)	53 (0.02)	57 (0.22)
4	81 (0.49)	76 (0.11)	77 (0.23)	65 (0.22)	74 (0.47)
5	32 (0.12)	25 (0.13)	27 (0.31)	17 (0.05)	19 (0.09)
6	46 (0.02)	36 (0.16)	38 (0.07)	34 (0.07)	33 (0.20)
7	71 (0.18)	60 (0.12)	62 (0.40)	54 (0.34)	56 (0.01)
8	42 (0.05)	24 (0.15)	26 (0.11)	18 (0.52)	30 (0.49)

Table 5.23: Mean Accuracy for Visible Testing Data, Test 3

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.7043	0.7135	0.7117	0.6753	0.6898	0.8053
2	0.7713	0.7850	0.7825	0.7726	0.7965	0.9796
3	0.6388	0.6757	0.6740	0.5443	0.6906	0.5535
4	0.7089	0.7220	0.7153	0.6722	0.6998	0.6826
5	0.7295	0.8496	0.8401	0.8765	0.8813	0.8885
6	0.7696	0.7839	0.7811	0.7832	0.7917	0.9613
7	0.6651	0.6860	0.6829	0.6656	0.6691	0.5680
8	0.8330	0.8627	0.8645	0.8799	0.8793	0.8925

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.6941	0.7257	0.7246	0.6893	0.7118
2	0.7440	0.7796	0.7786	0.7791	0.7674
3	0.6736	0.6943	0.6917	0.6616	0.6708
4	0.7044	0.7269	0.7286	0.6879	0.6875
5	0.7423	0.7705	0.7687	0.7694	0.7716
6	0.7716	0.7866	0.7834	0.6172	0.7861
7	0.6765	0.6979	0.6961	0.6665	0.6693
8	0.7219	0.7664	0.7688	0.7626	0.7649

Table 5.24: Standard Deviation of Accuracy for Visible Testing Data, Test 3 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	19.64	18.60	19.52	20.52	43.93	6.655
2	24.81	23.73	23.17	19.81	22.64	1.781
3	128.1	25.89	22.71	294.0	27.58	10.20
4	32.90	17.83	13.17	21.76	17.02	7.870
5	266.7	9.750	15.58	6.856	8.417	4.339
6	28.81	30.09	30.72	23.24	20.80	1.591
7	13.55	11.78	12.13	19.09	11.34	9.484
8	10.77	6.552	6.726	4.235	5.134	4.214

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	125.9	29.15	25.14	14.13	24.80
2	152.8	30.74	26.29	50.71	22.00
3	17.63	11.78	11.55	19.69	14.58
4	47.86	16.61	17.73	41.11	64.49
5	29.49	23.48	22.40	27.28	39.46
6	22.18	22.62	21.04	327.7	33.06
7	28.61	19.33	15.92	20.13	16.52
8	151.5	38.71	36.58	26.87	33.88

Table 5.25: Accuracy Ranking for Visible Testing Data, Test 3 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	56 (0.27)	51 (0.06)	53 (0.14)	73 (0.04)	64 (0.01)	14 (0.45)
2	31 (0.04)	19 (0.03)	23 (0.04)	28 (0.02)	15 (0.18)	1 (0.96)
3	84 (0.09)	72 (0.01)	74 (0.02)	88 (1.00)	63 (0.01)	87 (0.05)
4	54 (0.21)	48 (0.00)	50 (0.09)	76 (0.06)	57 (0.08)	70 (0.32)
5	43 (0.01)	11 (0.55)	12 (0.52)	8 (0.87)	5 (0.18)	4 (0.72)
6	33 (0.00)	20 (0.01)	24 (0.04)	22 (0.02)	16 (0.17)	2 (0.95)
7	82 (0.22)	68 (0.21)	69 (0.02)	81 (0.02)	79 (0.13)	86 (0.13)
8	13 (0.80)	10 (0.85)	9 (0.22)	6 (0.09)	7 (0.37)	3 (0.64)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	61 (0.06)	46 (0.03)	47 (0.04)	65 (0.03)	52 (0.00)
2	41 (0.03)	25 (0.00)	27 (0.17)	26 (0.01)	37 (0.02)
3	75 (0.07)	60 (0.00)	62 (0.04)	83 (0.21)	77 (0.08)
4	55 (0.00)	45 (0.04)	44 (0.08)	66 (0.00)	67 (0.03)
5	42 (0.16)	32 (0.03)	36 (0.04)	34 (0.01)	30 (0.00)
6	29 (0.00)	17 (0.01)	21 (0.00)	85 (0.19)	18 (0.03)
7	71 (0.05)	58 (0.08)	59 (0.04)	80 (0.04)	78 (0.01)
8	49 (0.14)	38 (0.03)	35 (0.00)	40 (0.32)	39 (0.06)

Table 5.26: Mean Accuracy for Hidden Testing Data, Test 3

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.4088	0.4164	0.4151	0.4247	0.3859	0.3093
2	0.5418	0.5528	0.5526	0.5291	0.5354	0.3676
3	0.4082	0.4133	0.4144	0.4106	0.4034	0.3605
4	0.3854	0.3915	0.4022	0.4212	0.4023	0.3693
5	0.4626	0.4846	0.4900	0.4460	0.4240	0.3854
6	0.5393	0.5524	0.5504	0.5418	0.5381	0.4239
7	0.4041	0.4113	0.4122	0.3851	0.4083	0.3542
8	0.4400	0.4688	0.4605	0.4323	0.4318	0.3799

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.3857	0.3996	0.3953	0.3909	0.4030
2	0.5251	0.5541	0.5513	0.4725	0.5377
3	0.4215	0.4304	0.4330	0.4278	0.4285
4	0.3922	0.4038	0.4080	0.4099	0.4143
5	0.4936	0.5290	0.5250	0.5206	0.5206
6	0.5472	0.5665	0.5684	0.5667	0.5477
7	0.4037	0.4174	0.4199	0.4094	0.4149
8	0.4689	0.5314	0.5215	0.5165	0.4562

Table 5.27: Standard Deviation of Accuracy for Hidden Testing Data, Test 3 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	39.39	39.16	39.18	49.46	103.8	32.82
2	70.43	76.22	73.94	74.76	77.75	53.15
3	51.13	56.23	54.88	57.48	49.22	34.32
4	56.28	29.07	40.72	57.40	45.53	34.81
5	38.91	40.23	38.56	26.00	55.28	41.96
6	70.61	81.37	79.24	72.72	74.80	50.63
7	45.50	52.57	52.25	98.48	51.15	33.32
8	33.73	32.23	31.32	31.64	35.23	42.78

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	49.37	41.06	35.75	75.41	33.00
2	77.80	82.31	81.45	181.1	78.00
3	48.66	50.95	52.74	47.53	46.67
4	50.98	44.59	51.11	59.66	50.49
5	67.34	56.34	52.70	55.46	59.60
6	76.27	89.66	84.36	99.73	92.24
7	49.57	48.17	47.42	42.53	39.99
8	112.1	49.52	46.29	36.59	112.8

Table 5.28: Accuracy Ranking for Hidden Testing Data, Test 3 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	62 (0.01)	51 (0.02)	52 (0.00)	44 (0.01)	78 (0.00)	88 (1.00)
2	13 (0.05)	5 (0.00)	6 (0.00)	19 (0.00)	17 (0.05)	85 (0.13)
3	64 (0.00)	56 (0.01)	54 (0.00)	59 (0.00)	69 (0.00)	86 (0.15)
4	81 (0.00)	76 (0.00)	72 (0.05)	48 (0.02)	71 (0.00)	84 (0.03)
5	33 (0.07)	29 (0.09)	28 (0.11)	36 (0.26)	45 (0.00)	80 (0.00)
6	14 (0.01)	7 (0.01)	9 (0.02)	12 (0.00)	15 (0.00)	46 (0.04)
7	66 (0.00)	58 (0.01)	57 (0.01)	82 (0.06)	63 (0.00)	87 (0.79)
8	37 (0.21)	32 (0.14)	34 (0.04)	39 (0.01)	40 (0.02)	83 (0.21)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	79 (0.00)	73 (0.09)	74 (0.06)	77 (0.04)	70 (0.01)
2	21 (0.00)	4 (0.01)	8 (0.00)	30 (0.02)	16 (0.02)
3	47 (0.00)	41 (0.03)	38 (0.01)	43 (0.05)	42 (0.01)
4	75 (0.02)	67 (0.00)	65 (0.07)	60 (0.00)	55 (0.01)
5	27 (0.08)	20 (0.06)	22 (0.05)	24 (0.00)	25 (0.07)
6	11 (0.08)	3 (0.11)	1 (0.01)	2 (0.00)	10 (0.00)
7	68 (0.00)	50 (0.01)	49 (0.04)	61 (0.01)	53 (0.01)
8	31 (0.00)	18 (0.03)	23 (0.01)	26 (0.32)	35 (0.12)

Table 5.29: Test 3, Mean Training CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	420.4	2102	2102	2445	2458	0
2	421.0	2105	2105	2450	2462	0
3	427.7	2138	2138	2487	2496	0
4	427.8	2139	2139	2481	2493	0
5	426.3	2132	2132	2477	2483	0
6	423.2	2116	2116	2459	2474	0
7	436.3	2182	2182	2535	2546	0
8	479.6	2398	2398	2716	2749	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	126.9	634.4	634.4	932.8	945.1	
2	126.4	631.8	631.8	928.3	941.4	
3	126.8	633.9	633.9	933.9	943.5	
4	125.9	629.5	629.5	927.9	941.2	
5	127.2	636.0	636.0	934.2	949.8	
6	126.2	630.8	630.8	930.8	941.5	
7	126.7	633.7	633.7	934.2	946.9	
8	123.9	619.7	619.7	918.0	930.4	

Table 5.30: Test 3, Mean Output CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	9.599	35.94	52.57	36.05	50.14	392.6
2	9.670	36.24	52.94	36.39	49.97	95.59
3	9.832	36.91	54.07	37.07	51.17	1264
4	9.851	37.03	54.00	37.16	51.20	1952
5	9.731	36.60	53.73	36.85	50.64	368.4
6	9.789	36.80	53.48	36.90	50.83	368.7
7	9.857	36.83	54.21	36.90	51.04	1195
8	11.07	41.19	61.00	40.99	57.57	348.9
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	4.863	13.08	28.78	13.50	26.35	
2	4.907	13.26	29.01	13.43	26.16	
3	4.810	12.84	28.64	12.98	25.89	
4	4.851	13.06	28.71	13.43	26.00	
5	4.941	13.38	29.52	13.38	26.75	
6	4.828	12.92	28.49	13.46	25.95	
7	4.947	13.46	29.42	13.69	26.59	
8	4.941	13.51	29.66	13.50	26.55	

5.3.2 Discussion

The results in Table 5.26 show small gains in hidden testing accuracy compared to Test 2. The best mean hidden testing accuracy was 56.84%, achieved by the averaged answer committee of small MLPs with Kalman filtering and residual normalization. This is an absolute gain of 1.95% hidden testing accuracy. The best mean hidden testing accuracy within the large MLP variants was 55.28%, achieved by the averaged probability committee with Kalman filtering. The best mean hidden testing accuracy using nearest-neighbour search was 42.39%, achieved with Kalman filtering and residual normalization.

While a small increase in hidden testing accuracy was observed, it is obvious that some factor is preventing the hidden testing accuracy from approaching the visible testing accuracy. It is possible that this factor is simply a lack of sufficient data. However, it was suspected that lack of environmental data, such as wind conditions, could also be to blame. In order to confirm this suspicion, the following test considers data from only a single day.

5.4 Test 4

Test 4 follows the same patterns of eligible training data selection as Test 2, but only considers data from Aug. 9, 2012. Therefore, both the testing and training data originate from this day only. The reason this day was chosen is that it was the day in which the most data was collected. The rationale behind this test is that the environmental conditions over a few hours should be more uniform than the environmental conditions over two months. If a marked increase in hidden testing accuracy is observed, it would imply that a lack of environmental data impaired the hidden testing accuracy of Test 2 and Test 3.

5.4.1 Results

The accuracy results of Test 4 are presented in Table 5.31 to Table 5.39. The CPU time results of Test 4 are presented in Table 5.40 and Table 5.41.

Table 5.31: Mean Accuracy for Training Data, Test 4

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.7061	0.7287	0.7274	0.7465	0.7378	1.000
2	0.9277	0.9522	0.9516	0.9732	0.9652	1.000
3	0.7753	0.7963	0.7946	0.8040	0.7725	0.9999
4	0.6846	0.7252	0.7229	0.7448	0.7362	1.000
5	0.9654	0.9846	0.9849	0.9940	0.9919	1.000
6	0.9341	0.9532	0.9519	0.9731	0.9308	1.000
7	0.7667	0.8013	0.7999	0.7882	0.8053	0.9999
8	0.9577	0.9842	0.9847	0.9926	0.9911	1.000

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.6487	0.6781	0.6755	0.7027	0.6932
2	0.7861	0.8977	0.8896	0.9247	0.9196
3	0.6747	0.7006	0.6984	0.7133	0.7096
4	0.6465	0.6771	0.6744	0.7037	0.6993
5	0.7974	0.9126	0.9049	0.9057	0.9320
6	0.8633	0.8986	0.8986	0.9383	0.9259
7	0.6246	0.6971	0.6900	0.7072	0.7044
8	0.8545	0.9214	0.9189	0.9092	0.9355

Table 5.32: Standard Deviation of Accuracy for Training Data, Test 4 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	56.57	30.66	31.25	27.83	28.67	0
2	50.81	9.411	9.936	8.087	9.011	0
3	24.63	25.42	24.79	25.43	59.64	0.078 67
4	95.01	26.54	26.08	21.14	21.71	0
5	50.27	2.805	2.957	1.463	2.482	0
6	48.63	13.16	13.19	10.35	91.87	0
7	53.55	22.43	22.53	57.20	22.92	0.078 67
8	60.52	3.958	3.704	3.559	4.273	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	58.50	37.68	40.01	30.87	31.87
2	208.5	22.38	28.17	20.52	23.56
3	39.77	24.69	25.04	24.88	24.64
4	46.50	33.67	35.72	28.76	26.68
5	195.9	26.35	26.65	75.60	18.95
6	52.85	23.66	24.72	17.64	18.81
7	124.8	20.98	15.89	18.98	22.46
8	103.1	19.48	19.85	77.67	19.40

Table 5.33: Accuracy Ranking for Training Data, Test 4 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	64 (0.06)	57 (0.03)	58 (0.06)	53 (0.05)	55 (0.05)	1 (0.90)
2	25 (0.07)	17 (0.02)	19 (0.35)	11 (0.00)	14 (0.34)	1 (0.90)
3	50 (0.06)	46 (0.05)	47 (0.12)	42 (0.09)	51 (0.11)	2 (0.90)
4	74 (0.13)	59 (0.07)	60 (0.30)	54 (0.23)	56 (0.23)	1 (0.90)
5	13 (0.01)	9 (0.11)	7 (0.04)	3 (0.43)	5 (0.19)	1 (0.90)
6	22 (0.05)	16 (0.06)	18 (0.02)	12 (0.38)	24 (0.04)	1 (0.90)
7	52 (0.58)	43 (0.05)	44 (0.03)	48 (0.02)	41 (0.04)	2 (0.90)
8	15 (0.19)	10 (0.78)	8 (0.00)	4 (0.19)	6 (0.83)	1 (0.90)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	80 (0.08)	75 (0.02)	77 (0.02)	67 (0.06)	72 (0.11)
2	49 (0.15)	37 (0.26)	38 (0.67)	27 (0.13)	29 (0.02)
3	78 (0.01)	68 (0.04)	70 (0.04)	61 (0.12)	62 (0.09)
4	81 (0.47)	76 (0.03)	79 (0.61)	66 (0.02)	69 (0.03)
5	45 (0.01)	31 (0.05)	34 (0.20)	33 (0.01)	23 (0.01)
6	39 (0.21)	35 (0.00)	36 (0.03)	20 (0.12)	26 (0.05)
7	82 (1.00)	71 (0.12)	73 (0.15)	63 (0.03)	65 (0.02)
8	40 (0.65)	28 (0.07)	30 (0.16)	32 (0.03)	21 (0.05)

Table 5.34: Mean Accuracy for Visible Testing Data, Test 4

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.6291	0.6568	0.6614	0.6766	0.6840	0.8412
2	0.8688	0.8877	0.8854	0.9332	0.9341	0.9914
3	0.6821	0.7008	0.7009	0.7077	0.7281	0.5560
4	0.6417	0.6699	0.6701	0.6707	0.6737	0.6848
5	0.9284	0.9449	0.9449	0.9730	0.9704	0.9371
6	0.8726	0.8910	0.8884	0.9350	0.9359	0.9829
7	0.6313	0.6926	0.6901	0.7247	0.7166	0.5704
8	0.9283	0.9436	0.9440	0.9686	0.9682	0.9401

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.6034	0.6334	0.6342	0.6477	0.6345
2	0.7474	0.8346	0.8325	0.8524	0.8602
3	0.6248	0.6455	0.6423	0.6729	0.6694
4	0.6122	0.6457	0.6386	0.6538	0.6637
5	0.7896	0.8527	0.8544	0.8841	0.8838
6	0.8063	0.8398	0.8368	0.8758	0.8725
7	0.5841	0.6378	0.6351	0.6554	0.6632
8	0.7830	0.8505	0.8429	0.8876	0.8840

Table 5.35: Standard Deviation of Accuracy for Visible Testing Data, Test 4 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	132.4	22.76	16.77	12.74	27.08	9.737
2	30.68	18.01	18.92	14.21	13.24	1.015
3	12.36	10.04	9.919	14.55	46.14	19.28
4	139.1	37.76	28.86	19.47	13.12	15.75
5	14.35	4.628	4.248	3.756	4.601	2.758
6	33.17	24.00	24.46	22.22	16.58	1.712
7	186.0	14.83	14.24	35.04	5.704	18.67
8	11.40	7.706	7.184	8.867	9.544	2.634

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	132.8	36.39	39.97	20.74	26.54
2	227.5	18.53	22.08	33.63	37.90
3	43.99	19.54	18.42	17.46	14.36
4	93.99	36.53	44.66	22.69	11.27
5	170.8	24.74	30.18	25.01	20.75
6	30.89	27.58	32.86	31.50	25.77
7	178.6	27.72	30.75	18.81	22.97
8	163.6	23.51	26.46	26.54	24.58

Table 5.36: Accuracy Ranking for Visible Testing Data, Test 4 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	82 (0.09)	67 (0.05)	66 (0.19)	57 (0.18)	55 (0.10)	37 (0.06)
2	30 (0.27)	21 (0.00)	23 (0.05)	16 (0.37)	15 (0.05)	1 (0.99)
3	56 (0.43)	51 (0.49)	50 (0.01)	49 (0.43)	46 (0.06)	88 (1.00)
4	74 (0.05)	62 (0.01)	61 (0.00)	60 (0.02)	58 (0.04)	54 (0.02)
5	17 (0.01)	8 (0.12)	7 (0.00)	3 (0.48)	4 (0.21)	12 (0.10)
6	28 (0.00)	19 (0.08)	20 (0.02)	14 (0.04)	13 (0.03)	2 (0.95)
7	81 (0.02)	52 (0.14)	53 (0.24)	47 (0.32)	48 (0.61)	87 (0.55)
8	18 (0.96)	10 (0.44)	9 (0.04)	5 (0.03)	6 (0.92)	11 (0.33)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	85 (0.24)	80 (0.03)	79 (0.01)	70 (0.05)	78 (0.00)
2	45 (0.22)	40 (0.08)	41 (0.64)	34 (0.05)	31 (0.14)
3	83 (0.34)	72 (0.08)	73 (0.01)	59 (0.09)	63 (0.26)
4	84 (0.15)	71 (0.00)	75 (0.01)	69 (0.22)	64 (0.02)
5	43 (0.07)	33 (0.00)	32 (0.05)	24 (0.00)	26 (0.24)
6	42 (0.31)	38 (0.08)	39 (0.07)	27 (0.09)	29 (0.15)
7	86 (0.23)	76 (0.07)	77 (0.01)	68 (0.06)	65 (0.07)
8	44 (0.34)	35 (0.24)	36 (0.07)	22 (0.08)	25 (0.00)

Table 5.37: Mean Accuracy for Hidden Testing Data, Test 4

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.5568	0.5644	0.5635	0.5372	0.5407	0.3564
2	0.5609	0.5856	0.5799	0.4738	0.4711	0.4118
3	0.4866	0.4949	0.4946	0.4844	0.4621	0.4544
4	0.5479	0.5676	0.5643	0.5516	0.5562	0.4585
5	0.4352	0.4577	0.4533	0.3404	0.3380	0.4376
6	0.5700	0.5837	0.5793	0.4782	0.4791	0.4946
7	0.4896	0.4912	0.4865	0.4500	0.4627	0.4475
8	0.4327	0.4521	0.4497	0.3355	0.3294	0.4335

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.5527	0.5672	0.5679	0.5633	0.5837
2	0.5438	0.6250	0.6322	0.5729	0.5544
3	0.5168	0.5347	0.5320	0.5214	0.5288
4	0.5237	0.5435	0.5437	0.5637	0.5599
5	0.4508	0.5325	0.5113	0.4608	0.4649
6	0.5839	0.6166	0.6057	0.5007	0.5128
7	0.4985	0.5409	0.5354	0.5267	0.5150
8	0.4985	0.5231	0.5217	0.4194	0.4400

Table 5.38: Standard Deviation of Accuracy for Hidden Testing Data, Test 4 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	82.11	93.81	93.71	94.56	96.22	64.36
2	125.9	139.1	139.8	132.6	129.5	118.3
3	62.11	71.00	69.75	72.13	78.69	53.07
4	129.9	102.5	95.47	96.92	88.90	65.49
5	112.5	137.2	134.7	117.8	113.9	72.86
6	139.5	145.0	144.7	153.7	139.2	121.7
7	75.14	60.97	61.80	66.28	69.24	51.50
8	118.5	138.0	139.6	102.8	97.90	71.95

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	88.21	97.84	94.89	75.94	98.46
2	169.2	136.3	132.0	152.7	148.4
3	64.21	58.03	57.52	66.16	68.85
4	95.27	73.79	77.97	89.55	97.07
5	184.2	138.9	147.0	144.3	121.5
6	138.4	160.1	168.3	121.8	122.4
7	95.83	55.44	63.80	71.62	65.28
8	124.8	128.1	134.1	159.6	136.8

Table 5.39: Accuracy Ranking for Hidden Testing Data, Test 4 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	23 (0.00)	16 (0.00)	19 (0.00)	34 (0.01)	33 (0.03)	84 (0.14)
2	21 (0.01)	5 (0.01)	9 (0.00)	62 (0.01)	63 (0.04)	83 (0.46)
3	57 (0.00)	52 (0.00)	54 (0.04)	59 (0.04)	66 (0.00)	70 (0.01)
4	28 (0.05)	14 (0.00)	17 (0.00)	27 (0.04)	24 (0.01)	68 (0.00)
5	79 (0.02)	69 (0.02)	71 (0.00)	85 (0.01)	86 (0.01)	78 (0.03)
6	12 (0.02)	8 (0.02)	10 (0.03)	61 (0.02)	60 (0.00)	53 (0.00)
7	56 (0.05)	55 (0.02)	58 (0.02)	74 (0.00)	65 (0.00)	76 (0.06)
8	81 (0.10)	72 (0.00)	75 (0.01)	87 (0.05)	88 (1.00)	80 (0.01)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	26 (0.01)	15 (0.02)	13 (0.00)	20 (0.03)	7 (0.00)
2	29 (0.00)	2 (0.04)	1 (0.04)	11 (0.02)	25 (0.01)
3	45 (0.03)	36 (0.01)	38 (0.04)	44 (0.07)	39 (0.02)
4	41 (0.00)	31 (0.02)	30 (0.00)	18 (0.00)	22 (0.03)
5	73 (0.00)	37 (0.00)	48 (0.06)	67 (0.01)	64 (0.01)
6	6 (0.00)	3 (0.05)	4 (0.10)	49 (0.02)	47 (0.00)
7	50 (0.00)	32 (0.00)	35 (0.01)	40 (0.02)	46 (0.01)
8	51 (0.03)	42 (0.00)	43 (0.00)	82 (0.04)	77 (0.01)

Table 5.40: Test 4, Mean Training CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	147.6	738.1	738.1	871.6	875.8	0
2	150.0	750.2	750.2	890.4	890.8	0
3	191.5	957.7	957.7	1139	1137	0
4	191.5	957.7	957.7	1131	1140	0
5	189.3	946.5	946.5	1121	1126	0
6	167.6	838.2	838.2	988.5	988.6	0
7	152.7	763.3	763.3	896.8	899.8	0
8	154.1	770.4	770.4	907.6	909.2	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	44.17	220.8	220.8	342.8	347.0
2	41.48	207.4	207.4	327.7	330.6
3	43.24	216.2	216.2	334.7	339.8
4	43.26	216.3	216.3	334.5	338.0
5	43.56	217.8	217.8	335.8	339.3
6	43.17	215.9	215.9	333.6	336.8
7	43.35	216.7	216.7	333.2	337.6
8	43.36	216.8	216.8	333.9	337.0

Table 5.41: Test 4, Mean Output CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	3.015	11.59	16.16	12.01	15.92	58.87
2	3.038	11.89	16.29	12.12	16.01	10.95
3	3.687	13.82	20.14	14.01	19.18	153.6
4	3.355	13.08	18.09	13.32	17.79	182.5
5	3.541	13.48	19.02	13.62	18.69	57.22
6	3.285	12.82	17.54	13.23	17.22	41.85
7	3.287	12.89	17.62	13.21	17.26	189.0
8	3.247	12.79	17.31	13.01	16.92	58.02
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	1.548	3.903	8.983	4.153	8.337	
2	1.504	3.891	8.742	4.171	8.265	
3	1.548	4.012	8.911	4.318	8.383	
4	1.491	3.738	8.624	3.978	8.096	
5	1.493	3.769	8.627	4.100	8.296	
6	1.478	3.785	8.511	4.028	8.131	
7	1.531	3.872	8.929	4.162	8.415	
8	1.473	3.813	8.477	4.303	8.252	

5.4.2 Discussion

The results in Table 5.37 show significant gains in hidden testing accuracy compared to Test 3. The best mean hidden testing accuracy was 63.22%, achieved by the averaged answer committee of small MLPs with Kalman filtering. This is an absolute gain of 6.38% hidden testing accuracy. The best mean hidden testing accuracy within the large MLP variants was 58.56%, achieved by the averaged probability committee with Kalman filtering. The best mean hidden testing accuracy using nearest-neighbour search was 49.46%, achieved with Kalman filtering and residual normalization.

The significant increase in hidden testing accuracy implies that additional environmental data would increase the hidden testing accuracy of Test 2 and Test 3. While the evidence of this implication is not overwhelming, it is significant enough to warrant investigation in future work.

5.5 Test 5

In order to be thorough, Test 5 follows the same patterns of eligible training data selection as Test 3, but only considers data from Aug. 9, 2012.

5.5.1 Results

The accuracy results of Test 5 are presented in Table 5.42 to Table 5.50. The CPU time results of Test 5 are presented in Table 5.51 and Table 5.52.

Table 5.42: Mean Accuracy for Training Data, Test 5

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.7155	0.7351	0.7329	0.5638	0.7477	1.000
2	0.9213	0.9557	0.9550	0.9752	0.9687	1.000
3	0.7489	0.8066	0.8064	0.7838	0.8174	0.9999
4	0.7213	0.7375	0.7366	0.7577	0.7523	1.000
5	0.9332	0.9891	0.9890	0.9480	0.9957	1.000
6	0.9023	0.9528	0.9526	0.9739	0.9658	1.000
7	0.7764	0.8119	0.8100	0.8227	0.6996	0.9999
8	0.9195	0.9866	0.9820	0.9472	0.9452	1.000

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.6710	0.6932	0.6916	0.6675	0.7059
2	0.8600	0.8963	0.8932	0.9305	0.9201
3	0.6608	0.7129	0.7116	0.7255	0.7207
4	0.6687	0.6942	0.6912	0.7217	0.7091
5	0.8708	0.9212	0.9189	0.9468	0.9389
6	0.8415	0.8996	0.8958	0.9353	0.7760
7	0.6522	0.7089	0.7064	0.7311	0.6325
8	0.8707	0.9268	0.9253	0.9486	0.9404

Table 5.43: Standard Deviation of Accuracy for Training Data, Test 5 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	73.03	39.23	38.82	270.0	35.90	0
2	120.1	10.47	11.22	11.45	9.883	0
3	104.3	27.86	29.17	93.54	27.57	0.057 11
4	53.90	40.58	40.14	39.74	41.75	0
5	143.8	3.225	3.903	111.7	2.729	0
6	132.2	16.10	16.88	12.15	14.38	0
7	57.35	25.46	27.06	27.11	158.6	0.057 11
8	167.3	6.633	16.86	110.4	112.5	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	49.33	40.12	41.46	124.2	37.84
2	90.85	33.89	33.45	23.28	27.82
3	102.3	34.78	35.32	33.67	37.17
4	52.52	40.90	41.62	38.00	39.89
5	100.8	24.23	25.37	19.68	20.63
6	96.29	28.29	32.99	24.62	216.9
7	132.6	42.84	42.76	37.06	214.0
8	103.8	19.55	20.44	15.58	20.06

Table 5.44: Accuracy Ranking for Training Data, Test 5 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	64 (0.06)	57 (0.04)	58 (0.03)	82 (1.00)	54 (0.21)	1 (0.78)
2	28 (0.00)	12 (0.05)	13 (0.11)	8 (0.09)	10 (0.19)	1 (0.78)
3	53 (0.02)	46 (0.00)	47 (0.21)	48 (0.04)	43 (0.17)	2 (0.78)
4	62 (0.01)	55 (0.01)	56 (0.03)	51 (0.10)	52 (0.06)	1 (0.78)
5	24 (0.05)	4 (0.03)	5 (0.35)	17 (0.00)	3 (0.72)	1 (0.78)
6	33 (0.05)	14 (0.00)	15 (0.07)	9 (0.37)	11 (0.27)	1 (0.78)
7	49 (0.00)	44 (0.06)	45 (0.10)	42 (0.16)	71 (0.04)	2 (0.78)
8	31 (0.01)	6 (0.32)	7 (0.39)	18 (0.00)	20 (0.07)	1 (0.78)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	76 (0.04)	73 (0.03)	74 (0.00)	78 (0.06)	70 (0.05)
2	40 (0.37)	35 (0.01)	37 (0.45)	25 (0.14)	30 (0.00)
3	79 (0.14)	65 (0.03)	66 (0.05)	60 (0.09)	63 (0.12)
4	77 (0.01)	72 (0.02)	75 (0.23)	61 (0.00)	67 (0.00)
5	38 (0.00)	29 (0.02)	32 (0.33)	19 (0.02)	22 (0.11)
6	41 (0.41)	34 (0.09)	36 (0.06)	23 (0.04)	50 (0.12)
7	80 (0.12)	68 (0.04)	69 (0.01)	59 (0.13)	81 (0.23)
8	39 (0.21)	26 (0.06)	27 (0.09)	16 (0.00)	21 (0.05)

Table 5.45: Mean Accuracy for Visible Testing Data, Test 5

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.6052	0.6128	0.6110	0.3935	0.6372	0.7771
2	0.8270	0.8762	0.8724	0.9229	0.9163	0.9779
3	0.6110	0.6555	0.6513	0.5428	0.6642	0.5028
4	0.6048	0.6125	0.6079	0.6393	0.6451	0.6283
5	0.8813	0.9353	0.9343	0.7856	0.9647	0.8853
6	0.8610	0.8768	0.8753	0.9194	0.9137	0.9583
7	0.6488	0.6677	0.6667	0.6685	0.7268	0.5165
8	0.8430	0.9258	0.9100	0.7800	0.9623	0.8890

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.5231	0.5318	0.5357	0.4877	0.5834
2	0.8003	0.8300	0.8308	0.8539	0.8476
3	0.5589	0.5859	0.5848	0.6252	0.6319
4	0.5411	0.5461	0.5486	0.6042	0.5788
5	0.8001	0.8369	0.8383	0.8706	0.8629
6	0.7999	0.8334	0.8339	0.8499	0.8355
7	0.5392	0.5900	0.5745	0.6293	0.4899
8	0.8084	0.8429	0.8473	0.8713	0.8685

Table 5.46: Standard Deviation of Accuracy for Visible Testing Data, Test 5 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	19.32	17.93	13.79	345.8	15.55	11.84
2	172.6	20.14	20.26	24.20	18.87	4.120
3	146.2	11.23	15.51	281.2	22.83	9.656
4	30.89	18.43	12.41	15.06	9.192	9.242
5	177.5	8.787	12.99	416.9	11.54	10.06
6	34.80	29.95	33.02	24.13	22.77	4.576
7	29.88	14.11	15.20	9.161	75.06	9.597
8	243.9	24.59	63.23	414.8	10.64	9.956

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	64.29	12.90	12.95	259.5	12.49
2	33.57	21.48	21.10	32.90	38.64
3	122.2	17.42	16.97	13.15	14.35
4	39.60	21.35	22.00	20.21	20.73
5	33.10	14.68	17.28	25.78	21.89
6	42.92	17.04	21.54	40.82	63.11
7	158.0	25.26	16.70	7.961	262.4
8	30.85	17.80	20.63	15.05	26.01

Table 5.47: Accuracy Ranking for Visible Testing Data, Test 5 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	66 (0.03)	61 (0.01)	64 (0.19)	88 (1.00)	56 (0.28)	45 (0.73)
2	38 (0.16)	17 (0.02)	19 (0.05)	8 (0.12)	10 (0.10)	1 (0.75)
3	63 (0.00)	51 (0.25)	52 (0.15)	78 (0.00)	50 (0.40)	85 (0.08)
4	67 (0.03)	62 (0.03)	65 (0.18)	55 (0.11)	54 (0.38)	59 (0.22)
5	15 (0.07)	5 (0.08)	6 (0.35)	43 (0.01)	2 (0.18)	14 (0.07)
6	24 (0.24)	16 (0.01)	18 (0.09)	9 (0.12)	11 (0.07)	4 (0.70)
7	53 (0.27)	48 (0.05)	49 (0.11)	47 (0.05)	46 (0.69)	84 (0.16)
8	29 (0.00)	7 (0.09)	12 (0.33)	44 (0.01)	3 (0.40)	13 (0.15)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	83 (0.19)	82 (0.25)	81 (0.24)	87 (0.26)	72 (0.22)
2	40 (0.01)	37 (0.05)	36 (0.02)	25 (0.09)	27 (0.00)
3	75 (0.08)	70 (0.05)	71 (0.07)	60 (0.30)	57 (0.19)
4	79 (0.03)	77 (0.01)	76 (0.03)	68 (0.33)	73 (0.19)
5	41 (0.00)	32 (0.03)	31 (0.05)	21 (0.06)	23 (0.08)
6	42 (0.06)	35 (0.11)	34 (0.02)	26 (0.04)	33 (0.03)
7	80 (0.07)	69 (0.16)	74 (0.17)	58 (0.09)	86 (0.00)
8	39 (0.10)	30 (0.15)	28 (0.05)	20 (0.02)	22 (0.19)

Table 5.48: Mean Accuracy for Hidden Testing Data, Test 5

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	0.5710	0.5875	0.5874	0.4669	0.5745	0.3792
2	0.5818	0.6032	0.6074	0.5173	0.5302	0.4633
3	0.4883	0.5150	0.5149	0.5015	0.4951	0.4656
4	0.5710	0.5874	0.5838	0.5473	0.5522	0.4744
5	0.4259	0.4717	0.4612	0.3783	0.3381	0.4731
6	0.5769	0.6103	0.6048	0.4979	0.5308	0.5319
7	0.4893	0.5070	0.5070	0.4902	0.3631	0.4615
8	0.4537	0.4749	0.4799	0.3838	0.3037	0.4718

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.5854	0.6131	0.6101	0.5918	0.6047
2	0.5912	0.6155	0.6166	0.5472	0.5594
3	0.5350	0.5664	0.5598	0.5409	0.5411
4	0.5670	0.6018	0.5950	0.5671	0.5872
5	0.4913	0.5348	0.5306	0.4552	0.4710
6	0.5747	0.5948	0.5965	0.5618	0.5108
7	0.5242	0.5431	0.5517	0.5288	0.5137
8	0.4817	0.5299	0.5166	0.4834	0.4777

Table 5.49: Standard Deviation of Accuracy for Hidden Testing Data, Test 5 [10^3]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	100.6	96.64	98.36	181.2	90.87	54.85
2	147.9	157.7	158.9	168.4	149.0	114.3
3	83.66	61.69	61.89	69.98	67.03	47.17
4	79.64	92.91	91.12	76.15	85.33	57.39
5	122.1	126.3	116.2	165.6	106.9	66.86
6	169.8	165.9	166.6	157.0	147.2	128.6
7	58.77	60.81	59.32	57.96	180.8	44.68
8	151.7	134.3	153.3	111.8	139.0	65.86

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	90.98	96.24	93.94	83.27	98.06
2	115.9	102.5	98.67	132.2	109.5
3	71.91	63.61	59.00	50.88	50.59
4	87.53	85.64	80.56	77.64	76.00
5	106.6	124.3	120.1	113.8	102.5
6	109.0	117.1	112.6	143.3	113.6
7	61.79	61.85	64.33	67.22	60.62
8	146.1	130.5	132.8	133.0	151.5

Table 5.50: Accuracy Ranking for Hidden Testing Data, Test 5 (Confidence)

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	26 (0.00)	16 (0.00)	17 (0.00)	75 (0.00)	25 (0.04)	84 (0.00)
2	22 (0.06)	9 (0.00)	6 (0.01)	50 (0.00)	46 (0.00)	77 (0.01)
3	64 (0.04)	52 (0.00)	53 (0.01)	58 (0.02)	60 (0.05)	76 (0.02)
4	27 (0.05)	18 (0.00)	21 (0.02)	36 (0.00)	34 (0.00)	70 (0.01)
5	82 (0.35)	73 (0.00)	79 (0.04)	85 (0.07)	87 (0.23)	71 (0.01)
6	23 (0.02)	4 (0.00)	7 (0.00)	59 (0.02)	44 (0.00)	43 (0.00)
7	63 (0.02)	56 (0.00)	57 (0.07)	62 (0.01)	86 (0.14)	78 (0.00)
8	81 (0.38)	69 (0.00)	67 (0.01)	83 (0.03)	88 (1.00)	72 (0.00)

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	20 (0.02)	3 (0.01)	5 (0.01)	14 (0.00)	8 (0.00)
2	15 (0.04)	2 (0.02)	1 (0.00)	37 (0.03)	33 (0.06)
3	41 (0.00)	30 (0.03)	32 (0.00)	40 (0.05)	39 (0.00)
4	29 (0.01)	10 (0.04)	12 (0.00)	28 (0.00)	19 (0.02)
5	61 (0.01)	42 (0.01)	45 (0.00)	80 (0.01)	74 (0.02)
6	24 (0.00)	13 (0.02)	11 (0.01)	31 (0.01)	55 (0.03)
7	49 (0.05)	38 (0.02)	35 (0.03)	48 (0.08)	54 (0.02)
8	66 (0.01)	47 (0.00)	51 (0.01)	65 (0.01)	68 (0.01)

Table 5.51: Test 5, Mean Training CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	97.00	485.0	485.0	554.9	575.0	0
2	96.78	483.9	483.9	570.5	574.1	0
3	97.16	485.8	485.8	572.0	575.1	0
4	96.68	483.4	483.4	569.6	573.4	0
5	97.71	488.6	488.6	572.7	576.2	0
6	96.91	484.5	484.5	574.8	577.5	0
7	97.22	486.1	486.1	575.1	578.8	0
8	99.26	496.3	496.3	586.0	591.0	0

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	30.44	152.2	152.2	235.5	240.5
2	30.33	151.7	151.7	237.6	240.1
3	31.57	157.9	157.9	244.7	248.0
4	34.17	170.8	170.8	264.0	269.5
5	30.53	152.6	152.6	238.3	240.9
6	30.69	153.5	153.5	239.1	242.7
7	30.48	152.4	152.4	237.8	240.8
8	30.08	150.4	150.4	234.2	238.3

Table 5.52: Test 5, Mean Output CPU Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	3.040	11.87	16.26	12.30	15.97	42.07
2	2.992	11.67	16.02	12.08	15.83	11.02
3	3.021	11.87	16.28	12.17	15.87	111.5
4	2.978	11.66	15.94	11.96	15.69	127.9
5	3.003	11.64	16.05	11.81	15.82	51.09
6	3.014	11.62	16.10	12.06	15.85	42.59
7	3.060	11.79	16.41	12.22	16.11	137.7
8	3.161	12.35	16.90	12.69	16.58	54.78
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	1.539	3.831	8.817	4.112	8.452	
2	1.527	3.994	8.795	4.356	8.496	
3	1.466	3.788	8.555	4.103	8.159	
4	1.543	4.034	8.914	4.493	8.524	
5	1.517	3.856	8.739	4.187	8.390	
6	1.466	3.816	8.552	4.044	8.137	
7	1.561	4.053	9.226	4.512	8.555	
8	1.490	3.822	8.586	4.246	8.290	

5.5.2 Discussion

The results in Table 5.37 show small losses in hidden testing accuracy compared to Test 4. The best mean hidden testing accuracy was 61.66%, achieved by the averaged answer committee of small MLPs with Kalman filtering. This is an absolute loss of 1.56% hidden testing accuracy. The best mean hidden testing accuracy within the large MLP variants was 61.03%, achieved by the averaged probability committee with Kalman filtering and residual normalization. The best mean hidden testing accuracy using nearest-neighbour search was 53.19%, achieved with Kalman filtering and residual normalization.

While the best mean hidden testing accuracy is less than in Test 4, other hidden testing accuracies benefited from the equal proportions of each class in the training data.

5.6 Conclusion

Nearest-neighbour search was superior for visible testing data, but inferior for hidden testing data. This is also true for the large MLP variants, however the difference was much smaller. The small MLP variants were superior for hidden testing data, but inferior for visible testing data. These patterns show that the larger number of weight and bias elements in the large MLP variants resulted in overfitting and a reduction in their capability to generalize compared to the small MLP variants. These patterns also confirm that nearest-neighbour search is a classification method which overfits heavily and generalizes poorly.

If a finite number of different turns is assumed, the large difference in classification accuracy between visible testing data and hidden testing data implies that collection of additional data would improve classification accuracy. The rationale for this is that by collecting additional data, coverage of the set of different turns increases. This increases the probability that part of a given turn to be classified has already been observed before, resulting in it being classified with an accuracy similar to visible testing data rather than hidden testing data.

Kalman filtering was a very effective pre-processing algorithm, with every top accuracy result making use of it. Residual normalization was also successful, being used along with Kalman filtering for many top accuracy results. Categorization using quantiles was the least successful of the optional pre-processing algorithms. Few top accuracy results made use of it, and it had detrimental effects on the nearest-neighbour search algorithm.

The classification method and pre-processing combination selected is the averaged answer committee of small MLPs with Kalman filtering and residual normalization as trained in Test 3. The rationale behind this selection is that it provides

the best mean hidden testing accuracy while having a reasonable mean visible testing accuracy. However, due to the large standard deviations present in the results, the confidence in this selection is low.

6 Conclusion and Future Work

The results imply that collection of additional data would increase the classification accuracy, as it should move hidden accuracy results towards visible accuracy results. The results also imply that collecting additional environmental measurements to add to the input data would increase the classification accuracy. It was hoped that the wind speed and direction sensors which were abandoned when switching hardware platforms would not be needed. The results imply that the measurements from these sensors would be beneficial to the classification accuracy.

The immediate future work is the development of an algorithm to determine the start and end points of each turn by utilizing the classification results. Additional future work consists of state estimation, computation of performance metrics, and dissemination of information to the vessel crew.

The conclusion of the thesis is that the averaged answer committee of small MLPs as trained in Test 3 is chosen as the classification method. Kalman filtering and residual normalization are chosen as the optional pre-processing algorithms to be applied. This selection of pre-processing algorithms and classification method provides the best accuracy on turns that are completely omitted from the training data, while maintaining competitive accuracy on regular testing data.

7 References

[1] V. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.

[2] K. Iagnemma and C. Ward, "Classification-based wheel slip detection and detector fusion for mobile robots on outdoor terrain," *Autonomous Robots*, vol. 26, no. 1, pp. 33–46, Jan. 2009.

[3] R. Begg and J. Kamruzzaman, "A machine learning approach for automated recognition of movement patterns using basic, kinetic and kinematic gait data," *Journal of Biomechanics*, vol. 38, no. 3, pp. 401–408, Mar. 2005.

[4] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[5] A. Subasi and E. Erelebi, "Classification of EEG signals using neural network and logistic regression," *Computer Methods and Programs in Biomedicine*, vol. 78, no. 2, pp. 87–99, May 2005.

[6] E. Haselsteiner and G. Pfurtscheller, "Using time-dependent neural networks for EEG classification," *IEEE Transactions on Rehabilitation Engineering*, vol. 8, no. 4, pp. 457–463, Dec. 2000.

[7] J. Barton and A. Lees, "An application of neural networks for distinguishing gait patterns on the basis of hip-knee joint angle diagrams," *Gait & Posture*, vol. 5, no. 1, pp. 28–33, Feb. 1997.

[8] S. Huang and Y. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *Neural Networks, IEEE Transactions on*, vol. 2, no. 1, pp. 47–55, 1991.

[9] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: four layers versus three," *Neural Networks, IEEE Transactions on*, vol. 8, no. 2, pp. 251–255, 1997.

[10] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.

[11] G. H. Elkaim, "System identification-based control of an unmanned autonomous wind-propelled catamaran," *Control Engineering Practice*, vol. 17, no. 1, pp. 158–169, Jan. 2009.

[12] M. Caccia, M. Bibuli, R. Bono, and G. Bruzzone, "Basic navigation, guidance and control of an unmanned surface vehicle," *Autonomous Robots*, vol. 25, no. 4, pp. 349–365, Aug. 2008.

[13] S. Sukkarieh, E. Nebot, and H. Durrant-Whyte, "A high integrity IMU/GPS navigation loop for autonomous land vehicle applications," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 572–578, Jun. 1999.

[14] J. Wendel, O. Meister, C. Schlaile, and G. F. Trommer, "An integrated GPS/MEMS-IMU navigation system for an autonomous helicopter," *Aerospace Science and Technology*, vol. 10, no. 6, pp. 527–533, Sep. 2006.

[15] B. M. Bolstad, "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias," *Bioinformatics*, vol. 19, no. 2, pp. 185–193, 2003.

[16] R. Auckenthaler, M. Carey, and H. Lloyd-Thomas, "Score normalization for text-independent speaker verification systems," *Digital Signal Processing*, vol. 10, no. 13, pp. 42 – 54, 2000.

[17] R. Lippmann, "Pattern classification using neural networks," *IEEE Communications Magazine*, vol. 27, no. 11, pp. 47–50, Nov. 1989.

[18] T. Ozyagcilar, *Calibrating an eCompass in the Presence of Hard and Soft-Iron Interference*, Freescale Semiconductor, 6501 William Cannon Drive West, Austin, Texas, Apr. 2012. [Online]. Available: http://cache.freescale.com/files/sensors/doc/app_note/AN4246.pdf

[19] ——, *Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors*, Freescale Semiconductor, 6501 William Cannon Drive West, Austin, Texas, Jan. 2012. [Online]. Available: http://cache.freescale.com/files/sensors/doc/app_note/AN4248.pdf

[20] A. Bergeron, "Design and development of a marine data acquisition system for inertial measurement in wind powered yachts," Master's thesis, University of Ottawa, 2012. [Online]. Available: www.ruor.uottawa.ca/en/bitstream/handle/10393/23116/Bergeron_Alexandre_thesis_2012.pdf

[21] *HMC5883L Datasheet*, Honeywell, 12001 Highway 55, Plymouth, Minnesota, Feb. 2013. [Online]. Available: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf

[22] "BlackBerry PlayBook teardown - iFixit." [Online]. Available: http://www.ifixit.com/Teardown/BlackBerry+PlayBook+Teardown/5265/1

[23] "WiLink7 information - WLAN applications forum - low power RF & wireless connectivity - TI E2E community." [Online]. Available: http://e2e.ti.com/support/low_power_rf/f/307/t/101655.aspx

[24] *TC6000GN-P1 Datasheet*, Global Navigation Systems, Adenauerstrasse 18, 52146 Würselen, Germany, July 2012. [Online]. Available: http://www.forum.gns-gmbh.com/TC6000GN.pdf

[25] *BMA150 Datasheet*, Bosch Sensortec, Stuttgart, Germany, June 2010. [Online]. Available: http://ae-bst.resource.bosch.com/media/products/dokumente/bma150/bst-bma150-ds000-07.pdf

[26] *MPU-3000/MPU-3050 Datasheet*, InvenSense, 1197 Borregas Ave., Sunnyvale, California, Nov. 2011. [Online]. Available: http://www.invensense.com/mems/gyro/documents/PS-MPU-3000A.pdf

[27] "Classes & equipment | ISAF | world sailing | official website : J/24." [Online]. Available: http://www.sailing.org/270.php

[28] *J/24 Class Rules*, The International Sailing Federation, Apr. 2013. [Online]. Available: http://www.sailing.org/tools/documents/J242013CR250213-%5B14530%5D.pdf

[29] G. Evensen, *Data Assimilation: The Ensemble Kalman Filter*. Springer Berlin Heidelberg, Jan. 2009.

[30] S. Theodoridis, A. Pikrakis, K. Koutroumbas, and D. Cavouras, *Introduction to Pattern Recognition: A Matlab Approach*. Academic Press, 2010.

[31] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993.

[32] T. Vogl, J. Mangis, A. Rigler, W. Zink, and D. Alkon, "Accelerating the convergence of the backpropagation method," *Biological Cybernetics*, vol. 59, pp. 257–263, 1988.

[33] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, Sep. 1977.

[34] S. Psarakis and J. Panaretoes, "The folded t distribution," *Communications in Statistics - Theory and Methods*, vol. 19, no. 7, pp. 2717–2734, 1990.

[35] M. Mandelkern, "Setting confidence intervals for bounded parameters," *Statistical Science*, vol. 17, no. 2, pp. 149–159, 2002.

[36] R. Barandela, J. Sánchez, V. García, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2002.

Appendices

A Detailed Results

A.1 Test 1

Table A.1: Test 1, Mean Training Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	309.1	1545	1545	1711	1725	0
2	315.2	1576	1576	1744	1758	0
3	293.4	1467	1467	1627	1640	0
4	274.0	1370	1370	1521	1535	0
5	274.9	1374	1374	1525	1539	0
6	280.1	1400	1400	1554	1567	0
7	291.6	1458	1458	1618	1631	0
8	305.4	1527	1527	1692	1707	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	88.19	441.0	441.0	584.3	597.1	
2	87.76	438.8	438.8	582.8	594.9	
3	88.23	441.2	441.2	585.3	597.2	
4	88.11	440.5	440.5	585.0	597.1	
5	90.91	454.5	454.5	600.7	614.1	
6	92.63	463.1	463.1	611.6	626.3	
7	92.31	461.6	461.6	609.5	621.2	
8	102.1	510.6	510.6	674.4	687.0	

Table A.2: Test 1, Mean Output Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	5.498	14.44	32.22	12.79	27.63	529.7
2	5.584	14.89	32.72	13.23	28.08	77.34
3	5.390	14.33	31.85	12.71	27.10	1562
4	5.105	13.51	29.93	11.97	25.67	2380
5	5.100	13.54	30.13	11.99	25.63	320.8
6	5.185	13.66	30.59	12.13	26.07	281.7
7	5.373	14.18	31.82	12.57	27.01	1857
8	5.490	14.51	32.49	12.89	27.60	393.3
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	3.447	5.912	21.75	5.244	18.28	
2	3.435	5.960	21.66	5.233	18.17	
3	3.458	6.010	21.76	5.276	18.12	
4	3.466	6.032	21.88	5.352	18.46	
5	3.522	6.097	22.42	5.351	18.54	
6	3.525	6.109	22.25	5.352	18.73	
7	3.539	6.054	22.32	5.300	18.63	
8	3.720	6.489	23.61	5.725	19.67	

A.2 Test 2

Table A.3: Test 2, Mean Training Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	271.9	1360	1360	1505	1525	0
2	285.0	1425	1425	1585	1597	0
3	271.2	1356	1356	1501	1515	0
4	235.6	1178	1178	1307	1321	0
5	236.3	1182	1182	1311	1325	0
6	285.9	1430	1430	1586	1601	0
7	254.9	1275	1275	1410	1425	0
8	247.8	1239	1239	1374	1388	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	89.34	446.7	446.7	590.5	604.8	
2	82.78	413.9	413.9	544.1	555.7	
3	77.31	386.5	386.5	511.4	524.0	
4	74.50	372.5	372.5	495.2	506.7	
5	74.53	372.6	372.6	496.0	508.2	
6	74.60	373.0	373.0	496.4	508.6	
7	74.80	374.0	374.0	497.1	509.0	
8	74.97	374.8	374.8	498.3	510.4	

Table A.4: Test 2, Mean Output Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	5.590	14.88	32.81	13.92	29.03	565.2
2	5.974	16.03	34.92	15.10	31.05	106.4
3	5.572	14.93	33.22	14.11	28.93	1801
4	5.097	13.56	29.83	12.72	26.48	2579
5	5.125	13.72	30.39	12.86	26.55	373.3
6	6.001	16.00	35.33	15.01	31.07	338.6
7	5.377	14.37	31.80	13.49	27.84	1685
8	5.300	14.23	31.52	13.34	27.53	369.4
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	3.813	6.591	23.83	5.805	20.24	
2	3.665	6.334	23.11	5.547	19.54	
3	3.502	6.062	22.09	5.290	18.36	
4	3.434	5.963	21.39	5.211	18.14	
5	3.451	6.013	22.12	5.305	18.31	
6	3.433	5.926	21.66	5.179	18.16	
7	3.450	5.992	21.91	5.244	18.33	
8	3.442	5.987	21.92	5.241	18.19	

A.3 Test 3

Table A.5: Test 3, Mean Training Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	166.7	833.4	833.4	933.8	947.4	0
2	167.0	834.9	834.9	935.6	949.1	0
3	169.7	848.4	848.4	950.2	963.4	0
4	169.9	849.6	849.6	950.1	963.6	0
5	169.7	848.3	848.3	949.6	961.3	0
6	168.1	840.7	840.7	941.3	955.1	0
7	173.8	869.1	869.1	972.0	985.9	0
8	195.6	977.8	977.8	1096	1111	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	50.37	251.9	251.9	339.4	352.3	
2	50.30	251.5	251.5	338.8	351.7	
3	50.22	251.1	251.1	339.3	351.4	
4	49.93	249.7	249.7	337.6	350.1	
5	50.59	252.9	252.9	340.3	353.9	
6	50.11	250.5	250.5	338.5	351.0	
7	50.40	252.0	252.0	340.2	352.7	
8	50.33	251.6	251.6	339.3	352.0	

Table A.6: Test 3, Mean Output Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	5.272	14.16	30.94	13.30	27.25	399.7
2	5.257	14.12	30.86	13.26	27.14	97.18
3	5.339	14.38	31.61	13.54	27.64	1285
4	5.323	14.25	31.36	13.37	27.63	1983
5	5.311	14.28	31.64	13.45	27.64	373.6
6	5.273	14.13	30.95	13.28	27.24	372.6
7	5.408	14.54	32.01	13.64	27.96	1202
8	6.232	16.76	36.87	15.64	32.34	350.3
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	3.444	5.964	21.68	5.239	18.15	
2	3.460	6.001	21.77	5.317	18.20	
3	3.441	5.968	21.78	5.238	18.04	
4	3.437	5.968	21.63	5.220	18.10	
5	3.459	5.974	22.11	5.237	18.20	
6	3.441	5.948	21.55	5.222	18.05	
7	3.461	6.009	22.00	5.269	18.22	
8	3.449	6.007	22.19	5.246	18.33	

A.4 Test 4

Table A.7: Test 4, Mean Training Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	58.51	292.5	292.5	328.3	331.3	0
2	59.49	297.5	297.5	335.3	338.0	0
3	78.31	391.6	391.6	445.7	448.1	0
4	77.98	389.9	389.9	440.2	446.4	0
5	76.93	384.7	384.7	435.1	439.7	0
6	67.66	338.3	338.3	380.3	382.8	0
7	61.14	305.7	305.7	342.2	345.3	0
8	61.54	307.7	307.7	344.8	347.9	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	17.46	87.32	87.32	119.6	122.8	
2	17.27	86.34	86.34	118.4	121.5	
3	17.32	86.60	86.60	118.2	121.7	
4	17.32	86.62	86.62	118.2	121.3	
5	17.40	86.98	86.98	118.5	121.7	
6	17.27	86.33	86.33	117.8	120.8	
7	17.38	86.92	86.92	118.0	121.3	
8	17.36	86.80	86.80	118.1	121.2	

Table A.8: Test 4, Mean Output Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	1.314	3.655	7.658	3.514	6.824	59.54
2	1.352	3.787	7.873	3.615	7.046	11.00
3	1.967	5.176	11.57	4.856	10.13	154.2
4	1.618	4.465	9.423	4.294	8.481	183.3
5	1.751	4.644	10.12	4.405	9.122	57.47
6	1.475	4.064	8.493	3.897	7.651	41.96
7	1.381	3.854	8.088	3.675	7.181	189.2
8	1.373	3.837	7.963	3.660	7.172	58.02

Pre-Processing Combination	Classification Method				
	7	8	9	10	11
1	0.9113	1.647	5.742	1.481	4.801
2	0.9041	1.647	5.720	1.536	4.785
3	0.9100	1.650	5.720	1.486	4.776
4	0.9069	1.636	5.702	1.481	4.779
5	0.9032	1.632	5.678	1.473	4.798
6	0.9025	1.636	5.638	1.478	4.763
7	0.9001	1.634	5.771	1.472	4.762
8	0.9045	1.649	5.636	1.492	4.784

A.5 Test 5

Table A.9: Test 5, Mean Training Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	38.17	190.8	190.8	209.3	216.6	0
2	38.06	190.3	190.3	213.1	216.2	0
3	38.17	190.8	190.8	213.6	216.6	0
4	37.99	190.0	190.0	212.6	215.9	0
5	38.40	192.0	192.0	214.1	217.3	0
6	38.08	190.4	190.4	214.2	217.0	0
7	38.21	191.1	191.1	214.4	217.6	0
8	39.13	195.7	195.7	219.2	222.8	0
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	12.22	61.12	61.12	83.17	86.62	
2	12.26	61.28	61.28	84.13	87.01	
3	12.70	63.52	63.52	86.64	89.73	
4	13.83	69.17	69.17	94.43	98.25	
5	12.26	61.32	61.32	83.95	86.76	
6	12.27	61.36	61.36	83.89	87.07	
7	12.19	60.94	60.94	83.42	86.46	
8	12.14	60.69	60.69	82.89	86.15	

Table A.10: Test 5, Mean Output Time [s]

Pre-Processing Combination	Classification Method					
	1	2	3	4	5	6
1	1.282	3.593	7.464	3.433	6.653	42.27
2	1.279	3.573	7.455	3.411	6.635	11.08
3	1.291	3.618	7.612	3.458	6.709	112.4
4	1.285	3.593	7.475	3.429	6.674	128.0
5	1.281	3.589	7.446	3.431	6.670	51.39
6	1.281	3.589	7.429	3.439	6.671	42.73
7	1.288	3.617	7.553	3.460	6.707	138.4
8	1.321	3.722	7.708	3.556	6.873	54.96
Pre-Processing Combination	Classification Method					
	7	8	9	10	11	
1	0.9098	1.643	5.676	1.477	4.772	
2	0.9246	1.680	5.783	1.511	4.880	
3	0.9243	1.684	5.845	1.515	4.915	
4	0.9916	1.794	6.152	1.637	5.217	
5	0.9161	1.670	5.733	1.500	4.825	
6	0.9045	1.643	5.677	1.480	4.789	
7	0.9123	1.674	5.979	1.520	4.813	
8	0.9050	1.630	5.660	1.470	4.796	

B MATLAB Functions and Scripts

B.1 Main.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Important Constants
% Changing these constants will significantly impact the results.

% Number of quantile categories.
% The actual number of quantiles used is one less than this number.
Num_Quantile_Categories = 10;

% Number of cross-validation folds.
CV_Folds = 5;

%% Load Data from CSV Files

Load;
save('..\Data\Active\Sensors');

%% Prepare All Pre-Processing Combinations

% For reference:
% Combination #1: None.
% Combination #2: Kalman filtering.
% Combination #3: Categorization using quantiles.
% Combination #4: Normalization.
% Combination #5: Kalman filtering and categorization using quantiles.
% Combination #6: Kalman filtering and normalization.
% Combination #7: Categorization using quantiles and normalization.
% Combination #8: All.

for Combination_Num = 1:8

    % Clear.
    clearvars -except Combination_Num CV_Folds

    % Load raw data.
    load('..\Data\Active\Sensors');

    % Name for the resultant file.
    filename = '..\Data\Active\Input';

    % Convert from raw data.
    Sensor_Correction;
    Derivative_Calculation;

    % Do Kalman filtering for Combination #2, #5, #6, and #8.
    if Combination_Num == 2 || Combination_Num == 5 ...
            || Combination_Num == 6 || Combination_Num == 8

        % Add to file name.
        filename = [filename, '_Kalman'];

        % Do Kalman filtering.
        Kalman_Filter;

    end

    % Refine the input by taking certain absolute values and trimming.
    Selective_Absolute_Values;

    % Categorize using quantiles for Combination #3, #5, #7, and #8.
    if Combination_Num == 3 || Combination_Num == 5 ...
            || Combination_Num == 7 || Combination_Num == 8

        % Add to file name.
        filename = [filename, '_Quantile'];

        % Do categorization.
        Categorization_Using_Quantiles;
```

```
    end

    % Rescale each input to the interval between zero and unity.
    Scale_Normalization;

    % Normalize for Combination #4, #6, #7, and #8.
    if Combination_Num == 4 || Combination_Num == 6 ...
            || Combination_Num == 7 || Combination_Num == 8

        % Add to file name.
        filename = [filename, '_Normal'];

        % Do normalization.
        Residual_Normalization;

    end

    % Save.
    save(filename);

end

%% Do Tests

for Test_Num = 1:5
    Main_Test;
end
```

B.2 Load.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Initialize

% Start timer.
tic;
cpu_tic = cputime;

% Initialize the structure and counter.
Run = struct([]);
ii = 1;

%% Load

% Load the data.
try
    Run(ii).Sensors = csvread('..\Data\2012_07_26\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_07_26\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_07_26\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_07_26\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_07_26\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_07_26\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
    Run(ii).Sensors = csvread('..\Data\2012_08_07\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_08_07\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_08_07\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_08_07\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_08_07\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
```

```
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_08_07\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
    Run(ii).Sensors = csvread('..\Data\2012_08_09\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_08_09\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_08_09\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_08_09\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_08_09\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_08_09\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
    Run(ii).Sensors = csvread('..\Data\2012_08_14\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_08_14\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_08_14\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_08_14\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_08_14\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_08_14\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
```

```
    Run(ii).Sensors = csvread('..\Data\2012_09_11\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_09_11\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_09_11\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_09_11\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_09_11\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_09_11\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
    Run(ii).Sensors = csvread('..\Data\2012_09_25\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_09_25\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
try
    Run(ii).Tack_Points = csvread('..\Data\2012_09_25\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_09_25\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_09_25\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_09_25\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end
% Next.
ii = ii + 1;
try
    Run(ii).Sensors = csvread('..\Data\2012_09_29\BP.csv');
catch
    Run(ii).Sensors = [];
end
try
    Run(ii).Trim_Points = csvread('..\Data\2012_09_29\BP_Trim.csv');
catch
    Run(ii).Trim_Points = [];
end
```

```
try
    Run(ii).Tack_Points = csvread('..\Data\2012_09_29\BP_Tacks.csv');
catch
    Run(ii).Tack_Points = [];
end
try
    Run(ii).Gybe_Points = csvread('..\Data\2012_09_29\BP_Gybes.csv');
catch
    Run(ii).Gybe_Points = [];
end
try
    Run(ii).Rounding_Points = csvread('..\Data\2012_09_29\BP_Roundings.csv');
catch
    Run(ii).Rounding_Points = [];
end
try
    Run(ii).Straight_Points = csvread('..\Data\2012_09_29\BP_Straight.csv');
catch
    Run(ii).Straight_Points = [];
end

clearvars ii

%% Column Identifiers

% Record what each column represents.
col = 1;
sensors_gpstime_col = col;
col = col + 1;
sensors_gpslatitude_col = col;
col = col + 1;
sensors_gpslongitude_col = col;
col = col + 1;
sensors_gpsaccuracy_col = col;
col = col + 1;
sensors_gpsheading_col = col;
col = col + 1;
sensors_gpsspeed_col = col;
col = col + 1;
sensors_accelerometerx_col = col;
col = col + 1;
sensors_accelerometery_col = col;
col = col + 1;
sensors_accelerometerz_col = col;
col = col + 1;
sensors_linearaccelerometerx_col = col;
col = col + 1;
sensors_linearaccelerometery_col = col;
col = col + 1;
sensors_linearaccelerometerz_col = col;
col = col + 1;
sensors_magnetometerx_col = col;
col = col + 1;
sensors_magnetometery_col = col;
col = col + 1;
sensors_magnetometerz_col = col;
col = col + 1;
sensors_gyroscopex_col = col;
col = col + 1;
sensors_gyroscopey_col = col;
col = col + 1;
sensors_gyroscopez_col = col;
col = col + 1;
sensors_azimuth_col = col;
col = col + 1;
sensors_pitch_col = col;
col = col + 1;
sensors_roll_col = col;
col = col + 1;
sensors_rotationvector0_col = col;
col = col + 1;
sensors_rotationvector1_col = col;
col = col + 1;
```

```
sensors_rotationvector2_col = col;
col = col + 1;
sensors_rotationvector3_col = col;
col = col + 1;
sensors_rotationmatrix0_col = col;
col = col + 1;
sensors_rotationmatrix1_col = col;
col = col + 1;
sensors_rotationmatrix2_col = col;
col = col + 1;
sensors_rotationmatrix3_col = col;
col = col + 1;
sensors_rotationmatrix4_col = col;
col = col + 1;
sensors_rotationmatrix5_col = col;
col = col + 1;
sensors_rotationmatrix6_col = col;
col = col + 1;
sensors_rotationmatrix7_col = col;
col = col + 1;
sensors_rotationmatrix8_col = col;
col = col + 1;
sensors_timeinterval_col = col;

clearvars col

%% Remove Unnecessary Columns

% Columns to remove.
remove = [sensors_gpstime_col:sensors_gpsaccuracy_col,...
    sensors_linearaccelerometerx_col:sensors_magnetometerz_col,...
    sensors_azimuth_col:sensors_rotationmatrix8_col];

% Remove the columns.
for ii = 1:size(Run, 2)
    Run(ii).Sensors(:,remove) = [];
end

% Update column identifiers.
sensors_gpsheading_col = 1;
sensors_gpsspeed_col = 2;
sensors_accelerometerx_col = 3;
sensors_accelerometery_col = 4;
sensors_accelerometerz_col = 5;
sensors_gyroscopex_col = 6;
sensors_gyroscopey_col = 7;
sensors_gyroscopez_col = 8;
sensors_timeinterval_col = 9;

clearvars  sensors_gpstime_col sensors_gpslatitude_col sensors_gpslongitude_col...
    sensors_gpsaccuracy_col sensors_linearaccelerometer* sensors_azimuth_col...
    sensors_pitch_col sensors_roll_col sensors_rotation* sensors_magnetometer*...
    ii remove

%% Invert Required Columns

for ii = 1:size(Run, 2)
    Run(ii).Sensors(:,sensors_accelerometerx_col) = ...
        -Run(ii).Sensors(:,sensors_accelerometerx_col);
    Run(ii).Sensors(:,sensors_accelerometery_col) = ...
        -Run(ii).Sensors(:,sensors_accelerometery_col);
    Run(ii).Sensors(:,sensors_gyroscopex_col) = ...
        -Run(ii).Sensors(:,sensors_gyroscopex_col);
    Run(ii).Sensors(:,sensors_gyroscopey_col) = ...
        -Run(ii).Sensors(:,sensors_gyroscopey_col);
end

%% Remove Data Points Where Any GPS Data is Invalid

for ii = 1:size(Run, 2)
    Run(ii).Sensors(any(Run(ii).Sensors...
        (:,sensors_gpsheading_col:sensors_gpsspeed_col)== -1,2),:)=[];
end
```

```
% Clean up cell.
clearvars ii

%% Trim the Data

for ii = 1:size(Run, 2)
    Run(ii).Sensors = Run(ii).Sensors(Run(ii).Trim_Points(1):Run(ii).Trim_Points(2),:);
    Run(ii).Tack_Points = Run(ii).Tack_Points - Run(ii).Trim_Points(1);
    Run(ii).Gybe_Points = Run(ii).Gybe_Points - Run(ii).Trim_Points(1);
    Run(ii).Rounding_Points = Run(ii).Rounding_Points - Run(ii).Trim_Points(1);
    Run(ii).Straight_Points = Run(ii).Straight_Points - Run(ii).Trim_Points(1);
end

%% Determine Straight Points

for ii = 1:size(Run, 2)
    list = [Run(ii).Tack_Points(:);Run(ii).Gybe_Points(:);Run(ii).Rounding_Points(:)];
    list = sort(list);
    Run(ii).Straight_Points = [1,list(1)-1];
    for jj = 2:2:length(list)-1
        Run(ii).Straight_Points = ...
            vertcat(Run(ii).Straight_Points,[list(jj)+1,list(jj+1)-1]);
    end
    Run(ii).Straight_Points = ...
        vertcat(Run(ii).Straight_Points,[list(end)+1,size(Run(ii).Sensors,1)]);
end

% Record time.
Load_Time = toc;
Load_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii cpu_tic list
```

B.3 Sensor_Correction.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Create Input Matrix

% Start timer.
tic;
cpu_tic = cputime;

% Column identifiers.
id = 1;
input_gpsheading_col = id;
id = id + 1;
input_dgpsheading_col = id;
id = id + 1;
input_ddgpsheading_col = id;
id = id + 1;
input_gpsspeed_col = id;
id = id + 1;
input_dgpsspeed_col = id;
id = id + 1;
input_accelerometerx_col = id;
id = id + 1;
input_accelerometery_col = id;
id = id + 1;
input_accelerometerz_col = id;
id = id + 1;
input_gyroscopex_col = id;
id = id + 1;
input_dgyroscopex_col = id;
id = id + 1;
input_gyroscopey_col = id;
id = id + 1;
input_dgyroscopey_col = id;
id = id + 1;
input_gyroscopez_col = id;
id = id + 1;
input_dgyroscopez_col = id;
id = id + 1;
input_time_col = id;

% Create the matrix.
for ii = 1:size(Run, 2)
    Run(ii).Input = zeros(size(Run(ii).Sensors,1),id);
end

% Clean up cell.
clearvars ii id

%% Populate Input Matrix

for ii = 1:size(Run, 2)

    Run(ii).Input(:,input_gpsheading_col) =  ...
        Run(ii).Sensors(:,sensors_gpsheading_col);
    Run(ii).Input(:,input_gpsspeed_col) =  ...
        Run(ii).Sensors(:,sensors_gpsspeed_col);
    Run(ii).Input(:,input_accelerometerx_col) =  ...
        Run(ii).Sensors(:,sensors_accelerometerx_col);
    Run(ii).Input(:,input_accelerometery_col) =  ...
        Run(ii).Sensors(:,sensors_accelerometery_col);
    Run(ii).Input(:,input_accelerometerz_col) =  ...
        Run(ii).Sensors(:,sensors_accelerometerz_col);
    Run(ii).Input(:,input_gyroscopex_col) =  ...
        Run(ii).Sensors(:,sensors_gyroscopex_col);
    Run(ii).Input(:,input_gyroscopey_col) =  ...
        Run(ii).Sensors(:,sensors_gyroscopey_col);
    Run(ii).Input(:,input_gyroscopez_col) =  ...
        Run(ii).Sensors(:,sensors_gyroscopez_col);

end
```

```
% Clean up cell.
clearvars ii

%% Calculate Time Vector

for ii = 1:size(Run, 2)

    % Sum.
    Run(ii).Input(:,input_time_col) = ...
        [0;cumsum(Run(ii).Sensors(2:end,sensors_timeinterval_col),1)];

    % Convert from microseconds to seconds.
    Run(ii).Input(:,input_time_col) = Run(ii).Input(:,input_time_col)/1e6;

end

clearvars ii

%% Calculate Continuous GPS Heading

% The GPS is quite noisy, so extreme changes must be removed.
% Merge data from all runs.
gpsheading = [];
for ii = 1:size(Run, 2)
    gpsheading = vertcat(gpsheading, Run(ii).Input(:,input_gpsheading_col));
end

% Calculate all changes.
delta = diff(gpsheading);

% Exclude zeros.
delta(delta == 0) = [];

% Handle wrap around.
delta(delta > 180) = delta(delta > 180) - 360;
delta(delta < -180) = delta(delta < -180) + 360;

% Take a 95% confidence interval, assuming normal distribution.
Valid_GPS_Heading_Difference_Interval = ...
    norminv([0.025 0.975],mean(delta),std(delta));

% Loop through all runs.
for ii = 1:size(Run, 2)

    % Calculate all changes.
    delta = diff(Run(ii).Input(:,input_gpsheading_col));

    % Handle wrap around.
    delta(delta > 180) = delta(delta > 180) - 360;
    delta(delta < -180) = delta(delta < -180) + 360;

    % Apply validity interval.
    delta(delta < min(Valid_GPS_Heading_Difference_Interval)) = 0;
    delta(delta > max(Valid_GPS_Heading_Difference_Interval)) = 0;

    % Copy into matrix.
    Run(ii).Input(2:end,input_gpsheading_col) = delta;

    % Sum.
    Run(ii).Input(:,input_gpsheading_col) = ...
        cumsum(Run(ii).Input(:,input_gpsheading_col));

end

% Clean up cell.
clearvars delta ii jj gpsheading

%% Correct Erroneous GPS Speed Measurements

% The GPS speed is also quite noisy, so a similar method must be used.
% Merge data from all runs.
gpsspeed = [];
for ii = 1:size(Run, 2)
```

```
    gpsspeed = vertcat(gpsspeed, Run(ii).Input(:,input_gpsspeed_col));
end

% Take a 95% confidence interval, assuming normal distribution.
Valid_GPS_Speed_Interval = norminv([0.025 0.975],mean(gpsspeed),std(gpsspeed));

% Loop through all runs.
for ii = 1:size(Run, 2)

    % Loop through rows.
    for jj = 2:size(Run(ii).Input,1)

        % Apply validity interval.
        if Run(ii).Input(jj,input_gpsspeed_col) < min(Valid_GPS_Speed_Interval) ...
                || Run(ii).Input(jj,input_gpsspeed_col) > max(Valid_GPS_Speed_Interval)

            Run(ii).Input(jj,input_gpsspeed_col) = Run(ii).Input(jj-1,input_gpsspeed_col);

        end

    end

end

% Record time.
Sensor_Correction_Time = toc;
Sensor_Correction_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii jj cpu_tic gpsspeed
```

B.4 Derivative_Calculation.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Calculate GPS Derivatives

% Start timer.
tic;
cpu_tic = cputime;

for ii = 1:size(Run, 2)

    % First derivatives.
    % Calculate forward difference for first point.
    Run(ii).Input(1,input_dgpsheading_col) = ...
        (Run(ii).Input(2,input_gpsheading_col)-...
        Run(ii).Input(1,input_gpsheading_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));
    Run(ii).Input(1,input_dgpsspeed_col) = ...
        (Run(ii).Input(2,input_gpsspeed_col)-...
        Run(ii).Input(1,input_gpsspeed_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_dgpsheading_col) = ...
        (Run(ii).Input(end,input_gpsheading_col)-...
        Run(ii).Input(end-1,input_gpsheading_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));
    Run(ii).Input(end,input_dgpsspeed_col) = ...
        (Run(ii).Input(end,input_gpsspeed_col)-...
        Run(ii).Input(end-1,input_gpsspeed_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_dgpsheading_col) = ...
            (Run(ii).Input(jj+1,input_gpsheading_col)-...
            Run(ii).Input(jj-1,input_gpsheading_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
        Run(ii).Input(jj,input_dgpsspeed_col) = ...
            (Run(ii).Input(jj+1,input_gpsspeed_col)-...
            Run(ii).Input(jj-1,input_gpsspeed_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end

    % Second derivatives.
    % Calculate forward difference for first point.
    Run(ii).Input(1,input_ddgpsheading_col) = ...
        (Run(ii).Input(2,input_dgpsheading_col)-...
        Run(ii).Input(1,input_dgpsheading_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_ddgpsheading_col) = ...
        (Run(ii).Input(end,input_dgpsheading_col)-...
        Run(ii).Input(end-1,input_dgpsheading_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_ddgpsheading_col) = ...
            (Run(ii).Input(jj+1,input_dgpsheading_col)-...
            Run(ii).Input(jj-1,input_dgpsheading_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
```

```
    end

end

% Clean up cell.
clearvars ii jj

%% Calculate Gyroscope Derivatives

for ii = 1:size(Run, 2)

    % Calculate forward difference for first point.
    Run(ii).Input(1,input_dgyroscopex_col) = ...
        (Run(ii).Input(2,input_gyroscopex_col)-...
        Run(ii).Input(1,input_gyroscopex_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));
    Run(ii).Input(1,input_dgyroscopey_col) = ...
        (Run(ii).Input(2,input_gyroscopey_col)-...
        Run(ii).Input(1,input_gyroscopey_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));
    Run(ii).Input(1,input_dgyroscopez_col) = ...
        (Run(ii).Input(2,input_gyroscopez_col)-...
        Run(ii).Input(1,input_gyroscopez_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_dgyroscopex_col) = ...
        (Run(ii).Input(end,input_gyroscopex_col)-...
        Run(ii).Input(end-1,input_gyroscopex_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));
    Run(ii).Input(end,input_dgyroscopey_col) = ...
        (Run(ii).Input(end,input_gyroscopey_col)-...
        Run(ii).Input(end-1,input_gyroscopey_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));
    Run(ii).Input(end,input_dgyroscopez_col) = ...
        (Run(ii).Input(end,input_gyroscopez_col)-...
        Run(ii).Input(end-1,input_gyroscopez_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_dgyroscopex_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopex_col)-...
            Run(ii).Input(jj-1,input_gyroscopex_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
        Run(ii).Input(jj,input_dgyroscopey_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopey_col)-...
            Run(ii).Input(jj-1,input_gyroscopey_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
        Run(ii).Input(jj,input_dgyroscopez_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopez_col)-...
            Run(ii).Input(jj-1,input_gyroscopez_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end

end

% Record time.
Derivative_Calculation_Time = toc;
Derivative_Calculation_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii jj cpu_tic
```

B.5 Kalman_Filter.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Filter GPS Heading

% Start timer.
tic;
cpu_tic = cputime;

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise covariances.
    R = 1;
    Q = 1e-5;

    % Heading.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_gpsheading_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_gpsheading_col) = x;
    end

end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Recalculate GPS Heading First Derivative

for ii = 1:size(Run, 2)

    % Calculate forward difference for first point.
    Run(ii).Input(1,input_dgpsheading_col) = ...
        (Run(ii).Input(2,input_gpsheading_col)-...
        Run(ii).Input(1,input_gpsheading_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_dgpsheading_col) = ...
        (Run(ii).Input(end,input_gpsheading_col)-...
        Run(ii).Input(end-1,input_gpsheading_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_dgpsheading_col) = ...
            (Run(ii).Input(jj+1,input_gpsheading_col)-...
            Run(ii).Input(jj-1,input_gpsheading_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end
```

```
end

% Clean up cell.
clearvars ii jj

%% Filter GPS Heading First Derivative

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise covariances.
    R = 1;
    Q = 1e-4;

    % Heading first derivative.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_dgpsheading_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_dgpsheading_col) = x;
    end

end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Recalculate GPS Heading Second Derivative

for ii = 1:size(Run, 2)

    % Calculate forward difference for first point.
    Run(ii).Input(1,input_ddgpsheading_col) = ...
        (Run(ii).Input(2,input_dgpsheading_col)-...
        Run(ii).Input(1,input_dgpsheading_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_ddgpsheading_col) = ...
        (Run(ii).Input(end,input_dgpsheading_col)-...
        Run(ii).Input(end-1,input_dgpsheading_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_ddgpsheading_col) = ...
            (Run(ii).Input(jj+1,input_dgpsheading_col)-...
            Run(ii).Input(jj-1,input_dgpsheading_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end

end

% Clean up cell.
```

```
clearvars ii jj

%% Filter GPS Heading Second Derivative

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise covariances.
    R = 1;
    Q = 1e-4;

    % Heading second derivative.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_ddgpsheading_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_ddgpsheading_col) = x;
    end

end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Filter GPS Speed

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise covariances.
    R = 1;
    Q = 1e-6;

    % Speed.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_gpsspeed_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_gpsspeed_col) = x;
    end
```

```
end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Recalculate GPS Speed First Derivative

for ii = 1:size(Run, 2)

    % Calculate forward difference for first point.
    Run(ii).Input(1,input_dgpsspeed_col) = ...
        (Run(ii).Input(2,input_gpsspeed_col)-...
        Run(ii).Input(1,input_gpsspeed_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_dgpsspeed_col) = ...
        (Run(ii).Input(end,input_gpsspeed_col)-...
        Run(ii).Input(end-1,input_gpsspeed_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_dgpsspeed_col) = ...
            (Run(ii).Input(jj+1,input_gpsspeed_col)-...
            Run(ii).Input(jj-1,input_gpsspeed_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end

end

% Clean up cell.
clearvars ii jj

%% Filter GPS Speed First Derivative

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise covariances.
    R = 1;
    Q = 1e-5;

    % Speed first derivative.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_dgpsspeed_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_dgpsspeed_col) = x;
    end

end

% Clean up cell.
```

```
clearvars ii jj x P z H F y S K Q R q

%% Filter Accelerometer

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise Covariances.
    R = 1;
    Q = 1e-3;

    % X.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_accelerometerx_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_accelerometerx_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Y.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_accelerometery_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_accelerometery_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Z.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_accelerometerz_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
```

```
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_accelerometerz_col) = x;
    end

end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Filter Gyroscope

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise Covariances.
    R = 1;
    Q = 1e-3;

    % X.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_gyroscopex_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_gyroscopex_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Y.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_gyroscopey_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_gyroscopey_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Z.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
```

```
        z = Run(ii).Input(jj, input_gyroscopez_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_gyroscopez_col) = x;
    end

end

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q

%% Recalculate Gyroscope Derivatives

for ii = 1:size(Run, 2)

    % Calculate forward difference for first point.
    Run(ii).Input(1,input_dgyroscopex_col) = ...
        (Run(ii).Input(2,input_gyroscopex_col)-...
        Run(ii).Input(1,input_gyroscopex_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));
    Run(ii).Input(1,input_dgyroscopey_col) = ...
        (Run(ii).Input(2,input_gyroscopey_col)-...
        Run(ii).Input(1,input_gyroscopey_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));
    Run(ii).Input(1,input_dgyroscopez_col) = ...
        (Run(ii).Input(2,input_gyroscopez_col)-...
        Run(ii).Input(1,input_gyroscopez_col))...
        /(Run(ii).Input(2,input_time_col)-...
        Run(ii).Input(1,input_time_col));

    % Calculate backwards difference for last point.
    Run(ii).Input(end,input_dgyroscopex_col) = ...
        (Run(ii).Input(end,input_gyroscopex_col)-...
        Run(ii).Input(end-1,input_gyroscopex_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));
    Run(ii).Input(end,input_dgyroscopey_col) = ...
        (Run(ii).Input(end,input_gyroscopey_col)-...
        Run(ii).Input(end-1,input_gyroscopey_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));
    Run(ii).Input(end,input_dgyroscopez_col) = ...
        (Run(ii).Input(end,input_gyroscopez_col)-...
        Run(ii).Input(end-1,input_gyroscopez_col))...
        /(Run(ii).Input(end,input_time_col)-...
        Run(ii).Input(end-1,input_time_col));

    % Calculate central difference for interior points.
    for jj = 2:size(Run(ii).Input,1)-1;
        Run(ii).Input(jj,input_dgyroscopex_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopex_col)-...
            Run(ii).Input(jj-1,input_gyroscopex_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
        Run(ii).Input(jj,input_dgyroscopey_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopey_col)-...
            Run(ii).Input(jj-1,input_gyroscopey_col))...
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
        Run(ii).Input(jj,input_dgyroscopez_col) = ...
            (Run(ii).Input(jj+1,input_gyroscopez_col)-...
            Run(ii).Input(jj-1,input_gyroscopez_col))...
```

```
            /(Run(ii).Input(jj+1,input_time_col)-...
            Run(ii).Input(jj-1,input_time_col));
    end

end

% Clean up cell.
clearvars ii jj

%% Filter Gyroscope First Derivatives

for ii = 1:size(Run, 2)

    % Initial input.
    x = 0;
    P = 1e6;

    % Input to observation matrix.
    H = 1;

    % Noise Covariances.
    R = 1;
    Q = 1e-2;

    % X.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_dgyroscopex_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_dgyroscopex_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Y.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_dgyroscopey_col);
        F = 1;
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_dgyroscopey_col) = x;
    end

    % Initial input.
    x = 0;
    P = 1e6;

    % Z.
    for jj = 1:size(Run(ii).Input,1)
        % Load observation.
        z = Run(ii).Input(jj, input_dgyroscopez_col);
        F = 1;
```

```
        % Predict.
        x = F*x;
        P = F*P*(F')+Q;
        % Update.
        y = z - H*x;
        S = H*P*(H')+R;
        K = P*(H')/S;
        x = x + K*y;
        P = (1-K*H)*P;
        % Record.
        Run(ii).Input(jj, input_dgyroscopez_col) = x;
    end

end

% Record time.
Kalman_Filter_Time = toc;
Kalman_Filter_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii jj x P z H F y S K Q R q cpu_tic
```

B.6 Selective_Absolute_Values.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Remove Dependence on Turn Direction

% Start timer.
tic;
cpu_tic = cputime;

for ii = 1:size(Run, 2)

    % Take the absolute value of quantities whose sign depends on turn
    % direction, and any derivatives of such a quantity.
    Run(ii).Input(:,input_dgpsheading_col) = ...
        abs(Run(ii).Input(:,input_dgpsheading_col));
    Run(ii).Input(:,input_ddgpsheading_col) = ...
        abs(Run(ii).Input(:,input_ddgpsheading_col));
    Run(ii).Input(:,input_accelerometery_col) = ...
        abs(Run(ii).Input(:,input_accelerometery_col));
    Run(ii).Input(:,input_gyroscopex_col) = ...
        abs(Run(ii).Input(:,input_gyroscopex_col));
    Run(ii).Input(:,input_dgyroscopex_col) = ...
        abs(Run(ii).Input(:,input_dgyroscopex_col));
    Run(ii).Input(:,input_gyroscopez_col) = ...
        abs(Run(ii).Input(:,input_gyroscopez_col));
    Run(ii).Input(:,input_dgyroscopez_col) = ...
        abs(Run(ii).Input(:,input_dgyroscopez_col));

end

% Clean up cell.
clearvars ii

%% Trim Input Matrix

for ii = 1:size(Run, 2)

    % The time and GPS heading columns are no longer needed.
    Run(ii).Input(:,input_time_col) = [];
    Run(ii).Input(:,input_gpsheading_col) = [];

end

% Redefine column identifiers.
id = 1;
input_dgpsheading_col = id;
id = id + 1;
input_ddgpsheading_col = id;
id = id + 1;
input_gpsspeed_col = id;
id = id + 1;
input_dgpsspeed_col = id;
id = id + 1;
input_accelerometerx_col = id;
id = id + 1;
input_accelerometery_col = id;
id = id + 1;
input_accelerometerz_col = id;
id = id + 1;
input_gyroscopex_col = id;
id = id + 1;
input_dgyroscopex_col = id;
id = id + 1;
input_gyroscopey_col = id;
id = id + 1;
input_dgyroscopey_col = id;
id = id + 1;
input_gyroscopez_col = id;
id = id + 1;
input_dgyroscopez_col = id;

% Clean up cell.
```

```
clearvars ii id input_time_col input_gpsheading_col

%% Remove Raw Sensor Data

% The raw sensor data is no longer needed.
Run = rmfield(Run, 'Sensors');

% Record time.
Selective_Absolute_Values_Time = toc;
Selective_Absolute_Values_CPU_Time = cputime-cpu_tic;

% Remove column identifiers.
clearvars sensors_*_col cpu_tic
```

B.7 Categorization_Using_Quantiles.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Find Quantiles

% Start timer.
tic;
cpu_tic = cputime;

% Merge data from all runs.
temp = [];
for ii = 1:size(Run,2)
    temp = vertcat(temp, Run(ii).Input);
end

% Express the quantile fractions.
q = (1:Num_Quantile_Categories-1)/Num_Quantile_Categories;

% Find the quantiles.
Quantiles = quantile(temp,q,1);

% Bound by infinity to make using the matrix easier.
Quantiles(Num_Quantile_Categories,:) = Inf;

% Clean up cell.
clearvars temp ii q

%% Convert

for nn = 1:size(Run,2)
    for ii = 1:size(Run(nn).Input,1)
        for jj = 1:size(Run(nn).Input,2)
            for kk = 1:Num_Quantile_Categories
                if Run(nn).Input(ii,jj) <= Quantiles(kk,jj)
                    Run(nn).Input(ii,jj) = kk;
                    break;
                end
            end
        end
    end
end

% Record time.
Categorization_Using_Quantiles_Time = toc;
Categorization_Using_Quantiles_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars nn ii jj kk cpu_tic
```

B.8 Scale_Normalization.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Start timer.
tic;
cpu_tic = cputime;

% Merge data from all runs.
temp = [];
for ii = 1:size(Run,2)
    temp = vertcat(temp, Run(ii).Input);
end

% Find minimum and maximum values.
min_val = min(temp,[],1);
max_val = max(temp,[],1);

% Loop through all runs.
for ii = 1:size(Run,2)

    % Loop through columns.
    for jj = 1:size(Run(ii).Input,2)

        % Rescale.
        Run(ii).Input(:,jj) = (Run(ii).Input(:,jj)-min_val(jj))...
            /(max_val(jj)-min_val(jj));

    end
end

% Record time.
Scale_Normalization_Time = toc;
Scale_Normalization_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii jj temp min_val max_val cpu_tic
```

B.9 Residual_Normalization.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Start timer.
tic;
cpu_tic = cputime;

% Merge data from all runs.
temp = [];
for ii = 1:size(Run,2)
    temp = vertcat(temp, Run(ii).Input);
end

% Find mean.
mean_val = mean(temp,1);

% Find standard deviation.
std_val = std(temp,1);

% Loop through all runs.
for ii = 1:size(Run,2)

    % Loop through columns.
    for jj = 1:size(Run(ii).Input,2)

        % Normalize.
        Run(ii).Input(:,jj) = (Run(ii).Input(:,jj)-mean_val(jj))...
            /std_val(jj);

    end
end

% Record time.
Residual_Normalization_Time = toc;
Residual_Normalization_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars ii jj temp mean_val std_val cpu_tic
```

B.10 Main_Test.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Set Up for Classification

for Combination_Num = 1:8

    % Clear.
    clearvars -except Test_Num Combination_Num CV_Folds

    % Construct combination segment of filename.
    combination = Combination_Num_To_String(Combination_Num);

    % Name of file to load.
    filename = ['..\Data\Active\Input', combination];

    % Load.
    load(filename, 'Run');

    % Do setup.
    Setup_For_Classification;

    % Name for the resultant file.
    filename = ['..\Data\Active\Test_', ...
        num2str(Test_Num), '\Input', combination];

    % Save.
    clearvars combination
    save(filename);

end

%% Do Large MLP

for Combination_Num = 1:8

    for Fold_Num = 1:CV_Folds

        % Clear.
        clearvars -except Test_Num Combination_Num Fold_Num CV_Folds

        % Construct combination segment of filename.
        combination = Combination_Num_To_String(Combination_Num);

        % Name of file to load.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Input', combination];

        % Load.
        load(filename);

        % Do MLP.
        Large_MLP;

        % Name for the resultant file.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Part_1\Fold_',...
            num2str(Fold_Num), '\MLP', combination];

        % Save.
        clearvars combination
        save(filename);

    end

end

%% Do NN

for Combination_Num = 1:8

    for Fold_Num = 1:CV_Folds
```

```
        % Clear.
        clearvars -except Test_Num Combination_Num Fold_Num CV_Folds

        % Construct combination segment of filename.
        combination = Combination_Num_To_String(Combination_Num);

        % Name of file to load.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Input', combination];

        % Load.
        load(filename);

        % Do NN.
        NN;

        % Name for the resultant file.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Part_1\Fold_',...
            num2str(Fold_Num), '\NN', combination];

        % Save.
        clearvars combination
        save(filename);

    end

end

%% Do Small MLP

for Combination_Num = 1:8

    for Fold_Num = 1:CV_Folds

        % Clear.
        clearvars -except Test_Num Combination_Num Fold_Num CV_Folds

        % Construct combination segment of filename.
        combination = Combination_Num_To_String(Combination_Num);

        % Name of file to load.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Input', combination];

        % Load.
        load(filename);

        % Do Small MLP.
        Small_MLP;

        % Name for the resultant file.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Part_1\Fold_',...
            num2str(Fold_Num), '\MLP_Small', combination];

        % Save.
        clearvars combination
        save(filename);

    end

end

%% Calculate Classification Results

for Test_Num = 1:3
    for Combination_Num = 1:8

        for Fold_Num = 1:CV_Folds

            % Clear.
```

```
            clearvars -except Test_Num Combination_Num Fold_Num CV_Folds

            % Construct combination segment of filename.
            combination = Combination_Num_To_String(Combination_Num);

            % Load MLP.
            filename = ['..\Data\Active\Test_', ...
                num2str(Test_Num), '\Part_1\Fold_',...
                num2str(Fold_Num), '\MLP', combination];
            load(filename);

            % Load NN.
            filename = ['..\Data\Active\Test_', ...
                num2str(Test_Num), '\Part_1\Fold_',...
                num2str(Fold_Num), '\NN', combination];
            load(filename);

            % Load Small MLP.
            filename = ['..\Data\Active\Test_', ...
                num2str(Test_Num), '\Part_1\Fold_',...
                num2str(Fold_Num), '\MLP_Small', combination];
            load(filename);

            % Calculate results.
            Classification_Result;

            % Name for the resultant file.
            filename = ['..\Data\Active\Test_', ...
                num2str(Test_Num), '\Part_1\Fold_',...
                num2str(Fold_Num), '\Classification', combination];

            % Save.
            clearvars combination
            save(filename);

        end

    end
end

%% Aggregate Classification Results

Classification_Aggregate;

%% Rank Classification Results

Classification_Rank;

%% Make Latex Tables

Classification_Latex;
```

B.11 Combination_Num_To_String.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
function combination = Combination_Num_To_String(Combination_Num)
% Returns the pre-processing combination segment of a filename for a given
% combination number.

combination = '';
if Combination_Num == 2 || Combination_Num == 5 ...
        || Combination_Num == 6 || Combination_Num == 8
    combination = [combination, '_Kalman'];
end
if Combination_Num == 3 || Combination_Num == 5 ...
        || Combination_Num == 7 || Combination_Num == 8
    combination = [combination, '_Quantile'];
end
if Combination_Num == 4 || Combination_Num == 6 ...
        || Combination_Num == 7 || Combination_Num == 8
    combination = [combination, '_Normal'];
end

end
```

B.12 Setup_For_Classification.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Trim Run

% Start timer.
tic;
cpu_tic = cputime;

% Test #4 and Test #5 only use the third run.
if Test_Num == 4 || Test_Num == 5
    Run = Run(3);
end

%% Common Setup

% End points of each event by class and run.
p = cell(1,4);
for ii = 1:4
    p{ii} = cell(1,size(Run, 2));
end

% Number of events of each class by run.
n = cell(1,4);
for ii = 1:4
    n{ii} = zeros(1,size(Run, 2));
end

% Number of points in each event by class and run.
np = cell(1,4);
for ii = 1:4
    np{ii} = cell(1,size(Run, 2));
end

% Construct input.
X = [];
for ii = 1:size(Run,2)
    X = vertcat(X,Run(ii).Input);
end

% Construct classification.
T = zeros(size(X,1),4);
count = 0;
for ii = 1:size(Run,2)
    p{1}{ii} = Run(ii).Tack_Points;
    p{2}{ii} = Run(ii).Gybe_Points;
    p{3}{ii} = Run(ii).Rounding_Points;
    p{4}{ii} = Run(ii).Straight_Points;
    for jj = 1:4
        n{jj}(ii) = size(p{jj}{ii},1);
        for kk = 1:n{jj}(ii)
            np{jj}{ii}(kk) = p{jj}{ii}(kk,2)-p{jj}{ii}(kk,1)+1;
            T(count+p{jj}{ii}(kk,1):count+p{jj}{ii}(kk,2),jj) = 1;
        end
    end
    count = count + size(Run(ii).Input,1);
end

% Determine the factor for straight data.
ep = 0;
sp = 0;
for nn = 1:3
    for ii = 1:size(Run, 2)
        for jj = 1:n{nn}(ii)
            ep = ep + np{nn}{ii}(jj);
        end
    end
end
for ii = 1:size(Run, 2)
    for jj = 1:n{4}(ii)
        sp = sp + np{4}{ii}(jj);
    end
```

```
end
ep = ep/3;
sf = ep/sp;

% Indicies to use for training and testing.
% Hidden testing indicies are from events that are not used for training.
Train_Ind = cell(1,CV_Folds);
Test_Ind = cell(1,CV_Folds);
Test_Ind_Visible = cell(1,CV_Folds);
Test_Ind_Hidden = cell(1,CV_Folds);
for ii = 1:CV_Folds
    Train_Ind{ii} = cell(1,4);
    Test_Ind{ii} = cell(1,4);
    Test_Ind_Visible{ii} = cell(1,4);
    Test_Ind_Hidden{ii} = cell(1,4);
end

% Clean up cell.
clearvars ii jj kk count ep sp

%% Test Dependent Setup

% Test #1:
% All data from all events is eligible training data.
if Test_Num == 1

    % Loop through all folds.
    for ff = 1:CV_Folds

        % Counter to keep track of position.
        count = 0;

        % Loop through all runs.
        for ii = 1:size(Run, 2)

            % Loop through all classes.
            for nn = 1:4

                % Loop through all events.
                for jj = 1:n{nn}(ii)

                    % All indices.
                    total = p{nn}{ii}(jj,1):p{nn}{ii}(jj,2);

                    % Treat straight data differently.
                    if nn == 4

                        % Apply factor.
                        eq = round(length(total)*sf);
                        eq = round(quantile(total,eq));

                        % Take removed points as testing data.
                        test = total(~ismember(total,eq));

                        % Divide remaining into testing and training.
                        test = horzcat(test, downsample(eq,CV_Folds,ff-1));
                        train = eq(~ismember(eq,test));

                        % Record.
                        Train_Ind{ff}{nn}= ...
                            horzcat(Train_Ind{ff}{nn}, train+count);
                        Test_Ind{ff}{nn} = ...
                            horzcat(Test_Ind{ff}{nn}, test+count);
                        Test_Ind_Visible{ff}{nn} = ...
                            horzcat(Test_Ind_Visible{ff}{nn}, test+count);

                    else

                        % Create test and validation sets.
                        test = downsample(total,CV_Folds,ff-1);

                        % Create training set.
                        train = total(~ismember(total,test));
```

```
                        % Record.
                        Train_Ind{ff}{nn} = ...
                            horzcat(Train_Ind{ff}{nn}, train+count);
                        Test_Ind{ff}{nn} = ...
                            horzcat(Test_Ind{ff}{nn}, test+count);
                        Test_Ind_Visible{ff}{nn} = ...
                            horzcat(Test_Ind_Visible{ff}{nn}, test+count);

                    end
                end
            end

            % Increment position.
            count = count + size(Run(ii).Input,1);

        end
    end
end

% Test #2:
% The data from some events is not eligible training data.
% These events are selected in the same manner as the cross-validation
% sets.
if Test_Num == 2 || Test_Num == 4

    % Loop through all folds.
    for ff = 1:CV_Folds

        % Determine which events to hide.
        % This is done evenly and in a repeatable manner by downsampling.
        h = cell(1,4);
        for nn = 1:4
            h{nn} = cell(1,size(n{nn},2));
        end
        for nn = 1:3

            % Sum number of events.
            sum_n = sum(n{nn});

            % Downsample.
            sum_ds = downsample(1:sum_n,CV_Folds,ff-1);

            % Restore association with run.
            count = 0;
            for ii = 1:size(n{nn}, 2)
                h{nn}{ii} = sum_ds - count;
                h{nn}{ii} = h{nn}{ii}(h{nn}{ii} > 0);
                h{nn}{ii} = h{nn}{ii}(h{nn}{ii} <= n{nn}(ii));
                count = count + n{nn}(ii);
            end
        end

        % Counter to keep track of position.
        count = 0;

        % Loop through all runs.
        for ii = 1:size(Run, 2)

            % Loop through all event classes.
            for nn = 1:4

                % Loop through all events.
                for jj = 1:n{nn}(ii)

                    % All indices.
                    total = p{nn}{ii}(jj,1):p{nn}{ii}(jj,2);

                    % Check if this event is hidden.
                    if ismember(jj, h{nn}{ii})

                        % The event is hidden.
                        % All test.
```

```
                        % Record.
                        Test_Ind{ff}{nn} = ...
                            horzcat(Test_Ind{ff}{nn}, total+count);
                        Test_Ind_Hidden{ff}{nn} = ...
                            horzcat(Test_Ind_Hidden{ff}{nn}, total+count);

                    else

                        % Visible.
                        % Treat straight data differently.
                        if nn == 4

                            % Apply factor.
                            eq = round(length(total)*sf);
                            eq = round(quantile(total,eq));

                            % Take removed points as testing data.
                            test = total(~ismember(total,eq));

                            % Divide remaining into testing and training.
                            test = horzcat(test, downsample(eq,CV_Folds,ff-1));
                            train = eq(~ismember(eq,test));

                            % Record.
                            Train_Ind{ff}{nn} = ...
                                horzcat(Train_Ind{ff}{nn}, train+count);
                            Test_Ind{ff}{nn} = ...
                                horzcat(Test_Ind{ff}{nn}, test+count);
                            Test_Ind_Visible{ff}{nn} = ...
                                horzcat(Test_Ind_Visible{ff}{nn}, test+count);

                        else

                            % Create test and validation sets.
                            test = downsample(total,CV_Folds,ff-1);

                            % Create training set.
                            train = total(~ismember(total,test));

                            % Record.
                            Train_Ind{ff}{nn} = ...
                                horzcat(Train_Ind{ff}{nn}, train+count);
                            Test_Ind{ff}{nn} = ...
                                horzcat(Test_Ind{ff}{nn}, test+count);
                            Test_Ind_Visible{ff}{nn} = ...
                                horzcat(Test_Ind_Visible{ff}{nn}, test+count);

                        end
                    end
                end
            end

            % Increment position.
            count = count + size(Run(ii).Input,1);

        end
    end
end

% Test #3:
% The data from some events is not eligible training data.
% These events are selected in the same manner as the cross-validation
% sets.
% The event with the minimum number of eligible data points after this
% process sets the maximum number of eligible data points for the other
% events.
% This results in an approximately equal number of training data points
% from each event.
if Test_Num == 3 || Test_Num == 5

    % Loop through all folds.
    for ff = 1:CV_Folds
```

```
% Determine which events to hide.
% This is done evenly and in a repeatable manner by downsampling.
h = cell(1,4);
for nn = 1:4
    h{nn} = cell(1,size(n{nn},2));
end
for nn = 1:3

    % Sum number of events.
    sum_n = sum(n{nn});

    % Downsample.
    sum_ds = downsample(1:sum_n,CV_Folds,ff-1);

    % Restore association with run.

    count = 0;
    for ii = 1:size(n{nn},2)
        h{nn}{ii} = sum_ds - count;
        h{nn}{ii} = h{nn}{ii}(h{nn}{ii} > 0);
        h{nn}{ii} = h{nn}{ii}(h{nn}{ii} <= n{nn}(ii));
        count = count + n{nn}(ii);
    end
end

% Determine the equalizing factor for each class.
ef = zeros(1,4);
for nn = 1:4

    % Loop through all runs.
    for ii = 1:size(Run, 2)

        % Loop through all events.
        for jj = 1:n{nn}(ii)

            % If the event is not hidden, add to total.
            if nn == 4 || ~ismember(jj, h{nn}{ii})
                ef(nn) = ef(nn) + np{nn}{ii}(jj);
            end

        end
    end
end
ef = min(ef)./ef;

% Counter to keep track of position.
count = 0;

% Loop through all runs.
for ii = 1:size(Run, 2)

    % Loop through all event classes.
    for nn = 1:4

        % Loop through all events.
        for jj = 1:n{nn}(ii)

            % All indices.
            total = p{nn}{ii}(jj,1):p{nn}{ii}(jj,2);

            % Check if this event is hidden.
            if ismember(jj, h{nn}{ii})

                % The event is hidden.
                % All test.
                % Record.
                Test_Ind{ff}{nn} = ...
                    horzcat(Test_Ind{ff}{nn}, total+count);
                Test_Ind_Hidden{ff}{nn} = ...
                    horzcat(Test_Ind_Hidden{ff}{nn}, total+count);

            else
```

```
                        % Visible.
                        % Apply equalizing factor.
                        eq = round(length(total)*ef(nn));
                        eq = round(quantile(total,eq));

                        % Take removed points as testing data.
                        test = total(~ismember(total,eq));

                        % Divide remaining into testing and training.
                        test = horzcat(test, downsample(eq,CV_Folds,ff-1));
                        train = eq(~ismember(eq,test));

                        % Record.
                        Train_Ind{ff}{nn} = ...
                            horzcat(Train_Ind{ff}{nn}, train+count);
                        Test_Ind{ff}{nn} = ...
                            horzcat(Test_Ind{ff}{nn}, test+count);
                        Test_Ind_Visible{ff}{nn} = ...
                            horzcat(Test_Ind_Visible{ff}{nn}, test+count);

                    end
                end
            end

            % Increment position.
            count = count + size(Run(ii).Input,1);

        end
    end
end

% Merge index lists.
Train_Ind_All = cell(1,CV_Folds);
Test_Ind_All = cell(1,CV_Folds);
Test_Ind_Visible_All = cell(1,CV_Folds);
Test_Ind_Hidden_All = cell(1,CV_Folds);
for ff = 1:CV_Folds
    for nn = 1:4
        Train_Ind_All{ff} = horzcat(Train_Ind_All{ff}, Train_Ind{ff}{nn});
        Test_Ind_All{ff} = horzcat(Test_Ind_All{ff}, Test_Ind{ff}{nn});
        Test_Ind_Visible_All{ff} = ...
            horzcat(Test_Ind_Visible_All{ff}, Test_Ind_Visible{ff}{nn});
        Test_Ind_Hidden_All{ff} = ...
            horzcat(Test_Ind_Hidden_All{ff}, Test_Ind_Hidden{ff}{nn});
    end
end

% Clean up cell.
clearvars nn ii jj ff n np count train test total eq ef h sum_n sum_ds p sf

%% Remove Run Data

% The data for each run is no longer needed.
clearvars Run

% Record time.
Setup_For_Classification_Time = toc;
Setup_For_Classification_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars cpu_tic
```

B.13 Large_MLP.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Important Classification Constants
% Changing these constants will significantly impact the results.

% Start timer.
tic;
cpu_tic = cputime;

% Parameters for pattern recognition networks.
% Size.
% With 13 inputs and 4 outputs, this results in 2279 weight elements.
MLP.Dim = [50, 25, 10];

% Maximum epochs.
MLP.Epochs = 250;

% Number of entities in each committee.
% Must be a prime number greater than four to work properly with this code.
Num_Committee = 5;

%% Set Data for Fold

Train_Ind = Train_Ind{Fold_Num};
Test_Ind = Test_Ind{Fold_Num};
Test_Ind_Visible = Test_Ind_Visible{Fold_Num};
Test_Ind_Hidden = Test_Ind_Hidden{Fold_Num};
Train_Ind_All = Train_Ind_All{Fold_Num};
Test_Ind_All = Test_Ind_All{Fold_Num};
Test_Ind_Visible_All = Test_Ind_Visible_All{Fold_Num};
Test_Ind_Hidden_All = Test_Ind_Hidden_All{Fold_Num};

%% Set up Neural Networks

% Create a committee of pattern recognition networks.
mlp_net = cell(1,Num_Committee);
mlp_net_tr = cell(1,Num_Committee);
for ii = 1:Num_Committee

    % Create the network.
    mlp_net{ii} = patternnet(MLP.Dim);
    mlp_net{ii}.trainFcn = 'trainscg';
    mlp_net{ii}.trainParam.epochs = MLP.Epochs;

    % Assign division.
    mlp_net{ii}.divideFcn = 'divideind';
    mlp_net{ii}.divideParam.trainInd = 1:length(Train_Ind_All);
    mlp_net{ii}.divideParam.valInd = [];
    mlp_net{ii}.divideParam.testInd = [];

    % Don't show training GUI.
    mlp_net{ii}.trainParam.showWindow = 0;

end

% Clean up cell.
clearvars ii

%% Train the Networks

% Storage for CPU time in the parallel loop.
par_cpu_time = zeros(1,Num_Committee);

% Loop through all the networks and train them.
% This will take a long time.
parfor ii = 1:Num_Committee

    % Train.
    par_cpu_tic = cputime;
    [mlp_net{ii}, mlp_net_tr{ii}] = ...
        train(mlp_net{ii}, X(Train_Ind_All,:)', T(Train_Ind_All,:)');
```

```
    par_cpu_time(ii) = cputime-par_cpu_tic;

end

% Add to structure.
MLP.Net = mlp_net;
MLP.Net_TR = mlp_net_tr;

% Record time.
MLP.Net_Train_Time = toc;
MLP.Net_Train_CPU_Time = sum(par_cpu_time)+cputime-cpu_tic;

% Clean up cell.
clearvars ii mlp_net mlp_net_tr cpu_tic par_cpu_tic par_cpu_time

%% Calculate Output

% Time storage.
prob_time = 0;
prob_cpu_time = 0;
ans_time = 0;
ans_cpu_time = 0;

% Storage.
MLP.Net_Output_Prob = cell(size(MLP.Net));
MLP.Net_Output_Ans = cell(size(MLP.Net));

% Loop through all networks.
for ii = 1:Num_Committee

    % Get the output.
    tic;
    cpu_tic = cputime;
    MLP.Net_Output_Prob{ii} = MLP.Net{ii}(X')';
    prob_time = prob_time+toc;
    prob_cpu_time = prob_cpu_time+cputime-cpu_tic;

    % Convert to answer.
    tic;
    cpu_tic = cputime;
    MLP.Net_Output_Ans{ii} = zeros(size(MLP.Net_Output_Prob{ii}));
    for jj = 1:size(MLP.Net_Output_Prob{ii},1)
        [~,max_index] = max(MLP.Net_Output_Prob{ii}(jj,:));
        MLP.Net_Output_Ans{ii}(jj,max_index) = 1;
    end
    ans_time = ans_time+toc;
    ans_cpu_time = ans_cpu_time+cputime-cpu_tic;
end

% Record time.
MLP.Net_Output_Time = prob_time+ans_time;
MLP.Net_Output_CPU_Time = prob_cpu_time+ans_cpu_time;
MLP.Avg_Prob_Commitee_Output_Time = prob_time;
MLP.Avg_Prob_Commitee_Output_CPU_Time = prob_cpu_time;
MLP.Net_Prob_Commitee_Output_Time = prob_time;
MLP.Net_Prob_Commitee_Output_CPU_Time = prob_cpu_time;
MLP.Net_Prob_Committee_Train_Time = MLP.Net_Train_Time+prob_time;
MLP.Net_Prob_Committee_Train_CPU_Time = MLP.Net_Train_CPU_Time+prob_cpu_time;

% Clean up cell.
clearvars ii jj max_index cpu_tic prob_time prob_cpu_time ans_time ans_cpu_time

%% Averaged Probability Committee

% Start timer.
tic;
cpu_tic = cputime;

% Storage.
MLP.Avg_Prob_Committee_Output = zeros(size(T));

% Loop through all networks.
for ii = 1:Num_Committee
```

```
    % Add to sum.
    MLP.Avg_Prob_Committee_Output = ...
        MLP.Avg_Prob_Committee_Output + MLP.Net_Output_Prob{ii};

end

% Average.
MLP.Avg_Prob_Committee_Output = MLP.Avg_Prob_Committee_Output/Num_Committee;

% Convert to answer.
temp = zeros(size(MLP.Avg_Prob_Committee_Output));
for ii = 1:size(MLP.Avg_Prob_Committee_Output,1)
    [~,max_index] = max(MLP.Avg_Prob_Committee_Output(ii,:));
    temp(ii,max_index) = 1;
end
MLP.Avg_Prob_Committee_Output = temp;

% Record time.
MLP.Avg_Prob_Commitee_Output_Time = MLP.Avg_Prob_Commitee_Output_Time+toc;
MLP.Avg_Prob_Commitee_Output_CPU_Time = ...
    MLP.Avg_Prob_Commitee_Output_CPU_Time+cputime-cpu_tic;

% Clean up cell.
clearvars ii cpu_tic temp

%% Averaged Answer Committee

% Start timer.
tic;
cpu_tic = cputime;

% Storage.
MLP.Avg_Ans_Committee_Output = zeros(size(T));

% Loop through all networks.
for ii = 1:Num_Committee

    % Add to sum.
    MLP.Avg_Ans_Committee_Output = ...
        MLP.Avg_Ans_Committee_Output + MLP.Net_Output_Ans{ii};

end

% Break ties.
for ii = 1:size(MLP.Avg_Ans_Committee_Output,1)

    % Pull out the row.
    row = MLP.Avg_Ans_Committee_Output(ii,:);

    % Find the maximum value in the row.
    [win_value, win_index] = max(row);

    % Make a mask of all the classes with the maximum votes.
    mask = row == win_value;

    % Check if there is a tie.
    if length(row(mask)) > 1

        % There is a tie.
        % Find which networks produced non-tied votes.
        % Loop through the committee members.
        for jj = 1:Num_Committee

            % Get the answer.
            [~, lose_index] = max(MLP.Net_Output_Ans{jj}(ii,:));

            % Check if the answer is a non-tied vote.
            if mask(lose_index) == 0

                % Apply the mask to only consider classes that break the tie.
                prob = MLP.Net_Output_Prob{jj}(ii,:).*mask;
                [~, win_index] = max(prob);
```

```
                % Adjust values to break the tie.
                MLP.Avg_Ans_Committee_Output(ii,win_index) = ...
                    MLP.Avg_Ans_Committee_Output(ii,win_index) + 1;
                MLP.Avg_Ans_Committee_Output(ii,lose_index) = ...
                    MLP.Avg_Ans_Committee_Output(ii,lose_index) - 1;

            end
        end
    end
end

% Average.
MLP.Avg_Ans_Committee_Output = MLP.Avg_Ans_Committee_Output/Num_Committee;

% Convert to answer.
temp = zeros(size(MLP.Avg_Ans_Committee_Output));
for ii = 1:size(MLP.Avg_Ans_Committee_Output,1)
    [~,max_index] = max(MLP.Avg_Ans_Committee_Output(ii,:));
    temp(ii,max_index) = 1;
end
MLP.Avg_Ans_Committee_Output = temp;

% Record time.
MLP.Avg_Ans_Commitee_Output_Time = MLP.Net_Output_Time+toc;
MLP.Avg_Ans_Commitee_Output_CPU_Time = MLP.Net_Output_CPU_Time+cputime-cpu_tic;

% Clean up cell.
clearvars ii jj temp row win_value win_index mask lose_index prob cpu_tic

%% Networked Probability Committee

% Start timer.
tic;
cpu_tic = cputime;

% Input.
x = zeros(size(X,1), Num_Committee*4);

% Loop through all networks.
for ii = 1:Num_Committee

    % Copy in output.
    x(:,(ii-1)*4+1:ii*4) = MLP.Net_Output_Prob{ii};

end

% Record time.
input_time = toc;
input_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Create the network.
MLP.Net_Prob_Committee = patternnet(4);
MLP.Net_Prob_Committee.trainFcn = 'trainscg';
MLP.Net_Prob_Committee.trainParam.epochs = MLP.Epochs;

% Assign division.
MLP.Net_Prob_Committee.divideFcn = 'divideind';
MLP.Net_Prob_Committee.divideParam.trainInd = 1:length(Train_Ind_All);
MLP.Net_Prob_Committee.divideParam.valInd = [];
MLP.Net_Prob_Committee.divideParam.testInd = [];

% Train.
MLP.Net_Prob_Committee.trainParam.showWindow = false;
[MLP.Net_Prob_Committee, MLP.Net_Prob_Committee_TR] = ...
    train(MLP.Net_Prob_Committee, x(Train_Ind_All,:)', T(Train_Ind_All,:)');

% Record time.
train_time = toc;
```

```
train_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Capture output.
MLP.Net_Prob_Committee_Output = MLP.Net_Prob_Committee(x')';

% Record time.
output_time = toc;
output_cpu_time = cputime-cpu_tic;

% Add times.
MLP.Net_Prob_Commitee_Output_Time = ...
    MLP.Net_Prob_Commitee_Output_Time+input_time+output_time;
MLP.Net_Prob_Commitee_Output_CPU_Time = ...
    MLP.Net_Prob_Commitee_Output_CPU_Time+input_cpu_time+output_cpu_time;
MLP.Net_Prob_Committee_Train_Time = ...
    MLP.Net_Prob_Committee_Train_Time+input_time+train_time;
MLP.Net_Prob_Committee_Train_CPU_Time = ...
    MLP.Net_Prob_Committee_Train_CPU_Time+input_cpu_time+train_cpu_time;

% Clean up cell.
clearvars ii x cpu_tic input_time input_cpu_time train_time train_cpu_time ...
    output_time output_cpu_time

%% Networked Answer Committee

% Start timer.
tic;
cpu_tic = cputime;

% Input.
x = zeros(size(X,1), Num_Committee*4);

% Loop through all networks.
for ii = 1:Num_Committee

    % Copy in output.
    x(:,(ii-1)*4+1:ii*4) = MLP.Net_Output_Ans{ii};

end

% Record time.
input_time = toc;
input_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Create the network.
MLP.Net_Ans_Committee = patternnet(4);
MLP.Net_Ans_Committee.trainParam.epochs = MLP.Epochs;

% Assign division.
MLP.Net_Ans_Committee.divideFcn = 'divideind';
MLP.Net_Ans_Committee.divideParam.trainInd = 1:length(Train_Ind_All);
MLP.Net_Ans_Committee.divideParam.valInd = [];
MLP.Net_Ans_Committee.divideParam.testInd = [];

% Train.
MLP.Net_Ans_Committee.trainParam.showWindow = false;
[MLP.Net_Ans_Committee, MLP.Net_Ans_Committee_TR] = ...
    train(MLP.Net_Ans_Committee, x(Train_Ind_All,:)', T(Train_Ind_All,:)');

% Record time.
train_time = toc;
train_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
```

```
cpu_tic = cputime;

% Capture output.
MLP.Net_Ans_Committee_Output = MLP.Net_Ans_Committee(x')';

% Record time.
output_time = toc;
output_cpu_time = cputime-cpu_tic;

% Add times.
MLP.Net_Ans_Commitee_Output_Time = ...
    MLP.Net_Output_Time+input_time+output_time;
MLP.Net_Ans_Commitee_Output_CPU_Time = ...
    MLP.Net_Output_CPU_Time+input_cpu_time+output_cpu_time;
MLP.Net_Ans_Committee_Train_Time = ...
    MLP.Net_Train_Time+MLP.Net_Output_Time+input_time+train_time;
MLP.Net_Ans_Committee_Train_CPU_Time = ...
    MLP.Net_Train_CPU_Time+MLP.Net_Output_CPU_Time+input_cpu_time+train_cpu_time;

% Clean up cell.
clearvars ii x cpu_tic input_time input_cpu_time train_time train_cpu_time ...
    output_time output_cpu_time
```

B.14 Small_MLP.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Important Classification Constants
% Changing these constants will significantly impact the results.

% Start timer.
tic;
cpu_tic = cputime;

% Parameters for pattern recognition networks.
% Size.
% With 13 inputs and 4 outputs, this results in 219 weight elements.
MLP_Small.Dim = [10, 5];

% Maximum epochs.
MLP_Small.Epochs = 250;

% Number of entities in each committee.
% Must be a prime number greater than four to work properly with this code.
Num_Committee = 5;

%% Set Data for Fold

Train_Ind = Train_Ind{Fold_Num};
Test_Ind = Test_Ind{Fold_Num};
Test_Ind_Visible = Test_Ind_Visible{Fold_Num};
Test_Ind_Hidden = Test_Ind_Hidden{Fold_Num};
Train_Ind_All = Train_Ind_All{Fold_Num};
Test_Ind_All = Test_Ind_All{Fold_Num};
Test_Ind_Visible_All = Test_Ind_Visible_All{Fold_Num};
Test_Ind_Hidden_All = Test_Ind_Hidden_All{Fold_Num};

%% Set up Neural Networks

% Create a committee of pattern recognition networks.
mlp_net = cell(1,Num_Committee);
mlp_net_tr = cell(1,Num_Committee);
for ii = 1:Num_Committee

    % Create the network.
    mlp_net{ii} = patternnet(MLP_Small.Dim);
    mlp_net{ii}.trainFcn = 'trainscg';
    mlp_net{ii}.trainParam.epochs = MLP_Small.Epochs;

    % Assign division.
    mlp_net{ii}.divideFcn = 'divideind';
    mlp_net{ii}.divideParam.trainInd = 1:length(Train_Ind_All);
    mlp_net{ii}.divideParam.valInd = [];
    mlp_net{ii}.divideParam.testInd = [];

    % Don't show training GUI.
    mlp_net{ii}.trainParam.showWindow = 0;

end

% Clean up cell.
clearvars ii

%% Train the Networks

% Storage for CPU time in the parallel loop.
par_cpu_time = zeros(1,Num_Committee);

% Loop through all the networks and train them.
% This will take a long time.
parfor ii = 1:Num_Committee

    % Train.
    par_cpu_tic = cputime;
    [mlp_net{ii}, mlp_net_tr{ii}] = ...
        train(mlp_net{ii}, X(Train_Ind_All,:)', T(Train_Ind_All,:)');
```

```
    par_cpu_time(ii) = cputime-par_cpu_tic;

end

% Add to structure.
MLP_Small.Net = mlp_net;
MLP_Small.Net_TR = mlp_net_tr;

% Record time.
MLP_Small.Net_Train_Time = toc;
MLP_Small.Net_Train_CPU_Time = sum(par_cpu_time)+cputime-cpu_tic;

% Clean up cell.
clearvars ii mlp_net mlp_net_tr cpu_tic par_cpu_tic par_cpu_time

%% Calculate Output

% Time storage.
prob_time = 0;
prob_cpu_time = 0;
ans_time = 0;
ans_cpu_time = 0;

% Storage.
MLP_Small.Net_Output_Prob = cell(size(MLP_Small.Net));
MLP_Small.Net_Output_Ans = cell(size(MLP_Small.Net));

% Loop through all networks.
for ii = 1:Num_Committee

    % Get the output.
    tic;
    cpu_tic = cputime;
    MLP_Small.Net_Output_Prob{ii} = MLP_Small.Net{ii}(X')';
    prob_time = prob_time+toc;
    prob_cpu_time = prob_cpu_time+cputime-cpu_tic;

    % Convert to answer.
    tic;
    cpu_tic = cputime;
    MLP_Small.Net_Output_Ans{ii} = zeros(size(MLP_Small.Net_Output_Prob{ii}));
    for jj = 1:size(MLP_Small.Net_Output_Prob{ii},1)
        [~,max_index] = max(MLP_Small.Net_Output_Prob{ii}(jj,:));
        MLP_Small.Net_Output_Ans{ii}(jj,max_index) = 1;
    end
    ans_time = ans_time+toc;
    ans_cpu_time = ans_cpu_time+cputime-cpu_tic;
end

% Record time.
MLP_Small.Net_Output_Time = prob_time+ans_time;
MLP_Small.Net_Output_CPU_Time = prob_cpu_time+ans_cpu_time;
MLP_Small.Avg_Prob_Commitee_Output_Time = prob_time;
MLP_Small.Avg_Prob_Commitee_Output_CPU_Time = prob_cpu_time;
MLP_Small.Net_Prob_Commitee_Output_Time = prob_time;
MLP_Small.Net_Prob_Commitee_Output_CPU_Time = prob_cpu_time;
MLP_Small.Net_Prob_Committee_Train_Time = MLP_Small.Net_Train_Time+prob_time;
MLP_Small.Net_Prob_Committee_Train_CPU_Time = ...
    MLP_Small.Net_Train_CPU_Time+prob_cpu_time;

% Clean up cell.
clearvars ii jj max_index cpu_tic prob_time prob_cpu_time ans_time ans_cpu_time

%% Averaged Probability Committee

% Start timer.
tic;
cpu_tic = cputime;

% Storage.
MLP_Small.Avg_Prob_Committee_Output = zeros(size(T));

% Loop through all networks.
```

```
for ii = 1:Num_Committee

    % Add to sum.
    MLP_Small.Avg_Prob_Committee_Output = ...
        MLP_Small.Avg_Prob_Committee_Output + MLP_Small.Net_Output_Prob{ii};

end

% Average.
MLP_Small.Avg_Prob_Committee_Output = ...
    MLP_Small.Avg_Prob_Committee_Output/Num_Committee;

% Convert to answer.
temp = zeros(size(MLP_Small.Avg_Prob_Committee_Output));
for ii = 1:size(MLP_Small.Avg_Prob_Committee_Output,1)
    [~,max_index] = max(MLP_Small.Avg_Prob_Committee_Output(ii,:));
    temp(ii,max_index) = 1;
end
MLP_Small.Avg_Prob_Committee_Output = temp;

% Record time.
MLP_Small.Avg_Prob_Commitee_Output_Time = ...
    MLP_Small.Avg_Prob_Commitee_Output_Time+toc;
MLP_Small.Avg_Prob_Commitee_Output_CPU_Time = ...
    MLP_Small.Avg_Prob_Commitee_Output_CPU_Time+cputime-cpu_tic;

% Clean up cell.
clearvars ii cpu_tic temp

%% Averaged Answer Committee

% Start timer.
tic;
cpu_tic = cputime;

% Storage.
MLP_Small.Avg_Ans_Committee_Output = zeros(size(T));

% Loop through all networks.
for ii = 1:Num_Committee

    % Add to sum.
    MLP_Small.Avg_Ans_Committee_Output = ...
        MLP_Small.Avg_Ans_Committee_Output + MLP_Small.Net_Output_Ans{ii};

end

% Break ties.
for ii = 1:size(MLP_Small.Avg_Ans_Committee_Output,1)

    % Pull out the row.
    row = MLP_Small.Avg_Ans_Committee_Output(ii,:);

    % Find the maximum value in the row.
    [win_value, win_index] = max(row);

    % Make a mask of all the classes with the maximum votes.
    mask = row == win_value;

    % Check if there is a tie.
    if length(row(mask)) > 1

        % There is a tie.
        % Find which networks produced non-tied votes.
        % Loop through the committee members.
        for jj = 1:Num_Committee

            % Get the answer.
            [~, lose_index] = max(MLP_Small.Net_Output_Ans{jj}(ii,:));

            % Check if the answer is a non-tied vote.
            if mask(lose_index) == 0
```

```
                % Apply the mask to only consider classes that break the tie.
                prob = MLP_Small.Net_Output_Prob{jj}(ii,:).*mask;
                [~, win_index] = max(prob);

                % Adjust values to break the tie.
                MLP_Small.Avg_Ans_Committee_Output(ii,win_index) = ...
                    MLP_Small.Avg_Ans_Committee_Output(ii,win_index) + 1;
                MLP_Small.Avg_Ans_Committee_Output(ii,lose_index) = ...
                    MLP_Small.Avg_Ans_Committee_Output(ii,lose_index) - 1;

            end
        end
    end
end

% Average.
MLP_Small.Avg_Ans_Committee_Output = ...
    MLP_Small.Avg_Ans_Committee_Output/Num_Committee;

% Convert to answer.
temp = zeros(size(MLP_Small.Avg_Ans_Committee_Output));
for ii = 1:size(MLP_Small.Avg_Ans_Committee_Output,1)
    [~,max_index] = max(MLP_Small.Avg_Ans_Committee_Output(ii,:));
    temp(ii,max_index) = 1;
end
MLP_Small.Avg_Ans_Committee_Output = temp;

% Record time.
MLP_Small.Avg_Ans_Commitee_Output_Time = MLP_Small.Net_Output_Time+toc;
MLP_Small.Avg_Ans_Commitee_Output_CPU_Time = ...
    MLP_Small.Net_Output_CPU_Time+cputime-cpu_tic;

% Clean up cell.
clearvars ii jj temp row win_value win_index mask lose_index prob cpu_tic

%% Networked Probability Committee

% Start timer.
tic;
cpu_tic = cputime;

% Input.
x = zeros(size(X,1), Num_Committee*4);

% Loop through all networks.
for ii = 1:Num_Committee

    % Copy in output.
    x(:,(ii-1)*4+1:ii*4) = MLP_Small.Net_Output_Prob{ii};

end

% Record time.
input_time = toc;
input_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Create the network.
MLP_Small.Net_Prob_Committee = patternnet(4);
MLP_Small.Net_Prob_Committee.trainFcn = 'trainscg';
MLP_Small.Net_Prob_Committee.trainParam.epochs = MLP_Small.Epochs;

% Assign division.
MLP_Small.Net_Prob_Committee.divideFcn = 'divideind';
MLP_Small.Net_Prob_Committee.divideParam.trainInd = 1:length(Train_Ind_All);
MLP_Small.Net_Prob_Committee.divideParam.valInd = [];
MLP_Small.Net_Prob_Committee.divideParam.testInd = [];

% Train.
MLP_Small.Net_Prob_Committee.trainParam.showWindow = false;
```

```
[MLP_Small.Net_Prob_Committee, MLP_Small.Net_Prob_Committee_TR] = ...
    train(MLP_Small.Net_Prob_Committee, x(Train_Ind_All,:)', T(Train_Ind_All,:)');

% Record time.
train_time = toc;
train_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Capture output.
MLP_Small.Net_Prob_Committee_Output = MLP_Small.Net_Prob_Committee(x')';

% Record time.
output_time = toc;
output_cpu_time = cputime-cpu_tic;

% Add times.
MLP_Small.Net_Prob_Commitee_Output_Time = ...
    MLP_Small.Net_Prob_Commitee_Output_Time+input_time+output_time;
MLP_Small.Net_Prob_Commitee_Output_CPU_Time = ...
    MLP_Small.Net_Prob_Commitee_Output_CPU_Time+input_cpu_time+output_cpu_time;
MLP_Small.Net_Prob_Committee_Train_Time = ...
    MLP_Small.Net_Prob_Committee_Train_Time+input_time+train_time;
MLP_Small.Net_Prob_Committee_Train_CPU_Time = ...
    MLP_Small.Net_Prob_Committee_Train_CPU_Time+input_cpu_time+train_cpu_time;

% Clean up cell.
clearvars ii x cpu_tic input_time input_cpu_time train_time train_cpu_time ...
    output_time output_cpu_time

%% Networked Answer Committee

% Start timer.
tic;
cpu_tic = cputime;

% Input.
x = zeros(size(X,1), Num_Committee*4);

% Loop through all networks.
for ii = 1:Num_Committee

    % Copy in output.
    x(:,(ii-1)*4+1:ii*4) = MLP_Small.Net_Output_Ans{ii};

end

% Record time.
input_time = toc;
input_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Create the network.
MLP_Small.Net_Ans_Committee = patternnet(4);
MLP_Small.Net_Ans_Committee.trainParam.epochs = MLP_Small.Epochs;

% Assign division.
MLP_Small.Net_Ans_Committee.divideFcn = 'divideind';
MLP_Small.Net_Ans_Committee.divideParam.trainInd = 1:length(Train_Ind_All);
MLP_Small.Net_Ans_Committee.divideParam.valInd = [];
MLP_Small.Net_Ans_Committee.divideParam.testInd = [];

% Train.
MLP_Small.Net_Ans_Committee.trainParam.showWindow = false;
[MLP_Small.Net_Ans_Committee, MLP_Small.Net_Ans_Committee_TR] = ...
    train(MLP_Small.Net_Ans_Committee, x(Train_Ind_All,:)', T(Train_Ind_All,:)');

% Record time.
```

```
train_time = toc;
train_cpu_time = cputime-cpu_tic;

% Start timer.
tic;
cpu_tic = cputime;

% Capture output.
MLP_Small.Net_Ans_Committee_Output = MLP_Small.Net_Ans_Committee(x')';

% Record time.
output_time = toc;
output_cpu_time = cputime-cpu_tic;

% Add times.
MLP_Small.Net_Ans_Commitee_Output_Time = ...
    MLP_Small.Net_Output_Time+input_time+output_time;
MLP_Small.Net_Ans_Commitee_Output_CPU_Time = ...
    MLP_Small.Net_Output_CPU_Time+input_cpu_time+output_cpu_time;
MLP_Small.Net_Ans_Committee_Train_Time = ...
    MLP_Small.Net_Train_Time+MLP_Small.Net_Output_Time+input_time+train_time;
MLP_Small.Net_Ans_Committee_Train_CPU_Time = ...
    MLP_Small.Net_Train_CPU_Time+MLP_Small.Net_Output_CPU_Time+...
    input_cpu_time+train_cpu_time;

% Clean up cell.
clearvars ii x cpu_tic input_time input_cpu_time train_time train_cpu_time ...
    output_time output_cpu_time
```

B.15 NN.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% Set Data for Fold

% Start timer.
tic;
cpu_tic = cputime;

Train_Ind = Train_Ind{Fold_Num};
Test_Ind = Test_Ind{Fold_Num};
Test_Ind_Visible = Test_Ind_Visible{Fold_Num};
Test_Ind_Hidden = Test_Ind_Hidden{Fold_Num};
Train_Ind_All = Train_Ind_All{Fold_Num};
Test_Ind_All = Test_Ind_All{Fold_Num};
Test_Ind_Visible_All = Test_Ind_Visible_All{Fold_Num};
Test_Ind_Hidden_All = Test_Ind_Hidden_All{Fold_Num};

%% Find Nearest Neighbors

% Find nearest neighbor to each point in the training set.
obj = KDTreeSearcher(X(Train_Ind_All,:));
idx = knnsearch(obj, X, 'K', 1);

% Find output.
NN.Output = T(Train_Ind_All(idx),:);

% Record time.
NN.Output_Time = toc;
NN.Output_CPU_Time = cputime-cpu_tic;

% Clean up cell.
clearvars cpu_tic obj idx
```

B.16 Classification_Result.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
%% MLP Individual Network Performance

% Column identifiers.
train_col = 1;
test_col = 2;
test_visible_col = 3;
test_hidden_col = 4;

% Storage.
MLP.Net_Confusion = cell(1,Num_Committee);

% Loop through all networks.
for ii = 1:Num_Committee

    % Storage.
    MLP.Net_Confusion{ii} = cell(1,size(T,2)+1);

    % Calculate confusion for each output class.
    for jj = 1:size(T,2)
        MLP.Net_Confusion{ii}{jj} = zeros(1,test_hidden_col);
        MLP.Net_Confusion{ii}{jj}(train_col) = ...
            confusion(T(Train_Ind{jj},:)', ...
            MLP.Net_Output_Ans{ii}(Train_Ind{jj},:)');
        MLP.Net_Confusion{ii}{jj}(test_col) = ...
            confusion(T(Test_Ind{jj},:)', ...
            MLP.Net_Output_Ans{ii}(Test_Ind{jj},:)');
        MLP.Net_Confusion{ii}{jj}(test_visible_col) = ...
            confusion(T(Test_Ind_Visible{jj},:)', ...
            MLP.Net_Output_Ans{ii}(Test_Ind_Visible{jj},:)');
        MLP.Net_Confusion{ii}{jj}(test_hidden_col) = ...
            confusion(T(Test_Ind_Hidden{jj},:)', ...
            MLP.Net_Output_Ans{ii}(Test_Ind_Hidden{jj},:)');
    end

    % Calculate total confusion.
    MLP.Net_Confusion{ii}{end} = zeros(1,test_hidden_col);
    MLP.Net_Confusion{ii}{end} = zeros(1,test_hidden_col);
    MLP.Net_Confusion{ii}{end}(train_col) = ...
        confusion(T(Train_Ind_All,:)', ...
        MLP.Net_Output_Ans{ii}(Train_Ind_All,:)');
    MLP.Net_Confusion{ii}{end}(test_col) = ...
        confusion(T(Test_Ind_All,:)', ...
        MLP.Net_Output_Ans{ii}(Test_Ind_All,:)');
    MLP.Net_Confusion{ii}{end}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible_All,:)', ...
        MLP.Net_Output_Ans{ii}(Test_Ind_Visible_All,:)');
    MLP.Net_Confusion{ii}{end}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden_All,:)', ...
        MLP.Net_Output_Ans{ii}(Test_Ind_Hidden_All,:)');

end

% Clean up cell.
clearvars ii jj

%% MLP Averaged Probability Committee Performance

% Storage.
MLP.Avg_Prob_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP.Avg_Prob_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP.Avg_Prob_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP.Avg_Prob_Committee_Output(Train_Ind{ii},:)');
    MLP.Avg_Prob_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP.Avg_Prob_Committee_Output(Test_Ind{ii},:)');
```

```
    MLP.Avg_Prob_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP.Avg_Prob_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP.Avg_Prob_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP.Avg_Prob_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP.Avg_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Avg_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Avg_Prob_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP.Avg_Prob_Committee_Output(Train_Ind_All,:)');
MLP.Avg_Prob_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP.Avg_Prob_Committee_Output(Test_Ind_All,:)');
MLP.Avg_Prob_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP.Avg_Prob_Committee_Output(Test_Ind_Visible_All,:)');
MLP.Avg_Prob_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP.Avg_Prob_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% MLP Averaged Answer Committee Performance

% Storage.
MLP.Avg_Ans_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP.Avg_Ans_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP.Avg_Ans_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP.Avg_Ans_Committee_Output(Train_Ind{ii},:)');
    MLP.Avg_Ans_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP.Avg_Ans_Committee_Output(Test_Ind{ii},:)');
    MLP.Avg_Ans_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP.Avg_Ans_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP.Avg_Ans_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP.Avg_Ans_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP.Avg_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Avg_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Avg_Ans_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP.Avg_Ans_Committee_Output(Train_Ind_All,:)');
MLP.Avg_Ans_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP.Avg_Ans_Committee_Output(Test_Ind_All,:)');
MLP.Avg_Ans_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP.Avg_Ans_Committee_Output(Test_Ind_Visible_All,:)');
MLP.Avg_Ans_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP.Avg_Ans_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% MLP Networked Probability Committee Performance

% Storage.
MLP.Net_Prob_Committee_Confusion = cell(1,size(T,2)+1);
```

```
% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP.Net_Prob_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP.Net_Prob_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP.Net_Prob_Committee_Output(Train_Ind{ii},:)');
    MLP.Net_Prob_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP.Net_Prob_Committee_Output(Test_Ind{ii},:)');
    MLP.Net_Prob_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP.Net_Prob_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP.Net_Prob_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP.Net_Prob_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP.Net_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Net_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Net_Prob_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP.Net_Prob_Committee_Output(Train_Ind_All,:)');
MLP.Net_Prob_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP.Net_Prob_Committee_Output(Test_Ind_All,:)');
MLP.Net_Prob_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP.Net_Prob_Committee_Output(Test_Ind_Visible_All,:)');
MLP.Net_Prob_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP.Net_Prob_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% MLP Networked Answer Committee Performance

% Storage.
MLP.Net_Ans_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP.Net_Ans_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP.Net_Ans_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP.Net_Ans_Committee_Output(Train_Ind{ii},:)');
    MLP.Net_Ans_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP.Net_Ans_Committee_Output(Test_Ind{ii},:)');
    MLP.Net_Ans_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP.Net_Ans_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP.Net_Ans_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP.Net_Ans_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP.Net_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Net_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP.Net_Ans_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP.Net_Ans_Committee_Output(Train_Ind_All,:)');
MLP.Net_Ans_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP.Net_Ans_Committee_Output(Test_Ind_All,:)');
MLP.Net_Ans_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP.Net_Ans_Committee_Output(Test_Ind_Visible_All,:)');
MLP.Net_Ans_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP.Net_Ans_Committee_Output(Test_Ind_Hidden_All,:)');
```

```
% Clean up cell.
clearvars ii

%% NN Performance

% Storage.
NN.Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    NN.Confusion{ii} = zeros(1,test_hidden_col);
    NN.Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', NN.Output(Train_Ind{ii},:)');
    NN.Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', NN.Output(Test_Ind{ii},:)');
    NN.Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', NN.Output(Test_Ind_Visible{ii},:)');
    NN.Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', NN.Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
NN.Confusion{end} = zeros(1,test_hidden_col);
NN.Confusion{end} = zeros(1,test_hidden_col);
NN.Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', NN.Output(Train_Ind_All,:)');
NN.Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', NN.Output(Test_Ind_All,:)');
NN.Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', NN.Output(Test_Ind_Visible_All,:)');
NN.Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', NN.Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% Small MLP Individual Network Performance

% Storage.
MLP_Small.Net_Confusion = cell(1,Num_Committee);

% Loop through all networks.
for ii = 1:Num_Committee

    % Storage.
    MLP_Small.Net_Confusion{ii} = cell(1,size(T,2)+1);

    % Calculate confusion for each output class.
    for jj = 1:size(T,2)
        MLP_Small.Net_Confusion{ii}{jj} = zeros(1,test_hidden_col);
        MLP_Small.Net_Confusion{ii}{jj}(train_col) = ...
            confusion(T(Train_Ind{jj},:)', ...
            MLP_Small.Net_Output_Ans{ii}(Train_Ind{jj},:)');
        MLP_Small.Net_Confusion{ii}{jj}(test_col) = ...
            confusion(T(Test_Ind{jj},:)', ...
            MLP_Small.Net_Output_Ans{ii}(Test_Ind{jj},:)');
        MLP_Small.Net_Confusion{ii}{jj}(test_visible_col) = ...
            confusion(T(Test_Ind_Visible{jj},:)', ...
            MLP_Small.Net_Output_Ans{ii}(Test_Ind_Visible{jj},:)');
        MLP_Small.Net_Confusion{ii}{jj}(test_hidden_col) = ...
            confusion(T(Test_Ind_Hidden{jj},:)', ...
            MLP_Small.Net_Output_Ans{ii}(Test_Ind_Hidden{jj},:)');
    end

    % Calculate total confusion.
    MLP_Small.Net_Confusion{ii}{end} = zeros(1,test_hidden_col);
    MLP_Small.Net_Confusion{ii}{end} = zeros(1,test_hidden_col);
    MLP_Small.Net_Confusion{ii}{end}(train_col) = ...
        confusion(T(Train_Ind_All,:)', ...
        MLP_Small.Net_Output_Ans{ii}(Train_Ind_All,:)');
    MLP_Small.Net_Confusion{ii}{end}(test_col) = ...
        confusion(T(Test_Ind_All,:)', ...
```

```
        MLP_Small.Net_Output_Ans{ii}(Test_Ind_All,:)');
    MLP_Small.Net_Confusion{ii}{end}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible_All,:)', ...
        MLP_Small.Net_Output_Ans{ii}(Test_Ind_Visible_All,:)');
    MLP_Small.Net_Confusion{ii}{end}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden_All,:)', ...
        MLP_Small.Net_Output_Ans{ii}(Test_Ind_Hidden_All,:)');

end

% Clean up cell.
clearvars ii jj

%% Small MLP Averaged Probability Committee Performance

% Storage.
MLP_Small.Avg_Prob_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP_Small.Avg_Prob_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP_Small.Avg_Prob_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP_Small.Avg_Prob_Committee_Output(Train_Ind{ii},:)');
    MLP_Small.Avg_Prob_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP_Small.Avg_Prob_Committee_Output(Test_Ind{ii},:)');
    MLP_Small.Avg_Prob_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP_Small.Avg_Prob_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP_Small.Avg_Prob_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP_Small.Avg_Prob_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP_Small.Avg_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Avg_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Avg_Prob_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP_Small.Avg_Prob_Committee_Output(Train_Ind_All,:)');
MLP_Small.Avg_Prob_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP_Small.Avg_Prob_Committee_Output(Test_Ind_All,:)');
MLP_Small.Avg_Prob_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP_Small.Avg_Prob_Committee_Output(Test_Ind_Visible_All,:)');
MLP_Small.Avg_Prob_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP_Small.Avg_Prob_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% Small MLP Averaged Answer Committee Performance

% Storage.
MLP_Small.Avg_Ans_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP_Small.Avg_Ans_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP_Small.Avg_Ans_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP_Small.Avg_Ans_Committee_Output(Train_Ind{ii},:)');
    MLP_Small.Avg_Ans_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP_Small.Avg_Ans_Committee_Output(Test_Ind{ii},:)');
    MLP_Small.Avg_Ans_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP_Small.Avg_Ans_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP_Small.Avg_Ans_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
```

```
        MLP_Small.Avg_Ans_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP_Small.Avg_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Avg_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Avg_Ans_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP_Small.Avg_Ans_Committee_Output(Train_Ind_All,:)');
MLP_Small.Avg_Ans_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP_Small.Avg_Ans_Committee_Output(Test_Ind_All,:)');
MLP_Small.Avg_Ans_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP_Small.Avg_Ans_Committee_Output(Test_Ind_Visible_All,:)');
MLP_Small.Avg_Ans_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP_Small.Avg_Ans_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% MLP Networked Probability Committee Performance

% Storage.
MLP_Small.Net_Prob_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP_Small.Net_Prob_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP_Small.Net_Prob_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
        MLP_Small.Net_Prob_Committee_Output(Train_Ind{ii},:)');
    MLP_Small.Net_Prob_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP_Small.Net_Prob_Committee_Output(Test_Ind{ii},:)');
    MLP_Small.Net_Prob_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP_Small.Net_Prob_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP_Small.Net_Prob_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP_Small.Net_Prob_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP_Small.Net_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Net_Prob_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Net_Prob_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP_Small.Net_Prob_Committee_Output(Train_Ind_All,:)');
MLP_Small.Net_Prob_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP_Small.Net_Prob_Committee_Output(Test_Ind_All,:)');
MLP_Small.Net_Prob_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP_Small.Net_Prob_Committee_Output(Test_Ind_Visible_All,:)');
MLP_Small.Net_Prob_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP_Small.Net_Prob_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii

%% MLP Networked Answer Committee Performance

% Storage.
MLP_Small.Net_Ans_Committee_Confusion = cell(1,size(T,2)+1);

% Calculate confusion for each output class.
for ii = 1:size(T,2)
    MLP_Small.Net_Ans_Committee_Confusion{ii} = zeros(1,test_hidden_col);
    MLP_Small.Net_Ans_Committee_Confusion{ii}(train_col) = ...
        confusion(T(Train_Ind{ii},:)', ...
```

```
        MLP_Small.Net_Ans_Committee_Output(Train_Ind{ii},:)');
    MLP_Small.Net_Ans_Committee_Confusion{ii}(test_col) = ...
        confusion(T(Test_Ind{ii},:)', ...
        MLP_Small.Net_Ans_Committee_Output(Test_Ind{ii},:)');
    MLP_Small.Net_Ans_Committee_Confusion{ii}(test_visible_col) = ...
        confusion(T(Test_Ind_Visible{ii},:)', ...
        MLP_Small.Net_Ans_Committee_Output(Test_Ind_Visible{ii},:)');
    MLP_Small.Net_Ans_Committee_Confusion{ii}(test_hidden_col) = ...
        confusion(T(Test_Ind_Hidden{ii},:)', ...
        MLP_Small.Net_Ans_Committee_Output(Test_Ind_Hidden{ii},:)');
end

% Calculate total confusion.
MLP_Small.Net_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Net_Ans_Committee_Confusion{end} = zeros(1,test_hidden_col);
MLP_Small.Net_Ans_Committee_Confusion{end}(train_col) = ...
    confusion(T(Train_Ind_All,:)', ...
    MLP_Small.Net_Ans_Committee_Output(Train_Ind_All,:)');
MLP_Small.Net_Ans_Committee_Confusion{end}(test_col) = ...
    confusion(T(Test_Ind_All,:)', ...
    MLP_Small.Net_Ans_Committee_Output(Test_Ind_All,:)');
MLP_Small.Net_Ans_Committee_Confusion{end}(test_visible_col) = ...
    confusion(T(Test_Ind_Visible_All,:)', ...
    MLP_Small.Net_Ans_Committee_Output(Test_Ind_Visible_All,:)');
MLP_Small.Net_Ans_Committee_Confusion{end}(test_hidden_col) = ...
    confusion(T(Test_Ind_Hidden_All,:)', ...
    MLP_Small.Net_Ans_Committee_Output(Test_Ind_Hidden_All,:)');

% Clean up cell.
clearvars ii
```

B.17 Classification_Aggregate.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Storage.
% Rows: Training, Testing, Visible Testing, Hidden Testing
% Columns: Tacks, Gybes, Roundings, Straights, All
Aggregate = cell(4,5);
for ii = 1:size(Aggregate,1)
    for jj = 1:size(Aggregate,2)
        Aggregate{ii,jj} = cell(8,11);
    end
end

% Time storage.
% Rows: Training Time, CPU Training Time, Output Time, CPU Output Time
Time = cell(1,4);
for ii = 1:length(Time)
    Time{ii} = cell(8,11);
end

% Loop through combinations.
for Combination_Num = 1:8

    % Loop through folds.
    for Fold_Num = 1:CV_Folds

        clearvars -except Test_Num Combination_Num Fold_Num CV_Folds ...
            Aggregate Time

        % Construct combination segment of filename.
        combination = Combination_Num_To_String(Combination_Num);

        % File to load.
        filename = ['..\Data\Active\Test_', ...
            num2str(Test_Num), '\Part_1\Fold_',...
            num2str(Fold_Num), '\Classification', combination];

        % Load.
        load(filename);

        % Aggregate.
        % Time.
        % Individual MLP Networks.
        col = 1;
        Time{1}{Combination_Num, col} = ...
            horzcat(Time{1}{Combination_Num, col}, ...
            MLP.Net_Train_Time/Num_Committee);
        Time{2}{Combination_Num, col} = ...
            horzcat(Time{2}{Combination_Num, col}, ...
            MLP.Net_Train_CPU_Time/Num_Committee);
        Time{3}{Combination_Num, col} = ...
            horzcat(Time{3}{Combination_Num, col}, ...
            MLP.Net_Output_Time/Num_Committee);
        Time{4}{Combination_Num, col} = ...
            horzcat(Time{4}{Combination_Num, col}, ...
            MLP.Net_Output_CPU_Time/Num_Committee);

        % MLP Averaged Probability Committee.
        col = 2;
        Time{1}{Combination_Num, col} = ...
            horzcat(Time{1}{Combination_Num, col}, MLP.Net_Train_Time);
        Time{2}{Combination_Num, col} = ...
            horzcat(Time{2}{Combination_Num, col}, MLP.Net_Train_CPU_Time);
        Time{3}{Combination_Num, col} = ...
            horzcat(Time{3}{Combination_Num, col}, ...
            MLP.Avg_Prob_Commitee_Output_Time);
        Time{4}{Combination_Num, col} = ...
            horzcat(Time{4}{Combination_Num, col}, ...
            MLP.Avg_Prob_Commitee_Output_CPU_Time);

        % MLP Averaged Answer Committee.
        col = 3;
```

```
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, MLP.Net_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, MLP.Net_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP.Avg_Ans_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP.Avg_Ans_Commitee_Output_CPU_Time);

% MLP Networked Probability Committee.
col = 4;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP.Net_Prob_Committee_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP.Net_Prob_Committee_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP.Net_Prob_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP.Net_Prob_Commitee_Output_CPU_Time);

% MLP Networked Answer Committee.
col = 5;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP.Net_Ans_Committee_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP.Net_Ans_Committee_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP.Net_Ans_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP.Net_Ans_Commitee_Output_CPU_Time);

% NN
col = 6;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, 0);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, 0);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, NN.Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, NN.Output_CPU_Time);

% Individual Small MLP Networks.
col = 7;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP_Small.Net_Train_Time/Num_Committee);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP_Small.Net_Train_CPU_Time/Num_Committee);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP_Small.Net_Output_Time/Num_Committee);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP_Small.Net_Output_CPU_Time/Num_Committee);

% Small MLP Averaged Probability Committee.
col = 8;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP_Small.Net_Train_Time);
Time{2}{Combination_Num, col} = ...
```

```
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP_Small.Net_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP_Small.Avg_Prob_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP_Small.Avg_Prob_Commitee_Output_CPU_Time);

% Small MLP Averaged Answer Committee.
col = 9;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP_Small.Net_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP_Small.Net_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP_Small.Avg_Ans_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP_Small.Avg_Ans_Commitee_Output_CPU_Time);

% Small MLP Networked Probability Committee.
col = 10;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP_Small.Net_Prob_Committee_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP_Small.Net_Prob_Committee_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP_Small.Net_Prob_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP_Small.Net_Prob_Commitee_Output_CPU_Time);

% Small MLP Networked Answer Committee.
col = 11;
Time{1}{Combination_Num, col} = ...
    horzcat(Time{1}{Combination_Num, col}, ...
    MLP_Small.Net_Ans_Committee_Train_Time);
Time{2}{Combination_Num, col} = ...
    horzcat(Time{2}{Combination_Num, col}, ...
    MLP_Small.Net_Ans_Committee_Train_CPU_Time);
Time{3}{Combination_Num, col} = ...
    horzcat(Time{3}{Combination_Num, col}, ...
    MLP_Small.Net_Ans_Commitee_Output_Time);
Time{4}{Combination_Num, col} = ...
    horzcat(Time{4}{Combination_Num, col}, ...
    MLP_Small.Net_Ans_Commitee_Output_CPU_Time);

% Accuracy.
for ii = 1:size(Aggregate,1)
    for jj = 1:size(Aggregate,2)

        % Individual MLP Networks.
        col = 1;
        for kk = 1:Num_Committee
            Aggregate{ii,jj}{Combination_Num, col} = ...
                horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                1-MLP.Net_Confusion{kk}{jj}(ii));
        end

        % MLP Averaged Probability Committee
        col = 2;
        Aggregate{ii,jj}{Combination_Num, col} = ...
            horzcat(Aggregate{ii,jj}{Combination_Num, col},...
            1-MLP.Avg_Prob_Committee_Confusion{jj}(ii));

        % MLP Averaged Answer Committee
```

```
                col = 3;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP.Avg_Ans_Committee_Confusion{jj}(ii));

                % MLP Networked Probability Committee
                col = 4;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP.Net_Prob_Committee_Confusion{jj}(ii));

                % MLP Networked Answer Committee
                col = 5;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP.Net_Ans_Committee_Confusion{jj}(ii));

                % NN
                col = 6;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-NN.Confusion{jj}(ii));

                % Individual MLP Networks.
                col = 7;
                for kk = 1:Num_Committee
                    Aggregate{ii,jj}{Combination_Num, col} = ...
                        horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                        1-MLP_Small.Net_Confusion{kk}{jj}(ii));
                end

                % MLP Averaged Probability Committee
                col = 8;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP_Small.Avg_Prob_Committee_Confusion{jj}(ii));

                % MLP Averaged Answer Committee
                col = 9;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP_Small.Avg_Ans_Committee_Confusion{jj}(ii));

                % MLP Networked Probability Committee
                col = 10;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP_Small.Net_Prob_Committee_Confusion{jj}(ii));

                % MLP Networked Answer Committee
                col = 11;
                Aggregate{ii,jj}{Combination_Num, col} = ...
                    horzcat(Aggregate{ii,jj}{Combination_Num, col},...
                    1-MLP_Small.Net_Ans_Committee_Confusion{jj}(ii));

            end
        end

    end

end

% Clean up.
clearvars -except Test_Num CV_Folds Aggregate Time

% Filename of resultant file.
filename = ['..\Data\Active\Test_', ...
    num2str(Test_Num), '\Part_1\Classification_Aggregate'];

% Save
save(filename);
```

B.18 Classification_Rank.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Filename to load.
filename = ['..\Data\Active\Test_', ...
    num2str(Test_Num), '\Part_1\Classification_Aggregate'];

% Load.
load(filename);

% Storage.
Rank = cell(size(Aggregate));
Confidence = cell(size(Aggregate));

% Loop through aggregate.
for ii = 1:size(Aggregate,1)
    parfor jj = 1:size(Aggregate,2)

        % Test #1 and straight events have no hidden testing data.
        if (ii == 3 || ii == 4) && (jj == 4 || Test_Num == 1)
            continue;
        end

        % Storage.
        Rank{ii,jj} = zeros(size(Aggregate{ii,jj}));
        Confidence{ii,jj} = zeros(size(Aggregate{ii,jj}));
        mu = zeros(size(Aggregate{ii,jj}));
        sigma = zeros(size(Aggregate{ii,jj}));
        n = zeros(size(Aggregate{ii,jj}));

        % Calculate quantities.
        for kk = 1:size(Aggregate{ii,jj},1)
            for ll = 1:size(Aggregate{ii,jj},2)
                mu(kk,ll) = mean(Aggregate{ii,jj}{kk,ll});
                sigma(kk,ll) = std(Aggregate{ii,jj}{kk,ll});
                n(kk,ll) = length(Aggregate{ii,jj}{kk,ll});
            end
        end

        % Storage.
        lvm_record = cell(100,1);
        uvm_record = cell(100,1);

        % Initialize mask.
        mask = ones(size(mu));

        % Initialize current rank.
        cur_rank = 1;

        % Loop until all ranks have been found.
        while 1

            % Find the current target.
            mu_masked = mu .* mask;
            target = (mu == max(mu_masked(:)));

            % Record rank.
            Rank{ii,jj}(target) = cur_rank;
            cur_rank = cur_rank + 1;

            % Check if last rank.
            if isequal(mask, target)
                Confidence{ii,jj}(target) = 1;
                break;
            end

            % Find confidence.
            for kk = 1:100

                % Check if record exists.
                if isempty(lvm_record{kk}) || isempty(uvm_record{kk})
```

```
                    % Record doesn't exist, add to it.
                    [lvm_record{kk}, uvm_record{kk}] = ...
                        TINV_Folded_Matrix(1-(kk*0.01), mu, sigma, n, mask);

                end

                % Apply mask.
                lvm = lvm_record{kk} .* mask;
                uvm = uvm_record{kk} .* mask;

                % Determine result.
                uv = max(uvm(:));
                t = max(lvm(uvm == uv));
                result = uvm >= t;

                % Check if target.
                if isequal(result, target)
                    Confidence{ii,jj}(target) = 1-(kk*0.01);
                    break;
                end

            end

            % Update mask.
            mask(target) = 0;

        end

    end
end

% Clean up.
clearvars -except Test_Num CV_Folds Rank Confidence

% Filename of resultant file.
filename = ['..\Data\Active\Test_', ...
    num2str(Test_Num), '\Part_1\Classification_Rank'];

% Save
save(filename);
```

B.19 Classification_Latex.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Filename to load.
filename = ['..\Data\Active\Test_', ...
    num2str(Test_Num), '\Part_1\Classification_Aggregate'];

% Load.
load(filename);

% Filename to load.
filename = ['..\Data\Active\Test_', ...
    num2str(Test_Num), '\Part_1\Classification_Rank'];

% Load.
load(filename);

% Labels.
data_label_text = {'Training','Testing','Visible Testing','Hidden Testing'};
data_label_file = {'Training','Testing','Visible_Testing','Hidden_Testing'};
class_label_text = {', Tacks Only',', Gybes Only',', Mark Roundings Only',', Straight Sailing Only',''};
class_label_file = {'_Tacks','_Gybes','_Mark_Roundings','_Straight_Sailing',''};
time_label_text = {'Training', 'Training CPU', 'Output', 'Output CPU'};
time_label_file = {'Training', 'Training_CPU', 'Output', 'Output_CPU'};

% Column headers.
result_header = ...
    ['\\multicolumn{1}{c}{\\multirow{2}{*}',...
    '{\\begin{tabular}[c]{c}Pre-Processing\\\\Combination\\end{tabular}}}',...
    ' & \\multicolumn{6}{c}{Classification Method}\\\\\\cmidrule{2-7}\n',...
    '& \\multicolumn{1}{c}{1} & \\multicolumn{1}{c}{2} & \\multicolumn{1}{c}{3}',...
    ' & \\multicolumn{1}{c}{4} & \\multicolumn{1}{c}{5} & \\multicolumn{1}{c}{6}\\\\\n'];
small_mlp_result_header = ...
    ['\\multicolumn{1}{c}{\\multirow{2}{*}',...
    '{\\begin{tabular}[c]{c}Pre-Processing\\\\Combination\\end{tabular}}}',...
    ' & \\multicolumn{5}{c}{Classification Method}\\\\\\cmidrule{2-6}\n',...
    '& \\multicolumn{1}{c}{7} & \\multicolumn{1}{c}{8} & \\multicolumn{1}{c}{9}',...
    ' & \\multicolumn{1}{c}{10} & \\multicolumn{1}{c}{11}\\\\\n'];

% Create master files.
master_a_fid = fopen(['..\Tables\Test_', num2str(Test_Num), '\Master_All.tex'], 'w');
master_e_fid = fopen(['..\Tables\Test_', num2str(Test_Num), '\Master_Events.tex'], 'w');
master_c_fid = fopen(['..\Tables\Test_', num2str(Test_Num), '\Master_CPU_Time.tex'], 'w');
master_t_fid = fopen(['..\Tables\Test_', num2str(Test_Num), '\Master_Time.tex'], 'w');

% Loop through aggregate.
for ii = 1:size(Aggregate,1)
    for jj = 1:size(Aggregate,2)

        % Test #1 and straight events have no hidden testing data.
        if (ii == 3 || ii == 4) && (jj == 4 || Test_Num == 1)
            continue;
        end

        % Correct rank.
        Rank{ii,jj}(Rank{ii,jj} == 0) = max(Rank{ii,jj}(:))+1;

        % Storage.
        mu = zeros(size(Aggregate{ii,jj}));
        sigma = zeros(size(Aggregate{ii,jj}));
        n = zeros(size(Aggregate{ii,jj}));

        % Calculate quantities.
        for kk = 1:size(Aggregate{ii,jj},1)
            for ll = 1:size(Aggregate{ii,jj},2)
                mu(kk,ll) = mean(Aggregate{ii,jj}{kk,ll});
                sigma(kk,ll) = std(Aggregate{ii,jj}{kk,ll});
                n(kk,ll) = length(Aggregate{ii,jj}{kk,ll});
            end
        end

        % Calculate orders.
```

```
mu_order = floor(log(abs(mu))/log(10));
mu_order(isinf(mu_order)) = 0;
sigma_order = floor(log(abs(sigma*1e3))/log(10));
sigma_order(isinf(sigma_order)) = 0;

% Calculate maximum orders.
mu_order_max = max(mu_order);
sigma_order_max = max(sigma_order);

% Calculate minimum orders.
mu_order_min = min(mu_order);
sigma_order_min = min(sigma_order);

% Open mean file.
mu_filename = ['..\Tables\Test_', num2str(Test_Num), ...
    '\', data_label_file{ii}, class_label_file{jj}, '_Mean.tex'];
fid = fopen(mu_filename, 'w');

% Write first header.
fprintf(fid,'\\begin{table}\n');
fprintf(fid,'\\centering\n');
fprintf(fid,['\\caption{Mean Accuracy for ', data_label_text{ii}, ...
    ' Data, Test ', num2str(Test_Num), class_label_text{jj}, '}\n']);
fprintf(fid,['\\label{tab:Test_', num2str(Test_Num), ...
    class_label_file{jj}, '_', data_label_file{ii}, '_Mean}\n']);
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 1:6
    if min(mu(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([mu_order_max(kk)+1,1]), ...
            max([-mu_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\toprule\n');
fprintf(fid, result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(mu,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 1:5
        fprintf(fid, [FORMAT_Sig_Fig(mu(kk,ll),4), ' & ']);
    end

    % Write last result.
    fprintf(fid, [FORMAT_Sig_Fig(mu(kk,6),4), ' \\\\\n']);

end

% Write second header.
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 7:11
    if min(mu(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([mu_order_max(kk)+1,1]), ...
            max([-mu_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\midrule\n');
fprintf(fid, small_mlp_result_header);
fprintf(fid,'\\midrule\n');
```

```
% Write data.
for kk = 1:size(mu,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 7:10
        fprintf(fid, [FORMAT_Sig_Fig(mu(kk,ll),4), ' & ']);
    end

    % Write last result.
    fprintf(fid, [FORMAT_Sig_Fig(mu(kk,11),4), ' \\\\\n']);

end

% Write footer.
fprintf(fid,'\\bottomrule\n');
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\end{table}');

% Close file.
fclose(fid);

% Open standard deviation file.
sigma_filename = ['..\Tables\Test_', num2str(Test_Num), ...
    '\', data_label_file{ii}, class_label_file{jj}, '_Std.tex'];
fid = fopen(sigma_filename, 'w');

% Write first header.
fprintf(fid,'\\begin{table}\n');
fprintf(fid,'\\centering\n');
fprintf(fid,['\\caption{Standard Deviation of Accuracy for ', data_label_text{ii}, ...
    ' Data, Test ', num2str(Test_Num), class_label_text{jj}, ' $[10^3]$}\n']);
fprintf(fid,['\\label{tab:Test_', num2str(Test_Num), ...
    class_label_file{jj}, '_', data_label_file{ii}, '_Std}\n']);
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 1:6
    if min(sigma(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([sigma_order_max(kk)+1,1]), ...
            max([-sigma_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\toprule\n');
fprintf(fid, result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(sigma,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 1:5
        fprintf(fid, [FORMAT_Sig_Fig(sigma(kk,ll)*1e3,4), ' & ']);
    end

    % Write last result.
    fprintf(fid, [FORMAT_Sig_Fig(sigma(kk,6)*1e3,4), ' \\\\\n']);

end

% Write second header.
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 7:11
    if min(sigma(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
```

```
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([sigma_order_max(kk)+1,1]), ...
            max([-sigma_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\midrule\n');
fprintf(fid, small_mlp_result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(sigma,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 7:10
        fprintf(fid, [FORMAT_Sig_Fig(sigma(kk,ll)*1e3,4), ' & ']);
    end

    % Write last result.
    fprintf(fid, [FORMAT_Sig_Fig(sigma(kk,11)*1e3,4), ' \\\\\n']);

end

% Write footer.
fprintf(fid,'\\bottomrule\n');
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\end{table}');

% Close file.
fclose(fid);

% Open rank file.
rank_filename = ['..\Tables\Test_', num2str(Test_Num), ...
    '\', data_label_file{ii}, class_label_file{jj}, '_Rank.tex'];
fid = fopen(rank_filename, 'w');

% Write first header.
fprintf(fid,'\\begin{table}\n');
fprintf(fid,'\\centering\n');
fprintf(fid,'\\footnotesize\n');
fprintf(fid,['\\caption{Accuracy Ranking for ', data_label_text{ii}, ...
    ' Data, Test ', num2str(Test_Num), class_label_text{jj}, ' (Confidence)}\n']);
fprintf(fid,['\\label{tab:Test_', num2str(Test_Num), ...
    class_label_file{jj}, '_', data_label_file{ii}, '_Rank}\n']);
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} crrrrrr}\n');
fprintf(fid,'\\toprule\n');
fprintf(fid, result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(Rank{ii,jj},1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 1:5
        fprintf(fid, '%u (%4.2f) & ', Rank{ii,jj}(kk,ll), Confidence{ii,jj}(kk,ll));
    end

    % Write last result.
    fprintf(fid, '%u (%4.2f) \\\\\n', Rank{ii,jj}(kk,6), Confidence{ii,jj}(kk,6));

end

% Write second header.
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} crrrrr}\n');
fprintf(fid,'\\midrule\n');
```

```
        fprintf(fid, small_mlp_result_header);
        fprintf(fid,'\\midrule\n');

        % Write data.
        for kk = 1:size(Rank{ii,jj},1)

            % Write pre-processing combination.
            fprintf(fid, '%u & ', kk);

            % Write results.
            for ll = 7:10
                fprintf(fid, '%u (%4.2f) & ', Rank{ii,jj}(kk,ll), Confidence{ii,jj}(kk,ll));
            end

            % Write last result.
            fprintf(fid, '%u (%4.2f) \\\\\n', Rank{ii,jj}(kk,11), Confidence{ii,jj}(kk,11));

        end

        % Write footer.
        fprintf(fid,'\\bottomrule\n');
        fprintf(fid,'\\end{tabular*}\n');
        fprintf(fid,'\\end{table}');

        % Close file.
        fclose(fid);

        if jj == 5

            % Write to all master.
            fprintf(master_a_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Mean}\n']);
            fprintf(master_a_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Std}\n']);
            fprintf(master_a_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Rank}\n']);

        else

            % Write to event master.
            fprintf(master_e_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Mean}\n']);
            fprintf(master_e_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Std}\n']);
            fprintf(master_e_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
                '/', data_label_file{ii}, class_label_file{jj}, '_Rank}\n']);

        end

    end
end

% Loop through time.
for ii = 1:length(Time)

    % Storage.
    mu = zeros(size(Time{ii}));

    % Calculate quantities.
    for kk = 1:size(Time{ii},1)
        for ll = 1:size(Time{ii},2)
            mu(kk,ll) = mean(Time{ii}{kk,ll});
        end
    end

    % Calculate order.
    mu_order = floor(log(abs(mu))/log(10));
    mu_order(isinf(mu_order)) = 0;

    % Calculate maximum order.
    mu_order_max = max(mu_order);

    % Calculate minimum order.
```

```
mu_order_min = min(mu_order);

% Open file.
filename = ['..\Tables\Test_', num2str(Test_Num), ...
    '\', time_label_file{ii}, '_Time.tex'];
fid = fopen(filename, 'w');

% Write header.
fprintf(fid,'\\begin{table}\n');
fprintf(fid,'\\centering\n');
fprintf(fid,['\\caption{Test ', num2str(Test_Num), ...
    ', Mean ', time_label_text{ii}, ' Time [s]}\n']);
fprintf(fid,['\\label{tab:Test_', num2str(Test_Num), ...
    '_', time_label_file{ii}, '_Time}\n']);
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 1:6
    if min(mu(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([mu_order_max(kk)+1,1]), ...
            max([-mu_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\toprule\n');
fprintf(fid, result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(mu,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 1:5
        fprintf(fid, [FORMAT_Sig_Fig(mu(kk,ll),4), ' & ']);
    end

    % Write last result.
    fprintf(fid, [FORMAT_Sig_Fig(mu(kk,6),4), ' \\\\\n']);

end

% Write second header.
fprintf(fid,'\\end{tabular*}\n');
fprintf(fid,'\\begin{tabular*}{\\textwidth}{@{\\extracolsep{\\fill}} c');
for kk = 7:11
    if min(mu(:,kk) == 0) == 1
        fprintf(fid, 'S[table-format=1.0]');
    else
        fprintf(fid, 'S[table-format=%u.%u]', ...
            max([mu_order_max(kk)+1,1]), ...
            max([-mu_order_min(kk)+3,0]));
    end
end
fprintf(fid,'}\n');
fprintf(fid,'\\midrule\n');
fprintf(fid, small_mlp_result_header);
fprintf(fid,'\\midrule\n');

% Write data.
for kk = 1:size(mu,1)

    % Write pre-processing combination.
    fprintf(fid, '%u & ', kk);

    % Write results.
    for ll = 7:10
        fprintf(fid, [FORMAT_Sig_Fig(mu(kk,ll),4), ' & ']);
    end
```

```
        % Write last result.
        fprintf(fid, [FORMAT_Sig_Fig(mu(kk,11),4), ' \\\\\n']);

    end

    % Write footer.
    fprintf(fid,'\\bottomrule\n');
    fprintf(fid,'\\end{tabular*}\n');
    fprintf(fid,'\\end{table}');

    % Close file.
    fclose(fid);

    % Write to master.
    if ii == 2 || ii == 4
        fprintf(master_c_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
            '/', time_label_file{ii}, '_Time}\n']);
    else
        fprintf(master_t_fid,['\\input{./Tables/Test_', num2str(Test_Num), ...
            '/', time_label_file{ii}, '_Time}\n']);
    end

end

% Close masters.
fclose(master_a_fid);
fclose(master_e_fid);
fclose(master_c_fid);
fclose(master_t_fid);
```

B.20 TINV_Folded_Matrix.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
function [lvm, uvm] = TINV_Folded_Matrix(p, mu, sigma, n, mask)
% Calculates a confidence interval on matricies of quantites.
% 'p' - Confidence interval. For example, 'p = 0.95' uses a 95%
% confidence interval.
% 'mu' - An 'm' by 'n' matrix of means.
% 'sigma' - An 'm' by 'n' matrix of standard deviations.
% 'n' - An 'm' by 'n' matrix of sample sizes.
% 'mask' - An 'm' by 'n' matrix. Only locations where the mask is non-zero
% will be evaluated.
% 'lvm' - An 'm' by 'n' matrix of lower confidence bounds.
% 'uvm' - An 'm' by 'n' matrix of upper confidence bounds.

% Max indicies.
ii_max = size(mu,1);
jj_max = size(mu,2);

% Storage.
lvm = zeros(ii_max, jj_max);
uvm = zeros(ii_max, jj_max);

% Loop.
parfor ii = 1:ii_max
    for jj = 1:jj_max

        % Mask.
        if ~mask(ii,jj)
            continue;
        end

        % Calculate confidence interval.
        [lvm(ii,jj),uvm(ii,jj)] = ...
            TINV_Folded(p,n(ii,jj),mu(ii,jj),sigma(ii,jj),0,1);

    end
end

end
```

B.21 TINV_Folded.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
function [lv,uv] = TINV_Folded(p,n,mu,sigma,lb,ub)
% Calculates a confidence interval on the mean using the Students-t
% distribution, with folding given bounds.
% 'p' - Confidence interval desired. For example, 'p = 0.95' calculates a 95%
% confidence interval.
% 'n' - Number of samples.
% 'mu' - Sample mean.
% 'sigma' - Sample standard deviation.
% 'lb' - Lower bound for folding.
% 'ub' - Upper bound for folding.

if isnan(mu) || isnan(sigma)
    lv = NaN;
    uv = NaN;
    return
end

if sigma <= 0
    lv = mu;
    uv = mu;
    return
end

% Calculate t-value at bounds.
tvlb = (lb-mu)*sqrt(n)/sigma;
tvub = (ub-mu)*sqrt(n)/sigma;

% Fit unity probability between the bounds.
% Options may have to be changed if the sample mean is outside of the
% bounds by a large margin.
opt = optimset('Display', 'off', 'TolFun', 1e-12, 'MaxFunEvals', 1000);
factor = lsqnonlin(@(y)(quad(@(x)tpdf(x,n-1)*y,tvlb,tvub)-1),1,1,[],opt);

% Calculate interval.
tvu = lsqnonlin(@(y)(quad(@(x)tpdf(x,n-1)*factor,0,y)-p/2),0,0,[],opt);
tvl = lsqnonlin(@(y)(quad(@(x)tpdf(x,n-1)*factor,y,0)-p/2),0,[],0,opt);
if tvu > tvub
    tvu = tvub;
    tvl = lsqnonlin(@(y)(quad(@(x)tpdf(x,n-1)*factor,y,tvub)-p),0,[],[],opt);
elseif tvl < tvlb
    tvl = tvlb;
    tvu = lsqnonlin(@(y)(quad(@(x)tpdf(x,n-1)*factor,tvlb,y)-p),0,[],[],opt);
end
lv = mu + tvl*sigma/sqrt(n);
uv = mu + tvu*sigma/sqrt(n);

end
```

B.22 FORMAT_Sig_Fig.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
function result = FORMAT_Sig_Fig(num, sig_fig)
% Formats a number into a string with a given number of significant
% figures.
% num - The number to format.
% sig_fig - The number of significant figures.
% result - The output string.

% Handle zero.
if num == 0
    result = '0';
    return;
end

% Format.
result = sprintf(['%#.', num2str(sig_fig), 'g'], num);

% Remove trailing decimal point.
if result(end) == '.'
    result(end) = [];
end

end
```

B.23 Classification_Rank_Demo.m

```
% Copyright (c) 2013 Ryan Sammon.
% All rights reserved.
% Demonstration of ranking algorithm.
% Input values.
mu = [0.8960,0.7300,0.6000];
sigma = [0.05683,0.01581,0.5477];
n = [5,5,5];

% Record.
csv_matrix = zeros(100,6);

% Calculate confidence intervals at every 1% probability.
for ii = 1:3
    for jj = 1:100
        [csv_matrix(jj,(ii-1)*2+1),csv_matrix(jj,(ii-1)*2+2)] = ...
            TINV_Folded(1-0.01*jj,n(ii),mu(ii),sigma(ii),0,1);
    end
end

% Write the file.
csvwrite('..\Figures\Classification_Rank_Demo.csv', csv_matrix);
```

C BlackBerry Playbook Code

C.1 main.c

```
/*
 * Copyright (c) 2013 Ryan Sammon.
 * All rights reserved.
 */

// Library includes.
#include <bps/sensor.h>
#include <bps/geolocation.h>
#include <bps/bps.h>
#include <bps/dialog.h>
#include <bps/navigator.h>
#include <screen/screen.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

// Dialog helper.
#include "dialogutil.h"

// Defines for converting time values.
#define MILLION  1000000L;
#define THOUSAND 1000L;

// Defines for buffer size.
#define MSG_SIZE 2048
#define ITEM_SIZE 128

// Define for file close and reopen.
#define FLUSH_INTERVAL 10000

// Define for writing period.
#define WRITE_PERIOD 20000

// Target sensor period in microseconds.
static const int SENSOR_RATE = 5000;

// Main.
int main(int argc, char *argv[])
{
// Flag to exit the program.
bool exit = false;

// Flag to control screen updates.
bool update_screen = true;

// Flags to track sensor data reception.
bool gps_received = false;
bool accelerometer_received = false;
bool magnetometer_received = false;
bool gyroscope_received = false;
bool azimuth_pitch_roll_received = false;
bool linear_accel_received = false;
bool rotation_vector_received = false;
bool rotation_matrix_received = false;

// Variable to store the number of data points recorded.
unsigned  int i_count = 0;

// Variables to store file related information.
FILE* fd;
uint16_t f_num;
char f_num_str[6];
char f_name[ITEM_SIZE];

// Variables to store the current event.
bps_event_t *event = NULL;
unsigned int event_code = 0;
```

```
// Variables to store the start, end, and interval time for each iteration.
struct timespec start_t, end_t;
double interval_t = 0;

// Variables to store sensor data.
double gps_la, gps_lo, gps_ac, gps_h, gps_s; // GPS readings.
long long gps_t; // GPS time.
float a_x, a_y, a_z; // Accelerometer readings.
float m_x, m_y, m_z; // Magnetometer readings.
float g_x, g_y, g_z; // Gyroscope readings.
float apr_a, apr_p, apr_r; // Azimuth, pitch, and roll readings.
float la_x, la_y, la_z; // Linear acceleration readings.
sensor_rotation_vector_t rv; // Rotation vector reading.
sensor_rotation_matrix_t rm; // Rotation matrix reading.

// Flags to track GPS validity.
bool gps_h_valid = false;
bool gps_s_valid = false;
bool gps_t_valid = false;

// Initialize platform.
bps_initialize();

// Setup the screen.
if (setup_screen() != EXIT_SUCCESS) {
printf("Unable to set up the screen. Exiting.");
return 0;
}

// Display the dialog.
create_dialog();

// Flags to track sensor support.
bool accelerometer_supported = sensor_is_supported(SENSOR_TYPE_ACCELEROMETER);
bool magnetometer_supported = sensor_is_supported(SENSOR_TYPE_MAGNETOMETER);
bool gyroscope_supported = sensor_is_supported(SENSOR_TYPE_GYROSCOPE);
bool azimuth_pitch_roll_supported = sensor_is_supported(SENSOR_TYPE_AZIMUTH_PITCH_ROLL);
bool linear_accel_supported = sensor_is_supported(SENSOR_TYPE_LINEAR_ACCEL);
bool rotation_vector_supported = sensor_is_supported(SENSOR_TYPE_ROTATION_VECTOR);
bool rotation_matrix_supported = sensor_is_supported(SENSOR_TYPE_ROTATION_MATRIX);

// Request core events.
navigator_request_events(0);
dialog_request_events(0);

// Set up the GPS and request events.
geolocation_set_period(1);
geolocation_request_events(0);

// Set up the supported sensors and request events.
if (accelerometer_supported)
{
sensor_set_rate(SENSOR_TYPE_ACCELEROMETER, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_ACCELEROMETER, true);
sensor_request_events(SENSOR_TYPE_ACCELEROMETER);
}
if (magnetometer_supported)
{
sensor_set_rate(SENSOR_TYPE_MAGNETOMETER, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_MAGNETOMETER, true);
sensor_request_events(SENSOR_TYPE_MAGNETOMETER);
}
if (gyroscope_supported)
{
sensor_set_rate(SENSOR_TYPE_GYROSCOPE, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_GYROSCOPE, true);
sensor_request_events(SENSOR_TYPE_GYROSCOPE);
}
if (azimuth_pitch_roll_supported)
{
sensor_set_rate(SENSOR_TYPE_AZIMUTH_PITCH_ROLL, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_AZIMUTH_PITCH_ROLL, true);
sensor_request_events(SENSOR_TYPE_AZIMUTH_PITCH_ROLL);
```

```
}
if (linear_accel_supported)
{
sensor_set_rate(SENSOR_TYPE_LINEAR_ACCEL, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_LINEAR_ACCEL, true);
sensor_request_events(SENSOR_TYPE_LINEAR_ACCEL);
}
if (rotation_vector_supported)
{
sensor_set_rate(SENSOR_TYPE_ROTATION_VECTOR, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_ROTATION_VECTOR, true);
sensor_request_events(SENSOR_TYPE_ROTATION_VECTOR);
}
if (rotation_matrix_supported)
{
sensor_set_rate(SENSOR_TYPE_ROTATION_MATRIX, SENSOR_RATE);
sensor_set_skip_duplicates(SENSOR_TYPE_ROTATION_MATRIX, true);
sensor_request_events(SENSOR_TYPE_ROTATION_MATRIX);
}

// Open the next available file for writing.
// Loop through file numbers.
for (f_num = 1; f_num <= UINT16_MAX; f_num++)
{
// Construct the filename.
itoa(f_num,f_num_str,10);
f_name[0]='\0';
strcat(f_name,"shared/documents/SailSmartData_");
strcat(f_name,f_num_str);
strcat(f_name,".csv");

// Attempt to open the file for reading.
fd = fopen(f_name, "r");

// Check if it opened.
if(fd)
{
// File exists, close it.
fclose(fd);
}
else
{
// File does not exist, break.
break;
}
}

// Open the file for writing.
fd = fopen(f_name, "w");

// Set the start time.
clock_gettime(CLOCK_REALTIME, &start_t);

// Processing loop.
while (!exit)
{
// Get the next event in the queue.
event = NULL;
bps_get_event(&event, 0);

// Check if there is an event.
if (event)
{
// Yes, determine its domain.
if (bps_event_get_domain(event) == navigator_get_domain())
{
// Its a navigator event.
// Get the event code.
event_code = bps_event_get_code(event);

// Determine its code.
if (event_code == NAVIGATOR_EXIT)
{
```

```
// Event indicating a request to close the application.
// Set the exit flag.
exit = true;
}
else if (event_code == NAVIGATOR_WINDOW_STATE)
{
// Its a window state event.
if (navigator_event_get_window_state(event) == NAVIGATOR_WINDOW_INVISIBLE)
{
// The window is invisible, don't update the screen.
update_screen = false;
}
else
{
// The window is visible, update the screen.
update_screen = true;
}
}
// All other codes are ignored.
}
else if (bps_event_get_domain(event) == geolocation_get_domain())
{
// Its a GPS event.
// Get the event code.
event_code = bps_event_get_code(event);

// Determine its code.
if (event_code == GEOLOCATION_INFO)
{
// GPS data event.
// Extract all of the event information.
gps_received = true;
gps_la = geolocation_event_get_latitude(event);
gps_lo = geolocation_event_get_longitude(event);
gps_ac = geolocation_event_get_accuracy(event);
gps_h = geolocation_event_get_heading(event);
gps_h_valid = geolocation_event_is_heading_valid(event);
gps_s = geolocation_event_get_speed(event);
gps_s_valid = geolocation_event_is_speed_valid(event);
gps_t = geolocation_event_get_utc_time(event)/(long long)THOUSAND;
gps_t_valid = geolocation_event_is_utc_time_valid(event);
}
// All other codes are ignored.
}
else if (bps_event_get_domain(event) == sensor_get_domain())
{
// Its a sensor event.
// Get the event code.
event_code = bps_event_get_code(event);

// Determine its code.
if (event_code == SENSOR_ACCELEROMETER_READING)
{
// Accelerometer data event.
// Extract all of the event information.
accelerometer_received = true;
sensor_event_get_xyz(event, &a_x, &a_y, &a_z);
}
else if (event_code == SENSOR_MAGNETOMETER_READING)
{
// Magnetometer data event.
// Extract all of the event information.
magnetometer_received = true;
sensor_event_get_xyz(event, &m_x, &m_y, &m_z);
}
else if (event_code == SENSOR_GYROSCOPE_READING)
{
// Gyroscope data event.
// Extract all of the event information.
gyroscope_received = true;
sensor_event_get_xyz(event, &g_x, &g_y, &g_z);
}
else if (event_code == SENSOR_AZIMUTH_PITCH_ROLL_READING)
```

```
{
// Azimuth, pitch, and roll data event.
// Extract all of the event information.
azimuth_pitch_roll_received = true;
sensor_event_get_apr(event, &apr_a, &apr_p, &apr_r);
}
else if (event_code == SENSOR_LINEAR_ACCEL_READING)
{
// Linear acceleration data event.
// Extract all of the event information.
linear_accel_received = true;
sensor_event_get_xyz(event, &la_x, &la_y, &la_z);
}
else if (event_code == SENSOR_ROTATION_VECTOR_READING)
{
// Rotation vector data event.
// Extract all of the event information.
rotation_vector_received = true;
sensor_event_get_rotation_vector(event, &rv);
}
else if (event_code == SENSOR_ROTATION_MATRIX_READING)
{
// Rotation vector data event.
// Extract all of the event information.
rotation_matrix_received = true;
sensor_event_get_rotation_matrix(event, &rm);
}
// All other codes are ignored.
}
}

// If we have received all of the supported sensors once, do recording logic.
if ((accelerometer_received)
&& (magnetometer_received)
&& (gyroscope_received)
&& (azimuth_pitch_roll_received)
&& (linear_accel_received)
&& (rotation_vector_received)
&& (rotation_matrix_received))
{
// Get the end time.
clock_gettime(CLOCK_REALTIME, &end_t);

// Calculate the time interval in microseconds.
interval_t = (double)(end_t.tv_sec - start_t.tv_sec)*(double)MILLION;
interval_t = interval_t + (double)(end_t.tv_nsec - start_t.tv_nsec)/(double)THOUSAND;

if (interval_t >= WRITE_PERIOD)
{

// This is the start time for the next iteration.
start_t = end_t;

// Update the screen if required.
if (update_screen)
{
// Create strings for the GPS data.
char gps_la_buffer[ITEM_SIZE];
char gps_lo_buffer[ITEM_SIZE];
char gps_ac_buffer[ITEM_SIZE];
char gps_h_buffer[ITEM_SIZE];
char gps_s_buffer[ITEM_SIZE];
char gps_t_buffer[ITEM_SIZE];
if (gps_received)
{
snprintf(gps_la_buffer, ITEM_SIZE, "%lf", gps_la);
snprintf(gps_lo_buffer, ITEM_SIZE, "%lf", gps_lo);
snprintf(gps_ac_buffer, ITEM_SIZE, "%lf", gps_ac);
if (gps_h_valid)
{
snprintf(gps_h_buffer, ITEM_SIZE, "%lf", gps_h);
}
else
```

```
{
snprintf(gps_h_buffer, ITEM_SIZE, "INVALID");
}
if (gps_s_valid)
{
snprintf(gps_s_buffer, ITEM_SIZE, "%lf", gps_s);
}
else
{
snprintf(gps_s_buffer, ITEM_SIZE, "INVALID");
}
if (gps_t_valid)
{
snprintf(gps_t_buffer, ITEM_SIZE, "%s", ctime(&gps_t));
gps_t_buffer[strlen(gps_t_buffer)-1] = '\0';
}
else
{
snprintf(gps_t_buffer, ITEM_SIZE, "INVALID");
}
}
else
{
snprintf(gps_la_buffer, ITEM_SIZE, "INVALID");
snprintf(gps_lo_buffer, ITEM_SIZE, "INVALID");
snprintf(gps_ac_buffer, ITEM_SIZE, "INVALID");
snprintf(gps_h_buffer, ITEM_SIZE, "INVALID");
snprintf(gps_s_buffer, ITEM_SIZE, "INVALID");
snprintf(gps_t_buffer, ITEM_SIZE, "INVALID");
}

// Create a buffer, write to it, and display it.
char msg[MSG_SIZE];
snprintf(msg, MSG_SIZE, "<b>GPS Time</b> %s\n<b>GPS Latitude</b> %s \
<b>GPS Longitude</b> %s \
<b>GPS Accuracy</b> %s\n<b>GPS Heading</b> %s \
<b>GPS Speed</b> %s\n\
<b>Accelerometer</b> X: %f Y: %f Z: %f\n\
<b>Linear Accelerometer</b> X: %f Y: %f Z: %f\n\
<b>Magnetometer</b> X: %f Y: %f Z: %f\n\
<b>Gyroscope</b> X: %f Y: %f Z: %f\n\
<b>Azimuth</b> %f\n<b>Pitch</b> %f\n<b>Roll</b> %f\n\
<b>Rotation Vector</b> [%f, %f, %f, %f]\n\
<b>Rotation Matrix</b>\n[%f, %f, %f;\n %f, %f, %f;\n %f, %f, %f]\n\
<b>Time Interval</b> %lf\n",
gps_t_buffer, gps_la_buffer, gps_lo_buffer, gps_ac_buffer,
gps_h_buffer, gps_s_buffer,
a_x, a_y, a_z, la_x, la_y, la_z, m_x, m_y, m_z,
g_x, g_y, g_z,
apr_a, apr_p, apr_r,
rv.vector[0], rv.vector[1], rv.vector[2], rv.vector[3],
rm.matrix[0], rm.matrix[1], rm.matrix[2],
rm.matrix[3],  rm.matrix[4],  rm.matrix[5],
rm.matrix[6],  rm.matrix[7],  rm.matrix[8], interval_t);
show_dialog_message(msg);
}

// Deal with the validity of GPS data.
if (gps_received)
{
if (!gps_h_valid)
{
gps_h = -1;
}
if (!gps_s_valid)
{
gps_s = -1;
}
if (!gps_t_valid)
{
gps_t = -1;
}
}
```

```
else
{
gps_la = -1;
gps_lo = -1;
gps_ac = -1;
gps_h = -1;
gps_s = -1;
gps_t = -1;
}

// Write to file.
fprintf(fd,"%lld,%lf,%lf,%lf,%lf,%lf,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\
%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%lf\r\n",
gps_t, gps_la, gps_lo, gps_ac, gps_h, gps_s, a_x, a_y, a_z,
la_x, la_y, la_z, m_x, m_y, m_z, g_x, g_y, g_z,
apr_a, apr_p, apr_r,
rv.vector[0], rv.vector[1], rv.vector[2], rv.vector[3],
rm.matrix[0], rm.matrix[1], rm.matrix[2],
rm.matrix[3], rm.matrix[4], rm.matrix[5],
rm.matrix[6],  rm.matrix[7],  rm.matrix[8], interval_t);

// Increment the counter.
i_count = i_count + 1;

// Check if the file should be flushed to disk.
if(i_count >= FLUSH_INTERVAL)
{
// Yes.
// Close the file.
fclose(fd);

// Open the file for appending.
fd = fopen(f_name, "a");

// Reset the count.
i_count = 0;
}
}
}
}

// Close the file.
fclose(fd);

// Stop requesting events.
geolocation_stop_events(0);
sensor_stop_events(SENSOR_TYPE_ACCELEROMETER);
sensor_stop_events(SENSOR_TYPE_MAGNETOMETER);
sensor_stop_events(SENSOR_TYPE_GYROSCOPE);
sensor_stop_events(SENSOR_TYPE_AZIMUTH_PITCH_ROLL);
sensor_stop_events(SENSOR_TYPE_LINEAR_ACCEL);
sensor_stop_events(SENSOR_TYPE_ROTATION_VECTOR);
sensor_stop_events(SENSOR_TYPE_ROTATION_MATRIX);

// Clean up.
destroy_dialog();
cleanup_screen();
bps_shutdown();
return 0;
}
```

C.2 dialogutil.h

```
/*
 * Copyright (c) 2011-2012 Research In Motion Limited.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


#ifndef DIALOGUTIL_H_
#define DIALOGUTIL_H_

/**
 * Set up a basic screen, so that the navigator will send window state events
 * when the window state changes.
 *
 * @return @c EXIT_SUCCESS or @c EXIT_FAILURE
 */
int setup_screen();

/**
 * Clean up all the resources that were allocated by setup_screen().
 */
void cleanup_screen();

/**
 * Show an alert dialog that will output general data.
 */
void create_dialog();

/**
 * Destroy dialog that was created by create_dialog.
 */
void destroy_dialog();

/**
 * Displays a message to the dialog created by create_dialog() and
 * outputs that message to stderr.  This means it will get added
 * to the log file in the sandbox and output to the console.
 *
 * @param msg the message to be displayed.
 */
void show_dialog_message(const char *msg);

#endif /* DIALOGUTIL_H_ */
```

C.3 dialogutil.c

```
/*
 * Copyright (c) 2011-2012 Research In Motion Limited.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Modified by Ryan Sammon.
 * Modifications Copyright (c) 2013 Ryan Sammon.
 * All rights reserved.
 */

#include <stdio.h>
#include <stdlib.h>

#include <screen/screen.h>
#include <bps/bps.h>
#include <bps/dialog.h>
#include <bps/navigator.h>
#include "dialogutil.h"

static screen_context_t screen_ctx = 0;
static screen_window_t screen_win = 0;
static dialog_instance_t main_dialog = 0;

/**
 * Use the PID to set the window group id.
 */
static char *
get_window_group_id()
{
    static char s_window_group_id[16] = "";

    if (s_window_group_id[0] == '\0') {
        snprintf(s_window_group_id, sizeof(s_window_group_id), "%d", getpid());
    }

    return s_window_group_id;
}

int
setup_screen()
{
    if (screen_create_context(&screen_ctx, SCREEN_APPLICATION_CONTEXT) != 0) {
        return EXIT_FAILURE;
    }

    //Signal BPS library that navigator orientation is to be locked
    if (BPS_SUCCESS != navigator_rotation_lock(true)) {
        screen_destroy_context(screen_ctx);
        return EXIT_FAILURE;
    }

    if (screen_create_window(&screen_win, screen_ctx) != 0) {
        screen_destroy_context(screen_ctx);
        return EXIT_FAILURE;
    }

    int usage = SCREEN_USAGE_NATIVE;
    if (screen_set_window_property_iv
     (screen_win, SCREEN_PROPERTY_USAGE, &usage)
     != 0) goto fail;
```

```
    int size[2];
    if (screen_get_window_property_iv
     (screen_win, SCREEN_PROPERTY_BUFFER_SIZE, size)
     != 0) goto fail;

    screen_display_t screen_disp;
    screen_get_window_property_pv(screen_win,
     SCREEN_PROPERTY_DISPLAY, (void **)&screen_disp);

    screen_display_mode_t screen_mode;
    if (screen_get_display_property_pv
     (screen_disp, SCREEN_PROPERTY_MODE, (void**)&screen_mode)
     != 0) goto fail;

    int buffer_size[2] = {size[0], size[1]};

    int angle = atoi(getenv("ORIENTATION"));
    if ((angle == 0) || (angle == 180)) {
       if (((screen_mode.width > screen_mode.height) && (size[0] < size[1])) ||
          ((screen_mode.width < screen_mode.height) && (size[0] > size[1]))) {
            buffer_size[1] = size[0];
        buffer_size[0] = size[1];
       }
    } else if ((angle == 90) || (angle == 270)){
       if (((screen_mode.width > screen_mode.height) && (size[0] > size[1])) ||
          ((screen_mode.width < screen_mode.height && size[0] < size[1]))) {
        buffer_size[1] = size[0];
        buffer_size[0] = size[1];
        }
    } else {
        goto fail;
    }

    if (screen_set_window_property_iv
     (screen_win, SCREEN_PROPERTY_BUFFER_SIZE, buffer_size)
     != 0) goto fail;

    if (screen_set_window_property_iv
     (screen_win, SCREEN_PROPERTY_ROTATION, &angle)
     != 0) goto fail;

    int idle_mode = SCREEN_IDLE_MODE_KEEP_AWAKE;
    if (screen_set_window_property_iv
     (screen_win, SCREEN_PROPERTY_IDLE_MODE, &idle_mode)
     != 0) goto fail;

    if (screen_create_window_buffers(screen_win, 1)
     != 0) goto fail;

    if (screen_create_window_group(screen_win, get_window_group_id())
     != 0) goto fail;

    screen_buffer_t buff;
    if (screen_get_window_property_pv
     (screen_win, SCREEN_PROPERTY_RENDER_BUFFERS, (void*)&buff)
     != 0) goto fail;

    int attribs[1] = {SCREEN_BLIT_END};
    if (screen_fill(screen_ctx, buff, attribs) != 0) goto fail;

    int dirty_rects[4] = {0, 0, buffer_size[0], buffer_size[1]};
    if (screen_post_window(screen_win, buff, 1, (const int*)dirty_rects, 0)
     != 0) goto fail;

    return EXIT_SUCCESS;

fail:
    perror(NULL);
    cleanup_screen();
    return EXIT_FAILURE;
}
```

```
void cleanup_screen() {
    screen_destroy_window(screen_win);
    screen_destroy_context(screen_ctx);
    screen_win = 0;
    screen_ctx = 0;
}



void
create_dialog()
{
    if (main_dialog) {
        return;
    }

    dialog_create_alert(&main_dialog);
    dialog_set_alert_message_text(main_dialog, "\n");
    dialog_set_size(main_dialog, DIALOG_SIZE_FULL);
    dialog_set_group_id(main_dialog, get_window_group_id());
    dialog_set_cancel_required(main_dialog, true);
    dialog_show(main_dialog);
}

void
destroy_dialog() {
    if (main_dialog) {
        dialog_destroy(main_dialog);
    }
    main_dialog = 0;
}

void
show_dialog_message(const char * msg) {
dialog_set_alert_html_message_text(main_dialog, msg);
    dialog_update(main_dialog);
}
```

D Apache License, Version 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each

Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following
boilerplate notice, with the fields enclosed by brackets "[]"
replaced with your own identifying information. (Don't include
the brackets!) The text should be enclosed in the appropriate
comment syntax for the file format. We also recommend that a
file or class name and description of purpose be included on the
same "printed page" as the copyright notice for easier
identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.