Curtin University
Curtin Institute of Radio Astronomy
Physics Project 1

Finding New Pulsars Using Machine Learning

Jacob Ian Matthews

June 5, 2020

Supervisors

Dr Sam McSweeney and Dr Ramesh Bhat

Abstract

As radio telescope technology improves, the number of candidate detections to be classified as pulsar or non-pulsar during pulsar surveys increases to the point that it becomes infeasible to classify them manually, a state of affairs known as the "pulsar candidate selection problem". To solve this problem, machine learning (ML) techniques are used to filter out the non-pulsar candidates in surveys, reducing the number of candidates requiring human evaluation to a manageable number. Here, we adapt an ML solution developed for the LOFAR Tied-Array All-Sky Survey (LOTAAS) for the Southern-sky Murchison Widefield Array Rapid Two-metre (SMART) survey. A group of 147 pulsar and 87 non-pulsar candidates generated by the SMART survey were used to test the LOTAAS ML classifiers trained on a set of 12 pulsar and 13 non-pulsar SMART survey candidates. The classifiers were found, on average, to make accurate pulsar and non-pulsar classifications on SMART survey candidates 87.3% of the time, compared to the 97.1% success rate obtained by Lyon et al. (2016). Using the machine learning classifiers in ensemble with a selection criteria of three separate positive classifications increased the success rate of non-pulsar classification by 5.2% (making 0 False Positive classifications), however it decreased the success rate of pulsar classification by 3.5%. The ensemble machine learning classifier can be further improved by increasing the diversity of pulsars within the machine learning training dataset.

Contents

1 Introduction

1.1 Aim

This project consists of three aims:

i. Investigate the use of Machine Learning (ML) techniques in surveying pulsars;

ii. Create a training dataset for a Machine Learning algorithm to find pulsars in data obtained by the Murchison Widefield Array (MWA); and

iii. Evaluate the utility of the Machine Learning algorithms used by the LOFAR Telescope for the Murchison Widefield Array, and adjust the algorithms as necessary to achieve optimum pulsar candidate classification.

1.2 Structure of this Report

In this report, I will first explain in *Section 1.3* how Pulsars, Radio Astronomy, and Machine learning work, and then explain the "Candidate Selection Problem" (Lyon et al. 2016) and why Machine Learning is necessary in completing future pulsar surveys.

In *Section 2*, I discuss the methods undertaken in: (i) developing the machine learning training dataset for the Murchison Widefield Array, (ii) developing software to validate the output of machine learning classifiers, (iii) evaluating the machine learning algorithms used by the LOFAR surveys for use with the Murchison Widefield Array, and (iv) developing an ensemble machine learning classifier to be used with the Murchison Widefield Array and other radio telescopes.

In *Section 3*, I analyse the results and findings obtained by the methods described in *Section 2*, and in *Section 4* I will discuss (i) the current pulsar classification methods used at the Curtin Institute of Radio Astronomy, (ii) the efficacy of the machine learning classifiers and training dataset produced in this report, and (iii) my concluding remarks about the utility of the machine learning classifiers with the Murchison Widefield Array.

This report will conclude with my recommendations for areas of further development in *Section 5*, and the detailed methodologies in creating the results of this project, and entirety of the source code created undertaking this project, in the *Appendices*.

1.3 Background Theory

1.3.1 Pulsars

Pulsars were first detected in 1967 at the Mullard Radio Astronomy Observatory when a periodically pulsing radio source was discovered (Hewish et al. 1968). The pulsing radio source was found to not be man-made or of terrestrial nature, as it could only be observed at a particular declination and right ascension (Hewish et al. 1968). Pulsars were thus suggested to be either rotating white dwarf stars or neutron stars (Hewish et al. 1968; Gold 1969), the latter suggestion of which was later supported by the works of Gold (1969). Gold (1969) found that pulsars were more often detected in supernova areas – supernovae being the producers of neutron stars – and that pulsars with faster rotations corresponded to younger neutron stars. The theory that a pulsar is a rapidly rotating neutron star was further confirmed by the works of Ostriker and Gunn (1969), as well as discovering that pulsars possess a dipole magnetic field that isn't parallel to its rotation axis. A diagram of a pulsar and its magnetic field can be seen in *Figure 1*.

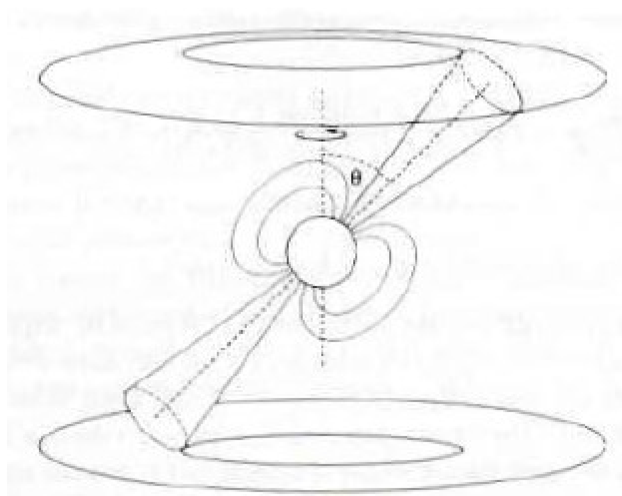

Figure 1: A pulsar (Maoz 2016).

When a star reaches the end of its life, if its initial mass was greater than eight times the mass of the sun, i.e. $8M_{\odot}$, the star will likely supernova, exploding its outer envelope into the surrounding space, leaving the gravitationally collapsed core of the star (Maoz 2016). This is a neutron star. Using the conservation of angular momentum, we can prove that the collapsed neutron star will have a speed of rotation much greater than the star did prior to the its gravitational collapse. If one considers an ice skater rotating with their arms stretched out, we intuitively know that their speed of rotation will increase if their arms are brought in closer to their body. The same effect applies for a star: if prior to the star's collapse it is rotating at an angular velocity ω_1, the angular velocity ω_2 after the star collapses to a radius of 11km from a radius thousands of times larger, will be much greater (Maoz 2016).

Despite our understanding of the formation of neutron stars and their rotation, the nature of their electromagnetic wave emission is still an active area of research (Lorimer and Kramer 2005). The reason that pulsars appear to emit a pulse of radio waves is due to a process called the 'lighthouse effect': as the neutron star rotates, the radiated electromagnetic waves sweep across the sky, crossing the line of sight of an observer and producing a pulse-like effect (Lorimer and Kramer 2005). When the radio wave passes the line of sight of a radio telescope, the pulse can be recorded. An Integrated Pulse Profile can then be generated from the pulse recorded by the telescope by a process called folding, where a number of separate observations of the pulse can be 'stacked' on top of each other to produce a clear representation of the pulse (Helfand, Manchester, and Taylor 1975). An example of an integrated pulse profile can be seen in *Figure 2*.

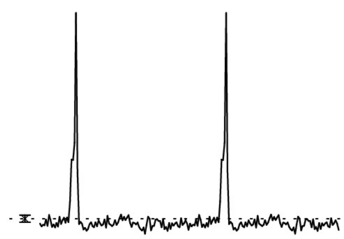

Figure 2: Integrated Pulse Profile of Pulsar 0459-0210 (Swainston 2020a).

1.3.2 Sky Surveys

When completing sky surveys in radio astronomy, it is common to detect many examples of radio frequency interference and noise (Hewish et al. 1968; Lorimer and Kramer 2005; Lyon et al. 2016; Tan et al. 2017). Radio frequency interference (RFI) is any radio signal that has been unintentionally detected; it normally arises from electrical devices with a periodic nature, such as devices with AC currents, communication systems like radar, or from electrical storms (Lorimer and Kramer 2005). Noise, on the other hand, can be detections of the Cosmic Microwave Background (depending on the sensitivity of the radio telescope), or detections of radio waves produced by the electronics of the telescope (Lorimer and Kramer 2005). An example of an integrated pulse profile produced by radio frequency interference and noise can be seen in *Figure 3*.

It is scientifically important to survey the skies to find new pulsars for a multitude of reasons. For example: due to their extremely reliable periodicity,

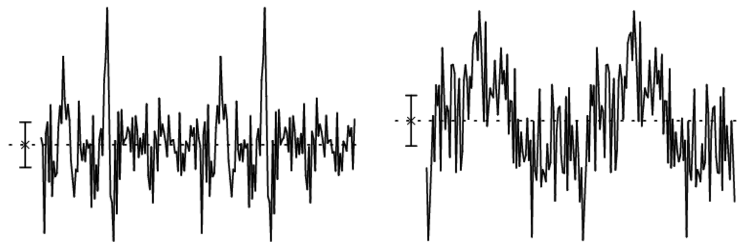

Figure 3: Examples of RFI (left) and Noise (right) (Swainston 2020a).

pulsars can be used as astronomical clocks of high accuracy (Matsakis, Taylor, and Eubanks 1997). Also, due to the massive gravitational fields surrounding pulsars, binary pulsars are the definitive area in which to test gravitational theories in the strong-field, such as Einstein's theory of General Relativity, and future research into quantum gravity (Lorimer and Kramer 2005).

Throughout the history of pulsar discovery, as the sky survey techniques of detection and radio telescope technology has improved, the number of candidates detected has grown at a very fast rate (Lyon et al. 2016). The growth rate of the number of candidates stands to increase further with the development of extremely large radio telescopes such as the Square Kilometre Array (SKA) (Lyon et al. 2016). This introduces a new problem to sky surveying: the candidate selection problem. The candidate selection problem occurs when there are too many detected pulsar candidates to be classified as pulsar or as a non-pulsar than can be feasibly examined by human eyes (Lyon et al. 2016).

To attempt to solve the candidate selection problem, Lyon et al. (2016) turned to using a machine learning classifier as an intelligent filter that will filter out the candidates that are radio frequency interference or noise, reducing the candidates to be classified by a human to a more manageable number.

1.3.3 Machine Learning

Valiant (1984) first described machine learning as a function in which a computer completes a task that hasn't been explicitly programmed, it has instead learned to do so. A supervised learning classifier is a function in which a machine learns examples of a particular class and then classifies other inputted data based on the learned model (Dietterich 1998).

In addressing the candidate selection problem, Lyon et al. (2016) created a program named `LOTAASClassifier` which contains four separate machine learning algorithms that can be chosen from:

1. The C4.5 Decision Tree;

2. the Multilayered Perceptron;

3. the Naive Bayes classifier; and
4. the Support Vector Machine.

The machine learning classifiers from `LOTAASClassifier` generate a classification model from an inputted training dataset, and then they can make classification predictions on the class of pulsar candidates (Lyon et al. 2016). The training dataset used by Lyon et al. (2016) contained a set of pulsars, radio frequency interference, and noise. This allows the machine learning classifiers to distinctly identify what is and isn't a pulsar.

By using the machine learning classifiers, Lyon et al. (2016) was able to accurately predict the class of the candidates 97.1%. of the time, reducing the number of candidates that had to be manually classified.

2 Methods

2.1 Developing the Machine Learning Training Dataset

Before a machine learning algorithm can make predictions and classify candidates as a pulsar or a non-pulsar, it must first build a classification model from a training dataset which contains similar data with known positive and negative classifications (Tan et al. 2017; Lyon et al. 2016). For the use case of pulsar classification, the training dataset must contain examples of data from both pulsars and from non-pulsars so that the algorithm can learn how to distinguish between the two classes.

To maximise the accuracy of the machine learning algorithm, the input data (including the training dataset) must be composed of a common group of features that can be determined for each candidate that maximises the differences between a pulsar and a non-pulsar. The candidate features used by Tan et al. (2017) to maximise the differences between pulsar and non-pulsar candidates are:

$$Prof_{\mu}, Prof_{\sigma}, Prof_{S}, Prof_{k} \tag{1}$$

$$DM_{\mu}, DM_{\sigma}, DM_{S}, DM_{k}, DM_{\mu'}, DM_{\sigma'}, DM_{|S'|}, DM_{k'} \tag{2}$$

$$Subband_{\mu}, Subband_{\sigma}, Subband_{S}, Subband_{k} \tag{3}$$

$$Subint_{\mu}, Subint_{\sigma}, Subint_{S}, Subint_{k} \tag{4}$$

Where candidate features are calculated from the: (1) Integrated Pulsar Profile, (2) the Dispersion Measure – Signal-to-Noise Ratio Curve (DM-S/N curve), (3) the correlation coefficients between each sub-band and the integrated pulsar profile, and (4) from the correlation coefficients between each sub-integration and the integrated pulsar profile.

A set of 12 pulsar and 13 non-pulsar candidates from the SMART survey were provided by N. Swainston to become the machine learning training dataset. The free Python software created by Lyon et al. (2016), `PulsarFeatureLab`, was used to process the PRESTO Prepfold `PFD` pulsar candidate files, extracting the above 20 machine learning features from each candidate into a single file of WEKA Data Mining `ARFF` filetype. To create a closed software environment in which the dependencies of the `PulsarFeatureLab` software were unaffected by the host operating system, a containerised virtual operating system was created using the free software Docker (https://docker.com) and the pulsar feature extraction software was thus ran using a bind mounted volume. For further details on how this was achieved, see *Appendix A.A*.

The file outputted by `PulsarFeatureLab` containing the feature extracted candidates was then edited, removing the '?' character and appending a '1' or '0' to the end of each candidate's line depending on whether the candidate was a pulsar or non-pulsar respectively. This tells the machine learning classifier that candidates with features similar to those preceding the appended binary classification belong to a pulsar (or a non-pulsar). The remaining file constitutes the machine learning training dataset.

2.2 Developing a Machine Learning Classifier Validation Tool

In order to automate the evaluation of predictions made by the machine learning pulsar classifiers, a Java program named `PulsarValidator` was developed.

`PulsarValidator` was designed to load the positive and negative output files generated from the machine learning classification of a controlled testing dataset of pulsar candidates, and compare these files to a list of the pulsars included in the testing dataset. The software outputs the following statistics: the number of pulsars and non-pulsars, the number of detected pulsars and non-pulsars, and the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

This was achieved by first creating individual Java 'ArrayList's to contain the filenames of each candidate found to be a True Positive, False Positive, True Negative, or False Negative classification. The positive and negative output files generated by the classifier (positive denoting pulsars and negative denoting non-pulsars), and the list of pulsars in the testing dataset are then also converted to Java 'ArrayList's. The items inside the positive and negative 'ArrayList's are then iterated over, checking each item against the list of pulsars. If an item in the positive list is found in the list of pulsars, it is added to the True Positive list, otherwise it is added to the False Positive list. If an item in the negative list is found in the list of pulsars, it is added to the False Negative list, otherwise it is added to the True Negative list. The number of items inside each list is then outputted to the user. This process can be seen in *Algorithm 1*. Further details on the development of this software can be found in *Appendix A.B*, and the full Java source code can be found in *Appendix C* or on GitHub (https://github.com/jacob-ian/PulsarValidator.git).

2.3 Developing an Ensemble Machine Learning Classifier

To implement the ensemble classifier feature outlined by Tan et al. (2017) to the `LOTAASClassifier` software, a new Java program named `PulsarClassifier` was developed.

`PulsarClassifier` was designed so that a new 'ensemble classifier' option can be chosen as the classification algorithm as well as the previously existing algorithms from `LOTAASClassifier`. The program outputs the classification results of all machine learning classifiers from the `LOTAASClassifier` program, as well as that of the new ensemble classifier. This was achieved by creating the following new Java classes: 'PulsarClassifier', 'ClassPredictor', 'Classifier-Builder', and 'ClassifierValidator', and extending the existing versions of those

Algorithm 1 Pulsar Validator (pseudocode)

```
Let truePositive = new List
Let falsePositive = new List
Let trueNegative = new List
Let falseNegative = new List
for each item in classifierPositive list do
    Let Boolean found = false
    Let Integer i = 0
    while found is false do
        if ith item in pulsarList is item then
            add item to truePositive list
            found = true
        else if i equals number of items in pulsarList then
            add item to falsePositive list
            found = true
        else
            i=i + 1
        end if
    end while
end for
for each item in classifierNegative list do
    Let Boolean found = false
    Let Integer i = 0
    while found is false do
        if ith item in pulsarList is item then
            add item to falseNegative list
            found = true
        else if i equals number of items in pulsarList then
            add item to truePositive list
            found = true
        else
            i=i + 1
        end if
    end while
end for
Let TP = number of items in truePositive
Let TN = number of items in trueNegative
Let FP = number of items in falsePositive
Let FN = number of items in falseNegative
Let Pulsars = TP + FN
Let NonPulsars = TN + FP
Output Pulsars, NonPulsars, TP, FP, TN, FN
```

classes from `LOTAASClassifier` to maintain some of the prior methods and properties.

The Java classes ‘ClassifierBuilder’ and ‘ClassifierValidator’ were modified such that if the user selected the ensemble classifier as the classification algorithm, the method involved in building or validating a machine learning classifier would iterate through the list of available machine learning classifiers (C4.5, Multi-Layered Perceptron, Naïve Bayes, and Support Vector Machine) to build a classification model for each ML classifier, or validate each ML classifier, respectively. These processes can be seen in *Algorithm 2* and *Algorithm 3*.

Algorithm 2 ClassifierBuilder (pseudocode)

if $algorithm = -1$ **then**
  for each algorithm $i = 1 \rightarrow 4$ **do** ▷ Loop through all classifiers
    buildClassifier(i, $trainingSet$, $modelsDirectory$)
  end for
else ▷ Build individual classifier
  buildClassifier($algorithm$, $trainingSet$, $modelPath$)
end if

Algorithm 3 ClassifierValidator (pseudocode)

if $algorithm = -1$ **then**
  for each algorithm $i = 1 \rightarrow 4$ **do** ▷ Loop through all classifiers
    testClassifier(i, $testSet$, $modelsDirectory$)
  end for
else ▷ Test individual classifier
  testClassifier($algorithm$, $testSet$, $modelPath$)
end if

The Java class ‘ClassPredictor’ was modified such that if the user chooses to use the ensemble classifier, a Java ‘ArrayList’ is created to store the filepaths to the outputs of each machine learning classifier. The list of available machine learning classifiers is then iterated over, generating the classification predictions from each classifier and adding their outputs to the created list of output files. The Java classes ‘Classification’ and ‘ClassificationList’ are then created to be a key-value pair data class and a list of key-value classifications respectively, where the key denotes the filename of a candidate and the value denotes the number of times this candidate has been given a particular classification. A ‘ClassificationList’ for positive and negative classifications are then constructed and the candidates in the positive and negative classifier output files are iterated over, adding the candidates to their respective lists and incrementing their associated value for every recurring classification. As discussed in Tan et al. (2017), a positive classification in an ensemble classifier is generally made when a candidate has received a positive classification from 3 separate classifiers. To implement this selection criteria, the positive ‘ClassificationList’ is iterated over and the candidates with an associated value less than 3 are moved into the negative ‘ClassificationList’. The remaining positive and negative ‘ClassificationList’s are then outputted to their respective positive and negative output files for the ensemble classifier. This process can be seen in *Algorithm 4*.

Algorithm 4 ClassPredictor (pseudocode)

```
if algorithm = −1 then
    list = new OutputFileList()
    for algorithm i = 1 → 4 do                        ▷ Loop through all classifiers
        makePredictions(i, inputData, modelsDirectory)
        list.add(ClassifierOutputFiles)               ▷ Add output filepaths to list
    end for
    positiveList = new ClassificationsList()
    negativeList = new ClassificationsList()          ▷ Create classification lists
    for each OutputFile from list do
        if OutputFile is .positive then
            for each line in OutputFile do
                positiveList.add(line)                ▷ Add to +ve classifications
            end for
        else if OutputFile is .negative then
            for each line in OutputFile do
                negativeList.add(line)                ▷ Add to -ve classifications
            end for
        end if
    end for
    for each classification in positiveList do
        if classification.occurrences < 3 then
            negativeList.add(classification)          ▷ cut-off at < 3 classifications
        else
            positiveOutput(classification)            ▷ Output classified as Pulsar
        end if
    end for
    for each classification in negativeList do
        negativeOutput(classification)                ▷ Output classified as non-Pulsar
    end for
else
    makePredictions(algorithm)                        ▷ Use individual classifier
end if
```

The final modification that was made included changing the Java class ‘PulsarClassifier’ to include the ensemble classifier as an option in the user selectable algorithms. Further details on the development of this software can be found in *Appendix A.C* and the full Java source code can be found in *Appendix D* or on GitHub (https://github.com/jacob-ian/PulsarClassifier.git).

2.4 Evaluating the Machine Learning Classifiers

A set of 147 pulsar and 87 non-pulsar candidates generated by the SMART survey were then provided by N. Swainston to test the machine learning classifiers on Murchison Widefield Array data. To test all machine learning classifiers present in `LOTAASClassifier`, and the ensemble classifier, the new `PulsarClassifier` program was used.

Before making predictions on the test data, it is necessary to create the machine learning classification models from the training dataset. Using `PulsarClassifier`, the training dataset was used to generate the classification models for each ML classifier. `PulsarFeatureLab` was then used to extract the machine learning features from the testing dataset of candidates. `PulsarClassifier` was then used to classify the candidates in the testing dataset with the ensemble classifier (and hence with each of the other ML classifiers). For details on the use of `PulsarClassifier`, see *Appendix A.D*.

To evaluate the accuracy of the predictions of the machine learning classifiers, a list of the pulsars included in the testing dataset was created and `PulsarValidator` was ran with the output files of each classifier. For details on the use of `PulsarValidator`, see *Appendix A.E*.

3 Results and Analyses

3.1 Machine Learning Training Dataset

The machine learning training dataset created with candidates detected by the Murchison Widefield Array can be found under *Appendix B.1*.

3.2 Classification Results from the LOTAASClassifier Algorithms

3.2.1 The J48 Algorithm

The output created by `PulsarValidator` on analysis of the classification results of the J48 (C4.5 Decision Tree) algorithm is as follows:

```
Number of Pulsars: 147
Pulsars Detected: 146
True Positives: 131
False Positives: 15

Number of Non-Pulsars: 87
Non-Pulsars Detected: 88
True Negatives: 72
False Negatives: 16
```

We can therefore calculate the pulsar classification success rate to be:

$$R_p = \frac{TP}{N_p} = \frac{131}{147} = 0.8911 = 89.11\%,$$

where N_p is the number of pulsars in the testing dataset and TP is the number of true positive classifications. The non-pulsar classification success rate can be found as:

$$R_{np} = \frac{TN}{N_{np}} = \frac{72}{87} = 0.8275 = 82.75\%,$$

where N_{np} is the number of non-pulsars in the testing dataset and TN is the number of true negative classifications.

3.2.2 The Multi-Layer Perceptron Algorithm

The output created by `PulsarValidator` on analysis of the classification results of the Multi-Layer Perceptron algorithm is as follows:

```
Number of Pulsars: 147
Pulsars Detected: 125
True Positives: 123
False Positives: 2
```

```
Number of Non-Pulsars: 87
Non-Pulsars Detected: 109
True Negatives: 85
False Negatives: 24
```

We can therefore calculate the pulsar classification success rate to be:

$$R_p = \frac{TP}{N} = \frac{123}{147} = 0.8367 = 83.67\%,$$

where N is the number of pulsars in the testing dataset, and TP is the number of true positive classifications. The non-pulsar classification success rate can be found as:

$$R_{np} = \frac{TN}{N_{np}} = \frac{85}{87} = 0.9770 = 97.70\%,$$

where N_{np} is the number of non-pulsars in the testing dataset and TN is the number of true negative classifications.

3.2.3 The Naïve Bayes Tester Algorithm

The output created by `PulsarValidator` on analysis of the classification results of the Naïve Bayes algorithm is as follows:

```
Number of Pulsars: 147
Pulsars Detected: 119
True Positives: 118
False Positives: 1

Number of Non-Pulsars: 87
Non-Pulsars Detected: 115
True Negatives: 86
False Negatives: 29
```

We can therefore calculate the pulsar classification success rate to be:

$$R_p = \frac{TP}{N} = \frac{118}{147} = 0.8027 = 80.27\%,$$

where N is the number of pulsars in the testing dataset, and TP is the number of true positive classifications. The non-pulsar classification success rate can be found as:

$$R_{np} = \frac{TN}{N_{np}} = \frac{86}{87} = 0.9885 = 98.85\%,$$

where N_{np} is the number of non-pulsars in the testing dataset and TN is the number of true negative classifications.

3.2.4 The Support Vector Machine Algorithm

The output created by `PulsarValidator` on analysis of the classification results of the Naïve Bayes algorithm is as follows:

```
Number of Pulsars: 147
Pulsars Detected: 97
True Positives: 97
False Positives: 0

Number of Non-Pulsars: 87
Non-Pulsars Detected: 137
True Negatives: 87
False Negatives: 50
```

We can therefore calculate the pulsar classification success rate to be:

$$R_p = \frac{TP}{N} = \frac{97}{147} = 0.6598 = 65.98\%,$$

where N is the number of pulsars in the testing dataset, and TP is the number of true positive classifications. The non-pulsar classification success rate can be found as:

$$R_{np} = \frac{TN}{N_{np}} = \frac{87}{87} = 1.00 = 100.00\%,$$

where N_{np} is the number of non-pulsars in the testing dataset and TN is the number of true negative classifications.

3.3 Classification Results from the PulsarClassifier Ensemble Classifier

The output created by `PulsarValidator` on analysis of the classification results of the `PulsarClassifier` ensemble classifier is as follows:

```
Number of Pulsars: 147
Pulsars Detected: 112
True Positives: 112
False Positives: 0

Number of Non-Pulsars: 87
Non-Pulsars Detected: 122
True Negatives: 87
False Negatives: 35
```

We can therefore calculate the pulsar classification success rate to be:

$$R_p = \frac{TP}{N} = \frac{112}{147} = 0.7619 = 76.19\%,$$

where N is the number of pulsars in the testing dataset, and TP is the number of true positive classifications. The non-pulsar classification success rate can be found as:

$$R_{np} = \frac{TN}{N_{np}} = \frac{87}{87} = 1.00 = 100.00\%,$$

where N_{np} is the number of non-pulsars in the testing dataset and TN is the number of true negative classifications.

3.4 Compilation of Results and Analyses

A table containing the number of true and false positives and negatives for each classifier can be found below in *Table 1*:

Table 1: Compiled Results of the Machine Learning Classifiers

	Pulsars		Non-Pulsars	
Classifier	TP	FN	TN	FP
LOTAASClassifier				
J48	131	16	72	15
MLP	123	24	85	2
NB	118	29	86	1
SVM	97	50	87	0
PulsarClassifier				
Ensemble	112	35	87	0

TP: Number of True Positives.
FN: Number of False Negatives.
TN: Number of True Negatives.
FP: Number of False Positives.

A table containing the pulsar and non-pulsar classification success rates of all algorithms and the ensemble classifier can be found below in *Table 2*:

Table 2: Success Rates of Each Pulsar Classifier

Classifier	R_p	R_{np}	*Combined*
LOTAASClassifier			
J48	89.11%	82.75%	85.93%
MLP	83.67%	97.70%	90.69%
NB	80.27%	98.85%	89.56%
SVM	65.98%	100.00%	82.99%
Average	79.76%	94.83%	87.30%
PulsarClassifier			
Ensemble	76.19%	100.00%	88.10%

R_p: Success rate of pulsar classification.
R_{np}: Success rate of non-pulsar classification.
Combined: Combined success rate (total accuracy).

4 Discussion and Conclusions

4.1 Evaluating Curtin Institute of Radio Astronomy's Pulsar Classification Pipeline

A pulsar classification pipeline can be defined as the process undertaken to classify a candidate as a pulsar (Swainston 2020b). The current pipeline at the Curtin Institute of Radio Astronomy (CIRA) for pulsar classification is as follows:

i. Generate pulsar candidates through the SMART (Southern-sky MWA Rapid Two-metre) survey;

ii. Extract machine learning features from the pulsar candidates using `PulsarFeatureLab`;

iii. Use the LOFAR Tied-Array All-Sky machine learning pulsar classifier, `LOTAASClassifier`, to to eliminate a large number of pulsar candidates; and

iv. Manually inspect the remaining pulsar candidates to confirm pulsar discovery.

(Smith and Swainston 2019). While the classification pipeline itself appears to be optimal, I have identified a few issues with the previous attempts at CIRA in using the machine learning classifier.

The first issue revolves around the use of mislabelled and incomplete software. In the research completed by Tan et al. (2017), the original `LOTAASClassifier`, created by Lyon et al. (2016), was upgraded to include two major new features: (i) ensemble classification, and (ii) radio frequency interference (RFI) classification. The machine learning feature set was also expanded from 8 features in Lyon et al. (2016) to 20 features in Tan et al. (2017), greatly improving the accuracy of classification. Upon analysis of the source code of `LOTAASClassifier`, it appears that the two new classification features were not released publicly, despite the software being labelled as `LOTAASClassifier v2.0` in its main Java class, the same name referenced by Tan et al. (2017). Despite this, the new, expanded set of machine learning features was released with `PulsarFeatureLab Version 1.3.2`. This created a mismatch between the feature extraction software and the machine learning classifier, which leads to the next issue in the CIRA pipeline: the training dataset.

In the pipeline provided by Smith and Swainston (2019), it appears that CIRA has been attempting to use the training dataset generated for the LOFAR Tied-Array All-Sky Survey included with the `LOTAASClassifier` software, for pulsar candidates generated in the SMART survey. In itself, this may not present issues, however due to the unknown mismatch in the machine learning software, an issue of feature dimensions occurs. The included training dataset creates a classification model with the original 8 machine learning features, therefore any predictions on new candidates would require the same 8 machine learning features to have been extracted prior to classification. Due to the mislabelled software, CIRA has been extracting the set of 20 features from pulsar candidates and attempting to make classifications against a set of 8 features, preventing classifications from occurring in `LOTAASClassifier` due to a mismatch in dimensions. This issue can be fixed by creating a training dataset with the same set of extracted features as will be present in the candidates for classification.

The final issue arising in the CIRA pipeline also spouts from the mislabelling of the `LOTAASClassifier` software. The CIRA pipeline expects that the ensemble classifier created in the upgraded software from Tan et al. (2017) will be used. Without the latest version of `LOTAASClassifier` having been released, the CIRA pipeline defaults to using the J48 machine learning classification algorithm. As seen in Table 2 from *Section 3.4*, the J48 algorithm does not appear to be optimal in combined pulsar and non-pulsar classification. To fix this issue, an ensemble machine learning classifier was created in *Section 2.3* to replace the `LOTAASClassifier` software.

4.2 Evaluating the Training Dataset and Machine Learning Classifiers

An evaluation of the machine learning classifiers: `LOTAASClassifier` and its algorithms, and `PulsarClassfier`, will in its nature be subject to the quality of the training dataset used. By inspecting the data produced in *Section 3.2*, we can see the success rates of pulsar and non-pulsar classification, which are defined as the ratio of the number of true positive (true negative) classifications with the number of pulsars (non-pulsars). The success rates of each classifier are then compiled into Table 2. Another metric, the combined success rate, is also introduced as the mean of the two success rates. The combined success rate can be used to rank the total accuracy of the classifiers, however this metric does not contain ample information. The classifiers can thus be ranked in order of highest total accuracy:

1. Multi-Layered Perceptron (90.69%)
2. Naïve Bayes Test (89.56%)
3. Ensemble Classifier (88.10%)
4. J48 (C4.5 Decision Tree) (85.93%)
5. Support Vector Machine (82.99%)

Despite being ranked third in combined success rates, the `PulsarClassifier`'s Ensemble Classifier was only one of two classifiers that correctly classified all examples of noise and radio frequency interference as *non-pulsar*, the other successful classifier being the Support Vector Machine. As a result of this, there were no false positive classifications completed by the Ensemble classifier, i.e. everything classified as a pulsar actually was a pulsar. The Ensemble Classifier was let down by misclassifying 35 pulsars, having the second-lowest success rate of pulsar classification - the lowest being the Support Vector Machine which misclassified 50 pulsars, and the highest being the J48 Classifier having misclassified only 16 pulsars.

For the use case of classifying pulsar candidates generated from the Murchison Widefield Array, the most important metric to be considered when evaluating a machine learning classifier is the pulsar classification success rate. It is more important to not miss the classification of a pulsar and less important if a non-pulsar is classified as a pulsar. For this reason, the objective is to maximise the rate of true positive classifications and minimise the rate of false negative classifications.

The training dataset created in this project contained 11 examples of pulsars and 12 examples of noise and radio frequency interference (RFI). While the training dataset appeared to have a satisfactory variety of noise and RFI, contributing to the ensemble classifier's perfect rate of non-pulsar classification, the set of pulsars appeared to be unsatisfactory. In order to improve the success rate of pulsar classification (positive success rate) in the ensemble classifier, we must use a modified training dataset that will improve the positive success rate in the worst performing individual classifiers: the Support Vector Machine (65.98%), and the Naïve Bayes Test (80.27%), whilst maintaining the higher positive success rates of the other classifiers. By doing so, more pulsars will survive the ensemble classifier's individual classification criteria of 3 or more, and thus the positive success rate will improve. This modification to the training dataset could involve changing the included set of pulsars to include a subset of pulsars of ordinary appearing features, and the remaining subset of pulsars should exhibit unusual or difficult to discern features.

In conclusion, I believe that the machine learning classifiers in the `LOTAASClassifier` software, particularly used in ensemble such as with the `PulsarClassifier` software created for this project, are of great utility for current and future Murchison Widefield Array pulsar surveys. As discussed in Lyon et al. (2016), the number of pulsar candidates generated in pulsar surveys stands to grow exponentially beyond the economical capacity of data storage and viability of manual examination. This problem requires a solution to filter out the non-pulsars from the group of candidates to maximise the discovery of pulsars, and I believe that the `PulsarClassifier` software is another step forward in solving the problem. While the training dataset does require further development to maximise the success of the Ensemble Classifier, the Machine Learning techniques themselves proved to be extremely valuable in pulsar classification.

5 Recommendations

For further development of the Machine Learning classifiers and strategies used in this project, I would recommend undertaking the following tasks:

1. Investigate and fix the Python `Traceback Error` produced by the software `PulsarFeatureLab v1.3.2`:

   During the usage of this feature extraction software, some `PFD` candidate files would cause a `Traceback` error to be produced, causing the feature extraction to fail for that particular candidate. The result of these failures was that there was a smaller dataset to train the machine learning classifiers on, and a smaller dataset to make machine learning classification predictions on.

2. Build the Radio Frequency Interference (RFI) classification feature discussed by Tan et al. (2017) into `PulsarClassifier`:

   According to Tan et al. (2017), by also including the classification category of RFI, the accuracy of the machine learning ensemble classifier was increased. The result of this feature would be three classifier output files: `output.pulsars`, `output.rfi`, and `output.other`.

3. Use a more diverse Training Dataset:

   Due to constraints on the available `PFD` candidate data during this project, I was unable to use a large set of pulsar and non-pulsar candidates in the training dataset. By using a training dataset that is more diverse, the `PulsarClassifier` will be more accurate in its predictions.

6 References

Dietterich. 1998. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms." *Neural Computation* 10 (7): 1895–1923. https://doi.org/10.1162/089976698300017197.

Gold, Thomas. 1969. "Rotating Neutron Stars and the Nature of Pulsars." *Nature* 221:25–27. https://doi.org/10.1038/221025a0.

Helfand, D. J., R. N. Manchester, and J. H. Taylor. 1975. "Observations of pulsar radio emission. III. Stability of integrated profiles." *Astrophysical Journal* 198 (June): 661–670. https://doi.org/10.1086/153644.

Hewish, A., S. J. Bell, J. D. H. Pilkington, P. F. Scott, and R. A. Collins. 1968. "Observation of Rapidly Pulsating Radio Source." *Nature* 217:709–713. https://doi.org/10.1038/217709a0.

Lorimer, D. R., and M. Kramer. 2005. *Handbook of Pulsar Astronomy.* Vol. 4. Cambridge, United Kingdom: Cambridge University Press.

Lyon, R. J., B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. 2016. "Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach." *Monthly Notices of the Royal Astronomical Society* 459, no. 1 (April): 1104–1123. https://doi.org/10.1093/mnras/stw656.

Maoz, Dan. 2016. *Astrophysics in a Nutshell.* 2nd ed. 64–94. Princeton, New Jersey: Princeton University Press.

Matsakis, D. N., J. H. Taylor, and T. M. Eubanks. 1997. "A statistic for describing pulsar and clock stabilities." *Astronomy and Astrophysics* 326:924–928.

Ostriker, J. P., and J. E. Gunn. 1969. "On the Nature of Pulsars. I. Theory." *Astrophysical Journal* 157 (September): 1395. https://doi.org/10.1086/150160.

Smith, Keegan, and Nick Swainston. 2019. "MWA Search Classifier Nextflow Script." https://github.com/NickSwainston/mwa_search/blob/master/nextflow/classifier.nf.

Swainston, Nick. 2020a. "Classifier Training Dataset V1." https://www.dropbox.com/sh/dwd50nzfb41es8k/AABAJ28UkgRLHnNc8WOaeOBwa?dl=0.

———. 2020b. "Investigating the Low-Frequency Population of Southern Pulsars with the MWA: Milestone 2," International Centre for Radio Astronomy Research.

Tan, C. M., R. J. Lyon, B. W. Stappers, S. Cooper, J. W. T. Hessels, V. I. Kondratiev, D. Michilli, and S. Sanidas. 2017. "Ensemble candidate classification for the LOTAAS pulsar survey." *Monthly Notices of the Royal Astronomical Society* 474 (4): 4571–4583. https://doi.org/10.1093/mnras/stx3047.

Valiant, L. G. 1984. "A Theory of the Learnable." *Communications of the ACM* 27 (11): 1134–1142. https://doi.org/10.1145/1968.1972.

Appendices

Appendix A Methods

Appendix A.A Feature Extraction with PulsarFeatureLab

First, a directory to store the Dockerfile and pulsar candidate data is created by completing the following commands in a UNIX terminal:

```
$ mkdir ~/pulsars
$ cd ~/pulsars
$ touch Dockerfile
```

To create the Docker image, the contents of the `Dockerfile` can be edited to contain:

Dockerfile

```
FROM alpine/git:latest as builder
WORKDIR /root/
RUN cd /root/ && git clone --single-branch --branch V1.3.2
    https://github.com/scienceguyrob/PulsarFeatureLab.git &&
    mkdir PulsarFeatureLab/PulsarFeatureLab/Data/IO

FROM python:2.7
WORKDIR /usr/src/app
COPY --from=builder /root/PulsarFeatureLab .
RUN pip install numpy scipy matplotlib astropy
ENTRYPOINT ["python", "./PulsarFeatureLab/Src/
    PulsarFeatureLab.py"]
```

The above `Dockerfile` instructs Docker to:

i. use an image of Alpine Linux with `git` preinstalled to download the `PulsarFeatureLab` software from GitHub (https://github.com/scienceguyrob/PulsarFeatureLab);

ii. create a directory inside the downloaded software to store the input and output data;

iii. create a Docker image based on Python 2.7;

iv. transfer the `PulsarFeatureLab` software into the Python 2.7 image; and

v. install `PulsarFeatureLab`'s library dependencies (`NumPy, SciPy, matplotlib` and `astropy`).

The above Docker image can now be built into a container (a virtual operating system) and a directory to hold the input data can be created by running the following commands on a UNIX terminal:

```
$ docker build -t jacobianm/pulsarfeaturelab:1.3.2 .
$ mkdir ~/pulsars/data/pfd
```

For future development of this project, the Docker image `jacobianm/pulsarfeaturelab:1.3.2` is available on Docker Hub (https://hub.docker.com) and will be automatically downloaded when running the `docker run` command below. Candidate `PFD` files of known pulsars and non-pulsars detected by the Murchison Widefield Array (MWA) can now populate the above created directory, and the following command can be ran to extract the features from the candidates:

```
$ docker run --rm -v ~/pulsars/data/pfd:/usr/src/app/
    PulsarFeatureLab/Data/IO jacobianm/pulsarfeaturelab:1.3.2
     -d "/usr/src/app/PulsarFeatureLab/Data/IO" -c 3 -t 6 -f
    "/usr/src/app/PulsarFeatureLab/Data/IO/output.arff" --
    arff --meta
```

This function instructs Docker to connect the directory containing the `PFD` files to the `PulsarFeatureLab` container's input/output directory and then run the `PulsarFeatureLab` software with arguments stating where the input files are, what filetype they are (`PFD`), which set of features to extract, and where to place the output file (`output.arff`).

Appendix A.B Developing PulsarValidator

We begin by creating a new Java project using the free software Maven (https://maven.apache.org) by running the following commands in a UNIX terminal:

```
$ cd ~/pulsars
$ mkdir PulsarValidator && cd PulsarValidator
$ mvn archetype:generate -DarchetypeGroupId=org.apache.maven.
    archetypes -DarchetypeArtifactId=maven-archetype-
    quickstart -DarchetypeVersion=1.4
```

We can then create the following file structure and resynchronise the project:

```
PulsarValidator/
    src/
        main/
            java/
                com/jacobianmatthews/pulsarvalidator/
                    PulsarValidator.java
        test/
            ...
    target/
            ...
    pom.xml
```

The file: `PulsarValidator.java` stands as the entry-point to the software and will be compiled into an executable `JAR` file upon completion of creating the software.

This software will process a user inputted `String` containing the path to a file with the list of pulsars included in the dataset classified by the machine learning classifier, a user inputted `String` containing the path to the `.positive` file created by the classifier, and a user inputted `String` containing the path to the `.negative` file created by the classifier[1]. These will be inputted as command-line arguments when the user runs the Java executable file.

To access the user inputted arguments, we can include the following function in the `main(String[] args)` method of the `PulsarValidator.java` class:

Algorithm 5 getCliVariables(args) (pseudocode)

> **for** Integer $i = 0 \rightarrow$ number of arguments in $args$ **do**
>   **if** ith argument in $args$ is "-v" **then**
>   Let Boolean $ValidationMode$ = true
>   Let String $pulsarListPath = (i + 1)$th argument in $args$
>   Let String $classifierPositive = (i + 2)$th argument in $args$
>   Let String $classifierNegative = (i + 3)$th argument in $args$
>   **end if**
> **end for**

We can then use a simple conditional statement after this function is ran to check if the `Boolean ValidationMode` has been set to `true`, to determine whether to continue the validation. The complete Java class `PulsarValidator.java` can be seen in *Appendix C.A*.

We can then create a new Java class, `ValidationMode.java`, with the following algorithm to validate the output of the classifier against the list of pulsars:

1. The '.positive' file contains the candidates classified as pulsars, and the '.negative' file contains the candidates classified as non-pulsars.

Algorithm 6 ValidationMode.java (pseudocode)

Let $truePositive$ = new List
Let $falsePositive$ = new List
Let $trueNegative$ = new List
Let $falseNegative$ = new List
for each $item$ in $classifierPositive$ list **do**
    Let Boolean $found$ = false
    Let Integer $i = 0$
    while $found$ is false **do**
        if ith item in $pulsarList$ is $item$ **then**
            add $item$ to $truePositive$ list
            $found$ = true
        else if i equals number of items in $pulsarList$ **then**
            add $item$ to $falsePositive$ list
            $found$ = true
        else
            i=$i + 1$
        end if
    end while
end for
for each $item$ in $classifierNegative$ list **do**
    Let Boolean $found$ = false
    Let Integer $i = 0$
    while $found$ is false **do**
        if ith item in $pulsarList$ is $item$ **then**
            add $item$ to $falseNegative$ list
            $found$ = true
        else if i equals number of items in $pulsarList$ **then**
            add $item$ to $truePositive$ list
            $found$ = true
        else
            i=$i + 1$
        end if
    end while
end for
Let TP = number of items in $truePositive$
Let TN = number of items in $trueNegative$
Let FP = number of items in $falsePositive$
Let FN = number of items in $falseNegative$
Let $Pulsars = TP + FN$
Let $NonPulsars = TN + FP$
Output $Pulsars$, $NonPulsars$, TP, FP, TN, FN

The complete Java class for `ValidationMode.java` can be found in *Appendix C.B*.

We can now compile the program by first adding the following lines of code to the `pom.xml` file at the root of the Maven project:

pom.xml

```
<project>
    ...
    <build>
    ...
    <pluginManagement>
        ...
        <plugins>
        ...
        <!-- Create a JAR containing the resources and
            dependencies -->
        <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
            <descriptorRefs>
                <descriptorRef>jar-with-dependencies</
                    descriptorRef>
            </descriptorRefs>
            <finalName>${project.artifactId}-${project.version
                }-full</finalName>
            <appendAssemblyId>false</appendAssemblyId>
            <archive>
                <manifest>
                <mainClass>com.jacobianmatthews.pulsarvalidator
                    .PulsarValidator</mainClass>
                </manifest>
            </archive>
            </configuration>
            <executions>
            <execution>
                <id>make-my-jar-with-dependenciess</id>
                <phase>package</phase>
                <goals>
                    <goal>single</goal>
                </goals>
            </execution>
            </executions>
        </plugin>
        </plugins>
    </pluginManagement>
    </build>
</project>
```

Which instructs Maven to create a single `JAR` file containing the program's dependencies and resources. The program can then be compiled and built by running the command:

```
$ mvn assembly:single
```

This will produce the file: `/target/pulsarvalidator-1.0-full.jar`, which is an executable Java program. The complete source code for `PulsarValidator` can be found in *Appendix C* or at
https://github.com/jacob-ian/PulsarValidator.git.

Appendix A.C Developing PulsarClassifier

To build the ensemble classification feature into the existing `LOTAASClassifier` tool, we can first begin by creating a new Java project named `PulsarClassifier` using the free software, Maven (https://maven.apache.org).

```
$ cd ~/pulsars
$ mkdir PulsarClassifier && cd PulsarClassifier
$ mvn archetype:generate -DarchetypeGroupId=org.apache.maven.
    archetypes -DarchetypeArtifactId=maven-archetype-
    quickstart -DarchetypeVersion=1.4
```

We can now copy the source code from `LOTAASClassifier` to be included in the `PulsarClassifier` software.

```
$ cp -R ~/pulsars/LOTAASClassifier/src/ ~/pulsars/
    PulsarClassifier/src/main/java
```

To use the WEKA suite of Machine Learning tools, we must then edit the `pom.xml` file inside `PulsarClassifier` to include it as a dependency, and resynchronise the project:

pom.xml

```
...
<dependencies>
...
    <dependency>
        <groupId>nz.ac.waikato.cms.weka</groupId>
        <artifactId>weka-stable</artifactId>
        <version>3.8.0</version>
    </dependency>
...
</dependencies>
...
```

We now have the following basic project directory structure:

```
PulsarClassifier/
    src/
        main/
            java/
                com/jacobianmatthews/pulsarclassifier/
                com/scienceguyrob/lotaasclassifier/
        test/
            java/
                com/jacobianmatthews/pulsarclassifier
    target/
            ...
    pom.xml
```

Where the new source code will be located under `/src/main/java/com/jacobianmatthews/pulsarclassifier`. To introduce the ensemble classification feature, we must write four main Java classes: `PulsarClassifier.java`, `ClassifierBuilder.java`, `ClassifierValidator.java`, and `ClassPredictor.java`.

The `LOTAASClassifier` tool accepts a command-line argument `-a` which accepts an integer that denotes the machine learning algorithm to use in building a classification model and making predictions (Lyon et al. 2016). Therefore, we will add an algorithm into the above listed Java classes that will accept an integer value of `-1` that will activate the ensemble classifier.

The class `ClassifierBuilder.java` handles training and building a classification model. To add ensemble classification to this class we will use the following algorithm:

Algorithm 7 ClassifierBuilder (pseudocode)

```
if algorithm = −1 then
    for each algorithm i = 1 → 4 do                ▷ Loop through all classifiers
        buildClassifier(i, trainingSet, modelsDirectory)
    end for
else                                               ▷ Build individual classifier
    buildClassifier(algorithm, trainingSet, modelPath)
end if
```

See *Appendix D.B* for the complete `ClassifierBuilder.java` class.

The class `ClassifierValidator.java` handles validating and testing the existing classification models. To implement the ensemble classifier into this class, we will use the following, similar algorithm:

Algorithm 8 ClassifierValidator (pseudocode)

```
if algorithm = −1 then
    for each algorithm i = 1 → 4 do                ▷ Loop through all classifiers
        testClassifier(i, testSet, modelsDirectory)
    end for
else                                               ▷ Test individual classifier
    testClassifier(algorithm, testSet, modelPath)
end if
```

See *Appendix D.C* for the complete `ClassifierValidator.java` class.

The class `ClassPredictor.java` handles making the classification predictions on new data using existing classifier models. We can add the ensemble classification feature to this class with the following algorithm:

Algorithm 9 ClassPredictor (pseudocode)

```
if algorithm = −1 then
    list = new OutputFileList()
    for algorithm i = 1 → 4 do                          ▷ Loop through all classifiers
        makePredictions(i, inputData, modelsDirectory)
        list.add(ClassifierOutputFiles)                  ▷ Add output filepaths to list
    end for
    positiveList = new ClassificationsList()
    negativeList = new ClassificationsList()            ▷ Create classification lists
    for each OutputFile from list do
        if OutputFile is .positive then
            for each line in OutputFile do
                positiveList.add(line)                   ▷ Add to +ve classifications
            end for
        else if OutputFile is .negative then
            for each line in OutputFile do
                negativeList.add(line)                   ▷ Add to -ve classifications
            end for
        end if
    end for
    for each classification in positiveList do
        if classification.occurrences < 3 then
            negativeList.add(classification)             ▷ cut-off at < 3 classifications
        else
            positiveOutput(classification)               ▷ Output classified as Pulsar
        end if
    end for
    for each classification in negativeList do
        negativeOutput(classification)                   ▷ Output classified as non-Pulsar
    end for
else
    makePredictions(algorithm)                           ▷ Use individual classifier
end if
```

The above algorithm preserves the individual classifiers' predictions and also makes an ensemble prediction based on all of the classifiers' predictions. According to Tan et al. 2017, it is common for ensemble machine learning classifiers to use a cut-off of three concurrent positive predictions in separate classifiers to make a positive ensemble classification. Therefore, for a candidate to be classified as a pulsar with the ensemble classifier it must first be classified as a pulsar by three of the underlying machine learning classifiers. See *Appendix D.D* for the complete `ClassPredictor.java` class.

The final class to add the ensemble classifier feature to is the entry point of the program, `PulsarClassifier.java`. This class only requires changes to the command-line inputs and outputs, so the completed Java class can be found in

Appendix D.A. The complete source code to the `PulsarClassifier` contains the following classes:

```
src/
    main/
        java/
            com/jacobianmatthews/pulsarclassifier/
                utils/
                    Classification.java
                    ClassificationList.java
                    Classifiers.java
                    Models.java
                PulsarClassifier.java
                ClassifierBuilder.java
                ClassifierValidator.java
                ClassPredictor.java

            com/scienceguyrob/lotaasclassifier/
                ...
```

Now that the source code for the package is complete, we can add the following lines to the `pom.xml` file at the root of the project:

pom.xml

```
<project>
    ...
    <build>
      <pluginManagement>
        <plugins>
          ...

          <!-- Create a JAR containing the resources and dependencies -->
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-assembly-plugin</artifactId>
            <configuration>
              <archive>
                <manifest>
                  <addClasspath>true</addClasspath>
                  <mainClass>com.jacobianmatthews.pulsarclassifier.
                       PulsarClassifier</mainClass>
                </manifest>
              </archive>
              <descriptorRefs>
                <descriptorRef>jar-with-dependencies</descriptorRef>
              </descriptorRefs>
            </configuration>
            <executions>
              <execution>
                <phase>package</phase>
                <goals>
                  <goal>single</goal>
```

```
            </goals>
          </execution>
        </executions>
      </plugin>

      ...
    </plugins>
  </pluginManagement>
</build>
...
</project>
```

This will instruct Maven to build a Java `JAR` file containing the package and its WEKA library dependency. To build the package, we can run the following commands in the UNIX terminal:

```
$ cd ~/pulsars/PulsarClassifier
$ mvn assembly:single
```

The complete source code and build of `PulsarClassifier` can be found at https://github.com/jacob-ian/PulsarClassifier.git, or in *Appendix D*.

Appendix A.D Using PulsarClassifier

We can use `PulsarClassifier` with the Murchison Widefield Array's (MWA) candidates by:

i. deleting the previous `output.positive` and `output.negative` files from the `PFD` candidates directory; and

ii. deleting the previous `model.m` file created by the `LOTAASClassifier` software.

We can now build the ensemble classification model with the training dataset created earlier, by running the following commands:

```
$ cd target
$ java -jar pulsarclassifier-1.0-jar-with-dependencies.jar -t
     ~/pulsars/data/trainingSet.arff -m ~/pulsars/data/models
     -a -1
```

Where the argument `-m` now denotes the path to a directory to store the various classifier models. Now that we have all of the classification models created, we can use the ensemble classifier to predict the class of new candidates. Using the feature extracted candidates compiled from *Section 2.2.2*, we can run the following command:

```
$ java -jar pulsarclassifier-1.0-jar-with-dependencies.jar -p
     ~/pulsars/data/pfd/output.arff -m ~/pulsars/data/models
    -a -1
```

`PulsarClassifier` will create the prediction output files for each classifier, and then the `output_ensemble.positive` and `output_ensemble.negative` files for the ensemble classifier.

Appendix A.E Using PulsarValidator

We can validate the ensemble classifier with the `PulsarValidator` program created in *Appendix A.B* by running the commands:

```
$ cd ~/pulsars/PulsarValidator/target
$ java -jar pulsarvalidator-1.0-full.jar -v ~/pulsars/data/
    pfd/pulsars.txt ~/pulsars/data/pfd/output_ensemble.
    positive ~/pulsars/data/pfd/output_ensemble.negative
```

The outputted validation statistics can then be compared to those created for the `LOTAASClassifier` algorithms and an opinion can be formed in regards to the utility of `PulsarClassifier` with the Murchison Widefield Array.

Appendix B Results

Subsection B.1 Machine Learning Training Dataset

trainingData.arff

```
@relation Pulsar_Feature_Data_Type_6
@attribute Feature_1 numeric
@attribute Feature_2 numeric
@attribute Feature_3 numeric
@attribute Feature_4 numeric
@attribute Feature_5 numeric
@attribute Feature_6 numeric
@attribute Feature_7 numeric
@attribute Feature_8 numeric
@attribute Feature_9 numeric
@attribute Feature_10 numeric
@attribute Feature_11 numeric
@attribute Feature_12 numeric
@attribute Feature_13 numeric
@attribute Feature_14 numeric
@attribute Feature_15 numeric
@attribute Feature_16 numeric
@attribute Feature_17 numeric
@attribute Feature_18 numeric
@attribute Feature_19 numeric
@attribute Feature_20 numeric
@attribute class {0,1,2}
@data
105.43918879794592,34.81562200057764,0.64091523062507,2.999965982775474,
    4.9731636,0.55617476,0.737705903258671,0.30015370463164537,
    0.2151762712941084,0.14214175185644565,-0.3166940138360127,
    -1.0084107149648813,0.17068485851918086,0.14475696370172492,
    1.632520131355678,4.059343847775194,47.576362204485356,
```

```
    29.306795113034152,0.05054613702000202,-1.2339760570632967,0,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1222697776_DM106.85_ACCEL_0_2_23_29_01.45_
    -10_35_14.03_8577.99ms_Cand_rfi.pfd
19.439267094649395,37.83110112790925,5.311899765657869,27.72625393618159,
    49.398266,17.023054,0.7582301777919424,-0.6690853550664837,
    0.7969783832836072,0.18564959248173177,-2.6111649508205854,
    7.75466957004941,0.8257942326119128,0.05310328864591472,
    -0.7025649619964046,-0.16942272159563032,49.637810769348576,
    24.759615998506728,0.004589172191101176,-0.7464438003865812,1,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1252780888_100_bins_PSR_0152-1637.pfd
18.18808199226157,29.34228539923202,5.972780846625737,41.45065223740317,
    31.696558,10.691088,0.632813129945887,-0.8541382878630457,
    0.7179793103088743,0.07111136671676592,-2.051028547817395,
    4.795786283258072,0.6697716962118456,0.0787967302217217,
    -0.591322615359726,-0.374506184841894,49.641593098999955,
    24.75138882693106,0.007109563095125316,-0.7563406788735434,1,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_0459-0210.pfd
21.22497781504873,33.50944234920476,4.928435748454679,26.392961286010472,
    19.536982,9.97172,1.0080046130377358,-0.01749195278819604,
    0.6288298612107428,0.14226525991343733,-0.07328453268261065,
    -0.7760081792257427,0.635532115110715,0.07569241239174013,
    -0.6454806403349908,-0.22201265419959704,63.29670555606886,
    27.73646944941322,0.5976574186441985,-0.7794129730580339,1,%/usr/src/app
    /PulsarFeatureLab/Data/IO/1253471952_100_bins_PSR_0255-5304.pfd
125.23524447122634,56.134362050023924,0.2229079772063587,-0.6216367448361071,
    11.638162,0.4708962,-0.9323500327291592,0.026974980798526893,
    0.2114757857992023,0.1690232851080658,0.058747599377539456,
    -0.5844300664122941,0.180508212537476,0.21259525842995097,
    -0.017456567184899085,-0.3139929597882456,50.040016795317,
    28.42321390118317,0.019428020518538806,-1.166645575344718,0,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1222697776_DM13.36_ACCEL_0_5_23_40_05.66_
    -13_34_55.98_591.37ms_Cand_noise.pfd
89.58174363355785,48.4446505060987,1.1788897000809406,2.1990474951218184,
    6.3838625,1.0415499,0.5775721247545433,-0.475129473762395,
    0.24557082776424885,0.09786188942598621,0.29275146998551704,
    -0.41424876112816467,0.2200571923781589,0.09503722553974074,
    -1.4319839206473817,2.8250537497993466,47.67179038646872,
    27.33291907433837,0.07145656830772441,-1.0385114275399896,1,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1252177744_100_bins_PSR_0151-0635.pfd
24.10863128259143,28.805247892957368,5.8820714970090355,41.4163908926057,
    26.94988,5.6278524,0.5874539163045516,-0.8389770182657368,
    0.4360563880324174,0.1297189968983776,-0.09540441239077253,
    -0.9301901127719643,0.3906387031999134,0.12027370376026593,
    -0.0529580678928215,-0.36995226959552374,47.80278524499236,
    26.565446325901135,0.0805551541059782,-0.9536043954878441,1,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1253991112_100_bins_PSR_0206-4028.pfd
19.659448914180896,33.71258916374586,6.212422938671684,38.92028743782727,
    33.78987,11.8825655,0.6139388470617142,-0.9493679705790381,
    0.7868883245856111,0.06691804191818253,-0.8309005284173321,
    -0.322551914444245,0.770521671609615,0.0403411973640893,
    0.04819999983836385,-0.9897646915402789,51.54462076437605,
    24.67525889451714,0.13313381803356525,-0.7286102199011202,1,%/usr/src/
    app/PulsarFeatureLab/Data/IO/1255803168_100_bins_PSR_0401-7608.pfd
130.37194427538753,48.5719656104399,0.03110721286401433,-0.28624231198914396,
    3.4748905,0.32001108,0.5444939189157489,-0.33239528639955696,
    0.16970626278252404,0.09649221205835012,0.4103571193662487,
    -0.7962143365852485,0.16034302263897418,0.08038131311036988,
```

```
      0.4063090998871781,0.35617007376255305,48.45058998307626,
      28.231337237194516,0.06120559028326374,-1.1489337808022277,0,%/usr/src/
      app/PulsarFeatureLab/Data/IO/1222697776_DM116.10_ACCEL_0_6_23_42_17.97_
      -12_35_31.39_3956.39ms_Cand_rfi.pfd
33  14.60760000762955,33.37521977816117,5.2251889142177985,29.534224184342584,
      80.282196,26.19628,0.5178206724750178,-0.9728259342130663,
      0.893477352431569,0.028687594940150146,-1.1385033438906254,
      0.7918283562790687,0.8729508030732867,0.04630670543612602,
      -1.661349783499672,3.460011989977854,49.58874503986989,
      24.792250514998546,0.01200259694803683,-0.7790806796208449,1,%/usr/src/
      app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_J0450-1248.pfd
34  19.160516821154644,52.612510321908914,3.498625248210871,10.92536155890088,
      221.91383,63.068035,0.46347635738061715,-1.085076006030645,
      0.9796727313010649,0.007708572312532982,-1.1449579309901678,
      0.6892826441998356,0.9615444721512544,0.018372881403763718,
      -1.2258505064734828,0.7964884426768761,49.70606769723089,
      25.32396221546309,0.006630948007153273,-0.8402679914296627,1,%/usr/src/
      app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_0452-1759.pfd
35  105.40739512137863,42.046982247914656,1.105401957592074,2.71606611188946,
      9.959542,1.508026,-0.10728643513973729,-0.575145272580543,
      0.21107789978405606,0.18110485489305306,1.2103695139475754,
      0.5497850271770104,0.14155672887756315,0.18398559172665105,
      2.090067439037396,5.155953247586126,45.394186429660124,28.416117397145,
      0.15258164708961688,-1.1267031372543894,0,%/usr/src/app/PulsarFeatureLab
      /Data/IO/1222697776_DM124.60_ACCEL_0_5_23_30_05.03_-15_32_28.86_7306.99
      ms_Cand_rfi.pfd
36  108.62704623126228,48.50808371245384,0.3888816698242244,0.10828707118695347,
      3.2382581,0.57282144,0.5061148662125752,-0.37190728802966433,
      0.16834501980712974,0.09483980211682852,0.5065252466218468,
      0.22607203657281394,0.1478631398839748,0.10268243782582831,
      1.2921170136328852,3.0823172664969274,45.7249804147162,
      28.856239988627337,0.11352983164931017,-1.1654127982083446,0,%/usr/src/
      app/PulsarFeatureLab/Data/IO/1222697776_DM144.10_ACCEL_0_4_23_41_42.76_
      -10_20_02.01_8566.62ms_Cand_rfi.pfd
37  30.65638094885709,36.39390906613557,4.334831666786916,20.661559138936177,
      22.137678,7.129584,0.5838741122318873,-0.7779836001168721,
      0.5429516337472768,0.068508294898713,-0.05956653663997695,
      -0.5072477532401791,0.46544379667343827,0.18186315104742065,
      -0.8913984152742472,0.23339557242434195,48.174632940592666,
      24.941747251772618,0.05804656648398697,-0.7838777635830874,1,%/usr/src/
      app/PulsarFeatureLab/Data/IO/1254594264_100_bins_PSR_0304+1932.pfd
38  110.46206868842188,46.251025957184034,0.37602640993908826,0.6004748040886194,
      3.5859008,0.48099223,0.9547631298163894,0.4135209351341804,
      0.20497692610421012,0.11970199393436176,0.07370434415143237,
      -0.48704911078586743,0.18721106211959027,0.11666707053309519,
      1.1853307807165687,2.25516233694235,50.58690926075272,28.58996541423567,
      0.08591965989680113,-1.1749014501039128,0,%/usr/src/app/PulsarFeatureLab
      /Data/IO/1222697776_DM133.85_ACCEL_0_4_23_34_32.65_-11_35_39.59_6359.79
      ms_Cand_rfi.pfd
39  67.96984095457957,30.670603367741087,2.9112873304761235,15.54008032933561,
      10.704723,3.0320911,-0.031139309413730974,-1.112557040322861,
      0.23665654949435502,0.2488763386952279,0.8534874503791406,
      -0.33346425705147986,0.14277266919223672,0.21040869818830255,
      2.339532390484435,6.038661020349938,41.44273829979434,27.623510996429545
      ,0.32711202934214134,-0.9930001536623023,0,%/usr/src/app/
      PulsarFeatureLab/Data/IO/1222697776_DM119.85_ACCEL_0_2_23_29_31.44_-15
      _47_03.39_13154.44ms_Cand_rfi.pfd
```

```
118.82100466581987,47.54490692614183,0.07781892865435884,-0.01638792599233385,3.4399548,0.8459567,0.7032480633089495,-0.7772255489180973,0.1697769818615784,0.06478195467823909,0.36310779421276496,0.7005773542260694,0.16145372201096284,0.10706223065327461,-0.1580528222963038,-0.45308023844697143,43.940007266955085,29.40433240665215,0.21689913203584288,-1.263495512374777,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM137.35_ACCEL_0_6_23_43_56.97_-11_20_36.48_5794.50ms_Cand_rfi.pfd
106.55367062734199,38.31922288316345,1.1201921873325247,3.524621849432644,4.6642995,1.4868807,1.2001879564459192,0.30185349985740917,0.21600790098220352,0.157004768317726,-0.03963724930253008,-0.6315962588469608,0.14498174340334713,0.17277641296712704,2.290547823032979,6.497604390941952,42.22435224003298,29.826089433097845,0.267641131812538,-1.2556978232502796,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM128.85_ACCEL_0_2_23_35_06.34_-10_20_02.01_13136.61ms_Cand_rfi.pfd
110.58047755262913,56.835988222404985,0.2597416198749472,-0.6009060312856693,6.2209997,1.3479991,0.03017693805887491,-1.488682979365772,0.21728005611823115,0.12862680601783838,-0.6671166056131778,0.02915403714241549,0.19370279062796897,0.1347179759873616,-0.27198972686586903,-0.36402365450393237,43.46834692311878,28.066122933832418,0.2827593499906235,-1.0752955947491496,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM124.10_ACCEL_0_12_23_36_13.35_-15_17_52.85_590.99ms_Cand_noise.pfd
22.63765924905709,32.25698418159722,5.017978110756053,28.558061328289384,23.729832,7.0639696,0.6582277084859545,-0.7967920282087153,0.6234012081321887,0.0553500395354285,-0.5533563848973001,-0.06756962085465013,0.5918865405107819,0.07053209744207622,0.043814399334462385,-0.09854663665519459,49.13623427277407,25.293472980718583,0.03635710958799211,-0.8209469526223003,1,%/usr/src/app/PulsarFeatureLab/Data/IO/1256407632_100_bins_PSR_J0450-1248.pfd
113.87621446459325,43.41151017852675,0.5893561441860531,1.71756136954156,5.65921,1.1683735,1.0193316046807013,1.4307229742168683,0.24277589148211937,0.13834994745613052,0.07107405638957241,-0.9824782006556352,0.16941307146894913,0.14966351742457934,2.164082062300479,5.916367297751334,44.50890374499167,28.802726541757913,0.15186962321586234,-1.15801903487134,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM119.35_ACCEL_0_2_23_29_35.58_-09_49_30.86_10373.16ms_Cand_rfi.pfd
116.41310084274251,46.52577061051363,0.1525172365016325,0.14280165135512535,3.0154004,0.65890557,1.4567341719163656,1.318452493060109,0.17635504213075298,0.13249954630159957,-0.19313971778172703,1.017867751573574,0.14723785581788873,0.12218666162430229,0.7585169054926933,2.0519787982530477,45.37813777753899,30.330805946058312,0.147494148971039,-1.304604403685838,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM106.85_ACCEL_0_2_23_37_20.08_-15_17_52.85_9862.48ms_Cand_rfi.pfd
115.05147692422577,59.943352711870936,0.06373023728528525,-0.5571994750116955,5.8320494,0.6752133,0.10013517067405343,-1.2867735020850521,0.2109214071604715,0.0983262916382064,-0.25916924236748073,-0.7968664315552898,0.19935938099018038,0.10038957671622305,0.9329047327135078,1.9245721329921315,46.236787709289835,28.703868432480217,0.14165640052355835,-1.1744718361297086,0,%/usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM114.60_ACCEL_0_6_23_42_50.44_-11_20_36.48_3514.62ms_Cand_rfi.pfd
13.688631621147106,29.885241623656917,6.142618064042354,42.30058250077879,42.656235,14.500364,0.6562798650382561,-0.8262654537167662,
```

```
0.836051931244822,0.023432197066059174,0.0226887983671618,
-0.6030241719786913,0.8133576587883088,0.03819319174623738,
-0.14087095518953835,-0.3893989970477949,49.34275912765738,
24.72845885609503,0.008595036799964983,-0.7534974947406083,?,%/usr/src/
app/PulsarFeatureLab/Data/IO/1256407632_100_bins_PSR_0459-0210.pfd
```

Appendix C Pulsar Validator

Appendix C.A PulsarValidator.java

PulsarValidator.java

```java
package com.jacobianmatthews.pulsarvalidator;

import java.io.IOException;

/**
 * This program is a validator for the machine learning software
 * PulsarClassifier.
 *
 * This program will generate a list of candidate filenames from an output file
 * of PulsarFeatureLab. It will then be possible to filter the list of known
 * pulsars in the original dataset to the pulsars that had successful feature
 * extraction. Finally, the program can compare the list of known pulsars in the
 * dataset to the pulsars classified in the classifier, and produce comparison
 * statistics.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 28/05/2020
 */
public class PulsarValidator {

    /**
     *
     * VARIABLES
     *
     */

    // Pulsar list mode
    private static boolean list = false;

    // Pulsar classification output validation mode
    private static boolean validation = false;

    // String containaing the path to the PulsarFeatureLab output file
    private static String pflOutput;

    // String containing the path to the list of pulsars
    private static String pulsarList;

    // String containing the path to the positive classifier output
    private static String positiveClassifier;

    // String containing the path to the negative classifier output
    private static String negativeClassifier;

    public static void main(String[] args) throws IOException {
        // Get the input arguments
        getCliVariables(args);

        // Check to see which flag has been given
```

```java
        if (list) {
            // LIST MODE CHOSEN
            // Print a message
            System.out.println("Pulsar list generation mode chosen.\n");

            // Create a list mode instance
            System.out.println("This feature is under development. Exiting program.");
            System.exit(0);

        } else if (validation) {

            // VALIDATION MODE CHOSEN
            // Print a message
            System.out.println("\nPulsar classifier validation mode chosen.");

            // Print the location of the pulsar list
            System.out.println("\nPulsar list location: "+pulsarList);

            // Print the location of the positive output file
            System.out.println("Classifier positive output location: "+positiveClassifier);

            // Print the location of the positive output file
            System.out.println("Classifier negative output location: "+negativeClassifier);

            // Create the validation mode instance
            ValidationMode validationMode = new ValidationMode(pulsarList, positiveClassifier, negativeClassifier);

            // Get the output string
            String output = validationMode.validate();

            // Output the string and then exit the program
            System.out.println("\nPulsar classifier validated successfully.\n");
            System.out.println(output);

            System.exit(0);

        } else if (list && validation) {

            // Display error that you can only do one thing at once
            System.out.println("'-l' and '-v' arguments entered. Please choose one mode only. \n");

            // Exit the application
            System.exit(0);

        } else if (!list && !validation) {

            // Display error that you need to pick a flag
            System.out.println("Please choose a mode by using this program with a '-l' or '-v' argument.\n");

            // Exit the application
```

```
            System.exit(0);
        }

    }

    /**
     * This function checks the command-line input arguments to decide how the
     * program should be ran.
     *
     * @param args
     */
    private static void getCliVariables(String[] args)
    {

        // Loop through arguments for the list and compare flags
        for(int i = 0; i<args.length; i++)
        {
            // Define the current argument
            String argument = args[i];

            // The list flag
            if( argument.equals("-l") ){

                // Set the list mode boolean to true
                list = true;

                // Get the next argument (the PulsarFeatureLab output file)
                pflOutput = args[i+1];

                // Get the input pulsar list
                pulsarList = args[i+2];

            // Check for compare flag
            } else if( argument.equals("-v") ){

                // Set the compare mode boolean to true
                validation = true;

                // Get the list of pulsars
                pulsarList = args[i+1];

                // Get the positive classifier output file
                positiveClassifier = args[i+2];

                // Get the negative classifier output file
                negativeClassifier = args[i+3];

            }
        }

    }
}
```

Appendix C.B ValidationMode.java

ValidationMode.java

```java
package com.jacobianmatthews.pulsarvalidator;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

import com.jacobianmatthews.pulsarvalidator.utils.StatisticList;
import com.jacobianmatthews.pulsarvalidator.utils.Utilities;

/**
 * This class contains the Pulsar Classifier validation mode of the program. It
 * will compare the output files of the Pulsar Classifier to the true list of
 * pulsars and output statistics.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 29/05/2020
 */
public class ValidationMode {

    /** VARIABLES */
    private String positiveOutputPath;

    private String negativeOutputPath;

    private List<String> pulsars;

    /** CONSTRUCTOR */
    /**
     * Instantiates the Validation mode.
     *
     * @param pulsarListPath A string containing the path to the list of pulsars
     * in the data set.
     * @param postiveOutputPath A string containing the path to the positive output
     * file of the classifier.
     * @param negativeOutputPath A string containing the path to the negative output
     * file of the classifier.
     * @throws IOException
     */
    public ValidationMode(String pulsarListPath, String positiveOutputPath, String negativeOutputPath)
            throws IOException {

        // Assign the variables
        this.positiveOutputPath = positiveOutputPath;
        this.negativeOutputPath = negativeOutputPath;

        // Validate the filepaths
```

```java
        if (!Utilities.isFile(pulsarListPath)) {
            // Print an error and end the program
            System.out.println("The path given to the pulsar list is not valid.\nExiting program.");
            System.exit(0);
        }

        if (!Utilities.isFile(positiveOutputPath)) {
            // Print an error and end the program
            System.out.println("The path given to the positive classifier output is not valid.\nExiting program.");
            System.exit(0);
        }

        if (!Utilities.isFile(negativeOutputPath)) {
            // Print an error and end the program
            System.out.println("The path given to the negative classifier output is not valid.\nExiting program.");
            System.exit(0);
        }

        // Get the list of pulsars
        this.pulsars = Files.readAllLines(Paths.get(pulsarListPath));

    };

    /**
     * This validates the classifier's outputs against the list of pulsars.
     *
     * @return String containg the validation statistics.
     * @throws IOException
     */
    public String validate() throws IOException
    {
        // Create a list of the statistics
        StatisticList statistics = new StatisticList();

        // Add the true and false positive statistics to the list
        StatisticList positiveStatistics = processPositive(statistics);

        // Add the true and false negative statistics to the list
        StatisticList allStatistics = processNegative(positiveStatistics);

        // Get the statistics
        int TP = allStatistics.getValueByName("TruePositives");
        int FP = allStatistics.getValueByName("FalsePositives");
        int TN = allStatistics.getValueByName("TrueNegatives");
        int FN = allStatistics.getValueByName("FalseNegatives");

        // To find the number of pulsars without generating a new list from the output data of
        // PulsarFeatureLab, we can just add the TruePositive count with the FalseNegative count.
        // The same can be applied for non-pulsars
        int pulsarCount = TP + FN;
        int nonpulsarCount = TN + FP;

```

```java
        int pulsarsDetected = TP + FP;
        int nonpulsarsDetected = TN + FN;

        // Output the statistics as a string
        String output = "Number of Pulsars: " + pulsarCount;
        output+= "\nPulsars Detected: " + pulsarsDetected;
        output+= "\nTrue Positives: " + TP;
        output+= "\nFalse Positives: " + FP;
        output+= "\n\nNumber of Non-Pulsars: " + nonpulsarCount;
        output+= "\nNon-Pulsars Detected: " + nonpulsarsDetected;
        output+= "\nTrue Negatives: " + TN;
        output+= "\nFalse Negatives: " + FN;

        // Return the output
        return output;

    }

    /**
     * This will create the true positive and false positive lists and output their
     * results
     *
     * @return a StatisticList object containing the postiive statistics
     * @throws IOException
     */
    private StatisticList processPositive(StatisticList statistics) throws IOException
    {
        // Create the True Positive list
        List<String> truePositive = new ArrayList<String>();

        // Create the False Positive list
        List<String> falsePositive = new ArrayList<String>();

        // Get the positive output file list
        List<String> classifier = Files.readAllLines(Paths.get(this.positiveOutputPath));

        // Check each line of the classifier's output
        for(String classification: classifier)
        {
            // Get the filename of the classification
            String name = classification.substring(classification.lastIndexOf("/")+1, classification.length());

            // Check if it is in the list of pulsars
            // Create search flag and index
            boolean found = false;
            int i = 0;

            // Loop through the list
            while(!found){

                // Check if the classified pulsar is in the real list
                if( this.pulsars.get(i).equals(name) ){

```

```java
                    // Add pulsar to the true positive list
                    truePositive.add(name);

                    // Set the flag to found
                    found = true;

                } else {
                    // Increment loop
                    i++;
                }

                // Check if we have reached the end of the list
                if ( !(i < this.pulsars.size()) ){

                    // Add this pulsar to the false positive list
                    falsePositive.add(name);

                    // End the loop
                    found = true;

                }
            }
        }

        // Count the number of true and false positives
        int truePositiveCount = truePositive.size();
        int falsePositiveCount = falsePositive.size();

        // Add the statistics to the list
        statistics.add("TruePositives", truePositiveCount);
        statistics.add("FalsePositives", falsePositiveCount);

        // Return the list
        return statistics;

    }

    /**
     * This creates the false and true negative lists and output the results of
     * their statistics
     *
     * @return a StatisticList containing the negative statistics
     * @throws IOException
     */
    private StatisticList processNegative(StatisticList statistics) throws IOException
    {
        // Create the False Negative List
        List<String> falseNegative = new ArrayList<String>();

        // Create the True Negative List
        List<String> trueNegative = new ArrayList<String>();

        // Get the negative output list
        List<String> classifier = Files.readAllLines(Paths.get(this.negativeOutputPath));
```

```

        // Loop through the negative output file
        for(String classification: classifier)
        {
            // Get the filename of the candidate
            String name = classification.substring(classification.lastIndexOf(
                 "/")+1,classification.length());

            // Create the loop variables
            boolean found = false;
            int i = 0;
            while(!found)
            {
                // Check if the name is in the list of pulsars
                if( this.pulsars.get(i).equals(name) ){
                    // Detected a false negative, therefore add to the list
                    falseNegative.add(name);

                    // End loop
                    found = true;
                } else {
                    // increment loop
                    i++;
                }

                // Check if the end of the pulsars list has been reached
                if ( !(i < this.pulsars.size() )){

                    // Add this classification to the true negatives list
                    trueNegative.add(name);

                    // End loop
                    found = true;
                }
            }
        }

        // Create the statistics
        int trueNegativeCount = trueNegative.size();
        int falseNegativeCount = falseNegative.size();

        // Add the statistics to the list
        statistics.add("TrueNegatives", trueNegativeCount);
        statistics.add("FalseNegatives", falseNegativeCount);

        // Return the statistics list
        return statistics;

    }


}
```

Appendix C.C utils/Statistic.java

utils/Statistic.java

```java
package com.jacobianmatthews.pulsarvalidator.utils;

/**
 * This is a data object to hold a statistic for the classifier.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 29/05/2020
 */
public class Statistic {

    /** VARIABLES */
    private String name;
    private int value;

    /** CONSTRUCTOR */
    public Statistic(String name, int value) {

        // Get the variables
        this.name = name;
        this.value = value;

    }

    /** GETTERS AND SETTERS */
    public void setValue(int value)
    {
        this.value = value;
    }

    public int getValue(){

        return this.value;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return this.name;
    }

}
```

Appendix C.D utils/StatisticList.java

utils/StatisticList.java

```java
package com.jacobianmatthews.pulsarvalidator.utils;

import java.util.ArrayList;
import java.util.List;

/**
 * A list to hold the classifier's statistics.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 29/05/2020
 */
public class StatisticList {

    /** VARIABLES */
    private List<Statistic> list;

    /** CONSTRUCTOR */
    public StatisticList() {

        // Create the list
        this.list = new ArrayList<Statistic>();

    }

    /**
     *
     * @param name
     * @param value
     */
    public void add(String name, int value)
    {
        // Create a statistic
        Statistic stat = new Statistic(name, value);

        // Add it to the list
        this.list.add(stat);
    }

    /**
     * Get the value of a statistic by its name
     * @param name String containing the name of the statistic
     * @return integer value of statistic
     */
    public int getValueByName(String name)
    {
        // Loop through the list
        for(Statistic stat: this.list)
        {
            if( stat.getName().equals(name) ){

                // Return the value
                return stat.getValue();
            }
```

```
        }

        // Couldn't find it, return 0
        return 0;
    }

}
```

Appendix C.E utils/Utilities.java

utils/Utilities.java

```java
package com.jacobianmatthews.pulsarvalidator.utils;

import java.io.File;

/**
 * This class contains the common utility functions across the program.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 29/05/2020
 */
public class Utilities {

    /** CONSTRUCTOR */
    public Utilities(){
        // Empty
    }

    /**
     * This method validates that a file exists.
     * @param path A string containing the path to a file.
     * @return True if it is a valid, existing file.
     */
    public static boolean isFile(String path)
    {
        // Trim the whitespace
        String trimmed = path.trim();

        // Create the file
        File file = new File(trimmed);

        // Validate the file
        if(file.isFile()){

            // Return true
            return true;
        } else {

            // Return false;
            return false;

        }

    }
}
```

Appendix D Pulsar Classifier

Appendix D.A PulsarClassifier.java

PulsarClassifier.java

```
package com.jacobianmatthews.pulsarclassifier;

import com.scienceguyrob.lotaasclassifier.classifiers.Classifiers;
import com.scienceguyrob.lotaasclassifier.cli.CLI;
import com.scienceguyrob.lotaasclassifier.cli.CLParameter;
import com.scienceguyrob.lotaasclassifier.cli.ICLI;
import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
import com.scienceguyrob.lotaasclassifier.utils.Common;

import java.net.URL;


/**
 *This class takes in command-line arguments and starts the PulsarClassifier
 *program.
 *
 *It is a derivative of the com.scienceguyrob.lotaasclassifier package.
 *
 *@author Jacob Ian Matthews & Rob Lyon
 *
 *@version 1.0, 22/5/2020
 */
public class PulsarClassifier {
    /**
     *
     * VARIABLES
     *
     */

    /**
     *Full path to the file containing training data.
     */
    private static String training_path = "";

    /**
     *Full path to the file (ARFF file) to be assigned classifier predictions.
     */
    private static String predict_path = "";

    /**
     *Full path to the classifier validation file (ARFF file).
     */
    private static String validate_path = "";

    /**
     *Full path to the file containing the classification model to use.
     */
    private static String model_path = "";

    /**
     *The algorithm to train/use to make predictions.
```

```java
     */
    private static int algorithm = -1;

    /**
     *Logging flag, if true, verbose logging outputs will be written to standard out.
     */
    private static boolean verbose = false;

    /**
     *Training flag. If true the system will attempt to build a new classifier.
     */
    private static boolean train = false;

    /**
     *Prediction flag. If true the system will attempt to classify data.
     */
    private static boolean predict = false;

    /**
     *Validation flag. If true, the system will attempt to validate the classifiers performance.
     */
    private static boolean validate = false;

    /**
     *The working directory for this code.
     */
    private static URL workingDir = PulsarClassifier.class.getProtectionDomain().getCodeSource().getLocation();

    /**
     *The object used to output debug/logging information.
     */
    private static BasicLogger log = new BasicLogger(verbose,workingDir.getFile().replace(".jar",".log"));

    //****************************************
    //****************************************
    // Main Method
    //****************************************
    //****************************************

    /**
     *The main entry point to the application.
     *@param args the command line arguments.
     */
    public static void main(String[] args)
    {
        processCommandLine(args);

        /**
         *Run desired commands...
         */

        if(!train & !predict & !validate)
```

```
{
    log.sout("Unable to train classifier model/make predictions - inputs invalid", true);
    safeExit();
}
else if(train & predict)
{
    log.sout("Valid training and classification inputs provided - system unsure what to do", true);
    safeExit();
}
else if(train)
{
    log.sout("Attempting to build a new classifier", true);
    ClassifierBuilder cb = new ClassifierBuilder(log,"ClassifierBuilder");

    // Actually build the ensemble classification system
    boolean result = cb.build(algorithm,training_path,model_path);

    // Check the results of the classifier build
    if( algorithm == -1 )
    {
        if (result)
        {
            log.sout("Ensemble classifier built successfully", true);

        } else {

            log.sout("Ensemble classifier couldn't be built.", true);
        }

    } else {
        if (result)
        {
            log.sout(Classifiers.getClassifierName(algorithm)+" classifier built successfully", true);

        } else {
            log.sout(Classifiers.getClassifierName(algorithm)+" classifier couldn't be built.", true);
        }
    }

}
else if(predict)
{
    log.sout("Attempting to apply predictions using existing classifier", true);

    ClassPredictor cp = new ClassPredictor(log,"ClassPredictor");

    // Actually build the classification system
    boolean result = cp.predict(algorithm,predict_path,model_path);

    // Log the results back to the user
    if ( algorithm == -1 )
```

```
            {
                if ( result )
                {
                    log.sout("Ensemble classifier made predictions successfully.", true);
                } else {
                    log.sout("Ensemble classifier was unsuccessful in applying predictions.", true);
                }
            } else {

                if( result )
                {
                    log.sout(Classifiers.getClassifierName(algorithm)+ " classifier made predictions successfully", true);

                } else {
                    log.sout(Classifiers.getClassifierName(algorithm)+ " classifier unsuccessful in applying predictions", true);
                }

            }


        }
        else if(validate)
        {
            log.sout("Attempting to validate existing classifier performance", true);

            ClassifierValidator cv = new ClassifierValidator(log,"ClassifierValidator");

            // Validate the classifier chosen
            boolean result = cv.validate(algorithm,validate_path,model_path);

            // Output the results for the ensemble classifier
            if(algorithm == -1) {
                if(result) {
                    log.sout("Ensemble classifier validated successfully.", true);
                } else {
                    log.sout("Ensemble classifier couldn't be valdated successfully.", true);
                }

            // Output the results for the individual classifiers
            } else {
                if(result) {
                    log.sout(Classifiers.getClassifierName(algorithm)+ " classifier validated successfully", true);
                } else {
                    log.sout(Classifiers.getClassifierName(algorithm)+ " classifier validation unsuccessful", true);
                }
            }
```

```java

        }

        /**
         *When done...
         */
        safeExit();
    }

    /**
     *Command line processing methods.
     */

    /**
     *Processes the command line parameters.
     *@param args the command line arguments
     */
    public static void processCommandLine(String[] args)
    {
        // Print some details, help etc to command line.
        printApplicationDetails();

        // Always make sure to write an app start message to the log file.
        printLogFileHeader();

        // Set logging to false before processing user input
        verbose = false;

        System.out.println( "\nReading Terminal Parameters...\n" );

        ICLI cli = getCommandlineOptions();
        cli.processArguments(args);

        // Update local variables based on user parameters.
        updateVariables(cli);
        printParameters();
    }

    /**
     *Updates class variables with user input parameters, if provided.
     *@param cli the command line parameters to use to obtain user input.
     */
    private static void updateVariables(ICLI cli)
    {
        /**
         *There are two main modes for the application. Either build a classifier, or
         *classify new data.
         *
         *Option one requires the full path to a training set file, the integer identifier
         *of the classifier to build, and the path to write the classification model to.
         *
         *Option two requires the full path to a file requiring predictions, the integer
```

```
         *identifier of the classifier used to make the predictions, and the path to the
         *classification model.
         *
         *Below we make sure this logic is correct.
         */

        // First deal with general simulation wide variables.
        if(cli.hasParameter(FLAG_VERBOSE))
            verbose = true;
        else
            verbose = false;

        // Check the algorithm supplied is valid, if not return since there's
        // no point checking the other variables.
        if (cli.hasParameter(FLAG_ALGORITHM))
            if (cli.getParameter(FLAG_ALGORITHM).toInt() > -2 && cli.getParameter(FLAG_ALGORITHM).toInt() < 6)
                algorithm = cli.getParameter(FLAG_ALGORITHM).toInt();
            else
            {
                log.sout("Algorithm value supplied via -a flag invalid (must be -1 to 5).", true);
                return;
            }

        if(cli.hasParameter(FLAG_TRAINING)) // if a training set has been provided.
        {
            // Try to load training data path (used for ML classification).
            if (cli.hasParameter(FLAG_TRAINING))
            {
                training_path = cli.getParameter(FLAG_TRAINING).getValue();

                if (Common.fileExist(training_path)) // If the training set is valid.
                {
                    if (cli.hasParameter(FLAG_MODEL))
                    {
                        model_path = cli.getParameter(FLAG_MODEL).getValue();

                        if (Common.isPathValid(model_path)) // If the training set is valid.
                            train = true;
                        else
                            log.sout("Model output path specified via -m flag invalid",true);
                    }
                    else
                        log.sout("No output model path supplied with -m flag", true);
                }
                else
                    log.sout("No machine learning training data supplied via -t flag",true);
            }
        }
```

```
        else if(cli.hasParameter(FLAG_PREDICT)) // if a training set has been provided.
        {
            // Try to load training data path (used for ML classification).
            if (cli.hasParameter(FLAG_PREDICT))
            {
                predict_path = cli.getParameter(FLAG_PREDICT).getValue();

                if (Common.fileExist(predict_path)) // If the file to classify is valid.
                {
                    if (cli.hasParameter(FLAG_MODEL))
                    {
                        model_path = cli.getParameter(FLAG_MODEL).getValue();

                        if (Common.isPathValid(model_path)) // If the model path is valid.
                            predict = true;
                        else
                            log.sout("Cannot load the classifier model via the -m flag",true);
                    }
                    else
                        log.sout("Cannot load the classifier model via the -m flag",true);
                }
                else
                    log.sout("No data to be classified supplied via -s flag",true);
            }
        }
        else if(cli.hasParameter(FLAG_VALIDATE)) // if a validation set has been provided.
        {
            // Try to load validation data path.
            if (cli.hasParameter(FLAG_VALIDATE))
            {
                validate_path = cli.getParameter(FLAG_VALIDATE).getValue();

                if (Common.fileExist(validate_path)) // If the file to classify is valid.
                {
                    if (cli.hasParameter(FLAG_MODEL))
                    {
                        model_path = cli.getParameter(FLAG_MODEL).getValue();

                        if (Common.isPathValid(model_path)) // If the model path is valid.
                            validate = true;
                        else
                            log.sout("Cannot load the classifier model via the -m flag",true);
                    }
                    else
                        log.sout("Cannot load the classifier model via the -m flag",true);
                }
```

```
                else
                    log.sout("No data to be used for validation supplied via -validate flag",true);
            }
        }
    }

    /**
     *@return the command line options for this application.
     */
    private static ICLI getCommandlineOptions()
    {
        ICLI cli = new CLI();

        cli.addParameter(FLAG_VERBOSE,
                "Verbose logging flag (optional, logging off by default)",
                CLParameter.BOOL_PARAM_TYPE);

        cli.addParameter(FLAG_VALIDATE,
                "The path to the validation data to use to build a classifier (required).",
                CLParameter.FILEPATH_PARAM_TYPE);

        cli.addParameter(FLAG_TRAINING,
                "The path to the training data to use to build a classifier (required).",
                CLParameter.FILEPATH_PARAM_TYPE);

        cli.addParameter(FLAG_MODEL,
                "The path to the classification model to load/create (required).",
                CLParameter.FILEPATH_PARAM_TYPE);

        cli.addParameter(FLAG_PREDICT,
                "The path to the observational data to label (required).",
                CLParameter.FILEPATH_PARAM_TYPE);

        cli.addParameter(FLAG_ALGORITHM,
                "The algorithm to use (required).",
                CLParameter.INT_PARAM_TYPE);

        return cli;
    }

    /**
     *Prints input parameters to the command line.
     */
    private static void printParameters()
    {
        String details = "\nAPPLICATION PARAMETERS\n";
        details += "Verbose logging : " + verbose + "\n";
        details += "Training set path : " + training_path + "\n";
        details += "Prediction path : " + predict_path + "\n";
        details += "Validation set path: " + validate_path + "\n";
        details += "Model path : " + model_path + "\n";
        details += "Algorithm : " + algorithm + "\n";
        System.out.println(details);
```

```
    }

    /**
     *Prints application details when beginning execution.
     */
    private static void printApplicationDetails()
    {
        String details = "\n";
        details += "****************************************************************************\n";
        details += "| |\n";
        details += "| PULSAR CLASSIFIER v1.0 |\n";
        details += "| |\n";
        details += "****************************************************************************\n";
        details += "| Description: |\n";
        details += "| |\n";
        details += "| A machine learning pulsar classification program derived from Rob. |\n";
        details += "| Lyon's LOTAASClassifier v1.0. Can be used in ensemble classification |\n";
        details += "| or individual classification modes. This software can create |\n";
        details += "| classification models and make predictions on data using those models. |\n";
        details += "| Requires Java 1.6 or later to run. |\n";
        details += "| |\n";
        details += "****************************************************************************\n";
        details += "| Author: Jacob Ian Matthews & Rob Lyon |\n";
        details += "| Email : jacob@jacobian.com.au |\n";
        details += "| web : jacobianmatthews.com |\n";
        details += "****************************************************************************\n";
        details += "| Required Command Line Arguments: |\n";
        details += "| |\n";
        details += "| Training mode (builds a new classifier model): |\n";
        details += "| |\n";
        details += "| -t (path) path to a file containing training data in ARFF format. |\n";
        details += "| This is used to train the machine learning classifier that |\n";
        details += "| assigns predicted candidate labels. |\n";
        details += "| |\n";
        details += "| -m (path) output directory for the created model. |\n";
        details += "| |\n";
        details += "| -a (int) the learning algorithm to build a model for. There are some |\n";
        details += "| possible choices listed below: |\n";
        details += "| |\n";
        details += "| -1 = Ensemble Classifier (builds all algorithms) |\n";
        details += "| 1 = J48 decision tree |\n";
        details += "| 2 = Multilayer perceptron (neural network) |\n";
        details += "| 3 = Naive Bayes |\n";
        details += "| 4 = Support vector machine |\n";
        details += "| |\n";
```

```
        details += "| Prediction mode (applies the classifier to new data): |\n";
        details += "| |\n";
        details += "| -m (path) path to the models directory, describing the pre-built |\n";
        details += "| classifier to use. The model must have been built using |\n";
        details += "| this tool or WEKA. |\n";
        details += "| |\n";
        details += "| -p (string) path to a file containing unlabelled data in ARFF format. |\n";
        details += "| The model loaded in via the -m flag will apply predicted |\n";
        details += "| labels to the data in this file. |\n";
        details += "| |\n";
        details += "| -a (int) the learning algorithm stored in the model. |\n";
        details += "| possible choices listed below: |\n";
        details += "| |\n";
        details += "| -1 = Ensemble Classifier |\n";
        details += "| 1 = J48 decision tree |\n";
        details += "| 2 = Multilayer perceptron (neural network) |\n";
        details += "| 3 = Naive Bayes |\n";
        details += "| 4 = Support vector machine |\n";
        details += "| |\n";
        details += "| Validation mode (checks a new classifier model): |\n";
        details += "| |\n";
        details += "| -m (path) path to the models directory, describing the pre-built |\n";
        details += "| classifier to use. The models must have been built using |\n";
        details += "| this tool or WEKA. |\n";
        details += "| |\n";
        details += "| -v (string) path to a file containing labelled data in ARFF |\n";
        details += "| format. The model loaded in via the -m flag will then be |\n";
        details += "| tested against the labels in the file. |\n";
        details += "| |\n";
        details += "| -a (int) the learning algorithm stored in the model. |\n";
        details += "| possible choices listed below: |\n";
        details += "| |\n";
        details += "| -1 = Ensemble Classifier |\n";
        details += "| 1 = J48 decision tree |\n";
        details += "| 2 = Multilayer perceptron (neural network) |\n";
        details += "| 3 = Naive Bayes |\n";
        details += "| 4 = Support vector machine |\n";
        details += "| |\n";
        details += "***************************************************************************\n";
        details += "| Optional Command Line Arguments: |\n";
        details += "| |\n";
        details += "| -d (boolean) verbose debugging flag. |\n";
        details += "| |\n";
        details += "***************************************************************************\n";
```

```
            details += "| |\n";
            details += "| EXAMPLE USAGE: |\n";
            details += "| |\n";
            details += "| java -jar LotaasClassifier.jar -a -1 -t /my/file.arff -m /my/models/ |\n";
            details += "| |\n";
            details += "| This would build an ensemble classifier using the supplied training |\n";
            details += "| set with the 'learned' models written to /models/ |\n";
            details += "| |\n";
            details += "*************************************************************************\n";
            details += "| License: |\n";
            details += "| |\n";
            details += "| Code made available under the GPLv3 (GNU General Public License), that |\n";
            details += "| allows you to copy, modify and redistribute the code as you see fit |\n";
            details += "| (http://www.gnu.org/copyleft/gpl.html). Though a mention to the |\n";
            details += "| original author using the citation above in derivative works, would be |\n";
            details += "| very much appreciated. |\n";
            details += "*************************************************************************\n";

            System.out.println(details);
        }

        /**
         *Prints application details to the log file.
         */
        private static void printLogFileHeader()
        {
            log.setVerbose(true);

            if(log != null)
                log.sout("Welcome to PULSAR CLASSIFIER 1.0",true);
            else
            {
                System.out.println("Log file cannot be initialised, exiting...");
                safeExit();
            }
        }

        /**
         *Safely exits the application and updates the log.
         */
        private static void safeExit()
        {
            log.setVerbose(true);
            log.sout("Exiting PULSAR CLASSIFIER 1.0 correctly",true);
            System.exit(0);
        }

        /**
         *The command line flags...
```

```
     */

    private static String FLAG_VERBOSE = "-d";
    private static String FLAG_VALIDATE = "-v";
    private static String FLAG_TRAINING = "-t";
    private static String FLAG_PREDICT = "-p";
    private static String FLAG_MODEL = "-m";
    private static String FLAG_ALGORITHM = "-a";
}
```

Appendix D.B ClassifierBuilder.java

ClassifierBuilder.java

```java
package com.jacobianmatthews.pulsarclassifier;

import com.jacobianmatthews.pulsarclassifier.utils.Models;
import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.NaiveBayesTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;


/**
 * This file is intended to build a Classification Model in four different Machine Learning algorithms
 * from a training data set provided by the user. It can be completed in ensemble (all classifiers) by using
 * algorithm=-1, or individually by using their respective integers.
 *
 * @author Jacob Ian Matthews
 * Contact: jacob@jacobian.com.au
 * @version 1.0, 22/05/2020
 *
 */

/* Create the Class */
public class ClassifierBuilder extends com.scienceguyrob.lotaasclassifier.mvc.ClassifierBuilder {

    // CONSTRUCTORS
    public ClassifierBuilder(BasicLogger l, String n) {
        super(l, n);
    }

    public ClassifierBuilder(String n) {
        super(n);
    }

    /**
     * Builds the classifier for all algorithms with the training data set.
     *
     * @param algorithm set to -1 for all algorithms, otherwise use original
     * integers for individual algorithms.
     * @return true if all classifiers have been successfully tested and trained.
     */
    public boolean build(int algorithm, String trainingSetPath, String modelPath) {

        // Check if user requested the ensemble classifier
        if (algorithm == -1) {

            // Create success count variable
```

```
            int successCount = 0;

            // All algorithms selected therefore loop through training and
                 testing them all
            for (int classifier : Classifiers.classifiers) {
                // Check if the result of building the selected classifier
                     returns true
                if (buildClassifier(classifier, trainingSetPath, modelPath)) {
                    // Add to the successCount as it was successful
                    successCount++;
                }
            }

            // Check to see if all classifiers have built successfully
            if (successCount == Classifiers.classifiers.length) {
                // All classifiers have been built successfully, therefore
                     return true
                return true;

            } else {

                // Not all classifiers were successful, therefore return false
                return false;
            }

        } else {

            // Not using the ensemble classifier, build the individual
                 classifier and return the result
            return buildClassifier(algorithm, trainingSetPath, modelPath);

        }


    }
    /**
     * This method will call the trainAndTest method on the algorithm/
          classifier desired.
     *
     * @param algorithm integer corresponding to the classifier algorithm
     * @param trainingSetPath String corresponding to the filepath of the
          training data set
     * @param modelPath String corresponding to the directory of the
          classifier models to be outputted
     * @return true if the selected classifier to be built is built
          successfully
     */
    private boolean buildClassifier(int algorithm, String trainingSetPath,
         String modelDir) {
        switch (algorithm) {
            case Classifiers.J48:
                return trainAndTest(new J48Tester(log, "J48Tester"),
                     trainingSetPath,
                        Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.MLP:
                return trainAndTest(new MLPTester(log, "MLPTester"),
                     trainingSetPath,
```

```
                    Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.NB:
                return trainAndTest(new NaiveBayesTester(log, "
                    NaiveBayesTester"), trainingSetPath,
                    Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.SVM:
                return trainAndTest(new SVMTester(log, "SVMTester"),
                    trainingSetPath,
                    Models.getModelFilePath(algorithm, modelDir));
            default:
                return false;
        }
    }

}
```

Appendix D.C ClassifierValidator.java

ClassifierValidator.java

```java
package com.jacobianmatthews.pulsarclassifier;

import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
import com.jacobianmatthews.pulsarclassifier.utils.Models;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.NaiveBayesTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
import com.scienceguyrob.lotaasclassifier.wekawrappers.I_WekaTest;
/**
 * This class validates the ensemble and individual classifiers on a
 * provided data set.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 22/05/2020
 */
public class ClassifierValidator extends com.scienceguyrob.lotaasclassifier.mvc.ClassifierValidator {

    // CONSTRUCTORS
    public ClassifierValidator(BasicLogger l, String n) {
        super(l, n);
    }

    public ClassifierValidator(String n){ super(n);}

    /**
     *
     * @param algorithm integer corresponding to the algorithm
     * @param validationPath String with the filepath of the data to test the classifier on
     * @param modelsDir String with the path of the directory containing the classifier models
     * @return true if the classifier(s) have been successfully tested
     */
    public boolean validate(int algorithm, String validationPath, String modelsDir)
    {
        // Check if using the ensemble classifier
        if( algorithm == -1 ){

            // Create a counter to keep track of the number of successful validations
            int successCount = 0;

            // Loop through the array of classifiers to validate them all
            for(int classifier: Classifiers.classifiers)
            {
                // Check for the result of the validation
                if( chooseClassifier(classifier, validationPath, modelsDir) )
                {
```

```java
                    // Successful, therefore add to the counter
                    successCount++;
                }
            }

            // Check if all classifiers validated successfully
            if(successCount == Classifiers.classifiers.length)
            {
                // Return true, as validation was successful
                return true;
            } else {
                // One or validations were unsuccessful, return false
                return false;
            }

        } else {

            // Make predictions on the chosen classifier and return the result
            return chooseClassifier(algorithm, validationPath, modelsDir);

        }

    }

    /**
     *
     * @param algorithm integer corresponding to the classifier.
     * @param validationPath String containing the filepath of the testing data
     * @param modelDir String containing the directory containing the classifier models
     * @return
     */
    private boolean chooseClassifier(int algorithm, String validationPath, String modelDir)
    {
        // Check which classifier needs to be validated and validate it
        switch (algorithm)
        {
            case Classifiers.J48:
                return performValidation(new J48Tester(log,"J48Tester"), algorithm, validationPath, Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.MLP:
                return performValidation(new MLPTester(log,"MLPTester"), algorithm, validationPath, Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.NB:
                return performValidation(new NaiveBayesTester(log,"NaiveBayesTester"), algorithm, validationPath, Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.SVM:
                return performValidation(new SVMTester(log,"SVMTester"), algorithm, validationPath, Models.getModelFilePath(algorithm, modelDir));
            default:
                return false;
```

```
        }

    }

    /**
     *
     * @param classifier a WEKA classifier
     * @param algorithm integer corresponding to the chosen algorithm
     * @param validationPath string containing the path to the data to test
         with
     * @param modelPath string containing the path to the classifier's model
     * @return
     */
    private boolean performValidation(I_WekaTest classifier, int algorithm,
         String validationPath, String modelPath)
    {
        // Check for successful loading and validation of the model
        boolean loaded = classifier.loadModel(modelPath);
        boolean validated = classifier.validate(validationPath);

        // Get the name of the classifier
        String classifierName = Classifiers.getClassifierName(algorithm);

        // Output the results of the validations
        if(loaded & validated)
            return true;
        else
        {
            if(!loaded)
            {
                log.sout("Could not load the "+classifierName+" classifier 
                     model",true);
                return false;
            }
            else if(!validated)
            {
                log.sout("Could not validate the "+classifierName+" model",
                     true);
                return false;
            }
            else
            {
                log.sout("Could not perform validation on the "+classifierName
                     +" classifier model",true);
                return false;
            }
        }
    }

}
```

Appendix D.D ClassPredictor.java

ClassPredictor.java

```java
package com.jacobianmatthews.pulsarclassifier;

import java.util.ArrayList;
import java.util.List;
import com.jacobianmatthews.pulsarclassifier.utils.Classification;
import com.jacobianmatthews.pulsarclassifier.utils.ClassificationList;
import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
import com.jacobianmatthews.pulsarclassifier.utils.Models;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.
     NaiveBayesTester;
import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
import com.scienceguyrob.lotaasclassifier.io.Writer;
import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
import com.scienceguyrob.lotaasclassifier.wekawrappers.I_WekaTest;

/**
 * This class contains the methods and properties required to make predictions
       on data
 * to classify pulsars.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 24/05/20
 */
public class ClassPredictor extends com.scienceguyrob.lotaasclassifier.mvc.
     ClassPredictor {

    // CONSTRUCTORS
    public ClassPredictor(BasicLogger l, String n) { super(l, n); }
    public ClassPredictor(String n){ super(n); }


    public boolean predict(int algorithm, String predictPath, String
         modelsDir)
    {
        // Check if the ensemble classifier is being used
        if ( algorithm == -1 )
        {
            // Get the prefix of the prediction output files (input file path
                 without extension)
            String fileName = predictPath.substring(0, predictPath.lastIndexOf
                 ("."));

            // Create an array of the [output].positive and [output].negative
                 files
            List<String> positiveFiles = new ArrayList<String>();
            List<String> negativeFiles = new ArrayList<String>();

            // Create an index to count the number of classifiers successfully
                 completing predictions
            int predictCount = 0;

```

```java
        // Loop through the array of classifiers to make all predictions
        for(int classifier: Classifiers.classifiers)
        {
            // Check the status of the classifier's predictions
            if ( chooseClassifier(classifier, predictPath, modelsDir) )
            {
                // Returned true, therefore add to prediction count
                predictCount++;
            }

            // Get the name of the classifier
            String classifierName = Classifiers.getClassifierName(
                 classifier);

            // Re-create the output file's name
            String outputName = fileName+"_"+classifierName;

            // Add the output file's name to the array of file names
            positiveFiles.add(outputName+".positive");
            negativeFiles.add(outputName+".negative");

        }

        // Check that all classifiers completed predictions
        if ( predictCount == Classifiers.classifiers.length )
        {
            // All classifiers have finished making predictions, now we can
                  create the final
            // classification output file by combining the positive pulsar
                 predictions of all classifiers.
            // According to Tan et al. (2017), having positive
                 classifications in 3 separate classifiers
            // indicates a more accurate classification.

            // Create the ensemble classifier's output file names
            String ensemblePositive = fileName+"_ensemble.positive";
            String ensembleNegative = fileName+"_ensemble.negative";

            // Create some flags to denote the ensemble classification
                 process completed successfully
            boolean positiveSuccess = false;
            boolean negativeSuccess = false;

            // Create a list containing the postitive classifications and a
                  count of their occurrences
            ClassificationList positiveList = new ClassificationList();

            // Create a list containing the postitive classifications and a
                  count of their occurrences
            ClassificationList negativeList = new ClassificationList();

            // Build the list of positive classifications using the outputs
                  from all the classifiers
            if( positiveList.buildList(positiveFiles) ){

                // We now have a list of positive candidate classifications
                      and we can apply
```

```
            // the cutoff of 3 separate positive classifications for the ensemble classifier.

            // Loop through the list of positive classifications
            for( Classification positive: positiveList.getList() )
            {
                // Check the classification key-value pair for a value >=3
                if( positive.getValue() > 2 )
                {
                    // Try add it to the ensemble positive output file
                    if ( !Writer.append(ensemblePositive, positive.getKey()+"\n") )
                    {
                        log.sout("Couldn't add "+positive.getKey()+" to positive ensemble classifier output file.", true);
                    }
                } else {

                    // Add it to the negative classification list as it didn't survive the cutoff
                    negativeList.add(positive.getKey(), positive.getValue());
                }

            }
            // Flag that the process was successful
            positiveSuccess = true;

        } else {

            // Return false and log the error
            log.sout("Ensemble classifier positive candidates list couldn't be compiled.", true);
            return false;
        }

        // Build a list of negative classifications from all classifiers
        if ( negativeList.buildList(negativeFiles) )
        {
            // Loop through the list to produce the negative output file
            for (Classification negative: negativeList.getList() )
            {
                // Get the key of the classification
                String key = negative.getKey();

                // Make sure that it isn't already in the positive or negative output files
                if( !Reader.checkStringIsInFile(ensembleNegative, key) && !Reader.checkStringIsInFile(ensemblePositive, key))
                {
                    // Append it to the output file
                    if( !Writer.append(ensembleNegative, key+"\n") )
```

```
                            {
                                // Log the error
                                log.sout("Couldn't add "+key+" to the negative ensemble classifier output file.", true);

                            }

                        }
                    }

                    // Flag that the process was successful
                    negativeSuccess = true;

                } else {
                    // Log the error and return false
                    log.sout("Ensemble classifier negative candidates list couldn't be compiled.", true);
                }

                // Check to see if both processes were successful
                if( positiveSuccess && negativeSuccess )
                {
                    // Return true to denote that it worked
                    return true;
                } else {

                    return false;
                }


            } else {

                // Return false as the ensemble classifier didn't complete
                log.sout("Not all classifiers completed predictions in ensemble classifier. Cannot produce final classifications file.", true);
                return false;
            }

        } else {

            // The ensemble classifier isn't being used, choose individual classifier
            return chooseClassifier(algorithm, predictPath, modelsDir);
        }

    }

    private boolean chooseClassifier(int algorithm, String predictPath, String modelDir)
    {
        // Determine which classifier to make predictions with
        switch (algorithm)
        {
            case Classifiers.J48:
                return makePredictions(new J48Tester(log,"J48Tester"), algorithm, predictPath, Models.getModelFilePath(algorithm,
```

```
                    modelDir));
            case Classifiers.MLP:
                return makePredictions(new MLPTester(log,"MLPTester"), algorithm, predictPath, Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.NB:
                return makePredictions(new NaiveBayesTester(log,"NaiveBayesTester"), algorithm, predictPath, Models.getModelFilePath(algorithm, modelDir));
            case Classifiers.SVM:
                return makePredictions(new SVMTester(log,"SVMTester"), algorithm, predictPath, Models.getModelFilePath(algorithm, modelDir));
            default:
                return false;
        }
    }

    private boolean makePredictions(I_WekaTest classifier, int algorithm, String predictPath, String modelPath)
    {
        // Get the Input Data's file name without the extension
        final String file = predictPath.substring(0, predictPath.lastIndexOf("."));

        // Get the name of the classifier
        final String classifierName = Classifiers.getClassifierName(algorithm);

        // Create the name of the Classifier's output file
        final String outputName = file+"_"+classifierName;

        // Check if the classifier's model loaded
        final boolean loaded = classifier.loadModel(modelPath);

        // Check if the predictions were made
        final boolean predicted = classifier.predict(predictPath, outputName);

        // Check the results of the booleans
        if(loaded & predicted)
            return true;
        else
        {
            if(!loaded)
            {
                log.sout("Could␣not␣load␣the␣"+classifierName+"␣classifier␣model",true);
                return false;
            }
            else if(!predicted)
            {
                log.sout("Could␣not␣make␣predictions␣using␣the␣"+classifierName+"␣classifier␣model",true);
                return false;
            }
            else
            {
```

```
                log.sout("Could not predict using the "+classifierName+" classifier model",true);
                return false;
            }
        }

    }

}
```

Appendix D.E utils/Classification.java

utils/Classification.java

```java
package com.jacobianmatthews.pulsarclassifier.utils;


/**
 * This class is a key-value pair datatype.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 24/02/20
 *
 */
public class Classification
{
    // Variables
    private String key;
    private int value;

    // Constructor
    public Classification(String key, int value)
    {
        // Get the values
        this.key = key;
        this.value = value;

    }

    public int getValue()
    {
        return this.value;
    }

    public String getKey()
    {
        return this.key;
    }

    public void setValue(int v)
    {
        this.value = v;
    }

    public void setKey(String k)
    {
        this.key = k;
    }

}
```

Appendix D.F utils/ClassificationList.java

utils/ClassificationList.java

```java
package com.jacobianmatthews.pulsarclassifier.utils;

import java.util.ArrayList;
import java.util.List;

import com.scienceguyrob.lotaasclassifier.io.Reader;

/**
 * This class creates a list with a key-value pairing system to use inside a
 * list. It is used as a part of ClassPredictor.java to keep track of the number
 * of occurrences of positive and negative pulsar classifications.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 24/05/20
 *
 */
public class ClassificationList
{

    /**
     * Variables
     */
    public List<Classification> list;

    /**
     * Constructor
     */
    public ClassificationList()
    {
        // Create the list
        this.list = new ArrayList<Classification>();
    }

    /**
     *
     * @param index index in the list of the item to retrieve
     * @return the Key-Value pair at the list index
     */
    public Classification get(int index)
    {
        // Return the list item
        return list.get(index);
    }

    public List<Classification> getList()
    {
        return this.list;
    }

    /**
     *
     * @return the number of items in the list
```

```java
     */
    public int size()
    {
        // Get the size of the list
        return list.size();
    }


    public void add(String key, int value)
    {
        // Create a new entry and add it to the list
        list.add(new Classification(key, value));

    }

    /**
     * Get the value of a key-value pair inside the list, found by its key.
     * @param key String with the key of the pair
     * @return Integer value of the pair or 0 if it doesn't exist.
     */
    public int getValueByKey(String key)
    {

        // Loop through the list until the pair is found
        for(Classification item: this.list)
        {
            // Check the string against the provided string
            if( item.getKey().equals(key) )
            {
                // Found the classification item, get its value
                return item.getValue();
            }
        }

        // Return 0 since it wasn't found
        return 0;

    }

    /**
     * Sets the value of a key-value pair inside the list, by its key.
     * @param key String key of the pair.
     */
    public void setValueByKey(String key, int value)
    {
        // Loop through the list until the key-value pair is found
        for(Classification item: this.list)
        {
            // Compare the string keys
            if ( item.getKey().equals(key) )
            {
                // Found the pair, updated the value
                item.setValue(value);
            }

        }
    }
```

```java

    /**
     * Builds an ensemble list of positive or negative classification instances from all classifiers.
     * @param files a List of filepaths to the positive or negative classifier outputs
     * @return true if build is successful
     */
    public boolean buildList(List<String> files)
    {
        // Loop through the classifiers' output files to count instances of candidate classifications
        for(String file: files)
        {
            // Get the number of lines in the file
            int lineCount = Reader.getLineCount(file);

            // Get the contents of each line and handle them individually
            for(int i=1; i<=lineCount; i++)
            {
                // Read the line of the file
                String line = Reader.readLine(file, i);

                // Check if ensemble candidates list is empty
                if ( this.size() == 0 ){

                    // List is empty, no point in checking for previous ocurrences of this line
                    this.add(line, 1);

                } else {

                    // List isn't empty, check if this classification is already in the list
                    int occurrences = this.getValueByKey(line);

                    // If it isn't 0, then it has already showed up and therefore we can add to its total count
                    if (occurrences > 0) {

                        // Add to the value
                        this.setValueByKey(line, occurrences+1);

                    } else {

                        // Occurrences are 0, therefore we can add this key-value pair to the list
                        this.add(line, 1);

                    }
                }
            }

        }
        return true;

    }
```

```java

    /**
     *
     * @return a string containing lines of all key value classification pairs
     */
    public String printList()
        {
            // Create an empty string to print to the command line
            String output = "";

            // Loop through the list items and print their key and value
            for(Classification item: this.list)
            {
                // Get the key and value of the classification
                String key = item.getKey();
                int value = item.getValue();

                // Add to the output string
                output+=key+", "+value+"\n";
            }

            // Return the output string
            return output;

        }

}
```

Appendix D.G utils/Classifiers.java

utils/Classifiers.java

```
package com.jacobianmatthews.pulsarclassifier.utils;


/**
 * This class contains additional methods and properties relating to the
      Classifiers.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 23/05/20
 */
public class Classifiers extends com.scienceguyrob.lotaasclassifier.
     classifiers.Classifiers {

    // An array of the classifiers
    public static final int[] classifiers = {J48, MLP, NB, SVM};

}
```

Appendix D.H utils/Models.java

utils/Models.java

```java
package com.jacobianmatthews.pulsarclassifier.utils;

import com.scienceguyrob.lotaasclassifier.utils.Common;

/**
 * The class Models contains methods to create the path to each classifier's model.
 *
 * @author Jacob Ian Matthews
 * @version 1.0, 23/05/20
 */
public class Models extends Common {

    /**
     * This method converts a directory modelDir of which to output the classifier models, into individual
     * model filepaths for each classifier.
     *
     * @param algorithm an integer relating to the classifier algorithm
     * @param modelDir String containing the directory to place the classifier models
     * @return String with the filepath of the model of the classifier.
    */
    public static String getModelFilePath(int algorithm, String modelDir)
    {
        // Check if the provided path is a valid directory
        if(isDirectory(modelDir)) {

            // Return the model's filepath
            return createModelFilePath(algorithm, modelDir);

        } else {
            // Create the directory as it doesn't exist
            if (dirCreateRecursive(modelDir)) {
                // Return the model's filepath
                return createModelFilePath(algorithm, modelDir);

            } else {
                // Couldn't create the directory, return a null filepath
                return null;
            }
        }
    }

    /**
     *
     * @param algorithm An integer corresponding to the chosen algorithm.
     * @param modelDir A string containing the path of the directory of models
     * @return A string with a filepath to a classifier's model
     */
    private static String createModelFilePath(int algorithm, String modelDir)
    {
        // Create the classifier's model's file name
```

```java
        String fileName = createModelFileName(algorithm);

        // Check if the directory path ends in a forward slash
        if (modelDir.endsWith("/")) {
            // Don't add an extra forward slash and return the full path
            return modelDir + fileName;
        } else {
            // Add a slash to the end of the path
            return modelDir + "/" + fileName;
        }

    }

    /**
     *
     * @param algorithm An integer corresponding to the chosen algorithm
     * @return Returns a string with the filename for the chosen algorithm
     */
    private static String createModelFileName(int algorithm)
    {
        // Get the algorithm/classifier's name
        String classifierName = Classifiers.getClassifierName(algorithm);

        // Return a file name dependent of the classifier's name
        return classifierName+"_model.m";
    }



}
```