

Curtin University
Curtin Institute of Radio Astronomy
Physics Project 1

Finding New Pulsars Using Machine Learning

Jacob Ian Matthews

Draft

June 4, 2020

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	3
1.1	Aim	3
1.2	Structure of this Report	3
1.3	Background Theory	3
1.3.1	Pulsars	3
1.3.2	What is a Pulsar Profile?	6
1.3.3	What is multi-path scattering?	6
1.3.4	What is a Pulsar Dispersion Measure?	6
1.3.5	What is a Radio Telescope and how do they work?	6
1.3.6	What is the Murchison Widefield Array?	6
1.3.7	Why are we conducting sky surveys?	6
1.3.8	How is a sky survey conducted with the Murchison Widefield Array?	6
1.3.9	What is tied-array beamforming?	6
1.3.10	How many Pulsar candidates are found in a Murchison Widefield Array sky survey?	6
1.3.11	How long do sky surveys take?	6
1.3.12	How much data is created from a sky survey?	6
1.3.13	What is Radio Frequency Interference?	6
1.3.14	What is a Signal-to-Noise Ratio?	6
1.3.15	What is PRESTO and a .PFD file?	6
1.3.16	What is Machine Learning?	6
1.3.17	How does Machine Learning work?	6
1.3.18	Why do we need to use Machine Learning in finding new Pulsars?	6
1.3.19	Why is this particular Machine Learning Classifier used?	6
2	Methods	7
2.1	Developing the Machine Learning Training Dataset	7
2.2	Developing a Machine Learning Classifier Validation Tool	8
2.3	Developing an Ensemble Machine Learning Classifier	8
2.4	Evaluating the Machine Learning Classifiers	12
3	Results and Analyses	13
3.1	Machine Learning Training Dataset	13
3.2	Classification Results from the LOTAASClassifier Algorithms	13
3.2.1	The J48 Algorithm	13
3.2.2	The Multi-Layer Perceptron Algorithm	13
3.2.3	The Naïve Bayes Classifier Algorithm	14
3.2.4	The Support Vector Machine Algorithm	15
3.3	Classification Results from the PulsarClassifier Ensemble Classifier	15
3.4	Combined Analyses	16

4	Discussion and Conclusions	17
4.1	Evaluating Curtin Institute of Radio Astronomy's Pulsar Classification Pipeline	17
4.2	Evaluating the Training Dataset and Machine Learning Classifiers	18
5	Recommendations	20
6	References	21
	Appendices	22
A	Methods	22
A.A	Feature Extraction with PulsarFeatureLab	22
A.B	Developing PulsarValidator	23
A.C	Developing PulsarClassifier	27
A.D	Using PulsarClassifier	32
A.E	Using PulsarValidator	33
B	Results	33
B.1	Machine Learning Training Dataset	33
C	Pulsar Validator	38
C.A	PulsarValidator.java	38
C.B	ValidationMode.java	41
C.C	utils/Statistic.java	46
C.D	utils/StatisticList.java	47
C.E	utils/Utilities.java	49
D	Pulsar Classifier	50
D.A	PulsarClassifier.java	50
D.B	ClassifierBuilder.java	63
D.C	ClassifierValidator.java	66
D.D	ClassPredictor.java	69
D.E	utils/Classification.java	75
D.F	utils/ClassificationList.java	76
D.G	utils/Classifiers.java	80
D.H	utils/Models.java	81

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*.

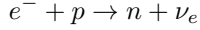
1.3 Background Theory

1.3.1 Pulsars

To answer the question of “what is a pulsar?” we must first investigate the evolution and death of stars.

A star can form when a cloud of hydrogen gas in the interstellar medium (ISM) collects mass over millions of years; as the mass of the gas cloud increases,

its gravitational pull to gather more mass also increases (Maoz 2016). This proto-star will eventually reach a critical mass in which the pressure of gravity within the gas causes enough friction between the gas particles to generate the required heat (thermal pressure) to begin fusing the hydrogen atoms into helium (Maoz 2016). This marks the beginning of the star's main sequence lifetime. Once the star has fused all of the available hydrogen gas in its core, the star will begin fusing helium into carbon and its outer envelope will expand, moving the star into its "red-giant" phase (Maoz 2016). If the initial mass of the star was greater than 8 times the mass of the Sun (i.e. $8M_{\odot}$), the star will continue to fuse the elements in its core until it reaches a core of iron. At this point phenomena called nuclear photodisintegration and neutronization occurs, the latter of which causes electrons and photons to combine and form neutrons and anti-neutrinos (Maoz 2016). Neutronization can be shown as:



This process removes the electron degeneracy pressure in the core of the star (a pressure which balances the star's gravitational pressure), causing the star to collapse under its own gravity in a timeframe of 0.1 seconds (Maoz 2016). The gravitational collapse stops once the gravitational pressure of the star is balanced by the neutron degeneracy pressure, i.e. the pressure from pushing neutrons together. The remaining star is incredibly dense, with a mass of approximately $1.4M_{\odot}$ and a radius of around 11km. This is called a neutron star (Maoz 2016).

Prior to the collapse of the star, we can imagine the star to be rotating at an angular velocity of ω_1 . We know from the conservation of angular momentum that when the radius of a rotating object decreases, the angular velocity will increase (a spinning ice skater pulling their arms in close increases the speed of their spinning). We can therefore show that the angular velocity of the star after the gravitational collapse, ω_2 , is much greater than the prior angular velocity:

$$L_1 = L_2$$

where L is the angular momentum, $L_1 = I_1\omega_1$ and $L_2 = I_2\omega_2$. Therefore:

$$I_1\omega_1 = I_2\omega_2$$

$$\omega_2 = \frac{I_1}{I_2}\omega_1$$

Assuming the star is a sphere, its moment of inertia, I is:

$$I = \frac{2}{5}MR^2$$

where M is the mass of the star and R is the radius of the star. We can thus show:

$$\omega_2 = \frac{\frac{2}{5}MR_1^2}{\frac{2}{5}MR_2^2}\omega_1$$

$$\omega_2 = \left(\frac{R_1}{R_2}\right)^2 \omega_1$$

where $R_1 \gg R_2$. We are left with a neutron star with a very large angular velocity. Analogous to the angular velocity of the star, the magnetic field of the star is also amplified. The ionised gas in the iron core of the star, which generates a magnetic field, is compressed by the gravitational collapse, forcing the flux of the magnetic field to be amplified such that the field strength is approximately 10^{10} times stronger in the neutron star compared to during the star's main sequence lifetime (Maoz 2016).

If the rotation of the neutron star is misaligned with the axis of the magnetic field by an angle θ , the spinning magnetic dipole will radiate electromagnetic waves (Maoz 2016). As the neutron star rotates, the radiated electromagnetic waves will periodically sweep across the line of sight of an observer, creating a pulse of light. See Figure 1. We can therefore define a pulsar as a rapidly rotating neutron star that appears to periodically emit electromagnetic waves (Maoz 2016; Lorimer and Kramer 2005; Swainston 2020).

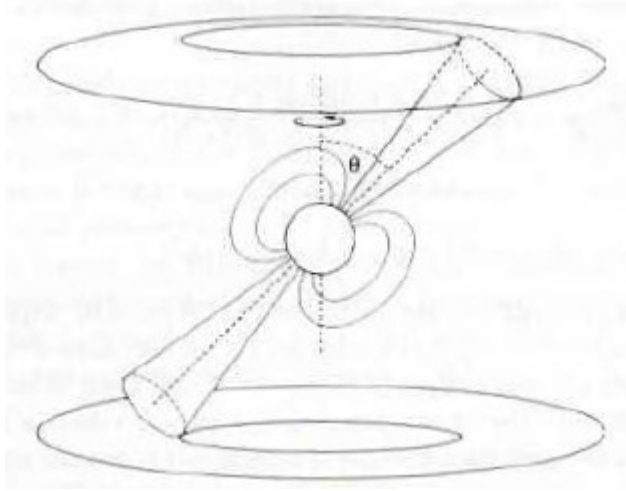


Figure 1: A pulsar (Maoz 2016).

While some pulsars, like the Crab Pulsar (Maoz 2016), emit electromagnetic waves in the visible spectrum and can therefore be detected by an optical telescope, the majority of pulsar emission is invisible to the human eye and requires a radio telescope to be detected.

- 1.3.2 What is a Pulsar Profile?
- 1.3.3 What is multi-path scattering?
- 1.3.4 What is a Pulsar Dispersion Measure?
- 1.3.5 What is a Radio Telescope and how do they work?
- 1.3.6 What is the Murchison Widefield Array?
- 1.3.7 Why are we conducting sky surveys?
- 1.3.8 How is a sky survey conducted with the Murchison Widefield Array?
- 1.3.9 What is tied-array beamforming?
- 1.3.10 How many Pulsar candidates are found in a Murchison Widefield Array sky survey?
- 1.3.11 How long do sky surveys take?
- 1.3.12 How much data is created from a sky survey?
- 1.3.13 What is Radio Frequency Interference?
- 1.3.14 What is a Signal-to-Noise Ratio?
- 1.3.15 What is PRESTO and a .PFD file?
- 1.3.16 What is Machine Learning?
- 1.3.17 How does Machine Learning work?
- 1.3.18 Why do we need to use Machine Learning in finding new Pulsars?
- 1.3.19 Why is this particular Machine Learning Classifier used?

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:

$$Prof_{\mu}, Prof_{\sigma}, Prof_S, Prof_k \quad (1)$$

$$DM_{\mu}, DM_{\sigma}, DM_S, DM_k, DM_{\mu'}, DM_{\sigma'}, DM_{|S'|}, DM_{k'} \quad (2)$$

$$Subband_{\mu}, Subband_{\sigma}, Subband_S, Subband_k \quad (3)$$

$$Subint_{\mu}, Subint_{\sigma}, Subint_S, Subint_k \quad (4)$$

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)

```

1: if algorithm = -1 then
2:   for each algorithm i = 1 → 4 do           ▷ Loop through all classifiers
3:     buildClassifier(i, trainingSet, modelsDirectory)
4:   end for
5: else                                           ▷ Build individual classifier
6:   buildClassifier(algorithm, trainingSet, modelPath)
7: end if

```

Algorithm 3 ClassifierValidator (pseudocode)

```

1: if algorithm = -1 then
2:   for each algorithm i = 1 → 4 do           ▷ Loop through all classifiers
3:     testClassifier(i, testSet, modelsDirectory)
4:   end for
5: else                                           ▷ Test individual classifier
6:   testClassifier(algorithm, testSet, modelPath)
7: 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)

```

1: if algorithm = -1 then
2:   list = new OutputFileList()
3:   for algorithm i = 1 → 4 do                                ▷ Loop through all classifiers
4:     makePredictions(i, inputData, modelsDirectory)
5:     list.add(ClassifierOutputFiles)                            ▷ Add output filepaths to list
6:   end for
7:   positiveList = new ClassificationsList()
8:   negativeList = new ClassificationsList()                      ▷ Create classification lists
9:   for each OutputFile from list do
10:    if OutputFile is .positive then
11:      for each line in OutputFile do
12:        positiveList.add(line)                                ▷ Add to +ve classifications
13:      end for
14:    else if OutputFile is .negative then
15:      for each line in OutputFile do
16:        negativeList.add(line)                                ▷ Add to -ve classifications
17:      end for
18:    end if
19:  end for
20:  for each classification in positiveList do
21:    if classification.occurrences < 3 then
22:      negativeList.add(classification)                          ▷ cut-off at < 3 classifications
23:    else
24:      positiveOutput(classification)                            ▷ Output classified as Pulsar
25:    end if
26:  end for
27:  for each classification in negativeList do
28:    negativeOutput(classification)                              ▷ Output classified as non-Pulsar
29:  end for
30: else
31:   makePredictions(algorithm)                                ▷ Use individual classifier
32: 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 Combined Analyses

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

Table 1: Success Rates of Each Pulsar Classifier

Classifier	R_p	R_{np}	<i>Combined</i>
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%
PulsarClassifier			
Ensemble	76.19%	100.00%	88.10%

R_p : Success rate of pulsar classification.
 R_{np} : Success rate of non-pulsar classification.

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 2020). 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 eliminate a large number of pulsar candidates; and
- iv. Manually inspect the remaining pulsar candidates to confirm pulsar discovery.

(Swainston 2020). 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 Swainston (2020), 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 1 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 2.2.4* 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 `PulsarClassifier`, 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 1. 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

- 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.
- Swainston, Nick. 2020. “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>.

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

```
1 FROM alpine/git:latest as builder
2 WORKDIR /root/
3 RUN cd /root/ && git clone --single-branch --branch V1.3.2
   https://github.com/scienceguyrob/PulsarFeatureLab.git &&
   mkdir PulsarFeatureLab/PulsarFeatureLab/Data/IO
4
5 FROM python:2.7
6 WORKDIR /usr/src/app
7 COPY --from=builder /root/PulsarFeatureLab .
8 RUN pip install numpy scipy matplotlib astropy
9 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 classifier¹. 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  $i$ th 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 ??.1*.

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 7.2.2*, and its supplementary classes in *Appendix 7.2*.

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 7.2* 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/lotaclassifier/
    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)

```

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

```

See *Appendix ??*.2 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)

```
1: if algorithm = -1 then
2:   for each algorithm  $i = 1 \rightarrow 4$  do           ▷ Loop through all classifiers
3:     testClassifier(i, testSet, modelsDirectory)
4:   end for
5: else                                           ▷ Test individual classifier
6:   testClassifier(algorithm, testSet, modelPath)
7: end if
```

See *Appendix ??*.3 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)

```
1: if algorithm = -1 then
2:   list = new OutputFileList()
3:   for algorithm i = 1 → 4 do                                ▷ Loop through all classifiers
4:     makePredictions(i, inputData, modelsDirectory)
5:     list.add(ClassifierOutputFiles)                            ▷ Add output filepaths to list
6:   end for
7:   positiveList = new ClassificationsList()
8:   negativeList = new ClassificationsList()                      ▷ Create classification lists
9:   for each OutputFile from list do
10:    if OutputFile is .positive then
11:      for each line in OutputFile do
12:        positiveList.add(line)                                ▷ Add to +ve classifications
13:      end for
14:    else if OutputFile is .negative then
15:      for each line in OutputFile do
16:        negativeList.add(line)                                ▷ Add to -ve classifications
17:      end for
18:    end if
19:  end for
20:  for each classification in positiveList do
21:    if classification.occurrences < 3 then
22:      negativeList.add(classification)                          ▷ cut-off at < 3 classifications
23:    else
24:      positiveOutput(classification)                            ▷ Output classified as Pulsar
25:    end if
26:  end for
27:  for each classification in negativeList do
28:    negativeOutput(classification)                              ▷ Output classified as non-Pulsar
29:  end for
30: else
31:   makePredictions(algorithm)                                ▷ Use individual classifier
32: 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 ??*.4 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 ??.1. 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>
        </plugin>
      </plugins>
    </build>
  </project>
```



```

        </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 *Section 2.2* 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

```
1 @relation Pulsar_Feature_Data_Type_6
2 @attribute Feature_1 numeric
3 @attribute Feature_2 numeric
4 @attribute Feature_3 numeric
5 @attribute Feature_4 numeric
6 @attribute Feature_5 numeric
7 @attribute Feature_6 numeric
8 @attribute Feature_7 numeric
9 @attribute Feature_8 numeric
10 @attribute Feature_9 numeric
11 @attribute Feature_10 numeric
12 @attribute Feature_11 numeric
13 @attribute Feature_12 numeric
14 @attribute Feature_13 numeric
15 @attribute Feature_14 numeric
16 @attribute Feature_15 numeric
17 @attribute Feature_16 numeric
18 @attribute Feature_17 numeric
19 @attribute Feature_18 numeric
20 @attribute Feature_19 numeric
21 @attribute Feature_20 numeric
22 @attribute class {0,1,2}
23 @data
24 105.43918879794592,34.81562200057764,0.64091523062507,2.999965982775474,
    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

25 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

26 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

27 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

28 125.23524447122634,56.134362050023924,0.2229079772063587,-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

29 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

30 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

31 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,
0.0481999983836385,-0.9897646915402789,51.54462076437605,
24.67525889451714,0.13313381803356525,-0.7286102199011202,1,%,usr/src/
app/PulsarFeatureLab/Data/IO/1255803168_100_bins_PSR_0401-7608.pfd

32 130.37194427538753,48.5719656104399,0.03110721286401433,-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

40 118.82100466581987,47.54490692614183,0.07781892865435884,-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

41 106.55367062734199,38.31922288316345,1.1201921873325247,3.524621849432644,
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_13136.61
ms_Cand_rfi.pfd

42 110.58047755262913,56.835988222404985,0.2597416198749472,-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

43 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

44 113.87621446459325,43.41151017852675,0.5893561441860531,1.71756136954156,
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_10373.16
ms_Cand_rfi.pfd

45 116.41310084274251,46.52577061051363,0.1525172365016325,0.14280165135512535,
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

46 115.05147692422577,59.943352711870936,0.06373023728528525,-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

47 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/I0/1256407632_100_bins_PSR_0459-0210.pfd
```

Appendix C Pulsar Validator

Appendix C.A PulsarValidator.java

PulsarValidator.java

```
1 package com.jacobianmatthews.pulsarvalidator;
2
3 import java.io.IOException;
4
5 /**
6  * This program is a validator for the machine learning software
7  * PulsarClassifier.
8  *
9  * This program will generate a list of candidate filenames from an output
10  * file
11  * of PulsarFeatureLab. It will then be possible to filter the list of known
12  * pulsars in the original dataset to the pulsars that had successful feature
13  * extraction. Finally, the program can compare the list of known pulsars in
14  * the
15  * dataset to the pulsars classified in the classifier, and produce comparison
16  * statistics.
17  *
18  * @author Jacob Ian Matthews
19  * @version 1.0, 28/05/2020
20  */
21 public class PulsarValidator {
22
23     /**
24      *
25      * VARIABLES
26      */
27
28     // Pulsar list mode
29     private static boolean list = false;
30
31     // Pulsar classification output validation mode
32     private static boolean validation = false;
33
34     // String containaing the path to the PulsarFeatureLab output file
35     private static String pflOutput;
36
37     // String containing the path to the list of pulsars
38     private static String pulsarList;
39
40     // String containing the path to the positive classifier output
41     private static String positiveClassifier;
42
43     // String containing the path to the negative classifier output
44     private static String negativeClassifier;
45
46     public static void main(String[] args) throws IOException {
47         // Get the input arguments
48         getCliVariables(args);
49
50         // Check to see which flag has been given
```

```

50     if (list) {
51         // LIST MODE CHOSEN
52         // Print a message
53         System.out.println("Pulsar_list_generation_mode_chosen.\n");
54
55         // Create a list mode instance
56         System.out.println("This_feature_is_under_development._Exiting_
           program.");
57         System.exit(0);
58
59     } else if (validation) {
60
61         // VALIDATION MODE CHOSEN
62         // Print a message
63         System.out.println("\nPulsar_classifier_validation_mode_chosen.");
64
65         // Print the location of the pulsar list
66         System.out.println("\nPulsar_list_location:_" + pulsarList);
67
68         // Print the location of the positive output file
69         System.out.println("Classifier_positive_output_location:_" +
           positiveClassifier);
70
71         // Print the location of the positive output file
72         System.out.println("Classifier_negative_output_location:_" +
           negativeClassifier);
73
74         // Create the validation mode instance
75         ValidationMode validationMode = new ValidationMode(pulsarList,
           positiveClassifier, negativeClassifier);
76
77         // Get the output string
78         String output = validationMode.validate();
79
80         // Output the string and then exit the program
81         System.out.println("\nPulsar_classifier_validated_successfully.\n"
           );
82         System.out.println(output);
83
84         System.exit(0);
85
86     } else if (list && validation) {
87
88         // Display error that you can only do one thing at once
89         System.out.println("'-'_and_'-v'_arguments_entered._Please_choose
           _one_mode_only._\n");
90
91         // Exit the application
92         System.exit(0);
93
94     } else if (!list && !validation) {
95
96         // Display error that you need to pick a flag
97         System.out.println("Please_choose_a_mode_by_using_this_program_
           with_a_'-l'_or_'-v'_argument.\n");
98
99         // Exit the application

```



```

100         System.exit(0);
101     }
102
103 }
104
105 /**
106  * This function checks the command-line input arguments to decide how the
107  * program should be ran.
108  *
109  * @param args
110  */
111 private static void getCliVariables(String[] args)
112 {
113
114     // Loop through arguments for the list and compare flags
115     for(int i = 0; i<args.length; i++)
116     {
117         // Define the current argument
118         String argument = args[i];
119
120         // The list flag
121         if( argument.equals("-l") ){
122
123             // Set the list mode boolean to true
124             list = true;
125
126             // Get the next argument (the PulsarFeatureLab output file)
127             pflOutput = args[i+1];
128
129             // Get the input pulsar list
130             pulsarList = args[i+2];
131
132             // Check for compare flag
133             } else if( argument.equals("-v") ){
134
135                 // Set the compare mode boolean to true
136                 validation = true;
137
138                 // Get the list of pulsars
139                 pulsarList = args[i+1];
140
141                 // Get the positive classifier output file
142                 positiveClassifier = args[i+2];
143
144                 // Get the negative classifier output file
145                 negativeClassifier = args[i+3];
146
147             }
148         }
149     }
150 }
151 }

```

Appendix C.B ValidationMode.java

ValidationMode.java

```
1 package com.jacobianmatthews.pulsarvalidator;
2
3 import java.io.IOException;
4 import java.nio.file.Files;
5 import java.nio.file.Paths;
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import com.jacobianmatthews.pulsarvalidator.utils.StatisticList;
10 import com.jacobianmatthews.pulsarvalidator.utils.Utilities;
11
12 /**
13  * This class contains the Pulsar Classifier validation mode of the program.
14  * It
15  * will compare the output files of the Pulsar Classifier to the true list of
16  * pulsars and output statistics.
17  *
18  * @author Jacob Ian Matthews
19  * @version 1.0, 29/05/2020
20  */
21 public class ValidationMode {
22     /** VARIABLES */
23     private String positiveOutputPath;
24
25     private String negativeOutputPath;
26
27     private List<String> pulsars;
28
29     /** CONSTRUCTOR */
30     /**
31      * Instantiates the Validation mode.
32      *
33      * @param pulsarListPath A string containing the path to the list of
34      * pulsars
35      * in the data set.
36      * @param positiveOutputPath A string containing the path to the positive
37      * output
38      * file of the classifier.
39      * @param negativeOutputPath A string containing the path to the negative
40      * output
41      * file of the classifier.
42      * @throws IOException
43      */
44     public ValidationMode(String pulsarListPath, String positiveOutputPath,
45         String negativeOutputPath)
46         throws IOException {
47
48         // Assign the variables
49         this.positiveOutputPath = positiveOutputPath;
50         this.negativeOutputPath = negativeOutputPath;
51
52         // Validate the filepaths
```

```

49     if (!Utilities.isFile(pulsarListPath)) {
50         // Print an error and end the program
51         System.out.println("The path given to the pulsar list is not valid
           \nExiting program.");
52         System.exit(0);
53     }
54
55     if (!Utilities.isFile(positiveOutputPath)) {
56         // Print an error and end the program
57         System.out.println("The path given to the positive classifier
           output is not valid.\nExiting program.");
58         System.exit(0);
59     }
60
61     if (!Utilities.isFile(negativeOutputPath)) {
62         // Print an error and end the program
63         System.out.println("The path given to the negative classifier
           output is not valid.\nExiting program.");
64         System.exit(0);
65     }
66
67     // Get the list of pulsars
68     this.pulsars = Files.readAllLines(Paths.get(pulsarListPath));
69
70 };
71
72 /**
73  * This validates the classifier's outputs against the list of pulsars.
74  *
75  * @return String containing the validation statistics.
76  * @throws IOException
77  */
78 public String validate() throws IOException
79 {
80     // Create a list of the statistics
81     StatisticList statistics = new StatisticList();
82
83     // Add the true and false positive statistics to the list
84     StatisticList positiveStatistics = processPositive(statistics);
85
86     // Add the true and false negative statistics to the list
87     StatisticList allStatistics = processNegative(positiveStatistics);
88
89     // Get the statistics
90     int TP = allStatistics.getValueByName("TruePositives");
91     int FP = allStatistics.getValueByName("FalsePositives");
92     int TN = allStatistics.getValueByName("TrueNegatives");
93     int FN = allStatistics.getValueByName("FalseNegatives");
94
95     // To find the number of pulsars without generating a new list from
           the output data of
96     // PulsarFeatureLab, we can just add the TruePositive count with the
           FalseNegative count.
97     // The same can be applied for non-pulsars
98     int pulsarCount = TP + FN;
99     int nonpulsarCount = TN + FP;
100

```

```

101     int pulsarsDetected = TP + FP;
102     int nonpulsarsDetected = TN + FN;
103
104     // Output the statistics as a string
105     String output = "Number of Pulsars: " + pulsarCount;
106     output+= "\nPulsars Detected: " + pulsarsDetected;
107     output+= "\nTrue Positives: " + TP;
108     output+= "\nFalse Positives: " + FP;
109     output+= "\n\nNumber of Non-Pulsars: " + nonpulsarCount;
110     output+= "\nNon-Pulsars Detected: " + nonpulsarsDetected;
111     output+= "\nTrue Negatives: " + TN;
112     output+= "\nFalse Negatives: " + FN;
113
114     // Return the output
115     return output;
116
117 }
118
119 /**
120  * This will create the true positive and false positive lists and output
121    their
122  * results
123  * @return a StatisticList object containing the positive statistics
124  * @throws IOException
125  */
126 private StatisticList processPositive(StatisticList statistics) throws
    IOException
127 {
128     // Create the True Positive list
129     List<String> truePositive = new ArrayList<String>();
130
131     // Create the False Positive list
132     List<String> falsePositive = new ArrayList<String>();
133
134     // Get the positive output file list
135     List<String> classifier = Files.readAllLines(Paths.get(this.
        positiveOutputPath));
136
137     // Check each line of the classifier's output
138     for(String classification: classifier)
139     {
140         // Get the filename of the classification
141         String name = classification.substring(classification.lastIndexOf(
            "/")+1, classification.length());
142
143         // Check if it is in the list of pulsars
144         // Create search flag and index
145         boolean found = false;
146         int i = 0;
147
148         // Loop through the list
149         while(!found){
150
151             // Check if the classified pulsar is in the real list
152             if( this.pulsars.get(i).equals(name) ){
153

```

```

154         // Add pulsar to the true positive list
155         truePositive.add(name);
156
157         // Set the flag to found
158         found = true;
159
160     } else {
161         // Increment loop
162         i++;
163     }
164
165     // Check if we have reached the end of the list
166     if ( !(i < this.pulsars.size()) ){
167
168         // Add this pulsar to the false positive list
169         falsePositive.add(name);
170
171         // End the loop
172         found = true;
173
174     }
175 }
176
177 // Count the number of true and false positives
178 int truePositiveCount = truePositive.size();
179 int falsePositiveCount = falsePositive.size();
180
181 // Add the statistics to the list
182 statistics.add("TruePositives", truePositiveCount);
183 statistics.add("FalsePositives", falsePositiveCount);
184
185 // Return the list
186 return statistics;
187
188 }
189
190 /**
191  * This creates the false and true negative lists and output the results
192  * of
193  * their statistics
194  *
195  * @return a StatisticList containing the negative statistics
196  * @throws IOException
197  */
198 private StatisticList processNegative(StatisticList statistics) throws
199     IOException
200 {
201     // Create the False Negative List
202     List<String> falseNegative = new ArrayList<String>();
203
204     // Create the True Negative List
205     List<String> trueNegative = new ArrayList<String>();
206
207     // Get the negative output list
208     List<String> classifier = Files.readAllLines(Paths.get(this.
209         negativeOutputPath));

```

```

208
209 // Loop through the negative output file
210 for(String classification: classifier)
211 {
212     // Get the filename of the candidate
213     String name = classification.substring(classification.lastIndexOf(
        "/" )+1,classification.length());
214
215     // Create the loop variables
216     boolean found = false;
217     int i = 0;
218     while(!found)
219     {
220         // Check if the name is in the list of pulsars
221         if( this.pulsars.get(i).equals(name) ){
222             // Detected a false negative, therefore add to the list
223             falseNegative.add(name);
224
225             // End loop
226             found = true;
227         } else {
228             // increment loop
229             i++;
230         }
231
232         // Check if the end of the pulsars list has been reached
233         if ( !(i < this.pulsars.size() ) ){
234
235             // Add this classification to the true negatives list
236             trueNegative.add(name);
237
238             // End loop
239             found = true;
240         }
241     }
242 }
243
244 // Create the statistics
245 int trueNegativeCount = trueNegative.size();
246 int falseNegativeCount = falseNegative.size();
247
248 // Add the statistics to the list
249 statistics.add("TrueNegatives", trueNegativeCount);
250 statistics.add("FalseNegatives", falseNegativeCount);
251
252 // Return the statistics list
253 return statistics;
254
255 }
256
257
258 }

```

Appendix C.C utils/Statistic.java

utils/Statistic.java

```
1  package com.jacobianmatthews.pulsarvalidator.utils;
2
3  /**
4   * This is a data object to hold a statistic for the classifier.
5   *
6   * @author Jacob Ian Matthews
7   * @version 1.0, 29/05/2020
8   */
9  public class Statistic {
10
11     /** VARIABLES */
12     private String name;
13     private int value;
14
15     /** CONSTRUCTOR */
16     public Statistic(String name, int value) {
17
18         // Get the variables
19         this.name = name;
20         this.value = value;
21     }
22
23
24     /** GETTERS AND SETTERS */
25     public void setValue(int value)
26     {
27         this.value = value;
28     }
29
30     public int getValue(){
31
32         return this.value;
33     }
34
35     public void setName(String name)
36     {
37         this.name = name;
38     }
39
40     public String getName()
41     {
42         return this.name;
43     }
44
45 }
```

Appendix C.D utils/StatisticList.java

utils/StatisticList.java

```
1  package com.jacobianmatthews.pulsarvalidator.utils;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  /**
7   * A list to hold the classifier's statistics.
8   *
9   * @author Jacob Ian Matthews
10   * @version 1.0, 29/05/2020
11   */
12  public class StatisticList {
13
14      /** VARIABLES */
15      private List<Statistic> list;
16
17      /** CONSTRUCTOR */
18      public StatisticList() {
19
20          // Create the list
21          this.list = new ArrayList<Statistic>();
22
23      }
24
25      /**
26       *
27       * @param name
28       * @param value
29       */
30      public void add(String name, int value)
31      {
32          // Create a statistic
33          Statistic stat = new Statistic(name, value);
34
35          // Add it to the list
36          this.list.add(stat);
37      }
38
39      /**
40       * Get the value of a statistic by its name
41       * @param name String containing the name of the statistic
42       * @return integer value of statistic
43       */
44      public int getValueByName(String name)
45      {
46          // Loop through the list
47          for(Statistic stat: this.list)
48          {
49              if( stat.getName().equals(name) ){
50
51                  // Return the value
52                  return stat.getValue();
53              }
54          }
55      }
56  }
```



```
54     }  
55  
56     // Couldn't find it, return 0  
57     return 0;  
58 }  
59  
60 }
```

Appendix C.E utils/Utilities.java

utils/Utilities.java

```
1  package com.jacobianmatthews.pulsarvalidator.utils;
2
3  import java.io.File;
4
5  /**
6   * This class contains the common utility functions across the program.
7   *
8   * @author Jacob Ian Matthews
9   * @version 1.0, 29/05/2020
10  */
11  public class Utilities {
12
13      /** CONSTRUCTOR */
14      public Utilities(){
15          // Empty
16      }
17
18      /**
19       * This method validates that a file exists.
20       * @param path A string containing the path to a file.
21       * @return True if it is a valid, existing file.
22       */
23      public static boolean isFile(String path)
24      {
25          // Trim the whitespace
26          String trimmed = path.trim();
27
28          // Create the file
29          File file = new File(trimmed);
30
31          // Validate the file
32          if(file.isFile()){
33
34              // Return true
35              return true;
36          } else {
37
38              // Return false;
39              return false;
40          }
41      }
42
43  }
44 }
```

Appendix D Pulsar Classifier

Appendix D.A PulsarClassifier.java

PulsarClassifier.java

```
1 package com.jacobianmatthews.pulsarclassifier;
2
3 import com.scienceguyrob.lotaasclassifier.classifiers.Classifiers;
4 import com.scienceguyrob.lotaasclassifier.cli.CLI;
5 import com.scienceguyrob.lotaasclassifier.cli.CLParameter;
6 import com.scienceguyrob.lotaasclassifier.cli.ICLI;
7 import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
8 import com.scienceguyrob.lotaasclassifier.utils.Common;
9
10 import java.net.URL;
11
12
13 /**
14  * This class takes in command-line arguments and starts the PulsarClassifier
15  * program.
16  *
17  * It is a derivative of the com.scienceguyrob.lotaasclassifier package.
18  *
19  * @author Jacob Ian Matthews & Rob Lyon
20  *
21  * @version 1.0, 22/5/2020
22  */
23 public class PulsarClassifier {
24     /**
25      *
26      * VARIABLES
27      *
28      */
29
30     /**
31      * Full path to the file containing training data.
32      */
33     private static String training_path = "";
34
35     /**
36      * Full path to the file (ARFF file) to be assigned classifier predictions
37      *
38      */
39     private static String predict_path = "";
40
41     /**
42      * Full path to the classifier validation file (ARFF file).
43      */
44     private static String validate_path = "";
45
46     /**
47      * Full path to the file containing the classification model to use.
48      */
49     private static String model_path = "";
50
51     /**
```

```

51      * The algorithm to train/use to make predictions.
52      */
53      private static int algorithm = -1;
54
55      /**
56       * Logging flag, if true, verbose logging outputs will be written to
57       * standard out.
58      */
59      private static boolean verbose = false;
60
61      /**
62       * Training flag. If true the system will attempt to build a new
63       * classifier.
64      */
65      private static boolean train = false;
66
67      /**
68       * Prediction flag. If true the system will attempt to classify data.
69      */
70      private static boolean predict = false;
71
72      /**
73       * Validation flag. If true, the system will attempt to validate the
74       * classifiers performance.
75      */
76      private static boolean validate = false;
77
78      /**
79       * The working directory for this code.
80      */
81      private static URL workingDir = PulsarClassifier.class.
82      getProtectionDomain().getCodeSource().getLocation();
83
84      /**
85       * The object used to output debug/logging information.
86      */
87      private static BasicLogger log = new BasicLogger(verbose,workingDir.
88      getFile().replace(".jar",".log"));
89
90      *****
91      *****
92      *****
93      *****
94      *****
95      *****
96
97      /**
98       * The main entry point to the application.
99       * @param args the command line arguments.
100     */
101     public static void main(String[] args)
102     {
103         processCommandLine(args);
104
105         /**
106          * Run desired commands...
107         */

```

```

103     if(!train & !predict & !validate)
104     {
105         log.sout("Unable_to_train_classifier_model/make_predictions-
            inputs_invalid", true);
106         safeExit();
107     }
108     else if(train & predict)
109     {
110         log.sout("Valid_training_and_classification_inputs_provided-
            system_unsure_what_to_do", true);
111         safeExit();
112     }
113     else if(train)
114     {
115         log.sout("Attempting_to_build_a_new_classifier", true);
116         ClassifierBuilder cb = new ClassifierBuilder(log,"
            ClassifierBuilder");
117
118         // Actually build the ensemble classification system
119         boolean result = cb.build(algorithm,training_path,model_path);
120
121         // Check the results of the classifier build
122         if( algorithm == -1 )
123         {
124             if (result)
125             {
126                 log.sout("Ensemble_classifier_built_successfully", true);
127
128             } else {
129
130                 log.sout("Ensemble_classifier_couldn't_be_built.", true);
131             }
132
133         } else {
134             if (result)
135             {
136                 log.sout(Classifiers.getClassifierName(algorithm)+"
                    classifier_built_successfully", true);
137
138             } else {
139                 log.sout(Classifiers.getClassifierName(algorithm)+"
                    classifier_couldn't_be_built.", true);
140             }
141         }
142     }
143 }
144 else if(predict)
145 {
146     log.sout("Attempting_to_apply_predictions_using_existing_
        classifier", true);
147
148     ClassPredictor cp = new ClassPredictor(log,"ClassPredictor");
149
150     // Actually build the classification system
151     boolean result = cp.predict(algorithm,predict_path,model_path);
152
153     // Log the results back to the user

```

```

154         if ( algorithm == -1 )
155         {
156             if ( result )
157             {
158                 log.sout("Ensemble_classifier_made_predictions_successfully
159                 .", true);
160             } else {
161                 log.sout("Ensemble_classifier_was_unsuccessful_in_applying_
162                 predictions.", true);
163             }
164         } else {
165             if( result )
166             {
167                 log.sout(Classifiers.getClassifierName(algorithm)+ "
168                 classifier_made_predictions_successfully", true);
169             } else {
170                 log.sout(Classifiers.getClassifierName(algorithm)+ "
171                 classifier_unsuccessful_in_applying_predictions", true
172                 );
173             }
174         }
175     }
176     else if(validate)
177     {
178         log.sout("Attempting_to_validate_existing_classifier_performance",
179         true);
180
181         ClassifierValidator cv = new ClassifierValidator(log,"
182         ClassifierValidator");
183
184         // Validate the classifier chosen
185         boolean result = cv.validate(algorithm,validate_path,model_path);
186
187         // Output the results for the ensemble classifier
188         if(algorithm == -1) {
189             if(result) {
190                 log.sout("Ensemble_classifier_validated_successfully.",
191                 true);
192             } else {
193                 log.sout("Ensemble_classifier_couldn't_be_validated_
194                 successfully.", true);
195             }
196         }
197
198         // Output the results for the individual classifiers
199         } else {
200             if(result) {
201                 log.sout(Classifiers.getClassifierName(algorithm)+ "
202                 classifier_validated_successfully", true);
203             } else {
204                 log.sout(Classifiers.getClassifierName(algorithm)+ "
205                 classifier_validation_unsuccessful", true);
206             }
207         }
208     }
209 }

```

```

200         }
201
202     }
203
204     /**
205      * When done...
206      */
207     safeExit();
208 }
209
210 /**
211  * Command line processing methods.
212  */
213
214 /**
215  * Processes the command line parameters.
216  * @param args the command line arguments
217  */
218 public static void processCommandLine(String[] args)
219 {
220     // Print some details, help etc to command line.
221     printApplicationDetails();
222
223     // Always make sure to write an app start message to the log file.
224     printLogFileHeader();
225
226     // Set logging to false before processing user input
227     verbose = false;
228
229     System.out.println( "\nReading Terminal Parameters...\n" );
230
231     ICLI cli = getCommandLineOptions();
232     cli.processArguments(args);
233
234     // Update local variables based on user parameters.
235     updateVariables(cli);
236     printParameters();
237 }
238
239 /**
240  * Updates class variables with user input parameters, if provided.
241  * @param cli the command line parameters to use to obtain user input.
242  */
243 private static void updateVariables(ICLI cli)
244 {
245     /**
246      * There are two main modes for the application. Either build a
247      * classifier, or
248      * classify new data.
249      *
250      * Option one requires the full path to a training set file, the
251      * integer identifier
252      * of the classifier to build, and the path to write the
253      * classification model to.
254      *
255      * Option two requires the full path to a file requiring predictions,
256      * the integer

```

```

253      * identifier of the classifier used to make the predictions, and the
254      * path to the
255      * classification model.
256      *
257      * Below we make sure this logic is correct.
258      */
259      // First deal with general simulation wide variables.
260      if(cli.hasParameter(FLAG_VERBOSE))
261          verbose = true;
262      else
263          verbose = false;
264
265      // Check the algorithm supplied is valid, if not return since there's
266      // no point checking the other variables.
267      if (cli.hasParameter(FLAG_ALGORITHM))
268          if (cli.getParameter(FLAG_ALGORITHM).toInt() > -2 && cli.
269              getParameter(FLAG_ALGORITHM).toInt() < 6)
270              algorithm = cli.getParameter(FLAG_ALGORITHM).toInt();
271          else
272          {
273              log.sout("Algorithm_value_supplied_via_a_flag_invalid_(must_
274                  be_1_to_5).", true);
275              return;
276          }
277
278      if(cli.hasParameter(FLAG_TRAINING)) // if a training set has been
279      provided.
280      {
281          // Try to load training data path (used for ML classification).
282          if (cli.hasParameter(FLAG_TRAINING))
283          {
284              training_path = cli.getParameter(FLAG_TRAINING).getValue();
285
286              if (Common.fileExist(training_path)) // If the training set is
287              valid.
288              {
289                  if (cli.hasParameter(FLAG_MODEL))
290                  {
291                      model_path = cli.getParameter(FLAG_MODEL).getValue();
292
293                      if (Common.isPathValid(model_path)) // If the training
294                      set is valid.
295                      train = true;
296                  }
297                  else
298                  {
299                      log.sout("Model_output_path_specified_via_m_flag_
300                          invalid",true);
301                  }
302              }
303              else
304              {
305                  log.sout("No_output_model_path_supplied_with_m_flag",
306                      true);
307              }
308          }
309          else
310          {
311              log.sout("No_machine_learning_training_data_supplied_via_t_
312                  _flag",true);
313          }
314      }
315      }

```



```

301     else if(cli.hasParameter(FLAG_PREDICT)) // if a training set has been
302         {
303         // Try to load training data path (used for ML classification).
304         if (cli.hasParameter(FLAG_PREDICT))
305         {
306             predict_path = cli.getParameter(FLAG_PREDICT).getValue();
307
308             if (Common.fileExist(predict_path)) // If the file to classify
309                 is valid.
310             {
311                 if (cli.hasParameter(FLAG_MODEL))
312                 {
313                     model_path = cli.getParameter(FLAG_MODEL).getValue();
314
315                     if (Common.isPathValid(model_path)) // If the model
316                         path is valid.
317                     {
318                         predict = true;
319                     }
320                     else
321                     {
322                         log.sout("Cannot load the classifier model via the
323                             -m flag",true);
324                     }
325                 }
326                 else
327                 {
328                     log.sout("Cannot load the classifier model via the -m
329                         flag",true);
330                 }
331             }
332             else
333             {
334                 log.sout("No data to be classified supplied via -s flag",
335                     true);
336             }
337         }
338     }
339
340     else if(cli.hasParameter(FLAG_VALIDATE)) // if a validation set has
341         been provided.
342     {
343         // Try to load validation data path.
344         if (cli.hasParameter(FLAG_VALIDATE))
345         {
346             validate_path = cli.getParameter(FLAG_VALIDATE).getValue();
347
348             if (Common.fileExist(validate_path)) // If the file to classify
349                 is valid.
350             {
351                 if (cli.hasParameter(FLAG_MODEL))
352                 {
353                     model_path = cli.getParameter(FLAG_MODEL).getValue();
354
355                     if (Common.isPathValid(model_path)) // If the model
356                         path is valid.
357                     {
358                         validate = true;
359                     }
360                     else
361                     {
362                         log.sout("Cannot load the classifier model via the
363                             -m flag",true);
364                     }
365                 }
366                 else
367                 {
368                     log.sout("Cannot load the classifier model via the -m
369                         flag",true);
370                 }
371             }
372         }
373     }

```

```

347         else
348             log.sout("No data to be used for validation supplied via -
                    validate_flag", true);
349     }
350 }
351 }
352
353 /**
354  * @return the command line options for this application.
355  */
356 private static ICLI getCommandLineOptions()
357 {
358     ICLI cli = new CLI();
359
360     cli.addParameter(FLAG_VERBOSE,
361         "Verbose logging flag (optional, logging off by default)",
362         CLParameter.BOOL_PARAM_TYPE);
363
364     cli.addParameter(FLAG_VALIDATE,
365         "The path to the validation data to use to build a classifier
366         (required).",
367         CLParameter.FILEPATH_PARAM_TYPE);
368
369     cli.addParameter(FLAG_TRAINING,
370         "The path to the training data to use to build a classifier
371         (required).",
372         CLParameter.FILEPATH_PARAM_TYPE);
373
374     cli.addParameter(FLAG_MODEL,
375         "The path to the classification model to load/create (required
376         ).",
377         CLParameter.FILEPATH_PARAM_TYPE);
378
379     cli.addParameter(FLAG_PREDICT,
380         "The path to the observational data to label (required).",
381         CLParameter.FILEPATH_PARAM_TYPE);
382
383     cli.addParameter(FLAG_ALGORITHM,
384         "The algorithm to use (required).",
385         CLParameter.INT_PARAM_TYPE);
386
387     return cli;
388 }
389
390 /**
391  * Prints input parameters to the command line.
392  */
393 private static void printParameters()
394 {
395     String details = "\nAPPLICATION PARAMETERS\n";
396     details += "Verbose logging: " + verbose + "\n";
397     details += "Training set path: " + training_path + "\n";
398     details += "Prediction path: " + predict_path + "\n";
399     details += "Validation set path: " + validate_path + "\n";
400     details += "Model path: " + model_path + "\n";
401     details += "Algorithm: " + algorithm + "\n";
402     System.out.println(details);

```

```

400     }
401
402
403     /**
404      * Prints application details when beginning execution.
405      */
406     private static void printApplicationDetails()
407     {
408         String details = "\n";
409         details += "
*****\n";
410         details += "|
\n";
411         details += "|
PULSAR_CLASSIFIER_v1.0
\n";
412         details += "|
\n";
413         details += "
*****\n";
414         details += "|_Description:
\n";
415         details += "|
\n";
416         details += "|_A_machine_learning_pulsar_classification_program_
derived_from_Rob.
\n";
417         details += "|_Lyon's_LOTAASClassifier_v1.0._Can_be_used_in_ensemble_
classification.
\n";
418         details += "|_or_individual_classification_modes._This_software_can_
create
\n";
419         details += "|_classification_models_and_make_predictions_on_data_
using_those_models.
\n";
420         details += "|_Requires_Java_1.6_or_later_to_run.
\n";
421         details += "|
\n";
422         details += "
*****\n";
423         details += "|_Author:_Jacob_Ian_Matthews_&_Rob_Lyon
\n";
424         details += "|_Email:_jacob@jacobian.com.au
\n";
425         details += "|_web:_jacobianmatthews.com
\n";
426         details += "
*****\n";
427         details += "|_Required_Command_Line_Arguments:
\n";
428         details += "|
\n";
429         details += "|_Training_mode_(builds_a_new_classifier_model):
\n";
430         details += "|
\n";

```

```

431 details += "|_t_(path)_path_to_a_file_containing_training_data_in_
      ARFF_format._____|\\n";
432 details += "|_____This_is_used_to_train_the_machine_learning_
      classifier_that_|\\n";
433 details += "|_____assigns_predicted_candidate_labels._____|\\n";
434 details += "|_____\\n";
435 details += "|_m_(path)_output_directory_for_the_created_model._____|\\n";
436 details += "|_____\\n";
437 details += "|_a_(int)_the_learning_algorithm_to_build_a_model_for._
      There_are_some_|\\n";
438 details += "|_____possible_choices_listed_below:_____|\\n";
439 details += "|_____\\n";
440 details += "|_____1=_Ensemble_Classifier_(builds_all_algorithms
      )_____|\\n";
441 details += "|_____1=_J48_decision_tree_____|\\n";
442 details += "|_____2=_Multilayer_perceptron_(neural_network)_____|\\n";
443 details += "|_____3=_Naive_Bayes_____|\\n";
444 details += "|_____4=_Support_vector_machine_____|\\n";
445 details += "|_____\\n";
446 details += "|_Prediction_mode_(applies_the_classifier_to_new_data):_|\\n";
447 details += "|_____\\n";
448 details += "|_m_(path)_path_to_the_models_directory,_describing_the_
      pre-built_____|\\n";
449 details += "|_____classifier_to_use._The_model_must_have_been_
      built_using_____|\\n";
450 details += "|_____this_tool_or_WEKA._____|\\n";
451 details += "|_____\\n";
452 details += "|_p_(string)_path_to_a_file_containing_unlabelled_data_
      in_ARFF_format._____|\\n";
453 details += "|_____The_model_loaded_in_via_the_-m_flag_will_
      apply_predicted_____|\\n";
454 details += "|_____labels_to_the_data_in_this_file._____|\\n";
455 details += "|_____\\n";
456 details += "|_a_(int)_the_learning_algorithm_stored_in_the_model._____|\\n";
457 details += "|_____possible_choices_listed_below:_____|\\n";
458 details += "|_____\\n";

```

```

459 details += "|_Ensemble_Classifier_
      _
460 details += "|_J48_decision_tree_
      _
461 details += "|_Multilayer_perceptron_(neural_network)_
      _
462 details += "|_Naive_Bayes_
      _
463 details += "|_Support_vector_machine_
      _
464 details += "|_
      _
465 details += "|_Validation_mode_(checks_a_new_classifier_model):_
      _
466 details += "|_
      _
467 details += "|_m_(path)_path_to_the_models_directory,_describing_the_
      pre-built_
468 details += "|_classifier_to_use._The_models_must_have_been_
      built_using_
469 details += "|_this_tool_or_WEKA._
      _
470 details += "|_
      _
471 details += "|_v_(string)_path_to_a_file_containing_labelled_
      data_in_ARFF_
472 details += "|_format._The_model_loaded_in_via_the_-m_flag_
      _will_then_be_
473 details += "|_tested_against_the_labels_in_the_file._
      _
474 details += "|_
      _
475 details += "|_a_(int)_the_learning_algorithm_stored_in_the_model._
      _
476 details += "|_possible_choices_listed_below:_
      _
477 details += "|_
      _
478 details += "|_1=_Ensemble_Classifier_
      _
479 details += "|_J48_decision_tree_
      _
480 details += "|_2=_Multilayer_perceptron_(neural_network)_
      _
481 details += "|_3=_Naive_Bayes_
      _
482 details += "|_4=_Support_vector_machine_
      _
483 details += "|_
      _
484 details += "
      *****\
      n";
485 details += "|_Optional_Command_Line_Arguments:_
      _
486 details += "|_
      _

```

```

487     details += "|_d_(boolean)_verbose_debugging_flag.
488     details += "|
489     details += "
490         n";
491     details += "|_EXAMPLE_USAGE:
492     details += "|
493     details += "|_java_-jar_LotaasClassifier.jar_-a_1_-t_/my/file.arff_-
494     details += "|
495     details += "|_This_would_build_an_ensemble_classifier_using_the_
496     details += "|_set_with_the_'learned'_models_written_to_/models/
497     details += "|
498     details += "
499         n";
500     details += "|_License:
501     details += "|_Code_made_available_under_the_GPLv3_(GNU_General_Public
502     details += "|_allows_you_to_copy,_modify_and_redistribute_the_code_as
503     details += "|_(http://www.gnu.org/copyleft/gpl.html)._Though_a_
504     details += "|_original_author_using_the_citation_above_in_derivative_
505     details += "|_very_much_appreciated.
506     details += "
507         n";
508     System.out.println(details);
509 }
510
511 /**
512  * Prints application details to the log file.
513  */
514 private static void printLogFileHeader()
515 {
516     log.setVerbose(true);
517
518     if(log != null)
519         log.sout("Welcome_to_PULSAR_CLASSIFIER_1.0",true);
520     else

```

```

521     {
522         System.out.println("Log_file_cannot_be_initialised,_exiting...");
523         safeExit();
524     }
525 }
526
527 /**
528  * Safely exits the application and updates the log.
529  */
530 private static void safeExit()
531 {
532     log.setVerbose(true);
533     log.sout("Exiting_PULSAR_CLASSIFIER_1.0_correctly",true);
534     System.exit(0);
535 }
536
537 /**
538  * The command line flags...
539  */
540
541 private static String FLAG_VERBOSE = "-d";
542 private static String FLAG_VALIDATE = "-v";
543 private static String FLAG_TRAINING = "-t";
544 private static String FLAG_PREDICT = "-p";
545 private static String FLAG_MODEL = "-m";
546 private static String FLAG_ALGORITHM = "-a";
547 }

```

Appendix D.B ClassifierBuilder.java

ClassifierBuilder.java

```
1 package com.jacobianmatthews.pulsarclassifier;
2
3 import com.jacobianmatthews.pulsarclassifier.utils.Models;
4 import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
5 import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
6 import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
7 import com.scienceguyrob.lotaasclassifier.classifiers.offline.
    NaiveBayesTester;
8 import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
9 import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
10
11
12 /**
13  * This file is intended to build a Classification Model in four different
14  * Machine Learning algorithms
15  * from a training data set provided by the user. It can be completed in
16  * ensemble (all classifiers) by using
17  * algorithm=-1, or individually by using their respective integers.
18  *
19  * @author Jacob Ian Matthews
20  * Contact: jacob@jacobian.com.au
21  * @version 1.0, 22/05/2020
22  */
23 /* Create the Class */
24 public class ClassifierBuilder extends com.scienceguyrob.lotaasclassifier.mvc
    .ClassifierBuilder {
25
26     // CONSTRUCTORS
27     public ClassifierBuilder(BasicLogger l, String n) {
28         super(l, n);
29     }
30
31     public ClassifierBuilder(String n) {
32         super(n);
33     }
34
35     /**
36      * Builds the classifier for all algorithms with the training data set.
37      *
38      * @param algorithm set to -1 for all algorithms, otherwise use original
39      * integers for individual algorithms.
40      * @return true if all classifiers have been successfully tested and
41      *         trained.
42      */
43     public boolean build(int algorithm, String trainingSetPath, String
44         modelPath) {
45
46         // Check if user requested the ensemble classifier
47         if (algorithm == -1) {
48
49             // Create success count variable
```



```

48         int successCount = 0;
49
50         // All algorithms selected therefore loop through training and
51         testing them all
52         for (int classifier : Classifiers.classifiers) {
53             // Check if the result of building the selected classifier
54             returns true
55             if (buildClassifier(classifier, trainingSetPath, modelPath)) {
56                 // Add to the successCount as it was successful
57                 successCount++;
58             }
59         }
60
61         // Check to see if all classifiers have built successfully
62         if (successCount == Classifiers.classifiers.length) {
63             // All classifiers have been built successfully, therefore
64             return true
65             return true;
66         }
67         // Not all classifiers were successful, therefore return false
68         return false;
69     }
70 }
71
72 // Not using the ensemble classifier, build the individual
73 classifier and return the result
74 return buildClassifier(algorithm, trainingSetPath, modelPath);
75 }
76
77 }
78
79 /**
80  * This method will call the trainAndTest method on the algorithm/
81  * classifier desired.
82  *
83  * @param algorithm integer corresponding to the classifier algorithm
84  * @param trainingSetPath String corresponding to the filepath of the
85  * training data set
86  * @param modelPath String corresponding to the directory of the
87  * classifier models to be outputted
88  * @return true if the selected classifier to be built is built
89  * successfully
90  */
91 private boolean buildClassifier(int algorithm, String trainingSetPath,
92     String modelDir) {
93     switch (algorithm) {
94         case Classifiers.J48:
95             return trainAndTest(new J48Tester(log, "J48Tester"),
96                 trainingSetPath,
97                 Models.getModelFilePath(algorithm, modelDir));
98         case Classifiers.MLP:
99             return trainAndTest(new MLPTester(log, "MLPTester"),
100                 trainingSetPath,

```

```
94         Models.getModelFilePath(algorithm, modelDir));
95     case Classifiers.NB:
96         return trainAndTest(new NaiveBayesTester(log, "
97             NaiveBayesTester"), trainingSetPath,
98             Models.getModelFilePath(algorithm, modelDir));
99     case Classifiers.SVM:
100         return trainAndTest(new SVMTester(log, "SVMTester"),
101             trainingSetPath,
102             Models.getModelFilePath(algorithm, modelDir));
103     default:
104         return false;
105     }
106 }
```

Appendix D.C ClassifierValidator.java

ClassifierValidator.java

```
1 package com.jacobianmatthews.pulsarclassifier;
2
3 import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
4 import com.jacobianmatthews.pulsarclassifier.utils.Models;
5 import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
6 import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
7 import com.scienceguyrob.lotaasclassifier.classifiers.offline.
    NaiveBayesTester;
8 import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
9 import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
10 import com.scienceguyrob.lotaasclassifier.wekawrappers.I_WekaTest;
11 /**
12  * This class validates the ensemble and individual classifiers on a
13  * provided data set.
14  *
15  * @author Jacob Ian Matthews
16  * @version 1.0, 22/05/2020
17  */
18 public class ClassifierValidator extends com.scienceguyrob.lotaasclassifier.
    mvc.ClassifierValidator {
19
20     // CONSTRUCTORS
21     public ClassifierValidator(BasicLogger l, String n) {
22         super(l, n);
23     }
24
25     public ClassifierValidator(String n){ super(n);}
26
27     /**
28      *
29      * @param algorithm integer corresponding to the algorithm
30      * @param validationPath String with the filepath of the data to test the
31      * classifier on
32      * @param modelsDir String with the path of the directory containing the
33      * classifier models
34      * @return true if the classifier(s) have been successfully tested
35      */
36     public boolean validate(int algorithm, String validationPath, String
37         modelsDir)
38     {
39         // Check if using the ensemble classifier
40         if( algorithm == -1 ){
41
42             // Create a counter to keep track of the number of successful
43             // validations
44             int successCount = 0;
45
46             // Loop through the array of classifiers to validate them all
47             for(int classifier: Classifiers.classifiers)
48             {
49                 // Check for the result of the validation
50                 if( chooseClassifier(classifier, validationPath, modelsDir) )
51                 {
```

```

48         // Successful, therefore add to the counter
49         successCount++;
50     }
51 }
52
53 // Check if all classifiers validated successfully
54 if(successCount == Classifiers.classifiers.length)
55 {
56     // Return true, as validation was successful
57     return true;
58 } else {
59     // One or validations were unsuccessful, return false
60     return false;
61 }
62
63 } else {
64
65     // Make predictions on the chosen classifier and return the result
66     return chooseClassifier(algorithm, validationPath, modelsDir);
67
68 }
69
70 }
71
72 /**
73  *
74  * @param algorithm integer corresponding to the classifier.
75  * @param validationPath String containing the filepath of the testing
76  *   data
77  * @param modelDir String containing the directory containing the
78  *   classifier models
79  * @return
80  */
81 private boolean chooseClassifier(int algorithm, String validationPath,
82     String modelDir)
83 {
84     // Check which classifier needs to be validated and validate it
85     switch (algorithm)
86     {
87         case Classifiers.J48:
88             return performValidation(new J48Tester(log,"J48Tester"),
89                 algorithm, validationPath, Models.getModelFilePath(
90                     algorithm, modelDir));
91         case Classifiers.MLP:
92             return performValidation(new MLPTester(log,"MLPTester"),
93                 algorithm, validationPath, Models.getModelFilePath(
94                     algorithm, modelDir));
95         case Classifiers.NB:
96             return performValidation(new NaiveBayesTester(log,"
97                 NaiveBayesTester"), algorithm, validationPath, Models.
98                 getModelFilePath(algorithm, modelDir));
99         case Classifiers.SVM:
100             return performValidation(new SVMTester(log,"SVMTester"),
101                 algorithm, validationPath, Models.getModelFilePath(
102                     algorithm, modelDir));
103         default:
104             return false;
105     }
106 }

```

```

94     }
95
96 }
97
98 /**
99  *
100  * @param classifier a WEKA classifier
101  * @param algorithm integer corresponding to the chosen algorithm
102  * @param validationPath string containing the path to the data to test
103  *      with
104  * @param modelPath string containing the path to the classifier's model
105  * @return
106  */
107 private boolean performValidation(I_WekaTest classifier, int algorithm,
108     String validationPath, String modelPath)
109 {
110     // Check for successful loading and validation of the model
111     boolean loaded = classifier.loadModel(modelPath);
112     boolean validated = classifier.validate(validationPath);
113
114     // Get the name of the classifier
115     String classifierName = Classifiers.getClassifierName(algorithm);
116
117     // Output the results of the validations
118     if(loaded & validated)
119         return true;
120     else
121     {
122         if(!loaded)
123         {
124             log.sout("Could_not_load_the_" + classifierName + "_classifier_"
125                 + modelPath, true);
126             return false;
127         }
128         else if(!validated)
129         {
130             log.sout("Could_not_validate_the_" + classifierName + "_model",
131                 true);
132             return false;
133         }
134     }
135 }
136 }
137
138 }

```

Appendix D.D ClassPredictor.java

ClassPredictor.java

```
1 package com.jacobianmatthews.pulsarclassifier;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import com.jacobianmatthews.pulsarclassifier.utils.Classification;
6 import com.jacobianmatthews.pulsarclassifier.utils.ClassificationList;
7 import com.jacobianmatthews.pulsarclassifier.utils.Classifiers;
8 import com.jacobianmatthews.pulsarclassifier.utils.Models;
9 import com.scienceguyrob.lotaasclassifier.classifiers.offline.J48Tester;
10 import com.scienceguyrob.lotaasclassifier.classifiers.offline.MLPTester;
11 import com.scienceguyrob.lotaasclassifier.classifiers.offline.
    NaiveBayesTester;
12 import com.scienceguyrob.lotaasclassifier.classifiers.offline.SVMTester;
13 import com.scienceguyrob.lotaasclassifier.io.Writer;
14 import com.scienceguyrob.lotaasclassifier.utils.BasicLogger;
15 import com.scienceguyrob.lotaasclassifier.wekawrappers.I_WekaTest;
16
17 /**
18  * This class contains the methods and properties required to make predictions
19  * on data
20  * to classify pulsars.
21  *
22  * @author Jacob Ian Matthews
23  * @version 1.0, 24/05/20
24  */
25 public class ClassPredictor extends com.scienceguyrob.lotaasclassifier.mvc.
    ClassPredictor {
26
27     // CONSTRUCTORS
28     public ClassPredictor(BasicLogger l, String n) { super(l, n); }
29     public ClassPredictor(String n){ super(n); }
30
31     public boolean predict(int algorithm, String predictPath, String
        modelsDir)
32     {
33         // Check if the ensemble classifier is being used
34         if ( algorithm == -1 )
35         {
36             // Get the prefix of the prediction output files (input file path
37             // without extension)
38             String fileName = predictPath.substring(0, predictPath.lastIndexOf
39             ("."));
40
41             // Create an array of the [output].positive and [output].negative
42             // files
43             List<String> positiveFiles = new ArrayList<String>();
44             List<String> negativeFiles = new ArrayList<String>();
45
46             // Create an index to count the number of classifiers successfully
47             // completing predictions
48             int predictCount = 0;
```

```

46 // Loop through the array of classifiers to make all predictions
47 for(int classifier: Classifiers.classifiers)
48 {
49     // Check the status of the classifier's predictions
50     if ( chooseClassifier(classifier, predictPath, modelsDir) )
51     {
52         // Returned true, therefore add to prediction count
53         predictCount++;
54     }
55
56     // Get the name of the classifier
57     String classifierName = Classifiers.getClassifierName(
58         classifier);
59
60     // Re-create the output file's name
61     String outputName = fileName+"_"+classifierName;
62
63     // Add the output file's name to the array of file names
64     positiveFiles.add(outputName+".positive");
65     negativeFiles.add(outputName+".negative");
66 }
67
68 // Check that all classifiers completed predictions
69 if ( predictCount == Classifiers.classifiers.length )
70 {
71     // All classifiers have finished making predictions, now we can
72     // create the final
73     // classification output file by combining the positive pulsar
74     // predictions of all classifiers.
75     // According to Tan et al. (2017), having positive
76     // classifications in 3 separate classifiers
77     // indicates a more accurate classification.
78
79     // Create the ensemble classifier's output file names
80     String ensemblePositive = fileName+"_ensemble.positive";
81     String ensembleNegative = fileName+"_ensemble.negative";
82
83     // Create some flags to denote the ensemble classification
84     // process completed successfully
85     boolean positiveSuccess = false;
86     boolean negativeSuccess = false;
87
88     // Create a list containing the postitive classifications and a
89     // count of their occurrences
90     ClassificationList positiveList = new ClassificationList();
91
92     // Create a list containing the postitive classifications and a
93     // count of their occurrences
94     ClassificationList negativeList = new ClassificationList();
95
96     // Build the list of positive classifications using the outputs
97     // from all the classifiers
98     if( positiveList.buildList(positiveFiles) ){
99
100         // We now have a list of positive candidate classifications
101         // and we can apply

```

```

94      // the cutoff of 3 separate positive classifications for
          the ensemble classifier.
95
96      // Loop through the list of positive classifications
97      for( Classification positive: positiveList.getList() )
98      {
99          // Check the classification key-value pair for a value
              >=3
100          if( positive.getValue() > 2 )
101          {
102              // Try add it to the ensemble positive output file
103              if ( !Writer.append(ensemblePositive, positive.
                  getKey()+"\n" ) )
104              {
105                  log.sout("Couldn't add "+positive.getKey()+" to
                      positive_ensemble_classifier_output_file.",
                          true);
106              }
107          } else {
108
109              // Add it to the negative classification list as it
                  didn't survive the cutoff
110              negativeList.add(positive.getKey(), positive.
                  getValue());
111          }
112
113      }
114      // Flag that the process was successful
115      positiveSuccess = true;
116
117  } else {
118
119      // Return false and log the error
120      log.sout("Ensemble_classifier_positive_candidates_list_
          couldn't be compiled.", true);
121      return false;
122  }
123
124  // Build a list of negative classifications from all
          classifiers
125  if ( negativeList.buildList(negativeFiles) )
126  {
127      // Loop through the list to produce the negative output
          file
128      for (Classification negative: negativeList.getList() )
129      {
130          // Get the key of the classification
131          String key = negative.getKey();
132
133          // Make sure that it isn't already in the positive or
          negative output files
134          if( !Reader.checkStringIsInFile(ensembleNegative, key)
              && !Reader.checkStringIsInFile(ensemblePositive,
                  key))
135          {
136              // Append it to the output file
137              if( !Writer.append(ensembleNegative, key+"\n" ) )

```



```

138         {
139             // Log the error
140             log.sout("Couldn't add "+key+" to the negative_
                ensemble_classifier_output_file.", true);
141
142         }
143
144     }
145 }
146
147 // Flag that the process was successful
148 negativeSuccess = true;
149
150 } else {
151     // Log the error and return false
152     log.sout("Ensemble_classifier_negative_candidates_list_
        couldn't be compiled.", true);
153 }
154
155 // Check to see if both processes were successful
156 if( positiveSuccess && negativeSuccess )
157 {
158     // Return true to denote that it worked
159     return true;
160 } else {
161
162     return false;
163 }
164
165 } else {
166
167     // Return false as the ensemble classifier didn't complete
168     log.sout("Not all classifiers completed predictions in_
        ensemble_classifier._Cannot produce final classifications_
        file.", true);
169     return false;
170 }
171
172 } else {
173
174     // The ensemble classifier isn't being used, choose individual
175     classifier
176     return chooseClassifier(algorithm, predictPath, modelsDir);
177 }
178
179 }
180
181 private boolean chooseClassifier(int algorithm, String predictPath,
    String modelDir)
182 {
183     // Determine which classifier to make predictions with
184     switch (algorithm)
185     {
186     case Classifiers.J48:
187         return makePredictions(new J48Tester(log,"J48Tester"),
            algorithm, predictPath, Models.getModelFilePath(algorithm,

```

```

188         modelDir));
189     case Classifiers.MLP:
190         return makePredictions(new MLPTester(log,"MLPTester"),
191             algorithm, predictPath, Models.getModelFilePath(algorithm,
192                 modelDir));
193     case Classifiers.NB:
194         return makePredictions(new NaiveBayesTester(log,"
195             NaiveBayesTester"), algorithm, predictPath, Models.
196             getModelFilePath(algorithm, modelDir));
197     case Classifiers.SVM:
198         return makePredictions(new SVMTester(log,"SVMTester"),
199             algorithm, predictPath, Models.getModelFilePath(algorithm,
200                 modelDir));
201     default:
202         return false;
203 }
204 }
205
206 private boolean makePredictions(I_WekaTest classifier, int algorithm,
207     String predictPath, String modelPath)
208 {
209     // Get the Input Data's file name without the extension
210     final String file = predictPath.substring(0, predictPath.lastIndexOf(".")
211         .));
212
213     // Get the name of the classifier
214     final String classifierName = Classifiers.getClassifierName(algorithm)
215         ;
216
217     // Create the name of the Classifier's output file
218     final String outputName = file+"_"+classifierName;
219
220     // Check if the classifier's model loaded
221     final boolean loaded = classifier.loadModel(modelPath);
222
223     // Check if the predictions were made
224     final boolean predicted = classifier.predict(predictPath, outputName);
225
226     // Check the results of the booleans
227     if(loaded & predicted)
228         return true;
229     else
230     {
231         if(!loaded)
232         {
233             log.sout("Could_not_load_the_"+classifierName+"_classifier_
234                 model",true);
235             return false;
236         }
237         else if(!predicted)
238         {
239             log.sout("Could_not_make_predictions_using_the_"+
240                 classifierName+"_classifier_model",true);
241             return false;
242         }
243     }
244     else
245     {

```

```
233         log.sout("Could not predict using the "+classifierName+"  
234                 classifier model",true);  
235         return false;  
236     }  
237  
238 }  
239  
240 }
```

Appendix D.E utils/Classification.java

utils/Classification.java

```
1  package com.jacobianmatthews.pulsarclassifier.utils;
2
3
4  /**
5   * This class is a key-value pair datatype.
6   *
7   * @author Jacob Ian Matthews
8   * @version 1.0, 24/02/20
9   *
10  */
11  public class Classification
12  {
13      // Variables
14      private String key;
15      private int value;
16
17      // Constructor
18      public Classification(String key, int value)
19      {
20          // Get the values
21          this.key = key;
22          this.value = value;
23      }
24
25
26      public int getValue()
27      {
28          return this.value;
29      }
30
31      public String getKey()
32      {
33          return this.key;
34      }
35
36      public void setValue(int v)
37      {
38          this.value = v;
39      }
40
41      public void setKey(String k)
42      {
43          this.key = k;
44      }
45
46  }
```

Appendix D.F utils/ClassificationList.java

utils/ClassificationList.java

```
1  package com.jacobianmatthews.pulsarclassifier.utils;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import com.scienceguyrob.lotaasclassifier.io.Reader;
7
8  /**
9   * This class creates a list with a key-value pairing system to use inside a
10  * list. It is used as a part of ClassPredictor.java to keep track of the
11  * number
12  * of occurrences of positive and negative pulsar classifications.
13  *
14  * @author Jacob Ian Matthews
15  * @version 1.0, 24/05/20
16  */
17  public class ClassificationList
18  {
19
20      /**
21       * Variables
22       */
23      public List<Classification> list;
24
25      /**
26       * Constructor
27       */
28      public ClassificationList()
29      {
30          // Create the list
31          this.list = new ArrayList<Classification>();
32      }
33
34      /**
35       *
36       * @param index index in the list of the item to retrieve
37       * @return the Key-Value pair at the list index
38       */
39      public Classification get(int index)
40      {
41          // Return the list item
42          return list.get(index);
43      }
44
45      public List<Classification> getList()
46      {
47          return this.list;
48      }
49
50      /**
51       *
52       * @return the number of items in the list
```

```

53     */
54     public int size()
55     {
56         // Get the size of the list
57         return list.size();
58     }
59
60
61     public void add(String key, int value)
62     {
63         // Create a new entry and add it to the list
64         list.add(new Classification(key, value));
65     }
66
67
68     /**
69      * Get the value of a key-value pair inside the list, found by its key.
70      * @param key String with the key of the pair
71      * @return Integer value of the pair or 0 if it doesn't exist.
72      */
73     public int getValueByKey(String key)
74     {
75
76         // Loop through the list until the pair is found
77         for(Classification item: this.list)
78         {
79             // Check the string against the provided string
80             if( item.getKey().equals(key) )
81             {
82                 // Found the classification item, get its value
83                 return item.getValue();
84             }
85         }
86
87         // Return 0 since it wasn't found
88         return 0;
89     }
90
91
92     /**
93      * Sets the value of a key-value pair inside the list, by its key.
94      * @param key String key of the pair.
95      */
96     public void setValueByKey(String key, int value)
97     {
98         // Loop through the list until the key-value pair is found
99         for(Classification item: this.list)
100         {
101             // Compare the string keys
102             if ( item.getKey().equals(key) )
103             {
104                 // Found the pair, updated the value
105                 item.setValue(value);
106             }
107         }
108     }
109 }

```

```

110
111  /**
112   * Builds an ensemble list of positive or negative classification
113   * instances from all classifiers.
114   * @param files a List of filepaths to the positive or negative classifier
115   * outputs
116   * @return true if build is successful
117   */
118  public boolean buildList(List<String> files)
119  {
120      // Loop through the classifiers' output files to count instances of
121      candidate classifications
122      for(String file: files)
123      {
124          // Get the number of lines in the file
125          int lineCount = Reader.getLineCount(file);
126
127          // Get the contents of each line and handle them individually
128          for(int i=1; i<=lineCount; i++)
129          {
130              // Read the line of the file
131              String line = Reader.readLine(file, i);
132
133              // Check if ensemble candidates list is empty
134              if ( this.size() == 0 ){
135
136                  // List is empty, no point in checking for previous
137                  occurrences of this line
138                  this.add(line, 1);
139
140              } else {
141
142                  // List isn't empty, check if this classification is
143                  already in the list
144                  int occurrences = this.getValueByKey(line);
145
146                  // If it isn't 0, then it has already showed up and
147                  therefore we can add to its total count
148                  if (occurrences > 0) {
149
150                      // Add to the value
151                      this.setValueByKey(line, occurrences+1);
152
153                  } else {
154
155                      // Occurrences are 0, therefore we can add this key-
156                      value pair to the list
157                      this.add(line, 1);
158
159                  }
160              }
161          }
162      }
163      return true;
164  }

```

```

160
161     /**
162     *
163     * @return a string containing lines of all key value classification pairs
164     */
165     public String printList()
166     {
167         // Create an empty string to print to the command line
168         String output = "";
169
170         // Loop through the list items and print their key and value
171         for(Classification item: this.list)
172         {
173             // Get the key and value of the classification
174             String key = item.getKey();
175             int value = item.getValue();
176
177             // Add to the output string
178             output+=key+" "+value+"\n";
179         }
180
181         // Return the output string
182         return output;
183     }
184 }
185
186 }

```


Appendix D.G utils/Classifiers.java

utils/Classifiers.java

```
1  package com.jacobianmatthews.pulsarclassifier.utils;
2
3
4  /**
5   * This class contains additional methods and properties relating to the
6   *   Classifiers.
7   * @author Jacob Ian Matthews
8   * @version 1.0, 23/05/20
9   */
10 public class Classifiers extends com.scienceguyrob.lotaasclassifier.
11     classifiers.Classifiers {
12
13     // An array of the classifiers
14     public static final int[] classifiers = {J48, MLP, NB, SVM};
15 }
```

Appendix D.H utils/Models.java

utils/Models.java

```
1  package com.jacobianmatthews.pulsarclassifier.utils;
2
3  import com.scienceguyrob.lotaasclassifier.utils.Common;
4
5  /**
6   * The class Models contains methods to create the path to each classifier's
7     model.
8   *
9   * @author Jacob Ian Matthews
10  * @version 1.0, 23/05/20
11  */
12  public class Models extends Common {
13
14      /**
15       * This method converts a directory modelDir of which to output the
16       * classifier models, into individual
17       * model filepaths for each classifier.
18       *
19       * @param algorithm an integer relating to the classifier algorithm
20       * @param modelDir String containing the directory to place the classifier
21       * models
22       * @return String with the filepath of the model of the classifier.
23       */
24      public static String getModelFilePath(int algorithm, String modelDir)
25      {
26          // Check if the provided path is a valid directory
27          if(isDirectory(modelDir)) {
28
29              // Return the model's filepath
30              return createModelFilePath(algorithm, modelDir);
31
32          } else {
33              // Create the directory as it doesn't exist
34              if (dirCreateRecursive(modelDir)) {
35                  // Return the model's filepath
36                  return createModelFilePath(algorithm, modelDir);
37
38              } else {
39                  // Couldn't create the directory, return a null filepath
40                  return null;
41              }
42          }
43      }
44
45      /**
46       *
47       * @param algorithm An integer corresponding to the chosen algorithm.
48       * @param modelDir A string containing the path of the directory of models
49       * @return A string with a filepath to a classifier's model
50       */
51      private static String createModelFilePath(int algorithm, String modelDir)
52      {
53          // Create the classifier's model's file name
```

```

51     String fileName = createModelFileName(algorithm);
52
53     // Check if the directory path ends in a forward slash
54     if (modelDir.endsWith("/")) {
55         // Don't add an extra forward slash and return the full path
56         return modelDir + fileName;
57     } else {
58         // Add a slash to the end of the path
59         return modelDir + "/" + fileName;
60     }
61
62 }
63
64 /**
65  *
66  * @param algorithm An integer corresponding to the chosen algorithm
67  * @return Returns a string with the filename for the chosen algorithm
68  */
69 private static String createModelFileName(int algorithm)
70 {
71     // Get the algorithm/classifier's name
72     String classifierName = Classifiers.getClassifierName(algorithm);
73
74     // Return a file name dependent of the classifier's name
75     return classifierName+"_model.m";
76 }
77
78
79
80 }

```