Finding New Pulsars Using Machine Learning

Jacob Ian Matthews

Draft

May 30, 2020

Abstract

Contents

1 Introduction

1.1 Aim

This project consists of three aims:

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

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

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

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 what has been dubbed 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 undertook in: (i) developing the machine learning training dataset for the Murchison Widefield Array algorithm, and (ii) evaluating the machine learning algorithm used by the LOFAR surveys for use with the Murchison Widefield Array.

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 efficacy of the training datasets, and (ii) the usefulness of the LOFAR machine learning algorithm with the Murchison Widefield Array and why changes were made to the algorithm.

This report will end with my final conclusions on the use of machine learning in discovering new pulsars (*Section 5*), and my recommendations to future researchers undertaking a similar project (*Section 6*).

1.3 Background Theory

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:

$$e^{-} + p \rightarrow n + \nu_e$$

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).

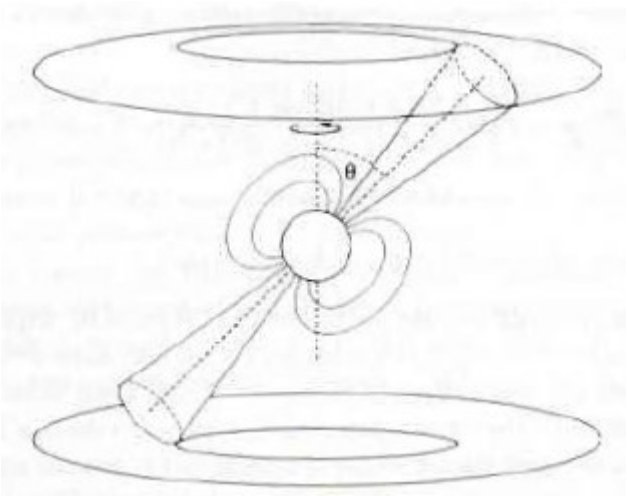

Figure 1: A pulsar (Maoz 2016).

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.

1.3.1 What is a Pulsar Profile?

1.3.2 What is multi-path scattering?

1.3.3 What is a Pulsar Dispersion Measure?

1.3.4 What is a Radio Telescope and how do they work?

1.3.5 What is the Murchison Widefield Array?

1.3.6 Why are we conducting sky surveys?

1.3.7 How is a sky survey conducted with the Murchison Widefield Array?

1.3.8 What is tied-array beamforming?

1.3.9 How many Pulsar candidates are found in a Murchison Widefield Array sky survey?

1.3.10 How long do sky surveys take?

1.3.11 How much data is created from a sky survey?

1.3.12 What is Radio Frequency Interference?

1.3.13 What is a Signal-to-Noise Ratio?

1.3.14 What is PRESTO and a .PFD file?

1.3.15 What is Machine Learning?

1.3.16 How does Machine Learning work?

1.3.17 Why do we need to use Machine Learning in finding new Pulsars?

1.3.18 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:

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

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

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

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

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

To extract the 20 above features from each classification candidate, we can use the software `PulsarFeatureLab` (Lyon et al. 2016).

2.1.1 Pulsar Candidate Feature Extraction

The Python software tool `PulsarFeatureLab` can be used to consume pulsar candidate files of the PRESTO Prepfold `PFD` filetype and output the 20 above features for each candidate into a single file of WEKA Data Mining `ARFF` filetype (Lyon et al. 2016).

To create a closed software environment in which the dependencies of the `PulsarFeatureLab` software are unaffected by the host operating system, a containerised virtual operating system can be created using the free software Docker (`https://docker.com`).

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
```

Candidate `PFD` files of known pulsars and non-pulsars detected by the Murchison Widefield Array (MWA) provided by N. Swainston 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.

2.1.2 Creating the Training Dataset

The `output.arff` file created by `PulsarFeatureLab`, contains a set of comma-separated features for each candidate, with an appended '?' character, per file line. Since the class of each candidate is already known, the '?' character on each line can be replaced by a '1' if the candidate is a pulsar, or a '0' if the candidate is a non-pulsar. This signals to the Machine Learning algorithm what a pulsar and a non-pulsar candidate's feature set may appear like.

The edited file can now be renamed and moved with the following command:

```
$ mv ~/pulsars/data/pfd/output.arff ~/pulsars/data/
    trainingSet.arff
```

The Machine Learning Training Dataset has now been created using Murchison Widefield Array data.

2.2 Evaluating the LOTAASClassifier Machine Learning Classification Tool for use with the Murchison Widefield Array

To address the Candidate Selection Problem discussed in *Section 1.3* of this report, a Java Machine Learning pulsar classification tool named `LOTAASClassifier` was created to classify the pulsar candidates produced by the LOFAR Tied-Array All-sky Survey (LOTAAS) (Lyon et al. 2016). Due to the similarities between the LOFAR radio telescope and the Murchison Widefield Array (MWA) radio telescope, it is logical to attempt to apply the `LOTAASClassifier` tool to candidates produced by the MWA.

The `LOTAASClassifier` tool contains four machine learning algorithms with which the user can choose to make pulsar classifications. They are: the J48 (C4.5 Decision Tree (Quinlan 1987)), the Multi-Layer Perceptron, the Naive Bayes, and the Support Vector Machines (SVM) algorithms (Lyon et al. 2016).

To begin evaluating the software, `LOTAASClassifier v1.0` can be downloaded from its GitHub repository

(`https://github.com/scienceguyrob/LOTAASClassifier`) with the following UNIX terminal commands:

```
$ cd ~/pulsars
$ git clone https://github.com/scienceguyrob/LOTAASClassifier
    .git
```

The tool's executable program can then be found by navigating to the following directory in the UNIX terminal:

```
$ cd ~/pulsars/LOTAASClassifier/dist
```

2.2.1 Creating a Pulsar Classifier Output Validation Tool

In order to accurately evaluate and test the predictions made by the `LOTAASClassifier` tool, we must create a program that will automate the process of checking the outputs of the classifier against the known list of pulsars.

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 consume a user inputted `String` containing the path to a file with the list of pulsars included in the dataset classified by the machine learning classifier, a user inputted `String` containing the path to the `.positive` file created by the classifier, and a user inputted `String` containing the path

to the `.negative` file created by the classifier[1]. These will be inputted as command-line arguments when the user runs the Java executable file.

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

Algorithm 1 getCliVariables(args) (pseudocode)

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

We can then use a simple conditional statement after this function is ran to check if the `Boolean ValidationMode` has been set to `true`, to determine whether to continue the validation. The complete Java class `PulsarValidator.java` can be seen in *Appendix 7.2.1.*

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 2 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`.

2.2.2 Creating a Classification Model

In order to use `LOTAASClassifier` to classify Murchison Widefield Array candidates, we must use the Machine Learning Training Dataset created in *Section 2.1* to create a classification model. This can be completed by running the following commands:

```
$ java -jar LOTAASClassifier.jar -t ~/pulsars/data/
    trainingSet.arff -m ~/pulsars/data/model.m -a 1
```

This instructs `LOTAASClassifier` to create a new classification model for the J48 machine learning algorithm with the previously constructed training dataset.

To ensure that the classification model was created successfully, we can test the model against the same dataset provided by N. Swainston to create the training dataset. Using `PulsarFeatureLab` we can produce a new `output.arff` file with the candidates' class unedited.

```
$ 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
```

It is now possible to test the classification model with the candidate feature sets with the following commands:

```
$ cd ~/pulsars/LOTAASClassifier/dist
$ java -jar LOTAASClassifier.jar -p ~/pulsars/data/pfd/output
    .arff -m ~/pulsars/data/model.m -a 1
```

The `LOTAASClassifier` tool will output two files: '`output.positive`' and '`output.negative`', in the same location as the input dataset. The J48 classification model was constructed successfully if the candidates inside the '`positive`' file are the known pulsars.

2.2.3 Evaluating LOTAASClassifier

With a successfully created machine learning classification model, it is now possible to test `LOTAASClassifier` against a previously unseen (by the classifier) dataset of Murchison Widefield Array candidates whose class is also known. We can first remove the existing candidates from inside the `PFD` files directory and delete the existing `output` files.

The candidates directory can then be populated by new `PFD` files and we can use the '`docker run`' command from *Section 2.2.1* to extract the features from the new candidates.

The `LOTAASClassifier` tool can now make classification predictions on the new candidates by running the UNIX terminal command:

```
$ cd ~/pulsars/LOTAASClassifier/dist
$ java -jar LOTAASClassifier.jar -p ~/pulsars/data/pfd/output
    .arff -m ~/pulsars/data/model.m -a 1
```

This can be repeated for each machine learning classifier present in the `LOTAASClassifier` software by changing the argument `-a` from 1 through to 4.

We can now run the `PulsarValidator` program created in *Section 2.2.1* to evaluate the accuracy of `LOTAASClassifier`'s algorithms by running the command:

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

Where the `[classifier]` variable can be changed depending on the name of the classifier algorithm being evaluated. The outputted validation statistics can then be compared across the algorithms to determine their usefulness with the Murchison Widefield Array.

2.2.4 Creating the PulsarClassifier Ensemble Classification Tool

According to Tan et al. 2017, using Machine Learning algorithms in ensemble to make pulsar classifications increases the accuracy of classifications, classifying pulsars that were often misclassified such as wide-pulse pulsars.

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

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

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

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

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

pom.xml

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

We now have the following basic project directory structure:

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

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

The `LOTAASClassifier` tool accepts a command-line argument `-a` which accepts an integer that denotes the machine learning algorithm to use in building a classification model and making predictions (Lyon et al. 2016). Therefore, we

will add an algorithm into the above listed Java classes that will accept an integer value of `-1` that will activate the ensemble classifier.

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

Algorithm 3 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
```

See Section 8.4.1 for 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 4 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
```

See Section 8.4.2 for 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 5 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 Section 8.4.3* 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

Section 8.4.4. The complete source code to the `PulsarClassifier` contains the following classes:

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

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

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

pom.xml

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

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

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

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

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

The complete source code and build of `PulsarClassifier` can be found at `https://github.com/jacob-ian/PulsarClassifier.git`, or in *Appendix 7.3*.

2.2.5 Evaluating PulsarClassifier

Now that we have successfully built `PulsarClassifier`, we can evaluate its usage with the Murchison Widefield Array's (MWA) candidates. We begin 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. We can validate the ensemble classifier with the `PulsarValidator` program created in *Section 2.2.1* 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.

3 Results and Outputs

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 7.1*.

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
```

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
```

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
```

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
```

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: 165
Pulsars Detected: 112
True Positives: 112
False Positives: 0

Number of Non-Pulsars: 87
Non-Pulsars Detected: 140
True Negatives: 87
False Negatives: 53
```

4 Discussion and Conclusions

4.1 How effective is the training dataset?

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

4.3 Why did/didn't the LOFAR Machine Learning Algorithm work with the MWA?

4.4 What changes to the algorithm were necessary for it to successfully classify Pulsars from the MWA?

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. In the `PFD` datasets that I used, the only commonality between the failed feature extraction candidates was that the candidates were not pulsars - they were either examples of Radio Frequency Interference (RFI) or Noise. The result of these failures was that there was a smaller dataset to train the machine learning classifiers on, or 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.

Quinlan, J. R. 1987. "Decision Trees as Probabilistic Classifiers." *Proceedings of the Fourth International Workshop of Machine Learning* 1:31–37.

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.

7 Appendices

7.1 Machine Learning Training Dataset

trainingData.arff

```
@relation Pulsar_Feature_Data_Type_6
@attribute Feature_1 numeric
@attribute Feature_2 numeric
@attribute Feature_3 numeric
@attribute Feature_4 numeric
@attribute Feature_5 numeric
@attribute Feature_6 numeric
@attribute Feature_7 numeric
@attribute Feature_8 numeric
@attribute Feature_9 numeric
@attribute Feature_10 numeric
@attribute Feature_11 numeric
@attribute Feature_12 numeric
@attribute Feature_13 numeric
@attribute Feature_14 numeric
@attribute Feature_15 numeric
@attribute Feature_16 numeric
@attribute Feature_17 numeric
@attribute Feature_18 numeric
@attribute Feature_19 numeric
@attribute Feature_20 numeric
@attribute class {0,1,2}
@data
105.43918879794592,34.81562200057764,0.64091523062507,2.999965982775474,4.9731636,0.55617476,0.73770590325867
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM106.85
    _ACCEL_0_2_23_29_01.45_-10_35_14.03_8577.99ms_Cand_rfi.pfd
19.439267094649395,37.83110112790925,5.311899765657869,27.72625393618159,49.398266,17.023054,0.75823017779194
    usr/src/app/PulsarFeatureLab/Data/IO/1252780888_100_bins_PSR_0152-1637.
    pfd
18.18808199226157,29.34228539923202,5.972780846625737,41.45065223740317,31.696558,10.691088,0.632813129945887
    usr/src/app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_0459-0210.
    pfd
21.22497781504873,33.50944234920476,4.928435748454679,26.392961286010472,19.536982,9.97172,1.0080046130377358
    usr/src/app/PulsarFeatureLab/Data/IO/1253471952_100_bins_PSR_0255-5304.
    pfd
125.23524447122634,56.134362050023924,0.2229079772063587,-0.6216367448361071,11.638162,0.4708962,-0.932350032
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM13.36
    _ACCEL_0_5_23_40_05.66_-13_34_55.98_591.37ms_Cand_noise.pfd
89.58174363355785,48.4446505060987,1.1788897000809406,2.1990474951218184,6.3838625,1.0415499,0.57757212475454
    usr/src/app/PulsarFeatureLab/Data/IO/1252177744_100_bins_PSR_0151-0635.
    pfd
24.10863128259143,28.805247892957368,5.8820714970090355,41.4163908926057,26.94988,5.6278524,0.587453916304551
    usr/src/app/PulsarFeatureLab/Data/IO/1253991112_100_bins_PSR_0206-4028.
    pfd
19.659448914180896,33.71258916374586,6.212422938671684,38.92028743782727,33.78987,11.8825655,0.61393884706171
    usr/src/app/PulsarFeatureLab/Data/IO/1255803168_100_bins_PSR_0401-7608.
    pfd
130.37194427538753,48.5719656104399,0.03110721286401433,-0.28624231198914396,3.4748905,0.32001108,0.544493918
    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
```

```
14.60760000762955,33.37521977816117,5.2251889142177985,29.534224184342584,80.282196,26.19628,0.51782067247501
    usr/src/app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_J0450-1248.
    pfd
19.160516821154644,52.612510321908914,3.498625248210871,10.92536155890088,221.91383,63.068035,0.4634763573806
    usr/src/app/PulsarFeatureLab/Data/IO/1255197408_100_bins_PSR_0452-1759.
    pfd
105.40739512137863,42.046982247914656,1.105401957592074,2.71606611188946,9.959542,1.508026,-0.107286435139737
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM124.60
    _ACCEL_0_5_23_30_05.03_-15_32_28.86_7306.99ms_Cand_rfi.pfd
108.62704623126228,48.50808371245384,0.3888816698242244,0.10828707118695347,3.2382581,0.57282144,0.5061148662
    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
30.65638094885709,36.39390906613557,4.334831666786916,20.661559138936177,22.137678,7.129584,0.583874112231887
    usr/src/app/PulsarFeatureLab/Data/IO/1254594264_100_bins_PSR_0304+1932.
    pfd
110.46206868842188,46.251025957184034,0.37602640993908826,0.6004748040886194,3.5859008,0.48099223,0.954763129
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM133.85
    _ACCEL_0_4_23_34_32.65_-11_35_39.59_6359.79ms_Cand_rfi.pfd
67.96984095457957,30.670603367741087,2.9112873304761235,15.54008032933561,10.704723,3.0320911,-0.031139309413
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM119.85
    _ACCEL_0_2_23_29_31.44_-15_47_03.39_13154.44ms_Cand_rfi.pfd
118.82100466581987,47.54490692614183,0.07781892865435884,-0.01638792599233385,3.4399548,0.8459567,0.703248063
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM137.35
    _ACCEL_0_6_23_43_56.97_-11_20_36.48_5794.50ms_Cand_rfi.pfd
106.55367062734199,38.31922288316345,1.1201921873325247,3.524621849432644,4.6642995,1.4868807,1.2001879564459
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM128.85
    _ACCEL_0_2_23_35_06.34_-10_20_02.01_13136.61ms_Cand_rfi.pfd
110.58047755262913,56.835988222404985,0.2597416198749472,-0.6009060312856693,6.2209997,1.3479991,0.0301769380
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM124.10
    _ACCEL_0_12_23_36_13.35_-15_17_52.85_590.99ms_Cand_noise.pfd
22.63765924905709,32.25698418159722,5.017978110756053,28.558061328289384,23.729832,7.0639696,0.65822770848595
    usr/src/app/PulsarFeatureLab/Data/IO/1256407632_100_bins_PSR_J0450-1248.
    pfd
113.87621446459325,43.41151017852675,0.5893561441860531,1.71756136954156,5.65921,1.1683735,1.0193316046807013
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM119.35
    _ACCEL_0_2_23_29_35.58_-09_49_30.86_10373.16ms_Cand_rfi.pfd
116.41310084274251,46.52577061051363,0.1525172365016325,0.14280165135512535,3.0154004,0.65890557,1.4567341719
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM106.85
    _ACCEL_0_2_23_37_20.08_-15_17_52.85_9862.48ms_Cand_rfi.pfd
115.05147692422577,59.943352711870936,0.06373023728528525,-0.5571994750116955,5.8320494,0.6752133,0.100135170
    usr/src/app/PulsarFeatureLab/Data/IO/1222697776_DM114.60
    _ACCEL_0_6_23_42_50.44_-11_20_36.48_3514.62ms_Cand_rfi.pfd
13.688631621147106,29.885241623656917,6.142618064042354,42.30058250077879,42.656235,14.500364,0.6562798650382
    usr/src/app/PulsarFeatureLab/Data/IO/1256407632_100_bins_PSR_0459-0210.
    pfd
```

7.2 PulsarValidator

7.2.1 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
        file
10  * of PulsarFeatureLab. It will then be possible to filter the list of known
11  * pulsars in the original dataset to the pulsars that had successful feature
12  * extraction. Finally, the program can compare the list of known pulsars in
        the
13  * dataset to the pulsars classified in the classifier, and produce comparison
14  * statistics.
15  *
16  * @author Jacob Ian Matthews
17  * @version 1.0, 28/05/2020
18  */
19 public class PulsarValidator {
20
21     /**
22      *
23      * VARIABLES
24      *
25      */
26
27     // Pulsar list mode
28     private static boolean list = false;
29
30     // Pulsar classification output validation mode
31     private static boolean validation = false;
32
33     // String containaing the path to the PulsarFeatureLab output file
34     private static String pflOutput;
35
36     // String containing the path to the list of pulsars
37     private static String pulsarList;
38
39     // String containing the path to the positive classifier output
40     private static String positiveClassifier;
41
42     // String containing the path to the negative classifier output
43     private static String negativeClassifier;
44
45     public static void main(String[] args) throws IOException {
46         // Get the input arguments
47         getCliVariables(args);
48
49         // 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("'-l'␣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 }
```

7.2.2 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.
        It
14  * will compare the output files of the Pulsar Classifier to the true list of
15  * pulsars and output statistics.
16  *
17  * @author Jacob Ian Matthews
18  * @version 1.0, 29/05/2020
19  */
20 public class ValidationMode {
21
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
           pulsars
34      * in the data set.
35      * @param postiveOutputPath A string containing the path to the positive
           output
36      * file of the classifier.
37      * @param negativeOutputPath A string containing the path to the negative
           output
38      * file of the classifier.
39      * @throws IOException
40      */
41     public ValidationMode(String pulsarListPath, String positiveOutputPath,
          String negativeOutputPath)
42             throws IOException {
43
44         // Assign the variables
45         this.positiveOutputPath = positiveOutputPath;
46         this.negativeOutputPath = negativeOutputPath;
47
48         // 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 containg 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
                their
121      * results
122      *
123      * @return a StatisticList object containing the postiive 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 
178         // Count the number of true and false positives
179         int truePositiveCount = truePositive.size();
180         int falsePositiveCount = falsePositive.size();
181 
182         // Add the statistics to the list
183         statistics.add("TruePositives", truePositiveCount);
184         statistics.add("FalsePositives", falsePositiveCount);
185 
186         // Return the list
187         return statistics;
188 
189     }
190 
191     /**
192      * This creates the false and true negative lists and output the results
                of
193      * their statistics
194      *
195      * @return a StatisticList containing the negative statistics
196      * @throws IOException
197      */
198     private StatisticList processNegative(StatisticList statistics) throws
            IOException
199     {
200         // Create the False Negative List
201         List<String> falseNegative = new ArrayList<String>();
202 
203         // Create the True Negative List
204         List<String> trueNegative = new ArrayList<String>();
205 
206         // Get the negative output list
207         List<String> classifier = Files.readAllLines(Paths.get(this.
                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 }
```

7.2.3 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 }
```

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

7.2.5 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 }
```

7.3 PulsarClassifier

7.3.1 PulsarClasifier.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     private static String predict_path = "";
39
40     /**
41      * Full path to the classifier validation file (ARFF file).
42      */
43     private static String validate_path = "";
44
45     /**
46      * Full path to the file containing the classification model to use.
47      */
48     private static String model_path = "";
49
50     /**
```

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

```
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
                            .", true);
159                 } else {
160                     log.sout("Ensemble␣classifier␣was␣unsuccessful␣in␣applying␣
                            predictions.", true);
161                 }
162             } else {
163 
164                 if( result )
165                 {
166                     log.sout(Classifiers.getClassifierName(algorithm)+ "␣
                            classifier␣made␣predictions␣successfully", true);
167 
168                 } else {
169                     log.sout(Classifiers.getClassifierName(algorithm)+ "␣
                            classifier␣unsuccessful␣in␣applying␣predictions", true
                            );
170                 }
171 
172             }
173 
174 
175         }
176         else if(validate)
177         {
178             log.sout("Attempting␣to␣validate␣existing␣classifier␣performance",
                      true);
179 
180             ClassifierValidator cv = new ClassifierValidator(log,"
                    ClassifierValidator");
181 
182             // Validate the classifier chosen
183             boolean result = cv.validate(algorithm,validate_path,model_path);
184 
185             // Output the results for the ensemble classifier
186             if(algorithm == -1) {
187                 if(result) {
188                     log.sout("Ensemble␣classifier␣validated␣successfully.",
                            true);
189                 } else {
190                     log.sout("Ensemble␣classifier␣couldn't␣be␣valdated␣
                            successfully.", true);
191                 }
192 
193             // Output the results for the individual classifiers
194             } else {
195                 if(result) {
196                     log.sout(Classifiers.getClassifierName(algorithm)+ "␣
                            classifier␣validated␣successfully", true);
197                 } else {
198                     log.sout(Classifiers.getClassifierName(algorithm)+ "␣
                            classifier␣validation␣unsuccessful", true);
199                 }
```

```
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
                classifier, or
247          * classify new data.
248          *
249          * Option one requires the full path to a training set file, the
                integer identifier
250          * of the classifier to build, and the path to write the
                classification model to.
251          *
252          * Option two requires the full path to a file requiring predictions,
                the integer
```

```
253          * identifier of the classifier used to make the predictions, and the
                 path to the
254          * classification model.
255          *
256          * Below we make sure this logic is correct.
257          */
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.
                      getParameter(FLAG_ALGORITHM).toInt() < 6)
269                 algorithm = cli.getParameter(FLAG_ALGORITHM).toInt();
270             else
271             {
272                 log.sout("Algorithm␣value␣supplied␣via␣-a␣flag␣invalid␣(must␣
                          be␣-1␣to␣5).", true);
273                 return;
274             }
275
276         if(cli.hasParameter(FLAG_TRAINING)) // if a training set has been
                  provided.
277         {
278             // Try to load training data path (used for ML classification).
279             if (cli.hasParameter(FLAG_TRAINING))
280             {
281                 training_path = cli.getParameter(FLAG_TRAINING).getValue();
282
283                 if (Common.fileExist(training_path)) // If the training set is
                          valid.
284                 {
285                     if (cli.hasParameter(FLAG_MODEL))
286                     {
287                         model_path = cli.getParameter(FLAG_MODEL).getValue();
288
289                         if (Common.isPathValid(model_path)) // If the training
                                  set is valid.
290                             train = true;
291                         else
292                             log.sout("Model␣output␣path␣specified␣via␣-m␣flag␣
                                      invalid",true);
293                     }
294                     else
295                         log.sout("No␣output␣model␣path␣supplied␣with␣-m␣flag",
                                  true);
296                 }
297                 else
298                     log.sout("No␣machine␣learning␣training␣data␣supplied␣via␣-t
                              ␣flag",true);
299             }
300         }
```

```
301         else if(cli.hasParameter(FLAG_PREDICT)) // if a training set has been
                provided.
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
                        is valid.
309                 {
310                     if (cli.hasParameter(FLAG_MODEL))
311                     {
312                         model_path = cli.getParameter(FLAG_MODEL).getValue();
313
314                         if (Common.isPathValid(model_path)) // If the model
                                path is valid.
315                             predict = true;
316                         else
317                             log.sout("Cannot␣load␣the␣classifier␣model␣via␣the␣
                                    -m␣flag",true);
318                     }
319                     else
320                         log.sout("Cannot␣load␣the␣classifier␣model␣via␣the␣-m␣
                                flag",true);
321                 }
322                 else
323                     log.sout("No␣data␣to␣be␣classified␣supplied␣via␣-s␣flag",
                            true);
324             }
325         }
326         else if(cli.hasParameter(FLAG_VALIDATE)) // if a validation set has
                been provided.
327         {
328             // Try to load validation data path.
329             if (cli.hasParameter(FLAG_VALIDATE))
330             {
331                 validate_path = cli.getParameter(FLAG_VALIDATE).getValue();
332
333                 if (Common.fileExist(validate_path)) // If the file to classify
                        is valid.
334                 {
335                     if (cli.hasParameter(FLAG_MODEL))
336                     {
337                         model_path = cli.getParameter(FLAG_MODEL).getValue();
338
339                         if (Common.isPathValid(model_path)) // If the model
                                path is valid.
340                             validate = true;
341                         else
342                             log.sout("Cannot␣load␣the␣classifier␣model␣via␣the␣
                                    -m␣flag",true);
343                     }
344                     else
345                         log.sout("Cannot␣load␣the␣classifier␣model␣via␣the␣-m␣
                                flag",true);
346                 }
```

```
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␣
                        (required).",
366                 CLParameter.FILEPATH_PARAM_TYPE);
367 
368         cli.addParameter(FLAG_TRAINING,
369                 "The␣path␣to␣the␣training␣data␣to␣use␣to␣build␣a␣classifier␣(
                        required).",
370                 CLParameter.FILEPATH_PARAM_TYPE);
371 
372         cli.addParameter(FLAG_MODEL,
373                 "The␣path␣to␣the␣classification␣model␣to␣load/create␣(required
                        ).",
374                 CLParameter.FILEPATH_PARAM_TYPE);
375 
376         cli.addParameter(FLAG_PREDICT,
377                 "The␣path␣to␣the␣observational␣data␣to␣label␣(required).",
378                 CLParameter.FILEPATH_PARAM_TYPE);
379 
380         cli.addParameter(FLAG_ALGORITHM,
381                 "The␣algorithm␣to␣use␣(required).",
382                 CLParameter.INT_PARAM_TYPE);
383 
384         return cli;
385     }
386 
387     /**
388      * Prints input parameters to the command line.
389      */
390     private static void printParameters()
391     {
392         String details = "\nAPPLICATION␣PARAMETERS\n";
393         details += "Verbose␣logging␣␣␣␣:␣" + verbose + "\n";
394         details += "Training␣set␣path␣␣:␣" + training_path + "\n";
395         details += "Prediction␣path␣␣␣␣:␣" + predict_path + "\n";
396         details += "Validation␣set␣path:␣" + validate_path + "\n";
397         details += "Model␣path␣␣␣␣␣␣␣␣:␣" + model_path + "\n";
398         details += "Algorithm␣␣␣␣␣␣␣␣␣:␣" + algorithm + "\n";
399         System.out.println(details);
```

```java

    }

    /**
     * Prints application details when beginning execution.
     */
    private static void printApplicationDetails()
    {
        String details = "\n";
        details += "**************************************************************************\n";
        details += "|                                                                        |\n";
        details += "|                          PULSAR CLASSIFIER v1.0                        |\n";
        details += "|                                                                        |\n";
        details += "**************************************************************************\n";
        details += "| Description:                                                           |\n";
        details += "|                                                                        |\n";
        details += "| A machine learning pulsar classification program derived from Rob.     |\n";
        details += "| Lyon's LOTAASClassifier v1.0. Can be used in ensemble classification   |\n";
        details += "| or individual classification modes. This software can create           |\n";
        details += "| classification models and make predictions on data using those models. |\n";
        details += "| Requires Java 1.6 or later to run.                                     |\n";
        details += "|                                                                        |\n";
        details += "**************************************************************************\n";
        details += "| Author: Jacob Ian Matthews & Rob Lyon                                  |\n";
        details += "| Email : jacob@jacobian.com.au                                          |\n";
        details += "| web   : jacobianmatthews.com                                           |\n";
        details += "**************************************************************************\n";
        details += "| Required Command Line Arguments:                                       |\n";
        details += "|                                                                        |\n";
        details += "| Training mode (builds a new classifier model):                         |\n";
        details += "|                                                                        |\n";
```

```
details += "|␣-t␣(path)␣path␣to␣a␣file␣containing␣training␣data␣in␣ARFF␣format.␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣This␣is␣used␣to␣train␣the␣machine␣learning␣classifier␣that␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣assigns␣predicted␣candidate␣labels.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣-m␣(path)␣output␣directory␣for␣the␣created␣model.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣-a␣(int)␣the␣learning␣algorithm␣to␣build␣a␣model␣for.␣There␣are␣some␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣possible␣choices␣listed␣below:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣-1␣=␣Ensemble␣Classifier␣(builds␣all␣algorithms)␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣1␣=␣J48␣decision␣tree␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣2␣=␣Multilayer␣perceptron␣(neural␣network)␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣3␣=␣Naive␣Bayes␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣4␣=␣Support␣vector␣machine␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣Prediction␣mode␣(applies␣the␣classifier␣to␣new␣data):␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣-m␣(path)␣path␣to␣the␣models␣directory,␣describing␣the␣pre-built␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣classifier␣to␣use.␣The␣model␣must␣have␣been␣built␣using␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣this␣tool␣or␣WEKA.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣-p␣(string)␣path␣to␣a␣file␣containing␣unlabelled␣data␣in␣ARFF␣format.␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣The␣model␣loaded␣in␣via␣the␣-m␣flag␣will␣apply␣predicted␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣labels␣to␣the␣data␣in␣this␣file.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣-a␣(int)␣the␣learning␣algorithm␣stored␣in␣the␣model.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣possible␣choices␣listed␣below:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
```

```
459         details += "|␣␣␣␣␣␣␣␣␣-1␣=␣Ensemble␣Classifier␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
460         details += "|␣␣␣␣␣␣␣␣␣␣1␣=␣J48␣decision␣tree␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
461         details += "|␣␣␣␣␣␣␣␣␣␣2␣=␣Multilayer␣perceptron␣(neural␣network)␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
462         details += "|␣␣␣␣␣␣␣␣␣␣3␣=␣Naive␣Bayes␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
463         details += "|␣␣␣␣␣␣␣␣␣␣4␣=␣Support␣vector␣machine␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
464         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
465         details += "|␣Validation␣mode␣(checks␣a␣new␣classifier␣model):␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
466         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
467         details += "|␣-m␣(path)␣path␣to␣the␣models␣directory,␣describing␣the␣
                pre-built␣␣␣␣␣␣␣|\n";
468         details += "|␣␣␣␣␣␣␣␣␣␣␣classifier␣to␣use.␣The␣models␣must␣have␣been␣
                built␣using␣␣␣␣␣|\n";
469         details += "|␣␣␣␣␣␣␣␣␣␣␣this␣tool␣or␣WEKA.␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
470         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
471         details += "|␣-v␣␣␣␣␣␣␣␣␣(string)␣path␣to␣a␣file␣containing␣labelled␣
                data␣in␣ARFF␣␣␣␣␣␣|\n";
472         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣format.␣The␣model␣loaded␣in␣via␣the␣-m␣flag
                ␣will␣then␣be␣␣␣␣|\n";
473         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣tested␣against␣the␣labels␣in␣the␣file.␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
474         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
475         details += "|␣-a␣(int)␣the␣learning␣algorithm␣stored␣in␣the␣model.␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
476         details += "|␣␣␣␣␣␣␣␣␣␣possible␣choices␣listed␣below:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
477         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
478         details += "|␣␣␣␣␣␣␣␣␣-1␣=␣Ensemble␣Classifier␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
479         details += "|␣␣␣␣␣␣␣␣␣␣1␣=␣J48␣decision␣tree␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
480         details += "|␣␣␣␣␣␣␣␣␣␣2␣=␣Multilayer␣perceptron␣(neural␣network)␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
481         details += "|␣␣␣␣␣␣␣␣␣␣3␣=␣Naive␣Bayes␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
482         details += "|␣␣␣␣␣␣␣␣␣␣4␣=␣Support␣vector␣machine␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
483         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
484         details += "
                **********************************************************************\
                n";
485         details += "|␣Optional␣Command␣Line␣Arguments:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
486         details += "|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|\n";
```

```
487         details += "| -d (boolean) verbose debugging flag.                    
                 |\n";
488         details += "|                                                         
                 |\n";
489         details += "
                 ************************************************************\
                 n";
490         details += "|                                                         
                 |\n";
491         details += "| EXAMPLE USAGE:                                          
                 |\n";
492         details += "|                                                         
                 |\n";
493         details += "| java -jar LotaasClassifier.jar -a -1 -t /my/file.arff -
                 m /my/models/    |\n";
494         details += "|                                                         
                 |\n";
495         details += "| This would build an ensemble classifier using the 
                 supplied training     |\n";
496         details += "| set with the 'learned' models written to /models/       
                 |\n";
497         details += "|                                                         
                 |\n";
498         details += "
                 ************************************************************\
                 n";
499         details += "| License:                                                
                 |\n";
500         details += "|                                                         
                 |\n";
501         details += "| Code made available under the GPLv3 (GNU General Public
                 License), that |\n";
502         details += "| allows you to copy, modify and redistribute the code as
                 you see fit     |\n";
503         details += "| (http://www.gnu.org/copyleft/gpl.html). Though a 
                 mention to the        |\n";
504         details += "| original author using the citation above in derivative 
                 works, would be |\n";
505         details += "| very much appreciated.                                  
                 |\n";
506         details += "
                 ************************************************************\
                 n";
507
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 }
```

7.3.2 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
        Machine Learning algorithms
14  * from a training data set provided by the user. It can be completed in
        ensemble (all classifiers) by using
15  * algorithm=-1, or individually by using their respective integers.
16  *
17  * @author Jacob Ian Matthews
18  * Contact: jacob@jacobian.com.au
19  * @version 1.0, 22/05/2020
20  *
21  */
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
            trained.
41      */
42     public boolean build(int algorithm, String trainingSetPath, String
           modelPath) {
43
44         // Check if user requested the ensemble classifier
45         if (algorithm == -1) {
46
47             // Create success count variable
```

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

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

7.3.3 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
            classifier on
31      * @param modelsDir String with the path of the directory containing the
            classifier models
32      * @return true if the classifier(s) have been successfully tested
33      */
34     public boolean validate(int algorithm, String validationPath, String
           modelsDir)
35     {
36         // Check if using the ensemble classifier
37         if( algorithm == -1 ){
38
39             // Create a counter to keep track of the number of successful
                   validations
40             int successCount = 0;
41
42             // Loop through the array of classifiers to validate them all
43             for(int classifier: Classifiers.classifiers)
44             {
45                 // Check for the result of the validation
46                 if( chooseClassifier(classifier, validationPath, modelsDir) )
47                 {
```

```
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
         data
76   * @param modelDir String containing the directory containing the
         classifier models
77   * @return
78   */
79  private boolean chooseClassifier(int algorithm, String validationPath,
       String modelDir)
80  {
81      // Check which classifier needs to be validated and validate it
82      switch (algorithm)
83      {
84          case Classifiers.J48:
85              return performValidation(new J48Tester(log,"J48Tester"),
                    algorithm, validationPath, Models.getModelFilePath(
                    algorithm, modelDir));
86          case Classifiers.MLP:
87              return performValidation(new MLPTester(log,"MLPTester"),
                    algorithm, validationPath, Models.getModelFilePath(
                    algorithm, modelDir));
88          case Classifiers.NB:
89              return performValidation(new NaiveBayesTester(log,"
                    NaiveBayesTester"), algorithm, validationPath, Models.
                    getModelFilePath(algorithm, modelDir));
90          case Classifiers.SVM:
91              return performValidation(new SVMTester(log,"SVMTester"),
                    algorithm, validationPath, Models.getModelFilePath(
                    algorithm, modelDir));
92          default:
93              return false;
```

```
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
             with
103      * @param modelPath string containing the path to the classifier's model
104      * @return
105      */
106     private boolean performValidation(I_WekaTest classifier, int algorithm,
             String validationPath, String modelPath)
107     {
108         // Check for successful loading and validation of the model
109         boolean loaded = classifier.loadModel(modelPath);
110         boolean validated = classifier.validate(validationPath);
111
112         // Get the name of the classifier
113         String classifierName = Classifiers.getClassifierName(algorithm);
114
115         // Output the results of the validations
116         if(loaded & validated)
117             return true;
118         else
119         {
120             if(!loaded)
121             {
122                 log.sout("Could␣not␣load␣the␣"+classifierName+"␣classifier␣
                        model",true);
123                 return false;
124             }
125             else if(!validated)
126             {
127                 log.sout("Could␣not␣validate␣the␣"+classifierName+"␣model",
                        true);
128                 return false;
129             }
130             else
131             {
132                 log.sout("Could␣not␣perform␣validation␣on␣the␣"+classifierName
                        +"␣classifier␣model",true);
133                 return false;
134             }
135         }
136     }
137
138 }
```

7.3.4 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
          on data
19  * to classify pulsars.
20  *
21  * @author Jacob Ian Matthews
22  * @version 1.0, 24/05/20
23  */
24 public class ClassPredictor extends com.scienceguyrob.lotaasclassifier.mvc.
       ClassPredictor {
25
26     // CONSTRUCTORS
27     public ClassPredictor(BasicLogger l, String n) { super(l, n); }
28     public ClassPredictor(String n){ super(n); }
29
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
                   without extension)
37             String fileName = predictPath.substring(0, predictPath.lastIndexOf
                   (("."));
38
39             // Create an array of the [output].positive and [output].negative
                   files
40             List<String> positiveFiles = new ArrayList<String>();
41             List<String> negativeFiles = new ArrayList<String>();
42
43             // Create an index to count the number of classifiers successfully
                   completing predictions
44             int predictCount = 0;
45
```

```
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(
                        classifier);
58
59                  // Re-create the output file's name
60                  String outputName = fileName+"_"+classifierName;
61
62                  // Add the output file's name to the array of file names
63                  positiveFiles.add(outputName+".positive");
64                  negativeFiles.add(outputName+".negative");
65
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
                        create the final
72                  // classification output file by combining the positive pulsar
                        predictions of all classifiers.
73                  // According to Tan et al. (2017), having positive
                        classifications in 3 separate classifiers
74                  // indicates a more accurate classification.
75
76                  // Create the ensemble classifier's output file names
77                  String ensemblePositive = fileName+"_ensemble.positive";
78                  String ensembleNegative = fileName+"_ensemble.negative";
79
80                  // Create some flags to denote the ensemble classification
                        process completed successfully
81                  boolean positiveSuccess = false;
82                  boolean negativeSuccess = false;
83
84                  // Create a list containing the postitive classifications and a
                        count of their occurrences
85                  ClassificationList positiveList = new ClassificationList();
86
87                  // Create a list containing the postitive classifications and a
                        count of their occurrences
88                  ClassificationList negativeList = new ClassificationList();
89
90                  // Build the list of positive classifications using the outputs
                        from all the classifiers
91                  if( positiveList.buildList(positiveFiles) ){
92
93                      // We now have a list of positive candidate classifications
                            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                     // Append it to the output file
134                     if( !Writer.append(ensembleNegative, key+"\n") )
135                     {
136                         // Log the error
137                         log.sout("Couldn't␣add␣"+key+"␣to␣the␣negative␣
                                ensemble␣classifier␣output␣file.", true);
138
139                     }
```

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

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

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

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

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

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

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

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

    }

```

```
235   }
```

7.3.5 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 }
```

7.3.6 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
        number
11  * of occurrences of positive and negative pulsar classifications.
12  *
13  * @author Jacob Ian Matthews
14  * @version 1.0, 24/05/20
15  *
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
             instances from all classifiers.
113      * @param files a List of filepaths to the positive or negative classifier
             outputs
114      * @return true if build is successful
115      */
116     public boolean buildList(List<String> files)
117     {
118         // Loop through the classifiers' output files to count instances of
                candidate classifications
119         for(String file: files)
120         {
121             // Get the number of lines in the file
122             int lineCount = Reader.getLineCount(file);
123
124             // Get the contents of each line and handle them individually
125             for(int i=1; i<=lineCount; i++)
126             {
127                 // Read the line of the file
128                 String line = Reader.readLine(file, i);
129
130                 // Check if ensemble candidates list is empty
131                 if ( this.size() == 0 ){
132
133                     // List is empty, no point in checking for previous
                            ocurrences of this line
134                     this.add(line, 1);
135
136                 } else {
137
138                     // List isn't empty, check if this classification is
                            already in the list
139                     int occurrences = this.getValueByKey(line);
140
141                     // If it isn't 0, then it has already showed up and
                            therefore we can add to its total count
142                     if (occurrences > 0) {
143
144                         // Add to the value
145                         this.setValueByKey(line, occurrences+1);
146
147                     } else {
148
149                         // Occurrences are 0, therefore we can add this key-
                                value pair to the list
150                         this.add(line, 1);
151
152                     }
153                 }
154             }
155
156         }
157         return true;
158
159     }
```

```
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 }
```

7.3.7 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
        Classifiers.
6   *
7   * @author Jacob Ian Matthews
8   * @version 1.0, 23/05/20
9   */
10 public class Classifiers extends com.scienceguyrob.lotaasclassifier.
       classifiers.Classifiers {
11
12    // An array of the classifiers
13    public static final int[] classifiers = {J48, MLP, NB, SVM};
14
15 }
```

7.3.8 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
        model.
7   *
8   * @author Jacob Ian Matthews
9   * @version 1.0, 23/05/20
10  */
11 public class Models extends Common {
12
13    /**
14     * This method converts a directory modelDir of which to output the
            classifier models, into individual
15     * model filepaths for each classifier.
16     *
17     * @param algorithm an integer relating to the classifier algorithm
18     * @param modelDir String containing the directory to place the classifier
            models
19     * @return String with the filepath of the model of the classifier.
20     */
21    public static String getModelFilePath(int algorithm, String modelDir)
22    {
23        // Check if the provided path is a valid directory
24        if(isDirectory(modelDir)) {
25
26            // Return the model's filepath
27            return createModelFilePath(algorithm, modelDir);
28
29        } else {
30            // Create the directory as it doesn't exist
31            if (dirCreateRecursive(modelDir)) {
32                // Return the model's filepath
33                return createModelFilePath(algorithm, modelDir);
34
35            } else {
36                // Couldn't create the directory, return a null filepath
37                return null;
38            }
39        }
40    }
41
42    /**
43     *
44     * @param algorithm An integer corresponding to the chosen algorithm.
45     * @param modelDir A string containing the path of the directory of models
46     * @return A string with a filepath to a classifier's model
47     */
48    private static String createModelFilePath(int algorithm, String modelDir)
49    {
50        // 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  }
```