6.864 Adv. Natural Language Processing Homework 1

Anonymous Author

1 Word Embeddings

In this section, we investigated the ability of multiple word embedding methods to encode the meaning of words based on learning from a text corpus. In particular, we focused on two methods: latent semantic analysis (LSA) and the Word2Vec model. In addition to evaluating embeddings by qualitatively observing nearest neighbors for select words, we also evaluated them on a down-stream logistic regression modeling problem. For both the unsupervised learning of word embeddings and model training, we used a dataset of 3500 labelled reviews with a vocabulary size of 2006, each of which had a $0-1$ label indicating if the review was bad or good, respectively. We set aside 500 reviews for the test dataset, making the remainder available for training.

1.1 Implementation Details

1.1.1 Latent Semantic Analysis

The first word embedding method we implemented was LSA. In LSA, we encode our text corpus as a term-document matrix W_{td}, where the entry w_{td} is the number of times term t appears in document d. This matrix is normalized with term frequency-inverse document frequency (TF-IDF) normalizationWe then factorize this matrix using singular-value decomposition (SVD), getting a decomposition of the form $U\Sigma V^T$, where U, V are unitary and Σ is diagonal. We truncate U to a predetermined embedding size to get our word embeddings, which we can then use to encode words in downstream tasks.

To implement the matrix factorization, we used the `svd` method in the `numpy.linalg` library to get the matrix U, which we then truncated the rows to the appropriate embedding size. We had considered the use of `TruncatedSVD` from scikit-learn, but found it inappropriate for our use case since it returns $U\Sigma$ instead of U. In addition, to implement TF-IDF normalization, we used numpy to calculate the normalization factor for each term.

1.1.2 Word2Vec

For the Word2Vec embeddings, we used the continuous bag-of-words (CBOW) formulation of the problem. We implemented a feed-forward neural network with a single hidden-layer of the embedding dimension size in PyTorch. Like in the original Word2Vec paper, we did not incorporate a non-linear activation function. This network takes in the context words around the word we want to predict, converts them into their learned embeddings in the hidden layer, and passes their average through the second weight matrix and a softmax activation function to get the prediction of the hidden word. For numeric stability, we used the log-softmax instead of softmax. This model was then trained on the reviews text corpus to get the learned word embeddings.

1.2 Questions

1.2.1 Latent Semantic Analysis

Question 1 and 2 After performing LSA with a embedding size of 500, we see that the nearest neighbors generally relate in meaning (see Table 1), either as synonym, antonyms, or topically, to the given word. For example, “bad" had nearest neighbors such as “disgusting" and “awful," which are similar in meaning, as well as words with opposite meanings like “positive." We see that this holds true for various embedding dimensions. For embedding sizes of similar magnitudes, like 100, we observed similar qualitative performance, with the embeddings having better, more related results for cookie like “cookies," “oreos," and “craving," but worse results for “bad." As we moved further from these sizes, the nearest neighbors tended to become qualitatively worse, with fewer similar words..

Question 3 Instead of the term-document matrix, we could have used the term-term matrix

Embed Size	bad	cookie	4
10	agree, entirely, positive, forward, overly	cookies, muffins, cake, tough, excellent	1, 6, 5, protein, 7
100	taste, strange, like, myself, nasty	cookies, nana's, oreos, bars, craving	1, 6, 70, concentrated, measure
500	disgusting, awful, positive, bland, gone	nana's, moist, odd, impossible, needs	mistake, 2nd, toast, table, 70
1000	disgusting, touch, wild, entirely, timely	nana's, moist, needs, chewy, odd	economical, mistake, total, 70, certainly

Table 1: Nearest neighbors for select words at various LSA embedding sizes. Words are order from nearest to farthest. We bolded the default size given in the lab. See code for more results.

$W_{tt} = W_{td}W_{td}^T$. Since V in the SVD decomposition $U\Sigma V^T$ is unitary, we know that $V^TV = I$ and get that

$$W_{tt} = W_{td}W_{td}^T = U\Sigma V^T V \Sigma^T U^T = USU^T$$

where S is diagonal. Thus we'd expect that the SVD decomposition of W_{tt} would result in the same word embeddings as W_{td}. In practice however, while the most columns do appear the same up to a constant factor of -1, the final columns are different. This could possibly be because of numeric instability and underflows, as the values tend to be very low in these columns.

Question 4 We found that in general, the LSA embeddings improved model performance, as seen in Figure 1. This effect was observed over various training data sizes, indicating the word embeddings' consistency in improving performance.

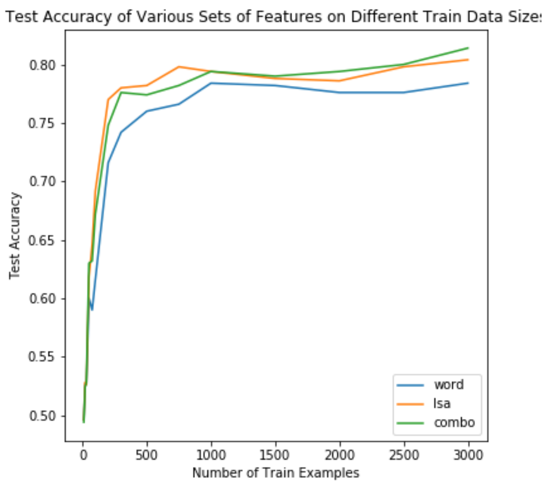

Figure 1: Test accuracy of various sets of features from LSA

Question 5 We found that embedding size is a hyperparameter in need of tuning. Test accuracy would increase with embedding size until around an embedding size of 500. At this point, test accuracy plateaued or even decreases at some train data sizes. The results are depicted in Figure 2.

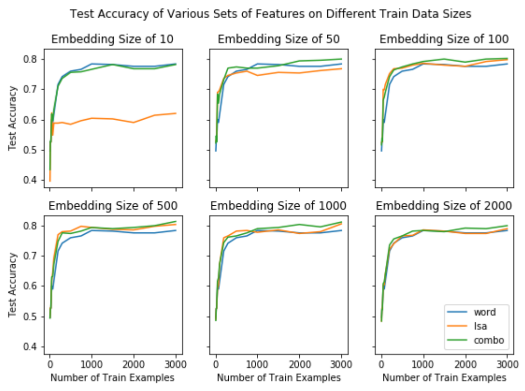

Figure 2: Test accuracy of various sets of features from LSA, over different embedding sizes

1.2.2 Word2Vec

Question 1 Qualitatively, we saw that the nearest neighbors to selected words did not seem to share a close similarity in meaning (see Table 2). Instead, adjectives like "bad" seemed to have many nouns, which makes sense as these are nouns that likely are described as bad and thus would have the word near them. Similarly, the word "cookie" had many nouns around it as well, which likely are also found close "cookie" in the corpus. This differed from our intuitive understanding of word similarity to be about meaning, which was better reflected in LSA.

Question 2 If we vary the context size in Word2Vec, we see still get similar types of nearest neighbors as those described in Question 1 (see Table 2). These results seem to worsen if we decrease the context size to the minimal 1, and improve if we increase the context size. This seems reasonable, as larger context sizes allow us to see more of the local neighbor of each word, and therefore notice related words that appear close but not right next to the word (i.e. within 5-10 words, but not 2).

Question 3 For the default hyperparameters and an embedding size of 500, as shown in Figure 3, we see that LSA outperforms Word2Vec on the test dataset in almost all train data sizes. The exception is a size of 2000; however, even in this case, LSA and Word2Vec are very close in test accuracy. We then varied the embedding size to get the results shown in Figure 4. While LSA does outperform Word2Vec in smaller embedding sizes, Word2Vec outperforms LSA at larger sizes such as 1000 and 2000. In addition, we note that like LSA, Word2Vec test accuracy increases with embedding sizes, although it plateaus later at the sizes of 1000 and 2000. This indicates that Word2Vec requires more dimensions to encode the same amount of the word similarity, but can ultimately provide better performance.

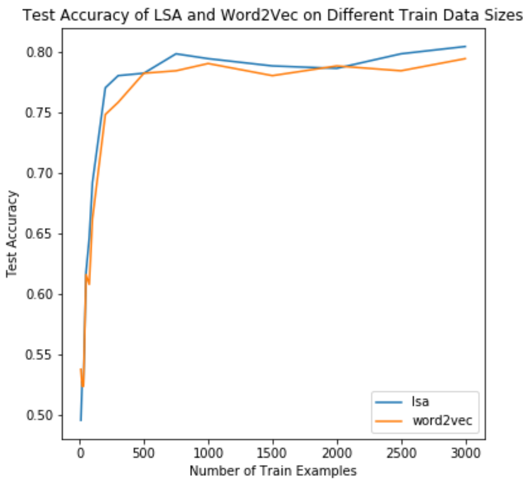

Figure 3: Test accuracy of LSA and Word2Vec over various train data sizes

Question 4 As we observed in figures for question 3 (Figures 3 and 4), while Word2Vec can achieve an overall better test accuracy than LSA at large enough embedding sizes, LSA is more space-efficient, as it can achieve the same test accuracy as Word2Vec at a lower embedding size until it begins plateauing. In addition, our qualitative analysis of nearest neighbors indicates that LSA

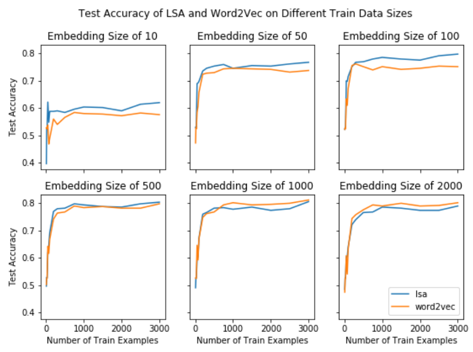

Figure 4: Test accuracy of LSA and Word2Vec over various train data sizes and embedding sizes

and Word2Vec tend to encode different types of word similarity. In particular, LSA seems more focused on meaning, presenting many synonyms and antonyms, whereas Word2Vec seems more focused on word proximity in sentences.

Question 5 One design choice in our implementation of the Word2Vec model was to average the embeddings of the context words. This can be problematic, as if two words have embeddings with opposite values of similar magnitude, the resulting vector will lose the signed information and pass along a zero to the output layer. If the magnitude of certain embedding dimensions actually encodes some meaningful information, say positivity vs. negativity, the intensity of this information can be lost when combining opposite vectors. This could be undesirable, as the unsigned intensity of this information might affect the probability of various words and would be lost in the averaging schema.

2 Hidden Markov Models

2.1 Implementation Details

To implement our hidden Markov model (HMM), we used Pytorch and the formulation introduced in lecture, training it with a expectation-maximization (EM) algorithm. We implemented the forward-backward algorithm, and used it along with the Baum-Welch EM algorithm to train our HMM. We used the same review corpus as introduced in Section 1. As we would work with discrete probability distributions with many states, we performed our calculations in log-space for numeric stability and preventing underflows in marginal probabilities. All models were trained for 10 epoches given the time and resource constraints of the experiment, unless otherwise specified.

Context	bad	cookie
1	cons, messy, inches, pound, pleasant	personally, highly, boy, support, cholesterol
2	inches, done, banana, wanting, weak	he's, caused, substitute, bit, common
5	17, strange, trust, overpowering, salty	carrot, supermarket, family, death, disappointed
10	sodas, clearly, cooking, stomach, thin	vanilla, thinking, substitute, craving, sorry

Table 2: Nearest neighbors for select words at various Word2Vec context sizes. Words are order from nearest to farthest. We bolded the default size given in the lab. See code for more results.

2.2 Questions

Question 1 We list select clusters for the HMM at various numbers of states in Table 4. When there are only two states, these clusters seem fairly meaningless, as the two states share many of the same most probably words and these words tend to be common words, like “a," “the," and “is," or punctuation marks. When we increases the number of states to 10 or even 50, we see a similar problem with the most common English words and punctuation marks appearing in multiple states. However, we do begin to see some longer words, such as “have" and “these" that previously did not appear. However, if we generate sentences, we see that for 10 or 50 states, less common words are selected and more coherent text samples are returned, although the results are not close to human writing.

Unfortunately, due to the performance of our HMM implementation, we could not run the experiment for 100 states.

S	**Example State Clusters**
2	. a it and i , is br to <unk>
	<unk> the , i of . to and in that
10	. i <unk> it , they that and not you
	i a , have and the is you . !
50	<unk> . , and to it of for have not
	, . the to my is this a are these

Table 3: Select state clusters for trained with $S = 2, 10$, and 50 states, presenting the ten most probable words of each cluster. Words are order from most probable to least. Note that punctuation count as words for the purposes of this experiment. See code for full results.

Question 2 We found that the logistic regression model fails to converge, even with 50 states and all 3000 available training points. We observe a spike in test accuracy around 500 training examples, after which accuracy slowly increases. For sizable training datasets, test accuracy lies between 0.6 and 0.65, much lower than test accuracy for either methods in Section 1, which had peak accuracies around 0.81.

This may in part be due to too small of an HMM, as well as the fact that an EM algorithm is only guaranteed to converge to a point of zero gradient, which might not be locally optimal much less globally so. Altogether, these concerns may provide an explanation to why our HMM models do not provide coherent sentences. In terms of the classification task, the word embeddings may be more appropriate for this task, as they encode some sense of word similarity that could include factors that strongly correlate to sentiment, whereas the distribution of next word from the HMM seems less tailored for such a task.

S	**Generated Text Samples**
2	'are not <unk> are are not not are <unk> these'
10	'during <unk> i up all , br i in single'
	'snack salt changed saw not , ! from have spent'
50	'allergic a it br over you purchase only about for'
	'just . this still if i are eaten the i'

Table 4: Select text samples of length 10 generated by HMMs trained with $S = 2, 10$, and 50 states. See code for full results.

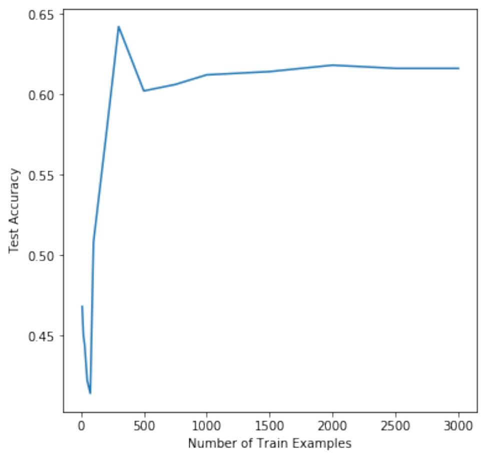

Figure 5: Test accuracy of the HMM with 50 states on various train data sizes