Homework 2
Advanced Natural Language Processing

Felipe del Canto

October, 2021

1. Part 1: Matrix Factorization

Starting with the scaffolding code provided in the notebook, in order to implement the function `learn_reps_lsa`, which learns the LSA representations given a corpus. To do this, I followed three steps. First, I defined the `idf_matrix` function, which given a term-document (TD) matrix returns the inverse-document-frequency (IDF) matrix given by

$$\texttt{idf}(w) = \log \frac{\text{\# documents}}{1 + \text{\# documents that contain } w} \tag{1}$$

where w is a word in the vocabulary V. The denominator is adjusted in order to account for words that do not appear in any document during the training process of the review classifier. Second, I defined the `tf_matrix` function, which given a TD matrix returns the term-frequency (TF) matrix according to a certain weight (given by `tf_weight`). All weights and the corresponding entries of the resulting TF matrix are summarized in Table 1. The third step is to finally implement the `learn_reps_lsa` function. For this, given a TD matrix, a representation size, and optional arguments `tf_weight`, `tf_idf`, and `tf_weight`, the function does the following:

1) Computes the TF matrix according to `tf_weight`. If it is not provided, the TF matrix is equal to the TD matrix.

2) If `tf_idf` is `True`, computes the IDF matrix according to `tf_weight`.

3) If `tf_idf` is `False`, sets the `LSA_matrix` (the matrix to be decomposed using SVD) equal to the TF matrix. Otherwise, `LSA_matrix` is set equal to the TF-IDF matrix given by

$$\texttt{tf-idf}(w,d) = \texttt{tf}(w,d) \cdot \texttt{idf}(w)$$

4) The `LSA_matrix` is then decomposed using the SVD algorithm provided by the library `scikit-learn` and the result is returned.

Table 1: Summarize of `tf_weight` options for function `tf_matrix` when applied to matrix TD.

`tf_weight`	$\texttt{tf}(w,d)$
‘`raw`’	$TD(w,d)$
‘`term frequency`’	$\frac{TD(w,d)}{\sum_{w,d} TD(w,d)}$
‘`log normalization`’	$\log(1 + TD(w,d))$
‘`double-K`’	$K + (1-K)\frac{TD(w,d)}{\max_{w'} TD(w',d)}$

After executing the `learn_reps_lsa` on the reviews corpus, I modified the `lsa_featurizer` function to accepts these representations as an argument. Although these can be accessed as a global variable, I do this for better code readability. The latter function takes a TD matrix for it to be featurized, and the learned representations. In its initial form, this function returns the features which are obtained by summing the learned representations for each word in each document. That is,

$$\text{feat}(d) = \sum_{w \in d} \texttt{lsa\_rep}(w)$$

where $\texttt{lsa\_rep}(w)$ is the learned LSA representation of word w. However, in order to make more experiments, the function also allows to weight this sum by the IDF matrix. That is, if the `idf_weighting` argument is `True`, then:

$$\text{feat}(d) = \sum_{w \in d} \texttt{idf}(w) \cdot \texttt{lsa\_rep}(w)$$

where `idf` is computed as in (1). This tweak allows the representation of each document to contain more information on words that are less frequent, potentially improving the features obtained.

Next, in order to carry out other experiments more easily, the functions `train_model` and `trainint_experiment` require a model to be passed as argument. Moreover, these functions also accept arguments and keyword arguments for each featurizer to be passed as optional arguments, in order to improve readability.

Now, for part (a), consider the singular value decomposition (SVD) of W_{td},

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

where U and V are orthonormal matrices. Then

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

but since $VV^T = I$, then

$$W_{tt} = U\Sigma^2 U^T \tag{2}$$

where

$$\Sigma^2 = \Sigma\Sigma^T = \Sigma^T\Sigma$$

is the diagonal matrix whose elements are the squares of the elements of Σ. This implies that (2) corresponds to the SVD of W_{tt} and thus the left singular vectors of W_{tt} and W_{td} coincide.

To check if this happens, I first run SVD on both matrices. Then, divide the embedded representation by the singular values, since by definition the `TruncatedSVD` algorithm of library `scikit-learn` returns the $U_k\Sigma_k$ representation. Then, I normalize the output and obtain the absolute value of each entry. Finally, to check if these representatios coincide, I compute the cosine similarity between each column. In Table 2 is resumed the behavior of the share of vectors that present cosine similarity values above different thresholds. As can be seen from the table, the cosine similarity is high for low embedding sizes and for an extremely hight value of 2000. However, for embedding sizes between 100 and 1000, the share diminishes. This pattern is most likely due to the algorithm used for decomposing the matrices and not produced by an implementation issue. This, because for all embedding sizes, the singular values of the TD matrix coincided with the square root of the singular values of the TT matrix. Consequently, one potential problem could be numerical instability, specially for the TT matrix, whose singular values are squared with respect to the TD matrix and thus the vectors have lower numerical precision.

Now, for part (b), in Table 3 are presented the 5 most similar words for each of the main words considered, under different embedding sizes. All these experiments were ran using LSA on the TF-IDF matrix, with a the `'term frequency'` TF weight. Other experiments, with other TF weights and with no TF-IDF matrix were performed but the results were similar, and thus they are not shown. From the table is possible to observe that the group of similar words for each main word is similar across embedding sizes. However, the small differences may not have the same effect for every word. For example, "switched" was similar to "dog" when the representation size was low, and disappeared for bigger embedding sizes, being replaced by "dogs". This could be a good as now the cluster represents similar words. On the other hand, the effect of the embedding size on the words related to "3" may be different. In this case, for an embedding size of 100, most of the similar tokens were numbers, which might be useful. However, after increasing the size of the low-rank approximation, the similar tokens to "3" are mostly punctuation marks and the "<unk>" token, which may not carry a lot on information for other downstream tasks.

Finally, for part (c), in Figure 1a is presented the

Table 2: Cosine similarity (in absolute value) between column vectors of low rank approximations of TD and TT matrices.

Embedding Size	Share of vectors with similarity greater or equal than		
	0.90	0.95	0.99
10	1	1	1
50	0.92	0.86	0.76
100	0.81	0.80	0.72
500	0.80	0.77	0.72
1000	0.85	0.83	0.80
2000	1	1	1

Table 3: Similar words for given terms, under different representation sizes. All experiments use `'term frequency'` as the TF weighting and compute the representations using the TF-IDF matrix.

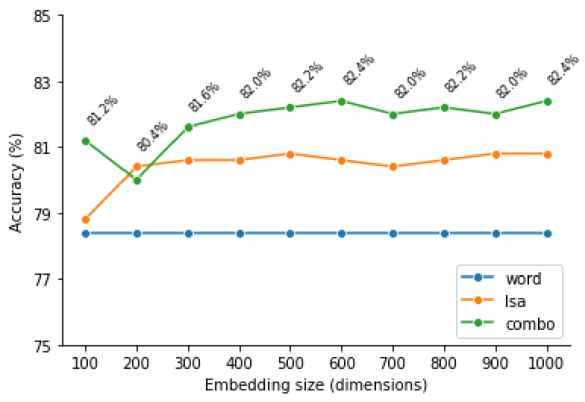
Embedding Size	Similar words to			
	the	dog	3	good
100	.	pet	8	but
	of	foods	2	.
	to	switched	1	a
	and	pets	per	is
	<unk>	food	to	for
500	.	food	8	.
	and	pets	.	but
	<unk>	pet	the	a
	of	foods	to	and
	to	switched	of	the
1000	.	food	.	.
	<unk>	pets	8	a
	and	pet	the	but
	of	foods	to	and
	to	dogs	<unk>	the

accuracy of the classifier for each featurizer, when varying the embedding size. In all experiments, the LSA representation is computed using the TF-IDF matrix with "term frequency" weighting, and the LSA featurizer is using the IDF weighting to sum the representation vectors. As can be seen from the image, the LSA features by themselves are in general more powerful than the word features. In fact, their accuracy is almost 2 percentile points above. However, the best performance in general is given by the "combo" featurizer, which in turn improves the accuracy by approximately 2 percentile points with respect to LSA. Another important point to note from the picture is that the performance of the classifier is somewhat stable across different embedding sizes. This is positive, since it shows that not many features are needed in order to get good performance in downstream tasks, which may improve computational time overall. For the second question, fixing the embedding size in 500, I repeat the experiment for different training sample sizes. The results are presented in Figure 1b. In general, the accuracy of the classifier shows an upward trend, which is a usual feature of supervised learning algorithms. Moreover, this happens in general for the three featurizers, and their ordering is repeated with respect to Figure 1a.

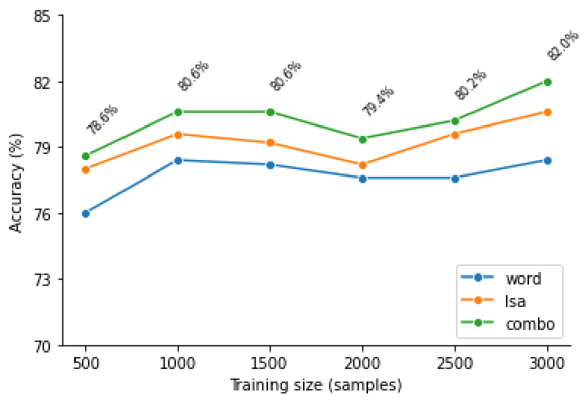
As an exploration exercise, I investigate if changing the classifier type or the TF-IDF weighting scheme change the previous results. In order to make these comparable with the ones obtained above, the number of training samples will be fixed in 3000 and the embedding size in 500. Additionally, the LSA representations will be computed using the TF-IDF and the LSA featurizer will use IDF weighting. The accuracies obtained by trying different classifiers is presented in Figure 2a. As seen in the picture, the accuracy of the Logistic Regression classifier is already one of the best across all methods considered. However, the SVM classifier is capable to slightly improve accuracy by 0.6 percentile points. It is interesting to note that in general, the ordering of the featurizer in terms of accuracy is not respected for the other models. In particular, QDA, 3-NN and 5-NN show a very good performance of the LSA features alone. In the case of the nearest neighbor classifiers, this could be explained if LSA features are indeed capturing some sense of similarity in terms of the labeling of each review. If that is the case, then these similarities are not perfect, as the performance is really poor compared to the Logit benchmark.

For the second part of this exercise, to test the effect of different weighting schemes, I fix the Logistic Regression classifier and consider different weighting schemes for the TF-IDF matrix. The results are presented in Figure 2b. Observe that the

Figure 1: Accuracy of each featurizer for different embedding (left) and training (right) sizes. The LSA representations are using the TF-IDF matrix with "term frequency" weighting and the LSA featurizer is using the IDF weighting to sum the representation vectors.



(a) Training size = 3000



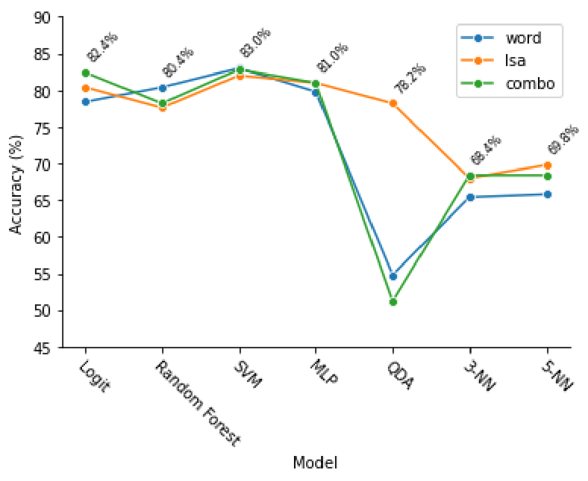
(b) Embedding size = 500

accuracy for the word featurizer does not change, since the only featurizers affected by the weighting scheme are "LSA" and "combo". The figure shows a relatively stable behavior of the accuracy for the combo featurizer, although its performance falls for the double-K features. However, the LSA featurizer is not reliable when using the double-K weighting scheme. This goes in line with the idea that double-K puts $(1-K)$ weight on the TD matrix. Hence, the accuracy falls when K grows because the information on the TD matrix fades. Overall, `log-normalization` is the scheme that achieves the better performance for LSA, and overall. In particular for the combo featurizer, the accuracy improves in 1.2 percentile points with respect to the `raw` weighting benchmark.

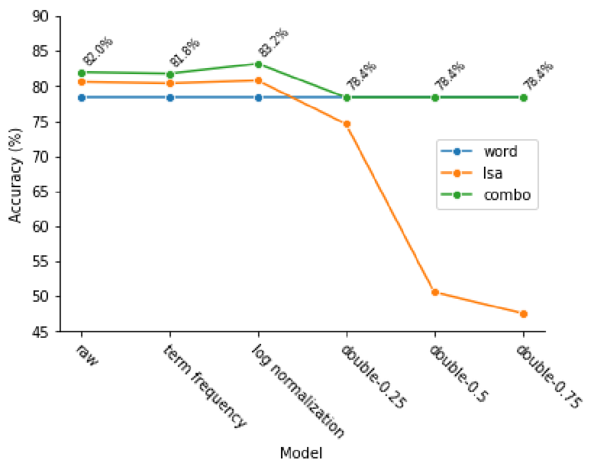
Part 2

In order to implement the CBOW formulation of Word2Vec in the function `learn_reps_word2vec`, the first step is to define a class inheriting from the `nn.Module` class. To initialize this new class, called `Word2VecModel`, the input and output matrices must be created. These will comprise the final representations for each word and have sizes $(|V|+1) \times k$, where $|V|$ is the vocabulary size and k is the embedding dimension. Observe that the additional row allows the model to include a padding token, which also has a representation.

Figure 2: Accuracy of each featurizer for different classifiers (left) and TF-IDF weights (right). The LSA representations are using the TF-IDF matrix with "term frequency" weighting and the LSA featurizer is using the IDF weighting to sum the representation vectors.



(a) TF weighting = `'term frequency'`



(b) Classifier: Logistic Regression

In the rest of the report, it is assumed these matrices only have $|V|$ rows for readability purposes. In code, these matrices are stored as weights of an `Embedding` Pytorch object.

After initializing the object, the `forward` function takes the context for a word and embed this using the input embedding matrix. The context is an array of size $2w$, where w is the window size and represents the $2w$ words that surround a certain word. After embeddding the context, this representation is averaged to obtain a single array (of length `embedding_size`) which is then transformed into a probability by multiplying it for the output embedding matrix and taking the (log) softmax of the result. Observe that for this last step, the last row associated with the padding token must be eliminated.

Now, to complete the implementation of the Word2Vec (CBOW) embeddings, first I obtain the context for each word in the training data. Next, I define the loss for this task, which is the Negative Log-likelihood. Finally, for each epoch and batch, I feed forward the context to the `Word2VecModel` object and obtain the predictions, to compute the loss and complete the backward pass. After training, the function can receive an optional argument to inicate which representation to return. In a Word2Vec model, both the input and output embedding matrices could be used as the word representations. Consequently, the function `learn_reps_word2vec` can accept three possible values for the `embedding_type` argument:

- `input`: only input embedding matrix.
- `output`: only output embedding matrix.
- `combo`: simple average of both matrices.

Now, for part (a), in Table 4 are presented the similar words to the same terms used for LSA, but under the Word2Vec representation. In general the representations observed by the nearest neighbors are better than those given by the LSA representations. For example, words "3" and "good" are no longer associated with punctuation marks, which is intuitive given that they do not belong to the same family. However, it seems that this improvement has its limits. For example, word "3" was previously associated with numbers (see the first row of Table 3), but with Word2Vec it is now associated with words that do not share an apparent relationship with "3", like "boxes" or "carton". Something similar happened with the word "dog", which was previously associated with "food" or its plural "dogs", and even the word "pet", but now is associated with non-related words like "blend", "stomach" and "again". There are two possible explanations for this. One is that the representation is not capturing similarities between words close to each other. The second is that the similarities that are being captured do not correspond to those we would intuitively expect. A closer look at the clustering technique shows that the pattern observed for words "dog" and "3" is not exclusive. Words like "picky" and "milk" appear together in the same cluster, as well as "disposable" and "shared". This may hint that the most plausible explanation is the second: Word2Vec is capturing some similarities, but not the ones we intuitively expect. Whether these similarities are useful for downstream tasks is something I will test below.

For part (b), the classification task when using the Word2Vec featurizer alone achieves an accuracy of 80.4%. In comparison, the "combo" featurizer (word + LSA features) under the same classifier and embedding size achieved an accuracy of 82.2% (see Figure 1a). Moreover, LSA features alone only achieved an accuracy of 80.2%. It seems that for this classification task, a combination of word and semantic features is needed to achieve a higher accuracy. Moreover, although Word2Vec

Table 4: Similar words for given terms, under Word2Vec CBOW representation.

Similar words to			
the	dog	3	good
case	blend	boxes	recommended
.	cat	carton	tasting
br	healthier	1	need
2	stomach	6	looked
which	again	20	at

CBOW embeddings appear to incorporate more information about words than LSA, their performance is similar in this task. Whether Word2Vec features combined with word features achieve a better accuracy than the original combo featurizer is something I will investigate further below.

For part (c), in Figure 3a is shown the accuracy of the classification task under different Word2Vec window sizes. It is interesting to note that Word2Vec features appear to improve with context size, when measured through the accuracy of the classifier. In the figure, the `word2vec` featurizer curve has an upward trend. However, using these features combined with word features does not have this upward trend. Instead, the behavior is mostly flat. The best performance, with 85.2% of accuracy is achieve for a window size of 10, that is, for a context size of 20. This is 2.8 percentile points higher than the LSA + word features of previous experiments.

Finally, as I did with the LSA representations, I turn to analyzing how the Word2Vec representations change and modify the accuracy of the classification task. First, I investigate if changing the embedding type has an impact on the accuracy score. This is, if choosing the output embedding matrix, the input embedding matrix or the average of the two modifies the previous results. This can be seen in Figure 3b. The figure shows that all three embedding types behave similarly. This means that output and input embedding matrices are capturing the same information for this task. Second, I repeat the experiment of testing the accuracy of different classifiers. Since there was a non-stable behavior of the LSA representations on this task, it might be expected that these learned representations also have a non-stable pattern. The results are presented in Figure 3c. As can be seen in the picture, the Word2Vec features show a similar pattern, but the accuracies can be quite different in some of the classifiers. In this case, the highest score for the Logit and SVM classifiers increases for the `word2vec (combo)` featurizer. However, this increment is small. In particular for SVM, the accuracy is only 0.4 percentile points higher. What is interesting is that Word2Vec features alone perform much better than LSA features for the nearest neighbor classifiers, improving from 68.4% for the 3-NN and 69.8% for the 5-NN, to 74% and 74.6%, respectively. If the theory of the semantic clustering is correct, then this means than Word2Vec representations are better than LSA features at capturing the similarities of words for this task in particular. Nevertheless, and as happened with the previous experiment, the performance of nearest neighbors is poor. This is expected, since sentiment analysis is not necessarily correlated with similarity of words.

Part 4

For part (a), we have that

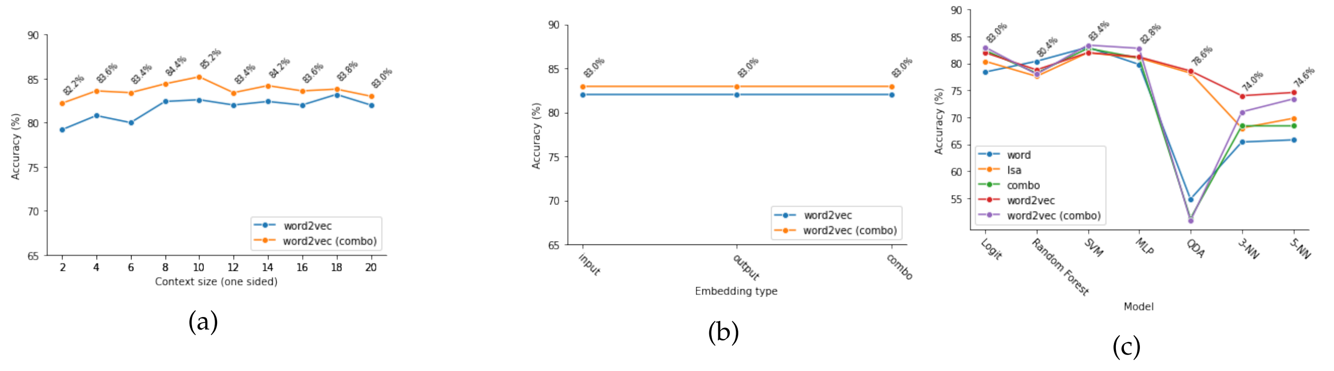
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Thus,

$$\frac{\partial \log \sigma(x)}{\partial x} = \frac{\partial - \log(1+e^{-x})}{\partial x} = \frac{e^{-x}}{1+e^{-x}} \tag{3}$$

For part (b) note that using the expression for

Figure 3: Word2Vec embedding results for the downstream classification task. The `word2vec (combo)` featurizer combines word features with Word2Vec embeddings. The left panel shows the accuracy of the Logit classifier under different context sizes. The central panel shows the accuracy when changing the embedding type. Finally, the right panel shows the accuracy under different classifiers (including the accuracy of the previous featurizers).



$\ell(w,c)$ we have that, using (3),

$$\frac{\partial \ell(w,c)}{\partial \vec{w}\cdot\vec{c}} = \#(w,c)\,\frac{e^{-\vec{w}\cdot\vec{c}}}{1+e^{-\vec{w}\cdot\vec{c}}} - k\cdot\#(w)\cdot\frac{\#(c)}{|D|}\cdot\frac{e^{-\vec{w}\cdot\vec{c}}}{1+e^{-\vec{w}\cdot\vec{c}}}. \tag{4}$$

Letting $\lambda = k\cdot\#(w)\cdot\frac{\#(c)}{|D|}$, we have that (4) is equal to 0 if and only if

$$\frac{\#(w,c)\,e^{-\vec{w}\cdot\vec{c}}}{1+e^{-\vec{w}\cdot\vec{c}}} = \frac{\lambda e^{\vec{w}\cdot\vec{c}}}{1+e^{\vec{w}\cdot\vec{c}}}.$$

Cross multiplying and rearranging we obtain

$$\#(w,c)e^{-\vec{w}\cdot\vec{c}} - \lambda e^{\vec{w}\cdot\vec{c}} + \#(w,c) - \lambda = 0$$

Multiplying this expression by $-\lambda^{-1}e^{\vec{w}\cdot\vec{c}}$ and rearranging we finally obtain

$$e^{2\vec{w}\cdot\vec{c}} - \left(\frac{\#(w,c)}{\lambda} - 1\right)e^{\vec{w}\cdot\vec{c}} - \frac{\#(w,c)}{\lambda} = 0 \tag{5}$$

which is a second-order equation on $e^{\vec{w}\cdot\vec{c}}$ with solutions:

$$\begin{aligned} e^{\vec{w}\cdot\vec{c}} &= \frac{1}{2}\left(\frac{\#(w,c)}{\lambda} - 1\right) \\ &\quad \pm \frac{1}{2}\sqrt{\left(\frac{\#(w,c)}{\lambda} - 1\right)^2 + \frac{4\,\#(w,c)}{\lambda}} \\ &= \frac{1}{2}\left(\frac{\#(w,c)}{\lambda} - 1\right) \\ &\quad \pm \frac{1}{2}\sqrt{\left(\frac{\#(w,c)}{\lambda} + 1\right)^2} \end{aligned}$$

And thus, the solutions to the second-order equation are

$$e^{\vec{w}\cdot\vec{c}} = \begin{cases} \dfrac{\#(w,c)}{\lambda} \\ -1 \end{cases}$$

Since e^x is positive, then the second solution is not feasible and, thus, the unique solution in this context is

$$e^{\vec{w}\cdot\vec{c}} = \frac{\#(w,c)}{\lambda} = \frac{\#(w,c)}{\#(w)\cdot\#(c)}\cdot|D|\cdot k$$

Consequently, for part (c) we have that the critical point satisfies

$$\vec{w}\cdot\vec{c} = PMI(w,c) + \log(k)$$

This is, the learned embedding vectors $\vec{w}$ and $\vec{c}$ are such that their dot product is equal to the corresponding entry of the PMI matrix, plus a bias given by the log of negative samples selected (which is positive since $k \geq 2$).