CS489/698: Introduction to Machine Learning

Homework 4

Due: 11:59 pm, November 16, 2017, submit on LEARN.

Include your name, student number and session!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!
[Text in square brackets are hints that can be ignored.]

Exercise 1: Gaussian Mixture Model (GMM) (40 pts)

Notation: For a matrix A, $|A|$ denotes its determinant. For a diagonal matrix $\mathrm{diag}(\mathbf{s})$, $|\mathrm{diag}(\mathbf{s})| = \prod_i s_i$.

Algorithm 1: EM for GMM.

```
Input: $X \in \mathbb{R}^{n\times d}$, $K \in \mathbb{N}$, initialization for $model$
// $model$ includes $\pi \in \mathbb{R}^K_+$ and for each $1 \le k \le K$, $\boldsymbol{\mu}_k \in \mathbb{R}^d$ and $S_k \in \mathbb{S}^d_+$
// $\pi_k \ge 0$, $\sum_{k=1}^K \pi_k = 1$, $S_k$ symmetric and positive definite.
// random initialization suffices for full credit.
// alternatively, can initialize $r$ by randomly assigning each data to one of the $K$
   components
Output: $model, \ell$
for $iter = 1 : \textsc{maxiter}$ do
    // step 2, for each $i = 1, \ldots, n$
    for $k = 1, \ldots, K$ do
        $r_{ik} \leftarrow \pi_k |S_k|^{-1/2} \exp[-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top S_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)]$        // compute responsibility
    // for each $i = 1, \ldots, n$
    $r_{i.} \leftarrow \sum_{k=1}^K r_{ik}$
    // for each $k = 1, \ldots, K$ and $i = 1, \ldots, n$
    $r_{ik} \leftarrow r_{ik}/r_{i.}$                                                   // normalize
    // compute negative log-likelihood
    $\ell(iter) = -\sum_{i=1}^n \log(r_{i.})$
    if $iter > 1$ && $|\ell(iter) - \ell(iter-1)| <= \textsc{tol} * |\ell(iter)|$ then
        break
    // step 1, for each $k = 1, \ldots, K$
    $r_{.k} \leftarrow \sum_{i=1}^n r_{ik}$
    $\pi_k \leftarrow r_{.k}/n$
    $\boldsymbol{\mu}_k = \sum_{i=1}^n r_{ik}\mathbf{x}_i / r_{.k}$
    $S_k \leftarrow \left(\sum_{i=1}^n r_{ik}\mathbf{x}_i\mathbf{x}_i^\top / r_{.k}\right) - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^\top$
```

1. (20 pts) Derive and implement the EM algorithm for the spherical Gaussian mixture model, where all covariance matrices are constrained to be diagonal. Algorithm 1 recaps all the essential steps and serves as a hint rather than a verbatim instruction. In particular, you must change the highlighted steps accordingly (with each S_k being a diagonal matrix), along with formal explanations. Analyze the space and time complexity of your implementation. [You might want to review the steps we took in class (slide 15) to get the updates in Algorithm 1 and adapt them to the simpler case here. The solution should look like $s_j = \frac{\sum_{i=1}^n r_{ik}(x_{ij}-\mu_j)^2}{\sum_{i=1}^n r_{ik}} = \frac{\sum_{i=1}^n r_{ik}x_{ij}^2}{\sum_{i=1}^n r_{ik}} - \mu_j^2$ for the j-th diagonal. Multiplying an $n \times p$ matrix with a $p \times m$ matrix costs $O(mnp)$. Do not maintain a diagonal matrix explicitly; using a vector for its diagonal suffices.]

   To stop the algorithm, set a maximum number of iterations (say $\textsc{maxiter} = 500$) and also monitor the change of the negative log-likelihood ℓ:

$$\ell = -\sum_{i=1}^n \log\left[\sum_{k=1}^K \pi_k |2\pi S_k|^{-1/2} \exp[-\tfrac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^\top S_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)]\right], \qquad (1)$$

where $\mathbf{x}_i$ is the i-th column of $X^\top$. As a debug tool, note that ℓ should decrease from step to step, and we can stop the algorithm if the decrease is smaller than a predefined threshold, say $\text{TOL} = 10^{-5}$.

2. (20 pts) Next, we apply (the adapted) Algorithm 1 to the MNIST dataset (also available on course website). For each of the 10 classes (digits), we can use its (only its) training images to estimate its (class-conditional) distribution by fitting a GMM (with say $K = 5$, roughly corresponding to 5 styles of writing this digit). This gives us the density estimate $p(\mathbf{x}|y)$ where $\mathbf{x}$ is an image (of some digit) and y is the class (digit). We can now classify the test set using the Bayes classifier (whose optimality you proved in A2):

$$\hat{y}(\mathbf{x}) = \arg \max_{c=0,\ldots,9} \underbrace{\Pr(Y = c) \cdot p(X = \mathbf{x}|Y = c)}_{\propto \, \Pr(Y=c|X=\mathbf{x})}, \tag{2}$$

where the probabilities $\Pr(Y = c)$ can be estimated using the training set, e.g., the proportion of the c-th class in the training set, and the density $p(X = \mathbf{x}|Y = c)$ is estimated using GMM for each class c separately. Report your error rate on the test set as a function of K (if time is a concern, using $K = 5$ will receive full credit). [Optional: Reduce dimension by PCA may boost accuracy quite a bit. Your running time should be on the order of minutes (for one K), if you do not introduce extra for-loops in Algorithm 1.]

[In case you are wondering, our classification procedure above belongs to the so-called plug-in estimators (plug the estimated densities to the known optimal Bayes classifier). However, note that estimating the density $p(X = \mathbf{x}|Y = c)$ is actually harder than classification. Solving a problem (e.g. classification) through some intermediate harder problem (e.g. density estimation) is almost always a bad idea.]

Ans: The updates for π, $\boldsymbol{\mu}_k$, and r remain the same. For the covariance matrix $S = \operatorname{diag}(\mathbf{s})$ which is constrained to be diagonal, we need to solve (for the k-th component):

$$\min_{\mathbf{s}} \sum_{i=1}^{n} r_{ik} \left[\frac{1}{2} \sum_{j=1}^{d} \log s_j + \frac{1}{2} (\mathbf{x}_i - \boldsymbol{\mu})^\top \operatorname{diag}(\mathbf{s})^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \right]. \tag{3}$$

In the above we omitted the subscript k for $\mathbf{s}$ and $\boldsymbol{\mu}$, to avoid unnecessary clutter. Taking derivative w.r.t. s_j and setting it to 0:

$$\sum_{i=1}^{n} r_{ik} [1/s_j - (x_{ij} - \mu_j)^2 / s_j^2] = 0. \tag{4}$$

Thus, we have the (simplified) update rule:

$$s_j = \frac{\sum_{i=1}^n r_{ik} (x_{ij} - \mu_j)^2}{\sum_{i=1}^n r_{ik}} = \frac{\sum_{i=1}^n r_{ik} x_{ij}^2}{\sum_{i=1}^n r_{ik}} - \mu_j^2. \tag{5}$$

Exercise 2: Gaussian Processes (GP) (30 pts)

Let $Z_t \sim \mathcal{GP}(m, \kappa)$ be a Gaussian process with $t \in T$, $m : T \to \mathbb{R}$ being the mean function, and $\kappa : T \times T \to \mathbb{R}$ being the covariance function. Suppose we have observed $Z_{t_1}, Z_{t_2}, \ldots, Z_{t_n}$. Complete the following two exercises to verify the consistency in GP prediction:

1. (15 pts) Suppose we are interested in predicting $Z_{t_{n+1}}$. What is the conditional distribution of $Z_{t_{n+1}}$ given $Z_{t_1}, Z_{t_2}, \ldots, Z_{t_n}$? Please state the form of the distribution and the essential parameters for specifying this distribution.

2. (15 pts) Suppose we want to predict jointly $Z_{t_{n+1}}, Z_{t_{n+2}}, \ldots, Z_{t_{n+p}}$. What is the conditional distribution of $Z_{t_{n+1}}, Z_{t_{n+2}}, \ldots, Z_{t_{n+p}}$ given $Z_{t_1}, Z_{t_2}, \ldots, Z_{t_n}$? Please state the form of the distribution and the essential parameters for specifying this distribution. Once we have this joint conditional distribution, we can also

predict $Z_{t_{n+1}}$, by integrating $Z_{t_{n+2}}, \ldots, Z_{t_{n+p}}$ out from the conditional distribution, i.e., compute the marginal distribution of $Z_{t_{n+1}}$ from the conditional distribution. Compare this predictive distribution of $Z_{t_{n+1}}$ with the one you derived in the previous subproblem.

Ans: Partition the kernel matrix

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}, \tag{6}$$

so that $K_{11} \in \mathbb{R}^{n \times n}$ is the kernel κ evaluated at every pair in the training set $t_1, \ldots, t_n$, $K_{22} \in \mathbb{R}^{p \times p}$ is the kernel κ evaluated at every pair in the test set $t_{n+1}, \ldots, t_{n+p}$, and K_{12} is the kernel κ evaluated at every cross-pair between the training set and test set. Similarly, define

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \tag{7}$$

where $\boldsymbol{\mu}_1$ is the mean function m evaluated at the training set and $\boldsymbol{\mu}_2$ is the mean function m evaluated at the test set.

Then, the conditional distribution of $Z_{t_{n+1}}, Z_{t_{n+2}}, \ldots, Z_{t_{n+p}}$, given $Z_{t_1}, Z_{t_2}, \ldots, Z_{t_n}$, is a Gaussian distribution, with the following mean and covariance matrix:

$$\boldsymbol{\mu} = \boldsymbol{\mu}_2 + K_{21} K_{11}^{-1} (\mathbf{Z} - \boldsymbol{\mu}_1) \tag{8}$$

$$S = K_{22} - K_{21} K_{11}^{-1} K_{12}, \tag{9}$$

where $\mathbf{Z} = [Z_{t_1}, Z_{t_2}, \ldots, Z_{t_n}]$. The marginal distribution of $Z_{t_{n+1}}$ in the above conditional distribution is again Gaussian, with the following mean and covariance matrix:

$$\mu = [\mu_2]_1 + [K_{21}]_{1:} K_{11}^{-1} (\mathbf{Z} - \boldsymbol{\mu}_1) \tag{10}$$

$$\sigma^2 = [K_{22} - K_{21} K_{11}^{-1} K_{12}]_{11}. \tag{11}$$

Let $\mu_{n+1} := m(t_{n+1})$, $\mathbf{k} := [\kappa(t_{n+1}, t_1), \kappa(t_{n+1}, t_2), \ldots, \kappa(t_{n+1}, t_n)]$, and $\gamma^2 = \kappa(t_{n+1}, t_{n+1})$. Then, clearly $[\mu_2]_1 = \mu_{n+1}$, $[K_{21}]_{1:} = \mathbf{k}$, and $[K_{22}]_{11} = \gamma^2$. Thus,

$$\mu = \mu_{n+1} + \mathbf{k} K_{11}^{-1} (\mathbf{Z} - \boldsymbol{\mu}_1) \tag{12}$$

$$\sigma^2 = \gamma^2 - \mathbf{k} K_{11}^{-1} \mathbf{k}^\top. \tag{13}$$

For the first subproblem, simply let $p = 1$ and it is immediate that the conditional distribution of $Z_{t_{n+1}}$ given $Z_{t_1}, \ldots, Z_{t_n}$ is exactly the same Gaussian distribution with mean μ and variance σ^2 above.

Exercise 3: Regularization (30 pts)

Notation: For the vector $\mathbf{x}_i$, we use x_{ij} to denote its j-th element.

Overfitting to the training set is a big concern in machine learning. One simple remedy to avoid overfitting to any particular training data is through injecting noise: we randomly perturb each training data before feeding it into our machine learning algorithm. In this exercise you are going to prove that injecting noise to training data is essentially the same as adding some particular form of regularization. We use least-squares regression as an example, but the same idea extends to other models in machine learning almost effortlessly.

Recall that least-squares regression aims at solving:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2, \tag{14}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \mathbb{R}$ are the training data. (For simplicity, we omit the bias term here.) Now, instead of using the given feature vector $\mathbf{x}_i$, we perturb it first by some independent noise $\boldsymbol{\epsilon}_i$ to get $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i, \boldsymbol{\epsilon}_i)$, with different choices of the perturbation function f. Then, we solve the following **expected** least-squares regression problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \mathbf{E}[(y_i - \mathbf{w}^\top \tilde{\mathbf{x}}_i)^2], \tag{15}$$

where the expectation removes the randomness in $\tilde{\mathbf{x}}_i$ (due to the noise $\boldsymbol{\epsilon}_i$). [To understand the expectation, think of n as so large that we have each data appearing repeatedly many times in our training set.]

1. (15 pts) Let $f(\mathbf{x}_i, \boldsymbol{\epsilon}_i) = \mathbf{x}_i + \boldsymbol{\epsilon}_i$ where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \lambda\mathbb{I})$ follows the standard Gaussian distribution. Simplify (15) as the usual least-squares regression (14), plus a familiar regularization function on $\mathbf{w}$.

   Ans: We clearly have the following identity:

$$\begin{aligned} \mathbf{E}[(y_i - \mathbf{w}^\top \tilde{\mathbf{x}}_i)^2] &= \mathbf{E}[(y_i - \mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \boldsymbol{\epsilon}_i)^2] && (16) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 - \mathbf{E}[2(y_i - \mathbf{w}^\top \mathbf{x}_i)\mathbf{w}^\top \boldsymbol{\epsilon}_i] + \mathbf{E}[\mathbf{w}^\top \boldsymbol{\epsilon}_i]^2 && (17) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 - 0 + \lambda \|\mathbf{w}\|_2^2, && (18) \end{aligned}$$

   where the last line follows from the assumption $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \lambda\mathbb{I})$. Thus, (15) is simply the usual ridge regression (with regularization constant $n\lambda$).

2. (15 pts) Let $\tilde{\mathbf{x}}_i = f(\mathbf{x}_i, \boldsymbol{\epsilon}_i) = \mathbf{x}_i \odot \boldsymbol{\epsilon}_i$, where $\odot$ denotes the element-wise product and $p\epsilon_{ij} \sim$ Bernoulli(p) independently for each j. That is, with probability $1-p$ we reset x_{ij} to 0 and with probability p we scale x_{ij} as x_{ij}/p. Note that for different training data $\mathbf{x}_i$, $\boldsymbol{\epsilon}_i$'s are independent. Simplify (15) as the usual least-squares regression (14), plus a different regularization function on $\mathbf{w}$. [This way of injecting noise, when applied to the weight vector $\mathbf{w}$ in a neural network, is known as Dropout (DropConnect).]

   Ans: As above, we have

$$\begin{aligned} \mathbf{E}[(y_i - \mathbf{w}^\top \tilde{\mathbf{x}}_i)^2] &= \mathbf{E}[y_i - \mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top (\mathbf{x}_i \odot (\boldsymbol{\epsilon}_i - 1))]^2 && (19) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 - \mathbf{E}[2(y_i - \mathbf{w}^\top \mathbf{x}_i)\mathbf{w}^\top (\mathbf{x}_i \odot (\boldsymbol{\epsilon}_i - 1))] + \mathbf{E}[\mathbf{w}^\top (\mathbf{x}_i \odot (\boldsymbol{\epsilon}_i - 1))]^2 && (20) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 - 0 + \mathbf{E}\left[\sum_j w_j x_{ij}(\epsilon_{ij} - 1)\right]^2 && (21) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \sum_j \sum_k w_j w_k x_{ij} x_{ik} \mathbf{E}[(\epsilon_{ij} - 1)(\epsilon_{ik} - 1)] && (22) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \sum_j w_j^2 x_{ij}^2 \mathbf{E}[(\epsilon_{ij} - 1)^2] && (23) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \sum_j w_j^2 x_{ij}^2 [(1-p) + p(1/p - 1)^2] && (24) \\ &= (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \frac{1-p}{p} \sum_j w_j^2 x_{ij}^2. && (25) \end{aligned}$$

   where we have used the fact that $p\epsilon_{ij} \sim$ Bernoulli(p) independently. Thus, (15) is simply the usual least squares regression, with the following regularization function:

$$\frac{1-p}{p} \sum_j \left(\sum_i x_{ij}^2 \right) w_j^2, \tag{26}$$

   i.e., a weighted (squared) ℓ_2 regularizer. In particular, if we normalize the training data so that $\sum_i x_{ij}^2 \equiv 1$, then we reduce again to ridge regression (this time with regularization constant $\frac{1-p}{p}$).