(PSL) Coding Assignment 1

Contents

Background

This assignment is related to the simulation study described in Section 2.3.1 (the so-called Scenario 2 or Example 2) of "Elements of Statistical Learning" (ESL).

Scenario 2: the two-dimensional data $X \in R^2$ in each class are generated from a mixture of 10 different bivariate Gaussian distributions with uncorrelated components and different means, i.e.,

$$X \mid Y = k, Z = j \quad \sim \mathcal{N}(\mathbf{m}_{kj}, s^2 I_2)$$

where $k = 0, 1$, and $j = 1, 2, \dots, 10$. Set

$$P(Y = k) = 1/2, \quad P(Z = j) = 1/10, \quad s^2 = 1/5.$$

In other words, given $Y = k$, X follows a mixture distribution with probability density function (PDF)

$$\frac{1}{10} \sum_{j=1}^{10} \left(\frac{1}{\sqrt{2\pi s^2}} \right)^2 e^{-\|\mathbf{x} - \mathbf{m}_{kj}\|^2/(2s^2)}.$$

Part 1: Generate Data

1. First generate the 20 centers from two-dimensional normal and randomly split them into two classes of 10. You can use any mean and covariance structure. **You should not regenerate the centers. Use these 20 centers throughout this simulation study.**
2. Given the 20 centers, generate a training sample of size 200 (100 from each class) and a test sample of size 10,000 (5,000 from each class).
3. Produce a **scatter plot** of the **training** data:
   - assign different colors to the two classes of data points;
   - overlay the 20 centers on this scatter plot, using a distinguishing marker (e.g., a star or a different shape) and color them according to their respective class.

Part 2: kNN

1. Implement kNN **from scratch**; use Euclidean Distance. Your implementation should meet the following requirements:
   - **Input**: Your kNN function should accept three input parameters: training data (with labels), test data (without labels), and k. No need to write your kNN function to handle any general input; it suffices to write a function that is able to handle the data for this specific simulation study: binary classification; features are two-dimensional numerical vectors.
   - **Output**: Your function should return a vector of predictions for the test data.
   - **Vectorization**: Efficiently compute distances between all test points and training points simultaneously. Make predictions for all test points in a single operation.
   - **No Loops**: Do not use explicit loops like `for` or `while` inside your kNN function to compute distances or make predictions. Instead, harness the power of vectorized operations for efficient computations. For example, you can use broadcasting in Numpy or command `outer` in R.
2. Explain how you handle distance ties and voting ties;
   - **distance ties** may occur when you have multiple (training) observations that are equidistant from a test observation.
   - **voting ties** may occur when K is an even number and you have 50% of the k-nearest-neighbors from each of the two classes.
3. Test your code with the training/test data you just generated when K = 1, 3, 5; and compare your results with `knn` in R or `sklearn.neighbors` in Python.
   - Report your results (**on the test data**) as a 2-by-2 table (confusion matrix) for each K value.
   - Report the results from `knn` or `sklearn.neighbors` as a 2-by-2 table (confusion matrix) for each K value.
   - Students who will use R for this assignment should read the post "The buffer zone issue with kNN in R."

Part 3:cvKNN

1. Implement KNN classification with K chosen by 10-fold cross-validation **from scratch**.
   - Set the candidate K values from 1 to 180. (The maximum candidate K value is 180. Why?)
   - From now on, you are allowed to use the built-in kNN function from R or Python instead of your own implementation from Part 2.
   - It is possible that multiple K values give the (same) smallest CV error; when this happens, pick the largest K value among them, since the larger the K value, the simpler the model.
2. Test your code with the training/test data you just generated. Report your results (**on the test data**) as a 2-by-2 table and also report the value of the selected K.
3. The 'No Loops' rule doesn't apply in this Part; you're allowed to loop over K.
4. Note that the CV procedure should be applied only to the training data to select the best K value. Once the optimal K is determined, apply kNN to the test data and report the corresponding test error.

Part 4: Bayes rule

1. Implement the Bayes rule. Your implementation should meet the following requirements:
   - Do not use explicit loops over the test sample size (10,000 or 5,000).
   - You are allowed to use loops over the number of centers (10 or 20), although you can avoid all loops.
2. Test your code with the test data you just generated. (Note that you do not need training data for the Bayes rule.) Report your results (on the test data) as a 2-by-2 table.

The Bayes rule for binary classification (under the zero-one loss), as derived in class, is: predict Y to be 1, if

$$P(Y = 1 \mid X = x) \geq P(Y = 0 \mid X = x),$$

or equivalently

$$\frac{P(Y = 1 \mid X = x)}{P(Y = 0 \mid X = x)} \geq 1.$$

Following the data generation process, we have

$$\frac{P(Y = 1 \mid X = x)}{P(Y = 0 \mid X = x)} = \frac{P(Y = 1) \cdot P(X = x \mid Y = 1)}{P(Y = 0) \cdot P(X = x \mid Y = 0)}$$

$$= \frac{(1/2) \cdot 10^{-1} \sum_{l=1}^{10} (2\pi s^2)^{-1} \exp\left(-\|\mathbf{x} - \mathbf{m}_{1l}\|^2/(2s^2)\right)}{(1/2) \cdot 10^{-1} \sum_{l=1}^{10} (2\pi s^2)^{-1} \exp\left(-\|\mathbf{x} - \mathbf{m}_{0l}\|^2/(2s^2)\right)}$$

$$= \frac{\sum_{l=1}^{10} \exp\left(-\|\mathbf{x} - \mathbf{m}_{1l}\|^2/(2s^2)\right)}{\sum_{l=1}^{10} \exp\left(-\|\mathbf{x} - \mathbf{m}_{0l}\|^2/(2s^2)\right)}.$$

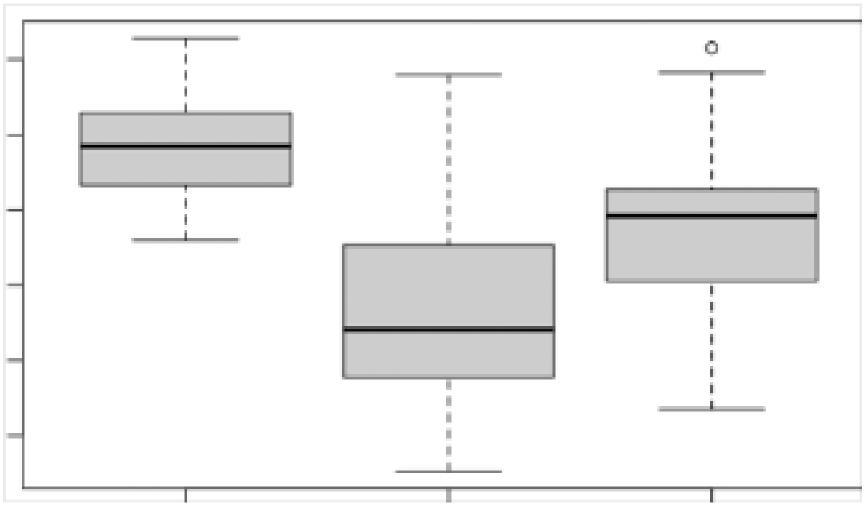
Part 5: Simulation Study

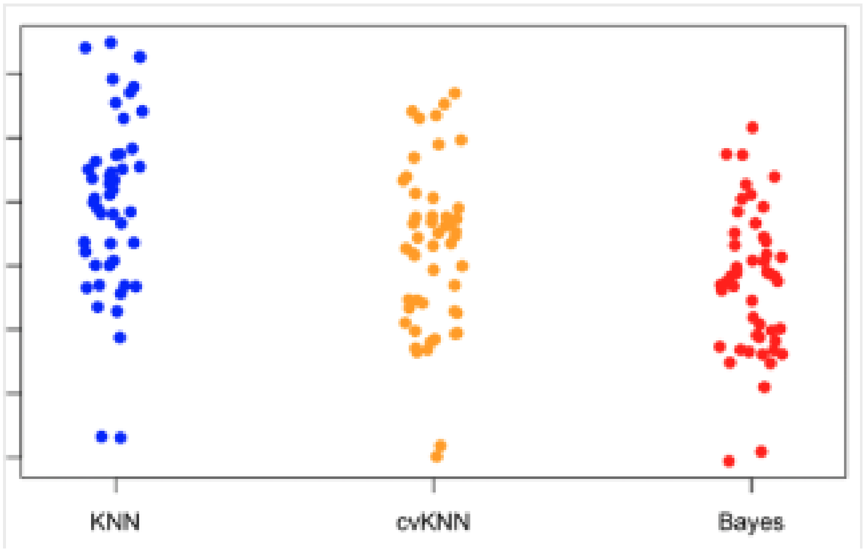
Given the 20 centers generated in Part 1, repeatedly generate 50 training/test datasets (training size = 200 and test size = 10,000). For each pair of training/test datasets, calculate the test errors (the averaged 0/1 loss on the test data set) for each of the following three procedures:

1. kNN with K = 7 (you can use the built-in kNN function from R or Python);
2. kNN with K chosen by 10-fold CV (your implementation from Part 3); and
3. the Bayes rule (your implementation from Part 4).

Graph

Present the test errors graphically, for example, using a boxplot or strip chart (see below).




KNN
cvKNN
Bayes