Timothy Truong Jr. September 23rd, 2017

6.806 Assignment 1

Bag of Words Model

1. I generated the vocabulary using the default parameters of the CountVectorizer. *This means a non-binary bag of words representation.*

2. Tuning on the dev set, we find that the best value of the hyperparameter is $c = 0.1$. This corresponds to a dev accuracy of 0.77981651376146788 and a test accuracy of 0.773750686436.

3. We interpret "most strongly weighted words" to mean the "the words that impact the prediction of the model the most". That is, if the corresponding coefficient in the logistic model for word w_i is $coef_i$, we look at the top 10 of the set $\{|coef_i|\}$. The 10 highest impact words, along with their coefficient absolute values, are as follows:

   (a) (u'bad', 1.0288018799631216)
   (b) (u'too', 0.9216975266322607)
   (c) (u'fun', 0.8491907454286777)
   (d) (u'dull', 0.8439325439862556)
   (e) (u'best', 0.8198748833524161)
   (f) (u'worst', 0.782226722213832)
   (g) (u'entertaining', 0.756426507127)
   (h) (u'solid', 0.7316634950721679)
   (i) (u'powerful', 0.6658376401756004)
   (j) (u'mess', 0.6600839805129926)

   It makes sense that these are the most heavily weighted words because the model is solving a sentiment task. Words like "bad" and "fun" are strong indicators of whether a movie is good or bad. The high weight on the word "too" is a bit strange - it has a negative corresponding coefficient, so it is likely often used to say that a movie is "too —insert negative adjective—".
   If we remove the stop words (by setting the stop_words parameter of CountVectorizer to 'english'), the best model has hyperparameter $c = 0.1$ with dev accuracy of 0.75458715596330272 and test accuracy of 0.784733662823. The highest impact features of this model are as follows:

   (a) (u'bad', 1.094598702203999)
   (b) (u'best', 0.8710982748404603)
   (c) (u'fun', 0.8622326104589116)
   (d) (u'worst', 0.8188091607311414)

(e) (u'entertaining', 0.7927253300958501)

(f) (u'solid', 0.7897760434619704)

(g) (u'dull', 0.783532406939836)

(h) (u'heart', 0.7612503673965126)

(i) (u'powerful', 0.7594981067822036)

(j) (u'mess', 0.6979642486029995)

The word "too" is gone and the word "heart" is added. The order of the words has also changed slightly. Overall, removing stop words does not appear to significantly change the model.

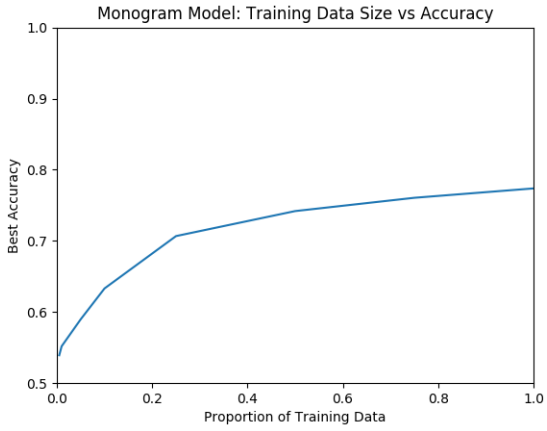
4. Switching to a bigram model seems to improve performance slightly. By setting ngram_range of the CountVectorizer to $(1, 2)$ and keeping all other defaults, we find that the best model has hyperparameter $c = 10.0$ with dev accuracy 0.80045871559633031 and test accuracy 0.802855573861.

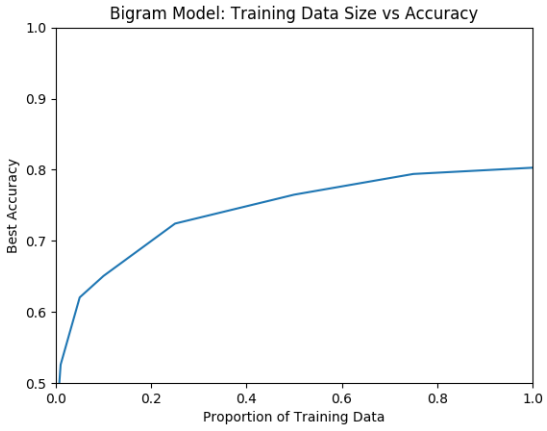
5. We attempt to train other models using the Cartesian product of the following parameter sets:

```
N_GRAMS = [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5),
           (2, 2), (2, 3), (2, 4), (2, 5),
           (3, 3), (3, 4), (3, 5),
           (4, 4), (4, 5)]
MIN_FREQUENCY = [1, 2, 5]
STOP_WORDS = [''english'', None]
CS = [1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1,
1e0, 1e1, 1e2, 1e3, 1e4, 1e5, 1e6, 1e7, 1e8, 1e9, 1e10]
```

However, we find that the best model is still the previous bigram model.

6.





EM Algorithm for Language Model Smoothing

1. We would get $\Lambda = \{0, 0, 1\}$ because we know for a fact that the log-likelihood of the trigram model will be greater than the log-likelihoods of either the unigram or bigram models. This is true because the trigram model has more explanatory power/degrees of freedom than either of the other two models. Thus setting Λ to give all the weight to the trigram model is most optimal.

2. Let T be the number of words in the development set.

   (a) $\hat{n}(\lambda_1) = \sum_{t=1}^{T} \frac{\lambda_1 p(w_t)}{\lambda_1 p(w_t)+\lambda_2 p(w_t|w_{t-1})+\lambda_3 p(w_t|w_{t_1},w_{t-2})}$,
   $\hat{n}(\lambda_2) = \sum_{t=1}^{T} \frac{\lambda_2 p(w_t|w_{t-1})}{\lambda_1 p(w_t)+\lambda_2 p(w_t|w_{t-1})+\lambda_3 p(w_t|w_{t_1},w_{t-2})}$,
   $\hat{n}(\lambda_3) = \sum_{t=1}^{T} \frac{\lambda_3 p(w_t|w_{t_1},w_{t-2})}{\lambda_1 p(w_t)+\lambda_2 p(w_t|w_{t-1})+\lambda_3 p(w_t|w_{t_1},w_{t-2})}$

   (b) $\lambda_y^{(k+1)} = \frac{\hat{n}(\lambda_y)}{\hat{n}(\lambda_1)+\hat{n}(\lambda_2)+\hat{n}(\lambda_3)}$

3. Then we will have that $\lambda_2^{(k)} = 0$, because the EM algorithm will assign zero probability to all bigrams on the first iteration, and thus all iterations. The formulas above confirm this.

EM Algorithm for Topic Models

1. $\hat{n}_t(z) = \sum_{i=1}^{N_t} p(z|w_{t_i}, t)$ and $\hat{n}(w, z) = \sum_{t'=1}^{n} \sum_{i=1}^{N_{t'}} p(z|w, t')[[w_{t_i} == w]]$.

2. $\theta_{z|t} = \frac{\hat{n}_t(z)}{N_t}$ and $\theta_{w|z} = \frac{\hat{n}(w,z)}{\sum_{w' \in V} \hat{n}(w,z)}$ where V is the vocabulary/the set of unique words from the t documents.

3. (a)

   (b) One way to determine the quality of generated topics would be to look at the most probable words and see if the topic makes sense to a human. For example, looking at the topics below, we see that while the topics may be meaningful, they are incredibly vague, particularly because some words are there simply because they are used often in the New York Times (such as "said" or "new").

   (c) i. Topic 1 - Mining?
   new (0.0087), said (0.0062), chile (0.0050), one (0.0042), times (0.0039), people (0.0039), year (0.0037), also (0.0037), york (0.0036), coal (0.0036)

   ii. Topic 2 - Indian/Pakistan-Company?
   said (0.0122), new (0.0097), york (0.0066), enron (0.0066), lay (0.0047), india (0.0045), would (0.0041), one (0.0038), pakistan (0.0035), company (0.0035)

   iii. Topic 3 - America?
   said (0.0175), one (0.0050), would (0.0050), american (0.0036), last (0.0033), year (0.0030), years (0.0030), like (0.0030), new (0.0029)

   iv. Topic 4 - Police?
   said (0.0186), enron (0.0080), lay (0.0070), new (0.0059), last (0.0049), one (0.0046), forces (0.0042), york (0.0041), year (0.0038), special (0.0037)

v. Topic 5 - Intestate News?
news (0.0097), new (0.0076), atlanta (0.0068), said (0.0065), service (0.0059), com (0.0056), journal (0.0043), washington (0.0043), moved (0.0041), constitution (0.0040)

vi. Topic 6 - Cars?
said (0.0139), enron (0.0093), company (0.0090), year (0.0071), percent (0.0058), last (0.0052), new (0.0046), one (0.0045), car (0.0042), would (0.0038)

vii. Topic 7 - Intestate News?
new (0.0102), atlanta (0.0096), news (0.0077), journal (0.0062), york (0.0059), constitution (0.0058), moved (0.0055), paint (0.0051), jan (0.0047), service (0.0044)

viii. Topic 8 - President?
said (0.0122), one (0.0053), year (0.0047), people (0.0044), bush (0.0044), president (0.0042), new (0.0041), percent (0.0036), years (0.0035), would (0.0035)

ix. Topic 9 - Energy Policy?
said (0.0178), bush (0.0080), would (0.0071), energy (0.0067), administration (0.0062), nuclear (0.0058), government (0.0055), rubin (0.0053), fuel (0.0049), could (0.0046)

x. Topic 10 - People?
said (0.0143), new (0.0078), one (0.0052), york (0.0045), two (0.0041), people (0.0035), jan (0.0034), year (0.0032), times (0.0032), would (0.0031)

The topics do not appear to be all different. Topics 5 and 7 appear to be particularly similar. In our model, the word distributions for topics are estimated independently, so there is no constraint that the topics should be different.

(d) If the parameters are initialized uniformly, all of the topics will converge to the same word distribution, as the model assumes that the topics are all independent of each other. This can be verified by setting

```
theta_t_z = np.ones((NUM_DOCS, NUM_TOPICS))
theta_z_w = np.ones((NUM_TOPICS, vocabSize))
```

in the code.