CSC 558 Data Mining and Predictive Analytics II, Fall 2021

Dr. Dale E. Parson, Assignment 1, Classification of audio data samples for waveform class using decision trees and Bayesian techniques with large training datasets (10-fold cross-validation), adding to these approaches three instance-based (lazy) approaches with small training datasets.[1]

DUE By 11:59 PM on Thursday September 30 via D2L Assessment -> Assignment 1 in our D2L course page. Details are at the end of this handout. The standard 10% per day deduction for late assignments applies.

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

https://faculty.kutztown.edu/parson/fall2021/lazy558fa2021.problem.zip
OR
https://acad.kutztown.edu/~parson/lazy558fa2021.problem.zip

You will see the following files in this unzipped **lazy558fa2021** directory:

README.txt Your answers to Q1 through Q15 below go here, in the required format.
csc558lazyraw10005fa2021.arff The handout ARFF file for assignment 1.

You can download Weka 3.8.5 at https://waikato.github.io/weka-wiki/downloading_weka/ . Do not use version 3.9.

How can you avoid running out of memory in Weka?

I will test out projects using the default Weka memory allocation to try to ensure that they work without expanding memory. However, I am including memory expansion instructions just in case we need them.

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 this semester's assignments, I have created batch file WekaWith4GBcampus.bat under the shared campus PC networked location S:\ComputerScience\WEKA\. There is also WekaWith2GBcampus.bat.

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. 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. I can give batch execution instructions if needed

[1] See http://faculty.kutztown.edu/parson/fall2021/CSC558Audio1_2021.html and in-class discussion on the Zoom archive from September 13, 2021.

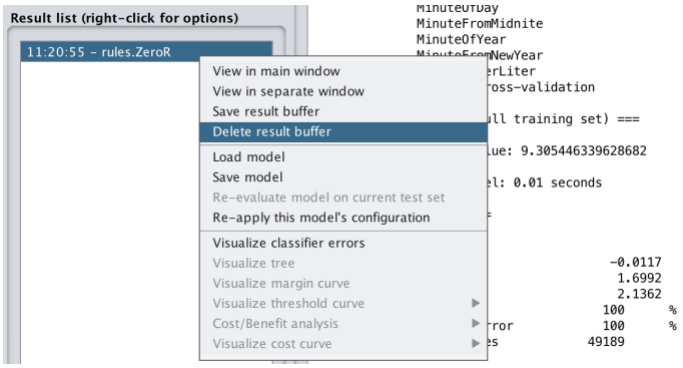

Figure 2: Deleting a Weka result buffer

http://faculty.kutztown.edu/parson/fall2021/CSC558Audio1_2021.html outlines the application dataset. We will go over this when preparing for the assignment. It explains the meaning of the attributes and their relationships.

ALL OF YOUR ANSWERS FOR Q1 through Q12 BELOW MUST GO INTO THE README.txt file supplied as part of assignment handout directory **lazy558fa2021**. You will lose an automatic 20% of the assignment if you do not adhere to this requirement. **Q13 through Q15** are the correct ARFF file contents that you must save following instructions below. Each of Q1 through Q15 is worth 6.6% of the project, and any glaring bug in ARFF file contents or your procedure can count up to 10%.

1. Open **csc558lazyraw10005fa2021.arff** in Weka's Preprocess tab.

Here are the attributes in csc558lazyraw10005fa2021.arff. Other than the 5 zero-noise training instances, I have generated new data with a similar distribution to 2020's data for 2021's assignment 1.

tid	Unique ID for each instance except that the 5 noiseless reference samples have ID 0.
tosc	Waveform type. This string must become the nominal class (target) attribute.
tfreq	Fundamental frequency in Hertz (cycles per second) passed to the audio generator.
toscgn	Waveform signal gain passed to the audio generator in the range [0.0, 1.0].
tnoign	White noise signal gain passed to the audio generator in the range [0.0, 1.0].
centroid	**Raw** spectral centroid extracted from the audio .wav file.[2]
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.
roll75	**Raw** frequency where 75% of the energy rolls off, extracted from the audio .wav file.
smprate	Rate at which the computer sampled audio, extracted from the audio .wav file.
fftbins	Number of raw bins used in frequency analysis, extracted from the audio .wav file.
hrmbins	Number of cooked bins used in Parson's data reduction, extracted from fftbins data.
shftfftfund	Number of fftbins used to normalize fundamental frequency, extracted from fftbins.
amplscale	Multiplier used to scale fundamental frequency to normalized 1.0, extracted from fftbins.
amplbin0	Normalized amplitude of fundamental frequency as extracted from the audio signal data.

amplbin1 through amplbin19 Normalized amplitudes of 1st through 19th overtones of the fundamental.

[2] See http://faculty.kutztown.edu/parson/fall2021/CSC558Audio1_2021.html for signal processing term definitions.

Raw indicates an attribute that you must normalize to the reference fundamental frequency or amplitude.

The first 5 attributes with names starting in "t" do not come from the audio signal. They were parameters to the audio generator. We are interested in predicting **tosc** (waveform oscillator type) from several of the non-"t" attributes. We must remove **tfreq**, **toscgn**, and **tnoign** before analyzing data relationships. We must get rid of **tid** after we use it to select the small training sets. We must convert **tosc** from a string to a nominal attribute and make it the final attribute in the list; **tosc** is what we are trying to predict. We will also get rid of some of the other attributes that impede analysis, as explained in class.

2. Use Weka's **unsupervised -> attribute -> StringToNominal** attribute filter to make **tosc** into a nominal attribute. Inspect its value set.
3. As directed in http://faculty.kutztown.edu/parson/fall2021/CSC558Audio1_2021.html , use Weka's **AddExpression** attribute filter to create derived attributes **nc**, **n25**, **n50**, **n75** that are **centroid** and the **rolloff** frequencies normalized in terms of the fundamental frequency. We will discuss this normalization. You will need to create some temporary "helper attributes" such as **nyfreq** and **funfreq**. Create derived attributes in the order from step 1a through step 2 on that web page. The **nc**, **n25**, **n50**, **n75** attributes correlate to the frequency-to-signal-strength distribution of the **tosc** waveform, regardless of the actual fundamental frequency **funfreq**, so they must be normalized via division by that frequency. Note that **funfreq** shows some **funfreq** fundamental frequencies outside the [100, 2000] Hz range of **tfreq**, caused by overtones and noise in some of the instances.

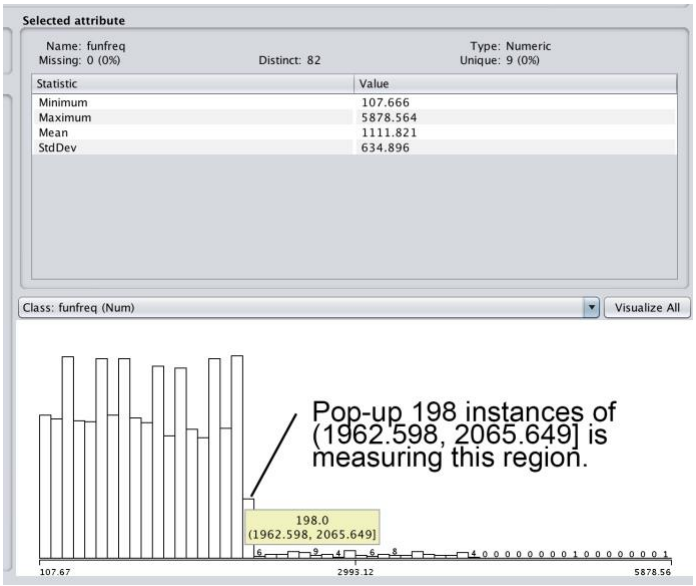

Figure3 : funfreq goes outside of the tfreq range. Label is for spring 2020 dataset. Our 2021 dataset differs in details but still has out-of-band funfreq values.

Q1. Go into Weka's Preprocess Edit window and sort on the **funfreq** attribute in descending order by holding down the SHIFT key and clicking on the **funfreq** heading. Look at the **tosc** classification for waveforms with a **funfreq** > 2040 Hz. Do the instances with **funfreq** > 2040 Hz correlate with a single category of **tosc**, and if so, what is the **tosc** value for these instances? If not, what are the **tosc** values for these instances?

Q2. Is it a good idea to keep these instances with **funfreq** > 2040 Hz in the dataset for classifying tosc, answer YES or NO (not both). Explain why.

Note that by inspecting the right side of the Weka Preprocess tab for derived attributes **centrfreq**, **roll25freq**, **roll50freq**, and **roll75freq**, they share the same distribution as their raw counterparts **centroid**, **roll25**, **roll50**, and **roll75**, because the **nyfreq** multiplier is a constant. In contrast, the normalized attributes **nc**, **n25**, **n50**, and **n75** have per-instance distributions because the **funfreq** divisor varies across instances.

4. As directed in http://faculty.kutztown.edu/parson/fall2021/CSC558Audio1_2021.html , create derived attribute **normrms** that normalizes the **rms** signal level. The **rms** is the square root of the average of the squares of signal amplitude across time for a waveform. The **amplscale** gives Parson's pre-ARFF scaling of the peak signal harmonic (the fundamental frequency) in script una2csv.py[3], while **rms** gives the raw average signal strength, which correlates to the **tosc** waveform; **normrms** scales the **rms** similarly to peak signal scaling done by Parson's preprocessing. Note that the graphical distribution of **normrms** as seen in Weka differs from that of **rms** because **amplscale** varies by instance.
5. Remove **tfreq**, **toscgn**, and **tnoign**. The fundamental frequency in the range [100, 2000] Hz does not determine the **tosc** waveform type. Also, **tfreq** does not come from the audio file; it was input to the generator, as were **toscgn** and **tnoign**. Derived attributes **funfreq** and **normrms** approximate the fundamental frequency and the signal strength from other attributes extracted from the waveform.
6. Use Weka's **Reorder** attribute filter to place **tosc** in the last position, after **normrms**, without changing the relative order of any of the other attributes.
7. Use Weka's **RemoveUseless** attribute filter to get rid of constant-valued attributes, some of which have been used temporarily in steps 3 and 4. Take notes on which attributes are removed.

Q3. Which attributes does **RemoveUseless** remove, including any derived attributes removed? Why?

Q4. Why did we keep some of these attributes until this point? Name the "useless" attribute(s) that we needed to keep to this point.

8. Save this dataset as csc558lazy10005fa2021.arff, without "raw" in its name. You will turn this file in to me via D2L at the end of the project. Reload this file using Weka's **Open file** button to get around the class identifying bug in the current Weka; class attribute **tosc** must be the final attribute.
9. Remove the **tid** attribute, because it pairs directly with **tosc** for all **tid** values except 0; the 5 noiseless reference instances all use **tid** == 0. Tid is not part of the audio data, and it "gives away" the result. We will later need to restore it temporarily from the file saved in step 8 or by executing **Undo** in the Preprocessor, in order to build some training dataset files.
10. Go to the Weka Classify tab and save the following result line only after running the following Weka classifiers on this dataset with 10-fold cross-correlation: ZeroR, OneR, J48, RandomTree, NaiveBayes, BayesNet, and IBk, keeping the default configuration parameters. IBk (a Weka **lazy** classifier) is the newcomer since csc458. It is related to K-nearest neighbor[4]. There is a paper relating to nearest-neighbor classification on our course page near assignment 1. There are two other instance-based (lazy) classifiers that run too slowly with this large training dataset that we

[3] http://faculty.kutztown.edu/parson/fall2021/una2csv.py.txt

[4] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

will use later. Also, varying the search algorithm and distance weighting parameters for IBk have no effect on its result, although changing the search algorithm may speed IBK somewhat.

Q5: Copy and paste all of these following results, this line only, with the classifier name in front:

ZeroR: Correctly Classified Instances N N.N %
Kappa statistic N

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

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

RandomTree: Correctly Classified Instances N N.N %
Kappa statistic N

NaiveBayes: Correctly Classified Instances N N.N %
Kappa statistic N

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

IBk: Correctly Classified Instances N N.N %
Kappa statistic N

11. Restore **tid**, either by executing **Undo** or by loading csc558lazy10005fa2021.arff that you saved.

NOTE: I experimented with removing all remaining attributes (raw or derived) that were used to derive the normalized attributes of steps 3 and 4. These attributes are redundant with normalized attributes **nc**, **n25**, **n50**, **n75** and **normrms**, which you are keeping. The ones I removed were centroid, rms, roll25, roll50, roll75, shftfftfund, amplscale, funfreq, centrfreq, roll25freq, roll50freq, and roll75freq. Results for re-running Q5 stayed almost exactly the same all classifiers. So, I am not having you remove these attributes.

Q6. Give one reason why removing redundant non-target attributes might have improved results for at least one machine learning algorithm tested in Q5.

Q7. Why does ZeroR have the result that it has? Relate this result to one of the terms in the Kappa statistic as explained on this page[5].

12. Next you must make two training sets with 5 elements each. Copy your csc558lazy10005fa2021.arff and makeTrainingData.sh files into a **lazy558fa2021/** directory on acad[6] and run **bash -x makeTrainingData.sh**, which performs the following steps. Or you can do this in a text editor in steps listed after these makeTrainingData.sh automated steps.

[5] http://faculty.kutztown.edu/parson/fall2019/Fall2019Kappa.html.
[6] You can run makeTrainingData.sh on a Mac or home Linux machine this way.

echo "making 5 noiseless training instances in **csc558lazytrain5fa2021.arff**"
making 5 noiseless training instances in csc558lazytrain5fa2021.arff
bash -c "echo '@relation csc558lazytrain5fa2021' > csc558lazytrain5fa2021.arff"
bash -c "grep @ csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazytrain5fa2021.arff"
bash -c "grep ^0, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazytrain5fa2021.arff"
echo "making 5 noisey training instances in **csc558lazynoise5fa2021.arff**"
making 5 noisey training instances in csc558lazynoise5fa2021.arff
echo "Sin, Tri, Sqr, Saw, Pulse:"
Sin, Tri, Sqr, Saw, Pulse:
bash -c "echo '@relation csc558lazynoise5fa2021' > csc558lazynoise5fa2021.arff"
bash -c "grep @ csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
bash -c "grep ^981018, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
 # first PulseOsc
bash -c "grep ^738502, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
 # first SawOsc
bash -c "grep ^526474, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
 # first SinOsc
bash -c "grep ^126978, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
 # first SqrOsc
bash -c "grep ^997716, csc558lazy10005fa2021.arff | grep -v @relation >> csc558lazynoise5fa2021.arff"
 # first TriOsc

MANUAL ALTERNATIVE TO RUNNING makeTrainingData.sh

A. Copy csc558lazy10005fa2021.arff to csc558lazytrain5fa2021.arff.
B. Change the @relation name in the first line of csc558lazytrain5fa2021.arff to csc558lazytrain5fa2021.
C. Keep all lines up through @data along with the first five subsequent lines that begin with the two characters “0,”, deleted all lines below those. These are the training instances for the five waveform types with 0 white noise levels.
D. Save that file as csc558lazytrain5fa2021.arff.
E. Copy csc558lazy10005fa2021.arff to csc558lazynoise5fa2021.arff.
F. Change the @relation name in the first line of csc558lazytrain5fa2021.arff to csc558lazynoise5fa2021.
G. Keep all lines up through @data along with the five lines that begin with the following character sequences, deleting all other lines below @data. These are the training instances for the five waveform types with non-zero white noise levels.
   981018,
   738502,
   526474,
   126978,
   997716,
H. Save that file as csc558lazynoise5fa2021.arff.

Running **makeTrainingData.sh** or performing these manual steps gives **csc558lazytrain5fa2021.arff** a @relation line of **csc558lazytrain5fa2021**, and **csc558lazynoise5fa2021.arff** a @relation name of **csc558lazynoise5fa2021**. Both files get all of the @attribute declarations of **csc558lazy10005fa2021.arff**, along with the ARFF @data line. Training file **csc558lazytrain5fa2021.arff** gets the five 0-noise instances with **tid** == 0, and **csc558lazynoise5fa2021.arff** gets five noise-bearing instances with **tid** values of 981018 (SinOsc),

738502 (TriOsc), 526474 (SqrOsc), 126978 (SawOsc), and 997716 (PulseOsc) with the tid attribute intact. **Verify** in Weka that each has one of each **tosc** type with the specified tids and exactly 5 instances.

13. In Weka load training set **csc558lazytrain5fa2021.arff** and Remove the **tid** attribute in memory. Leave it in the file. In the Classify tab of Weka set the Supplied test set to **csc558lazy10005fa2021.arff** instead of using cross-validation on the small training set. Figure 1 below shows how to set up a supplied test dataset. This test is similar to the sonic survey and machine listener research projects previously discussed, in that there is a small training set (a.k.a. reference set) of 5 instances and a large test of 10,005 instances against which to test it. The 5 redundant training instances in the test dataset are not a significant number of instances for testing.

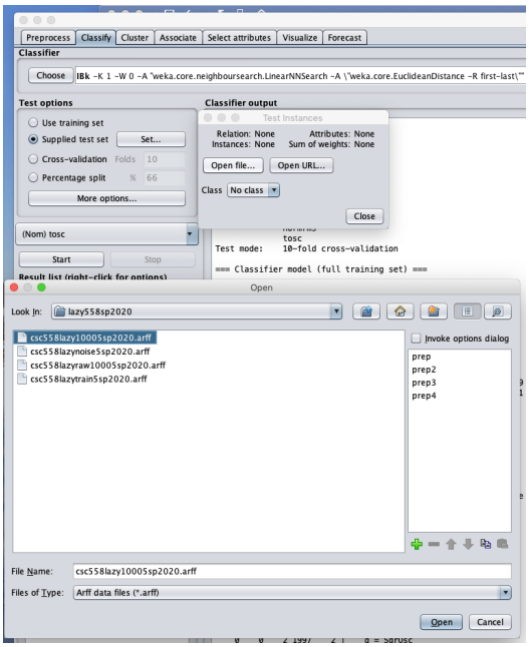

Figure 1: Using a Supplied test dataset in Weka's Classify tab

Q8. Repeat the tests of Q5 with the tid-deleted **csc558lazytrain5fa2021.arff**, adding lazy classifiers KStar and LWL into the set below. Give their Correct instances as before. **Note that Weka may ask you to accept attribute-to-attribute mappings from the training set to the test set. The attributes have the same names and positions in the ARFF files, so this should run OK**. You will see InputMappedClassifier messages. Make sure that you have removed tid from the in-memory training set. You can leave it in the test set file, since it is not mapped from the training set.

ZeroR: Correctly Classified Instances N N.N %
Kappa statistic N

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

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

RandomTree: Correctly Classified Instances N N.N %

Kappa statistic N

NaiveBayes: Correctly Classified Instances N N.N %
Kappa statistic N

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

IBk: Correctly Classified Instances N N.N %
Kappa statistic N

KStar: Correctly Classified Instances N N.N %
Kappa statistic N

LWL: Correctly Classified Instances N N.N %
Kappa statistic N

Q9. Account for the top classifier for Q8. Why is its performance substantially better than most of the remaining classifiers and at least marginally better than all the others?

14. In Weka load training set **csc558lazynoise5fa2021.arff** and delete the **tid** attribute in memory. Leave it in the file. In the Classify tab of Weka keep the Supplied test set at **csc558lazy10005fa2021.arff** instead of using cross-validation on the small training set. This is a repeat of the previous test run using a training set that has some noise in the signals.

Q10. Repeat the tests of Q8 with the tid-deleted **csc558lazynoise5fa2021.arff**, adding lazy classifiers KStar and LWL into the set below. Give their Correct instances as before. Note that Weka may ask you to accept attribute-to-attribute mappings from the training set to the test set. The attributes have the same names and positions in the ARFF files, so this should run OK. You will see InputMappedClassifier messages. Make sure that you have removed tid from the in-memory training set. You can leave it in the test set file, since it is not mapped from the training set.

ZeroR: Correctly Classified Instances N N.N %
Kappa statistic N

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

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

RandomTree: Correctly Classified Instances N N.N %
Kappa statistic N

NaiveBayes: Correctly Classified Instances N N.N %
Kappa statistic N

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

IBk: Correctly Classified Instances N N.N %

Kappa statistic N

KStar: Correctly Classified Instances N N.N %
Kappa statistic N

LWL: Correctly Classified Instances N N.N %
Kappa statistic N

Q11. Account for performance improvements in the top 3 classifiers of Q8&Q9 in going to Q10. What accounts for the improvements?

Q12. Why does IBk perform significantly better than KStar for Q8 through Q11 for this signal dataset?

Q13 points are for a correctly saved **csc558lazy10005fa2021.arff** in the project directory.

Q14 points are for a correctly saved **csc558lazytrain5fa2021.arff** in the project directory.

Q15 points are for a correctly saved **csc558lazynoise5fa2021.arff** in the project directory.

After making certain that the completed README.txt file and the files required in Q13, Q14, and Q15 are in the project directory, go to **D2L -> Assessments -> Assignments -> Assignment 1 instance-based (lazy) classification** and upload the three ARFF files of **Q13, Q14, and Q15 along with README.txt.**