Information Retrieval

Programming Assignment 4

Daniel Erenrich & Daniel Rosenberg

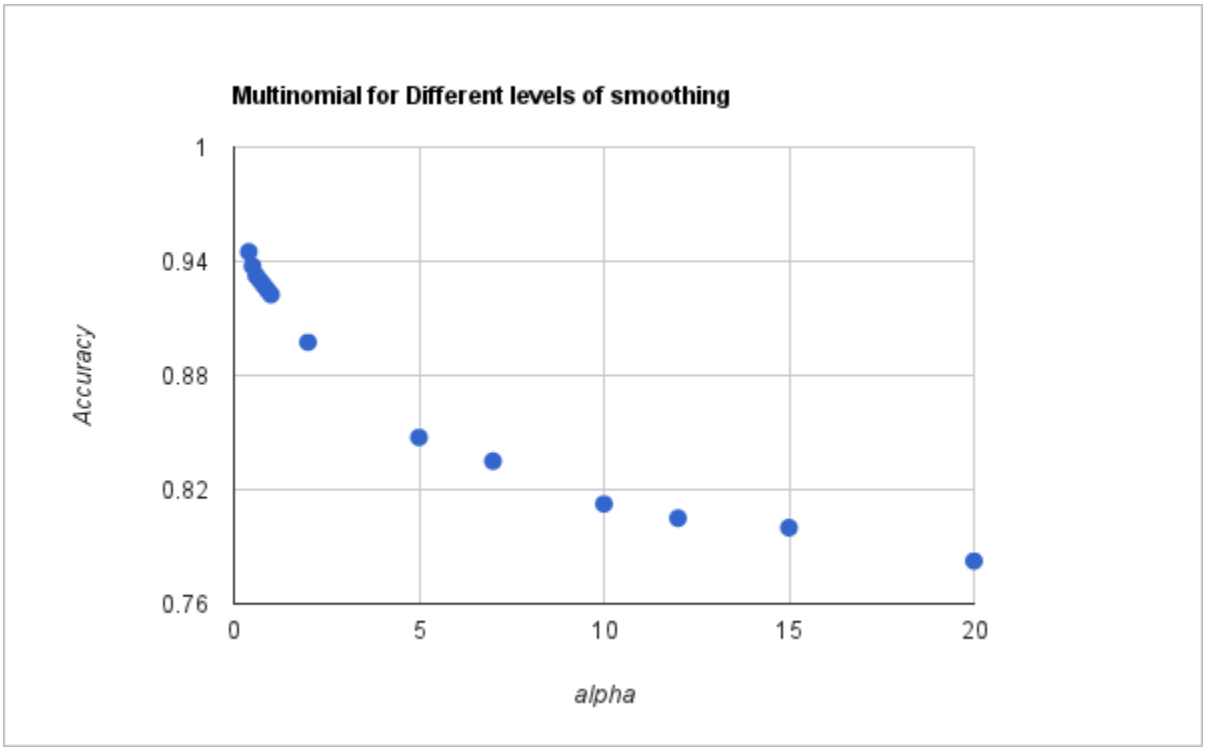
Multivariate and multinomial classifiers

One design decision that we made right from the beginning was what to do with the body and subject fields. Perhaps optimally one would train them separately however for simplicity wedecided to simply add the fields together and pretend they were just a single field. Since the body's are much longer than the subjects this has a (perhaps unwanted) result of decreasing the impact of the subject in the classification process, but it is the decision we made.

One problem with multivariate classifier is its speed of operation. We have to iterate over the entire dictionary for each document and evaluate which of them are present and account for the probability mass for each. The dictionary is quite large (hundreds of thousands of terms) and so this operation takes about 4 seconds per document. The multinomial classifier is only concerned with terms that are actually present and so its runtime is proportional only to the length of the document itself. One design decision one can make when assessing the probability of a word being in some class for the multivariate model is whether to use a dictionary built of words found in that class's documents or the complete dictionary. The latter is probably more correct but the former was considerably faster. Empirically we found little difference in accuracy and so we went with the faster option.

	Multivariate	**Multinomial**
Accuracy	0.915	0.9375 (for alpha = .5)

For the multinomial classifier, we found that adjusting the alpha level for our smoothing greatly affected our accuracy. Smaller levels of alpha led to greater accuracy, although this comes at the cost of overfitting. From the plot below, we find that we take losses up until around an alpha of about 15 when we were around 80% accurate. From our chi2 results, this seems to be around where our out of sample performance is.



χ^2 feature selection

We ran chi-squared on the presence of words within a particular news group, versus presence in all other newsgroups. We sorted these by chi-squared values, and chose the top 300 from each category as our dictionary. One immediate result of this was that our binomial algorithm ran much faster, as its main limiting factor was the large dictionary it was working with. We also showed a slight decrease in performance, although the decrease was pretty small, especially with respect to our decrease in runtime. Our overall accuracy using words appearing in the top 300 most important words in at least one newsgroup was .87

.

Cross validation

Running cross validation gives us a much more accurate picture of how well our algorithms are performing, as we test on data that is separate from our training data. We randomized the order of the messages, and then divided them into 10 groups for cross validation. One concern we had is that we might over represent one of the sets in testing or training, but the large sample sizes should cause any such irregularities to average out. Our results were surprising, as we see our multivariate model outperforming our multinomial model, suggesting that multinomial was significantly overfitting the data previously.

Cross validation	**Binomial**	**Multinomial**
Average accuracy	0.785	0.677

TWCNB

	CNB	WCNB	TWCNB
Accuracy	100%	82%	89.25%

The fact that CNB alone achieved 100% accuracy was somewhat troubling. To confirm that no accidental cheating was occurring we reran the experiment over a larger test set. Here it achieved 99.65% accuracy. This suggests that it simply was overfitting by a large margin and not that it was always correct as we feared.

Why performance dropped off so much after running the reweighting was not entirely clear. At first we believed this to be a bug but the few reweighting lines of code we needed were checked several times. As expected though the final step of transforming the original word frequencies did lift the performance of the algorithm though so much ground had been lost previously that it still underperforms the other algorithms we used.

We also tried excluding the reweighting from the analysis and using just frequency transforms and complementing. That also achieved 100% accuracy. Looking at the larger test set (500 from each class) TCNB achieved 99.7% accuracy while CNB achieved 99.68% accuracy. These differences are too small to draw conclusions on but it is possible both are helping. [Note that just TNB achieves 71% accuracy but running it from the command line does not work properly and it is not clear anyone would use just TNB in practice]

Domain specific techniques

All testing for these techniques were done using the binomial classifier. This was done somewhat arbitrarily since one would expect similar performance gains from each of the classifiers.

The first thing we examined was the source of the errors we were seeing. Most of our errors were from the usenet group “talk.religion.misc” which is easily confused for “soc.religion.christian.” We read the posts in both (which we should note normally would not be acceptable since it introduces large concerns of overfitting but we are assuming that there is some large training/testing corpus outside of the one we are using) and identified that the variants of the word “sex” were being used at different (though somewhat similar) rates between the two sets. These were not variants that the stemmer would combine correctly (homosexual, sexuality or sex). So we added this as a new feature. Performance did not change much.

Indeed most new features had zero impact on performance. While they might change the probabilities assigned to each document it often did not change them enough to change rankings. Along the same lines we added features for punctuation (exclamation marks, question marks, periods, dollar signs). Additionally we added a feature for all words related to Judaism (“jew”, “hebrew”, etc) since those were seen less in the the christian group. We saw less than percentage point gain in accuracy here.

We noted that often a user's signature has no bearing on the group subject. We detected the ends of posts by looking for lines of dashes and stopped parsing at that point. This hurt accuracy until we noted that we were deleting all PGP posts. After correcting that this technique bought us a little more accuracy.

In the end all the domain specific features together bought us half a percentage point increase bringing us to 92% from about 91%. It seems that most trivial features really have very little effect on the overall accuracy of the classifier at least for these sizes of training and testing sets.

Multi-class support vector machines

A standard support vector machine can only perform binary classification. At first we started building many one versus all classifiers but we found that someone had already designed an svm library that met our needs. We used svm-multiclass (http://svmlight.joachims.org/svm_multiclass.html) a svmlight variant which allows for multiple different classes. The algorithm it uses is described in "K. Crammer and Y. Singer. On the Algorithmic Implementation of Multi-class SVMs, JMLR, 2001." but in essence it solves for all of the hyperplanes simultaneously. We used a linear kernel but we still had to decide upon the regularization constant. The following table shows our accuracy on the standard 400 document test set for various values of the regularization constant (the default value for which, for context, is 0.01).

Constant value	Error rate
50	17.75%
5000	3.50%
50000	0.75%
5000000	0%

Note also that train time was quite long compared to the bayes models we used in class. Indeed the (essentially) unregularized model took nearly a half hour to train. We attempted quadratic kernels but found the training time to be unacceptable (it is unclear if that is a product of the number of classes or the regularization parameter or the codebase).

One would expect the 0% error model to have very poor out of sample performance. Without more data (or using k-fold training) we cannot really say which of the constant values are optimal here. Still this experiment does make us more impressed with TWCNB as SVM performance for reasonable regularization values was not exceptional.

The script that generates files of the correct format for the svm learner is called "data_dump.py."