Computer lab: Statistical models for textual data

Måns Magnusson, Leif Jonsson

May 10, 2013

a)

We used the naive bayes classifier in the `nltk` package to train a simple classifier on whether the movie rating were positive or negative in the `movie_reviews` corpus in the `nltk` package. All the code can be found in the two python files (one with the functions and another with the main code).

We used 10 % of the corpus (the reviews) as a test set (randomly selected) for testing the accuracy of the final models. We also set aside 5 % of the corpus to use as a development set to be able to identify potential problems in the modeling of the data without studying the results in the test set specifically.

Using the naive bayesian classifier with the featureset `contains:[word]` resulted in a quite good classifier (especially compared with the other classifiers tested in this lab). The result of this classifier were:

```
Accuracy : 0.7525
Precision : 0.7411
Recall : 0.7526
F-value : 0.7468
```

In general for Måns it was a quite different way of representing features in the model this way. The way the features is expressed (as items in a hash table) is not very common in the way features is expressed in machine learning and in statistics. Is there any benefits of using this way of features definitions instead of using an ordinary matrix notation used in "ordinary statistics"?

b)

To improve the performance of the classifier were tried different ways of normalizing the textual data. We tried to remove "uninteresting" data such as stop words, punctuations and numbers. By removing stop words, punctuations and numbers the classifier were slightly improved the model and we got the following results when classifying the test data set:

```
Accuracy : 0.7725
Precision : 0.7668
Recall : 0.7629
F-value : 0.7649
```

But when we used lemmatization and/or stemming with the same featureset `contains:[word]`, we instead reduced the classifier accuracy. The reason for this is that probably different inflections of the same word can have a limited effect in the classification when the inflections are removed. One example of this is the word `fails` that increases the likelihood of the category `neg` by 2.3 while the word `fail` do not have almost any effect on the likelihood of the different categories. So using the naive bayes classifier and normalizing words using stemming and/or lemmatization does not necessarily improves the classification performance. And this is of course true for all classifiers, not just the naive bayes classifier.

It is interesting to see the effect of removing stop words, numbers and punctuations. By removing these tokens the classification accuracy increased slightly (by 0.02). So the naive bayesian classifier were hence better when a few non-informative features were excluded. As a comparison, when another 1000 word/tokes were included (instead of the 1000 most common words, the 2000 most common words were used), the accuracy increased to 0.79 from 0.7525.

c)

To continue with the modeling of the text data both document length and average word length were included as features in the model. We also included the number of times a specific word were included in the model (counts). The main problem that we encounter were that even since there are many words included in the model, the result of including (binned) counts in the model increases the number of features dramatically and this resulted in an over-fitting of the model. There are just too many features that is used in the model.

To reduce the risk of over-fitting we only included the count of words with the bins 0,1 and 2+ words, but even with this small addition to the number of features, the model were over-fitted with poor accuracy as a result. This model gave the following results for the test data:

```
Accuracy : 0.515
Precision : 0.5
Recall : 0.6134
F-value : 0.5509
```

To reduce the risk as much as possible we tried that instead of studying the number of features this way we instead used a "lexical approach" were we counted the number of positive words and negative words in each document. The negative and positive words, that is often used in opinion mining, were found here.

The positive and negative word lexical method were used simply by counting the number of positive and negative words in each document. The number of positive and negative words were then categorized into 10 bins (based on the quantiles) and used as features. By only using the counts of positive and negative words (a total of 20 features) the classification accuracy were quite high:

```
Accuracy : 0.65
Precision : 0.645161290323
Recall : 0.618556701031
F-value : 0.631578947368
```

When we included the 1000 most common words in the classifiers (as in section a) together with the counts of positive and negative word the classifier is a little better than the classifier in a, but not much better:

```
Accuracy : 0.7675
Precision : 0.758974358974
Recall : 0.762886597938
F-value : 0.760925449871
```

So the main conclusion is that it is hard to get the classifier to do much better than an accuracy of approximately 80% using the Naive Bayesian classifyer with this data.

d)

As a last thing we also tried using the tf-idf weighted features. Since the results in c) resulted in that just adding one more bin than compared with part a) resultet in a heavily over-fitted model our guess were that the tf-idf weighted features would have the same problem, if the tf-idf features just were used instead of ordinary counts.

Instead we tried to use the tf-idf to identify which features that is of interest in the classification task. So what we did was that we computed the tf-idf for all documents (or reviews). We then summed the tf-idf for the negative documents and the positive documents and calculated the difference between the sums of the two classes. The purpose was to mimic the cosine similarity but at the term level (instead of the document level), but instead be intereseted in the terms with a large cosine difference.

This resulted in that word such as boring, worse, realistic, superb etc. were identified as having the largest tf-idf difference between neg and pos categories. We then included the 50 words with the largest differencein the model with quite good classifying result:

```
Accuracy : 0.7125
Precision : 0.7158
Recall : 0.6753
F-value : 0.6950
```