MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2017

Assignment 1, Due: 9am on Tuesday Sep 26. Upload pdf or scan to Stellar.

Problem: Bag of Words model

In this section, you will use a simple Bag-of-Words feature set to solve the sentiment analysis task, by modeling it as a binary classification problem. The goal is to make you understand the impacts of using different feature sets and settings of classification models on a real world task.

As a part of this problem, you will explore the performance of different feature sets. You will observe the impact of training size on performance. Specific tasks are explained in detail below.

For each model, you will have many hyper-parameters and feature-set settings to play with. *To pick the best setting, you must evaluate each run on the dev set. The best-dev performing model will then be used to evaluate the test set. While stating performances, mention absolute accuracies on both the Dev and Test set.*

Set Up: The data for this assignment consists of excerpts of movie reviews from `rottentomatoes.com` consolidated in the Stanford Sentiment Tree Bank [1]. Each data point in the dataset will look like

$$rating\ excerpt$$

where $rating$ has been normalized to $\{0, 1\}$, and $excerpt$ is a string excerpt of the review. $rating$ and $excerpt$ are delimited by a singular space.

We have also divided the dataset into three parts — Train (located in 'data/stsa.binary.train'), Dev (located in 'data/stsa.binary.dev'), and Test (located in 'data/stsa.binary.test') of 6920, 872 and 1821 instances respectively.

Software: You will implement this assignment in Python 2.X and will be required to use the scikit_learn [2] toolbox and PyTorch `https://github.com/pytorch/pytorch`.

You should also develop your code with the following structure

```
assignment1
├── sentiment
│   ├── src
│   └── data
└── topic_model_em
```

and *only* turn in a zipped version of your src directory for this problem.

(Question 1.1) [30 points]

1. Compile all words that appear in a training set of reviews into a dictionary, thereby producing a list of d unique words. Transform each of the reviews into a feature vector of length d by setting the i^{th} coordinate of the feature vector to 1 if the i^{th} word in the dictionary appears in the review or zero, otherwise. You should observe 13789 unique words in the training corpus (located in 'stsa.binary.train'),

(using the tokenizer from CountVectorizer). (Hint: You might find CountVectorizer from sklearn useful.) *You can also use non-binary bag of words where you indicate in the vector, the number of times a word appears in the sequence. Please specify in detail, what specifications of the model you are using while reporting results.*

2. Learn a LogisticRegression classifier for this feature representation on the training set, and evaluate. You are supposed to use the scikit_learn machine learning toolbox for implementation. You will have to play around with the regularization parameter (C) for optimum performance. Pick C from the set $\{1e-1, 1e1, 1e3, 1e5, 1e7\}$. *State the performance that you get by tuning on the Dev set. Mention both Dev and Test accuracies. Save this code in a file named* ***P12.py***

3. Analyze the features i.e. parameters of the output model. Consider the parameters corresponding words which are weighted strongly by the classifier. *Argue if the top 10 words weighted by the classifier make sense in the context of the dataset.* Remove stop words from your vocabulary and run the model with this new feature space. You can use the python stop-words `https://pypi.python.org/pypi/stop-words` library to import a fixed set of stop words for English. *State your performance with this change. Save this code in a file named* ***P13.py***

4. Increase your feature space by using not just *unigrams* but also *bigrams*, i.e. add every bigram phrase in the training corpus to the feature space, along with the unigram phrases. Don't remove stop-words. *State if this improves performance as well as the performance numbers. Save this code in a file named* ***P14.py***

5. Explore using more number of n-gram features. While this explodes the feature space, you should trim it by using a threshold on the frequency of the features. You may also play around with whether or not to include stopwords as a part of these n-grams. You can also play around with C from a larger set.*State your best performance with this setting. Save this code in a file named* ***P15.py***. State the range of hyper-parameters and the range of features that you are using for the models.

6. This part needs to be implemented for the unigram model from Part 3 and your best performing model. *Plot the performance (Test Accuracy) on using just 25%, 50%, 75%, or 100% of the training data points*. You will present two separate plots, and can pick the training data points randomly. Where, X-axes represent the number of training points and the Y-axes represent the performances.

Note, that some results might seem counter-intuitive. However, these steps present a baseline that one considers for tasks in NLP.

Problem: EM Algorithm for Language Model Smoothing

In this problem, we consider smoothing language models via linear interpolation. It turns out that estimating the associated linear weights is a problem that EM can solve very well. Recall that in linear interpolation, the desired smoothed trigram model is written as a combination of maximum likelihood estimates of trigram, bigram, and unigram models as follows

$$p_\Lambda(w|u,v) = \lambda_3 p_{\text{ML}}(w|u,v) + \lambda_2 p_{\text{ML}}(w|u) + \lambda_1 p_{\text{ML}}(w)$$

where the interpolation weights $\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ satisfy $\lambda_y \geq 0$ and $\sum_{y=1}^{3} \lambda_y = 1$. Here $p_{\text{ML}}(w|u,v)$, $p_{\text{ML}}(w|v)$, and $p_{\text{ML}}(w)$ are the maximum likelihood estimates of the trigram, bigram, and unigram models, respectively, estimated on the basis of the training corpus. The interpolation weights, however, are

obtained by maximizing the log-likelihood that the smoothed model $p_\Lambda(w|u,v)$ assigns to the words in the development set, not training set. Note that the p_{ML} estimates remain unchanged during this process.

Let (w_{t-2}, w_{t-1}, w_t) specify the t^{th} trigram in the development set D. w_{t-2} and w_{t-1} are set to specific start symbols STA_{-2} and STA_{-1} whenever the trigram appears in the beginning of a sentence. In this case, the log-likelihood of the words in the development set can be written as

$$l(\text{D};\Lambda) = \sum_{t=1}^{T} \log p_\Lambda(w_t|w_{t-1}, w_{t-2}) \tag{1}$$

$$= \sum_{t=1}^{T} \log \Big(\lambda_3 p_{\text{ML}}(w_t|w_{t-1}, w_{t-2}) + \lambda_2 p_{\text{ML}}(w_t|w_{t-1}) + \lambda_1 p_{\text{ML}}(w_t) \Big) \tag{2}$$

You are asked to use the EM algorithm to find the maximum likelihood estimates (MLEs) of parameters Λ based on D, i.e., to find

$$\Lambda^* = \arg\max_{\Lambda} \log l(D;\Lambda)\,.$$

This way of modeling suggests the following method for generating documents: For each word w_t, select one of the trigram, bigram, or unigram maximum likelihood models according to the distribution Λ. Then generate w_t using that model and necessary context (for the bigram and trigram models). Thus the parameters Λ are our latent variables.

(Question 2.1) [4 points] Suppose we deviated from the procedure and instead estimated the interpolation weights on the basis of the same training corpus from which $p_{\text{ML}}(w|u,v)$, $p_{\text{ML}}(w|u)$, and $p_{\text{ML}}(w)$ are derived. Which setting of the weights Λ in $p_\Lambda(w|u,v)$ maximize the log-likelihood of the words in the training corpus? Briefly justify your answer.

(Question 2.2) [5+3 = 8 points] Now we can make use of the EM algorithm to estimate the parameters Λ based on the development set. Let $\Lambda^{(k)} = \{\lambda_1^{(k)}, \lambda_2^{(k)}, \lambda_3^{(k)}\}$ be the parameters after the kth EM iteration.

(a) *E-Step:* Provide expressions for the fractional counts $\hat{n}(\lambda_1)$, $\hat{n}(\lambda_2)$, and $\hat{n}(\lambda_3)$.

(b) *M-Step:* Now estimate the new values $\lambda_y^{(k+1)}$, $y = 1, 2, 3$ using the above fractional counts.

(Question 2.3) [3 points] Let $\Lambda^{(0)} = \{\lambda_1^{(0)}, \lambda_2^{(0)}, \lambda_3^{(0)}\}$ be the initial setting for the EM algorithm. Explain what will happen during the EM iterations if $\lambda_2^{(0)} = 0$.

Problem: EM Algorithm for Topic Models

Topic models are useful for discovering patterns in text and creating groups of words called *topics*. Recall from class that a latent topic model aims to assign a topic $z \in [1, K]$ to each word w in a given set of documents $\{d_t : t \in [1, n]\}$. We saw that the posterior probability can be modeled as:

$$p(z|w,t) = \frac{\theta_{z|t}\theta_{w|z}}{\sum_{z'=1}^{k} \theta_{z'|t}\theta_{w|z'}}$$

where $\theta_{z|t}$ and $\theta_{w|z}$ are the parameters of the topic model representing the conditional probabilities $p(z|t)$ and $p(w|z)$, respectively.

(Question 3.1) [4 points] *E-Step:* Write down the expressions for the soft counts (or fractional counts) $\hat{n}_t(z)$ and $\hat{n}(w,z)$ using the above posterior probability. You can assume the number of documents to be n and the number of words in document t to be N_t.

(Question 3.2) [4 points] *M-Step:* Using these soft counts, provide the update equations for $\theta_{z|t}$ and $\theta_{w|z}$ using MLE.

(Question 3.3) [12 points] *Programming:* The questions in this part require you to work with the Python code template provided in the folder `topic_model_em/`, and experiment you code with the data provided in `topic_model_em/data_topic_em/`.

(a) Using the expressions you derived in (1) and (2), complete the sections marked with a *TODO* in the code template to obtain a working implementation of EM for topic modeling. You can test it on the sample data included.

(b) Using the code, experiment with various settings of number of topics and number of iterations to obtain "good" topics. Can you think of a way to determine the quality of a topic obtained from the model?

(c) Run the model with 10 topics for 50 iterations. Interpret the "topics" that you obtain (Please provide a list of the top 10 words in each topic and a single word description of the topic). Are all the topics unique? Provide a short (one or two sentence) explanation for your observations.

(d) In the code, you will have noticed that the parameters (θ) of the model are initialized randomly at the start of the EM algorithm. What do you expect will happen if you initialize the parameters uniformly? That is, for a given document, there is a uniform distribution over topics. Similarly, for a given topic, there is uniform distribution over words. You can verify your intuitions using the code (only your reasoning is required as an answer though).

Latent Dirichlet Allocation (LDA)

This problem is for graduate students only.

In this problem, you are required to read the paper on Latent Dirichlet Allocation (LDA), which is a more sophisticated topic model than the one covered in class. Please answer the following question after you read the paper:

(Question 4.1) [2 points] Write down the probability density function of Dirichlet distribution and its support.

(Question 4.2) [3 points] Write down the generative process of LDA for a document w. What's the meaning of α and β?

(Question 4.3) [4 points] What is the likelihood of a document w under LDA and probabilistic latent semantic indexing (PLSI)? Describe the difference between the two and provide intuition of why LDA could be better than PLSI.

(Question 4.4) [6 points] Run LDA model with 10 topics on the same data from the previous problem, with 50 iterations as a maximum. Use the scikit_learn implementation `http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html` with n_components=10, max_iter=50 and other parameters set as default. You should remove all stop words from the document. Use the same stop word list as provided in the python code of Question 3.3. Interpret the "topics" that you obtain (Please provide a list of the top 10 words in each topic and a single word description of the topic). Compare the topics learned from LDA and the topic model in previous question.

Submission details

1. Create and submit report with answers to questions **as a PDF named Answers.pdf**.
2. Submit report and code to Stellar **independently**. Submit the code in a zipped file. Do not include the data directory in the zipped file, only the src directory.

References

[1] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642, 2013.

[2] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.