CSC 523 Scripting for Data Science, Fall 2020

Dr. Dale E. Parson, Assignment 2, scikit-learn for numeric->numeric regression.

DUE By 11:59 PM on Thursday November 12, 2020 via make turnitin on mcgonagall or acad. The standard 10% per day deduction for late assignments applies.

This project reuses the data and project outline for Assignment 2 from CSC558 in spring 2020. That project includes an overview of its audio signal data, including several tagged attributes. The current project focuses on exploring scikit numeric regression classes for fitting non-target numeric attributes to numeric target attribute **tnoign**, the tagged level of noise gain in each audio signal in the range [0.0, 1.0]. We will have class work time during at least one session.

From acad it will be necessary to **ssh mcgonagall** from acad in order to **time make test**. You can also download and run the VPN and then putty or ssh into mcgonagall.kutztown.edu directly. Testing must occur on mcgonagall because of increasing CPU load in these projects. This test runs about 7 CPU-bound minutes for me. We do not want to swamp interactive acad users. If you are on campus, you can log directly into mcgonagall.kutztown.edu without going through acad. Also, you can perform file editing via acad, since acad and mcgonagall share the same student and faculty networked file systems. You can **make turnitin** at the end of the project from either machine.

Perform the following steps to set up for this project. Start out in your login directory.

```
cd  $HOME
mkdir  DataMine  # This should already be there.
cd  ./DataMine
cp  ~parson/DataMine/whitenoise523fall2020.problem.zip  whitenoise523fall2020.problem.zip
unzip  whitenoise523fall2020.problem.zip
cd  ./whitenoise523fall2020
```

This is the directory from which you must run **make turnitin** by the project deadline to avoid a 10% per day late penalty. **Please do not change the name of this directory**, since my test scripts depend on it. Some of the larger data files for this project are in directory ~parson/DataMine, with symbolic links installed to our project directory by **make test** for ease of examining them with an editor.
~parson/DataMine/ csc558wn10Ksp2020.arff # The input to your script, copied to your directory.
~parson/DataMine/csc558wn10Ksp2020.arff.ref # Edited output ARFF ref file, a symbolic link.

You will see the following files in this **whitenoise523fall2020** directory:

whitenoise523fall2020.py	The Python program you must complete.
csc558wn10Ksp2020.arff	Your program's initial input file. Testing copies a fresh copy into your directory from ~parson/DataMine each time, since your program must modify the initial data and save it.
makefile & makelib	Files needed to **time make test** and **make turnitin**.
arfflib_3_1.py	My ARFF attribute & data I/O and manipulation module. This module is also useful for manipulating in-memory CSV data.

There are several testing .ref files containing expected output, and __init__.py for module initialization. Detailed STUDENT [A-K] comments in whitenoise523fall2020.py give instructions. Most code is very similar to what you coded for Assignment 2.

A successful **time make clean test** looks like this. The **bold-highlighted** test will pass after your complete STUDENT preprocessing requirements A and B discussed below. I have lettered them in correct working order. The red-highlighted test will pass after you complete STUDENT C through K, the analysis steps. If the "import graphviz" statement reports an error on your **time make test**, try running **pip install graphviz** from your command line. You should not need to do this; let me know if you do.

$ **time make clean test**

/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc __pycache__/*.pyc
/bin/rm -f junk* *.pyc csc558wn10Ksp2020.arff.ref whitenoise523fall2020.out.txt metrics.txt
/bin/rm -f *.tmp *.o *.dif *.out *.csv __pycache__/* whitenoise523fall2020.trace.txt timetrace.txt
/bin/ln -s /home/kutztown.edu/parson/DataMine/csc558wn10Ksp2020.arff.ref csc558wn10Ksp2020.arff.ref
/bin/cp -p /home/kutztown.edu/parson/DataMine/csc558wn10Ksp2020.arff csc558wn10Ksp2020.arff
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. time /usr/local/bin/python3.7 whitenoise523fall2020.py csc558wn10Ksp2020.arff > whitenoise523fall2020.out.txt 2>whitenoise523fall2020.trace.txt"
egrep -v '@relation' csc558wn10Ksp2020.arff | egrep -v '^%' > csc558wn10Ksp2020.tmp
diff csc558wn10Ksp2020.tmp csc558wn10Ksp2020.arff.ref > csc558wn10Ksp2020.arff.dif
OUTPUT csc558wn10Ksp2020.arff IS OK
diff whitenoise523fall2020.out.txt whitenoise523fall2020.out.ref > whitenoise523fall2020.out.dif
egrep COEF whitenoise523fall2020.trace.txt | sort -r -k2 | sort --stable -n -k16 > metrics.txt
egrep TIME whitenoise523fall2020.trace.txt | sort -r -k2 > timetrace.txt
OUTPUT whitenoise523fall2020.out.txt IS OK, see also whitenoise523fall2020.trace.txt and metrics.txt

real 7m0.662s
user 12m37.757s
sys 23m3.406s

Raw output files are whitenoise523fall2020.out.txt (via stdout) and whitenoise523fall2020.trace.txt (via stderr). Testing extracts file **metrics.txt**, sorted on accuracy as measured by the correlation coefficient, and model running time as the secondary sort key, with the most accurate test run at the bottom of the file. Testing also extracts **timetrace.txt**, sorted on model run time, with the fastest at the bottom of the file. File **whitenoise523fall2020.trace.txt** shows linear regression coefficients. These appear at the end of this document. File metrics shows the relative timing of the three search strategies for instance-based KNN. Note that the highest-scoring number of neighbors averaged is the same as for csc558 assn2, 9 neighbors. The reported target **tnoign** value is the **mean** of the K nearest neighbors; it is the **mode** for discrete target classification like assn2. On-the-fly search trees derived from the training set show more acceleration for larger training sets. For tiny training sets, linear ("brute") search may be faster, since it does not have to build a search tree.

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

TIME **0:00:00.647157** DATA 684 **csc558wn10Ksp2020.arff_RAND_NORM** REGRESSOR KNeighborsRegressor(**n_neighbors=9**,p=1,algorithm=**'brute'**,weights='distance') TRAIN # 5000 TEST # 5000 CORR COEF **0.881102** MSQERROR **0.000443** MABSERROR **0.016161**
TIME **0:00:00.467477** DATA 685 **csc558wn10Ksp2020.arff_RAND_NORM** REGRESSOR KNeighborsRegressor(**n_neighbors=9**,p=1,algorithm=**'kd_tree'**,weights='distance') TRAIN # 5000 TEST # 5000 CORR COEF **0.881102** MSQERROR **0.000443** MABSERROR **0.016161**
TIME **0:00:00.443327** DATA 686 **csc558wn10Ksp2020.arff_RAND_NORM** REGRESSOR KNeighborsRegressor(n_neighbors=9,p=1,algorithm=**'ball_tree'**,weights='distance') TRAIN # 5000 TEST # 5000 CORR COEF **0.881102** MSQERROR **0.000443** MABSERROR **0.016161**

Testing handout code fails because your code has not yet run to mutate its input file.
Running **less csc558wn10Ksp2020.arff.dif** or **less whitenoise523fall2020.out.dif** lets you step through .dif files on failed tests. Hit **Return** to step a line at a time, **Space** for a screen, and **q** to get out.

If you program crashes, the error may report like this:

/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. time /usr/local/bin/python3.7 whitenoise523fall2020.py csc558wn10Ksp2020.arff > whitenoise523fall2020.out.txt 2>whitenoise523fall2020.trace.txt"
make: *** [test] Error 1

The Linux standard output of the your program is redirected to **whitenoise523fall2020.out.txt**, which contains printResults() print output and any print() statements you add temporarily for debugging. Linux standard error goes to **whitenoise523fall2020.trace.txt**, including timing data, LinearRegression coefficient listings, and error messages from program aborts. You can apply the **less** or **tail** commands to whitenoise523fall2020.trace.txt in order to see error messages when make aborts on an error.

$ **tail -30 whitenoise523fall2020.trace.txt**
Traceback (most recent call last):
  File "whitenoise523fall2020.py", line 480, in <module>
    analyze(infilename, dataAttributes, dataInstances)
  File "whitenoise523fall2020.py", line 241, in analyze
    bugcall();
NameError: name 'bugcall' is not defined
3.68user 1.90system 0:02.77elapsed 201%CPU (0avgtext+0avgdata 74772maxresident)k
320inputs+5768outputs (1major+44356minor)pagefaults 0swaps

$ **less whitenoise523fall2020.trace.txt** # scroll with **Return** or **Space**, abort using **q**

Traceback (most recent call last):
  File "whitenoise523fall2020.py", line 480, in <module>
    analyze(infilename, dataAttributes, dataInstances)
  File "whitenoise523fall2020.py", line 241, in analyze
    bugcall();
NameError: name 'bugcall' is not defined
3.68user 1.90system 0:02.77elapsed 201%CPU (0avgtext+0avgdata 74772maxresident)k
320inputs+5768outputs (1major+44356minor)pagefaults 0swaps
whitenoise523fall2020.trace.txt (END)

All project code requirements are documented with upper case **STUDENT** comments in whitenoise523fall2020.py. Each of the 11 STUDENT [A-T] instructions gives its percentage value. When

time make clean test runs and you have re-checked all project requirements, use shell command **make turnitin** and follow the prompts to turn the assignment in to me by the deadline.

$ **make STUDENT**
grep "CSC523 STUDENT.*%" whitenoise523fall2020.py | sed -e 's/^[^#]*# //' |sort
CSC523 STUDENT A 5%: Read input ARFF data.
CSC523 STUDENT B 10%: Filter & write output ARFF data; call analyze().
CSC523 STUDENT C 5%: REGRESSORLIST
CSC523 STUDENT D 10%: Train (fit()) the regressor to the training data
CSC523 STUDENT E 10%: TEST (.predict()) THE TEST DATASET
CSC523 STUDENT F 10%: Invoke, on separate command lines,
CSC523 STUDENT G 10%: Derive INPUTARFFNAME, INPUTARFFATTRS, INPUTARFFSATA
CSC523 STUDENT H 10%: Write outer for loop over 4 dataset variants.
CSC523 STUDENT I 10%: Separate non-target, target data columns.
CSC523 STUDENT J 10%: Partition training & test instances by halving.
CSC523 STUDENT K 10%: Iterate over regressors in INNER LOOP.
grep "CSC523 STUDENT.*%" whitenoise523fall2020.py | wc -l
11

When you have completed coding and **time make test** passes, **make sure to insert your name and other project documentation at the top of whitenoise523fall2020.py**. Run **time make test** a final time, then **make turnitin** and follow the prompt to turn in this project by the due date.

You can use your code for Assignment 2 as a template for most of your steps in this project.

LinearRegression error measures and coefficients.
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

 # NORMalization of data allows comparison of coefficients on a single scale. None has greater range of magnitude than any other to skew the weights. RANDomization of instance order corrects for the following instance order after elimination of the first 5 instances by STUDENT B. Without RANDomization, training occurs only on 2000 'SinOsc', 2000 'TriOsc', and 1000 'PulseOsc' instances, and testing occurs on 1000 'PulseOsc', 2000 'SawOsc', and 2000 'SqrOsc' instances. RANDomization makes the training and testing data instances much more similar to reach other by hitting all signal types.

1 'PulseOsc', 1 'SawOsc', 1 'SinOsc' 1 'SqrOsc' 1 'TriOsc'
2000 'SinOsc' ,2000 'TriOsc', 2000 'PulseOsc', 2000 'SawOsc', 2000 'SqrOsc'

TIME 0:00:00.057154 DATA 579 **csc558wn10Ksp2020.arff_RAND_NORM** REGRESSOR LinearRegression(**normalize=False**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.799666** MSQERROR **0.000681** MABSERROR **0.020504**
tnoign =
0.101176 * centroid
+ 0.108854 * rms
+ -0.014134 * roll25
+ 0.032842 * roll50
+ 0.081759 * roll75
+ -0.033896 * amplbin1
+ -0.080772 * amplbin2
+ 0.007617 * amplbin3
+ -0.031435 * amplbin4

+ 0.004050 * amplbin5
+ -0.035148 * amplbin6
+ 0.043504 * amplbin7
+ -0.042395 * amplbin8
+ 0.034212 * amplbin9
+ 0.025941 * amplbin10
+ 0.027834 * amplbin11
+ 0.004402 * amplbin12
+ 0.003247 * amplbin13
+ 0.021983 * amplbin14
+ -0.005489 * amplbin15
+ 0.018671 * amplbin16
+ 0.014017 * amplbin17
+ 0.038444 * amplbin18
+ 0.025787 * amplbin19
+ 0.101176 * centrfreq
+ -0.014134 * roll25freq
+ 0.032842 * roll50freq
+ 0.081759 * roll75freq
+ 1.162172 * nc
+ 0.241190 * n25
+ -0.635815 * n50
+ -0.591462 * n75
+ 0.128266 * normrms

 # Renormalization within LinearRegression has no measurable effect.

TIME 0:00:00.074933 DATA 580 **csc558wn10Ksp2020.arff_RAND_NORM** REGRESSOR LinearRegression(**normalize=True**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.799666** MSQERROR **0.000681** MABSERROR **0.020504**
tnoign =
0.101176 * centroid
+ 0.108854 * rms
+ -0.014134 * roll25
+ 0.032842 * roll50
+ 0.081759 * roll75
+ -0.033896 * amplbin1
+ -0.080772 * amplbin2
+ 0.007617 * amplbin3
+ -0.031435 * amplbin4
+ 0.004050 * amplbin5
+ -0.035148 * amplbin6
+ 0.043504 * amplbin7
+ -0.042395 * amplbin8
+ 0.034212 * amplbin9
+ 0.025941 * amplbin10
+ 0.027834 * amplbin11
+ 0.004402 * amplbin12
+ 0.003247 * amplbin13
+ 0.021983 * amplbin14
+ -0.005489 * amplbin15

+ 0.018671 * amplbin16
+ 0.014017 * amplbin17
+ 0.038444 * amplbin18
+ 0.025787 * amplbin19
+ 0.101176 * centrfreq
+ -0.014134 * roll25freq
+ 0.032842 * roll50freq
+ 0.081759 * roll75freq
+ 1.162172 * nc
+ 0.241190 * n25
+ -0.635815 * n50
+ -0.591462 * n75
+ 0.128266 * normrms

 # Lack of NORMalization retains the above accuracy, but coefficients are harder to interpret because they adapt to the application weights of the non-target attribute values.

TIME 0:00:00.048382 DATA 197 **csc558wn10Ksp2020.arff_RAND** REGRESSOR LinearRegression(**normalize=False**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.799666** MSQERROR **0.000681** MABSERROR **0.020504**
tnoign =
0.000000 * centroid
+ 9.142047 * rms
+ -0.000000 * roll25
+ 0.000000 * roll50
+ 0.000000 * roll75
+ -0.034048 * amplbin1
+ -0.081089 * amplbin2
+ 0.008104 * amplbin3
+ -0.035273 * amplbin4
+ 0.005475 * amplbin5
+ -0.050548 * amplbin6
+ 0.070096 * amplbin7
+ -0.074805 * amplbin8
+ 0.070860 * amplbin9
+ 0.074362 * amplbin10
+ 0.080711 * amplbin11
+ 0.013664 * amplbin12
+ 0.009329 * amplbin13
+ 0.069412 * amplbin14
+ -0.017251 * amplbin15
+ 0.058115 * amplbin16
+ 0.041911 * amplbin17
+ 0.141417 * amplbin18
+ 0.117618 * amplbin19
+ 0.000034 * centrfreq
+ -0.000007 * roll25freq
+ 0.000007 * roll50freq
+ 0.000016 * roll75freq
+ 0.014577 * nc
+ 0.016322 * n25

+ -0.008654 * n50
+ -0.004190 * n75
+ 0.429543 * normrms

 # Renormalization within LinearRegression has trivial effect on MABSERROR 0.020505.

TIME 0:00:00.083534 DATA 198 **csc558wn10Ksp2020.arff_RAND** REGRESSOR LinearRegression(normalize=True,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.799658** MSQERROR **0.000681** MABSERROR **0.020505**
tnoign =
150734707895.382233 * centroid
+ 9.142884 * rms
+ -207070881170.449554 * roll25
+ -17351977267.355591 * roll50
+ 130196878241.064621 * roll75
+ -0.034036 * amplbin1
+ -0.081098 * amplbin2
+ 0.008111 * amplbin3
+ -0.035259 * amplbin4
+ 0.005469 * amplbin5
+ -0.050571 * amplbin6
+ 0.070187 * amplbin7
+ -0.074797 * amplbin8
+ 0.070777 * amplbin9
+ 0.074393 * amplbin10
+ 0.080752 * amplbin11
+ 0.013640 * amplbin12
+ 0.009360 * amplbin13
+ 0.069457 * amplbin14
+ -0.017315 * amplbin15
+ 0.058177 * amplbin16
+ 0.041940 * amplbin17
+ 0.141401 * amplbin18
+ 0.117633 * amplbin19
+ -6836041.174360 * centrfreq
+ 9390969.667587 * roll25freq
+ 786937.744558 * roll50freq
+ -5904620.328377 * roll75freq
+ 0.014577 * nc
+ 0.016323 * n25
+ -0.008654 * n50
+ -0.004190 * n75
+ 0.429535 * normrms

 # UnRANDomized data gives a skewed training to testing data relationship as noted above.
.49 / .80 is 61.25% of the RANDomized correlation coefficient.

TIME 0:00:00.107281 DATA 6 **csc558wn10Ksp2020.arff** REGRESSOR LinearRegression(**normalize=False**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.487810** MSQERROR **0.002759** MABSERROR **0.042350**
tnoign =

0.000000 * centroid
+ 13.757887 * rms
+ 0.000000 * roll25
+ -0.000000 * roll50
+ 0.000000 * roll75
+ -0.011189 * amplbin1
+ -0.022364 * amplbin2
+ -0.048301 * amplbin3
+ 0.014517 * amplbin4
+ -0.098863 * amplbin5
+ 0.021042 * amplbin6
+ 0.039421 * amplbin7
+ 0.049959 * amplbin8
+ 0.152768 * amplbin9
+ 0.019592 * amplbin10
+ 0.047527 * amplbin11
+ 0.012228 * amplbin12
+ 0.040005 * amplbin13
+ 0.035144 * amplbin14
+ 0.036696 * amplbin15
+ 0.025128 * amplbin16
+ 0.023530 * amplbin17
+ 0.048802 * amplbin18
+ 0.247452 * amplbin19
+ 0.000059 * centrfreq
+ 0.000002 * roll25freq
+ 0.000004 * roll50freq
+ 0.000005 * roll75freq
+ 0.014198 * nc
+ 0.002779 * n25
+ -0.007118 * n50
+ -0.004267 * n75
+ 0.033932 * normrms

TIME 0:00:00.080919 DATA 7 **csc558wn10Ksp2020.arff** REGRESSOR LinearRegression(**normalize=True**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.487810** MSQERROR **0.002759** MABSERROR **0.042347**
tnoign =
68099348921.014061 * centroid
+ 13.759771 * rms
+ 307465347040.179871 * roll25
+ -113232957855.042694 * roll50
+ 78755952564.503326 * roll75
+ -0.011188 * amplbin1
+ -0.022360 * amplbin2
+ -0.048304 * amplbin3
+ 0.014518 * amplbin4
+ -0.098858 * amplbin5
+ 0.021047 * amplbin6
+ 0.039403 * amplbin7
+ 0.049986 * amplbin8

+ 0.152794 * amplbin9
+ 0.019580 * amplbin10
+ 0.047490 * amplbin11
+ 0.012226 * amplbin12
+ 0.040024 * amplbin13
+ 0.035148 * amplbin14
+ 0.036710 * amplbin15
+ 0.025094 * amplbin16
+ 0.023536 * amplbin17
+ 0.048856 * amplbin18
+ 0.247367 * amplbin19
+ -3088405.846699 * centrfreq
+ -13944006.668487 * roll25freq
+ 5135281.535380 * roll50freq
+ -3571698.528997 * roll75freq
+ 0.014198 * nc
+ 0.002781 * n25
+ -0.007118 * n50
+ -0.004267 * n75
+ 0.033947 * normrms

TIME 0:00:00.050320 DATA 388 **csc558wn10Ksp2020.arff_NORM** REGRESSOR LinearRegression(**normalize=False**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.487809** MSQERROR **0.002759** MABSERROR **0.042350**
tnoign =
0.176226 * centroid
+ 0.163815 * rms
+ 0.004018 * roll25
+ 0.016910 * roll50
+ 0.026055 * roll75
+ -0.011139 * amplbin1
+ -0.022277 * amplbin2
+ -0.045399 * amplbin3
+ 0.012938 * amplbin4
+ -0.073132 * amplbin5
+ 0.014631 * amplbin6
+ 0.024466 * amplbin7
+ 0.028313 * amplbin8
+ 0.073757 * amplbin9
+ 0.006834 * amplbin10
+ 0.016390 * amplbin11
+ 0.003939 * amplbin12
+ 0.013926 * amplbin13
+ 0.011130 * amplbin14
+ 0.011676 * amplbin15
+ 0.008073 * amplbin16
+ 0.007869 * amplbin17
+ 0.013267 * amplbin18
+ 0.054252 * amplbin19
+ 0.176226 * centrfreq
+ 0.004018 * roll25freq

+ 0.016910 * roll50freq
+ 0.026055 * roll75freq
+ 1.131896 * nc
+ 0.041067 * n25
+ -0.522975 * n50
+ -0.602368 * n75
+ 0.010133 * normrms

TIME 0:00:00.076522 DATA 389 **csc558wn10Ksp2020.arff_NORM** REGRESSOR LinearRegression(**normalize=True**,copy_X=True) TRAIN # 5000 TEST # 5000 CORR COEF **0.487809** MSQERROR **0.002759** MABSERROR **0.042350**
tnoign =
0.176226 * centroid
+ 0.163815 * rms
+ 0.004018 * roll25
+ 0.016910 * roll50
+ 0.026055 * roll75
+ -0.011139 * amplbin1
+ -0.022277 * amplbin2
+ -0.045399 * amplbin3
+ 0.012938 * amplbin4
+ -0.073132 * amplbin5
+ 0.014631 * amplbin6
+ 0.024466 * amplbin7
+ 0.028313 * amplbin8
+ 0.073757 * amplbin9
+ 0.006834 * amplbin10
+ 0.016390 * amplbin11
+ 0.003939 * amplbin12
+ 0.013926 * amplbin13
+ 0.011130 * amplbin14
+ 0.011676 * amplbin15
+ 0.008073 * amplbin16
+ 0.007869 * amplbin17
+ 0.013267 * amplbin18
+ 0.054252 * amplbin19
+ 0.176226 * centrfreq
+ 0.004018 * roll25freq
+ 0.016910 * roll50freq
+ 0.026055 * roll75freq
+ 1.131896 * nc
+ 0.041067 * n25
+ -0.522975 * n50
+ -0.602368 * n75
+ 0.010133 * normrms