IMGS 682 Spring 2018 - Homework 1
Linear Algebra, Classifiers, and PCA

Due: See My Courses

Instructions

Your homework submission must cite any references used (including articles, books, code, websites, and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. If you do work with others, you must list the people you worked with. Submit your solutions as a PDF to the Dropbox Folder on MyCourses.

Your homework solution must be prepared in LaTeXand output to PDF format. I suggest using `http://overleaf.com` or BaKoMa TeXto create your document.

Your programs must be written in Python. The relevant code to the problem should be in the PDF you turn in. If a problem involves programming, then the code should be shown as part of the solution to that problem. One easy way to do this in LaTeXis to use the verbatim environment, i.e., \begin{verbatim} YOUR CODE \end{verbatim}

If you have forgotten your linear algebra, you may find the Matrix Cookbook useful, which can be readily found online. I like to do math using a program called MathType, which can easily export equations to AMS LaTeXso that you don't have to actually write the equations in LaTeXdirectly: `http://www.dessci.com/en/products/mathtype/`

If told to implement an algorithm, don't just call a toolbox.

Problem 1 (2 points)

Let $\mathbf{X}$ be a 4×2 matrix, $\mathbf{Y}$ be a 2×2 matrix, and $\mathbf{Z}$ be a matrix such that $\mathbf{X} = \mathbf{ZY}$. What is the size of $\mathbf{Z}$?

Solution:
X's rows and columns equal (4,2) which is the result of a multiplication of Y (2,2) and Z (?,?) The size of matrix Z, denoted as $Z(m \times p)$, is determined by the dimensions of the matrices being multiplied, $X(m \times n)$ and $Y(n \times p)$. In this case, X is a 4×2 matrix, and Y is a 2×2 matrix. To find the size of Z, we multiply X and Y as follows:

$$Z = XY$$

$$Z = (4 \times 2) \times (2 \times 2)$$

Using the matrix multiplication rule, the resulting matrix Z will be a 4×2 matrix.

So, the size of matrix Z is 4×2.

Problem 2 (4 points)

Let
$$\mathbf{A} = \begin{pmatrix} 5 & -1 \\ -1 & 5 \end{pmatrix}.$$

Compute a formula for $\mathbf{A}^k$, i.e., the matrix power operation defined as the matrix product of k copies of $\mathbf{A}$. **Your answer must be a single matrix.** Show the necessary steps of your derivation.

Solution:
To find the formula for A^k, recall that $A^k = PD^kP^{-1}$

We need to find eigenvectors and eigenvalues using

$$\mathbf{A} = \begin{pmatrix} 5-\lambda & -1 \\ -1 & 5-\lambda \end{pmatrix}.$$

Lets take the determinant $\begin{vmatrix} 5-\lambda & -1 \\ -1 & 5-\lambda \end{vmatrix} = (5-\lambda)*(5-\lambda)-(-1)*(-1) = \lambda^2-10\lambda+24 = (\lambda-6)(\lambda-4)$

The roots are $\lambda_1 = 6, \lambda_2 = 4$

for $\lambda = 6$

$$\begin{pmatrix} 5-\lambda & -1 \\ -1 & 5-\lambda \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}$$

We now find the null space of this matrix of this matrix where we first get the row echelon form of it which is self-evident We then solve

$$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The Null space vector is

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

for $\lambda = 4$

$$\begin{pmatrix} 5-\lambda & -1 \\ -1 & 5-\lambda \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

We now find the null space of this matrix of this matrix where we first get the row echelon form of it which is self-evident We then solve

$$\begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The Null space vector is

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Armed with the eigenvectors and eigenvalues we can now construct the PDP^{-1} decomposition

$$\mathbf{P} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}.$$

$$\mathbf{D} = \begin{pmatrix} 6 & 0 \\ 0 & 4 \end{pmatrix}.$$

$$\mathbf{P^{-1}} = \begin{pmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}.$$

A is diagonalizable and thus $A^n = PD^NP^{-1}$

So,

$$\mathbf{A^k} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 6^k & 0 \\ 0 & 4^k \end{pmatrix} \cdot \begin{pmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$$

Simplifying into one matrix...

$$\mathbf{A^k} = \begin{pmatrix} \frac{2^{2k}+2^k*3^k}{2} & \frac{2^{2k}-2^k*3^k}{2} \\ \frac{2^{2k}-2^k*3^k}{2} & \frac{2^{2k}+2^k*3^k}{2} \end{pmatrix}.$$

Problem 3

Suppose $\mathbf{A}$ is an $n \times n$ matrix with eigenvalue λ and the corresponding eigenvector $\mathbf{v}$.

Part 1 (2 points)

If A is invertible, is $\mathbf{v}$ an eigenvector of $\mathbf{A}^{-1}$ If so, what will the new eigenvalue be? If not, explain why not.

Solution:
First part: yes, from the eigenvector decomposition,

$Av = \lambda v$ multiply by A^{-1} on the left we get $v = \lambda A^{-1} v$

Because λ is not zero since A is invertible.

$A^{-1}v = \frac{1}{\lambda}v$ since we know that v is nonzero vector, it is implied that v is an eigenvector with eigenvalue $\frac{1}{\lambda}$ of A

Part 2 (2 points)

Is 3$\mathbf{v}$ an eigenvector of $\mathbf{A}$? If so, what will the eigenvalue be? If not, explain why not.

Solution:
Yes, any scalar multiplication is communicable throughout $A(3v) = \lambda(3v)$

$A(3v) = 3Av = 3\lambda(v) = \lambda(3v)$

Problem 4

Part 1 (2 points)

Identify, with proof, the possible values of the determinant of an orthogonal matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$.

Solution:

Since A is orthogonal, $AA^\top = I = A^\top A$
Using $det(BC) = det(B)det(C)$
we have $det(I) = 1 = det(AA^\top) = det(A)det(A^\top) = det(A)det(A) = [det(A)]^2$

Since we have $[det(A)]^2 = 1$, then $det(A) = \pm\sqrt{1} = \pm 1$.

Part 2 (2 points)

Assume $\mathbf{A}$ is an invertible matrix. Prove that

$$\det\left(\mathbf{A}^{-1}\right) = \frac{1}{\det\left(\mathbf{A}\right)}.$$

Solution:

Using that we know that A is an invertible matrix, we know that $AA^{-1} = I$ where I is the identity matrix.
So, $det(AA^{-1}) = det(I)$.
As previously stated in Part 1 and the determinant of the identity matrix is 1,
$det(A)det(A^{-1}) = 1$
Being that the determinant is a scalar value we can rearrange the previous equation so that $det(A^{-1}) = \frac{1}{det(A)}$

Problem 5

In this problem, and several others, you will use the classic MNIST digit dataset. MNIST is made up of 70,000 28×28 images of the handwritten digits 0–9, where 60,000 of the images have been designated for use in training algorithms and 10,000 images have been designated for testing/evaluating algorithms to see how well they work on data they have never seen before (i.e., how well they generalize to new data). The image values should be converted from integers to floating point values. Using single precision will make the code run faster and use less memory.

Go to this webpage `http://yann.lecun.com/exdb/mnist/` and download these four files:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

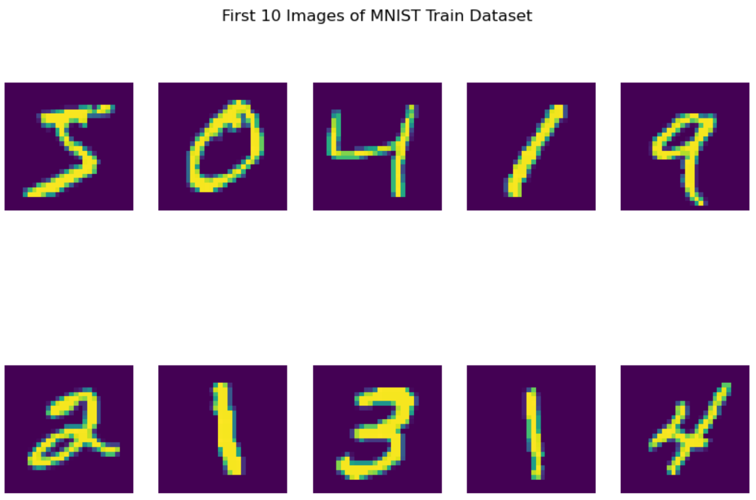
After downloading them, you likely will need to unzip them and place them in a folder.

Important: mnist_hard.mat is not used for this problem. It is used later on a different problem.

Part 1 (2 Points)

Display the first 10 data points in the training set as images. You will likely need to use the numpy.reshape function to do this by reshaping each 784-dimensional vector into a 28×28 image to display it.

Solution:



This code is the beginning of all the mnist-related code

```
import gzip
import matplotlib.pyplot as plt
import numpy as np
import random
from collections import Counter

from tqdm import tqdm
#https://stackoverflow.com/questions/40427435/extract-images-from-idx3-ubyte-file-or-gzip-v
#https://www.geeksforgeeks.org/how-to-display-multiple-images-in-one-figure-correctly-in-ma
#https://www.kaggle.com/code/walidmourou/k-nearest-neighbors-on-mnist-dataset
def get_images(location):
    with gzip.open(location, 'r') as f:
        # first 4 bytes is a magic number
        magic_number = int.from_bytes(f.read(4), 'big')
        # second 4 bytes is the number of images
        image_count = int.from_bytes(f.read(4), 'big')
        # third 4 bytes is the row count
        row_count = int.from_bytes(f.read(4), 'big')
        # fourth 4 bytes is the column count
        column_count = int.from_bytes(f.read(4), 'big')
        # rest is the image pixel data, each pixel is stored as an unsigned byte
        # pixel values are 0 to 255
        image_data = f.read()
        images = np.frombuffer(image_data, dtype=np.uint8)\
            .reshape((image_count, row_count, column_count))
        return images

def get_labels(location):
    with gzip.open(location, 'r') as f:
        # first 4 bytes is a magic number
        magic_number = int.from_bytes(f.read(4), 'big')
        # second 4 bytes is the number of labels
        label_count = int.from_bytes(f.read(4), 'big')
        # rest is the label data, each label is stored as unsigned byte
        # label values are 0 to 9
        label_data = f.read()
        labels = np.frombuffer(label_data, dtype=np.uint8)
        return labels

train_labels = "train-labels-idx1-ubyte.gz"
train_images = "train-images-idx3-ubyte.gz"
test_labels = "t10k-labels-idx1-ubyte.gz"
test_images = "t10k-images-idx3-ubyte.gz"
```

```
#Make 10 image plot

image_size = 28
num_images = 10


f = gzip.open(train_images,'r')
f.read(16)
buf = f.read(image_size * image_size * num_images)
data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)
data = data.reshape(num_images, image_size, image_size, 1)
i=0
fig = plt.figure(figsize=(10, 7))
plt.title("First 10 Images of MNIST Train Dataset")
plt.axis('off')
rows = 2
columns = 5
while i < num_images:
    fig.add_subplot(rows, columns, i+1)
    image = np.asarray(data[i]).squeeze()
    plt.axis('off')
    plt.imshow(image)
    i+=1

plt.show()
```

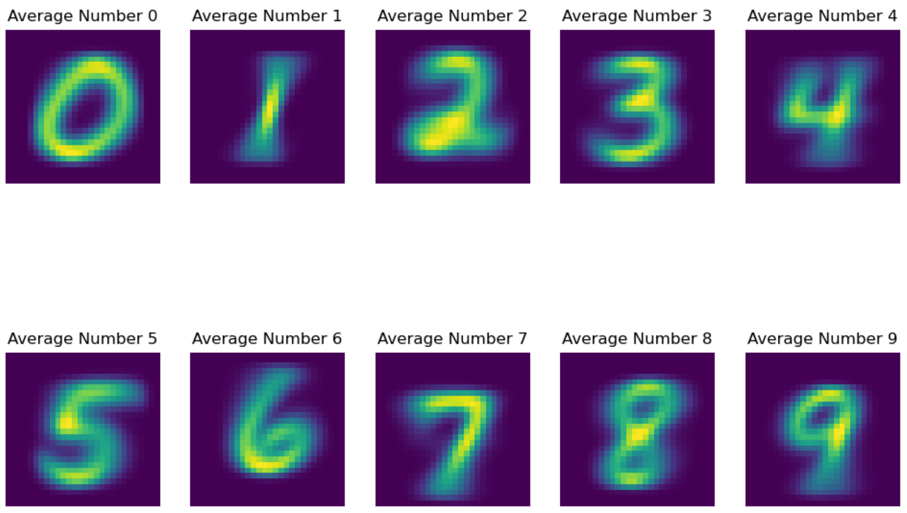
Part 2 (2 Points)

Compute the mean of each digit in the training set, i.e., compute the mean of all the zeros, all the ones, etc. Your answer should display 10 images (i.e., the average '0', the average '1', etc.). You should provide your code as part of your answer.

Solution:

```
#Make average plots
train_images = get_images(train_images)
train_labels = get_labels(train_labels)
test_images = get_images(test_images)
test_labels = get_labels(test_labels)
print(train_images[0])
print(train_images.shape)
print(train_labels)
print(train_labels.shape)

fig = plt.figure(figsize=(12, 8))
rows = 2
columns = 5
for i in range(10):
    avgImg = np.average(train_images[train_labels==i],0)
    plt.subplot(rows, columns, i+1).set_title(f'Average Number {i}')
    plt.imshow(avgImg)
    plt.axis('off')
plt.show()
```



Problem 6 (4 points)

Implement k nearest neighbors using Euclidean distance and evaluate your algorithm by "training" it on the 60,000 MNIST training images and evaluating it on the 10,000 testing images. To do this, write a function that takes as input the test features (images), training images, training labels, and k, and it should output predicted class labels for each test feature vector (or image). For debugging purposes, I suggest you only use a small portion of the training images (first 300 images) and that you use vectorization to ensure your code is fast. Note that with vectorization you may need to manage memory to ensure your machine doesn't run out of resources.

What's the percent accuracy with k equal to 1, 3, and 5 on the test images? What would the accuracy be on the training images be when $k = 1$?

Solution:
The percent accuracy I got was 95.3% for K=1, 94.7% for K=3, and 95.2% for K=5. The accuracy on the training images would be 100% as is the purpose of K-NN

```
data_size = 60000
train_images_subset = train_images[:data_size]
train_labels_subset = train_labels[:data_size]



# Combine the data to create the final dataset
idx = np.random.permutation(len(train_images_subset))
shuffled_train_images_subset,shuffled_train_labels_subset = train_images_subset[idx], train

# Set random seed for reproducibility
random.seed(42)



# Use the K-NN function to make predictions
traindata = shuffled_train_images_subset.reshape(shuffled_train_images_subset.shape[0], -1)
testdata = test_images.reshape(test_images.shape[0], -1)

def distance(x,y):
    return np.sqrt(np.sum(np.square(x-y)))

def kNN(x, k, data, label):
    distances =[distance(x,data[i]) for i in range(len(data))]
    idx = np.argpartition(distances, k)
    clas, freq = np.unique(label[idx[:k]], return_counts=True)
    return clas[np.argmax(freq)]

def accuracy_set(data, label, train_data, train_label, k):
    cnt = 0
    for x, lab in zip(data,label):
        if kNN(x,k, train_data, train_label) == lab:
            cnt += 1
    return cnt/len(label)

k_acc = [accuracy_set(testdata, test_labels, traindata, shuffled_train_labels_subset, k) fo
print(k_acc)
```

Problem 7 - Principal Component Analysis

Consider the following dataset of three points in $\mathbb{R}^2$: $\mathbf{x}_1 = (-1,-1)^T, \mathbf{x}_2 = (0,0)^T, \mathbf{x}_3 = (1,1)^T$.

Part 1 (2 points)

What is the first principal component when normalized to unit length? Write down the vector.

Solution:
$[0.70710678 - 0.70710678]$

```
    covariance_matrix = np.cov(data, ddof = 0, rowvar = False)
    print("Covariance matrix: ", covariance_matrix)

    ### Step 3: Eigendecomposition on the Covariance Matrix
    eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
    print("Eigenvalues: ", eigenvalues)
    print("Eigenvectors: ", eigenvectors)

    ### Step 4: Sort the Principal Components
    # np.argsort can only provide lowest to highest; use [::-1] to reverse the list
    order_of_importance = np.argsort(eigenvalues)[::-1]
    print("Order of Importance: ", order_of_importance)
    # utilize the sort order to sort eigenvalues and eigenvectors
    sorted_eigenvalues = eigenvalues[order_of_importance]
    sorted_eigenvectors = eigenvectors[:,order_of_importance] # sort the columns
    print("Sorted Eigenvalues: ", sorted_eigenvalues)
    print("Sorted Eigenvectors: ", sorted_eigenvectors)
```

Part 2 (2 points)

Use the vector you found in Part 1 to reduce the dimensionality of the data to one dimension. What are the new coordinates? What is the variance of the projected data?

Solution:

Reduced Data: [[-1.73205081] [0.] [1.73205081]] and the variance is 3

```
    k = 1 # select the number of principal components
reduced_data = np.matmul(data, sorted_eigenvectors[:,:k]) # transform the original data
print("Reduced Data: ", reduced_data)
```

Part 3 (2 points)

Compute the reconstruction error when the data is projected back to two-dimensional space.

Solution:

Reconstruction error is 1.8748

```
from numpy import mean
from numpy import cov
import numpy as np
from numpy.linalg import eig
from numpy import linalg as LA
#https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/
#https://medium.com/@nahmed3536/a-python-implementation-of-pca-with-numpy-1bbd3b21de2e
# define a matrix
A = np.array([[-1, -1], [0, 0], [1, 1]])
A = (A - A.mean(axis = 0)) / A.std(axis = 0)
### Step 2: Calculate the Covariance Matrix
covariance_matrix = np.cov(data, ddof = 0, rowvar = False)
print("Covariance matrix: ", covariance_matrix)

### Step 3: Eigendecomposition on the Covariance Matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
print("Eigenvalues: ", eigenvalues)
print("Eigenvectors: ", eigenvectors)

### Step 4: Sort the Principal Components
```

```
# np.argsort can only provide lowest to highest; use [::-1] to reverse the list
order_of_importance = np.argsort(eigenvalues)[::-1]
print("Order of Importance: ", order_of_importance)
# utilize the sort order to sort eigenvalues and eigenvectors
sorted_eigenvalues = eigenvalues[order_of_importance]
sorted_eigenvectors = eigenvectors[:,order_of_importance] # sort the columns
P = sorted_eigenvectors.T.dot(C.T)
print("Reconstruction erorr: ", LA.norm((A-P.T),None))
print("Sorted Eigenvalues: ", sorted_eigenvalues)
print("Sorted Eigenvectors: ", sorted_eigenvectors)
### Step 5: Calculate the Explained Variance
# use sorted_eigenvalues to ensure the explained variances correspond to the eigenvectors
explained_variance = sorted_eigenvalues / np.sum(sorted_eigenvalues)
print("Explained Variance: ", explained_variance)

### Step 6: Reduce the Data via the Principal Components
k = 1 # select the number of principal components
reduced_data = np.matmul(data, sorted_eigenvectors[:,:k]) # transform the original data
print("Reduced Data: ", reduced_data)

### Step 7: Determine the Explained Variance
total_explained_variance = sum(explained_variance[:k])
print("Total Explained Variance: ", total_explained_variance)
```