LAB2: Local Methods and Cross Validation

Goal: to offer some familiarity with the k Nearest Neighbors (kNN) algorithm and get a practical understanding of the bias-variance trade-off using synthetic data.

This lab is divided into two parts depending of their level of complexity (**Beginner, Advanced**). Your goal is to complete entirely, at least, one of the two parts. Please note that a different notation can be used in the code as we used in the lectures.

PART I: Beginner

Warm up: Data Generation

1. Using the class `MixGauss.py` with appropriate parameters, 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 variance 0.3.
   To run the example: Xtr, Ytr = MixGauss([….])
   To visualise the data: plt.scatter(Xtr[:,0], Xtr[:,1], 25, Ytr)
   plt.show()
2. Manipulate the data so to obtain a 2-class problem where data on opposite corners share the same class with labels +1 and -1. Hint: if you produced the data following the centers order provided above, you can use a mapping like
   Ytr = (Ytr % 2) * 2 - 1;
3. Similarly generate a "test set" `Xte`, `Yte` drawn from the same distribution (start with 200 samples per class) and plot it using plt.scatter().

The Geek part

The *k-Nearest Neighbors* algorithm (kNN) assigns to a test point the most frequent label of its k closest examples in the training set. Study the code of class `kNNClassify.py`.

4. Use `kNNClassify.py` on the 2-class data generated at step 1. Pick a "reasonable" number of neigbours k.
5. Evaluate the classification performance by comparing the estimated to the true outputs:
   sum(Ypred~=Yte)/Nt   %Nt number of test data
6. To visualise the separating function (and thus visualise what areas of the 2D plane are

associated with each class) you can use the provided class separatingFkNN.py and again plot the test points with plt.scatter().

Analysis

So far we considered an arbitrary choice for k.

7. Perform a hold-out cross validation procedure on the available training data for a large range of candidate values for k (e.g. k=1, 3, 5. . . , 41). Repeat the hold-out experiment for rep=10 time using at each iteration p=30% of the training set for validation. You can use the provided class holdoutCVkNN.py (check the code for an example use). Look at 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"? Note: for the parameters rep=10 and p=0.3, the hold-out procedure may be quite unstable.
8. How is the value affected by p (percentage of points held out) and number rep (number of repetitions e.g., 1, 5, 30, 50, 100) of the experiment? What does a large number of repetitions provide?
9. Apply the model obtained by cross validation (i.e., best k) to the test set (Xte) and see if there is an improvement on the classification error over the result of point 5 above.

PART II: Advanced

Advanced Analysis

10. *Dependence on training size:* Evaluate the performance as the size of the training set grows (varies), e.g., n = {3, 5, 20, 50, 100, 300, 500, . . . } How would you choose a good range for k as n changes? Repeat the validation and test multiple times. What can you say about the stability of the solution/performances?
11. Try classifying more difficult datasets, generated through MixGauss.py, for instance, by increasing variance.
12. Modify the class kNNClassify.py to handle multi-class problems.
13. Modify the class kNNClassify.py to handle regression problems.