Programming Assignment

PART I

Data generation

(a) Use Gaussian distribution with appropriate parameters and produce a dataset with four classes and 30 samples per class: the classes must live in the 2D space and be centered on the corners of the unit square (0,0), (0,1), (1,1), (1,0), all with independent components each with variance 0.3.

(b) Obtain a 2-class train set [X, Y] by having data on opposite corners sharing the same class with labels +1 and -1.

(c) Generate a test set [X_{te}, Y_{te}] from the same distribution, starting with 200 samples per class.

(d) Visualize both sets using a scatter plot on a 2-D plane.
(e) Repeat (a)-(d) for Laplace distribution.

[Hint: The pdf of scalar Laplace distribution takes the following form.

$$f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

Relate mean and variance to μ and b *(>0)*.]

PART II

kNN classification (Attempt first for the Gaussian dataset, then repeat for the Laplacian dataset)

1. The k-Nearest Neighbors algorithm (kNN) assigns to a test point the most frequent label of its k closest examples in the training set.

(a) Write a function kNNClassify to generate predictions Y_p for the 2-class data generated at Section 1. Pick a "reasonable" k.

(b) Evaluate the classification performance (prediction error) by comparing the predicted labels Y_p to the true labels Y_{te}.

(c) Visualize the obtained results, e.g. by plotting the wrongly classified points using different colors/markers:

(d) Write a function to generate & visualize the decision regions of the 2D plane that are associated with each class, for a given classifier. Overlay the test points using scatter.

2. Parameter selection: What is a good value for k? So far we considered an arbitrary choice for k. You will now use the function hold outCVkNN for model selection

(a) Perform hold-out cross-validation by setting aside a fraction (ρ of the training set for validation. Note: You may use ρ = 0.3, and repeat the procedure 10 times. The hold-out procedure may be quite unstable.

- Use a large range of candidate values for k (e.g. k = 1, 3, 5..., 21). Notice odd numbers are considered to avoid ties.
- Repeat the process for 10 times using a random cross-validation set each time with a ρ = 0.3
- Plot the training and validation errors for the different values of k. • How would you now answer the question ”what is the best value for k”?

(b) How is the value of k affected by ρ (percentage of points held out) and number of repetitions? What does a large number of repetitions provide?

(c) Apply the model obtained by cross-validation (i.e., best k) to the test set and check if there is an improvement on the classification error over the result of (1).

PART III (modified based on student feedback)

Soft-margin (linear) SVM classifier

To extend SVM to cases in which the data are not linearly separable, we introduced in class the hinge loss function (for the *i*-th data point)

$$\max\left(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b)\right).$$

Note that y_i is the *i*-th target (i.e., in this case, 1 or −1), and

$$\mathbf{w}^\mathsf{T}\mathbf{x}_i - b$$

is the *i*-th output.

This function is zero if x_i lies on the correct side of the decision boundary. For data on the wrong side of the decision boundary, the function's value is proportional to the distance from the margin.

The goal of the optimization then is to minimize

$$\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b)\right)\right]$$

where the parameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the x_i lies on the correct side of the margin. By deconstructing the hinge loss, this optimization problem can be massaged into the following:

P1:

$$\begin{array}{ll} \underset{\mathbf{w},\, b,\, \zeta}{\text{minimize}} & \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\zeta_i \\ \text{subject to} & y_i(\mathbf{w}^\top\mathbf{x}_i - b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \quad \forall i \in \{1, \ldots, n\} \end{array}$$

Thus, for large values of C , it will behave similarly to the hard-margin SVM, if the input data are linearly classifiable, but will still learn if a classification rule is viable or not.

[Use data generated in Part I with the following modification: Samples generated with the mean at corners (0,0) and (0,1) are assigned the label +1, and those corresponding to corners (1,1) and (1,0) are assigned the label -1.]

(a) Optimize the problem P1 for various values of C.

(b) Repeat Q2(a) and Q2(b) of Part II to obtain best C for the current problem (instead of k in kNN classifier) and the current dataset.

(c) Make a comparison of the performance of the soft-margin SVM with best value of C and kNN with best k (for the current modified dataset), that is, which one performs better.