CSC 523 Scripting for Data Science, Fall 2020

Dr. Dale E. Parson, Assignment 5, final assignment, time series for a string-valued class value.

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

This assignment is a sklearn implementation of **PREP STEP H** on page 13 through **Q12** on page 16 of spring 2020's CSC558 Assignment 3 in time-lagging MIDI musical performance data.
http://faculty.kutztown.edu/parson/spring2020/csc558sp2020assn3ANSWERS.pdf
I will take questions **ONLY** in class from 6-8:50 PM on November 30 and December 7, and then only to clarify any confusion. I will go over the assignment requirements in detail on 11/30. This is meant to take the place of an exam and introduces no new concepts or programming techniques not already covered by Assignments 1 through 4. It is a review assignment.

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. 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/TimeMIDICSC523.problem.zip  TimeMIDICSC523.problem.zip
unzip  TimeMIDICSC523.problem.zip
cd  ./TimeMIDICSC523
```

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. The Zoom class video from November 9 that introduced time series uses the CSC558 project as an example. The Zoom video from November 30 will go over this assignment in detail.

You will see the following files in this **TimeMIDICSC523** directory:

TimeMIDICSC523.py The Python program you must complete.
genmidi.py Jython program used to generate this dataset used in a performance for backing tracks.

OneNotePerTimeCSC523Fall2020.arff

Your program's initial input file. It contains these attributes.

```
@attribute movement numeric
@attribute channel numeric
@attribute tonic numeric
@attribute tick numeric
```

```
@attribute notenormalized numeric
@attribute tmode string
% @attribute tmode {Ionian,Mixolydian,Lydian,Aeolian,Dorian,Phrygian,Locrian,Chromatic}
```

tmode is the nominal target attribute representing a musical scale used by a musician playing on a MIDI **channel** (instrument) during each **movement** (major time period). The **tonic** is the "do note" for that movement-channel pair, and the **tick** is the time for that instance's note within the movement. Attribute **notenormalized** gives the actual note played, relative to the **tonic** note. A **notenormalized** value of 0 is the tonic. Here are the notes in each mode in the statistical generator for this dataset.

```
# Major modes (major 3rd)
__IonianMode__ = [0, 2, 4, 5, 7, 9, 11, 12]     # a.k.a. major scale
__LydianMode__ = [0, 2, 4, 6, 7, 9, 11, 12]     # fourth is sharp
__MixolydianMode__ = [0, 2, 4, 5, 7, 9, 10, 12]   # seventh is flat
# Minor modes (minor 3rd)
__AeolianMode__ = [0, 2, 3, 5, 7, 8, 10, 12]    # natural minor scale
__DorianMode__ = [0, 2, 3, 5, 7, 9, 10, 12]     # sixth is sharp
__Phrygian__ = [0, 1, 3, 5, 7, 8, 10, 12]       # 2nd is flat
# Locrian has Minor third, also known as a diminished mode because of flat 5th
__LocrianMode__ = [0, 1, 3, 5, 6, 8, 10, 12]        # 2nd is flat, 5th is flat
# Chromatic is not a mode, it is just all the notes.
__Chromatic__ = [i for i in range(0, 13) # actual notes are [0, 12]
```

OneNoteSorted.arff.ref

The reference file that your return result from preprocess(…) must match. It contains these attributes.

```
@attribute movement numeric
@attribute channel numeric
@attribute tonic numeric
@attribute tick numeric
@attribute notenormalized numeric
@attribute tmode string
@attribute partitionedticks numeric
```

Attribute **partitionedticks** is created by your preprocess(…) function, which combines the following attributes in this way and then sorts the instances on partitionedticks values. Attribute partitionedticks is the compound time stamp.

```
partitionedticks = (movement * 20000) + (channel * 2000) + tick
```

The "* 20000" makes movement the primary sort key; "* 2000" makes channel the secondary sort key, and tick is the tertiary sort key. This partitioning separates the 4 movements, and within a movement separates the 4 musicians that play on distinct channels 0 through 3.

NotesLagged.arff.ref

The reference file that your return result from analyze(…) must match. It contains these attributes.

```
@attribute movement numeric
@attribute channel numeric
@attribute tonic numeric
@attribute tick numeric
@attribute notenormalized numeric
@attribute tmode string
@attribute partitionedticks numeric
@attribute notenormalized_lag1 numeric
@attribute notenormalized_lag2 numeric
@attribute notenormalized_lag3 numeric
@attribute notenormalized_lag4 numeric
@attribute notenormalized_lag5 numeric
@attribute notenormalized_lag6 numeric
@attribute notenormalized_lag7 numeric
@attribute notenormalized_lag8 numeric
@attribute notenormalized_lag9 numeric
@attribute notenormalized_lag10 numeric
```

Each lagged **notenormalized** value is that attribute value from the preceding instance lag**N** notes earlier in the movement-channel, e.g., 1 note earlier for lag1, etc. For leading notes that do not have these predecessors, i.e., notes early in a movement-channel, the lagged value is just the current notenormalized value. Sklearn does not allow unknown values, and imputing the current notenormalized value into unknown lagged attributes does not add data noise. It is simply redundant with the notenormalized note in terms of inferring the tmode.

TimeMIDICSC523.out.ref	Reference file for sys.stdout and print statements from printResults(…).
makefile & makelib	Files needed to **time make test** and **make turnitin**.
arfflib_3_1.py	ARFF attribute & data I/O and manipulation module. This module is also useful for manipulating in-memory CSV data.

A successful **make clean test** looks like this. The **bold-highlighted** test will pass after your complete STUDENT requirements in function preprocess(…).The red-highlighted test will pass after you complete STUDENT requirements in function analyze(…). Running this takes just over a minute.

```
$ make clean test
/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc __pycache__/*.pyc
/bin/rm -f junk* *.pyc OneNoteSorted.arff NotesLagged.arff TimeMIDICSC523.out.txt
/bin/rm -f *.tmp *.o *.dif *.out *.csv __pycache__/*
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. time /usr/local/bin/python3.7
TimeMIDICSC523.py OneNotePerTimeCSC523Fall2020.arff OneNoteSorted.arff NotesLagged.arff >
TimeMIDICSC523.out.txt"
58.47user 2.72system 1:01.04elapsed 100%CPU (0avgtext+0avgdata 110516maxresident)k
0inputs+9064outputs (1major+347765minor)pagefaults 0swaps
```

```
egrep -v '@relation' OneNoteSorted.arff | egrep -v '^%' > OneNoteSorted.tmp
diff OneNoteSorted.tmp OneNoteSorted.arff.ref > OneNoteSorted.arff.dif
```
OUTPUT OneNoteSorted.arff IS OK
```
egrep -v '@relation' NotesLagged.arff | egrep -v '^%' > NotesLagged.tmp
diff NotesLagged.tmp NotesLagged.arff.ref > NotesLagged.arff.dif
OUTPUT NotesLagged.arff IS OK
egrep '^DATA' TimeMIDICSC523.out.txt | sort --stable -n -k16 > metrics.txt
diff TimeMIDICSC523.out.txt TimeMIDICSC523.out.ref > TimeMIDICSC523.out.dif
OUTPUT TimeMIDICSC523.out.txt IS OK
```

Syntax and related error messages to sys.stderr will appear on your terminal without looking into output files. If you receive an error from an output difference from a reference file, apply **less** or **tail -40** to the .dif file at the end of the error report.

All project code requirements are documented with upper case **STUDENT** comments in TimeMIDICSC523.py. Percentage values appear below. When **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.

ADDED 12/2/2020 per an email I sent to the class:

A dif here indicates a problem with your preprocess() function:

diff OneNoteSorted.tmp OneNoteSorted.arff.ref > OneNoteSorted.arff.dif

A dif here indicates a problem with your analyze() function:

diff NotesLagged.tmp NotesLagged.arff.ref > NotesLagged.arff.dif

And a dif here indicates a problem in the output from printResults. These used to be bad arguments to printResults, but I have supplied those this time. So it is likely an assignment into an incorrect variable in analyze() or helpAnalyze(). Assigning into variables names that were never used was a bug that cropped up in several assignments:

diff TimeMIDICSC523.out.txt TimeMIDICSC523.out.ref > TimeMIDICSC523.out.dif

Note the **tail -40** and **less** commands for inspecting .dif files, from the handout:

Syntax and related error messages to sys.stderr will appear on your terminal without looking into output files. If you receive an error from an output difference from a reference file, apply **less** or **tail -40** to the .dif file at the end of the error report.

Note **head -40** on a .dif file shows the first 40 lines, while **tail -40** shows the final 40 lines. Also, within **less**, hit **space bar** to scroll a whole screen, hit **Return** to scroll a line at a time, hit **q** to quit, **/STRING** to search forward for your string, and **?STRING** to search backward for your string

```
$ make STUDENT
grep "STUDENT.*%" TimeMIDICSC523.py | sed -e 's/^[^#]*# //' |sort
STUDENT A 10%: Create derived attribute partitionedticks
STUDENT B 5%: Sort the dataInstances on their partitionedticks
STUDENT C 5%: Train (fit()) the classifier to the training data
STUDENT D 5%  per assignment 2 call to classifier.predict,
STUDENT E 5%: per assignment 2 confusion_matrix() and kappa():
STUDENT F 5%: Assign partitionedticks_column and notenormalized_column
STUDENT G 25%: Implement this pseudocode for lagging notenormalized
STUDENT H 5%: shuffle tmpInstances to randomize timestamps
STUDENT I 10%: Use projectARFF on tmpAttributes, tmpInstances
STUDENT J 10%: Use projectARFF on tmpAttributes, tmpInstances
STUDENT K 5%: Compute half of the length of tmpInstances, then use
STUDENT L 10%: Iterate over classifiers in INNER LOOP.
grep "STUDENT.*%" TimeMIDICSC523.py | wc -l
12
```

When you have completed coding and **make test** passes, **make sure to insert your name and other project documentation at the top of TimeMIDICSC523.py**. Run **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 Assignments 2 through 4 as templates for many of your steps in this project. Assignment 2 is especially relevant, since this assignment performs classification, not regression. Assignment 4 is relevant because it, too, performs time series analysis. We are not using Python datetime objects as our timestamps in this assignment, instead using partitionedticks.

The best-performing DecisionTreeClassifier for this assignment is graphed here.

https://acad.kutztown.edu/~parson/OneNoteSorted_arff_lag6_DecisionTreeClassifier_criterion__gini__random_state_42_.pdf with measures %correct = 0.942947, kappa = 0.929626.

```
ATTRIBUTES FOR DATA215 ['channel', 'tonic', 'notenormalized',
'notenormalized_lag1', 'notenormalized_lag2', 'notenormalized_lag3',
'notenormalized_lag4', 'notenormalized_lag5', 'notenormalized_lag6',
'notenormalized_lag7', 'notenormalized_lag8', 'notenormalized_lag9',
'notenormalized_lag10'] -> tmode
DATA215 OneNoteSorted.arff_lag10 CLASSIFIER
RandomForestClassifier(random_state=42) TRAIN # 4154 TEST # 4154 Correct 4053
%correct 0.975686 kappa 0.969936

class labels ('Ionian', 'Mixolydian', 'Lydian', 'Aeolian', 'Dorian',
'Phrygian', 'Locrian', 'Chromatic')
[[1413    0    0    0    0    0    1    0]
 [   0  369    0    0    0    0    0   29]
 [   0    0  654    0    0    0    0    9]
 [   0    0    0  506    0    0    0    0]
 [   0    0    0    0  127    0    0    0]
 [   0    0    0    0    0  282    0    0]
 [  13    0    0    0    0    0  224    0]
 [   0   23   26    0    0    0    0  478]]
```

The pattern of errors in the above confusion matrix correspond closely to those made by Weka in the CSC558 Assignment 3 at this step. Weka's matrix numbers are double because we used 10-fold cross-validation in Weka, where 9/10th of the instances are selected for training, 1/10th for testing, and the process repeats 10 times, using a different testing 10th each time. Here in sklearn we simply split the shuffled dataset in halves for training and testing. Sometimes cross-validation overfits the model to the training set, an overfitting that can be examined using an external test set as we are doing in sklearn. The correspondences of the kappa value and confusion matrix between sklearn and Weka indicate lack of overfitting in Weka's cross-validation.

BayesNet classifier in Weka:

```
Correctly Classified Instances        8220               98.9408 %
Incorrectly Classified Instances        88                1.0592 %
Kappa statistic                          0.9869
Mean absolute error                      0.0034
Root mean squared error                  0.0465
Relative absolute error                  1.6624 %
Root relative squared error             14.6324 %
=== Confusion Matrix ===

    a    b    c    d    e    f    g    h   <-- classified as
 2884    2    0    3    0    0    0    1 |    a = Ionian
    1  757    4    0    0    0    0    6 |    b = Mixolydian
    0    4 1301    0    0    0    2   15 |    c = Lydian
    1    0    0 1023    0    0    0    0 |    d = Aeolian
    0    0    0    0  256    0    0    0 |    e = Dorian
    0    0    0    0    0  512    0    0 |    f = Phrygian
    7    0    3    0    0    0  502    0 |    g = Locrian
    0   24   13    0    2    0    0  985 |    h = Chromatic
```

The minimal but well-performing DecisionTreeClassifier for this assignment is graphed here. https://acad.kutztown.edu/~parson/OneNoteSorted_arff_lag0_DecisionTreeClassifier_criterion__gini__random_state_42_.pdf with measures %correct = 0.904911, kappa = 0.881727. Here is its confusion matrix.

```
ATTRIBUTES FOR DATA3 ['channel', 'tonic', 'notenormalized'] -> tmode
DATA3 ONENOTESORTED.ARFF_LAG0 CLASSIFIER
DECISIONTREECLASSIFIER(CRITERION='GINI',RANDOM_STATE=42) TRAIN # 4154 TEST # 4154
CORRECT 3759 %CORRECT 0.904911 KAPPA 0.881727
CLASS LABELS ('IONIAN', 'MIXOLYDIAN', 'LYDIAN', 'AEOLIAN', 'DORIAN', 'PHRYGIAN',
'LOCRIAN', 'CHROMATIC')
[[1414    0    0    0    0    0    0    0]
 [   0  384    0    0    0    0    0   14]
 [   0    0  646    0    0    0    0   17]
 [   0    0    0  506    0    0    0    0]
 [   0    0    0    0  127    0    0    0]
 [   0    0    0    0    0  282    0    0]
 [  87    0    0    0    0    0  150    0]
 [   0  151  126    0    0    0    0  250]]
```

This tree that uses entropy instead of gini decision making as the same %correct and kappa measures. https://acad.kutztown.edu/~parson/OneNoteSorted_arff_lag0_DecisionTreeClassifier_criterion__entropy__random_state_42_.pdf

The fact that lag0 models perform relatively well indicates that the simple combination of tonic and notenormalized – the note being played against that tonic – are relatively good indicators of the mode (scale) **for this composition**. Even though we have reduced the non-target attributes down to channel (musician), tonic, and notenormalized, those are set against statistical regularities in this piece of music that contribute to the machine learning process, as demonstrated by the following tmode distributions.

```
$ egrep '^[0-9]'  OneNotePerTimeCSC523Fall2020.arff | cut -d, -f6 | sort | uniq -c
   1024 Aeolian
   1024 Chromatic
    256 Dorian
   2890 Ionian
    512 Locrian
   1322 Lydian
    768 Mixolydian
    512 Phrygian
```

```
$ $ egrep '^[0-9]'  OneNotePerTimeCSC523Fall2020.arff | cut -d, -f6 | sort | uniq -c
   1024 Aeolian
   1024 Chromatic
    256 Dorian
   2890 Ionian
    512 Locrian
   1322 Lydian
    768 Mixolydian
    512 Phrygian
```

```
$ grep attribute  OneNotePerTimeCSC523Fall2020.arff | cat -n
     1	@attribute movement numeric                # attributesToIgnoreInClassification
     2	@attribute channel numeric
     3	@attribute tonic numeric
     4	@attribute tick numeric                    # attributesToIgnoreInClassification
     5	@attribute notenormalized numeric
     6	@attribute tmode string
     7	% @attribute tmode {Ionian,Mixolydian,Lydian,Aeolian,Dorian,Phrygian,Locrian,Chromatic}
```

```
$ egrep '^[0-9]'  OneNotePerTimeCSC523Fall2020.arff | cut -d, -f2,3,5,6 | sort | uniq -c
    496 0,7,0,Ionian        # Blue highlights – channel,tonic determine mode in this composition
    206 0,7,11,Ionian
    122 0,7,2,Ionian
    294 0,7,4,Ionian
    164 0,7,5,Ionian
    260 0,7,7,Ionian
    122 0,7,9,Ionian
    192 0,9,0,Aeolian
     70 0,9,10,Aeolian
      6 0,9,2,Aeolian
     78 0,9,3,Aeolian
     34 0,9,5,Aeolian
    116 0,9,7,Aeolian
     16 0,9,8,Aeolian
    442 1,7,0,Ionian        # __IonianMode__ =  [0, 2, 4, 5, 7, 9, 11]    # a.k.a. major scale
    144 1,7,0,Locrian       # __LocrianMode__ = [0, 1, 3, 5, 6, 8, 10]      # 2nd is flat, 5th is flat
     72 1,7,10,Locrian      # Green – channel,tonic,notenormalized deteremine mode in composition
    140 1,7,11,Ionian
     62 1,7,1,Locrian
     44 1,7,2,Ionian
```

```
 62 1,7,3,Locrian
206 1,7,4,Ionian
100 1,7,5,Ionian
 60 1,7,5,Locrian
 52 1,7,6,Locrian
234 1,7,7,Ionian
 60 1,7,8,Locrian
 60 1,7,9,Ionian
188 1,9,0,Aeolian
 64 1,9,10,Aeolian
 80 1,9,3,Aeolian
 38 1,9,5,Aeolian
122 1,9,7,Aeolian
 20 1,9,8,Aeolian
 76 2,7,0,Chromatic    # __Chromatic__ = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
268 2,7,0,Mixolydian # __MixolydianMode__ = [0, 2, 4, 5, 7, 9, 10]   # seventh is flat
 84 2,7,10,Mixolydian # Magenta could have been Chromatic, but none present
 50 2,7,1,Chromatic
 46 2,7,2,Chromatic
 28 2,7,2,Mixolydian
 66 2,7,3,Chromatic
 68 2,7,4,Chromatic
114 2,7,4,Mixolydian
 68 2,7,5,Chromatic
 76 2,7,5,Mixolydian
 60 2,7,6,Chromatic
 78 2,7,7,Chromatic
166 2,7,7,Mixolydian
 32 2,7,9,Mixolydian
 94 2,9,0,Dorian
 20 2,9,10,Dorian
  6 2,9,2,Dorian
 40 2,9,3,Dorian
 16 2,9,5,Dorian
 62 2,9,7,Dorian
 18 2,9,9,Dorian
 64 3,7,0,Chromatic    # __Chromatic__ = [i for i in range(0, 13) # actual notes are [0, 12]
470 3,7,0,Lydian       # __LydianMode__ = [0, 2, 4, 6, 7, 9, 11, 12]     # fourth is sharp
132 3,7,11,Lydian
 82 3,7,1,Chromatic
 78 3,7,2,Chromatic
 40 3,7,2,Lydian
 56 3,7,3,Chromatic
 64 3,7,4,Chromatic
212 3,7,4,Lydian
 62 3,7,5,Chromatic
 52 3,7,6,Chromatic
112 3,7,6,Lydian
 54 3,7,7,Chromatic
288 3,7,7,Lydian
 68 3,7,9,Lydian
```

```
196 3,9,0,Phrygian
 70 3,9,10,Phrygian
  6 3,9,1,Phrygian
 94 3,9,3,Phrygian
 30 3,9,5,Phrygian
102 3,9,7,Phrygian
 14 3,9,8,Phrygian
```

I experimented with increasing the maximum lag from lag10 to lag12 (not part of your assignment) to help eliminate some of mis-classifications of the Chromatic scale, which uses all 12 notes in an octave. Using lag12 made only a minor improvement in accuracy, because with a uniform random distribution of notes, there is no guarantee that lag12 will actually hit all 12 notes in the Chromatic scale. In fact, the probability is small because of the uniform and Gaussian distributions of repeated notes within any 12-note consecutive sequence.