COMP 401 – Project in Biology and Computer Science

Final Report

Supervised by: Dr. Thomas Bureau & Mr. Emilio Vello

Lea Collin, 260618407

12/05/17

Terms to know: unswitched case: case where training dataset has 185 instances and testing dataset has 908 instances (original way datasets were formed); switched case: case where training dataset has 908 instances and testing dataset has 185 instances; ranked case: case where training dataset has 908 instances with attributes ranked and testing dataset has 185 instances

1. Introduction

Climate change is becoming an ever-growing problem. The increase in global temperatures will lead to more drought events and some areas of the world are expected to experience reductions in their crop yields. For these areas, plant breeding programs and agricultural technology must be developed (Vello et al., 2015).

The use of different phenomics technologies in plants is a key element to improve our knowledge of the genotype-phenotype association of desired agricultural traits such as the response to water deficit (Vello et al., 2015). In the McGill Plant Phenomics Platform (MP3), research is being conducted in the hopes of better understanding plant response to water deficit, among others. The MP3 is instrumental in conducting this research. The MP3 has two main components, the 3D system and the Hts systems. The 3D system has the following cameras: 2 visible light, 2 neo-infrared, and 2 infrared, consisting of top and side views for each type of camera. This allows the phenomics lab to capture most of the characteristics of the plant. The system has a lifter with two different positions, high and low, and it can take images at single degree differences, ie 0 degrees, 1 degree, 2 degrees, etc. A conveyor belt transports the plants and the cabinets in which the plants are held have different positions that can rotate. There is a weighing system which allows for the tracking of automatic watering of the plant. The system can be programmed to keep the plant at a certain watering level or absolute weight value, eg. the plant will always receive 100 mL of water or the plant will receive water until it weighs 300 g (including the weight of the carrier). MP3 is located in the McGill greenhouse in the Stewart Biology building.

In this research, soybeans were studied because they are economically important. As the meat-packing industry is known to have very negative effects on the environment, soybeans have become even more important for the environmental sustainability of the future. Because soybean products are ingredients in many meat and dairy substitutes and soybeans are an exceptional source of essential nutrients, the shift from meat to soybeans may be essential (Arnarson et al., 2015). Learning more about soybeans would better allow us to effectively use it as a meat and dairy substitute and alleviate some negative environmental effects.

Machine learning and image analysis are important in this field so that we can better predict the effect that climate change will have on crop production and better prepare for it (Vello et al., 2015, Stavness et al., 2017). With the ability to instantly analyze different plants grown under different watering conditions, we will be able to get a better sense of what needs to be done in order to still meet the global demand for plant agriculture.

2. Background

2.1. Image Processing

All of the necessary image analysis was performed by Emilio Vello and the processed image data was stored in the MP3 soybean database. Figure 1 shows an example of the image analysis pipeline that was performed prior to this research.

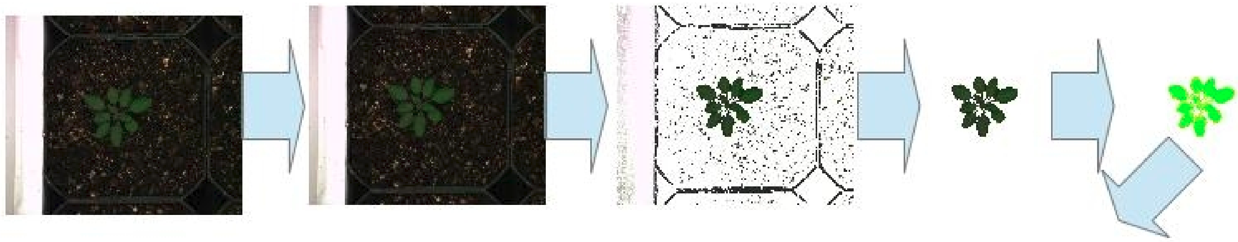


Figure 1 – Image analysis pipeline performed prior to research and stored in arabidopsis database; From “Image Analysis Pipeline”, by Emilio Vello, 2017, http://mp3.biol.mcgill.ca/mcgill_mp3_analysis.html. Copyright 2017 by Emilio Vello. Reprinted with permission.

The final image in Figure 1 shows the image from which the data is taken and stored in the phenomics lab database.

2.2. Weka

Machine learning has an ever-growing important role in many fields, including image-based plant phenotyping. Weka is a collection of state-of-the-art machine learning algorithms (Frank and Witten, 1999). It was developed at the University of Waikato in New Zealand and the name stands for *Waikato Environment for Knowledge Analysis (*Frank and Witten, 1999). The system is written in Java and runs on almost any platform. Weka provides implementations of learning algorithms that one can easily apply to a dataset. All of the algorithms take their input in ARFF format which can be read from a file or generated by a database query (Frank and Witten, 1999). For this research, the database queries were saved as ARFF files and then read in by the program. One way of using Weka is to use learned models to generate predictions on new instances, which is the way that Weka was used for this research (Frank and Witten, 1999). Besides just training and building models based on a given training dataset, Weka also has several methods for “Attribute Selection”. These different “Attribute Selection” methods read the training dataset and determine which attributes in the dataset are most important in predicting the developmental stage of the plant. One of these methods ranks the attributes by level of importance and these rankings can then be used while training and building the classification models. Weka is available through an easy-to-use graphical user interface or directly in code through the Java Weka library.

3. Methods

3.1. Software

During this research, a ‘Weka Application’ Java project was developed consisting of one class and several methods, that will be used by Dr. Thomas Bureau and Mr. Emilio Vello in their phenomics lab. For simplicity of building an initial program that Mr. Vello can later build upon, the developmental stage of the plant was based on its DAS (days after seeding). The different das were split into 5 categories, each containing 10 values and the developmental stages were then assigned to one of these 5 categories (Stage 1, Stage 2, Stage 3, Stage 4, or Stage 5). For example, a plant who was 34 days after seeding was considered to be in Stage 4.

This class provides an easy way to query the image data from the phenomics lab’s soybean database, write the data to an ARFF file that is understandable to Weka methods, build and train classification models provided by the Java Weka library, and test these models on a testing dataset, which also comes from the soybean database. The jdbc drivers were used to be able to write SQL queries within the class and the Java Weka library was used to directly import Weka methods. The Weka library provided all of the machine learning classification algorithms that were trained and tested within the class. The program simply reads the training data, builds a classification model based on this data and the name of the model inputted by the user (or builds the program's default model options). The program then runs on the testing dataset and calculates a precision of the model. All of the predictions of the different classification algorithms tested were outputted to a CSV file which also contained the barcode, das, and true stage of the plant. Another CSV is produced that stores the name of each algorithm tested and their respective precision value. The precision value was simply calculated as the number of instances that the model classifies correctly divided by the total number of instances within the dataset. These precision values matched those reported by Weka when performing the training and testing of the models directly within the Weka GUI. The CSV containing the stage prediction of each plant was then analyzed in R.

Both the Java code and R analysis code can be viewed at the end of this report.

3.2. Datasets and Tests

The training and testing sets were formed by querying the phenomics lab's soybean database. The public schema was used as were the dasplusev, imageev, imgobjectev, and soyidentification tables. These tables allowed the joining of the identification of the plant along with all of the important image characteristics that would be used to determine the developmental stage of the plant. The original training data only looked at the vis-side-1-0 camera view , line 1, and set 3. The original testing data also only looked at the vis-side-1-0 camera, lines 1, 2, and 3 from set 2 and lines 1 and 2 from set 3. It is important to only choose one camera for training and testing otherwise this will lead to non-comparable image views. The area of the plant from an aerial view will be calculated differently than the area of the plant from a side view. Therefore, it is important to be consistent with the type of camera when querying the data. The data selected additionally only included "well-watered" plants as opposed to also including "drought" plants. This is because the developmental stage of plants in these different conditions vary greatly. The term 'original' training data refers to the fact that this data is what was initially used to build the program (this will later on in the paper be referred to as the 'unswitched case', as described in 'terms to know' above). The testing data was later on used as the training data and vice versa to see if there would be any improvement in precision (this will later on in the paper be referred to as the 'switched case', as described in 'terms to know' above).

3.3. Approach

The user was left with the option of inputting which classification algorithms they want to try. This was possible due to the "AbstractClassifier" class within the Weka library. The "AbstractClassifier" class would instantiate any classifier object simply based on the name of the classifier. This prevented the necessity of importing every classifier available from Weka regardless of user input. An array list of strings containing all of the possible classification algorithms was created in order to ensure that the user input was valid. This array list is also the 'default' set of classification algorithms that the program will run if the user decides not to input any algorithms on their own. Because the datasets are fairly small, it only took a few seconds for all of the models to be built and tested. The user was also allowed to decide whether they wanted to rank the attributes in the datasets to test if there would be any improvement in precision from doing so.

4. Results

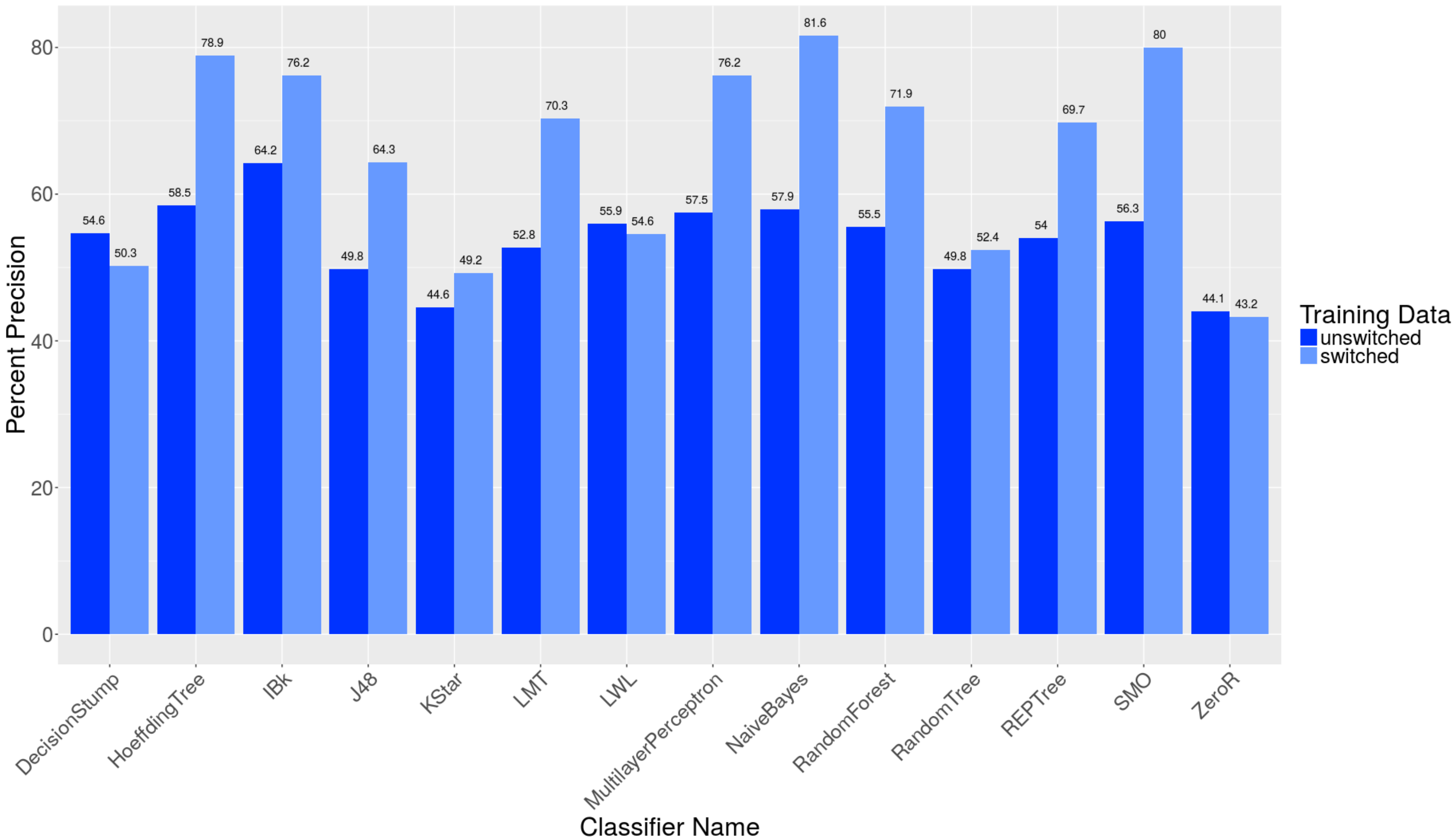
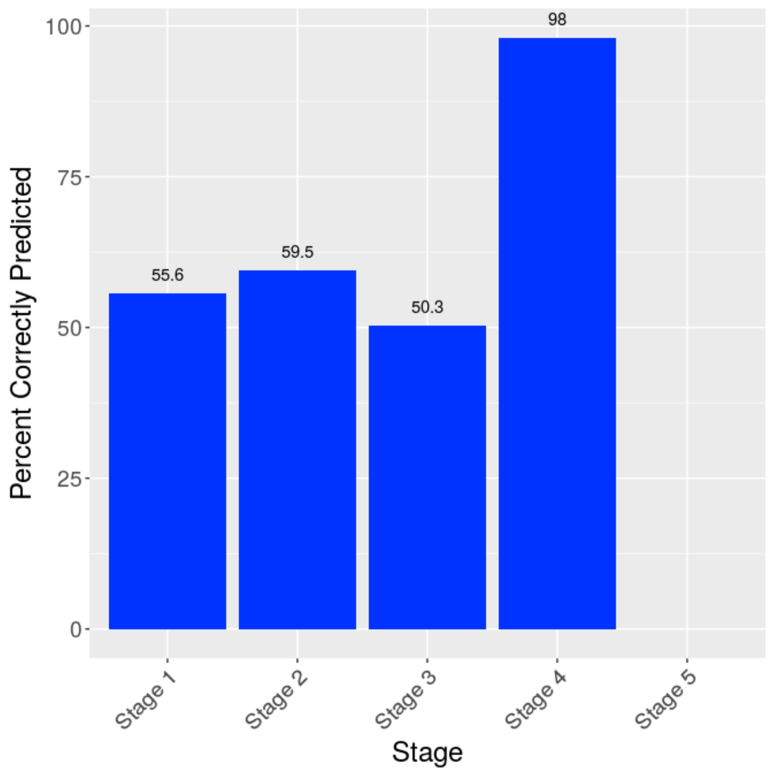


Figure 2 - All default algorithms tested with their precisions graphed side by side with the unswitched (dark blue) and switched (light blue) case

The precision values of all of the default algorithms are visible in Figure 2. The 'unswitched' bars in the figure refer to the 'original' organization of the training and testing data as described in 'terms to know' and Methods above. The 'switched' bars therefore refer to switching which dataset was used for training versus which dataset was used for testing. The following algorithms had better precision with the 'unswitched' datasets: ZeroR, LWL, and Decision Stump. The following algorithms had better precision with the 'switched' datasets: NaiveBayes, MultilayerPerceptron, SMO, IBk, KStar, HoeffdingTree, J48, LMT, RandomForest, RandomTree, and REPTree. Therefore, 11 algorithms performed better when the larger dataset was used for training and only 3 algorithms performed better when the smaller dataset was used for training.Overall, the best classification algorithm was NaiveBayes, in the 'switched' case, with 81.6% precision.



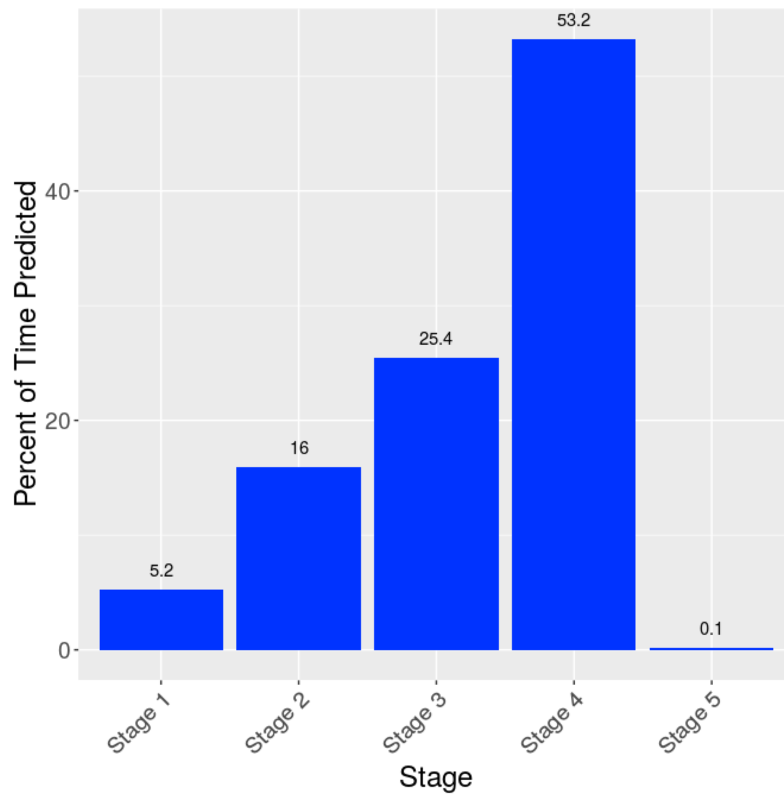


Figure 3A – Number of times each stage was correctly predicted as a percentage of the total number of times each stage was predicted; Sum of number of times each classifier correctly predicted the stage was calculated and divided by total number of times each stage was predicted at all; Average over all default algorithms except for ZeroR

Figure 3B – Frequency that each stage was predicted; Number of times each stage was predicted by each classifier divided by total number of prediction made; Average over all default algorithms except for ZeroR

If we look at just the 'switched' case, ie with the larger dataset used for training and the smaller used for testing, Figure 3A shows the number of times that each stage was correctly predicted, as a percentage of the total number of times each stage was predicted. The sum of the number of times each classifier correctly predicted the stage was calculated and then divided by the total number of times each stage was predicted at all. Note that in these calculations, the ZeroR algorithm was ignored as this algorithm simply predicts the most common stage in the training dataset. For example, in the original training set, Stage 4 was the most common stage so the ZeroR algorithm predicted Stage 4 for each instance in the testing set, therefore including this algorithm would skew the data too much. The frequency that each stage was predicted for this case was also calculated. This frequency was calculated by counting the number of times each stage was predicted by all of the algorithms and then dividing by the total number of predictions made by all of the algorithms, except ZeroR. The frequencies resulted in the following breakdown: Stage 1 – 5.2%, Stage 2 – 16%, Stage 3 – 25.4%, Stage 4 – 53.2%, Stage 5 – 0.1%,which is shown in Figure 3B.

When looking at the 'unswitched' case, the stage precision breakdown we saw was the following: Stage 1 – 63.5%, Stage 2 – 73%, Stage 3 – 29%, Stage 4 – 79.5%, Stage 5 – 0%. The frequency that each stage was predicted for this case had the following breakdown: Stage 1 – 7.2%, Stage 2 – 32.2%, Stage 3 – 17.1%, Stage 4 – 43.5%, Stage 5 – 0%.

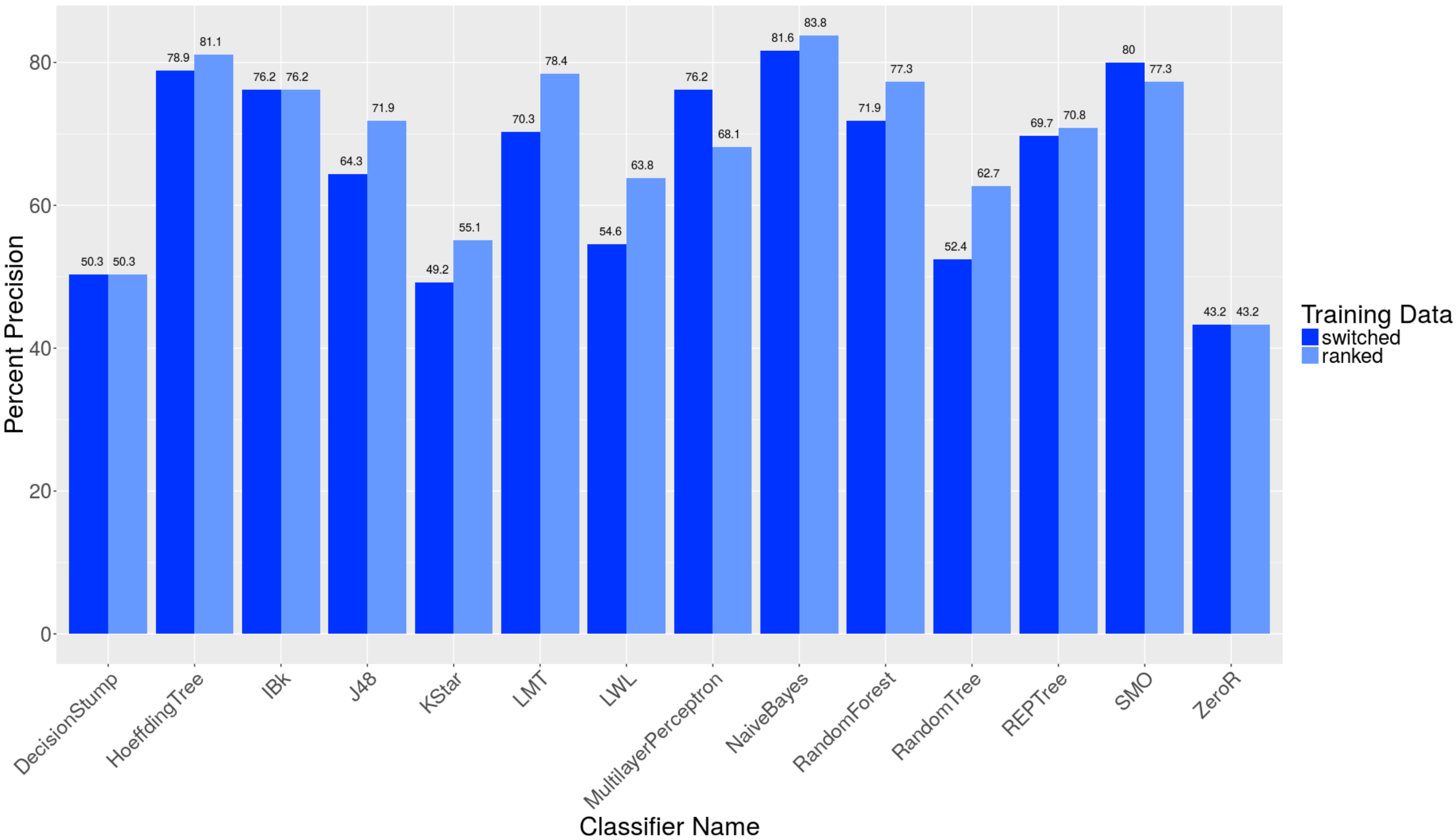


Figure 4 - All defaut algorithms tested with their precisions graphed side by side with the ranked (dark blue) and switched (light blue) case

Figure 4 shows the different precision values for the 'switched' case and the 'ranked' case. The 'ranked' case took the 'switched' data and first ranked the attributes before training and building the classification models. The same default set of models were built and tested and the precision values were compared to those achieved when the ranking algorithm was not run. The algorithms that performed better when the attributes were ranked first were: NaiveBayes, KStar, LWL, HoeffdingTree, J48, LMT, RandomForest, RandomTree, and REPTree. The algorithms that performed better when the attributes were not ranked first were: MultilayerPerceptron and SMO. The algorithms that showed no difference in precision when the attributes were ranked or unranked were: ZeroR, IBk, DecisionStump. Therefore, 9 algorithms performed better when the attributes were ranked, 2 performed worse, and 3 performed the same. The best classification algorithm after ranking the attributes was the NaiveBayes algorithm with a precision of 83.8%, a 2.2% increase in precision from NaiveBayes in just the 'switched' case with no attribute ranking.

When looking at the 'ranked' case, the stage precision breakdown we saw was the following: Stage 1 – 79.6%, Stage 2 – 81.5%, Stage 3 – 42.1%, Stage 4 – 99.5%, Stage 5 – 0%. The frequency that each stage was predicted for this case had the following breakdown: Stage 1 – 6.6%, Stage 2 – 19.8%, Stage 3 – 18.3%, Stage 4 – 55.3%, Stage 5 – 0%.

5. Discussion

From Figure 3A (the ‘switched’ case), we see that Stage 4 is predicted drastically more precisely than any other stage. From Figure 3B, we also see that Stage 4 was predicted the most often. Looking at a breakdown of the number of instances belonging to each stage in the dataset we have: (Stage 1: 32, Stage 2: 163, Stage 3: 280, Stage 4: 400, Stage 5: 33). The combination of Stage 4 being the most prevalent stage in the set used for training and the fact that it was predicted the most often, gives a possible reason that this stage was predicted most accurately. Additionally, Stage 4 plants are larger since they are more days after seeding. These plants may be more developed and more defined than smaller plants which have less information to be gained from them. This gives another possible reason that Stage 4 plants were predicted most precisely. Stage 5 was never predicted correctly because in this case the testing set had zero plants in Stage 5.

In the ‘unswitched’ case, we see again that Stage 4 was correctly predicted the most whereas Stage 3 was correctly predicted the least often (excluding Stage 5). We also see that Stage 4 was predicted the highest number of times but we also see that Stage 3 was not predicted the fewest number of times, Stage 1 was (again excluding Stage 5). Looking at a breakdown of the number of instances belonging to each stage in the dataset we have: (Stage 1: 9, Stage 2: 36, Stage 3: 60, Stage 4: 80, Stage 5: 0). We can see from this that there were no instances in the dataset that belonged to Stage 5 and this is why Stage 5 was never predicted nor correctly predicted, the classification algorithms had nothing to learn from. Similar to the ‘switched’ case, Stage 4 was most likely predicted most accurately due to the fact that there is more information to be gained from plants in this stage than plants in earlier stages. However, there is not yet a good explanation why Stage 3 was correctly predicted the least often instead of Stage 1.

From Figure 4 (the ‘ranked’ case) we see that ranking the attributes before building and training the models improved (or at least did not affect) the precision of most algorithms. For some algorithms, it is obvious why there were no improvements in precision when the attributes were ranked. The ZeroR algorithm, for example, always predicts the value that is most common in the training dataset. Therefore, it does not matter how well the data is prepared before building the model, the ZeroR algorithm will always predict the same value. Most of the algorithms experienced an improvement in precision as a result of ranking the attributes. This makes sense as the ranking tells the algorithms which attributes are most important to analyze to determine the developmental stage of the plant and therefore these attributes receive more weight. It seems reasonable that in the worst-case, this ranking would just be extra computational work that added no benefit to precision and the precision would simply be the same as if we did not rank first, which is what we saw for three of the algorithms tested. We once again saw that Stage 4 was the best predicted stage when the attributes were ranked and we can again guess that this is due to the plants in Stage 4 being larger and better defined than the plants in earlier stages. What is interesting, however, is that we see a 24% increase in precision in predicting Stage 1, a 22% increase in precision in predicting Stage 2, and an 8.2% decrease in predicting Stage 3. As just stated, the increase in precision for certain stages is most likely due to the fact that the more important attributes have more weight being assigned to them. However, there is not a good explanation yet for why Stage 3 would have experienced a decrease in precision compared to when the attributes were unranked.

6. Conclusion

In this research, a program was developed as a way of automating the querying of any database from the McGill phenomics lab, reading this data, using the training data to build machine learning classification models that would predict the developmental stage of a plant based on its image data, and testing these models on a new test set of plant image data. Based on the program, the NaiveBayes algorithm is most precise in correctly predicting the developmental stage based on a new set of data. This research also showed that a larger training set leads to better models, which seems reasonable and makes sense. As we saw here, the initial training set had 185 instances while the testing set had 908 instances. When the smaller dataset is used for training, the highest precision we achieved from any algorithm was only 64.2% (from the IBk algorithm). However, when the larger dataset was used for training and the smaller dataset was used for testing, ie the ‘switched’ case, we were able to achieve 81.6% precision (from the NaiveBayes algorithm). We were then able to reach an even slightly higher precision by looking at the ‘switched’ case and ranking the attributes before building the models. With this method, we were able to achieve 83.8% precision (again from the NaiveBayes algorithm).

We saw from the R analysis that Stage 5 was predicted the least precisely both in the ‘switched’ and ‘unswitched’ cases and we safely concluded that this is due to the small number of plants that were in Stage 5 in both datasets, relative to the other stages. We saw that Stage 4 was predicted the most precisely in both the ‘switched’ and ‘unswitched’ cases. We concluded that this was due to the fact that plants in this stage were larger and more developed, thereby being more defined than smaller plants which would have less information available to learn about them.

This program provides an easy way to test many different algorithms quickly and efficiently connected to the MP3 database. A user can easily build and train models with any algorithm that exists within the Weka library. Additionally, the program also provides direct querying from the MP3 database, though it can be easily modified to query another database. A possible future opportunity is to see if different datasets lead to even higher precision. For example, in this research we only looked at the vis-side-1-0 camera but it is very possible to choose a different camera and angle to see if there would be an improvement in precision results. Though I implemented the option for the user to rank the attributes in the dataset by which attributes are most important in determining the developmental stage, I did not have the chance to implement the other attribute selection options that Weka provides. For example, Weka has different attribute selection options that will actually eliminate certain attributes and only keep the attributes that are most important in determining the developmental stage of the plant. A further step for this research would be to implement these different attribute selection options that the user can choose from to see if there would be any improvement in precision, similar to the improvement that was observed when the attribute ranking was performed.

References

Arnarson, Atli. (2015, January 17). *Soybeans 101: Nutritions Facts and Health Effects*. Retrieved from https://www.healthline.com/nutrition/foods/soybeans.

Frank, Eibe and Witten, Ian H. (1999). *Data Mining: Practical Machine Learning Tools and Techniques*. London: Morgan Kaufmann Publishers.

Stavness, Ian and Ubbens, Jordan R. (2017). “Deep Plant Phenomics: A Deep Learning Platform for Complex Plant Phenotyping Tasks”. *Frontiers in Plant Science, 8,* 11.

Vello E., Diallo Oury, A., Tomita, A., and Bureau T.E., (2015). “A Comprehensive Approach to Assess Arabidopsis Survival Phenotype in Water-Limited Condition Using a Non-invasive High-Throughput Phenomics Platform”. *Frontiers in Plant Science*, *6,* 10.

Vello, Emilio. (2017). *Image Analysis Pipeline.* Retrieved from http://mp3.biol.mcgill.ca/mcgill_mp3_analysis.html

Vello, Emilio. (2017). *The Platform in a Short Presentation.* Retrieved from http://mp3.biol.mcgill.ca/mcgill_mp3_summary.html

SUPPLEMENTARY MATERIALS

```java
/*
 * @author Lea Collin
 */
import weka.core.Instance;
import weka.core.Instances;
import weka.classifiers.Classifier;
import weka.classifiers.AbstractClassifier;
import weka.classifiers.meta.FilteredClassifier;
import weka.core.converters.ArffLoader;
import weka.filters.unsupervised.attribute.Remove;
import weka.attributeSelection.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import org.postgresql.util.PSQLException;
import java.sql.*;

public class Driver {

    static Scanner sc = new Scanner(System.in);

    public static void main(String [] args) throws Exception {

        //private database information
        String dbConfig = args[0];

        //ArrayList will contain all the possible algorithms that the user can input
        ArrayList<String> possibleClassifiers = new ArrayList<String>();
        possibleClassifiers.add("ZeroR");
        possibleClassifiers.add("NaiveBayes");
        possibleClassifiers.add("MultilayerPerceptron");
        possibleClassifiers.add("SMO");
        possibleClassifiers.add("IBk");
        possibleClassifiers.add("KStar");
        possibleClassifiers.add("LWL");
        possibleClassifiers.add("DecisionStump");
        possibleClassifiers.add("HoeffdingTree");
        possibleClassifiers.add("J48");
        possibleClassifiers.add("LMT");
        possibleClassifiers.add("RandomForest");
        possibleClassifiers.add("RandomTree");
        possibleClassifiers.add("REPTree");

        System.out.println("Please enter the directory name of where you
```

```
would like to store all program outputs:");

        //add control to check for valid directory
        String outputDir = setOutputDirectory();

        System.out.println();
        System.out.println("Please enter the name of the file you'd like to store the TRAINING data. Please end the file name in '.arff'");
        String trainingFile = setFileName(".arff", outputDir);
        //String trainingFile = outputDir + "trainingData.arff";
        System.out.println();

        System.out.println("Please enter the name of the file you'd like to store the TESTING data. Please end the file name in '.arff'");
        String testingFile = setFileName(".arff", outputDir);
        System.out.println();

        //try to connect to database given username and password, user prompted to enter username and password again if connection is unsuccessful
        boolean successfulConnection = false;
        while(!successfulConnection) {
                System.out.println("Please enter your database username:");
                String dbUsr = sc.next();
                System.out.println("Password:");
                String dbPwd = sc.next();
                sc.nextLine();
            try {
                connectToDatabase(dbUsr, dbPwd, dbConfig, testingFile, trainingFile);

                successfulConnection = true;
            }catch (PSQLException s){
                System.out.println("Username or password was incorrect. Please try again.");
            }
        }

        boolean rankAttributes = false;
        boolean isValid = false;
        while(!isValid) {
            System.out.println("Would you like to rank the attributes? (Y/n), (Could lead to better results)");
            String answer = sc.next();
            if(answer.equals("Y") || answer.equals("y")) {
                isValid = true;
                rankAttributes = true;
            }
            else if(answer.equals("N") || answer.equals("n")){
```

```java
                isValid = true;
                rankAttributes = false;
            }
            else {
                System.out.println("Could not understand input. Please enter a name again. \n");
            }
        }

        //getting user input for classifier names, checking if input is valid
        String [] classifiers = null;

        boolean validClassifier = false;
        while(!validClassifier){
            System.out.println("Do you want to run the default set of classification algorithms? (Y/n)");
            String answer = sc.next();
            if(answer.equals("Y") || answer.equals("y")) {
                classifiers = new String[possibleClassifiers.size()];
                for(int i = 0; i < classifiers.length; i++) {
                    classifiers[i] = possibleClassifiers.get(i);
                }
                validClassifier = true;
            }
            else if(answer.equals("N") || answer.equals("n")){
                System.out.println("Please enter the names of the classifiers you'd like to test, separated by a single space.");
                sc.nextLine();
                String classifierInput = sc.nextLine();
                classifiers = classifierInput.split("\\s+");

                for(int i = 0; i < classifiers.length; i++) {
                    if(!possibleClassifiers.contains(classifiers[i])) {
                        System.out.println(classifiers[i] + " is not a valid classifier name.");
                        System.out.println("Please try again.");
                        System.out.println();
                    }

                    if(i == classifiers.length - 1 && possibleClassifiers.contains(classifiers[i])) {
                        validClassifier = true;
                    }
                }
            }
            else {
                System.out.println("Could not understand input.");
            }
```

```java

        }

        System.out.println();
        System.out.println("Please enter the name of the file you would like to store all of the predictions. Please end the file in '.csv'");
        String predictionFile = setFileName(".csv", outputDir);

        System.out.println();
        System.out.println("Finally, please enter the name of the file you would like to store the precision of each algorithm you are testing. "
                + "Please end the file in '.csv'");
        String precisionFile = setFileName(".csv", outputDir);
        System.out.println();

        sc.close();

        //reading the files and getting all the instances of each one
        Instances instancesTrain = fileReader(trainingFile);
        Instances instancesTest = fileReader(testingFile);

        //what attribute do we want to predict
        String classAttribute = "Stage";

        //these are the attributes to not be included in the classifier, das is a giveaway of the stage, barcode is irrelevant
        String [] attributesToRemove = {"barcode", "das"};
        //returns the indices of the attributes to be ignored
        String indicesToRemove = removeAttribute(instancesTrain, attributesToRemove);

        double maxPrecision = 0.0;

        String bestMethod = "";

        //run all the different classifiers
        Double [] precisions = predict(instancesTrain, instancesTest, classifiers, indicesToRemove, classAttribute, predictionFile, rankAttributes);

        //writing precision values to a csv
                File output = new File(precisionFile);
                PrintWriter pw = new PrintWriter(output);
                StringBuilder sb = new StringBuilder();
                sb.append("Algorithm");
                sb.append(',');
                sb.append("Precision");
                sb.append("\n");
```

```

        //find best algorithm and write to file
        for(int i = 0; i < classifiers.length; i++) {
            if(precisions[i] > maxPrecision) {
                maxPrecision = precisions[i];
                bestMethod = classifiers[i];
            }
        }
        //simply printing out for the user which algorithm was best and its precision
        System.out.println("Best Algorithm: " + bestMethod + " with precision: " + maxPrecision);

        pw.write(sb.toString());
        pw.close();
    }

    public static boolean validArff(String file) {
        //string must contain and end in .arff to be a valid arff file
        return (file.contains(".arff") && file.indexOf(".arff") == file.length() - 5);
    }

    public static boolean validCsv(String file) {
        return (file.contains(".csv") && file.indexOf(".csv") == file.length() - 4);
    }

    public static boolean fileExists(String file) {
        File newFile = new File(file);
        return newFile.exists();
    }

    public static String setOutputDirectory() {
        String dirName = "";
        boolean isValid = false;
        while(!isValid) {
            dirName = sc.next();
            File dir = new File(dirName);

            if(dir.exists() && (dirName.charAt(dirName.length()-1) == '/')) {
                isValid = true;
            }
            else {
                System.out.println("Sorry that was an invalid directory. Please try again.");
            }
        }
```

```

        return dirName;
    }

    public static String setFileName(String fileType, String outputDir) {
        String outputFile = "";
        boolean isValid = false;
        while(!isValid) {
            outputFile = outputDir + sc.next();
            if(fileType.equals(".arff")) {
                if(validArff(outputFile) == isValid) {
                    System.out.println("Sorry, the file you entered does not end in '.arff'. Please try again.");
                    continue;
                }
            }
            if(fileType.equals(".csv")) {
                if(validCsv(outputFile) == isValid) {
                    System.out.println("Sorry, the file you entered does not end in '.csv'. Please try again.");
                    continue;
                }
            }
            if(fileExists(outputFile)) {
                System.out.println("This file already exists in this directory. Do you want to overwrite it? (Y/n)?");
                String answer = sc.next();
                if(answer.equals("Y") || answer.equals("y")) {
                    isValid = true;
                }
                else if(answer.equals("N") || answer.equals("n")){
                    System.out.println("Please enter another name.");
                }
                else {
                    System.out.println("Could not understand input. Please enter a name again.");
                }
            }
            else {
                isValid = true;
            }
        }
        return outputFile;
    }

    public static Instances fileReader(String input) throws IOException {

        File inputFile = new File(input);
```

```
        ArffLoader atf = new ArffLoader();
        atf.setFile(inputFile);
        Instances data = atf.getDataSet();

        return data;

    }

    public static String removeAttribute(Instances data, String [] attributes) throws Exception{

        String [] options = new String[2];
        options[0] = "-R";

        String indices = "";

        //getting all the indices of attributes we want to later remove (or ignore)
        for(int i = 0; i < attributes.length; i++) {
            int index = (data.attribute(attributes[i])).index() + 1;
            indices += index;
            if(i != attributes.length-1) {
                indices += ",";
            }
        }

        return indices;
    }

    public static Double [] predict(Instances train, Instances test, String [] classifierNames,
        String indicesToRemove, String classAttribute, String outputFile, boolean rankAttributes) throws Exception {

        //removing attributes we don't want to include such as das and barcode
                Remove rm = new Remove();
                rm.setAttributeIndices(indicesToRemove);

        //running an attribute selector if user chose to rank attributes
                if(rankAttributes) {
                    Instances [] reduced = attributeSelector(train, test);

                    train = reduced[0];

                    test = reduced[1];
                }

```

```
        //set the Class (what we want to predict)
        test.setClass(test.attribute(classAttribute));

        //setting the train class index to be the same as the testing class index
        train.setClassIndex(test.classIndex());

        //going to make an array of classifiers
        Classifier [] classifiers = new Classifier [classifierNames.length];

        //need somewhere to store the precision of each classifier + initializing the array
        Double precision [] = new Double [classifiers.length];
        for(int i = 0; i < precision.length; i++) {
            precision[i] = 0.0;
        }

        //creating filtered versions of each classifier (removing das and barcode)
        for(int i = 0; i < classifiers.length; i++) {
            Classifier temp = AbstractClassifier.forName(classifierNames[i], null);

            FilteredClassifier fc = new FilteredClassifier();
            fc.setFilter(rm);
            fc.setClassifier(temp);

            classifiers[i] = fc;
        }

        //building the models for each classifier
        for(int i = 0; i < classifiers.length; i++) {
            classifiers[i].buildClassifier(train);
        }

        //writing the header to the output csv
        File output = new File(outputFile);
        PrintWriter pw = new PrintWriter(output);
        StringBuilder sb = new StringBuilder();
        sb.append("Barcode");
        sb.append(',');
        sb.append("Das");
        sb.append(",");
        sb.append("Actual");
        sb.append(',');

        //making the names of the classifiers part of the header, will indicate the stage that classifier has predicted for a particular plant
```

```
        for(int i = 0; i < classifierNames.length; i++) {
            sb.append(classifierNames[i]);
            if(i != classifierNames.length - 1) {
                sb.append(",");
            }
        }
        sb.append("\n");

        double numInst = test.numInstances();

        //actually running the classifier
        for(int i = 0; i < numInst; i++){

            Instance current = test.instance(i);

            //will set the Stage of this 'temp' instance to be the predicted value to then compare it to the actual value of 'current'
            Instance temp = (Instance)current.copy();

            //attributes are given as array positions, getting the string value
            String actualVal = current.stringValue(test.classIndex());

            sb.append((int) current.value(test.attribute("barcode")));
            sb.append(',');
            sb.append((int) current.value(test.attribute("das")));
            sb.append(",");
            sb.append(actualVal);
            sb.append(',');

           for(int j = 0; j < classifiers.length; j++) {
            //getting the predicted value of the class attribute of this instance
            double predicted = classifiers[j].classifyInstance(test.instance(i));

            //setting this value to the temp class attribute
            temp.setValue(test.classIndex(), predicted);

            //getting the string value
            String predictedVal = temp.stringValue(temp.classIndex());

            //comparing predicted with actual value
            if(predictedVal.equals(actualVal)) {
                precision[j]++;
            }

            sb.append(predictedVal);
```

```java

            if(j != classifiers.length - 1) {
                sb.append(",");
            }

            else if( j == classifiers.length - 1) {
                sb.append('\n');
            }

           }
        }

        for(int i = 0; i < precision.length; i++) {
            precision[i] = 100*precision[i]/numInst;
        }

        sb.append('\n');
        pw.write(sb.toString());
        pw.close();

        return precision;
    }

    private static void connectToDatabase(String usrDB, String passwordDB, String conDB, String trainName, String testName) throws SQLException, FileNotFoundException {

        File trainingOutput = new File(trainName);
        PrintWriter trainingPw = new PrintWriter(trainingOutput);
        StringBuilder sb = new StringBuilder();

        File testingOutput = new File(testName);
        PrintWriter testingPw = new PrintWriter(testingOutput);

        //creating the header for the arff file
        sb.append("@relation databasetraining" + "\n" +  "\n" + "@attribute barcode numeric"  + "\n" + "@attribute area numeric" + "\n" + "@attribute perimeter numeric" + "\n" +
        "@attribute circularity numeric" + "\n" + "@attribute compactness numeric" + "\n" + "@attribute major numeric" + "\n" +
        "@attribute minor numeric" + "\n" + "@attribute eccentricity numeric" + "\n" + "@attribute hisgreypeak numeric" + "\n" +
        "@attribute q1grey numeric" + "\n" + "@attribute q2grey numeric" + "\n" + "@attribute q3grey numeric" + "\n" +
        "@attribute q1r numeric" + "\n" + "@attribute q2r numeric" + "\n" + "@attribute q3r numeric" + "\n" + "@attribute q1g numeric" + "\n" +
        "@attribute q2g numeric" + "\n" + "@attribute q3g numeric" + "\n" + "@attribute q1b numeric" + "\n" + "@attribute q2b numeric" + "\n" +
```

```
        "@attribute q3b numeric" + "\n" + "@attribute das numeric" + "\n"
+"@attribute Stage {'Stage 1','Stage 2','Stage 3','Stage 4', 'Stage 5'}" +
"\n" + "\n" + "@data" + "\n");

        trainingPw.write(sb.toString());
        testingPw.write(sb.toString());

        try {
            Class.forName("org.postgresql.Driver");
            Connection conn = DriverManager.getConnection(conDB, usrDB,
passwordDB);

            String trainingSql = "SELECT s.barcode, o.area, "
                    + "o.perimeter, o.circularity, o.compactness, "
                    + "o.major, o.minor, o.eccentricity, o.hisgreypeak, "
                    + "o.q1grey, o.q2grey, o.q3grey, "
                    + "o.q1r, o.q2r, o.q3r, "
                    + "o.q1g, o.q2g, o.q3g, "
                    + "o.q1b, o.q2b, o.q3b, d.das, "
                    + "CASE WHEN ( d.das <= 10 ) THEN 'Stage 1' "
                    + "WHEN ( d.das > 10 AND d.das <= 20 ) THEN 'Stage 2' "
                    + "WHEN ( d.das > 20 AND d.das <= 30 ) THEN 'Stage 3' "
                    + "WHEN ( d.das > 30 AND d.das <= 40 ) THEN 'Stage 4' "
                    + "WHEN ( d.das > 40 AND d.das <= 50 ) THEN 'Stage 5' "
                    + "ELSE 'Stage 6' END Stage "
                    + "FROM imageev AS i, imgobjectev AS o, soyidentification
AS s, dasplusev AS d "
                    + "WHERE i.assayid = o.assayid "
                    + "AND i.imgid = o.imgid "
                    + "AND s.barcode = ( CAST( i.barcode AS INTEGER ) ) "
                    + "AND i.assayid = d.assayid AND i.fdate = d.fdate "
                    + "AND i.set = d.set "
                    + "AND s.line = 1 "
                    + "AND i.camera = 'vis-side-1-0' "
                    + "AND i.set = '3'";

            String testingSql =  "SELECT s.barcode, o.area, "
                    + "o.perimeter, o.circularity, o.compactness, "
                    + "o.major, o.minor, o.eccentricity, o.hisgreypeak, "
                    + "o.q1grey, o.q2grey, o.q3grey, "
                    + "o.q1r, o.q2r, o.q3r, "
                    + "o.q1g, o.q2g, o.q3g, "
                    + "o.q1b, o.q2b, o.q3b, d.das, "
                    + "CASE WHEN ( d.das <= 10 ) THEN 'Stage 1' "
                    + "WHEN ( d.das > 10 AND d.das <= 20 ) THEN 'Stage 2' "
                    + "WHEN ( d.das > 20 AND d.das <= 30 ) THEN 'Stage 3' "
                    + "WHEN ( d.das > 30 AND d.das <= 40 ) THEN 'Stage 4' "
                    + "WHEN ( d.das > 40 AND d.das <= 50 ) THEN 'Stage 5' "
```

```
                    + "ELSE 'Stage 6' END Stage "
                    + "FROM imageev AS i, imgobjectev AS o, soyidentification AS s, dasplusev AS d "
                    + "WHERE i.assayid = o.assayid "
                    + "AND i.imgid = o.imgid "
                    + "AND s.barcode = ( CAST( i.barcode AS INTEGER ) ) "
                    + "AND i.assayid = d.assayid "
                    + "AND i.fdate = d.fdate "
                    + "AND i.set = d.set "
                    + "AND ( s.line = 1 OR s.line = 2 OR s.line = 3 ) "
                    + "AND i.camera = 'vis-side-1-0' "
                    + "AND i.set = '2' "
                    + "UNION "
                    + "SELECT s.barcode, o.area, o.perimeter, o.circularity, "
                    + "o.compactness, o.major, o.minor, o.eccentricity, o.hisgreypeak, "
                    + "o.q1grey, o.q2grey, o.q3grey, "
                    + "o.q1r, o.q2r, o.q3r,"
                    + " o.q1g, o.q2g, o.q3g, "
                    + "o.q1b, o.q2b, o.q3b, d.das, "
                    + "CASE WHEN ( d.das <= 10 ) THEN 'Stage 1' "
                    + "WHEN ( d.das > 10 AND d.das <= 20 ) THEN 'Stage 2' "
                    + "WHEN ( d.das > 20 AND d.das <= 30 ) THEN 'Stage 3' "
                    + "WHEN ( d.das > 30 AND d.das <= 40 ) THEN 'Stage 4' "
                    + "WHEN ( d.das > 40 AND d.das <= 50 ) THEN 'Stage 5' "
                    + "ELSE 'Stage 6' END Stage "
                    + "FROM imageev AS i, imgobjectev AS o, soyidentification AS s, dasplusev AS d "
                    + "WHERE i.assayid = o.assayid "
                    + "AND i.imgid = o.imgid "
                    + "AND s.barcode = ( CAST( i.barcode AS INTEGER ) ) "
                    + "AND i.assayid = d.assayid "
                    + "AND i.fdate = d.fdate "
                    + "AND i.set = d.set "
                    + "AND (s.line = 2 OR s.line = 3 ) "
                    + "AND i.camera = 'vis-side-1-0' "
                    + "AND i.set = '3' ";

            PreparedStatement trainingPs = conn.prepareStatement(trainingSql);
            ResultSet trainingSet = trainingPs.executeQuery();
            while(trainingSet.next()) {
                Double barcode = trainingSet.getDouble("barcode");
                Double area = trainingSet.getDouble("area");
                Double perimeter = trainingSet.getDouble("perimeter");
                Double circularity = trainingSet.getDouble("circularity");
                Double compactness = trainingSet.getDouble("compactness");
```

```
                Double major = trainingSet.getDouble("major");
                Double minor = trainingSet.getDouble("minor");
                Double eccentricity = trainingSet.getDouble("eccentricity");
                Double hisgreypeak = trainingSet.getDouble("hisgreypeak");
                Double q1grey = trainingSet.getDouble("q1grey");
                Double q2grey = trainingSet.getDouble("q2grey");
                Double q3grey = trainingSet.getDouble("q3grey");
                Double q1r = trainingSet.getDouble("q1r");
                Double q2r = trainingSet.getDouble("q2r");
                Double q3r = trainingSet.getDouble("q3r");
                Double q1g = trainingSet.getDouble("q1g");
                Double q2g = trainingSet.getDouble("q2g");
                Double q3g = trainingSet.getDouble("q3g");
                Double q1b = trainingSet.getDouble("q1b");
                Double q2b = trainingSet.getDouble("q2b");
                Double q3b = trainingSet.getDouble("q3b");
                Double das = trainingSet.getDouble("das");
                String stage = trainingSet.getString("Stage");

                //writing all the different attributes from the query to the output arff file
                trainingPw.write(barcode + "," + area + ", " + perimeter + ", " + circularity + ", " + compactness + ", " + major + ", " + minor + ", " + eccentricity
                        + ", " + hisgreypeak + ", " + q1grey + ", " + q2grey + ", " + q3grey + ", " + q1r + ", " + q2r + ", " + q3r
                        + ", " + q1g + ", " + q2g + ", " + q3g + ", " + q1b + ", " + q2b + ", " + q3b + "," + das + "," + "'" + stage + "'" + "\n");
            }
            trainingSet.close();
            trainingPw.close();

            PreparedStatement  testingPs = conn.prepareStatement(testingSql);
            ResultSet testingSet = testingPs.executeQuery();
            while(testingSet.next()) {
                Double barcode = testingSet.getDouble("barcode");
                Double area = testingSet.getDouble("area");
                Double perimeter = testingSet.getDouble("perimeter");
                Double circularity = testingSet.getDouble("circularity");
                Double compactness = testingSet.getDouble("compactness");
                Double major = testingSet.getDouble("major");
                Double minor = testingSet.getDouble("minor");
                Double eccentricity = testingSet.getDouble("eccentricity");
                Double hisgreypeak = testingSet.getDouble("hisgreypeak");
                Double q1grey = testingSet.getDouble("q1grey");
                Double q2grey = testingSet.getDouble("q2grey");
                Double q3grey = testingSet.getDouble("q3grey");
                Double q1r = testingSet.getDouble("q1r");
```

```
                Double q2r = testingSet.getDouble("q2r");
                Double q3r = testingSet.getDouble("q3r");
                Double q1g = testingSet.getDouble("q1g");
                Double q2g = testingSet.getDouble("q2g");
                Double q3g = testingSet.getDouble("q3g");
                Double q1b = testingSet.getDouble("q1b");
                Double q2b = testingSet.getDouble("q2b");
                Double q3b = testingSet.getDouble("q3b");
                Double das = testingSet.getDouble("das");
                String stage = testingSet.getString("Stage");

                testingPw.write(barcode + "," + area + ", " + perimeter + ", " + circularity + ", " + compactness + ", " + major + ", " + minor + ", " + eccentricity
                        + ", " + hisgreypeak + ", " + q1grey + ", " + q2grey + ", " + q3grey + ", " + q1r + ", " + q2r + ", " + q3r
                        + ", " + q1g + ", " + q2g + ", " + q3g + ", " + q1b + ", " + q2b + ", " + q3b + "," + das + "," + "'" + stage + "'" + "\n");
            }
            testingSet.close();
            testingPw.close();

            conn.close();
        }
        catch (ClassNotFoundException e) {

            System.out.println("Improper database connection set-up.");
            e.printStackTrace();

        }
    }

public static Instances [] attributeSelector(Instances train, Instances test) throws Exception {

        AttributeSelection selector = new AttributeSelection();
        Ranker search = new Ranker();
        OneRAttributeEval eval = new OneRAttributeEval();
        selector.setEvaluator(eval);
        selector.setSearch(search);
        selector.SelectAttributes(train);

        //rankedAttributes gives the ranking of the attributes along with their weights
        //selected Attributes just gives the order of the ranking
        Instances trainTemp = selector.reduceDimensionality(train);
        Instances trainTest = selector.reduceDimensionality(test);

```

```

        Instances [] reduced = {trainTemp, trainTest};

        return reduced;
    }

}
```

```
library(ggplot2)
library(reshape2)
setwd("~/Documents/U4/Comp401/output")

predictions <- read.csv("originalPredictions.csv", header = TRUE)
unswitched <- rep(0, length(predictions) - 3)
classifier.name <- colnames(predictions[4:length(predictions)])
actual.stage <- as.vector(predictions[, "Actual"])
n <- length(actual.stage)

for(i in 4:length(predictions)){
    v <- as.vector(predictions[,i])
    unswitched[i-3] <- round(100*length(which(actual.stage == v))/n, 3)
}


### RESULTS WHEN TRAINING DATA AND TESTING DATA ARE SWITCHED
switchedPredictions <- read.csv("switchedPredictions.csv", header = TRUE)
switched <- rep(0, length(switchedPredictions) - 3)
classifier.name.s <- colnames(switchedPredictions[4:length(switchedPredictions)])
actual.stage.s <- as.vector(switchedPredictions[, "Actual"])
sn <- length(actual.stage.s)

for(i in 4:length(switchedPredictions)){
  sv <- as.vector(switchedPredictions[,i])
  switched[i-3] <- round(100*length(which(actual.stage.s == sv))/sn, 3)
}

allPrecisions <- data.frame(classifier.name, unswitched, switched)
allPrecisions<- melt(allPrecisions, id.vars = "classifier.name")
precision.plot <- ggplot(allPrecisions, aes(y = value, x = classifier.name, fill = variable))
precision.plot <- precision.plot + geom_bar(stat = "identity", position = "dodge")
precision.plot <- precision.plot + xlab("Classifier Name") + ylab("Percent Precision")
precision.plot <- precision.plot + guides(fill=guide_legend(title = "Training Data"))
precision.plot <- precision.plot + scale_fill_manual(values = c("#0033FF", "#6699FF"))
precision.plot <- precision.plot + geom_text(aes(label = round(value, 1)), position = position_dodge(0.75), vjust = -1)
precision.plot <- precision.plot + theme(axis.text.x = element_text(angle = 45, hjust = 1), text = element_text(size =
24))
precision.plot

### Finding out which algorithms performed better between switched data
###BETTER UNSWITCHED
classifier.name[which(unswitched > switched)]

###BETTER SWITCHED
classifier.name[which(unswitched < switched)]

### RESULTS WHEN ATTRIBUTES ARE RANKED FIRST
rankedPredictions <- read.csv("RankedPredictions.csv", header = TRUE)
ranked <- rep(0, length(rankedPredictions) - 3)
classifier.name.r <- colnames(rankedPredictions[4:length(rankedPredictions)])
actual.stage.r <- as.vector(rankedPredictions[, "Actual"])
rn <- length(actual.stage.r)

for(i in 4:length(rankedPredictions)){
  rv <- as.vector(rankedPredictions[,i])
  ranked[i-3] <- round(100*length(which(actual.stage.r == rv))/rn, 3)
}

allPrecisions <- data.frame(classifier.name.r, switched, ranked)
allPrecisions<- melt(allPrecisions, id.vars = "classifier.name.r")
precision.plot <- ggplot(allPrecisions, aes(y = value, x = classifier.name.r, fill = variable))
precision.plot <- precision.plot + geom_bar(stat = "identity", position = "dodge")
precision.plot <- precision.plot + xlab("Classifier Name") + ylab("Percent Precision")
precision.plot <- precision.plot + guides(fill=guide_legend(title = "Training Data"))
precision.plot <- precision.plot + scale_fill_manual(values = c("#0033FF", "#6699FF"))
precision.plot <- precision.plot + geom_text(aes(label = round(value, 1)), position = position_dodge(0.75), vjust = -1)
precision.plot <- precision.plot + theme(axis.text.x = element_text(angle = 45, hjust = 1), text = element_text(size =
24))
precision.plot

###BETTER RANKED
classifier.name.r[which(ranked > switched)]

###NO DIFFERENCE
classifier.name.r[which(ranked == switched)]

###BETTER UNRANKED
classifier.name.r[which(ranked < switched)]

###LOOKING AT THE STAGE PREDICTION BREAKDOWN WHEN THE ATTRIBUTES ARE RANKED
predictions <- read.csv("RankedPredictions.csv", header = TRUE)
classifier.name <- colnames(predictions[5:length(predictions)])
actual.stage <- as.vector(predictions[, "Actual"])
n <- length(actual.stage)

stages <- c("Stage 1", "Stage 2", "Stage 3", "Stage 4", "Stage 5")
numStages <- rep(0, length(stages))
for (i in 1:length(stages)){
  numStages[i] <- length(which(predictions[,"Actual"] == stages[i]))
```

```
}

percent.correct <- rep(0, length(stages))

for(i in 1:length(stages)){
  indices <- which(predictions[,"Actual"] == stages[i])
  for(j in 5:length(predictions)){
    percent.correct[i] <- percent.correct[i] + length(which(predictions[indices, j] == stages[i]))
  }
}

### percent.correct is an AVERAGE of the percent time a stage is predicted correctly
percent.correct <- 100*(round(percent.correct/(numStages*(length(classifier.name)-1)),3))
percent.correct <- data.frame(stages, percent.correct)

stage.plot <- ggplot(percent.correct, aes(y = percent.correct, x = stages))
stage.plot <- stage.plot + geom_bar(stat = "identity", fill = "#0033FF")
stage.plot <- stage.plot + xlab("Stage") + ylab("Percent Correctly Predicted") + ggtitle("Average Precision per Stage")
stage.plot <- stage.plot + geom_text(aes(label = round(percent.correct, 1)), position = position_dodge(0.75), vjust =
-1)
stage.plot

###What percentage of the time was stage 1 predicted?
times.predicted <- rep(0, length(stages))

for(i in 1:length(stages)){
  for(j in 5:length(predictions)){
    times.predicted[i] <- times.predicted[i] + length(which(predictions[,j] == stages[i]))
  }
}

times.predicted <- 100*(round(times.predicted/(sum(times.predicted)),5))
times.predicted <- data.frame(stages, times.predicted)
times.predicted.plot <- ggplot(times.predicted, aes(y = times.predicted, x = stages))
times.predicted.plot <- times.predicted.plot + geom_bar(stat = "identity", fill = "#0033FF")
times.predicted.plot <- times.predicted.plot + xlab("Stage") + ylab("Percent of Time Predicted") + ggtitle("Percentage
Predicted per Stage")
times.predicted.plot <- times.predicted.plot + geom_text(aes(label = round(times.predicted, 1)), position =
position_dodge(0.75), vjust = -1)
times.predicted.plot

### WORKING WITH JUST THE SWITCHED DATA SINCE THIS LED TO OVERALL BETTER RESULTS
predictions <- read.csv("switchedPredictions.csv", header = TRUE)
classifier.name <- colnames(predictions[5:length(predictions)])
actual.stage <- as.vector(predictions[, "Actual"])
n <- length(actual.stage)

stages <- c("Stage 1", "Stage 2", "Stage 3", "Stage 4", "Stage 5")
numStages <- rep(0, length(stages))
for (i in 1:length(stages)){
  numStages[i] <- length(which(predictions[,"Actual"] == stages[i]))
}

percent.correct <- rep(0, length(stages))

for(i in 1:length(stages)){
  indices <- which(predictions[,"Actual"] == stages[i])
  for(j in 5:length(predictions)){
   percent.correct[i] <- percent.correct[i] + length(which(predictions[indices, j] == stages[i]))
  }
}

### percent.correct is an AVERAGE of the percent time a stage is predicted correctly
percent.correct <- 100*(round(percent.correct/(numStages*(length(classifier.name)-1)),3))
percent.correct <- data.frame(stages, percent.correct)

stage.plot <- ggplot(percent.correct, aes(y = percent.correct, x = stages))
stage.plot <- stage.plot + geom_bar(stat = "identity", fill = "#0033FF")
stage.plot <- stage.plot + xlab("Stage") + ylab("Percent Correctly Predicted")
stage.plot <- stage.plot + geom_text(aes(label = round(percent.correct, 1)), position = position_dodge(0.75), vjust =
-1)
stage.plot <- stage.plot + theme(axis.text.x = element_text(angle = 45, hjust = 1), text = element_text(size = 18))
stage.plot

###What percentage of the time was stage 1 predicted?
predictions <- read.csv("switchedPredictions.csv", header = TRUE)
times.predicted <- rep(0, length(stages))

for(i in 1:length(stages)){
  for(j in 5:length(predictions)){
    times.predicted[i] <- times.predicted[i] + length(which(predictions[,j] == stages[i]))
  }
}

times.predicted <- 100*(round(times.predicted/(sum(times.predicted)),5))
times.predicted <- data.frame(stages, times.predicted)
times.predicted.plot <- ggplot(times.predicted, aes(y = times.predicted, x = stages))
times.predicted.plot <- times.predicted.plot + geom_bar(stat = "identity", fill = "#0033FF")
times.predicted.plot <- times.predicted.plot + xlab("Stage") + ylab("Percent of Time Predicted")
times.predicted.plot <- times.predicted.plot + geom_text(aes(label = round(times.predicted, 1)), position =
position_dodge(0.75), vjust = -1)
```

```
times.predicted.plot <- times.predicted.plot + theme(axis.text.x = element_text(angle = 45, hjust = 1), text =
element_text(size = 18))
times.predicted.plot

###How many of each stage was there in the testing set?
trainingSet <- read.csv("DatabaseTesting.csv", header = TRUE)
stages.training <- rep(0, length(stages))
for (i in 1:length(stages)){
  stages.training[i] <- length(which(trainingSet[,"Stage"] == stages[i]))
}


####WORKING WITH THE ORIGINAL DATA, COPIED CODE BASICALLY FROM ABOVE
predictions <- read.csv("originalPredictions.csv", header = TRUE)
classifier.name <- colnames(predictions[5:length(predictions)])
actual.stage <- as.vector(predictions[, "Actual"])
n <- length(actual.stage)

stages <- c("Stage 1", "Stage 2", "Stage 3", "Stage 4", "Stage 5")
numStages <- rep(0, length(stages))
for (i in 1:length(stages)){
  numStages[i] <- length(which(predictions[,"Actual"] == stages[i]))
}

percent.correct <- rep(0, length(stages))

for(i in 1:length(stages)){
  indices <- which(predictions[,"Actual"] == stages[i])
  for(j in 5:length(predictions)){
    percent.correct[i] <- percent.correct[i] + length(which(predictions[indices, j] == stages[i]))
  }
}

### percent.correct is an AVERAGE of the percent time a stage is predicted correctly
percent.correct <- 100*(round(percent.correct/(numStages*(length(classifier.name)-1)),3))
percent.correct <- data.frame(stages, percent.correct)

stage.plot <- ggplot(percent.correct, aes(y = percent.correct, x = stages))
stage.plot <- stage.plot + geom_bar(stat = "identity", fill = "#0033FF")
stage.plot <- stage.plot + xlab("Stage") + ylab("Percent Correctly Predicted") + ggtitle("Average Precision per Stage")
stage.plot <- stage.plot + geom_text(aes(label = round(percent.correct, 1)), position = position_dodge(0.75), vjust =
-1)
stage.plot

###What percentage of the time was stage 1 predicted?
times.predicted <- rep(0, length(stages))

for(i in 1:length(stages)){
  for(j in 5:length(predictions)){
    times.predicted[i] <- times.predicted[i] + length(which(predictions[,j] == stages[i]))
  }
}

times.predicted <- 100*(round(times.predicted/(sum(times.predicted)),5))
times.predicted <- data.frame(stages, times.predicted)
times.predicted.plot <- ggplot(times.predicted, aes(y = times.predicted, x = stages))
times.predicted.plot <- times.predicted.plot + geom_bar(stat = "identity", fill = "#0033FF")
times.predicted.plot <- times.predicted.plot + xlab("Stage") + ylab("Percent of Time Predicted") + ggtitle("Percentage
Predicted per Stage")
times.predicted.plot <- times.predicted.plot + geom_text(aes(label = round(times.predicted, 1)), position =
position_dodge(0.75), vjust = -1)
times.predicted.plot
```