Assignment #5

Due: 23:55pm March 25, 2020

Upload at: https://www.gradescope.com/courses/75501/assignments/379110

Assignments in COS 302 should be done individually. See the course syllabus for the collaboration policy.

Remember to append your Colab PDF as explained in the first homework, with all outputs visible.
When you print to PDF it may be helpful to scale at 95% or so to get everything on the page.

Problem 1 (10pts)

We sometimes need to introduce norms for matrices. This comes up when we want to talk about the magnitude of a matrix, or when we need a notion of distance between matrices. One important matrix norm is the Frobenius norm, which can be written in several ways for a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$:

$$||\boldsymbol{A}||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}^2} = \sqrt{\text{trace}(\boldsymbol{A}^T \boldsymbol{A})}$$

(A) Show that the Frobenius norm is also the square root of the sum of the squared singular values.

(B) Assume that $\boldsymbol{A}$ is square and invertible, with a very small Frobenius norm. What kind of value would you expect to get for the Frobenius norm of $\boldsymbol{A}^{-1}$?

Problem 2 (23pts)

In this problem, you'll look at singular value decomposition as a way to find a low-rank approximation to a matrix.

(A) Upload any (tasteful) image you want to Colab. Note that life is a little easier if you put it in your Google drive and access it from there so you don't have to constantly re-upload it every time you come back after a break. Load the image as a NumPy array. There are various ways to do this, but the easiest may be to use `imread` from matplotlib. There is a useful tutorial here also. Once you have a NumPy array, look at its shape: it is probably $width \times height \times 3$ or something to that effect, with the third dimension usually reflecting red, green, and blue channels. Make the image grayscale by taking the mean across this third dimension. This should make it into a $width \times height$ array. Use `imshow` as in previous homeworks to render the grayscale image.

(B) Import `numpy.linalg` and use the `svd` function to compute the singular value decomposition of the image matrix. This will return three things: a $\boldsymbol{U}$ matrix, a vector containing the singular values, and a $\boldsymbol{V}^T$ matrix. Print the shapes of these arrays, and then figure out how to "reassemble" these three arrays with multiplication to reconstruct the image. Render the reconstruction and verify that it looks like the original image.

(C) Plot the cumulative sum of the singular values. Sum them from largest to smallest.

(D) Create three different low-rank approximations to your image. Create one of rank 5, one of rank 10, and one of rank 25. (Hint: you can do this by reconstructing the image matrix as above, but with zeros for all the singular values with index larger than the rank of your approximation. Render each of the three images.

(E) Create three new matrices, each with the pixel-wise squared difference between the original image and the low-rank approximation. Render these as images and qualitatively describe what kind of visual structure seems to be lost in the low-rank approximations.

Problem 3 (25pts)

In this problem you will use SVD to model a text corpus, a small subset of New York Times articles. You will use a "bag of words" representation in which documents are represented by the counts of words, usually excluding very common "stop words" like *the* and *and*. Download `nyt.pkl.gz` and upload it to your Google drive so you don't have to upload it every time you open the Colab; it's a fairly big file. Here's some code to get you going:

```
import pickle as pkl
import numpy as np
import gzip

filename = 'drive/My Drive/COS 302/nyt.pkl.gz'
with gzip.open(filename, 'rb') as fh:
  nyt = pkl.load(fh)
documents = nyt['docs']
vocab = nyt['vocab']

# Create reverse lookup table.
vocab_indices = dict([(w, i) for (i, w) in enumerate(vocab)])

M = len(documents)
N = len(vocab)
print('%d documents, %d words' % (M,N))

count_mat = np.zeros((M,N))
for mm, doc in enumerate(documents):
  for word, count in doc.items():
    count_mat[mm, vocab_indices[word]] = count
```

(A) Typically, raw counts don't lead to discovery of interesting structure. Instead, it is common to use something like TF-IDF, which stands for *term frequency-inverse document frequency*. Term frequency is the number of times word n appeared in document m, divided by the total number of words in document m.

$$\text{tf}_{m,n} = \frac{\text{\# times word } n \text{ appears in doc } m}{\text{total \# of words in doc } m}$$

Transform `count_mat` from the code above into a term frequency matrix.

(B) Inverse document frequency is typically the natural log of the number of documents, divided by the number of documents in which word m appears. (The plus-one ensures that you're not dividing by zero.)

$$\text{idf}_n = \log \frac{\text{total \# of documents}}{1 + \text{\# of documents with word } n}$$

Compute the idf vector from `count_mat`.

(C) Now compute the TF-IDF matrix by multiplying (broadcasting) $\text{tf}_{m,n}$ and idf_n. Use `numpy.linalg.svd` to take the SVD of this TF-IDF matrix; it may take a minute since the matrix is relatively large. Plot the singular values in decreasing order.

(D) The right singular vectors (columns of the V matrix) will ideally represent interesting topical dimensions. For each of the top 20 right singular vectors: identify the words in the vocabulary that have the largest entries in the vector and print them. That will probably mean looping over the first 20 right singular vectors, doing an appropriate `argsort` and then finding that entry in the `vocab` variable. Do you see any interesting qualitative structure in these groups of words?

Problem 4 (25pts)

In various data analysis problems, it is often easier to reason about the pairwise distances between data, rather than features of the data directly. This comes up particularly when dealing with discrete data where there aren't vectors, but perhaps there is a sensible concept of distance. A prime example is strings, in which there are various sensible edit distances. We previously examined principal component analysis (PCA) as a way to find low-dimensional representations of data, but if you only have distances, vanilla PCA doesn't apply. Instead, the classic approach is to use principal *coordinates* analysis, also called multidimensional scaling (MDS) to map the data to $\mathbb{R}^d$ in such a way that the pairwise distances are approximately preserved. That is, you are given a matrix of squared distances $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, and your goal is to discover reasonable locations $\{\boldsymbol{x}_i\}_{i=1}^n$ for, say, $\boldsymbol{x} \in \mathbb{R}^2$, so that $D_{ij} \approx ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2$. The details of MDS are beyond the scope of this course, but the steps are straightforward applications of tools that you've been learning to use in COS 302.

(1) Compute a squared distance matrix $\boldsymbol{D}$.

(2) Subtract off the column-wise means and the row-wise means of $\boldsymbol{D}$, i.e., perform "double centering".

(3) Compute the largest eigenvalues $\lambda_1, \lambda_2, \ldots$ and associated eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots$ of $-\frac{1}{2}\boldsymbol{D}$.

(4) The jth entry of the vector $\sqrt{\lambda_i}\boldsymbol{v}_i$ can now be used as the ith coordinate of $\boldsymbol{x}_j$. That is, for mapping data into $\mathbb{R}^2$, you would form a $n \times 2$ matrix $\boldsymbol{X} = \begin{bmatrix} \sqrt{\lambda_1}\boldsymbol{v}_1 & \sqrt{\lambda_2}\boldsymbol{v}_2 \end{bmatrix}$ whose rows are locations to embed each datum.

In this problem, you're going to take a list of strings, find edit distances between them all, and then make a visualization of them in $\mathbb{R}^2$. By default, you can use `dog_names1000.txt`, a list of dog names taken from a random subset of registered dogs in Anchorage, Alaska; you can use any list of strings you want, however, as long as it has at least 500 or so entries: lists of cities, street names, metal bands, etc. You'll need to figure out how to load the string from file into a list, but then the function below will compute a squared distance matrix for you.

```
import numpy as np
import editdistance
def sq_distances(names):
  '''Takes a list of strings. Returns squared edit distances.'''
  N = len(names)
  sq_dists = np.eye(N)
  for ii, name1 in enumerate(names):
    for jj, name2 in enumerate(names[:ii]):
      sq_dists[ii,jj] = (editdistance.eval(name1, name2) \
                         / np.maximum(len(name1), len(name2)))**2
      sq_dists[jj,ii] = sq_dists[ii,jj]
  return sq_dists
```

Implement the MDS procedure above to put your strings into locations in $\mathbb{R}^2$ and then plot them. Assuming you have a $n \times 2$ matrix `X`, the code below should get you started making the figure.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(20,20))
plt.plot(X[:,0], X[:,1], '.')
for ii in range(n):
  plt.text(X[ii,0], X[ii,1], names[ii], fontsize=10)
plt.show()
```

Problem 5 (2pts)
Approximately how many hours did this assignment take you to complete?

My notebook URL: https://colab.research.google.com/XXXXXXXXXXXXXXXXXXXXXX

Changelog

- 5 March 2020 – Initial version.