Programming Assignment 1

CS 474

`https://github.com/alexander-novo/CS479-PA1`

Nikhil Khedekar
–% Work

Mehar Mangat
–% Work

Alexander Novotny
–% Work

Due: March 10, 2021
Submitted: March 10, 2021

Contents

1 Parts 1 & 2

1.1 Theory

1.1.1 Generating arbitrary multivariate Gaussian-distributed random vectors

While generating random vectors distributed as a standard normal distribution is simple enough (using a Box-Muller transform or the quantile function), we're still left with transforming it to an arbitrary Gaussian distribution. The key to being able to do this is through the Whitening transform, defined as

$$A_w^\top = \left(\Phi\Lambda^{-1/2}\right)^\top, \tag{1}$$

where $\Sigma = \Phi\Lambda\Phi^{-1}$ is an eigendecomposition (Φ is an orthonormal matrix of eigenvectors of Σ and Λ is a diagonal matrix of eigenvalues of Σ) of the covariance matrix Σ of a multivariate Gaussian distribution. Since Σ is a real symmetric matrix, such an eigendecomposition always exists. Applying this whitening transform to $\vec{X} \sim \mathcal{N}(\vec{0}, \Sigma)$ as

$$\vec{Y} = A_w^\top \vec{X} \tag{2}$$

yields $\vec{Y} \sim \mathcal{N}(\vec{0}, A_w^\top \Sigma A_w)$ and

$$\begin{aligned}
A_w^\top \Sigma A_w &= \left(\Phi\Lambda^{-1/2}\right)^\top \left(\Phi\Lambda\Phi^{-1}\right)\left(\Phi\Lambda^{-1/2}\right) \\
&= \left(\Lambda^{-1/2}\right)^\top \Phi^\top \Phi\Lambda\Lambda^{-1/2} \\
&= \Lambda^{-1/2}\Phi^{-1}\Phi\Lambda^{1/2} \\
&= \Lambda^0 \\
&= I,
\end{aligned}$$

so $Y \sim \mathcal{N}(\vec{0}, I)$, the standard Gaussian distribution. Then by multiplying both sides of eq. (2) by the inverse transformation

$$\begin{aligned}
\left(A_w^\top\right)^{-1} &= \left(\left(\Phi\Lambda^{-1/2}\right)^\top\right)^{-1} \\
&= \left(\left(\Phi\Lambda^{-1/2}\right)^{-1}\right)^\top \\
&= \left(\Lambda^{1/2}\Phi^{-1}\right)^\top \\
&= \Phi\Lambda^{1/2} \\
&= \Phi\Lambda^{1/2}\Phi^{-1}\Phi \\
&= \Sigma^{1/2}\Phi, \qquad (3)
\end{aligned}$$

we arrive at

$$\vec{X} = \left(A_w^\top\right)^{-1}\vec{Y}. \tag{4}$$

Then we can simply add $\vec{\mu}$ to obtain $\vec{Z} = \vec{X} + \vec{\mu}$ where $\vec{Z} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ is our "goal" distribution.

1.1.2 Bayesian classifier for Gaussian-distributed classes

When designing a classifier, we want to pick discriminant functions for each class that indicate when we should pick that class, i.e. we should pick class ω_i if $g_i(\vec{X}) \geq g_j(\vec{X})$ for all j. For a Bayesian classifier, these discriminant functions should be the probability that $\vec{X}$ came from the class, but any function of the discriminant that preserves this inequality (monotonic increasing functions) will also work as a discriminant. So, in the general case (referred to as "case 3" in the textbook) of a Bayesian classifier with multivariate Gaussian distribution, we can have (for Y the class of $\vec{X}$)

$$\begin{aligned}
g_i(\vec{x}) &= \ln\Big(P(Y=\omega_i \mid \vec{X}=\vec{x})P(\vec{X}=\vec{x})\Big)\\
&= \ln\left(\left(\frac{P(\vec{X}=\vec{x}\mid Y=\omega_i)P(Y=\omega_i)}{P(\vec{X}=\vec{x})}\right)P(\vec{X}=\vec{x})\right)\\
&= \ln\Big(P(\vec{X}=\vec{x}\mid Y=\omega_i)\Big)+\ln(P(Y=\omega_i))\\
&= \ln\left(\frac{1}{(2\pi)^{1/d}|\Sigma_i|^{1/2}}\exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)\right)\right)+\ln(P(Y=\omega_i))\\
&= -\frac{1}{d}\ln(2\pi)-\frac{1}{2}\ln|\Sigma_i|-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)+\ln(P(Y=\omega_i)),
\end{aligned}$$

which we can further refine by subtracting the constant $-\frac{1}{d}\ln(2\pi)$:

$$g_i(\vec{x}) = -\frac{1}{2}\ln|\Sigma_i|-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)+\ln(P(Y=\omega_i)). \tag{5}$$

By further narrowing down certain cases, we can simplify this discriminant by removing other terms which become constant across classes. For instance, when $\Sigma_i=\Sigma$ (referred to as "case 2" in the textbook), then the $-\frac{1}{2}\ln|\Sigma_i|$ term becomes constant and we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma^{-1}(\vec{x}-\vec{\mu}_i)+\ln(P(Y=\omega_i)) \qquad (6)\\
&= -\frac{1}{2}(\vec{x}^\top-\vec{\mu}_i^\top)(\Sigma^{-1}\vec{x}-\Sigma^{-1}\vec{\mu}_i)+\ln(P(Y=\omega_i))\\
&= -\frac{1}{2}(\vec{x}^\top(\Sigma^{-1}\vec{x}-\Sigma^{-1}\vec{\mu}_i)-\vec{\mu}_i^\top(\Sigma^{-1}\vec{x}-\Sigma^{-1}\vec{\mu}_i))+\ln(P(Y=\omega_i))\\
&= -\frac{1}{2}(\cancelto{const}{\vec{x}^\top\Sigma^{-1}\vec{x}}-\vec{x}^\top\Sigma^{-1}\vec{\mu}_i-\vec{\mu}_i^\top\Sigma^{-1}\vec{x}+\vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i)+\ln(P(Y=\omega_i))\\
&= -\frac{1}{2}(-((\Sigma^{-1}\vec{\mu}_i)^\top\vec{x})^\top-(\Sigma^{-\top}\vec{\mu}_i)^\top\vec{x}+\vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i)+\ln(P(Y=\omega_i))\\
&= -\frac{1}{2}(-(\Sigma^{-1}\vec{\mu}_i)^\top\vec{x}-(\Sigma^{-1}\vec{\mu}_i)^\top\vec{x}+\vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i)+\ln(P(Y=\omega_i))
\end{aligned}$$

(since $(\Sigma^{-1}\vec{\mu}_i)^\top\vec{x}$ is a scalar, and Σ,Σ^{-1} are symmetric)

$$= (\Sigma^{-1}\vec{\mu}_i)^\top\vec{x}-\frac{1}{2}\vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i+\ln(P(Y=\omega_i)). \tag{7}$$

Furthermore, if Σ is a scalar matrix $\Sigma=\sigma^2 I$ (referred to as "case 1" in the book), then $\Sigma^{-1}=\frac{1}{\sigma^2}$ and from eq. (6) above, we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2\sigma^2}(\vec{x}-\vec{\mu}_i)^\top(\vec{x}-\vec{\mu}_i)+\ln(P(Y=\omega_i))\\
&= -\frac{||\vec{x}-\vec{\mu}||}{2\sigma^2}+\ln(P(Y=\omega_i)). \qquad (8)
\end{aligned}$$

Further simplification can be found in section 2.1.

1.1.3 Finding the decision boundary

It is of interest to find the boundary along which the classifier changes its decision between classes. It is a defined as

$$B = \{\vec{x} \mid g_i(\vec{x}) = g_j(\vec{x}) \geq g_k(\vec{x}) \text{ for some } i \neq j \text{ and all } k\}, \tag{9}$$

and for classifiers which are classifying between only two classes (such as the ones in this assignment), the inequality is trivially satisfied. Therefore, we're interesting in finding the roots $\vec{x}$ such that $g_i(\vec{x}) - g_j(\vec{y}) = 0$. Taking eq. (5) above (the most general case) and reorganizing it in a similar way in eq. (7) to separate out $\vec{x}$ more cleanly, we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2}(\vec{x} - \vec{\mu}_i)^\top \Sigma_i^{-1}(\vec{x} - \vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top - \vec{\mu}_i^\top)(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i) - \vec{\mu}_i^\top(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i)) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top\Sigma_i^{-1}\vec{x} - \vec{x}^\top\Sigma_i^{-1}\vec{\mu}_i - \vec{\mu}_i^\top\Sigma_i^{-1}\vec{x} + \vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}\vec{x}^\top\Sigma_i^{-1}\vec{x} + (\Sigma_i^{-1}\vec{\mu}_i)^\top\vec{x} - \frac{1}{2}\vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)),
\end{aligned}$$

which gives us

$$g_i(\vec{x}) - g_j(\vec{x}) = \vec{x}^\top W_{ij}\vec{x} + \vec{w}_{ij}^\top\vec{x} + w_{0_{ij}} = 0, \tag{10}$$

$$W_{ij} = -\frac{1}{2}\left(\Sigma_i^{-1} - \Sigma_j^{-1}\right), \tag{11}$$

$$\vec{w}_{ij} = \Sigma_i^{-1}\vec{\mu}_i - \Sigma_j^{-1}\vec{\mu}_j, \tag{12}$$

$$\begin{aligned}
w_{0_{ij}} = &\left(-\frac{1}{2}\vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i))\right) - \\
&\left(-\frac{1}{2}\vec{\mu}_j^\top\Sigma_j^{-1}\vec{\mu}_j - \frac{1}{2}\ln|\Sigma_j| + \ln(P(Y = \omega_j))\right).
\end{aligned} \tag{13}$$

This is in the general form of a quadric, which in two dimensions looks like

$$ax^2 + bx + cy^2 + dx + ey + f = 0,$$

where

$$W_{ij} = \begin{pmatrix} a & \frac{1}{2}b \\ \frac{1}{2}b & c \end{pmatrix}, \tag{14}$$

$$\vec{w}_{ij} = \begin{bmatrix} d \\ e \end{bmatrix}, \tag{15}$$

$$w_{0_{ij}} = f. \tag{16}$$

1.1.4 The derivative of the error bound function

When trying to minimize the error bound function for 2-class multivariate Gaussian Bayesian classifiers,

$$\begin{aligned}
f(\beta) &= (P(Y = \omega_1))^\beta (P(Y = \omega_2))^{1-\beta}\exp(-k(\beta)), \\
k(\beta) &= \frac{\beta(1-\beta)}{2}(\vec{\mu}_1 - \vec{\mu}_2)^\top[(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(\vec{\mu}_1 - \vec{\mu}_2) + \frac{1}{2}\ln\left(\frac{|(1-\beta)\Sigma_1 + \beta\Sigma_2|}{|\Sigma_1|^{1-\beta}|\Sigma_2|^\beta}\right),
\end{aligned} \tag{17}$$

is it useful to know the derivative, $f'(\beta)$. However, since this involves many chain rules, product rules, quotient rules, and matrix derivative rules, we should first break the derivatives up into parts. Note that

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[a^{f(\beta)}\right] &= \frac{\mathrm{d}}{\mathrm{d}\beta}\left[\exp\left(\ln\left(a^{f(\beta)}\right)\right)\right] \\ &= \frac{\mathrm{d}}{\mathrm{d}\beta}[\exp(f(\beta)\ln(a))] \\ &= \exp(f(\beta)\ln(a))\frac{\mathrm{d}}{\mathrm{d}\beta}[f(\beta)\ln(a)] \\ &= a^{f(\beta)}\left(\ln(a)\frac{\mathrm{d}f}{\mathrm{d}\beta}\right), \qquad (18)\end{aligned}$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}\left[[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\right] &= -[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\frac{\mathrm{d}}{\mathrm{d}\beta}[(1-\beta)\Sigma_1+\beta\Sigma_2][(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1} \\ &= -[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(-\Sigma_1+\Sigma_2)[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}, \qquad (19)\end{aligned}$$

and by Jacobi's Formula,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}|(1-\beta)\Sigma_1+\beta\Sigma_2| &= |(1-\beta)\Sigma_1+\beta\Sigma_2|\,\mathrm{tr}\left([(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\frac{\mathrm{d}}{\mathrm{d}\beta}[(1-\beta)\Sigma_1+\beta\Sigma_2]\right) \\ &= |(1-\beta)\Sigma_1+\beta\Sigma_2|\,\mathrm{tr}\left([(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(-\Sigma_1+\Sigma_2)\right). \qquad (20)\end{aligned}$$

Finally, if we rewrite

$$\begin{aligned}f(\beta) &= f_1(\beta)g_1(\beta)h_1(\beta), \\ f_1(\beta) &= (P(Y=\omega_1))^{\beta}, \\ g_1(\beta) &= (P(Y=\omega_2))^{1-\beta}, \\ h_1(\beta) &= \exp(-k(\beta)), \\ k(\beta) &= \frac{1}{2}f_2(\beta)g_2(\beta)h_2(\beta)+\frac{1}{2}\ln\left(\frac{f_3(\beta)}{g_3(\beta)h_3(\beta)}\right), \\ f_2(\beta) &= \beta, \\ g_2(\beta) &= 1-\beta, \\ h_2(\beta) &= (\vec{\mu}_1-\vec{\mu}_2)^{\top}[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(\vec{\mu}_1-\vec{\mu}_2), \\ f_3(\beta) &= |(1-\beta)\Sigma_1+\beta\Sigma_2|, \\ g_3(\beta) &= |\Sigma_1|^{1-\beta}, \\ h_3(\beta) &= |\Sigma_2|^{\beta},\end{aligned}$$

then we have

$$\begin{aligned}
f'(\beta) &= f_1'(\beta)g_1(\beta)h_1(\beta) + f_1(\beta)g_1'(\beta)h_1(\beta) + f_1(\beta)g_1(\beta)h_1'(\beta), \qquad (21)\\
f_1'(\beta) &\overset{(18)}{=} (P(Y=\omega_1))^\beta \ln(P(Y=\omega_1)),\\
g_1'(\beta) &\overset{(18)}{=} -(P(Y=\omega_2))^{1-\beta} \ln(P(Y=\omega_2)),\\
h_1'(\beta) &= -\exp(-k(\beta))k'(\beta),\\
k'(\beta) &= \frac{1}{2} f_2'(\beta)g_2(\beta)h_2(\beta) + f_2(\beta)g_2'(\beta)h_2(\beta) + f_2(\beta)g_2(\beta)h_2'(\beta))\\
&\quad + \frac{\left(\frac{f_3'(\beta)g_3(\beta)h_3(\beta) - f_3(\beta)g_3'(\beta)h_3(\beta) - f_3(\beta)g_3(\beta)h_3'(\beta)}{g_3(\beta)^2 h_3(\beta)^2}\right)}{2\left(\frac{f_3(\beta)}{g_3(\beta)h_3(\beta)}\right)},\\
f_2'(\beta) &= 1,\\
g_2'(\beta) &= -1,\\
h_2'(\beta) &\overset{(19)}{=} -(\vec{\mu}_1 - \vec{\mu}_2)^\top [(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(-\Sigma_1 + \Sigma_2)[(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(\vec{\mu}_1 - \vec{\mu}_2),\\
f_3'(\beta) &\overset{(20)}{=} |(1-\beta)\Sigma_1 + \beta\Sigma_2| \operatorname{tr}\left([(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(-\Sigma_1 + \Sigma_2)\right),\\
g_3'(\beta) &\overset{(18)}{=} -|\Sigma_1|^{1-\beta} \ln|\Sigma_1|,\\
h_3'(\beta) &\overset{(18)}{=} |\Sigma_2|^\beta \ln|\Sigma_2|,
\end{aligned}$$

1.2 Implementation

All matrix calculations are done using the Eigen C++ library (`http://eigen.tuxfamily.org/`). Random standard multivariate Gaussian-distributed vectors are generated using the C++11 `<random>` library, and then the transform described in eq. (3) is applied to get the correct distribution. Eigen provides a library (`SelfAdjointEigenSolver`) for quickly finding the eigendecomposition of symmetric matrices. This library includes a way to retrieve Φ and a way to calculate $\Sigma^{1/2}$. A classifier is then implemented by automatically detecting the case and choosing from three different discriminant functions (described in eqs. (5), (7) and (8)) based on the detected case. The prior probabilities for each class is calculated as the quotient of the sample size for the class and the sum of the sample sizes for all classes.

The inverse variance matrices and variance determinants are pre-computed based on the case detected. If the case is 1, then the determinant is not needed and the inverse matrix is calculated by Eigen very easily using its `Diagonal` library, which presumably just finds the reciprocal of each diagonal element. If the case is 2, then the determinant is still not needed, but the matrices are now arbitrary and more difficult to find inverses for. However, since covariance matrices are symmetric positive semi-definite, we can use the Cholesky decomposition, which is provided by Eigen's `LLT` library. Finally, if the case is 3, then each covariance matrix is still positive semi-definite, but we need to calculate the determinant of each one (see eq. (5)), and the Cholesky decomposition does not help with that. Instead we use the LU decomposition, which gives both the matrix inverse and determinant easily via Eigen's `PartialPivLU` library.

Then each vector generated in the sample is classified and checked against the class it was generated from. If they mismatch, the mismatch is tallied and the classifier continues until each sample is classified. The error rate is then calculated as the quotient of the number of mismatched vectors and the total number of vectors. The Bhattacharyya bound is calculated from eq. (17) with $\beta = 0.5$. Then, a hybrid iterative root-finding algorithm that uses the secant method (or the bisection method if the secant method attempts to diverge and to set the second initial guess for the secant method) with an initial guess of $\beta = 0.5$ is run on eq. (21) to find the Chernoff Bound. This should always work since there should be only one local minimum in $(0, 1)$ and $\beta = 0.5$ should be quite close. Finally, the decision boundary quadric parameters are calculated using eqs. (14) to (16) and then written out into an output file, where there will be used by plotting software to plot the boundary.

To make plots, `gnuplot` was used. To plot the decision boundary, `gnuplot` first plots the quadric as a surface, then finds the contour curve at $z = 0$ according to eq. (10). This curve is the decision boundary. In addition, the joint pdf is pre-calculated and each point on the surface is classified according to the more likely class. This is included as a surface which lies above the sample scatter plots and decision boundary to illustrate that the decision boundary lines up with which class is actually more likely at each point.

1.3 Results and Discussion

2 Parts 3 & 4

2.1 Theory

2.2 Implementation

2.3 Results and Discussion