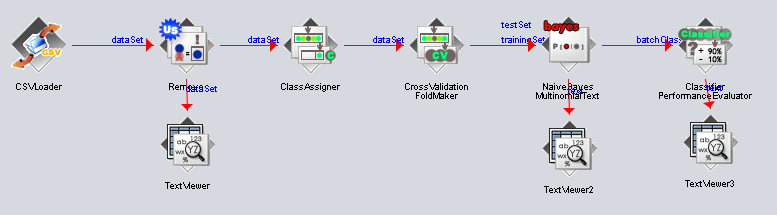
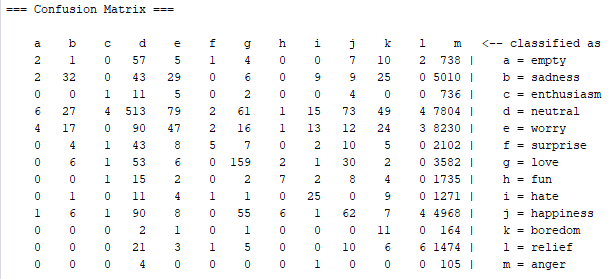
# Homework Writeup

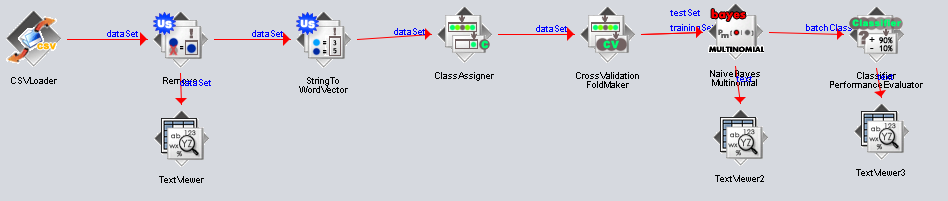
## Part 1

For this part, we were supposed to do sentiment analysis using the provided text\_emotion dataset that has the following structure (tweet\_id, sentiment, author, content). The image provided below is the structure that was set up following the direct instructions given.

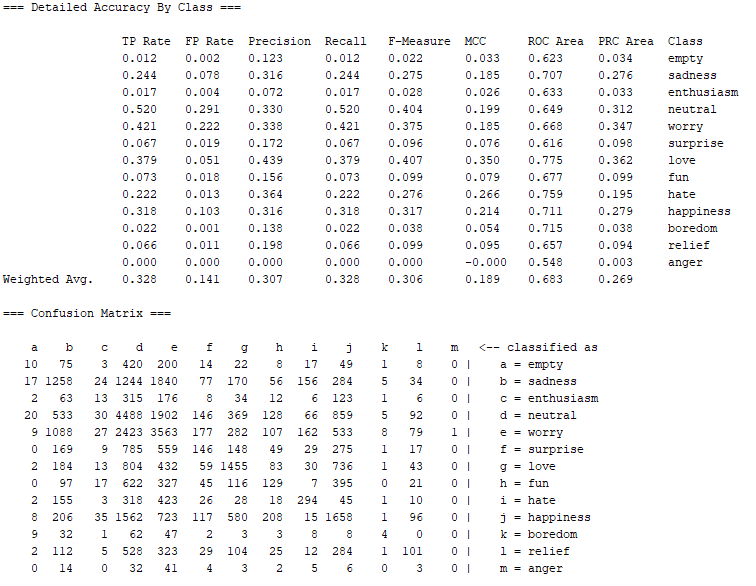


There is a few things wrong with this model. For one, each run takes over a minute and half each time and slows the rest of the PC to a crawl. This wouldn’t be that big of a problem if the output gave something that was desirable. The model was only capable of classifying 975 correctly, out of the 40,000 instance dataset. This is approximately 2.4%... This score is just beyond horrible. Below is the confusion matrix

One thing of interest, from this output is that the model somehow consistently classified most objects as anger. Which is extremely peculiar since the dataset least common sentiment was anger. This was checked quickly by using the Weka explorer. Since we are already here in the Weka explorer, lets run some basic tests, like ZeroR and OneR. The ZeroR decided that neutral was the best sentiment, given that neutral had 8638 counts. This received a classification score of 21.595% correct. Weird to think this is the baseline, but it makes sense due to the complicated nature of the subject. As for the OneR, this created a massive amount of subrules that related keywords to sentiment (like funeral -> worry). The end result was 929 classified and 2.325%... This score is higher than our model from earlier. Why is that? Well, to be fairly honest, I have no idea. I spent an entire night double checking every parameter, every link, and anything else. I tried classifying each item as a different type (nominal, string) and regardless, the highest score I got was when I classified sentiment as nominal and content as numerical, the other two don’t need to be labeled since they are being removed. Regardless, I wanted to score something better than that. So I realized, why dont I take the best of both worlds? This modification can be seen below.



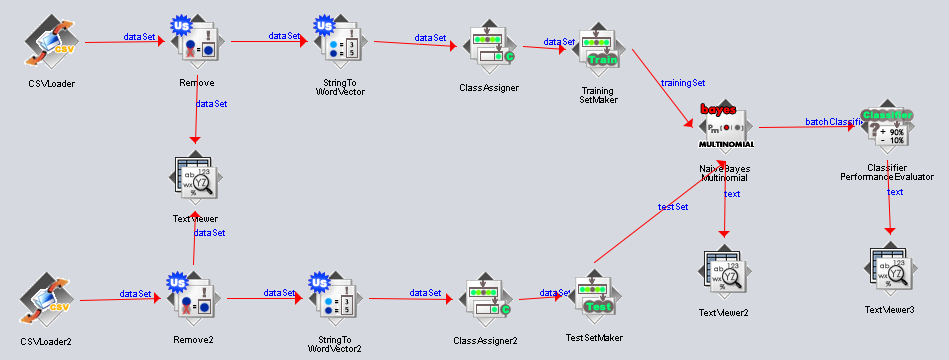
So if you remember from the tutorial provided by the professor that used NaiveBayes, she converted all her strings to vector. But the professor mentioned that NaiveBayes shouldn’t be used, so instead we used NaiveBayesMultinominalText. Since I am in this case converting the data from strings to vector, I no longer needed to use the previously mentioned method, but instead I used just a NaiveBayesMultinominal. And then, this is where the magic happened! The run literally took less than 5 seconds, cuz god bless vectors instead of a giant bag of text. The data scored 13,119 instances correctly, which is 32.7975! THIS SCORE IS HIGHER THAN WHAT THE PROFESSOR ACHIEVED, that and its so much faster. The kappa stat was 0.189, which is still regrettably staging that this model is still basically randomly guessing.

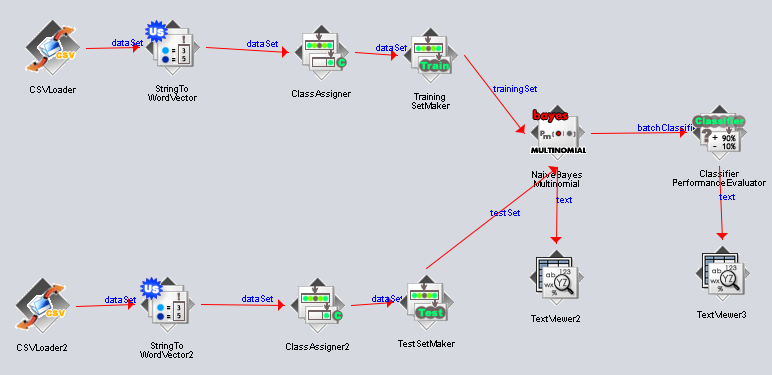


Based on the ROC area, the best classes were love (0.775), hate (0.759), boredom (0.715), happiness (0.711). The one I was to point out that is unique in this area is boredom. Boredom only has 179 instances. Every other item had over 2000 instances, with happiness at max 5209. The instances where boredom is mentioned, it is clearly more formulated, with “bored” and other synonyms being visible. The most command predictions here were neutral and worry, which are also the most common instances found too. What is almost funny is how the dataset now only ever guessed anger once, which was incorrect. I even tried to “fatten” the dataset in a modified version. Regardless, I am just glad this system worked.

## Part 2

Even though this part of the assignment should have been super easy (based off the instructions) this is where I had the most difficulty. So below is the modified version of part 1 to work instead on a new test dataset, generated by me. This was atrocious due to the “training set and test set are structured differently.” Over and Over I got this message no matter how similar the dataset was. At one point, i straight up opened the original file, copied everything from it, and pasted it into a new file. Even though the two files were an exact match in visual text, it somehow differed internally. You can see all the hell variations that it took for me to figure it out in the writeup zip.

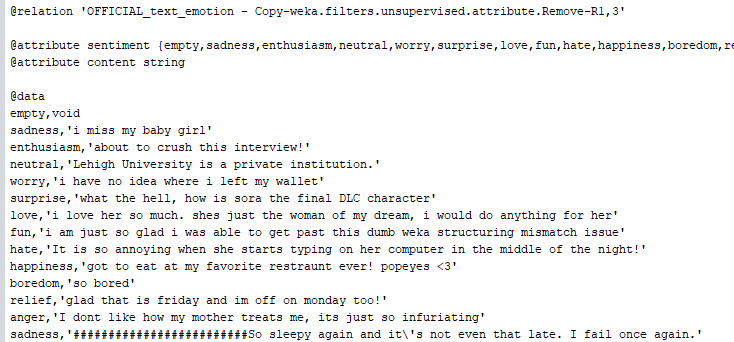


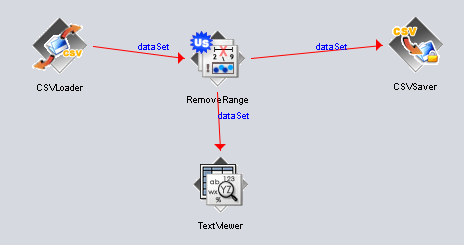
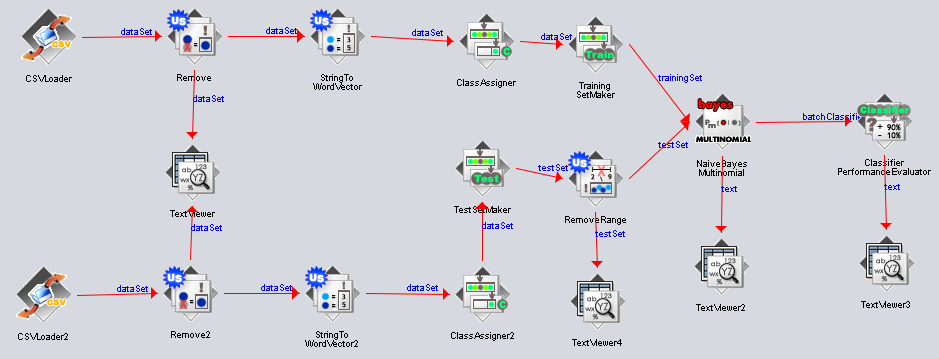
So this method was a deadend. Then I realized, HMMMM maybe the output of the two will be same if I just saved the results outwards and use that as the input next time. So I modified the run above to link the Remove module to a CSVSave module. Those exported modified files were then shoved back in to the model on the left. AND GUESS WHAT, IT STILL DIDN’T WORK. At this point, i was just desperate. IF you look through all the failed attempts folder, you will see that I even tried replacing every “ and , with my own computers “ and , which did nothing. Then after a bit of research I found this magical statement. 

“

[One of WEKA's fundamental assumption is that the structure of the training and test sets are **exactly** the same. This does not only mean that you need the exact *same number* of attributes, but also the exact *same type*. In case of *nominal* attributes, you must ensure that the *number* of labels and the *order* of the labels are the same.](https://waikato.github.io/weka-wiki/faqs/why_do_i_get_the_error_message_training_and_test_set_are_not_compatible/)

“

ARE YOU SERIOUS? So then I made sure that each nominal value appeared in the same occurrence as the original dataset. After making sure this wasn’t the issue, I gave it my final last college try and magic happened. What I did was I copied the file (in file explorer) and pasted it within the same directory, which gave me the original and original-Copy. Then I went through the original copy, making sure I don’t do anything by cut and paste, I cutted the first instance of each sentiment, in order, and pasted it at the top of the file (so the first 13 instances would contain an instance for each sentiment). What was excruciating about this process was hitting the run button after each modification, making sure that the two structures still matched. This is why I became so dearly grateful that my modified model runs in seconds versus minutes. Once I had the 13 instances working (which still contained outdated information instead of mine), I then modified each author and content, one at a time, running the entire model after each change. This all took about 1-2 hours. Which was hell. Then I modified the pipeline to include a RemoveRange so it removed all the other instances except the first 13, which were the ones I excruciatingly copied and modified. Then… We finally had everything matchup. 



I do have to preface, the only reason I figured out the above idea, just modifying inplace once it was done converting to a hex was from messing around with exporting the saved result

←-----



So with my modified instances I was able to get a 6/13 instances or 46%. This actually was a pretty decent run. Looking at it, it seemed to nail relief, happiness, love and worry (with a full 1 precision). There were a few others that classified a right one, but also classified as a wrong one such as sadness and neutral (which was by far the most common). Overall the results here aren’t bad for a generated test data. Granted, it wasn’t able to guess boredom from a tweet that said “so bored,” so take it with a grain of salt, but another grain of salt, it guessed empty, which isn’t the worst guess either.

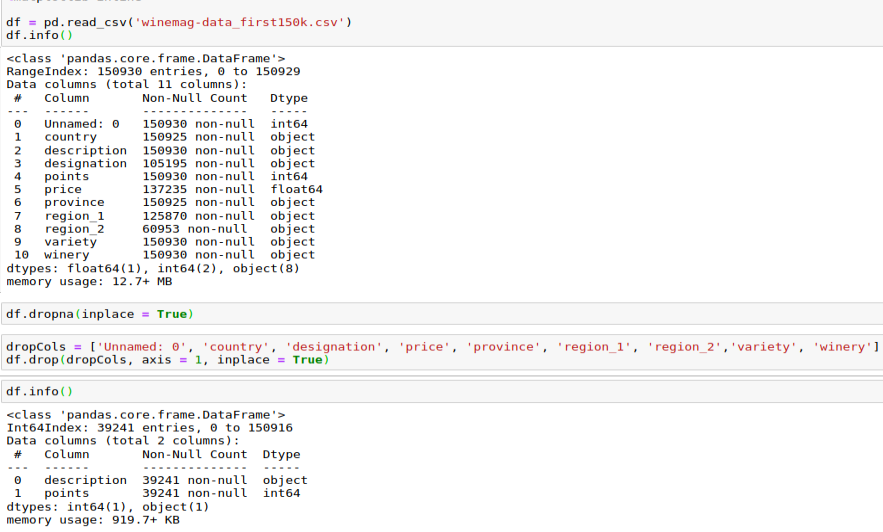
## 

# Extension Writeup

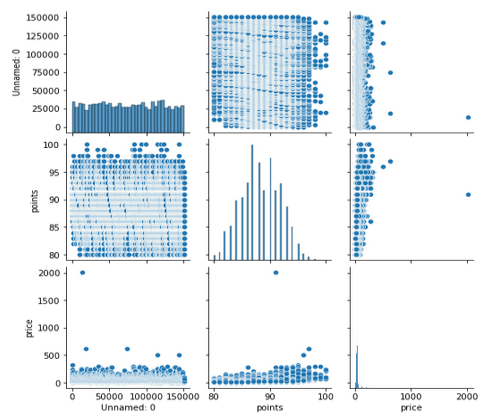
### The Dataset

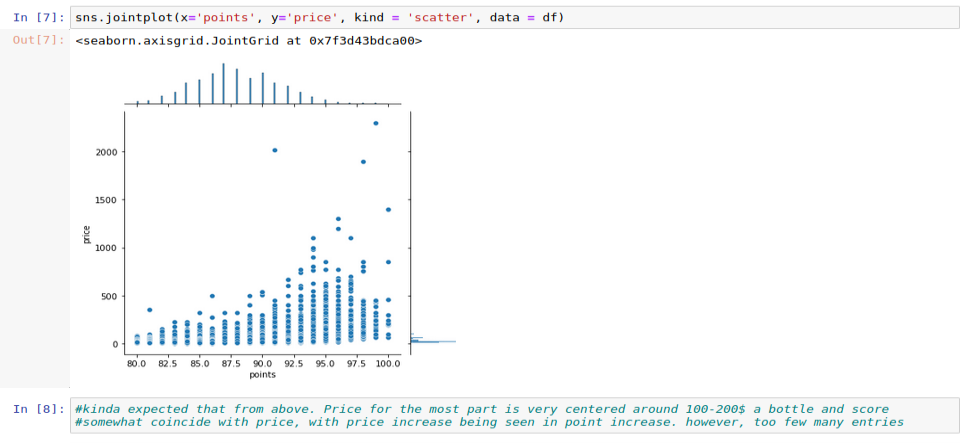
For this extension, I choose to move over to python/jupyter following the Wine Review dataset (<https://www.kaggle.com/zynicide/wine-reviews>). This dataset has about 150930 instances. I decided this set was a bit large as it was and decided to drop any na in place (this is done before dropping any unnecessary attributes). Once that was done, our working dataset was cut to 40,000 entries. A bit easier dataset to be working with.





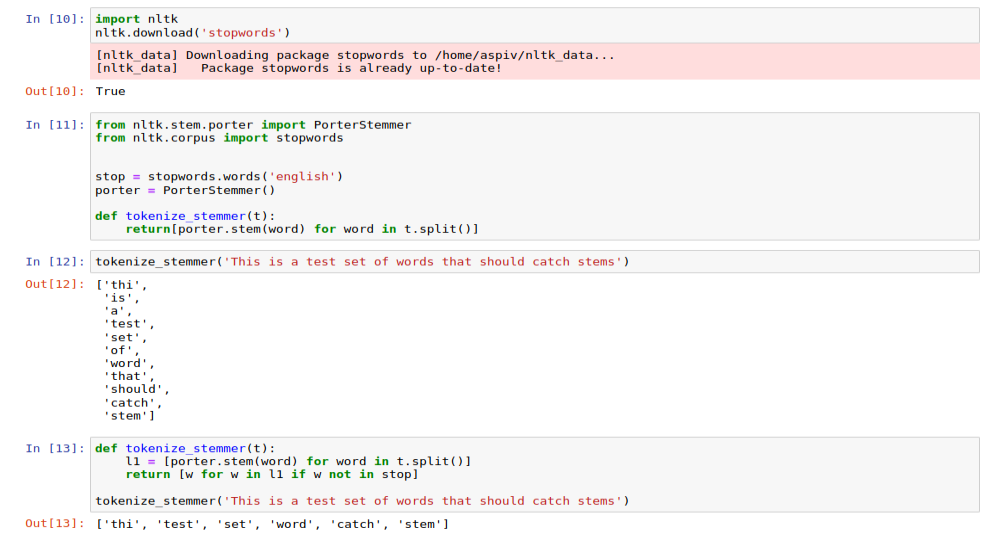
Before moving onwards, I wanted to see just some numerical characteristics of the dataset.



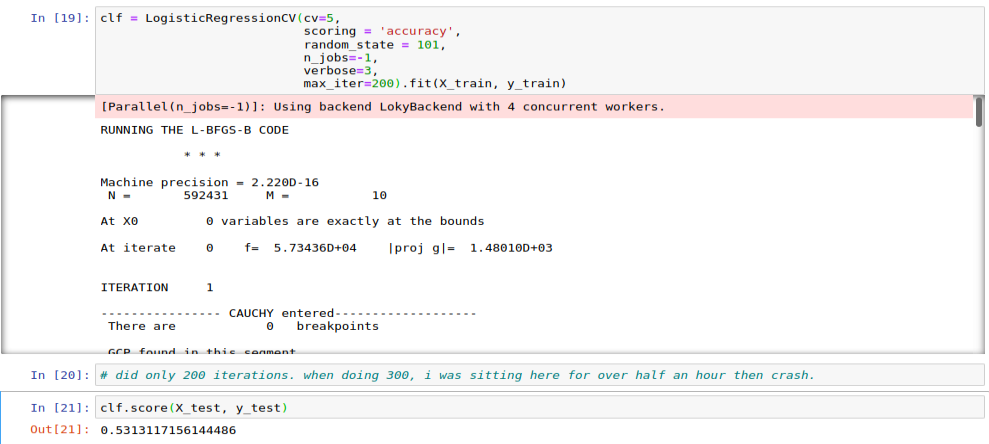
It wasnt hard to see that most elements werent of focus other than price and point. Doing a closer up investigation… 

### Sentiment Analysis

Before we move on, since we are working with Logistic regression, we want to drop dead words and use only word stems. We do so by importing nltk and downloading the specific languages stop text. Then we wrote our own tokenizer, I modified mine to do such in one step.

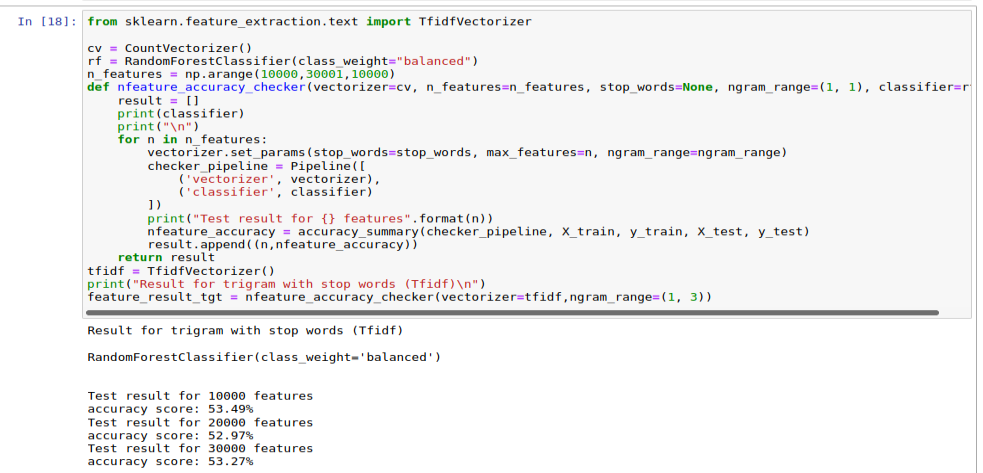


Once this was done, we move onto taking Vectorize the remaining instances and columns (points and description). The tutorial suggested 300, but that processing time was too large and cause crashes. So sticking to just 200, we try to score our test set and the score we got wasn’t all too great…



However, when just scoring the training data, it wasn’t hard to see that the data was too overfitted. 

The generated confusion matrix was a bit too large to even analyze properly. So to compare our dataset, I followed a tutorial using Random Forest Classifier to do the john and trying out 10,000-30,000 features.



This is fairly insync with the scoring provided by logistic regression model. Sadly, I couldn’t get enough time to score precision and recall on the dataset, given that jupyter just didn’t want to stay open.