Assignment 6

Learning Objectives:

1. Get more practice using LightSide.
2. Think more about implications of feature space design for text classification.
3. Gain experience with error analysis in LightSide

Description:

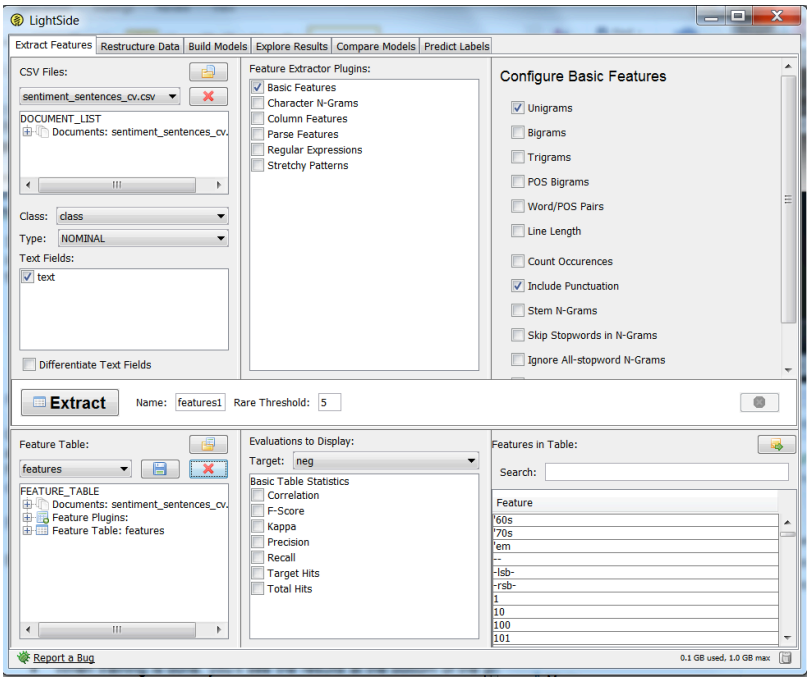
We'll be working with one of the data sets that comes with LightSide. In the last assignment you did a simple classification experiment with LightSide. This time you'll do more intensive experimentation. Refer to the LightSide Manual and the lecture slides.

1. Start with the *sentiment_sentences.csv* you'll find in the *LightSide/data* folder. These are individual sentences showing positive or negative sentiment, from movie reviews. Notice that the file has 10663 instances and that they are sorted by class value.
    - Read the file into a spreadsheet program (like Excel) and randomize the order of the instances – the common way to do this is to add a column with a random value (=RAND()), and sort the spreadsheet by that column.
    - Set aside 1/3 of the data as development data and the use rest as a cross-validation dataset. Save the development set as *sentiment_sentences-dev.csv* and the cross-validation set as *sentiment_sentences-cross-validation.csv*.

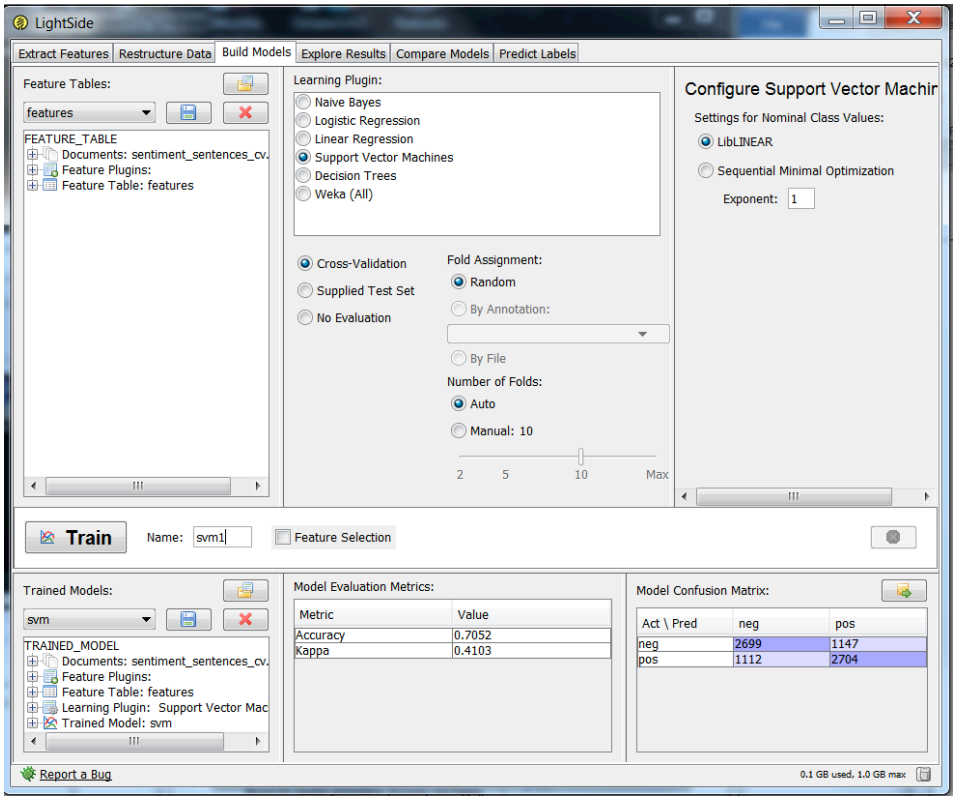
    Files included in answers folder

2. On the first ("Extract Features") panel, load *sentiment_sentences-cross-validation.csv* and configure the panel so that you are using only unigram features. Under "Basic Features", only "Unigrams" and "Binary N-Grams" should be checked (Note that in the current version, you should instead leave "Count Occurences" unchecked.)
    - **Save the unigram feature table as an ARFF file called *baseline.arff*.**

    File included in answers folder

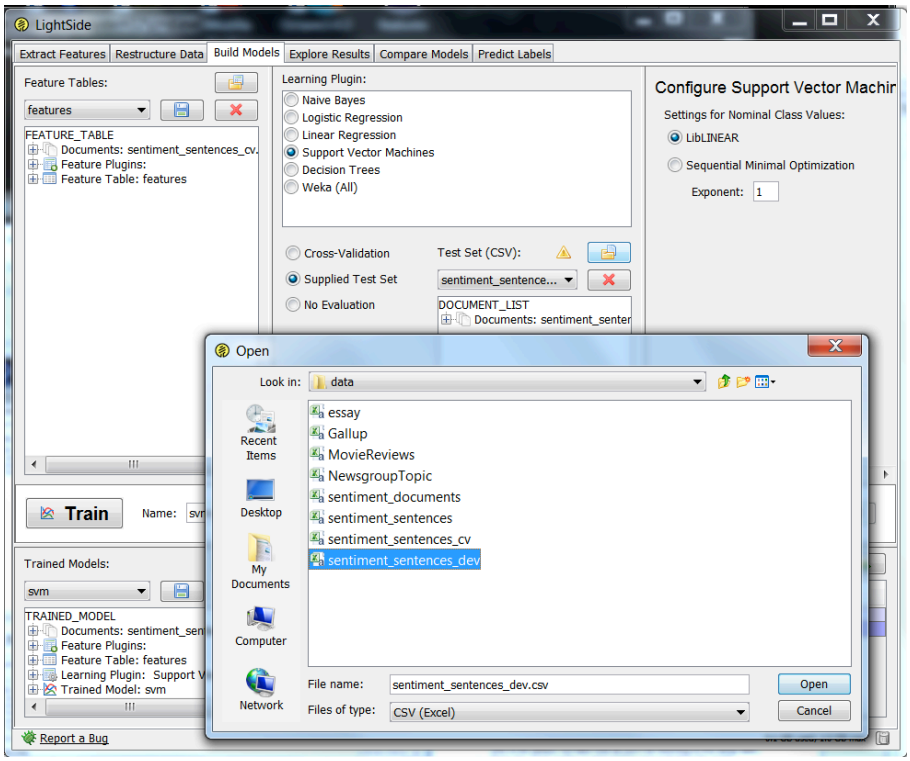


3. Build a baseline model.
    - On the “Build Models” panel under “Learning Plugins”, select the LibLINEAR SVM (under “Support Vector Machines”) as your machine learning algorithm.
    - In the center pane, make sure you’re set up for random 10-fold cross-validation. (this is the default, but take note as you’ll be returning to this setup later)
    - Press the “Train” button to start training and evaluating the model.
    - When training is done, you’ll see the results at the bottom of the pane. **Record the baseline performance (Accuracy and Kappa), under “Model Evaluation Metrics”.**



Baseline performance is 71%, with a Kappa of .41.

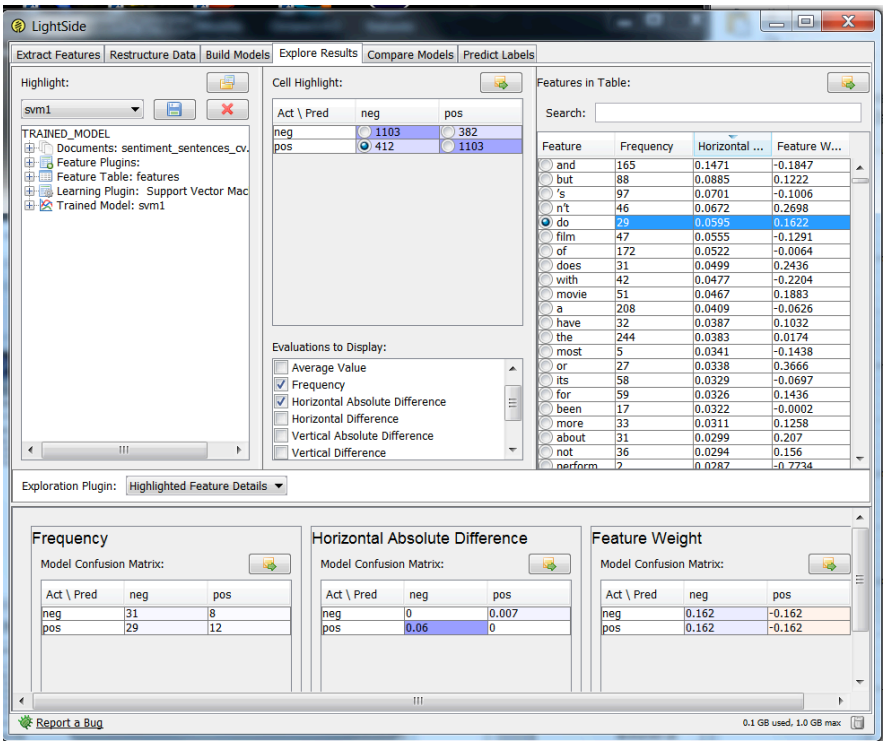
4. You're going to prepare to do an error analysis, to determine where the machine learning model is making mistakes. We want to avoid "cheating" by analyzing the training data directly, so you'll train on the cross-validation set, and test on the development set. You'll perform the error analysis on the test results from the development set.



- Under the center pane of “Build Models”, set up the evaluation to use the “Supplied Test Set” option. Load the development set as the test set.
- Press the “Train” button. The result should be similar to what you saw when you did the cross-validation, but it won’t be exactly the same.

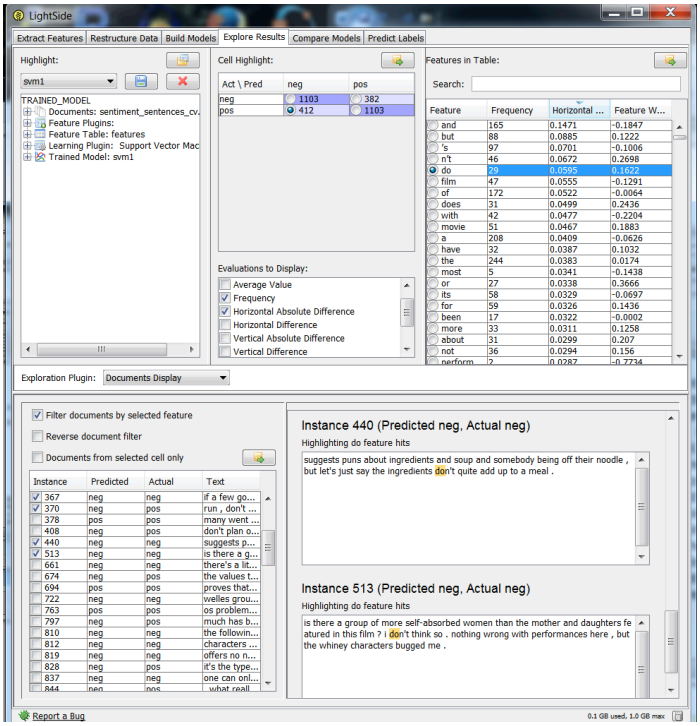
When I did this I got an accuracy of 74% and a kappa of .47.

5. Switch to the “Explore Results” panel to do your error analysis on these test results.
   - Make sure the second trained model is selected in the top-left pane.
   - Use the error analysis methodology demonstrated in class. This includes using horizontal and vertical comparisons to identity problematic features, then using additional tests to try to identify the reasons why these features are problematic.
   - The Explore Results pane (including all of the built-in error analysis metrics) is described in Chapter 7 of the LightSide manual.
   - **Describe the problematic features you identified, and explain how you determined they were problematic.**



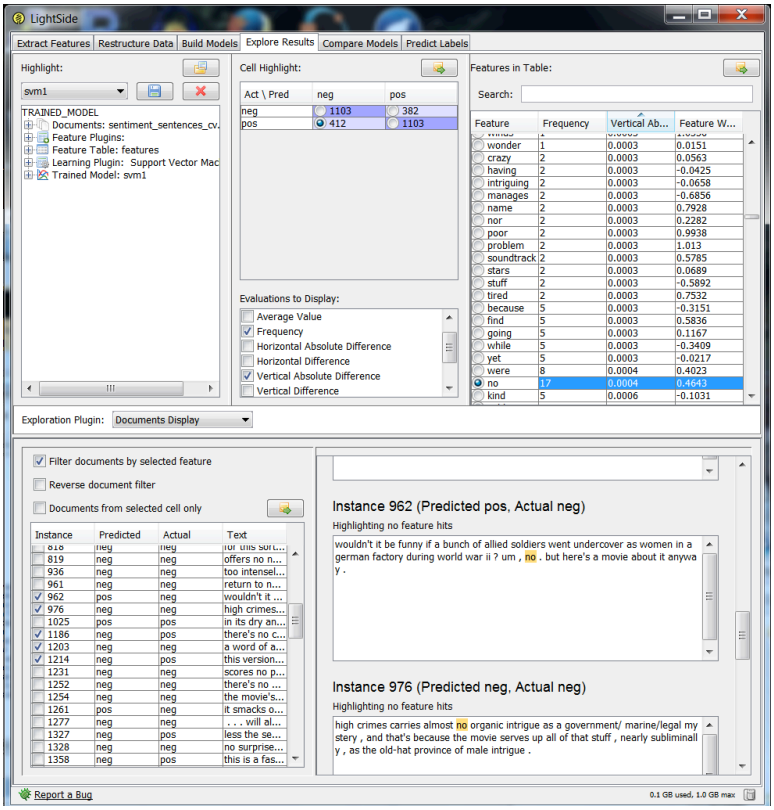
I went to the Explore Results panel. The most recently created model was selected in the upper left. Looking at the confusion matrix in the top/center of the screen, I decided to choose the bottom left error cell because it was the bigger one. I selected that, and the list of features came up in the Features in Table panel.

I started with a horizontal comparison. In Evaluations to Display I selected Frequency, Horizontal Absolute Difference, and Feature Weight. Then I clicked the column heading for Horizontal Absolute Difference to sort. I clicked twice so that the highest values would be at the top. Looking at the sorted list, “n’t” looked like the most promising combination of frequency, horizontal difference, and feature weight, but since we went over that one in class, I decided to select “do” to give you another example. Looking at the frequency table at the bottom left, I can see that the distribution of “do” is such that it made sense for the model to treat it as negative, although we can see there are a substantial number of cases also where it is in positive examples.



One thing I noticed while poking around is that LightSIDE breaks “don’t” into “do” and “n’t”. As I looked at both positive and negative instances predicted as negative with “do”, more of the really negative ones had “don’t”, although there were some positive ones with “don’t” as well. Now that I see that “don’t” is getting split up, I think LightSIDE might be having trouble distinguishing between those two. Once they are split up, the “n’t” it wouldn’t be able to tell whether it came from “don’t”, “can’t”, “aren’t”, etc., which might have different connotations in reviews. “don’t” sounds like a negative recommendation, but other “n’t” words might not be. That seemed like an important problem.

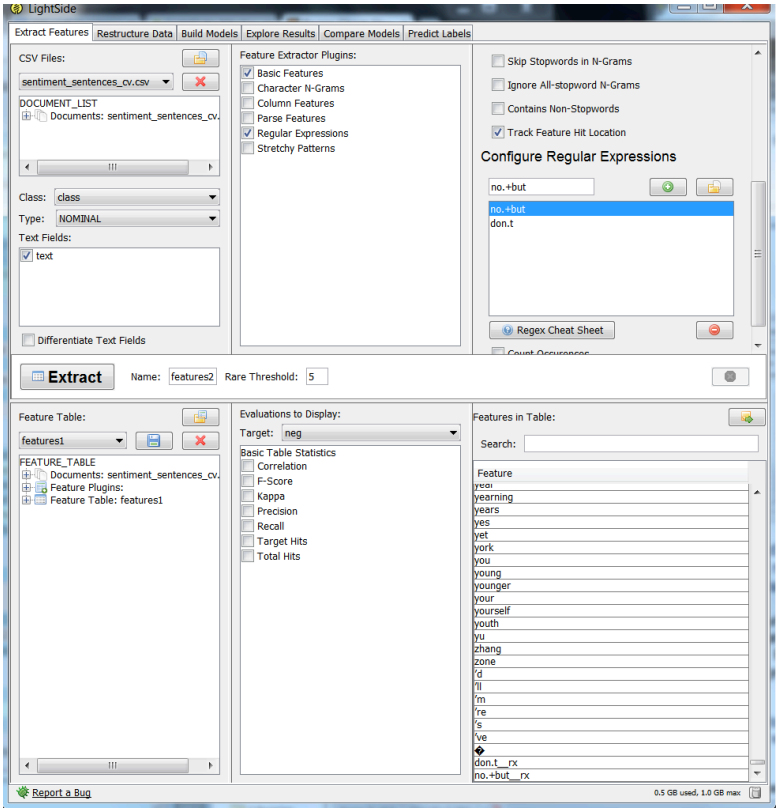
I then went on to do a vertical comparison. I unchecked vertical absolute difference and checked instead Vertical Absolute difference. Then I sorted by that column so that smaller numbers were at the top of the list. I then scrolled down to where there were some entries with high frequency, relatively small vertical distance, and high weight.



"no" looked like a good choice. And as I poked around in misclassified examples with "no" I noticed a lot of "no ,,, but". That seemed like the kind of thing that could be easily missed in a unigram space, and easily fixed.

6. Based on what you learned from the error analysis, come up with some ideas for new features that you think might help the machine learning algorithm overcome the errors you found. **Explain your proposed features and why you think they will solve the problems you noticed.**

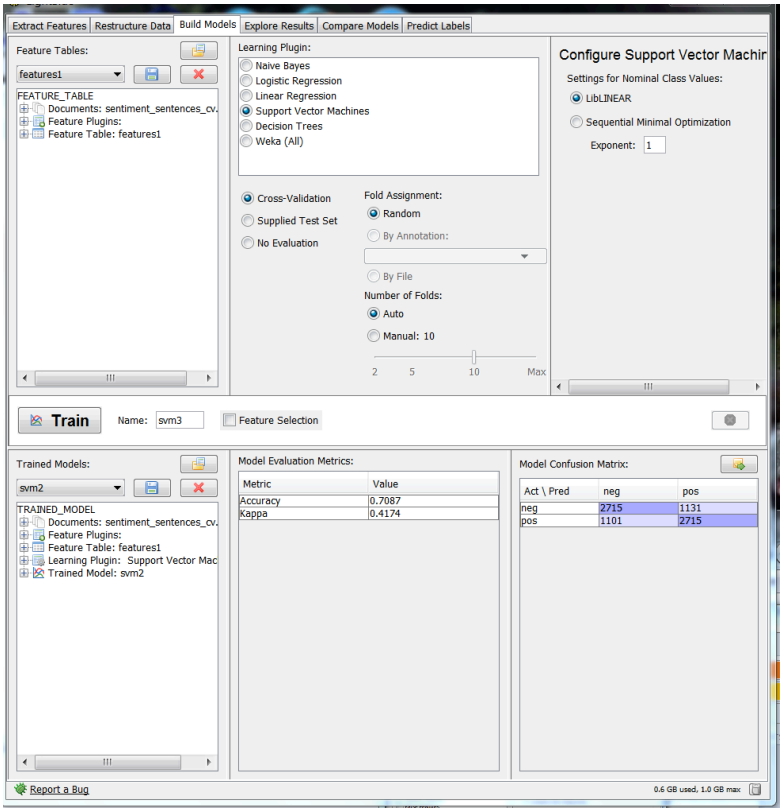
It seemed to me the "do" + "n't" problem and the "no…but" problem could both be addressed with regular expressions. This would allow me to find places where these two strings are one after the other, in the case of "don't" with nothing in between, and in the case of "no…but", with an arbitrary amount of text in between. So I decided to test that.



7. Test whether your ideas are effective at addressing the problems you found. You will evaluate the effect on the cross-validation set.
    - Return to “Extract Features” panel and set up the feature extraction using the ideas you listed under #6. See Chapter 4 of the LightSide manual for more information on advanced feature extraction. In addition to the options under Basic Features, Regular Expressions may be especially useful.
    - **Extract the features and then save the new feature table as an ARFF file named *final.arff*.**

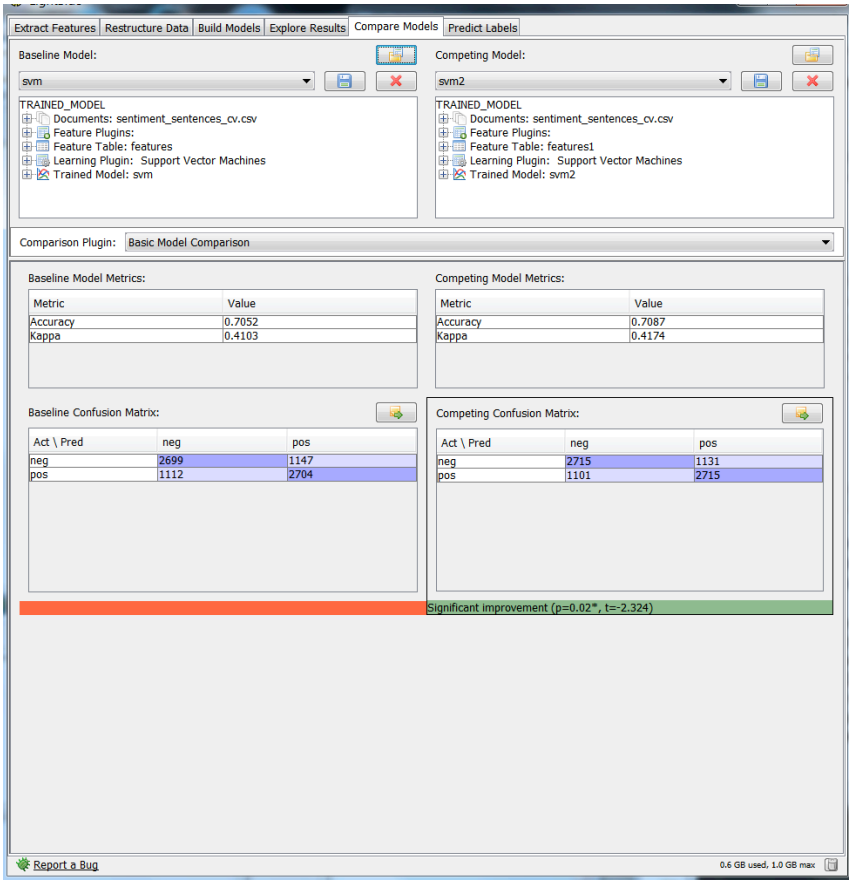
      See screen above for extracting the features. File included in answers folder.

    - Return to the Build Models pane. Reset the options in the center pane for random 10-fold cross-validation.
    - Train and evaluate a cross-validated model to test the performance of the new feature space. **Record the new performance values (Accuracy and Kappa).**



Here we see we have 71% accuracy and .42 kappa. It's just a slight improvement over the baseline, which is maybe not that surprising since the solution was very specific.

8. Compare the results.
    - Switch to the "Compare Models" pane. Make sure the baseline model is selected on the left and the final model is selected on the right. See Chapter 8 of the LightSide manual for more information.
    - **Test whether the performance difference is statistically significant.**
    - **Try to determine (and describe) the kinds of instances on which the two models perform differently.**



Surprisingly, although it was just a small improvement, it is a significant improvement over baseline!! We can see that we have indeed reduced the number of errors in our target error cell. And we have increased the number of correctly classified examples in both diagonal cells. ☺

Deliverables:

Turn in *baseline.arff* and *final.arff*

Turn in answers to the questions in 3, 5, 6, 7, and 8.