CSC 458 Data Mining and Predictive Analytics I, Final Exam Mini-Project, Fall 2018, Answer Sheet

Dr. Dale E. Parson, Final Assignment 5, Comprehensive Assignment/Exam. Due by 9 AM on Thursday December 13 via make turnitin. I will not accept late solutions; I need to grade these in a timely manner. Assignments coming in any amount after 9 AM on December 13 earn 0%.

Our final exam class is scheduled for Tuesday, December 11, 6-8 PM. I will post this assignment and the necessary files by noon on that Tuesday I will answer questions only in Tuesday's class between 6-8 PM, so come prepared to ask questions. Your **make turnitin** is due by 9 AM on Thursday and no later.

Perform the following steps to set up for this project. Start out in your login directory on csit (a.k.a. acad).

cd $HOME
mkdir DataMine # This should already be there from earlier assignments.
cp ~parson/DataMine/finalexam458fall2018.problem.zip DataMine/finalexam458fall2018.problem.zip
cd ./DataMine
unzip finalexam458fall2018.problem.zip
cd ./finalexam458fall2018

This is the directory from which you must run **make turnitin** by the project deadline to avoid an exam grade of 0%. If you run out of file space in your account, you can perform the following steps from within your DataMine/ directory. **Be extremely careful, and do NOT use any file name wildcards**. This will discard your results from previous assignments. If you wish to keep those, do not remove directories **csc458fall2018assn1, csc458fall2018assn2**, **linear458fall2018, or bayes458fall2018.**

rm -rf csc458fall2018assn1.problem.zip csc458fall2018assn1
rm -rf csc458fall2018assn2.problem.zip csc458fall2018assn2
rm -rf linear458fall2018.problem.zip linear458fall2018
rm -rf bayes458fall2018.problem.zip bayes458fall2018

You will see the following files in this **finalexam458fall2018** directory:

readme.txt	Your answers to Q1 through Q15 below go here, in the required format. Each of Q1..Q15 is worth 6.66% of the exam.
Q1before.arff & Q7before.arff	The ARFF files that are the handout datasets for this exam.
makefile	Files needed to **make turnitin** to get your solution to me.
checkfiles.sh	
makelib	

How can you avoid running out of memory in Weka?

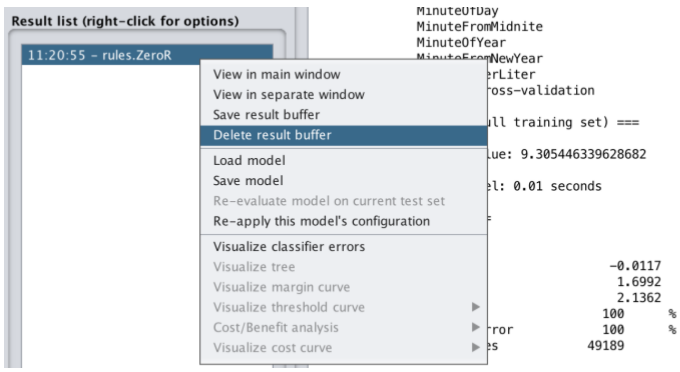
1. Run Weka using a command line or batch script that sets memory size. I run it this way on my Mac:

java -server -Xmx4000M -jar /Applications/weka-3-8-0/weka.jar

That requires having the Java runtime environment (not necessarily the Java compiler) installed on your machine (true of campus PCs), and locating the path to the weka.jar Java archive that contains the Weka class libraries and other resources. This line allocates 4,000,000 bytes of storage for Weka. As for assignment 2, I have created batch file S:\ComputerScience\WEKA\WekaWith2GBcampus.bat for campus PCs, with handout data files in S:\ComputerScience\Parson\Weka\. I plan to create a 4Gb. Byte script S:\ComputerScience\WEKA\WekaWith4GBcampus.bat after I return to campus on November 8. Try using that. It will contain this command line:

java –Xmx4096M -jar "S:\ComputerScience\WEKA\weka.jar"

2. Right-click results buffers in the Weka -> Classify window, or use Alt-click on Mac (control-click on PC) to Delete result buffer after you are done with one. They take up space. You can also save these results to text files via this menu.



3. Some of these models take a long time to execute. I have noted that condition in these instructions. In such cases, it may save time just to exit Weka and restart it via the command line or a batch file with a large memory limit, rather than just deleting result buffers.

STEPS

1. **Open file Q1before.arff** as the training and test set in Weka.
2. Reorder attributes to make **targetAttribute** the last attribute (as usual) without changing the relative order of the non-class attributes.
3. Run the **LinearRegression** model after setting its **attributeSelectionMethod** parameter to **No Attribute Selection**, and run the **M5P** model tree with its default configuration parameters on this data; use 10-fold cross correlation, and compare their formulas, tree, Correlation coefficients, and error measures. (Note: If setting **attributeSelectionMethod** parameter to **No attribute selection** is not supported on your version of Weka, just report the **attributeSelectionMethod** value in your Q1 answer.)

Q1: Which one, LinearRegression or M5P, gives the Minimum Description Length formula, considering both formula length and prediction accuracy, for this dataset? Explain your answer.

Answer: M5P. It is just as accurate as LinearRegression for this data, but with a much smaller formula:

Linear Regression Model
targetAttribute =

2 * uniform +
-0 * gaussian +
0 * noisygau +
0 * exponential +
0 * revexponential +
-0 * angle +
-0.0001 * sinwave +
-0 * coswave +
0 * logcurve +
0 * expcurve +
9.9995

Correlation coefficient	1
Mean absolute error	0.0003
Root mean squared error	0.0167
Relative absolute error	0 %
Root relative squared error	0.0003 %
Total Number of Instances	50000

M5 pruned model tree:
(using smoothed linear models)
LM1 (50000/0%)
LM num: 1
targetAttribute =
2 * uniform
+ 9.9998

Correlation coefficient	1
Mean absolute error	0.0002
Root mean squared error	0.0167
Relative absolute error	0 %
Root relative squared error	0.0003 %
Total Number of Instances	50000

4. Discretize **only** this targetAttribute into **10** nominal bins. Leave useEqualFrequency at False in order to maintain the statistical distribution of the values.

Q2: Save this file as **Q1after.arff** and turn it in using **make turnitin** from the project directory after completing all steps in this exam.

5. Run the ZeroR, OneR, J48, BayesNet, and NaiveBayes classifiers on this dataset. Compare their "Correctly Classified Instances" and all error measures.

2017:

ZeroR:

Correctly Classified Instances	5130	10.26 % (SAME FOR 2018)
Incorrectly Classified Instances	44870	89.74 %
Kappa statistic	0	
Mean absolute error	0.18	
Root mean squared error	0.3	
Relative absolute error	100 %	
Root relative squared error	100 %	

Total Number of Instances 50000

OneR:
Correctly Classified Instances 49994 99.988 %
Incorrectly Classified Instances 6 0.012 %
Kappa statistic 0.9999
Mean absolute error 0
Root mean squared error 0.0049
Relative absolute error 0.0133 %
Root relative squared error 1.633 %
Total Number of Instances 50000

J48:
Correctly Classified Instances 49991 99.982 %
Incorrectly Classified Instances 9 0.018 %
Kappa statistic 0.9998
Mean absolute error 0
Root mean squared error 0.006
Relative absolute error 0.02 %
Root relative squared error 2 %
Total Number of Instances 50000

BayesNet:
Correctly Classified Instances 49994 99.988 %
Incorrectly Classified Instances 6 0.012 %
Kappa statistic 0.9999
Mean absolute error 0.0002
Root mean squared error 0.0049
Relative absolute error 0.1243 %
Root relative squared error 1.6359 %
Total Number of Instances 50000

NaiveBayes:
Correctly Classified Instances 49562 99.124 %
Incorrectly Classified Instances 438 0.876 %
Kappa statistic 0.9903
Mean absolute error 0.0209
Root mean squared error 0.0768
Relative absolute error 11.6139 %
Root relative squared error 25.5869 %
Total Number of Instances 50000

2018 has OneR and BayesNet tied:

OneR 2018:
Correctly Classified Instances 49994 99.988 %
Incorrectly Classified Instances 6 0.012 %
Kappa statistic 0.9999
Mean absolute error 0
Root mean squared error 0.0049
Relative absolute error 0.0133 %

Root relative squared error 1.633 %
Total Number of Instances 50000

J48 2018:
Correctly Classified Instances 49991 99.982 %
Incorrectly Classified Instances 9 0.018 %
Kappa statistic 0.9998
Mean absolute error 0
Root mean squared error 0.006
Relative absolute error 0.02 %
Root relative squared error 2 %
Total Number of Instances 50000

NaiveBayes 2018:
Correctly Classified Instances 49562 99.124 %
Incorrectly Classified Instances 438 0.876 %
Kappa statistic 0.9903
Mean absolute error 0.0209
Root mean squared error 0.0768
Relative absolute error 11.6138 %
Root relative squared error 25.5868 %
Total Number of Instances 50000

BayesNet 2018:
Correctly Classified Instances 49994 99.988 %
Incorrectly Classified Instances 6 0.012 %
Kappa statistic 0.9999
Mean absolute error 0.0002
Root mean squared error 0.0049
Relative absolute error 0.1243 %
Root relative squared error 1.6359 %
Total Number of Instances 50000

Q3: Is there an unconditional winner from among the above classifiers in terms of "Correctly Classified Instances" and error measures? If so, which one, and give its "Correctly Classified Instances" and error measures. If not, give the "Correctly Classified Instances" and error measures for the contending approaches, and explain why there is no clear winner. Explain the reasoning behind your answer, showing model structure and/or "Correctly Classified Instances"/error measures as needed.

OneR is winner because some of its error measures, underlined above, are smaller than BayesNet in second place. (**2018 BayesNet ties OneR**.)

Q4: Which approach from Q2 represents the "Minimal Description Length" (MDL) model? Explain the reasoning behind your answer, showing model structure and/or "Correctly Classified Instances"/error measures as needed.

I give it to OneR.n

OneR:
uniform:
    < 1000.317242 -> '(-inf-2010.607418]'

< 2000.1581434999998 -> '(2010.607418-4010.507342]'
< 3000.0774300000003 -> '(4010.507342-6010.407265]'
< 4000.124552 -> '(6010.407265-8010.307189]'
< 5000.0880985 -> '(8010.307189-10010.207112]'
< 6000.145431999999 -> '(10010.207112-12010.107035]'
< 7000.0288255 -> '(12010.107035-14010.006959]'
< 7999.8571885 -> '(14010.006959-16009.906882]'
< 8999.2698815 -> '(16009.906882-18009.806806]'
>= 8999.2698815 -> '(18009.806806-inf)'
(50000/50000 instances correct)

Correctly Classified Instances	49994	99.988 %
Incorrectly Classified Instances	6	0.012 %
Kappa statistic	0.9999	
Mean absolute error	0	
Root mean squared error	0.0049	
Relative absolute error	0.0133 %	
Root relative squared error	1.633 %	
Total Number of Instances	50000	

BayesNet is a contender after you throw away the useless nodes in the graph with constant probabilities of 1.

targetAttribute	'(-inf-1000...	'(1000.317...	'(2000.158...	'(3000.077...	'(4000.124...	'(5000.088...	'(6000.145...	'(7000.028...	'(7999.857...	'(8999.269...
'(-inf-2010.607418]'	0.999	0	0	0	0	0	0	0	0	0
'(2010.607418-4010.507342]'	0	0.999	0	0	0	0	0	0	0	0
'(4010.507342-6010.407265]'	0	0	0.999	0	0	0	0	0	0	0
'(6010.407265-8010.307189]'	0	0	0	0.999	0	0	0	0	0	0
'(8010.307189-10010.207112]'	0	0	0	0	0.999	0	0	0	0	0
'(10010.207112-12010.107035]'	0	0	0	0	0	0.999	0	0	0	0
'(12010.107035-14010.006959]'	0	0	0	0	0	0	0.999	0	0	0
'(14010.006959-16009.906882]'	0	0	0	0	0	0	0	0.999	0	0
'(16009.906882-18009.806806]'	0	0	0	0	0	0	0	0	0.999	0
'(18009.806806-inf)'	0	0	0	0	0	0	0	0	0	0.999

J48 is more complicated than OneR with less accuracy:

```
uniform <= 4999.982539
|   uniform <= 1999.853854
|   |   uniform <= 1000.303289: '(-inf-2010.607418]' (5015.0)
|   |   uniform > 1000.303289: '(2010.607418-4010.507342]' (5054.0)
|   uniform > 1999.853854
|   |   uniform <= 4000.0147
|   |   |   uniform <= 2999.907739: '(4010.507342-6010.407265]' (4938.0)
|   |   |   uniform > 2999.907739: '(6010.407265-8010.307189]' (5061.0)
|   |   uniform > 4000.0147: '(8010.307189-10010.207112]' (4994.0)
uniform > 4999.982539
|   uniform <= 7000.002567
|   |   uniform <= 5999.877829: '(10010.207112-12010.107035]' (4969.0)
|   |   uniform > 5999.877829: '(12010.107035-14010.006959]' (5130.0)
|   uniform > 7000.002567
|   |   uniform <= 8998.280291
|   |   |   uniform <= 7999.748943: '(14010.006959-16009.906882]' (4886.0)
|   |   |   uniform > 7999.748943: '(16009.906882-18009.806806]' (4988.0)
|   |   uniform > 8998.280291: '(18009.806806-inf)' (4965.0)
```

NaiveBayes is much more complicated to read.

Q5: Based on your analysis of this ARFF file's dataset up to this point, how can you get NaiveBayes to maximize its performance in terms of perform "Correctly Classified Instances" without any degradation to BayesNet's "Correctly Classified Instances"? Describe how you achieved this result and why your change or changes to the data make this improvement in NaiveBayes. Explain the reasoning behind your answer, showing model structure and/or "Correctly Classified Instances"/error measures as needed.

Drop all attributes except uniform and targetAttribute. This gives shortest NaiveBayes description and greatest accuracy:

NaiveBayes:

Correctly Classified Instances	49832	99.664 %
Incorrectly Classified Instances	168	0.336 %
Kappa statistic	0.9963	
Mean absolute error	0.0208	
Root mean squared error	0.0762	
Relative absolute error	11.5702 %	
Root relative squared error	25.4028 %	
Total Number of Instances	50000	

with no impact on BayesNet:
BayesNet:

Correctly Classified Instances	49994	99.988 %
Incorrectly Classified Instances	6	0.012 %
Kappa statistic	0.9999	
Mean absolute error	0.0002	
Root mean squared error	0.0049	
Relative absolute error	0.1243 %	
Root relative squared error	1.6359 %	
Total Number of Instances	50000	

The removed attributes are either uncorrelated with targetAttribute, or statistically interdependent with each other. Both of those conditions violate NaiveBayes' need for statistical independence of non-class attributes. In this case some attributes exhibit both condition. Also, BayesNet's graph shows that all attributes except uniform are statistically uncorrelated with the targetAttribute.

Q6: What formula did I use to derive Q1before.arff's targetAttribute from the remaining attributes?

targetAttribute = 2 * uniform + 10
FROM M5P. NOTE PYTHON CODE:
derv1 = genDerived(lambda i, l : l[0][i] * 2.0 + 10, datarecords)
WHERE l[0] is the uniform distribution attribute.

LinearRegression is a valid answer with this formula:

```
Linear Regression Model
targetAttribute =
      2     * uniform +
      0     * noisygau +
     -0     * angle +
     -0     * sinwave +
      0     * logcurve +
```

0 * expcurve +
10

There is also a rule-structured variant of M5P called M5rules.
I don't use it much because M5P tends to be more accurate, but in some cases M5rules gives a better MDL with little or no loss in accuracy:

M5 pruned model rules
Number of Rules : 1
Rule: 1
targetAttribute =
2 * uniform
+ 10 [50000/0%]

Correlation coefficient	1
Mean absolute error	0
Root mean squared error	0
Relative absolute error	0 %
Root relative squared error	0 %
Total Number of Instances	50000

6. Open file Q7before.arff as the training and test set in Weka.
7. Run the **LinearRegression** model and the **M5P** model on this data, with 10-fold cross correlation, and compare their formulas, tree, Correlation coefficients, and error measures.

Q7: Which one, LinearRegression or M5P, gives the Minimum Description Length formula, considering both formula length and prediction accuracy, for this dataset? Explain your answer.

M5P has shorter, clearer formulas and better accuracy.

Linear Regression Model
targetAttribute =
-0.0124 * uniform +
-3.6598 * gaussian +
0.0146 * noisygau +
-3.991 * angle +
-161.509 * sinwave +
-151.2686 * coswave +
49.857 * logcurve +
32.8475 * expcurve +
16037.9959

Correlation coefficient	0.7859
Mean absolute error	3998.2323
Root mean squared error	4731.9904
Relative absolute error	53.2491 %
Root relative squared error	61.8344 %
Total Number of Instances	50000

M5 pruned model tree:
gaussian <= 4999.98 : LM1 (25060/0%)
gaussian > 4999.98 : LM2 (24940/0%)

LM num: 1
targetAttribute =
1.4969 * gaussian
+ 9.7696
LM num: 2
targetAttribute =
-1.5013 * gaussian
+ 9.8165

Number of Rules : 2

Correlation coefficient	1
Mean absolute error	2.9618
Root mean squared error	67.1281
Relative absolute error	0.0394 %
Root relative squared error	0.8772 %
Total Number of Instances	50000

8. Remove the attributes except for those that appear in the **more accurate** of LinearRegression and M5P for this dataset. Keep only the attributes appearing in the more accurate model.

Q8: What attributes remain?

targetAttribute & gaussian

Q9: Re-run LinearRegression and M5P on these attributes. Do the results differ from the full-attribute set of Q7before.arff? If so, summarize what has changed.

Slight, insignificant change in LinearRegression, none in M5P.

Linear Regression Model
targetAttribute =
-3.656 * gaussian +
16331.4506

Correlation coefficient	0.7859 **same**
Mean absolute error	3997.8913 **slightly better**
Root mean squared error	4731.6702 **slightly better**
Relative absolute error	53.2446 % **slightly better**
Root relative squared error	61.8302 % **slightly better**
Total Number of Instances	50000

M5 pruned model tree: **same**
gaussian <= 4999.98 : LM1 (25060/0%)
gaussian > 4999.98 : LM2 (24940/0%)
LM num: 1
targetAttribute =
1.4969 * gaussian
+ 9.7696
LM num: 2
targetAttribute =
-1.5013 * gaussian

+ 9.8165

Correlation coefficient	1	**same**
Mean absolute error	2.9618	**same**
Root mean squared error	67.1281	**same**
Relative absolute error	0.0394 %	**same**
Root relative squared error	0.8772 %	**same**
Total Number of Instances	50000	

9. Discretize **only** this targetAttribute into **2** nominal bins. Leave useEqualFrequency at False in order to maintain the statistical distribution of the values.

Q10: Save this file as **Q7after.arff** and turn it in using **make turnitin** from the project directory after completing all steps in this exam.

Q11: Run the OneR, J48, and RandomTree classifiers on this dataset. Copy & paste the actual rule and trees, along with the following accuracy measures in your answer. Which of the above numeric-targetAttribute classifiers (LinearRegression or M5P) do these rule & trees most closely resemble, in terms of structure? Which is most accurate, OneR, J48, or RandomTree? Explain your answer.

OneR
INSERT RULE HERE
Correctly Classified Instances N N %
Kappa statistic N
Mean absolute error N
Root mean squared error N
Relative absolute error N %
Root relative squared error N %
Total Number of Instances N

J48 pruned tree
INSERT TREE HERE
Correctly Classified Instances N N %
Kappa statistic N
Mean absolute error N
Root mean squared error N
Relative absolute error N %
Root relative squared error N %
Total Number of Instances N

RandomTree
INSERT TREE HERE
Correctly Classified Instances N N %
Kappa statistic N
Mean absolute error N
Root mean squared error N
Relative absolute error N %
Root relative squared error N %
Total Number of Instances 50000

Which of the above numeric-targetAttribute classifiers (LinearRegression or M5P) do these trees most closely resemble, in terms of structure? **M5P. M5P splits attribute gaussian's range identically to**

RandomTree. Which is most accurate, OneR, J48, or RandomTree? **OneR** & **RandomTree**. **See bold in RandomTree below for illustration of better accuracy than J48.**

OneR:
gaussian:
< 4999.9798835 -> '(-3745.172703-inf)'
>= 4999.9798835 -> '(-inf--3745.172703]'
Correctly Classified Instances 50000 100 %
Kappa statistic 1
Mean absolute error 0
Root mean squared error 0
Relative absolute error 0 %
Root relative squared error 0 %
Total Number of Instances 50000

J48 pruned tree
gaussian <= 4999.940849: '(-3745.172703-inf)' (25060.0)
gaussian > 4999.940849: '(-inf--3745.172703]' (24940.0)
Correctly Classified Instances 49999 99.998 %
Kappa statistic 1
Mean absolute error 0
Root mean squared error 0.0045
Relative absolute error 0.004 %
Root relative squared error 0.8944 %
Total Number of Instances 50000

RandomTree
gaussian < 4999.98 : '(-3745.172703-inf)' (25060/0)
gaussian >= 4999.98 : '(-inf--3745.172703]' (24940/0)
Correctly Classified Instances 50000 100 %
Kappa statistic 1
Mean absolute error 0
Root mean squared error 0
Relative absolute error 0 %
Root relative squared error 0 %
Total Number of Instances 50000

Q12: Run the NaiveBayes and BayesNet statistical classifiers on this dataset. Copy & paste the actual tables and BayesNet graph (manually type the BayesNet graph per instructions below), along with the following accuracy measures in your answer. Which is more accurate, NaiveBayes or BayesNet? Explain your answer.

BayesNet is more accurate in all measures. See below.

Naive Bayes Classifier
Class
Attribute '(-inf--3745.172703]' '(-3745.172703-inf)'
(0.5) (0.5)
===
gaussian
mean 6322.9886 3688.3679

std. dev. 987.9011 983.0047
Correctly Classified Instances 49925 99.85 %
Kappa statistic 0.997
Mean absolute error 0.1162
Root mean squared error 0.1833
Relative absolute error 23.2444 %
Root relative squared error 36.6532 %
Total Number of Instances 50000

BayesNet GRAPH
GRAPH: targetAttribute → Gaussian
targetAttribute TABLE:

Probability Distribution Table For targetAttribute

'(-inf--3745.172703]'	'(-3745.172703-inf)'
0.499	0.501

Gaussian TABLE:

Probability Distribution Table For gaussian

targetAttribute	'(-inf-4999.979884]'	'(4999.979884-inf)'
'(-inf--3745.172703]'	0	1
'(-3745.172703-inf)'	1	0

Correctly Classified Instances 50000 100 %
Kappa statistic 1
Mean absolute error 0
Root mean squared error 0
Relative absolute error 0.0044 %
Root relative squared error 0.0044 %
Total Number of Instances 50000

Naive Bayes Classifier (STUDENT – PASTE THE ACTUAL VALUES FOR NaiveBayes results)
Class
Attribute 'LOWER-NOMINAL-RANGE' '(UPPER-NOMINAL-RANGE)'
(fraction-in-range) (fraction-in-range)
===
non-target-attribute
mean N N
std. dev. N N
Correctly Classified Instances N N %
Kappa statistic N
Mean absolute error N
Root mean squared error N
Relative absolute error N %

BayesNet GRAPH – STUDENT – TYPE IN BOTH THE GRAPH STRUCTURE AND THE TABLES WITHIN THE BAYESNET GRAPH HERE AFTER INSPECTING IT IN WEKA.
Correctly Classified Instances N N %
Kappa statistic N
Mean absolute error N
Root mean squared error N
Relative absolute error N %

<u>Q13</u>: The formula to find the Kappa statistic is

Kappa = (observed accuracy - expected accuracy)/(1 - expected accuracy).

What is the **expected accuracy** for the targetAttribute as a percentage for the dataset of Q12? How did you arrive at this answer?

expected accuracy = 50.12%. 25060/50000 = .5012 for the larger of two targetAttribute bins. This is the random guess of ZeroR.

ZeroR predicts class value: '(-3745.172703-inf)'

Correctly Classified Instances	25060	50.12	%
Incorrectly Classified Instances	24940	49.88	%
Kappa statistic	0		
Mean absolute error	0.5		
Root mean squared error	0.5		
Relative absolute error	100 %		
Root relative squared error	100 %		
Total Number of Instances	50000		

Q14: Run Simple K-means clustering using 2 clusters for this dataset. Copy & paste the table below, showing the actual data:

kMeans
======
…
Final cluster centroids:

		Cluster#	
Attribute	Full Data	0	1
	(N)	(N)	(N)
non-target-attribute	N	N	N
targetAttribute	RANGE	RANGE	RANGE

===
===============

Clustered Instances

0 REMAINDER OF THIS LINE
1
0 REMAINDER OF THIS LINE

kMeans
======
…
Cluster 0: 4793.837155,'\'(-3745.172703-inf)\''
Cluster 1: 4649.303971,'\'(-3745.172703-inf)\''
Missing values globally replaced with mean/mode
Final cluster centroids:

		Cluster#	
Attribute	Full Data	0	1
	(50000.0)	(24940.0)	(25060.0)
gaussian	5002.5168	6322.989	3688.3678
targetAttribute	'(-3745.172703-inf)'	'(-inf--3745.172703]'	'(-3745.172703-inf)'

===
===============

Clustered Instances

0 24940 (50%)
1 25060 (50%)

Setup for **Q15**: Use Weka's Preprocess tab to consult the Mean and the value-distribution curve (histogram) for the non-target attribute (**NOT** targetAttribute). Note how the colors of the two-bin targetAttribute distribute across the non-target attribute curve in the lower right part of the Preprocess tab.

Use the Weka Preprocess filter **Unsupervised -> Instance -> RemoveWithValues** to remove one of the targetAttribute bins. (**NOTE**: RemoveWithValues' *attributeIndex* refers to the attribute with values-to-remove, such as *first* or *last*, just like other filters you have used; nominalIndicies is a value of 1 or 2, depending on which targetAttribute bin you want to remove; you may have to change invertSelection to true to remove the other bin; use Undo after each step to get back to the full dataset.)

After removing one of the targetAttribute bins, note the following:

Which targetAttribute bin did you remove?
What is the mean of the non-target-attribute?
What is the minimum of the non-target-attribute?
What is the maximum of the non-target-attribute?

Removed bin 1 (kept 2).
Mean 3688.368
Minimum 0.422
Maximum 4999.941

Execute UNDO, then remove the OTHER targetAttribute bin.
Which targetAttribute bin did you remove?
What is the mean of the non-target-attribute?
What is the minimum of the non-target-attribute?
What is the maximum of the non-target-attribute?

Removed bin 2 (kept 1).
Mean 6322.989
Minimum 5000.019
Maximum 9993.504

Q15: Relate these non-target-attribute mean, min, and max values back to the values that appear in J48, RandomTree, NaiveBayes, BayesNet, and Simple K-means clustering in Q11, Q12, and Q14. Where do these values show up? What is the significance of that fact?

Means show up in NaiveBayes and K-means, and central split point show up in OneR and all of the trees.

Significance is that the lower non-target-attribute values (Gaussian) show up in the upper targetAttribute range, and vice versa. For example:

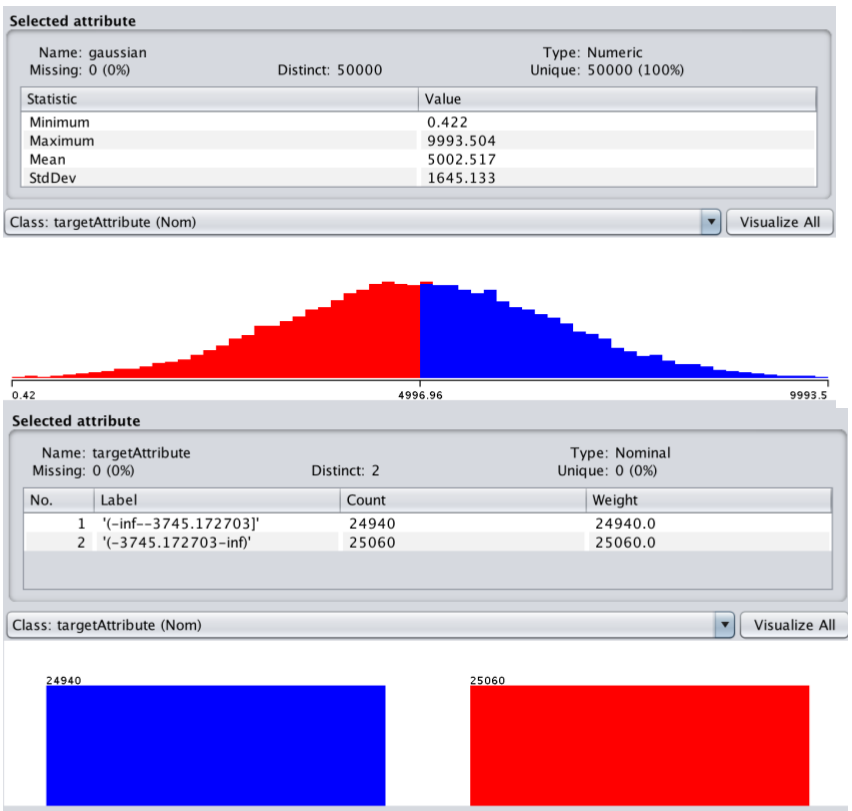
M5 pruned model tree: **same**
gaussian <= 4999.98 : LM1 (25060/0%)
gaussian > 4999.98 : LM2 (24940/0%)
LM num: 1

targetAttribute =
1.4969 * gaussian
+ 9.7696
LM num: 2
targetAttribute =
-1.5013 * gaussian
+ 9.8165

OR

RandomTree
gaussian < 4999.98 : '(-3745.172703-inf)' (25060/0)
gaussian >= 4999.98 : '(-inf--3745.172703]' (24940/0)

OR


Selected attribute

Name: gaussian | Type: Numeric
Missing: 0 (0%) | Distinct: 50000 | Unique: 50000 (100%)

Statistic	Value
Minimum	0.422
Maximum	9993.504
Mean	5002.517
StdDev	1645.133

Class: targetAttribute (Nom) | Visualize All

Selected attribute

Name: targetAttribute | Type: Nominal
Missing: 0 (0%) | Distinct: 2 | Unique: 0 (0%)

No.	Label	Count	Weight
1	'(-inf--3745.172703]'	24940	24940.0
2	'(-3745.172703-inf)'	25060	25060.0

Class: targetAttribute (Nom) | Visualize All

Make sure to run **make turnitin** in directory **finalexam458fall2018** that contains your saved files **readme.txt**, **Q1after.arff** and **Q7after.arff** as instructed above.

BONUS EXTRA credit question. Add this sequence of answers tagged as **BONUS** at the bottom of **readme.txt** if you decide to do it. It is worth 10 bonus points on the exam if it is exactly correct. I will not award any points to incorrect or partially correct solutions to this BONUS problem. It is all or none, although you cannot lose points by attempting it. Read all steps below before starting.

A. Open Q1before.arff as you did before. Do NOT save any changes that you make to the ARFF file.

B. Remove attribute **targetAttribute**.

C. Create a new derived attribute called **derivedAttribute** using the appropriate Weka filter. **derivedAttribute** will serve as your class attribute (target attribute). Attribute **derivedAttribute** must satisfy the following constraints:

C.1 It must derive from one or more attributes in this dataset.

C.2 It must correlate exactly linearly with one attribute in this dataset that does not appear in C.1. In other words, you cannot derive it in part or entirely from attribute A and then assert that it correlates linearly with that same attribute A.

C.3 By correlating exactly in step C.2 I mean that this derivation must give the highest correlation coefficient and the lowest error measures possible for a linear classifier.

D. Type into **readme.txt** the Weka formula that appears in the filter line panel after you Apply it.

E. Repeat step D, using the Weka attribute Name in place of its Position number for each original attribute used in the derivation. I need to be able to tell the attribute or attributes from which **derivedAttribute** derives.

F. Find the most accurate classifier that also exhibits the minimum description length (MDL) in predicting **derivedAttribute**. Remove any attribute that increases the description length without increasing accuracy, but be careful. Do NOT remove **derivedAttribute** or the attribute with which it correlates exactly linearly per step C.2 above. Also, the derivation formula of steps C through E must give the highest correlation coefficient and the lowest error measures possible for this dataset.

G. Copy and paste the classifier's rule, rules, formula, formulas, tree, or other structure that establishes its standing as the MDL classifier, along with the following measure of accuracy.

```
Correlation coefficient            N
Mean absolute error                N
Root mean squared error            N
Relative absolute error            N %
Root relative squared error        N %
Total Number of Instances          50000
```

D: AddExpression –E sin(a6 / 360.0 * 6.28318530717959)" –N derivedAttribute

NOTE: 6.28318530717959 is 2.0 * PI. "/ 360.0 * 6.28318530717959" converts degrees to radians. "6.28318530717959" came from multiplying 2 X PI on a calculator.

E: AddExpression –E sin(angle / 360.0 * 6.28318530717959)" –N derivedAttribute

```
Attributes:   2
              sinwave
              derivedAttribute
Linear Regression Model
derivedAttribute =
      1      * sinwave +
      0
Correlation coefficient            1
Mean absolute error                0
Root mean squared error            0
Relative absolute error            0      %
Root relative squared error        0      %
Total Number of Instances          50000
OR
M5 pruned model tree:
```

LM num: 1
derivedAttribute =
    1 * sinwave
    + 0

Correlation coefficient	1
Mean absolute error	0
Root mean squared error	0
Relative absolute error	0 %
Root relative squared error	0 %
Total Number of Instances	50000

OR (next page)

D: AddExpression –E cos(a6 / 360.0 * 6.28318530717959)" –N derivedAttribute
E: AddExpression –E cos(angle / 360.0 * 6.28318530717959)" –N derivedAttribute
Attributes: 2

coswave
derivedAttribute

Linear Regression Model: derivedAttribute = 1 * coswave + 0
M5 pruned model tree: derivedAttribute = 1 * coswave + 0
Simple Linear regression on coswave: 1 * coswave + 0

Correlation coefficient	1
Mean absolute error	0
Root mean squared error	0
Relative absolute error	0 %
Root relative squared error	0 %
Total Number of Instances	50000