CS168, Spring 2020

Mini-Project #5

Due by 11:59 PM on Tuesday, May 12th.

Instructions

- You can work in groups of up to four students. If you work in a group, please submit one assignment via Gradescope (with all group members' names).
- Detailed submission instructions can be found on the course website (https://web.stanford.edu/class/cs168) under "Coursework - Assignments" section.
- Use 12pt or higher font for your writeup.
- Make sure the plots you submit are easy to read at a normal zoom level.
- If you've written code to solve a certain part of a problem, or if the part explicitly asks you to implement an algorithm, you must also include the code in your pdf submission.
- Code marked as Deliverable should be pasted into the relevant section. Keep variable names consistent with those used in the problem statement, and with general conventions. No need to include import statements and other scaffolding, if it is clear from context. Use the `verbatim` environment to paste code in LaTeX.

```
def example():
     print "Your code should be formatted like this."
```

- **Reminder:** No late assignments will be accepted, but we will drop your lowest assignment grade.

Part 1: SVD for learning word embeddings

This question is about *word embeddings*. A word embedding is a mapping from words, to vector representations of the words. Ideally, the geometry of the vectors will capture the semantic and syntactic meaning of the words—for example, words similar in meaning should have representations which are close to each other in the vector space. A good word embedding provides a means for mapping text into vectors, from which you can then apply all the usual learning algorithms that take, as input, a set of vectors. Word embeddings have taken NLP by storm in the past 5-10 years, and have become the backbone for numerous NLP tasks such as question answering and machine translation. There are neural-network approaches to learning word embeddings, but in this question we will study a simple SVD based scheme which does a surprisingly good job at learning word embeddings.

We have created a word co-occurrence matrix M of the 10,000 most frequent words from a Wikipedia corpus with 1.5 billion words. Entry M_{ij} of the matrix denotes the number of times in the corpus that the ith and jth words occur within 5 words of each other. The file `co_occur.csv` contains the symmetric co-occurence matrix, M. The file `dictionary.txt` contains the dictionary for interpreting this matrix, the ith row of the dictionary is the word corresponding to the ith row and column of M. The dictionary is sorted according to the word frequencies. Therefore the first word in the dictionary—"the" is the most common word in the corpus and the first row and column of M contains the co-occurrence counts of "the" with every other word in the dictionary.

a. Make sure you can import the given datasets into whatever language you're using. If you're using MATLAB, you can import the data using the GUI. Also, make sure you can interpret the entries of the co-occurrence matrix using the dictionary, you can try to find the co-occurrence of a few pairs of common words to make sure of this.

b. (3 points) Let matrix M denote the 10000×10000 matrix of word co-occurrences. In light of the power law distribution of word occurrences, it is more suitable to work with the normalized matrix $\tilde{M}$ such that each entry $\tilde{M}_{ij} = \log(1 + M_{ij})$. For the remainder of this problem, we will work with this scaled matrix, $\tilde{M}$. Compute the rank-100 approximation of $\tilde{M}$ by computing the SVD $\tilde{M} = UDV^T$. Plot the singular values of $\tilde{M}$. Does $\tilde{M}$ seem to be close to a low rank matrix? [Hint: Computing the full SVD will take a bit of time—instead you should compute just the top 100 singular values/vectors...and save this decomposition rather than recomputing every time you work on this miniproject!]

c. (5 points) Note that as the matrix $\tilde{M}$ is symmetric, the left and right singular vectors are the same, up to flipping signs of some columns. We will now interpret the singular vectors (columns of U or V). For any i, denote v_i as the singular vector corresponding to the ith largest singular value. Note that the coordinates of this vector correspond to the 10,000 words in our dictionary. For a given vector v_i, you can see which words are most/least relevant for that vector by looking at the words corresponding to the coordinates of v_i that have the largest or smallest values. This allows you to get a rough sense for the semantic information that is captured in each of the singular vectors. Find 5 interesting/interpretable singular vectors, and describe what semantic or syntactic structures they capture. For each of the 5 vectors you choose, provide a list of the 10 words corresponding to the coordinates with the largest values and the 10 words corresponding to the coordinates of that vector with the smallest values. Not all of the singular vectors have easy-to-interpret semantics; why would you expect this to be the case?

d. Let U denote the 10000×100 matrix corresponding to the top 100 singular vectors. Normalize the rows of U such that each row has unit ℓ_2 norm. We will regard the ith row of U as the (100-dimensional) embedding of the ith word. We will now explore a curious property of these word embedding—that certain directions in the embedded space correspond to specific syntactic or semantic concepts. Let $\mathbf{v}_1$ be the word embedding for "woman" and $\mathbf{v}_2$ be the word embedding for "man". Let $\mathbf{v} = \mathbf{v}_1 - \mathbf{v}_2$.

   i (5 points) Project the embeddings of the following words onto $\mathbf{v}$: *boy, girl, brother, sister, king, queen, he, she, john, mary, wall, tree.* Present a plot of projections of the embeddings of these words marked on a line. For example, if the projection of the embedding for "girl" onto $\mathbf{v}$ is 0.1, then you should label 0.1 on the line with "girl". What do you observe?

   ii (5 points) Present a similar plot of the projections of the embeddings of the following words onto $\mathbf{v}$: *math, matrix, history, nurse, doctor, pilot, teacher, engineer, science, arts, literature, bob, alice.* What do you observe? Why do you think this is the case? Do you see a potential problem with this? (For example, suppose LinkedIn used such word embeddings to extract information from candidates' resumes to improve their "search for qualified job candidates" option. What might be the result of this?

   iii (3 points) Propose one method of mitigating the problem discussed in the previous part. There are many acceptable answers, and this is an ongoing area of research, so be creative and do not worry about making your answer overly rigorous. [Hint: If you need inspiration, see the original paper that surfaced this issue.]

e. In this question we will explore in more depth the property that directions in the embedded space correspond to semantic or syntactic concepts.

   i (4 points) First, we will define a similarity metric for words: the similarity of two words will be defined as the cosine-similarity between their embeddings. As all the word embedding vectors have unit ℓ_2 norm, the cosine similarity between two words i and j with embeddings $\mathbf{w_i}$ and $\mathbf{w_j}$ is equal to the inner product $\langle \mathbf{w_i}, \mathbf{w_j} \rangle$. Now that we have a similarity metric defined, we can have some fun with these embeddings by querying for the closest word to any word we like! Using this definition of similarity, what are the most similar words to "stanford"?

ii (10 points) Because word embeddings capture semantic and syntactic concepts, they can be used to solve word analogy tasks. For example, consider an analogy question— *"man is to woman as king is to ___"*, where the goal is to fill in the blank space. This can be solved by finding the word whose embedding is closest to $\mathbf{w}_{\text{woman}} - \mathbf{w}_{\text{man}} + \mathbf{w}_{\text{king}}$ in cosine similarity. You can do this by a nearest neighbor search across the entire dictionary—excluding the three words *man, woman, king* which already appear in the analogy as they cannot be valid answers to the question. Here $\mathbf{w}_i$ represents the word embedding for the word i.

We have provided a dataset, `analogy_task.txt`, which tests the ability of word embeddings to answer analogy questions. Using the cosine similarity metric, find and report the accuracy of the word embedding approach to solving the analogy task. Comment on the results–which analogies seemed especially difficult for this approach?

iii *(Bonus 5 points)* Your goal is to now improve the score of the word embeddings on the analogy task. Feel free to try any alternative scheme for learning word embeddings, but using only the co-occurrence matrix we have provided. Discuss the approaches you tried, what informed the choices, and the results they obtained. Showing that certain approaches obtain a poor accuracy is also valuable too. Your score will be based on a combination of the discussion and the best accuracy you obtain.

Deliverables: Discussions for part b, c, d, e. Plot for part b, d. Code for all parts in the appendix.

Part 2: SVD for image compression

Download http://web.stanford.edu/class/cs168/p5_image.gif. It is a 1600×1170, black and white drawing of Alice conversing with a Cheshire Cat. We will think of this image as a 1600×1170 matrix, with each black pixel represented as a 0, and each white pixel represented as a 1. We will observe this matrix under various approximations induced by its SVD.

a. (4 points) Before running SVD on Alice, think about what you expect the rank 1 approximation given by SVD to look like. To guide your thinking, consider the following simpler picture, where black pixels have value 0. Qualitatively describe the rank 1 approximation (given by SVD) of the following picture of the moon. Explain your reasoning. [Hint: it might be helpful for you to sketch what you expect the answer to be—and make sure it is rank 1!]

Figure 1: The Moon. Assume this is represented as a matrix of pixel values, with black = 0.

b. (6 points) Run SVD and recover the rank k approximation for the image of Alice, for $k \in \{1, 3, 10, 20, 50, 100, 150, 200, 400, 800, 1170\}$. In your assignment, include the recovered drawing for $k = 150$. Note that the recovered drawing will have pixel values outside of the range $[0, 1]$; feel free to either scale things so that the smallest value in the matrix is black and the largest is white (default for most python packages and matlab), or to clip values to lie between 0 and 1.

c. (2 points) Why did we stop at 1170?

d. (3 points) How much memory is required to efficiently store the rank 150 approximation? Assume each floating point number takes 1 unit of memory, and don't store unnecessary blocks of 0s. How much better is this than naively saving the image as a matrix of pixel values?

e. *(Bonus 3 points)* Details of the drawing are visible even at relatively low k, but the gray haze / random background noise persists till almost the very end (you might need to squint to see it at $k = 800$). Why is this the case? [For full credit, you need to explain the presence of the haze and say more than just "the truncated SVD is an approximation of the original image."]

Deliverables: Discussions for part a, b, c, d, e. Code for all parts in the appendix. Plot for part b.