732A47 Text Mining

Lab 5 - Statistical Models for Textual Data

Sebastiano Milardo (sebmi912)
Tomas Peterka (tompe625)

To deeply understand the implemented functionalities it is important to realize that the process of getting the final features is composed of three steps:

- **Document cleaning** - In this step we used a set of functions which will prepare input documents to be processed. All documents have to go through this phase. An example of this kind of functions can be: string.lower, stemming, lemmatizing, stop-words removing etc.
- **Features selection** - function will generate the features candidates from a list of all words in the documents. This will reduce the size of feature candidates and keep only the valuable ones. An example might be: the most frequent 1000 words, words with proper document frequency weight etc. These functions produces sets of features which can be joined into one big set of feature candidates.
- **Feature extraction** - For each document in the documents set our program extracts a set of features and it stores these values with their corresponding labels for training purposes.

We splitted our code in four different parts. Each part correspond to a question of the lab. We recommend for this lab to use "pypy" instead of python, because it can speed up the execution of the code.
To run our program just use the command "pypy /path/to/file/Lab5.py n X" where n is the number of repetitions for each analysis in a part and X is the corresponding letter for that part. Therefore to run the whole analysis just type "pypy /path/to/file/Lab5.py 10 A B C D"

a) *Use the movie_reviews data from the nltk.corpus module to construct a naive Bayes classifier from a training sample consisting of 80% of the data. Use only binary features [has('word')] based on the most frequent 1000 words in the corpus. Do not pre-process the text in the reviews. Evaluate the predictive performance on the test data (the 20% that you left out from training) using measures like Accuracy, Precision, Recall and the F-measure*

We evaluated the predictive performance of our algorithm by using it ten times on ten differently shuffled document sets. These are the average results that we obtained:

```
------------------------------------------------------------------------------------------------
| F-Meas.| Acc.   | Prec.  | Rec.   | Setup                                                     |
------------------------------------------------------------------------------------------------
|  0.769 |  0.781 |  0.792 |  0.748 | Lower + Freq. Filter + Has(feat)                          |
------------------------------------------------------------------------------------------------
```

Table A.1. Results using binary features [has('word')] based on the most 1000 frequent words in the corpus.*

*In this report we always compute F-measure using alpha = 0.5

***b)** Redo the analysis in a), but this time do a careful pre-processing of the data before training and prediction. Play around with normalization, stemming, removing stop words, removing punctuation, remove numbers, or whatever you find interesting. Use the same subset for training and testing as in a). Compare the predictive performance of the models to the model in a). Be creative. Analyze.*

In this step we used different preprocessing functions. We used the Port stemmer and the Lancaster stemmer. We also used the Lemmatizer and we removed stop words and punctuations. We tried these functions alone and in different combinations. As in the previous case we used the first 1000 frequent words in the corpus as features and we computed the featureset by using the binary property "has(feature)".
As before, we ran our analysis ten times and we computed averages of the results. Is important to notice that due to the long time needed to run a single test we run every configuration only ten times, so the absolute values that we get are not really significatives. More important are the relatives values between different configuration.

The following table shows the results:

F-Meas.	Acc.	Prec.	Rec.	Setup
0.752	0.767	0.809	0.703	Rm Stop words + Rm Puncts + Port Stemmer + Freq. Filter + Has(feat)
0.750	0.765	0.808	0.700	Rm Stop words + Port Stemmer + Freq. Filter + Has(feat)
0.749	0.764	0.807	0.700	Port Stemmer + Freq. Filter + Has(feat)
0.746	0.762	0.808	0.694	Rm Puncts + Port Stemmer + Freq. Filter + Has(feat)
0.736	0.745	0.769	0.707	Rm Puncts + Freq. Filter + Has(feat)
0.735	0.746	0.775	0.700	Rm Stops + Freq. Filter + Has(feat)
0.732	0.741	0.765	0.703	Lower + Freq. Filter + Has(feat)
0.728	0.742	0.777	0.685	Rm Stops + Lemmatizer + Freq. Filter + Has(feat)
0.728	0.739	0.769	0.691	Lemmatizer + Freq. Filter + Has(feat)
0.725	0.737	0.767	0.688	Rm Puncts + Lemmatizer + Freq. Filter + Has(feat)

Table B.1. Results using different pre-processing functions. Red means best value.

The results are really similar each other, but there are some exception when we consider the use of the Port Stemmer and removing stop words and punctuations.

***c)** Continue with the best version of the pre-processed data from b), but now do a more careful selection of features. Consider not only binary features, but also perhaps frequencies of individual words, measures of lexical diversity, average word length, maybe bigrams or collocations. Experiment as much as you like and have the time for. Always compare the predictive performance of the models.*

According to the previous step we decided to use two combinations of "Removing stop words", “Removing punctuations” and “Port Stemming” as preprocessing steps because they gave the best results in term of f-measure. We used the first 1000 frequent words as features then we computed all the possible bigrams and trigrams from these words. After that we computed the two and three terms collocations for the movie_reviews corpus. Moreover we calculated for every document the lexical diversity and the average word length (see all the functions called “xxxx_feats(document)” for the corresponding code). Where possible we used both “has” and “count” features.
By the way, these kind of features can be divided into two categories: “binary” and “continuous”. For the first one we used the corresponding true/false value while for the second type we discretized the continuous value into three different intervals. For example, for the average word length we used the following function:

```
def avg_w_l_feats(document):
        features = {}
        avg = sum([len(w) for w in document])/len(document)
```

```
    features['avg<2'] = (avg<2)
    features['2<=avg<=4'] = (avg>=2 and avg<=4)
    features['avg>4'] = (avg>4)
    return features
```

These are the results for this step:

```
---------------------------------------------------------------------------------------------------------
| F-Meas.| Acc.   | Prec.  | Rec.   | Setup                                                           |
---------------------------------------------------------------------------------------------------------
|  0.758 |  0.771 |  0.803 |  0.719 | Rm Stop w. + P. Stem. + Has(feat) + Has(t-coll)                 |
|  0.755 |  0.768 |  0.798 |  0.719 | Rm Stop w. + P. Stem. + Has(feat) + Has(b-coll)                 |
|  0.754 |  0.767 |  0.797 |  0.717 | Rm Stop w. + P. Stem. + Has(feat) + Has(trigr)                  |
|  0.754 |  0.767 |  0.796 |  0.716 | Rm Stop w. + P. Stem. + Has(feat) + Has(bigr)                   |
|  0.753 |  0.767 |  0.797 |  0.715 | Rm Stop w. + P. Stem. + Has(feat) + Lexical Diversity           |
|  0.753 |  0.767 |  0.797 |  0.715 | Rm Stop w. + P. Stem. + Has(feat) + Average W. Length           |
|  0.753 |  0.767 |  0.797 |  0.715 | Rm Stop w. + P. Stem. + Has(feat)                               |
|  0.752 |  0.766 |  0.798 |  0.713 | Rm Stop w. + P. Stem. + Has(feat) + Count(bigr)                 |
|  0.752 |  0.766 |  0.798 |  0.713 | Rm Stop w. + P. Stem. + Has(feat) + Count(b-coll)               |
|  0.752 |  0.766 |  0.796 |  0.715 | Rm Stop w. + P. Stem. + Has(feat) + Count(trigr)                |
|  0.752 |  0.766 |  0.796 |  0.715 | Rm Stop w. + P. Stem. + Has(feat) + Count(t-coll)               |
|  0.750 |  0.770 |  0.821 |  0.691 | Rm Stop w. + P. Stem. + Has(feat) + Count(feat)                 |
---------------------------------------------------------------------------------------------------------
```

Table C.1. Results using with different features (pre-processing A).

```
---------------------------------------------------------------------------------------------------------
| F-Meas.| Acc.   | Prec.  | Rec.   | Setup                                                           |
---------------------------------------------------------------------------------------------------------
|  0.759 |  0.772 |  0.805 |  0.718 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(t-coll)    |
|  0.759 |  0.772 |  0.803 |  0.721 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(b-coll)    |
|  0.758 |  0.770 |  0.801 |  0.720 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(bigr)      |
|  0.758 |  0.770 |  0.799 |  0.721 | Rm Stop w. + Rm Puncts + P. Stem. + Has(feat)                   |
|  0.758 |  0.770 |  0.799 |  0.721 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Lexical Diversity  |
|  0.758 |  0.770 |  0.799 |  0.721 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Average W. Length  |
|  0.757 |  0.770 |  0.799 |  0.720 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(trigr)     |
|  0.756 |  0.769 |  0.800 |  0.718 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Count(bigr)    |
|  0.756 |  0.769 |  0.800 |  0.718 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Count(b-coll)  |
|  0.756 |  0.769 |  0.799 |  0.719 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Count(trigr)   |
|  0.756 |  0.769 |  0.799 |  0.719 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Count(t-coll)  |
|  0.746 |  0.767 |  0.818 |  0.686 | Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Count(feat)    |
---------------------------------------------------------------------------------------------------------
```

Table C.2. Results using with different features (pre-processing B).

As we can see from the results, the best f-measure values correspond to the use of *[has(feature)], [has('TrigramCollocation')]* and *[has('BigramCollocation')].*
We can justify these result because in this contest Trigram Collocations and Bigram Collocations correspond most of the times to names of famous actors or famous characters. Then usually a positive or negative review is somehow related to the actors in that movie.

We also tried using the document frequency of a word as a way to select feature candidates. We assumed that there are some words that characterize a review to be negative or positive (e.g. "awesome" or "ugly"). These words will probably be in approximately half documents, because we have half positive and half negative reviews. Therefore, if we select words which appear between N-times and 0.7N-times, where N is the number of documents divided by 2,we should cover all the important words. The results seems to support this idea.

We compared the values in following table with the same configuration of question A which resulted in a f-measure of 0.77. As you can see there is almost no information loss by using df <= N.

	df <= N	df <= .9 * N	df <= .6 * N	df >> .4 * N	df >= .5 * N
avg f-measure	0.77	0.77	0.73	0.73	0.69

Table 3.1. Results using different feature thresholds

The following table shows that df <= N is a really good metric because the f-measure on selected features does not change a lot by lowering the number of features generated.

# of features	1000	800	600	400	200
avg f-measure	0.77	0.77	0.77	0.76	0.70

Table 3.2. Results of using df-value features selection with variable number of features generated

d) *Finally, consider the use of tf–idf weighted features.*

The last step of this analysis involves the use of the tf-idf weight as feature. For every word in the feature set, we computed the Inverse document frequency and we stored these values in a set. Then for every word of a document that appears in the features set we computed the Term Frequency and finally we use the formula Tf*Idf to compute the [tf-idf('word')] feature.

We compared the best configurations so far with and without using the Tf-Idf feature. The results of this test are shown below.

```
---------------------------------------------------------------------------------------------
| F-Meas.| Acc.   | Prec.  | Rec.   | Setup                                                   |
---------------------------------------------------------------------------------------------
|  0.763 |  0.772 |  0.810 |  0.721 | 1) Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(b-coll)      |
|  0.762 |  0.772 |  0.810 |  0.721 | 2) Rm Stop w. + P. Stem. + Has(feat) + Has(t-coll)                   |
|  0.761 |  0.773 |  0.817 |  0.712 | 3) Rm Stop w. + P. Stem. + Has(feat) + Tf-Idf(feat)                  |
|  0.760 |  0.773 |  0.821 |  0.709 | 4) RmS. + RmP.  + P.St. + H(feat) + H(bcoll) + H(tcoll) + Tf-Idf     |
|  0.760 |  0.770 |  0.807 |  0.719 | 5) Rm Stop w. + P. Stem. + Has(feat) + Has(b-coll)                   |
|  0.757 |  0.769 |  0.815 |  0.707 | 6) Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Tf-Idf(feat)     |
|  0.757 |  0.768 |  0.805 |  0.715 | 7) Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat) + Has(t-coll)      |
|  0.757 |  0.767 |  0.805 |  0.715 | 8) Rm Stop w. + P. Stem. + Has(feat)                                 |
|  0.755 |  0.764 |  0.801 |  0.714 | 9) Rm Stop w. + Rm Puncts  + P. Stem. + Has(feat)                    |
---------------------------------------------------------------------------------------------
```

Table D.2. Comparison using tf-idf.

The use of Tf-idf increases the performance of our program. As we can see setup n.6 has better performance than n.9. Same for n.3 and n.8. We can also notice that the setup with all the best features, reaches the highest Precision but lacks in Recall. By the way, the increment that we gain using tf-idf is smaller than the one we obtain by using for example Trigram Collocations.