Project Documentation

Lijuan Geng, Kefan Chen

Working towards the vision of a general analysis engine, this project has implemented two PLSA extensions for MeTA to support more flexible and complex topic modeling tasks.

Probabilistic Latent Semantic Analysis (PLSA) is a mixture model with k Unigram Language Models (k topics). PLSA uses Expectation-Maximization (EM) Algorithm to compute the Maximum Likelihood (ML) of mixture models.

ML estimate discovers the following topical knowledge from text data: k word distributions, i.e., k topics, and proportion of each topic in each document. EM algorithm is a general algorithm for computing ML likelihood of mixture models. It’s a hill-climbing algorithm which can converge to a local maximum depending on the initialization. The EM algorithm has two steps, namely Expectation step (E-step) and Maximization step (M-step). E-step augments data by predicting values of useful hidden variables, while M-step exploits the augmented data to improve estimate of parameters.

1. Support PLSA with prior

The standard PLSA blindly listens to data by using maximum likelihood to fit data as much as possible and get insights about data. In practice, when PLSA is applied to analyze text data, we might have additional knowledge to guide the analysis. For example, a user might have expectations about which topics to analyze. “Mount Rainier” is likely expected to be a topic in documents about Seattle outdoor/nature activities. We can incorporate such knowledge as priors of PLSA.

In this case, Maximum a Posteriori (MAP) estimate should be used instead of the standard ML likelihood estimate.

$$\Lambda^* = \arg\max_{\Lambda} p(\Lambda) p(Data \mid \Lambda)$$

$p(\Lambda)$ is used to encode constraints and preferences. In this project we focus on using a conjugate prior where $p(\Lambda)$ favors a Λ with topics that assign high probabilities to some particular words. The MAP estimate (with conjugate prior) can be computed using EM algorithm with smoothing to reflect prior preferences:

E-step:

$$p(z_{d,w} = j) = \frac{\pi_{d,j}^{(n)} p^{(n)}(w \mid \theta_j)}{\sum_{j'=1}^{k} \pi_{d,j'}^{(n)} p^{(n)}(w \mid \theta_{j'})}$$

M-step:

$$\pi_{d,j}^{(n+1)} = \frac{\sum_{w \in V} c(w,d) p(z_{d,w} = j)}{\sum_{j'} \sum_{w \in V} c(w,d) p(z_{d,w} = j')}$$

$$p^{(n+1)}(w \mid \theta_j) = \frac{\sum_{d \in C} c(w,d) p(z_{d,w} = j) + \overbrace{\mu p(w \mid \theta'_j)}^{\text{Pseudo counts of w from prior } \theta'}}{\sum_{w' \in V} \sum_{d \in C} c(w',d) p(z_{d,w'} = j) + \underbrace{\mu}_{\text{Sum of all pseudo counts}}}$$

It's important to note that there's only one conjugate prior θ'_j in the model, and the conjugate prior is a word distribution with words that the user is expecting to be included in a topic. The prior is applied to smooth one word distribution θ_j so that words in the prior would get a higher probability in θ_j, and words that are not in the prior would get a lower probability.

The sum of all pseudo counts μ determines how much $\boldsymbol{\theta}_j$ would be influenced by the prior. When μ =0, it's essentially the same as if there's no prior. An increased value of μ creates a bigger impact from prior. For the extreme case when μ is $+\infty$, $\boldsymbol{\theta}_j$ is forced to be same as the prior $\boldsymbol{\theta}'_j$, and all the probabilities in the word distribution would concentrate on words in the prior and all the other words have zero probability.

Documentation

"plsa_prior" folder contains code and data for this part of the project.

- Dataset is located in "data" folder. Dataset from MP3 is used for demonstration due to its manageable vocabulary size and number of topics.
- Algorithm results are included in a Jupyter notebook "result.ipynb", which also shows the command to run the program: !python plsa_with_prior.py.
- plsa_with_prior.py is the main source code with many helper functions and the following important functions that are directly related to how conjugate prior is incorporated into PLSA:
    - expectation_step(self): E-step
    - maximization_with_prior(self,number_of_topics,pseudo_count, conjugate_prior): M-step
    - plsa_with_prior (self, number_of_topics, max_iter, epsilon, pseudo_count, conjugate_prior): implement PLSA algorithm with conjugate prior.
- To experiment with different prior parameters, go to main() in plsa_with_prior.py and make changes in the "prior parameters" section.

Results & Discussions

Here are some results from our implementation of supporting PLSA with conjugate prior. We used a prior distribution with words “mount” and “rainier”.

The 1st scenario shows the default case when pseudo_count is 0, which means there’s zero influence from prior. In Topic-Word-Probability, topic0 gives high probabilities to “mount”, “rainier” and “seattle”, and the topic1 has high probabilities for “chicago”, “tower” and “willis”. In Likelihoods, we can see the log likelihood of the algorithm is increasing, which aligns well with our optimization goal. Document-Topic-Probability snippet shows some documents’ topic coverage of the two topics, with the first two documents covering only topic0 and the remaining documents covering both topics.

```
!python plsa_with_prior.py
```

```
Vocabulary:  ['chicago', 'mount', 'rainier', 'seattle', 'tower', 'willis']
Vocabulary size: 6
Number of documents: 1000

----------- Conjugate Prior -------------
pseudo_count:  0
Prior:  {'rainier': 0.5, 'mount': 0.5}

---------- Likelihoods -----------
[-179246.2852669692, -178449.54987504674, -177770.94669127284, -176844.94469549146, -175310.11061781805, -17278
7.0857764463, -169206.0297564499, -165311.18459178187, -162306.1990261008, -160650.63809936595, -159925.0441617
8263, -159608.1354495811, -159445.83770824203, -159348.55039436577, -159282.0041532881, -159231.91965367508, -1
59191.94949092326, -159158.94622813718, -159131.02207509466, -159106.93038516233, -159085.89837460915, -159067.
38624036388, -159050.87780693674, -159035.84927311345, -159022.04784534522, -159009.77314975005, -158999.310340
83499, -158990.42244037802, -158982.6770890425, -158975.79131916442, -158969.71172953036, -158964.37553868117,
-158959.60145454764, -158955.20947258166, -158951.1455622167, -158947.44926936718, -158944.0277717352, -158940.
69638392775, -158937.40931803102, -158934.2846739519, -158931.51988617476, -158929.14061557245, -158927.1100269
36, -158925.3793137941, -158923.8683089509, -158922.48818225577, -158921.13187158303, -158919.6955297982, -1589
18.14905057463, -158916.61637278512]

-------------- Document-Topic-Probability snippet ---------------
[[1.         0.        ]
 [1.         0.        ]
 [0.2924568  0.7075432 ]
 [0.3657417  0.6342583 ]
 [0.10222804 0.89777196]
 [0.40395803 0.59604197]
 [0.99755613 0.00244387]
 [0.89925624 0.10074376]
 [0.89016864 0.10983136]
 [0.90095236 0.09904764]
 [0.31746348 0.68253652]
 [0.75287681 0.24712319]]

-------------- Topic-Word-Probability ---------------
[[0.00091961 0.29874077 0.3965925  0.30350957 0.00017325 0.0000643 ]
 [0.29680591 0.05498912 0.00055507 0.05124555 0.29617518 0.30022917]]
```

The 2nd scenario has a pseudo_count of 8000, which means across all the documents in the corpus, we have introduced an additional 8000*0.5 “mount” and 8000*0.5 “rainier” to increase the probabilities of the two words in the topic that has been influenced by prior. Topic-Word-Probability clearly shows that p(mount|topic0) and p(rainier|topic0) are both higher than in the 1st scenario. Meanwhile in Document-Topic-Probability, we have observed a slight decrease of topic0 coverage in documents that cover both topics. Because as the probabilities of “mount” and “rainier” are getting higher in topic0, the probabilities of other words are getting lower. For documents that only cover topic0, a 100% coverage of topic0 is still the best for maximum document probability. However, for documents covering both topics, In order to optimize for Maximum a Posteiori, the algorithm has to shift some coverage to other topics to maintain a high overall probability.

```
!python plsa_with_prior.py
```

```
Vocabulary:  ['chicago', 'mount', 'rainier', 'seattle', 'tower', 'willis']
Vocabulary size: 6
Number of documents: 1000

----------- Conjugate Prior -------------
pseudo_count:  8000
Prior:  {'rainier': 0.5, 'mount': 0.5}

---------- Likelihoods -----------
[-179193.8609762327, -177824.92871615326, -175848.4099481389, -172754.32753138235, -168651.40621190672, -16465
1.0252244155, -161968.67800107243, -160648.42646106877, -160050.8143687899, -159759.77253195472, -159598.051554
6376, -159496.09212296887, -159426.37593815252, -159375.88047189458, -159337.4775879929, -159307.53243984358, -
159284.39643516272, -159267.0428617333, -159254.00220087904, -159243.88987558533, -159236.15349294353, -159230.
47292053525, -159226.21041806744, -159222.5286281167, -159218.70511461512, -159214.8330886944, -159211.65150335
108, -159209.45209560386, -159207.9118253011, -159206.6963147896, -159205.69922282346, -159204.8711971152, -159
204.17765290826, -159203.59243651037, -159203.09542912652, -159202.67098148525, -159202.30678743366, -159201.99
306066005, -159201.72193016816, -159201.486994664, -159201.2829925025, -159201.10555574755, -159200.9510255451
6, -159200.8163123425, -159200.69878899894, -159200.5962081746, -159200.50663772848, -159200.42840958235, -1592
00.36007875376, -159200.30039012188]

-------------- Document-Topic-Probability snippet ---------------
[[1.         0.        ]
 [1.         0.        ]
 [0.27795107 0.72204893]
 [0.34663243 0.65336757]
 [0.09661426 0.90338574]
 [0.38711258 0.61288742]
 [0.98844277 0.01155723]
 [0.89723368 0.10276632]
 [0.88623365 0.11376635]
 [0.89766441 0.10233559]
 [0.31798288 0.68201712]
 [0.74544391 0.25455609]]

-------------- Topic-Word-Probability ---------------
[[0.00000182 0.33429933 0.41510741 0.25059102 0.00000031 0.00000012]
 [0.29420851 0.04825171 0.00004188 0.06798233 0.29281153 0.29670402]]
```

The 3rd scenario shows an extreme case with pseudo_count of 50000. The influence of the prior is so heavy that topic0 is forced to be same as the prior distribution, i.e., all the probabilities are concentrated on “mount” and “rainier”. In Document-Topic-Probability, doc0 and doc1, which previously covered only topic0, have now also been forced to shift some coverage for topic1 to avoid near zero word probabilities in topic0. Therefore, when applying conjugate prior to influence topic modeling, we would like to balance the pseudo_count of the prior, so the prior’s intention is achieved but other important words in the topic are not completely ignored or unnecessarily forced to other unrelated topics.

```
!python plsa_with_prior.py
```

```
Vocabulary:  ['chicago', 'mount', 'rainier', 'seattle', 'tower', 'willis']
Vocabulary size: 6
Number of documents: 1000

----------- Conjugate Prior -------------
pseudo_count:  50000
Prior:  {'mount': 0.5, 'rainier': 0.5}

---------- Likelihoods -----------
[-183643.8325391589, -179699.17457387422, -174580.40220929458, -170742.22629301323, -169245.06604671778, -16880
0.9856401644, -168642.3626794908, -168572.51395115105, -168531.6047248377, -168504.1156826725, -168483.95902227
494, -168467.97744428375, -168454.6224351626, -168443.13155706233, -168433.05737644824, -168424.06443381935, -1
68415.87902845533, -168408.2947185101, -168401.171025132, -168394.41507205457, -168387.96390371135, -168381.782
44673228, -168375.86865446754, -168370.24654334862, -168364.94684525576, -168359.9912456436, -168355.3879900133
5, -168351.13492655207, -168347.22365555196, -168343.64008226444, -168340.3626424108, -168337.3632734259, -1683
34.61216954066, -168332.08243056925, -168329.74929536245, -168327.5835014196, -168325.54808024564, -168323.6088
3944685, -168321.75206224, -168319.98245821602, -168318.29576175782, -168316.6715111475, -168315.10745806023, -
168313.63251690645, -168312.26003396884, -168310.97622800572, -168309.77633899852, -168308.66738777465, -16830
7.64231156022, -168306.6659277268]

-------------- Document-Topic-Probability snippet ---------------
[[0.66740808 0.33259192]
 [0.61556083 0.38443917]
 [0.18559812 0.81440188]
 [0.20596621 0.79403379]
 [0.07451818 0.92548182]
 [0.23062809 0.76937191]
 [0.63429469 0.36570531]
 [0.6533448  0.3466552 ]
 [0.59550521 0.40449479]
 [0.63477369 0.36522631]
 [0.27289833 0.72710167]
 [0.47737757 0.52262243]]

-------------- Topic-Word-Probability ---------------
[[0.         0.47164013 0.52835985 0.00000002 0.         0.        ]
 [0.22467333 0.04201575 0.00360639 0.2795219  0.22360515 0.22657747]]
```

2. Support interpretation of any topic in any given context of a collection of articles

Documentation

The standard PLSA algorithm will derive word distribution inside each topic based on maximum likelihood via the EM algorithm. This is helpful in some cases especially when we want to retrieve the most representative topics. However, sometimes we do want the model to come up with topic weights of a documentation based on certain predefined topics. In this case, we will need a model or software which is able to take in user inputs as the topic distribution and try to maximize the likelihood by running an EM algorithm on topic weights for each document.

The actual implementation follows very closely with the above high level idea and the actual usage is also illustrated in the jupyter notebook attached in the source code. First of all, the user will need to specify their desired topics in a separate file. Then the user needs to instantiate the Corpus object with the target topic file path. One good thing about the current implementation is that the same class will support both use cases whether predefined topics are supplied or not, which makes the usage easier in some sense. Then once the document file and topic file are loaded, the actual EM algorithm will start to maximize the likelihood by improving the document topic weights (as well as the topic word distribution if topics are not predefined). After running the EM algorithm, we will be able to examine the exact topic weight. We also implemented nice printing functions within the Corpus function to better check the results.

For the sample usage, please check topic_interpretation.ipynb in the github repository. Some sample commands include:

```
from topicinterpretation import Corpus

documents_path = "data/test.txt"
topic_path = "data/test_topics_1.txt"
corpus = Corpus(documents_path, topic_path)
corpus.build_corpus()
corpus.build_vocabulary()
```

```
corpus.em(2, 100, 0.001)
corpus.print_topics()
corpus.print_topic_distribution()
```

Results & Discussions

Let us check the results we are getting below.

1. Results with normal PLSA.

This is using the standard PLSA algorithm to derive topic distribution and topic weights based on maximum likelihood. We can see that since the documents are quite distinct between two topics, each topic focuses on distinct words and each document has a very clear topic emphasis among the two.

```
topic 1
chicago: 1.1821032490738054e-05
mount: 0.2987910784204859
1: 0.0
0: 0.001071497414360191
rainier: 0.39658931157524957
tower: 2.8969626199439534e-07
seattle: 0.3035359367155584
willis: 6.514558976723548e-08

topic 2
chicago: 0.2975682530561838
mount: 0.05448114299801302
1: 0.0009226990461883934
0: 0.0
rainier: 1.1620953775706166e-06
tower: 0.29616558450000513
seattle: 0.050758488202532835
willis: 0.30010267010170083
```

```
document 1
topic 1: 1.0
topic 2: 1.0511715438144668e-80

document 2
topic 1: 1.0
topic 2: 5.857162885519661e-95
```

```
document 3
topic 1: 8.34624633498599e-19
topic 2: 1.0

document 4
topic 1: 1.3264558463968237e-19
topic 2: 1.0

document 5
topic 1: 7.448535543282083e-27
topic 2: 1.0

document 6
topic 1: 2.140098935900383e-31
topic 2: 1.0
```

2. Results with predefined topics with target words.

This is using predefined topics which have similar word distribution as the real identified topics. We can see in this case for each document, it still clearly favors one topic over the other, this is because these documents have very distinct topic concentration. However the topic weight is not as biased as the one we see using normal PLSA, since PLSA is optimizing another set of the parameters to achieve even better likelihood.

```
topic 1
chicago: 1.4285697959202333e-07
mount: 0.571428061225073
1: 1.4285697959202333e-07
0: 1.4285697959202333e-07
rainier: 0.4285710816330496
tower: 1.4285697959202333e-07
seattle: 1.4285697959202333e-07
willis: 1.4285697959202333e-07

topic 2
chicago: 1.4285697959202333e-07
mount: 1.4285697959202333e-07
1: 1.4285697959202333e-07
0: 1.4285697959202333e-07
rainier: 1.4285697959202333e-07
tower: 0.4285710816330496
seattle: 1.4285697959202333e-07
willis: 0.571428061225073
```

```
document 1
topic 1: 0.9999998551109679
topic 2: 1.448890320813485e-07

document 2
topic 1: 0.9999999892024466
topic 2: 1.0797553488808389e-08

document 3
topic 1: 0.057970825709337426
topic 2: 0.9420291742906625

document 4
topic 1: 0.09374989427223838
topic 2: 0.9062501057277617

document 5
topic 1: 0.05970128117242958
topic 2: 0.9402987188275704

document 6
topic 1: 0.09230756091214762
topic 2: 0.9076924390878525
```

3. Results with predefined topics with mixed words.

This is using predefined models which actually have mixed words in each topic compared to PLSA results. In this case, each of the topics actually contains high probability words which are supposed to be in different topics in normal PLSA. In the result, we can clearly see that the weights among two topics for all documents are much closer compared to the above twp results, as each topic will contain some desired words for all documents.

```
topic 1
chicago: 0.125
mount: 0.499999625000375
1: 1.2499987500012498e-07
0: 1.2499987500012498e-07
rainier: 0.37499975000024993
tower: 1.2499987500012498e-07
seattle: 1.2499987500012498e-07
```

```
willis: 1.2499987500012498e-07

topic 2
chicago: 1.2499987500012498e-07
mount: 1.2499987500012498e-07
1: 1.2499987500012498e-07
0: 1.2499987500012498e-07
rainier: 1.2499987500012498e-07
tower: 0.37499975000024993
seattle: 0.125
willis: 0.499999625000375
```

```
document 1
topic 1: 0.6400007314169397
topic 2: 0.35999926858306025

document 2
topic 1: 0.7000004258345061
topic 2: 0.29999957416549405

document 3
topic 1: 0.2699999301985811
topic 2: 0.730000069801419

document 4
topic 1: 0.3599999116784573
topic 2: 0.6400000883215428

document 5
topic 1: 0.30999926722488363
topic 2: 0.6900007327751163

document 6
topic 1: 0.3799997416716714
topic 2: 0.6200002583283286
```

The above results and observations actually match with our expectation about predefined topics, which also justifies our implementation.