# Assignment 2 Report: Data Classification

Annelise Wittenberg & Cody Stammer

([wittenberg.18@osu.edu](mailto:wittenberg.18@osu.edu); [stammer.3@osu.edu](mailto:stammer.3@osu.edu))

## Approach

We decided to use python again after our success with using it on the first assignment. When transforming the data, we first eliminated the places list and the word frequency feature vector in order to focus solely on the bigram feature vector for classification. Any articles not containing topics were eliminated from the dataset so we could improve our runtime and allow ourselves more time for testing. Any articles that had multiple topics were given the first topic in the list as their identifying topic for simplicity’s sake. Since all of the topics in a multi-topic article were closely related or synonymous, we figured that the effects choosing the first topic each time were nominal.

Our classifier.py reads in the output file from Assignment 1 and transforms it into Weka-friendly formatting, placing it in out.arff. The out.arff is organized as follows:

@Relation word\_frequency # The attribute we’re classifying on is document word freq.

@ATTRIBUTE word1 NUMERIC

@ATTRIBUTE word2 NUMERIC

…

@ATTRIBUTE wordN NUMERIC

@ATTRIBUTE class-zzz {comma-separated list of topics}

# named class-zzz so as not to be confused with the word “class” which may appear in a

# document

@DATA

# followed by a matrix for each article of 1 column per word containing the

# frequency of the word

We loaded this file into the Weka GUI and used it to train and build two classifiers. We chose to use Naïve Bayes and Decision Tree classifiers.

## Results

Note: The data was classified in Weka on a VM with 11GB of RAM and 61GB of swap space. Using the command: java –jar –Xms11G weka.jar

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Split** | **Time to Build NaiveBayes** | **Time to Test NaiveBayes** | **NaiveBayes Accuracy** | **Time to Build DecisionTree** | **Time to Test DecisionTree** | **DecisionTree Accuracy** |
| 60/40 | 61.68s | 3013.12s | 66.5% | 110.36 | .69s | 45.7472% |
| 80/20 | 59.21s | 4486.54s | 73.56% | 173.75s | .32s | 47.333% |

Running our pre-processing python file in python took a massive 42 Gigabytes total of memory. All 11 gigabytes of RAM were used as well as 30 Gigabytes of swap space. The output was so large the program had troubles closing the file so a forced program kill must be done to release the memory.

Actually importing the data file in Weka results in a usage of 6.7 Gigabytes of RAM and proceeded to use all 11 gigabytes of ram during the Naïve Bayes Classification with and additional 1-2 Gigabytes of swap space.

## Issues

Having enough RAM to process the file was our first issue encountered. We ended up testing our script on an Ubuntu VM that had 11GB of RAM and 61GB of swap space to prevent it from crashing. Had we not had access to this VM, we may not have been able to complete this assignment. We were able to do some other testing on smaller input sets to temporarily get around this issue.

Due to the massive amount of attributes passed into Weka, any visualization done in Weka did not display any useful or interpretive graphics. We have included some in the screenshots section but the images do not communicate much about what is happening to the data after training.

The offline cost for Weka to build our classifiers grew significantly as input size got larger. There was not a noticeable difference in time to build classifiers when the training/testing data split increased.

The online cost for Weka to classify a new tuple wasn’t as bad as the size of the testing set got smaller.

Decoupling of training and testing data (robustness) had a noticeable effect. When the split was 80/20, the classifier had better performance than the 60/40 split. This difference most likely resulted from the fact that the classifier was more intelligent from training with 80% of the data, therefore more quickly identifying the TOPIC label of the testing data. There was also less test data for it to classify resulting in a smaller time for classification.

The larger the split, the more accurate our classifiers were. Again, this accuracy resulted from the higher amount of data training the classifier.

## Assumptions

An assumption we have made while completing this assignment is that when transforming the data, the input data was of the form:

DocID <topic list> <place list> <bigrams> <(word, frequency)>.

Knowing the format of our input helped us to straightforwardly transform the input data into a format that could be used by our classifier packages.

We also assumed that choosing the first topic in a list of topics for a given article would be sufficient when training the classifiers for the data.

## Work Distribution

Input data filtering: Annelise

Input data transformation: Annelise & Cody

Testing: Cody

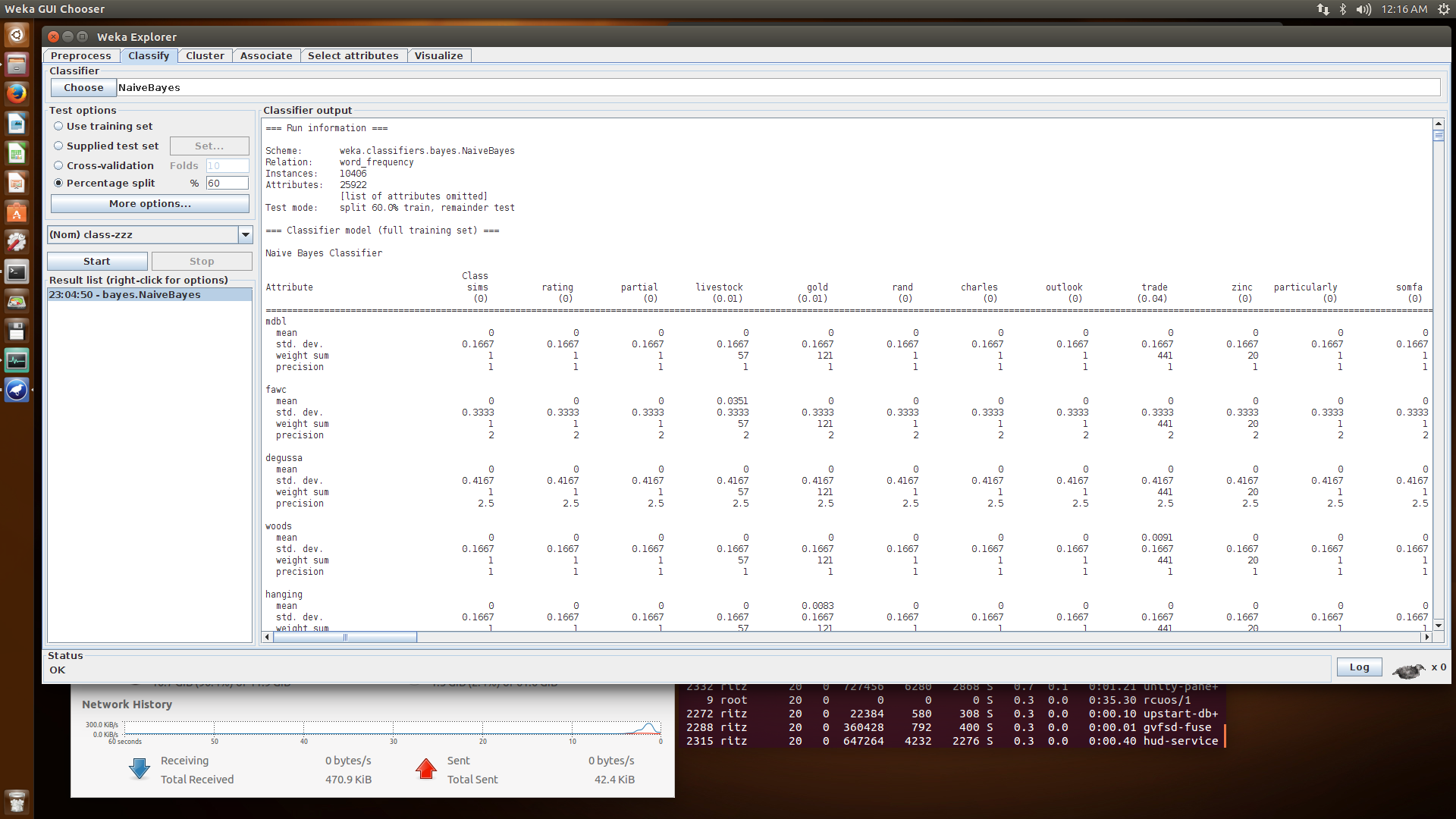
Debugging: Cody

README: Annelise

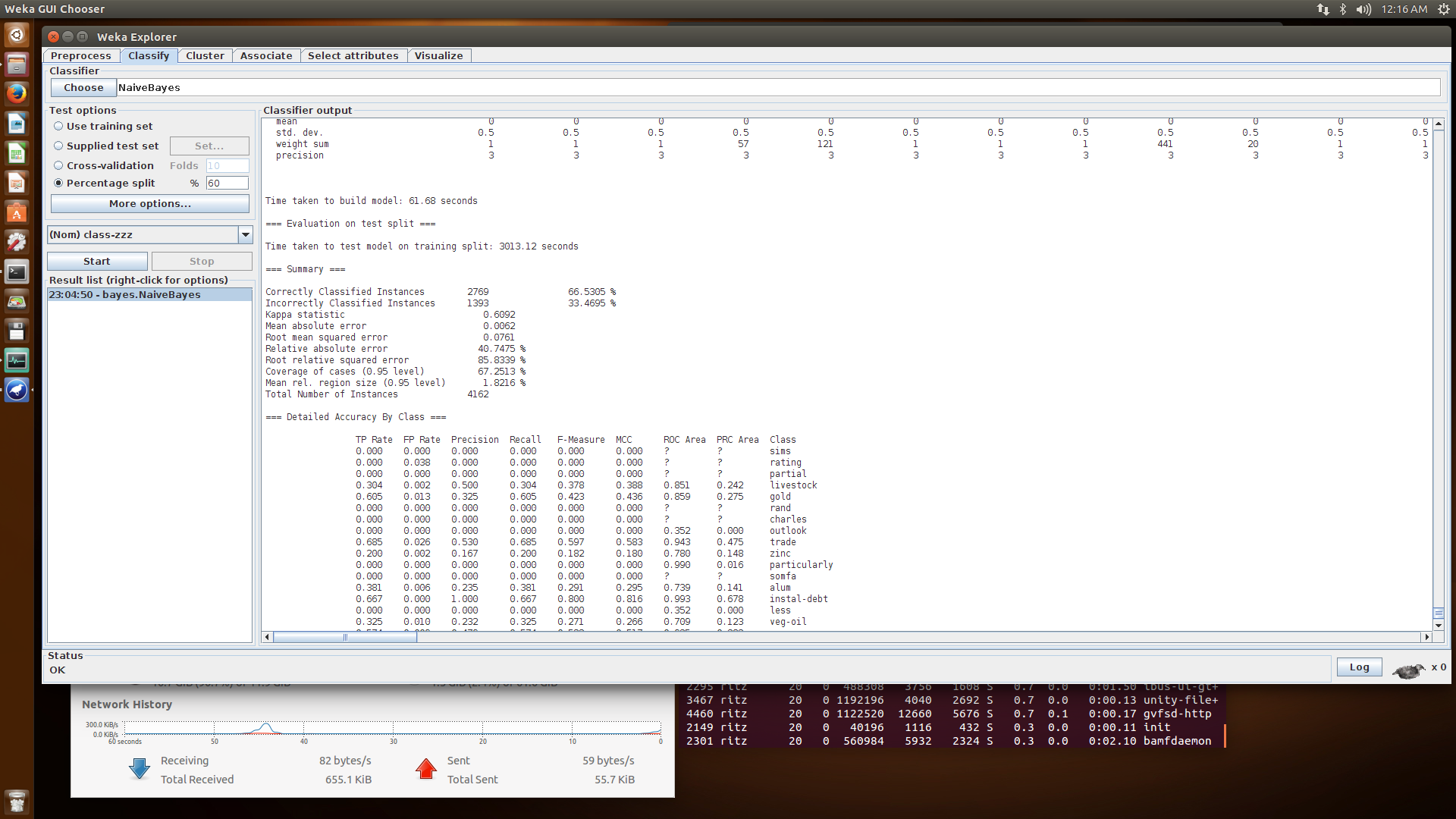
Makefile: Annelise

Report: Annelise & Cody

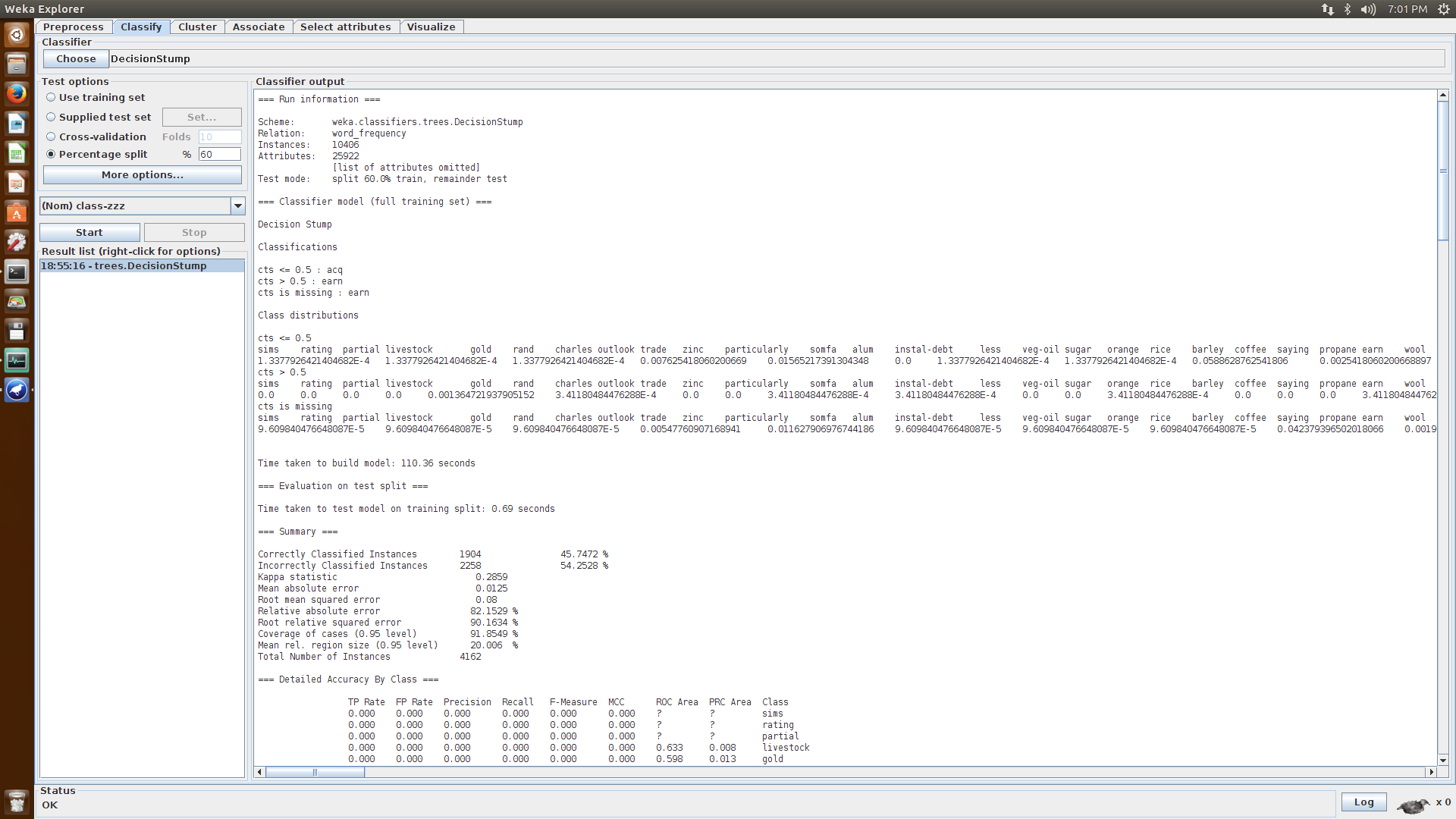
## Screenshots of Weka Analysis



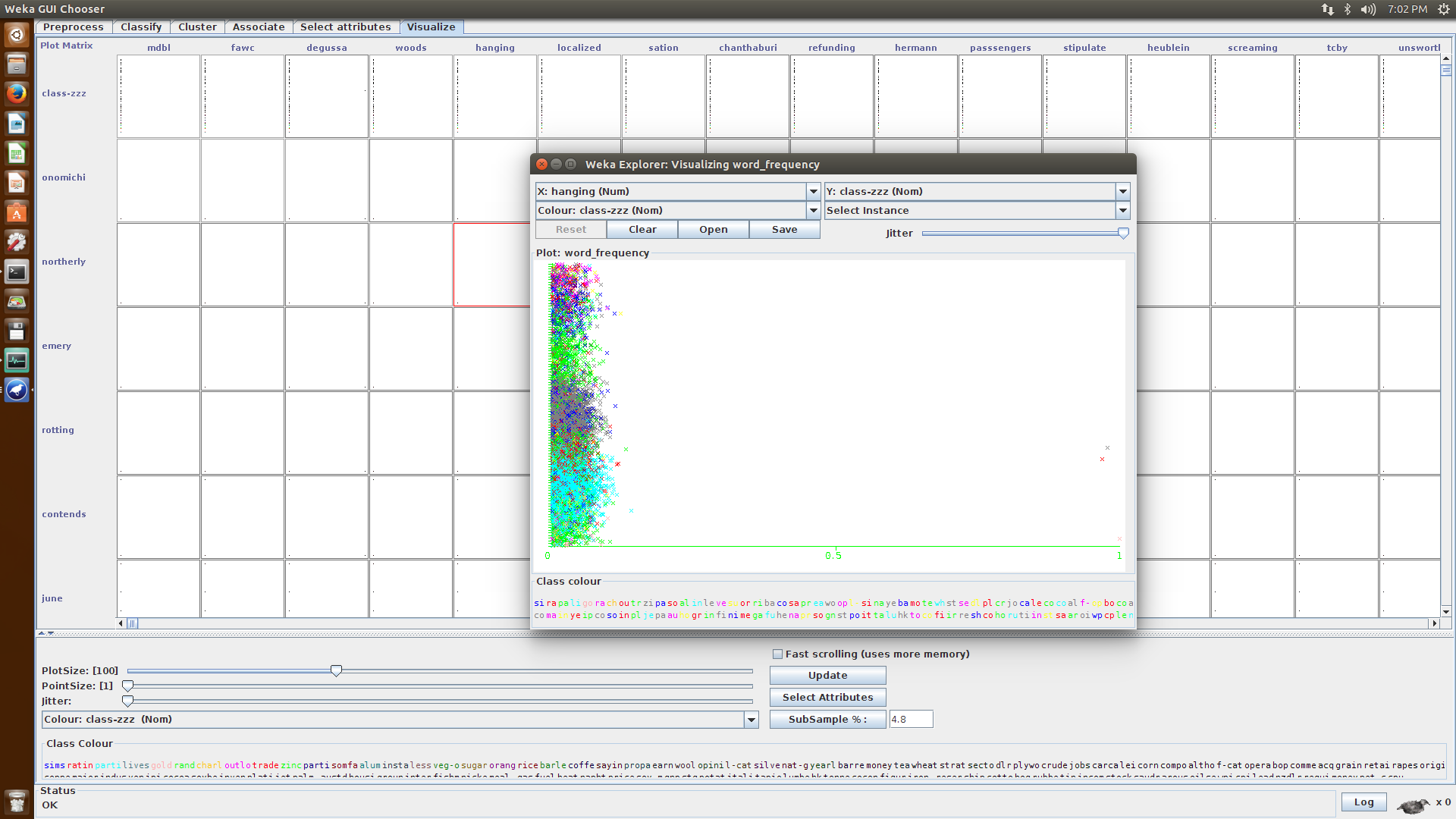
Bayes 60/40 results (Model Creation)



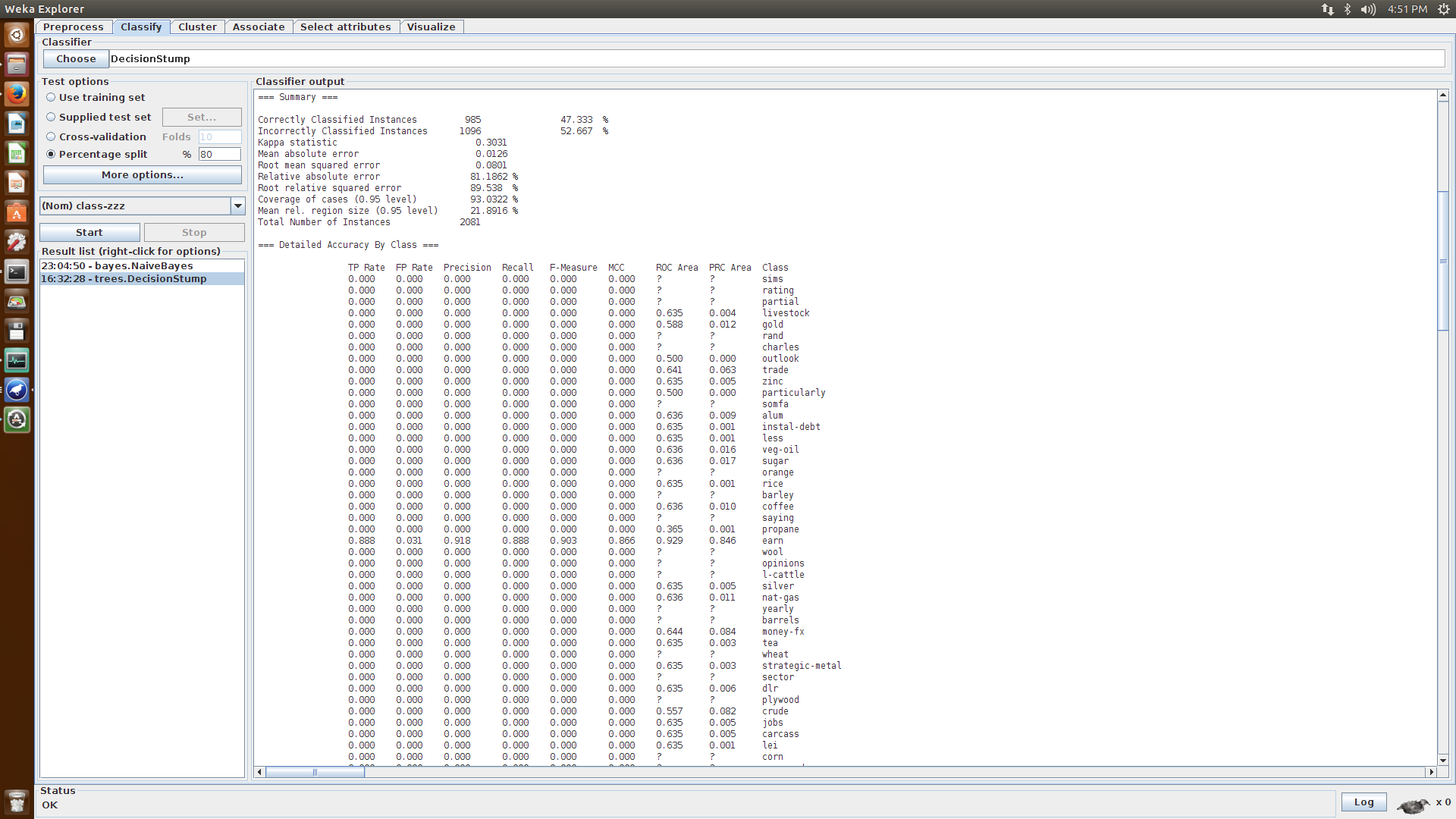
Bayes 60/40 results continued (Accuracy)



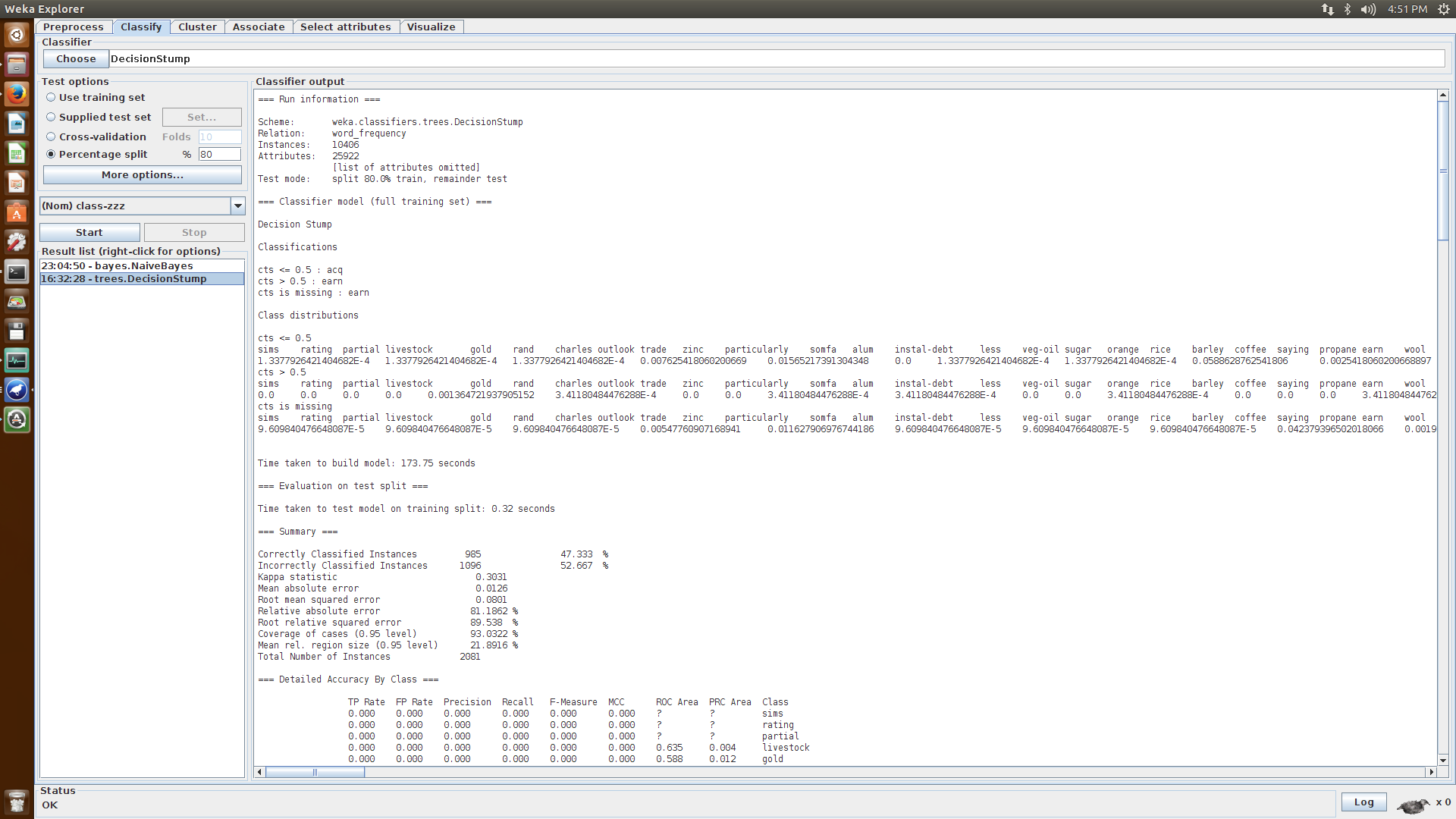
60/40 split Decision Tree Results



60/40 split Visualization of decision tree results



80/20 split Decision Tree Results



80/20 split Decision Tree Results Continued