ML2016 HW4 Unsupervised Clustering & Dimensionality Reduction

December 9, 2016

0.1 Most common words

The following table is the most common word in each cluster using the setting of my kaggle best (TF-DF + autoencoder, discussed in the third section) if I set the number of clusters to be 20.

cluster	most common word	count	Number of samples
0	mac	471	2776
1	linq	860	884
2	ajax	744	742
3	matlab	827	828
4	svn	602	1040
5	visual	720	766
6	bash	665	749
7	sharepoint	738	766
8	wordpress	876	910
9	magento	873	877
10	drupal	843	868
11	hibernate	880	894
12	qt	623	624
13	cocoa	58	1932
14	oracle	773	803
15	apache	613	957
16	spring	804	1050
17	haskell	723	822
18	scala	803	829
19	excel	874	883

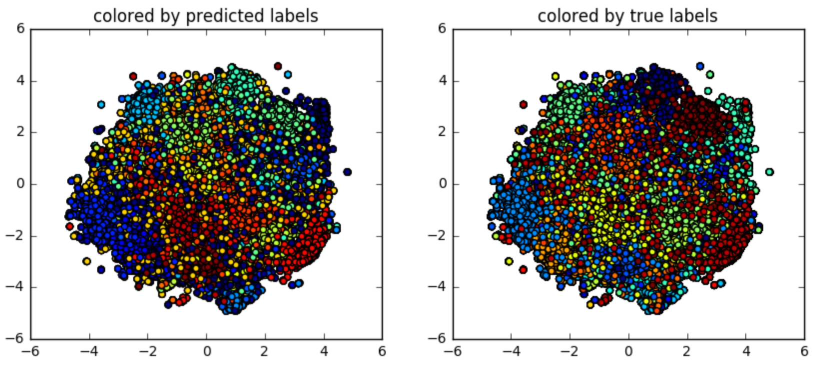
It can be seen that, these most common words happen to be the underlying tags in most cases, with only two exceptions: `osx` (somewhat being represented by `mac` in cluster 0) and `visual-studio` (somewhat being represented by `visual` in cluster 5).

The first 6 common words in cluaster 0 are: `mac` (471), `x` (357), `os` (344), `cocoa` (291), `application` (153), and `c` (131). I think this is because people usually use tag `os-x` or `os x` while asking questions related to `osx`. The first 6 common words in cluaster 5 are: `visual` (720), `studio` (694), `project` (73), `c` (47), `solution` (40), and `projects` (36). Again, people usually use `visual studio` or `visual-studio` while asking these questions, and my tokenizer split it into two distinct words. Maybe a more sophisticated tokenizer that takes these kind of connected words into account will help.

The most common word `cocoa` in cluster 13 only appears 58 times. Other common words are: `variable` (47), `value` (43), `object` (37), `values` (34), and `text` (32). It's hard to say what does this cluster mean since these common words are very general and can be related to many fields of computer programming. This cluster seems to be a bad cluster.

0.2 Visualization

The visualization of data with predicted and true labels are shown below. I use the setting of my kaggle best (TF-DF + autoencoder). The code is 512-dim. I use PCA and take the first 64 principle components, and then use t-SNE to project the 64-dim vector onto a 2-D space. It can be seen that the two figures are similar since most of the data points still aggregate together.



0.3 Compare Feature Extraction Methods

First, I tokenize all the 20000 titles and compute their bag-of-word representations and TF-IDF representations. After removing some stopwords, I also remove words that appear in less than or equal to 20 titles. The resulting data frames are both 20000-by-908. Then, I use these representations directly as features and apply a K-Means algorithm for clustering. In order to get higher $F_{0.25}$, I choose the number of clusters to be 40 instead of 20.

0.3.1 Bag-of-words

$$TP = 131864; TN = 4462087; FN = 118828; FP = 287221$$

$$\text{Precision} = 0.314647; \text{Recall} = 0.471403; F_{0.25} = 0.322264$$

0.3.2 TF-IDF

$$TP = 100716; TN = 4386949; FN = 149976; FP = 362359$$

$$\text{Precision} = 0.217494; \text{Recall} = 0.401752; F_{0.25} = 0.223524$$

It can be seen that the simplest bag-of-words representation achieves higher score than TF-IDF representation. I think the reason may be: the important words for a certain tag usually appear quite often in the title of these stackoverflow questions, and hence tend to get a smaller IDF than unimportant words. Based on this assumption, I also try **TF-DF**. That is, I take the (log-scaled) document frequency directly. After removing some stopwords and rarely-seen words, the resulting data frame is also 20000-by-908. Then, I use these **TF-DF** representations directly as features and apply a K-Means algorithm with number of clusters being 40.

0.3.3 TF-DF

$$TP = 118177; TN = 4297656; FN = 132515; FP = 451652$$

$$\text{Precision} = 0.207390; \text{Recall} = 0.471403; F_{0.25} = 0.214455$$

The score is slightly lower than TF-IDF approach. It seems that the TF-IDF representations still make more sense than TF-DF even if we are considering titles.

0.3.4 TF-DF + autoencoder (Kaggle best)

The above results are not good because the naive frequency-based features are not good text representations. So I implement an autoencoder. In this case, I only remove words that appear in less than or equal to 3 titles and the resulted data frame is 20000-by-3162. I train a `3162-1024-512-1024-3162` autoencoder to obtain a 512-dim representation for each title and apply a K-Means algorithm with number of clusters being 40.

$$TP = 107297; TN = 4711960; FN = 143395; FP = 37348$$

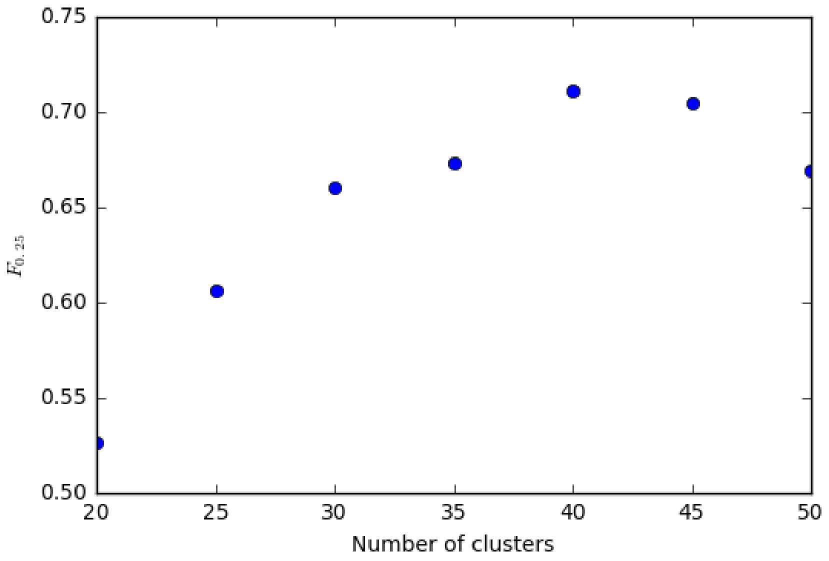
$$\text{Precision} = 0.741795; \text{Recall} = 0.428003; F_{0.25} = 0.711127$$

It seems that the autoencoder indeed find good representation for these titles.

0.4 Compare Cluster Numbers

Since the performance metric is defined as $F_{0.25} = \frac{17 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + 16 \cdot \text{recall}} = \frac{17 \cdot TP}{17 \cdot TP + FN + 16 \cdot FP}$.

It can been seen that the precision is much important than recall. More specifically, we want a smaller FP. So, we can choose the number of clusters larger than 20 to achieve higher score. The following figure shows the relation between number of clusters and score using the setting of my kaggle best (TF-DF + autoencoder) and a fixed `random_state` in the `KMeans` function. It explains why I choose number of cluster to be 40 for my kaggle submission.


$F_{0.25}$
Number of clusters