Homework IV

STAT/CS 5810/6655 - Spring semester 2024

Due: Friday, March 15, 2024 - 5:00 PM

Please upload your solutions in a single pdf file in Canvas. Any requested plots should be sufficiently labeled for full points. Include any code requested.

Unless otherwise stated, programming assignments should use built-in functions in your chosen programming language (Python, R, or Matlab). However, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

1. **Deep Learning Paper (10 pts).** Read the the paper "Deep Learning" by LeCunn et al. You can find it on Canvas. Write a short (1-2 paragraph) summary of the paper. Was there anything you learned in it that you didn't previously know? Was there anything you disagree with?

2. **Conceptual questions (10 pts)**. Write short answers to each of the following questions.

   (a) What is the difference between PCA and LDA?

   (b) Multicollinearity in data occurs when two or more features are highly correlated, which can cause problems in the regression setting. Is multicollinearity an issue in PCA?

   (c) Discuss the limitations of PCA and how they can be overcome.

   (d) Suppose you wanted to use a trained Gaussian Mixture Model to generate new data points. How would you do that?

3. **PCA (5810 - 30 pts).** In this exercise you will apply PCA to a modified version of the Extended Yale Face Database B. The modified database is available in the file `Data/HW4/yalefaces.mat` on Canvas. The modification was simply to reduce the resolution of each image by a factor of $4 \times 4 = 16$ to hopefully avoid computational and memory bottlenecks.
   Plot images of a few of the samples. The data consist of several different subjects (38 total) under a variety of lighting conditions.

   (a) (15 pts) By viewing each image as a vector in a high dimensional space, perform PCA on the full dataset using the equations derived in class. Do not use a built-in method that performs PCA

automatically. Hand in a plot of the sorted eigenvalues (use the `semilogy` command in Matlab; `plt.semilogy` in Python) of the sample covariance matrix. How many principal components are needed to represent 95% of the total variation? 99%? What is the percentage reduction in dimension in each case? Useful commands in Matlab: `reshape, eig, svd, mean, diag`, and Python: `np.reshape, np.linalg.eig, np.linalg.svd, np.mean, np.diag`.

(b) (15 pts) Include a 4×5 array of subplots showing principal eigenvectors ('eigenfaces') 0 through 19 as images, treating the sample mean as the zeroth order principal eigenvector. Comment on what facial or lighting variations some of the different principal components are capturing. Useful commands in Matlab: `subplot, imagesc, colormap(gray)`, in Python: `plt.imshow(x, cmap=plt.get cmap('gray'))`

(c) (5 pts) What happens if you apply PCA to an isotropic Gaussian distribution, i.e. the covariance matrix is given by $\Sigma = \sigma I$.

4. **PCA for preprocessing (5810 - 15 pts)** Download the Glass Identification dataset located at: `https://archive.ics.uci.edu/dataset/42/glass+identification`.
In this problem, we'll see if running PCA prior to doing logistic regression helps or hurts the classification performance for different dimensions on this dataset. You may use any built-in methods for this problem. Note that the prediction variable is the type of glass.

   (a) Split the data into a training and test dataset, making sure that the relative percentage of classes is close to the same for the training and test data.

   (b) Apply PCA to the training data with reduced dimensions ranging from 1 to 8. Train a logistic regression classifier for each setting. Project the test data onto the learned dimensions for each setting and apply the learned classifier to the test data. Repeat this for the full data without dimensionality reduction. Report the results in a plot with dimension along the x-axis (a dimension of 9 is the full data case). Which dimension gives the best test performance? Does doing PCA seem to help at all?

   (c) Plot the eigenvalues of the covariance matrix in decreasing order. Based on the "knee method", what dimension would you select for this data? Does that dimension match what you obtain in part b? If not, explain why that could be the case.

5. **GMM (5810 - 10 pts)** Using the same Glass Identification dataset as before, learn a Gaussian Mixture Model (GMM) with six Gaussian components in it. You may use any built-in methods to do this. After learning the GMM, assign each data point to one of the six clusters. Compare this clustering result to the class labels using the Adjusted Rand Index (ARI). You do not need to code the ARI up from scratch. You should be able to find code online. How do the GMM clusters compare to the class labels. Do they mostly align or are they different? If they are different, does that mean that the clustering has "failed"?

6. **Canonical Correlation Analysis (CCA) (25 pts)** Consider n observations described by two sets of variables $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$ (both matrices are mean centered). We are interested in finding projections $\mathbf{z} = X\mathbf{a}$ and $\mathbf{w} = Y\mathbf{b}$, such that the correlation between $\mathbf{z}$ and $\mathbf{w}$ is maximized:

$$\mathbf{a}, \mathbf{b} = \underset{\mathbf{a},\mathbf{b}}{\operatorname{argmax}} \operatorname{corr}(X\mathbf{a}, Y\mathbf{b}), \tag{1}$$

where $\mathbf{a} \in \mathbb{R}^p, \mathbf{b} \in \mathbb{R}^q$.

(a) (5pts) Explain why the previous problem can be rewritten as:

$$\begin{aligned} \max_{\mathbf{a},\mathbf{b}} \quad & \mathbf{a}^T X^T Y \mathbf{b} \\ \text{s.t.} \quad & ||\mathbf{z}||^2 = 1, ||\mathbf{w}||^2 = 1. \end{aligned}$$

(b) (8pts) Solve the previous optimization problem. **Hint: Use Lagrange multipliers λ_1 and λ_2 for the constraints. You should be able to find that $\lambda_1 = \lambda_2$.**

(c) (2pts) Briefly explain the similarities and difference between CCA and PCA.

(d) (10pts) Apply CCA to `Canvas - Files/Data/HW4/JobPerformance.csv`. You can use any built-in methods. The data contains the following 9 variables.

The dependent data set contains three variables that each are alternative measurements of Job Performance:
- ClientSat: A satisfaction rating between 1 and 100 by a person's main client
- SuperSat: A rating on Job Performance between 1 and 100 by a person's superior
- ProjCompl: The percentage of a person's projects that were successfully delivered

The independent data contains six variables, of which two variables are measurements of Social Skills:
- PsychTest1: score between 0 (bad) and 100 (good)
- PsychTest2: score between 0 (bad) and 100 (good)

The data also contain two variables on Intellectual Skills:
- YrsEdu: Number of years of higher education obtained
- IQ: Score on an IQ test

Also two variables on Motivation:
- HrsTrain: Number of hours spent on training
- HrsWrk: Average number of hours in a workweek

The goal of this analysis is determine whether the three independent concepts are also identified by the model as Canonical Variables, or whether there are more interesting Canonical Variables to be defined. So, take `'PsychTest1', 'PsychTest2', 'YrsEdu', 'IQ', 'HrsTrn', 'HrsWrk'` as X variables and `'ClientSat', 'SuperSat', 'ProjCompl'` as Y variables. Apply CCA, find

two canonical correlation components and identify which variables have potential correlations with job performance. **Hint**: In R, you can use the `cancor()` function; In Python, you can use the `CCA` function from the `sklearn.cross_decomposition` library; In MATLAB, you can use the `canoncorr` function. To identify variables that have higher correlations, you may extract the weights (namely **a** and **b**) that maximize the correlations and plot them.

7. **Kernel PCA (6655 - 30 pts)**. In our discussion about PCA we established that the principal components are given by the eigenvectors of the centered covariance matrix in the original feature space. Now, let's suppose we want to transform our data to a new feature space given by $\phi(\boldsymbol{x})$ with $\phi : \mathbb{R}^d \to \mathbb{R}^m$. Thus, we could solve the following eigenvalue problem in the new feature space:

$$C = \frac{1}{N} \sum_{n=1}^{N} \phi(\boldsymbol{x}_n)\phi(\boldsymbol{x}_n)^T \tag{2}$$

$$C\mathbf{v}_i = \lambda_i \mathbf{v}_i,\ i = 1, \ldots, m. \tag{3}$$

This setting requires us to work directly in the new feature space which we would prefer to avoid.

(a) (5pts) Find a way to transform the problem using the kernel trick. I.e. find a way that the new formulation depends only on inner products which can be replaced with a kernel function and corresponding kernel matrix K. **Hint: Combine equations (2) and (3). Realize that v_i can be written as a linear combination of $\phi(\boldsymbol{x}_n)$: $\mathbf{v}_i = \sum_{n=1}^{N} a_{in}\phi(\boldsymbol{x}_n)$.**

(b) (10pts) Find an expression for the projections to the principal components in terms of K.

(c) (5pts) Note that in the previous formulations we are assuming the data is centered in the new feature space, thus, we use (2). But in general this is not the case. We could center the data as follows:

$$\tilde{\phi}(\boldsymbol{x}_n) = \phi(\boldsymbol{x}_n) - \frac{1}{N} \sum_{n=1}^{N} \phi(\boldsymbol{x}_n)$$

But, can you find an inconvenience in doing so? Find a way to fix this issue.

(d) (15pts) Implement Kernel PCA in R, Matlab or Python. Apply your code to perform kernel PCA vs PCA for the concentric circles dataset, and the swiss roll, and find the projections to the first **two** principal components. The data is available on Canvas under `Data/HW4/circles.csv` and `Data/HW4/swr.csv`. The last column in both cases corresponds to the labels. Use these only to color your plots.

For kernel PCA use the **rbf kernel** for three different σ values. You can visualize the original datasets since they have only 2 and 3 dimensions, respectively. From these plots you might be able to estimate good values for σ (choose two that "make sense" and one that does not perform well). Include the four plots of the 2D projections for each dataset (PCA + three Kernel PCA) and interpret the results. **Turn in your code**.

8. **EM Algorithm for Mixed Linear Regression (6655 - 25 pts)**. Consider regression training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ which are iid realizations of $(\boldsymbol{X}, Y) \in \mathbb{R}^d \times \mathbb{R}$ where the conditional distribution of Y given $\boldsymbol{X}$ is modeled by the pdf

$$f(y|\boldsymbol{x};\boldsymbol{\theta}) = \sum_{k=1}^{K} \epsilon_k \phi\left(y; \boldsymbol{w}_k^T \boldsymbol{x} + b_k, \sigma_k^2\right),$$

where $\boldsymbol{\theta} = \left(\epsilon_1, \ldots, \epsilon_K, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_K, b_1, \ldots, b_K, \sigma_1^2, \ldots, \sigma_K^2\right)$ is a list of model parameters, and ϕ is a univariate Guassian density as described in the lecture notes. Derive an EM algorithm for maximizing the likelihood by following these steps.

(a) (10 pts) Denote $\underline{\mathbf{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\underline{\mathrm{y}} = (y_1, \ldots, y_n)$. First, write down the formula for the log-likelihood $\ell(\theta; \underline{\mathbf{y}}|\underline{\mathbf{x}}) = \log f(\underline{\mathrm{y}}|\underline{\boldsymbol{x}};\boldsymbol{\theta})$ where $f(\underline{\mathrm{y}}|\underline{\boldsymbol{x}};\boldsymbol{\theta})$ is the model (with parameters $\boldsymbol{\theta}$) for $\underline{\mathrm{y}}$ given $\underline{\mathbf{x}}$. Then introduce an appropriate hidden variable and write down the complete-data log-likelihood. Here and below use notation consistent with the EM algorithm for GMMs whenever possible. Finally, determine the E-step. Give an explicit formula for $Q\left(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}\right)$ in terms of $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{(j)}$, and the data.

(b) (10 pts) Determine the M-step. *Suggestions*: Use Lagrange multiplier theory to optimize the weights ϵ_k (recall that they must sum to 1). To optimize $\left(\boldsymbol{w}_k, b_k, \sigma_k^2\right)$, first hold σ_k^2 fixed and find the optimal $(\boldsymbol{w}_k, b_k)$, then plug that in and find the optimal σ_k^2. Just treat σ_k^2 as a variable (not the square of a variable).

(c) (10 pts) Now let's put these ideas into practice. Generate training data using the following code. In Matlab:

```
clear all
close all
rng(0);
n = 200; % sample size
K = 2; % number of lines
e = [.7 .3]; % mixing weights
w = [-2 1]; % slopes of lines
b = [.5 -.5]; % offsets of lines
v = [.2 .1]; % variances
for i=1:n
x(i) = rand;
if rand < e(1);
y(i) = w(1)*x(i) + b(1) + randn*sqrt(v(1));
else
y(i) = w(2)*x(i) + b(2) + randn*sqrt(v(2));
end
end
plot(x,y,'bo')
```

```
hold on
t=0:0.01:1;
plot(t,w(1)*t+b(1),'k')
plot(t,w(2)*t+b(2),'k')
```

In Python:

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
n = 200 #sample size
K = 2 #number of lines
e = np.array([0.7,0.3]) #mixing weights
w = np.array([-2,1]) #slopes of lines
b = np.array([0.5,-0.5]) #offsets of lines
v = np.array([0.2,0.1]) #variances
x = np.zeros([n])
y = np.zeros([n])
for i in range(0,n):
x[i] = np.random.rand(1)
if np.random.rand(1) < e[0]:
y[i] = w[0]*x[i] + b[0] + np.random.randn(1)*np.sqrt(v[0])
else:
y[i] = w[1]*x[i] + b[1] + np.random.randn(1)*np.sqrt(v[1])
plt.plot(x,y,'bo')
t = np.linspace(0, 1, num=100)
plt.plot(t,w[0]*t+b[0],'k')
plt.plot(t,w[1]*t+b[1],'k')
plt.show()
```

In R:

```
#Libraries library(ggplot2) library(dplyr)
#Set Seed set.seed(0)
#Initialize Parameters
n = 200 #Sample Size
K = 2   # Number of Lines
e = c(0.7,0.3) #mixing weights
w = c(-2,1) #Slopes of the Line
b = c(0.5, -0.5) #Offset of the lines
v = c(0.2, 0.1) #Variances
x = runif(n=200, min = 0,max = 1)
```

```
y = rep(0,n)
for (i in 1:n){
if(runif(n=1, min = 0,max = 1) < e[1]){
        y[i] = w[1]*x[i] + b[1] + runif(n=1, min = 0,max = 1)*sqrt(v[1])
  }
  else{
          y[i] = w[2]*x[i] + b[2] + runif(n=1, min = 0,max = 1)*sqrt(v[2])
       }
}
#Create a Datframe with x, y
data = data.frame(cbind(x,y))
#Data frame to draw the lines
t = seq(0, 1, by=0.01)
l1 = t*w[1] + b[1]
l2 = t*w[2] + b[2]
plt_set = data.frame(cbind(t,l1,l2))
#Visualize the Plot
data %>% ggplot(aes(x=x,y=y)) + geom_point(col="#f0dc82") +
   geom_line(color='red',data = plt_set, aes(x=t, y=l1)) +
    geom_line(color='blue',data = plt_set, aes(x=t, y=l2))
```

Implement and run the EM algorithm and report or turn in the following:

- The number of iterations to reach convergence
- A plot of the log-likelihood as a function of iteration number
- The estimated model parameters
- A plot showing the data, true lines (solid), and estimated lines (dotted) together.
- Your code.

Initialize your variables as follows:

- $\epsilon_k = 0.5$ for each k
- The w's to 1 and -1
- The b's to 0
- The variances to the sample variance of the y data

Other comments:

- Stop iterating when the increase in log-likelihood is less than 10^{-4}
- You can use `normpdf` in Matlab or other similar functions in other languages.