CS 189:289A Introduction to Machine Learning
Fall 2021 Jennifer Listgarten and Jitendra Malik

HW5

Due 11/15 at 11:59pm

- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.

Deliverables:

1. Submit a PDF of your homework to the Gradescope assignment entitled "HW5 Write-Up". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
   - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats. *"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."*
   - **Replicate all your code in an appendix**. Begin code for each coding question in a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from appendix to correct questions.

1 Kernel SVM and Kernel Ridge Regression

In this problem, we will give a derivation of kernel SVM and kernel ridge regression from the view of function approximation with penalization. The objective function of a linear SVM can be interpreted as the hinge loss combined with L_2 regularization over the space of linear functions. The objective function of a kernel SVM can be interpreted in the same way: hinge loss plus L_2 regularization over a particular space of functions defined by a kernel function.

Assume we are doing classification or regression over $\mathbb{R}^d$. We first introduce the following abstract vector space:

$$H = \{f : \mathbb{R}^d \to \mathbb{R} : f(x) = \sum_{m=1}^{M} \alpha_m k(x, y_m) : \alpha_i \in \mathbb{R}, M \in \mathbb{N}, y_m \in \mathbb{R}^d\}, \tag{1}$$

where $k(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a kernel function that satisfies the property $k(x, y) = k(y, x)$ for any $x, y \in \mathbb{R}^d$ and for any distinct $y_1, y_2, \ldots, y_M \in \mathbb{R}^d$, the matrix $K \in \mathbb{R}^{M\times M}$ defined by $K_{ij} = k(y_i, y_j)$ is positive definite.

This is a vector space since it has a zero and linear combinations make sense. It can be infinite-dimensional however since there are lots of possible y_m.

(a) Now we introduce an inner product on the above vector space H. We define the inner product between any two functions

$$f(x) = \sum_{m=1}^{M} \alpha_m k(x, y_m),\ g(x) = \sum_{s=1}^{S} \beta_s k(x, x_s) \in H,$$

in H as

$$\langle f, g\rangle_H = \sum_{m=1}^{M}\sum_{s=1}^{S} \alpha_m\beta_s k(y_m, x_s). \tag{2}$$

Show that the defined inner product is valid. That is, it satisfies the symmetry, linearity and positive-definiteness properties stated below: For any for any $f, g, h \in H$ and any $a \in \mathbb{R}$, we have

- $\langle f, g\rangle_H = \langle g, f\rangle_H$.
- $\langle af, g\rangle_H = a\langle f, g\rangle_H$ and $\langle f + h, g\rangle_H = \langle f, g\rangle_H + \langle h, g\rangle_H$.
- $\langle f, f\rangle_H \geq 0$; $\langle f, f\rangle = 0$ if and only if f is an constant zero function.

What is the norm of the function f? (The natural norm in an inner product space is defined as $\|f\|_H = \sqrt{\langle f, f\rangle_H}$.)

Solution:

$$\langle f, g\rangle_H = \sum_{m=1}^{M}\sum_{s=1}^{S} \alpha_m\beta_s k(y_m, x_s)$$

$$= \sum_{s=1}^{S}\sum_{m=1}^{M}\beta_s\alpha_m k(x_s, y_m)$$
$$= \langle g, f\rangle_H,$$

where the second equation follows from the symmetry of kernel function.

$$\begin{aligned}\langle af, g\rangle_H &= \sum_{m=1}^{M}\sum_{s=1}^{S}(a\alpha_m)\beta_s k(y_m, x_s)\\ &= a\sum_{m=1}^{M}\sum_{s=1}^{S}\alpha_m\beta_s k(y_m, x_s)\\ &= a\langle f, g\rangle_H,\end{aligned}$$

We write h as $h(x) = \sum_{t=1}^{N}\gamma_t k(x, z_t)$. Then

$$(f+h)(x) = \sum_{m=1}^{M}\alpha_m k(x, y_m) + \sum_{t=1}^{N} k(x, z_t).$$

$$\begin{aligned}\langle f+h, g\rangle_H &= \sum_{m=1}^{M}\sum_{s=1}^{S}\alpha_m\beta_s k(y_m, x_s) + \sum_{t=1}^{T}\sum_{s=1}^{S}\gamma_t\beta_s k(z_t, x_s)\\ &= \langle f, g\rangle_H + \langle h, g\rangle_H,\end{aligned}$$

where the first equality follows directly from the definition of the inner product.

Without loss of generality, we assume x_i are distinct. Then

$$\begin{aligned}\langle f, f\rangle_H &= \sum_{m=1}^{M}\sum_{s=1}^{M}\alpha_m\alpha_s k(y_m, y_s)\\ &= \alpha^T K\alpha \geq 0,\end{aligned}$$

where $\alpha \in \mathbb{R}^n$ with ith element being α_i and $K \in \mathbb{R}^{n\times n}$ with i, jth element being $k(y_i, y_j)$. Because K is positive definite, the last inequality follows. And the equality holds if and only if $\alpha = 0$, which is equivalent to f is a constant zero function.

The norm of a function f with $f(x) = \sum_{m=1}^{M}\alpha_m k(x, y_m)$ is

$$\|f\|_H = \sqrt{\langle f, f\rangle_H} = \sqrt{\sum_{m=1}^{M}\sum_{s=1}^{M}\alpha_m\alpha_s k(y_m, y_s)} = \sqrt{\alpha^T K\alpha}. \tag{3}$$

(b) Below we introduce a general optimization problem over the space H. We will see many kernalized machine learning algorithms, including kernel SVM, can be written in the following form: Given a data set with N points $x_i, y_i, i = 1, \ldots, N$, the optimization problem is

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2, \tag{4}$$

where L is any loss function on pairs of real numbers. Surprisingly, despite being an optimization over an infinite-dimensional space of functions, $f \in H$, it turns out the solution to this problem has the form:

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{5}$$

That is, the soluton can be expressed as a weighted sum of kernel functions based on the training points. This phenomenon that reduces an infinite-dimensional optimization problem to be finite-dimensional is called the *kernel property*.

With this machinery in place, we can derive numerous kernalized machine learning problems simply by changing the loss function, $L(y_i, f(x_i))$, in equation 4. The kernel SVM, for example, is nothing but defining the loss function L concretely as a hinge loss:

$$L(y, f(x)) = \max(0, 1 - y f(x)). \tag{6}$$

In other words, kernel SVM is

$$\min_{f \in H} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i f(x_i)) + \lambda \|f\|_H^2. \tag{7}$$

Show kernel SVM is of the form

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha. \tag{8}$$

Solution: Any $f \in M$ can be expressed as $f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i)$ for some $\alpha \in \mathbb{R}^N$. Then $\|f\|_H^2 = \alpha^T K \alpha$ from Part (a).

The optimization problem

$$\min_{f \in M} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda \|f\|_H^2, \tag{9}$$

can be rewriten as

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{N} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha. \tag{10}$$

Write out L specifically as a hinge loss, we get the kernel SVM:

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \alpha^T K \alpha. \tag{11}$$

(c) (Kernel ridge regression) Take L as l_2 loss, that is, $L(a,b) := \|a-b\|_2^2$. **Show that optimization problem of kernel ridge regression has the following form**

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{N}\|Y - K\alpha\|_2^2 + \lambda \alpha^T K \alpha, \tag{12}$$

where $Y \in \mathbb{R}^n$ with the ith element of Y being y_i. **Derive a closed form solution for the kernel ridge regression:**

$$\alpha = (K + \lambda N I_N)^{-1} Y, \tag{13}$$

where I_N is an n-dimensional identity matrix.

Solution: Plugging L with the l_2 loss, we get

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^N L(y_i, \sum_{j=1}^N \alpha_j k(x_i, x_j)) &= \frac{1}{N}\sum_{i=1}^N (y_i - \sum_{j=1}^N \alpha_j k(x_i, x_j))^2 \\ &= \frac{1}{N}\|K\alpha - Y\|^2\end{aligned}$$

Then the optimization problem can be written as

$$\min_{\alpha \in \mathbb{R}^N} \frac{1}{N}\|K\alpha - Y\|^2 + \lambda \alpha^T K \alpha. \tag{14}$$

Write $F(\alpha) = \frac{1}{N}\|K\alpha - Y\|^2 + \lambda \alpha^T K \alpha$. Then

$$\begin{aligned}\nabla_\alpha F(\alpha) &= \nabla_\alpha \frac{1}{N}(\alpha^T K^2 \alpha - 2Y^T K \alpha + Y^T Y) + \nabla_\alpha(\alpha^T K \alpha) \\ &= \frac{2}{N}(K^2\alpha - KY) + 2\lambda K \alpha \\ &= 2(\frac{1}{N}K^2 + \lambda K)\alpha - \frac{2}{N}KY.\end{aligned}$$

The Hessian is $2(\frac{1}{N}K^2 + \lambda K)$, which is positive definite. Therefore, the optimum is obtained when $\nabla_\alpha F(\alpha) = 0$. That is

$$\alpha = (K + \lambda N I_N)^{-1} Y. \tag{15}$$

(d) (Polynomial Regression from a kernelized view) In this part, we will show that polynomial regression with a particular Tiknov regularization is the same as kernel ridge regression with a polynomial kernel for second-order polynomials. Recall that a polynomial kernel function on $\mathbb{R}^d$ is defined as

$$k(x,y) = (1 + x^T y)^2, \tag{16}$$

for any $x, y \in \mathbb{R}^d$. Given a dataset (x_i, y_i) for $i = 1, 2, \ldots, N$. **Show the solution to kernel ridge regression is the same as the least square solution to polynomial regression for $d = 2$ given the right choice of Tikhonov regularization for the polynomial regression.** That is, show for any new point x given in the prediction stage, both methods give the same y. What is the Tikhonov regularization matrix here?

Hint: You may or may not use the following matrix identity:

$$A(aI_d + A^T A)^{-1} = (aI_N + AA^T)^{-1} A, \tag{17}$$

for any matrix $A \in \mathbb{R}^{n\times d}$ and any positive real number a.

Solution: Define a vector-valued function from $\mathbb{R}^2 \to \mathbb{R}^6$ such that

$$\phi(a) = (1, a_1^2, a_2^2, \sqrt{2}a_1, \sqrt{2}a_2, \sqrt{2}a_1 a_2)^T$$

for $a = (a_1, a_2)^T$.

Define a matrix in $\mathbb{R}^{N\times 6}$ such that

$$\Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} \tag{18}$$

We observe that $k(x, y) = \phi(x)^T \phi(y)$. For a kernel matrix $K \in \mathbb{R}^{n\times n}$ with $K_{ij} = k(x_i, x_j)$, we have

$$K_{ij} = \phi(x_i)^T \phi(x_j) = (\Phi\Phi^T)_{ij}. \tag{19}$$

That is

$$K = \Phi\Phi^T.$$

Recall the solution to Kernel ridge regression is a function f with

$$\begin{aligned} f(x) &= \sum_{i=1}^{N} \alpha_i k(x_i, x) \\ &= \sum_{i=1}^{N} \alpha_i \phi(x_i)^T \phi(x) \\ &= \alpha^T \Phi\phi(x), \end{aligned}$$

where

$$\alpha = (K + \lambda N I_N)^{-1} Y. \tag{20}$$

Therefore, we can write $f(x)$ as

$$f(x) = Y^T (\Phi\Phi^T + \lambda N I_N)^{-1} \Phi\phi(x). \tag{21}$$

For polynomial regression, define a vector-valued function from $\mathbb{R}^2 \to \mathbb{R}^6$ such that

$$\tilde{\phi}(a) = (1, a_1^2, a_2^2, a_1, a_2, a_1 a_2)^T$$

for $a = (a_1, a_2)^T$.

Define a matrix in $\mathbb{R}^{N\times 6}$ such that

$$\tilde{\Phi} = \begin{bmatrix} \tilde{\phi}(x_1)^T \\ \tilde{\phi}(x_2)^T \\ \vdots \\ \tilde{\phi}(x_N)^T \end{bmatrix} \tag{22}$$

Observe the relationship between ϕ and $\tilde{\phi}$: We have

$$\tilde{\phi}(x) = D\phi(x), \tilde{\Phi} = \Phi D, \tag{23}$$

for a diagonal matrix $D \in \mathbb{R}^{6\times 6}$, with

$$D = \text{diag}(1, 1, 1, 1/\sqrt{2}, 1/\sqrt{2}, 1/\sqrt{2}).$$

A polynomial regression is nothing but replacing linear feature X by $\tilde{\phi}(X) \in \mathbb{R}^6$ and add a Tikhonov regularization over the parameters $w \in \mathbb{R}^6$. Recall in a previous homework, we've shown it has a closed form solution

$$w = (\tilde{\Phi}^T\tilde{\Phi} + \Lambda)^{-1}\tilde{\Phi}^T Y, \tag{24}$$

for a polynomial regression with Tikhonov regularization matrix $\Lambda \in \mathbb{R}^{d\times d}$. Let Λ be a diagonal matrix defined by

$$\Lambda = \text{diag}(\lambda N, \lambda N, \lambda N, \lambda N/2, \lambda N/2, \lambda N/2). \tag{25}$$

Then

$$\Lambda = D(\lambda N I_6)D. \tag{26}$$

Then the predictor produced by Tikhonov regression is

$$\begin{aligned} g(x) &= w^T\tilde{\phi}(x) \\ &= [(\tilde{\Phi}^T\tilde{\Phi} + \Lambda)^{-1}\tilde{\Phi}^T Y]^T\tilde{\phi}(x) \\ &= Y^T\tilde{\Phi}(\tilde{\Phi}^T\tilde{\Phi} + \Lambda)^{-1}\tilde{\phi}(x) \\ &= Y^T\Phi D(D\Phi^T\Phi D + D(D^{-1}\Lambda D^{-1})D)^{-1}D\phi(x) \\ &= Y^T\Phi DD^{-1}(\Phi^T\Phi + (D^{-1}\Lambda D^{-1}))^{-1}D^{-1}D\phi(x) \\ &= Y^T\Phi(\Phi^T\Phi + (D^{-1}\Lambda D^{-1}))^{-1}\phi(x) \\ &= Y^T\Phi(\Phi^T\Phi + \lambda N I_6)^{-1}\phi(x) \\ &= Y^T(\Phi\Phi^T + \lambda N I_N)^{-1}\Phi\phi(x) = f(x), \end{aligned}$$

where the last equation follows from the hint. Hence, we have shown the equivalence between the two predictors.

(e) (Bonus) In general, for any polynomial regression with pth order polynomial on $\mathbb{R}^d$, with a selected Tikhonov regression, we can show the equivalence between it and kernel ridge regression with a polynomial kernel of order p. (You are not required to show this.) **Comment on the computational complexity of doing least squares for polynomial regression with a Tikhonov regression directly and that of doing kernel ridge regression in the training stage.** (That is, the complexity of finding α and finding w.) **Compare with the computational complexity of actually doing prediction as well.**

Solution: In the polynomial regression with Tikhonov regularization, for any data point (x_i, y_i), computing its polynomial features of order p takes $O(d^p)$. The complexity of solving least square is $O(d^{3p} + d^p n)$. The total complexity is $O(d^{3p} + d^p n)$.

In the kernel ridge regression, the complexity of computing the kernel matrix is $O(n^2 d \log p)$. The complexity of getting α after that is $O(n^3)$. The total complexity is $O(n^3 + n^2 d \log p)$. It only has a log dependence on p but cubic dependence on n. Kernel ridge regression is preferred when p is large.

2 Kernel SVMs via Gradient Descent

Feature-based SVMs take in an $n \times d$ data matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix}$ and an n-dimensional vector $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ of binary labels (i.e. each label $y_i \in \{-1, 1\}$). The SVM aims to learn a d-dimensional weight vector $\mathbf{w}$, classifying inputs $\mathbf{x}$ as $\text{sign}(\mathbf{w}^\top \mathbf{x})$. For simplicity, we will ignore the bias term in this problem until the last two parts.

For soft-margin SVMs, we aim to choose $\mathbf{w}$ to minimize the cost function

$$L(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i),$$

where S is the set of indices i such that $y_i(\mathbf{w}^\top \mathbf{x}_i) < 1$.

In this problem, we will examine how gradient descent can be used to minimize $L(\mathbf{w})$, before generalizing to learn kernel SVMs.

(a) **Give the gradient update step (i.e. $\mathbf{w}^{(t+1)} = \cdots$) at time t for $\mathbf{w}$ to minimize $L(\mathbf{w})$ as defined above, with a step size of η.** Let $S^{(t)}$ be the set S (as defined previously) at any time t. Assume that you will update $S^{(t+1)}$ in the natural manner after you have updated $\mathbf{w}$, but will use $S^{(t)}$ during the gradient update step that computes $\mathbf{w}^{(t+1)}$.

Solution: Taking the gradient, we find that

$$\nabla_{\mathbf{w}} L = \mathbf{w} - C \sum_{i \in S} y_i \mathbf{x}_i. \tag{27}$$

Consequently, the gradient update step at time t is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - \eta \left(\mathbf{w}^{(t)} - C \sum_{i \in S^{(t)}} y_i \mathbf{x}_i \right) = (1 - \eta)\mathbf{w}^{(t)} + \eta C \sum_{i \in S^{(t)}} y_i \mathbf{x}_i.$$

(b) Imagine that we start with some weight vector $\mathbf{w}^{(t)} = \mathbf{X}^\top \mathbf{a}^{(t)}$ that is a linear combination of the $\{\mathbf{x}_i\}$. (Notice that if we start with the zero vector as our initial condition for $\mathbf{w}^{(0)}$, this is true as a base case.)

For notational convenience, define $\mathbf{z}^{(t)}$ to be an n-dimensional vector with the ith component equal to y_i if $y_i(\mathbf{X}^\top \mathbf{a}^{(t)})^\top \mathbf{x}_i = y_i(\mathbf{w}^{(t)})^\top \mathbf{x}_i < 1$, and equal to zero otherwise.

Show that after one gradient step, $\mathbf{w}^{(t+1)}$ will remain a linear combination of the $\{\mathbf{x}_i\}$, and so is expressible as $\mathbf{X}^\top \mathbf{a}^{(t+1)}$ for some "dual weight vector" $\mathbf{a}^{(t+1)}$. Then write down the gradient-descent update step for the dual weights $\mathbf{a}^{(t)}$ directly without referring to $\mathbf{w}^{(t)}$ at all. In other words, tell us how $\mathbf{a}^{(t+1)}$ is obtained from the step size η, the previous dual weights $\mathbf{a}^{(t)}$, and the $\mathbf{z}^{(t)}$ vector.

Solution: We have $\mathbf{w}^{(t)} = \mathbf{X}^\top \mathbf{a}^{(t)}$. Thus, from the previous part,

$$\mathbf{w}^{(t+1)} = (1-\eta)\mathbf{X}^\top \mathbf{a}^{(t)} + \eta C \sum_{i \in S^{(t)}} y_i \mathbf{x}_i.$$

Here, we use the definition of $\mathbf{z}^{(t)}$ and notice that $\mathbf{X}^\top \mathbf{z}^{(t)} = \mathbf{0} + \sum_{i \in S^{(t)}} y_i \mathbf{x}_i$. This means we can rewrite the update as:

$$\begin{aligned}\mathbf{w}^{(t+1)} &= (1-\eta)\mathbf{X}^\top \mathbf{a}^{(t} + \eta C \mathbf{X}^\top \mathbf{z}^{(t)} \\ &= \mathbf{X}^\top \left((1-\eta)\mathbf{a}^{(t)} + \eta C \mathbf{z}^{(t)}\right).\end{aligned}$$

Thus by remembering the definition of the dual weights $\mathbf{a}$:

$$\mathbf{a}^{(t+1)} = (1-\eta)\mathbf{a}^{(t)} + \eta C \mathbf{z}^{(t)}.$$

(c) In order to understand how we can use gradient descent with Kernel SVMs, we want to show that $\mathbf{z}^{(t)}$ can be computed knowing only the Gram matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$, the labels $\mathbf{y}$, and the dual weights $\mathbf{a}^{(t)}$. To argue this, we notice that $\mathbf{z}_i^{(t)}$ is either y_i or 0 depending on whether

$$y_i(\mathbf{X}^\top \mathbf{a}^{(t)})^\top \mathbf{x}_i < 1 \tag{28}$$

How can you check this condition (28) **if you only had access to $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$, the index i, the labels y, and the dual weights $\mathbf{a}^{(t)}$, but did not have access to the features $\mathbf{x}_i$?**

Solution: Observe that $\mathbf{z}_i$ depends only on the quantity $y_i(\mathbf{X}^\top \mathbf{a}^{(t)})^\top \mathbf{x}_i$, which can be rewritten as

$$y_i(\mathbf{a}^{(t)})^\top \mathbf{X}\mathbf{x}_i = y_i(\mathbf{a}^{(t)})^\top (\mathbf{X}\mathbf{X}^\top)_i,$$

where $(\mathbf{X}\mathbf{X}^\top)_i$ is the ith column of $\mathbf{X}\mathbf{X}^\top$.

This follows from the nature of $\mathbf{X}^\top = [\mathbf{x}_1 \mathbf{x}_2 \cdots \mathbf{x}_n]$ and the nature of matrix multiplication.

Consequently, using the definition of K, we can simply check $y_i \mathbf{a}^{(t)} K_i$ to see if this is less than 1. Here K_i refers to the i-th column of the Gram matrix K.

(d) Recall that the solution for the hyperplane is given by

$$\arg\min_{\mathbf{w}} L(\mathbf{w}) = \arg\min_{\mathbf{w}} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i)\right). \tag{29}$$

Show that the optimization problem can be equivalently expressed as

$$\arg\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i)) + \lambda \|\mathbf{w}\|^2. \tag{30}$$

for an appropriate choice of λ.

Solution: Since $C > 0$, we can multiply by $\frac{1}{NC}$ without changing the value

$$\arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) = \arg\min_{\mathbf{w}} \frac{1}{2NC}\|\mathbf{w}\|^2 + \frac{1}{N} \sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i). \tag{31}$$

Now, letting $\lambda = \frac{1}{2NC}$ and remembering that,

$$\sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) = \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i)),$$

we have,

$$\begin{aligned} \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) &= \arg\min_{\mathbf{w}} \frac{1}{2NC}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) \quad (32)\\ &= \arg\min_{\mathbf{w}} \frac{1}{2NC}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) \quad (33)\\ &= \arg\min_{\mathbf{w}} \lambda\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) \quad (34)\end{aligned}$$

(e) Part (a-c) suggest that we can use the kernel trick to compute gradient steps for the dual weights $\mathbf{a}^{(t)}$ (explained in part b) so long as we can compute the inner products of any pair of featurized observations. So if we have access to an appropriate similarity kernel function $k(\cdot,\cdot)$, we can use it to compute $k(\mathbf{x}_i, \mathbf{x}_j)$ whenever we need (i, j) entries of the Gram matrix $\mathbf{K}$. This gives us a way to train a soft-margin kernel SVM using gradient descent. **Show that the kernel trick combined with the equivalent optimization problem derived in Part (d) produces the kernel SVM derived in Problem (1):**

$$L(\mathbf{a}) = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i \sum_{j=1}^{N} \mathbf{a}_j k(\mathbf{x}_i, \mathbf{x}_j)) + \lambda \mathbf{a}^T K \mathbf{a}. \quad (35)$$

Solution: From Part (d) we have

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i) + \lambda\|\mathbf{w}\|^2.$$

Substituting $\mathbf{w} = \mathbf{X}^\top \mathbf{a}$ we get,

$$L(\mathbf{a}) = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{a}^\top \mathbf{X}\mathbf{x}_i) + \lambda \mathbf{a}^\top \mathbf{X}\mathbf{X}^\top \mathbf{a}.$$

Now by substituting the Gram matrix in place of $\mathbf{x}$ and $\mathbf{X}$ as in Part (c),

$$\begin{aligned} L(\mathbf{a}) &= \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{a}^\top \mathbf{K}_i) + \lambda \mathbf{a}^\top \mathbf{K}\mathbf{a}\\ &= \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i \sum_{j=1}^{N} \mathbf{a}_j k(\mathbf{x}_i, \mathbf{x}_j)) + \lambda \mathbf{a}^\top \mathbf{K}\mathbf{a}\end{aligned}$$

(f) We will now consider the case when we have a bias term b, so our cost function becomes

$$L(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i \in S} 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b),$$

where S is the set of indices i such that $y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1$. **Write the gradient update step for b.** (i.e. $b^{(t+1)} = \cdots$) As before, assume that you will update $S^{(t+1)}$ in the natural manner after you have updated b, but will use $S^{(t)}$ during this gradient update step.

For a fixed value of w, when does $\frac{\partial L}{\partial b} = 0$?

(HINT: This has something to do with the balance of the points in the S that corresponds to $\mathbf{w}, b$.)

Solution: Differentiating, we obtain

$$b^{(t+1)} = b^{(t)} + \eta C \sum_{i \in S} y_i.$$

Remembering that S keeps track of all the margin violators, we see that $b^{(t+1)} = b^{(t)}$ when there are equallly many points of each class in the margin generated by $\mathbf{w}^{(t)}, b^{(t)}$. This is because the y_i are ± 1.

Notice also an interesting challenge here. If the step size η is a constant, then b is forced to move on a lattice with step size ηC. This combined with the fact above that the bias won't move if the margin is balanced tells us that this particular approach to optimizing isn't guaranteed to maximize the margin.

(g) After our gradient descent converges or we decide to stop it, we need to use the learned dual weights $\hat{\mathbf{a}}$ and bias $\hat{b}$ to classify a test point x_{test}. **Express its predicted class $y_{pred} \in \{-1, +1\}$ in terms of the dual weights $\hat{\mathbf{a}}$, the similarity kernel function $k(\cdot, \cdot)$, the training data $\{(x_i, y_i)\}$, the test input x_{test}, and the bias $\hat{b}$.**

You do not have access to the features for the test point, and so you are going to have to use the $k(x_{test}, x_i)$ as a part of your solution.

(HINT: Don't forget that the final class prediction has to be either +1 or -1. This is a binary classification problem. Also, don't forget to use the y_i appropriately.)

Solution: If we had features, we would use

$$y_{pred} = \text{sign}(\hat{\mathbf{w}}^\top \mathbf{x}_{test} + \hat{b}).$$

But we know that our $\mathbf{w}^{(t)} = \mathbf{X}^\top \mathbf{a}^{(t)}$ and so $\hat{\mathbf{w}}^\top = \hat{\mathbf{a}}^\top \mathbf{X}$. Plugging that in, we get:

$$\begin{aligned} y_{pred} &= \text{sign}\left(\hat{\mathbf{w}}^\top \mathbf{x}_{test} + \hat{b}\right) \\ &= \text{sign}\left(\hat{\mathbf{a}}^\top \mathbf{X}\mathbf{x}_{test} + \hat{b}\right) \end{aligned}$$

$$
\begin{aligned}
&= \operatorname{sign}\left(\hat{\mathbf{a}}^\top \begin{bmatrix} \langle \mathbf{x}_{test}, \mathbf{x}_1 \rangle \\ \vdots \\ \langle \mathbf{x}_{test}, \mathbf{x}_i \rangle \\ \vdots \\ \langle \mathbf{x}_{test}, \mathbf{x}_n \rangle \end{bmatrix} + \hat{b}\right) \\
&= \operatorname{sign}\left(\hat{b} + \sum_{i=1}^{n} \hat{\mathbf{a}}_i \langle \mathbf{x}_{test}, \mathbf{x}_i \rangle\right) \\
&= \operatorname{sign}\left(\hat{b} + \sum_{i=1}^{n} \hat{\mathbf{a}}_i k(\mathbf{x}_{test}, \mathbf{x}_i)\right),
\end{aligned}
$$

where at the end, we used the standard substitution of inner-products with kernel similarity.

Notice that there are no y_i in this expression. All the impact of the y_i is captured in the dual weights $\hat{\mathbf{a}}$ in this gradient-descent approach.

3 Running Time of k-Nearest Neighbor and Kernelized Nearest Neighbor Search Methods

The method of k-nearest neighbors is a fundamental conceptual building block of machine learning. A classic example is the k-nearest neighbor classifier, which is a non-parametric classifier that finds the k closest examples in the training set to the test example, and then outputs the most common label among them as its prediction. Generating predictions using this classifier requires an algorithm to find the k closest examples in a possibly large and high-dimensional dataset, which is known as the k-nearest neighbor search problem. More precisely, given a set of n points, $\mathcal{D} = \{\mathbf{x}_1 \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ and a test point $\mathbf{q} \in \mathbb{R}^d$, there are 2 steps to decide what class to predict:

1. Find the k training points nearest $\mathbf{q}$.
2. Return the class with the most votes from the k training points.

(a) What is the runtime to classify a newly given test point q, using Euclidean distance? (**Hint:** Use heaps to keep track of the k smallest distances.)

Solution: To classify a point q, we first need to calculate the distances to every other point. The Euclidean distance is $\sqrt{(x_1 - q_1)^2 + (x_2 - q_2)^2 + \ldots + (x_d - q_d)^2}$, and so for one point, takes $O(d)$ time to compute. For n points, we get $O(nd)$ time.

As we compute the distances, we can use a max-heap to keep track of the k shortest distances seen so far. In the worst case, we have n insertions, taking $O(\log k)$ time each.

This gives us a total run time of $O(nd + n \log k)$.

(b) Let us now 'lift' the points into a higher dimensional space by adding all possible mononomials of the original features with degree at most p. What dimension would this space be? What would the new runtime for classification of a test point q be, in terms of n, d, k, and p?

Solution: Our training points now have $O(d^p)$ features. Therefore, it takes $O(d^p)$ time to compute the Euclidean distance between two points. For n points, the total time is $O(nd^p)$.

We again use a max-heap to keep track of the k shortest distances seen so far. In the worst case, we have n insertions, taking $O(\log k)$ time each.

This gives us a total runtime of $O(nd^p + n \log k)$.

(Alternative Solution): It is also possible to create a heap of all n elements in $O(n)$ time. We can then get the top k elements in $k \log n$ time, giving a total runntime of $O(nd + k \log n)$.

(c) Instead, we can use the polynomial kernel to compute the distance between 2 points in the 'lifted' $O(d^p)$-dimensional space without having to move all of the points into the higher dimensional space. Using the polynomial kernel, $k(x, y) = (x^T y + \alpha)^p$ instead of Euclidean distance, what is the runtime for k-NN to classify a test point q?

Solution: Let the $O(d^p)$-dimensional 'lifted' point corresponding to a point x in d-dimensional space be represented by $\Phi(x)$. The Euclidean distance squared between the test point q and a training point x in the 'lifted' space is $||\Phi(q) - \Phi(x)||^2$. We then have:

$$\begin{aligned}||\Phi(q) - \Phi(x)||^2 &= \Phi^T(q)\Phi(q) - 2\Phi^T(q)\Phi(x) + \Phi^T(x)\Phi(x)\\ &= k(q,q) - 2k(q,x) + k(x,x)\end{aligned}$$

We note that the kernel calculations take $O(d)$ time. Therefore, for n points, we get $O(nd)$ time. Following the same logic as before, we end up with a total runtime of $0(nd + n\log k)$.

Note: We assume that exponentiation (a^b) of two floating-point numbers a, b takes $O(1)$ time.

(d) Decades of research have focused on devising a way of preprocessing the data so that the k-nearest neighbors for each query can be found efficiently. "Efficient" means the time complexity of finding the k-nearest neighbors is lower than that of the naïve exhaustive search algorithm—meaning that the complexity must be *sublinear* in n.

Many efficient algorithms for k-nearest neighbor search rely on a divide-and-conquer strategy known as space partitioning. The idea is to divide the feature space into cells and maintain a data structure that keeps track of the points that lie in each. Then, to find the k-nearest neighbors of a query, these algorithms look up the cell that contains the query and obtain the subset of points in $\mathcal{D}$ that lie in the cell and adjacent cells. Adjacent cells must be included in case the query point is in the corner of its cell. Then, exhaustive search is performed on this subset to find the k points that are the closest to the query.

For simplicity, we'll consider the special case of $k = 1$ in the following questions, but note that the various algorithms we'll consider can be easily extended to the setting with arbitrary k. We first consider a simple partitioning scheme, where we place a Cartesian grid (a rectangular grid consisting of hypercubes) over the feature space.

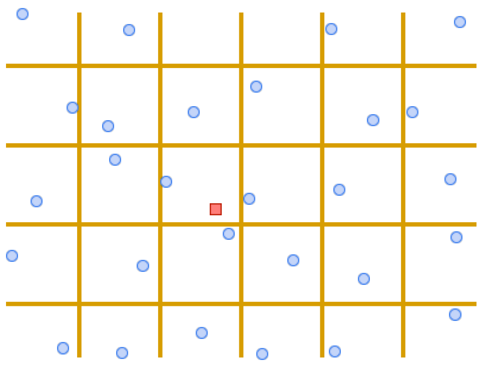


Figure 1: Illustration of the space partitioning scheme we consider. The data points are shown as blue circles and the query is shown as the red square. The cell boundaries are shown as gold lines.

How many cells need to be searched in total if the data points are one-dimensional? Two-dimensional? d-dimensional? If each cell contains one data point, what is the time complexity for finding the 1-nearest neighbor in terms of d, assuming accessing any cell takes constant time?

Solution: In 1D, there are two adjacent cells, so three cells need to be searched. In 2D, there are eight adjacent cells, so nine cells need to be searched. By induction, it is easy to see that in d dimensions, there are 3^d cells that need to be searched in $\mathbb{R}^d$ since you need to search the neighbors in each new dimension, both above and below you. This multiplies the number of cells by three with each additional dimension. The reason we have to look at all the neighbors is that our query point might be in a corner, and the points in the adjacent cells might also be in corners. So, the exact configuration matters if we want to find the true nearest neighbor.

Since each cell contains a data point, there are 3^d data points that need to be searched exhaustively. Because computing the distance for each data point takes $\Theta(d)$ time and finding the minimum of 3^d distances takes $O(3^d)$ time, the time complexity is $O(d3^d + 3^d) = O(d3^d)$.

4 Random Forests: Entropy and Bagging

This problem helps build intuition behind the core techniques that make random forests work: entropy and bagging. We will first clarify the concepts of surprise and entropy. Recall that entropy is one of the standards for us to split the nodes in decision trees until we reach a certain level of homogeneity.

(a) Suppose you have a bag of balls, all of which are black. How surprised are you if you take out a black ball?

Solution: 0. We aren't surprised at all when events with probability 1 occur.

(b) With the same bag of balls, how surprised are you if you take out a white ball?

Solution: ∞. We are infinitely surprised when an event with probability 0 occurs.

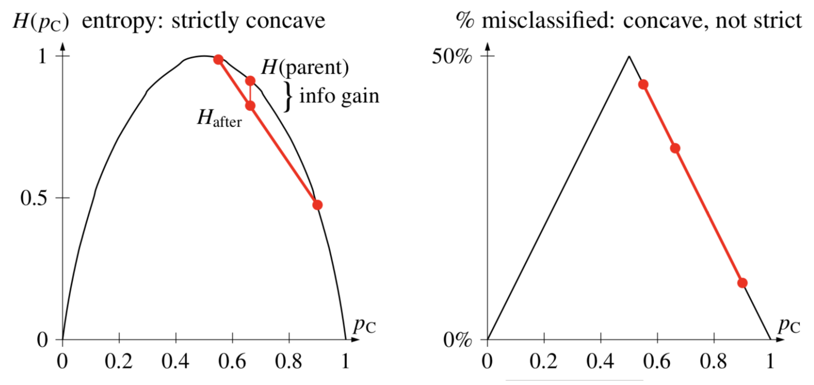
(c) Now we have 10 balls in the bag, each of which is black or white. Under what color distribution(s) is the entropy of the bag minimized? And under what color distribution(s) is the entropy maximized? Calculate the entropy in each case.
Recall: The entropy of an index set S is a measure of expected surprise from choosing an element from S; that is,

$$H(S) = -\sum_{C} p_C \log_2(p_C), \text{ where } p_C = \frac{|i \in S : y_i = C|}{|S|}.$$

Solution: The entropy is minimized when, for example, all the balls are black or all the balls are white. In this case the entropy is 0. The entropy is maximized when half the balls are black and half the balls are white, in which case the entropy is $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$.

(d) Draw the graph of entropy $H(p_c)$ when there are only two classes C and D, with $p_D = 1 - p_C$. Is the entropy function strictly concave, concave, strictly convex, or convex? Why? What is the significance?
Hint: For the significance, recall the information gain.

Solution: The function is strictly concave. Notice that the function $-x \log x$ is strictly concave



in [0, 1], and a sum of strictly concave functions is strictly concave.
Significance: (from lecture) Suppose we pick two points on the entropy curve, then draw a line segment connecting them. Because the entropy curve is strictly concave, the interior of the line segment is strictly below the curve. Any point on that segment represents a weighted average of the two entropies for suitable weights. If you unite the two sets into one parent set, the parent set's value p_C is the weighted average of the children's p_c's. Therefore, the point directly above that point on the curve represents the parent's entropy. The information gain is the vertical distance between them. So the information gain is positive unless the two child sets both have exactly the same p_C and lie at the same point on the curve.

On the other hand, for the graph on the right, plotting the % misclassified, if we draw a line segment connecting two points on the curve, the segment might lie entirely on the curve. In that case, uniting the two child sets into one, or splitting the parent set into two, changes neither the total misclassified sample points nor the weighted average of the % misclassified. The bigger problem, though, is that many different splits will get the same weighted average cost; this test doesn't distinguish the quality of different splits well.

(e) **Ensemble Learning - the motivation of averaging.** Ensemble learning is a general technique to combat overfitting, by combining the predictions of many varied models into a single prediction based on their average or majority vote. Consider a set of uncorrelated random variables $\{Y_i\}_{i=1}^n$ with mean μ and variance σ^2. Calculate the expectation and variance of their average. (In the context of ensemble methods, these Y_i are analogous to the prediction made by classifier i.)

Solution: The average of the Y_is has the same expectation as each individual Y_i:

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} Y_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[Y_i] = \frac{1}{n}\cdot n\mu = \mu$$

but reduced variance compared to each of the individual Y_i's:

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \left(\frac{1}{n}\right)^2\sum_{i=1}^{n}\mathrm{Var}(Y_i) = \frac{1}{n^2}\cdot n\sigma^2 = \frac{\sigma^2}{n}$$

(f) **Ensemble Learning – Bagging.** In lecture, we covered bagging (Bootstrap AGGregatING). Bagging is a randomized method for creating many different learners from the same data set.

Given a training set of size n, generate B random subsamples of size n' by sampling with replacement. Some points may be chosen multiple times, while some may not be chosen at all. If $n' = n$, around 63% of the points are chosen, and the remaining 37% are called out-of-bag (OOB) samples.

(a) Why 63%?
Hint: Consider the probability of a point not being chosen as $n \to \infty$

Solution: Each sample has probability $(1 - 1/n)^n$ of not being selected. For large n, $(1 - 1/n)^n \approx 1/e \approx .368$

(b) If we use bagging to train our model, How should we choose the hyperparameter B? Recall, B is the number of subsamples, and typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set.

Solution: An optimal number of subsamples B can be found by using cross-validation. Alternatively, we can observe the OOB error.

(g) In part (a), we see that averaging reduces variance for uncorrelated classifiers. Real world prediction will of course not be completely uncorrelated, but reducing correlation will generally reduce the final variance. Reconsider a set of correlate random variables $\{Z_i\}_{i=1}^n$. Suppose $\forall i \neq j$, $\mathrm{Corr}(Z_i, Z_j) = \rho$. Calculate the variance of their average.

Solution:

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \frac{1}{n^2}\mathrm{Var}\left(\sum_{i=1}^{n} Z_i\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n}\mathrm{Var}(Z_i) + \sum_{i\neq j}\sum \mathrm{Cov}(Z_i, Z_j)\right)$$

$$= \frac{n\sigma^2 + n(n-1)\sigma^2\rho}{n^2} = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

We can see that for large n, the first term dominates, which limits the benefit of averaging.