NLP classification investigation

Summary:

This investigation focuses on looking at the performance of various textual classifiers and their configurations against the amazon review corpus of documents. These documents detail customer product reviews grouped by product category and contain both classified and rated documents for the training and testing processes of the candidate classifiers. These documents are classified based on whether they represent a positive or negative review of a particular product.

Approach:

This investigation focuses on evaluating the performance of a number of candidate classifiers against a pre-classified set of documents. These documents are either hand classified from the corpus, or sentiment assumed based on rating measures where < 1.1 equates to a negative review and >4.9 denotes a positive review. Using a cross validation averaging technique, the corpus is broken into a number of equally sized buckets, randomly distributed throughout. For each bucket or 'run', the candidates are evaluated in turn by training against a designated 'training' set and then assessed against a 'test set'. The buckets iterated over determines a test set as being defined as the bucket number and the training set is the remainder of the buckets. For each bucket or 'run' the trained candidate classifiers are tested in turn against the test set and the differences between the estimated classifications and their true classifications are observed and recorded. There are a total of 5 measures of performance calculated:

- Accuracy – The proportion of the test set that have the same estimated and true classifications. How many classifications the classifier correctly predicted.

- Error Rate – The proportion of the test set that have different estimated and true classifications. How many classifications the classifier wrongly predicted.

- Precision – In a binary classifier, the precision is the proportion of correctly predicted positive classifications over the total of all positive classification predictions. The proportion of positive classifications that were made were correct.

- Recall – The proportion of correctly predicted positive classifications over the actual number of positive classifications. How many positive classifications the classifier got right.

- F-Measure – The harmonic mean of the precision and recall.

Over the entire bucket set, these values are then averaged to provide an accurate measure of performance of each of the candidates. These candidates are broken down into 5 main test types:

- *Comparisons between classifier implementations:*

This test involves trialing different classifiers against each other using the 'book' domain of the corpus with the Document Words feature extractor configured:

- Naïve Bayes

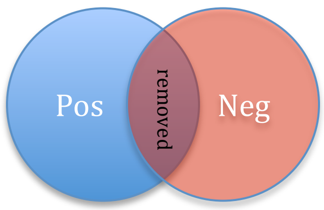
  This approach builds a probability model that represents conditional probabilities of classification/feature pairing weighed by the prior probabilities of the classifications. Once the model is constructed based on representative proportions in the training set, classification is achieved by trialing all possible classifications using the combined probabilities in the model that are present in the document under test. The joint product of all these probabilities is then computed, multiplied by the prior probability of that particular classification and the classification with the maximum probability is returned.

- Static word list classifier counter

  The approach uses two hand crafted lists of words, one representing positive sentiments and the other representing negative ones. A review is classified based on which of these lists has more members occurring in a given list. In the event of a tie break, a random classification is made (theory being that this will be, on average correct 50% of the time assuming equal probabilistic distributions of the classifications).

- Dynamic word list classifier counter

  This approach is similar to the static classifier above except it dynamically generates the word lists. Two methods of extracting these words are tested in this investigation. The first uses the n most frequent words present in each of the pre classified training data sets, whilst the second uses all words above a particular frequency count. Due to the brute force nature of these approaches, each of the positive and negative wordlists have their intersections removed so that words that appear in both are assumed to hold no sentiment in regard to classification - the nature of the accumulator scoring process means that words shared between the word lists will cancel each other out and therefore are unrequired.



- *Comparisons of different feature extractors*

  This looks at how features are extracted from each of the documents in the corpus. There are 4 types that are considered:

  - Document words – Each word that occurs in a document is used. This is the simplest type of feature extractor possible
  - Simple feature extractor – All lowercase translations of words in a document are considered if they are alpha numeric and do not appear in a stoplist. The stoplist is a list of words that are so common that no sentiment can be accurately determined from them. The stoplist is calculated using the NLTK library plus some additional words added as part of observing common positive and negative feature sets (such as ‘book’, ‘read’, ‘write’ etc)
  - PorterStemmer – This using the NLTK implementation of the porter stemmer algorithm[1]. Stemming is a technique used to remove certain postfixes of words, reducing common meanings in words to a single prefix. An example would be to remove a distinction between singular and plural variations or compress the words "universal", "university", and "universe" to "univers". Stoplist words are also removed.
  - LancasterStemmer – Also known as the Paice/Husk stemmer, this is another iterative stemmer that comes bundled with the NLTK library[2]. Stoplist words are also removed

- *Comparisons between most informative features of each of the classifiers*

  This involves observing the internal state of the classifier are a testing run has occurred to view what features each of the classifiers where most utilized in order to determine the estimated classifications. For these tests, all candidate and feature extractor combinations specified above are tested against the full ‘book’ categorized corpus.

- *Comparisons of different training set sizes*

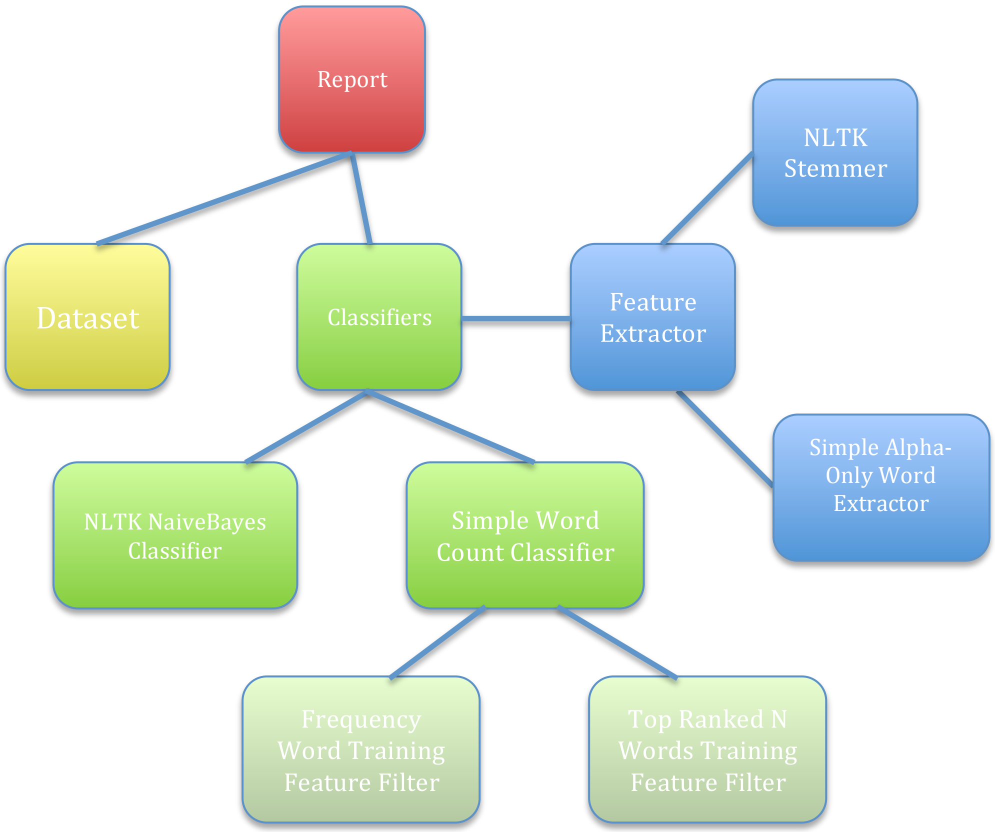
  The amazon corpus contains a number of documents in each of the categories. This test is to determine what impact the size of the training set has on the accuracy of the classification. To achieve this, the book category is broken into training sets of 50, 100, 500, 1000, and full (2831 reviews). The size of the test set remains the same throughout.

- *Comparisons of different review categories used for training and testing processes*

  As already alluded to, the amazon corpus contains reviews based on four distinct different categories or domains – books, dvd, kitchen and electronics.

This test is to determine whether there is a difference in the way positive and negative reviews of one domain can effect the classification accuracy of another. To accomplish this, each classifier is trained against documents from a particular domain and its performance observed by testing against a test document set from a different domain. To evaluate all combinations, 16 tests will be conducted with a further test to observe how all combined domains perform in the training and testing components. Only the classifier that achieved the best operation in the first test will be used in this comparison so as not to overload the investigation with unwanted noise. Preliminary tests will be conducted with other classifiers to ensure that there is no bias on one domain set/classifier pairing but there is no obvious reason to suggest that this is the case.

Implementation:


Report
NLTK Stemmer
Dataset
Classifiers
Feature Extractor
Simple Alpha-Only Word Extractor
NLTK NaiveBayes Classifier
Simple Word Count Classifier
Frequency Word Training Feature Filter
Top Ranked N Words Training Feature Filter

The responsibilities of these components can be briefly summarized as follows:

Dataset

These collections of classes are responsible for encapsulating the training and test sets, providing explicit associations with classifications where required, and also facilitate in cross validation functionality via the 'bucketing' functionality. Note that this is a custom implementation of n-fold cross validation whereby the proportion of |training|/|test| can be altered as well. To facilitate the domain comparison tests, the cross validation functionality works with different training and test corpuses.

Report

This is the main testing harness of the framework. It is responsible for coordinating the execution of a classifier against a dataset, iterating over all buckets of that dataset's training and test set, and averaging the accuracy, error, recall, precision and f-measure values from the resulting training and classification steps. This module also contains static utility methods for displaying results in tabular and graphic forms.

Classifiers

This contains the main implementation of the framework. It defines the concept of a supervised classifier factory that builds a classifier by training it against a list of ClassifiedDataset objects defined in the Dataset module. The factory then returns a classifier that can be used to catalog an unknown document. Note that although this investigation focuses on a corpus with binary classifications, this part of the framework is built to work with multiple classifications.

There are two main classifiers in this module – the NLTK naïve bayes classifier, and a custom simple accumulator based classifier. Both have pluggable components that can be injected in to determine the configurable properties under test (feature extraction, stoplists, wordlist filters) with common properties abstracted into base classes.

The classifiers returned from the factory initially work with the types defined in the Dataset module but also contain a translation wrapper (NltkClassifierTransformer) that converts the typed document sets into the large homogenous dictionary format that the NLTK library requires. This translation is performed transparently to the rest of the framework. The reason why a more typed domain model is used for the framework is because, although python as a language does not unfortunately enforce compile time type safety, using a more structured datamodel is easier to debug and construct then the brittle format that NLTK uses.

Main

This module contains the actual tests that have been performed in this experiment. Utility methods are present to aid in the construction of the corpus domain extractions and the truncation of their document sets for the purpose of evaluating the last two test types.

Naming

Given the different experiences detailed above, each configuration of classifier, feature extractor pairing has been given a name to denote them in the tests. The naming follows a consistent format of the form:

```
<classifier{[<wordlistFilter>]}>@{<featureSetExtract>}
```

For example, the classifier with the name “NaiveBayes@” represents an NLTK naïve bayes classifier with no feature set extractor (just uses the words in each document). Whereas the classifier name “SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@SimpleExtractor” represents the simple count classifier trained using a static word list and the simple feature extractor that considers all lower case alpha words not present in the default stoplist.

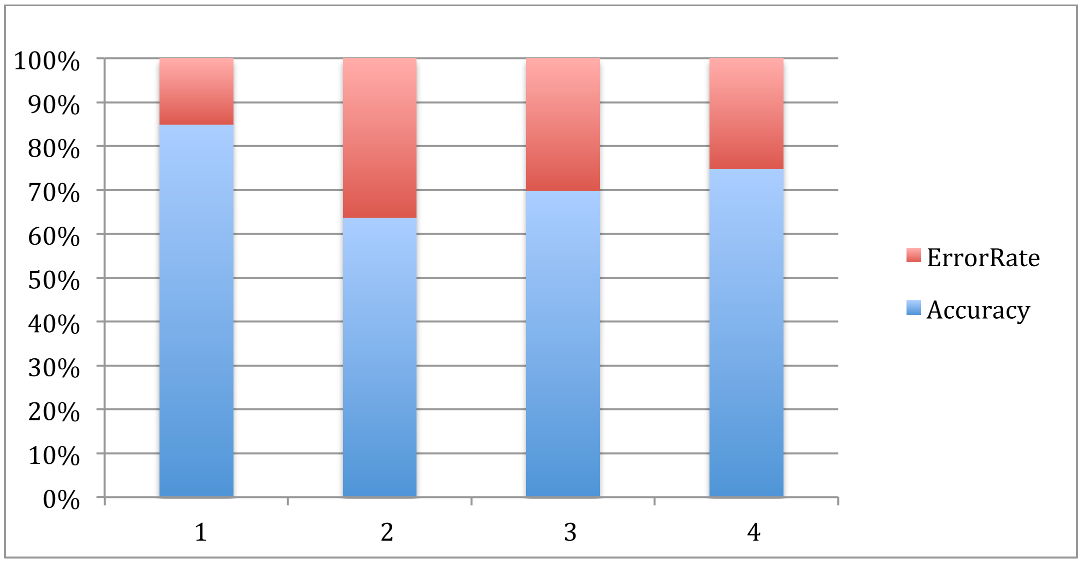
Results and Conclusions:

The following are results of the tests outlined previously, all using a 10 ten fold cross validation using a 1:10 split between test and training datasets. Note that, where applicable and due to the verbose nature of the classifier names, graphs are labeled with the id of the classifier rather then with their names directly.

- *Comparisons between classifier implementations:*

Here the classifiers of 4 different implementations are considered and evaluated. The classifiers are naïve bayes and 3 derivations of the simple wordlist accumulator classifiers. All use just the words in the document as the featureset.

Id	*Classifier Name*	*Accuracy*	*Error Rate*	*Precision*	*Recall*	*FMeasure*
1	`NaiveBayes@`	0.8498	0.1502	0.9928	0.7856	0.8764
2	*`SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@`*	0.6375	0.3625	0.7227	0.7609	0.7411
3	*`SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@`*	0.6987	0.3013	0.6974	0.987	0.8173
4	*`SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@`*	0.7479	0.2521	0.8508	0.7647	0.8053



The results show that the naïve bayes implementation offers the best all round performance on all but the recall measure. The results are particularly impressive given that not even stop words have been removed from the featureset. Its performance can be explained at least partly due to the fact that the classifier weights every word based on the probabilities of occurs it encountered in each of the classifications of the training phase. This means that common words such as 'and', 'the' etc will provide little bias towards any particular classification.

Using a static word list accumulator provided the worst performance. This can be expected as the number of words used to classify is the lowest of all the techniques evaluated. Using the results from the informative feature investigation, the numbers of times a particular feature was used to aid in classification is very small. In order to have a successful classifier, it is clear that a multitude of features need to be considered to provide accurate classification.

All the other metrics collected reflect the accuracy measure except recall where the simple counter classifier using a generated word list using all words above a particular frequency occurrence in the training set out performs. An explanation for this could be that possibly very few positive classifications were made by the classifier but the ones that were made were highly accurate. The low precision rating indicates that this classifier missed a lot of real positive classifications so this classifier appears to be good if confidence in a positive classification is required.

- *Comparisons of different feature extractors*

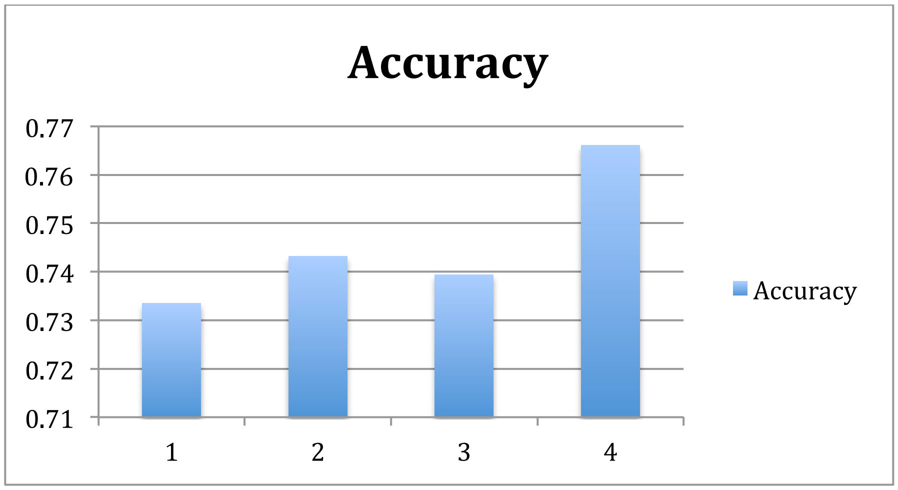
In this test the classifiers in the previous test are configured to use the four different featureset extractors mentioned in the approach section.

	Classifier Name	*Accuracy*	*Error Rate*	*Precision*	*Recall*	*FMeasure*
1	`NaiveBayes@`	0.8498	0.1502	0.9928	0.7856	0.8764
2	`NaiveBayes@SimpleExtractor`	0.9263	0.0737	0.9863	0.9047	0.9436
3	`NaiveBayes@nltk<LancasterStemmer>`	0.8952	0.1048	0.9783	0.866	0.9183
4	`NaiveBayes@nltk<PorterStemmer>`	0.9098	0.0902	0.9836	0.8828	0.9302
5	`SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@`	0.6375	0.3625	0.7227	0.7609	0.7411
6	`SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@SimpleExtractor`	0.6562	0.3438	0.7311	0.7851	0.7566
7	`SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@nltk<LancasterStemmer>`	0.6492	0.3508	0.7225	0.7893	0.7543
8	`SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@`	0.6987	0.3013	0.6974	0.9870	0.8173
9	`SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@Simp`	0.6981	0.3019	0.696	0.9902	0.8174

	leExtractor					
10	SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@nltk<LancasterStemmer>	0.6889	0.3111	0.6932	0.9763	0.8107
11	SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@nltk<PorterStemmer>	0.6946	0.3054	0.6954	0.9833	0.8146
12	SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@	0.7479	0.2521	0.8508	0.7647	0.8053
13	SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@SimpleExtractor	0.7838	0.2162	0.8724	0.8005	0.8348
14	SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@nltk<LancasterStemmer>	0.7244	0.2756	0.8451	0.7302	0.7832
15	SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@nltk<PorterStemmer>	0.7600	0.2400	0.8658	0.7674	0.8133
16	NaiveBayes@	0.8498	0.1502	0.9928	0.7856	0.8764
17	NaiveBayes@SimpleExtractor	0.9263	0.0737	0.9863	0.9047	0.9436
18	NaiveBayes@nltk<LancasterStemmer>	0.8952	0.1048	0.9783	0.866	0.9183

Using the simple, alpha only and stoplist exclusion extractor the naïve bayes classifier can be observed to exceed a 92% accuracy rating with a precision of >98%. This classifier is exceptionally performant when it is necessary to classify all positive documents correctly. It also exhibits only a small degree of error mis-classifying negative documents. Taking the average performance of all classifiers gives an indication of how the feature extractors perform overall.

Id	*Classifier Name*	*Accuracy*	*Error Rate*	*Precision*	*Recall*	*FMeasure*
1	Words in document	0.733475	0.266525	0.815925	0.82455	0.810025
2	Porter stemmer	0.743175	0.256825	0.81435	0.836975	0.817875
3	Lancester stemmer	0.739425	0.260575	0.809775	0.84045	0.816625
4	Simple alpha extractor	0.7661	0.2339	0.82145	0.870125	0.8381



The results show that the best extractor is the simple alpha extractor. The stemming approaches, although did better then just taking the document words, really didn't perform as was expected. On consideration, stripping a word into stems could easily remove unwanted sentiment in a word (such as compressing a noun and verb into the same stem) and could explain why they didn't beat the simple alpha word extractor.

- *Comparisons between most informative features of each of the classifiers*

This test considers the internal state of each of the classifiers to determine what features have been most informative in classifying a particular document instance.

```
classifier: NaiveBayes@
Most Informative Features
           disappointing = True              NEG : POS    =     29.6 : 1.0
                    weak = True              NEG : POS    =     16.7 : 1.0
           dysfunctional = True              NEG : POS    =     13.6 : 1.0
                 kidding = True              NEG : POS    =     13.6 : 1.0
                rambling = True              NEG : POS    =     13.6 : 1.0
                Danielle = True              NEG : POS    =     13.6 : 1.0
                  Unless = True              NEG : POS    =     13.6 : 1.0
                  poorly = True              NEG : POS    =     13.6 : 1.0
                 shallow = True              NEG : POS    =     13.3 : 1.0
                   Steel = True              NEG : POS    =     12.2 : 1.0
classifier: NaiveBayes@nltk<PorterStemmer>
Most Informative Features
                  poorli = True              NEG : POS    =     14.5 : 1.0
                 endless = True              NEG : POS    =     13.3 : 1.0
                   bottl = True              NEG : POS    =     10.7 : 1.0
                 mediocr = True              NEG : POS    =     10.7 : 1.0
                  pigeon = True              NEG : POS    =     10.7 : 1.0
                    rear = True              NEG : POS    =     10.7 : 1.0
               caricatur = True              NEG : POS    =     10.7 : 1.0
                outstand = True              POS : NEG    =     10.4 : 1.0
               superfici = True              NEG : POS    =     10.1 : 1.0
                 shallow = True              NEG : POS    =     10.1 : 1.0
classifier: NaiveBayes@nltk<LancasterStemmer>
Most Informative Features
                outstand = True              POS : NEG    =     10.7 : 1.0
                   bottl = True              NEG : POS    =     10.7 : 1.0
                 mediocr = True              NEG : POS    =     10.7 : 1.0
                  pigeon = True              NEG : POS    =     10.7 : 1.0
                    rear = True              NEG : POS    =     10.7 : 1.0
                 shallow = True              NEG : POS    =     10.1 : 1.0
                   rambl = True              NEG : POS    =     10.1 : 1.0
                  superf = True              NEG : POS    =      9.5 : 1.0
                   moron = True              NEG : POS    =      9.3 : 1.0
                disjoint = True              NEG : POS    =      9.3 : 1.0
classifier: NaiveBayes@SimpleExtractor
Most Informative Features
           disappointing = True              NEG : POS    =     30.4 : 1.0
                    weak = True              NEG : POS    =     16.7 : 1.0
                rambling = True              NEG : POS    =     15.0 : 1.0
                  poorly = True              NEG : POS    =     14.5 : 1.0
                 shallow = True              NEG : POS    =     14.2 : 1.0
           dysfunctional = True              NEG : POS    =     13.6 : 1.0
                 kidding = True              NEG : POS    =     13.6 : 1.0
                 endless = True              NEG : POS    =     13.3 : 1.0
                  pigeon = True              NEG : POS    =     10.7 : 1.0
                 borders = True              NEG : POS    =     10.7 : 1.0

classifier: SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@
POS: good -> 74
POS: great -> 53
POS: best -> 47
```

```
POS: wonderful -> 16
POS: excellent -> 16
NEG: bad -> 17
NEG: worst -> 5
NEG: terrible -> 3
NEG: awful -> 2
NEG: inconsequential -> 1
classifier:
SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@nltk<PorterStemmer>
POS: good -> 80
POS: great -> 58
POS: best -> 48
POS: wonder -> 26
POS: excel -> 19
NEG: bad -> 17
NEG: worst -> 5
NEG: terribl -> 5
NEG: trash -> 2
NEG: nightmar -> 2
classifier:
SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@nltk<LancasterStemmer>
POS: good -> 80
POS: gre -> 62
POS: best -> 48
POS: wond -> 26
POS: excel -> 19
NEG: bad -> 18
NEG: aw -> 7
NEG: worst -> 5
NEG: trash -> 2
NEG: nightm -> 2
classifier: SimpleCountClassifier[StaticWordListTrainingFeatureFilter]@SimpleExtractor
POS: good -> 79
POS: great -> 58
POS: best -> 48
POS: excellent -> 18
POS: wonderful -> 17
NEG: bad -> 17
NEG: worst -> 5
NEG: terrible -> 3
NEG: awful -> 2
NEG: trash -> 2
classifier: SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@
POS: treat -> 5
POS: appreciation -> 5
POS: strategic -> 5
POS: shift -> 5
POS: revealing -> 4
NEG: remotely -> 3
NEG: ha -> 2
NEG: input -> 2
NEG: obstacle -> 2
NEG: blah -> 2
classifier:
SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@nltk<PorterStemmer>
POS: valley -> 3
POS: battlefield -> 3
POS: wheel -> 3
POS: delic -> 3
POS: ian -> 3
NEG: curio -> 3
NEG: firstli -> 2
NEG: consolid -> 2
NEG: texa -> 2
NEG: knit -> 2
classifier:
SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@nltk<LancasterStemmer>
POS: strategic -> 5
POS: exc -> 4
POS: valley -> 3
POS: battlefield -> 3
POS: wheel -> 3
NEG: clo -> 17
NEG: ana -> 12
```

```
NEG: ha -> 3
NEG: curio -> 3
NEG: anom -> 2
classifier: SimpleCountClassifier[TopRankedNWordsTrainingFeatureFilter]@SimpleExtractor
POS: treat -> 6
POS: appreciation -> 5
POS: strategic -> 5
POS: shift -> 5
POS: revealing -> 4
NEG: ha -> 3
NEG: remotely -> 3
NEG: input -> 2
NEG: firstly -> 2
NEG: obstacle -> 2
classifier: SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@
POS: the -> 286
POS: and -> 282
POS: of -> 277
POS: to -> 270
POS: a -> 266
NEG: the -> 286
NEG: and -> 282
NEG: of -> 277
NEG: to -> 270
NEG: a -> 266
classifier: SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@nltk<PorterStemmer>
POS: like -> 96
POS: good -> 80
POS: would -> 80
POS: author -> 77
POS: stori -> 74
NEG: like -> 96
NEG: good -> 80
NEG: would -> 80
NEG: author -> 77
NEG: stori -> 74
classifier:
SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@nltk<LancasterStemmer>
POS: lik -> 96
POS: us -> 95
POS: ev -> 94
POS: real -> 81
POS: good -> 80
NEG: lik -> 96
NEG: us -> 95
NEG: ev -> 94
NEG: real -> 81
NEG: good -> 80
classifier: SimpleCountClassifier[FrequencyWordTrainingFeatureFilter]@SimpleExtractor
POS: like -> 91
POS: would -> 80
POS: good -> 79
POS: reading -> 76
POS: well -> 71
NEG: like -> 91
NEG: would -> 80
NEG: good -> 79
NEG: reading -> 76
NEG: well -> 71
```

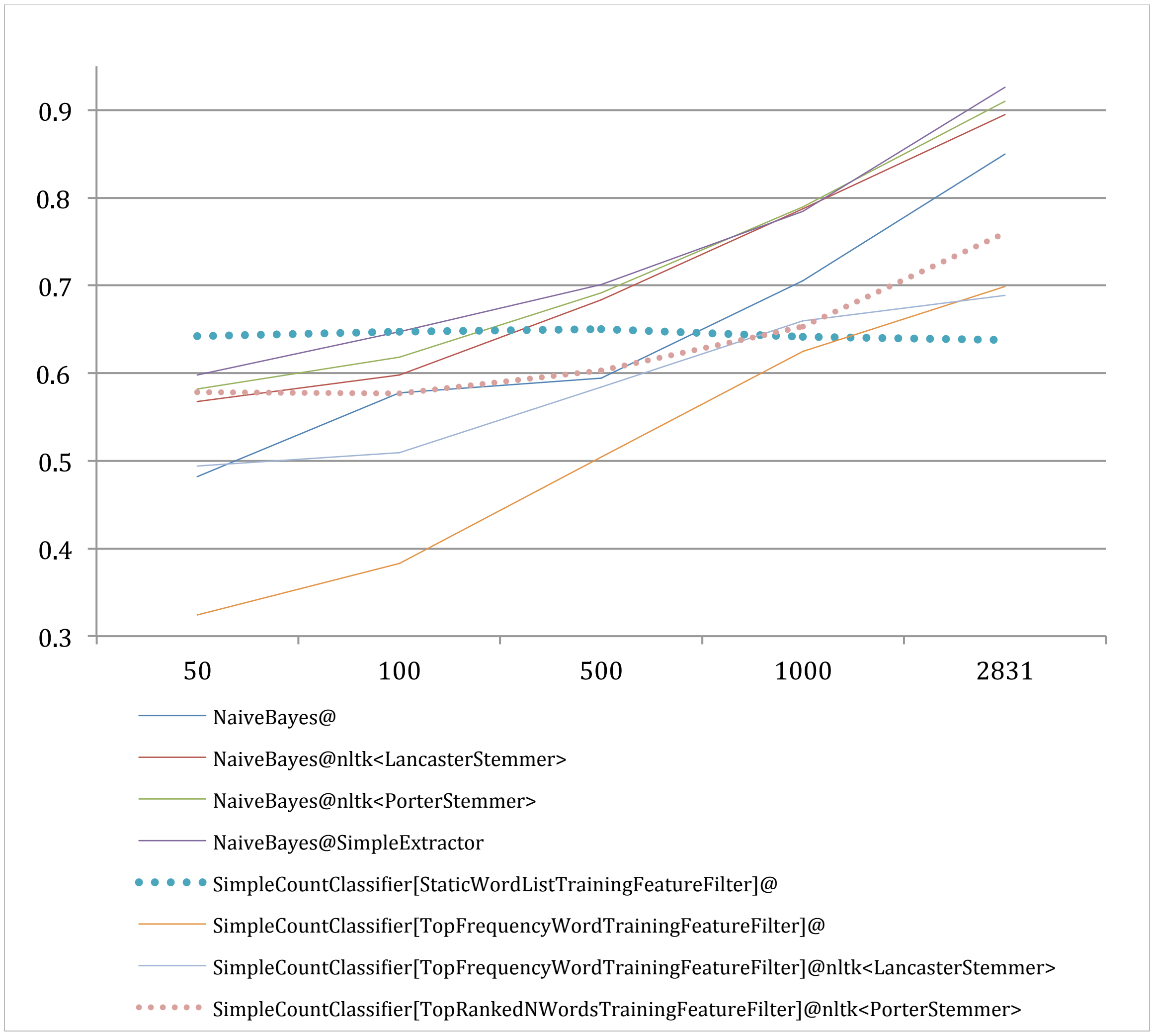
The results show that when no feature set extractors are used Naïve bayes is still successful in modeling words that express positive and negative sentiments. Using the dynamic word list simple accumulator classifiers on the other hand are clearly extracting words that imply no obvious opinion towards positive or negative classifications (such as and, the, of, shift, strategic, remotely etc). Even after including feature set extractors with stop lists, the most informative features used still do not express any immediately obvious meaning towards the required classifications. The stemming examples reveal an indication as to why their performance might be hampered. Stems such as ‘ev’ or ‘us’ could represent many different words that are all contributing to a particular classification that their full form may not do. This could easily result in certain documents being erroneously classified.

The other interesting outcome of this test is the low frequency that words are used to classify a document. Using the top n ranked word list extractor (that also filters out words contained in both positive and negative documents), the classifier only uses the most important words in a maxiumum of 6 documents. Compare that with the statically assigned word list that peeks at 80 documents classified using the word ‘good’. Given that the performance of these classifiers is comparable (see previous tests), this implies that choosing an appropriate but small word list is just as effective as generating a large but ultimately unfocused word list based on the classified training document set.

There is a concerning result regarding the FrequencyWordTrainingFeatureFilter. It appears that both the positive and negative word lists are the same. At first this anomaly was assumed to be the result of a bug in the implementation but after a long period of debugging, if there is a bug in the implementation, it remains elusive. The only other explanation is that, due to the fact that this classifier doesn’t filter out the intersection of classification word lists, that these words are so common in the corpus that they appear in both document classification equally.

- *Comparisons of different training set sizes*

This test explores the impact of the training set size on classifier performance. The data table is too large to present in its entirety in this report to retain focus so just the graphical results are presented. The raw data can be found in the companion source tree under the ‘Results’ file. Some classifiers have also been omitted from the graph as they all follow the same trend.



The results clearly show a positive trend in accuracy as the training set size is increased. The only classifier that doesn't exhibit this movement is the static word lists. This is because this classifier ignores the training phase as it has its own statically assigned word lists. Increasing the training set on this classifier has no impact on this classifier.

This result is largely unsurprising. By increasing the number of documents shown to the classifier in the training stage, is going to show it a bigger representation of the language domain. Seeing a bigger representation is going to allow all the classifiers to tune their internal models to better reflect the real language sentiments present in a particular domain. To take an extreme case to illustrate this point, take a training set of just one document. The classifier will only be able to class other documents that closely follow the training documents word structure. The classifier would fail to classify any other documents with any degree of accuracy. Using a larger training set allows the classifier to learn bigger feature sets so that they can accurately classify more and more different document types.

- *Comparisons of different review categories used for training and testing processes*

The following shows the accuracy results of a naïve bayes classifier with simple feature extraction trained and tested against differing domains. The column headers represent the training set domain source whilst the rows denote the test set domain.

	book	dvd	electronics	kitchen
book	0.9168	0.7962	0.5317	0.5708
dvd	0.6378	0.9515	0.6064	0.6010
electronics	0.5069	0.6680	0.8778	0.6527
kitchen	0.5820	0.6982	0.7577	0.9164

The results here show that the classifier performs best when training and test sets come from the same category. In particular the language used in the 'dvd' domain appears to be the most descriptive in defining sentiments of our classification types. Electronics appear to be the least descriptive. It might be worth commenting on the size of the corpus in each of these domains. The book domain's 2831 documents is comparatively dwarfed by the dvd's 3793 documents which, given the results of the training set size test, could explain why dvd outperforms book. This logic breaks down however when considering the electronics domain which contains a comparable 3771 documents.

The results also show that there are certain domains that perform reasonably well when trained and tested against each other. Training a classifier using the electronics domain and testing it against the kitchen domain offers fair results. This implies that there might be a similar language and tone structure between these categories. Kitchen and electronic appliances share a similar cost and marketing strategy in society (you can normally find these products in the same store for example) and therefore share demographics so this could be indicative of this relationship. A similar observation can also seen between the dvd and book classes. Again, stores like HMV can be seen stocking these products and the correlation seen here in this test could be reflective of similar people commenting on product reviews. Conversely, products that are wildly different such as book and electronic products show very poor performance.

Bibliography

[1] http://tartarus.org/martin/PorterStemmer/
[2] http://www.comp.lancs.ac.uk/computing/research/stemming/paice/article.htm

word count: 3415