Work 1: RULES System

Supervised and Experiential Learning

Nikita Belooussov

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Barcelona, April 20, 2022

Contents

1 Description

The purpose of this report is to gain a better understanding the classification method of the Rule Extraction System(RULES) Algorithm. Specifically how it works and what are its advantages and disadvantages. The report will describe the data that was used, how the algorithm was programmed, the rules that it created, an analysis of the rules created, and lastly how to implement the code and improvements in future implementations of the code. It is important to note that while the code was based on the report [4]. The goal was to follow the report as closely as possible but some slight things were changed.

The data sets used in this report will be the Iris Dataset[2], Pima Diabetes Dataet[3], and the Acoustic Extinguisher Fire Dataset[1]. These were mainly chosen due to the number of attributes that they have and their sizes. For this report three sizes were required: Small(instance $\leq$ 500), Medium(500 $<$ instances $\leq$ 2000), and Large(instances $>$ 2000). The reason for choosing data sets with smaller amounts of attributes was in order to limit how many different combinations would be possible. Since the code had to go through every possible combination of attribute. The data sets mostly use numerical values, this was again done to limit the attributes, and will be described more in detail in the report. In the end, in all of the datasets the algorithm performed better than random chance.

2 Methods

This section will first describe how to implement the code. This is mainly to help set up everything up correctly and be able to see if similar results can be obtained. It is important to note, that random shuffling is implemented in the code. As a result, the same results will not be produced. The results should still be similar to the results achieved in this report.

The section will then go into a explanation of what each of the programs do, as well as some of the pseudo code. The code that relates to the algorithm is in the train.py section. The arffToCSV.py is just converting the data sets and the test.py is to test the rules created during training and extract the results from them.

2.1 Instructions

As mentioned before, this section is to describe how to be able to run the program. It is possible to run the notebook file that contains everything, or to do everything individually in the terminal. This section will describe how to run everything individually. The notebook file will describe how to run it. The first thing is going the be the structure of the directory system. A more detailed description of the inputs required to run the codes can be found in the rulesRunner.ipynb. The requirements for the versions of the libraries used can be found in the requirements.txt file. The directories need to arranged in this structure for the codes to work. If the structure is different then the paths in the programs will need to be changed. The structure setup can be seen in Figure 1. As mentioned before the codes being used are the arffToCSV.py, test.py, and train.py. The rest are files that are either produced by the codes or are the datasets. It is important to note that in Figure 1 not all of the files that are not the codes are listed. If the name has "*", it means multiple files with that ending are located in that directory. These files will generally just have the dataset name, in the case of .txt files there will be "training" in front of the data set name. Some of the dataset names have also been shortened.

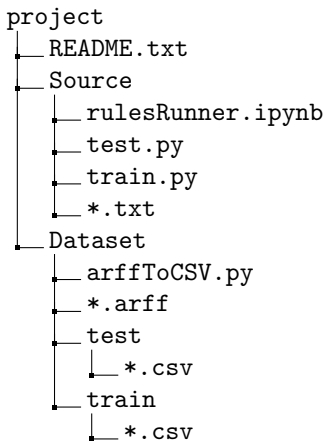

Figure 1: Directory Structure

The first code to run is the arffToCSV.py. This code takes in the dataset in the arff format and converts it to a csv file, splits the dataset into the train and test datasets, and lastly shuffles the dataset. The split is .85 of the total dataset is the training dataset and .15 of the total dataset is the test dataset. The shuffling is done in case the data set has some order to it as seen in the iris dataset. The code does take in the name of the file as system argument. As a result if the data sets are already split and in csv format, running arffToCSV is not necessary, and is only needed as a preprocessing step. The next step is to run the train.py code. This again takes in the name of the csv file being used without the extension as a system argument. This code will run and create the rules that will be used to classify the data. The rules will be stored in txt files that match the dataset names. These files also store some other useful information such as coverage of the rule to the training dataset, which instances they applied to, and the original statistics of the training dataset. Once this is done the test.py. Same input requirements are required as the train.py, with the name of the dataset being a system argument. The test.py code will not produce files and will only display the output in the terminal.

2.2 Program Explanation

This section will be giving a more in depth discussion of how each of the programs work. As mentioned before, the RULES algorithm is implemented in the training.py script. The rest are used to either prepare the datasets or test the rules created. The sections will also contain some psuedo code to help convey what happens in the scripts. The scripts will be discussed in order that they are used. It is important to note that the notebook will not be discussed, since it will be explained in side of the program and is mainly used to run all of these codes.

2.2.1 arffToCSV.py

As mentioned before this code is mainly used to convert the arff datasets to a csv dataset, split the dataset into a train and test dataset, and lastly shuffle the dataset. This is a fairly simple code and the psuedo code can be seen in Figure 2.

```
read dataset
shuffle dataset
train=.85 of dataset
test=remaining part of dataset
save train
save test
```

Figure 2: arffToCSV.py pseudo code

For this report it was necessary to do some one hot encoding on string values, but it was decided to do this in the train and test codes. This was done so that the data sets would remain intact through out the process.

Since the one hot encoding orders the columns alphabetically in this implementation, it was not an issue that the columns would be in the incorrect order. As mentioned before the datasets require to be shuffled since several of the datasets were ordered. The iris dataset had an ordering by the class, and the Acoustic Extinguisher Fire Dataset also seemed to be ordered. Thus, the dataset needed to be shuffled so that the training code would receive a more fair distribution of the different instances.

2.2.2 train.py

The train.py code is responsible for carrying out the RULES algorithim. As mentioned before the goal was to try to stick as close as possible to the process outlined in the report[4]. The simplified idea behind this algorithm is to find a set of rules that when applied only get one class of instances. These rules then can be used in the future to classify other instances. The approach taking in this report can be seen in Figure 3

```
read dataset
preprocess dataset
separate class from features
define input labels used for rules
for features in dataset:
    for label in input labels:
        store instances that fit input label in dictionary and statistics of the column

iterate i through [1:# of columns]:
    rule list=list of product of input labels list of input labels multiplied i times
    column list= list of all column combinations length of i, no repeats
    loop through rule list:
        loop through column list:
            go though input label and column list pairing column[i] with inputLabel[i]
            take only indexes that are present in all pairings
            check if all produce same class label
            if class label is the same remove instances and add rule to dictionary

store dictionary with rules and statistics
```

Figure 3: train.py pseudo code

As seen in Figure 3, the training code consists of two for loops. The first is to preprocess the dataset. The preprocessing done was only normalizing the dataset, and one hot encoding any of the features that were strings. The reasoning for one hot encoding was to help standardize the format of data, allowing for a simple process to be applied to everything. If the data was allowed to be a string, special rules would have to written for these features. By doing one hot encoding, the data is still preserved of it being different, but still allows for the rest of the code developed for numerical values to be applied. The RULES algorithm depends on having checking each unique value of the features to create the rules. Since it would be impractical to try to create with any numerical values labels were created for the features. For each feature the *standard deviation* and *mean* was calculated. The instance where then labeled if they where less than the $standard\,deviation - mean$, between $standard\,deviation - mean$ and the *mean*, between *mean* and $standard\,deviation + mean$, and lastly if it is bigger than $standard\,deviation + mean$. This allows to use these labels to create the rules, without having too many combinations to go through. This method was chosen due to it being a quick approach on how the data can be labeled and still being statistically relevant. The instance were then labeled by which of the categories it falls under. This was done so that all of these calculations would only have to be done once. The mean and std features for each feature were also stored so that can be used for the testing in the future.

The second main of the loop is the implementation of the RULES algorithm. The top level of the loop controls how long the combinations are. So through the first iteration the code is testing only one column with each rule possible, i.e. feature 1 and being less than $standard\,deviation - mean$. The next iteration would go through all of the product of the rules multiplied by the iteration it is through and all of the possible columns combinations, but limiting to no reproductions. So for example for rules testing if the value is $standard\,deviation - mean$ in all three columns was part of the list, but for the features list having the same feature tested 3 times with different rules was

not allowed. To further reduce the iterations, the order that feature appeared in did not matter, so if feature 1 was tested with feature 2 in the second iteration. There wouldn't be a test where feature 2 was listed first and feature 1 was listed after. Although this process does attempt to reduce how many iterations take place, the number of iterations that have to be done still increase very rapidly. A solution for this will be discussed later in the report but was not implemented due to the realization of the solution came too late to be implemented and delivered on time.

In order to test if the instances follow each of the rules a simple solution was implemented. There would be a list of all the possible instance indexes. Since the indexes that follow the input labels for certain features has been already determined, for each feature and input label combination, only those indexes would be kept. At the end if the indexes kept all provided the same class label, then this rule would be saved, otherwise it was discarded. The indexes were also removed from the possible matches in the future. At the end, when the code was testing all the columns with all of the possible rules, it was decided that if the rules do not provide a singular label, then the mode would be taken. This is because according to the algorithm, that if an instance is not label, then each of its feature labels will be taken as a rule. This would lead to an issue that the same rule might produce different class labels. In order to avoid this, the mode method was implemented. Both the mode method implemented and the removal of instances may be slight differences from the original RULES algorithm, but they were implemented to help either improve the performance of the code or future issues in labeling.

Once all of the possible combinations were tested, the resulting rules and some of the statistics about the dataset and rules were saved into a text file to be used later. According to the RULES algorithm once the combinations are tested the algorithm requires to go through the remaining instances and make the rules based off of their all of their features. This was covered by the last iteration of the second loop, since it would tested rules that had every feature in them and every possible individual rule and then taken the mode label. As mentioned, previously making rules based off these remaining instance would have made two different labels having the same set of rules. The RULES algorithm did not address how this problem should be resolved. As a result, it was decided that it would be safe to assume that in this case the label with most occurring instances would be the most likely to appear in the future.

Going through this section it is seen that the overall code follows the RULES algorithm, but there were some small changes. The first one was the mode implementation at the end. As mentioned, this would there was no direct method of solving this and the options were either completely ignoring the rule or taking the mode. Since the algorithm specifies to take the last instances and use their features to make the rules, the mode solution seemed to be the one that was most faithful to the idea of the RULES algorithm. The second was removing the instances as they were classified. This was done in order to help improve the speed of the code. This may produce an effect where rules become viable after certain other rules. This is because they would remove instances that are preventing a certain rule. Since rules are not going to be tested twice, and get more specific, it is not likely to happen. Overall, even though there were slight changes made to the RULES algorithm, they are not big enough to stop consider the produced algorithm something different and should still produce similar results.

2.2.3 test.py

The last code that is going to be discussed is the test.py script. This script was used to test how well the defined rules would preform on unseen data. The test script follows many of the same concept seen in the training code. This is due to it doing a very similar thing. The pseudo code for it can be seen if Figure 4

```
read dataset
read rules file
preprocess dataset
separate class from features
define rules
for features in dataset:
    for rule in rules:
        store instances that fit rule in dictionary and statistics of the column


iterate through rule combinations found in rule file:
    get instances following rule and remove them from overall pool
    get coverage of rule for total dataset

iterate through labeled indexes:
    calculate recall and precision for label

calculate overall accuracy, how much of the instances were covered
calculate accuracy of the covered instances
```

Figure 4: train.py pseudo code

As seen in Figure 4, the test script can be broken down into three main parts. The first part is the preprocessing of the data. Similar to the training the data is normalized and one hot encoded for any strings. It was then checked for the input labels that each instance feature follows. The input labels are the same as outlined in the train.py section. The values for the *mean* and *standarddeviation* can either be used from the training dataset or the ones calculated for this data set. The report will cover both situations. The next part is the classifying each of the instances according to the labeling rules developed during the training section. In this part we iterate through those rules and store the instances that match the rule in a dictionary and remove them from the overall pool. This mirrors what happens in the training process. The final part is calculating all of the performance values of the rules. This was done by simple going through the dictionary and calculating the true positives, false positives, true negatives, and false negatives for each class label. This should provide a good overall way of testing how well the rules performed with each dataset.

2.3 Dataset

This section will quickly go over the datasets chosen for this report. As mentioned previously, it was mandatory to have three different sizes of datasets: Small(instance ≤ 500), Medium($500 <$ instances ≤ 2000), and Large(instances > 2000). The Iris data was chosen for the small dataset. The Pima Diabetes dataset was chosen for the medium. Lastly the Acuostic Extinguisher Fire dataset was chosen for the large one.

2.3.1 Iris Dataset

The Iris dataset is a relatively small data set. It only has 150 instances, with three classes, and four features. The data set was ordered by its class, so it was important to shuffle the order of the instances in the dataset before training. One nice thing about the data set is that there are equal amounts of each label, as a result there is no biasing to a certain solution. Besides the class feature, the rest are numerical values so as a result no one hot encoding was done on this dataset. This dataset was used to test how the algorithm performs on a smaller data set and a method of quickly checking that everything appeared to work.

2.3.2 Pima Diabetes Dataset

The Pima Diabetes dataset has 768 instance with 8 attributes and two classes. This data set's order already seemed to be randomized, but in case it was not it was shuffled anyway. The data set again had only numerical values for the features, besides the string for the label. As a result, again no one hot encoding was done on this data set. The data set was not balanced, which may cause slight biasing. It is about is about 2 to 1 of "tested_negative"

label to "tested_positive". Although there is a bias, it should not heavily impact the results of the performance the algorithm. This dataset was used to further test algorithm after it was shown to work with the smaller dataset.

2.3.3 Acoustic Extinguisher Fire Dataset

Lastly is the Acoustic Extinguisher Fire dataset, which may be referred to as the Fire dataset in the future. This was by far the largest dataset out of the ones tested. There are 17,442 instances in this data set, with six attributes and two classes. One of the attributes does contain string values, and as a result needed to be one hot encoded. There were four values for the string values. This means that the dataset ended up having nine attributes instead of six. Since the string attribute was dropped, but replaced by the one hot encoded ones. As mentioned previously, this was done to help simplify the training code and the operations done in it. This dataset also seemed to have some order to it. As a result the shuffling was again an important step to perform. The label was a numerical value, but this should not interfere with any part of the scripts developed. The dataset was also pretty much balanced between the two labels, so again avoiding some sort of biasing. This data set was used to help show how the algorithm operates on large amounts of data and make sure that it could handle data sets with string values.

2.4 Measurements

In this report there were several measurements that were taken. This will be broken down by the measurements taken for the rules created, then the measurements taken per output label, and lastly the measurements of the system performance depending on which statistics were used. First was the precision of each rule. This was calculated by dividing the number of times the rule predicted correctly out of the total predictions it did. the next was the coverage of the data sets. This was just the number of instances it covered, correct or incorrectly divided by the the total data set length. The Accuracy of the system overall. This was calculated by how many times it made a correct prediction divided by the total length of the test dataset. The coverage the labels, which was calculated by dividing the number of correct predictions of the rule by the total number of that label that the rules is predicting in the data set. These all will be on one table. It is important to note that the system accuracy will be in the precision column, in order to save space. It will be mentioned in each explanation of the table, to make sure it is clear.

The next set of measurements taken were how will the system performed with each output label. The equations for used to measure the performance for each label can be seen below.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{1}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

In the equations above a $TruePositive$ is when the system correctly predicts an instance. A $FalseNegative$ is when the system says that the label is not the label that it is trying to predict. So for example if the rule is for predicting cars, any instance that is not labeled as a car but is one, would be a false negative. A $FalsePositive$ is when a the true label is not what the rule was predicting. So for example if the instance was labeled as bike by the rule, but the true label is car.

The last set of measurements will be the total accuracy, covered accuracy, and covered. The total accuracy is gotten the same was as described above, by just dividing the total amount of times an instance was predicted correctly by the total amount of instances. In the covered accuracy the total amount of instances is changed to how many instances were predicted. The covered measurement will be just dividing how many instances were covered by a rule divided by the total length of the dataset. This was done to see how well the rules work when they are triggered, in case the data set is not well covered. This will be in the table where looking at the comparison between if recalculating the statistics for the input label helps create a better system.

These measurements will provide a good representation of how well the algorithm is preforming. The measurements will be able to show how well each individual rule is preforming, how well the system is preforming, and if it is worth to do recalculations for the input label ranges. Another benefit of using these measurements will be that it will be able to get an idea of how well the system would preform if it could cover all of the instances. Overall, the measurements will help obtain information about the general performance as well as a more detailed overview.

3 Results

This section will be discussing the results of the RULES algorithm applied. The section is broken down into the different datasets. The main focus will be describing how the rules preformed and the overall system. There will also be a comparison between using the train dataset statistics for the rules or the test dataset statistics for the rules. By this it means that during labeling if it is better to use the *standarddeviation* and *mean* from the training data set or to recalculate them using the test dataset. In general it should be better to use the *standarddeviation* and *mean* from the training dataset. This is due to having more control of the datasets and being able to adjust for biasing or any other things that could mess up the the outcome. If a biased test data set is given and the statistics are biased off of it, then the rules will not preform as well as they would if the statistics being used were from the training dataset. Unless stated otherwise the values obtained are going to be when the test script is using the *mean* and *standarddeviation* values obtained from the train dataset. This is due to it being the more robust approach and wont depend on the test dataset being given. It is important to understand the labels that are used by the rules. As mentioned before there are four possible labels, and an explanation of each can be seen in Table 1.

Table 1: Labels

Label	Meaning
higherOut	greater than *standarddeviation+ mean*
higherIn	between *mean* and *mean+standarddeviation*
lowerOut	less than *mean-standarddeviation*
lowerIn	between *mean* and *mean-standarddeviation*

As mentioned before, these were chosen so that it would be easier to handle multiple different types of data sets. If all of the columns have the same amount of labels and there are no special cases, then the process of automating all of the steps becomes much easier. It is likely that having more labels, would make the system perform better. This is due it being more likely that a rules covers only one instance, and that any outliers of another class would be excluded.

3.1 Iris Dataset

As mentioned before the iris dataset was the smallest dataset that was tested. In total the RULES algorithm created a total of 11 rules. These rules can be seen in Table 2. The table shows the feature uses and the label for the feature. It is important to note that the feature is in line with the label that is being used. The coverage columns show how much of the overall dataset that rule covers. So for example the first rule covers about 33% of the instances in the train data set and about 30% of the the test dataset. The forced columns shows if the mode rule was implemented, which means that the algorithm was testing all of the feature and label combinations and that there was more than one expected output label, so it took which appeared most often. Since there is not many rules for this dataset all were able to be shown that the algorithm produced. It is important to note that the bottom right value in 2 is the accuracy of the entire system and not the precision even though it is in the precision column.

Table 2: Iris Rules Coverage and Precision

Features	Labels	Expected Output	Forced	coverage of Train Dataset	coverage of Train Label	coverage of Test Dataset	coverage of Test Label	Precision
Petal Length	lowerOut	Iris-setosa	No	33.07%	100%	30.43%	87.5%	100%
Sepal Width	higherOut	Iris-virginica	No	1.57%	4.76%	26.09%	50%	66.6%
Petal Length	higherOut	Iris-virginica	No	14.96%	45.24%	4.35%	12.5%	100%
Petal Width	higherOut	Iris-virginica	No	7.87%	23.81%	0%	0%	-
Petal Length	lowerIn	Iris-versicolor	No	5.51%	16.28%	0%	0%	-
Petal Width	lowerIn	Iris-versicolor	No	3.14%	9.30%	0%	0%	-
Sepal Length, Sepal Width	lowerOut lowerOut	Iris-virginica	No	7.87%	2.38%	0%	0%	-
Sepal Width, Petal Length	lowerOut higherIn	Iris-versicolor	No	4.72%	13.96%	4.35%	0%	0%
Sepal Length, Sepal Width	higherOut LowerIn	Iris-versicolor	No	1.57%	4.65%	0%	0%	-
Sepal Length, Sepal Width	higherOut higherIn	Iris-versicolor	No	3.14%	9.30%	0%	0%	-
Sepal Length, Sepal Width, Petal Length, Petal Width	lowerIn lowerIn higherIn higherIn	Iris-versicolor	Yes	6.30%	18.60%	4.35%	14.29%	100%
Sepal Length, Sepal Width, Petal Length, Petal Width	higherIn lowerIn higherIn higherIn	Iris-versicolor	Yes	7.09%	20.93%	4.35%	0%	0%
Sepal Length, Sepal Width, Petal Length, Petal Width	higherIn higherIn higherIn higherIn	Iris-versicolor	Yes	2.36%	6.98%	8.70%	28.57%	100%
Total				92.12%		82.60%		65.21%

In table 2 it can be observed that in most cases the rules are fairly accurate if they show up. A lot of the rules did not show up at all in the test dataset and only appeared in the train data set. This is likely due to how small the test dataset ended up being. The test dataset was only 23 instances. So it was likely to be missing the less common rules. This is backed up because many of the rules that had a smaller percentage coverage in the train data set, were the ones that were missing in the test dataset. There were also some combinations that were not covered by the train dataset. This is seen because not the entire test dataset was covered. This could be resolved with a larger dataset. The overall accuracy shown is 65.21% but this is because of how many instances it did not have a rule for. If looking at the the instances that it was able to cover then it got the correct label 78.94% of the time. Although this is not a very high accuracy, it is fairly decent for how simple the algorithm is.

Looking at the classes it is seen that only one rule is needed to identify pretty much all of the iris setosas and it has a very high precision rating. It is interesting to note that a lot more of the specific rules, rules with more requiring more input labels, are attributed to the iris versicolor. This is likely due to the most of the iris virginca instances being labeled by the more general rules. As a result, at then end when the when there is a rule that needs to be determined by the mode application, there will be more iris versicolor instances to outweigh the iris virginicas still not labeled. Another interesting note is that there is not much of a difference between how much coverage there is between more specific rules and more general rules. It is seen in the test dataset that the second rule has the second highest coverage of its labels while in the train it has one of the lowest. As a result, it shows that even though a rule may not be very useful in the train dataset it can be in the test. Although this may be due to the fact that the dataset is small and as result randomly shuffling and splitting instances is more likely to lead to a stronger bias. A better look at how it performed overall with each output label can be seen if Figure 3

Table 3: Iris Prediction Analysis

Label	Precision	Recall	F1
Iris-setosa	1	0.875	.933
Iris-virginica	.714	.625	.667
Iris-versicolor	.6	.428	.500

Figure 3 shows that out of all the labels the Iris setosa seemed to be the best performing one. This is likely due to it being more distinct than the other two, and as its values being considered outliers to the other output labels.

This idea is mainly biased off of the fact that only one attribute was required to get all of the instances in the training dataset. It is surprising to note that the precision for Iris setosa is 1, meaning that every instance that it labeled as an Iris setosa was labelled correctly. This helps reinforce the idea that Iris setosa was very different than the other two labels at least in the feature that the rule is using. In the case of iris verginica and iris versicolor, it is clear that they are much closer related to each other. It is very likely that if a deeper analysis is done it would be seen that they are often mistaken for each other by the rules. It is also likely that if a more input labels would have been used, with smaller numerical ranges, then it would have created a better system. In Table 4, the performance of the system is shown when the ranges for the input labels are recalculated.

Table 4: Iris Statistics Comparison

Method	Covered	Total Accuracy	Covered Accuracy
Test Statistics	100%	86.96%	86.96%
Train Statistics	82.61%	65.22%	87.95%

In table 4, the comparison of how the rules are made for testing is seen. The train statistics is when the the *average* and *standarddeviation* used for creating the rules for the test are from the training data set, and the test statistics is when they are created from the test dataset. Interestingly, the results show that the use of the test data set seemed to produce better results. This may be due to that by allowing for the *average* and *standarddeviation* to be recalculated, it allows the rules to be better fitted to the test data set than they were when created for the train. It is important to note that only the ranges that the input labels apply to are changing, the rules themselves still are the same.

The RULES algorithm appears to be able to work for iris dataset, when it is able to apply the rules. The mode rules that were implemented seem to be slightly more beneficial then a hindrance. The main issue that the algorithm seems to be having is that there is not enough data to create all the rules it needs to and that iris virginica and iris versicolor are very similar to each other. It might be possible that if the once the setosa instances were labeled, they could be removed and then the average and standard deviation taken again. This might make it easier to label the iris versicolor. A larger dataset would also be more beneficial. Another solution would be adding more input labels, and having them each cover a smaller range of possible values. As a result making it less likely that different instances would be mixed into one input label. It is also surprising how much of the dataset is covered by rules with only one input label. The petal length is the only rule used to determine if the instance is an iris setosa. Since about a third of the data set is this label, it shows that this was pretty much the only rule required differentiate the iris setosa from the other two output labels. All the rest of the rules were used to differentiate iris virginica and iris versicolor from each other. Overall, the system seems to require either more data, more input labels, or an extra step in preprocessing the data to make it work better due to how close iris virginica and iris versicolor are when compared to iris setosa.

3.2 Pima Diabetes Dataset

The next data set that was will be discussed is the Pima Diabetes dataset. This dataset has more instances and attributes when compared to the iris. So it was expected that there would be more rules and that they will be less general, so they would contain more input labels. This expectation was shown to be true. In the rules there was not a single rule that only used a single input label. All rules depended on at least two input labels to make their classifications. This is likely due to their being more instances and as a result more chances for outlier instances of one label interfering with a rule for another, that could be general. There was a total 333 rules developed for this dataset by the algorithm, many of which covered less than 1% of the training and test data sets. In table 5, some of the rules can be seen. Due to the large amount of rules developed it was decided to only show the rules that covered more than 1% of the test dataset. Since these were shown to be the more important rules than the other ones. If it was below 1% it also meant that it was also labeling only one instance. There were also many rules that did not apply to the test dataset at all. As a reminder the value in the total row and in the precision column is the measurement for the accuracy of the entire system and not the precision.

Table 5: Pima Diabetes Rules Coverage and Precision

Features	Labels	Expected Output	Forced	coverage of Train Dataset	coverage of Train Labels	coverage of Test Dataset	coverage of Test Labels	Precision
Glucose Concentration Skin Thickness	lowerOut lowerOut	tested_negative	No	2%	3.07%	2.59%	3.95%	100%
Glucose Concentration BMI	lowerOut higherOut	tested_negative	No	0.5%	0.47%	4.31%	6.58%	100%
# of time Pregnant Skin Thickness BMI	lowerOut lowerOut higherOut	tested_negative	No	0.3%	0.47%	1.72%	1.32%	50%
Skin Thickness BMI Age	lowerOut higherOut higherOut	tested_negative	No	0.2%	0.24%	5.17%	1.32%	16.7%
# of time Pregnant Glucose Concentration Insulin	lowerOut higherOut lowerIn	tested_positive	No	0.3%	0.88%	2.59%	7.50%	100%
Glucose Concentration Skin Thickness Diabetes Pedigree	lowerOut higherOut lowerIn	tested_negative	No	0.2%	0.24%	3.45%	5.25%	100%
# of time Pregnant Glucose Concentration Insulin	lowerOut higherIn lowerIn	tested_negative	No	0.3%	0.47%	1.72%	2.63%	100%
Blood Pressure Skin Thickness BMI	higherOut lowerOut higerOut	tested_positive	No	0.2%	0.44%	1.72%	2.5%	50%
# of time Pregnant Glucose Concentration Skin Thickness	higherOut higherOut higherOut	tested_positive	No	0.3%	0.88%	2.59%	7.50%	100%
# of time Pregnant Skin Thickness Insulin	higherOut higherOut higherOut	tested_positive	No	0.5%	1.32%	2.59%	5.00%	66.7%
# of time Pregnant Skin Thickness Age	higherOut higherOut higherOut	tested_positive	No	0.3%	0.31%	1.72	2.5%	50.0%
Blood Pressure Skin Thickness Insulin	higherOut higherOut higherOut	tested_positive	No	0.2%	0.44%	5.17	7.5%	50.0%
Thickness BMI Age	higherOut higherOut higerIn	tested_negative	No	0.3%	0.44%	4.31%	5.00%	40%
Insulin BMI Diabetes Pedigree	higherOut lowerIn higerOut	tested_negative	No	0.3%	0.47%	1.72%	2.63%	100%
Skin Thickness Insulin Diabetes Pedigree	higherOut lowerIn higherIn	tested_positive	No	0.3%	0.88%	1.72%	0%	0%
# of time Pregnant Glucose Concentration Skin Thickness	higherOut higherIn lowerOut	tested_positive	No	0.3%	0.88%	1.72%	0%	0%
Insulin Diabetes Pedigree Age	higherOut higherIn lowerIn	tested_negative	No	0,3%	0.47%	3.45%	5.26%	100%
# of time Pregnant Skin Thickness Insulin	lowerIn higherOut higherOut	tested_negative	No	0.2%	0.24%	5.17%	6.57%	83.3%
Glucose Concentration Skin Thickness BMI	lowerIn higherOut higherOut	tested_negative	No	0.2%	0.24%	1.72%	2.63%	100%
Blood Pressure BMI Age	higherIn lowerIn higherIn	tested_positive	No	0.5%	1.32%	3.45%	0%	0%
Total				92.12%		99.14%		62.9%

In table 5, it is seen that the RULES system was able to cover pretty much all of the instances. There was a significant improvement in the coverage of the test dataset. This is likely due to there being more instances. As a result, more of the possible combinations were present in the training and had rules created for them. There was a slight dip in the accuracy for the data set. Going down to 62.9% for the overall data set from 65.23%. The accuracy of the test dataset if only the covered instances of covered dropped significantly from 78.94% down to 63.45%. This may be due to that even though the dataset is larger than the iris dataset, due to it having more attributes it is more complex. As a result, the algorithm was not able to create the correct set of rules to cover this problem. It is also possible that the input labels created did not allow for enough variations, and as a result had a more difficult time differentiating the two classes from each other. It was also seen that the coverage of the labels between the test and train datasets did not have a strong correlation. Since this is a larger dataset it would have been expected that two datasets would have been more equally mixed, and as a result similar coverage would be expected.

Although none were listed there were a total of 27 rules that were forced. This means that at the end all of the instances that were left over were very similar to each other. Again, it was shown to be fairly helpful to use the mode system for the last set of combinations. There were no unforced rules in the system, showing that those instances were very similar to each other. The system was also not able to make any new rules between using four attributes and using all eight attributes. Again showing that the remaining instances were very similar to each

other.

A more broad look into how it preformed with each label shows that the system's lower performance may be due issues with one of the classifications. The system had issues with the tested_positive label. Specifically its precision was preforming poorly. It had a score of 0.479 compared to the tested_negative precision score of 0.746. This is likely the reason of why the recall score of the tested_negative is so much lower than its precision score. This likely means that the tested_negative instances have a broader range than the tested positive. As a result, they are more likely to end up in the ranges that the tested_positive are than the other way around. All of these discussed results can be seen in Table 6

Table 6: Pima Diabetes Prediction Analysis

Label	Precision	Recall	F1
tested_negative	0.746	0.658	0.699
tested_positive	0.479	0.575	0.523

As mentioned before, in Table 6 it is shown that test_negative is preforming a lot better than tested_positive. Another possibility not mentioned previously could be that the lower performance is due to there being less tested_positive instances than the tested_negative. Due to there being less instances, the algorithm had greater difficulty identifying what rules are required for the tested_positive. The best solution would be to try to obtain more instances that ended up being tested_positive. This would allow for the algorithm to make more rules for that category. Under sampling the tested_negative instances would likely only harm the overall performance. It was also discovered that recalculating the statistics used for the labels did improve the performance. This can be seen in Table 7.

Table 7: Pima Diabetes Statistics Comparison

Method	Covered	Total Accuracy	Covered Accuracy
Test Statistics	97.41%	71.55%	73.45%
Train Statistics	99.14%	62.93%	63.48%

In table 7, it is seen that again taking the statistics from the test data set seems to increase the performance of the model. Although the coverage does drop slightly. The drop might be due to some of the instances that were near edges of an input label being shifted out by the new calculations. Again the performance increase might be due to the more information being obtained from the test data set and allowing the rules to be adapted slightly more to the data set, instead of being fixed by the training data set.

Overall on the pima diabetes dataset, it was shown that the algorithm is able to work on larger data sets as well. As expected the rules with a single attribute stopped being produced, with the lowest number of attributes used per rule being two. The performance was slightly worse across the entire dataset. This was mainly due to the algorithm being able to cover a much larger percentage of the dataset. It performed significantly worse if only accounting for the instances that it did cover. This is likely due to the data set being more complex than the iris dataset, and the algorithm either not having enough flexibility or data to be able to perform well. Also as previously it was shown that recalculating the ranges for the input labels was proven to be beneficial.

3.3 Acoustic Extinguisher Fire Dataset

The last dataset that was tested was the Acoustic Extinguisher Fire dataset(fire dataset). This dataset officially only has six attributes, but due to one of the attributes containing string values it was one hot encoded. As a result there was a total of nine attributes in this dataset. It is important to not that the hot one encoding in this script arranges the columns in alphabetical order. As a result, in most scenarios in this dataset it will not matter that it is happening after the data was split. The output labels from the dataset were also either a "1" or a "0" as numerical values. This will not be an issue for the scripts developed.

The analysis of the rules show that there are some input labels that were a lot more important to the labeling of data than others. Looking at Table 8, it is seen that in most of the rules created and had a good coverage of the data set the distance feature is used. There are also some features that are not used at all in the ones listed, but all of the features do show up. There was a total of 389 rules created. In order to focus on the more important one, in

Table 8, only ones with higher than 1% coverage of the test dataset were listed. It is important to note that again the last value in the precision column is the accuracy of the entire system, and not the precision.

Table 8: Acoustic Extinguisher Fire Coverage and Precision

Features	Labels	Expected Output	Forced	Coverage of Train Dataset	Coverage of Train Labels	Coverage of Test Dataset	Coverage of Test Labels	Precision
Distance Frequency	lowerOut higherOut	1	No	13.05%	26.15%	12.46%	25.39%	100%
Frequency LPG	lowerOut higherOut	0	No	1.90%	3.79%	2.41%	4.74%	100%
Distance Gasoline	lowerOut lowerIn	1	No	2.16%	4.32%	1.75%	4.21%	100%
Distance Gasoline	higherOut higherOut	0	No	2.65%	2.65%	2.83%	5.55%	100%
Distance Frequency	higherIn lowerOut	0	No	4.34%	8.67%	4.62%	9.08%	100%
Distance Gasoline Kerosene	higherOut higherIn higherOut	0	No	1.11%	2.22%	1.14%	2.25%	100%
Distance Gasoline Thinner	higherOut higherIn lowerIn	0	No	1.75%	3.49%	1.57%	3.08%	100%
Distance Gasoline Kerosene	lowerIn lowerIn higherOut	1	No	1.40%	2.81%	1.68%	3.43%	100%
Size Distance Gasoline	lowerIn lowerIn lowerIn	1	No	1.41%	2.82%	1.64%	3.35%	100%
Size Distance Thinner	higherIn higherOut higherOut	0	No	1.19%	2.38%	1.34%	2.63%	100%
Size Frequency Gasoline	higherIn lowerIn higherIn	0	No	1.34%	2.68%	1.57%	3.08%	100%
Total				93.15%		100%		92.9%

In table 8, it quickly becomes evident that the algorithm was able to do a very good job in being able classify the instances with a 92.9% accuracy over the entire test dataset. It was also the only dataset where all the instances were covered by the rules. This is likely due to how large the dataset was, and how relatively few attributes there was. This allowed for the algorithm to encounter all the possible combinations of features that would appear and be able to classify them. The algorithm was not able to cover the entire train dataset but still was able to cover 93.15% of it. Same as before, the rules that included all of the attributes were forced, meaning that the mode of the instances was taken as the rule. There was a total of 233 of these rules. It is hard to determine if these were a net positive on the performance, but it does appear as that they were, although each individual one covered less than 1% of the test data set. It is also interesting to note that this was the only dataset were the coverage of the test labels and the coverage of the train labels are pretty close to each other. This is likely due to the data set being so large that by splitting the dataset did not introduce any biases, so the distribution between the two datasets was still fairly similar.

There are some rules that were created, which do not seem to make sense. These rules are the ones where two different fuels are used for the rule creation. This can be seen in the rule such as Distance = lowerIn, Gasoline=lowerIn, Kerosene=higherOut. This rules implies that the distance needs to be a lower outlier, the fuel is not Gasoline and the fuel is Kerosene. Since the fuel can only be Gasoline,LPG,Kerosene, or Thinner, it means that naming two types of fuel would redundant. The likely explanation is that when the rules was tested for an earlier combination, such as Distance=lowerIn, Kerosene=higherOut, some of the instances that it captured had other labels. Since when instances are matched to a rule they are removed to see if they fit future rules, when the script came back to another combination like this, it was able to create it since the instances that were problematic before were removed. This shows that the order in which the rules are applied is important to classification. As mentioned before this is not the exact implementation that is stated in the RULES report [4], but this small change does not change the overall concept of how the algorithm works, and may be a benefit in the end.

The analysis of the labels further shows that the algorithm preformed extraordinarily well for this dataset. Both the precision and recall scores are above 90%. These results are seen in Table 9 This shows that the algorithm is function very well, especially considering how simple the algorithm is.

Table 9: Acoustic Extinguisher Fire Prediction Analysis

Label	Precision	Recall	F1
0	0.914	0.923	0.919
1	0.924	0.914	0.919

This data set was very balanced, which may have aided in making the precision and recall of the two classes be so close to each other. Although, it is far more likely that the reason for this was actually the large amount of instances that there were. Due to the large amount of instances the algorithm was able to see every combination that would produce a certain output. Using the original features there were only 4096 combinations of instances possible after labeling was applied. This does not count how the rules could be combined with each other. The data set has 17,442 instances, which means that it likely covered most of the cases that typically occur. The significantly larger number of instances than combinations is the likely reason of why the algorithm performed so well, instead of it being well balanced, although likely helped as well. In Table 10, the results of recalculating the input label ranges can be seen.

Table 10: Acoustic Extinguisher Fire Statistics Comparison

Method	Covered	Total Accuracy	Covered Accuracy
Test Statistics	99.85%	92.85%	92.99%
Train Statistics	100%	92.93%	92.93%

In Table 10 it is shown that similar to the pima diabetes, the coverage of the test data set does drop slightly when its statistics are used to create the rules. Unlike the other two datasets the total accuracy does drop slightly. This is likely due to the rows that are not being covered being treated as incorrect labels. As a result, the slight drop in performance decreases the total accuracy of the system. This is confirmed by the covered accuracy where the accuracy increases slightly. This may not be due to more instances being labeled correctly, but instead it may be just that the instances that were not labeled correctly before are the ones being not covered now. As a result the accuracy increases for the covered accuracy without actually classifying more correctly.

Overall, the fire dataset was the best performing dataset. This was likely due to the large amount of data and possibly the fact that it was more distributed than other datasets. The data set also demonstrated that with the system implemented by the scripts the order of testing rules is important, and that some rules were created due to some instances being removed by other rules. It was also the first dataset that had the test dataset completely covered, which was due to the mode implementation for the last round of rule creation. This and the order of how the rules are created and tested seem to help create a better preforming algorithm than the original RULES system, while still remaining faithful to the outline of how it is supposed to operate.

4 Conclusion

The conclusion of this report is that the RULES algorithm is clearly an algorithm that works fairly well considering how simple of a algorithm it is. There were a couple of things that could have been adjusted in the scripts that this report used to help the functionality of the algorithm. There are also lessons learned from each data set that showed the strengths of the RULES algorithm. Lastly, the developed script did end up some slight differences from the original algorithm that seemed to prove beneficial. All of these topics will be discussed further in depth.

The main disadvantage of how the script was programmed is that it takes a very long time to go through all of the combinations of the input labels and features. Especially if there are a lot of features. Thus is not surprising since with the number of possible combinations increase extremely rapidly with each feature added. The way to fix this would be go through the instances and testing the combinations in them, until a rule can be created. This makes it so that only the combinations present are tested, instead of testing all of the possible combinations. Thus, the number of iterations being done would be decreased significantly. It would not hurt the performance of doing the script this way in anyway, due to skipping the combinations that are not present in the training data set would not be created anyway. The realization of creating the script this way was realized too late and there was not enough time to implement it.

Another disadvantage of the implementation of rules that was implemented is that if one classification of data is very different from another two, then it will cause a lot of issues with classifying the two more similar classes, even if when they were tested with only each other then it would have been easier. For example if one attribute class one is typically around 1, class two is typically around 5, and class 3 is typically around 100, then the algorithm would not be able to use the feature to find the difference between class one and class two. This is due to how the labels were decided to be created. This may have happened with the Iris dataset with the Petal Length feature. If the example is less extreme then having more possible labels may be able to solve this issue. Another solution would be trying to go through the features and trying to see if there are already some sort of clustering happening. Then create a label system around that clustering that was discovered. This system was not applied due to the time limitations, and it was felt that the system developed would be enough to demonstrate the functionality of the RULES algorithm.

The next topic is the stuff learned about the RULES algorithm from the datasets. The datasets showed that due to the increase of possible combinations, if there are more attributes in a data set then there needs to be more instances, in order to cover the possible combinations. The best performing dataset was the fire dataset, which had more than enough instances to cover the possible combinations. It is important to note that is not important to cover every possible combination, since not every combination is a possible outcome. It just increases the chances of catching outliers and being able to cover them with a rule. As seen in the fire data set, where the test data set was completely covered because all the outlier possibilities were covered in the training dataset. This is lesson also is generally applied to other classification algorithms where having more instances, generally allows them to preform better.

One of the surprising things about the algorithm was that it was hard to tell if the rules with more attributes had a higher precision than the rules with less when they were able to be used. It would have been expected that the rules with more attributes would perform better, because they are covering a narrower possible field of possibly instances. This appears not to be the case, an explanation could be that since the rules are created from the present data, any outliers are as likely to happen to any of the rule sets as the others. Although as previously stated it is hard to tell if this is the case. In the iris data set most of the rules that were more specific were not triggered. In the fire dataset pretty much rule that was developed was preforming well, and it is hard to tell only from the pima diabetes data set.

Another interesting thing shown with the datasets was that changing the input labels to use the statistics from the training dataset to the test dataset improved the performance of the system. For the larger datasets this may be due to the labels that were labeled incorrectly before being on the edge of a rule, and as a result with the changes of the range it would cause them to not be in the rule anymore. As a result improving the accuracy of the when looking at what was covered, while only slightly damaging the performance and coverage. For datasets where it seems like there may have not been enough instances, like iris and pima diabetes, the change greatly improved the accuracy. This may be due to by allowing the rules to change the rules were allowed to fit better to the dataset while still maintaining the patterns learned from the training. Although this seems like it might be a nice quick way to improve the performance it does come at a risk. If the test dataset is not well balanced then, the accuracy would decrease significantly, or if it is not close to the training dataset at all. For example, this method would not work if a class is missing, or only testing a singular instance. As a result, it is best to use this with caution.

As mentioned previously in the report there were some things that might have been changed from the original implementation. The main ones were the mode implementation applied during the last test for the rules and the fact that the order that the rules were tested mattered. The mode rule was implemented to help account for the last step in the RULES algorithm to go over the remaining instances and make rules from them. In order to avoid having the same rules with multiple rules the mode of the instances fitting the rule was used to assign the label to the rule. Overall, it seems that this implementation was beneficial.It appears that all of the datasets', the rules with that had to be created using the mode implementation were used to cover the entire dataset and generally had an precision higher than 50%. So as a result it seems that this change was beneficial and without drastically changing the RULES algorithm.

The other change is a much larger to the basic RULES algorithm. This was that if a instance matched to a rule, it would be removed and future rules would not check if that instance would fall into the new rule. The effect becomes very evident in the fire data set where some of the rules would not make sense unless this is the case. It does seem to be a beneficial thing, since it allows for more complex rules to be developed, even though it does not appear so at first. Since the way the testing works follows the same order that the rules are made there was no issue, but this does mean that if a single rule is tested without going through the same path then the results could be very different. All of this being said, the algorithm was still being followed, so the main idea was still being implemented and used. As a result this was a slight change that allowed for a more complex system, but still

followed the main idea of RULES, and as a result likely improved the performance of the system developed.

Overall this report shows that the RULES algorithm is a valid system to try to predict labels for data sets. As with other methods some data analysis and having more instances will help create a better system, but even when that is not the case it still produces a working model. The algorithm can have slight changes to it in order to help improve its performance. This implementation also clearly was not the best version of the algorithm and there a couple of things that could have been improved if there was more time. These mainly would help it run faster by limiting the search space, and improving the performance by allowing more flexible labels being applied to the numerical data. In the end, it seems that the main issue that the algorithm had with some of the data sets tested in this report was that they were too small and it required more instances to be able to work well, as it did in the fire data set. If this requirement was meet then the algorithm is able to work very well.

References

[1] Murat KOKLU. Acoustic extinguisher fire dataset, Apr 2022.

[2] UCI Machine Learning. Iris species, Sep 2016.

[3] UCI Machine Learning. Pima indians diabetes database, Oct 2016.

[4] D.T. Pham and M.S. Aksoy. Rules: A simple rule extraction system. *Expert Systems with Applications*, 8(1):59–65, 1995.