SLDM II - Homework 5

Matt Isaac

November 30, 2018

1. Variance of correlated random variables.

Suppose we have random variables $X_1, ..., X_B$ that are identically distributed but not independent. Assume that for each i, $\mathbb{V}[X_i] = \sigma^2$ where $\mathbb{V}[X_i]$ denotes the variance of X_i. Furthermore, assume that each pair of random variables is positively correlated with correlation coefficient ρ. I.e., $corr(X_i, X_j) = \rho > 0$ for each $i \neq j$.

a. Prove that the variance of the sample mean is $\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. This provides motivation for selecting random features in the random forest algorithm.

Beginning with the given hints,

$$\begin{aligned}
\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] &= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} Cov(X_i, X_j) \\
&= \frac{1}{B^2}\left[Cov(X_1, X_1) + \cdots + Cov(X_B, X_B) + \sum_{i=1, i\neq j}^{B}\sum_{j=1}^{B} Cov(X_i, X_j)\right] \\
&= \frac{1}{B^2}\left[\mathbb{V}[X_1] + \cdots + \mathbb{V}[X_B] + \sum_{i=1, i\neq j}^{B}\sum_{j=1}^{B} Cov(X_i, X_j)\right]
\end{aligned}$$

The definition of the correlation coefficient $\rho := \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$ can be rearranged so that $Cov(X, Y) = \rho\sqrt{Var(X)Var(Y)}$. Plugging this into the equation,

$$\begin{aligned}
\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] &= \frac{1}{B^2}\left[\mathbb{V}[X_1] + \cdots + \mathbb{V}[X_B] + \sum_{i=1, i\neq j}^{B}\sum_{j=1}^{B} \rho\sqrt{\mathbb{V}[X_i]\mathbb{V}[X_j]}\right] \\
&= \frac{1}{B^2}\left[B\sigma^2\right] + \frac{1}{B^2}\left[\sum_{i=1, i\neq j}^{B}\sum_{j=1}^{B} \rho\sqrt{\mathbb{V}[X_i]\mathbb{V}[X_j]}\right] \\
&= \frac{1}{B^2}\left[B\sigma^2\right] + \frac{1}{B^2}\left[\sum_{i=1, i\neq j}^{B}\sum_{j=1}^{B} \rho\sigma^2\right]
\end{aligned}$$

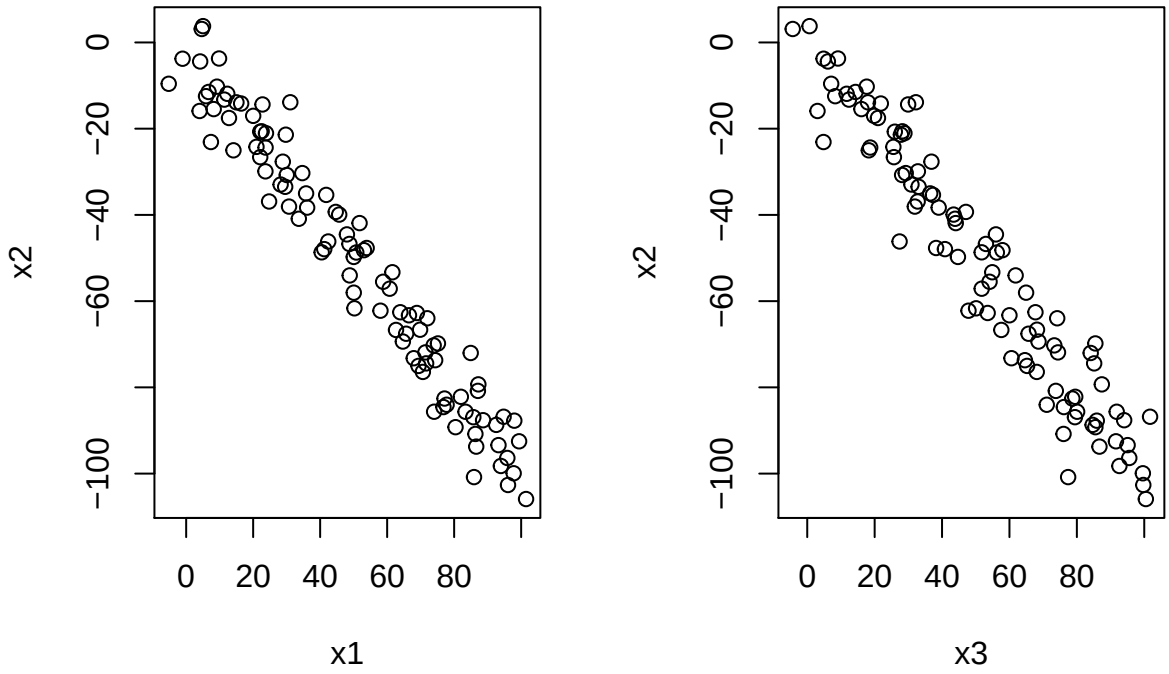
Since the double sum is limited to cases where $i \neq j$, there will be $B^2 - B$ terms in the double sum. So, we have

$$\begin{aligned}
\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] &= \frac{1}{B^2}\left[B\sigma^2\right] + \frac{1}{B^2}(B^2 - B)\left[\rho\sigma^2\right] \\
&= \frac{\sigma^2}{B} + \frac{1}{B^2}(B^2\rho\sigma^2 - B\rho\sigma^2) \\
&= \frac{\sigma^2}{B} + \rho\sigma^2 - \frac{\rho\sigma^2}{B} \\
&= \rho\sigma^2 + \frac{\sigma^2 - \rho\sigma^2}{B} \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2
\end{aligned}$$

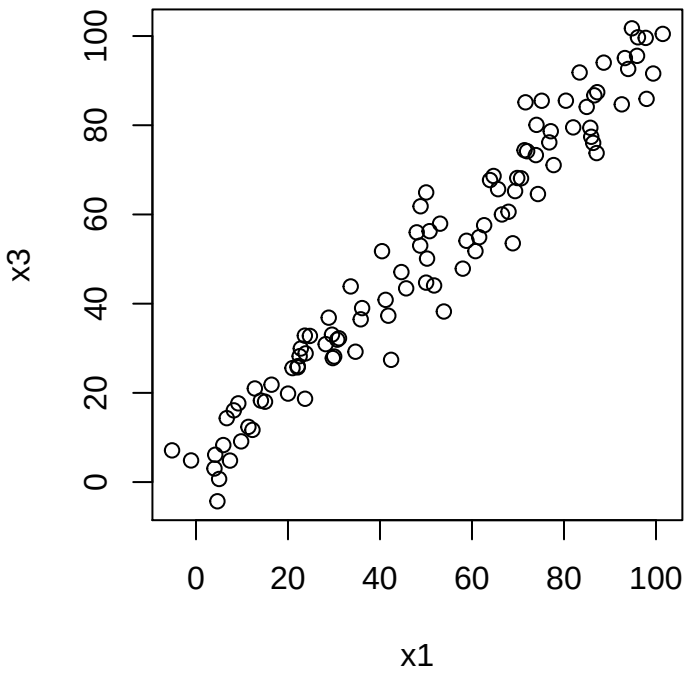
b. Explain why it would not make sense (i.e. it isn't possible) for ρ to be negataive when $B \geq 3$.

Begin by assuming that $Corr(X_i, X_j) = \rho < 0$ for all $i \neq j$. Let us begin with the simplest case - that is, let $B = 3$. In other words, we have three variables, X_1, X_2, X_3, all of which are negatively correlated with each other, with correlation coefficient $\rho < 0$.

So lets think through the specifics a little more. Below are plots for X_1 by X_2 and X_2 by X_3.



As X_1 increases, X_2 decreases. Likewise, as X_3 increases, $X_2 decreases$. That means that X_1 and X_3 are exhibiting the same behavior as X_2 decreases; that is, X_1 and X_3 both increase as X_2 decreases. So, X_1 and X_3 are positively correlated (see plot below).



2. Support Vector Regression

Support vector regression (SVR) is a method for regression analogous to the support vector classifier. Let $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}, i = 1, ..., n$ be training data for a regression problem. In the case of linear regression, SVR solves

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

$$\begin{aligned} s.t. \quad y_i - \mathbf{w}^T\mathbf{x}_i - b &\leq \epsilon + \xi_i^+ \quad \forall i \\ \mathbf{w}^T\mathbf{x}_i + b - y_i &\leq \epsilon + \xi_i^+ \quad \forall i \\ \xi_i^+ &\geq 0 \quad \forall i \\ \xi_i^- &\geq 0 \quad \forall i \end{aligned}$$

where $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}, \xi^+ = (\xi_1^+, ..., \xi_n^+)^T, \xi^- = (\xi_1^-, ..., \xi_n^-)^T$

a. show that for an appropriate choice of λ, SVR solves

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

where $\ell_\epsilon(y, t) = \max(0, |y - t| - \epsilon)$.

Essentially, we will need to show that the solution of

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

(subject to the given constraints) is also a solution to

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

For conciseness, let

$$* = \min_{\mathbf{w},b,\xi^+,\xi^-} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-)$$

First, choose $\lambda = \frac{1}{C}$. Then, we have

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

Letting $t_i = \mathbf{w}^T\mathbf{x}_i + b$, $\lambda*$ has the following constraints:
(1) $y_i - t_i \leq \epsilon + \xi_i^+ \forall i$
(2) $t_i - y_i \leq \epsilon + \xi_i^- \forall i$
(3) $\xi_i^+ \geq 0$
(4) $\xi_i^- \geq 0$

We then have 3 cases to consider:

- Case 1: $\mathbf{y_i - t_i > 0}$
- Case 2: $\mathbf{t_i - y_i > 0}$
- Case 3: $\mathbf{t_i - y_i = 0}$

For Case 1, if $\xi^+ \geq 0$, then constraints (1) and (3) are satisfied. Then, if we let $\xi_i^- = 0$ (in order to minimize), constraints (2) and (4) will also be satisfied, since $\epsilon > 0$.

Likewise, for Case 2, if $\xi_i^- \geq 0$, then (2) and (4) are satisfied. If we let $\xi_i^+ = 0$, then, since $\epsilon > 0$, constraints (1) and (3) will also be satisfied.

Lastly, for Case 3, if $\xi_i^+ = \xi_i^- = 0$, then all four constraints will be satisifed.

Combining the three cases, we can see that for Case 1 and Case 2, $\xi_i^+ = 0$ or $\xi_i^- = 0$. For these cases, we end up with $|y_i - t_i| \leq \epsilon + \xi_i^+$ or $|y_i - t_i| \leq \epsilon + \xi_i^-$, which implies that $|y_i - t_i| - \epsilon \leq \xi_i^+$ and $|y_i - t_i| - \epsilon \leq \xi_i^+$. Bringing Case 3 into consideration, if $\xi_i^+ = \xi_i^- = 0$, we can see that

$$(\xi_i^+ + \xi_i^-) = \max(0, |y_i - t_i| - \epsilon).$$

Making our substituion back for t_i, we get

$$(\xi_i^+ + \xi_i^-) = \max(0, |y_i - \mathbf{w}^T\mathbf{x}_i + b| - \epsilon).$$

Thus,

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

b. The optimization problem is convex with affine contraints and therefore strong duality holds. Use the KKT conditions to derive the dual optimization problem in a manner analogous to the support vector classifier (SVC). As in the SVC, you should eliminate the dual variables corresponding to the constraints $\xi_i^+ \geq 0, \xi_i^- \geq 0$.

For reference, the SVR program is as follows:

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

$$
\begin{aligned}
s.t. \quad y_i - \mathbf{w}^T\mathbf{x}_i - b &\leq \epsilon + \xi_i^+ \quad \forall i \\
\mathbf{w}^T\mathbf{x}_i + b - y_i &\leq \epsilon + \xi_i^+ \quad \forall i \\
\xi_i^+ &\geq 0 \quad \forall i \\
\xi_i^- &\geq 0 \quad \forall i
\end{aligned}
$$

We will begin by computing the Lagrangian, $L(\mathbf{w}, b, \xi^+, \xi^-, \alpha^+, \alpha^-, \beta^+, \beta^-)$. In order to do this, we must solve each of the constrains to it is in terms of ≤ 0. We obtain the following:

$$
\begin{aligned}
s.t. y_i - \mathbf{w}^T\mathbf{x}_i - b - \epsilon - \xi_i^+ &\leq 0 \quad \forall i \\
\mathbf{w}^T\mathbf{x}_i + b - y_i - \epsilon - \xi_i^- &\leq 0 \quad \forall i \\
-\xi_i^+ &\leq 0 \quad \forall i \\
-\xi_i^- &\leq 0 \quad \forall i
\end{aligned}
$$

Then, the Lagrangian is

$$
\begin{aligned}
L = &\frac{1}{2}||\mathbf{w}||^2 + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \\
&\sum_{i=1}^{n}\alpha_i^+(y_i - \mathbf{w}^T\mathbf{x}_i - b - \epsilon - \xi_i^+) + \\
&\sum_{i=1}^{n}\alpha_i^-(-y_i + \mathbf{w}^T\mathbf{x}_i + b - \epsilon - \xi_i^+) + \\
&\sum_{i=1}^{n}\beta_i^+(-\xi_i^+) + \sum_{i=1}^{n}\beta^-(-\xi_i^-)
\end{aligned}
$$

Then, in order to further simplify, we must calculate the partial derivatives of L with respect to $\mathbf{w}, b, \xi_i^+$, and ξ_i^-.

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} + \sum_{i=1}^{n} -\alpha_i^+\mathbf{x}_i + \sum_{i=1}^{n}\alpha_i^-\mathbf{x}_i = 0 \\
&= \mathbf{w} + \sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^- - \alpha_i^+) = 0 \\
&\implies \mathbf{w} = \sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial b} &= \sum_{i=1}^{n}\alpha_i^+ + \sum_{i=1}^{n}\alpha_i^- = 0 \\
&\implies \sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) = 0 \\
&\implies \sum_{i=1}^{n}(\alpha_i^+) = \sum_{i=1}^{n}(\alpha_i^-)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial \xi_i^+} &= \frac{C}{n} - \alpha_i^+ - \beta_i^+ = 0 \\
&\implies \frac{C}{n} = \alpha_i^+ + \beta_i^+
\end{aligned}
$$

$$\frac{\partial L}{\partial \xi_i^-} = \frac{C}{n} - \alpha_i^- - \beta_i^- = 0$$
$$\implies \frac{C}{n} = \alpha_i^- + \beta_i^-$$

From here, we will expand L, and then use the identities derived from the partial derivatives above to simplify. First, we will plug in $\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)$ for $\mathbf{w}$.

$$\begin{aligned} L =& \frac{1}{2}\left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T \left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) + \\ & \sum_{i=1}^{n} \alpha_i^+(y_i - \mathbf{w}^T\mathbf{x}_i - b - \epsilon - \xi_i^+) + \\ & \sum_{i=1}^{n} \alpha_i^-(-y_i + \mathbf{w}^T\mathbf{x}_i + b - \epsilon - \xi_i^+) + \\ & \sum_{i=1}^{n} \beta_i^+(-\xi_i^+) + \sum_{i=1}^{n} \beta^-(-\xi_i^-) \end{aligned}$$

$$\begin{aligned} L =& \frac{1}{2}\left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T \left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^+) + \frac{C}{n}\sum_{i=1}^{n}(\xi_i^-) + \\ & \sum_{i=1}^{n} \alpha_i^+(y_i) - \sum_{i=1}^{n} \alpha_i^+(\mathbf{w}^T\mathbf{x}_i) - \sum_{i=1}^{n} \alpha_i^+(b) - \sum_{i=1}^{n} \alpha_i^+(\epsilon) - \sum_{i=1}^{n} \alpha_i^+(\xi_i^+) - \\ & \sum_{i=1}^{n} \alpha_i^-(y_i) + \sum_{i=1}^{n} \alpha_i^-(\mathbf{w}^T\mathbf{x}_i) + \sum_{i=1}^{n} \alpha_i^-(b) - \sum_{i=1}^{n} \alpha_i^-(\epsilon) - \sum_{i=1}^{n} \alpha_i^-(\xi_i^+) + \\ & \sum_{i=1}^{n} \beta_i^+(-\xi_i^+) + \sum_{i=1}^{n} \beta^-(-\xi_i^-) \end{aligned}$$

Recalling that $\frac{C}{n} = \alpha_i^+ + \beta_i^+ = \alpha_i^- + \beta_i^-$

$$\begin{aligned} L =& \frac{1}{2}\left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T \left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \sum_{i=1}^{n}(\alpha_i^+ + \beta_i^+)(\xi_i^+) + \sum_{i=1}^{n}(\alpha_i^- + \beta_i^-)(\xi_i^-) + \cdots + \\ & \sum_{i=1}^{n} \beta_i^+(-\xi_i^+) + \sum_{i=1}^{n} \beta^-(-\xi_i^-) \end{aligned}$$

$$\begin{aligned} L =& \frac{1}{2}\left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T \left[\sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \sum_{i=1}^{n}(\alpha_i^+\xi_i^+) + \sum_{i=1}^{n}(\beta_i^+\xi_i^+) + \sum_{i=1}^{n}(\alpha_i^-\xi_i^-) + \sum_{i=1}^{n}(\beta_i^-\xi_i^-) + \\ & \sum_{i=1}^{n} \alpha_i^+(y_i) - \sum_{i=1}^{n} \alpha_i^+(\mathbf{w}^T\mathbf{x}_i) - \sum_{i=1}^{n} \alpha_i^+(b) - \sum_{i=1}^{n} \alpha_i^+(\epsilon) - \sum_{i=1}^{n} \alpha_i^+(\xi_i^+) - \\ & \sum_{i=1}^{n} \alpha_i^-(y_i) + \sum_{i=1}^{n} \alpha_i^-(\mathbf{w}^T\mathbf{x}_i) + \sum_{i=1}^{n} \alpha_i^-(b) - \sum_{i=1}^{n} \alpha_i^-(\epsilon) - \sum_{i=1}^{n} \alpha_i^-(\xi_i^+) + \\ & \sum_{i=1}^{n} \beta_i^+(-\xi_i^+) + \sum_{i=1}^{n} \beta^-(-\xi_i^-) \end{aligned}$$

$$L = \frac{1}{2}\left[\sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T\left[\sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \sum_{i=1}^{n}\alpha_i^+(y_i) - \sum_{i=1}^{n}\alpha_i^+(\mathbf{w}^T\mathbf{x}_i) - \sum_{i=1}^{n}\alpha_i^+(b) - \sum_{i=1}^{n}\alpha_i^+(\epsilon) - \sum_{i=1}^{n}\alpha_i^-(y_i) + \sum_{i=1}^{n}\alpha_i^-(\mathbf{w}^T\mathbf{x}_i) + \sum_{i=1}^{n}\alpha_i^-(b) - \sum_{i=1}^{n}\alpha_i^-(\epsilon) - \sum_{i=1}^{n}\alpha_i^-(\xi_i^+)$$

$$L = \frac{1}{2}\left[\sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right]^T\left[\sum_{i=1}^{n}\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \sum_{i=1}^{n}\alpha_i^+(y_i) - \sum_{i=1}^{n}\alpha_i^+(\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\mathbf{x}_i) - \sum_{i=1}^{n}\alpha_i^+(b) - \sum_{i=1}^{n}\alpha_i^+(\epsilon) - \sum_{i=1}^{n}\alpha_i^-(y_i) + \sum_{i=1}^{n}\alpha_i^-(\mathbf{x}_i(\alpha_i^+ - \alpha_i^-)\mathbf{x}_i) + \sum_{i=1}^{n}\alpha_i^-(b) - \sum_{i=1}^{n}\alpha_i^-(\epsilon)$$

$$L = \left[\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2||\mathbf{x}_i||^2\right] + \left[\sum_{i=1}^{n}(\alpha_i^+)^2||\mathbf{x}_i||^2\right] + \left[\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^-)^2||\mathbf{x}_i||^2\right] - \left[\sum_{i=1}^{n}(\alpha_i^+)^2||\mathbf{x}_i||^2\right] + \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)||\mathbf{x}_i||^2\right] + \left[\sum_{i=1}^{n}(\alpha_i^-)^2||\mathbf{x}_i||^2\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)||\mathbf{x}_i||^2\right] + \sum_{i=1}^{n}\alpha_i^+(y_i) - \sum_{i=1}^{n}\alpha_i^+(b) - \sum_{i=1}^{n}\alpha_i^+(\epsilon) - \sum_{i=1}^{n}\alpha_i^-(y_i) + \sum_{i=1}^{n}\alpha_i^-(b) - \sum_{i=1}^{n}\alpha_i^-(\epsilon)$$

$$L = \left[-\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2||\mathbf{x}_i||^2\right] + \left[\frac{3}{2}\sum_{i=1}^{n}(\alpha_i^-)^2||\mathbf{x}_i||^2\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)||\mathbf{x}_i||^2\right] + \sum_{i=1}^{n}\alpha_i^+(y_i) - \sum_{i=1}^{n}\alpha_i^+(b) - \sum_{i=1}^{n}\alpha_i^+(\epsilon) - \sum_{i=1}^{n}\alpha_i^-(y_i) + \sum_{i=1}^{n}\alpha_i^-(b) - \sum_{i=1}^{n}\alpha_i^-(\epsilon)$$

$$L = \left[-\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2||\mathbf{x}_i||^2\right] + \left[\frac{3}{2}\sum_{i=1}^{n}(\alpha_i^-)^2||\mathbf{x}_i||^2\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)||\mathbf{x}_i||^2\right] + \sum_{i=1}^{n}y_i(\alpha_i^+ - \alpha_i^-) + \sum_{i=1}^{n}b(\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n}\epsilon(\alpha_i^+ + \alpha_i^-)$$

$$L = \left[-\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2\langle\mathbf{x}_i,\mathbf{x}_i\rangle\right] + \left[\frac{3}{2}\sum_{i=1}^{n}(\alpha_i^-)^2\langle\mathbf{x}_i,\mathbf{x}_i\rangle\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)\langle\mathbf{x}_i,\mathbf{x}_i\rangle\right] + \sum_{i=1}^{n}y_i(\alpha_i^+ - \alpha_i^-) + \sum_{i=1}^{n}b(\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n}\epsilon(\alpha_i^+ + \alpha_i^-)$$

I know I need to combine constraints, but wasn't exactly sure how to do this. I believe it will be something like (using the lecture notes as a model)

$$\sum_{i=1}^{n} \alpha_i^+ y_i = 0$$
$$\sum_{i=1}^{n} \alpha_i^- y_i = 0$$
$$0 \leq \alpha_i^+ \leq \frac{C}{n}$$
$$0 \leq \alpha_i^- \leq \frac{C}{n}$$

So, the dual optimization problem will be

$$\max_{\alpha^+,\alpha^-} \quad \left[-\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2\langle \mathbf{x}_i, \mathbf{x}_i\rangle\right] + \left[\frac{3}{2}\sum_{i=1}^{n}(\alpha_i^-)^2\langle \mathbf{x}_i, \mathbf{x}_i\rangle\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-)\langle \mathbf{x}_i, \mathbf{x}_i\rangle\right] + \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) + \sum_{i=1}^{n} b(\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n} \epsilon(\alpha_i^+ + \alpha_i^-)$$

$$s.t. \quad \sum_{i=1}^{n} \alpha_i^+ y_i = 0$$
$$\sum_{i=1}^{n} \alpha_i^- y_i = 0$$
$$0 \leq \alpha_i^+ \leq \frac{C}{n}$$
$$0 \leq \alpha_i^- \leq \frac{C}{n}$$

c. Explain how to kernelize SVR. Be sure to explain how to recover w^* and b^*.

To kernelize SVR, we will use the dual optimization problem derived in part (b), and will replace the dot products $\langle \mathbf{x}_i, \mathbf{x}_i\rangle$ with a kernel $k(\mathbf{x}_i, \mathbf{x}_i)$.

So, our dual optimization problem becomes

$$\max_{\alpha^+,\alpha^-} \quad \left[-\frac{1}{2}\sum_{i=1}^{n}(\alpha_i^+)^2 k(\mathbf{x}_i, \mathbf{x}_i)\right] + \left[\frac{3}{2}\sum_{i=1}^{n}(\alpha_i^-)^2 k(\mathbf{x}_i, \mathbf{x}_i)\right] - \left[\sum_{i=1}^{n}(\alpha_i^+\alpha_i^-) k(\mathbf{x}_i, \mathbf{x}_i)\right] + \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) + \sum_{i=1}^{n} b(\alpha_i^+ + \alpha_i^-) + \sum_{i=1}^{n} \epsilon(\alpha_i^+ + \alpha_i^-)$$

$$s.t. \quad \sum_{i=1}^{n} \alpha_i^+ y_i = 0$$
$$\sum_{i=1}^{n} \alpha_i^- y_i = 0$$
$$0 \leq \alpha_i^+ \leq \frac{C}{n}$$
$$0 \leq \alpha_i^- \leq \frac{C}{n}$$

We can then recover $\mathbf{w}^*$ by recalling the property derived from the partial derivative of L with respect to $\mathbf{w}$. That is,

$$\mathbf{w} = \sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^+ - \alpha_i^-)$$

Defining the optimal values of α_i^+ and α_i^- as α_i^{+*} and α_i^{-*} respectively, we can recover $\mathbf{w}^*$ in the following manner:

$$\mathbf{w}^* = \sum_{i=1}^{n} \mathbf{x}_i(\alpha_i^{+*} - \alpha_i^{-*})$$

Next, we can recover b^* by letting α^{+*} be dual optimal. Then we will consider an i such that $0 \leq \alpha_i \leq \frac{C}{n}$. Since $\alpha_i > 0$, we know that (according to one of the KKT conditions) $y_i(k(\mathbf{w}^*, \mathbf{x}_i) + b^*) = 1 - \xi_i^*$. Since $\alpha_i^* < \frac{C}{n}$, by another KKT condition, we know that $\frac{C}{n} - \alpha_i^{+*} = \beta_i^* \geq 0 \implies \xi_i^* = 0$. We can then solve for b^* with the following equation:

$$b^* = y_i - k(\mathbf{w}^*, \mathbf{x}_i) = y_i \sum_{j=1}^{n} \alpha_j^* y_j k(\mathbf{x}_j, \mathbf{x}_i)$$

d. Argue that the final predictor will only depend on a subset of training examples (i.e. support vectors) and characterize those training examples.

The KKT conditions indicate that α^+ and α^- will be non-zero when $|f(\mathbf{x}_i) - y_i| \geq \epsilon$. In other words, the only observations that will impact α^+ and α^- will be those such that $f(\mathbf{x}_i)$ is further than ϵ away from y_i. The observations $\mathbf{x}_i$ such that $f(\mathbf{x}_i)$ that are within ϵ of y_i will not have an impact on the final predictor.

From complimentary slackness,

$$\alpha_i^{+*}(1 - \xi_i^{+*} - y_i(\langle \mathbf{w}^*, \mathbf{x}_i \rangle + b^*)) = 0.$$

If $\mathbf{x}_i$ satisfies $y_i(\langle \mathbf{w}^*, \mathbf{x}_i \rangle + b^*)) = 1 - \xi_i^{+*}$, then $\mathbf{x}_i$ is a support vector. Otherwise, $\alpha_i^* = 0$. So, $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^{+*} y_i \mathbf{x}_i = \sum_{Supportvectors} \alpha_i^{+*} y_i \mathbf{x}_i$.

3. Bagging and Logistic Regression

Download the `mnist_49_3000.mat` file from Canvas. This is the same handwritten digit dataset that was used in Homework 3. You will apply bagging to the logistic regression classifier on this dataset (you may use existing packages for logistic regression) and compare to other classifiers. Train using the first 2000 samples and test your classifier on the last 1000 samples.

a. Report the test error for 1) Random forests (you may use existing software; report what package and any parameters you use), 2) SVM without bagging (you may use existing software; report what package and any parameters you use including those selected with cross-validation), 3) logistic regression without bagging, 4) bagging+logistic regression with 50 bootstrapped samples, and 5) bagging+logistic regression with 100 bootstrapped samples. Which classifier performs the best? Does bagging seem to help the logistic regression classifier in this case? Make sure to keep your trained models for part (b).

- **Random Forests**: I used the `randomForest` package in R. The values I used for the parameters were the defaults set in the `randomForest()` function. (500 trees, $\sqrt{785}$ variables at each split, and a minimum node size of 1). The test error obtained with Random Forests was 0.015.
- **Support Vector Machines**: I used the implementation of SVMs in the `e1071` R package. I fit the SVM with a linear kernel, and settled on letting the cost parameter be 0.05 after some tuning. The SVM yielded a test error of 0.042.
- **Logistic Regression**: I used the `glm()` function in base R to fit the logistic regression model. Logistic regression without bagging gave a test error of 0.152.
- **Logistic Regression - bagging w/ 50 bootstrapped samples**: Obtained a test error rate of 0.116.
- **Logistic Regression - bagging w/ 100 bootstrapped samples**: Obtained a test error rate of 0.121

Yes, the bagging did help logistic regression perform better.

b. Download the mnist_49_1000_shifted.mat file from Canvas. This is the same test data as before except each of the images have been shifted. Apply each of the six trained classifiers from part (a) to this new, shifted test data and report the test error. Which classifier performs the best? Does bagging seem to help the logistic regression classifier in this case? If you knew your test data were likely to be shifted but your training data wasn't, what training strategy or strategies could you employ to obtain better test results? FYI, if the distribution of your test data differs from the distribution of your training data, then this problem is called transfer learning or covariate shift.

- **Random Forests**: Test error rate = 0.477
- **Support Vector Machines**: Test error rate = 0.504
- **Logistic Regression**: Test error rate = 0.391
- **Logistic Regression - bagging w/ 50 bootstrapped samples**: Test error rate = 0.491
- **Logistic Regression - bagging w/ 100 bootstrapped samples**: Test error rate = 0.551

No, bagging did not seem to help the logistic regression predictor in this case. This was surprising to me. As I understand bagging to work, I thought it might have done better on the shifted data, but it did not. If I had more time, I would investigate this further.

If I knew that my test data would be randomly shifted, I think I would purposefully introduce some sort of randomness into my training process to train on more similar data to my test data. For example, randomly shifting your training data so your algorithm learns on that data. Another option could be to incorporate bagging with the shifting training data. We could randomly shift each bootstrapped sample as we train each of our individual classifiers. That way, we would have introduced randomness into each classifier, and would likely get better lower test error when using the bagged classifiers to 'vote' on our final prediction.

Code:

```
library(R.matlab)
library(dplyr)
library(randomForest)
library(caret)
library(magrittr)
library(e1071)

# Read in and format data into dataframe with column "Y" and columns "X_1", "X_2", ...
df <- R.matlab::readMat("mnist_49_3000.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$Y <- as.factor(df$Y)


# Read in and format data into dataframe with column "Y" and columns "X_1", "X_2", ...
s_df <- R.matlab::readMat("mnist_49_1000_shifted.mat") %>% lapply(t) %>% lapply(as_tibble)
colnames(s_df[[1]]) <- sprintf("X_%s",seq(1:ncol(s_df[[1]])))
colnames(s_df[[2]]) <- c("Y")
s_df <- bind_cols(s_df) %>% select(Y, everything())
s_df$Y <- as.factor(s_df$Y)
test_sh <- s_df

################################################################################
### 0. Test/Train split ########################################################
################################################################################
```

```r
# create and format training data
train <- dplyr::slice(df, 1:2000)
trainPred <- select(train, -Y)
trainResp <- resp <- select(train, Y)

# create and format test data
test <- dplyr::slice(df, 2001:3000)
testPred <- select(test, -Y)
testResp <- select(test, Y)

# ##############################################################################
# ### 1. Random Forests ########################################################
# ##############################################################################

# Train random forest model, using default parameters
# ntree = 500
# mtry = sqrt(785) = 28.01785
# nodesize = 1
rf.model <- randomForest(Y~., data = train)

# predict onto test data
rf.predict <- predict(rf.model, testPred)
# rf.predict <- predict(rf.model, test_sh) # for predicting on shifted

rf.correct <- rf.predict == testResp$Y
rf.pcc <- sum(rf.correct)/length(rf.correct)
rf.testError <- 1 - rf.pcc
rf.testError
# Random Forest Test Error: 0.015

# ##############################################################################
# ### 2. Suppor Vector Machines ################################################
# ##############################################################################

# Tune linear-kernel SVM
svm.tune.linear <- tune.svm(Y~., data = train, kernel = 'linear', cost = 1.0*10^(-3:2))
# best: cost = 0.1

svm.tune.linear <- tune.svm(Y~., data = train, kernel = 'linear', cost = seq(0.01, 0.17, by = 0.03))
# best: cost = 0.07

svm.tune.linear <- tune.svm(Y~., data = train, kernel = 'linear', cost = seq(0.05, 0.07, by = 0.01))
# best: cost = 0.05

# train model
svm.model <- svm(Y~., data = train, kernel = 'linear', cost = 0.05)

svm.predict <- predict(svm.model, testPred)
# svm.predict <- predict(svm.model, test_sh) # for predicting on shifted

svm.correct <- svm.predict == testResp$Y
svm.pcc <- sum(svm.correct)/length(svm.correct)
svm.testError <- 1 - svm.pcc
```

```r
svm.testError
# SVM (linear kernel) Test Error: 0.042

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

# train logistic regression model
logReg.model <- glm(Y ~ ., data = train, family = binomial)

# predict on test data
logReg.predictProb <- predict(logReg.model, test, type = "response")
# logReg.predictProb <- predict(logReg.model, test_sh, type = "response") # for predicting on shifted

logReg.predict <- ifelse(logReg.predictProb > 0.5, "1", "-1")
logReg.correct <- logReg.predict == testResp$Y
logReg.pcc <- sum(logReg.correct)/length(logReg.correct)
logReg.testError <- 1 - logReg.pcc
logReg.testError # test error = 0.152

##############################################################################
#### Generate Bootstrap Samples ##############################################
##############################################################################
samp.size <- 2000

genBoot <- function(data, sample.size){
  smpl <- dplyr::sample_n(tbl = data, size = sample.size, replace = TRUE)
  return(smpl)
}

bs_list_50 <- list()
for (i in 1:50) {
  bs_list_50[[i]] <- genBoot(train, sample.size = samp.size)
}

bs_list_100 <- list()
for (i in 1:100) {
  bs_list_100[[i]] <- genBoot(train, sample.size = samp.size)
}

##############################################################################
#### Logistic Regression - Bagging w/ 50 Bootstrap Samples ###################
##############################################################################

lr.bag50 <- data.frame(n = double(nrow(test)))
lr.bag50.sh <- data.frame(n = double(nrow(test_sh)))

for(bs in bs_list_50){
  # train logistic regression model
  lr.bs.model <- glm(Y ~ ., data = bs, family = binomial)

  # predict on test data
  logReg.predictProb <- predict(lr.bs.model, test, type = "response")
```

```
  logReg.predictProb.shift <- predict(lr.bs.model, test_sh, type = "response")

  lr.bag50 <- cbind(lr.bag50, logReg.predictProb)
  lr.bag50.sh <- cbind(lr.bag50.sh, logReg.predictProb.shift)
}

lr.bag50.vote <- rowMeans(lr.bag50[,-1])
lr.bag50.predict <- ifelse(lr.bag50.vote > 0.5, "1", "-1")
lr.bag50.correct <- lr.bag50.predict == testResp$Y
lr.bag50.pcc <- sum(lr.bag50.correct)/length(lr.bag50.correct)
lr.bag50.testError <- 1 - lr.bag50.pcc
lr.bag50.testError # test error = 0.116

lr.bag50.vote <- rowMeans(lr.bag50.sh[,-1])
lr.bag50.predict <- ifelse(lr.bag50.vote > 0.5, "1", "-1")
lr.bag50.correct <- lr.bag50.predict == testResp$Y
lr.bag50.pcc <- sum(lr.bag50.correct)/length(lr.bag50.correct)
lr.bag50.testError.sh <- 1 - lr.bag50.pcc
lr.bag50.testError.sh # test error =

################################################################################
#### Logistic Regression - Bagging w/ 100 Bootstrap Samples ####################
################################################################################

lr.bag100 <- data.frame(n = double(nrow(test)))
lr.bag100.sh <- data.frame(n = double(nrow(test_sh)))

for(bs in bs_list_100){
  # train logistic regression model
  lr.bs.model <- glm(Y ~ ., data = bs, family = binomial)

  # predict on test data
  logReg.predictProb <- predict(lr.bs.model, test, type = "response")

  logReg.predictProb.shift <- predict(lr.bs.model, test_sh, type = "response")

  lr.bag100 <- cbind(lr.bag100, logReg.predictProb)
  lr.bag100.sh <- cbind(lr.bag100.sh, logReg.predictProb.shift)
}

lr.bag100.vote <- rowMeans(lr.bag100[,-1])
lr.bag100.predict <- ifelse(lr.bag100.vote > 0.5, "1", "-1")
lr.bag100.correct <- lr.bag100.predict == testResp$Y
lr.bag100.pcc <- sum(lr.bag100.correct)/length(lr.bag100.correct)
lr.bag100.testError <- 1 - lr.bag100.pcc
lr.bag100.testError # test error =

lr.bag100.vote <- rowMeans(lr.bag100.sh[,-1])
lr.bag100.predict <- ifelse(lr.bag100.vote > 0.5, "1", "-1")
lr.bag100.correct <- lr.bag100.predict == testResp$Y
lr.bag100.pcc <- sum(lr.bag100.correct)/length(lr.bag100.correct)
lr.bag100.testError.sh <- 1 - lr.bag100.pcc
lr.bag100.testError.sh # test error =
```

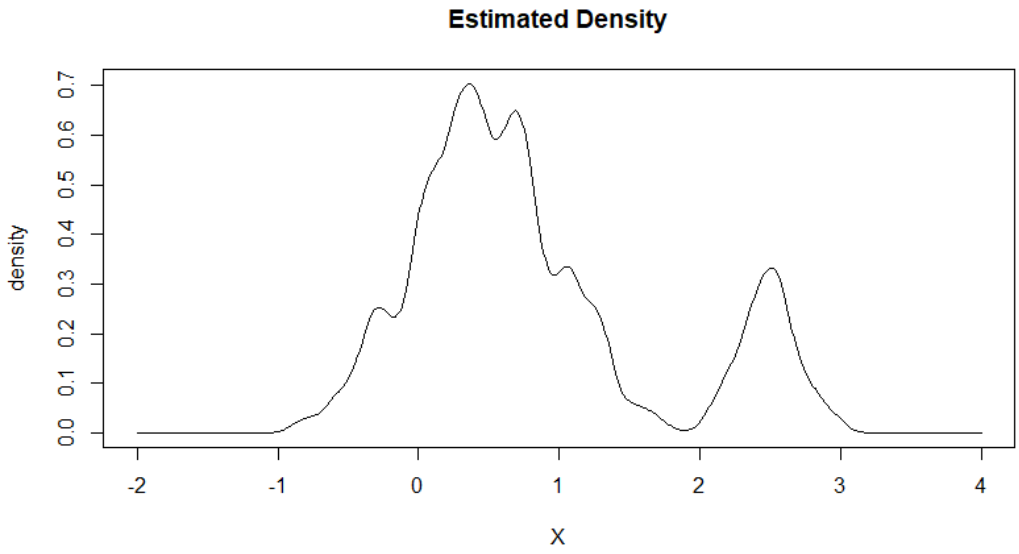
4. Outlier Detection with Kernel Density Estimation.

Download the `anomaly.mat file` from Canvas. This file has training data sampled from a univariate density f stored in the variable `X` and two test points `xtest1` and `xtest2`. The goal of this problem is to calculate a KDE of the density f using the training data and use the KDE to determine whether the two test points are outliers/anomalies.

a. Using least squares leave-one-out cross-validation with a Gaussian kernel an the training data, select a value for the bandwidth parameter. Report the bandwidth parameter.

Estimated bandwidth parameter: $\hat{h} = 0.08$.

b. Using the training data, estimate the density f at points uniformly spaced between -2 and 4 using a KDE with a Gaussian kernel and the bandwidth parameter selected in part (a). Include a plot of the KDE. Choose enough points between -2 and 4 so that the plot looks smooth. Based on the KDE, what do you think the true density f looks like?



Based on the KDE in the plot above, I think the true density is bimodal, with one peak around 0.5 and the other around 2.5.

c. There are multiple approaches for using the KDE in anomaly detection. Here is one approach. Let N be the number of points in the training data. Let x_j, $j = 1, ..., N$ be the training data. Let x be a point that you wish to test. Define an outlier score for x as

$$OutlierScore_1(x) = \frac{\hat{f}_h(x)}{\frac{1}{n}\sum_{j=1}^{N} \hat{f}_h^{(-j)}(x_j)}$$

Provide a conceptual interpretation of this score. I.e., would the score be low or high if x is an outlier? What if x is not an outlier? Are there any potential weaknesses of this score?

$OutlierScore_1(x)$ takes the kernel density estimate (with bandwidth parameter h) at x and divides by the average density as estimated by the leave one out estimator. If $\hat{f}_h(x)$ is close to the average density, $OutlierScore_1 \approx 1$. Otherwise, it will be "far away" from 1. If the density at x is high (relative to the average), the score will be large; if the density at x is low (relative to the average), the score will be small. So a small score will correspond with an outlier.

However, this score does have some weaknesses. If a distribution has small varaiance, with a large unimodal peak, the average density will be relatively high. On the other hand, if a distribution has larger variance

with several multimodal peaks, the average density will be relatively low. If the variance or dispersion was somehow taken into account in this score, it coule be improved.

d. Calculate the $OutlierScore_1$ for `xtest1` and `xtest2` using the same h you selected in part (a). Based on the calculated scores, do you think either of these points are outliers?

$OutlierScore_1(\texttt{xtest1}) = 0.178$
$OutlierScore_1(\texttt{xtest2}) = 1.029 \times 10^{-16}$

Because $OutlierScore_1(\texttt{xtest2})$ is so small, it is likely that it is an outlier.

e. $OutlierScore_1$ has some weaknesses. Let's define another score based on k-nearest neighbors. Let $\rho_k(x)$ be the distance of x to its kth nearest neighbor in the training data. Let $\mathcal{N}(x)$ be the set of k nearest neighbors of x in the training data. So $|\mathcal{N}(x)| = k$. Define another outlier score for x as

$$OutlierScore_2(x) = \frac{\rho_k(x)}{\frac{1}{k}\sum_{x_i \in \mathcal{N}(x)} \rho_k(x_i)},$$

where $\rho_k(x_i)$ is the distance of x_i to its kth nearest neighbor in the training data (not including x_i). Provde a conceptual interpretation of this score. I.e., would the score be low or high if x is an outlier? What if x is not an outlier? How is this score related to k-nn density estimation? What potential advantages would this score have over the one defined in part (c)?

This $OutlierScore_2$ is constructed by taking the distance from the point in question to the kth nearest neighbor and dividing it by the average distance from the point in question to its k nearest neighbors. If $OutlierScore_2$ is large, that means that the distance to the kth nearest neighbor is much larger than the average distance to the k nearest neighbors. This implies that there are many points closer to the point in question than the kth nearest neighbor, and the point is likely not an outlier. On the other hand, if $OutlierScore_2$ is small, that means that the average distance to the k nearest neighbors is larger (or as large as) the distance to the k nearest neighbors. So a smaller score implies that the observation is more likely to be an outlier.

It is related to k nearest neighbors in that it has an inherently adaptive bandwidth, and will yield more robust results than $OutlierScore_1$.

f. Calculate the $OutlierScore_2$ for `xtest1` and `xtest2` using the same h you selected in part (a). Based on the calculated scores, do you think either of these points are outliers?

$k = \mathbf{100}$: $OutlierScore_2(\texttt{xtest1}) = 2.847$
$k = \mathbf{150}$: $OutlierScore_2(\texttt{xtest1}) = 2.008$
$k = \mathbf{200}$: $OutlierScore_2(\texttt{xtest1}) = 1.757$

$k = \mathbf{100}$: $OutlierScore_2(\texttt{xtest2}) = 1.213$
$k = \mathbf{150}$: $OutlierScore_2(\texttt{xtest2}) = 1.228$
$k = \mathbf{200}$: $OutlierScore_2(\texttt{xtest2}) = 1.235$

According to the scores, if either of these points are outliers, it would be `xtest2`, since it has the smaller value of $OutlierScore_2$.

g. Include your code:

```
library(geometry)
library(tidyr)
library(dplyr)

################################################################################
#### Read in Data ##############################################################
################################################################################
```

```r
# training data is in train$X
df <- R.matlab::readMat("anomaly.mat") %>% lapply(t) %>% lapply(as_tibble)
train <- as.data.frame(df$X)
names(train)[1] <- 'X'
train$index <- rownames(train)

# test points 1 and 2
tp1 <- df$xtest1$V1
tp2 <- df$xtest2$V1

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

gaussKernel <- function(vec, h){
  # calculates gaussian kernel for a given vector of values.
  # Args:
  #   vec: vector of numeric values for which the kernel is to be estimated.
  #   h: bandwidth parameter.
  d <- length(vec)
  vec.h <- vec/h
  if(d > 1){
    kern <- ((2*pi)^(-d/2))*exp(-0.5*(dot(vec.h, vec.h)))
  } else {
    kern <- ((2*pi)^(-d/2))*exp(-0.5*(vec.h * vec.h))
  }
  return((h^(-d)) * kern)
}

estBandWidth <- function(df, h){
  # calculates objective function for LS-LOOCV
  # Args:
  #   df: data frame of training data (train) containing index and X .
  #   h: bandwidth parameter.
  n <- nrow(df)
  t1 <- 0
  t2 <- 0
  ind <- df$index
  X <- df$X

  for(i in 1:n){
    for(j in 1:n){
      newTerm1 <- gaussKernel(vec = (X[i] - X[j]), h = (sqrt(2)*h))
      t1 <- t1 + newTerm1
      if(j != i){
        newTerm2 <- gaussKernel(vec = (X[i] - X[j]), h = h)
        t2 <- t2 + newTerm2
      }
    }
  }
  t1 <- (1/(n^2)) * t1
  t2 <- (2/(n*(n-1))) * t2
  result <- t1 - t2
```

```
  return(result)
}


kdeRange <- function(range, X, h){
  # calculates kernel density estimate for points in range. Calls kdePoint for
  # all values in range.
  # Args:
  #   range: vector of numeric values for which the KDE is to be calculated
  #   X: training data
  #   h: bandwidth parameter.
  #
  densityEst <- data.frame(x = range, density = double(length(range)))
  for(i in 1:length(range)){
    densityEst$density[i] <- kdePoint(point = range[i], X = X, h = h)
  }
  return(densityEst)
}

kdePoint <- function(point, X, h){
  # calculates kernel desnsity estimate for point. Called by kdeRange
  # Args:
  #   point: point for which density is to be estimated.
  #   X: training data
  #   h: bandwidth parameter.
  term <- 0
  n <- length(X)
  for(i in 1:n){
    val <- gaussKernel(vec = (X[i] - point), h = h)
    term <- term + val
  }
  fhat <- (1/n) * term
  return(fhat)
}

fhat.i <- function(point, X, exclude.i, h){
  # calculates leave one out kernel density estiamate.
  # Used by outlierScore1().
  # Args:
  #   point: point for which density is to be estimated.
  #   X: training data
  #   exclude.i: index for point which is to be excluded
  #   h: bandwidth parameter.
  term <- 0
  n <- length(X)
  for(i in 1:n){
    if(i != exclude.i){
      val <- gaussKernel(vec = (point - X[i]), h = h)
      term <- term + val
    }
  }
  fhat <- (1/(n-1)) * term
  return(fhat)
```

```
}

outlierScore1 <- function(point, X, h){
  # calculates OutlierScore_1 for point.
  # Args:
  #   point: point for which outlier score is to be calculated.
  #   X: training data
  #   h: bandwidth parameter.
  n <- length(X)
  numerator <- kdePoint(point = point, X = X, h = h)
  denominator <- 0
  for(i in 1:n){
    term<- fhat.i(point = X[i], X = X, exclude.i = i, h = h)
    denominator <- term + denominator
  }
  denominator <- (1/n) * denominator
  return(numerator/denominator)
}

getKnn <- function(point, X, k){
  # Finds k nearest neighbors and distances. Used by outlierScore2()
  # Args:
  #   point: point for which k nearest neighbors are to be obtained.
  #   X: training data
  #   k: number of nearest neighbors to get
  pointvec <- rep(point, length(X))
  dist <- abs(X - pointvec)
  df <- data.frame(X = X, dist = dist)
  df <- arrange(df, dist)
  knn <- slice(df, 1:k)
  return(knn)
}

outlierScore2 <- function(point, X, k){
  # calculates OultierScore_2 for point
  # Args:
  #   point: point for which OutlierScore_2 is to be estimated.
  #   X: training data
  #   k: number of nearest neighbors to calculate
  knn <- getKnn(point, X, k)
  numerator <- knn$dist[k]

  denominator <- (1/k) * sum(knn$dist)

  return(numerator/denominator)
}

################################################################################
#### To Run ####################################################################
################################################################################

# grid of possible bandwidths
grid <- seq(.01,.5, by = .01)
```

```r
gridn <- length(grid)

# calculate objective function from LS-LOOCV for each value in grid
hs <- data.frame(index = 1:gridn, h = grid, objF = rep(0, gridn))
for(i in 1:nrow(hs)){
  objF <- estBandWidth(df = train, h = hs$h[i])
  hs$index[i] <- i
  hs$objF[i] <- objF
}

# plot objective function by h
plot(hs$h, hs$objF)

# get h for which objective function is minimized.
min.h <- hs[hs$objF == min(hs$objF),]
hhat <- min.h$h

# hhat is optimal value of h.
hhat

densRange <- seq(-2, 4, by = 0.01)
# estimate density
densEst <- kdeRange(range = densRange, X = train$X, h = hhat)
# plot estimated density
plot(densEst$x, densEst$density, type = 'l', main = "Estimated Density", xlab = "X", ylab = "density")

# calculate OutlierScore_1 for xtest1 and xtest2
os1.tp1 <- outlierScore1(point = tp1, X = train$X, h = 0.08)
os1.tp2 <- outlierScore1(point = tp2, X = train$X, h = 0.08)

os1.tp1
os1.tp2

# calculate OutlierScore_2 for xtest1 and xtest2, with various values of k
os2.tp1.100 <- outlierScore2(point = tp1, X = train$X, k = 100)
os2.tp1.150 <- outlierScore2(point = tp1, X = train$X, k = 150)
os2.tp1.200 <- outlierScore2(point = tp1, X = train$X, k = 200)

os2.tp1.100
os2.tp1.150
os2.tp1.200

os2.tp2.100 <- outlierScore2(point = tp2, X = train$X, k = 100)
os2.tp2.150 <- outlierScore2(point = tp2, X = train$X, k = 150)
os2.tp2.200 <- outlierScore2(point = tp2, X = train$X, k = 200)

os2.tp2.100
os2.tp2.150
os2.tp2.200
```

5. How long did this assignment take you?

About 35 hours.

6. Type up homework solutions.

Check.