Week 5 - Learning from data

Chris Pool

December 17, 2015

1 Support Vector Machines

svm_chrispool.py

I tried several settings for a support vector machine. I use a development set to test my system. I created the development set by splitting the given train data in 75% train set and the remaining 25% is used for testing. The results with using the default settings can be found in table 1. I also tried several settings for C, these results

	Precision	recall	f1-score	Support
neg	0.82	0.84	0.83	731
pos	0.85	0.82	0.83	769
avg / total	0.83	0.83	0.83	1500

Table 1: Results with default settings (linear kernel / C = 1.0)

can be found in table 2

	precision	recall	f1-score	support
C = 0.1	0.80	0.79	0.79	1500
C = 0.5	0.83	0.83	0.83	1500
C = 0.75	0.83	0.83	0.83	1500
C = 1.0	0.83	0.83	0.83	1500
C = 5.0	0.81	0.81	0.81	1500
C = 20.0	0.81	0.81	0.81	1500
C = 200.0	0.81	0.81	0.81	1500

Table 2: Results with different values of C

C is the penalty of the errors. You can use it to define how much you want to avoid misclassifying for each trainingsexample. The larger the value the smaller the gap.

	precision	recall	f1-score	support
C = 0.1, G = 0.7	0.76	0.54	0.42	1500
C = 0.5, G = 0.7	0.83	0.83	0.83	1500
C = 0.75, G = 0.7	0.83	0.83	0.83	1500
C = 1.0 G = 0.0005	0.24	0.49	0.32	1500
C = 1.0 G = 0.9	0.84	0.84	0.84	1500
C = 1.0 G = 1.5	0.83	0.83	0.83	1500

Table 3: Results with radial basis function

My final version is using Radial basis function with C = 1.0 and G = 0.9. This is based on testing different settings. As features I use bigrams where stopwords are ignord. The bigrams are bigrams of the stemmed tokens

using the Snowball NLTK stemmer[1]. For creating the vector I use the TF-idf vectorizer with only tokens with a df higher then 0.05. The results can be found in the table below:

	Precision	recall	f1-score	Support
neg	0.81	0.89	0.85	731
pos	0.89	0.80	0.84	769
avg / total	0.85	0.85	0.85	1500

Table 4: Results best performing settings

2 K-Means

kmeans_chrispool.py

I first calculated the most common label in each cluster, the data was not so good distributed:

```
0 [('camera', 1245), ('health', 3), ('software', 3), ('books', 2), ('dvd', 2)]
1 [('books', 1679), ('dvd', 37), ('software', 28), ('health', 9), ('music', 5)]
2 [('music', 1680), ('dvd', 56), ('software', 15), ('books', 13), ('camera', 3), ('health', 1)]
3 [('dvd', 1535), ('books', 128), ('music', 81), ('software', 17), ('health', 6), ('camera', 4)]
4 [('software', 1351), ('camera', 94), ('health', 75), ('dvd', 9), ('books', 7), ('music', 3)]
5 [('health', 1906), ('camera', 653), ('software', 501), ('dvd', 361), ('music', 231), ('books', 171)]
```

I tried several things to calculate the confusion matrix but I found it a hard task. I wrote a script that calculates the most common label in each cluster as can be found in the output above.

	Books	Music	Software	DVD	Camera	Health
Books	1779	13	5	38	2	163
Music	7	1695	2	51	0	245
Software	37	15	1353	7	3	500
Dvd	49	67	10	1497	2	375
Camera	0	3	79	4	1258	655
Health	11	2	70	0	3	1914

Table 5: Confusion matrix for best settings (not the same run as distribution output

There seems to be a lot of confusion in the health cluster, a lot of documents are clustered in the health cluster. Looking at the data I cannot see what the cause is. Maybe the health class is to general.

I tried several settings: The clustering with 10 iterations, not using stopwords and using the Snowball stemmer

Settings	**Homogeneity**	**Completeness**	**V-measure**	**Adj. Rand-Index**
Iter=1, stopwords = false, stemmer = true	0.602	0.632	0.617	0.507
iter=10, stopwords = false, stemmer = true	0.627	0.657	0.641	0.539
iter=100, stopwords = false,stemmer = true	0.632	0.661	0.646	0.550
iter=1, stopwords = true, stemmer = true	0.581	0.602	0.591	0.490

Table 6: Different settings

got the best results although still not great. The randindex of 0.657 indicates that the two clusterings (gold labels and cluster index) are 0.657 similair, so the largest category in each cluster makes up on average 0.657% of each cluster.

[1] http://www.nltk.org/howto/stem.html

2.1 Binary clustering

kmeansBinary_chrispool.py

I tried to cluster the documents in the categories music and health. The clustering using the category labels worked pretty good as can be seen in the table below. The clustering using sentiment labels was very bad.

Settings	**Homogeneity**	**Completeness**	**V-measure**
Iter=1, stopwords = true, stemmer = true, gold_label = category	0.786	0.788	0.787
iter=10, stopwords = true, stemmer = true, gold_label = category	0.782	0.784	0.783
iter=10, stopwords = true, stemmer = true, gold_label = sentiment	0.0	0.0	0.0

Table 7: Binary clustering

the score of 0 using the sentiment label means that in both clusters the same amount of labels are present.

I tried other things to improve the clustering on sentiment. I used two lists with positive and negative words and replaced each positive word with ¡POS¿ and each negative word with ¡NEG¿ but that resulted only in a bit higher score. Using the counts of each token instead of TF-IDF also did not improve the system.

```
Homogeneity: 0.082
Completeness: 0.112
V-measure: 0.095
Adjusted Rand-Index: 0.070
```