# Stat 6910, Section 003
# Statistical Learning and Data Mining II
# Fall 2018

## Homework 5


## Jared Hansen

A-number: A01439768

e-mail: jrdhansen@gmail.com


Due: 5:00 PM, Friday 11/30/18

1. **Variance of correlated random variables (5 pts).** Suppose we have random variables $X_1, \ldots, X_B$ 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 $\rho$. I.e., $corr(X_i, X_j) = \rho > 0$ for each $i \neq j$.

   (a) (3 pts) 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.
   *Hint:* You may use the facts that $\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] = \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} Cov(X_i, X_j)$ and $Cov(X_i, X_i) = \mathbb{V}[X_i]$.

   The idea I'm going to use is taking the first hint (definition of $\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right]$) and manipulating it until it looks like $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$.

   Via the hints:

$$\begin{aligned}
\mathbb{V}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] &= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} Cov(X_i, X_j) \\
&= \frac{1}{B^2}\left[Cov(X_1, X_1) + Cov(X_2, X_2) + \ldots + Cov(X_B, X_B) + \sum_{i=1, j=i,}^{B}\sum_{i\neq j}^{B} Cov(X_i, X_j)\right] \\
&= \frac{1}{B^2}\left[\mathbb{V}[X_1] + \mathbb{V}[X_2] + \ldots + \mathbb{V}[X_B] + \sum_{i=1, j=i,}^{B}\sum_{i\neq j}^{B} Cov(X_i, X_j)\right]
\end{aligned}$$

   Now we'll need to make use of the mathematical definition of the correlation coefficient $\rho$:

$$\rho_{X,Y} := \frac{Cov(X,Y)}{\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}} \implies Cov(X,Y) = \rho_{X,Y}\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}$$

   Now, distribute the $\frac{1}{B^2}$, use the fact that $V[X_i] = \sigma^2$, and use the definition of $\rho$ to get:

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

   The double sum would have $B^2$ terms, but since there will be exactly $B$ terms excluded (per the condition that $i \neq j$) we'll have $B^2 - B$ terms. So now:

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

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

   We have proven what was desired, so we're done with this problem. Nice.

(b) (2 pts) Explain why it would not make sense (i.e. it isn't possible) for $\rho$ to be negative when $B \geq 3$.

Let's think about the simplest case: when $B = 3$ and $Corr(X_i, X_j) = \rho < 0 \ \forall i \neq j$, and these three random variables are $X_1$, $X_2$, and $X_3$.
Per our assumption, all of the pairwise correlations $\rho_{i,j} \ \forall i \neq j$ are $< 0$ (where $i, j \in \{1, 2, 3\}$). However, we will reason that this is not possible (as the question asks).

- If $X_1$ and $X_2$ are negatively correlated, we can say something like: as $X_1$ increases, $X_2$ decreases.
- Similarly, if $X_2$ and $X_2$ are negatively correlated, we can say something like: as $X_3$ increases, $X_2$ decreases.
- BUT, if both $X_1$ and $X_3$ are increasing as $X_2$ is decreasing, it means that these two variables ($X_1$ and $X_3$) are positively correlated with one another, and
  $Corr(X_1, X_3) = \rho_{1,3} > 0 \implies$ it isn't possible for all $\rho_{i,j} < 0 \ \forall i \neq j$ when $B = 3$.
- This generalizes to cases where $B > 3$ by realizing that there will always be at least one $\rho_{i,j} > 0$ by applying the same logic to more random variables $X_i$ where $i, j \in \{1, 2, 3, \ldots, B\}$ and $i \neq j$ and $B > 3$.

2. **Support Vector Regression (25 pts).** Support vector regression (SVR) is a method for regression analogous to the support vector classifier. Let $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, $i = 1, \ldots, n$ be training data for a regression problem. In the case of linear regression, SVR solves

$$\begin{aligned} \min_{\mathbf{w},b,\xi^+,\xi^-} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\left(\xi_i^+ + \xi_i^-\right) \\ \text{s.t.} \quad & y_i - \mathbf{w}^T\mathbf{x}_i - b \leq \epsilon + \xi_i^+ \ \forall i \\ & \mathbf{w}^T\mathbf{x}_i + b - y_i \leq \epsilon + \xi_i^- \ \forall i \\ & \xi_i^+ \geq 0 \ \forall i \\ & \xi_i^- \geq 0 \ \forall i \end{aligned}$$

where $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, $\xi^+ = \left(\xi_1^+, \ldots, \xi_n^+\right)^T$, and $\xi^- = \left(\xi_1^-, \ldots, \xi_n^-\right)^T$. Here $\epsilon$ is fixed.

(a) (5 pts) Show that for an appropriate choice of $\lambda$, SVR solves

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

where $\ell_\epsilon(y,t) = \max\{0, |y-t| - \epsilon\}$is the $\epsilon$-insensitive loss, which does not penalize prediction errors below a level of $\epsilon$.

- In order to do this, I'm going to show that for a specific $\lambda$, the SVR equation $\left[\min_{\mathbf{w},b,\xi^+,\xi^-} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\left(\xi_i^+ + \xi_i^-\right)\right]$ solves $\left[\min_{\mathbf{w},b} \frac{1}{n}\sum_{i=1}^{n}\ell_\epsilon\left(y_i, \mathbf{w}^T\mathbf{x}_i + b\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right]$.
- Since we're attempting to minimize both of these functions, what we'll need to do is show that $\left[\frac{1}{n}\sum_{i=1}^{n}\ell_\epsilon\left(y_i, \mathbf{w}^T\mathbf{x}_i + b\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2\right] \equiv \left[\frac{C}{n}\sum_{i=1}^{n}\left(\xi_i^+ + \xi_i^-\right) + \frac{1}{2}\|\mathbf{w}\|^2\right]$ through manipulation by scaling SVR and manipulating/combining constraints.
- Start by scaling the SVR equation by $\lambda = \frac{1}{C}$. (This is legal and won't change the solution. We did the same thing for the SVC in course notes.) Now we have: $\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n}\left(\xi_i^+ + \xi_i^-\right)$. All we need to do now is make $\left(\xi_i^+ + \xi_i^-\right)$ equivalent to $\ell_\epsilon(y,t)$ by working with the SVR constraints.
- Re-write the constraints, letting $t_i = \langle \mathbf{w}^T, \mathbf{x}_i \rangle + b$:
  i. $y_i - t_i \leq \epsilon + \xi_i^+$
  ii. $-y_i + t_i \leq \epsilon + \xi_i^-$
  iii. $0 \leq \xi_i^+$
  iv. $0 \leq \xi_i^-$

- Now we'll consider 3 cases in order to combine these constraints:
  - **CASE 1:** $(y_i - t_i) > 0$
    When $(y_i - t_i) > 0$ and $\xi_i^+ > 0$ (satisfying constraints (i) and (iii)), we can satisfy constraints (ii) and (iv) by letting $\xi_i^- = 0$.
    We know if $(y_i - t_i) > 0 \implies (-y_i + t_i) < 0 \implies$ constraint (ii) is satisfied by $\xi_i^- = 0$ since $\epsilon > 0$, $\implies \left[(-y_i + t_i) < 0\right] < \left[(\epsilon + \xi_i^-) = (\epsilon > 0)\right]$, satisfying constraint (ii).
    The reason we set $\xi_i^- = 0$ is to minimize $(\xi_i^+ + \xi_i^-)$, which will minimize our objective function since the objective function is just the sum $\forall i$ of $(\xi_i^+ + \xi_i^-)$ + the regularization term.
    We could set $\xi_i^- > 0$ since it would still satisfy all 4 constraints, BUT it would give a larger value of the objective function than $\xi_i^- = 0$. So, since $\xi_i^- = 0$ minimizes the objective function for this case AND still satisfies all 4 constraints, this is what we'll have for CASE 1.
  - **CASE 2:** $(-y_i + t_i) > 0$
    Similarly, when $(-y_i + t_i) > 0$ and $\xi_i^- > 0$ (satisfying constraints (ii) and (iv)), we can satisfy constraints (i) and (iii) by letting $\xi_i^+ = 0$. This uses the same logic as above, simply switching $\xi_i^+$ and $\xi_i^-$. In the interest of space and redundancy, I won't repeat this logic here.
  - **CASE 3:** $(y_i - t_i) = 0$
    When $(y_i - t_i) = 0 \implies (-y_i + t_i) = 0$, and all 4 constraints are satisfied with $\xi_i^+ = \xi_i^- = 0$. This obviously minimizes $(\xi_i^+ + \xi_i^-)$, and hence minimizes the objective function.

- Combining CASE 1 and CASE 2, we can simply say either $\xi_i^+ = 0$ or $\xi_i^- = 0$. The other case will give a value $\geq (|y_i - t_i| - \epsilon)$. We'll obviously choose the other case such that we use the $= (|y_i - t_i| - \epsilon)$ in order to give the smallest quantity possible and minimize the objective function.
  Obviously CASE 3 is the ideal since $\xi_i^+ + \xi_i^- = 0$, but this doesn't happen in all cases. Therefore, to account for all possibilities, we'll combine the values of CASE1+CASE2 with CASE3.
  In order to combine CASE3 $\left(\xi_i^+ = \xi_i^- = 0 \implies \xi_i^+ + \xi_i^- = 0\right)$ and
  CASE1+CASE2 $\left(\xi_i^+ = 0, \xi_i^- > 0 \text{ OR } \xi_i^+ > 0, \xi_i^- = 0 \implies \xi_i^+ + \xi_i^- = 0 + |y - t| - \epsilon\right)$ we must take the maximum to ensure that the 4 constraints remain satisfied.
  That is, $(\xi_i^+ + \xi_i^-) \equiv max(0, |y - t| - \epsilon)$.

- Substituting into our $\lambda$-scaled SVR equation of $\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^n(\xi_i^+ + \xi_i^-)$ we now have $\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^n max(0, |y - t| - \epsilon)$. From the prompt, we have defined $max(0, |y - t| - \epsilon) = \ell_\epsilon(y, t)$. If we let $t_i = \langle \mathbf{w}^T, \mathbf{x}_i \rangle + b$ as done previously in the problem $\implies max(0, |y - t| - \epsilon) = \ell_\epsilon(y_i, \mathbf{w}^T\mathbf{x}_i + b)$. Substituting this into our current equation, $\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^n max(0, |y-t| - \epsilon) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^n \ell_\epsilon(y_i, \mathbf{w}^T\mathbf{x}_i + b)$. This was the desired equation.

- Therefore, by choosing $\lambda = \frac{1}{C}$ and manipulating/combining constraints, we've shown that $\left[\frac{1}{n}\sum_{i=1}^n \ell_\epsilon\left(y_i, \mathbf{w}^T\mathbf{x}_i + b\right) + \frac{\lambda}{2}\left\|\mathbf{w}\right\|^2\right] \equiv \left[\frac{C}{n}\sum_{i=1}^n \left(\xi_i^+ + \xi_i^-\right) + \frac{1}{2}\left\|\mathbf{w}\right\|^2\right]$, implying that SVR solves $min_{\mathbf{w},b}\frac{1}{n}\sum_{i=1}^n \ell_\epsilon\left(y_i, \mathbf{w}^T\mathbf{x}_i + b\right) + \frac{\lambda}{2}\left\|\mathbf{w}\right\|^2$ which is what the prompt wanted. So we're done. Yay.

(b) (13 pts) 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 $\xi_i^+ \geq 0$, $\xi_i^- \geq 0$.

- Lagrangian function: $L(\boldsymbol{x}, \lambda, \nu)$
  - $L(\boldsymbol{x}, \lambda, \nu) := f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{i=1}^{n} \nu_j h_j(x)$
    where $g_i(x) \leq 0$ for $i \in \{1, 2, \ldots, n\}$
    and $h_j(x) = 0$ for $j \in \{1, 2, \ldots, n\}$

  - First, rewrite the given constraints for this problem such that they're of the form $g_i(x) \leq 0$ (we aren't given any equality constraints in this problem):
    1.....$y_i - \boldsymbol{w}^T \boldsymbol{x}_i - b - \epsilon - \xi_i^+ \leq 0, \forall i$
    2.... $- y_i + \boldsymbol{w}^T \boldsymbol{x}_i + b - \epsilon - \xi_i^- \leq 0, \forall i$
    3..... $- \xi_i^+ \leq 0, \forall i$
    4..... $- \xi_i^- \leq 0, \forall i$

  - Now, write the Lagrangian, $L(\boldsymbol{w}, b, \xi_i^+, \xi_i^-, \alpha_i^+, \alpha_i^-, \beta_i^+, \beta_i^-)$:
    $$L(\boldsymbol{w}, b, \xi_i^+, \xi_i^-, \alpha_i^+, \alpha_i^-, \beta_i^+, \beta_i^-) = \left[\tfrac{1}{2}\|\boldsymbol{w}\|^2\right] + \left[\tfrac{C}{n} \sum_{i=1}^{n} \left(\xi_i^+ + \xi_i^-\right)\right]$$
    $$+ \left[\sum_{i=1}^{n} \alpha_i^+ \left(y_i - \boldsymbol{w}^T \boldsymbol{x}_i - b - \epsilon - \xi_i^+\right)\right] + \left[\sum_{i=1}^{n} \alpha_i^- \left(- y_i + \boldsymbol{w}^T \boldsymbol{x}_i + b - \epsilon - \xi_i^-\right)\right]$$
    $$+ \left[\sum_{i=1}^{n} \beta_i^+ (-\xi_i^+)\right] + \left[\sum_{i=1}^{n} \beta_i^- (-\xi_i^-)\right]$$

- Now take partial derivatives of $L$ wrt to $\boldsymbol{w}, b, \xi_i^+, \xi_i^-$ and set $= 0$ to define KKT conditions for this problem:
  - $\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i^+ \boldsymbol{x}_i + \sum_{i=1}^{n} \alpha_i^- \boldsymbol{x}_i \overset{set}{=} 0 \implies \boldsymbol{w} = \sum_{i=1}^{n} \boldsymbol{x}_i \left(\alpha_i^+ - \alpha_i^-\right)$
  - $\frac{\partial L}{\partial b} = - \sum_{i=1}^{n} \alpha_i^+ + \sum_{i=1}^{n} \alpha_i^- \overset{set}{=} 0 \implies \sum_{i=1}^{n} \alpha_i^+ = \sum_{i=1}^{n} \alpha_i^-$
  - $\frac{\partial L}{\partial \xi_i^+}$ : we only care about taking the derivative wrt to terms that have an $\xi^+$ in them
    Take $\frac{\partial L}{\partial \xi_i^+}$ of $\left[\frac{C}{n}\left(\xi_1^+ + \xi_2^+ + \ldots\right) - \left(\alpha_1^+ \xi_1^+ + \alpha_2^+ \xi_2^+ + \ldots\right) - \left(\beta_1^+ \xi_1^+ + \beta_2^+ \xi_2^+ + \ldots\right)\right]$
    The partial wrt some $\xi_i^+$ (some specific $i$) will be: $\left[\frac{C}{n}\left(1\right) - \left(1\alpha_i^+\right) - \left(1\beta_i^+\right)\right]$
    $= \frac{C}{n} - \alpha_i^+ - \beta_i^+$ giving $\frac{C}{n} - \alpha_i^+ - \beta_i^+ \overset{set}{=} 0$
  - $\frac{\partial L}{\partial \xi_i^-}$ : exact same process as for $\frac{\partial L}{\partial \xi_i^+}$, but use $\xi_i^-$ instead, giving: $\frac{C}{n} - \alpha_i^- - \beta_i^- \overset{set}{=} 0$

- Now expand the Lagrangian, giving:
  $L =$
  $$\tfrac{1}{2}\|\boldsymbol{w}\|^2 + \tfrac{C}{n} \sum_{i=1}^{n} \xi_i^+ + \tfrac{C}{n} \sum_{i=1}^{n} \xi_i^-$$
  $$+ \sum_{i=1}^{n} \alpha_i^+ y_i - \sum_{i=1}^{n} \alpha_i^+ \boldsymbol{w}^T \boldsymbol{x}_i - \sum_{i-1}^{n} \alpha_i^+ b - \sum_{i=1}^{n} \alpha_i^+ \epsilon - \sum_{i=1}^{n} \alpha_i^+ \xi_i^+$$
  $$- \sum_{i=1}^{n} \alpha_i^- y_i + \sum_{i=1}^{n} \alpha_i^- \boldsymbol{w}^T \boldsymbol{x}_i + \sum_{i-1}^{n} \alpha_i^- b - \sum_{i=1}^{n} \alpha_i^- \epsilon - \sum_{i=1}^{n} \alpha_i^- \xi_i^-$$
  $$+ \sum_{i=1}^{n} \beta_i^+ (-\xi_i^+) + \sum_{i=1}^{n} \beta_i^- (-\xi_i^-)$$

- Now substitute terms in $L$ using some of the following values:

  $\frac{C}{n} = \alpha_i^+ + \beta_i^+$ OR $\frac{C}{n} = \alpha_i^- + \beta_i^-$, $\sum_{i=1}^n \alpha_i^- = \sum_{i=1}^n \alpha_i^+$

  Now we'll have:

  $$L = \left[\frac{1}{2}\sum_{i=1}^n \boldsymbol{x}_i(\alpha_i^+ - \alpha_i^-)\right] + \left[\sum_{i=1}^n \alpha_i^+\xi_i^+\right] + \left[\sum_{i=1}^n \beta_i^+\xi_i^+\right] + \left[\sum_{i=1}^n \alpha_i^-\xi_i^-\right] + \left[\sum_{i=1}^n \beta_i^-\xi_i^-\right]$$
  $$+ \leftarrow \ldots \text{ same as } L \text{ above } \ldots \rightarrow + \ldots - \left[\sum_{i=1}^n \alpha_i^+\xi_i^+\right]$$
  $$- \leftarrow \ldots \text{ same as } L \text{ above } \ldots \rightarrow + \ldots - \left[\sum_{i=1}^n \alpha_i^-\xi_i^-\right]$$
  $$- \left[\sum_{i=1}^n \beta_i^+\xi_i^+\right] - \left[\sum_{i=1}^n \beta_i^-\xi_i^-\right]$$

- Now sub in for $\boldsymbol{w} = \sum_{i=1}^n \boldsymbol{x}_i(\alpha_i^+ + \alpha_i^-)$ and cancel things out from previous $L$ to get:

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

- For the next part we'll need to make use of this identity:

  $$\frac{1}{2}\left[\sum_{i=1}^n (\alpha_i^+ - \alpha_i^-)^2\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\right] = \frac{1}{2}\left[\sum_{i=1}^n (\alpha_i^+)^2\|\boldsymbol{x}_i\|^2 - 2\sum_{i=1}^n \alpha_i^+\alpha_i^-\|\boldsymbol{x}_i\|^2 + \sum_{i=1}^n (\alpha_i^-)^2\|\boldsymbol{x}_i\|^2\right]$$

- So now we have:

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

  using the KKT condition $\sum_{i=1}^n \alpha_i^+ = \sum_{i=1}^n \alpha_i^- \implies \sum_{i=1}^n (\alpha_i^+ - \alpha_i^-) = 0$

  $$L = \leftarrow \left[\ldots \text{ same as } L \text{ above } \ldots\right] \rightarrow$$
  $$+ \left[\sum_{i=1}^n y_i(\alpha_i^+ - \alpha_i^-)\right] + \left[b\sum_{i=1}^n 0\right] + \left[\sum_{i=1}^n -\epsilon(\alpha_i^+ + \alpha_i^-)\right]$$

- Now we have:
  $L_D(\alpha^+, \alpha^-) =$
  $-\frac{1}{2}\left[\sum_{i=1}^{n}(\alpha_i^+)^2\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\right] - \left[\sum_{i=1}^{n}\alpha_i^+\alpha_i^-\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\right] + \frac{3}{2}\left[\sum_{i=1}^{n}(\alpha_i^-)^2\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\right]$
  $+\left[\sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-)\right] + \left[0\right] - \epsilon\left[\sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-)\right]$
- Dual Optimization Problem:
  $\max_{\alpha^+, \alpha^-, \beta^+, \beta^-}\left[L_D(\alpha^+, \alpha^-)\right]$ such that the KKT conditions and the $\nabla$ conditions are met.
  KKT conditions: $\left[\alpha_i^+ \geq 0, \alpha_i^- \geq 0, \beta_i^+ \geq 0, \beta_i^- \geq 0\right] \forall i$
  $\nabla$ conditions: $\left[\sum_{i=1}^{n}\alpha_i^+ - \sum_{i=1}^{n}\alpha_i^- = 0, \frac{C}{n} - \alpha_i^+ - \beta_i^+ = 0, \frac{C}{n} - \alpha_i^- - \beta_i^- = 0\right]$
  We can combine these conditions to get a more succinct set of constraints.
  Combined constraints:
  $0 \leq \alpha_i^+ \leq \frac{C}{n} \forall i$
  $0 \leq \alpha_i^- \leq \frac{C}{n} \forall i$
  $0 = \sum_{i=1}^{n}\alpha_i^+ - \sum_{i=1}^{n}\alpha_i^-$

- In summary, the dual optimization can be written as (this is the final answer for the problem):

  $$\max_{\alpha^+, \alpha^-}\Bigg( -\frac{1}{2}\Big[\sum_{i=1}^{n}(\alpha_i^+)^2\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\Big] - \Big[\sum_{i=1}^{n}\alpha_i^+\alpha_i^-\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\Big] + \frac{3}{2}\Big[\sum_{i=1}^{n}(\alpha_i^-)^2\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle\Big]$$
  $$+\Big[\sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-)\Big] - \epsilon\Big[\sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-)\Big]\Bigg)$$

  such that:
  $0 \leq \alpha_i^+ \leq \frac{C}{n} \forall i$
  $0 \leq \alpha_i^- \leq \frac{C}{n} \forall i$
  $0 = \sum_{i=1}^{n}\alpha_i^+ - \sum_{i=1}^{n}\alpha_i^-$

(c) (4 pts) Explain how to kernelize SVR. Be sure to explain how to recover $\mathbf{w}^*$ and $b^*$.

We will kernelize SVR similar to how we kernelized the SVC in class. After looking at doing it a few different ways, I talked to you and you suggested doing it just for one dual optimal variable at a time, so that's what I'm going to do.

- Let $\alpha^{+*}$ be dual optimal.
- From the KKT conditions we saw that $\boldsymbol{w} = \sum_{i=1}^{n} \boldsymbol{x}_i\big((\alpha^+) - (\alpha^-)\big)$.

  Therefore $\boldsymbol{w}^* = \sum_{i=1}^{n} \boldsymbol{x}_i\big(\alpha^{+*} - \alpha^{-*}\big)$ (**This is the answer for how to recover $\boldsymbol{w}^*$.**)
- Consider $i$ such that $0 < \alpha_i^{+*} < \frac{C}{n}$.
- Since $\alpha_i^{*+} > 0$ we know by the KKT conditions that
  $y_i - \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle - b^* = \epsilon + \xi_i^+$ and $-y_i + \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^* = \epsilon + \xi_i^-$
- Since $\alpha^{+*} < \frac{C}{n}$ by another of the KKT conditions we get $\frac{C}{n} - \alpha_i^{+*} = \beta_i^{+*} > 0 \implies \xi_i^+ = 0$
  (Similar logic is used to show $\xi_i^- = 0$, replacing $\alpha_i^{+*}$ with $\alpha_i^{+*}$)
- So now we can solve for $b^*$ using $\alpha^{+*}$:
  $b^* = y_i - \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle - \epsilon - \xi_i^+$
  $= y_i - \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle - \epsilon - 0$
  (**This is the answer for how to recover $b^*$.**)
  $b^* = y_i - \sum_{j=1}^{n} (\alpha_j^{+*} - \alpha_j^{-*}) \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle - \epsilon$
  This same thing holds when using $\alpha^{-*}$ to solve for $b^*$.
- To kernelize the SVR equation, we simply need to replace inner products with a kernel function. The SVR equation will be:
  $f(x) = \langle \boldsymbol{w}^*, \boldsymbol{x} \rangle + b^*$
  $= \sum_{j=1}^{n} (\alpha_i^{+*} - \alpha_i^{-*}) \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle + b^*$
  $f(x) = \sum_{j=1}^{n} (\alpha_i^{+*} - \alpha_i^{-*}) k(\boldsymbol{x}_j, \boldsymbol{x}_i) + b^*$
  for some $j$ with $0 < \alpha_j^{+*} < \frac{C}{n}$ and $0 < \alpha_j^{-*} < \frac{C}{n}$ and $b^*$ as defined above

(d) (3 pts) Argue that the final predictor will only depend on a subset of training examples (i.e. support vectors) and characterize those training examples.

By examining KKT conditions, we can see that the Lagrange multipliers ($\alpha^+$ or $\alpha^-$) may be non-zero only when $|f(\boldsymbol{x}_i) - y_i| \geq \epsilon$. So for all samples within $\epsilon$ of $f(\boldsymbol{x})$ the values of $\alpha^+$ and $\alpha^-$ will be 0. Stated mathematically, when we have $|f(\boldsymbol{x}_i) - y_i| < \epsilon$ either $\xi_i^+$ or $\xi_i^-$ is non-zero, implying that $\alpha^+$ and $\alpha^-$ both have to be zero in order for the KKT conditions to be satisfied. So in order to describe $\boldsymbol{w}$ we don't need all values of $\boldsymbol{x}_i$, only those for which either $\alpha^+$ or $\alpha^-$ is non-zero (meaning these values of $\boldsymbol{x}_i$ are outside of the $\epsilon$ area around $f(x)$ and are being penalized). This is why the loss is referred to as $\epsilon$-insensitive: we don't care about (or "use") observations within $\epsilon$ of $f(x)$. Only those observations that are at least $\epsilon$ away from $f(x)$ are used to construct the SVR equation. In contrast, the SVC equation only cares about support vectors that are within $\epsilon$ of $f(x)$ in order to describe $\boldsymbol{w}$.

3. **Bagging and Logistic Regression (25 pts).** 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 methods for logistic regression) and compare to other classifiers. Train using the first 2000 samples and test your classifier on the last 1000 samples.

   (a) (20 pts) 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 10 bootstrapped samples, 5) bagging+logistic regression with 50 bootstrapped samples, and 6) 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).

   **TEST ERRORS**

   i. Random forests: used the *randomForest* package in R, parameters ntree = 500, mtry = sqrt(785) $\approx$ 28, nodesize = 1. Achieved test error of **0.017**.
   ii. SVM w/out bagging: used *e1071* and *caret* packages in R. The best test error for the linear kernel SVM was **0.049**, achieved with C = 0.01. For the Gaussian kernel SVM, the best test error of **0.018** was achieved with C = 100 and $\gamma = 0.01$.
   iii. Logistic regression w/out bagging: The test error achieved was **0.152**.
   iv. Bagging + logistic regression w/51 bootstrapped samples: The test error achieved was **0.158**.
   v. Bagging + logistic regression w/101 bootstrapped samples: The test error achieved was **0.158**.

   The best classifier is essentially a tie between the random forests model and the SVM with a Gaussian kernel. The difference between their test error rates (0.017 and 0.018 respectively) is negligible, so I'd say they are co-champions.
   It does not appear that bagging helps logistic regression at all. In fact, it gives a (very) slightly worse test error, as the un-bagged model has a test error of 0.152 and the bagged models both have test error of 0.158. I'm guessing that this is because logistic regression is already a fairly low-variance learner, so aggregating results from multiple models trained on bootstrapped samples won't help results.

   **IMPORTANT NOTE**: I think that my bagged logistic regression results (and their lack of improvement) have to do with the way that I constructed my bagged classifiers (*bagging votes VS bagging probabilities*). Here was the process:

   - Take B (B $\in \{51, 101\}$) bootstrapped samples of size 2000 (samples made w/replacement).
   - Train a logistic regression model on each of the B samples. Your final predictor is the aggregation (via majority vote) of these B models.
   - Then, to get the test error, fit each of these B models onto the test data, generating B predictions for each of the 1000 test data points.
   - Next, take the majority vote for each of the 1000 observations over the B models. For example, say that observation 400 in the test data has 25/51 predictions as 1's and 26/51 predictions as -1's: then the final predicted value for observation 400 is -1. In practice I just summed the B outputs for a single observation. For the observation-400 example we'd have: $(1)(25) + (-1)(26) = -1$. If the sum of the B outputs is negative then the final prediction is -1, and if the sum of the B outputs is positive then the final prediction is 1.
   - In retrospect, I think that averaging the B probabilities to get a final probability and **THEN** using a cutoff of 0.5 to give the final prediction may have given better results. But since the problem doesn't specify this and I have code that already runs (and that took awhile to write and had to run overnight) I have to leave it as-is to get to other problems and homework :(

(b) (5 pts) 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 five 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 with one of these methods 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*.

**TEST ERRORS**

i. Random forests: **0.477**
ii. SVM w/out bagging: **0.517** (linear kernel), **0.478** (Gaussian kernel)
iii. Logistic regression w/out bagging: **0.391**
iv. Bagging + logistic regression w/51 bootstrapped samples: **0.407**
v. Bagging + logistic regression w/101 bootstrapped samples: **0.407**

The logistic regression classifiers perform the best in this case, with the single-model regression slightly outperforming the bagged models. Once, again, bagging does nothing to help the logistic regression models, slightly harming their performance.

If we know that our test data are shifted relative to our training data, we could utilize a combination of bagging with shifted base learners to get a final model. Analogous to the average shifted histograms discussed in the notes, we could randomly shift each bootstrapped sample (by applying some transformation that perturbs the bootstrap sample and results in a shifted training data set). Then, using majority voting (bagging) with all of these base learners we'd hopefully get a better result than what we have here.
To elaborate on the shifting transformation: we're shifting the pixel values in our dataframe, moving all pixel values in the same manner (same change in the horizontal and vertical position for all pixels). For example: have a function that randomly chooses a horizontal shift value $\in \{-5, -4, \ldots, 4, 5\}$ and also a randomly chosen vertical shift value $\in \{-5, -4, \ldots, 4, 5\}$. Then, simply move all pixel values according to this shift. Some pixels would "fall off the edge", and our result would have other edges without any values. I'm guessing that the easiest thing for these missing edge values would be to just assign these blank pixels as having a value of being "white" (if the pixels comprising the actual number are "black"). Or we could employ a more involved technique like imputation, taking into account the pixels around it.
Another thought I had is that (ideally) all of the images are shifted in precisely the same manner. If this were the case, what we'd have to do is find the specific shifting transformation that "makes the training data like the test data." However, this would likely be a more difficult task to do in practice than using a combination of random shifting combined with bagging as discussed above. Trying to find the exact shift transformation would be impractical because: (1.) it's very possible that the images aren't all shifted in the same way, and (2.) trying to find the correct shift transformation would likely take longer than just using random shifting + bagging.

**MY CODE FOR THIS PROBLEM IS BELOW**

```
#================================================================
#==== CODE FOR STATISTICAL LEARNING II, hw5 PROBLEM 3 ===========
#================================================================

# Necessary libraries for the script
library(dplyr)
library(R.matlab)
library(geometry)
library(pracma)
library(randomForest)
library(caret)
library(magrittr)
library("e1071")

# Change the working directory so that the image file can be read in.
# The line below needs to be changed if you want to run the code.
setwd("C:/Users/jrdha/OneDrive/Desktop/USU_Fa2018/Moon__SLDM2/hw5/problem3")

# Read in the UN-SHIFTED MNIST data as a dataframe with
# response "Y" and predictors "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())

# Split the data into a training set (first 2000 obs).
# Add a column of leading 1s for correct dimensions/calculations when dealing
# with b in theta.
set.seed(2345)
trainData <- dplyr::slice(df, 1:2000)
trainPredictors <- select(trainData, -Y)
X_0 <- rep(1, 2000)
trainPredictors <- cbind(X_0, trainPredictors)
trainResponse <- select(trainData, Y)
trainData$Y <- as.factor(trainData$Y)

# Split the data into a test set too (last 1000 obs).
# Add a column of leading 1s for correct dimensions/calculations when dealing
# with b in theta.
set.seed(2345)
testData <- dplyr::slice(df, 2001:3000)
testPredictors <- select(testData, -Y)
X_0 <- rep(1000)
testPredictors <- cbind(X_0, testPredictors)
testResponse <- select(testData, Y)
testData$Y <- as.factor(testData$Y)

# Read in the SHIFTED MNIST data as a dataframe with
# response "Y" and predictors "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]])))
```

```r
colnames(s_df[[2]]) <- c("Y")
s_df <- bind_cols(s_df) %>% select(Y, everything())



#=============================================================================
#==== RANDOM FORESTS ========================================================
#=============================================================================
set.seed(2345)

# Fitting the initial model (left default function arguments)
# ntree = 500
# mtry = sqrt(785) = 28.01785
# nodesize = 1
rf_model <- randomForest(Y ~., data = trainData)

# Predicting onto the UN-SHIFTED test data
rf_prediction <- predict(rf_model, newdata = testData)

#===== Return the UN-SHIFTED test error FOR RANDOM FORESTS: 0.017 ==============
1 - mean(rf_prediction == testData$Y)

# Predicting onto the SHIFTED test data
rf_prediction <- predict(rf_model, newdata = s_df)

#===== Return the SHIFTED test error for RANDOM FORESTS: 0.477 =================
1 - mean(rf_prediction == testData$Y)



#=============================================================================
#===== SVM ==================================================================
#=============================================================================

# Tuning LINEAR KERNEL: Round 1 (used this to get a set of values that did well)
# Ran it by changing cost = 1.digit*10^(-3:3) where I set digit = 0,1,2,....,9
set.seed(2345)
svm_tune_linear <- tune.svm(Y ~.,
                            data = trainData,
                            kernel = "linear",
                            cost = 1.0*10^(-3:2))
print(svm_tune_linear)

# FIT FINAL SVM MODEL, USING THE 10-FOLD CV cost=0.01 FOR LINEAR KERNEL
svm_model <- svm(Y ~., trainData, kernel = "linear", cost = 0.01)
testData$pred <- predict(svm_model, testData)
testData$correct <- with(testData, ifelse(testData$Y == pred, 1, 0))
# Generate the test error
# Total number of correct classifications
totalCorrect <- sum(testData$correct)
testError_SVM <- 1 - (totalCorrect / nrow(testData))
```

```r
testError_SVM
# Remove the "correct" column for the SVM portion
testData <- subset(testData, select =  -c(correct, pred))
#========== RESULTS: SVM, LINEAR-KERNEL ====================================
#========== UN-SHIFTED Test error rate of 0.049 ============================

# FIT FINAL SVM MODEL, USING THE 10-FOLD CV cost=0.01 FOR LINEAR KERNEL
s_df$pred <- predict(svm_model, s_df)
s_df$correct <- with(s_df, ifelse(s_df$Y == pred, 1, 0))
# Generate the test error
# Total number of correct classifications
totalCorrect <- sum(s_df$correct)
testError_SVM <- 1 - (totalCorrect / nrow(s_df))
testError_SVM
# Remove the "correct" column for the SVM portion
s_df <- subset(s_df, select =  -c(correct, pred))
#========== RESULTS: SVM, LINEAR-KERNEL ====================================
#========== SHIFTED Test error rate of 0.517 ===============================

#========== TUNING GAUSSIAN-KERNEL MODEL ====================================
# INITIAL PASS (leave base values at 10)
set.seed(2345)
svm_tune_gaussian <- tune.svm(Y~., data = trainData,
                              cost = 10^(-3:2),
                              gamma = 10^(-5:2))
print(svm_tune_gaussian)
# Parameter tuning of 'svm':
# - sampling method: 10-fold cross validation
# - best parameters:
#   gamma  cost
#    0.01  100
# - best performance: 0.0185

# THIS FITS A FINAL SVM MODEL, USING THE CROSSVALIDATED TUNING PARAMETERs
# of C = 100, and gamma = 0.01
costVal = 100
gammaVal = 0.01
svm_model <- svm(Y ~., trainData, kernel = "radial",
                 cost = costVal, gamma = gammaVal)
testData$pred <- predict(svm_model, testData)
testData$correct <- with(testData, ifelse(testData$Y == pred, 1, 0))
# Generate the test error
# Total number of correct classifications
totalCorrect <- sum(testData$correct)
testError_SVM <- 1 - (totalCorrect / nrow(testData))
testError_SVM
# Remove the "correct" column for the SVM portion
testData <- subset(testData, select =  -c(correct, pred))
#========= RESULTS: SVM, GAUSSIAN-KERNEL ====================================
#========= Test error UN-SHIFTED: 0.018 =====================================

s_df$pred <- predict(svm_model, s_df)
```

```r
s_df$correct <- with(s_df, ifelse(s_df$Y == pred, 1, 0))
# Generate the test error
# Total number of correct classifications
totalCorrect <- sum(s_df$correct)
testError_SVM <- 1 - (totalCorrect / nrow(s_df))
testError_SVM
# Remove the "correct" column for the SVM portion
s_df <- subset(s_df, select =  -c(correct, pred))
#========= RESULTS: SVM, GAUSSIAN-KERNEL ======================================
#========= Test error SHIFTED: 0.478 ==========================================




# Re-run this just to be sure we're working with the right data before doing
# logistic regression
set.seed(2345)
trainData <- dplyr::slice(df, 1:2000)
trainPredictors <- select(trainData, -Y)
X_0 <- rep(1, 2000)
trainPredictors <- cbind(X_0, trainPredictors)
trainResponse <- select(trainData, Y)
trainData$Y <- as.factor(trainData$Y)

# Split the data into a test set too (last 1000 obs).
# Add a column of leading 1s for correct dimensions/calculations when dealing
# with b in theta.
set.seed(2345)
testData <- dplyr::slice(df, 2001:3000)
testPredictors <- select(testData, -Y)
X_0 <- rep(1000)
testPredictors <- cbind(X_0, testPredictors)
testResponse <- select(testData, Y)
testData$Y <- as.factor(testData$Y)




#==============================================================================
#===== LOGISTIC REGRESSION W/OUT BAGGING ======================================
#==============================================================================

# Fit logistic regression model
logRegr_model <- glm(Y ~ . ,
                     data = trainData,
                     family = binomial
                     )

# Generates the predictions for our UN-SHIFTED testData, cutoff of 0.5
testData$pred <- predict(logRegr_model, newdata = testData, type = "response")
testData$pred <- ifelse(testData$pred > 0.5, "1", "-1")
```

```r
# Generates a column that says whether or not a test observation was correctly
# classified (1==correct, 0==incorrect)
testData$correct <- with(testData, ifelse(Y == pred, 1, 0))

# Generate the test error
totalCorrect <- sum(testData$correct)
testError_logReg <- 1 - (totalCorrect / nrow(testData))
testError_logReg
#============= RESULTS: LOGISTIC REGRESSION, 1 model ============================
#============= UN-SHIFTED Test error of 0.152 ====================================

# Remove the "correct" and "pred" columns so that we'll have the original data
# to work with for the SVM portion
testData <- subset(testData, select =  -c(correct, pred))

# Generates the predictions for our SHIFTED s_df, cutoff of 0.5
s_df$pred <- predict(logRegr_model, newdata = s_df, type = "response")
s_df$pred <- ifelse(s_df$pred > 0.5, "1", "-1")

# Generates a column that says whether or not a test observation was correctly
# classified (1==correct, 0==incorrect)
s_df$correct <- with(s_df, ifelse(Y == pred, 1, 0))

# Generate the test error
totalCorrect <- sum(s_df$correct)
testError_logReg <- 1 - (totalCorrect / nrow(s_df))
testError_logReg
#============= RESULTS: LOGISTIC REGRESSION, 1 model ============================
#============= SHIFTED Test error of 0.391 =======================================

# Remove the "correct" and "pred" columns so that we'll have the original data
# to work with for the SVM portion
s_df <- subset(s_df, select =  -c(correct, pred))



#================================================================================
#===== LOGISTIC REGRESSION W/BAGGING, 51 SAMPLES ================================
#================================================================================

# First, generate 51 bootstrapped samples
set.seed(2345)

bootModel <- function(trainData, testData, i){

  loop_seed = 1000+i
  set.seed(loop_seed)

  lengthTrain <- nrow(trainData)

  # Randomly sample indices from 1 - 2000 (observations from training data)
```

```r
  btstrp_ind <- sample(seq_len(lengthTrain), size = lengthTrain)

  # Make the observations at these indices your new training data
  bt_train <- trainData[btstrp_ind, ]

  # Fit logistic regression model
  logRegr_model <- glm(Y ~ . ,
                       data = bt_train,
                       family = binomial
  )

  # Generates the predictions for our testData, cutoff of 0.5
  testData$pred <- predict(logRegr_model, newdata = testData, type = "response")
  testData$pred <- ifelse(testData$pred > 0.5, "1", "-1")

  # Generates a column that says whether or not a test observation was correctly
  # classified (1==correct, 0==incorrect)
  testData$correct <- with(testData, ifelse(Y == pred, 1, 0))

  # Generate the test error
  totalCorrect <- sum(testData$correct)
  testError_logReg <- 1 - (totalCorrect / nrow(testData))
  testError_logReg

  # Store the predictions in here before removing that column from testData
  predictions <- testData$pred

  # Remove the "correct" and "pred" columns so that we'll have the original data
  # to work with for the SVM portion
  testData <- subset(testData, select =  -c(correct, pred))

  return(predictions)
}

# Initialize empty dataframe of predictions
all_51_preds <- data.frame(matrix(NA, nrow = 1000, ncol = 51))

# Generate the predictions from the 51 models fitted on bootstrapped data
for (i in 1:51){
  preds <- bootModel(trainData, testData, i)
  all_51_preds[ ,i] <- preds
}

# Make sure we're getting some different predictions
# all_51_preds[,1] == all_51_preds[,3]

# We'll use these two functions to make the data be numeric and not character
asNumeric <- function(x){
  as.numeric(as.character(x))
}
factorsNumeric <- function(d){
```

```r
  modifyList(d, lapply(d[, sapply(d, is.character)], asNumeric))
}

# Used this during de-bugging to save on computation time
# all_51_preds <- all_51_preds[, 1:3]

# Make the whole dataframe numeric so we can sum the rows (easier to get vote)
all_51_preds <- factorsNumeric(all_51_preds)

# Sum each of the rows, adding a new column that gives this sum
all_51_preds$sum <- rowSums(all_51_preds)

# If the sum of a row is negative then the majority vote is for -1, if the sum
# of the row is positive then the majority vote is for 1
all_51_preds$vote <- ifelse(all_51_preds$sum < 0, -1, 1)

# Add on the true labels to the dataframe
all_51_preds$truth <- testData$Y

# Finally, make a column that says whether or not the majority vote was correct
all_51_preds$correct <- ifelse(all_51_preds$vote == all_51_preds$truth, 1, 0)

# Generate test error
# Total number of correct classifications
totalCorrect <- sum(all_51_preds$correct)
testError_LR_51 <- 1 - (totalCorrect / nrow(testData))
testError_LR_51
#============= RESULTS: LOGISTIC REGRESSION, 51 models VOTING ==================
#============= UN-SHIFTED Test error of 0.158 ===================================

#==== NOW FOR THE SHIFTED DATA, USING THE SAME 51 PREDICTION MODELS ============
#================================================================================
# Initialize empty dataframe of predictions
all_51_shift <- data.frame(matrix(NA, nrow = 1000, ncol = 51))


# Generate the predictions from the 51 models fitted on bootstrapped data
for (i in 1:51){
  preds <- bootModel(trainData, s_df, i)
  all_51_shift[ ,i] <- preds
}

# Make the whole dataframe numeric so we can sum the rows (easier to get vote)
all_51_shift <- factorsNumeric(all_51_shift)

# Sum each of the rows, adding a new column that gives this sum
all_51_shift$sum <- rowSums(all_51_shift)

# If the sum of a row is negative then the majority vote is for -1, if the sum
# of the row is positive then the majority vote is for 1
all_51_shift$vote <- ifelse(all_51_shift$sum < 0, -1, 1)
```

```r
# Add on the true labels to the dataframe
all_51_shift$truth <- s_df$Y

# Finally, make a column that says whether or not the majority vote was correct
all_51_shift$correct <- ifelse(all_51_shift$vote == all_51_shift$truth, 1, 0)

# Generate test error
# Total number of correct classifications
totalCorrect <- sum(all_51_shift$correct)
testError_LR_51_SHIFT <- 1 - (totalCorrect / nrow(s_df))
testError_LR_51_SHIFT
#============= RESULTS: LOGISTIC REGRESSION, 51 models VOTING ==================
#============= SHIFTED Test error of 0.407 ====================================




#=============================================================================
#===== LOGISTIC REGRESSION W/BAGGING, 101 SAMPLES ============================
#=============================================================================

# First, generate 101 bootstrapped samples
set.seed(2345)

# Initialize empty dataframe of predictions
all_101_preds <- data.frame(matrix(NA, nrow = 1000, ncol = 101))

# Generate the predictions from the 101 models fitted on bootstrapped data
for (i in 1:101){
  preds <- bootModel(trainData, testData, i)
  all_101_preds[ ,i] <- preds
}

# Make the whole dataframe numeric so we can sum the rows (easier to get vote)
all_101_preds <- factorsNumeric(all_101_preds)

# Sum each of the rows, adding a new column that gives this sum
all_101_preds$sum <- rowSums(all_101_preds)

# If the sum of a row is negative then the majority vote is for -1, if the sum
# of the row is positive then the majority vote is for 1
all_101_preds$vote <- ifelse(all_101_preds$sum < 0, -1, 1)

# Add on the true labels to the dataframe
all_101_preds$truth <- testData$Y

# Finally, make a column that says whether or not the majority vote was correct
all_101_preds$correct <- ifelse(all_101_preds$vote == all_101_preds$truth, 1, 0)

# Generate test error
# Total number of correct classifications
```

```r
totalCorrect <- sum(all_101_preds$correct)
testError_LR_101 <- 1 - (totalCorrect / nrow(testData))
testError_LR_101
#============= RESULTS: LOGISTIC REGRESSION, 101 models VOTING ================
#============= UN-SHIFTED Test error of 0.158 ==================================

#==== NOW FOR THE SHIFTED DATA, USING THE SAME 101 PREDICTION MODELS ===========
#==============================================================================
# Initialize empty dataframe of predictions
all_101_shift <- data.frame(matrix(NA, nrow = 1000, ncol = 101))

# Generate the predictions from the 101 models fitted on bootstrapped data
for (i in 1:101){
  preds <- bootModel(trainData, s_df, i)
  all_101_shift[ ,i] <- preds
}

# Make the whole dataframe numeric so we can sum the rows (easier to get vote)
all_101_shift <- factorsNumeric(all_101_shift)

# Sum each of the rows, adding a new column that gives this sum
all_101_shift$sum <- rowSums(all_101_shift)

# If the sum of a row is negative then the majority vote is for -1, if the sum
# of the row is positive then the majority vote is for 1
all_101_shift$vote <- ifelse(all_101_shift$sum < 0, -1, 1)

# Add on the true labels to the dataframe
all_101_shift$truth <- s_df$Y

# Finally, make a column that says whether or not the majority vote was correct
all_101_shift$correct <- ifelse(all_101_shift$vote == all_101_shift$truth, 1, 0)

# Generate test error
# Total number of correct classifications
totalCorrect <- sum(all_101_shift$correct)
testError_LR_101_SHIFT <- 1 - (totalCorrect / nrow(s_df))
testError_LR_101_SHIFT
#============= RESULTS: LOGISTIC REGRESSION, 101 models VOTING ================
#============= SHIFTED Test error of 0.407 =====================================
```

4. **Outlier Detection with Kernel Density Estimation (35 pts).** 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) (5 pts) Using least squares leave-one-out cross-validation with a Gaussian kernel and the training data, select a value for the bandwidth parameter. Report the bandwidth parameter.

   The bandwith parameter my code came up with was h = 0.08 (this is different than the value of h = 0.2 you gave during office hours, but I can't find an error in my code).

   (b) (7 pts) 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). Do not use packages that automatically compute the KDE but use the equations instead. If in doubt, you can ask me. 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?

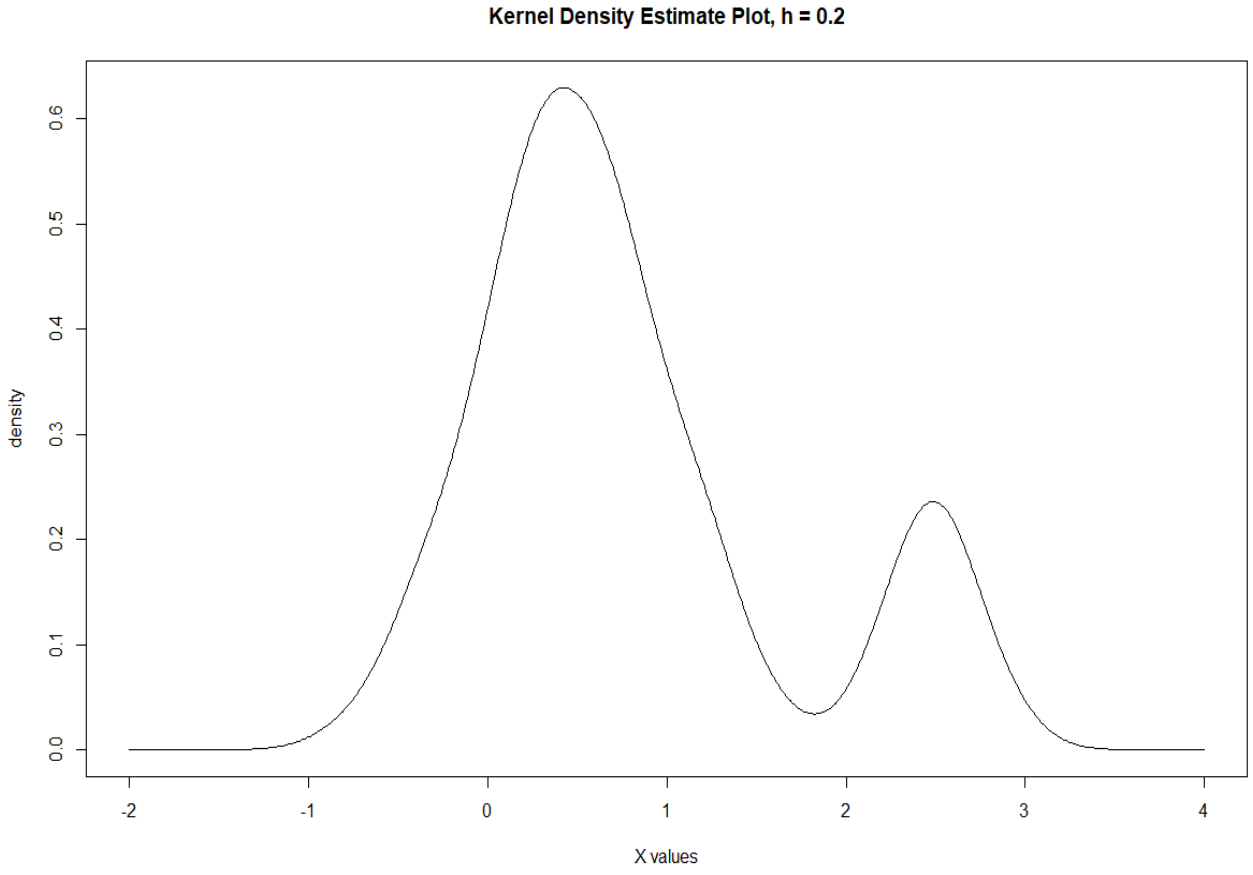


Figure 1: KDE with bandwidth parameter h = 0.2 (recommended during office hours)

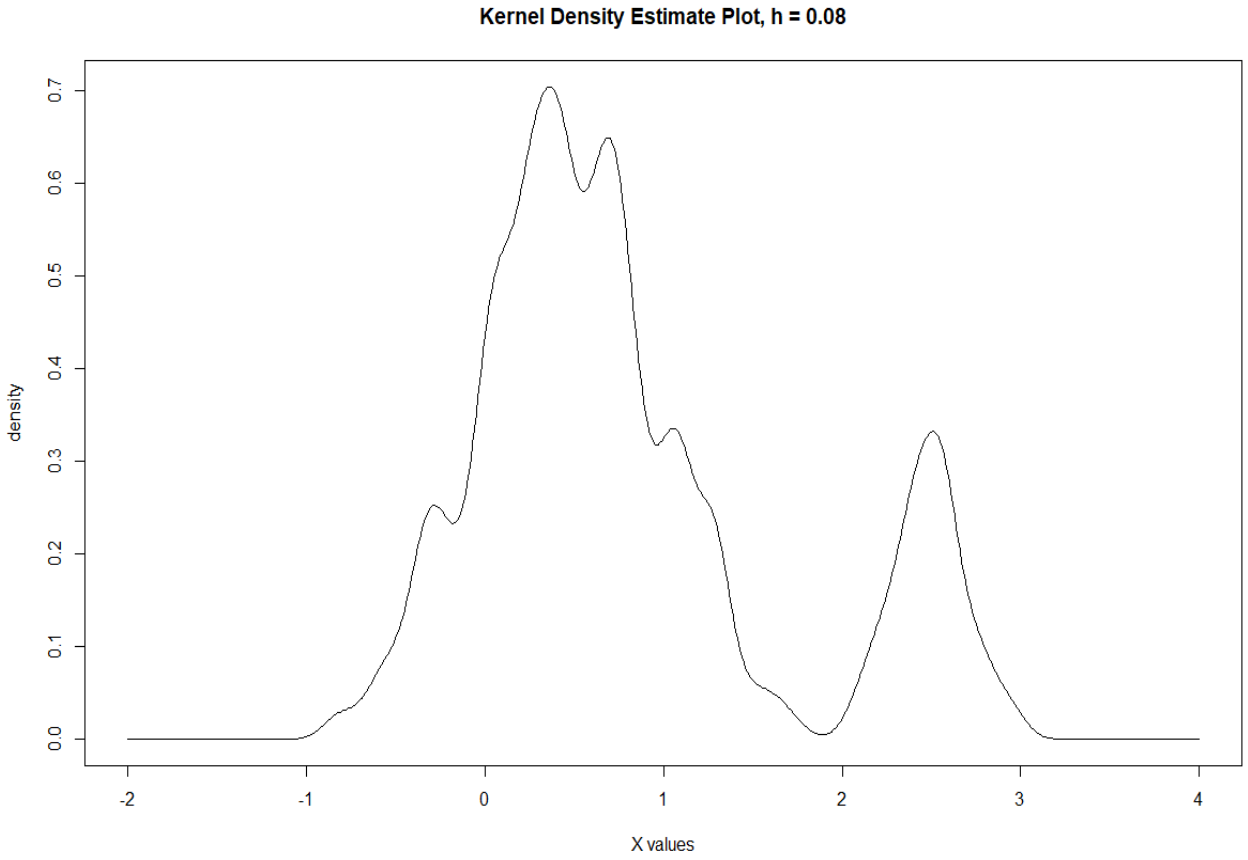


Figure 2: KDE with bandwidth parameter h = 0.08 (obtained in part (a))

I included KDE plots for both h = 0.2 and h = 0.08. The h = 0.2 comes from your recommendation during office hours, but my code never returned this, and I couldn't find a bug or logic error. (I also included it because it looks nicer). The KDE using h = 0.08 is given because this was the estimated bandwidth from my code.

Based on the KDE (moreso on the h = 0.2 KDE plot), it looks like the true density of $f$ is bimodal, with both of these modes looking to be somewhat Gaussian. Potentially, the true density of $f$ is simply concatenated data from two separate Gaussian distributions. If I had to guess, I'd say that the larger distribution (on the left) is something like $Normal(\mu = 0.5, \sigma^2 = 0.5^2)$ and the smaller distribution (on the right) is something like $Normal(\mu = 2.5, \sigma^2 = 0.1666^2)$.

(c) (4 pts) 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 $x_j$, $j = 1, \ldots, N$ be the training data. Let $x$ be a point that you wish to test. Define an outlier score for $x$ as

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

See lecture slides for a definition of $\hat{f}_h^{(-j)}(x_j)$. 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?

The denominator of $OutlierScore_1$ is the average density as given by the LOO (leave-one-out) density estimator. The numerator is the KDE determined by the bandwidth parameter h. If the density at a given $x$ (numerator) is similar to the average density (denominator), we'll obtain an $OutlierScore_1$ close to 1. If $OutlierScore_1$ is greater than 1, we can look at this $x$ as being in a more dense part of $f$. If $OutlierScore_1$ is less than 1 (especially if it's very small), we can look at this $x$ as being in a less dense part of $f$, and the lower the $OutlierScore_1$, the more of an outlier that $x$ is.
A potential weakness of this score is that it wouldn't give good performance for data that are very densely concentrated and have long tails. It would determine that $x$ that isn't very close to the center is an outlier. Many points that aren't actually outliers might be classified as outliers depending on how "much less than 1" is defined for $OutlierScore_1$.
Another potential weakness is actually for data that have large variability. In this case, the estimated density at some $x$ (numerator of $OutlierScore_1$) is fairly likely to be close to the average density (denominator), so it will have a hard time detecting outliers. Again, this will depend on how "much less than 1" are defined for $OutlierScore_1$.
In summary, the weakness of $OutlierScore_1$ is that it doesn't adapt to the inherent variability of $f$ and the KDE.

(d) (5 pts) 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?

- For `xtest1` $OutlierScore_1 = 0.1783$
- For `xtest2` $OutlierScore_1 = 1.029214 \times 10^{-16}$

It's hard to say whether `xtest1` is an outlier, since 0.1783 seems much smaller than 1. However, seeing that `xtest2` has such a ridiculously small $OutlierScore_1$ that is far smaller than 1, we can confidently say that it's an outlier. But without more information about the relative variability of $f$ and how we should characterize that variability, I don't know whether or not we'd call `xtest1` an outlier. I'd lean toward not calling `xtest1` an outlier.

(e) (4 pts) $OutlierScore_1$ has some weaknesses. Let's define another score based on $k$-nearest neighbors. Let $\rho_k(x)$ be the distance of $x$ to its $k$th 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 $k$th 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)?

The denominator of $OutlierScore_2$ is the average distance from $x$ to its k nearest neighbors. The numerator is the distance of $x$ to its kth nearest neighbor in the training data (given in prompt). This ratio gives us some sense of whether or not $x$ is similar to its k nearest neighbors. If $OutlierScore_2$ is large, this means that $x$ is far from its $k^{th}$ nearest neighbor relative to how close it is (on average) to all k of its nearest neighbors. This would mean that the KNN points around $x$ are fairly similar to $x$ and it likely isn't an outlier.
On the other hand, if $OutlierScore_2$ is small, this essentially means that the KNN points for $x$ are not very similar to it (where similarity is characterized by distances). A small value of $OutlierScore_2$ means that the $k^{th}$ nearest neighbor of $x$ is similar to all the other (k-1) neighbors of $x$, which likely means that $x$ isn't really similar to any of them, and is thus an outlier.

(f) (5 pts) Calculate $OutlierScore_2$ for `xtest1` and `xtest2` using $k = 100$, 150, and 200. Based on the calculated scores, do you think either of these points are outliers?

- For `xtest1` with $k = 100$ $OutlierScore_2 = 2.84671$
- For `xtest1` with $k = 150$ $OutlierScore_2 = 2.00772$
- For `xtest1` with $k = 200$ $OutlierScore_2 = 1.757143$
- For `xtest2` with $k = 100$ $OutlierScore_2 = 1.212911$
- For `xtest2` with $k = 150$ $OutlierScore_2 = 1.228357$
- For `xtest2` with $k = 200$ $OutlierScore_2 = 1.235166$

It's hard to say whether either of these is an outlier. For $OutlierScore_1$ we knew that anything greater than 1 was very unlikely to be an outlier. However, in this case both are greater than 1, and we have an outlier scoring estimate that is adaptive to variability, so looking at things relative to their proximity with 1 is useless. At least relative to each other, we'd certainly say that `xtest2` is much more of an outlier than `xtest1`.

(g) (5 pts) Include your code.

**See below**

```r
#==============================================================================
#==== CODE FOR STATISTICAL LEARNING II, hw5 PROBLEM 3 =========================
#==============================================================================



# Necessary libraries
library(geometry)
library(tidyr)
library(dplyr)

# Set the working directory
setwd("C:/Users/jrdha/OneDrive/Desktop/USU_Fa2018/Moon__SLDM2/hw5/problem4")

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

# Store the given test points 1 and 2
testPt_1 <- given_data$xtest1$V1
testPt_2 <- given_data$xtest2$V1




#==============================================================================
#==== ALL FUNCTIONS ===========================================================
#==============================================================================


# This function returns (calculating first) the Gaussian kernel for the vector
# of values passed in.
# Takes as arguments: vector (self-explanatory), h is the bandwidth parameter.
g_Kernel <- function(vector, h){
  vec_len <- length(vector)
  vector.h <- vector/h
  if(vec_len > 1){
    kernel_val <- ((2*pi)^(-vec_len/2))*exp(-0.5*(dot(vector.h, vector.h)))
  } else {
    kernel_val <- ((2*pi)^(-vec_len/2))*exp(-0.5*(vector.h * vector.h))
  }
  return((h^(-vec_len)) * kernel_val)
}
```

```r
# This function calculates (and returns) the value of the objective function for
# the data that is passed in via LS-LOOCV using a given bandwidth.
# Takes as arguments: given_data (self-explanatory), h is the bandwidth param.
est_BandWidth <- function(given_data, h){
  data_len <- nrow(given_data)
  t1 <- 0
  t2 <- 0
  ind <- given_data$index
  X <- given_data$X

  # Loop through all data
  for(i in 1:data_len){
    for(j in 1:data_len){
      newTerm1 <- g_Kernel(vector = (X[i] - X[j]), h = (sqrt(2)*h))
      t1 <- t1 + newTerm1
      if(j != i){
        newTerm2 <- g_Kernel(vector = (X[i] - X[j]), h = h)
        t2 <- t2 + newTerm2
      }
    }
  }
  t1 <- (1/(data_len^2)) * t1
  t2 <- (2/(data_len*(data_len-1))) * t2
  final_val <- t1 - t2
  return(final_val)
}


# This function returns (after calculating) the KDE for all points in a given
# range. It calls the KDE_one_pt function for each of these points.
# Takes as arguments: range (range of values for which KDE is calculated),
# X (the training dataframe), h (bandwidth parameter).
KDE_all_pts <- function(range, X, h){

  density_estimate <- data.frame(x = range, density = double(length(range)))
  for(i in 1:length(range)){
    density_estimate$density[i] <- KDE_one_pt(point = range[i], X = X, h = h)
  }
  return(density_estimate)
}
```

```r
# This function returns (after calculating) the KDE for a single point.
# Takes as arguments: point (the point for which KDE is being estimated),
# X (training dataframe), h (bandwidth parameter).
KDE_one_pt <- function(point, X, h){
  term <- 0
  len_data <- length(X)
  for(i in 1:len_data){
    val <- g_Kernel(vector = (X[i] - point), h = h)
    term <- term + val
  }
  f_hat <- (1/len_data) * term
  return(f_hat)
}


# This function returns (after calculating) the LOO (leave one out) KDE, and is
# used by the outlierScore_1 function.
# Takes as arguments: point (point on which we're calculating KDE), X (training
# dataframe), exclude.i (index indicating which single point is to be excluded),
# h (bandwidth).
f_hat.i <- function(point, X, exclude.i, h){

  term <- 0
  n <- length(X)
  for(i in 1:n){
    if(i != exclude.i){
      val <- g_Kernel(vector = (point - X[i]), h = h)
      term <- term + val
    }
  }
  f_hat <- (1/(n-1)) * term
  return(f_hat)
}


# This function returns (after calculating) the OutlierScore_1 that is defined
# in part (c) of problem 4.
# Takes as arguments: point (calculating OutlierScore_1 for this point),
# X (training dataframe), h (bandwidth).
outlierScore_1 <- function(point, X, h){

  len_data <- length(X)
  numer_frac <- KDE_one_pt(point = point, X = X, h = h)
  denom_frac <- 0
  for(i in 1:len_data){
    term<- f_hat.i(point = X[i], X = X, exclude.i = i, h = h)
    denom_frac <- term + denom_frac
  }
  denom_frac <- (1/len_data) * denom_frac
  return(numer_frac/denom_frac)
}
```

```r
# This function returns (after calculating) an object, KNN, that contains the K
# nearest neighbors to a point, and the distances of each of these neighbors to
# that point.
# Takes as arguments: point (finding KNN and their distances for this point),
# X (training dataframe), k (the "k" in KNN: number of neighbors).
findKNN_distances <- function(point, X, k){

  pointvector <- rep(point, length(X))
  distance <- abs(X - pointvector)
  given_data <- data.frame(X = X, distance = distance)
  given_data <- arrange(given_data, distance)
  KNN <- slice(given_data, 1:k)
  return(KNN)
}


# This function returns (after calculating) the OutlierScore_2 that is defined
# in part (e) of problem 4.
# Takes as arguments: point (calculating OutlierScore_1 for this point),
# X (training dataframe), k (the "k" in KNN: number of neighbors).
outlierScore_2 <- function(point, X, k){

  KNN <- findKNN_distances(point, X, k)
  numer_frac <- KNN$distance[k]

  denom_frac <- (1/k) * sum(KNN$distance)

  return(numer_frac/denom_frac)
}




#============================================================================
#===== FUNCTION CALLS, CODE THAT GIVES ANSWERS TO QUESTIONS IN PROBLEM 4 =======
#============================================================================

# Define a grid of bandwidths to search through.
grid_list <- seq(.01,.5, by = .01)
len_grid <- length(grid_list)

# Using LS-LOOCV w/Gaussian kerenel calculate objective function for
# each value in grid_list
h_vals <- data.frame(index = 1:len_grid, h = grid_list, objF = rep(0, len_grid))
for(i in 1:nrow(h_vals)){
  objF <- est_BandWidth(given_data = trainData, h = h_vals$h[i])
  h_vals$index[i] <- i
  h_vals$objF[i] <- objF
}
```

```r
# Get the bandwith parameter h for which objective function is minimized (h_hat)
min_h <- h_vals[h_vals$objF == min(h_vals$objF),]
h_hat <- min_h$h
h_hat

# Define a sequence such that the KDE representation looks smooth (granularity).
density_range <- seq(-2, 4, by = 0.01)
# Estimate the density.
density_estimate <- KDE_all_pts(range = density_range,
                                X = trainData$X,
                                h = h_hat)
# Plot the estimated density obtained via KDE
plot(density_estimate$x,
     density_estimate$density,
     type = 'l',
     main = "Kernel Density Estimate Plot, h = 0.08",
     xlab = "X values",
     ylab = "density")

# OutlierScore_1 for xtest1 and xtest2 using optimal bandwith parameter h
os1_testPt_1 <- outlierScore_1(point = testPt_1, X = trainData$X, h = 0.08)
os1_testPt_2 <- outlierScore_1(point = testPt_2, X = trainData$X, h = 0.08)
os1_testPt_1
os1_testPt_2

# OutlierScore_2 for xtest1 with values of k specified in the prompt
os2_testPt_1.100 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 100)
os2_testPt_1.150 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 150)
os2_testPt_1.200 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 200)
os2_testPt_1.100
os2_testPt_1.150
os2_testPt_1.200

# OutlierScore_2 for xtest2, with values of k specified in the prompt
os2_testPt_2.100 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 100)
os2_testPt_2.150 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 150)
os2_testPt_2.200 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 200)
os2_testPt_2.100
os2_testPt_2.150
os2_testPt_2.200
```

5. How long did this assignment take you? (5 pts)

| Date | Times | | | Day Total |
|---|---|---|---|---|
| Tue, 11/19 | 4:30 pm – 8:15 pm | | | **3.75** HRS |
| Wed, 11/20 | 9:00 am – 12:45 pm | 6:00 pm – 10:00 pm | | **7.75** HRS |
| Thu, 11/21 | 2:15 am – 6:15 am | | | **4.0** HRS |
| Mon, 11/26 | 4:00 pm – 8:30 pm | | | **4.5** HRS |
| Tue, 11/27 | 8:15 am – 10:45 am | 5:15 pm – 10:15 pm | | **7.5** HRS |
| Wed, 11/28 | 8:30 am – 10:45 am | 4:30 pm – 11:00 pm | | **8.75** HRS |
| **TOTAL: 36.25 HOURS** | | | | |

6. Type up homework solutions (5 pts)