Boston University
Department of Electrical and Computer Engineering

ENG EC 414 Introduction to Machine Learning

HW 6

Issued: Fri 16 Oct 2020 **Due:** 10:00am Fri 23 Oct 2020 in Gradescope (non-code) + Blackboard

Important: Before you proceed, please read the documents pertaining to *Homework formatting and submission guidelines* in the Homeworks section of Blackboard. **In particular, for computer assignments you are prohibited from using any online code or built-in MATLAB functions except as indicated in the problem or skeleton code (when provided).**

Important: To obtain full grade, please clearly motivate all your answers.

Note: Problem difficulty = number of coffee cups ☕

Problem 6.1 [6pts] *(Verify that a function is a Kernel)*
In class we said that in general it is a difficult task to understand if a function is a kernel or not. However, in some case it is easy: it is enough to find a transformation ϕ such that $\mathcal{K}(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^\top \phi(\mathbf{v})$ for all $\mathbf{u}$ and $\mathbf{v}$.

(a) [2pts] Consider the kernel $\mathcal{K}_{\text{linear}}(\mathbf{u}, \mathbf{v}) := \mathbf{u}^\top \mathbf{v}$, where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Show that it is a kernel by finding the corresponding function ϕ.

(b) [2pts] Consider the kernel $\mathcal{K}_{\text{product}}(\mathbf{u}, \mathbf{v}) := u_1 u_2 v_1 v_2$, where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$. Show that it is a kernel by finding the corresponding function ϕ.

(c) [2pts] Consider the kernel $\mathcal{K}_{\text{one}}(\mathbf{u}, \mathbf{v}) := 1$, where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Show that it is a kernel by finding the corresponding function ϕ.

Problem 6.2 [22pts] *(Soft-Margin Binary Kernel SVM)*
In this problem, we will implement and use the Subgradient Descent Algorithm to learn the parameters of the soft-margin Kernel SVM classifier. Let's start summarizing the derivation we did in class. The following notes are a useful companion to the lecture slides as well.

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be a training set of m examples consisting of feature vector $\mathbf{x}_j \in \mathbb{R}^d$ and label $y_j \in \{-1, +1\}$ for each j. The cost function (in the unconstrained form) of the binary soft-margin SVM optimization problem is given by:

$$(\mathbf{w}_{\text{SVM}}, b_{\text{SVM}}) := \arg\min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 + c \sum_{j=1}^{m} \text{hinge}\left(y_j \left(\mathbf{w}^\top \mathbf{x}_j + b\right)\right)$$

where $\text{hinge}(t) := \max(0, 1 - t)$. Since $\mathbf{w}_{\text{SVM}}$ is a linear combination of the feature vectors in the training set, we can set

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j \mathbf{x}_j = X^\top \boldsymbol{\alpha}, \quad \text{where } \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}_{m \times 1} \text{ and } X = \begin{bmatrix} — & \mathbf{x}_1^\top & — \\ & \vdots & \\ — & \mathbf{x}_m^\top & — \end{bmatrix}_{m \times d},$$

and solve for $\mathbf{w}$ in terms of α. Then, $\mathbf{w}_{\text{SVM}} = X\boldsymbol{\alpha}_{\text{SVM}}$ where,

$$(\boldsymbol{\alpha}_{\text{SVM}}, b_{\text{SVM}}) := \arg\min_{\alpha,b} \ \frac{1}{2}\boldsymbol{\alpha}^\top XX^\top \boldsymbol{\alpha} + c\sum_{j=1}^{m} \text{hinge}\left(y_j\left(\boldsymbol{\alpha}^\top X\mathbf{x}_j + b\right)\right) \ .$$

For convenience, let us define the following quantities:

$$K := XX^\top, \qquad \text{and for } j = 1, \ldots, m, \ K_j := X\mathbf{x}_j, \ K_j^{\text{ext}} := \begin{bmatrix} 1 \\ K_j \end{bmatrix}_{(m+1)\times 1} \ .$$

K is an $m \times m$ matrix consisting of inner products between all pairs of training feature vectors, i.e., $\forall i, j = 1, \ldots, m$, the ij-th entry of K is given by $K_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$. Moreover, K_j is the j-th column of K and K_j^{ext} is the column vector K_j extended by one row by appending the value 1 at the beginning. We introduce K_j^{ext} to easily deal with the bias term b, similarly to what we did in LS. If we further define $\tilde{\boldsymbol{\alpha}} := \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}$, then the soft-margin SVM is the solution to the following optimization problem:

$$\tilde{\alpha}_{\text{SVM}} := \arg\min_{\tilde{\alpha}} F(\tilde{\alpha}), \qquad \text{where } F(\tilde{\alpha}) := \frac{1}{2}\tilde{\boldsymbol{\alpha}}^\top \begin{bmatrix} 0 & 0 \\ 0 & K \end{bmatrix} \tilde{\boldsymbol{\alpha}} + c\sum_{j=1}^{m} \text{hinge}\left(y_j \tilde{\boldsymbol{\alpha}}^\top K_j^{\text{ext}}\right) \tag{1}$$

Given a test feature vector $\mathbf{x}_{\text{test}}$, the label predicted by the soft-margin SVM is given by:

$$f_{\text{SVM}}(\mathbf{x}_{\text{test}}) = \text{sign}\left(\mathbf{w}_{\text{SVM}}^\top \mathbf{x}_{\text{test}} + b_{\text{SVM}}\right) = \text{sign}\left(\boldsymbol{\alpha}_{\text{SVM}}^\top X\mathbf{x}_{\text{test}} + b_{\text{SVM}}\right) = \text{sign}\left(\tilde{\boldsymbol{\alpha}}_{\text{SVM}}^\top K_{\mathbf{x}_{\text{test}}}^{\text{ext}}\right)$$

where $K_{\mathbf{x}_{\text{test}}} := X\mathbf{x}_{\text{test}}$ is an $m \times 1$ column vector whose i-th element is equal to $\mathbf{x}_i^\top \mathbf{x}_{\text{test}}$. Thus,

$$\boxed{f_{\text{SVM}}(\mathbf{x}_{\text{test}}) = \text{sign}\left(\tilde{\boldsymbol{\alpha}}_{\text{SVM}}^\top K_{\mathbf{x}_{\text{test}}}^{\text{ext}}\right)} \tag{2}$$

Observe that the training of the soft-margin SVM depends on the feature vectors in the training set only through their pairwise inner products which is captured by the matrix K (see Eq. (1)). Similarly, predicting the label of a test sample also depends on the feature vectors of the training set and the test feature vector only through the inner products between the training and test feature vectors captured by the vector $K_{\mathbf{x}_{\text{test}}}^{\text{ext}}$ (see Eq. (2)). The *Kernel SVM* generalizes this idea by replacing the inner products between any two feature vectors $\mathbf{u}$ and $\mathbf{v}$ by $\mathcal{K}(\mathbf{u}, \mathbf{v})$, where $\mathcal{K}(\cdot, \cdot)$ is a real-valued function of two vector-variables, referred to as a *kernel*.

To summarize, the Kernel SVM is learned by solving Eq. (1) using K equal to an $m \times m$ matrix with $K_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ for all $i, j = 1, \ldots, m$ and K_j equal to the j-th column of K. The prediction for a test point $\mathbf{x}_{\text{test}}$ is obtained by evaluating Eq. (2) using an $m \times 1$ vector $K_{\mathbf{x}_{\text{test}}}^{\text{ext}}$ whose i-th row is equal to $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_{\text{test}})$.

The optimization problem described by Eq. (1) is convex and unconstrained and can therefore be solved by the Subgradient Descent algorithm. A subgradient of the hinge function hinge(t) is the following:

$$\begin{cases} -1, & \text{when } t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

We can use the above subgradient to have the pseudocode for implementing subgradient descent for soft-margin Kernel SVM loss as follows:

We will use the *kernel-svm-2rings* dataset. We will use the entire dataset for training and visualization. There will be no testing done. We will use the Gaussian kernel which is defined as $\mathcal{K}(\mathbf{u}, \mathbf{v}) = \exp(-\gamma\|\mathbf{u} - \mathbf{v}\|^2)$, where γ is called the bandwidth parameter. Use the following choices of hyperparameters:

$$T = 10000, \quad \eta_t = \frac{1}{t}, \quad c = 100, \quad \gamma = 2 \ .$$

Algorithm 1 Subgradient Descent for Kernel SVM

input: Training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, Maximum number of iterations T, Kernel function, Trade-off hyperparameter c

initialize: $\tilde{\boldsymbol{\alpha}}_1 = \mathbf{0}$

for $t = 1, 2, \ldots, T$ **do**

  $obj_t = F(\tilde{\boldsymbol{\alpha}}_t)$ *//Evaluate the value of the objective function in* $\tilde{\boldsymbol{\alpha}}_t$

  $zeroOneAverageLoss_t = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}[y_i \neq f_{\tilde{\boldsymbol{\alpha}}_t}(\boldsymbol{x}_i)]$ *//Average 0/1 loss on the training set in* $\tilde{\boldsymbol{\alpha}}_t$

  $\mathbf{g}_t = \begin{bmatrix} 0 & 0 \\ 0 & K \end{bmatrix} \tilde{\boldsymbol{\alpha}}_t$ *// Compute subgradient first term*

  //Add subgradient second term

  for $j = 1, 2, \ldots, m$ **do**

    if $(y_j \tilde{\boldsymbol{\alpha}}_t^\top K_j^{\text{ext}} < 1)$ **then**

      $\mathbf{g}_t = \mathbf{g}_t - c y_j \, K_j^{\text{ext}}$

    end if

  end for

  Update parameters: $\tilde{\boldsymbol{\alpha}}_{t+1} = \tilde{\boldsymbol{\alpha}}_t - \eta_t \mathbf{g}_t$

end for

Extract $\boldsymbol{\alpha}$ and b from $\tilde{\boldsymbol{\alpha}}_{T+1}$

return $\boldsymbol{\alpha}, b, obj, zeroOneAverageLoss$

Note that for non-differentiable functions we said that we should take the average of the solutions and use a learning rate of $\frac{1}{\sqrt{t}}$. However, this is an easier problem to solve (because it is 1-*strongly convex*), so we can use a learning rate that goes to zero faster and we can avoid the averaging.

(a) [10pts] Implement the algorithm in Algorithm 1 in a Matlab function with prototype

```
[alpha, b, obj, zeroOneAverageLoss] = train_ksvm_sd(X, y, T, c, gamma)
```

where `alpha` $\in \mathbb{R}^m$ and `b` $\in \mathbb{R}$ are the solution found by subgradient descent, `obj` $\in \mathbb{R}^T$ is the vector of objective function values, `zeroOneAverageLoss` $\in \mathbb{R}^T$ is the vector of the average 0/1 loss on the training samples, `X` is the matrix of the training samples $\in \mathbb{R}^{m \times d}$, `y` is the vector of labels $\in \mathbb{R}^m$, `T` is the maximum number of iterations to run, `c` is the trade-off hyperparameter in Eq. (1), `gamma` is the bandwidth parameter of the Gaussian kernel. There is no skeleton code this time.

(b) [2pts] Create a plot which shows how the objective function F evolves with iteration number t. Is the objective function decreasing? Should it be decreasing? Discuss its behavior.

(c) [2pts] Create a plot which shows how the average 0/1 loss on the training set evolves with iteration number. Is it decreasing? Should it be decreasing? Discuss its behavior.

(d) [2pts] After the subgradient descent algorithm terminates, report the final training error measured with the 0/1 loss. Discuss your observations.

(e) [6pts] *(Visualization of decision boundary and margins)* Plot on the same graph the following: (1) the training set (use different colors for the two classes) and (2) the decision boundary and the margins of the trained Kernel SVM. *Hint:* To plot the decision boundary and the margins, you can use the function `contour`, look for its Matlab help.

Problem 6.3 [13pts] *(Questions)*
Clearly explain your answers.

(a) [2pts] Consider the soft-margin SVM formulation. Does the variance increases or decreases with c?

(b) [2pts] Consider a linearly separable training set of m samples. Let's now gather other n different training samples for the same classification problem, to have a total of $m + n$ samples. Is the new dataset still linearly separable?

(c) [2pts] We have a linearly separable dataset. We run the Perceptron till convergence and measure the margin of the obtained solution, that is we measure the minimum distance between the hyperplane and the training points. Will this margin be smaller, equal, or bigger than the margin of the hard-margin SVM solution on the same dataset?

(d) [2pts] You have a binary classification dataset with roughly equal number of positive and negative examples. You use a soft-margin SVM with some setting of c and the 0/1 loss on the training and validation sets are close to 50%. You want to try a different value of c to try to improve the performance: Would you decrease or increase c? Why?

(e) [2pts] You have a binary classification dataset with roughly equal number of positive and negative examples. You use a soft-margin SVM with some setting of c and the 0/1 loss on the training set is close to 0 and validation error is close to 50%. You want to try a different value of c to try to improve the performance: Would you decrease or increase c? Why?

(f) [3pts] ☕ Using the definition of subgradient we gave in class, find a subgradient of the function $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2$ in $\boldsymbol{x} = \boldsymbol{0} \in \mathbb{R}^d$. This would be useful if we want to change the regularizer from $\|\boldsymbol{x}\|_2^2$ to $\|\boldsymbol{x}\|_2$ and use subgradient descent. Hint: there are infinite subgradients in that point, any of them will do.

Code-submission via Blackboard: You must prepare 1 file: `train_ksvm_sd.m` for Problem 6.2(a). Place it in a **single** directory which should be zipped and uploaded into Blackboard. Your directory must be named as follows: `<yourBUemailID>_hwX` where `X` is the homework number. For example, if your BU email address is `charles500@bu.edu` then for homework number 6 you would submit a single directory named: `charles500_hw6.zip` which contains all the MATLAB code (and only the code).

Three corresponding skeleton code files are provided for your reference. Reach-out to the TAs via Piazza and their office/discussion hours for questions related to coding.