Part 1. Text Classification with Bag-of-Words

(a) After fitting two logistic regression models on the vectorized BoW matrix and transformed TF-IDF matrix respectively, the two classifiers were evaluated using the validation and test datasets. Based on the results shown below, it appears that the TF-IDF transformation does not produce any noticeable effect on the classification task.

```
Logistic regression with bag of word features:
Training accuracy: 0.9853
Validation accuracy: 0.8260
Test accuracy: 0.8180

Logistic regression with tf-idf features:
Training accuracy: 1.0000
Validation accuracy: 0.8000
Test accuracy: 0.8180
```

(b) Extracting the five most positive/negative coefficients from `model.coef_` and obtaining the corresponding words from `vectorizer.tokenizer.token_to_word`, we find:

```
BoW                                    TF-IDF
-----                                  -----
delicious      1.72716                 great          1.55104
great          1.67073                 best           1.03295
best           1.40872                 tassimo        0.99075
thanks         1.38444                 delicious      0.83433
long           1.32134                 long           0.76265
...                                    ...
yuck          -1.43332                 gf            -0.83958
gf            -1.49177                 disappointing -0.84949
unfortunately -1.49587                 disappointed  -0.88703
disappointed  -1.57602                 unfortunately -0.88746
stick         -1.61904                 not           -1.41170
```

Part 2. Learning Word Representations with Matrix Factorization and Word2Vec

(a) Using the properties of SVD, we can show that the left singular vectors of W_{td} and W_{tt} are the same as follows:

$$\begin{aligned} W_{tt} &= W_{td}W_{td}^{\intercal} \\ &= (U\Sigma V^{\intercal})(V\Sigma^{\intercal}U^{\intercal}) \\ &= U(\Sigma\Sigma^{\intercal})U^{\intercal} \qquad \text{(since } V \text{ is orthogonal, so } V^{\intercal}V = I\text{)} \\ &= U\Sigma_{tt}U^{\intercal} \qquad \text{(where } \Sigma_{tt} = \Sigma\Sigma^{\intercal}\text{)} \end{aligned}$$

The last line is the decomposition of W_{tt}. Thus, the left singular vectors of W_{td} (which are the columns of U) are also the left singular values of W_{tt}. This result also holds experimentally, as our implementations of `learn_reps_lsa` and `learn_reps_lsa_wtt` both return the same matrix of values (see Experiments: Part 2a of the code).

(b) Implementation: we apply LSA to the approaches from Part 1 and train logistic regression-based word embedding model on it, as well as train another model using Word2Vec's continuous bag of words objective instead of matrix factorization. The Word2Vec model consists of a "bag" of embedding layers feeding into a linear layer with softmax as the activation function, and the resulting predictions are transformed into log-probabilities.

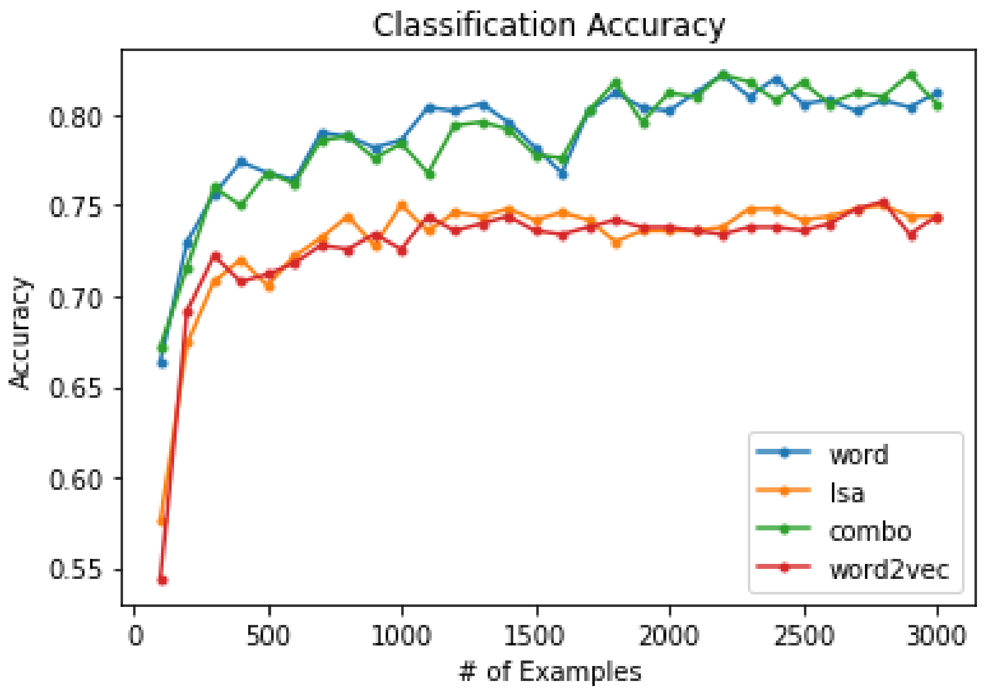
The nearest neighbors of *the*, *dog*, *3*, and *good* in representation space are shown below for LSA and Word2Vec word embeddings.

```
LSA:
the 36
  of 0.723
  . 0.775
  in 0.824
  on 0.926
  to 0.930
dog 925
  foods 0.574
  pet 0.636
  nutritious 0.652
  pets 0.687
  switched 0.719
3 289
  per 0.821
  8 0.857
  gallon 0.876
  1 0.907
  2 0.924
good 47
  quick 0.594
  everyone 0.673
  decide 0.674
  than 0.713
  better 0.726
```

```
Word2Vec:
the 36
  my 0.568
  their 0.623
  a 0.657
  an 0.679
  our 0.767
dog 925
  cat 0.609
  baby 0.806
  dogs 0.878
  junk 0.930
  salsa 0.997
3 289
  5 0.343
  2 0.470
  1 0.564
  4 0.643
  6 0.759
good 47
  great 0.425
  bad 0.686
  strong 0.941
  delicious 0.974
```

These results show that the nearest neighbors of any given word are semantically connected to each other. In the case of *the*, the nearest neighbors are other common articles in English. Similarly, the nearest neighbors of *3* are other numbers or measurements, and the nearest neighbors of *good* are synonyms (as well as antonyms). For *dog*, the nearest neighbors include other animals, as well as terms related to pet ownership, including feeding. As the size of the representations increase beyond 50, the similarity of nearest neighbors tends to become more sparse, and words get grouped with other words that are not semantically related (e.g. when `rep_size` = 500, the nearest neighbors of *3* become ”facts”, ”supermarket”, and ”closer”). On the other hand, when the size of the representations is too small, not enough information is captured by the Latent Semantic Analysis, so the results are not meaningful.

(c) After training a logistic regression model on each of the four featurizers (”word”, ”lsa”, ”combo”, and ”word2vec”) for varying numbers of labeled examples ranging from 100 to 3,000, it seems that the learned representations for LSA and Word2Vec do not help with the review classification problem, judging by the accuracy rates as shown below (”word” and ”combo” are universally better performing than ”lsa” and ”word2vec”, which are more or less equal). This is in part due to the inability of learned representations to differentiate between synonyms and antonyms; for instance, Word2Vec puts ”bad” as a near-neighbor of ”good”, leading it to potentially bad reviews as good and vice versa. However, we note that the combination of the word and LSA featurizers leads to a decent-performing model that can outperform the word featurizer alone in some cases, which suggests that the latent information uncovered by LSA does play a certain role in classification. Lastly, increasing the number of examples seems to boost the effect of the word embeddings, increasing the accuracy of the model up to a degree (based on the chart, between 500 to 1,000 examples are needed to reach maximum accuracy).



Part 3. Analyzing Word2Vec Skip-Gram with Negative Sampling

(a)

$$\begin{aligned}
\frac{\partial \log \sigma(x)}{\partial x} &= \frac{\partial}{\partial x} \log\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{\partial}{\partial x} \log\left(\frac{e^x}{e^x+1}\right) \\
&= \frac{\partial}{\partial x} \log(e^x) - \log(e^x+1) \\
&= \frac{\partial x}{\partial x} - \frac{\partial}{\partial x} \log(e^x+1) \\
&= 1 - \frac{e^x}{e^x+1} \\
&= \frac{1}{e^x+1}
\end{aligned}$$

(b) Setting $\overrightarrow{w} \cdot \overrightarrow{c} = x$, we get

$$\ell(w,c) = \#(w,c) \cdot \log \sigma(x) + k \cdot \frac{\#(w)\#(c)}{|D|} \cdot \log \sigma(-x)$$

Taking the derivative with respect to x gives

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

Setting the derivative to 0 and substituting $\lambda = k \cdot \frac{\#(w)\#(c)}{|D|}$, we get

$$\begin{aligned}
0 &= \frac{\#(w,c)}{e^x+1} - \frac{\lambda}{e^{-x}+1} \\
\frac{\lambda}{e^{-x}+1} &= \frac{\#(w,c)}{e^x+1} \\
\lambda e^x + \lambda &= \#(w,c)e^{-x} + \#(w,c) \\
\lambda e^{2x} + \lambda e^x &= \#(w,c) + \#(w,c)e^x \\
\lambda e^{2x} + \lambda e^x - \#(w,c)e^x - \#(w,c) &= 0 \\
e^{2x} - \left(\frac{\#(w,c)}{\lambda} - 1\right) e^x - \frac{\#(w,c)}{\lambda} &= 0 \\
\left(e^x - \frac{\#(w,c)}{\lambda}\right)(e^x+1) &= 0
\end{aligned}$$

The second root is imaginary, so we obtain

$$e^x - \frac{\#(w,c)}{\lambda} = 0 \implies x = \log\left(\frac{\#(w,c)}{\lambda}\right) = \log\left(\frac{\#(w,c)}{k \cdot \#(w)\#(c)} \cdot |D|\right)$$

Thus, the critical point of $\overrightarrow{w} \cdot \overrightarrow{c}$ is $\log\left(\frac{\#(w,c)}{k \cdot \#(w)\#(c)} \cdot |D|\right)$.

(c) When $k = 1$, the critical point of $\overrightarrow{w} \cdot \overrightarrow{c}$ is

$$\log\left(\frac{\#(w,c)}{\#(w)\#(c)} \cdot |D|\right)$$

which is exactly the value of the PMI matrix for (w, c). Otherwise, the number of negative samples k that we choose to draw for each positive sample corresponds to a $\log k$ shift in the PMI value, which makes sense because we are generating more negative samples in total. The embedding vectors $\overrightarrow{w}$ and $\overrightarrow{c}$ learned by the SGNS model are the two vectors whose product is most likely to occur; that is, the most likely word-context pair in the PMI matrix.