DSP Final Project
A modification of latent class analysis

B03901003 許晉嘉 B03901133 施順耀

June 27, 2016

1 Introduction

In the course about "Linguistic Processing and Latent Topic Analysis", we study some methods and algorithms to deal with classification and clustering. After the course, we are much interested in the performance of those methods and more extension versions. Thus, we start to read some papers given in the references.
However, most of the methods are based on complicated math model or need to train the model with multiple iterations. Therefore, we are wondering whether there is some relatively simple method with low complexity(no need to have multiple iterations).
Then, we come up with an idea similar to the normal latent class method. We changed the math model into a probability distribution. And, we can also turn each unknown document into a probability distribution. Then, we can calculate the similarity of them. Those models with higher similarity will be more probable topic of that unknown document.
Finally, we implement the method and collect several datas to train the model and test the performance. Surprisingly, the performance is not quite bad. In most of the case, it turns out to have more than 90% accuracy with our method.

2 Modification of latent class analysis

2.1 Main ideas and Math model

For each class(topic), we first collect several documents known to belong to this class(In big data era, we assume it's applicable). We first modify the data into only words where we define word as continuous alphabets or numbers. Moreover, we think that preposition doesn't affect class much. Thus, we exclude it by discarding too short word(the length will be specified). And, also for the too long word. Since, we think that it may be some parsing error. Then, we calculate the model of this class as a words distribution. Words distribution defined as follow:

$$P[w_i] = \frac{\sum_k \#\text{ of } w_i \text{ in document } d_k}{\sum_k \sum_j \#\text{ of } w_j \text{ in document } d_k}, \text{ where } w_i \text{ denoted as i-th word}$$

Therefore, for each unknown document, we can define its words distribution in similar way:

$$Q[w_i] = \frac{\#\text{ of } w_i \text{ in this document}}{\sum_j \#\text{ of } w_j \text{ in this document}}, \text{ where } w_i \text{ denoted as i-th word}$$

Then, we are going to calculate their similarity. For similarity, we think that if the document has similar topic with that class, their words distribution will be similar. Then, for a word w_i, if $P[w_i]$ is different to $Q[w_i]$ very much, it implies that this document probably doesn't have similar topic with this model. Thus, we define a cost function for each word where cost function should be a function with positive increasing slope. Therefore, we choose the square of difference to be the cost function.
Moreover, if some word with zero probability in model but with non-zero probability in document, or vise versa, we think that it may be some key words of that class. Thus, we add more penalty on it. But, to avoid adding penalty to some rare word, the non-zero probability should exceed some chosen threshold. Finally, the cost function of a word defined as follow:

$$C(w_i) = \begin{cases} \texttt{penalty} \times (P[w_i] - Q[w_i])^2, & \text{if} \quad P[w_i] \geq \texttt{threshold} \cap Q[w_i] = 0 \\ & \text{or } Q[w_i] \geq \texttt{threshold} \cap P[w_i] = 0 \\ (P[w_i] - Q[w_i])^2, & \text{otherwise} \end{cases}$$

Then, similarity is defined as follow:

$$\texttt{Sim} = \frac{1}{\sum C(w_i)}$$

That is, less total cost of word, more probable to have similar topic to that class.

2.2 detals of procedure

1. We first collect several known topic data and divide them into two set(one for building model, the other for testing).

2. For each topic, we calculate its words distribution based on those known topic data.

3. For each testing data, we calculate its words distribution also.

4. Calculate the similarity of the words distribution with each model.

5. First several models with highest similarity will have most probable similar topic with the testing data.

6. Since the topic of testing data is actually known, we can calculate the accuracy of our method.

2.3 details of implementation

1. Words distribution:
   In case that storing a word costs much memory, we decide to store its **hash value** instead. Since in natural language, there are about only tens of thousands of meaningful words. And the chosen hash space is $10^9 + 7$(which is a prime!). Thus, the probability of collision is very low(Actually, expected collision is less than 1).

2. Building model:

   - `WordsDistrib read( path, rangel, ranger )`:
     From the `path`, we read in a pure data. And, calculate out the words distribution of this data. And we only count the word with length between `rangel` and `ranger`. Moreover, the words distribution is composed of pair of hash value and probability **sorted by hash value** which will improve following implementation. Therefore, the total time complexity of this function is $O(\text{length of document} + N \lg N)$ where N is # of words.
   - `initModel( model, filelist )`:
     From the `filelist`, we will have bunch of path linked to data belonging to this topic. Thus, we call the read function to help us calculate out the words distribution. Since the words distribution of each document is sorted by hash value. We can merge two words distribution in $O(N)$ where N is # of words. Then, store the final words distribution into `model`. Therefore, total complexity of this function is $O(M \times N)$ where M is # of documents.

3. Testing datas:

   - `prob calProb( data, model )`:
     Since both words distributions of `data` and `model` are sorted by hash value, we can calculate out the similarity in $O(N)$ where N is # of words.
   - `models testModel( path )`:
     Calculate the words distribution from the pure data of `path`. Then, calculate the similarity with each models. Finally, find out the most probable ones. The time complexity of this function is $O(M \times N)$ where M is # of models and N is from the complexity of `calProb` function.

4. Overal analysis:

   - For building model: the total complexity is linear to the total length of data and linearithmic to the # of words.
   - For testing data: the time complexity to calculate with each model is linear to the length of data.

3 Experimental results

In general, our method performs about 90% accuracy. But, there are still some parameters we can improve its performance.

1. Adjusting `penalty` and `threshold`

`threshold` \ `penalty`	1	10	100	1000	10000
10^{-2}	58.56%	92.84%	92.29%	92.09%	92.00%
10^{-3}	58.56%	92.92%	92.81%	92.79%	92.78%
10^{-4}	58.56%	92.99%	92.95%	92.90%	92.91%
10^{-5}	58.56%	92.98%	92.96%	92.94%	92.95%
10^{-6}	58.56%	92.98%	92.95%	92.95%	92.95%

   - First, if `penalty`=1, the cost function isn't affected by the `penalty`. Thus, the `threshold` is no effect.
   - For `penalty`, we can find that setting `penalty` too high will make some models lose its advantage easily. And, setting it too low won't take enough effect to discard non-similar topics. Thus, `penalty` should be set carefully. In our case, it's best to set `penalty` into 10.
   - For `threshold`, we can find that setting `threshold` too high will make it hard to take the `penalty`. Since it's almost impossible to find a word appearing 1% in a document. And, setting it too low will make some models take the `penalty` easily. Thus, `threshold` should be also chosen carefully. In our case, it's best to set `threshold` to between 10^{-4} and 10^{-5}.

2. Adjusting `rangel` with `ranger`=255

`rangel`	0	1	2	3	4
	89.07%	91.42%	92.22%	92.92%	92.19%
`rangel`	5	6	7	8	9
	91.11%	89.75%	88.24%	85.03%	83.27%

   - As our assumption, pruning out some preposition(short word) will improve the performace.
   - Cutting out words with length 3 or more will likely discard some very important key words. Therefore, the accuracy is decreasing.
   - In our case, it's best to discard words with length shorter than 3.

3. Adjusting `rangel` and `ranger`

`rangel` \ `ranger`	1	2	3	4	5	6	7
1	22.33%	40.33%	56.30%	66.99%	75.39%	82.54%	86.70%
2		39.41%	58.56%	68.96%	77.06%	83.43%	88.09%
3			57.65%	71.09%	79.20%	85.80%	89.78%
4				63.49%	75.85%	84.02%	88.56%
5					67.33%	81.27%	87.11%
6						73.30%	83.43%
7							74.02%

- Most of the case, including more kinds of length will improve performance.
- Specially, from $(1, 3)$ to $(2, 3)$, the accuracy increases, and also several similar cases(e.g. $(2, 4)$ to $(3, 4)$). It's confirmed again that triming out some short words with length less than about 3 will improve the performance.

4 Conclusions

1. We derive a similar method with simple math model to deal with latent class problem. The method just needs almost linear time to build the model and linear time to calculate the similarity between data and model.

2. With the method, we collect several testdata and get overall accuracy about 90%.

3. With adjusting some parameters, we can further improve the performance and find out some reasonable explanation about variation of the accuracy.

5 Further improvement

1. In our case, the model is based on some chosen data. If the data is too extreme or wrongly classified, it may ruin our model. Thus, it's better to let the model reject some data or lower the weight of this data.

2. For the parameters `penalty`, `threshold`, `rangel`, `ranger`, it's better to put them into model and adjust them with datas. Because, for different topic, the length of key word may differ. And, for some topic, some key word may appear extremely lots of times.

3. There may be more useful adjustable parameter. Maybe the cost function can be changed into a parabola function. Maybe when calculating the words distribution, the number of a word can take the form of logarithm first.

6 Appendix

1. All of our project can be found here:
   `https://github.com/eddy1021/DSP_Final_Project`
   Also uploaded to ceiba.

2. Usage of our program:

   - Build: `make all`
   - Build model:
     `./train -default`
     `./train -wordLen #rangel #ranger`

- Test data:
  `./test -default`
  `./test [-penalty #penalty] [-threshold #threshold] [-wordLen #rangel #ranger]`

7 References

- testing datas: `http://qwone.com/~jason/20Newsgroups/`(provided by TA)
- Peter G.M. van der Heijden(1992), The EM Algorithm for latent class analysis with equality constraints
- Miin-Shen Yang(2003), Estimation of parameters in latent class models using fuzzy clustering algorithms
- Thomas Hofmann, Probabilistic Latent Semantic Analysis
- Thomas Hofmann, Probabilistic Latent Semantic Indexing