IMGS 682 Spring 2016 - Homework 1
Linear Algebra, Classifiers, and PCA

John Doe - `jdoe@rit.edu`

Due: 9:00 PM EST, February 16, 2016

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 either MATLAB or 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 $\mathbf{X} = \mathbf{Z}\mathbf{Y}$. What is the size of $\mathbf{Z}$?

Solution:
Given X is a 4x2 matrix

Y is a 2x2 matrix

And $\mathbf{X} = \mathbf{Z}\mathbf{Y}$

Then $\mathbf{Z} = \mathbf{X}\mathbf{Y}^{-1}$

From the above equation and the given conditions we can say that Z will be a 4x2 matrix.

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:
We know that, eigen decomposition of matrix $\mathbf{A} = \mathbf{Q} \times \lambda \times \mathbf{Q}^{-1}$
For finding $\mathbf{A}^2$ we can use eigen decomposition such as:

$$\mathbf{A}^2 = \mathbf{A}\times\mathbf{A}$$
$$=\mathbf{Q}\times\lambda\times\mathbf{Q}^{1}\times\mathbf{Q}\times\lambda\times\mathbf{Q}^{-1}$$
$$= \lambda^2$$

$$\mathbf{A}^k = \lambda^k$$

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:

$$\begin{aligned}\mathbf{A}^{-1}\mathbf{V} &= \mathbf{A}^{-1}\ (1/\lambda \times \lambda \times \mathbf{V})\\ &= 1/\lambda \times \mathbf{A}^{-1}(\lambda\mathbf{V})\\ &= 1/\lambda \times \mathbf{A}^{-1}(\mathbf{A}\mathbf{V})\end{aligned}$$

$$[Since\mathbf{A}\mathbf{V} = \lambda\mathbf{V}]$$

$$= 1/\lambda\mathbf{V}$$

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:
PUT SOMETHING HERE

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:
PUT SOMETHING HERE

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:
we know that det(AB) = det(A) det(B)

also we know that

$\mathbf{A}A^{-1} = I$

then,

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

which is equal to 1 hence,

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

Problem 5

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:
From the given data, we can find the covariance matrix as

$\mathbf{C} = \mathbf{X}\mathbf{X}^{\mathbf{T}}$

$$C = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \tag{1}$$

$$\tag{2}$$

Solving for eigen values $\mathbf{C}V = \mathbf{Lambda}V$

we get two eigen values 0 and 4

first principal component will be the eigen vector corresponding to highest eigen value

$$PC = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix} \tag{3}$$

$$\tag{4}$$

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:
PUT SOMETHING HERE

Part 3 (2 points)

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

Solution:
PUT SOMETHING HERE

Problem 6

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). I have provided you with some helper files to load the MNIST images into MATLAB, and similar functions can be readily written (or found online) for Python. The images should be turned into floating point images, and 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. Note that if you used the MATLAB helper files, the images will be stored as 784-dimensional vectors. You can use the 'reshape' command to reshape each one of these image vectors to be 28×28 to display it.

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

Part 1 (2 Points)

Compute the mean and standard deviation of each digit in the training set, i.e., compute the mean and standard deviation of all the zeros, all the ones, etc. Your answer should display 20 images in which the top row has the image mean for each digit (i.e., the average '0', the average '1', etc.) and the second row has their corresponding variance. You should provide your code as part of your answer (my solution was 10 lines of MATLAB).

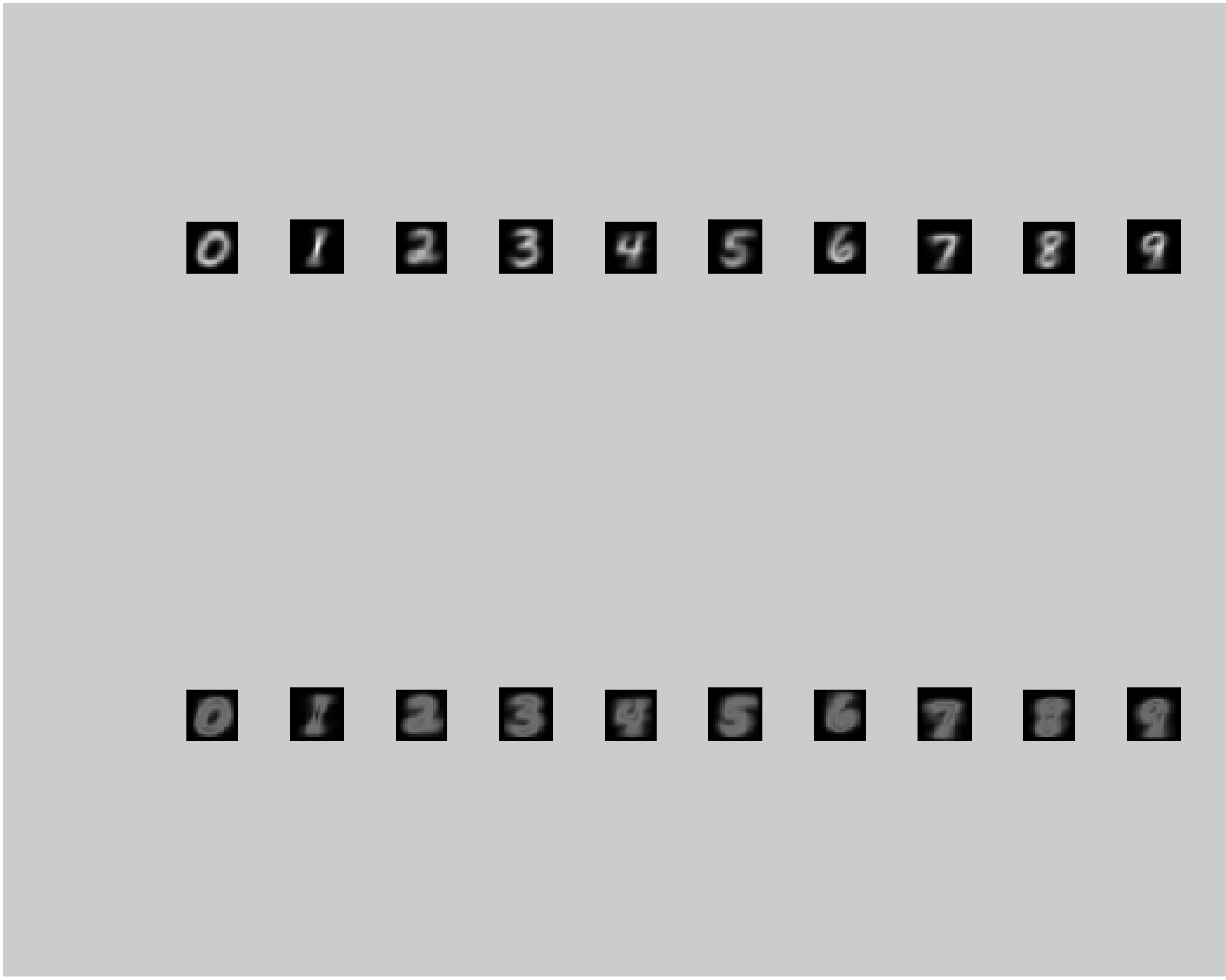
Solution:

```
clear all
close all

images = loadMNISTImages('train-images-idx3-ubyte');
labels = loadMNISTLabels('train-labels-idx1-ubyte');

imagesR = reshape(images,28,28,60000);
figure;
for i = 0:9
   M(:,:,i+1) = mean(imagesR(:,:,(labels==i)),3);
   S(:,:,i+1) = std(imagesR(:,:,(labels==i)),0,3);

   subplot(2,10,i+1); imshow(M(:,:,i+1));
   subplot(2,10,i+11); imshow(S(:,:,i+1));
end
```



Part 2 (2 Points)

Principal Component Analysis (PCA) is typically explained using an eigendecomposition of the $d \times d$ data covariance matrix $\mathbf{C}$; however, due to finite-precision arithmetic on a computer this algorithm for PCA can be numerically unstable. In practice, Singular Value Decomposition (SVD) of the data itself (instead of the covariance matrix) is typically used, i.e., $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ (see Appendix A.1 of Szeliski).

Given a data matrix $\mathbf{X}$ show mathematically how SVD can be used to compute the principal components instead of using the eigendecomposition of **C**. Assume that **X** is mean zero (i.e., centered). What would be the formula for the eigenvalues using the SVD algorithm for PCA?

Hint: This problem requires you to use the formula for PCA using the covariance matrix **C** to find a formula for PCA using SVD instead.

Solution:
Let X be NxD matrix

$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ (SVD of X)

$\mathbf{C} = \frac{\mathbf{1}}{\mathbf{N-1}} XX^T$ (dxd covariance matrix)

Now, $\mathbf{X^T X} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$

$\mathbf{X^T X} = \mathbf{V}\boldsymbol{\Sigma}\boldsymbol{\Sigma}\mathbf{V^T}$

$\mathbf{X^T X} = \mathbf{V}\boldsymbol{\Sigma}^{\mathbf{2}}\mathbf{V^T}$

$\mathbf{C} = \mathbf{V}\frac{\boldsymbol{\Sigma}^{\mathbf{2}}}{\mathbf{N-1}} V^T$

But C is symmetric, Hence $\mathbf{C} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V^T}$

Therefore, the eigen vectors of C are same as matrix V and eigen values of C can be derived from

$\lambda_{\mathbf{i}} = \frac{\sigma_{\mathbf{i}}^{\mathbf{2}}}{\mathbf{N-1}}$

Part 3 (4 Points)

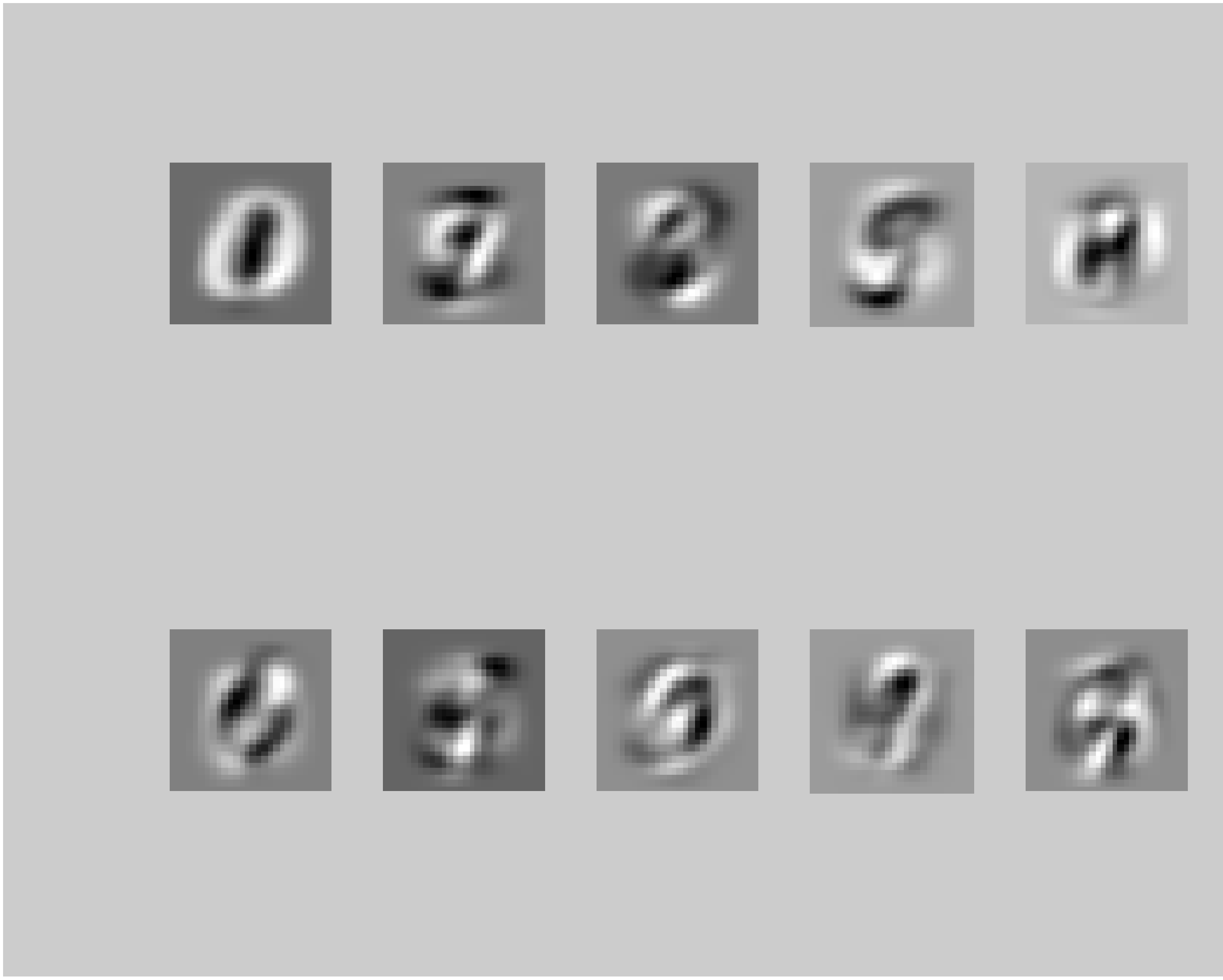
Write a function that implements PCA using SVD. Your algorithm should take a $d \times m$ matrix D and an integer k as input, where d is the dimensionality of the m data points, and k is the number of principal components to retain. Do not assume that D is mean zero. The function should return the top k principal components in a matrix and the d dimensional mean.

Apply your function to the MNIST training data to reduce the dimensionality to 10 dimensions. Display the first 10 principal components as images. When displaying the image, contrast normalize it by constraining the values to be between 0 and 1 by subtracting the minimum and then dividing by the maximum (this is called contrast stretching). (This problem took about 13 lines of MATLAB code, including code to display the answer) **Solution:**
Funtion that implements PCA using SVD

```
%PC = principle components
%dM = mean of 'd' dimension vectors
%D  = input matrix

function [PC,dM] = PCA_SVD(D,k)
mn = mean(D,2);
m = size(D,2);
D = D - repmat(mn,1,m);
Y = D'/sqrt(m-1);
[u,S,PCno] = svd(Y);
PC = PCno(:,1:k);
dM = mean(mn);
end


images = loadMNISTImages('train-images-idx3-ubyte');%loading MNIST train images
imagesR = reshape(images,28,28,60000);%reshaping them
[PC,dM] = PCA_SVD(images,10);%applying PCA_SVD
 redctn = PC'*images;%reducing the train data to 10 dimensions
imagesRnew = reshape(PC,28,28,10);%reshaping principle components to display as image
figure;
for i=1:10
    subplot(2,5,i);
    imshow(imagesRnew(:,:,i),[]);%displaying the images doing contrast stretching
end
```



Part 4 (4 points)

Plot the mean reconstruction error (squared distance between the original data and the reconstructed data) as a function of the number of principal components, i.e., from 1 to 783. To do this efficiently, you should make a new version of your function for PCA to in which k is varied. Use vectorization to ensure your code runs fast. Even if you use vectorization, expect this script to take decent amount of time to run (30-90 minutes). Make sure to label your plot's axes.

Solution:

```
im_train = loadMNISTImages('train-images-idx3-ubyte');
for i = 1:784;
PC = PCA_SVD(im_train,i);
recon = PC' * im_train;
normal = recon * PC;
diff = sum(normal-im_train)^2;
diff = diff(:)/(60000 *784);
diff1(i) = diff;
end
figure();
plot('diff');
```

Problem 7 (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. You may find the file Compute_L2_Distance_Matrix useful for this. 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:

```
function label= E_D(train_imgs,test_imgs,train_labels,k);


 M = Compute_L2_Distance_Matrix(test,train);
 [~, loc] = sort(M,2,'ascend');
 class = train_labels(loc);

  label = class(:,1:k);
 label = mode(label,2);
```

```
train_imgs = loadMNISTImages('train-images.idx3-ubyte');
test_imgs  = loadMNISTImages('t10k-images.idx3-ubyte');
train_labels = loadMNISTLabels('train-labels.idx1-ubyte');
test_labels = loadMNISTLabels('t10k-labels.idx1-ubyte');

train = train_imgs(:,1:60000);
test = test_imgs ( :,1:10000);
train_labels = train_labels(1:60000);
test_labels = test_labels(1:10000);

label= knnf(train_imgs,test_imgs,train_labels,1);%for k = 1,3,5


match = test_labels;
num = find(match == 0 );
acc = size(num,1);


acc = acc/100;
disp(acc);
```

Problem 8

One of the major problems with nearest neighbor is that it doesn't have a way to weigh the importance of each dimension of the feature vector. It also is computationally expensive because it requires comparing a vector of test features to every training data point. A computationally faster approach are linear classifiers.

For a K category classification problem, a multi-class linear algorithm will use a collection of K d-dimensional vector to classify a d dimensional vector of features (e.g., an image). Formally, this prediction would be given by $y = \arg\max_k \mathbf{w}_k^T \mathbf{x}$, where $\mathbf{w}_k$ is the weight vector for category k and $\mathbf{x}$ is the input vector of features.

Many algorithms have been devised to find the collection $\mathbf{w}_k$ vectors, and they each perform differently. We will look at some of the simplest methods.

Solution:

Part 1 (4 points)

A linear machine is a generalization of the Perceptron algorithm to multiple categories. To train a linear machine, it sequentially classifies each vector in the training data $\mathbf{x}_t$, i.e.,

$$y_t = \arg\max_k \mathbf{w}_k^T \mathbf{x}_t.$$

If the predicted category y_t is not the correct category v, then the weights are updated as follows:

$$\mathbf{w}_v \leftarrow \mathbf{w}_v + \alpha \mathbf{x}_t$$

$$\mathbf{w}_{y_t} \leftarrow \mathbf{w}_{y_t} - \alpha \mathbf{x}_t$$

where α is the learning rate. During training, the algorithm will review all of the training data multiple times (each loop through the entire dataset is called an epoch).

Implement a linear machine and compute the accuracy on the training data *and* the test data. Use a learning rate of $\alpha = 1$ and use 2 epochs, and initialize all of your weight vectors, i.e., all 10 of the $\mathbf{w}_k$ vectors, to be all zeros. How do the results on the train and test data compare to the nearest neighbor results from the previous problem? If there is a significant difference, explain why this might be the case.

Solution:

```
train_imgs = loadMNISTImages('train-images.idx3-ubyte');
test_imgs  = loadMNISTImages('t10k-images.idx3-ubyte');
train_labels = loadMNISTLabels('train-labels.idx1-ubyte');
test_labels = loadMNISTLabels('t10k-labels.idx1-ubyte');

w = zeros(784,10);
alpha = 1;

for epoch = 1:2
    for i = 1:60000
        im = train_imgs(:,i);
        curve = w' * im;
        max_pt = max(curve);
        loc = find(curve == max_pt,1);
        actual = train_labels(i)+1;

        if loc == actual
            % yay
        else
            w(:,loc) = w(:,loc) - alpha*im;
            w(:,actual) = w(:,actual) + alpha*im;
        end
    end
end

loc1 = zeros(10000,1);

for i = 1:10000
```

```
    im = test_imgs(:,i);
    curve = w' * im;
    max_pt = max(curve);
    loc = find(curve == max_pt,1);
    loc1(i) = loc-1;

end


match = test_labels - loc1;
num = find(match == 0 );
acc = size(num,1);


acc = acc/100;
disp(acc);
```

Part 2 (4 points)

While linear machines are easy to program and understand, they are not regularized and cannot output probabilities. One of the best regularized linear classifiers is the Support Vector Machine (with a linear kernel), and one of the best methods for generating probabilities is logistic regression.

Download LIBLINEAR (`https://www.csie.ntu.edu.tw/~cjlin/liblinear/`) and train a linear SVM model and a logistic regression model and compute the test accuracy for each of the two models on the MNIST test data. Use options '-s 0' and '-s 1' Try tuning the cost parameter.

Solution:

```
images = loadMNISTImages('train-images-idx3-ubyte');
labels = loadMNISTlabels('train-labels-idx3-ubyte');

ImTrain = loadMNISTImages('train-images-idx3-ubyte');
LbTrain = loadMNISTLabels('train-labels-idx1-ubyte');
ImTrainSparse=sparse(ImTrain');

model=train(MNISTLbTrain,MNISTImTrainSparse,'-s 1 -c 0.0125');

testimages = loadMNISTImages('t10k-images-idx3-ubyte');
testlabels = loadMNISTlabels('t10k-labels-idx1-ubyte');


ImTest = loadMNISTImages(filenameTestImages);
```

```
LbTest = loadMNISTLabels(filenameTestLabels);
MNISTImTestSparse=sparse(MNISTImTest');

[l,a,prob_est]=predict(MNISTLbTest, MNISTImTestSparse, model);
%accuracy 91,53; accu
```

Problem 9 (2 points)

Discuss the limitations of a linear classifier. What are some ways in which these limitations could be overcome?

Solution:
PUT SOMETHING HERE

Problem 10 (2 points)

Discuss the limitations of the naive nearest neighbor algorithm. What are some ways in which these limitations could be overcome?

Solution:
PUT SOMETHING HERE

Problem 11

Download the file mnist_hard.mat. Note that you can load MATLAB data files (*.mat files) into Python using the function loadmat in SciPy. It should be clear from the variables in the file which are the appropriate data and labels.

Solution:

Part 1 (2 points)

Train 1-nearest neighbor and report the accuracy on the test data.

Solution:
PUT SOMETHING HERE

Part 2 (2 points)

Apply PCA to the training data to reduce the dimensionality to 600, and then re-train the 1-nearest neighbor. The PCA transformation should be found using the training data, and then that transformation should be applied to both the train and test data. Report the accuracy on the test data. If PCA helped, explain why you think it might have done so.

Solution:
PUT SOMETHING HERE

Part 3 (2 points)

Why did accuracy differ significantly from when MNIST was used earlier? List two ways in which you might be able to increase accuracy and explain why you think they will help.

Solution:
PUT SOMETHING HERE

Part 4 (4 points)

Implement one of the ideas you came up with in the previous section and evaluate it on the test data. How did it affect the accuracy on the test data? Make sure to provide whatever code you wrote (which you should have been doing for all of the programming questions anyway).

Solution:
PUT SOMETHING HERE