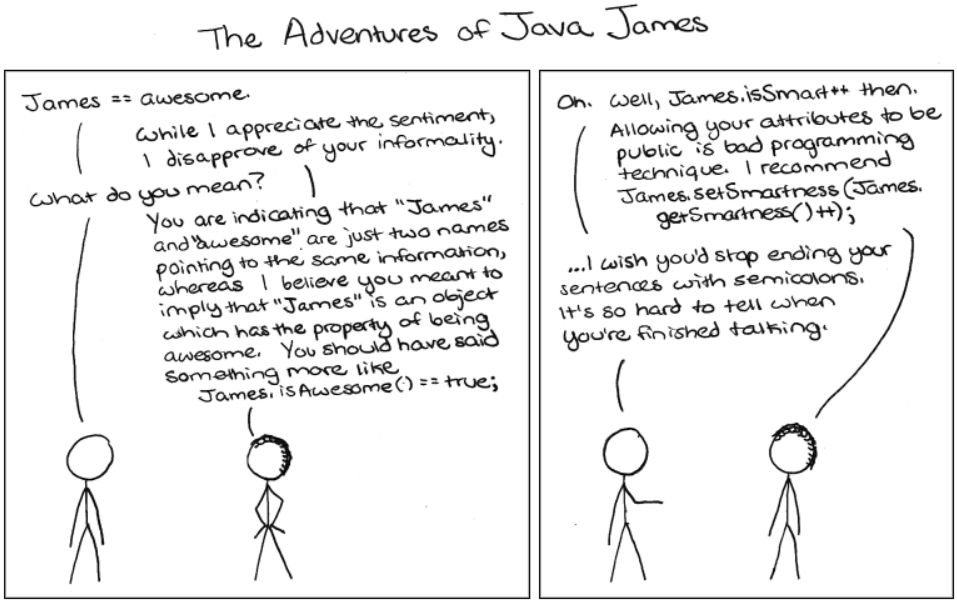
CS451 - Assignment 5
Multiclass Standoff: AVA vs. OVA

Due: Friday, October 18 by 6pm



http://www.decorumcomics.com/comic.php?id=152

Continuing with our trend towards tackling more and more interesting problems, for this assignment we're going to try out a few approaches for multiclass classification.

As always, make sure to read through the entire handout before starting. You may (and I would strongly encourage you to) work with a partner on this assignment. If you do, you must both be there whenever you are working on the project. If you'd like a partner for the lab, e-mail me asap and I can try and pair people up.

1 High-level Requirements

For this assignment you will:

- Experiment with the Decision Tree classifier for multiclass classification.
- Implement OVA.
- Implement AVA.
- Run some experiments comparing OVA and AVA.

2 Starter Code

I have included some starter code at:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs451/assignments/assign5/assign5-starter.tar.gz`

This code includes some new classes/interfaces and some changes to some previous ones:

- I have added a third method to our `Classifier` interface, `confidence`, that takes an example as a parameter and returns the classifier's confidence in the prediction for the example. In practice, we should just require the classify method to return both the prediction and the confidence, however, for simplicity (and ease of backward compatibility) we'll do it this way for now. I have added these methods to the decision tree and perceptron classifiers (dt = proportion of examples at leaf with that label and perceptron = distance from hyperplane).
- `DecisionTreeClassifier`: I have included my implementation of the decision tree learning algorithm. This version can handle multiclass classification and can also handle non-binary features. However, non-binary features are split as examples that have a zero value vs. a non-zero value.
- To allow us to play with more varied data, I've added a `TextDataReader` class that reads in text-based examples. The `DataSet` class constructor has also been changed now to allow us to instantiate it with either a csv file or a text-based file. **Make sure you look at this and change any experimentation code appropriately if you reuse any of it from previous assignments.**
- `ClassifierFactory`: Both the AVA and OVA algorithms can use any binary classifier. To allow us to support this behavior in Java, a common type of class is a "factory" class, which allows you to produce objects of a particular type. The `ClassifierFactory` class allows you to create classifiers so that within your AVA and OVA implementations you can create as many as you need while still giving us the flexibility to swap different classifiers in, should we choose to.

I've generated the html documentation from the starter code and made it available at:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs451/assignments/assign5/doc/`

3 Data

For this assignment, we'll be playing with a new data set. The data set consists of 1945 examples and 20 different labels. The task is to predict the type of wine (e.g. zinfandel) based on the text description of the wine. The data was collected from allwines.com.[1]

The data can be found at:

[1] Thanks to Hal Daume for sharing the dataset.

`http://www.cs.middlebury.edu/~dkauchak/classes/cs451/assignments/assign5/data/`

To be consistent with our previous data `wines.train` has numerical labels for each of the wines. The file `wines.names` has the association between label number and wine. To generate a new data set use:

```
DataSet wineDataset = new DataSet(wineFile, DataSet.TEXTFILE);
```

where `wineFile` is the location of the `wines.train` file.

Take a look at a few of the examples in the training file to help you get a feeling for what the data looks like. When the data is read, each word is considered a feature and the value for that feature is the number of times that word occurred in the text of the example.

4 A Blast From the Past

To get you warmed up and comfortable with the new data set the first things we'll do is run a quick experiment with the decision tree classifier on our new wine data set. On this data set, each internal decision tree node amounts to asking whether or not a particular word occurred on the data.

Include answers to the following three questions in your assignment writeup.

1. Train a decision tree classifier on ALL of the data and set the depth limit at 5. Use the `toString` method to print out the learned tree. Look at the words. Do they make sense? Do any of them stand out? *Include 1-2 sentences describing what you see.*
2. If you just predicted the majority class, what would the accuracy be? This is often a reasonable baseline to measure against (hopefully we can beat this!).
3. On a random 80/20 split of the data, train the decision tree classifier and evaluate both the training and testing accuracy for depth limits ranging from 0 to 50 (for consistency, all tests should be run on exactly the **SAME** split). Include the output of your results. What is the best depth to use? Do you see evidence of overfitting?

5 One versus all

Implement a one versus all classifier called `OVAClassifier` that implements the `Classifier` interface.

- The constructor should take a single parameter of type `ClassifierFactory`. For example, to train an `OVAClassifier` using depth 2 decision trees, we would do the following:

```
ClassifierFactory factory = new ClassifierFactory(ClassifierFactory.DECISION_TREE, 2);
OVAClassifier classifier = new OVAClassifier(factory);
classifier.train(trainDataSet);
```

Notice again the power of the factory. If we wanted to try our approach out with perceptrons, we could simply change the factory line to:

```
ClassifierFactory factory = new ClassifierFactory(ClassifierFactory.PERCEPTRON);
```

- For now, you can just leave the confidence prediction as returning 0.
- To classify an example, this classifier should pick the label of the most confident one-vs-all classifier that predicts positive. If none predict positive, it should return the label of the least confident one-vs-all classifier.

6 All versus all

Implement an all versus all classifier called `AVAClassifier` that implements the `Classifier` interface.

- The constructor should take a single parameter of type `ClassifierFactory`.
- For now, you can just leave the confidence prediction as returning 0.
- To classify an example, we will use the weighted vote based on the confidence of the classifiers. You will need to calculate a running total for all of the labels. For each of your paired classifiers you'll update the total for *two* labels, increasing one and decreasing the other.

http://xkcd.com/892/

7 Experiments

Answer the following questions (in addition to the three above) and put them in a file called `experiments` (pick some reasonable file type). Explicitly label each answer with the question number.

4. On a 10-fold cross validation of the wine data set, calculate accuracies for the following:

   - OVA with decision trees sized 1, 2 and 3.
   - AVA with decision trees sized 1, 2 and 3.
   - Multiclass decision tree (i.e. just by itself) with your best limit found above.

   Put these all in a spreadsheet or other format (you'll have 70 numbers, 10 for each experiment).

   Run a t-test to validate which approach is best. What is the best approach? Is this surprising? Include a short write-up describing your results.

5. Time both OVA and AVA on some reasonable test of the wine data set and measure both training time and testing time (separately). You can use the `ClassifierTimer` class if you'd like or just do it yourself. Do the timings make sense? Include 1-2 sentences describing how you generated the timings and explaining the results.

6. On this data set, what would you say is the best approach to use?

7. Train the OVA classifier with decision trees sized 3 on all of the data. What is the tree for the `zinfandel` classifier (again, use the `toString` method to print it out)? Does this make sense? What words are indicative of the class? What words not indicative of the class? Include a short summary describing your analysis of the tree.

8 Hints/Advice

- Neither the OVA or the AVA class should require tons of code or complicated procedures (my implementations are each a little over a hundred lines of code).
- Make sure you understand how each approach works. If you're fuzzy, come talk to me.
- Both the AVA and OVA approach will involve training a number of classifiers, so you should expect it to take some time to train on this data set. On my laptop, my implementations took on the order of a minute to train on a single split of the wine data.
- For the OVA classifier, you'll need to store one classifier for each label in the data. You can do this fairly straightforwardly using some sort of map (e.g. `HashMap`).
- For the AVA classifier, you'll need to store classifiers that distinguish between a pair of classes. When storing these classifiers, you'll need to store both the the classifier itself, as well as which label the classifier predicts between. There are a number of ways you can do this:

  - If you traverse the pairs of labels in exactly the same order during both training (when you create them) and classifying (when you apply them) you only need to store the classifiers themselves (and not a mapping). However, do NOT rely on a non-sequential ordering of the labels (like a `Set`). Instead, make your own copy of the labels in some sort of sequential structure (like a `List`).
  - Create a unique `String` that stores the information (i.e. which two classes). You can then associate the classifier with this string. You should be able to deconstruct this string during classify time to be able to tell which two classes it represents.

- You will need to create new `DataSet`s during training for both the OVA and AVA classifiers. To do this,

  1. Create a new `DataSet` using feature map from the original data set
  2. Create *new* examples that are copies of the examples in the original data set. There is a copy constructor in the `Example` class that makes this easy.
  3. Set the label of these new examples to be either positive or negative. If you do not create new copies of the examples, when you set the label you're going to lose the original label (that's a bad thing).
  4. Finally, add each new example to the new data set using the `addData` method.

  Notice that for each binary classifier you're learning you're creating a new data set that is a *copy* of the original data set, but with the labels changed (for the OVA, a copy of the whole thing, for the AVA, a copy of two labels).

- These types of meta classifiers can be particularly hard to debug since there is a lot going on. I *strongly* suggest coming up with a very simple data set on your own (say with just a few features and three classes). When training:

  - Print out each of the trees learned and make sure they make sense.
  - When classifying the examples, print out each of the results from the individual classifiers and check that they are doing the right thing. Then, you can check that your aggregation/voting method is doing the correct thing.

9 When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Create a directory with your last name, followed by the assignment number, for example, for this assignment mine would be `kauchak4`. If you worked with a partner, put both last names.

Inside this directory, create a `code` directory and copy all of your code into this directory, maintaining the package structure.

Finally, also include your `experiments` file with the answers from Section 7.

`tar` then `gzip` this folder and submit that file on the submission page on the course web page.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate JavaDoc
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.