Tweets Sentiment Mining

Classification of tweets into +ve or -ve sentiment using a Naive Bayes supervised learning classifier.

Kaushal Hooda, 2012CS10228

Methodology Used:

I have used a very minimal classifier based on the Naive Bayes algorithm.

The steps I have used are :

1. Divide the training set into ten parts.
2. For each of the ten parts, I tokenised each tweet.
3. Used each of the tokens for training the classifier.
4. Did a ten-fold cross validations (using one tenth as test set, rest for training)
5. Accuracy was measured as : no. of correct predictions\no. of tweets. The overall accuracy of the model was taken as the average of accuracy for all the evaluations in the ten-fold cross validations.

Effects of various factors on accuracy:

The main points that I tweaked were the tokeniser, the equation used to check the probability of a given tweet in a category, feature set and smoothing. The final best accuracy I got was 78.7 %

Tokeniser :

I tried out three different tokenisers, the one I made for the first assignment, the NLTK tokeniser (http://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.punkt.PunktWordTokenizer), and a twitter specific tokeniser from http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py

Interestingly the NLTK punkt word tokeniser resulted in the most accuracy, even though the other two were designed with twitter specifically in mind - though the difference was only around 2% or so.

Playing Around with the Bayes equation:

I tried to work with counts instead of probabilities (i.e., compared the number of times the tokens in the tweet occurred in positive vs. negative tweets). This however reduced the accuracy by ~12%.

Working in the normal space vs. working with the logarithms of probabilities had no effect whatsoever. (except that log gave errors without smoothing), so I have stuck to using probabilities without taking log, to avoid extra calculations.

I also skipped multiplying the tweet probability with the class probability, since probability of all classes is equal (ie, 0.5).

Features :

I simply took each individual token as a feature. Since Naive Bayes is quite fast, this did not have any real impact on performance. Training time was ~2 minutes. Similarly, while evaluating, since I use the default python dictionary (based on hashmap) which gives O(1) access time on average, a smaller feature set did not alter the running speed. Further, removing sparser words - ie, words that occur only once or twice in whole training set, did not have any affect on the accuracy.

I might add that I tried to handle negations (i.e., replace word by NOT_word for tokens that occur after a negation). For this, I used the following pseudocode:

```
bool saw_not = False
if saw_not :
      word = 'NOT_' + word
add_to_model(word)
if word is punctuation (',', '.') or negation (not, isn't, wasn't, doesn't):
      saw_not = not saw_not
```

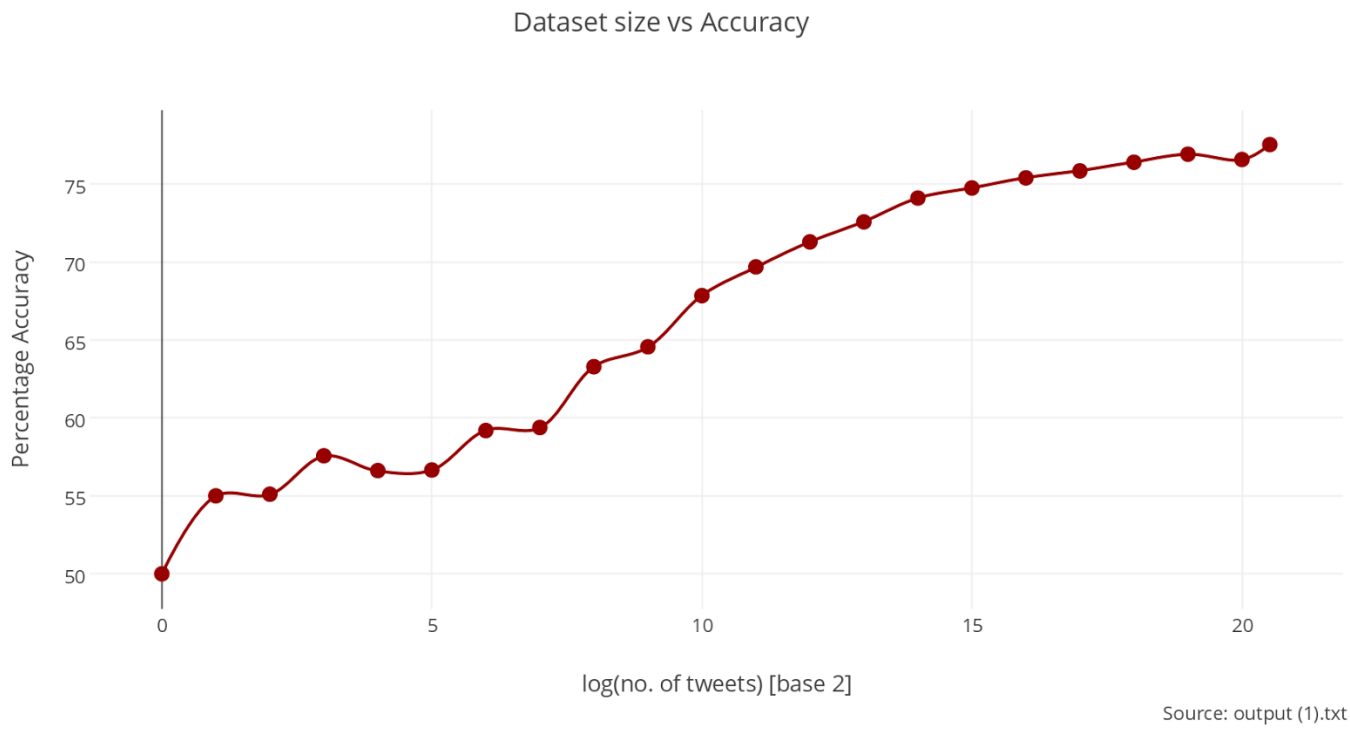
But this didn't result in any increase in accuracy. Thus I took the simplest features - all the words as is.

Smoothing :

I used add - 1 Laplacian smoothing. The effect was most pronounced from this, increasing the accuracy by about 15%. However this improvement was observed only while using all the words as features. If I used only words that occur more than a certain number of times - say 4, then there wasn't that much change - hardly 2-3 %.

Effect of training dataset on accuracy (Research Question)

Once I had settled on the method for training my model - 10-fold cross-validation, NLTK tokeniser, all words that occur at least twice as features, add - 1 smoothing; I evaluated the model generated by varying amounts of training data. For this I used the entire training set as the test except the actual data used to train as the test set. The graph of the same is shown below :



Source: output (1).txt

The plot starts out at 50% accuracy - as expected, since without any data it will be right around half the time, as half the tweets in the set are positive and half are negative. The accuracy increases as training set grows, but not quite monotonically - there are slight fluctuations in the data especially for smaller subsets. This could be eliminated by repeating the test on multiple subsets and taking the average.

The accuracy gains leveled off after the training set had become a few hundred thousand in size, maximising at about 78% accuracy. The plot is similar to an exponential decay curve.