CSC 523 Scripting for Data Science, Fall 2020

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

DUE By 11:59 PM on Thursday October 22, 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 1 from CSC558 in spring 2020. That project includes an overview of its audio signal data, including several tagged attributes. The goal of the current project differs somewhat from that project. CSC558 added the concepts and practices of instance-based classification (a.k.a. lazy classification), ensemble learning, and time-series modeling, to CSC458 topics including classification of nominal values, regression of numeric values, Bayesian modeling using conditional probabilities, and clustering. CSC523 projects manipulate their datasets at the finer-grain, more detailed level of Python code and libraries than the tool-based approach of CSC458 and CSC558. We will work through this assignment in three stages.

1. On October 5 we will go over the detailed mechanics needed so you can start coding. We will hit as much of the data and project outline for Assignment 1 from CSC558 in spring 2020 and the overview of its audio signal data from CSC558 as time allows.
2. On October 12 we will complete examining the CSC558 perspective on this data, along with brief overviews of conditional probability, instance-based classification, and ensemble learning. We have already had overviews of using information entropy and the gini index in building decision trees during the September 28 class. We will allot some time to in-class project work.
3. On October 19 we will complete discussing any conceptual matters that remain for this project, and spend at least half the class on project work.

The plan after October 22 is to complete a scikit-learn based project on regression – predicting a numeric target attribute and examining modeling techniques – and a fourth project on time-series analysis within Python and scikit-learn. I am hoping to work a fifth project on running Weka in command line mode into the course, but not at the expense of a coding/debugging death march. Coding is more time-consuming and labor-intensive than using an off-the-shelf tool like Weka, requiring spending more time in CSC523 on mechanics, and somewhat less on concepts, than in CSC458 and CSC558. Focus within CSC523 is necessarily on mechanics of coding and library use. However, students who have not taken those courses should ask questions whenever a concept is vague. There is enough time to do this right.

From acad it will be necessary to **ssh mcgonagall** from acad in order to **make test**. Testing must occur on mcgonagall because of increasing CPU load in these projects. 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/lazy523fall2020.problem.zip  lazy523fall2020.problem.zip
unzip  lazy523fall2020.problem.zip
cd  ./lazy523fall2020
```

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. All 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/csc523lazyraw10005sp2020.arff # The input to your script.
~parson/DataMine/csc523lazy10005sp2020.arff.ref
^^^ The reference file for your csc523lazy10005sp2020.arff output.

You will see the following files in this **lazy523fall2020** directory:

lazy523fall2020.py	The Python program you must complete. We will go over October 5.
makefile	Files needed to **make test** and **make turnitin** to get your solution to me.
makelib	
arfflib_3_0.py	My ARFF attribute & data I/O and manipulation module.
	This module is also useful for manipulating in-memory CSV data.
diffarff.py	A test scripting for diffing logical contents of ARFF files, used by make.

There are several testing .ref files containing expected output, and __init__.py for module initialization. Example code for using the arfflib_3_0 data manipulation functions and scikit-learn classification functions appear in example ~parson/DataMine/CSC523Example2/CSC523Example2.py on acad & mcgonagall. We went over this example code in September. Detailed comments in lazy523fall2020.py point you even to line numbers in CSC523Example2.py. There are also files ending in a .extended extension from my test runs that include extra code for normalizing numeric values into the range [0.0, 1.0], as part of my investigation of scikit's K-Nearest-Neighbor, instance-based classifier. You can ignore the .extended files for your programming & testing. I will go over them as we analyze the project.

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

$ **make clean test**
/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc __pycache__/*.pyc
/bin/rm -f junk* *.pyc csc523lazy10005sp2020.arff csc523lazynoise5sp2020.arff csc523lazytrain5sp2020.arff csc523lazyraw10005sp2020.arff
/bin/rm -f *.tmp *.o *.dif *.out *.csv /home/kutztown.edu/parson/tmp/parson*arff __pycache__/*
/bin/ln -sf /home/kutztown.edu/parson/DataMine/csc523lazyraw10005sp2020.arff csc523lazyraw10005sp2020.arff
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. time /usr/local/bin/python3.7 lazy523fall2020.py csc523lazyraw10005sp2020.arff /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff csc523lazytrain5sp2020.arff csc523lazynoise5sp2020.arff > lazy523fall2020.tmp"
28.10user 2.12system 0:27.92elapsed 108%CPU (0avgtext+0avgdata 110124maxresident)k
9728inputs+9856outputs (1major+180981minor)pagefaults 0swaps
grep -v 'ing directory' < lazy523fall2020.tmp | egrep -v '^making|^echo|^bash|^Sin' | /bin/sed -e "s/[^]*parson/STUDENT/g" > lazy523fall2020.out

/bin/ln -sf /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff csc523lazy10005sp2020.arff
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. /usr/local/bin/python3.7 diffarff.py /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff /home/kutztown.edu/parson/DataMine/csc523lazy10005sp2020.arff.ref rel_tol=0.0001 abs_tol=0.000001"
FILES /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff,/home/kutztown.edu/parson/DataMine/csc523lazy10005sp2020.arff.ref OK.
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. /usr/local/bin/python3.7 diffarff.py csc523lazytrain5sp2020.arff csc523lazytrain5sp2020.arff.ref rel_tol=0.0001 abs_tol=0.000001"
FILES csc523lazytrain5sp2020.arff,csc523lazytrain5sp2020.arff.ref OK.
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. /usr/local/bin/python3.7 diffarff.py csc523lazynoise5sp2020.arff csc523lazynoise5sp2020.arff.ref rel_tol=0.0001 abs_tol=0.000001"
FILES csc523lazynoise5sp2020.arff,csc523lazynoise5sp2020.arff.ref OK.
diff lazy523fall2020.out lazy523fall2020.ref > lazy523fall2020.dif
egrep kappa lazy523fall2020.out | sort -n -k16 > kappa.out
egrep csc523lazy10005sp2020 lazy523fall2020.out | egrep kappa | sort -n -k16 > kappa_10005.out
egrep csc523lazytrain5sp2020 lazy523fall2020.out | egrep kappa | sort -n -k16 > kappa_train5.out
egrep csc523lazynoise5sp2020 lazy523fall2020.out | egrep kappa | sort -n -k16 > kappa_noise5.out

The handout **make clean test** looks like this. Any test ending with a non-ignored Error line has failed:

$ **make clean test**
/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc __pycache__/*.pyc
/bin/rm -f junk* *.pyc csc523lazy10005sp2020.arff csc523lazynoise5sp2020.arff csc523lazytrain5sp2020.arff csc523lazyraw10005sp2020.arff
/bin/rm -f *.tmp *.o *.dif *.out *.csv /home/kutztown.edu/parson/tmp/parson*arff __pycache__/*
/bin/ln -sf /home/kutztown.edu/parson/DataMine/csc523lazyraw10005sp2020.arff csc523lazyraw10005sp2020.arff
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. time /usr/local/bin/python3.7 lazy523fall2020.py csc523lazyraw10005sp2020.arff /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff csc523lazytrain5sp2020.arff csc523lazynoise5sp2020.arff > lazy523fall2020.tmp"
3.68user 1.81system 0:02.41elapsed 227%CPU (0avgtext+0avgdata 85728maxresident)k
0inputs+112outputs (1major+49681minor)pagefaults 0swaps
grep -v 'ing directory' < lazy523fall2020.tmp | egrep -v '^making|^echo|^bash|^Sin' | /bin/sed -e "s/[^]*parson/STUDENT/g" > lazy523fall2020.out
/bin/ln -sf /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff csc523lazy10005sp2020.arff
/bin/bash -c "PYTHONPATH=/home/kutztown.edu/parson/DataMine:.:.. /usr/local/bin/python3.7 diffarff.py /home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff /home/kutztown.edu/parson/DataMine/csc523lazy10005sp2020.arff.ref rel_tol=0.0001 abs_tol=0.000001"
Traceback (most recent call last):
  File "diffarff.py", line 41, in <module>
    leftattrs, leftdata = readARFF(leftfile)
  File "/home/kutztown.edu/parson/private/csc523_fall2020/problems/lazy523fall2020/arfflib_3_0.py", line 244, in readARFF

```
    af = open(fname, 'r')
FileNotFoundError: [Errno 2] No such file or directory:
'/home/kutztown.edu/parson/tmp/parson_csc523lazy10005sp2020.arff'
make: *** [test] Error 1
```

That diff test fails because your code has not yet run to create its output files.

All project code requirements are documented with upper case **STUDENT** comments in lazy523fall2020.py. Each of the 20 STUDENT [A-T] instructions is worth 5% of the assignment. 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.

```
$ make STUDENT
grep 'STUDENT [A-Z].*%' lazy523fall2020.py  | sort
   # STUDENT A %: Write your make_centrfreq() ... make_roll75freq() closures:
   # STUDENT B %: Add your (NAME, TYPE, FUNC) triplets to this list:
   # STUDENT C %: Write your closures to make: nc, n25, n50, n75
   # STUDENT D %: Add your (NAME, TYPE, FUNC) triplets to this list:
   # STUDENT E %:
   # STUDENT F %:
   # STUDENT G %:
   # STUDENT H %: Just use writeARFF with file name arg in the variable
# STUDENT I % - Complete CLASSIFIERLIST entries to get all the classifiers
       # STUDENT J %: Train (fit()) the classifier to the training data
       # STUDENT K %  per CSC523Example2.py's "# 2".
       # STUDENT L %: per CSC523Example2.py's "# 3".
   # STUDENT M %: Run shell subprocess "make train" to make the 5-instance
   # STUDENT N %:
   #   STUDENT O %:
   # STUDENT P %:
   # STUDENT Q %:
      # STUDENT R %:
      # STUDENT S %:
      # STUDENT T %:
grep 'STUDENT [A-Z].*%' lazy523fall2020.py  | wc -l
20
```

When you have completed coding and **make test** passes, **make sure to insert your name and other project documentation at the top of lazy523fall2020.py**. Run **make test** a final time, then **make turnitin** and follow the prompt to turn in this project by the due date.

Below are the Python doc strings for all of the functions in arfflib_3_0.py used in this project. There are example uses of all of these in CSC523Example2.py, and there are many examples and direct instructions for library functions to use in lazy523fall2020.py STUDENT comments. Please read all STUDENT comments in the handout code. Examples preceded by line numbers below are from CSC523Example2.py. There are no unknown values (? In Weka, None in Python) in this dataset, so there is no need to use imputeARFF. You can probably ignore the following and just use the example code as a template.

FUNCTIONS

readARFF(fname)

Reads ARFF file named fname and returns (**attrmap, dataset**), where **attrmap is the map from attrname -> (offset, type)** returned by __getAttrIndices__, and **dataset is a 2D list indexed on [row][offset] that holds actual data instances**.
This offset is attribute position, starting at 0, and **type** is one of a date-3-tuple, **'numeric'**, **'string'**, a nominal set in {} delimiters, or a ARFF datetime value. A nominal type field is a 3-tuple of

(**'nominal'**,{NOMINAL_LIST_IN_STRING_FORM},PYTHON_LIST_OF_NOMINAL_SYMBOLS), and a datetime (Weka date) is a 3-tuple consisting of
(**'date'**, Weka-format-string, Python-datetime-strptime-format-string).
A **nominal attribute-value in the dataset is a simple strin**g as read from an ARFF file, and a **date attribute-value is a 2-tuple (STRING_VALUE, Python datetime.datetime object**).

```
464    attrmap, dataset = readARFF(infilename)
```

writeARFF(fname, attrmap, dataset, isDebugMode=False, clobber=False)

Writes ARFF file named fname with data in attrmap and dataset, where attrmap is the map from attrname -> (offset, type) returned by __getAttrIndices__, and dataset is a 2D list indexed on [row][offset] that holds actual data instances. Set isDebugMode to True (default is False) for debugging output to sys.stderr.
Set clobber to True (default is False) to over-write output fname without warning, added 11/24/2019.

```
466    writeARFF(outfilename, prpattrmap, prpdataset, isDebugMode=False)
```

projectARFF(attrmap, dataset, attributesToProject, isKeepingAttributes)

Return new ARFF data that is a projection of a copy of attrmap, dataset, where attrmap is the map from attrname -> (offset, type) returned by __getAttrIndices__ as in readARFF & writeARFF, dataset is a 2D list indexed on [row][offset] that holds actual data instances, attributesToProject is a list of attributes to keep or remove, and isKeepingAttributes is True if attributesToProject should be KEPT in the return value, False if attributesToProject should be DISCARDED in the return value. The attributesToProject can include int attribute offsets, str attribute names, and/or 1-tuples containing ('numeric',), ('string',), ('nominal',), ('date',), and/or ('useless',), where ('useless',) matches all single-value columns in the data.
The return value is a new (attrmap, dataset) pair as in readARFF's return value. added 09/13/2020

```
361    aNOBW, dNOBW = projectARFF(au, du, ['BW'], False)
362    aONLYBW, dONLYBW = projectARFF(au, du, ['BW'], True)
```

joinARFF(attrmap, dataset, nameTypePairs, rowsOfNewColumns)

Return new ARFF data that is a join of a copy of attrmap, dataset, where attrmap is the map from attrname -> (offset, type) returned by

__getAttrIndices__ as in readARFF & writeARFF, dataset is a 2D list
indexed on [row][offset] that holds actual data instances,
nameTypePairs is an ordered list of (NAME, TYPE) attribute columns
to concatenate with the data rows in dataset, and rowsOfNewColumns are
the actual rows of data to concatenate with the number of columns and
the current types given in nameTypePairs. In nameTypePairs, NAME
must be the string name of an attribute not already in attrmap, and
TYPE must be the string 'string' or 'numeric'; 'numeric' columns
must be cast-able to float, and they are cast to float within
rowsOfNewColumns. Weka nominal and date types are not supported
in the added columns. The input attrmap and dataset are not mutated.
joinARFF returns a newattrmap, newdataset pair.

```
367     au, du = joinARFF(aNOBW, dNOBW, [('BW','numeric')], dONLYBW)
```

deriveARFF(attrmap, dataset, nameTypeFunctionTriplets)
Return new ARFF data that is an edit of a copy of attrmap, dataset,
where attrmap is the map from attrname -> (offset, type) returned by
__getAttrIndices__ as in readARFF & writeARFF, dataset is a 2D list
indexed on [row][offset] that holds actual data instances,
nameTypeFunctionTriplets is an ordered list of (NAME, TYPE, FUNC)
attribute columns to mutate OR to concatenate with the data rows in
dataset. In nameTypeFunctionTriplets, NAME may be the string name
of an attribute that is already in attrmap, in which case the copy
of (attrmap, dataset) is mutated, or NAME is a new attribute name to
be joined to the copy of (attrmap, dataset). The result TYPE must be either
'string' or 'numeric'. FUNC is a function that takes 1 argument,
the instance (as a tuple) to be read by FUNC. FUNC should not hard-code
attribute offsets into its data instances; instead, pass FUNC as a closure
that has already captured its attribute indices to read. FUNC returns
a value that deriveARFF either substitutes into the resulting
NAMEd instance for mutation, or appends to a list of instance fields
to be joined per joinARFF. Weka nominal and date types are not supported
in the mutated/or/added columns. The input attrmap and dataset are not
mutated. deriveARFF returns a newattrmap, newdataset pair. The order
of evaluation of FUNC is left-to-right in nameTypeFunctionTriplets.
UPDATE 1 October 2020: New attribute NAMEs in nameTypeFunctionTriplets
are added to the copy of attrmap to be returned BEFORE applying their
FUNCs in left-to-right order of declaration in nameTypeFunctionTriplets.
Their offset columns are incremented starting at the original number
of attributes, for example, if there are 3 original attributes at columns
0, 1, and 2 in the dataset, then two new ones would be added at columns
3 and 4, yielding 5 attributes total, with their columns initialized
to None in the dataset copy BEFORE iterating over the FUNCs. That way,
new derived attribute FUNCs can refer to other new attribute values
in columns to their left. FUNCs apply left-to-right in a given instance
before going on to the next instance.

```
180     def Q12Discretizer(anInstance):
181         # Needed for deriveARFF(attrmap, dataset, nameTypeFunctionTriplets)
182         BW = anInstance[0] # In this example, BW is the only attribute.
```

```
183         if BW < 1:
184             return 0
185         elif BW < 94:
186             return 1
187         elif BW < 180:
188             return 2
189         else:
190             return 3
191     aQ12numSTEPS, dQ12numSTEPS = deriveARFF(aONLYBW, dONLYBW,
192         [('BW', 'numeric', Q12Discretizer)])
```

sortARFF(attrmap, dataset, attributeKeys, sreverse=False)
Sort a copy of the dataset list of instances without mutating
the original, returning only the sorted result list,
where attrmap is the map from attrname -> (offset, type) returned by
__getAttrIndices__ as in readARFF & writeARFF, dataset is a 2D list
indexed on [row][offset] that holds actual data instances,
attributeKeys is a sequence of attribute offsets to be used as keys
in the sort, with most significant attribute coming first, least
significant last, and sreverse as in Python's sort()'s reverse argument.
attributeKeys can contain either numeric indicies or string names
of attribute indicies. Returns a sorted copy of dataset

```
98     # Sort so timedCopy() can look back to previous instance for a value.
99     # sdataset = sortARFF(sattrmap, sdataset, ('HawkYear', 'msnyHstart'))
```

kappa(confusionMatrix)
Compute the Kappa statistic of a confusion matrix.
Param confusionMatrix is 2D list of lists, each sublist is a row
Each row shows the actual class. Each column shows the predicted class.
That row (actual), column (predicted is same as Weka. Also see
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
Return value is a 5-tuple:
(kappa %correct numberCorrect %incorrect numberIncorrect)

```
276             kapp,Pcorrect,numberCorrect,Pincorrect,numberIncorrect  \
277                 = kappa(confuseAuto)
```

Decision Trees graphs from this assignment are on the course page under Assignment 2.

https://faculty.kutztown.edu/parson/fall2020/CSC523Fall2020.html