PSM Fall 2015, Exercise 10 (the last one)

Overview. This is again a programming exercise. The goal is to train a digits classifier from our "workhorse" digits dataset. The classification algorithm will be based on Equation (3) from the Part 4 PSM lecture notes. This boils down to estimate class-conditional distributions $P(X \mid Y = C_k)$ from the training data. The formalism to represent these distributions will be Gaussian Parzen windows (GPW, explained below). The capacity of these GPW models is regulated by a certain parameter h^2. You will optimize the model capacity by cross validation. This exercise is quite comprehensive because it uses almost all concepts introduced in Part 4 of the lecture, so the due date is December 6. I will now take you through the steps of this assignment.

Step 1: familiarize yourself with GPW models. You find a detailed description of these models in my lecture notes

http://minds.jacobs-university.de/sites/default/files/uploads/teaching/lectureNotes/LN_AlgMod.pdf

in Section 3.3. GPW models estimate a pdf on pattern space $\mathbb{R}^n$ from a set of training points $x_i \in \mathbb{R}^n$ by placing a separate symmetric "Gaussian bell-curve pdf" around each training point, and then use the mean of all of these point-centered mini-Gaussians as the sought-after pdf. The formula for this is Equation 3.41 in those online lecture notes. Using GPWs for a quick estimation and representation of a pdf is simple and popular and not stupid – a method worth knowing.

The width of these local Gaussians is given by the variance h^2. This works as a *regularizer* (read Section 6.5 of the PSM lecture notes). Large h^2 means "soft" Gaussians, hence low model capacity. Small h^2 means "sharp" Gaussians, hence large capacity. I created Figure 16 in the PSM LNs by such GPW models. Finding the right value of h^2 (by cross-validation) is the key for good model quality.

Step 2: Prepare data for *k*-fold cross-validation. Use the first 100 digits per digit class as training data. Keep the other 100 examples per class in a secrete place and don't touch them during the entire learning procedure. We pretend we don't know them at model learning time. They will be used only after you have trained the model for an ultimate test. Let us denote the available training data by $D = (x_i, y_i)_{i = 1, ..., 1000}$. Split this dataset into k equally sized subsets $D_k = (x_i^j, y_i^j)_{i = 1, ..., 1000/k; j = 1, ..., k}$. The number k of folds is your choice. Make sure that each D_k contains equally many examples from each of the 10 digit classes. Denote the set $D \setminus D_k$ by $D_{\setminus k}$. In a crossvalidation fold, $D_{\setminus k}$ is used as training set and D_k as validation set.

Step 3: Program the core cross-validation subroutine. The core subroutine – call it `XvalFold`($D_{\setminus k}$, D_k, h^2) – gets three inputs, a training set, a validation set, and a regularization parameter h^2. `XvalFold` does the following:

1. Using $D_{\setminus k}$, for each digit class C_m, `XvalTest` estimates a pdf p_m on the pattern space $\mathbb{R}^{240}$ for the class-conditional distribution $P(X \mid Y = C_m)$, in the GPW format. Note that this pdf is mathematically given by

$$p_m(x) = \frac{1}{L}\sum_{i=1}^{L} \frac{1}{(2\pi h^2)^{120}} \exp\left(-\frac{\|x - x_i\|^2}{2h^2}\right).$$

This looks like numerical trouble because the denominator $(2\pi h^2)^{120}$ will be almost zero or almost infinity, in either case beyond your computer's numerical range. Fortunately, the ultimate use of this pdf for classification via the formula (3) from the Part 4 PSM lecture notes is invariant to any global scaling factor. Therefore we may simply drop the cumbersome normalization factors from the above equation and use un-normalized pseudo-pdfs

$$\tilde{p}_m(x) = \sum_{i=1}^{L} \exp\left(-\frac{\|x - x_i\|^2}{2h^2}\right)$$

instead. The index i in this equation ranges over all patterns from class m contained in your $D_{\backslash k}$.

Note that $\tilde{p}_m$ is a function $\tilde{p}_m : \mathbb{R}^{240} \to \mathbb{R}$. "Computing" this pdf therefore means to create a function object.

2. Using $\tilde{p}_1,\ldots,\tilde{p}_{10}$, classify the patterns from the validation set D_k on the basis of the formula (3) from the Part 4 PSM lecture notes. Because in our data all digit classes have the same prior probability, the factor $P(Y = C_m)$ in that formula can be omitted. Similarly, classify all patterns in the training set $D_{\backslash k}$.
3. The numbers of misclassifications on the validation and the training sets is the output of the call `XvalFold`($D_{\backslash k}$, D_k, h^2).

Step 4: program a complete *k*-fold cross-validation procedure. Making use of `XvalFold`, program a routine `Xval`(D_1, ..., D_k, h^2) which returns the training and validation misclassification number averaged over all k folds.

Step 5: Search for best model capacity. Run `Xval`(D_1, ..., D_k, h^2) for a decreasing sequence of h^2, that is a sequence of growing model capacity. Create a plot similar to Figure 13 from the Part 4 PSM lecture notes. Determine the regularizer h^2_{opt} which gives the lowest validation misclassification number.

Step 6: Compute final model and test it on the "secret" test data. Use h^2_{opt} to obtain a function `classify`(x) which upon input of test pattern $x \in \mathbb{R}^{240}$ returns a classification label. Compute the misclassification rate on the "secret" test data set that wasn't touched until this point.

Deliverables. 1. A short typeset documentation of your procedure, including the plot from Step 5 and the final test misclassification rate. 2. Your Matlab or Python code, suitably commented such that we can quickly understand it. Don't use any additional add-on tools beyond basic Matlab or Python + numpy + scipy. Submission deadline is December 6 midnight (10 % point subtraction for every day of delay, we will launch this penalty even when the delay is only one minute after midnight). Email your documentation and code to Owen and myself as usual.

Final comment. This entire procedure with Gaussian Parzen windows and cross-validation is a "professional" (albeit still simple) method for classification learning that you may find useful on some future occasion in your carreer. While it can be improved in many ways (e.g. using non-symmetric Gaussians), in its basic format it has a good tradeoff between simplicity and classification quality.