6.867 Machine Learning

Problem set 2 - solutions

Thursday, October 3

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). When answering question of the form *What is X* or *Write down X* show how exactly you arrived at the answer, without missing crucial steps in reasoning. You may turn in answers to questions marked "optional"— they will be read and corrected, but a grade will not be recorded for them.

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. Do **not** turn in code supplied to you with the problem set.

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

We saw in the lectures how active learning can be used in the context of regression problems. Here we will apply the ideas of active learning to another task - classification. We are concerned with a binary classification problem, where input vectors $\mathbf{x}$ are associated with one of the two classes, called "positive" (labeled 1) and "negative" (labeled 0). The learner is presented with a labeled set of training examples; in addition, an "oracle" is available - that is, a function that given a data point, returns the label of the class that generated this point. Note that no other information is available available to the learner - for example, no knowledge regarding the form of underlying probability distributions. The goal, of course, is to develop a classification rule that will predict the class label for any new input vectors.

The classifier we want to use in this problem is the *K-nearest neighbor* (kNN) classifier. "Training" of kNN consists simply of collecting n labeled points $X_{train} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$. At the "testing" phase, when presented with an unlabeled input vector $\mathbf{x}$, the classifier finds

the k points in the training set that are the closest to $\mathbf{x}$ (here relative to the Euclidean distance $\|\mathbf{x}-\mathbf{x}_i\|$), and assigns $\mathbf{x}$ the label shared by the majority of these k nearest neighbors. Note that k is a parameter of the classifier; it is typically set to an odd value, in order to prevent "ties".

We will experiment with the active learning method on a simple binary classification task over 2-dimensional Euclidean space in order to be able to view the results. You can use the implementation of kNNavailable for download from the course site (it is a slightly modified part of a software library accompanying Bishop's "Neural Network for Pattern Recognition"). You can prepare a kNNclassifier by a call to

```
knn_classifier = knn(d,k,Xtrain,Ytrain)
```

where `d` is the dimension of input vectors (2 in our case), `k` is the number of neighbors to be used by kNN, and `Xtrain,Ytrain` are respectively the input vectors (one per row) and the corresponding labels (column of 1/0). Once a classifier has been constructed, it can be applied on a set of input vectors `Xtest` by running

```
nncount=knntest(knn_classifier,Xtest)
```

which returns in `nncount` the relative count, for each test example, of the negative and positive examples among its k nearest neighbors *in the training set* on which the classifier is based. More precisely, let us denote the counts of positive and negative examples among the k nearest neighbors of $\mathbf{x}$ by $n_k^+(\mathbf{x})$ and $n_k^-(\mathbf{x})$ respectively. The value in `nncount` corresponding to the test example $\mathbf{x}$ is $\left(n_k^+(\mathbf{x}) - n_k^-(\mathbf{x})\right)/k$ - for instance, with 2 positive and 5 negative nearest neighbors (in 7-NN classifier) it would be (2-5)/7=-3/7. Obviously, the sign of this relative count conveys the predicted label of the example (positive corresponds to 1, negative to 0).

Note that $n_k^+(\mathbf{x})$ and $n_k^+(\mathbf{x})$ are related to our (perceived) confidence regarding the label of $\mathbf{x}$. If all $n_k^+(\mathbf{x}) = k$, i.e. all k nearest neighbors are positive, then we are fairly confident about predicting +1 as the label for $\mathbf{x}$. If $n_k^+(\mathbf{x}) = (k+1)/2$ and $n_k^+(\mathbf{x}) = (k-1)/2$, we will still label $\mathbf{x}$ as positive, but the training data provides much less support for this decision.

We shall measure our confidence in the predicted label for $\mathbf{x}$ by

$$C(\mathbf{x}) = \frac{|n_k^+(\mathbf{x}) - n_k^-(\mathbf{x})|}{k}.$$

Note that $C(\mathbf{x})$ takes values between $1/k$ and 1 for odd k.

We can now make use of $C(\mathbf{x})$ to construct a simple active learning strategy. We start with a small training set of n labeled points, and a large set of unlabeled points (for instance, all the points on a bounded 2D grid). The unlabeled points here define the points we can query labels for. The active learning algorithm proceeds by adding new labeled points to the labeled set, one at a time. In each iteration, it chooses the unlabeled point whose label

we have least confidence in (breaking ties randomly). The labels are provided by the oracle. The algorithm stops when it has labeled sufficiently many points, and returns kNNclassifier based on the accumulated set of labeled points.

(Note: a more reasonable active learning method would be based on "value of information calculations" alluded to in the lectures. Such an approach would inspect the expected consequences of labeling a particular point in addition to looking at the measure of confidence. For example, labeling a point generally affects the confidence in other labels as well – the labels are "tied" in this sense. We should take into account such effects in deciding which point is the most useful to label. We will revisit these issues in later on in the course. The simple active learning method suffices for our purposes here.)

1. [**7pt**] Using the training data provided in `XYtrain.dat`, construct kNNclassifier with $k = 7$. These data points were sampled independently from the corresponding class-conditionals of each class. What is the training error of the kNNclassifier? Test it on the test data provided in `XYtest.dat`. (All the data files mentioned in this problem set are in text format, with the input vectors first on each row, and the last column occupied by the class label.) What is the test error?

   Answer: The data used in this problem was drawn from two Gaussian distributions (as would become clear, for instance, from looking at the code of `oracle.m`). The i.i.d. sampling generates data points of which many are relatively far from the other class, and from the decision boundary. One can expect to do better with points which are more concentrated enough to the other class to allow "fine tuning" of the decision boundary.

   The training error of the kNNclassifier is 5%. The test error is 8.3%.

   Scoring: *7 points if the numbers are right*

   Advice: A simple way of generating a random sampling of m out of n points, without replacement: let the rows of `X` be your n points. Then, construct a random permutation of indices in `X` by calling `rp=randperm(size(X,1));`, take the first m elements, and your sample is in `X(rp(1:m),:)`.

2. [**23pt**] Now randomly select 4 data points from each class in `XYtrain`, and run active learning algorithm by call to

   ```
   [al_classifier,selectedX]=activelearn(Xstart,Ystart,k,grid,queries);
   ```

   with `Xstart,Ystart` being the randomly selected labeled points and their labels, $k = 7$ is the parameter of kNN, `grid` is the grid constructed using `make2Dgrid`, and `queries` is the number of points that the active learner may label during its run. Let `queries` be half of the number of examples in the labeled training set (that is, 50).

What is the training error (i.e. the error on the set of labeled points used by the active learning)? What is the test error? Explain the differences, if any, between the test error of the actively learned kNNand the classifier learned with the same number of labeled points but sampled i.i.d. from each class. Plot the originally labeled points and the points selected by the active learner, and refer to the plot in your explanation.

You may find the following Matlab functions (available from the course website) useful; see the comments in the code for the semantics and usage.

```
y = oracle(x);
grid = make2Dgrid(limX,limY,N);
plotres(x,y,selected);
```

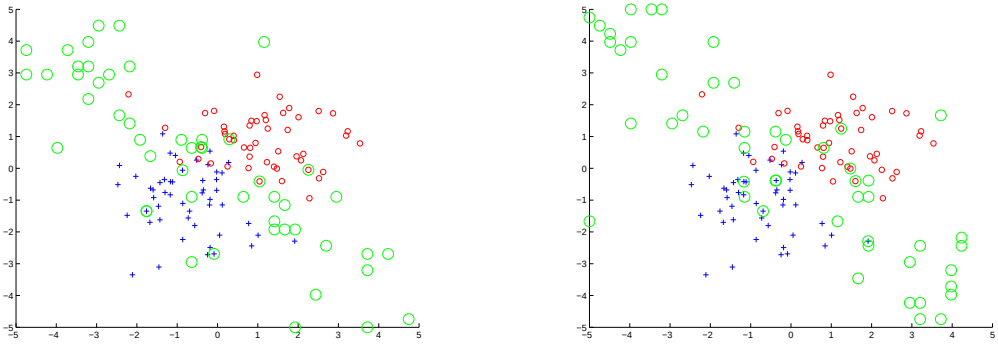
Answer: Before we proceed with a solution, let us mention an example of a practical situation where active learning (AL) with "oracle" makes sense. Consider a fatal decease which we would like to learn to diagnose with a few inexpensive blood tests. The decease already can be diagnosed with 100% precision (or almost that) by running an extremely expensive series of hi-tech tests (MRI and alike). In this situation, we could just send lots of randomly selected people to MRI and use the returned results as labels for training; AL could be more compelling however since we are interested in doing expensive "oracle" tests only on the patients who will contribute to a better blood test diagnosis.

Let us now return to our simple AL algorithm. There is, of course, an element of randomness introduced by the choice of the first 8 labeled points, as well as by the random tie-breaking between points with equal C, and therefore the test error of the AL classifier is a random variable. However, repeating this experiment for a large number of times would show that the distribution of this random variable is peaked around value very close to the test error of the original kNN. You were not required to repeat the experiment, although doing so certainly helps to understand the underlying process.

Figure 1 shows two quite extreme samples from the said distribution. To the left, the AL classifier has training error 14% and test error 12.8%, which is worse than the test error of the original classifier. To the right, the training and test error are respectively 8% and 6.8%, which means we do better than the original classifier, even with half of the number of labeled points. Note that the selected points (circles) tend to be close to the correct decision boundary; more so on the right, which leads to a better generalization performance. On the other hand, many of these points are concentrated in areas where the probability of input (under the true Gaussian class-conditionals), and therefore these points' contribution to reducing the generalization error, is low (this has to do with the localized nature of the kNNclassifier: the points other than the closest k neighbors of $\mathbf{x}$ do not matter for the classification of $\mathbf{x}$).

If we allow for 100 queries in the AL algorithm, we get results like that shown in Figure 2(a). The distribution of the test error in that case becomes even more peaked, and its mean becomes lower than the test error of the original classifier. Another way to improve the performance of AL is to run it on a smaller grid, for example bounded by (-2,2), as shown in Figure 2(b). The advantage here is in concentrating on the "important" areas in terms of data density - of course, knowing this requires some knowledge, or estimate about the true distribution of the data.

In summary, in this case we get essentially the same results with AL as without it, but with half of the number of labeled points; this is due to the fact that the points selected for query are generally closer to the decision boundary. If we allow same number of queries as used by the original algorithm, we slightly improve the results.

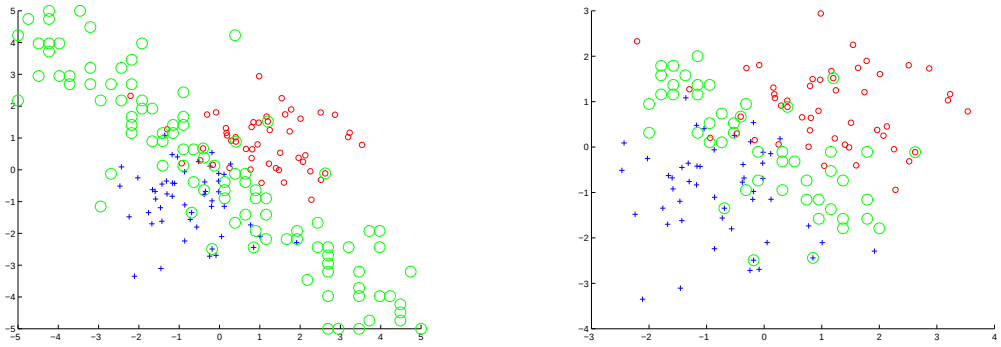


(a) Test error 12.8%

(b) Test error 6.8%

Figure 1: Results of active learning for different random choices of the "seed" points

Scoring: *7 points for providing correct plot; 5 points for understanding that the error is a random variable; 6 points for reasonable values of errors; 6 points for explanation of the trend of AL points towards the boundary; 3 points for explanation that points*



(a) AL with 100 queries;test error 7.7%

(b) AL on a smaller grid; test error 7.5%

Figure 2: Possible improvements on AL performance

in low-density areas do not improve result by much; 3 points for various other correct remarks.

Problem 2: Optimal classification

In this problem we shall investigate decision making process, with the assumption (often unrealistic in practice) that the underlying probability densities are known for each class.

We saw in class that the optimal classification rule for binary classification problem when the class-conditional densities $p(x|y=0)$ and $p(x|y=1)$ are known, is given by choosing the label with maximum posterior:

$$y^*(x) = \operatorname*{argmax}_y p(y|x). \tag{1}$$

We will now show that this rule is indeed optimal, in the sense defined shortly. Let $L(y^*, \hat{y})$ be the loss that we would incur if we predict $\hat{y}$ when the true label is y^*. For example, the *0/1 loss* corresponds to the case where there is no penalty for the correct decision and a fixed penalty 1 for any incorrect decision (that is, if $y^* \neq \hat{y}$). We can express the 0/1 loss as $1 - \delta_{y^*,\hat{y}}$, where $\delta_{i,j}$ is Kronecker's delta taking a value 1 when $i = j$ and zero otherwise. The *expected loss* associated with the decision rule $\hat{y}(x)$ is the expectation $\mathbf{E}_{(x,y)\sim p}\left[L(y, \hat{y}(x))\right]$. In other words, we measure the average loss that we would incur if we sampled examples from the true distribution $p(x, y)$ over examples and labels. Note that with 0/1 loss this expected loss is exactly the probability of miss-classifying an example. Make sure you understand why before continuing.

1. [**25pt**] Show that the classification rule (1) indeed minimizes the expected loss of the decision for the 0/1 loss.

   Hint: consider the expected loss as the expectation of the loss value on pairs (x, y) sampled from the joint distribution; think of how we should sample from the joint distribution when we know that $p(x, y) = p(x)p(y|x) = p(y)p(x|y)$, i.e., we can either first sample x from $p(x)$ and then y from $p(y|x)$ or vice versa; note that since we don't have any restrictions on how the decision rule can depend on x, the optimal decision rule has to be optimal in some sense for each x separately.

   Answer: Here is the explicit expression for the expected loss:

$$\begin{aligned}\mathbf{E}_{p(x,y)}\left[L(y, \hat{y}(x))\right] &= \int_x \sum_y L(y, \hat{y}(x)) p(x, y) dx \\ &= \int_x p(x) \sum_y \left(1 - \delta_{y,\hat{y}(x)}\right) p(y|x) dx.\end{aligned}$$

   Let us restate our sampling model. We have a fixed rule $\hat{y}(x)$ - that is, we specify how $\hat{y}$ should be chosen for any x, prior to seeing any data. Then, we sample $x' \sim p(x)$, compute $\hat{y}(x')$ based on our rule, sample $y \sim p(y'|x = x')$

and calculate $L(y', \hat{y}(x'))$. Note that the value of L will be 1 in the event of $y' \neq \hat{y}(x')$, and 0 in the event $y' = \hat{y}(x')$. For fixed x', therefore, the loss associated with $\hat{y}(x')$ on that x', called *conditional risk* of deciding $\hat{y}$ given x', is

$$\begin{aligned} R(\hat{y}|x') = \mathbf{E}_{p(y|x')}\left[L(y, \hat{y})\right] & \\ &= 1 \cdot \sum_{y' \neq \hat{y}} p(y'|x = x') + 0 \cdot p(\hat{y}|x = x') \\ &= \sum_{y' \neq \hat{y}} p(y'|x = x') \\ &= 1 - p(\hat{y}|x = x'), \end{aligned}$$

where $\hat{y} = \hat{y}(x')$. In the last step we have used the fact that $\sum_y p(y|x) = 1$. It is now sufficient to show that the optimal classification rule minimizes the conditional risk for *every* x'. Indeed, if this is true then we can sample (or otherwise choose) x' and have the guarantee that we always make the decision with the smallest loss (error).

The conditional risk $R(\hat{y}|x)$ is minimized when $p(\hat{y}(x)|x = x')$ is maximized - which finally leads to our rule for y^* in Equation (1).

Scoring: *7 points for using the hint for loss expression; 8 points for explaining why minimizing conditional risk for each x is enough; 10 for the remainder of the derivation*

Now, let us depart from the implicit assumption of cost symmetry. That is, we now assume that there is a *loss* $L(y^*, \hat{y})$ associated with each pair of our decision $\hat{y}$ and the true class label is y^*. Such loss function can be defined by the elements of the $K \times K$ *cost matrix* $\mathbf{C}$; the element c_{ij} gives the loss when $\hat{y} = i$ and the true label is $y^* = j$. We require that the elements of $\mathbf{C}$ are non-negative, but impose no further constraints. A practical example of such situation is the cost associated with doctor's decision on the patient's illness, given the symptoms (and test results). It is much more harmful to assume that the patient has common cold when in fact the patient has cholera, than vice versa.

2. [**10pt**] Show how the classification rule (1) is modified under this more general loss function.

Answer: We can follow the same idea of imaginary sampling first from $x \sim p(x)$ and then from $y \sim p(y|x)$ as in 1; we have also established that it is sufficient to minimize the conditional risk for any fixed x', that is, we must choose

$$y^*(x') = \operatorname*{argmin}_y L_y(x') = \operatorname*{argmin}_y \sum_{y'} p(y'|x') c_{y,y'}.$$

For the binary case, we can further simplify this rule: we choose $y^*(x') = 1$ if and only if

$$\begin{aligned} & L_1(x') \leq L_0(x') \\ \Leftrightarrow \; & p(y=0|x')c_{10} + p(y=1|x')c_{11} \leq p(y=0|x')c_{00} + p(y=1|x')c_{01} \\ \Leftrightarrow \; & p(0|x')\left(c_{10} - c_{00}\right) \leq p(1|x)\left(c_{01} - c_{11}\right), \end{aligned}$$

so that the optimal rule can be expressed as

$$y^*(x) = \operatorname*{argmax}_y p(y|x)\left(c_{\tilde{y},y} - c_{y,y}\right), \tag{2}$$

where $\tilde{y}$ means "not y", i.e. $\tilde{0} = 1, \tilde{1} = 0$. The modified rule (2) means that we should scale the posteriors by the cost differences between being wrong and being correct. For example, if the costs are $c_{00} = c_{11} = 0$,$c_{10} = 3$,$c_{01} = 1$, then for a given x the posterior $p(1|x)$ must be at least 3 times higher than $p(0|x)$ in order to justify classification $y(x) = 1$.

Scoring: *6 points for having the correct rule; 4 for derivation*

Problem 3: discriminative versus generative models for classification

In this problem we will investigate how generative and discriminative models might differ as classification methods.

1. [**10pt**] We consider here a two-class classification problem. Let's assume that the class-conditional distributions are bi-variate Gaussians and that the prior class frequencies are the same for the two classes. Fit the Gaussian models to the data in each of the files

   `XYsmall.dat`
   `XYtrain.dat`
   `XYLarge.dat`

   Report the classification error for the resulting generative model and the test error on the data in `XYtest.dat` for each of these 3 training sets. What do you expect the training and test error to become as the number of both training and test examples approaches infinity?

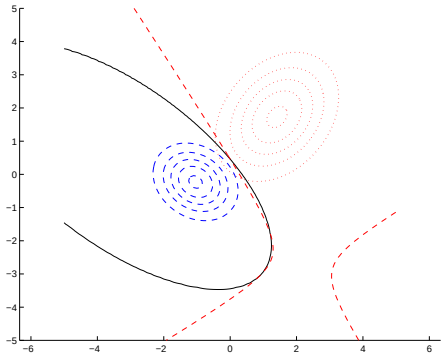
   Advice: You may find the standard Matlab functions `mean` and `cov` useful for estimating the parameters of a generative model, and the functions `multigaussian` and `plotgaussians` available on the course website, for plotting the bivariate Gaussian pdf and for using it for prediction.

   Answer: In Figure 3, the solid line shows the decision induced by the generative model, and the dashed line shows the boundary found by logistic regression. For the reference, we note again that both classes have Gaussian distributions with covariance $\mathbf{I}$ and means respectively in (-1,-1) and (1,1). The Bayes optimum decision boundary, computed from this information, is shown by the dotted line in Fig. 3(d). Of course, this is for analysis only - we can not assume that this information about the true nature of the problem is accessible to any of the learning algorithms.

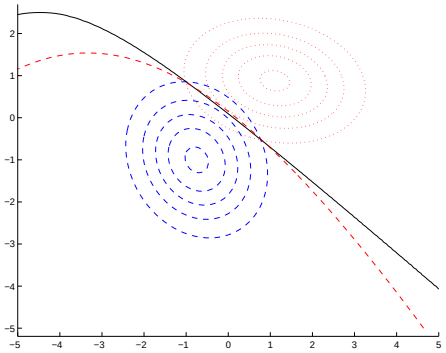
   The errors of the generative model (after fixing the bug in the function `predictgaussians`) are:

	`XYsmall`	`XYtrain`	`XYlarge`
Training	0	5%	7.1%
Test	9.2%	7.8%	7.4%

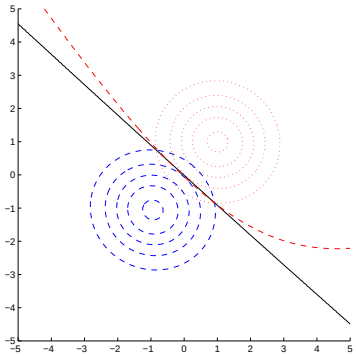
   The higher the number of samples in the training set, the closer are the estimated parameters of the Gaussians to the true parameters (graphically, it corresponds to the ellipsoidal contours estimated for the two Gaussians getting



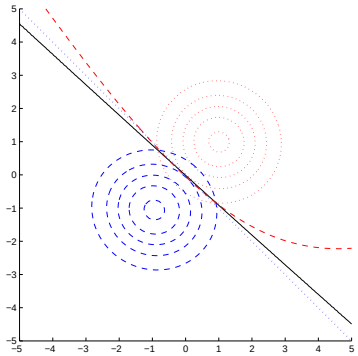
(a) `XYsmall.dat`



(b) `XYtrain.dat`



(c) `XYlarge.dat`



(d) The true class-conditionals and the optimal classifier

Figure 3: Results for problem 3. Displayed decision boundaries: solid line - generative model, dashed - logistic regression, dotted - Bayes optimal classifier

more similar to the true contours, shown in Fig. 3(d). As a result, the induced decision boundary is getting closer to to the optimal decision boundary. The test error is reduced (and comes closer to the Bayes minimal error), and the training error grows (and also, of course, comes closer to the same limit).

Scoring: *2 points for right numbers, 2 for plots; 4 for stating the boundary approaches optimal boundary; 2 for explaining effect of very few points.*

2. [**25pt**] Now, train the logistic regression model with quadratic boundary on the same training data, using the Matlab function

```
w = logisticreg(X,y)
```

which returns parameters `w` of logistic regression trained on input vectors `X` and labels `y`. The model can be tested using

```
labels = logistictest(x,y,w);
```

Note that in order to train and test quadratic logistic regression, you must transform the input to include the quadratic features. This can be done with the function `fullexpand2`, available on the website. The corresponding calls would be

```
w = logisticreg(fullexpand2(x),y);
y_logreg = logistictest(fullexpand2(x),y,w);
```

Report the training error of the logistic regression model, as well as the test error on `XYtest.dat`, for each of the same 3 training sets as in question 1. For each of the training sets, turn in the plots, on the same figure, of the decision boundaries corresponding to logistic regression and the corresponding generative model from question 1 (functions `plotgaussians` and `plotlogreg`, available from the website, will be useful here). Explain the differences in the boundaries on each plot, and the differences between plots. What would you expect top see on a plot corresponding to an infinite number of training examples provided to both models?

Answer: The error obtained by logistic regression:

	`XYsmall`	`XYtrain`	`XYlarge`
Training	0	6%	6.9%
Test	8.5%	7.8%	7.4%

The dynamics of the logistic regression performance is, not surprisingly, similar to that of the generative model - as the training set increases, the generalization error (as estimated by the test error) gets closer to the Bayes

error. For a very small data set, logistic regression shows a bit better performance - the generative model is too much off due to poor estimation of the Gaussian parameters (note the two different topologies of the decision regions, corresponding to two quadratic boundaries in Fig. 3(a)). However, with large enough training set the logistic regression loses the advantage, and eventually leads to the decision boundary which is farther from the optimal one (as is evident from Fig. 3(d)).

It is important to remember that although the decision boundary for logistic regression in this case "looks" much worse, the test error on a fairly large data set is very similar to that of the generative model - this is because the regions in which the decision boundaries are far correspond to very low density of examples; the error in posterior estimated by the log. regression there is large, but it is weighted by a low probability within the expected error.

If an unlimited (infinite) number of training examples is available, one can estimate the parameters of the generative model as close as desired to the true values; therefore, the decision boundary will approach the Bayes optimum decision boundary for the true parameters; for our two Gaussians, that is the line $x_2 = -x_1$. As for the discriminative model (logistic regression), its evaluation of the posteriors also will become as close as desired to the true posterior in each point, therefore bringing the decision boundary to the same Bayes optimum boundary. In particular, we would expect the quadratic terms in both boundaries to vanish.

Scoring: *2 points for error values, 2 for plots; 3 for explaining similar test performance despite the difference in boundaries; 15 for explaining that both models' boundaries approach the optimal boundary; 5 for other correct observations.*