Assignment #3
Due: 12:00 noon EDT, September 28, 2020

Upload at:

Problem 1 (14pts)

A convex set is a set with the property that it is closed under convex combination. That is, if C is a convex set, then $\forall x, y \in C$ and $\alpha \in [0, 1]$:

$$\alpha x + (1 - \alpha)y \in C .$$

For example, a convex polygon has the property that you can pick two points, and the line segment between them (which is the set of linear combinations for which $\alpha \in [0, 1]$) lies entirely within the polygon.
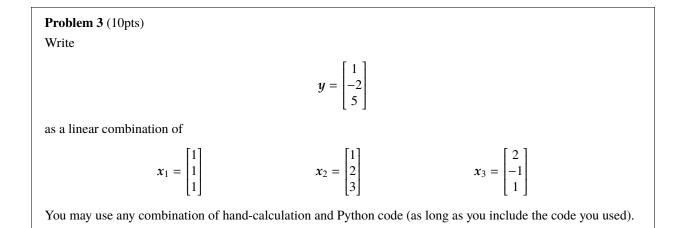
Prove that the set of $n \times n$ Symmetric Positive Definite (SPD) matrices is a convex set. (Hint: Think about how positive definiteness is affected by addition and by scaling with a positive number.)

Problem 2 (14pts)

Determine whether the following vectors are linearly independent. You may use any combination of hand-calculation and Python code (as long as you include the code you used).

(A)

$$x_1 = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 3 \\ -3 \\ 8 \end{bmatrix}$$

(B)

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad x_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Problem 3 (10pts)

Write

$$y = \begin{bmatrix} 1 \\ -2 \\ 5 \end{bmatrix}$$

as a linear combination of

$$x_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad x_3 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

You may use any combination of hand-calculation and Python code (as long as you include the code you used).

Problem 4 (30pts)

One important Pythonic skill is the ability to load, manipulate, and store external data in files. There are several different ways to do this, but one of the most powerful and convenient tools is the pickle library. (Pickling being a way to safely prepare food for long-term storage.) The first step towards pickling data in Python is to understand how to open files. Tutorials and documentation abound online, but for pickling, the most straightforward thing to do is to use the `with open` pattern. For reading, this typically looks like:

```python
import pickle as pkl

with open('in_file.pkl', 'rb') as fh:
    unpickled_data = pkl.load(fh)
```

What's happening here is that we're opening a file (that here conventionally has the extension `.pkl`) in "read binary" mode as indicated by the `'rb'`. The variable `fh` is a handle to the opened file that here we just hand to the pickle library to process. Writing pickled data is very similar:

```python
with open('out_file.pkl', 'wb') as fh:
    pkl.dump(data_to_pickle, fh)
```

We're opening a file in "write binary" mode and then handing the `dump` function a Python variable and the filehandle we just opened. The pickle library knows how to handle many different kinds of Python data structures; one powerful thing to do is to give it a `dict` with a bunch of different things you'd like to save.

(A) Download the `coords.pkl` file from the course website. Create a Colab notebook as in the previous two assignments. Upload the `coords.pkl` file and write code to open and unpickle it. After you load it, you'll have a NumPy array that is a 296×2 matrix. As the name indicates, these are coordinates in 2d. The first column are x and the second column are y. Plot these data using Matplotlib. This will look best if it is large with equal aspect ratio; the code below might be helpful for that.

```python
fig, ax = plt.subplots(figsize=(15,15))
# ... plotting ...
ax.set_aspect(1)
```

(B) In graphics and computer vision, affine transformations are a powerful way to deform objects and images. We'll look at a simple version of this here, in which we'll use 2×2 matrix to transform the two-dimensional coordinates you loaded above. For example, these linear maps rotate, scale, and shear:

$$\text{rotate:} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \text{scale:} \begin{bmatrix} \text{scale}_x & 0 \\ 0 & \text{scale}_y \end{bmatrix} \qquad \text{shear:} \begin{bmatrix} 1 & \text{shear}_x \\ \text{shear}_y & 1 \end{bmatrix}$$

Use `subplot` to make three subplots, rendering the coordinate data *after* applying each of these three transformations for parameter values of your choosing. Make sure to set the subtitles appropriately.

(C) Do these transformation commute? Create a rotation matrix and a shear matrix for some non-trivial parameters. Create a figure with two subplots, in which you compose these transformations: one with rotate-then-shear and another that is shear-then-rotate. Make sure to set the subtitles appropriately.

Problem 5 (30pts)

Even with just knowledge of vector norms, you already have a powerful way to construct machine learning classifiers using the *k*-nearest neighbors (KNN) algorithm. In KNN, as in many kinds of machine learning problems, we have a *training set* of data that we use to build the classifier, and our objective is to make accurate predictions on data we haven't seen before, i.e., the *test data*. If an algorithm makes good predictions on previously-unseen ("out of sample") data, then we say that it *generalizes*. Building algorithms that can generalize to new data is what machine learning is all about.

The nearest neighbor algorithm centers on a simple concept: when given a new piece of data, find the datum in your training set that is most similar to it, and return that label. The *k*-nearest neighbor algorithm does the same thing, but takes the majority vote of the *k* training examples that are most similar to it. Of course for this to work, we need to specify what it means to be "similar" and for that we can use a metric: a measure of distance as determined by the norm of the distance between the data.

(A) Download the `mnist2000.pkl` file from the course website and then upload it to Colab. MNIST is a famous data set for the problem of handwritten digit recognition for, e.g., reading zip codes off of envelopes. The data are 28×28 greyscale images of the digits 0 through 9. This pickle file contains a subset with 2000 training images and 150 test images; the pickle contains a dictionary with keys `train_images`, `train_labels`, `test_images`, and `test_labels`. The associated values are NumPy arrays: the images for train and test are in arrays of size (2000,28,28) and (150,28,28), respectively, and the labels are in 1d arrays. The pixels of the images take values in $[0, 1]$ and the labels are integers in the range $0, 1, \ldots, 9$. Use the `imshow` function from Matplotlib to render one of the training images via a command along the lines of `plt.imshow(train_images[n,:,:])`. Print the corresponding training label for whatever `n` you used. Do they make sense?

(B) Write a function called `classify` that takes in a test image in the form of a 28×28 NumPy array, and returns a digit based on the nearest neighbor. Use basic squared Euclidean distance in pixels to make the classification. The steps for this will be: 1) compute all the squared distances between the test image and the training data, probably using a broadcast and a sum, 2) use the `argmin` function to determine the index of the training image with the smallest distance to the test image, 3) use that index to select the right training label and return it.

```
def classify(test_image):
  sq_dists = # ???
  smallest = # ???
  return # ???
```

Test your function on a couple of the images from the test set and see if it behaves reasonably.

(C) Loop over all the test images and report the test accuracy (percent correct) of your classifier.

(D) That was a *k*-nearest neighbor classifier for $k = 1$. Make a similar function called `knn_classify` that also takes *k* as an argument and works for any *k*. You'll probably want to use `argsort` instead of `argmin`, to get the closest *k*. Determining the most common label among the *k* nearest might be slightly more involved. There are several ways to do it, but I suggest looking at the `unique` function for ideas. Try it out for a couple of values of *k* (say 5 and 10) and report the accuracies. For an additional challenge, see if you can do this without writing any explicit loops.

Problem 6 (2pts)

Approximately how many hours did this assignment take you to complete?

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

Changelog

- 13 September 2020 – Updated for fall 2020.
- 16 February 2020 – Initial version.