CSC 558 Data Mining and Predictive Analytics II, Spring 2020 Parson's solution

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 February 22 via make turnitin on acad. The standard 10% per day deduction for late assignments applies.

If you are not accustomed to using the Linux acad system, see me during office hours, or an in-class lab session, or consult a graduate assistant in Old Main 257. I will not accept student work via D2L for this assignment. You can do all of your work on your own machine or on the campus PCs, obtaining the starting files via S:\ComputerScience\Parson\Weka. You can also log into acad and perform the following steps to retrieve the same files. You can use the FileZilla client utility or a similar file transfer program to copy files from acad and to place your solution files back onto acad.[2] Here is what the FileZilla setup for acad looks like. Just click the "Open the Site Manager" icon at the upper left of the FileZilla window to access this panel.

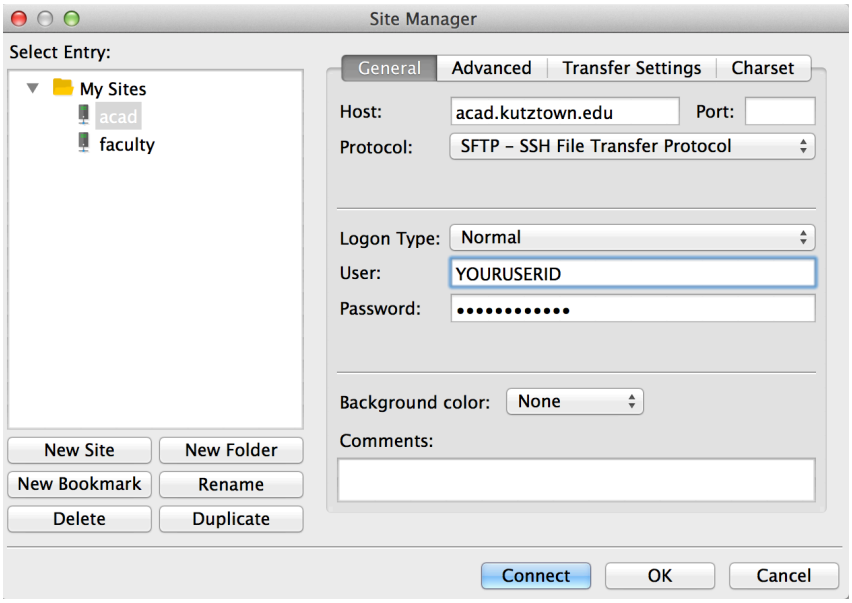

Figure 1: Filezilla

There will be one in-class work session for this assignment. You may attend in person or on-line. I encourage attending the work session in person. Come prepared to ask questions.

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 may already be there from last semester.
cp  ~parson/DataMine/lazy558sp2020.problem.zip  DataMine/lazy558sp2020.problem.zip
```

[1] See http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.html and in-class discussion on the Zoom archive from February 4.

[2] Download the FileZilla **client** at https://sourceforge.net/projects/filezilla/ .

cd ./DataMine
unzip lazy558sp2020.problem.zip
cd ./lazy558sp2020

This is the directory from which you must run **make turnitin** by the project deadline to avoid a 10% per day late penalty. If you run out of file space in your account and you took csc458, you can perform **rm -rf FILEORDIRECTORY** from within your DataMine/ directory, where **FILEORDIRECTORY** is any of the handout zip files or project directories from csc458. **Be extremely careful, and do NOT use any file name wildcards**. This will discard your results from csc458 assignments. Do not remove turned-in directories that you wish to keep.

You will see the following files in this **lazy558sp2020** directory:

README.txt	Your answers to Q1 through Q20 below go here, in the required format.
csc558lazyraw10005sp2020.arff	The handout ARFF file for assignment 1.
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 S:\ComputerScience\WEKA\WekaWith4GBcampus.bat for campus PCs, with handout data files in S:\ComputerScience\Parson\Weka\. Try using that. It contains 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. 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

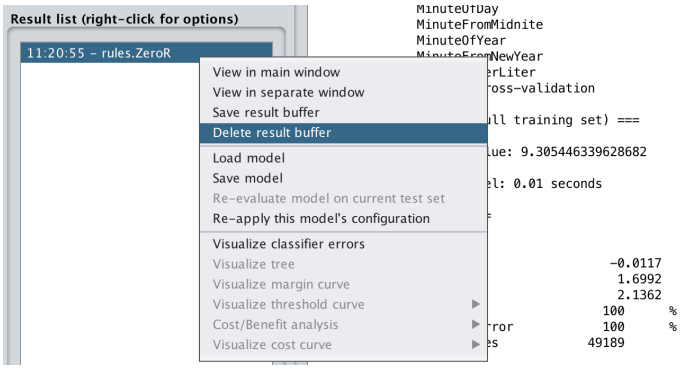

Figure 2: Deleting a Weka result buffer

http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.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 **lazy558sp2020**. 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 **csc558lazyraw10005sp2020.arff** in Weka's Preprocess tab.

Here are the attributes in csc558lazyraw10005sp2020.arff. Other than the 5 zero-noise training instances, I have generated new data with a similar distribution to 2018's data for 2020'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.[3]
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.

[3] See http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.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/spring2020/CSC558Audio1_2020.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.

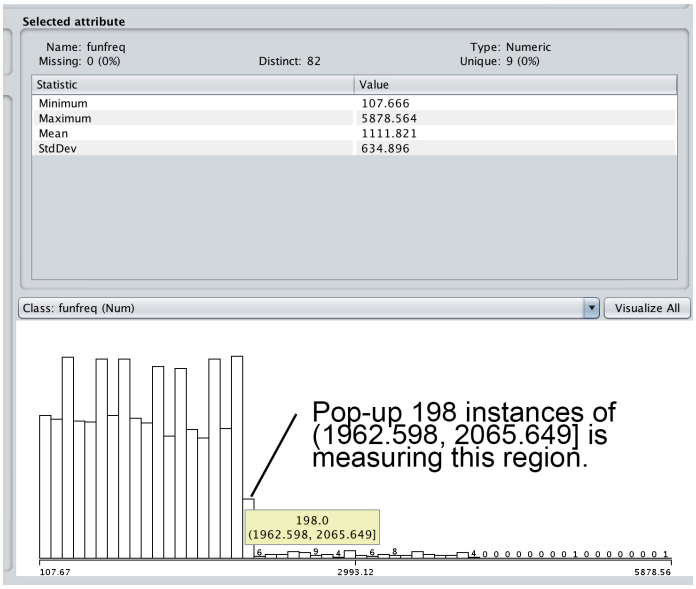

Figure3 : funfreq goes outside of the tfreq range.

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** > 2005 Hz. Do the instances with **funfreq** > 2005 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?

They all correlate with PulseOsc.

Q2. Is it a good idea to keep these instances with **funfreq** > 2005 Hz in the dataset for classifying tosc, answer YES or NO (not both). Explain why.

YES, keep them, because a high **funfreq** and attributes derived from it correlate unambiguously with **PulseOsc**.

NOTE I removed them temporarily using unsupervised -> instance -> RemoveWithValues –S 2001.0 –C 28 –V, thereby losing 10005 – 9740 (remaining) = 265 discarded instance. The modified results for Q5 below follow.

ZeroR:	Correctly Classified Instances	1964	20.5504 %
OneR:	Correctly Classified Instances	7587	79.3868 %
J48:	Correctly Classified Instances	9475	99.142 %
RandomTree:	Correctly Classified Instances	9381	98.1584 %
NaiveBayes:	Correctly Classified Instances	9248	96.7668 %
BayesNet:	Correctly Classified Instances	9066	94.8624 %
IBk:	Correctly Classified Instances	9355	97.8864 %

Those results are about the same as for Q5, slightly worse for IBk, and we have lost classification of 265 / 10005 = 2.65% of the instances, or 13.24% of the 2001 PulseOsc instances. There is no need to remove these instances.

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/spring2020/CSC558Audio1_2020.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[4], 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?

[4] http://faculty.kutztown.edu/parson/spring2020/una2csv.py.txt

@attribute smprate numeric	is the constant sampling rate of 44,100.
@attribute fftbins numeric	is the constant from waveform analysis of 512 FFT bins.
@attribute hrmbins numeric	is the constant from una2csv.py preprocessing of 20 bins.
@attribute amplbin0 numeric	is the normalized constant of 1.0.
@attribute nyfreq numeric	is the constant smprate / 2.0 = 22,050.

Removed because they are constants, and so do not contribute to per-instance classification.

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.

We needed **smprate** for **nyfreq**, and we needed **nyfreq** to compute **nc**, **n25**, **n50**, and **n75** for normalization. Other normalization steps depend on per-instance, varying attribute values.

8. Save this dataset as csc558lazy10005sp2020.arff, without "raw" in its name. You will turn this file in to me in your **lazy558sp2020/** project directory 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.
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. It 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[5]. 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 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 %
OneR:	Correctly Classified Instances	N	N.N %
J48:	Correctly Classified Instances	N	N.N %
RandomTree:	Correctly Classified Instances	N	N.N %
NaiveBayes:	Correctly Classified Instances	N	N.N %
BayesNet:	Correctly Classified Instances	N	N.N %
IBk:	Correctly Classified Instances	N	N.N %

2020 results:

ZeroR:	Correctly Classified Instances	2000	19.99 %
OneR:	Correctly Classified Instances	7876	78.7206 %
J48:	Correctly Classified Instances	9916	99.1104 %
RandomTree:	Correctly Classified Instances	9824	98.1909 %
NaiveBayes:	Correctly Classified Instances	9673	96.6817 %
BayesNet:	Correctly Classified Instances	9469	94.6427 %
IBk:	Correctly Classified Instances	9832	98.2709 %

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

2018 results:

ZeroR:	Correctly Classified Instances	2000	19.99 %
OneR:	Correctly Classified Instances	8013	80.09 %
J48:	Correctly Classified Instances	9918	99.1304 %
RandomTree:	Correctly Classified Instances	9845	98.4008 %
NaiveBayes:	Correctly Classified Instances	9687	96.8216 %
BayesNet:	Correctly Classified Instances	9507	95.0225 %
IBk:	Correctly Classified Instances	9810	98.051 %

11. Restore **tid**, either by executing **Undo** or by loading csc558lazy10005sp2020.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 became marginally worse for all classifiers except for ZeroR, which had an insignificant seeming improvement; essentially, it stayed the same. So, I am not having you remove these attributes.

Repeat step Q5 after experimental removal of centroid, rms, roll25, roll50, roll75, shftfftfund, amplscale, funfreq, centrfreq, roll25freq, roll50freq, and roll75freq.

ZeroR:	Correctly Classified Instances	1964	20.5504 %
OneR:	Correctly Classified Instances	7587	79.3868 %
J48:	Correctly Classified Instances	9425	98.6188 %
RandomTree:	Correctly Classified Instances	9299	97.3004 %
NaiveBayes:	Correctly Classified Instances	9221	96.4843 %
BayesNet:	Correctly Classified Instances	9054	94.7368 %
IBk:	Correctly Classified Instances	9245	96.7354 %

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

NaiveBayes typically performs better when the non-class attributes are independent, but in this case it did not. BayesNet might do so as well, although it tries to compensate for interdependent attributes. In this case the derived attributes had straightforward linear relations that did not degrade these Bayesian algorithms. Removing the attributes made NaiveBayes and BayesNet worse.

Q7. Why does ZeroR have the result that it has? Relate this result to one of the terms in the Kappa statistic as explained for csc458 assignment 4[6].

ZeroR just guesses one out of five possible classes, which are roughly equal in size, hence 20%. This is the **expected accuracy** of Kappa = (observed accuracy – **expected accuracy**)/(1 – **expected accuracy**)

[6] http://faculty.kutztown.edu/parson/fall2019/Fall2019Kappa.html.

12. Next you must make two training sets with 5 elements each. The easiest way is to copy your csc558lazy10005sp2020.arff file into your project's **lazy558sp2020/** directory on acad and run **make train**, which performs the following steps.[7]

```
echo "making 5 noiseless training instances in csc558lazytrain5sp2020.arff"
making 5 noiseless training instances in csc558lazytrain5sp2020.arff
bash -c "echo '@relation csc558lazytrain5sp2020' > csc558lazytrain5sp2020.arff"
bash -c "grep @ csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazytrain5sp2020.arff"
bash -c "grep ^0, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazytrain5sp2020.arff"
echo "making 5 noisey training instances in csc558lazynoise5sp2020.arff"
making 5 noisey training instances in csc558lazynoise5sp2020.arff
echo "Sin, Tri, Sqr, Saw, Pulse:"
Sin, Tri, Sqr, Saw, Pulse:
bash -c "echo '@relation csc558lazynoise5sp2020' > csc558lazynoise5sp2020.arff"
bash -c "grep @ csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
bash -c "grep ^265544, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
bash -c "grep ^657867, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
bash -c "grep ^866860, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
bash -c "grep ^320328, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
bash -c "grep ^296306, csc558lazy10005sp2020.arff | grep -v  @relation >> csc558lazynoise5sp2020.arff"
```

Running **make train** gives **csc558lazytrain5sp2020.arff** a @relation line of **csc558lazytrain5sp2020**, and **csc558lazynoise5sp2020.arff** a @relation name of **csc558lazynoise5sp2020**. Both files get all of the @attribute declarations of **csc558lazy10005sp2020.arff**, along with the ARFF @data line. Training file **csc558lazytrain5sp2020.arff** gets the five 0-noise instances with **tid** == 0, and **csc558lazynoise5sp2020.arff** gets five noise-bearing instances with **tid** values of 265544 (SinOsc), 657867 (TriOsc), 866860 (SqrOsc), 320328 (SawOsc), and 296306 (PulseOsc) with the tid attribute intact. You can make these two training files via a text editor by editing **csc558lazy10005sp2020.arff**, but **make train** on acad is easier. **Verify** in Weka that each has one of each **tosc** type with the specified tids and exactly 5 instances. Leave these in your project directory when you turn it in.

13. In Weka load training set **csc558lazytrain5sp2020.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 **csc558lazy10005sp2020.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.

[7] The TIME and FREQ waveform graphs and the .wav audio files for these two training sets are linked at http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.html .

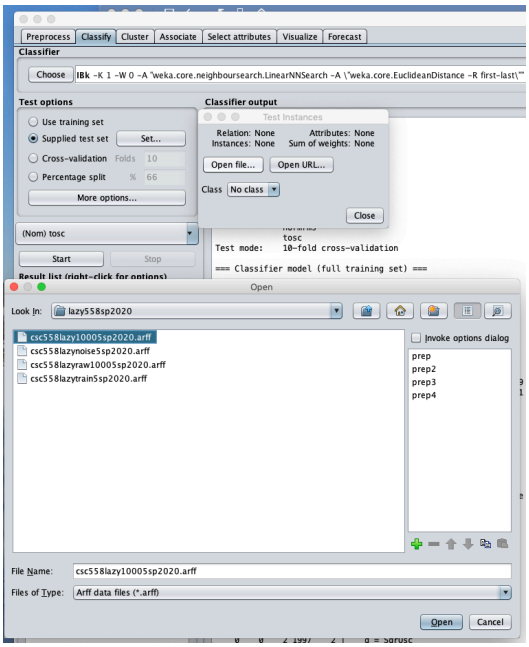

Figure 1: Using a Supplied test dataset in Weka's Classify tab

Q8. Repeat the tests of Q5 with the tid-deleted **csc558lazytrain5sp2020.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 %
OneR:	Correctly Classified Instances	N	N.N %
J48:	Correctly Classified Instances	N	N.N %
RandomTree:	Correctly Classified Instances	N	N.N %
NaiveBayes:	Correctly Classified Instances	N	N.N %
BayesNet:	Correctly Classified Instances	N	N.N %
IBk:	Correctly Classified Instances	N	N.N %
KStar:	Correctly Classified Instances	N	N.N %
LWL:	Correctly Classified Instances	N	N.N %

2020:

ZeroR:	Correctly Classified Instances	2001	20 %
OneR:	Correctly Classified Instances	2001	20 %
J48:	Correctly Classified Instances	2397	23.958 %
RandomTree:	Correctly Classified Instances	2652	26.5067 %
NaiveBayes:	**Correctly Classified Instances**	**4470**	**44.6777 %**
BayesNet:	Correctly Classified Instances	2001	20 %
IBk:	**Correctly Classified Instances**	**4913**	**49.1054 %**
KStar:	Correctly Classified Instances	1957	19.5602 %
LWL:	**Correctly Classified Instances**	**2889**	**28.8756 %**

2018:

ZeroR:	Correctly Classified Instances	2001	20 %
OneR:	Correctly Classified Instances	2001	20 %
J48:	Correctly Classified Instances	2441	24.3978 %
RandomTree:	Correctly Classified Instances	2654	26.5267 %
NaiveBayes:	**Correctly Classified Instances**	**4524**	**45.2174 %**
BayesNet:	Correctly Classified Instances	2001	20 %
IBk:	**Correctly Classified Instances**	**4982**	**49.7951 %**
KStar:	Correctly Classified Instances	1964	19.6302 %
LWL:	**Correctly Classified Instances**	**3240**	**32.3838 %**

Q9. Account for the top three classifiers for Q8. Why is their performance substantially better than the remaining classifiers?

The instance-based, lazy classifiers IBk and LWL are intended to work with small training sets. The linear relations among the attributes of an audio file continue to work best with the instance-attribute distance measures of IBk and LWL, and the statistical analysis of NaiveBayes. LWL is essentially lazy NaiveBayes[8]. The signal attributes of the instances correlate in a linear, conditional-probability manner to the tosc attribute, hence the three winners here. KStar does not work as well because the audio application distance among the attributes of IBk is linear, and hence a better measure than the information entropy of KStar.

14. In Weka load training set **csc558lazynoise5sp2020.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 **csc558lazy10005sp2020.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 **csc558lazynoise5sp2020.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 %
OneR:	Correctly Classified Instances	N	N.N %
J48:	Correctly Classified Instances	N	N.N %
RandomTree:	Correctly Classified Instances	N	N.N %
NaiveBayes:	Correctly Classified Instances	N	N.N %
BayesNet:	Correctly Classified Instances	N	N.N %
IBk:	Correctly Classified Instances	N	N.N %
KStar:	Correctly Classified Instances	N	N.N %
LWL:	Correctly Classified Instances	N	N.N %

2020:

ZeroR:	Correctly Classified Instances	2001	20 %
OneR:	Correctly Classified Instances	2001	20 %
J48:	Correctly Classified Instances	1842	18.4108 %
RandomTree:	Correctly Classified Instances	3887	38.8506 %
NaiveBayes:	**Correctly Classified Instances**	**7165**	**71.6142 %**

[8] http://faculty.kutztown.edu/parson/spring2020/LocallyWeightednaivebayes.pdf

BayesNet:	Correctly Classified Instances	2001	20 %
IBk:	**Correctly Classified Instances**	**7480**	**74.7626 %**
KStar:	Correctly Classified Instances	5001	49.985 %
LWL:	**Correctly Classified Instances**	**5629**	**56.2619 %**

2018:

ZeroR:	Correctly Classified Instances	2001	20 %
OneR:	Correctly Classified Instances	2001	20 %
J48:	Correctly Classified Instances	3141	31.3943 %
RandomTree:	Correctly Classified Instances	2777	27.7561 %
NaiveBayes:	**Correctly Classified Instances**	**7060**	**70.5647 %**
BayesNet:	Correctly Classified Instances	2001	20 %
IBk:	**Correctly Classified Instances**	**7543**	**75.3923 %**
KStar:	Correctly Classified Instances	3887	38.8506 %
LWL:	**Correctly Classified Instances**	**5976**	**59.7301 %**

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

The noise in the 5-instance training set **csc558lazynoise5sp2020.arff** is more representative of noise-bearing test set instances than the noiseless training instances of **csc558lazytrain5sp2020.arff**. A look at their FREQ graphs on http://faculty.kutztown.edu/parson/spring2020/CSC558Audio1_2020.html shows that the white noise affects the baseline frequency-amplitude levels substantially, and the primary waveforms to a visible degree. Since white noise is uniformly random across all noisy audio .wav files in this study, it has a similar effect across them all. The noiseless tid==0 training set is missing these level adjustments and signal interference.

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

KStar does not work as well because the audio application distance among the attributes of IBk is linear, and hence a better measure than the information entropy of KStar.

Q13 points are for a correctly saved **csc558lazy10005sp2020.arff** in the project directory.

Q14 points are for a correctly saved **csc558lazytrain5sp2020.arff** in the project directory.

Q15 points are for a correctly saved **csc558lazynoise5sp2020.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, run **make turnitin** and hit Enter at the prompt by the project deadline. You will not receive an email.