6.867 Machine learning and neural networks

Problem set 2

Deadline: October 12, 11am in recitation

What and how to turn in?

Turn in short written answers to the questions explicitly stated, and when requested to explain or prove. Do **not** turn in answers when requested to "think", "consider", "try" or "experiment" (except when specifically instructed).

Turn in all MATLAB code explicitly requested, or that you used to calculate requested values. It should be clear exactly what command was used to get the answer to each question.

To help the graders (including yourself...), please be neat, answer the questions briefly, and in the order they are stated. **Staple each "Problem" separately**, and be sure to write your name on the top of every page.

Problem 1: Active Learning

Reference: Lecture three; Chapter 5-5.4.1

Our goal here is to get a better understanding of the sequential active learning method described in lecture 3.

Let's start by defining our model and assumptions. We use additive regression models with fixed basis functions $\{\phi_i(\mathbf{x})\}_{i=1,\ldots,m}$ to model outputs as a function of inputs $\mathbf{x}$:

$$f(\mathbf{x};\mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + \ldots + w_m\phi_m(\mathbf{x}) \tag{1}$$

The basis functions could simply return a specific component of the input vector as in $\phi_i(\mathbf{x}) = x_i$ which would reduce the above model to linear regression. Alternatively, the basis functions could measure similarity to "prototypes" as in radial basis functions:

$$\phi_i(\mathbf{x}) = \exp\left\{ \frac{1}{2s^2} \|\mathbf{x} - \mu_i\|^2 \right\} \tag{2}$$

where μ_i specifies a location in the input space and the parameter s gives the spread around μ_i.

The observed outputs y are assumed to be related to the inputs x through some "*true*" weights $\mathbf{w}^*$, and independently corrupted with zero mean Gaussian noise. Thus, for any n input points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we model the observed outputs $\{y_1, \ldots, y_n\}$ as

$$\begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_m(\mathbf{x}_1) \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \phi_1(\mathbf{x}_n) & \cdots & \phi_m(\mathbf{x}_n) \end{bmatrix} \begin{bmatrix} w_0^* \\ \cdots \\ w_m^* \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \cdots \\ \epsilon_m \end{bmatrix} \tag{3}$$

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon \tag{4}$$

where $\epsilon \sim N(\mathbf{0}, \sigma^2 I)$. The overall noise variance σ^2 is unknown but insignificant for our purposes here.

It will be helpful to change our notation slightly. We define $\phi(\mathbf{x}) = [1, \phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^T$ as the *feature vector* (column vector) corresponding to an input point $\mathbf{x}$. We can therefore write the matrix $\mathbf{X}$ as

$$\mathbf{X} = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \cdots \\ \phi(\mathbf{x}_n)^T \end{bmatrix} \tag{5}$$

Moreover, we already know from lectures that given $\mathbf{X}$, the noise in the observed outputs cause our parameter estimates $\hat{\mathbf{w}}$ to be Gaussian random variables: $\hat{\mathbf{w}} \sim N(\mathbf{w}^*, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$. For simplicity (and with no real loss of generality), we will assume hereafter that $\sigma^2 = 1$. It will be also helpful to define the inverse covariance matrix $A = (\mathbf{X}^T\mathbf{X})$.

Now, given $\mathbf{x}$, the variance in our predictions $\hat{y}(\mathbf{x}) = f(\mathbf{x}, \hat{\mathbf{w}}) = \phi(\mathbf{x})^T\hat{\mathbf{w}}$ due to the noise in the training outputs is given by

$$\text{Var}(\, \hat{y}(\mathbf{x}) \,) = \phi(\mathbf{x})^T(\mathbf{X}^T\mathbf{X})^{-1}\phi(\mathbf{x}) = \phi(\mathbf{x})^T A^{-1}\phi(\mathbf{x}) \tag{6}$$

We are finally ready to turn to our problem...

1. Let A_n be the inverse covariance matrix based on n training examples. Show that $A_{n+1} = A_n + \phi(\mathbf{x}_{n+1})\phi(\mathbf{x}_{n+1})^T$. Is A_n symmetric for all n? Positive definite?

Answer:

$$
\begin{aligned}
A_{n+1} &= X_{n+1}^T X_{n+1} \\
X_{n+1} &= \begin{bmatrix} \phi(x_1)^T \\ \cdots \\ \phi(x_{n+1})^T \end{bmatrix} \\
A_{n+1_{i,j}} &= \sum_{k=1}^{n+1} X_{n+1_{i,k}}^T X_{n+1_{k,j}} \\
&= \sum_{k=1}^{n+1} \phi_i(x_k)\phi_j(x_k) \\
&= \sum_{k=1}^{n} \phi_i(x_k)\phi_j(x_k) + \phi_i(x_{n+1})\phi_j(x_{n+1}) \\
&= A_{n_{i,j}} + [\phi(x_{n+1})\phi(x_{n+1})^T]_{i,j} \\
A_{n+1} &= A_n + \phi(x_{n+1})\phi(x_{n+1})^T
\end{aligned}
$$

A_n is always symmetric, since $\phi(x)\phi(x)^T$ is symmetric, and a sum of symmetric matrices is always symmetric. If the feature vectors are of full rank, then the covariance matrix is invertible and A_n, being the inverse of a covariance matrix, is positive definite. However, if the feature vectors are not of full rank (e.g. if there are only a few samples), then the predictor is ill-defined, and A_n is non-invertible, and thus only positive semi-definite.

Scoring: *5 points, 3 for the proof, 1 for noting A_n is symmetric, and 1 for discussing weather it is positive-definite*

2. Find the *positive* coefficient λ such that $A_{n+1}^{-1} = A_n^{-1} - \lambda A_n^{-1}\phi(\mathbf{x}_{n+1})\phi(\mathbf{x}_{n+1})^T A_n^{-1}$. Express λ as a function of $\mathrm{Var}_n(\hat{y}(\mathbf{x}_{n+1}))$, the expected prediction error (after training on the first n samples) at $\mathbf{x}_{n+1}$. (Hint: In order to check that a matrix B is the inverse of a matrix A, check that $AB = I$).

Answer:

$$
A_{n+1}^{-1}A_{n+1} = (A_n^{-1} - \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T A_n^{-1})A_{n+1}
$$

As we have seen in the previous problem...

$$I = [A_n^{-1} - \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T A_n^{-1}][A_n + \phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T]$$

$$I = (A_n^{-1}A_n) + A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T - \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T A_n^{-1}A_n -$$
$$\lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\underbrace{\phi(\mathrm{x_{n+1}})^T A_n^{-1}\phi(\mathrm{x_{n+1}})}_{Var_n(\hat{y}(\mathrm{x}_{n+1}))}\phi(\mathrm{x_{n+1}})^T$$

$$I = I + A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T - \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T -$$
$$\lambda A_n^{-1}\phi(\mathrm{x_{n+1}})Var_n(\hat{y}(\mathrm{x_{n+1}}))\phi(\mathrm{x_{n+1}})^T$$

$$0 = A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T - \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T(I + Var_n(\hat{y}(\mathrm{x_{n+1}})))$$

$$A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T = \lambda A_n^{-1}\phi(\mathrm{x_{n+1}})\phi(\mathrm{x_{n+1}})^T(I + Var_n(\hat{y}(\mathrm{x_{n+1}})))$$

$$I = \lambda(I + Var_n(\hat{y}(\mathrm{x_{n+1}})))$$

$$\lambda = \frac{1}{(1 + Var_n(\hat{y}(\mathrm{x_{n+1}})))}$$

Since we know $Var_n(\hat{y}(\mathrm{x}_{n+1}))$ is positive, λ is a *positive coefficient*.

Scoring: *5 points*

3. Based on 1) and 2), show that the variance in our predictions at any point $\mathbf{x}$ cannot increase as a result of including *any* new point $\mathbf{x}_{n+1}$ in the training set.

Answer:

For all $\mathbf{x}$

$$\begin{aligned} Var_{n+1}(\hat{y}(\mathbf{x_n})) - Var_n(\hat{y}(\mathbf{x_n})) &= -\lambda\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})\phi(\mathbf{x_{n+1}})^T A_n^{-1}\phi(\mathbf{x}_n) \\ &= -\lambda[\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})][\phi(\mathbf{x}_n)^T (A_n^{-1})^T\phi(\mathbf{x_{n+1}})]^T \\ &= -\lambda[\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})][\phi(\mathbf{x}_n)^T (A_n^{T})^{-1}\phi(\mathbf{x_{n+1}})]^T \end{aligned}$$

Since A_n, and so also A_n^{-1} are symmetric:

$$\begin{aligned} &= -\lambda[\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})][\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})]^T \\ &= -\lambda[\phi(\mathbf{x}_n)^T A_n^{-1}\phi(\mathbf{x_{n+1}})]^2 \end{aligned}$$

$[\phi(\mathbf{x})^T A_n^{-1}\phi(\mathbf{x_{n+1}})]$ is a scalar, so its square must be positive. We know λ is also positive, so $Var_{n+1}(\hat{y}(\mathbf{x_n})) - Var_n(\hat{y}(\mathbf{x_n}))$ must be negative. Therefore the the change in variance must be less than or equal to 0.

Scoring: *6 points*

What we show here is that adding new data, regardless of how we choose the point $\mathbf{x}_{n+1}$, and even if we do not get to choose it, cannot hurt the predictions. This is true in expectation. That is, for any point $\mathbf{x}_{n+1}$, the expected error at another arbitrary point $\mathbf{x}$ is lower than the error at $\mathbf{x}$ before incorporating $(\mathbf{x}_{n+1}, y_{n+1})$ into the prediction. The expectations here are with respect to all the observed outputs.

This result is independent of the type of feature vectors we use. This is good since the feature vectors are typically rather constrained (e.g., $\phi(x) = [1, x, x^2]^T$ traces one dimensional set of points in three dimensions).

4. Show that if we include *any* $\mathbf{x}_{n+1}$ in the training set, then the variance at $\mathbf{x}_{n+1}$ comes down by a multiplicative factor that depends on the current variance at $\mathbf{x}_{n+1}$. Indeed, the larger the variance, the faster it will come down.

Answer:

$$\begin{aligned}
\mathsf{Var}_{n+1}(\hat{y}(\mathbf{x}_{n+1})) &= \phi(\mathbf{x}_{n+1})^T A_{n+1}^{-1}\phi(\mathbf{x}_{n+1}) \\
&= \phi(\mathbf{x}_{n+1})^T (A_n^{-1} - \lambda A_n^{-1}\Phi_{n+1}A_n^{-1})\phi(\mathbf{x}_{n+1}) \\
&= \mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1})) - \lambda\phi(\mathbf{x}_{n+1})^T A_n^{-1}\phi(\mathbf{x}_{n+1})\phi(\mathbf{x}_{n+1})^T A_n^{-1}\phi(\mathbf{x}_{n+1}) \\
&= \mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1})) - \frac{\mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1}))^2}{1 + \mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1}))} \\
&= \frac{\mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1}))}{1 + \mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1}))}
\end{aligned}$$

And the variance at $\mathbf{x}_{n+1}$ decreases by $(1 + \mathsf{Var}_n(\hat{y}(\mathbf{x}_{n+1})))$.

Scoring: *5 points*

This result guarantees that we can drive the prediction variance to zero at any desired input point so long as we are willing to query the same point multiple times.

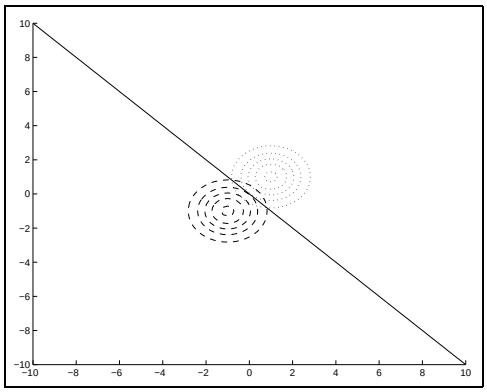
Problem 2: Gaussian Mixtures, Logistic, and Softmax models

The following MATLAB functions can be found in the course Athena locker, and on the course web page: `multigaussian.m`, `gaussian2d` and `plotgaussians.m`.

1. Use these functions to investigate different possible Gaussian decision boundaries. For each of the following specifications, find an appropriate pair of Gaussians and prior class probabilities. Turn in the appropriate plot, and the full specification of the Gaussians (means and covariance matrices) and prior class probabilities.

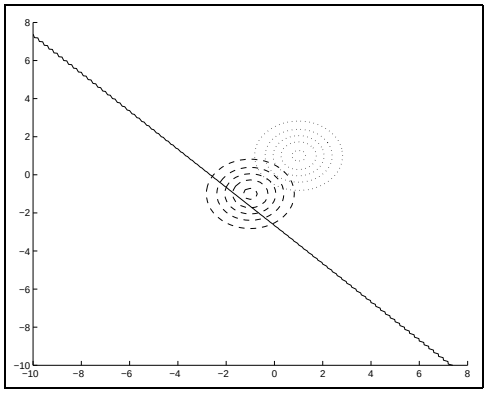
   (a) A linear decision boundary.

   Answer: $\mu_0 = [1, 1], \mu_1 = [-1, -1]. P_0 = 0.5, P_1 = 0.5, \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



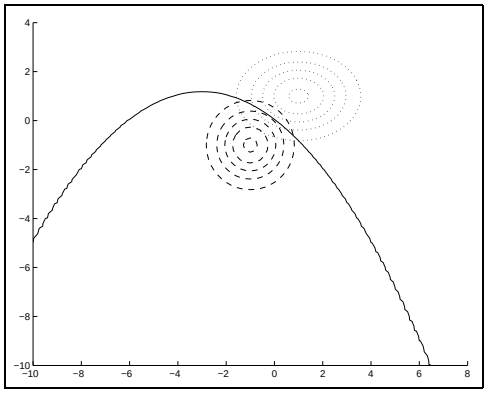
   (b) A linear decision boundary, where both means are on the same side of the decision boundary.

   Answer: $\mu_0 = [1, 1], \mu_1 = [-1, -1]. P_0 = 0.995, P_1 = 0.005, \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



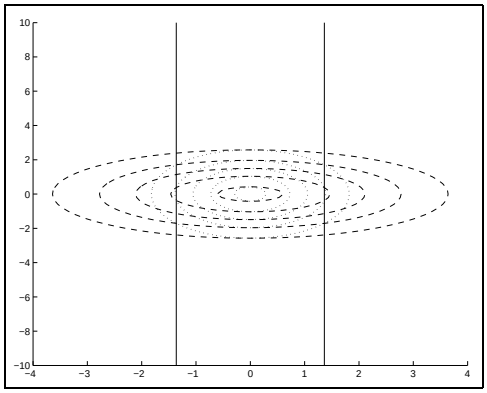
   (c) A parabolic decision boundary.

   Answer: $\mu_0 = [1, 1], \mu_1 = [-1, -1]. P_0 = 0.5, P_1 = 0.5, \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$



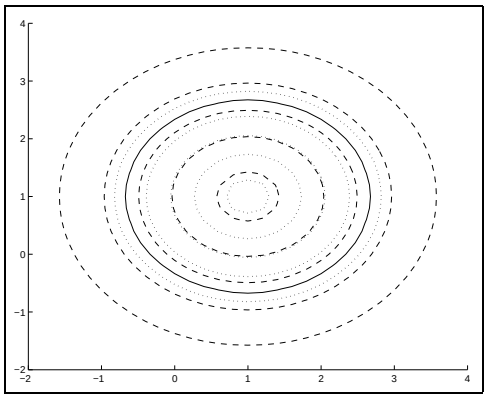
(d) A non-continues decision boundary (i.e. one of the classes is represented by two disconnected regions).

Answer: $\mu_0 = [0,0], \mu_1 = [0,0]. P_0 = 0.5, P_1 = 0.5, \Sigma_0 = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$



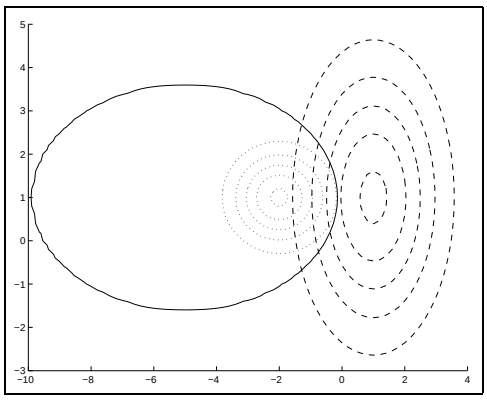
(e) A circular decision boundary.

Answer: $\mu_0 = [1,1], \mu_1 = [1,1]. P_0 = 0.5, P_1 = 0.5, \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$



(f) A skewed (non-circular) ellipsoid decision boundary, with only one of the means inside the ellipsoid.

Answer: $\mu_0 = [-2,1], \mu_1 = [1,1]. P_0 = 0.5, P_1 = 0.5, \Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1/2 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$



(g) No decision boundary– the entire plane is one decision region.

Answer: In order to have the entire plane be classified as a single class, the priors *must* be unequal. The simplest solution is with biased priors (either with a very small bias, e.g. $P_1 = 0.51$, or even with a definite prior

$P_1 = 1$) and otherwise identical Gaussians: since the Gaussian density at any point will be equal, the prior will always be the deciding factor. If the priors are equal, then there must be a point in space (perhaps outside the plotted region) in which each one is greater than the other (since the integral of both of them is one, the integral of the difference is zero, and so if it is negative somewhere it must be positive elsewhere).

If the covariance matrices are equal, the means must also be equal: otherwise, if you go far enough in the direction of one of the means, that Gaussian will eventually have higher posterior. To see this, consider the exponent of the logistic model giving to the class postiriors (we know that the class postiriors of equal-covariance Gaussians is given by a logistic). It is a linear function x, with non-zero slope, and so *must* cross zero somewhere. The only exception is if the prior is deterministic ($P_1 = 1$), in which case no matter what we do, that class will always have a higher posterior (in fact a posterior probability of one).

If the covariance matrices are not equal, the exponent in the class posterior is quadratic, and so it might or might not cross zero, depending on weather the quadratic function has real-valued roots.

Answers in which the mixture components where identical (identical means and covariances, and priors of $1/2$) are not acceptable, as one cannot really discuss a decision region. Rather, the whole plane is a decision boundary, in some sense.

Scoring: *5 points, roughly 2/3 per item. 2 points were taken off if the priors did not sum up to one.*

2. Which of the above decision boundaries can represent the decision boundary for a logistic model ?

   Answer: The decision boundaries (a), (b) and (g) can represent those of a logistic model. We know that the logistic model yields linear decision boundaries. We can get the entire plane in one decision region by choosing a non-zero afine weight w_0, but setting all the other weights to zero. The classification is then independent of the input vector.

   Scoring: *3 points*

In many cases, it is necessary to classify into more than two classes. A natural extension of the Gaussian mixture approach is to fit a Gaussian distribution for each class, and classify each input vector to the class with the highest posterior probability for it.

3. We would like to modify the function `plotgaussians()` to plot the decision boundaries between three Gaussian. A partial modification can be found in the MATLAB

file `plotgaussians3()`, but it is missing the core instructions for plotting the decision boundaries. Complete this function (turn in the MATLAB code) and use it to plot decision boundaries on several examples.

Answer: The following lines complete the function `plotgaussians3.m`:

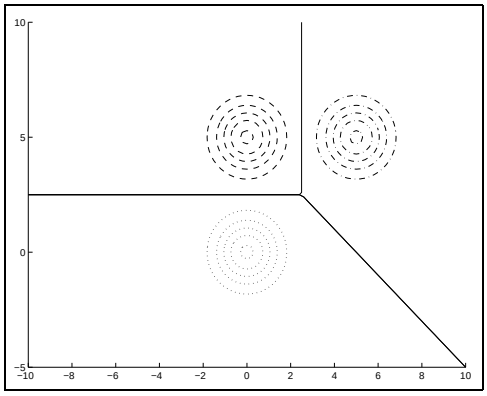
```
contour(x,y,g1*p1-max(g2*p2,g3*p3),[0 0],linespecs{1});
contour(x,y,g2*p2-max(g1*p1,g3*p3),[0 0],linespecs{1});
contour(x,y,g3*p3-max(g2*p2,g1*p1),[0 0],linespecs{1});
```

Each of these lines plots the decision region of one of the three classes. Two of these plots are actually enough, and so we could have dropped the last line.

Turn in plots (and associated parameters) for settings where

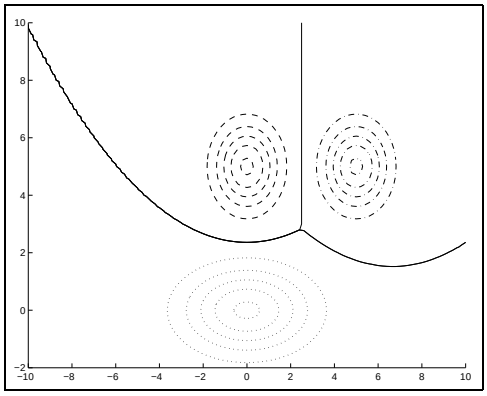
(a) All decision boundaries are linear.

Answer: $\mu_0 = [0,0], \mu_1 = [0,5], \mu_2 = [55]. P_0 = 0.33333, P_1 = 0.33333, P_3 = 0.33333$, $\Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,



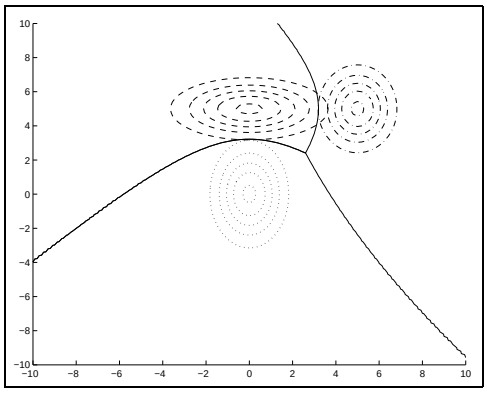
(b) Some decision boundaries are linear, while others are quadratic.

Answer: $\mu_0 = [0,0], \mu_1 = [0,5], \mu_2 = [55]. P_0 = 0.33333, P_1 = 0.33333, P_3 = 0.33333$, $\Sigma_0 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,



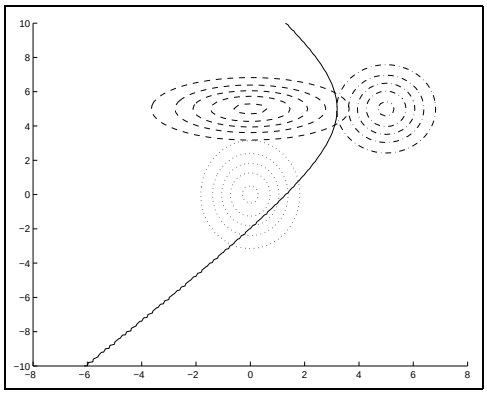
(c) All decision boundaries are quadratic.

Answer: $\mu_0 = [0,0], \mu_1 = [0,5], \mu_2 = [55]. P_0 = 0.33333, P_1 = 0.33333, P_3 = 0.33333$, $\Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$,



(d) There are only two decision regions (one class never gets selected).

Answer: $\mu_0 = [0,0], \mu_1 = [0,5], \mu_2 = [55]. P_0 = 0, P_1 = 0.5, P_3 = 0.5$, $\Sigma_0 = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$, $\Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, (see notes for 1(g)).



ALSO: mark each decision region in the plots with an appropriate label.

Scoring: *3 points for the plotting routine, and 3 points for the plots*

Another possible approach is to generalize the the logistic model. Let $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ be the input vector, and suppose we would like to classify to k classes, that is the output y can take a value in $1, \ldots, k$. The *softmax* generalization of the logistic model uses $k(d+1)$ weights $\mathbf{w} = (w_{ij})$, $i = 1, \ldots, k$, $j = 0, \ldots, d$, which define the following k intermediate values:

$$z_1 = w_{10} + \sum_j w_{1j} x_j$$

$$\ldots$$

$$z_i = w_{i0} + \sum_j w_{ij} x_j$$

$$\ldots$$

$$z_k = w_{k0} + \sum_j w_{kj} x_j$$

The classification probabilities under the softmax model are:

$$\Pr(y = i|\mathbf{x}; \mathbf{w}) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

4. Show that when $k = 2$ the softmax model reduces to the logistic model. That is, show how both give rise to the same classification probabilities $\Pr(y|\mathbf{x})$. Do this by constructing an explicit transformation between the weights: for any given set of $2(d+1)$ softmax weights, show an equivalent set of $(d+1)$ logistic weights.

   Answer: Under a logistic model with weights w':

   $$p(Y = 1|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-z'}} \tag{7}$$

   Where $z' = w'_0 + \sum_j w'_j x_j$. In the softmax model,

   $$p(Y = 1|\mathbf{x}; \mathbf{w}) = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} \tag{8}$$

   equating the two:

   $$\begin{aligned} \tfrac{1}{1+e^{-z'}} &= \tfrac{e^{z_1}}{e^{z_1}+e^{z_2}} \\ e^{z_1} + e^{z_2} &= e^{z_1} + e^{z_1} e^{-z'} \\ e^{z_1} + e^{z_2} &= e^{z_1} + e^{z_1 + z'} \\ e^{z_1 - z'} &= e^{z_2} \\ z' &= z_1 - z_2 \qquad \text{Taking the log} \end{aligned} \tag{9}$$

   With this result we can see that if we set $w'_j = w_{1j} - w_{2j}$ for each j, the the two models are equivalent.

   Scoring: *4 points*

5. Which of the decision regions from question 3 can represent decision boundaries for a softmax model ?

   Answer: Again, only linear decision boundaries are possible. Hence, (a) is possible, but (b) and (c) aren't. Having only two decision regions is possible if the decision boundary between them is linear, as in our example for (d). Other examples for (d) might have a quadratic decision boundary, which is not attainable with a logistic model.

   Scoring: *2 points*

6. Show that the softmax model, for any k, can always be represented by a Gaussian mixture model. What type of Gaussian mixture models are equivalent to a softmax models ?

Answer: Consider a softmax model with k classes and weights w_{ij} and denote $\mathbf{w}_i$ the d-element vector with components $(\mathbf{w}_i)_j = w_{ij}$ for $1 \leq j \leq d$ then the softmax posterior is given by:

$$\Pr(y|\mathbf{x}) = \frac{e^{\mathbf{w}_y^T \mathbf{x} + w_{y0}}}{\sum_i e^{\mathbf{w}_i^T \mathbf{x} + w_{i0}}}.$$

We would like like to find Gaussian mixture parameters $(\pi_i, \mu_i, \Sigma_i)_{i=1..k}$ that would yield the same posterior. The postiriors class probabilities of such a Gaussian mixture is given by:

$$\begin{aligned}
\Pr(y|\mathbf{x}; (\pi_i, \mu_i, \Sigma_i)_{i=1..k}) &= \frac{\pi_y |\Sigma_y|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_y)^T \Sigma_y^{-1}(\mathbf{x}-\mu_y)}}{\pi_i |\Sigma_i|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i)}} \\
&= \frac{\pi_y |\Sigma_y|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma_y^{-1}\mathbf{x} - \mathbf{x}^T \Sigma_y^{-1}\mu_y - \mu_y^T \Sigma_y^{-1}\mathbf{x} + \mu_y^T \Sigma_y^{-1}\mu_y)}}{\sum_i \pi_i |\Sigma_i|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma_i^{-1}\mathbf{x} - \mathbf{x}^T \Sigma_i^{-1}\mu - \mu_i^T \Sigma_i^{-1}\mathbf{x} + \mu_i^T \Sigma_i^{-1}\mu_i)}}
\end{aligned}$$

Taking advantage of covariance matrices being symmetric:

$$= \frac{\pi_y |\Sigma_y|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma_y^{-1}\mathbf{x} - 2\mu_y^T \Sigma_y^{-1}\mathbf{x} + \mu_y^T \Sigma_y^{-1}\mu_y)}}{\sum_i \pi_i |\Sigma_i|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma_i^{-1}\mathbf{x} - 2\mu_i^T \Sigma_i^{-1}\mathbf{x} + \mu_i^T \Sigma_i^{-1}\mu_i)}}$$

The exponents are quadratic functions of $\mathbf{x}$. But in the softmax posterior the exponents are linear functions of $\mathbf{x}$. In order to get linear functions in the exponents, we know we must choose identical covariance matrices, causing the quadratic terms to cancel out. Setting all covariance matrices to Σ:

$$\begin{aligned}
&= \frac{\pi_y |\Sigma|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma^{-1}\mathbf{x} - 2\mu_y^T \Sigma^{-1}\mathbf{x} + \mu_y^T \Sigma^{-1}\mu_y)}}{\sum_i \pi_i |\Sigma|^{-\frac{d}{2}} e^{-\frac{1}{2}(\mathbf{x}^T \Sigma^{-1}\mathbf{x} - 2\mu_i^T \Sigma^{-1}\mathbf{x} + \mu_i^T \Sigma^{-1}\mu_i)}} \\
&= \frac{\pi_y e^{-\frac{1}{2}(-2\mu_y^T \Sigma^{-1}\mathbf{x} + \mu_y^T \Sigma^{-1}\mu_y)}}{\sum_i \pi_i e^{-\frac{1}{2}(-2\mu_i^T \Sigma^{-1}\mathbf{x} + \mu_i^T \Sigma^{-1}\mu_i)}} \\
&= \frac{e^{\mu_y^T \Sigma^{-1}\mathbf{x} - \frac{1}{2}\mu_y^T \Sigma^{-1}\mu_y + \log \pi_y}}{\sum_i e^{\mu_i^T \Sigma^{-1}\mathbf{x} - \frac{1}{2}\mu_i^T \Sigma^{-1}\mu_i + \log \pi_i}}
\end{aligned}$$

This posterior looks very much like the softmax posterior, with a linear function of $\mathbf{x}$ in the exponent. To get the coefficients of $\mathbf{x}$ to be as in the softmax posterior, we must set the means and covariances such that:

$$\mu_i \Sigma^{-1} = \mathbf{w}_i \quad \text{for all } i \tag{10}$$

We can do this for *any* invertible Σ. But it is enough for us to show that there is *one* set of Gaussian mixture parameters that yields the desired posterior, and so we will arbitrarily set $\Sigma = I$. To satisfy (10), we get that μ_i must be

equal to $\mathbf{w}_i$ (for this specific choice of the covariance matrix). This gives us a Gaussian mixture posterior of:

$$\Pr\left(y|x;(\pi_i,\mu_i=\mathbf{w}_i,\Sigma_i=I)_{i=1..k}\right)=\frac{e^{\mathbf{w}_y^T\mathbf{x}-\frac{1}{2}\mathbf{w}_y^T\mathbf{w}_y+\log\pi_y}}{\sum_i e^{\mathbf{w}_i^T\mathbf{x}-\frac{1}{2}\mathbf{w}_i^T\mathbf{w}_i+\log\pi_i}} \tag{11}$$

We would like to set the priors π_i such that the affine terms would agree with w_{i0}. It is tempting to set the priors such that $\log\pi_i-\frac{1}{2}\mathbf{w}_i^T\mathbf{w}_i=w_{i0}$, which would yield the softmax posterior. However, this might result in negative priors, or priors that do not sum up to one. Instead, we first multiply both the numerator and denominator by some constant Z (to be determined), which will effectively normalize the priors:

$$=\frac{Z\pi_y e^{\mathbf{w}_y^T\mathbf{x}-\frac{1}{2}\mathbf{w}_y^T\mathbf{w}_y}}{\sum_i Z\pi_i e^{\mathbf{w}_i^T\mathbf{x}-\frac{1}{2}\mathbf{w}_i^T\mathbf{w}_i}}$$

Now we can get $Z\pi_i e^{-\frac{1}{2}\mathbf{w}_i^Y\mathbf{w}_i}=e^{w_{i0}}$ by setting $\pi_i=e^{\frac{1}{2}\mathbf{w}_i^Y\mathbf{w}_i+w_{i0}}/Z$, and in order to normalize the priors we get that $Z=\sum_i e^{\frac{1}{2}\mathbf{w}_i^Y\mathbf{w}_i+w_{i0}}$.

To summarize, we saw how for any softmax weights, we can always choose the following parameters:

$$\begin{aligned}\pi_i&=\frac{e^{\frac{1}{2}\mathbf{w}_i^Y\mathbf{w}_i+w_{i0}}}{\sum_j e^{\frac{1}{2}\mathbf{w}_j^Y\mathbf{w}_j+w_{j0}}}\\ \mu_i&=\mathbf{w}_i\\ \Sigma_i&=I\end{aligned}$$

and get a Gaussian mixture model that has the softmax posterior.

Although we saw how to get a Gaussian mixture model with unit covariance matrices, in fact any Gaussian mixture model with equal covariance matrices in all of its components has a posterior which can be represented as a softmax. If all components share the same covariance matrix, regardless of what this covariance matrix is, the quadratic terms will cancel out (as we saw), leading to a linear functions in the exponents, and thus a softmax posterior.

In our derivation, note that we arbitrarily choose $\Sigma=I$, but in fact we could have satisfied (10) using any invertible Σ (regardless of the weights of the softmax). Choosing a different covariance matrix, we would get a different Gaussian mixture model, with different joint distribution $\Pr(X,Y)$, but with the same posterior $\Pr(Y|X)$. Note that the softmax model only specifies a posterior, and not a joint distribution.

Scoring: *6 points (2 points lost for neglecting to verify the priors were legitimate)*

The stochastic gradient ascent learning rule for softmax is given by:

$$w_{ij} \leftarrow w_{ij} + \epsilon \sum_t \frac{\partial}{\partial w_{ij}} \log \Pr\left(y^t | \mathbf{x}^t, \mathbf{w}\right)$$

where $(\mathbf{x}^t, y^t)$ are the training examples. We would like to rewrite this rule as a *delta rule*. In a delta rule the update is specified as a function of the difference between the target and the prediction. In our case, our *target* for each example will actually be a vector $\underline{y}^t = (y_1^t, \ldots, y_k^t)$ where:

$$y_i^t = \begin{cases} 1 & \text{if } y^t = i \\ 0 & \text{if } y^t \neq i \end{cases}$$

Our prediction will be a corresponding vector of probabilities:

$$\underline{\hat{y}}^t = (\Pr\left(y = 1 | \mathbf{x}^t\right), \ldots, (\Pr\left(y = i | \mathbf{x}^t\right))).$$

7. Calculate the derivative above, and rewrite the update rule as a function of $\underline{y}^t - \underline{\hat{y}}^t$.

Answer:

$$\log(\Pr(y = i)) = z_i - log\left(\sum_l e^{z_l}\right) \tag{12}$$

$$\frac{\partial z_i}{\partial w_{ij}} = x_j \tag{13}$$

Two cases:

$$\frac{\partial \log \Pr(y = i)}{\partial w_{ij}} = x_j - \frac{e^{z_i}}{\sum_l e^{z_l}} x_j = y_i x_j - \hat{y}_i x_j \tag{14}$$

$$\frac{\partial \log \Pr(y = k \neq i)}{\partial w_{ij}} = -\frac{e^{z_i}}{\sum_l e^{z_l}} x_j = y_i x_j - \hat{y}_i x_j \tag{15}$$

Combining them in matrix notation:

$$\begin{aligned} \frac{\partial \log \Pr(\underline{y}^t | \mathbf{x}^t)}{\partial w_{ij}} &= y_i^t x_j - \hat{y}_i^t x_j \\ &= \left[\left(\underline{y}^t - \underline{\hat{y}}^t\right)^T \mathbf{x}^t\right]_{ij} \end{aligned}$$

Giving the new update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \epsilon \left(\underline{y}^t - \underline{\hat{y}}^t\right)^T \mathbf{x}^t \tag{16}$$

Scoring: *4 points*

Problem 3: Regularization

Reference: Lecture five

We elaborate here a bit on the relation between the "effective" number of parameter choices and regularization discussed in lecture 5. We do this in the context of a simple 1-dim logistic regression model

$$P(y=1|x,\mathbf{w}) = g\,(\,w_0 + w_1 x\,) \tag{17}$$

where $g(z) = (1+\exp\{-z\})^{-1}$. We assume that $x \in [-1,1]$.

To understand regularization in this context, we'll try to carve up our parameter space $\mathbf{w} = [w_0, w_1]^T \in \mathcal{R}^2$ into regions such that our loss (or log-probability) is roughly constant within each region. This will help us determine how many "effective" parameter choices we really have. Ideally, the regularization that we impose would directly limit (our estimate of) the number of parameter choices.

A bit more precisely, we want the log-probability $\log P(y|x,\mathbf{w})$ to vary by no more than ϵ within each region in the parameter space. To find such regions, we first examine how the log of the logistic function varies as a function of its input: (this result will be useful to you later on)

$$\frac{\partial}{\partial z}\log g(z) = \frac{1}{g(z)}\,\frac{\partial}{\partial z} g(z) = \frac{1}{g(z)}\,g(z)(1-g(z)) = 1 - g(z) \tag{18}$$

In other words, since $g(z) \in [0,1]$ (probability), the derivative here is also bounded by 1. The function $\log g(z)$ therefore varies at most with slope 1 as a function of z, or

$$|\log g(z) - \log g(z')| \le |z - z'| \tag{19}$$

for any two points z and z'. To use this result, we define $z = w_0 + w_1 x$ and $z' = w_0' + w_1' x$ for any $x \in [-1,1]$. This gives

$$\begin{aligned}
|\log P(y=1|x,\mathbf{w}) - \log P(y=1|x,\mathbf{w}')| &= |\log g(z) - \log g(z')| \le |z - z'| && (20)\\
&= |(w_0 - w_0') + (w_1 - w_1')x| && (21)\\
&\le |(w_0 - w_0')| + |(w_1 - w_1')x| && (22)\\
&\le |(w_0 - w_0')| + |(w_1 - w_1')| && (23)
\end{aligned}$$

since $|x| \le 1$ by assumption. So, whenever $|w_0 - w_0'| + |w_1 - w_1'| \le \epsilon$, the corresponding losses or (negative) log-probabilities are also bounded by ϵ. We can therefore carve up the parameter space by finding discrete points $\mathbf{w}^{(i)}$ and regions around them such that

$$|w_0^{(i)} - w_0'| + |w_1^{(i)} - w_1'| \le \epsilon \tag{24}$$

whenever $\mathbf{w}'$ belongs to the i^{th} region. These regions are shown in Figure 1 for $\epsilon = 0.4$. We have also included in the Figure the area limited by the Euclidean norm of the parameter

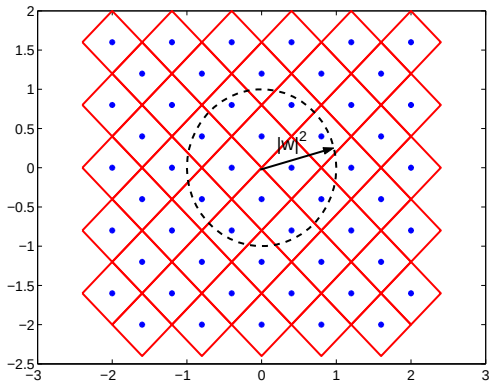


Figure 1: Regions in the parameter space corresponding to roughly constant losses for simple logistic regression model.

vector, $\|\mathbf{w}\|^2$. Increasing the the limit $\|\mathbf{w}\|^2$ clearly incorporates more "choices" and it makes sense to use this type of norm in regularization.

Note: Explicitly finding the regions as we have done here is merely a conceptual device in understanding (and analyzing) estimation methods. We never have to identify such regions nor the discrete "effective" parameter choices in practice.

The problem:

The goal here is to understand regularization a bit better in the context of a simple logistic regression model

$$P(y = 1|\mathbf{x}, \mathbf{w}) = g\,(\,w_0 + w_1 x_1 + \ldots + w_d x_d\,) \tag{25}$$

We have provided you with a few helpful MATLAB functions for this purpose:

> `w = logisticreg(X,y,c) % (logisticreg.m)` performs fast optimization of logistic regression coefficients with squared norm regularization (regularization parameter c).
>
> `[X,y] = sample(n) % (sample.m)` generates training or test examples from a very simple classification problem
>
> `EcvlogP = crossvalid(X,y) % (crossvalid.m)` computes leave-one-out cross-validation estimate of the expected log-probability of labels for logistic regression models.

1. Generate $n = 20$ training and $n = 200$ (or more) test examples using the `sample` function. The resulting examples will be 10 dimensional vectors. We ask you to plot training and test performance for different levels of regularization. More precisely, for each $c = \{1, \ldots, 20\}$, use the `logisticreg` function to get the corresponding parameter estimate $\hat{\mathbf{w}}$. Evaluate and plot the average training and test log-probabilities

of labels as a function of the regularization parameter c. Keep the training and test sets fixed while varying c. Return the resulting figure.

Answer: We use the following MATLAB functions:

```
function [trainll, testll, cvll] = testlogistic(n,maxc)
  [trainx, trainy] = sample(n);
  [testx, testy] = sample(1000);
  trainll = zeros(maxc,1);
  testll = zeros(maxc,1);
  cvll = zeros(maxc,1);
  for c=1:maxc
     w = logisticreg(trainx,trainy,c);
     trainll(c) = logisticll(trainx,trainy,w);
     testll(c) = logisticll(testx,testy,w);
     cvll(c) = crossvalid(trainx,trainy,c);
  end

function ll = logisticll(x,y,w)
  p = g(w(1) + x*w(2:end));
  ll = mean(y.*log(p) + (1-y).*log(1-p));

function p = g(z)
  p = 1./(1+exp(-z));
```

Using these functions, the following code generates the graphs for this, and subsequent, questions:

```
>> [train20,test20,cv20] = testlogistic(20,20);
>> [train50,test50,cv50] = testlogistic(50,20);
>> plot([train20,test20,cv20]);
>> hold
>> legend('train','test','cv');
>> plot([train50,test50,cv50],'--');
>> gtext('training set sizes: 20 (solid), 50 (dashed)')
```

Scoring: *5 points*

2. Explain the *relevant* qualitative properties of the two curves in your figure.

   Answer: The training performance decreases as we increase the regularization parameter C, since we are decreasing the relative importance of fitting the training data.

   The test performance improves due to less overfitting.

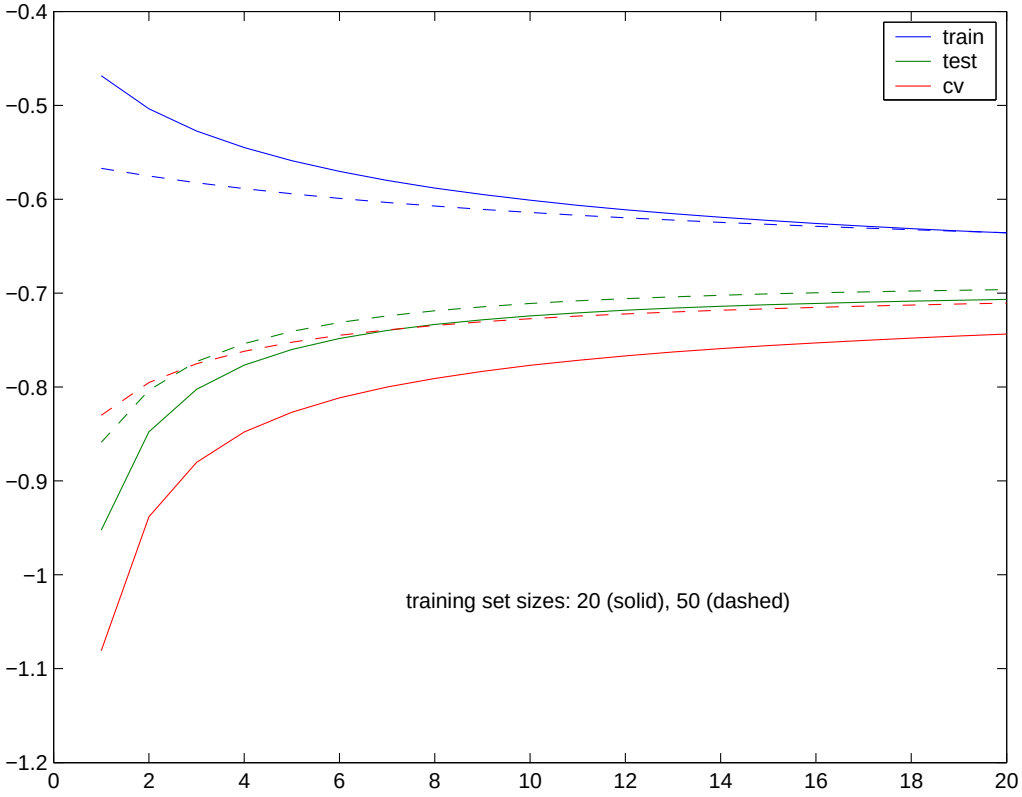


Figure 2: Logistic regression performance (Problem 3)

Most importantly, the training performance becomes a more accurate predictor of the test performance (i.e. the curves get closer), which is the goal of regularization.

Scoring: *5 points*

3. Use the `crossvalid` function to generate leave-one-out cross-validation estimates for each $c = \{1, \ldots, 20\}$ and include these results in your figure. How useful is cross-validation in approximating test performance? For setting the regularization parameter c?

   Answer: The cross-validation performance is slightly worse than the test error (since we are plotting the log-likelihood, higher is better). This is reasonable, since the cross-validation training uses only 19, instead of 20, examples. It is thus not a very accurate predictor of the test performance. However, although it is consistently worse, it does mirror the test error, and so is useful for setting the regularization parameter C.

   Scoring: *5 points, 2 for the generating the figure, 3 for the explanation*

4. Explain the changes in the *three* curves when you add 30 examples to the existing training set. (Note: it may be helpful to plot the new results using the same axis. You can store the previous axis with `prevaxis = axis` and recover the settings later on with `axis(prevaxis)`).

   Answer: The training performance decreases, since it becomes harder to fit all the training points. This is especially pronounced when there is not much regularization, and the training can more freely try to (over)fit the training points.

   The test performance, and with it also the cross-validation estimate, increase, as more data is available from training. The cross-validation more accurately reflects the test error, since the relative difference in the training set size drops from $1/20$ to $1/50$.

   Scoring: *5 points*

Problem 4: Stochastic Gradient Ascent

Reference: Lectures four, five; Chapter 6

Here you will solve a digit classification problem with logistic regression models. We have made available the following training and test sets:

`digit_x.dat, digit_y.dat, digit_x_test.dat, digit_y_test.dat`.

1. Derive the stochastic gradient ascent learning rule for a logistic regression model starting from the regularized likelihood objective

$$J(\mathbf{w}; c) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2}\|\mathbf{w}\|^2 \tag{26}$$

   where $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$, or by modifying your derivation of the delta rule for the softmax model.

   (Normally we would not include w_0 in the regularization penalty but have done so here for simplicity of the resulting update rule).

   Answer: As shown in class, we rewrite:

$$J(\mathbf{w}; c) = \sum_{i=1}^{n} \left(\log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2n}\|\mathbf{w}\|^2 \right)$$

And the stochastic online gradient ascent learning rule using the ith pattern is given by:

$$\begin{aligned} w_j \leftarrow & w_j + \epsilon \frac{\partial}{\partial w_j} \left(\log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2n} \|\mathbf{w}\|^2 \right) \\ & = w_j + \epsilon \left(\frac{\partial}{\partial w_j} \log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2n} \frac{\partial}{\partial w_j} \|\mathbf{w}\|^2 \right) \end{aligned}$$

We will calculate both of these derivatives.

The first is just the regular (no regularization) gradient ascent derivative:

$$\frac{\partial}{\partial w_j} \log P(y_i|\mathbf{x}_i, \mathbf{w})$$

using the identity g(z)=1-g(z) and checking for $y_i = 0$ and $y_i = 1$:

$$= \frac{\partial}{\partial w_j} \log g\left((-1)^{1-y_i} (\mathbf{w}^T \mathbf{x}_i + w_0) \right)$$

recalling that $(\log g(z))' = \frac{g'(z)}{g(z)} = \frac{g(z)(1-g(z))}{g(z)} = 1 - g(z)$:

$$= \left(1 - g((-1)^{1-y_i}(\mathbf{w}^T \mathbf{x}_i + w_0))\right) (-1)^{1-y_i} \frac{\partial (\mathbf{w}^T \mathbf{x}_i + w_0)}{\partial w_j}$$

again checking for $y_i = 0$ and $y_i = 1$ we get:

$$= (y_i - P(1|\mathbf{x}_i; \mathbf{w})) \frac{\partial (\mathbf{w}^T \mathbf{x}_i + w_0)}{\partial w_j}$$

and for $j \neq 0$:

$$= (y_i - P(1|\mathbf{x}_i; \mathbf{w})) x_{i,j}$$

while for $j = 0$ we get $\frac{\partial}{\partial w_0} \log P(y_i|\mathbf{x}_i, \mathbf{w}) = (y_i - P(1|\mathbf{x}_i; \mathbf{w}))$.

The derivative in the second term is:

$$\frac{\partial}{\partial w_j} \|\mathbf{w}\|^2 = \frac{\partial}{\partial w_j} \sum_k w_k^2 = 2w_j$$

Combining we get:

$$\begin{aligned} w_j \leftarrow & w_j + \epsilon \frac{\partial}{\partial w_j} \left(\log P(y_i|\mathbf{x}_i, \mathbf{w}) - \frac{c}{2n} \|\mathbf{w}\|^2 \right) \\ & = w_j + \epsilon \left((y_i - P(1|\mathbf{x}_i; \mathbf{w})) x_{i,j} - \frac{c}{2n} 2w_j \right) \\ & = (1 - \frac{\epsilon c}{n}) w_j + \epsilon (y_i - P(1|\mathbf{x}_i; \mathbf{w})) x_{i,j} \end{aligned}$$

Where $x_{i,0}$ is taken to be one. Considering this as an initial coordinate of $\mathbf{x}_i$, we can write this in matrix form:

$$\mathbf{w} \leftarrow (1 - \frac{\epsilon c}{n})\mathbf{w} + \epsilon(y_i - P(1|\mathbf{x}_i; \mathbf{w}))\mathbf{x}_i$$

Scoring: *5 points*

2. Write a MATLAB function `w = SGlogisticreg(X,y,c,epsilon)` that takes inputs similar to `logisticreg` from the previous section, and a learning rate parameter ϵ, and uses stochastic gradient ascent to learn the weights. You may include additional parameters to control when to stop, or hard-code it into the function.

Answer:

```
function [w] = SGlogisticreg(X,y,c,epsilon,stopdelta)
  [n,d] = size(X);
  X = [ones(n,1),X];
  w = zeros(d+1,1);
  cont = 1;
  while (cont)
     perm = randperm(n);
     oldw = w;
     for i=1:n
         w = (1-epsilon*c/n)*w+epsilon*(y(i)-g(X(i,:)*w))*X(i,:)' ;
     end
     cont = norm(oldw-w)>=stopdelta*norm(oldw) ;
  end
```

Scoring: *5 points*

3. Provide a rationale for setting the learning rate and the stopping criterion in the context of the digit classification task. You should assume that the regularization parameter c remains fixed at 1. (You might wish to experiment with different learning rates and stopping criterion but do **NOT** use the test set. Your justification should be based on the available information before seeing the test set.)

Answer: Learning rate: If the learning rate is too high, any memory of previous updates will be wiped out (beyond the last few points used in the updates). It's important that all the points affect the resulting weights and so the learning rate should scale somehow with the number of examples. But how? When the stochastic gradient updates converge, we are not changing the weights on average. So each update can be seen as a slight random perturbation around the correct weights. We'd like to keep such stochastic effects from pushing the weights too far from the optimal solution. One way to

deal with this is to simply average the random effects by making the learning rate scale as $\epsilon = \frac{c}{n}$ for a constant c, somewhat less than one.

But this would be slow. It's good to keep the variance of the sum of the random perturbation at a constant and instead set $\epsilon = \frac{c}{\sqrt{n}}$: You may recall from the first problem set that if Z_i is a Gaussian with zero zero and unit variance, then $\sum_{i=1}^{n} Z_i$ has variance n. Here Z_i corresponds to a gradient update based on the ith example. Dividing by the standard deviation of the sum, $\sqrt{n}$, makes the gradient updates have an overall fixed variance.

Since the update is also proportional to the norm of the input examples you might also divide the learning rate by the overall scale of the inputs. If we have d binary coordinates, the norm is at most $\sqrt{d}$. We get a learning rate of $\epsilon = \frac{c}{\sqrt{nd}}$.

Stopping criterion: We want to stop when a full iteration through the training set does not make much difference *on average*. Note that unless we can perfectly separate the training set, we would still expect to get specific training examples that will cause change, but at convergence they should cancel each other out. We should also not stop just because one, or a few, examples did not cause much change– it might be that other examples will.

And so, after each full iteration through the training set, we see how much the weight changed since before the iteration. As we do not know what the scale of the weights will be, we check the magnitude of the change relative to the magnitude of the weights. We stop if the change falls bellow some low threshold, which represents out desired accuracy of the result (this ratio is the parameter `stopdelta`).

Scoring: *1 point*

4. Set $c = 1$ and apply your procedure for setting the learning rate and the stopping criterion to evaluate the average log-probability of labels in the training and test sets. Compare the results to those obtained with `logisticreg`. For each optimization method, report the average log-probabilities for the labels in the training and test sets as well as the corresponding mean classification errors (estimates of the miss-classification probabilities). (Please include all MATLAB code you used for these calculations)

Answer: To calculate also the classification errors, we use a slightly expanded version of `logisticll.m`:

```
function [ll,err] = logisticle(x,y,w)
 p = g(w(1) + x*w(2:end));
 ll = mean(y.*log(p) + (1-y).*log(1-p));
 err = mean(y ~= (p>0.5));
```

We set the learning rate to: $\epsilon = \frac{0.1}{\sqrt{nd}} = \frac{0.1}{80}$, try a stopping granularity of $\delta = 0.001$, and get:

Average log probabilities:

	Newton-Raphson	Stochastic Gradient Ascent
Train	-0.0829	-0.1190
Test	-0.2876	-0.2871

Classification errors:

	Newton-Raphson	Stochastic Gradient Ascent
Train	0.01	0.02
Test	0.125	0.1125

Results for various stopping granularities are presented in figures 3 and 4.

Scoring: *5 points*

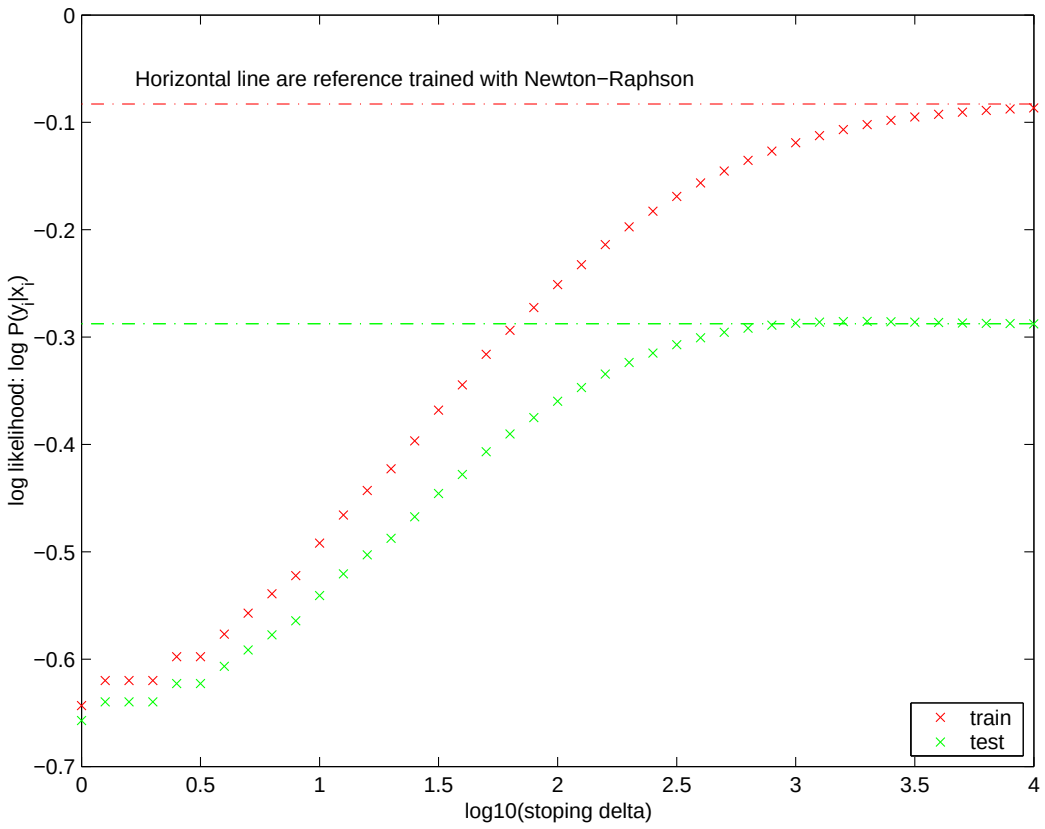


Figure 3: Logistic regression log-likelihood, when trained with stochastic gradient ascent, for varying stopping criteria

5. Are the train/test differences between the optimization methods reasonable? Why? (Repeat the gradient ascent procedure a couple of times to ensure that you are indeed looking at a "typical" outcome)

   Answer: Although both optimization methods are trying to optimize the same objective function, neither of them is perfect, and so we expect to see some discrepancies, as we do in fact see.

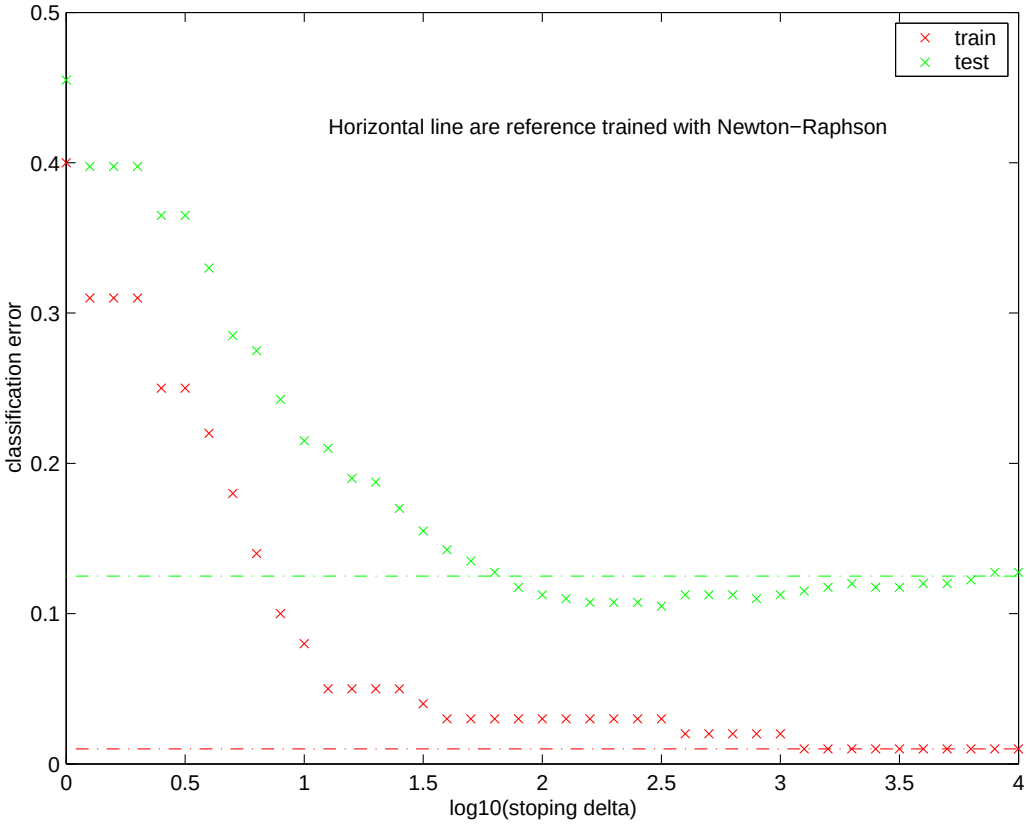


Figure 4: Logistic regression mean classification error, when trained with stochastic gradient ascent, for varying stopping criteria

In general, we would expect the Newton-Raphson method implemented in `logisticreg.m` to perform better, i.e. come closer to the true optimum. This should lead to a better objective function, which especially for small values of c, would translate into higher training performance / lower training error. On the other hand, the stochastic gradient ascent might not come as close to the optimum, especially when the stopping criteria is very relaxed. This can be clearly seen in figures 3 and 4, where training performance improves as the stopping criteria becomes more stringent, and eventually converges to the (almost) true optimum found with Newton-Raphson. Note also the slight deviations from monotonicity, which are a result of the randomness in the stochastic gradient ascent procedure.

However, the same cannot necessarily be said about the test error. In fact, early stopping of stochastic gradient ascent can in some cases be seen as a form of regularization, that might lead to better generalization, and hence better training error. This can be seen in the figures (as well as in the tables for $\delta = 0.001$), especially when comparing the classification errors. For values of δ of around 0.01 to 0.0005, the logistic model found with stochastic gradient

outperforms the "optimum" logistic model found with Newton-Raphson. This does not mean that Newton-Raphson did not correctly solve the optimization problem— we tried to optimize maximize training log likelihood, which indeed we did. We simply did too good of a job and overfit the training data.

Early stopping can sometimes be useful as a regularization technique. In this case, we could have also increased c to get stronger regularization.

Scoring: *1 point*

6. The classifiers we found above are both linear classifiers, as are all logistic regression classifiers. In fact, if we set c to a different value, we are still searching the same set of linear classifiers. Try using `logisticreg` with different values of c, to see that you get different classifications. Why are the resulting classifiers different, even though the same set of classifiers is being searched ? Contrast the reason with the reason for the differences you explained in the previous question.

   Answer: We are searching the same space of classifiers, but with a different objective function. This time not the optimization method is different (which in theory should not make much difference), but the actual objective is different, and hence the true optimum is different. We would not expect to find the same classifier.

   Scoring: *5 points*

7. Gaussian mixture models with identical covariance matrices also lead to linear classifiers. Is there a value of c such that training a Gaussian mixture model necessarily leads to the same classification as training a logistic regression model using this value of c ? Why ?

   Answer: There is no such value of c. The objective functions are different, even for $c = 0$. The logistic regression objective function aims to maximize the likelihood of the labels *given* the input vectors, while the Gaussian mixture objective is to fit a probabilistic model for the training input vectors *and* labels, by maximizing their joint *joint* likelihood.

   Scoring: *5 points*