XCS224N Assignment 1 Exploring Word Embeddings

Due Friday, July 2nd at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs224n-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some extra credit questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit only the `src/submission.py` file in the online student portal. For further details, see Writing Code and Running the Autograder below.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be foudn at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.

- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Hidden Tests

All hidden tests rely on files that are not provided to students. Therefore, the tests can only be run remotely. When a hidden test like `1a-1-hidden` is executed locally, it will produce the following result:

```
----- START 1a-1-hidden:  Test multiple instances of the same word in a sentence.
----- END 1a-1-hidden [took 0:00:00.011989 (max allowed 1 seconds), ???/3 points] (hidden test ungraded)
```

Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:

```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:

```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```


Remote Execution

Basic and hidden tests are treated the same by the remote autograder. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.

1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

Just like in the local autograder, this error caused the test to fail.

Just like in the local autograder, start your debugging in line 23 of grader.py.

1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

This error caused the test to fail.

Start your debugging in line 31 of grader.py.

Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.

1a-0-basic) Basic test case. (2.0/2.0)

1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

Before You Begin

Welcome to the first assignment of XCS224N! Please note that the core Assignment 1 (not including the Extra Credit which is presented separately) is divided into two tasks:

1. A coding assignment in which you will implement a co-occurrence based word embedding model.
2. A Google CoLab notebook in which you will experiment with pre-trained Word2Vec embeddings, using the results of these experiments to pass a multiple choice quiz. We recommend opening the notebook in a Google Chrome browser.

We highly recommend that you complete these two tasks in order (Part 1 before Part 2).

Word Embeddings (Refresher)

Presented below is a quick review on word embeddings and their role in NLP. Word Vectors are often used as a fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to build some intuitions as to their strengths and weaknesses. Here you will explore two types of word vectors: those derived from co-occurrence matrices (following section), and those derived via word2vec (last section). In this refresher, we will focus more intently on the details of count-based (co-occurrence) embeddings.

Note on Terminology: The terms "word vector" and "word embedding" are often used interchangeably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space. As Wikipedia states, "conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension".

0.1 Count-Based Word Vectors (Co-occurence)

A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ and $w_{i+1} \dots w_{i+n}$. We build a co-occurrence matrix $\mathbf{M}$, which is a symmetric word-by-word matrix in which $\mathbf{M}_{ij}$ is the number of times w_j appears inside w_i's window. We provide an example of such a matrix $\mathbf{M}$ with window-size $n = 1$ for the mock documents below:

Document 1: "all that glitters is not gold"
Document 2: "all is well that ends well"

$$\mathbf{M} = \begin{array}{c} \\ \text{START} \\ \text{all} \\ \text{that} \\ \text{glitters} \\ \text{is} \\ \text{not} \\ \text{gold} \\ \text{well} \\ \text{ends} \\ \text{END} \end{array} \begin{array}{c} \begin{array}{cccccccccc} \text{START} & \text{all} & \text{that} & \text{glitters} & \text{is} & \text{not} & \text{gold} & \text{well} & \text{ends} & \text{END} \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Note: In NLP, we often add START and END tokens to represent the beginning and end of sentences, paragraphs or documents. In this case we imagine START and END tokens encapsulating each document, e.g., "START All that glitters is not gold END", and include these tokens in our co-occurrence counts.

The rows (or columns) of this matrix provide one type of word vector (those based on word-word co-occurrence), but the vectors will be large in general (linear in the number of distinct words in a corpus). Thus, our next step is to run *dimensionality reduction*. In particular, we will run *SVD (Singular Value Decomposition)*, which is a kind of generalized *PCA (Principal Components Analysis)* to select the top k principal components. Figure 1 provides a visualization of dimensionality reduction using SVD. In this picture our co-occurrence matrix is $\mathbf{A}$ with n rows corresponding to n words. We obtain a full matrix decomposition, with the singular values ordered in the diagonal $\mathbf{S}$ matrix, and our new, shorter-length-k word vectors in $\mathbf{U}_k$.

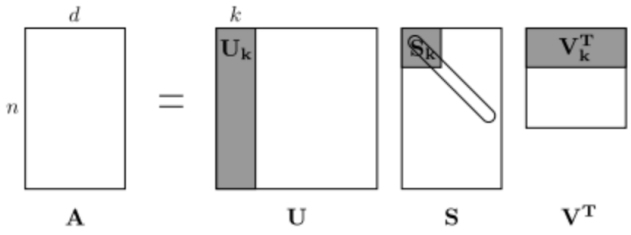


Figure 1: Dimensionality reduction by truncated SVD

This reduced-dimensionality co-occurrence representation preserves semantic relationships between words, e.g. doctor and hospital will be closer than doctor and dog.

Those interested in getting a deeper mathematical understanding of dimensionality reduction should complete the Extra Credit homework assignment. A slow, friendly introduction to SVD can be found here: https://davetang.org/file/Singular_Value_Decomposition_Tutorial.pdf. Alternatively, the SVD is a commonly discussed mathematical concept, and a quick Google search will yield a number of valuable resources!

1 Computing Co-occurence Embeddings

In this part of the assignment you will implement a co-occurrence based word embedding model. We will be using the Reuters (business and financial news) corpus. This corpus consists of 10,788 news documents totaling 1.3 million words. These documents span 90 categories and are split into train and test sets. For more details, please see https://www.nltk.org/book/ch02.html.

(a) **[2 points (Coding)]** Implement the `distinct_words` function in `src-co_occurence/submission.py`. You can do this with `for` loops, but it's more efficient to do it with Python list comprehensions.

(b) **[5 points (Coding)]** Implement the `compute_co_occurrence` function in `src-co_occurence/submission.py`. If you aren't familiar with the python `numpy` package, we suggest walking yourself through this tutorial: http://cs231n.github.io/python-numpy-tutorial.

(c) **[2 points (Coding)]** Implement the `reduce_to_k_dim` function in `src-co_occurence/submission.py`.

Note: All of numpy, scipy, and scikit-learn (sklearn) provide some implementation of SVD, but only scipy and sklearn provide an implementation of Truncated SVD, and only sklearn provides an efficient randomized algorithm for calculating large-scale Truncated SVD. Please use sklearn.decomposition.TruncatedSVD.

(d) (0 points) Show time! Now we are going to load the Reuters corpus and compute some word vectors with everything you just implemented! There is no additional code to write for this part; just run `python submission.py` from within the `src-co_occurence/` subdirectory. This will generate a plot of the embeddings for hand-picked words which have been dimensionally reduced to 2-dimensions. Try and take note of some of the clusters and patterns you do or don't see. Think about what you can and can't make sense of. **The plot will be referenced in Question 1 of the multiple choice question set in Part 2 below**.

Once finished with this part of your assignment, upload your `src-co_occurence/submission.py` file via the Gradescope submission link in the Assignment 1 block of your SCPD learning portal. The submission link is titled **Assignment 1 Coding Submission Link**.

2 Experimenting with Word2Vec Embeddings

In this section you will be experimenting with pre-trained Word2Vec embeddings in a Google CoLab python notebook. We highly recommend opening the notebook in a Google Chrome browser. To get started, you will need to access the notebook here: http://bit.ly/2rZZj1y.

Google CoLab notebooks are shared and saved just like Google Docs. Upon visiting the notebook, go to"File > Save a copy in Drive". You will then be able to see a copy of the notebook in a folder named "Colab Notebooks" inside your Google Drive. This is the file you should work with.

- The notebook is divided into individual cells. When running code in the notebook, one can run one cell at a time, and the output of the code will populate under the cell. The values of any variables used in this cell will be stored and can be used when running future cells. The variable values will be written over if the cell is run again.
- One of the cells loads in the Word2Vec embeddings. This takes a good bit of time (10-15 mins)! We highly recommend that you run this cell once and then use the stored word vectors in your future experiments without reloading the embeddings. The notebook is set up in a fashion conducive to this use.

This remainder of this homework is a series of multiple choice and True/False questions related to the CoLab notebook exercise described on the previous page.

How to submit: Even though these are not coding questions, you will submit your response to each question in the `src-quiz/submission.py` file. This file will act as your 'bubble sheet' for multiple choice questions in this course. A sample response might look like this:

```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  ### END CODE HERE ###
  return response
```

If you believe that `a` and `b` are the correct responses to this question, you will type `response = ['a', 'b']` between the indicated lines like this:

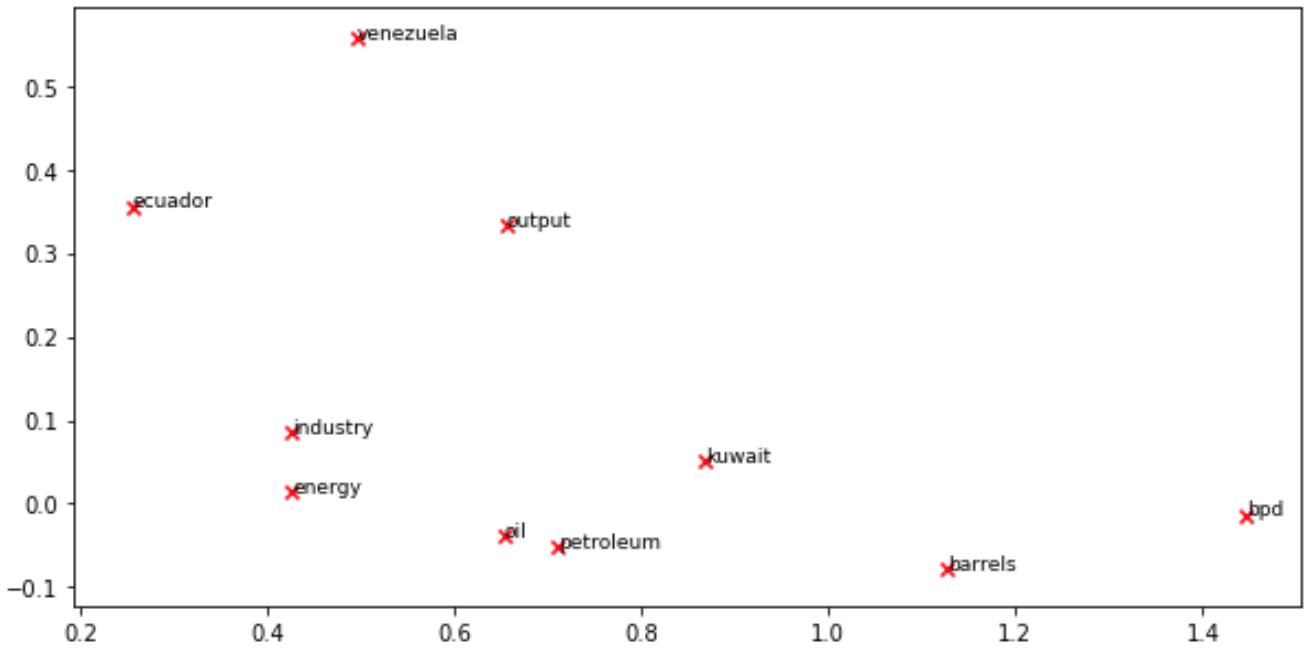
```
def multiple_choice_1a():
  """
  # Return a python collection with the option(s) that you believe are correct
  # like this:
  # `return ['a']`
  # or
  # `return ['a', 'd']`
  response = []
  ### START CODE HERE ###
  response = ['a', 'b']
  ### END CODE HERE ###
  return response
```

How to verify your submission: You can run the student version of the autograder locally like all coding problem sets. In the case of this problem set, the helper tests will verify that your responses are within the set of possible choices for each question (e.g. the helper functions will flag if you forget to answer a question of if you respond with `['a', 'd']` when the choices are `['a', 'b', 'c']`.) See the front pages of this assignment for instructions to run the autograder.

Once you have opened the Google CoLab notebook, begin following the steps there and inputting your answers into this quiz.

1. **[2 points]**

   We provide a similar plot to the one you generated in part one of the assignment, this time using Word2Vec embeddings.



   Why aren't countries "venezuela","ecuador" and "kuwait" clustered together in the Word2Vec plot while as they in your co-occurence plot? - Select all the reasonable possibilities

   A) Word2Vec was trained on a larger dataset in which the countries did not always appear in the same contexts as the small dataset used to compute the co-occurence matrix.

   B) Word2Vec recognizes that the words are countries, and thus groups them by geography, while the co-occurence method does not.

   C) For the Word2Vec embeddings, the first two principal components may represent attributes of the words "venezuela", "ecuador" and "kuwait" which are not similar, while in the co-occurence embeddings, they may represent attributes of the words which are similar.

   D) In the Word2Vec corpus, the countries "venezuela", "ecuador" and "kuwait" do not appear in each others' contexts, so they are not similar in the embeddings.

2. **[2 points]**

   For the following homonyms, respond True ("T") if the top 10 most similar words represent more than one meaning for the homonym and False ("F") otherwise. Please make sure to input the word exactly as specified below:

   a. mole

   b. nuts

   c. pen

   d. right

e. drive

f. rose

g. mean

h. saw

3. **[2 points]**

Why do you think many of the homonyms you tried didn't work? - Select all the reasonable possibilities.

A) A different word vector is trained for each meaning of the homonym, so synonyms for a particular meaning of the word will be most similar depending on the which vector we are using for the original word.

B) In some scenarios, one meaning of the homonym is much more common in the training data than the other meanings, resulting in the vector for the homonym is much closer to words from one meaning than the other(s).

C) When a particular meaning of a homonym is much more common than the others, the algorithm that trains the word vectors is able to recognize this and only learns to encode one of the definitions in the word vector.

4. **[2 points]**

Mark True ("T") if the antonyms pair has a smaller cosine distance than the synonyms pair. Mark False ("F") otherwise. Please make sure to input the words exactly as specified below:

a. happy, sad | happy, cheerful

b. right, wrong | right, correct

c. big, small | big, large

d. good, bad | good, nice

e. day, night | day, daytime

f. insane, sane | insane, crazy

g. several, one | several, numerous

h. antonym, synonym | antonym, opposite

5. **[2 points]**

Why do you think there are times where synonym-antonym pairs have a smaller cosine distance than the synonym-synonym pairs? - Select all the reasonable possibilities.

A) Sometimes, the synonym-antonym pair has two words that are found with much higher frequency in the corpus than the other synonym word. The word vectors capture this frequency, making the synonym-antonym pair more similar than the synonym-synonym pair.

B) In some cases, the antonym is a homonym and carries multiple meanings, some of which may have definitions slightly similar to the synonym, making them overall fairly similar.

C) We may find that the synonym-antonym words are actually used in similar contexts, more so than the synonym-synonym words, making the synonym-antonym words more similar.

6. **[3 points]**

Mark True ("T") if the word vectors are able to successfully complete the analogy. Mark False ("F") otherwise. Please make sure to input the words exactly as specified below:

a. man:king::woman:QUEEN

b. air:plane::sea:BOAT

c. happy:laugh::sad:CRY

d. cat:kitten::dog:PUPPY

e. France:Paris::Germany:BERLIN

f. carnivore:meat::herbivore:VEGETABLES

g. dog:bark::cat:MEOW

h. bat:baseball::racquet:TENNIS

i. mosque:Islam::church:CHRISTIANITY

j. long:longer::short:SHORTER

k. longer:longest::more:MOST

l. talk:talked::help:HELPED

7. [**2 points**]

What factors might contribute to the biases observed in the word vectors presented in the notebook examples? - Select all the reasonable possibilities.

A) Text data is generated by people, and thus word vectors trained on a corpus are prone to have the same biases of the people who generated the corpus

B) The training corpus may have many instances of sentences that relate certain races, genders, etc. to certain properties or behaviours

3 Primer on Singular Value Decomposition

3.1 What is the SVD?

The following section serves as a primer about **Singular Value Decomposition** (SVD). All the material in this section will be presented as fact (without proof) and we will ask you to use it to answer the extra credit questions in the sections below. Suppose we have some matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with rank r. Then $\mathbf{A}$ has a decomposition given by

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T \tag{1}$$

which has the following (along with other) properties

- $\mathbf{U} \in \mathbb{R}^{n \times r}$, $\mathbf{D} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{d \times r}$.
- The matrix $\mathbf{D}$ is diagonal with positive entries sorted in descending order. We often let let $\sigma_i := \mathbf{D}_{ii}$, and we then have $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$. We call σ_i the ith singular value of the $\mathbf{A}$.
- The columns of $\mathbf{V}$ form an orthonormal basis for the row space of $\mathbf{A}$, and thus $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. The ith column $\mathbf{v}_i$ of $\mathbf{V}$ is known as the ith right singular vector of $\mathbf{A}$.
- The columns of $\mathbf{U}$ form an orthonormal basis for the column space of $\mathbf{A}$, and thus $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. The ith column $\mathbf{u}_i$ of $\mathbf{U}$ is known as the ith left singular vector of $\mathbf{A}$.

The above decomposition is known as the SVD of $\mathbf{A}$.

3.2 Projections

Recall that for a linear subspace $W \subset \mathbb{R}^p$, the projection of some vector $\mathbf{v}$ onto W is defined as

$$\text{proj}_W(\mathbf{v}) := \underset{w \in W}{\arg\min} ||w - \mathbf{v}||_2^2 \tag{2}$$

In the special case that W is a one dimensional vector space spanned by some $\mathbf{w}$, we call this projection the projection of $\mathbf{v}$ onto $\mathbf{w}$ which has simple closed form. We give it below, where we have let $\hat{\mathbf{w}} := \mathbf{w}/||\mathbf{w}||_2$ represent the unit vector pointing in the direction of $\mathbf{w}$.

$$\text{proj}_{\mathbf{w}}(\mathbf{v}) = \left(\frac{\mathbf{v}^T\mathbf{w}}{||\mathbf{w}||_2^2}\right)\mathbf{w} = (\mathbf{v}^T\hat{\mathbf{w}})\hat{\mathbf{w}} \tag{3}$$

Intuitively, the quantity $\mathbf{v}^T\hat{\mathbf{w}}$ captures the amount that $\mathbf{v}$ points in the direction of $\mathbf{w}$.

3.3 Best Fit Sub-spaces

Suppose we have a set of vectors $S = \{\mathbf{w}_1, \ldots, \mathbf{w}_n\}$ where each $\mathbf{w}_i \in \mathbb{R}^d$. We define the k-dimensional best-fit subspace for S to be the k-dimensional linear subspace W such that

$$W = \underset{\dim(W)=k}{\arg\min} \sum_{i=1}^{n} ||\mathbf{w}_i - \text{proj}_W(\mathbf{w}_i)||_2^2 \tag{4}$$

Intuitively, W is the k-dimensional subspace that is closest to the vectors in set S.

It is an important and interesting fact that, for a matrix $\mathbf{A}$ with SVD as given above, the subspace given by $V_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is the k-dimensional best-fit subspace for the rows of $\mathbf{A}$. Informally, σ_i tells us the degree to which $\mathbf{v}_i$ contributes to fitting the rows of $\mathbf{A}$. Thus the vector which best fits the rows of A is $\mathbf{v}_1$, the vector which best fits the rows of $\mathbf{A}$ when the contribution of $\mathbf{v}_1$ is accounted for is $\mathbf{v}_2$, etc.

4 Extra Credit Challenge (5 Points)

Suppose we have a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with SVD $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{n \times r}$, $\mathbf{D} \in \mathbb{R}^{r \times r}$, $\mathbf{V} \in \mathbb{R}^{d \times r}$.

(a) **[1 point (Written, Extra Credit)]** Show that

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{5}$$

(b) **[1 point (Written, Extra Credit)]** Show that

$$\mathbf{u}_i = \frac{1}{\sigma_i}\mathbf{A}\mathbf{v}_i \tag{6}$$

In particular, the components of $\mathbf{u}_i$ represent the size of the projection of the rows of $\mathbf{A}$ onto $\mathbf{v}_i$ (scaled by σ_i).

(c) **[1 point (Written, Extra Credit)]** One way of finding a reduced rank approximation of $\mathbf{A}$ is by hard-setting all but the k largest σ_i to 0. This approximation is called the truncated SVD, and by (a) we see it can be written as

$$\mathbf{A}_k := \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{7}$$

From (a), we see the truncated SVD can also be written as $\mathbf{A}_k = \mathbf{U}_k \mathbf{D}_k \mathbf{V}_k^T$, where $\mathbf{U}_k \in \mathbb{R}^{n \times k}$, $\mathbf{V} \in \mathbb{R}^{d \times k}$ are the first k columns of $\mathbf{U}, \mathbf{V}$, and $\mathbf{D} \in \mathbb{R}^{k \times k}$ has the first k singular values.

Show that the rows of $\mathbf{A}_k$ are the projections of the rows of $\mathbf{A}$ onto the subspace of V_k spanned by the first k right singular vectors.

Hint: Recall that the projection of a vector $\mathbf{a}$ onto a subspace spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_k$ where the $\mathbf{v}_i$ are pairwise orthogonal is given by the sum of projections of $\mathbf{a}$ onto the individual $\mathbf{v}_i$.

(d) **[2 points (Written, Extra Credit)]** The Frobenius norm of a matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ is defined as

$$||\mathbf{M}||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} M_{ij}^2} \tag{8}$$

Show that

$$\mathbf{A}_k = \underset{\text{rank}(\mathbf{B})=k}{\arg\min} \; ||\mathbf{A} - \mathbf{B}||_F \tag{9}$$

where the arg min is taken over matrices of rank k.

Hint: Use the fact that V_k is the best-fit k-dimensional subspace for the rows of $\mathbf{A}$.

5 Interpretation

Suppose we have some data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ with rows x_i. The x_i represent n data samples each with d entries. In the case that d is a very large number, we may want to try and reduce the dimensionality of our data. Suppose we want to reduce the dimension of each sample from d to $k < d$. How can we go about doing this?

Consider taking the truncated SVD $\mathbf{X}_k$ of our data. From (c) we know that we have transformed our data so all the data samples live within a k-dimensional linear subspace, in particular, the k-dimensional linear subspace that best-fit our original data. Part (d) tells us that under the Frobenius norm, our new matrix $\mathbf{X}_k$ is as close to $\mathbf{X}$ as it can be given that it has this property.

Our new approximate data samples (given by the rows of $\mathbf{X}_k$) have dimension d, but they certainly don't need to. Since each sample lives in the same k-dimensional linear subspace, each sample's place in this subspace relative to other samples can be determined by how much it "points" in k fixed directions which span the subspace. Thus it suffices to fix k basis vectors for the subspace (these are our k directions) and re-parametrize each sample based on how much it "points" in the direction of each of these basis vectors.

From (c) we know that we can let $\mathbf{v}_i, \ldots, \mathbf{v}_k$ be a basis of our subspace. From (b), we then know that the ith row of $\mathbf{U}$ denoted u_i is such that its jth component $(u_i)_j$ tells us how much the ith sample points in the direction of $\mathbf{v}_j$. Thus, based on the above discussion, we can use the vector $((u_i)_1, \ldots, (u_i)_k)$ as a substitute for the ith row of $\mathbf{X}_k$. This leaves us with a k dimensional embedding for each of our original d dimensional data samples. The end result is that we use $\mathbf{U}_k$ as our new data, and it represents an approximated and transformed $\mathbf{X}$.

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the `README.md` for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

4.a

4.b

4.c

4.d