SLDM II - Homework 4

Matt Isaac

November 19, 2018

1. Convex Losses.

a. Show that the logistic loss is convex.

The logistic loss is $L(y,t) = \log(1+\exp(-yt))$. Recall the fact that a function f is convex if and only if $\nabla^2 f(x)$ is positive semi-definite (PSD) for all $x \in \mathbb{R}^d$. We will proceed by finding $\nabla^2 L(y,t)$ and showing that it is PSD.

$$L(y,t) = \log(1+\exp(-yt))$$

$$\nabla_t L(y,t) = -\frac{y\exp(-yt)}{1+\exp -yt}$$

$$\nabla_t^2 L(y,t) = -\nabla\left(\frac{y\exp(-yt)}{1+\exp -yt}\right)$$

$$\nabla_t^2 L(y,t) = -\left(\frac{(1+\exp(-yt))(y\exp(-yt))(-y) - [(y\exp(-yt))(\exp(-yt))(-y)]}{(1+\exp(-yt))^2}\right)$$

$$\nabla_t^2 L(y,t) = -\left(\frac{-y^2\exp(-yt)) + (-y^2\exp(-yt)\exp(-yt)) + (y^2\exp(-yt)\exp(-yt))}{(1+\exp(-yt))^2}\right)$$

$$\nabla_t^2 L(y,t) = -\left(\frac{-y^2\exp(-yt)}{(1+\exp(-yt))^2}\right)$$

$$\nabla_t^2 L(y,t) = \left(\frac{y^2\exp(-yt)}{(1+\exp(-yt))^2}\right)$$

The numerator of $\nabla_t^2 L(y,t)$ consists of the product of y^2 (≥ 0) and $\exp(-yt)$ (≥ 0). Thus, the numerator will always result in something ≥ 0. Likewise, the denominator, $(1+\exp(-yt))^2$ will always be ≥ 1. Combining these two results indicates that $\nabla_t^2 L(y,t) \geq 0$. This means that $\nabla_t^2 L(y,t)$ is PSD. Thus we have shown that the Logistic loss is convex.

b. Show that if L is a convex loss, then

$$\hat{R}(\mathbf{w},b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \mathbf{w}^T\mathbf{x}_i + b)$$

is a conovex function of θ where

$$\theta = \begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}$$

Since we are treating $\mathbf{x}_i$ and y_i as constant, we can say $L(y_i, \mathbf{w}^T\mathbf{x}_i + b) = L(\theta)$. Since we began with the assumption that L is convex, we know, by definition of convexity that

$$L(t\theta_1 + (1-t)\theta_2) \leq tL(\theta_1) + (1-t)L(\theta_2)$$
$$\implies L(y_i, [t\theta_1 + (1-t)\theta_2]^T\mathbf{x}_i) \leq tL(y_i, \theta_1^T\mathbf{x}_i) + (1-t)L(y_i, \theta_2^T\mathbf{x}_i)$$

So, L can be written as a convex function of θ. Since $\hat{R}(\theta)$ is a sum of convex functions, and since the sum of convex functions is convex, $\hat{R}(\theta)$ is also a convex function.

2. Conceptual questions.

a. Linear classifiers. Discuss the differences between LDA, Logistic Regression, separating hyperplanes, and the Optimal Soft-Margin Hyperplane. In what situations would you use each of the classifiers?

A major difference between LDA and Logistic regression is how the parameters of the decision boundary are estimated. For LDA they are estimated using the mean and variance parameters from an assumed gaussian distribution. For Logistic regression they are obtained from the maximum likelihood estimates. For separating hyperplanes and the optimal soft-margin hyperplane, one key difference is that, if the data is linearly seperable, there is an infinite number of possible separating hyperplanes, but only one possible optimal soft-margin hyperplane. So, if a separating hyperplane is used, cross-validation will be required to make an informed choice on which separating hyperplane will provide the best predictive power.

These classifiers are all possible choices if the decision boundary is linear. However, these different classifiers lend themselves to slightly different situations. LDA is appropriate for use when the data comes from a normal distribtion. Logistic regression is appropriate when this assumption cannot be made. We could use a separating hyperplane when the classes are completely linearly separable. The optimal soft-margin hyperplane would be more appropriate when there is some overlap between classes.

b. Describe how you would apply the SVM to the multiclass case.

There are two popular methods for applying SVM to the multiclass case. I will describe one method, and briefly comment on the second method.
The first method is referred to as "One-Versus-One" (ISL p. 355-356). The algorithm proceeds as follows. For $K > 2$ classes, this method constructs an SVM for each of the $\binom{K}{2}$ pairs of classes. The class of each observation is then predicted by each of the $\binom{K}{2}$ SVMs. The $\binom{K}{2}$ classifiers then essentially "vote" for the final predict class of each observation.

Another possible method is the "One-Versus-All" method (ISL p. 356). This method only fits K SVMs, each one comparing the one class to the rest. This method will be computationally more efficient and feasible, since fitting and tuning SVMs can be quite computationally expensive.

3. Kernels

a. To what feature map Φ does the kernel

$$k(\mathbf{u}, \mathbf{v}) = (\langle u, v \rangle + 1)^3$$

correspond? Assume the inputs have an arbitrary dimension d and the inner product is the dot product.

We begin by expanding things out as follows:

$$\begin{aligned} k(\vec{u}, \vec{v}) &= ((\vec{u}^T \vec{v}) + 1)^3 \\ &= \left(\left(\sum_{i=1}^{d} u^{(i)} v^{(i)} \right) + 1 \right)^3 \end{aligned}$$

For simplicity, let us define $g(\vec{u}, \vec{v}) = \sum_{i=1}^{d} u^{(i)} v^{(i)}$. Then,

$$\begin{aligned} k(\vec{u}, \vec{v}) &= (g(\vec{u}, \vec{v}) + 1)^3 \\ &= (g(\vec{u}, \vec{v}) + 1)(g(\vec{u}, \vec{v}) + 1)(g(\vec{u}, \vec{v}) + 1) \\ &= [(g(\vec{u}, \vec{v}))^2 + 2(g(\vec{u}, \vec{v}) + 1)](g(\vec{u}, \vec{v}) + 1) \\ &= (g(\vec{u}, \vec{v}))^3 + (g(\vec{u}, \vec{v}))^2 + 2(g(\vec{u}, \vec{v}))^2 + 2(g(\vec{u}, \vec{v})) + g(\vec{u}, \vec{v}) + 1 \\ &= (g(\vec{u}, \vec{v}))^3 + 3(g(\vec{u}, \vec{v}))^2 + 3(g(\vec{u}, \vec{v})) + 1 \end{aligned}$$

Substituting $\sum_{i=1}^{d} u^{(i)} v^{(i)}$ back in for $g(\vec{u}, \vec{v})$,

$$k(\vec{u}, \vec{v}) = \sum_{i=1}^{d} {u^{(i)}}^3 {v^{(i)}}^3 + \sum_{i=1}^{d} 3{u^{(i)}}^2 v^{(i)} + \sum_{i=1}^{d} 3u^{(i)} {v^{(i)}}^2 + \sum_{i=1}^{d} {u^{(i)}}^2 {v^{(i)}}^2 + \sum_{i \le j} 2u^{(i)} u^{(j)} v^{(i)} v^{(j)} + \sum_{i=1}^{d} u^{(i)} v^{(i)} + 1$$

Thus, the non-linear feature map $\Phi(u)$ is

$$\Phi(\vec{u}) = \left[(u^{(1)})^3, \ldots, (u^{(d)})^3, \sqrt{3}{u^{(1)}}^2, ..., \sqrt{3}{u^{(d)}}^2, \sqrt{3}u^{(i)}, ..., \sqrt{3}u^{(d)}, \sqrt{2}u^{(1)}u^{(2)}, ..., \sqrt{2}u^{(d-1)}u^{(d)}, u^{(i)}, ..., u^{(d)}, 1 \right]$$

Finally, $k(\vec{u}, \vec{v}) = \langle \Phi(\vec{u}), \Phi(\vec{v}) \rangle$

b. Let k_1, k_2 be symmetric, positive-definite kernels of $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real valued function, and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with positive coefficients. For each of the functions k below, state wheter it is necessarily a positive-definite kernel. If you think it is, prove it. If you think it is not, give a counterexample.

(Complete parts i - vi)

i. $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z}) + k_2(\vec{x}, \vec{z})$

Let the kernel matrix K_i be defined as

$$K_i = \begin{bmatrix} k_i(u_1, u_1) & \cdots & k_i(u_1, u_n) \\ \vdots & \ddots & \vdots \\ k_i(u_n, u_1) & \cdots & k_i(u_n, u_n) \end{bmatrix}$$

for $i \in \{1, 2\}$.
As was given in the problem statement, k_1 and k_2 are positive definite (PD) kernels. Thus, by definition, the kernel matrices K_1 and K_2 are positive semi-definite matrices. So, by definition of positive semi-definite matrices, for $\vec{v} \in \mathbb{R}^d$,

$$\vec{v}^T K_1 \vec{v} \ge 0$$
and
$$\vec{v}^T K_2 \vec{v} \ge 0.$$

We also know, by matrix properties, that

$$\vec{v}^T K_1 \vec{v} + \vec{v}^T K_2 \vec{v} = \vec{v}^T [K_1 + K_2] \vec{v}.$$

Since both $\vec{v}^T K_1 \vec{v}$ and $\vec{v}^T K_2 \vec{v}$ are ≥ 0,

$$\begin{aligned} &\vec{v}^T K_1 \vec{v} + \vec{v}^T K_2 \vec{v} \ge 0 \\ \implies &\vec{v}^T [K_1 + K_2] \vec{v} \ge 0 \end{aligned}$$

This means that $[K_1 + K_2]$ is positive semi-definite. Since $[K_1 + K_2]$ is the kernel matrix of $k(\vec{x}, \vec{z})$, $k(\vec{x}, \vec{z})$ is a positive definite kernel.

ii. $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z}) - k_2(\vec{x}, \vec{z})$
Let K_1 and K_2 be defined as in part (i). Also, let $K_2 = 2K_1$. Since K_1 and K_2 are both positive semi-definite matrices, we know that

$$\vec{v}^T K_1 \vec{v} \geq 0$$
and
$$\vec{v}^T K_2 \vec{v} = \vec{v}^T 2K_1 \vec{v} \geq 0.$$

We can also see that

$$\vec{v}^T K_1 \vec{v} - \vec{v}^T K_2 \vec{v} = \vec{v}^T [K_1 - K_2] \vec{v}$$
$$\implies \vec{v}^T K_1 \vec{v} - \vec{v}^T 2K_1 \vec{v} = \vec{v}^T [K_1 - K_2] \vec{v}.$$

Since

$$\vec{v}^T 2K_1 \vec{v} = 2[\vec{v}^T K_1 \vec{v}]$$
$$\implies 2[\vec{v}^T K_1 \vec{v}] \geq \vec{v}^T K_1 \vec{v} \geq 0,$$
$$\vec{v}^T K_1 \vec{v} - \vec{v}^T 2K_1 \vec{v} \leq 0$$

Recalling that

$$\vec{v}^T K_1 \vec{v} - \vec{v}^T 2K_1 \vec{v} = \vec{v}^T [K_1 - K_2] \vec{v},$$

it is clear that

$$\vec{v}^T [K_1 - K_2] \vec{v} \leq 0$$

Thus, we have shown (by way of counterexample) that $[K_1 - K_2]$ is not necessarily positive semi-definite. It follows that $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z}) - k_2(\vec{x}, \vec{z})$ is not necessarily a positive definite kernel.

iii. $k(\vec{x}, \vec{z}) = ak_1(\vec{x}, \vec{z})$
Let K_1 (the kernel matrix) be defined as in parts (i) and (ii). As before, K_1 is a positive semi-definite matrix due to the condition that $k(\vec{x}, \vec{z})$ is a positive definite kernel. Because K_1 is positive semi-definite, we know that

$$\vec{v}^T K_1 \vec{v} \geq 0$$

Since it is given that a is a constant where $a \geq 0$, it follows that

$$a(\vec{v}^T K_1 \vec{v}) \geq 0$$
$$\vec{v}^T [aK_1] \vec{v} \geq 0$$

Thus, aK_1 is a positive semi-definite matrix, which implies that $k(\vec{x}, \vec{z}) = ak_1(\vec{x}, \vec{z})$ is necessarily a positive definite kernel.

iv. $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z})$ Since k_1 and k_2 are positive definite kernels, they are also inner prodect kernels (by definition). Thus, k_i can be written as (for $i \in \{1, 2\}$):

$$k_i(\vec{x}, \vec{z}) = \phi^{(i)} \vec{x} \phi^{(i)} \vec{z} = \sum_j \phi_j^{(i)} x \phi_j^{(i)} z$$

Then,

$$k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z}) = \left(\sum_{j=1}^{d} \phi_j^{(1)} x \phi_j^{(1)} z \right) \left(\sum_{i=1}^{d} \phi_i^{(2)} x \phi_i^{(2)} z \right)$$
$$\implies k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z}) = \sum_{j=1}^{d} \sum_{i=1}^{d} \phi_j^{(1)} x \phi_j^{(1)} z \phi_i^{(2)} x \phi_i^{(2)} z$$
$$\implies k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z}) = \sum_{j=1}^{d} \sum_{i=1}^{d} (\phi_j^{(1)} x \phi_j^{(1)} z)(\phi_i^{(2)} x \phi_i^{(2)} z)$$
$$\implies k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z}) = \sum_{j=1}^{d} \sum_{i=1}^{d} \phi_j^{(1)} \phi_i^{(2)} x^2 \phi_j^{(1)} \phi_i^{(2)} z^2$$
$$\implies k_1(\vec{x}, \vec{z}) k_2(\vec{x}, \vec{z}) = \psi x^2 \psi z^2$$

where $\psi = \phi^{(1)}\phi^{(2)}$.

Thus, $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z})k_2(\vec{x}, \vec{z})$ can be written as an inner product kernel. From the properties of inner product kernels, we conclude that $k(\vec{x}, \vec{z}) = k_1(\vec{x}, \vec{z})k_2(\vec{x}, \vec{z})$ is necessarily a positive definite kernel.

v. $k(\vec{x}, \vec{z}) = f(\vec{x})f(\vec{z})$
Recall that, if $k(\vec{x}, \vec{z})$ is an inner product kernel, then there exists a non-linear feature space such that $k(\vec{x}, \vec{z}) = \psi(x)^T\psi(z)$. In this case, we notice that there are no restrictions on $\psi(\cdot)$. So if we let $\psi(\cdot) = f(\cdot)$, then we can write

$$k(\vec{x}, \vec{z}) = \psi(\vec{x})^T\psi(\vec{z}).$$

Thus, we have shown that $k(\vec{x}, \vec{z}) = f(\vec{x})f(\vec{z})$ is an inner product kernel. Recalling that all inner product kernels are positive definite kernels, we conclude that $k(\vec{x}, \vec{z}) = f(\vec{x})f(\vec{z})$ is necessarily a positive definite kernel.

vi. $k(\vec{x}, \vec{z}) = p(k_1(\vec{x}, \vec{z}))$
Since the polynomial $p(\cdot)$ is a polynomial with positive coefficients, we can write a polynimal of degree l as $\sum_{i=0}^{l} a_i(k_1(\vec{x}, \vec{z}))^l$. As shown in part (iv) of this problem, the individual terms of the polynomial without the coefficients $((k_1(\vec{x}, \vec{z}))^0 , \ldots, (k_1(\vec{x}, \vec{z}))^l)$ are positive definite kernels, since they are simply products of $k(\vec{x}, \vec{z})$ which is a positive definite kernel. Then, considering the terms with the coefficients $(a_0(k_1(\vec{x}, \vec{z}))^0 , \ldots, a_l(k_1(\vec{x}, \vec{z}))^l)$ are also positive definite kernels (as shown in part (iii)). Lastly, recalling what was shown in part (i), the sum of all these positive definite kernels results in a positive definite kernel. Thus, the kernel $k(\vec{x}, \vec{z}) = p(k_1(\vec{x}, \vec{z}))$ is necessarily a positive definite kernel.

4. Alternative OSM hyperplane.

An alternative way to extend the max-margin hyperplane to nonseperable data is to solve the following quadratic program:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i^2$$
$$s.t. y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \forall i$$
$$\xi_i \geq 0, \forall i$$

a. Which loss is associated with the above quadratic program? In other words, show that learning a hyperplane by the above optimiation problem is equivalent to ERM with a certain loss.

First, we can scale the problem without changin the solution by introducing a constant. Let $\lambda = \frac{1}{C}$. Our optimization program then becomes the following (with the same constraints as before).

$$\min_{\mathbf{w},b,\xi} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n}\xi_i^2$$

Next, we can reduce the constraints by getting the first constraint in terms of ξ_i.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$
$$\implies \xi_i \geq 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

And then, combining our two constraints:

$$\xi_i \geq \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

Since the right side will always be ≥ 0, the following also holds:

$$\xi_i^2 \geq \left[\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))\right]^2$$

The solution to the optimization program clearly must also satisfy

$$\xi_i^2 = \left[\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))\right]^2.$$

Thus, the optimal solution $(\mathbf{w}^*, b^*)$ will also solve

$$\min_{\mathbf{w},b,\xi} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n}\left[\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))\right]^2.$$

Which is equivalent to regularized ERM with squared hinge loss.

b. Argue that the second set of constraints can be dropped without changing the solution.
In looking at the squared combined constraints from the original optimization problem,

$$\xi_i^2 \geq \left[\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))\right]^2,$$

it is easy to see that since $(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))^2$ will always be greater than or equal to 0, which means that ξ_i^2 will always be greater than 0. Thus, the constraint of $\xi_i \geq 0$ can be dropped, resulting in the following constraint:

$$\xi_i^2 \geq \left[1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right]^2.$$

c. Identify an advantage and a disadvantage of this loss compared to the hinge loss.
The squared hinge loss will be more suceptible to misclassified points. The more "wrong" a classification is, the more the decision boundary will be penalized (i.e. the more it will shift) as compared with the usual hinge loss. This could be an advantage or a disadvantage compared to the usual hinge loss, depending on the context and goal of the classification. If the classification needs to strongly avoid misclassification, using the squared hinge loss could have an advantage over the regular hinge loss. On the other hand, if we don't need to overly penalize misclassifications, this high sensitivity to misclassification could be a disadvantage.

5. Subgradient methods for the optimal soft margin hyperplane.

In this problem you will implement the subgradient and stochastic subgradient methods for minimizing the convex but nondifferentiable function

$$J(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \mathbf{w}^T\mathbf{x}_i + b) + \frac{\lambda}{2}||\mathbf{w}||^2$$

a. Determine $J_i(\mathbf{w}, b)$ such that

$$J(\mathbf{w}, b) = \sum_{i=1}^{n} J_i(\mathbf{w}, b)$$

As was given in the problem statement, $L(y_i, \mathbf{w}^T\mathbf{x}_i + b) = max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)$. Thus,

$$J(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \frac{\lambda}{2}||\mathbf{w}||^2$$

Then pulling the multiplied constant $\frac{1}{n}$ into the summation, as well as the added constant $\frac{\lambda}{2}||\mathbf{w}||^2$, we get that

$$J(\mathbf{w}, b) = \sum_{i=1}^{n}\left[\frac{1}{n}max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \frac{\lambda}{2n}||\mathbf{w}||^2\right]$$

Then, it is clear that if $J(\mathbf{w}, b) = \sum_{i=1}^{n} J_i(\mathbf{w}, b)$ that

$$J_i = \frac{1}{n}max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \frac{\lambda}{2n}||\mathbf{w}||^2$$

Next, we need to determine a subgradient $\mathbf{u}_i$ of each J_i.

We will consider three cases.

Case 1: $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$.

$$J_i = \frac{1}{n}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) + \frac{\lambda}{2n}||\mathbf{w}||^2$$

$$\nabla J_i = \frac{1}{n}(-y_i \begin{bmatrix} 1 & \mathbf{x}_i \end{bmatrix}^T) + \frac{\lambda}{n} \begin{bmatrix} 0 & \mathbf{w} \end{bmatrix}^T$$

Case 2: $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$.

$$J_i = \frac{1}{n}(0) + \frac{\lambda}{2n}||\mathbf{w}||^2$$

$$J_i = \frac{\lambda}{2n}||\mathbf{w}||^2$$

$$\nabla J_i = \frac{1}{n}(0) + \frac{\lambda}{n} \begin{bmatrix} 0 & \mathbf{w} \end{bmatrix}^T$$

$$\nabla J_i = \frac{\lambda}{n} \begin{bmatrix} 0 & \mathbf{w} \end{bmatrix}^T$$

Case 3: $1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) = 0$.

$$J_i = \frac{1}{n}(max(0, 0)) + \frac{\lambda}{2n}||\mathbf{w}||^2$$

In which case the gradient doesn't exist. For this case we will to pick a subgradient. Each of the gradients in case 1 and case 2 are subgradients for case 3.

So, in summary,

$$\nabla J_i = \begin{cases} \frac{1}{n}(-y_i \begin{bmatrix} 1 & \mathbf{x}_i \end{bmatrix}^T) + \frac{\lambda}{n} \begin{bmatrix} 0 & \mathbf{w} \end{bmatrix}^T & 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 0 \\ \\ \frac{\lambda}{n} \begin{bmatrix} 0 & \mathbf{w} \end{bmatrix}^T & 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0 \end{cases}$$

b. Implement the subgradient method for minimizing J and apply it to the nuclear data. Submit two figures: One showing the data and the learned line, the other showing J as a function of iteration number. Also report the estimated hyperplane parameters and the minimum achieved value of the objective function.

Figure 1: Data and learned line

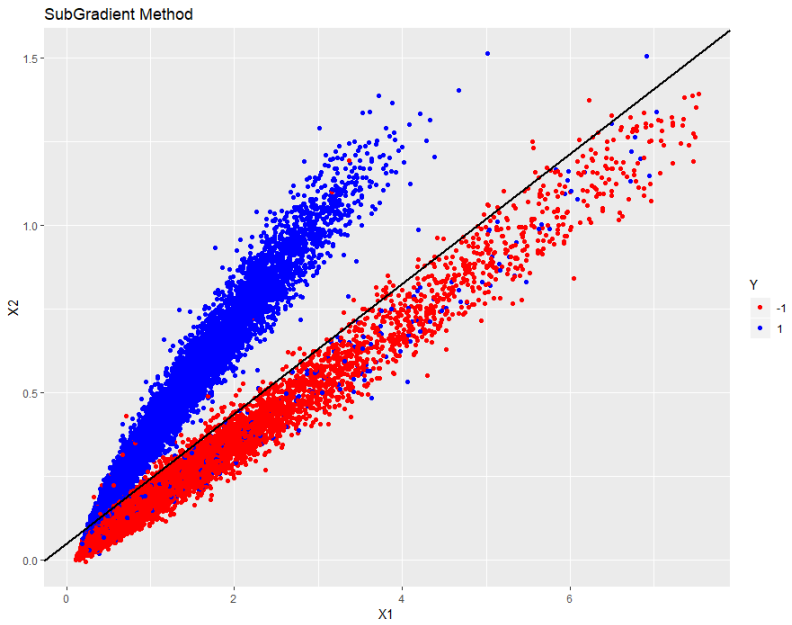
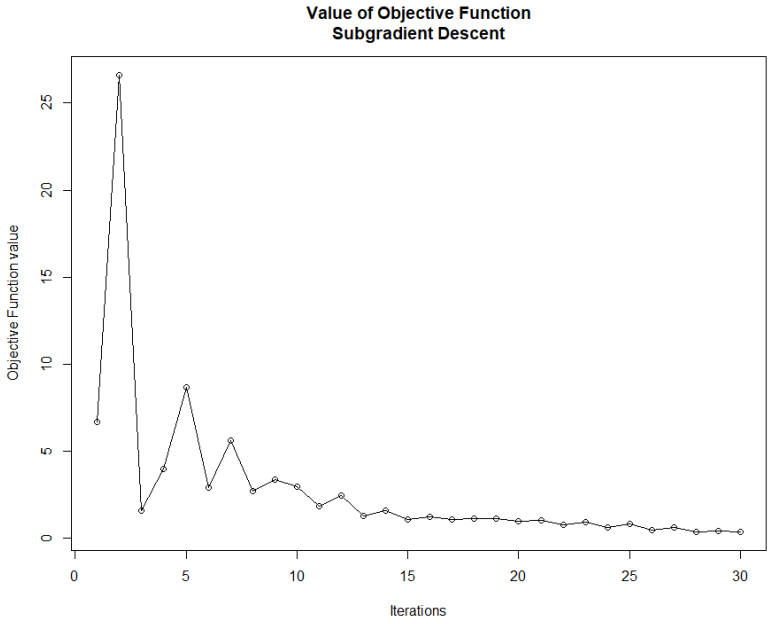


Figure 2: J by iteration number



Estimated hyperplane parameters: $b = -0.942$, $w_1 = -3.681$, $w_2 = 18.972$

Minimum achieved value of Objective Function: 0.3714

c. Now implement the stochastic subgradient method with a mnibatch of size $m = 1$. Be sure to cycle through all the data points before starting a new loop through the data. Report the same items as for part (b).

Figure 3: Data and learned line

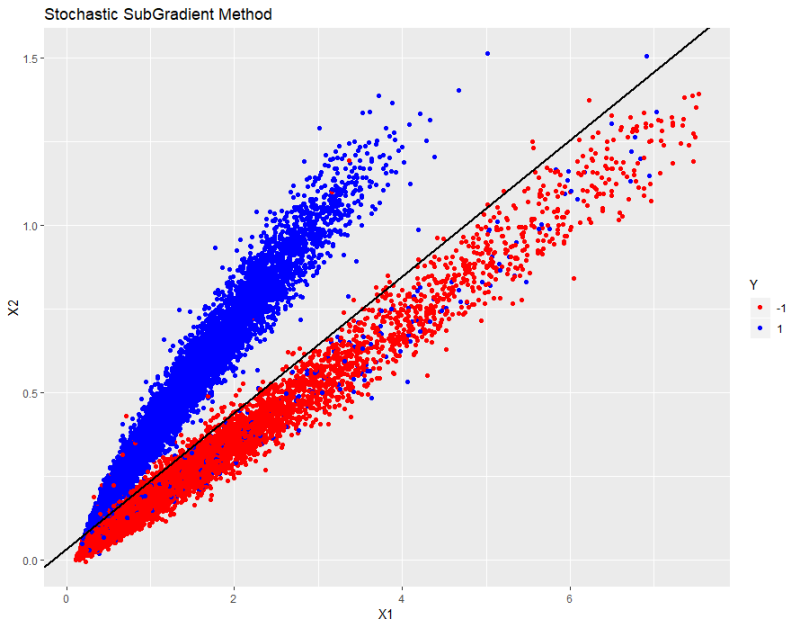
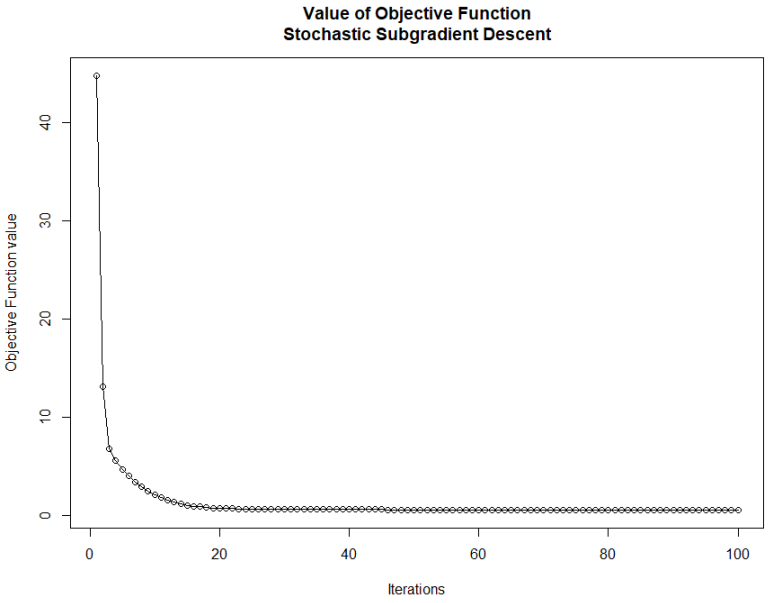


Figure 4: J by iteration number



Estimated hyperplane parameters: $b = -1.255$, $w_1 = -5.629$, $w_2 = 27.677$

Minimum achieved value of Objective Function: 0.561

d. Comment on the eperical rate of convergence of the stochastic subgradient method relative to the subgradient method.

From my observations, it seems like the stochastic subgradient method took more iterations to converge than the regular subgradient method, although the iterations seemed to go faster.

e. Submit your code:

```
# Exercise 5 (b)
```

```
library(R.matlab)
library(dplyr)
library(geometry)
library(ggplot2)

################################################################################
#### Function Definitions ######################################################
################################################################################

subGradInit <- function(maxitr, predictors, response, lambda){
  # Kicks off subGrad() recursive function
  # Args:
  #   maxitr: maximum iterations for subgradient method
  #   predictors: dataframe of predictor variables
  #   response: dataframe (tibble?) of response values
  #   lambda: regularization paramter
  #
  # Returns:
  #   The optimized value of theta
  #

  b = 1 # initial guess of b
  ws = rep(0, ncol(predictors) - 1) # initial guess of w's
  inittheta <- c(b, ws)

  # initialize history data frame
  history <- data.frame(numItr = double(0), objFun = double(0),
                        w1 = double(0), w2 = double(0), b = double(0))

  # start subGrad() recursive function
  theta <- subGrad(theta = inittheta, nitr = 1, maxitr = maxitr,
                   xs = predictors, y = response,
                   lambda = lambda, history = history)
  return(theta)
}

subGrad <- function(theta, nitr, maxitr, xs, y, lambda, history){
  # Recursive function that implements the sub-gradient method
  # Args:
  #   theta: vector of w's and b's
  #   nitr: current iteration number
  #   maxitr: maximum number of iterations for subgradient method
  #   xs: dataframe of predictor variables
  #   y: dataframe (tibble?) of response
  #   lambda:
  #   history: a dataframe used to store parameter estimates and objective function
  #            values.
  #
  # Returns:
  #   The optimized value of theta, or calls itself again.

  alpha <- 100/nitr # step size
  n <- nrow(preds)
```

```
  if(nitr > maxitr){
    return(list("theta" = theta, "history" = history))
  } else {
    print(paste("Iteration Number: ", nitr))

    u <- double(length(theta))
    for(i in 1:nrow(xs)){
      # calculate gradient
      grad <- calcGrad(b = theta[1], w = theta[2:length(theta)],
                       yi = as.numeric(y$Y[i]), xi = as.numeric(xs[i,]),
                       n = n, lambda = lambda)

      # add to u
      u <- u + grad
    }

    theta.1 <- theta - (alpha * u) # take step in negative direction of gradient

    # calculate value of objective function
    of <- objFunc(preds = xs, resp = y, theta = theta.1, lambda = lambda)

    # new row to be added to history dataframe
    hnew <- data.frame(numItr = nitr, objFun = of, b = theta.1[1],
                       w1 = theta.1[2], w2 = theta.1[3])

    history <- rbind(history, hnew)
    nitr <- nitr + 1 # increment iteration number

    return(subGrad(theta = theta.1, nitr = nitr, maxitr = maxitr,
                   xs = xs, y = y, lambda = lambda, history = history))
  }
}

stochSubGradInit <- function(maxitr, predictors, response, lambda){
  # Kicks off stochSubGrad() recursive function
  # Args:
  #   maxitr: maximum iterations for subgradient method
  #   predictors: dataframe of predictor variables
  #   response: dataframe (tibble?) of response values
  #   lambda: regularization parameter
  #
  # Returns:
  #   The optimized value of theta
  #

  b = 1 # initial guess of b
  ws = rep(0, ncol(predictors) - 1) # initial guess of w's
  inittheta <- c(b, ws)

  # initialize history data frame
  history <- data.frame(numItr = double(0), objFun = double(0),
                        w1 = double(0), w2 = double(0), b = double(0))
```

```
  # begin recursion of stochSubGrad()
  theta <- stochSubGrad(theta = inittheta, nitr = 1, maxitr = maxitr,
                        xs = predictors, y = response,
                        lambda = lambda, history = history)
  return(theta)
}

stochSubGrad <- function(theta, nitr, maxitr, xs, y, lambda, history){
  # Recursive function that implements the sub-gradient method
  # Args:
  #   theta: vector of w's and b's
  #   nitr: current iteration number
  #   maxitr: maximum number of iterations for subgradient method
  #   xs: dataframe of predictor variables
  #   y: dataframe (tibble?) of response
  #   lambda: regularization parameter
  #   history: dataframe recording parameter values and objective function values
  #
  # Returns:
  #   The optimized value of theta, or calls itself again.

  alpha <- 100/nitr # step size
  n <- nrow(preds)
  m <- 1 # minibatch size

  if(nitr > maxitr){
    return(list("theta" = theta, "history" = history))
  } else {
    print(paste("Iteration Number: ", nitr))

    index <- 1:n
    rand.index <- sample(index, n) # randomize indices for easy "random sampling"

    for(i in rand.index){
        grad <- calcGrad(b = theta[1], w = theta[2:length(theta)],
                         yi = as.numeric(y$Y[i]), xi = as.numeric(xs[i,]),
                         n = 1, lambda = lambda)
        theta <- theta - (alpha * grad) # update theta each time

    }

    theta.1 <- theta

    # calculate value of objective function
    of <- objFunc(preds = xs, resp = y, theta = theta.1, lambda = lambda)

    # new row to be added to history data frame
    hnew <- data.frame(numItr = nitr, objFun = of, b = theta.1[1],
                       w1 = theta.1[2], w2 = theta.1[3])

    history <- rbind(history, hnew)
    nitr <- nitr + 1
    return(subGrad(theta = theta.1, nitr = nitr, maxitr = maxitr, xs = xs, y = y, lambda = lambda, hist
```

```
  }
}

calcGrad <- function(w, b, yi, xi, n, lambda){
  # Calculates gradient
  # Args:
  #   w: vector of w's
  #   b: intercept
  #   yi: response
  #   xi: vector of predictor variables
  #   n: number of observations
  #   lambda: regularization parameter

  wvec <- c(0, w)

  term <- 1 - (yi * (dot(wvec, xi) + b))
  if(term >= 0){
    gradJi <- (1/n)*(-yi*xi) + (lambda/n)*wvec
  } else {
    gradJi <- (lambda/n)*wvec
  }
  sg <- (gradJi)
  return(sg)
}

objFunc <- function(preds, resp, theta, lambda){
  # Calculates value of the objective function
  # Args:
  #   preds: vector of predictor variable (x_i)
  #   resp: response value (y_i)
  #   theta: vector of b and w's
  #   lambda: regularization parameter
  #
  n <- nrow(preds)
  b <- theta[1]
  w <- theta[2:length(theta)]
  wvec <- c(0, w)

  summation <- 0
  for(i in 1:n){
    yi = as.numeric(resp$Y[i])
    xi = as.numeric(preds[i,])
    m1 <- 0
    m2 <- 1 - yi*(dot(wvec, xi) + b)
    term <- max(m1, m2)
    summation <- summation + term
  }
  objf <- 1/n * (summation) + (lambda/2)*dot(w,w)
  return(objf)
}

plotObjFun <- function(history, title){
  # Creates plot of objective function by iterations
```

```
  # Args:
  #   history: history dataframe as returned by stochSubGrad() or subGrad()
  #   title: string to be title of plot
  #
  plot(history$numItr, history$objFun, xlab = "Iterations",
       ylab = "Objective Function value", type = 'o',
       main = title)
}

################################################################################
### Read in and format data ####################################################
################################################################################

df <- R.matlab::readMat("nuclear.mat") %>% lapply(t) %>% lapply(as_tibble)
colnames(df[[1]]) <- sprintf("X_%s",seq(1:ncol(df[[1]])))
colnames(df[[2]]) <- c("Y")
df <- bind_cols(df) %>% select(Y, everything())

# df <- slice(df, 1:300) # uncomment this line for faster debugging
preds <- select(df, X_1, X_2)
X0 <- rep(1, nrow(df))
preds <- cbind(X0, preds)
resp <- select(df, Y)
resp$Y <- as.numeric(resp$Y)


################################################################################
### Function Calls #############################################################
################################################################################


### Sub Gradient Method ########################################################

# [s]ub [g]radient [m]ethod [res]ults
sgm_res <- subGradInit(maxitr = 30, predictors = preds, response = resp, lambda = 0.001)

plotObjFun(sgm_res$history, title = "Value of Objective Function\nSubgradient Descent")

b <- sgm_res$theta[1]
w1 <- sgm_res$theta[2]
w2 <- sgm_res$theta[3]

slope <- w1/-w2
intercept <- b/-w2

df$Y <- as.factor(df$Y) # needed for correct plotting

ggplot(data = df, aes(x = X_1, y = X_2, color = Y)) +
  geom_point() +
  scale_color_manual(values=c("-1" = "red", "1" = "blue")) +
  geom_abline(slope = slope, intercept = intercept, lwd = 1) +
  xlab("X1") +
  ylab("X2") +
```

```
  ggtitle("SubGradient Method")


### Stochastic Sub Gradient Method ##############################################

# [st]ochastic [sub [g]radient [m]ethod [res]ults
stsgm_res <- stochSubGradInit(maxitr = 100, predictors = preds,
                          response = resp, lambda = 0.001)

plotObjFun(stsgm_res$history, title = "Value of Objective Function\nStochastic Subgradient Descent")

b <- stsgm_res$theta[1]
w1 <- stsgm_res$theta[2]
w2 <- stsgm_res$theta[3]

slope <- w1/-w2
intercept <- b/-w2

plot(preds$X_1, preds$X_2)
abline(a = intercept, b = slope)

df$Y <- as.factor(df$Y)

ggplot(data = df, aes(x = X_1, y = X_2, color = Y)) +
  geom_point() +
  scale_color_manual(values=c("-1" = "red", "1" = "blue")) +
  geom_abline(slope = slope, intercept = intercept, lwd = 1) +
  xlab("X1") +
  ylab("X2") +
  ggtitle("Stochastic SubGradient Method")
```

6. Classification

a. Download a dataset for classification from the UCI ML Repository.

I downloaded the Parkinsons data set (https://archive.ics.uci.edu/ml/datasets/parkinsons). This data set has 22 features and 2 classes.

b. Apply logistic regression to this dataset. Calculate the test error based on k-fold cross-validation and report the training and test error.

Cross-validated training error: 0.211
Test-error: 0.154

c. Apply the SVM to this dataset using 2 different kernels: linear and Gaussian. Describe the tuning parameters in each case and set these parameters by cross-validation. Report your final test error, training error, and tuning parameters after cross-validation.

Linear Kernel:

For the linear kernel, there is only one tuning parameter: cost. Cost indicates the magnitude of penalty assigned for an observation violating the margin of our decision boundary. A small cost corresponds to a large margin, while a large cost corresponds to a small margin.

Tuning parameter: Cost = 2.0001 Training cross-validated error: 0.166 Test error: 0.103

Gaussian Kernel:

For the gaussian kernal, there are two tuning parameters: Cost, which functions as described in the linear kernel section, and gamma. Gamma is related to the variance of the gaussian distribution, in that $\gamma = \frac{1}{\sigma^2}$.

Tuning parameter: Cost = 2.0001, Gamma = 0.17 Training cross-validated error: 0.0325 Test error: 0.077

d. Which of the three methods you applied worked best? Do your results make sense?

The tuned SVM with tuned gaussian kernel worked the best. This makes sense to me because SVM is known to perform better than logistic regression, so the better results are not a surprise. The fact that the gaussian kernel worked better is not much of a surprise either, since the classes may not have been linearly seperable.

Code for Logistic Regression and SVMs.

```
library(caret)
library(dplyr)

################################################################################
### Logistic Regression ########################################################
################################################################################

park <- read.csv("parkinsons.csv", header = TRUE)
park <- select(park, -name)

traintest <- function(data, ptrain){
  n <- nrow(data)
  ntrain <- round(n * ptrain)
  ntest <- n - ntrain

  trainInd <- sample(1:n, size = ntrain)

  train <- slice(data, trainInd)
  test <- slice(data, -trainInd)

  return(list("train" = train, "test" = test))
}

park$status <- as.factor(park$status)
out <- traintest(park, ptrain = 0.8)
ptrain <- out$train
ptest <- out$test

# fit logistic regression
logreg <- train(status ~ ., data = ptrain, method = 'glm', family = binomial, trControl = trainControl(

# cross validated accuracy
logreg$results$Accuracy

pred <- predict(logreg, ptest)
iscorrect <- (ptest$status == pred)

sum(iscorrect)/nrow(ptest)

################################################################################
### SVM ########################################################################
################################################################################
```

```
library(e1071)

## Linear Kernel

svm.default <- svm(status~., data = ptrain, kernel = "linear")

preds <- predict(svm.default, ptest)
iscorrect <- (ptest$status == preds)
sum(iscorrect)/nrow(ptest)
notcorrect <- (ptest$status != preds)
sum(notcorrect)/nrow(ptest)

svm.tune <- tune.svm(status~. , data = ptrain, kernel = "linear", cost = 10^-4:4)
svm.tune

svm.mod <- svm(status~., data = ptrain, kernel = "linear", cost = 2.0001)

preds <- predict(svm.mod, ptest)
iscorrect <- (ptest$status == preds)
sum(iscorrect)/nrow(ptest)
notcorrect <- (ptest$status != preds)
sum(notcorrect)/nrow(ptest)


## Gaussian Kernel

svm.default.gaus <- svm(status~., data = ptrain)
preds <- predict(svm.default.gaus, ptest)
iscorrect <- (ptest$status == preds)
sum(iscorrect)/nrow(ptest)
notcorrect <- (ptest$status != preds)
sum(notcorrect)/nrow(ptest)

svm.tune.gaus <- tune.svm(status~. , data = ptrain, cost = 10^-4:4, gamma = seq(0.01, 1.0, 0.02))

svm.gaus <- svm(status~., data = ptrain, cost = 2.0001, gamma = 0.17)
preds <- predict(svm.gaus, ptest)
iscorrect <- (ptest$status == preds)
sum(iscorrect)/nrow(ptest)
notcorrect <- (ptest$status != preds)
sum(notcorrect)/nrow(ptest)
```

7. Type up homework solutions.

Check.

8. How long did this homework assignment take you?

About 33 hours.