# CS 231A Computer Vision (Autumn 2012)
# Problem Set 0

**Grading Set**

Due: Oct. 2, 2012 (2:15 pm)

## 1 Monty Hall (20 points)

In Computer Vision, probability is a widely used mathematical tool to solve problems ranging from low-level to high-level vision. This exercise will serve as a reminder of some basic probability concepts you find useful later in the course.

Consider the following problem: Gold is placed behind one of three curtains. A contestant chooses one of the curtains. Monty Hall, the game host, opens an unselected empty curtain. The contestant can choose either to switch his selection to the third curtain or not. In this problem you will use basic probability to maximize the contestant's probability of winning the gold.

(a) List the set of all possible outcomes i.e. the sample space for the Monty Hall problem. Let the three curtains be labeled $A$, $B$, $C$. An outcome consists of the curtain with gold, $x$, the curtain chosen by the contestant, $y$, and the curtain chosen by Monty, $z$, denoted as $(x, y, z)$. For example, the outcome $(A, B, C)$ means that the curtain with the gold is $A$, the curtain chosen by the contestant is $B$, and the curtain chosen by Monty is $C$.

(b) Assume that placement of the gold behind the three curtains is random, that the contestant's choice of curtains is random and independent of the gold placement, and that Monty Hall's choice of an empty curtain is equally likely between the two alternatives (if the two unselected curtains are empty). Specify the probabilities of each outcome for this random experiment and use it to compute the probability of winning the gold if the contestant decides to switch. For instance, consider computing $P(A, B, C)$. This can be rewritten as $P(A, B, C) = P(C \mid A, B)P(A, B) = P(C \mid A, B)P(A)P(B)$. The first equality holds from the chain rule of probability, and the second equality from the fact that the choice of position of the gold is independent of the choice of the contestant.

(c) Given your answer to part b, should the contestant switch their choice? That is, compute $P(W \mid S = 1)$ (the probability of winning if the contestant switches) and $P(W \mid S = 0)$ (the probability of winning if the contestant doesn't switch), which tell you whether the contestant should switch curtains or not.

**Solution:**

(a) The sample space consists of triplets of the form (curtain with gold behind it, curtain chosen by player, curtain that Monty opens). If we denote the curtains by $A$,$B$, and $C$ then the sample space, denoted by $\Omega$ has 12 outcomes:

$$\begin{aligned}\Omega = \{&(A,A,B),(A,A,C),(A,B,C),(A,C,B),(B,B,A),(B,B,C),\\ &(B,A,C),(B,C,A),(C,C,A),(C,C,B),(C,B,A),(C,A,B)\}\end{aligned}$$

(b) For a discrete sample space, the probability measure is completely specified by the probabilities of the single outcome events. For this problem we can calculate the probabilities as follows:

$$\begin{aligned}P(A,A,B) &= P(\text{gold is behind } A, \text{ player's first choice is } A)\cdot\\ &\quad P(\text{Monty opens } B \mid \text{gold is behind } A, \text{ player's first choice is } A)\\ &= \tfrac{1}{2}\cdot P(\text{gold is behind } A)\cdot P(\text{player's first choice is } A)\\ &= \tfrac{1}{2}\cdot\tfrac{1}{3}\cdot\tfrac{1}{3} = \tfrac{1}{18}\end{aligned}$$

Note that Monty Hall's choice of curtain to open is not independent of where the gold is placed and what the player's initial choice was. That is why we used conditional probability in the above derivation. Using the same approach we find

$$P(A,A,C) = P(B,B,A) = P(B,B,C) = P(C,C,A) = P(C,C,B) = \tfrac{1}{18}$$

This covers all the cases where the gold placement and the initial choice of contestant coincide. For the other cases,

$$\begin{aligned}P(A,B,C) &= P(\text{gold is behind } A, \text{ player's first choice is } B)\cdot\\ &\quad P(\text{Monty opens } C \mid \text{gold is behind } A, \text{ player's first choice is } B)\\ &= 1\cdot P(\text{gold is behind } A)\cdot P(\text{player's first choice is } B)\\ &= 1\cdot\tfrac{1}{3}\cdot\tfrac{1}{3} = \tfrac{1}{9}\end{aligned}$$

Using the same argument,

$$P(A,C,B) = P(B,A,C) = P(B,C,A) = P(C,A,B) = P(C,B,A) = \tfrac{1}{9}$$

(c) We now have a complete description of the probability space for this random experiment. To find the probability of winning if the player switches, we need to find the subset $W$ of the sample space corresponding to this event, which is

$$W = \{(A,B,C),(A,C,B),(B,A,C),(B,C,A),(C,A,B),(C,B,A)\}$$

The probability of W is the sum of the probabilities of its members, and therefore the probability of winning is

$$P(W) = 6\cdot\tfrac{1}{9} = \tfrac{2}{3}$$

The probability of winning if the contestant switches is $P(W \mid S=1) = \frac{2}{3}$, the probability of winning if the contestant switches is $P(W \mid S=0) = \frac{1}{3}$. Thus to maximize to the probability of the contestant winning, the contestant should switch their choice.

**Grading Guideline:**

Total (20 points)
(a) (4 points)
(2 points) half or more of sample space correct
(2 points) entire sample space correct
(b) (8 points)
(2 points) correctly defining all 1/18 probabilities
(2 points) correctly defining all 1/9 probabilities
(2 points) correctly computing 1/18 probability
(2 points) correctly computing 1/9 probability
(c ) (8 points)
(3 points) correctly computing $P(W \mid S = 1)$ probability
(3 points) correctly computing $P(W \mid S = 0)$ probability
(2 points) correctly concluding that contestant should switch

## 2 Support Vector Machines (20 points)

Machine learning is an important tool to accomplish many tasks in Computer Vision, especially classification problems. One staple algorithm in binary classification is the Support Vector Machine (SVM). In this problem you will implement a SVM package and use it to separate a sample dataset.

First, lets get some insight into why we might want to use a SVM in a binary classification problem. SVMs are an example of supervised learning; that is, after training a model with known input/output pairs, the model can classify new datapoints. In the binary classification problem you have a set of $n$ data points,$\{x^{(1)}, \ldots, x^{(n)}\}$, with labels $\{y^{(1)}, \ldots, y^{(n)}\}$ where $y^{(i)} \in \{-1, 1\}$. We are looking for a function $h(x)$ to separate the data points with different labels as shown in Fig 1. The data points which lie along the dotted line in Fig 1 are called the support vectors. Specifically the SVM is the solution to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ i = 1, \ldots, n \\ & \xi_i \geq 0,\ i = 1, \ldots, n \end{aligned}$$

where $w$ and $b$ are parameters that define our classifier $h(x) = w^T x + b$. The parameter $C$ controls the relative weighting between the twin goals of making the $\|w\|^2$ small and of ensuring that most examples are correctly classified. If you want more specifics on SVMs, please read Prof. Andrew Ng's Notes at the CS 229 course webpage `http://www.stanford.edu/class/cs229/materials.html`.

In this problem you will install the SVM package `liblinear`. You will use this package through its Matlab interface. The `liblinear` package is written in C, so you will need to have a compiler linked to your Matlab install.

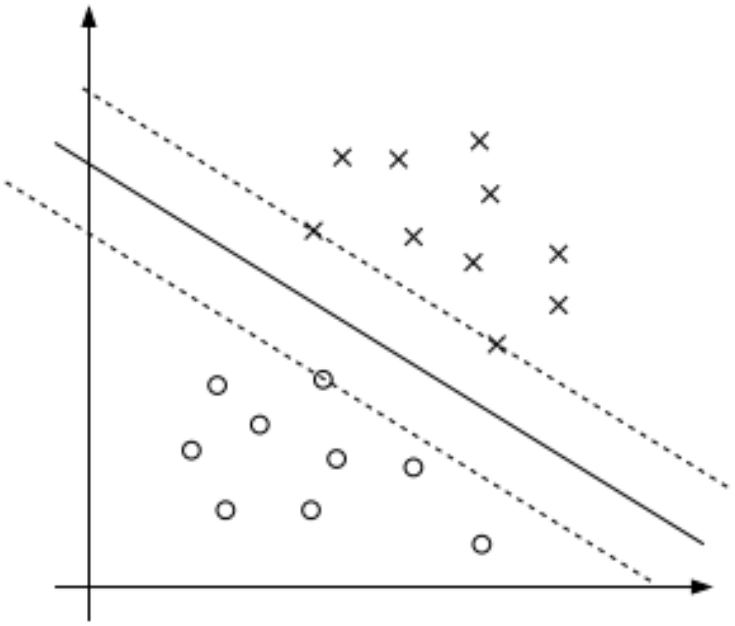

Figure 1: An example of the output of a separating $h(x)$ for a linearly separable data set.

(a) Use the `liblinear` package to test and train a SVM on an idealized dataset (similar to that shown in Fig 1). This will also act as a tutorial through installation and use of `liblinear`.

Download the current version of the `liblinear` package from `http://www.csie.ntu.edu.tw/~cjlin/liblinear/#download`. First read the `README`, then if you are using windows set your current Matlab directory to `.\liblinear-version\matlab\` and run the `make.m` script. If you are using Mac OS X or a Linux distribution you should edit Makefile such that the variable `MATLABDIR` is the installation directory of Matlab. To find the Matlab installation directory, open Matlab and use the `matlabroot` command to return the installation path. Then set `MATLABDIR` to this directory in your Makefile. Then type `make` in the terminal to build. In case you have issues making `liblinear`, we have included a compiled version of its `train` function in `PS0_data.zip`.

Download `PS-0_data.zip` from the course webpage and run the `SVM_data_parta` script which will load the data for this problem into your Matlab workspace. This data will act as a test for your installation of `liblinear`.

Once you have installed `liblinear`, write a Matlab script that trains a classifier using `data` and plots your separating $h(x)$ and the given data. The script `plotSVM` is written for you that plots the separating $h(x)$ and the given data. You will know if your answer is correct because it should be very similar to the data above. There should be no incorrect classifications because the data for this problem is linearly separable. *Hint: the input matrices need to be sparse, so Matlab's* `sparse` *command will be useful*

Turn in a plot of the classifer and the data, as well as your code.

(b) In practice, data is rarely linearly separable and so regularization is needed to make

the model more robust to outliers. In this part you will experiment with the regularization parameter $C$ to find an optimal fit. Run the `SVM_data_partb` script to get the new data set for this problem. Using `liblinear` and varying the parameter $C$ by using `C = logspace(-4,4,5)`, find a classifier that is visually similar to the classifier you found in part a.

Turn in a plot of the classifier and the data, your value of $C$, as well as your code.

**Solution:**

(a) Once you have liblinear installed this code will train the classifier and give the plot shown in Fig 2. The code that generates this is given below.

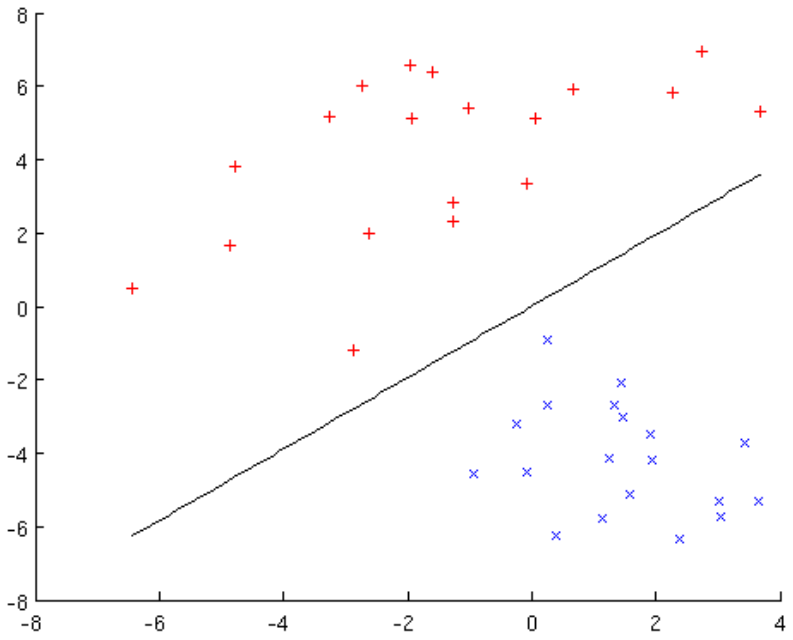


Figure 2: Classifier for data in part a

(b) The code below trains the classifier such and makes it similar to the classifier generated in part a, the classifier for part b is shown in Fig 3. The value of $C$ which achieves this is the 4th element, or `C(4)` in `C = logspace(-4,4,5)`.

```
% Problem 2 part a
close all
% load the data
SVM_data_parta

% train the model
```

```
model_parta = train(labels,sparse(data));

% plot the classifier and the data
plotSVM(data, labels, model_parta);


% Problem 2 part b
close all;
% load the data
SVM_data_partb

% train the model
C = logspace(-4, 4, 5);
for i = 1:length(C)
    model = train(labels,sparse(data),[ '-c ' num2str(C(i))]);

    % plot the classifier and the data
    plotSVM(data, labels, model);
end
```

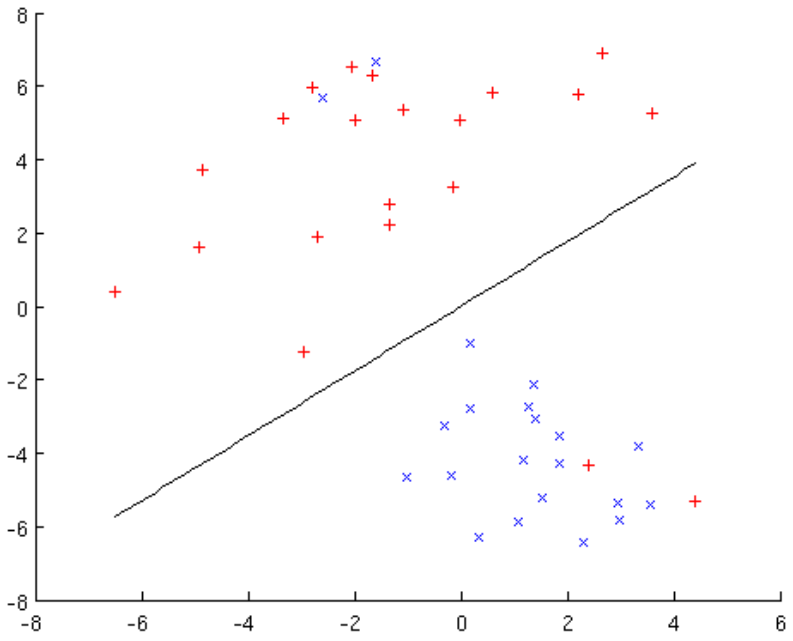


Figure 3: Classifier for data in part b

**Grading Guideline:**

Total (20 points)
(a) (8 points)
(4 points) Plot with incorrect hyperplane
(b) (12 points)
(3 points) correct C parameter chosen
(9 points) correct plot

## 3 Lagrangian derivation of least norm (20 points)

Many times in computer vision it is necessary to formulate an optimization problem to solve the vision task at hand. In this problem you will use the Lagrangian to solve an optimization problem. Consider finding an $x \in \mathbb{R}^n$ such that $Ax = b$, where $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. In this problem we will assume that a solution $x$ exists. There is the possibility that there is more than one solution. We explore how to pick a single solution when many exist. There is also the possibility that no solution exists for $Ax = b$, and we wish to find an $x$ which nearly satisfies $Ax = b$. We will not address this situation in this problem.

In the scenario where there exists multiple solutions to $Ax = b$ we can formulate an optimization problem to choose one of these solutions. Consider the following optimization problem

$$\begin{array}{ll} \text{minimize} & \|x\|_2^2 \\ \text{subject to} & Ax = b \end{array}$$

The solution to this optimization problem is the least norm solution to $Ax = b$ when there are multiple values of $x$. This may occur when e.g. $n \geq m$, which we shall assume is true for the remainder of the problem.

(a) Formulate the Lagrangian $\mathcal{L}(x, \lambda)$ of the optimization problem.

(b) Find the partial derivatives of $\mathcal{L}(x, \lambda)$ with respect to $x$ and $\lambda$, then set the derivatives equal to zero. The $x$ that satisfies these conditions is the optimal solution $x_{\text{opt}}$. Give the closed form solution for $x_{\text{opt}}$ in terms of A and b. Assume that $A$ is full rank (has rank $m$).

**Solution:**

(a) The Lagrangian for the optimization problem is given by

$$\mathcal{L}(x, \lambda) = x^T x + \lambda^T (Ax - b)$$

(b) Taking the partial derivatives with respect to $x$ and $\lambda$ and setting equal to zero yields

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x} &= 2x_{\text{opt}}^T + \lambda^T A = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= (Ax_{\text{opt}} - b)^T = 0 \end{aligned}$$

These are the optimality conditions. From the first condition $x_{opt} = -\frac{1}{2}A^T\lambda$. Substituting into the second condition we have $AA^T\lambda = -2b$. Solving for lambda we have

$$\lambda = -2(AA^T)^{-1}b$$

Note that we can be sure this inverse exists as $A$ is full rank, implying that $A^TA$ is also full rank and hence invertible. Substituting back into the expression for $x_{\text{opt}}$ we get a close form solution for $x_{\text{opt}}$

$$x_{\text{opt}} = A^T(AA^T)^{-1}b$$

**Grading Guideline:**

Total (20 points)
(a) (5 points)
(0 points) incorrect Lagrangian setup
(b) (15 points)
(5 points) correct partial derivative x
(5 points) correct partial derivative for lambda
(5 points) correct closed form solution for $x_{opt}$

# 4 Interacting with the SVD (20 points)

Linear algebra is essential in Computer Vision, especially in camera models and epipolar/multiview-geometry. Eigenvectors, eigenvalues and the fundamental spaces (nullspace, rowspace, etc.) are important concepts with which one has to be comfortable. One popular tool is the Singular Value Decomposition (similar to Principle Component Analysis), which is a powerful technique to understand how a matrix transforms a space. From this information, many techniques (such as compression and sensitivity analysis) can be reasonably implemented in very large dimensions. An excellent resource for this topic can be found in Prof. Andrew Ng's lecture notes at `www.stanford.edu/class/cs229/notes/cs229-notes10.pdf`.

The purpose of this problem is to familiarize yourself with properties of the Singular Value Decomposition through experiments in a low-dimensional setting. We will use the following matrix throughout this problem:

$$A = \begin{bmatrix} -2 & 11 \\ -10 & 5 \end{bmatrix}$$

(a) Find all solutions to the equation $Ax = b$ where $b = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$

(b) Calculate the U, $\Sigma$ , V matrices for the SVD of $A$. Is this unique? If not, give an alternative solution. *hint: you may find the Matlab command* `svd` *useful*

(c) Sketch the ellipsoid corresponding to the SVD of $A$. The ellipsoid will have semi-axis directions of the left singular vectors $u_i$, and with corresponding semi-axis half lengths of $\sigma_i$.

(d) Find the rank-1 approximation of $A$, denoted $\tilde{A}$. That is, set the smallest magnitude singular value to zero and use this new SVD to calculate $\tilde{A}$. Sketch the ellipsoid for the SVD of this matrix, $\tilde{A}$.

(e) Repeat part a but replace $A$ with its rank-1 approximation, $\tilde{A}$. *hint: make sure to include the solution for* $\tilde{A}x = 0$

**Solution:**

(a) Either by eyeballing it, using matlab's backslash operator ( `A  b` ), or using what you learned in Problem 3 ( `inv(A'A)*A'*b` ), we see that the answer is $x = [-.06 \quad .08]^T$ . Note that this is just a particular solution, and the full set of solutions for such a problem are expressed as

$$x = x_{particular} + null(A)$$

In this case, however, $null(A)$ is empty and therefore the solution set is just the particular solution we calculated above.

(b) In general, the SVD is not unique.

$$U = \begin{bmatrix} -0.7071 & -0.7071 \\ -0.7071 & 0.7071 \end{bmatrix} \Sigma = \begin{bmatrix} 14.1421 & 0 \\ 0 & 7.0711 \end{bmatrix} V = \begin{bmatrix} 0.6 & -0.8 \\ -0.8 & -0.6 \end{bmatrix}$$

Here is another possible arrangement:

$$U = \begin{bmatrix} 0.7071 & 0.7071 \\ 0.7071 & -0.7071 \end{bmatrix} \Sigma = \begin{bmatrix} 14.1421 & 0 \\ 0 & 7.0711 \end{bmatrix} V = \begin{bmatrix} -0.6 & 0.8 \\ 0.8 & 0.6 \end{bmatrix}$$

(c) Sketch of ellipsoid: `A = [-2 11; -10 5];`

```
%create a circle of x-y coordinates in 1 degree intervals
circ = zeros(2, 360);
deg = [ 1 :  1 :  360 ];
circ(1, :)  = cos(deg * pi / 180);
circ(2, :)  = sin(deg * pi / 180);
[U, S, V] = svd(A)

figure;
hold on;
axis equal;

%transform the unit circle by the A matrix
Acirc = A * circ;
plot(Acirc(1, :), Acirc(2, :), '.');

%The columns of the U matrix are the eigenvectos of AA'
%We illustrate this by drawing two lines from the origin, in the director
```

```
%of the eigenvectors and scaled by the corresponding singular values /
%eigenvalues of AA'
plot([0 S(1,1)*U(1,1)],[0 S(1,1)*U(1,2)],'r')
plot([0 S(2,2)*U(2,1)],[0 S(2,2)*U(2,2)],'r')
```

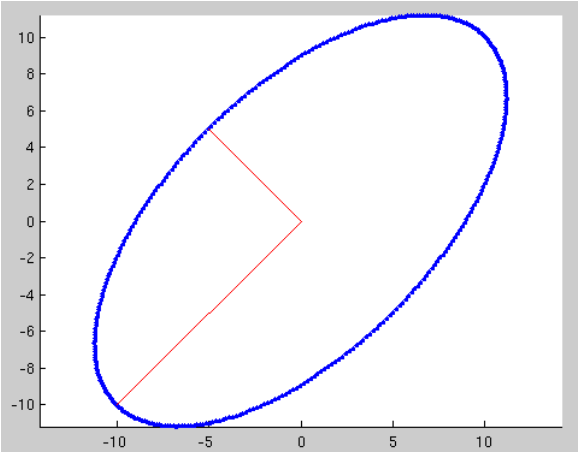


Figure 4: Output of Ellipsoid code

(d) The Rank-1 Approximation is achieved by setting the second singular value to 0. The resulting matrix is:

$$\tilde{A} = \begin{bmatrix} -6 & 8 \\ -6 & 8 \end{bmatrix}$$

This produces a degenerate ellipse: The minor axis is now zero!

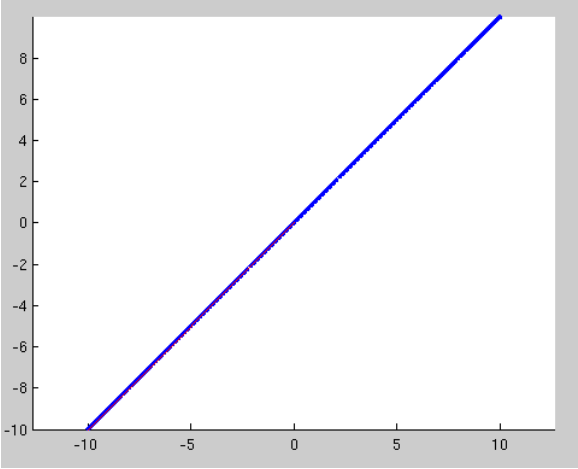


Figure 5: Output of Ellipsoid code from the Rank-1 Approximation

(e) $\tilde{A}$ now has a non-trivial nullspace. Thus the solution is

$$x = \begin{bmatrix} 7/6 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}$$

**Grading Guideline:**

Total (20 points)
(a) (4 points)
(4 points) correct solution for x
(b) (4 points)
(2 points) valid SVD solution
(2 points) stating that SVD is not unique
(c) (4 points)
(2 points) correct ellipse dimensions
(2 points) correct ellipse orientation
(d) (4 points)
(4 points) correct matrix
(e) (4 points)
(2 points) correct solution without nullspace
(2 points) correctly include nullspace

# 5 Maximum Likelihood Estimation (20 points)

There are many problems in Computer Vision that are most suitable for solving through probabilistic techniques. In fact, the foundation for many state-of-the-art high level vision algorithms (scene understanding, human pose estimation, etc.) is in probabilistic approaches. Maximum Likelihood Estimation, a popular yet straightforward method, is the process of maximizing the probability that a parameterized model would behave in a specified manner. Prof. Andrew Ng gives a good overview of MLE in the context of its similarity to Linear Regression in pages 11-13 of his notes at `www.stanford.edu/class/cs229/notes/cs229-notes1.pdf`.

In this problem, Consider that $N$ independent samples $x_i$, $1 \leq i \leq N$ are drawn from a Gaussian distribution with unknown mean and unknown variance $\sigma^2$. Recall that the formula for a Gaussian density is

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Your task is to estimate these unknown parameters given the $N$ samples by maximum likelihood.

(a) Write an expression for the probability of the sample $x_k$ given the parameters $\mu$ and $\sigma$.

(b) Now write an expression for the likelihood (the joint probability), $L$, of all $N$ samples given that all samples are identically and independently distributed.

(c) Find the values of $\mu$ and $\sigma$ that will maximize the log-likelihood, $l = \ln L$. This is equivalent to maximizing the likelihood, but applying the logarithm to part (b) turns the product into a sum, which is much easier to manipulate.

**Solution:**

(a) The expression is:

$$p(x_k; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x_k-\mu)^2/2\sigma^2}$$

(b) Because all the samples are drawn independently, we see that

$$L = p(x_1, x_2, x_3, \ldots, x_N; \mu, \sigma) = p(x_1; \mu, \sigma)p(x_2; \mu, \sigma)p(x_3; \mu, \sigma)\ldots p(x_N; \mu, \sigma)$$

(c) First we take the logarithm of the equation. Then we calculate the derivatives with respect to $\mu$ and $\sigma$.

$$\ln L = l = \ln p(x_1; \mu, \sigma) + \ln p(x_2; \mu, \sigma) + \ldots + \ln p(x_N; \mu, \sigma)$$

Looking at a single term in this sum we have

$$\ln p(x_i; \mu, \sigma) = -\frac{1}{2}\ln 2\pi\sigma^2 - \frac{1}{2\sigma^2}(x_i - \mu)^2$$

and taking the partials and setting them equal to zero we see

$$\sum_{i=1}^{N} \frac{1}{\sigma^2}(x_i - \mu) = 0$$

and

$$-\sum_{i=1}^{N} \frac{1}{\sigma^2} + \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{\sigma^4} = 0$$

Rearranging and solving we have our solution

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

And

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

**Grading Guideline:**

Total (20 points)
(a) (3 points)
(3 points) correctly writing down probability
(b) (3 points)
(3 points) correctly writing down product of probabilities
(c ) (14 points)
(2 points) changing product into sums from log
(4 points) correct partial derivative for $\mu$
(4 points) correct partial derivative for $\sigma$ or $\sigma^2$
(2 points) correct closed form solution for $\mu$
(2 points) correct closed form solution for $\sigma$ or $\sigma^2$