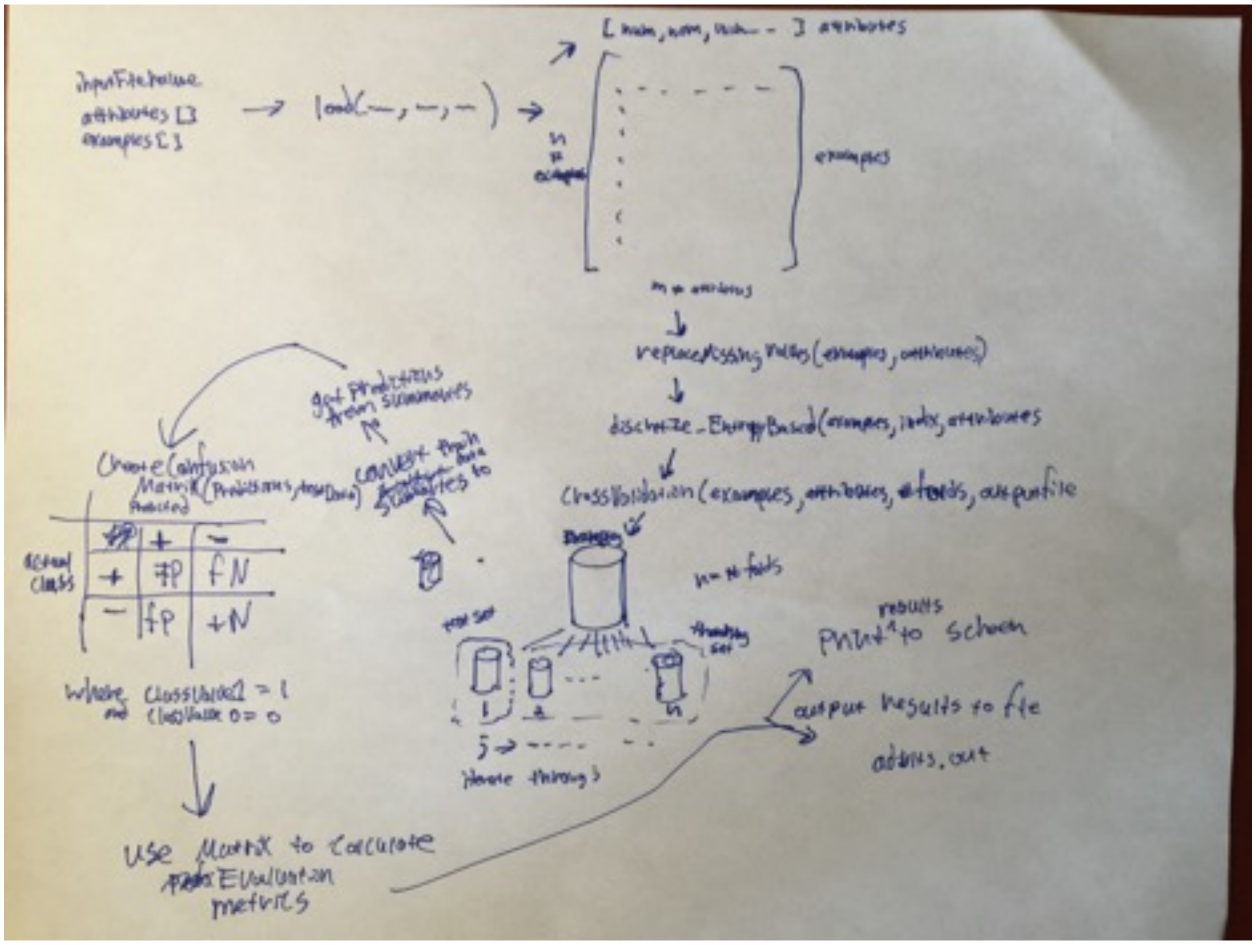
Konrad Rauscher
Introduction to Data Mining
Project 1
2/19/16

Status:
Completed

Things I wish I had been told prior to being given assignment:
Nothing in particular pertaining to the concepts of this class. For this project, I had to learn learning Python on the go, which cost me a lot of time.

Design


inputFilename
attributes []
examples []
load(__, __, __)
[num, nom, num...] attributes
n # examples
examples
m # attributes
replaceMissingValues(examples, attributes)
discretize_EntropyBased(examples, index, attributes
CrossValidation(examples, attributes, #folds, outputfile
n = # folds
test set
training set
1
2
n
5 →
iterate through 5
convert training data to summaries
get predictions from summaries
CreateConfusionMatrix(Predictions, testData)
predicted
+
−
actual class
+
TP
fN
−
fP
TN
where ClassValue1 = 1
and ClassValue 0 = 0
use Matrix to calculate Evaluation metrics
Print results to screen
output results to file
adults.out

A core component of my approach was to create an attributes list of that stored the type (nominal or numeric) of attributes at a given index (column) in my two-dimensional list of examples. Doing this, rather than storing all possible values nominal attributes, made decisions and coding simpler for the rest of the implementation. Being able to determine whether an attribute was numeric or nominal was useful in many instances: replacing missing values (mode for nominal, mean more numeric), summarizing the data (storing mean and std dev if num, or freq if nom), etc.

A particularly difficult challenge for me was implementing the entropy based discretization. In order to obtain the number and location of optimal splits for assigning nominal attributes to ranges of numeric attributes, I used a recursive function that continued to obtain split values until the entropy gain for the best splits were no longer above .05. for attributes 10-12, this usually results in 2 - split locations. Once split, I had to come up with a creative way of assigning informative and functional (meaningfully unique) string values to assign to numeric values found in the given ranges. This solution involved assigning a unique string to all values below the lowest split value and another unique string value for all above the highest. If more than 1 split value existed, a for-loop iterated through in middle split values, assigning values based on the ranges that the values fell in. In order to do this, I had to check if a given value was solely comprised of digits. if it was not, the assumption is made that this value has already been discretized.

Another key decision was to add 1 to the denominators of my Gaussian probability calculations so as to avoid divide by zero errors.

Other than this, my design choices do not deviate from what I believe to be convention. However, I do believe much of my implementation could have been a good deal simpler if i was more familiar with data structures in Python (dicts, counters, etc.).

A feature that I would have like to have implemented with more time is a function that is able to remove redundant/uninformative columns (attributes) from a data set.

Evaluation Results for my implementation:

Models	**Micro precision**	**Micro recall**	**Micro F1**	**Macro precision**	**Macro recall**	**Macro F1**	**Accuracy**
Model- 1	0.83067158	0.83067158	0.83067158	0.90922063	0.64230103	0.67162729	83.0671580
Model- 2	0.82555282	0.82555282	0.82555282	0.90648666	0.63913595	0.66611374	82.5552825
Model- 3	0.82568932	0.82568932	0.82568932	0.90643317	0.63959102	0.66708108	82.6781326
Model- 3	0.82442669	0.82442669	0.82442669	0.90609943	0.63526159	0.66110684	82.4426699
Model- 4	0.82780507	0.82780507	0.82780507	0.90759460	0.64194482	0.67021010	82.7805077
Model- 5	0.82292519	0.82292519	0.82292519	0.90546892	0.63182914	0.65644675	82.2925197
Model- 6	0.82502632	0.82502632	0.82502632	0.90631460	0.63719068	0.66362072	82.5026325
Model- 7	0.82777948	0.82777948	0.82777948	0.90761436	0.64142598	0.66959207	82.7779484
Model- 8	0.82575757	0.82575757	0.82575757	0.90682935	0.63573670	0.66213923	82.5757575
Model- 9	0.83093775	0.83093775	0.83093775	0.90908590	0.64671401	0.67685770	83.0937755
Model- 10	0.82678132	0.82678132	0.82678132	0.90716219	0.63959102	0.66708108	82.6781326
Average Accuracy over all 10 models	0.82518980	0.82518980	0.82518980	0.90642459	0.63701647	0.66341962	82.5189808

	Precision	Recall	F1-measure	Specificity	Accuracy
Weka	0.59531970	0.77273894	0.67252485	0.83477324	0.81992956

Interpretation:

The percentage of positive cases that my implementation was able to catch (recall) were significantly lower that the percentage of positive cases that were currently identified (~.20). Apprently, it is easier for my (and Weka's) Naive Baye's classifier to correctly calculate negative (>50K) value than positive values.

The Weka recall is interestingly significantly lower than macro recall of my model. The implication here is that with my model, when categories are weighted equally, doesn't perform as well because there are poorly contributing categories that negatively affect the effectiveness of informative categories.

Otherwise, the fact that Weka results line up with mine is a good indication of the effectiveness of my classifier.