EE538 Homework 3

Quang Minh Nguyen (20200854)

Monday 5th April, 2021

Contents

1 Bayes classifier and perceptron

We start with Gaussian distributions. The subscripts $1, 2$ to be used are associated with regions/classes A and B, respectively. Consider

$$p_{\mathbf{X}|C_1}(\mathbf{x}) = \frac{1}{2\pi\sigma_1^2} \exp\left(-\frac{(\mathbf{x} - (1\ 2)^\top)^\top(\mathbf{x} - (1\ 2)^\top)}{2\sigma_1^2}\right), \tag{1.1}$$

$$p_{\mathbf{X}|C_2}(\mathbf{x}) = \frac{1}{2\pi\sigma_2^2} \exp\left(-\frac{\mathbf{x}^\top\mathbf{x}}{2\sigma_1^2}\right), \tag{1.2}$$

$$p_1 = \frac{N_1}{N_1 + N_2}, \quad p_2 = \frac{N_2}{N_1 + N_2}, \tag{1.3}$$

$$c_{11} = c_{22} = 0, \quad c_{12} = c_{21} > 0, \tag{1.4}$$

where $\mathbf{x} = (x\ y)^\top$.

Then

$$\log\Lambda(\mathbf{x}) = 2\log\frac{\sigma_2}{\sigma_1} + \left(\frac{1}{2\sigma_2^2} - \frac{1}{2\sigma_1^2}\right)\mathbf{x}^\top\mathbf{x} + (\frac{1}{\sigma_1^2}\ \frac{2}{\sigma_1^2})\mathbf{x} - \frac{5}{2\sigma_1^2}, \tag{1.5}$$

$$\log\xi = \log\frac{N_2}{N_1}. \tag{1.6}$$

So the Bayes classifier reduces to a quadratic classifier with decision boundary

$$\left(\frac{1}{2\sigma_2^2}-\frac{1}{2\sigma_1^2}\right)\mathbf{x}^\top\mathbf{x}+(\frac{1}{\sigma_1^2}\ \frac{2}{\sigma_1^2})\mathbf{x}+\log\frac{\sigma_2^2N_1}{\sigma_1^2N_2}-\frac{5}{2\sigma_1^2}=0\,. \tag{1.7}$$

Note that the quadratic form vanishes and we get a linear classifier if $\sigma_1=\sigma_2$.

We are given the special cases

(a) $\sigma_1=\sigma_2=1, N_1=N_2=1000$. The decision boundary is

$$(1\ 2)\mathbf{x}-\frac{5}{2}=0\,, \tag{1.8}$$

i.e.,

$$x+2y-\frac{5}{2}=0\,. \tag{1.9}$$

(c) $\sigma_1=1,\sigma_2=\sqrt{2}, N_1=N_2=1000$. The decision boundary is

$$-\frac{1}{4}\mathbf{x}^\top\mathbf{x}+(1\ 2)\mathbf{x}+\log 2-\frac{5}{2}=0\,, \tag{1.10}$$

i.e.,

$$(x-2)^2+(y-4)^2=4\log 2+10\,. \tag{1.11}$$

(e) $\sigma_1=1,\sigma_2=\sqrt{2}, N_1=700, N_2=1400$. The decision boundary is

$$-\frac{1}{4}\mathbf{x}^\top\mathbf{x}+(1\ 2)\mathbf{x}-\frac{5}{2}=0\,, \tag{1.12}$$

i.e.,

$$(x-2)^2+(y-4)^2=10\,. \tag{1.13}$$

The plots for parts (a)-(f) are included in Figures 1, 2, and 3. We use a variant of the perceptron learning algorithm, which stores the sum of perceptron weights at each time step then uses this sum to predict. Since the prediction of a weight w is the same as its multiple aw, we can call this the *average perceptron*. Also, besides the N_1+N_2 data points for training, we generate additionally $3(N_1+N_2)$ data points for testing. The number at each perceptron decision boundary plot is the accuracy on the test set. All codes can be found in the files `learn.py` and `generate.py`.

Now we will work with uniform distributions for part (g). Let us denote two regions

$$A=\{\mathbf{x}|(x-1)^2+(y-2)^2<2\sigma_1^2\}\,, \tag{1.14}$$

$$B=\{\mathbf{x}|x^2+y^2<2\sigma_2^2\}\,. \tag{1.15}$$

The distributions, prior, and costs are

$$p_{\mathbf{X}|C_1}(\mathbf{x}) = \begin{cases} \frac{1}{2\pi\sigma_1^2}\,, & \text{if } \mathbf{x} \in A \\ 0\,, & \text{otherwise} \end{cases}\,, \tag{1.16}$$

$$p_{\mathbf{X}|C_2}(\mathbf{x}) = \begin{cases} \frac{1}{2\pi\sigma_2^2}\,, & \text{if } \mathbf{x} \in B \\ 0\,, & \text{otherwise} \end{cases}\,, \tag{1.17}$$

$$p_1 = \frac{N_1}{N_1 + N_2}\,, \quad p_2 = \frac{N_2}{N_1 + N_2}\,, \tag{1.18}$$

$$c_{11} = c_{22} = 0\,, \quad c_{21} = c_{12} > 0\,. \tag{1.19}$$

Here we cannot take the logarithm or compute the likelihood ratio because the probability density functions can be zero at some values of x and y. However, we can use the criteria

$$(c_{21} - c_{11})p_1 p_{\mathbf{X}|C_1}(\mathbf{x}) > (c_{12} - c_{21})p_2 p_{\mathbf{X}|C_2}(\mathbf{x}) \tag{1.20}$$

to decide if a point $\mathbf{x}$ is in class A. With $c_{11} = c_{22} = 0$ and $c_{12} = c_{21} > 0$ as we defined, equation 1.20 is equivalent to

$$p_1 p_{\mathbf{X}|C_1}(\mathbf{x}) > p_2 p_{\mathbf{X}|C_2}(\mathbf{x})\,. \tag{1.21}$$

Further substituting p_1 and p_2 by the expressions at 1.18 and the probability density functions by formulas at 1.16 and 1.17, we obtain three cases

1. A point $\mathbf{x} \in A \cap B$ is decided to be in class A if

$$\frac{N_1}{\sigma_1^2} > \frac{N_2}{\sigma_2^2}\,. \tag{1.22}$$

2. A point $\mathbf{x} \in A \setminus B$ is decided to be in class A if

$$\frac{N_1}{\sigma_1^2} > 0\,, \tag{1.23}$$

   i.e., for all $\mathbf{x}$.

3. A point $Bx \in B \setminus A$ is decided to be in class A if

$$0 > \frac{N_2}{\sigma_2^2}\,, \tag{1.24}$$

   i.e., for no $\mathbf{x}$.

We did not consider the case $\mathbf{x} \in (A \cup B)^c$ because such data appears in neither the training set nor the test set.

The special cases of parameters we are given are

(ga) $\sigma_1 = \sigma_2 = 1, N_1 = N_2 = 1000$. This case does not satisfy 1.22 and 1.24. So the decided region for class A is $A \setminus B$. The decision boundary is the boundary of $A \setminus B$ itself.

(gc) $\sigma_1 = 1, \sigma_2 = \sqrt{2}, N_1 = N_2 = 1000$. For this case, 1.22 and 1.23 are satisfied. So the region for class A is A. The decision boundary is the circle

$$(x-1)^2 + (y-2)^2 = 2\,. \tag{1.25}$$

(ge) $\sigma_1 = 1, \sigma_2 = \sqrt{2}, N_1 = 700, N_2 = 1400$. This case is similar to (fa), with the boundary of $A \setminus B$ as decision boundary.

The plots for part (g) can be found in Figures 4, 5, and 6. The perceptron algorithm used is the same as parts (a)-(f). The codes are all in the files `learn.py` and `generate.py`.

When the data is Gaussian, different classes become more mixed with each other, further enhancing the linear non-separability. Both the linear perceptron and the Bayes classifier, even with quadratic separability, performs badly with this kind of data and cannot capture major patterns.

Numerically, we see that the accuracy of part (g) is significantly higher than that of parts (a)-(f). This might have come from the fact that the data is now more locally and densely distributed, which makes classification more straightforward. Particularly, Bayes classifier has the ability to capture exactly the circular patterns that we intuitively would recognize.

2 Derivation of the backpropagation algorithm

Let us first introduce the notation.

- The network contains 2 hidden layers. There are N input neurons, K and L hidden neurons, and M output neurons.
- We index these layers from 0 to 3, in the same order. These indices are used as superscripts.
- A weight connecting the ith neuron of layer $(j-1)$ with the i' neuron of layer (j) is denoted by $w_{i'i}^{(j)}$. Here $w_{i'0}^{(j)}$ is the bias term between two layers.
- The induced local field at neuron i of layer (j) is

$$v_i^{(j)} = \sum_k w_{ik}^{(j)} y_k^{(j-1)}\,, \tag{2.1}$$

where each $y_k^{(j-1)}$ is a function signal at neuron k of layer $(j-1)$. Note that $y_k^{(0)} = x_k$, the input, and $y_0^{(j-1)}$ is always 1 (to multiply with the bias term).

- The activation function at neuron i of layer (j) is $\phi_m^{(3)}(.)$.
- For simplicity, we consider online learning where the cost function is the total instantaneous error energy

$$E = \frac{1}{2}\sum_{j=1}^{M} e_j^2 \,. \tag{2.2}$$

Here $e_j^2 = d_j - y_j^{(3)}$ is the error signal, i.e., the difference between desired output d_j and computed output $y_j^{(3)}$. Since online learning involves one stimulus at a time, we dropped the time step index for cleaner equations. For the case of batch/mini-batch learning, we perform the same gradient calculation for each stimulus and then take average to get new learning rules.

(a) The first approach is through chain rule. We first derive the gradient of E with respect to all weights and biases then describe the algorithm later.

Right before the output layer,

$$\begin{aligned}\frac{\partial E}{\partial w_{ml}^{(3)}} &= \frac{\partial v_m^{(3)}}{\partial w_{ml}^{(3)}}\frac{\partial y_m^{(3)}}{\partial v_m^{(3)}}\frac{\partial E}{\partial y_m^{(3)}} \\ &= y_l^{(2)}\phi_m'^{(3)}(v_m^{(3)})(-e_m)\,.\end{aligned} \tag{2.3}$$

Then, before layer (2),

$$\begin{aligned}\frac{\partial E}{\partial w_{lk}^{(2)}} &= \frac{\partial v_l^{(2)}}{\partial w_{lk}^{(2)}}\frac{\partial y_l^{(2)}}{\partial v_l^{(2)}}\frac{\partial E}{\partial y_l^{(2)}} \\ &= y_k^{(1)}\phi_l'^{(2)}(v_l^{(2)})\sum_m \frac{\partial E}{\partial y_m^{(3)}}\frac{\partial y_m^{(3)}}{\partial y_l^{(2)}} \\ &= y_k^{(1)}\phi_l'^{(2)}(v_l^{(2)})\sum_m (-e_m)w_{ml}^{(3)}\phi_m'^{(3)}(v_m^{(3)})\,.\end{aligned} \tag{2.4}$$

Finally, before layer (1),

$$\begin{aligned}\frac{\partial E}{\partial w_{kn}^{(1)}} &= \frac{\partial v_k^{(1)}}{\partial w_{kn}^{(1)}}\frac{\partial y_k^{(1)}}{\partial v_k^{(1)}}\frac{\partial E}{\partial y_k^{(1)}} \\ &= x_n\phi_k'^{(1)}(v_k^{(1)})\sum_l \frac{\partial E}{\partial y_l^{(2)}}\frac{\partial y_l^{(2)}}{\partial y_k^{(1)}} \\ &= x_n\phi_k'^{(1)}(v_k^{(1)})\sum_{l,m}(-e_m)w_{ml}^{(3)}\phi_m'^{(3)}(v_m^{(3)})w_{lk}^{(2)}\phi_l'^{(2)}(v_l^{(2)})\,.\end{aligned} \tag{2.5}$$

So the backpropagation algorithm can be written in a recursive fashion as follows:

1. Initialize weights and biases.

2. Repeat for each input in each epoch until convergence:

   (a) **Forward computation**: for each layer (j) from (1) and each neuron i, compute

$$v_i^{(j)} = \sum_k w_k^{(j)} y_k^{(j-1)} , \tag{2.6}$$

$$y_i^{(j)} = \phi_i^{(j)}(v^{(j)_i}) . \tag{2.7}$$

   (b) For each output $y_m^{(3)}$, compute error signal

$$e_m = d_m - y_m^{(3)} \tag{2.8}$$

   and

$$\frac{\partial E}{\partial y_m^{(3)}} = -e_m . \tag{2.9}$$

   (c) **Backward computation**: for each layer (j) from (3) to (1), compute

$$\frac{\partial E}{\partial w_{ab}^{(j)}} = y_b^{(j-1)} \phi'^{(j)}(v_a^{(j)}) \frac{\partial E}{\partial y_a^{(j)}} , \tag{2.10}$$

   where

$$\frac{\partial E}{\partial y_a^{(j)}} = \sum_c \frac{\partial E}{\partial y_c^{(j+1)}} \frac{\partial y_c^{(j+1)}}{\partial y_a^{(j)}} \tag{2.11}$$

   if $j \neq 3$.

   (d) Update weights and biases by the rule

$$\Delta w_{ab}^{(j)} = \alpha w_{ab}^{(j)} - \eta \frac{\partial E}{\partial w_{ab}^{(j)}} , \tag{2.12}$$

   where α is the momentum constant and η is the learning rate.

To adapt this algorithm to batch learning, we simply repeat parts 2(a)-2(c) for every input in the batch and average the matrices $\left[\frac{\partial E}{\partial w_{ab}^{(j)}}\right]$ to get the gradient of the empirical risk. Then the weight update part is the same, just with a modification $\left[\frac{\partial E}{\partial w_{ab}^{(j)}}\right] \to$ gradient of empirical risk.

(b) Since the full algorithm has been discussed in part (a), now we just derive the partial derivatives with sensitivity analysis.

1. Suppose we have a small pertubation $\Delta w_{ml}^{(3)}$. Then the function signal of layer (3) accordingly has the pertubation

$$\Delta y_m^{(3)} = \phi_m'^{(3)}(v_m^{(3)}) y_l^{(2)} \Delta w_{ml}^{(3)} . \tag{2.13}$$

   Therefore the error is changed by

$$\begin{aligned} \Delta E &= -(d_m - y_m^{(3)}) \Delta y_m^{(3)} \\ &= -(d_m - y_m^{(3)}) \phi_m'^{(3)}(v_m^{(3)}) y_l^{(2)} \Delta w_{ml}^{(3)} . \end{aligned} \tag{2.14}$$

2. Next, we examine the effect of $\Delta w_{lk}^{(2)}$:

$$\begin{aligned} & \Delta w_{lk}^{(2)} \\ \rightarrow \Delta y_l^{(2)} &= \phi_l'^{(2)}(v_l^{(2)}) y_k^{(1)} \Delta w_{lk}^{(2)} \\ \rightarrow \Delta y_m^{(3)} &= \phi_m'^{(3)}(v_m^{(3)}) w_{ml}^{(3)} \phi_l'^{(2)}(v_l^{(2)}) y_k^{(1)} \Delta w_{lk}^{(2)} \forall m \\ \rightarrow \Delta E &= -\sum_m (d_m - y_m^{(3)}) \phi_m'^{(3)}(v_m^{(3)}) w_{ml}^{(3)} \phi_l'^{(2)}(v_l^{(2)}) y_k^{(1)} \Delta w_{lk}^{(2)} \,. \end{aligned} \tag{2.15}$$

3. Finally,

$$\begin{aligned} & \Delta w_{kn}^{(1)} \\ \rightarrow \Delta y_k^{(1)} &= \phi_k'^{(1)}(v_k^{(1)}) x_n \Delta w_{kn}^{(1)} \\ \rightarrow \Delta y_l^{(2)} &= \phi_l'^{(2)}(v_l^{(2)}) w_{lk}^{(2)} \phi_k'^{(1)}(v_k^{(1)}) x_n \Delta w_{kn}^{(1)} \forall l \\ \rightarrow \Delta y_m^{(3)} &= \sum_l \phi_m'^{(3)}(v_m^{(3)}) w_{ml}^{(3)} y_l^{(2)} \\ \rightarrow \Delta E &= -\sum_{m,l} (d_m - y_m^{(3)}) \Delta y_m^{(3)} \,. \end{aligned} \tag{2.16}$$

We recognize that 2.16, 2.15, and 2.14 are essentially identical to 2.3, 2.4, and 2.5. So we can equivalently derive the backpropagation algorithm from sensitivity analysis.

A Sampling method for Gaussian distribution

In problem 1, we sampled the Gaussian distribution of two independent random variables x and y using an approximation and rejection strategy.

Denote the mean by $\mu = (\mu_1 \ \mu_2)^\top$ and the variance of x and y by σ_x and σ_2, respectively. Most of the distribution lies within the rectangle of width $8\sigma_x$ and height $8\sigma_y$ centered at μ. So we first make an approximation that all sampled data will be in this region.

Next, note that the maximum value of the density function is $\max f = (2\pi\sigma_x\sigma_y)^{-1}$. We sample a uniformly random vector (x, y, z) where $x \in [\mu_1 - 4\sigma_x, \mu_1 + 4\sigma_x]$, $y \in [\mu_2 - 4\sigma_y, \mu_2 + 4\sigma_y]$, and $z \in [0, \max f]$. If $z < f(x, y)$ then we admit (x, y) as a sample of the Gaussian distribution. This process is repeated until we reach the desired number of samples.

The code for this is in `generate.py`.

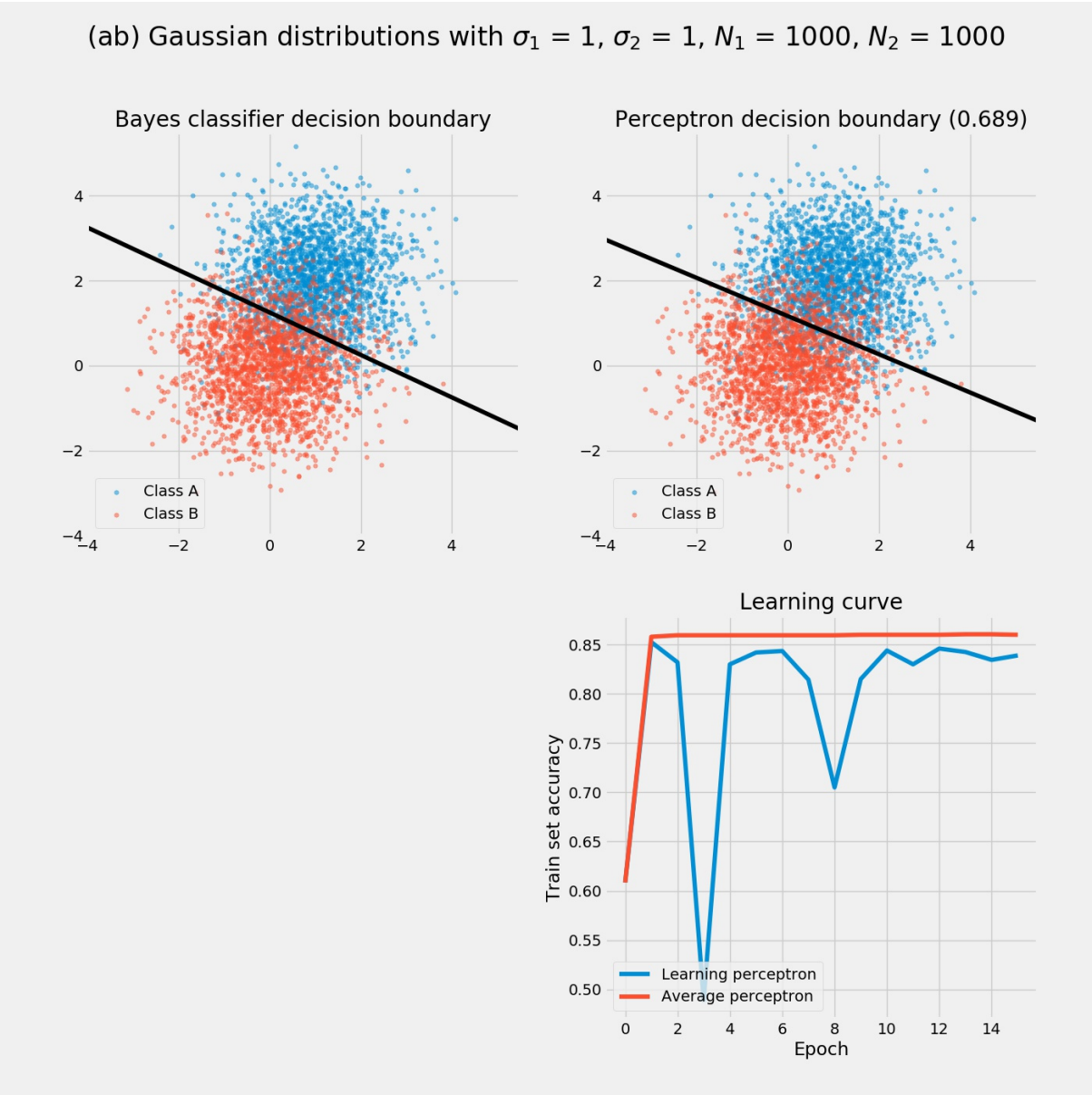

Figure 1: Problem 1a and 1b—Gaussian distributions with $\sigma_1 = \sigma_2 = 1$ and $N_1 = N_2 = 1$

.

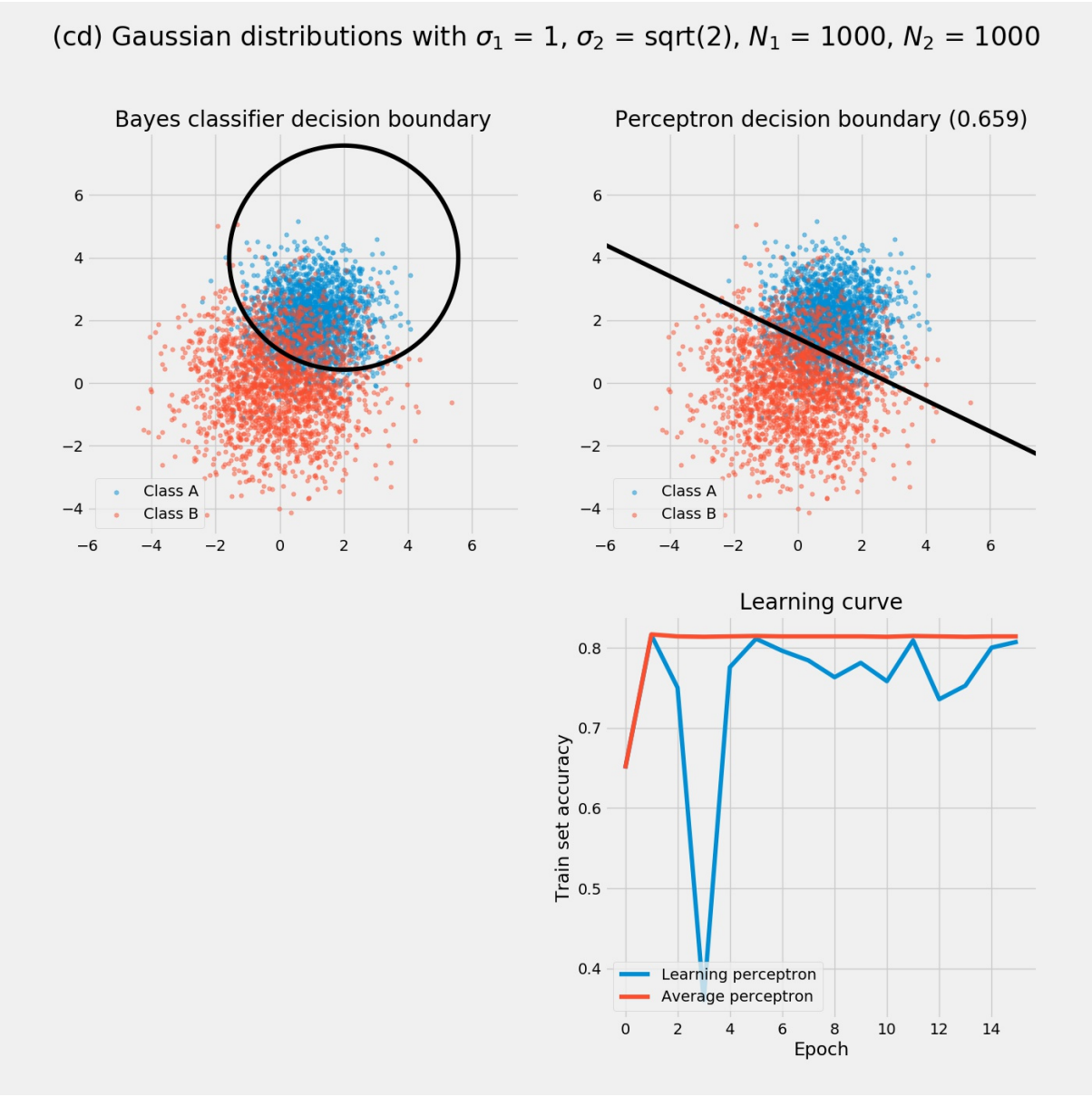

Figure 2: Problem 1c and 1d—Gaussian distributions with $\sigma_1 = 1$, $\sigma_2 = \sqrt{2}$, and $N_1 = N_2 = 1000$

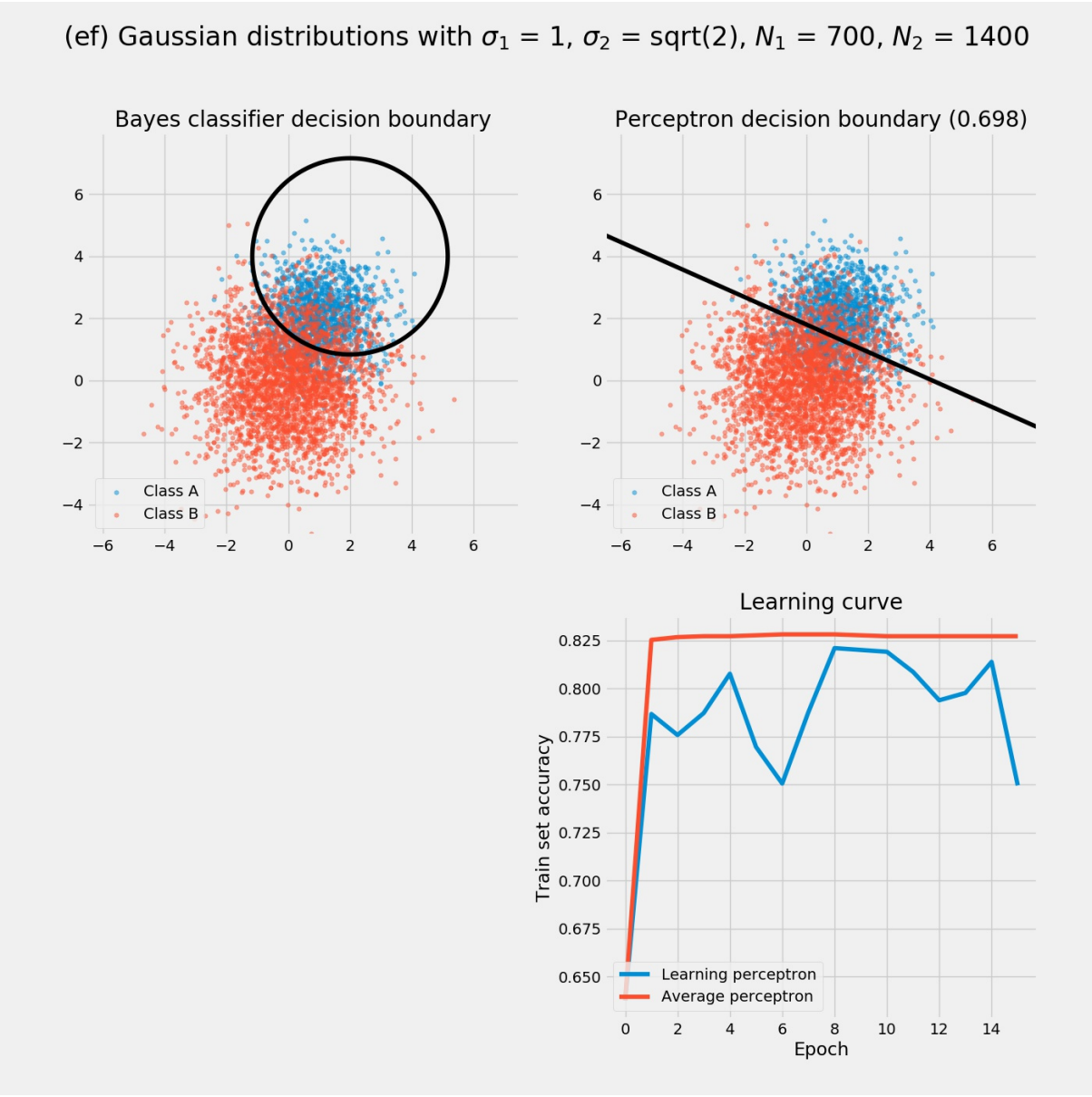

Figure 3: Problem 1e and 1f—Gaussian distributions with $\sigma_1 = 1$, $\sigma_2 = \sqrt{2}$, $N_1 = 700$ and $N_2 = 1400$

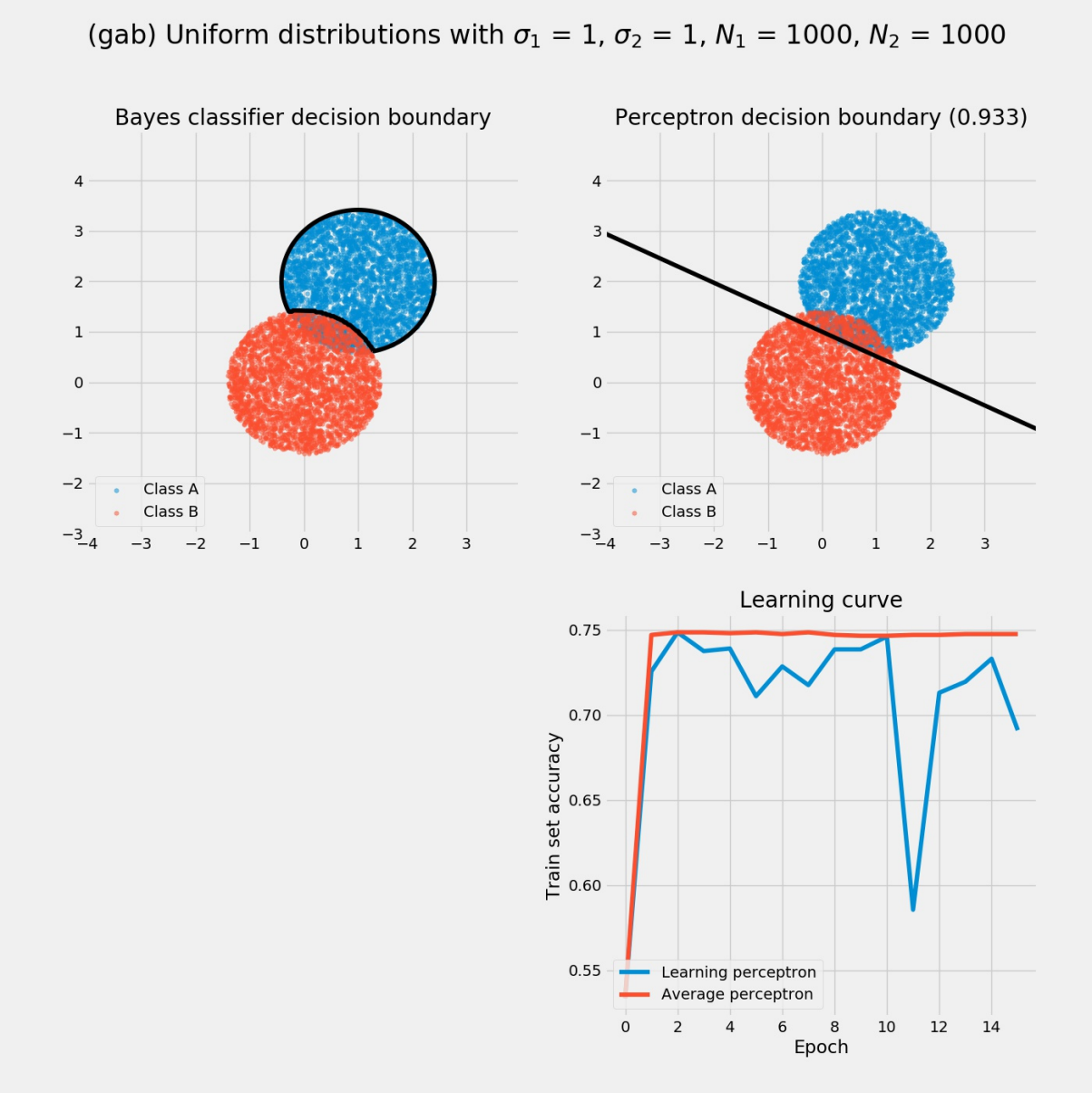

Figure 4: Problem 1ga and 1gb—Uniform distributions with $\sigma_1 = \sigma_2 = 1$ and $N_1 = N_2 = 1$

.

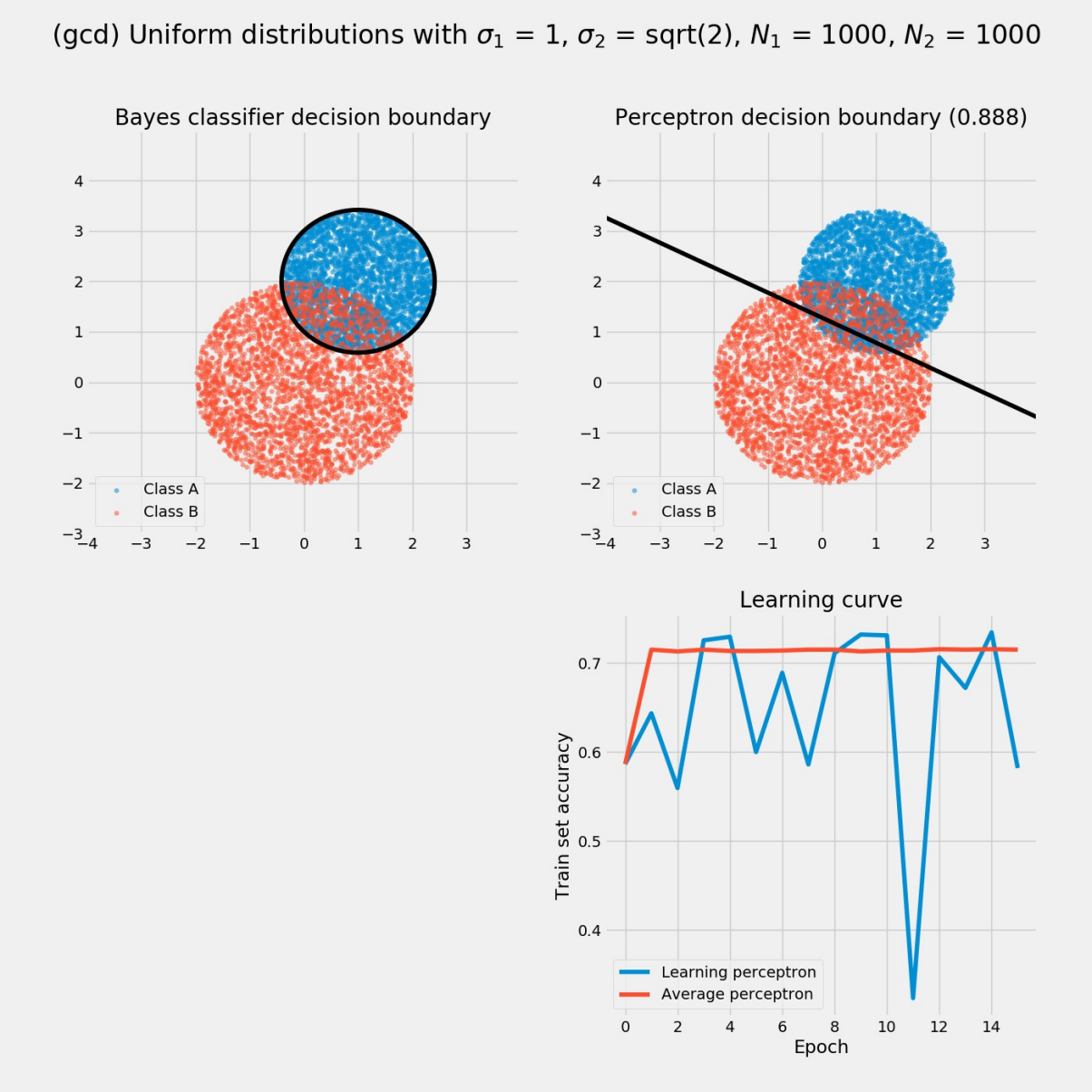

Figure 5: Problem 1gc and 1gd—Uniform distributions with $\sigma_1 = 1$, $\sigma_2 = \sqrt{2}$, and $N_1 = N_2 = 1000$

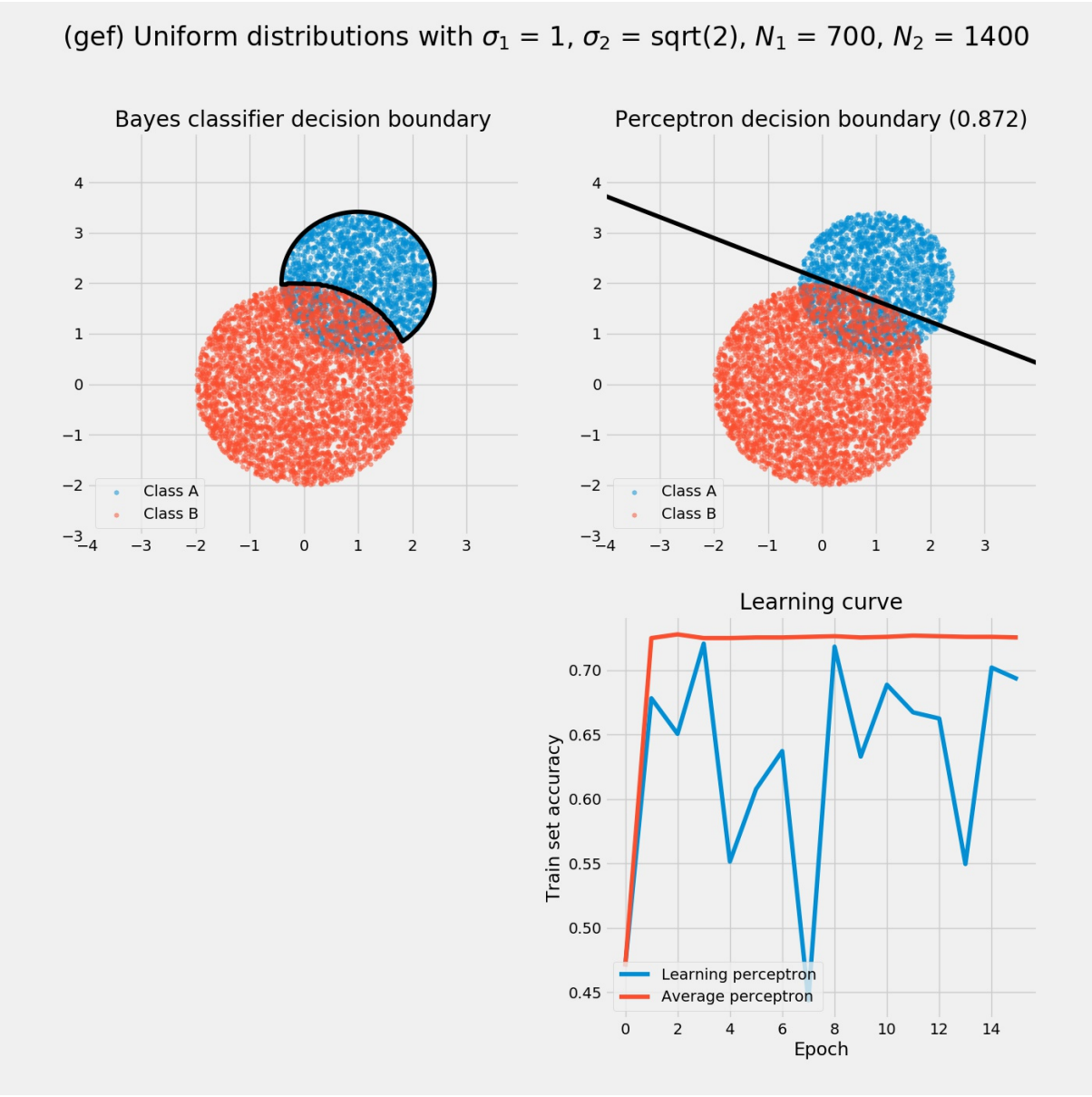

Figure 6: Problem 1ge and 1gf—Uniform distributions with $\sigma_1 = 1$, $\sigma_2 = \sqrt{2}$, $N_1 = 700$ and $N_2 = 1400$