Joy Jen

yjen2@illinois.edu

CS410 – Tech Review

Dec 9, 2018

Weka Error Classification

Introduction

Weka with its Graphical User Interference (GUI) allows users to do data mining and to learn machine learning without coding. It contains a collection of machine learning algorithms and tools for data preprocessing, classification, clustering, and etc. This post describes the essential processes to do classification using Weka: data preparation, data processing, and choose of classifiers.

Background

Automatic speech recognition (ASR) is an application to convert human speech to a corresponding set of words. Words error rate is a common metric of the performance of a speech recognition. The formula is $WER = \frac{S+D+I}{N}$, where

- S: number of substitutions
- D: number of deletions
- I: number of insertions
- N: total number of words

Different combinations of S, D, I can show certain type of errors such as word splitting or merging. The other common errors include homonym error resulting from the substitution of phonetically identical words, words with similar sound structure and etc. The goal of this classification is to further classify a pair of errors as *homophone*, *similar sound*, *split*, or *merge*.

Data

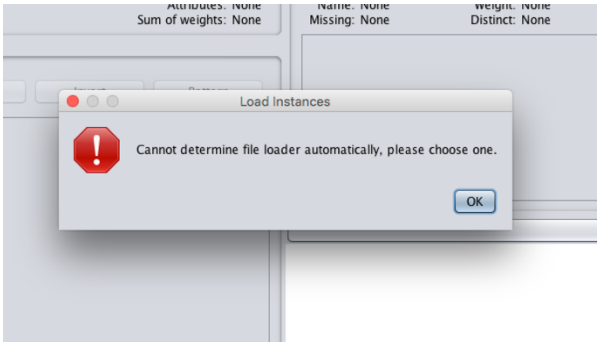
In many machine learning tasks, the default file type is an ARFF (Attribute-Relation File Format) file. It has two sections, header, data. The header of the ARFF file is the name of the relation, a list of the attributes, and their types.

For many text datasets such as movie reviews, articles and journals, and in my case, errors, it is not common to save the dataset in an ARFF file. If one is using its own dataset, he or she can create a text directory. The file structure for the database could be similar to the following:

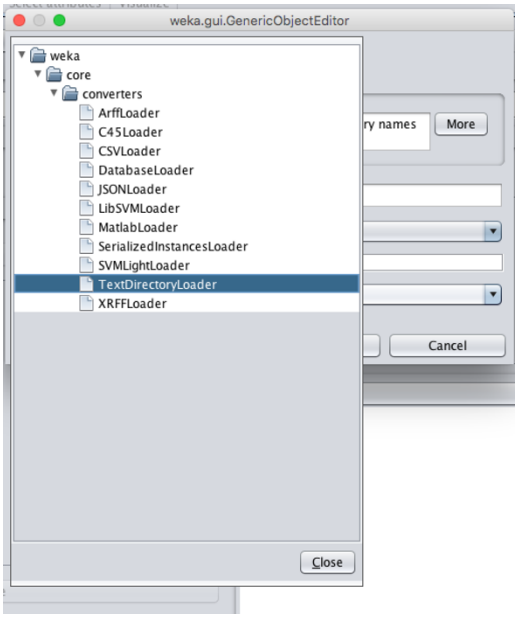
```
txt_pairtoken /

├── split/
│    ├── file0011.txt
│    ├── file0012.txt
│    ├── file0013.txt
│    └── file001...
├── merge/
│    ├── file0021.txt
│    └── file0022.txt
├── similarsound/
│    ├── file0031.txt
└── homophone/
     ├── file0041.txt
     ├── file0042.txt
     ├── file004...
```

I adapted CMU Sphinx (a speech recognition toolkit) to recognize speeches from 650 audio files (155MB) and successfully identified 348 errors under the categories of *homophone*, *similar sound*, *split*, or *merge*. Each error is a plain text file. They are in a single directory **txt_pairtoken** with sub-directories *homophone*, *similar sound*, *split*, and *merge*.



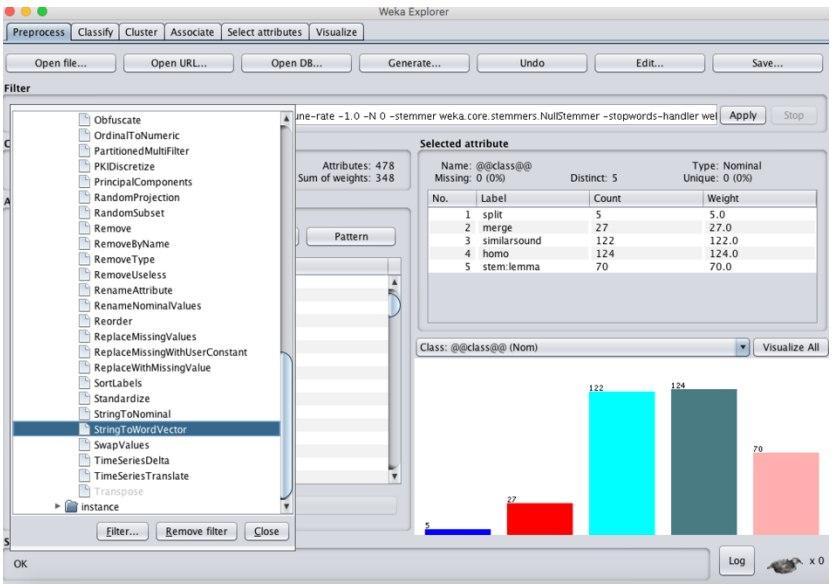
As we open the file directory, a warning window pop up. Click OK.



A *GenericObjectEditor* window appears. Choose *TextDirectoryLoader* and click OK. By using the *TextDirectoryLoader*, Weka automatically creates a relation with 2 attributes: text, class.

To further extract the attributes, we need a language process approach.

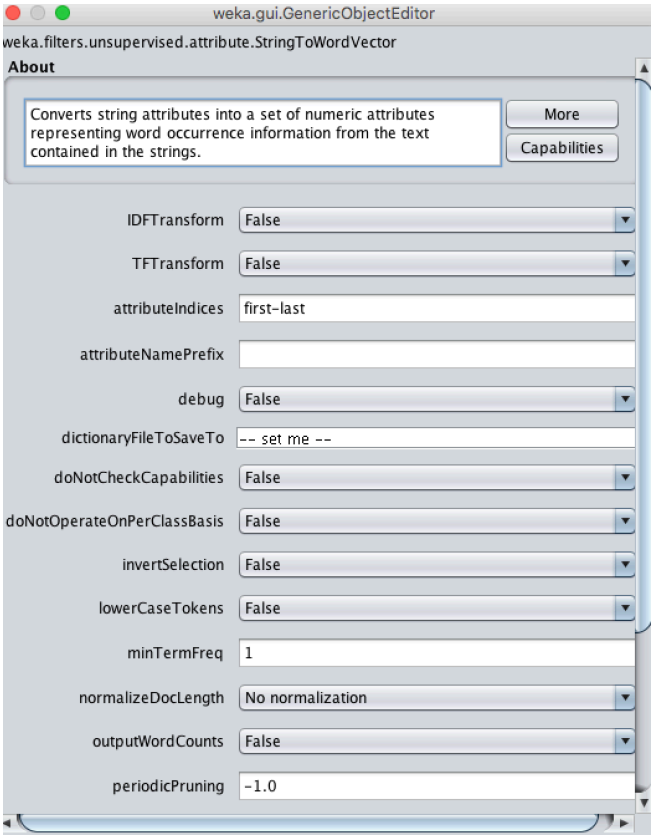
Weka provide an unsupervised attribute filter *StringtoWordVector*. It transfers the text data into bag-of-words, or in the other words, this approach coverts a sequence of words to numerical feature vectors. After those steps, we can save the attributes and relation in an ARFF file by clicking the top right *Save* bottom.



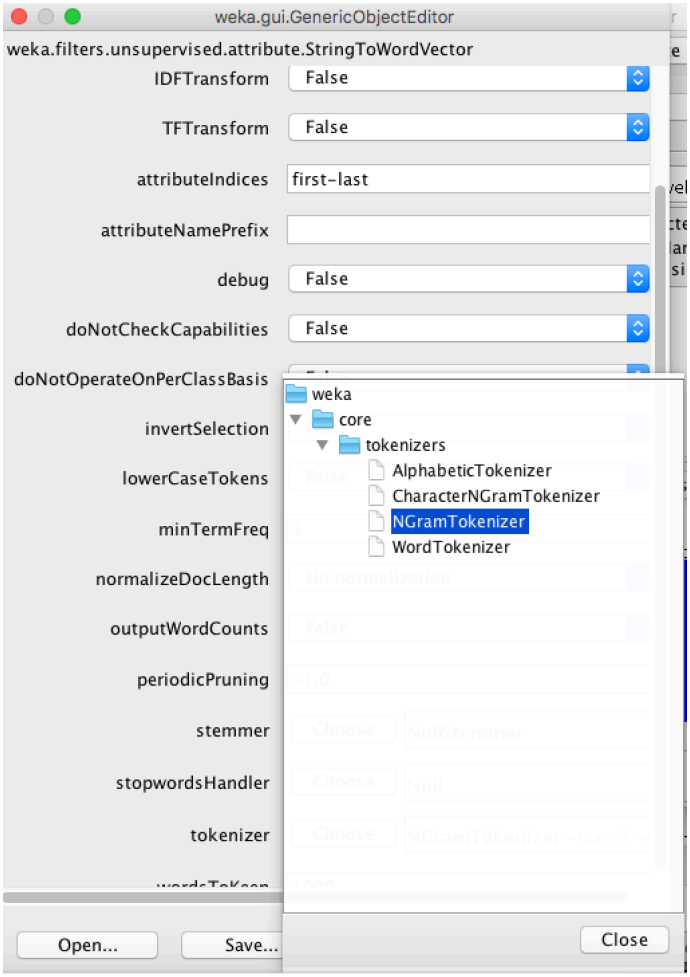
Preprocessing

We can now try the baseline classification without further preprocessing on the dataset. Next to the *preprocess* window is *classify* window. The classifier I choose is NaïveBayes with 5 folds cross-validation. The result for accuracy, precision, recall, F-measure is 59.77%, 67.4%, 59.8% and 57.9%.

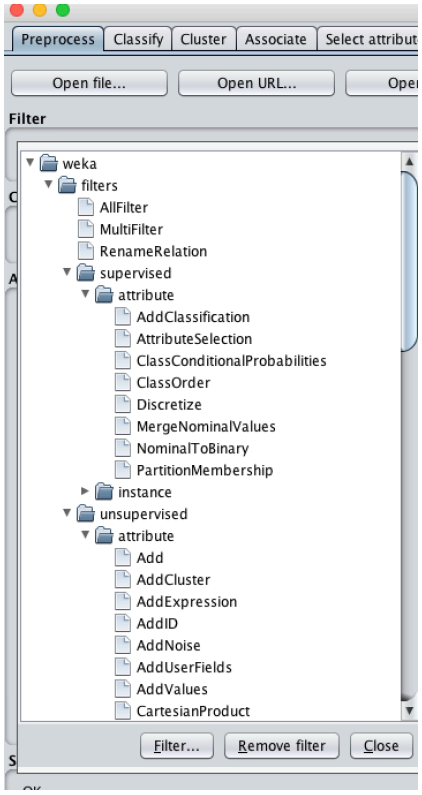
There are various of preprocessing we can try in Weka. Under the filter of *StringToWordVector*, the options include TF transform, IDF transforms, stemmer, *stopwordsHandler,* tokenizer and etc. These options allow us to do word parsing, stop-words removal, lemmatization and stemming, to transform the data based on sentences length and frequency.



However, because the error dataset I have is relatively small, the optimal feature extraction in Weka we can do might be the N-gram. Under the filter of *StringToWordVector,* there is an option for tokenizer, and then pick *NGramTokenizer*.



In *NGramTokenizer*, there are more options. We can require a *NGramMaxSize* and a *NGramMinSize*. With MinSize 1 and MaxSize 5, the accuracy increases to 62.35%. One big problem is that split class has zero correctness. This can be caused by the size of dataset.

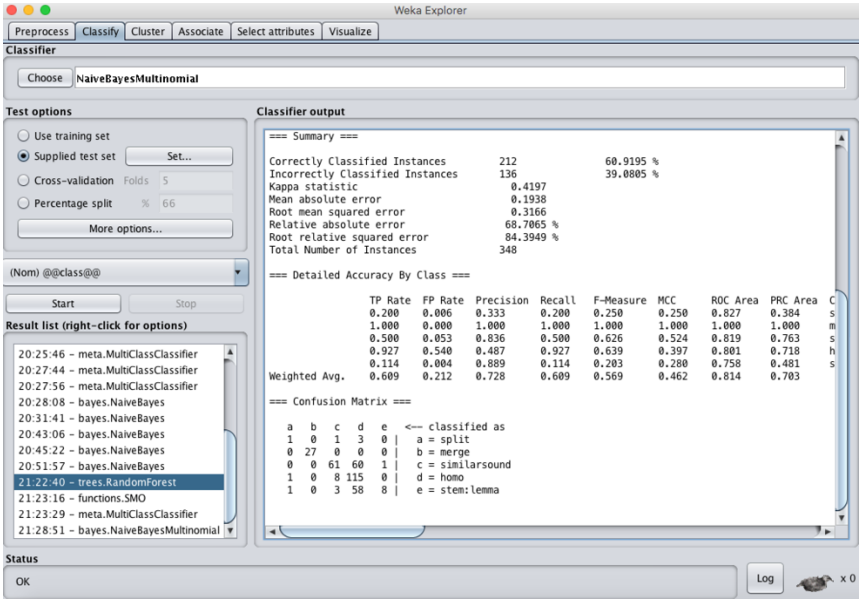


Besides changing the configuration of *StringtoWordVector* filter, we can use more filters. For instance, Weka provides a supervised *AttributeSelection* filter. Sometimes when we have a large dataset with many instances, it could be possible that not all of them are important. *AttributeSelectio* filter allows us to choose an attribute evaluation method and a search strategy. *CfsSubsetEval,* and *BestFirst* are the default evaluation method and search strategy respectively.

After all the attributes is ranked, one can just manually choose the top features and remove the relatively irrelevant features.

Classification

Weka has a variety of classifiers include Bayes, functions, lazy, meta, misc, rules, and trees. For the test options, one can choose to use training set, supplied test set, cross-validation, percentage split, and more. All the classification we run will appear in the result list box and output in classifier output window.



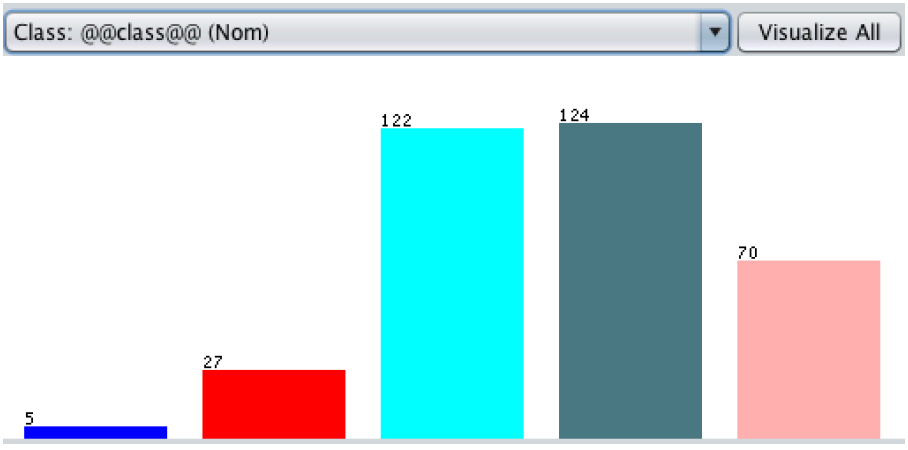
It is easily to compare each result. Weka has a number of performance matrixes. Some of which are accuracy, Kappa statistics, Precision, Recall, F-measure, and confusion matrix. For example, we can see from the confusion Matrix, there is only one correctly identified *split* using Random Forest classifier.

With the baseline system includes the 1-5 N-grams, the machine learning algorithm I chooses includes NaïveBayes, Random Forest, SVM, and meta.MultiClassClassifer. The result is as following:

Classifier	Accuracy	Precision	Recall	F-measure
Naïve Bayes	0.6235	-	0.624	-
Random Forest	0.6092	0.728	0.609	0.569
SVM	0.6322	-	0.623	-
MultiClassClassifier	0.6006	0.605	0.601	0.576

Future Improvement

The result of the error classification is not ideal. From Weka's attribute visualization window, we can see another problem besides the size of the training set. The classes we have is not balanced. It has very few split and much more homophones. The unbalanced dataset has a negative impact on the classifier, such as the unidentifiable precision. We should have a balanced training set with equal classes number in order to improve the classification.



Conclusion

Weka is powerful for doing machine learning analysis without coding. It can be a transition for people to do data science. It supplies a great number of filters, features, and classifiers. Weka also provide a nice data visualization on attributes and very intuitive on result comparison.