CSC 558 Data Mining and Predictive Analytics II, Fall 2023

Dr. Dale E. Parson, Assignment 2, Classification of audio data samples from assignment 1 for predicting numeric white-noise amplification level for the signals' generators.[1] We will also investigate discretizing the white-noise target attribute (class) and other non-target attributes.

DUE By 11:59 PM on Monday March 13, 2023 via D2L Assignment 2. The standard 10% per day deduction for late assignments applies.

There will be one in-class work session for this assignment. Start early and come prepared to ask questions. The 13th is the start of spring break, so no office hours that day. The preceding weeks' office hours are at usual times & modalities.

Download the following ZIP file via a web browser and unzip.

https://acad.kutztown.edu/~parson/whitenoise558sp2023.problem.zip

You can do all your work on a Kutztown PC or your home machine, no need for acad.

You will see the following files in this **whitenoise558sp2023** directory:

README.txt	Your answers to Q1 through Q20 below go here, in the required format.
csc558wn10Ksp2023.arff	The handout ARFF file for assignment 2, **wn** means white noise.
The following four files are from csc558wn10Ksp2023.arff without the five tnoign==0 instances.	
csc558wnTrain100sp2023.arff	100 initial-order training instances from csc558wn10Ksp2023NoTid0.arff.
csc558wnTest9900sp2023.arff	9900 remaining initial-order test instances of csc558wn10Ksp2023NoTid0.arff.
csc558wnTrain100Rndsp2023.arff	100 random-order instances from csc558wn10Ksp2023NoTid0.arff.
csc558wnTest9900Rndsp2023.arff	9900 other random instances from csc558wn10Ksp2023NoTid0.arff.

ALL OF YOUR ANSWERS FOR Q1 through Q20 BELOW MUST GO INTO THE README.txt file supplied as part of assignment handout directory **whitenoise558sp2023**. You will lose an automatic 20% of the assignment if you do not adhere to this requirement.

1. Open **csc558wn10Ksp2023.arff** in Weka's Preprocess tab. This is the same dataset used for assignment 1, with AddExpression's derived attributes already in place, and with **tosc** and **tid** removed; tagged numeric attribute **tnoign**, which is the gain on the white-noise generator, is the class (a.k.a. target attribute) of assignment 2. Where assignment 1 had a nominal attribute as the class, this assignment has **tnoign** as a numeric class attribute.

Here are the attributes in csc558wn10Ksp2023.arff.

centroid	**Raw** spectral centroid extracted from the audio .wav file.
rms	**Raw** root-mean-squared measure of signal strength extracted from the audio .wav file.
roll25	**Raw** frequency where 25% of the energy rolls off, extracted from the audio .wav file.
roll50	**Raw** frequency where 50% of the energy rolls off, extracted from the audio .wav file.

[1] See Assn1AudioOverview http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.html and in-class discussion on the Zoom archives.

roll75 **Raw** frequency where 75% of the energy rolls off, extracted from the audio .wav file. **amplbin1** through **amplbin19** Normalized amplitudes of 1st through 19th overtones of the fundamental. Filter **RemoveUseless** has removed **amplbin0** because of its constant value of 1.0.

Raw indicates an attribute that you normalized in assignment 1 to the reference fundamental frequency or amplitude. Attributes **centrfreq**, **roll25freq**, **roll50freq**, **roll75freq**, **nc**, **n25**, **n50**, **n75**, and **normrms** are Derived Attributes we created in assignment 1. Even though they are redundant with attributes from which they derive, they turn out to be useful for fine-tuning classifiers. We are keeping them for now. There are 34 attributes in the ARFF data of this assignment.

tnoign Target white noise signal gain passed to the audio generator in the range [0.0, 1.0]. Except for the five **tnoign**=0.0 samples that we will remove, the signal generator for this dataset generates **tnoign** in the range [0.1, 0.25). Note the Weka Preprocess statistics for **tnoign** below.

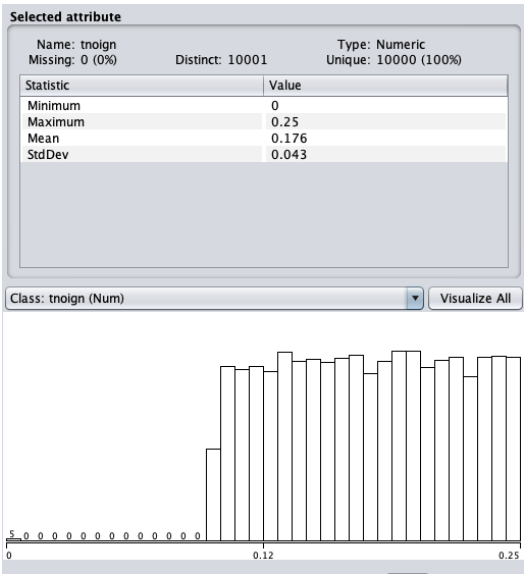

Figure 1: Class attribute tnoign in the handout dataset.

Since this assignment is about predicting white noise gain tagged as attribute **tnoign**, it is important to review the definition of white noise. As linked from Assn1AudioOverview, "White noise is a random signal having equal intensity at different frequencies, giving it a constant power spectral density…In discrete time, white noise is a discrete signal whose samples are regarded as a sequence of serially uncorrelated random variables with zero mean and finite variance.[2]" This white noise signal is distinct from the Sine, Triangle, Square, Sawtooth, and Pulse wave signals that were the focus of assignment 1, added into the composite signal with a random gain in the range [0.5, 0.75). The dataset of assignments 1 and 2 add white noise with a random gain in the range [0.1, 0.25) to each signal-record in the dataset, with 5 exceptions that you will remove in step 2 below. Compare the frequency domain plot of the noiseless 1000 Hz training sine wave of assignment 1[3] with the 1001 Hz sine wave with a **tnoign**= 0.139453694281 used as a noise-bearing training instance in assignment 1[4]. Both peak at about 1000 Hz, but the signal without white noise loses

[2] Wikipedia page on white noise https://en.wikipedia.org/wiki/White_noise , quotation checked for accuracy.
[3] http://faculty.kutztown.edu/parson/fall2023/lazy1_SinOsc_1000_0.9_0.0_0.FREQ.png
[4] http://faculty.kutztown.edu/parson/fall2023/lazy1_SinOsc_1001_0.500235007566_0.139453694281_615143.FREQ.png

most of its strength after that. The signal with **tnoign**= 0.139453694281 white noise falls off considerably less, maintaining an almost constant signal strength all the way to the Nyquist frequency of 22050 Hz. The contribution of white noise at each frequency is random and seemingly small, but the net contribution of white noise is to add signal strength evenly across the frequency spectrum. We are trying to determine that contribution based strictly on audio data in the WAV files in this assignment. Important points to note include the following.

- Most of the frequency spectrum in the range [0, 22050] Hz lies above the non-noise signal generation (sine, triangle, etc.) fundamental frequency of [100, 2000] Hz. White noise spans the [0, 22050] Hz range. While the non-sine waves contribute harmonics that push measures such as centroid and the rolloff frequencies higher than the fundamental frequency, white noise pushes these measures even further up the frequency spectrum because it spans the [0, 22050] Hz range.

- White noise contributes additional power beyond the non-noise signals across the wave + white noise signal. Attribute **rms** is the measure of power across the time-varying, time-domain signal. Unlike the normalized fundamental frequency of **amplbin0**, which represents only the strongest frequency component of a signal, **rms** integrates signal strength across the frequency spectrum.

The five **tnoign**=0.0 samples illustrated in Figure 1 are outliers in relation to the other 10,000 instances.

2. Use Weka's Unsupervised -> Instance -> RemoveWithValues Preprocess filter to remove the five outlying instances with **tnoign**=0.0. Use the attributeIndex to select tnoign, use the splitPoint to select a value for this attribute above which OR below which instances will be discarded, using invertSelection if necessary to change the direction of the split. Successful application of RemoveWithValues to tnoign results in 10,000 instances with tnoign in the range [0.1, 0.25), which is the range of white noise gain for the signal generator. **SAVE THIS 10000-INSTANCE DATASET INTO FILE csc558wn10Ksp2023NoTid0.arff**.

Q1: What is your exact RemoveWithValues command line from the top of Weka's Preprocess tab?

3. Run Classify -> Functions -> LinearRegression using 10-fold cross-validation on this 10,000-instance dataset.

Q2: Paste the following measures into README.txt Q2. We will use Correlation coefficient as the primary measure of accuracy in this assignment.

Correlation coefficient	n.n
Mean absolute error	n.n
Root mean squared error	n.n
Relative absolute error	n.n %
Root relative squared error	n.n %
Total Number of Instances	10000

Examine the Weka LinearRegression formula that starts out like this:

Linear Regression Model

tnoign =
    C.c * centroid +
    C.c * rms +
    …

-0.4307

Q3: In terms of absolute value of the coefficients C.c, what are the top six, starting with the one with the highest magnitude in descending order? Include the ones with minus signs in front of them, using their magnitude (absolute value) to determine their ranking. Negative coefficients occur in one of two ways: a) the attribute being multiplied has a negative correlation with the target numeric attribute, which is still an informative correlation, or b) the negative coefficient is an adjustment for a positive coefficient elsewhere in the formula. The second case usually occurs for nominal non-target attributes that appear more than once in the formula. Do not use the constant, non-multiplier value at the end of the formula. Here is the first line of your answer; supply the other 5.

9.1739 * rms +

Q4: Apply the unsupervised -> attribute -> Normalize preprocessing filter to all attributes using its default configuration parameters. With its default parameters Normalize adjusts each attribute except target **tnoign** to a value in the range [0.0, 1.0] using the formula **(value – min) / (max – min)** for the min and max of that attribute. Check several attributes to see the [0.0, 1.0] range and make sure tnoign has not been Normalized. Again run LinearRegression and paste the following measures into README.txt Q4:

Correlation coefficient	n.n
Mean absolute error	n.n
Root mean squared error	n.n
Relative absolute error	n.n %
Root relative squared error	n.n %
Total Number of Instances	10000

Q5: In terms of absolute value of the coefficients C.c for this Normalized LinearRegression model, what are the top six, starting with the one with the highest magnitude in descending order? Use the approach to coefficient absolute values that you used for Q3. List which attributes have been **Removed** from the top six, which have been **Added**, and which have been **Retained**.

Figure 1 shows the non-target attribute with the highest magnitude LinearRegression coefficient along the X axis at the top plot from Weka's Visualize tab, the second highest magnitude coefficient in the middle, and the third highest magnitude coefficient at the bottom. The Y axis shows target attribute **tnoign**. The centers of their slopes are rather steep. A vertical slope would be useless because all values of **tnoign** would correlate with a single value of the non-target attribute. A horizontal slope would be useless because all values of the non-target attribute would correlate with a single value of **tnoign**. Use Weka's Visualize tab to see that most other attributes have much more scattered correlations of the non-target attributes to **tnoign**.

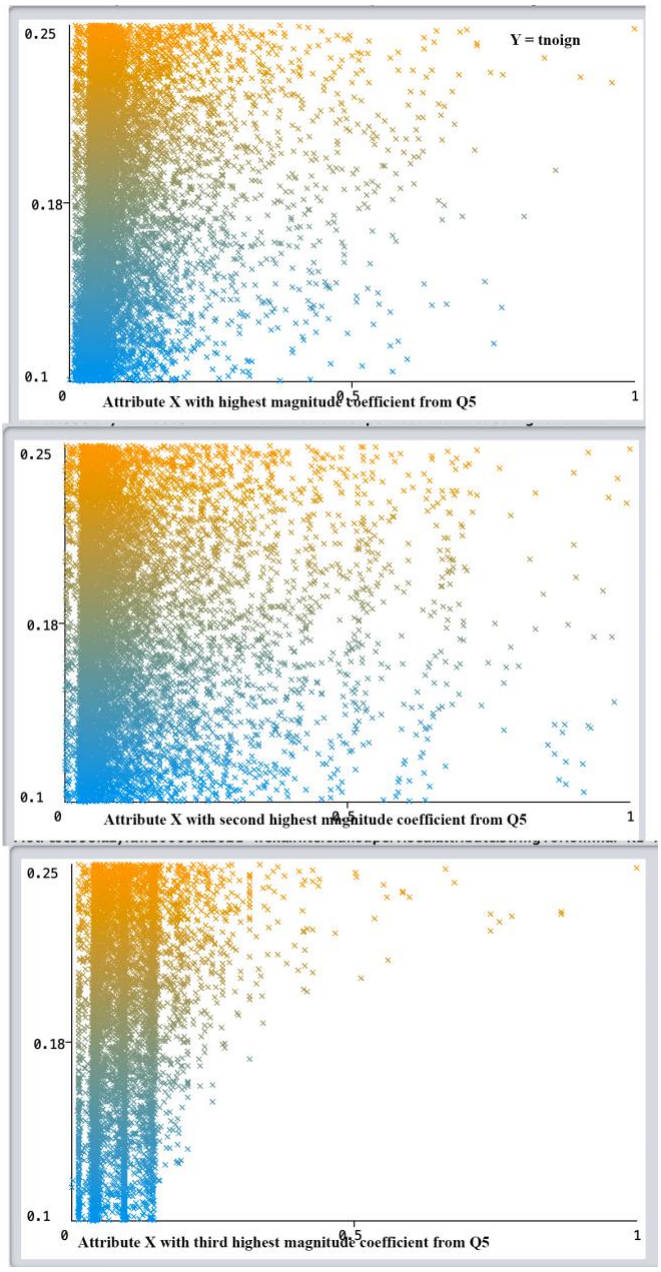

Figure 1

Q6: For some datasets Normalization to the range [0.0, 1.0] or some other fixed range addresses the problem of making some attributes appear more important in the LinearRegression formula than they are when interpreting the formula. Which attribute had the highest coefficient C.c in your answer to Q2 & Q3, and what happened to that attribute's importance in Normalized Q4 & Q5 relative to other attributes? Why was its coefficient C.c so very high in Q2 compared to Normalized Q4? (Hint: You probably need to execute Undo in the Preprocess tab to see its original range of min and max values to answer this question.)

Q7: Re-Normalize if necessary to get Normalized non-target attributes. Continue using Normalized non-target attributes unless otherwise instructed. Run Classify -> Trees -> M5P model tree on this 10,000-instance Normalized dataset, and record the Results (not the Model) for Q7. How do the M5P Results

(correlation coefficient and error measures) compare with those of LinearRegression for this Normalized dataset? Make sure to include M5P's Number of Rules measure, which is the number of leaf-linear-regression formulas in the M5P decision tree. (Note: If you executed Undo to discard Normalization in answering Q6, you will need to run the Normalize attribute filter now.)

Number of Rules : N
Correlation coefficient ?
Mean absolute error ?
Root mean squared error ?
Relative absolute error ? %
Root relative squared error ? %
Total Number of Instances 10000

Q8: Continue using the Normalized dataset. Run the instance-based (lazy) classifier IBk repeatedly with its default configuration parameters, increasing the KNN (number of nearest neighbors with **tnoign** values to be averaged together) parameter on each run until its performance begins to degrade, inspecting only correlation coefficient for its peak. If CC hits a plateau, keep going until it goes up or down. What lowest value of KNN gives the most accurate result in terms of correlation coefficient? Shows its Results.

KNN = N
Correlation coefficient n.n?
Mean absolute error n.n?
Root mean squared error n.n ?
Relative absolute error n.n? %
Root relative squared error n.n? %
Total Number of Instances 10000

Q9: Run the instance-based (lazy) classifier IBk one more time with its KNN as determined in Q8, then run it again after changing the nearest neighbor search algorithm from LinearNNSearch to KDTree with default parameters, and run it again using BallTree instead of KDTree. What change in behavior or performance do you notice compared to using the default LinearNNSearch nearest neighbor search algorithm?

In preparation for the next steps, run Preprocess filter Unsupervised -> Attribute -> Discretize on the target attribute **tnoign**, making sure to set the **ignoreClass** configuration parameter to **true**, allowing the Filter to Discretize target attribute **tnoign** (the Last attribute). Leave the useEqualFrequency parameter at false, leave bins at 10, and check **tnoign** before and after using the filter to make sure its distribution histograms look similar, and that it is not numeric after discretization. Do NOT discretize any numeric attributes other than **tnoign**. Check in the Preprocess tab to make sure no other attributes are discretized.

Q10: Now, run Preprocess filter Unsupervised -> Attribute -> Discretize on all remaining attributes with useEqualFrequency parameter at the default false and bins at 10. Inspect some of them in the Preprocess tab. Run classifiers rule **OneR**, tree **J48**, **BayesNet**, and instance (lazy) classifier **IBk** with the KNN parameter found in Q8 and nearest neighbor search algorithm of KDTree, and give their Results as outlined below, preceding each Result with the name of its classifier.

OneR
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

J48

Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

BayesNet
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

IBk with the KNN parameter of
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

Q11: Execute Preprocess -> Undo once, then check to make sure that only class **tnoign** is still Discretized. All other attributes except **tnoign** should be numeric. Now, run Preprocess filter **Supervised** -> Attribute -> Discretize on all remaining attributes (not **tnoign**). Inspect some of them in the Preprocess tab. Supervised Discretization attempts to correlate the non-target attribute bins with the target attribute ahead of classification model building. Run classifiers rule **OneR**, tree **J48**, **BayesNet**, and instance (lazy) classifier **IBk** with the KNN parameter found in Q8 and nearest neighbor search algorithm of KDTree, and give their Results as in Q10, preceding each Result with the name of its classifier. Which classifiers became BETTER as measured by Kappa when compared with Q10, and which became WORSE. Just write BETTER or WORSE or SAME behind their classifier names.

OneR
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

J48
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

BayesNet
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

IBk with the KNN parameter of
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

Q12: Execute Preprocess -> Undo once, then check to make sure that only class **tnoign** is still Discretized. All other attributes except **tnoign** should be numeric. Run classifiers rule **OneR**, tree **J48**, **BayesNet**, and instance (lazy) classifier **IBk** with the KNN parameter found in Q8 and nearest neighbor search algorithm of KDTree, and give their Results as before, preceding each Result with the name of its classifier. Which classifiers became BETTER as measured by Kappa when compared with **Q10**, and which became WORSE. Just write BETTER or WORSE or SAME behind their classifier names.

OneR
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

J48
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

BayesNet
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

IBk with the KNN parameter of
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

In general, increasing the resolution of the non-target attributes by keeping them numeric may help accuracy of prediction, since discretized non-target attributes only approximate the precision found in numeric non-target attributes. Unfortunately, precise numeric attributes may be harder for some classifiers to analyze. Bayesian analysis, for example, does its own discretization of numeric non-target attributes; this discretization may be better or worse than the Supervised Weka discretization filter at correlating non-target attributes to the target class.

Q13. All attributes numeric except **tnoign** should still be numeric and Normalized to the range [0.0, 1.0]. Try using ensemble meta-classifier **Bagging**, using your most accurate classifier (in terms of Kappa) configuration from Q12 as its base classifier. What base classifier did you select, and does it improve performance over Q12 in terms of Kappa by more than .02 of 1.0 of the non-bagged Result of Q12? Show your Result as before. All attributes except the target **tnoign** should be numeric at this point.

BASE CLASSIFIER USED:
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

Q14. Try using ensemble meta-classifier **AdaBoostM1**, using your most accurate classifier configuration form Q12 as its base classifier. What base classifier did you select, and does it improve performance over Q12 in terms of Kappa by more than .02 of 1.0 of the non-boosted Result of Q12? Show your Result as before. All attributes except the target **tnoign** should be numeric at this point.

BASE CLASSIFIER USED:
Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

Q15. Try using ensemble meta-classifier **RandomForest**, which uses RandomTree as its base classifier, running 100 RandomTrees by default. Does it improve performance over Q12 in terms of Kappa by more

than .02 of 1.0 of the non-boosted Result of Q12? Show your Result as before. All attributes except the target **tnoign** should be Normalized numeric at this point.

Correctly Classified Instances N N.N %
Incorrectly Classified Instances N N.N %
Kappa statistic N.N

Q16. What accounts for any performance improvements in terms of kappa in Q13 Q14 and Q15 over Q12 results?

For Q17 through Q20 I used the following bash shell script to create these files.

csc558wnTrain100sp2023.arff 100 initial training instances from csc558wn10Ksp2023NoTid0.arff.
csc558wnTest9900sp2023.arff 9900 remaining test instances from csc558wn10Ksp2023NoTid0.arff.
csc558wnTrain100Rndsp2023.arff 100 random-order instances from csc558wn10Ksp2023NoTid0.arff.
csc558wnTest9900Rndsp2023.arff 9900 other random instances from csc558wn10Ksp2023NoTid0.arff.

Bash script maker.sh. The non-SHUFFLED, non-Rnd instances are in original order.

```
echo "making 100 training instances in csc558wnTrain100sp2023.arff"
bash -c "echo '@relation csc558wnTrain100sp2023' >
csc558wnTrain100sp2023.arff"
bash -c "grep @ csc558wn10Ksp2023NoTid0.arff | grep -v  @relation >>
csc558wnTrain100sp2023.arff"
bash -c "grep ^[0-9] csc558wn10Ksp2023NoTid0.arff | head -100 >>
csc558wnTrain100sp2023.arff"
echo "making 9900 test instances in csc558wnTest9900sp2023.arff"
bash -c "echo '@relation csc558wnTest9900sp2023' >
csc558wnTest9900sp2023.arff"
bash -c "grep @ csc558wn10Ksp2023NoTid0.arff | grep -v  @relation >>
csc558wnTest9900sp2023.arff"
bash -c "grep ^[0-9] csc558wn10Ksp2023NoTid0.arff | tail -9900 >>
csc558wnTest9900sp2023.arff"
```

The SHUFFLED, Rnd instances are in random order

```
bash -c "grep ^[0-9] csc558wn10Ksp2023NoTid0.arff | sort --random-sort >
junk.txt"
echo "making 100 SHUFFLED training instances in
csc558wnTrain100Rndsp2023.arff"
bash -c "echo '@relation csc558wnTrain100Rndsp2023' >
csc558wnTrain100Rndsp2023.arff"
bash -c "grep @ csc558wn10Ksp2023NoTid0.arff | grep -v  @relation >>
csc558wnTrain100Rndsp2023.arff"
bash -c "head -100 < junk.txt >> csc558wnTrain100Rndsp2023.arff"
echo "making 9900 SHUFFLED test instances in csc558wnTest9900Rndsp2023.arff"
bash -c "echo '@relation csc558wnTest9900Rndsp2023' >
csc558wnTest9900Rndsp2023.arff"
bash -c "grep @ csc558wn10Ksp2023NoTid0.arff | grep -v  @relation >>
csc558wnTest9900Rndsp2023.arff"
bash -c "tail -9900 < junk.txt >> csc558wnTest9900Rndsp2023.arff
```

Q17. Load csc558wnTrain100sp2023.arff in the Preprocess tab as the training set, and set csc558wnTest9900sp2023.arff to be the supplied test set in the Classify tab. Do **NOT** Normalize or Discretize any attributes from Q17 through Q20. Run M5P and record its Results here. How many rules (linear formulas) does M5P generate?

Number of Rules : N
Correlation coefficient ?
Mean absolute error ?
Root mean squared error ?
Relative absolute error ? %
Root relative squared error ? %
Total Number of Instances 9900

Q18. Load csc558wnTrain100Rndsp2023.arff in the Preprocess tab as the training set, and set csc558wnTest9900Rndsp2023.arff to be the supplied test set in the Classify tab. Run M5P and record its Results here. How many rules (linear formulas) does M5P generate?

Number of Rules : N
Correlation coefficient ?
Mean absolute error ?
Root mean squared error ?
Relative absolute error ? %
Root relative squared error ? %
Total Number of Instances 9900

Q19. Before I removed **tosc** from your handout data, the instances were in the following order by **tosc** values. They remained in this order until my shell script randomized instance order in **`csc558wnTrain100Rndsp2023.arff`** and **`csc558wnTest9900Rndsp2023.arff`**. Note the five initial, 0-noise instances that you have deleted at the start of the current assignment in the above command output:

```
$ grep Osc csc558lazyraw10005sp2018.arff | cut -d, -f2 |uniq -c
      1 'PulseOsc'
      1 'SawOsc'
      1 'SinOsc'
      1 'SqrOsc'
      1 'TriOsc'
   2000 'PulseOsc'
   2000 'SawOsc'
   2000 'SinOsc'
   2000 'SqrOsc'
   2000 'TriOsc'
```

What accounts for the improvement in accuracy measures in going from Q17 to Q18? Note that before randomization, instances in file csc558wn10Ksp2023NoTid0.arff were in the same order as they are in the above csc558lazyraw10005sp2018.arff file.

Q20. Can you improve performance of M5P further by bagging it? Give Results showing improvement, or explain why this attempt at improvement fails. Make sure to use the randomized training and test files csc558wnTrain100Rndsp2023.arff and csc558wnTest9900Rndsp2023.arff of Q18, with M5P as the base classifier.

Correlation coefficient ?
Mean absolute error ?
Root mean squared error ?
Relative absolute error ? %

Root relative squared error	? %
Total Number of Instances	9900