Project 1

Bilal Baig (215734320)
bilalb@yorku.ca

April 25, 2021

Note: This is the project for 4404 students only. You need to work individually for this project. You must use this latex template to write up your report. Remember to fill in your information (name, student number, email) at above. Submit your codes/scripts (*.zip) and a project report (*.pdf) (maximum 6 pages) from eClass before the deadline. No late submission will be accepted. No handwriting is accepted. Direct your queries to Hui Jiang (hj@eecs.yorku.ca).

Latent Semantic Analysis for Natural Language Processing

In this project, you will use a text corpus, called the English Wikipedia Dump, to construct document-word matrices and then use the latent semantic analysis (LSA) technique to factorize the matrices to derive word representations, a.k.a. *word embeddings* or *word vectors*. You will first use the derived word vectors to investigate semantic similarity between different words based on the Pearson's correlation coefficient obtained by comparing cosine distance between word vectors and human assigned similarity scores in a data set called *WordSim353* (http://www.cse.yorku.ca/~hj/wordsim353_human_scores.txt). Furthermore, the derived word vectors will be visualized in a 2-dimensional space using the t-SNE method to inspect the semantic relationship among English words. In this project, you will implement several machine learning methods to factorize large sparse matrices to study how to produce meaningful word representations for natural language processing.

1. Use a small data set, called *enwiki8* (downloading from http://www.cse.yorku.ca/~hj/enwiki8.txt.zip) to construct a document-word frequency matrix in Figure 7.7. In this project, you treat each paragraph in a line as a document. You construct the matrix in a sparse format for the top 10,000 most frequent words in *enwiki8* and all words in *WordSim353*.

2. You first use a standard SVD procedure from a linear algebra library to factorize the sparse document-word matrix, and truncate it to $k = 20, 50, 100$. Examine the run-in time and memory consumption for SVD.

3. You can choose any algorithm from the following two choices to implement matrix factorization from scratch[1]:

   (a) The alternating Algorithm 7.6; or
   (b) The stochastic gradient descent (SGD) method in Exercise 7.5.

   Use your implementation to factorize the document-word matrix for $k = 20, 50, 100$. Examine the run-in time and memory consumption.

4. Investigate the quality of the above derived word vectors based on the correlation with some human assigned similarity scores. For each pair of words in *WordSim353*, compute the cosine distance between their word vectors and then compute the Pearson's correlation coefficient between these cosine distances and human scores. Tuning your learning hyperparameters towards higher correlation.

[1] You are only allowed to use linear algebra libraries, such as matrix multiplication, matrix inversion, eigenvalues and eigenvectors.

5. Visualize the above word representations for the top 300 most frequent words in *enwiki8* using the t-SNE method[2] by projecting each set into a 2-dimensional space. Investigate how these 300 word representations are distributed and inspect whether the semantically relevant words are located closer in the space. Explain why.

6. Refer to [1] to re-construct the document-word matrix based on the positive pointwise mutual information (PPMI). Repeat the above steps 2-5 to see how much the performance is improved.

What to submit?

You need to submit all of your codes written for this project. Please provide a clear instruction on how to repeat your experiments in a separate readme file. You need to submit a project report (in pdf, maximum 6 pages) to summarize what you have done in terms of algorithm development and experimental fine-tuning, also report the best settings for each case and discuss your findings from this project.

References

[1] Peter D. Turney and Patrick Pantel. 'From Frequency to Meaning: Vector Space Models of Semantics.' In: *J. Artif. Int. Res.* 37.1 (Jan. 2010), pp. 141–188. (`https://arxiv.org/abs/1003.1141`)

[2] You can use any existing t-SNE library.

Your Report (maximum 6 pages):

Algorithm Development

1. **SVD vs SGD Matrix Factorization**
   To compute the SVD I used the scipy library as it allowed for choosing how many features are needed. The time it took for K=20,50 and 100 along with the Pearson Correlation Coefficient using the SVD and Human scores are in Figure 1. As seen it takes roughly 0.15*k seconds for the algorithm to compute the matrices. Memory usage can get very high, using a sparse matrix it was possible to compute the entire matrices however it still used a lot of memory compared to Stochastic Gradient Descent.
   On the other hand, the stats using Stochastic Gradient Descent to compute the matrix factorization are in Figure 2. As seen it runs much quicker than SVD at about roughly Memory usage was much lower than what SVD requires, while it was especially when we are using larger matrices.

```
Part 2 started...
When K = 20:
U shape:  (10076, 20)
S shape:  (20,)
V shape:  (20, 489860)
Runtime: 3.645872116088867
PCC:  0.1459420697412974
When K = 50:
U shape:  (10076, 50)
S shape:  (50,)
V shape:  (50, 489860)
Runtime: 8.470862865447998
PCC:  0.3105452082208548
When K = 100:
U shape:  (10076, 100)
S shape:  (100,)
V shape:  (100, 489860)
Runtime: 17.68289542198181
PCC:  0.32094525386077183
Part 2 completed
```

```
Part 3 started.
When K = 20:
U shape:  (10076, 20)
V shape:  (20, 489860)
Runtime:  4.327005863189697
Iterations:  13
PCC:  0.13327521372850545
When K = 50:
U shape:  (10076, 50)
V shape:  (50, 489860)
Runtime:  4.36522102355957
Iterations:  7
PCC:  0.17063167589999859
When K = 100:
U shape:  (10076, 100)
V shape:  (100, 489860)
Runtime:  12.013965129852295
Iterations:  24
PCC:  0.179150166584646

Part 3 and 4 completed
```

Figure 1: Computing Matrix Factorization with SVD(left) with the scipy linalg library and with SGD(right) for k=20,50,100

Experimental Fine-tuning

1. **SGD Matrix Factorization**
   After initially obtaining the Pearson Correlation Coefficients for each $n \times k$ matrix I found that the correlation values were very low, even compared to the SVD coefficients so I spent decided to test out the algorithm with different learning rates, different regularization parameter values, different step sizes and also checked for convergence at different points. I found that higher k values required smaller parameters and larger step-sizes because values would more easily get large. I also adaptively tuned the step size instead of altering it with a consistent value each iteration, based on the change in loss it would either get larger or smaller.

2. **Positive Pointwise Mutual Information**
   Performance did not change much if at all after the PPMI optimization for both algorithms, however accuracy greatly improved. As seen in the figure below, the Pearson Correlation Coefficient was much higher when we used the document word frequency matrix constructed using PPMI.

```
Part 2 started...
When K = 20:
U shape:  (10076, 20)
S shape:  (20,)
V shape:  (20, 489860)
Runtime: 3.562746524810791
PCC:  0.3595720641353792
When K = 50:
U shape:  (10076, 50)
S shape:  (50,)
V shape:  (50, 489860)
Runtime: 8.072681903839111
PCC:  0.5296393839317874
When K = 100:
U shape:  (10076, 100)
S shape:  (100,)
V shape:  (100, 489860)
Runtime: 17.247297525405884
PCC:  0.5416925764564476
Part 2 completed
```

Figure 2: PPMI SVD Matrix Factorization for k=20,50,100

Semantically relevant words

1. **Semantically Relevant words**
   Yes, semantically relevant words are located closer to each other in the space as seen in Figure 3. This would be because they would often be used in the same context. i.e. ”white” is beside ”person” whereas something like ”men” and ”years” are on opposite ends. These semantically relevant words form ”communities” in which words often used in the same context are found, if we were to use more than the top 300 words we would see much larger communities but as we only used the top 300 the communities are going to be pretty small.

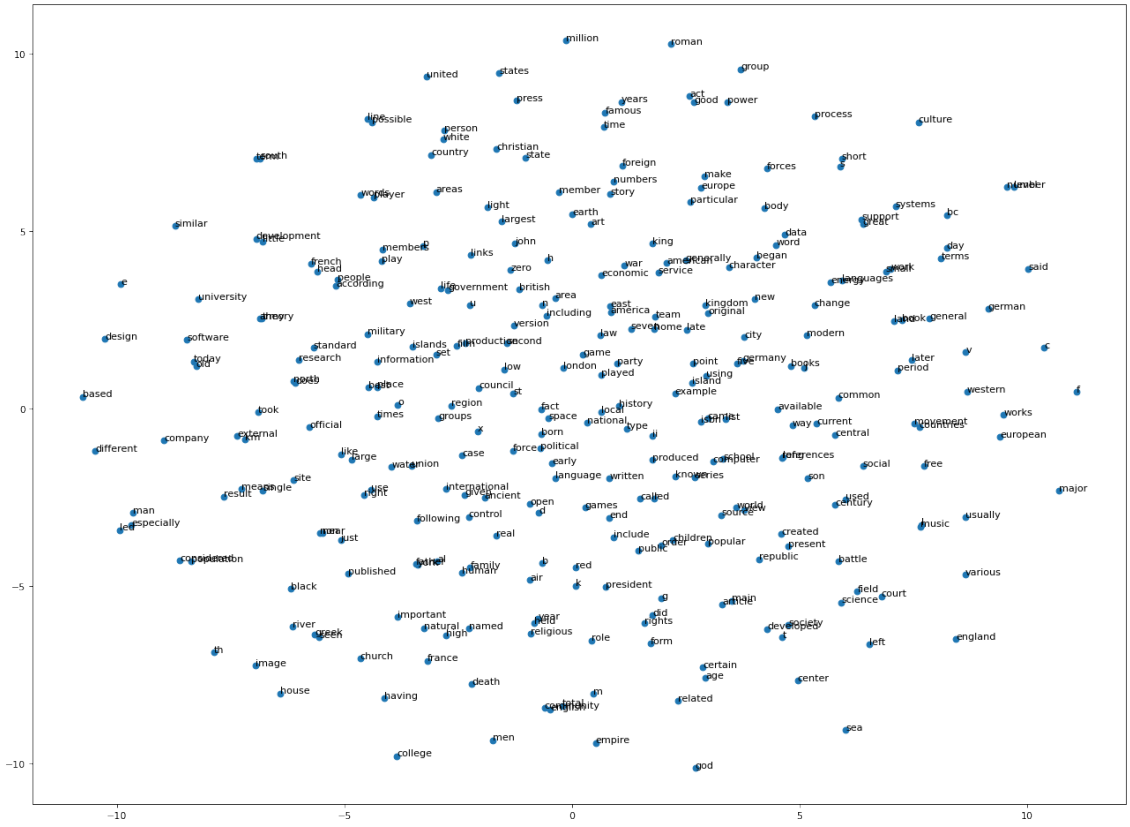

Figure 3: Visual representation of the top 300 most frequent words