Machine Learning - Problem Set #3

Correction -

1 Problem 1: Casino and The Dice

/5

First of all let's compute the probability of the player winning (outcomes sum is 7) given the type of dice he is using.

$$p(\text{wins}|\text{dice=fair}) = \sum_{i=1}^{6} p(\text{roll=i})p(\text{roll=7-i}) = \frac{1}{6}$$

$$p(\text{wins}|\text{dice=unfair}) = \sum_{i\in\{3,4\}} p(\text{roll=i})p(\text{roll=7-i}) + \sum_{i\notin\{3,4\}} p(\text{roll=i})p(\text{roll=7-i}) = \frac{2}{3^2} + \frac{4}{12^2} = \frac{1}{4}$$

Now recall that a binomial distribution with parameters n, μ (which captures here the probability of the player winning in each of the n independent rolls of the dice) is

$$p(k) = \binom{n}{k}\mu^k(1-\mu)^{n-k}, \quad k \in [n]\,.$$

We want to compute the probability that the dice are unfair, given some n and k. Then we will use this expression to compute the threshold t such that $p(\text{dice=unfair}|n, \text{k=t}) \geq 0.5$, but $p(\text{dice=unfair}|n, \text{k=t-1}) < 0.5$

$$
\begin{aligned}
p(\text{dice=unfair}|n,k) &= \frac{p(k|n,\text{dice=unfair})p(\text{dice=unfair}|n)}{p(k|n)} \\
&= \frac{p(k|n,\text{dice=unfair})p(\text{dice=unfair}|n)}{p(k|n,\text{dice=unfair})p(\text{dice=unfair}|n)+p(k|n,\text{dice=fair})p(\text{dice=fair}|n)} \\
&= \frac{1}{1+\frac{p(k|n,\text{dice=fair})p(\text{dice=fair}|n)}{p(k|n,\text{dice=unfair})p(\text{dice=unfair}|n)}} \\
&= \frac{1}{1+\frac{\binom{n}{k}(\frac{1}{6})^k(\frac{5}{6})^{n-k}\frac{999}{1000}}{\binom{n}{k}(\frac{1}{4})^k(\frac{3}{4})^{n-k}\frac{1}{1000}}} = \frac{1}{1+\frac{\cancel{\binom{n}{k}}5^{n-k}6^{-n}999}{\cancel{\binom{n}{k}}3^{n-k}4^{-n}}} = \frac{1}{1+999(\frac{5}{3})^{n-k}(\frac{3}{2})^{-n}} \\
0.5 &\leq \frac{1}{1+999(\frac{5}{3})^{n-k}(\frac{3}{2})^{-n}} \\
1 &\geq 999\left(\frac{5}{3}\right)^{n-k}\left(\frac{2}{3}\right)^{n} \\
0 &\geq \log(999)+(n-k)\log\left(\frac{5}{3}\right)+n\log\left(\frac{2}{3}\right) \\
k &\geq \frac{\log(999)}{\log 5-\log 3}+\left(1+\frac{\log 2-\log 3}{\log 5-\log 3}\right)n \\
k &\geq 13.5208+0.2063n
\end{aligned}
$$

So when k, the number or player wins is bigger than $13.5208 + 0.2063n$, where n is the total number of plays, it is more likely that the player is using unfair dice.

To compute the minimum number of rounds we must observe to possibly make an inference like that, we analyze the extreme case where $k = n$, i.e. when the player wins all rounds.

$$
\begin{aligned}
n &\geq 13.5208+0.2063n \\
n &\geq \frac{13.5208}{1-0.2063} \\
n &\geq 17.0351 \\
n &\geq 18
\end{aligned}
$$

So the player must start playing and win 18 rounds in a row for the probability of the dice being unfair to be more than half. This means the casino must wait at least 18 rounds before suspecting that the dice are loaded.

2 Problem 2: Coin Toss with Prior

/10

We will solve this problem without the use of previously computed Beta distribution statistics. It is ok to identify that both the prior and the posterior distributions are instances of the Beta distribution, and use formulas for this distribution (http://en.wikipedia.org/wiki/Beta_distribution).

Uniform prior and D = {H,T}

$$p(H) = \int_0^1 p(H|\mu)p(\mu)d\mu = \int_0^1 \mu d\mu = \boxed{\frac{1}{2}}$$

$$p(\mu|D) = \frac{p(D|\mu)p(\mu)}{\int_0^1 p(D|\mu)p(\mu)d\mu} = \frac{\mu(1-\mu)}{\int_0^1 \mu(1-\mu)d\mu} = \frac{\mu(1-\mu)}{\frac{1}{2}-\frac{1}{3}} = \boxed{6\mu(1-\mu)}$$

$$\begin{aligned} \mu_{ML} &= \arg_\mu \max[\log(p(D|\mu))] = \arg_\mu \max[\log\mu + \log(1-\mu)] \\ \frac{\partial}{\partial\mu}[\log(p(D|\mu))] &= \frac{1}{\mu_{ML}} - \frac{1}{1-\mu_{ML}} = 0 \\ \mu_{ML} &= \boxed{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} \mu_{MAP} &= \arg_\mu \max[\log(p(D|\mu)) + \log(p(\mu))] = \arg_\mu \max[\log\mu - \log(1-\mu)] = \mu_{ML} \\ \mu_{MAP} &= \boxed{\frac{1}{2}} \end{aligned}$$

$$p(H|D) = \int_0^1 p(H|\mu, D)p(\mu|D)d\mu = \int_0^1 6\mu^2(1-\mu)d\mu = 6\left(\frac{1}{3}-\frac{1}{4}\right) = \boxed{\frac{1}{2}}$$

$$\begin{aligned} Var\{p(\mu|D)\} &= E\{\mu^2|D\} - (E\{\mu|D\})^2 = \int_0^1 6\mu^3(1-\mu)d\mu - \left(\int_0^1 6\mu^2(1-\mu)d\mu\right)^2 \\ Var\{p(\mu|D)\} &= 6\left(\frac{1}{4}-\frac{1}{5}\right) - 36\left(\frac{1}{3}-\frac{1}{4}\right)^2 \\ Var\{p(\mu|D)\} &= \boxed{\frac{1}{20}} \end{aligned}$$

Uniform prior and D = {T,T,T}

$$p(H) = \int_0^1 p(H|\mu)p(\mu)d\mu = \int_0^1 \mu d\mu = \boxed{\frac{1}{2}}$$

$$p(\mu|D) = \frac{p(D|\mu)p(\mu)]}{\int_0^1 p(D|\mu)p(\mu)d\mu} = \frac{(1-\mu)^3}{\int_0^1 (1-\mu)^3 d\mu} = \frac{(1-\mu)^3}{2-\frac{1}{4}-\frac{3}{2}} = \boxed{4(1-\mu)^3}$$

$$\mu_{ML} = \arg_\mu \max[\log(p(D|\mu))] = \arg_\mu \max[3\log(1-\mu)] = \boxed{0}$$

$$\mu_{MAP} = \arg_\mu \max[\log(p(D|\mu)) + \log(p(\mu))] == \arg_\mu \max[3\log(1-\mu)] = \boxed{0}$$

$$p(H|D) = \int_0^1 p(H|\mu, D)p(\mu|D)d\mu = \int_0^1 4\mu(1-\mu)^3 d\mu = 4\left(-\frac{1}{5} + \frac{3}{4} - 1 + \frac{1}{2}\right) = \boxed{\frac{1}{5}}$$

$$\begin{aligned} Var\{p(\mu|D)\} &= E\{\mu^2|D\} - (E\{\mu|D\})^2 = \int_0^1 4\mu^2(1-\mu)^3 d\mu - \left(\int_0^1 4\mu(1-\mu)^3 d\mu\right)^2 \\ Var\{p(\mu|D)\} &= 4\left(-\frac{1}{6} + \frac{3}{5} - \frac{3}{4} + \frac{1}{3}\right) - \left(\frac{1}{5}\right)^2 \\ Var\{p(\mu|D)\} &= \boxed{\frac{2}{75}} \end{aligned}$$

Beta Distribution prior and D = {H,T}
Prior: $p(\mu) = 6\mu(1-\mu) = Beta(\alpha = 2, \beta = 2)$

$$p(H) = \int_0^1 p(H|\mu)p(\mu)d\mu = \int_0^1 6\mu^2(1-\mu)d\mu = 6\left(\frac{1}{3} - \frac{1}{4}\right) = \boxed{\frac{1}{2}}$$

$$p(\mu|D) = \frac{p(D|\mu)p(\mu)]}{\int_0^1 p(D|\mu)p(\mu)d\mu} = \frac{6\mu^2(1-\mu)^2}{\int_0^1 6\mu^2(1-\mu)^2 d\mu} = \frac{\mu^2(1-\mu)^2}{\frac{1}{5} - \frac{1}{2} + \frac{1}{3}} = \boxed{30\mu^2(1-\mu)^2}$$

$$\begin{aligned} \mu_{ML} &= \arg_\mu \max[\log(p(D|\mu))] = \arg_\mu \max[\log\mu + \log(1-\mu)] \\ \frac{\partial}{\partial\mu}[\log(p(D|\mu))] &= \frac{2}{\mu_{ML}} - \frac{2}{1-\mu_{ML}} = 0 \\ \mu_{ML} &= \boxed{\frac{1}{2}} \end{aligned}$$

$$\begin{aligned} \mu_{MAP} &= \arg_\mu \max[\log(p(D|\mu)) + \log(p(\mu))] == \arg_\mu \max[2\log\mu + 2\log(1-\mu)] = \mu_{ML} \\ \mu_{MAP} &= \boxed{\frac{1}{2}} \end{aligned}$$

$$p(H|D) = \int_0^1 p(H|\mu, D)p(\mu|D)d\mu = \int_0^1 30\mu^3(1-\mu)^2 d\mu = 30\left(\frac{1}{5} - \frac{1}{2} + \frac{1}{4}\right) = \boxed{\frac{1}{2}}$$

$$\begin{aligned} Var\{p(\mu|D)\} &= E\{\mu^2|D\} - (E\{\mu|D\})^2 = \int_0^1 30\mu^4(1-\mu)^2 d\mu - \left(\int_0^1 30\mu^3(1-\mu)^2 d\mu\right)^2 \\ Var\{p(\mu|D)\} &= 30\left(\frac{1}{7} - \frac{1}{3} + \frac{1}{5}\right) - \left(\frac{1}{2}\right)^2 \\ Var\{p(\mu|D)\} &= \boxed{\frac{1}{28}} \end{aligned}$$

Beta Distribution prior and D = {T,T,T}
Prior: $p(\mu) = 6\mu(1-\mu) = Beta(\alpha = 2, \beta = 2)$

$$p(H) = \int_0^1 p(H|\mu)p(\mu)d\mu = \int_0^1 6\mu^2(1-\mu)d\mu = 6\left(\frac{1}{3} - \frac{1}{4}\right) = \boxed{\frac{1}{2}}$$

$$p(\mu|D) = \frac{p(D|\mu)p(\mu)]}{\int_0^1 p(D|\mu)p(\mu)d\mu} = \frac{6\mu(1-\mu)^4}{\int_0^1 6\mu(1-\mu)^4 d\mu} = \frac{\mu(1-\mu)^4}{\frac{1}{6} - \frac{4}{5} + \frac{3}{2} - \frac{4}{3} + \frac{1}{2}} = \boxed{30\mu(1-\mu)^4}$$

$$\begin{aligned} \mu_{ML} &= \arg_\mu \max[\log(p(D|\mu))] = \arg_\mu \max[3\log(1-\mu)] \\ \mu_{ML} &= \boxed{0} \end{aligned}$$

$$\begin{aligned} \mu_{MAP} &= \arg_\mu \max[\log(p(D|\mu)) + \log(p(\mu))] = \arg_\mu \max[\log\mu + 4\log(1-\mu)] \\ \frac{\partial}{\partial\mu}[\log(p(D|\mu)) + \log(p(\mu))] &= \frac{1}{\mu_{MAP}} - \frac{4}{1-\mu_{MAP}} \\ \mu_{MAP} &= \boxed{\frac{1}{5}} \end{aligned}$$

$$p(H|D) = \int_0^1 p(H|\mu, D)p(\mu|D)d\mu = \int_0^1 30\mu^2(1-\mu)^4 d\mu = 30\left(\frac{15}{7} - \frac{2}{3} + \frac{6}{5} - 1 + \frac{1}{3}\right) = \boxed{\frac{2}{7}}$$

$$\begin{aligned} Var\{p(\mu|D)\} &= E\{\mu^2|D\} - (E\{\mu|D\})^2 = \int_0^1 30\mu^3(1-\mu)^4 d\mu - \left(\int_0^1 30\mu^2(1-\mu)^4 d\mu\right)^2 \\ Var\{p(\mu|D)\} &= \frac{3}{28} - \left(\frac{2}{7}\right)^2 \\ Var\{p(\mu|D)\} &= \boxed{\frac{5}{196}} \end{aligned}$$

Summary of Results

Case	$p(H)$	$p(\mu\|D)$	μ_{ML}	μ_{MAP}	$p(H\|D)$	$Var\{p(\mu\|D)\}$
$\mu = Uniform[0,1]; D = \{H,T\}$	$\frac{1}{2}$	$6\mu(1-\mu)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{20}$
$\mu = Uniform[0,1]; D = \{T,T,T\}$	$\frac{1}{2}$	$4(1-\mu)^3$	0	0	$\frac{1}{5}$	$\frac{2}{75}$
$\mu = Beta(2,2); D = \{H,T\}$	$\frac{1}{2}$	$30\mu^2(1-\mu)^2$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{28}$
$\mu = Beta(2,2); D = \{T,T,T\}$	$\frac{1}{2}$	$30\mu(1-\mu)^4$	0	$\frac{1}{5}$	$\frac{2}{7}$	$\frac{5}{196}$

Reason why maximum likelihood estimation might not be good in this case
We have too few observations, what makes ML estimation too brittle and prone to extreme results. Moreover, in this case we do have some prior knowledge about how a coin should behave (even if it is not fair it should not always land on the same side) and ML estimation ignores that.

3 Problem 3: Conditional Independence

/5

1. $a \perp\!\!\!\perp b|c \rightarrow a \perp\!\!\!\perp b$ **is False** . Conditional independence does not imply unconditional independence. For a counterexample let's consider a, b, c are binary random variables, with joint probabilities:

	c=0		c=1	
	b=0	b=1	b=0	b=1
$a=0$	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{2}{15}$	$\frac{4}{15}$
$a=1$	$\frac{2}{15}$	$\frac{2}{15}$	$\frac{1}{15}$	$\frac{2}{15}$

In this case $a \perp\!\!\!\perp b|c$ because:

$$p(a|b=0,c) = p(a|b=1,c) = p(a|c)$$

But $a \not\perp\!\!\!\perp b$ because:

$$p(a=1|b=0) = \frac{1}{2} \neq p(a=1|b=1) = \frac{4}{9}$$

2. $a \perp\!\!\!\perp b \rightarrow a \perp\!\!\!\perp b|c$ **is False** . Unconditional independence does not imply conditional independence. For a counterexample let's consider a, b, c are binary random variables, with joint probabilities:

	c=0		c=1	
	b=0	b=1	b=0	b=1
$a=0$	$\frac{3}{18}$	$\frac{1}{18}$	$\frac{0}{18}$	$\frac{2}{18}$
$a=1$	$\frac{0}{18}$	$\frac{2}{18}$	$\frac{6}{18}$	$\frac{4}{18}$

In this case $a \perp\!\!\!\perp b$ because:

$$p(a=1|b=0) = p(a=1|b=1) = \frac{2}{3}$$
$$p(a=0|b=0) = p(a=0|b=1) = \frac{1}{3}$$

But $a \not\perp\!\!\!\perp b|c$ because, for example:

$$p(a=1|b=0,c=0) = 0 \neq p(a=1|b=1,c=0) = \frac{2}{3}$$

3. $(a \perp\!\!\!\perp b) \wedge (b \perp\!\!\!\perp c) \rightarrow a \perp\!\!\!\perp c$ **is False** . This statement is trivially false. Consider the case where $a = c$, for example. By definition, a is not independent of c, since they are the same variable, but they can be both independent of b.

4 Problem 4: Maximum Likelihood

/10

Let's compute the likelihood, take the derivative in respect to λ and set it to zero:

$$\begin{aligned}
p(x_1 \dots x_n|\alpha,\lambda) &= \prod_{i=1}^{n} \frac{1}{\Gamma(\alpha)} \lambda^\alpha x_i^{\alpha-1} e^{-\lambda x_i} \\
\log p(x_1 \dots x_n|\alpha,\lambda) &= \sum_{i=1}^{n} -\log(\Gamma(\alpha)) + \alpha \log \lambda + (\alpha-1)\log x_i - \lambda x_i \\
\frac{\partial}{\partial \lambda} \log p(x_1 \dots x_n|\alpha,\lambda) &= \sum_{i=1}^{n} \frac{\alpha}{\lambda} - x_i \\
0 &= \sum_{i=1}^{n} \frac{\alpha}{\lambda_{MLE}} - x_i \\
\lambda_{MLE} &= \frac{\sum_{i=1}^{n} x_i}{n\alpha}
\end{aligned}$$

Now to completely prove that this is a maximum point of likelihood (and not a minimum), we'll take the second derivative of the log-likelihood and verify that it is negative at this point.

$$\begin{aligned}
\frac{\partial^2}{(\partial\lambda)^2} \log p(x_1 \dots x_n|\alpha,\lambda) &= \sum_{i=1}^{n} -\frac{\alpha}{\lambda^2} \\
&= -\frac{n\alpha}{\lambda^2} \\
&\leq 0
\end{aligned}$$

Since both α and n are strictly positive values, and λ^2 will always be non-negative, we conclude that the second derivative of the log-likelihood is always non-positive, and therefore the λ_{MLE} we computed is the maximum likelihood estimate.

5 Problem 5: Kernel Logistic Regression

/20

5.1 Gradient Expression and Update Rule

First we will compute the gradient of the cost and the update rule:

$$\begin{aligned}
\nabla_w J(w) &= \frac{\partial}{\partial w}\left[-\sum_{i=1}^{N}\log\left(\sigma(y_i w^T k_i)\right) + \lambda w^T w\right] \\
&= -\sum_{i=1}^{N}\frac{\sigma(y_i w^T k_i)(1-\sigma(y_i w^T k_i))y_i k_i}{\sigma(y_i w^T k_i)} + 2\lambda w \\
&= -\sum_{i=1}^{N}\left[\left(1-\sigma(y_i w^T k_i)\right)y_i k_i\right] + 2\lambda w \\
w^j &:= w^{j-1} + \sum_{i=1}^{N}\left[1-\sigma(y_i w^T k_i)\right]y_i k_i - 2\lambda w
\end{aligned}$$

This is the update rule for the gradient descent. For the stochastic gradient descent the only difference is that we randomly choose p points to do the summation instead of considering all points in the training set.

5.2 Gradient Descent

The step size used was the maximum value found that made the descent smooth (without increasing the cost from time to time).

The parameters of the classifier with best performance in terms of accuracy are shown in the table below. We also show how the cost decreased through time during the training.

step size (η)	0.05
stopping criteria (ϵ)	1e-5
regularization factor (λ)	1e-3
iterations to train	29722
time to train	52s
test accuracy	**92.7%**

Table 1: Results of gradient descent.

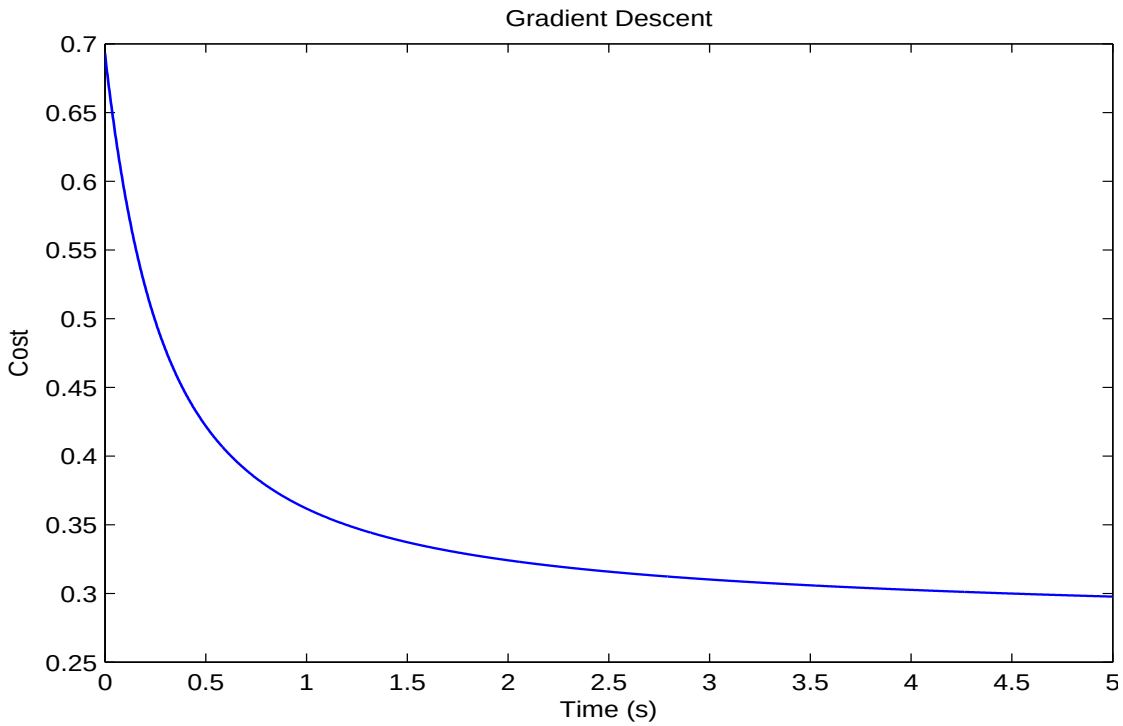

Figure 1: Evolution of the cost through time (during training).

5.3 Stochastic Gradient Descent (p=1)

For the stochastic gradient descent few changes were necessary in the algorithm:

1. The gradient in each iteration was computed with a single random observation of the training set.
2. The algorithm always kept stored the lowest cost achieved with its corresponding w. The algorithm has a time limit of 60 seconds and if no solution was found until then it would outputs the best solution seen so far.
3. The step size was reduced because otherwise the algorithm would not converge.

The parameters of the classifier with best performance in terms of accuracy are shown in the table below. We also show how the cost decreased through time during the training.

step size (η)	0.005
stopping criteria (ϵ)	1e-5
regularization factor (λ)	1e-3
iterations to train	67165
time to train	57s
test accuracy	**92.5%**

Table 2: Results of stochastic gradient descent with p=1.

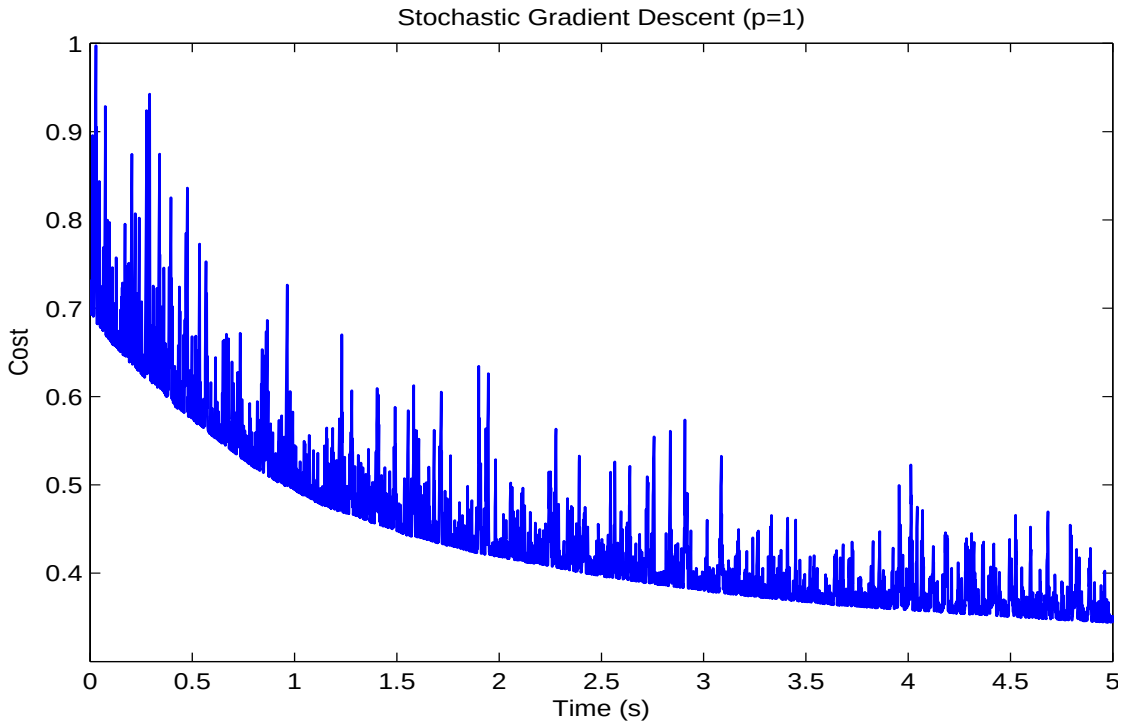

Figure 2: Evolution of the cost through time (during training).

Now with stochastic gradient descent we can see that the cost indeed goes down with time but has a lot of noise. Since we are always storing the best solution seen so far, what really matters is the lowest cost ever achieved, and we can see that the lower bound of the cost decreases consistently.

We can also conclude that the stochastic gradient descent needs more iterations to achieve the same low cost solution as gradient descent.On the other hand, it iterates much faster. It is expected that with bigger data sets the advantage of the stochastic gradient descent is greater, but in this specific case both gradient descent and stochastic gradient descent achieved practically the same accuracy with the same training time.

5.4 Stochastic Gradient Descent (p=100)

This version of the stochastic gradient descent is in the middle between the gradient descent and the stochastic gradient descent with p=1. In this method we select a batch of observations to compute the gradient in each iteration, and this batch is selected randomly from the training set (without replacement).

The parameters of the classifier with best performance in terms of accuracy are shown in the table below. We also show how the cost decreased through time during the training.

step size (η)	0.01
stopping criteria (ϵ)	1e-5
regularization factor (λ)	1e-3
iterations to train	62394
time to train	60s
test accuracy	**92.8%**

Table 3: Results of stochastic gradient descent with p=100.

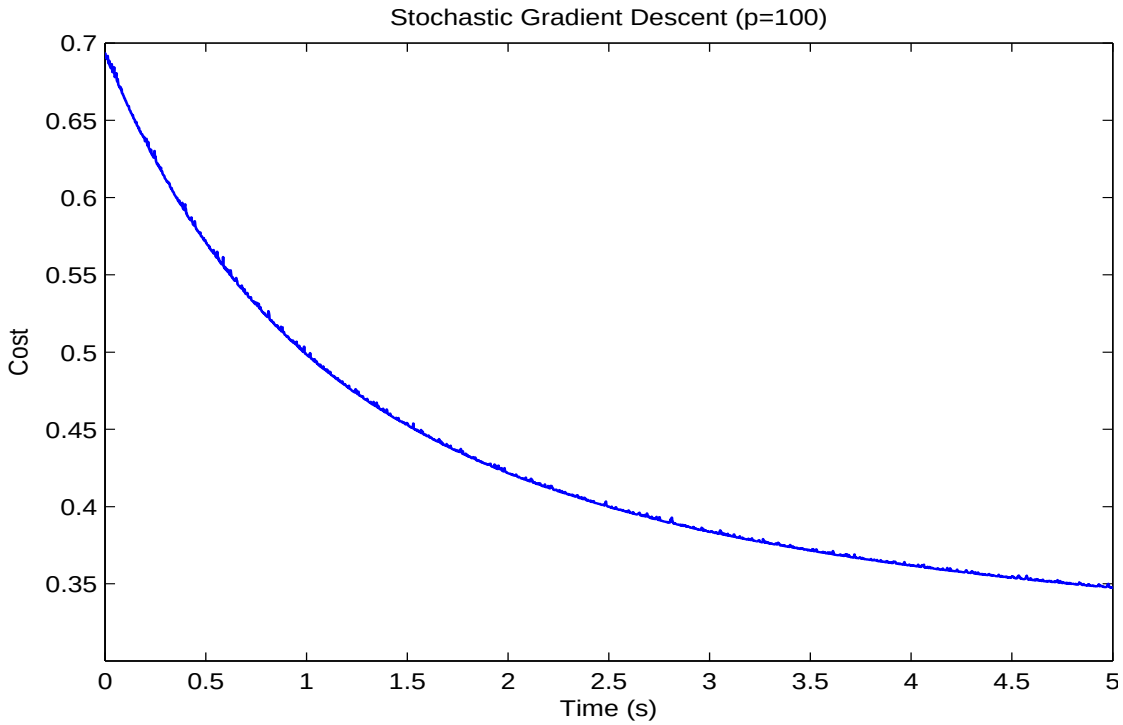

Figure 3: Evolution of the cost through time (during training).

We can see in this case that there is some noise during the descent, but not nearly as much as when p=1.

5.5 Comparison of the descent of the tree methods

To compare how the cost function decreases for the different methods let's do another run of the 3 algorithms with the same step size for all three (step size = 0.05). We measured the cost function along the time (not the iteration count) and got the following results:

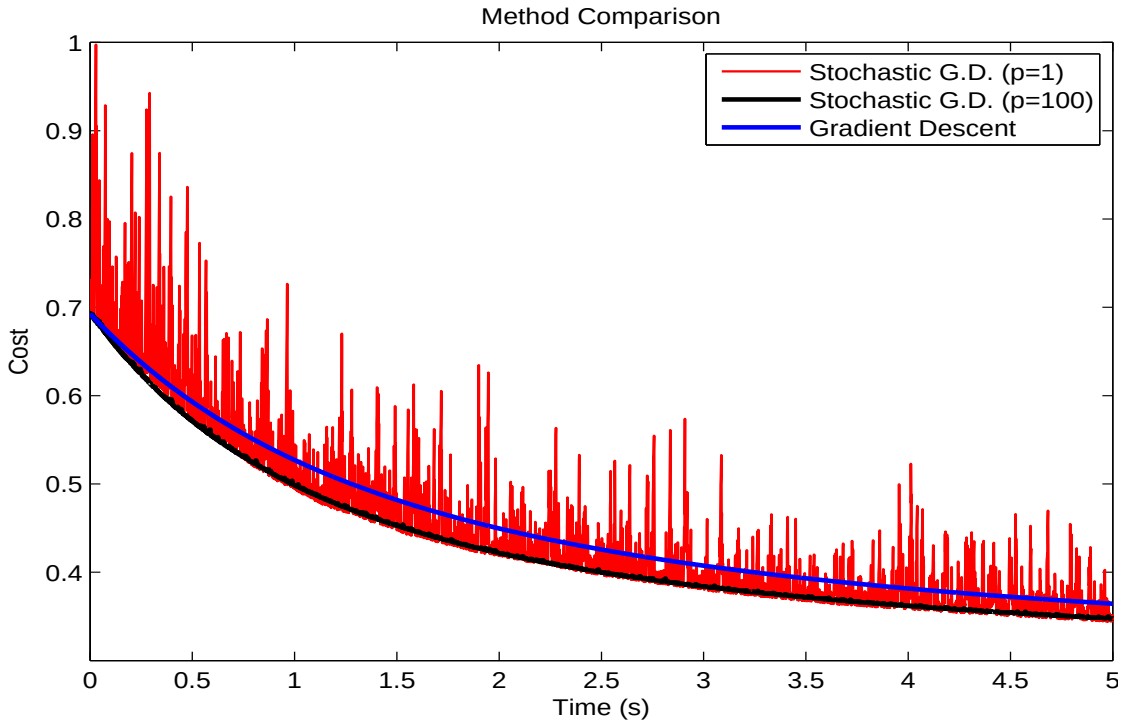

Figure 4: Evolution of the cost through time (during training).

The stochastic gradient descent with p=100 seems to be the best practical choice in this case, although for this dataset size all three methods are very close.

This is the code used to generate the reference solutions:

```
%% Pre-processing
clear
load data1.mat

N = size(TrainingX, 1);

% Compute k2 (variance of kernel)
distances_squared = zeros(N,N);
for i = 1:N
    for j = 1:N
        distances_squared(i,j) = norm(TrainingX(i,:) - TrainingX(j,:));
    end
end

k2 = sum(sum(distances_squared)) / (N^2);

% Compute the Gram Matrix
G = zeros(N,N);
for i = 1:N
    for j = 1:N
```

```
        G(i,j) = exp(-(distances_squared(i,j))/k2);
    end
end
clear distances_squared

%% Gradient Descent

time_to_record = 5;
step_size = .01;
epsilon = 1e-5;
lambda = 1e-3;
w = zeros(N,1);
cost_by_iteration = [];
i = 1;
recording_cost = true;
tic;

while(true)
    % compute the cost and gradient of the current w
    cost_now = cost(TrainingY, G, w, lambda);
    grad = gradient(TrainingY, G, w, lambda);

    if (recording_cost)
        cost_by_iteration(i) = cost_now;
    end

    if (mod(i, 1000) == 0)
        display(sprintf('Cost: %.5f', cost_now));
        display(sprintf('Grad: %.5f', norm(grad)));
    end

    if (recording_cost && toc > time_to_record)
       recording_cost = false;
    end

    % if gradient too small, stop
    if (norm(grad) < epsilon)
        break;
    end

    w = w - step_size * grad;
    i = i +1;
end

% evaluate accuracy
Ypredicted = predict(TrainingX, w, TestX, k2);
right = sum(Ypredicted == TestY);
accuracy = right / size(TestY,1);

% plot cost through time
time = linspace(0, time_to_record, size(cost_by_iteration,2));
plot(time, cost_by_iteration, 'b-', 'LineWidth', 1);
xlabel('Time (s)'); ylabel(sprintf('Cost'));
title('Gradient Descent');

%% Stochastic gradient descent p = 1

time_to_record = 5;
step_size = .005;
epsilon = 1e-5;
lambda = 1e-3;
w = zeros(N,1);
cost_by_iteration_sto = size(1,5000);
i = 1;
recording_cost = true;
```

```
tic;
best_w = w;
min_cost = 1000;

while(true)
    % compute the cost and gradient of the current w
    cost_now = cost(TrainingY, G, w, lambda);

    % get one random input
    chosen = ceil(N*rand);
    grad = gradient(TrainingY(chosen), G(chosen,:), w, lambda);

    if (recording_cost)
        cost_by_iteration_sto(i) = cost_now;
    end

    if (mod(i, 1000) == 0)
        display(sprintf('Cost: %.5f', cost_now));
        display(sprintf('Grad: %.5f', norm(grad)));
    end

    if (recording_cost && toc > time_to_record)
       recording_cost = false;
    end

    % if gradient too small, stop
    if (norm(grad) < epsilon)
        break;
    end

    w = w - step_size * grad;
    i = i +1;

    if (cost_now < min_cost)
        min_cost = cost_now;
        best_w = w;
    end
end
w = best_w;

% evaluate accuracy
Ypredicted = predict(TrainingX, w, TestX, k2);
right = sum(Ypredicted == TestY);
accuracy = right / size(TestY,1);

time = linspace(0, time_to_record, size(cost_by_iteration_sto,2));
plot(time, cost_by_iteration_sto, 'b-', 'LineWidth', 1);
xlabel('Time (s)'); ylabel(sprintf('Cost'));
title('Stochastic Gradient Descent (p=1)');

%% Stochastic gradient descent p = 100

time_to_record = 5;
step_size = .01;
epsilon = 1e-5;
lambda = 1e-3;
w = zeros(N,1);
cost_by_iteration_sto2 = size(1,5000);
i = 1;
recording_cost = true;
tic;
best_w = w;
min_cost = 1000;

while(true)
```

```
151     % compute the cost and gradient of the current w
        cost_now = cost(TrainingY, G, w, lambda);
153
        % get random inputs
155     chosen = randsample(N, 100);
        grad = gradient(TrainingY(chosen), G(chosen,:), w, lambda);
157
        if (recording_cost)
159         cost_by_iteration_sto2(i) = cost_now;
        end
161
        if (mod(i, 1000) == 0)
163         display(sprintf('Cost: %.5f', cost_now));
            display(sprintf('Grad: %.5f', norm(grad)));
165     end

167     if (recording_cost && toc > time_to_record)
           recording_cost = false;
169     end

171     % if gradient too small, stop
        if (norm(grad) < epsilon)
173         break;
        end
175
        w = w - step_size * grad;
177     i = i +1;

179     if (cost_now < min_cost)
            min_cost = cost_now;
181         best_w = w;
        end
183 end
    w = best_w;
185
    % evaluate accuracy
187 Ypredicted = predict(TrainingX, w, TestX, k2);
    right = sum(Ypredicted == TestY);
189 accuracy = right / size(TestY,1);

191 time = linspace(0, time_to_record, size(cost_by_iteration_sto2,2));
    plot(time, cost_by_iteration_sto2, 'b-', 'LineWidth', 1);
193 xlabel('Time (s)'); ylabel(sprintf('Cost'));
    title('Stochastic Gradient Descent (p=100)');
195
    %% Descent comparison
197


199 time = linspace(0, time_to_record, size(cost_by_iteration_sto,2));
    plot(time, cost_by_iteration_sto, 'r-', 'LineWidth', 1);
201
    hold all;
203 xlabel('Time (s)'); ylabel(sprintf('Cost'));
    title('Method Comparison');
205
    time = linspace(0, time_to_record, size(cost_by_iteration_sto2,2));
207 plot(time, cost_by_iteration_sto2, 'k-', 'LineWidth', 2);

209 time = linspace(0, time_to_record, size(cost_by_iteration,2));
    plot(time, cost_by_iteration, 'b-', 'LineWidth', 2);
211


213 hold off;

215 legend('Stochastic G.D. (p=1)', 'Stochastic G.D. (p=100)', 'Gradient Descent', 'Location', '
```

```
    Northeast')
```

Extra functions:

```
function [ cost ] = cost( Y, G, w, lambda )
%COST Returns the cost of the current w
    cost = lambda*(w'*w) - sum(log(sigmoid((G*w).*Y)))/size(Y,1);
end

function [ output ] = sigmoid( x )
%SIGMOID
    output = 1./(1+exp(-x));
end

function [ output ] = predict(Xtr, w, X, k2)
%PREDICT

    Xpanded = zeros(size(X,1),size(Xtr,1));
    for i = 1:size(X,1)
        for j = 1:size(Xtr,1)
            Xpanded(i,j) = exp(-(norm(X(i,:) - Xtr(j,:))^2)/k2);
        end
    end

    output = 2*(sigmoid(Xpanded*w) > 0.5) - 1;
end

function [ output ] = gradient( Y, G, w, lambda )
%GRADIENT
    output = 2*lambda*w - (G')*(Y.*(1-sigmoid((G*w).*Y)))/size(Y,1);
end
```