UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Department of Electrical and Computer Engineering

ECE 544NA Pattern Recognition

Solutions 6

Fall 2013

Assigned: Thursday, October 17, 2013 Due: Tuesday, October 29, 2013

Reading: NNPR Chapters 5, 6, 7

Problem 6.1

The Bayes rule for some loss $L(y, f(x))$ is said to be $f(x)$ that minimizes the expected loss:

$$f_B(x) = \arg\min_{f(x)} E_{X,Y}[L(y, f(x)]$$

We saw in class that when $L(y, f(x)) = |f(x) - y|^2$, $f_B(x) = E[Y|X]$, the conditional mean. We show that the optimal rule for $L(y, f(x)) = |f(x) - y|$ is the conditional *median*:

$$f_B(x) = \arg\min_{f(x)} \int P(y|x)|f(x) - y|dy$$

$$= \arg\min_{f(x)} \left[\int_{-\infty}^{f(x)} P(y|x)(f(x) - y)dy + \int_{f(x)}^{\infty} P(y|x)(y - f(x))dy \right]$$

Setting the derivative (w.r.t. $f(x)$) of this expression to 0 yields:

$$\int_{-\infty}^{f(x)} P(y|x)dy = \int_{f(x)}^{\infty} P(y|x)dy$$

Since there is exactly as much density to the left of $f(x)$ as there is to its right, $f(x)$ (by definition) is the median. Hence, $f_B(x) = median(P(Y|X = x))$.

Problem 6.2

Given K mixtures and a dataset with N examples, the E-M algorithm for a GMM computes

$$\gamma_{ij} = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_{j'} \pi_{j'} \mathcal{N}(x_i|\mu_{j'}, \Sigma_{j'})}$$

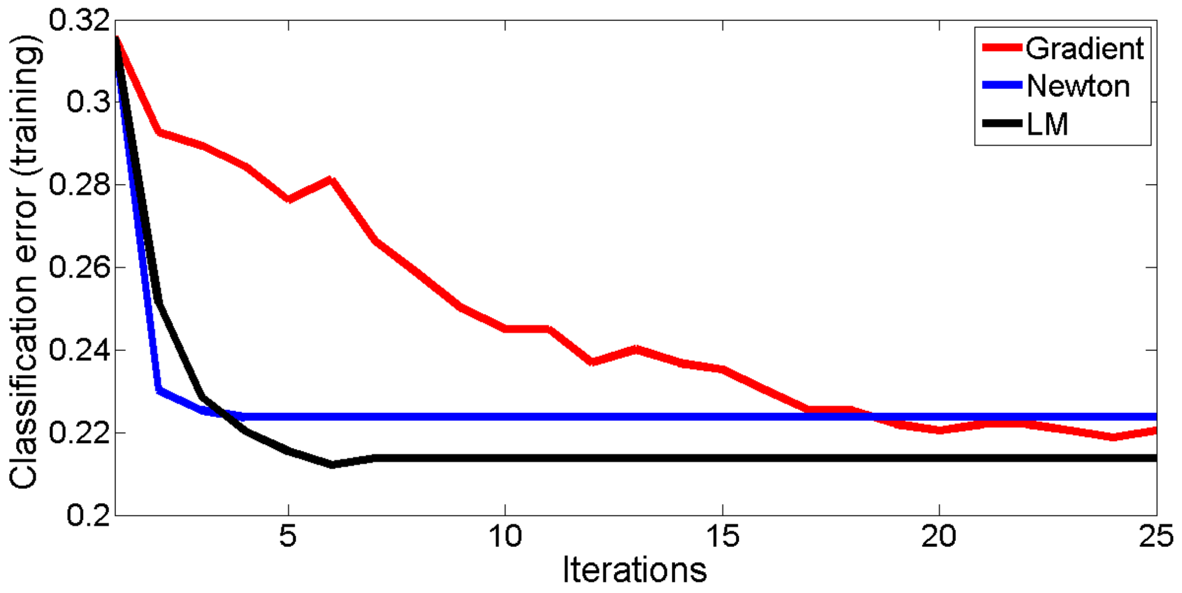
If we further assume that the covariances are diagonal, that is $\Sigma_{j'} = \epsilon I \ \forall j'$, we have

$$\gamma_{ij} = \frac{\pi_j e^{-\frac{\|x_i - \mu_j\|_2^2}{\epsilon}}}{\sum_{j'} \pi_{j'} e^{-\frac{\|x_i - \mu_{j'}\|_2^2}{\epsilon}}}$$

We are interested in the setting $\epsilon \to 0$. For convenience, let us rewrite γ_{ij} as

$$\gamma_{ij} = \frac{\pi_j}{\sum_{j'} \pi_{j'} e^{\frac{-\|x_i - \mu_{j'}\|_2^2 + \|x_i - \mu_j\|^2}{\epsilon}}}$$

Figure 6.3-1: Training set classification error as a function of the number of iterations



It is clear that if $\|x_i - \mu_j\|_2^2 > \|x_i - \mu_{j'}\|_2^2$ for any j', $e^{\frac{-\|x_i-\mu_{j'}\|_2^2+\|x_i-\mu_j\|^2}{\epsilon}} \to \infty$ and hence $\gamma_{ij} \to 0$. It is only when $\|x_i - \mu_j\|_2^2 \leq \|x_i - \mu_{j'}\|_2^2 \ \forall j'$; that is, when $j = \arg\min_{j'} \|x_i - \mu_{j'}\|_2^2$, that $\gamma_{ij} \to 1$. Hence, in the limit as $\epsilon \to 0$,

$$\gamma_{ij} = \begin{cases} 1 & \text{if } j = \arg\min_{j'} \|x_i - \mu_{j'}\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

The update equations for μ_j and γ_{ij} are therefore identical to the k-means algorithm in the limit as $\epsilon \to 0$ of a GMM in which the covariances are ϵI.

Matlab Exercises

Problem 6.3

Figure 6.3-1 reports the training set classification error as a function of the number of iterations for a random partitioning of the diabetes dataset. It is clear that both second order methods – Newton and Levenberg-Marqadt (LM) approximation – converge a lot faster than gradient descent. The biggest disadvantage with Newton's method is computing the inverse of the Hessian, which can be singular or close to singular. Many heuristics such as first finding a good initial point using gradient descent, regularization of the Hessian, etc. were used (see code). The LM approximation (coupled with the matrix inversion lemma), however, can directly estimate a non-singular and consistent inverse Hessian.

Table 1: Average classification error across 100 random trials

classifier	average error	standard deviation
gradient descent	0.24	0.03
Newton's method	0.26	0.08
LM approximation	**0.23**	**0.03**

Table 1 reports the average classification error across 100 trials. It is clear that LM approximation is at least as good as the other two methods both in terms of performance and consistency. It is computationally inefficient as implemented in this problem set; a much more natural setting is online learning/stochastic gradient descent in which we alternate between updating the weight parameters w and the inverse Hessian alternatively, one sample at a time. The Levenberg Marqadt approximation is therefore a good approach for any optimization problem wherein we want to harness the faster convergence properties of second-order methods (e.g. Newton) without their irregularities.

```
function [w, trainerror] = newtdesc(traindata, trainlabels, eta, startw, numtrials)
%given the training data and labels, newtdesc computes the optimal weight
%vector for a linear classifier y = w'x + b with a tanh(.) non-linearity
%using Newton's descent

w = startw; %initialize to something small
w = w./norm(w);
%Run a few stages of gradient descent
[w, graderr] = graddesc(traindata, trainlabels, 0.001, w, 5);
[~,numfeats] = size(traindata);
trainerror = zeros(numtrials, 1);
trainerror(1:5) = graderr;
for n = 6:numtrials
    a = tanh(traindata*w);
    y = 2.*(traindata*w >= 0) - 1; %classify training set
    trainerror(n) = sum(y ~= trainlabels)/length(trainlabels);
    %compute the gradient
    grad = 2.*traindata'*((a - trainlabels) .* (1-a).^2);
    H = traindata'*diag((1-a).^2 - 2.*(a - trainlabels).*(a .* (1-a).^2))*traindata;
    %regularize
    H = H + 0.0005.*eye(numfeats);

    w = w - eta*(H\grad);
    w = w./norm(w); %make sure w is not too large


end

function [w, trainerror] = lmdesc(traindata, trainlabels, eta, startw, numtrials)
%given the training data and labels, lmdesc computes the optimal weight
%vector for a linear classifier y = w'x + b with a tanh(.) non-linearity
%using the Levenberg Marqadt approximation

w = startw; %initialize to something small
w = w./norm(w);
%Run a few stages of gradient descent
[w, graderr] = graddesc(traindata, trainlabels, 0.001, w, 5);
[numtrain,numfeats] = size(traindata);
trainerror = zeros(numtrials, 1);
trainerror(1:5) = graderr;
for n = 6:numtrials
    a = tanh(traindata*w);
    y = 2.*(traindata*w >= 0) - 1; %classify training set
    trainerror(n) = sum(y ~= trainlabels)/length(trainlabels);

    %LM approximation
    G = bsxfun(@times, traindata, (a-trainlabels).*(1-a.^2));
    grad = sum(G, 1)';

    %Matrix inversion lemma: note that if we were to do stochastic grad.
    %descent, we would both 1) update Hinv and 2) update w for each sample
    Hinv = 20.*eye(numfeats);
    for i = 1:numtrain
        b = Hinv*G(i,:)';
        Hinv = Hinv - (b*b')/(1+b'*G(i,:)');
    end

    w = w - eta*Hinv*grad;
    w = w./norm(w); %make sure w is not too large

end
```