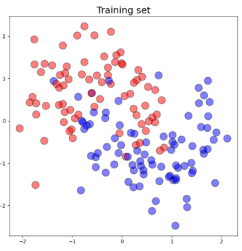
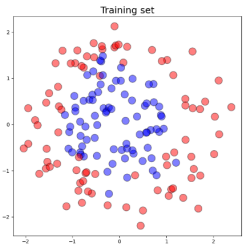
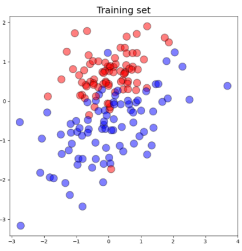
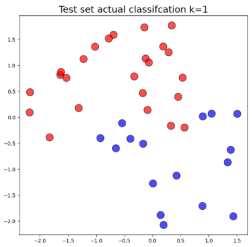
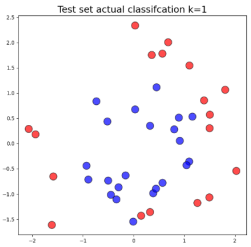
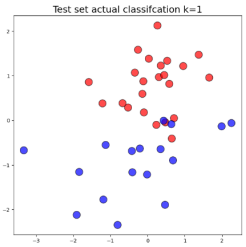
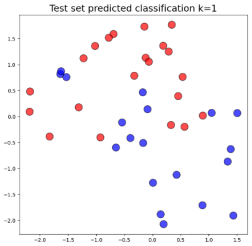
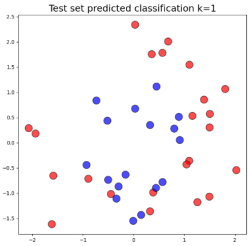
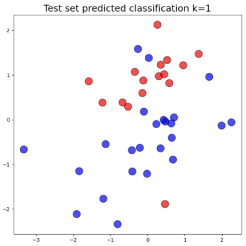
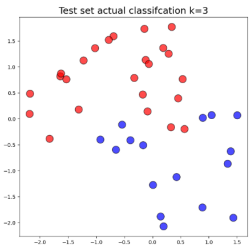
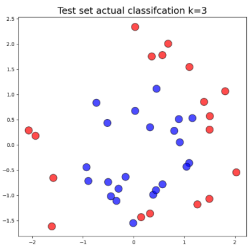
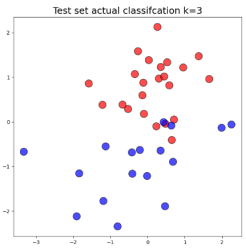
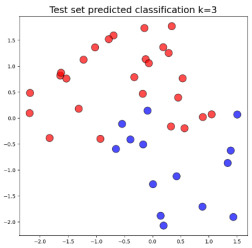
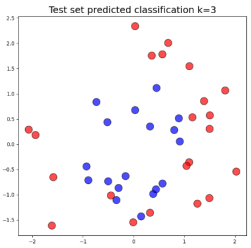
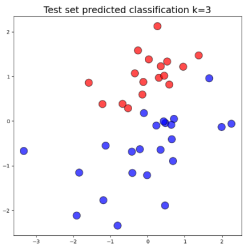
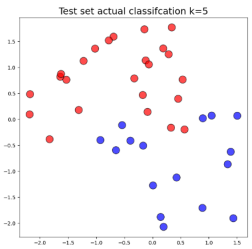
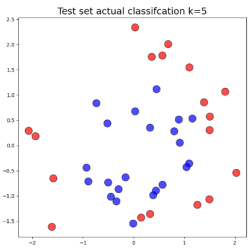
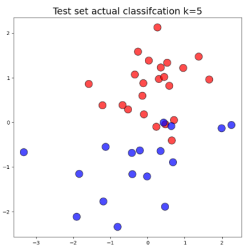
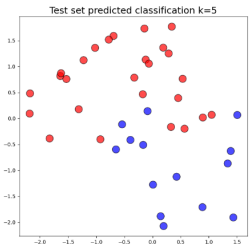
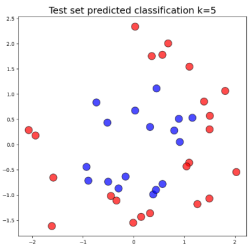
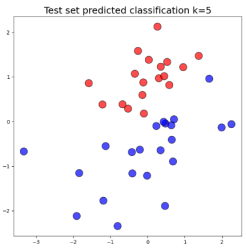
## KNNClassifier Code:

```
class KNNClassifier:
    def __init__(self,k):
        self.k = k

    def train(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def getNN(self, testEntry):
        distances = {}

        # get the distance between the test Entry and every training entry
        for trainEntryIdx in range(len(self.X_train)):
            dist = distance.euclidean(self.X_train[trainEntryIdx], testEntry)
            distances[trainEntryIdx] = dist

        #list of training entry index to that entry's distance to the test entry
        sorted_distance_map = list(sorted(distances.items(), key=lambda item: item[1], reverse=False))
        sorted_distance_map = sorted_distance_map[0:self.k]
        return [self.y_train[idx] for idx,distance in sorted_distance_map]

    def predict(self, X_test): # returns a vector of the predictions y_predict
        numTestEntries = len(X_test)
        #initialize y_predict
        y_predict = [0 for i in range(numTestEntries)]

        for testEntryIdx in range(numTestEntries):
            #get the values  of the K nearest neighbors
            nn = self.getNN(X_test[testEntryIdx])

            nnCounts = Counter(nn)
            val, count = nnCounts.most_common()[0]
            y_predict[testEntryIdx] = val
        return y_predict

    def score(self, X_test, y_test):
        y_predict = self.predict(X_test)
        meanAccuracy = 0
        for i in range(len(y_test)):
            if y_test[i] == y_predict[i]:
                meanAccuracy += 1
        meanAccuracy = meanAccuracy/len(y_test)
        return meanAccuracy
```

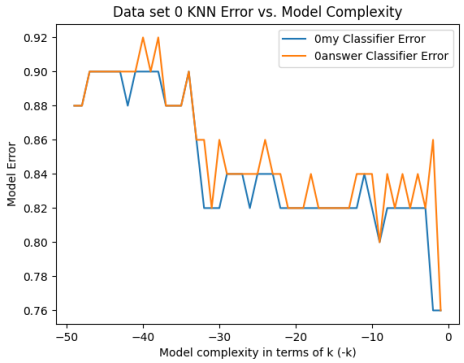
For the KNNRegression class, everything is the same, except instead of returning the majority class in the **predict** funtion, it returns the average of the y_train values of its nearest neighbors. Additionally, in the **score()** function, it uses the R^2 error metric instead of the mean accuracy

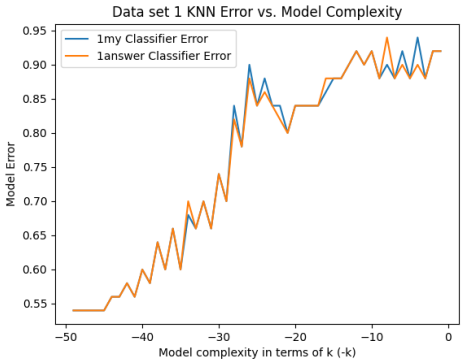
## KNNRegression Class

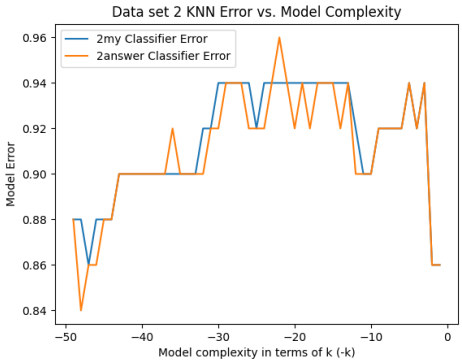
```python
class KNNRegression:
    def __init__(self,k):
        self.k = k

    def train(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def getNN(self, testEntry):
        distances = {}

        # get the distance between the test Entry and every training entry
        for trainEntryIdx in range(len(self.X_train)):
            dist = distance.euclidean(self.X_train[trainEntryIdx], testEntry)
            distances[trainEntryIdx] = dist

        #list of training entry index to that entry's distance to the test entry
        sorted_distance_map = list(sorted(distances.items(), key=lambda item: item[1], reverse=False))
        sorted_distance_map = sorted_distance_map[0:self.k]
        return [self.y_train[idx] for idx,distance in sorted_distance_map]


    def predict(self, X_test):
        numTestEntries = len(X_test)
        #initialize y_predict
        y_predict = [0 for i in range(numTestEntries)]

        for testEntryIdx in range(numTestEntries):
            #get the values  of the K nearest neighbors
            nn = self.getNN(X_test[testEntryIdx])

            nnCounts = Counter(nn)
            val, count = nnCounts.most_common()[0]
            y_predict[testEntryIdx] = sum(nnCounts.values()) / len(nnCounts)
        return y_predict

    def score(self, X_test, y_test):
        y_predict = self.predict(X_test)
        error = r2_score(y_test, y_predict)
        return error
```

| | Data set 1 | Data set 2 | Data set 3 |
|---|---|---|---|
| |   |   |   |
| k=1 |   |   |   |
| |   |   |   |
| k=3 |   |   |   |
| |   |   |   |
| k=5 |   |   |   |
| |   |   |   |

# (Model Complexity vs. Error)
# My KNN Classification against Correct Solution



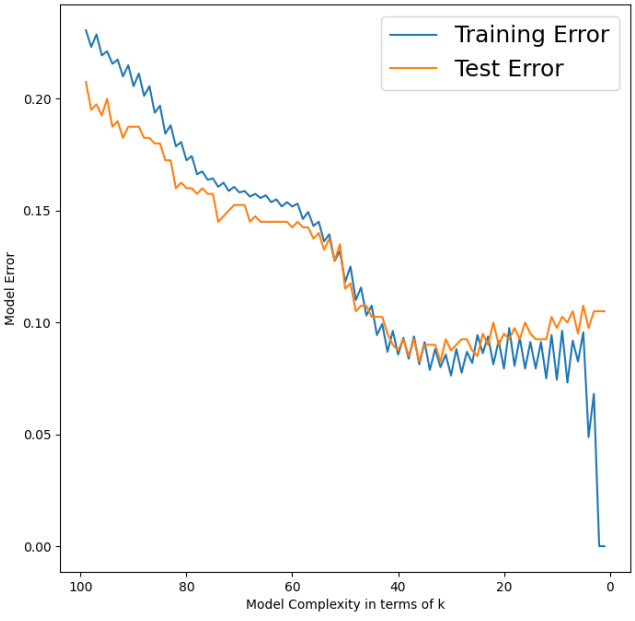




The point of these figures is to compare the error values I got using my KNN Classifier to an existing KNN classifier. My results matched up relatively closely with that of the imported KNN classifier.
Moreover, they also show model complexity versus the model's error (shown in better detail below).

---

# My KNNClassifier Training and Testing Error vs Model  Complextiy




### How do Results change with different values of k?

**k** is the **hyperparameter** for this model which controls the model's complexity. When k is large (k = n), the model complexity is low as the model will just predict the majority group every time. When k is small (k=1), the model complexity is high.

In the figures above, I plotted the model's error *(comparing y_predict = KNN(X_test) with y_test)* against the model's complexity (since small K means high model complexity and large k means small model complexity, I plotted it against -k so as the x-axis increases, so does the model complexity). Additionally, I compared my results to a correct implementation of KNNClassifier and KNNRegressor (my results are in **blue**, the correct implementation results are in **orange**).

When comparing the training and testing error, we see that when model complexity is low, both the training and testing errors are similar. When model complexity is high, although training error is low, the testing error is high. This makes sense as if we *overfit* our model to our training data with high model complexity, it's unlikely to generalize well on future/test data.

*Linear Regression:*

2. Consider the regression model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has rank $p < n$, $\boldsymbol{\beta} \in \mathbb{R}^p$, and $\boldsymbol{\epsilon} \in \mathbb{R}^n$ a random vector. Recall that the ordinary least squares (OLS) estimator is given by $\hat{\boldsymbol{\beta}}_{OLS} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$. Assume $\boldsymbol{\epsilon}$ is such that $\mathrm{E}(\boldsymbol{\epsilon}) = \mathbf{0}$ and $\mathrm{cov}(\boldsymbol{\epsilon}) = \sigma^2 \boldsymbol{I}$.

(a) Provide an expression for $\hat{\boldsymbol{\beta}}_{OLS}$ by solving the optimization problem above.

$$\hat{\beta}_{OLS} = \underset{\beta}{\operatorname{argmin}} \|y - X\beta\|_2^2 = \underset{\beta}{\operatorname{arg\,min}} (y - X\beta)^T (y - X\beta)$$

$$= \underset{\beta}{\operatorname{arg\,min}} \left( (y^T - \beta^T X^T)(y - X\beta) \right)$$

$$= \underset{\beta}{\operatorname{argmin}} \left( y^T y - 2(X\beta)^T y + (X\beta)^T (X\beta) \right)$$

To find $\hat{\beta}$ (the argmin), take the derivative wrt $\beta$ and set to 0 to find the min.

$$0 = -2X^T y + 2X^T X \hat{\beta}_{OLS}$$

$$X^T X \hat{\beta}_{OLS} = X^T y$$

$$\hat{\beta}_{OLS} = (X^T X)^{-1} (X^T y)$$

(b) Calculate $\mathrm{E}\left(\hat{\boldsymbol{\beta}}_{OLS}\right)$ and $\mathrm{cov}\left(\hat{\boldsymbol{\beta}}_{OLS}\right)$.

$$E(\hat{\beta}_{OLS}) = E\left((X^T X)^{-1} (X^T y)\right)$$

$$= E(\beta + X^{-1}\varepsilon)$$

$$= E(\beta) + X^{-1} \underbrace{E(\varepsilon)}_{0} \quad \text{linearity of expectation}$$

$$= \beta + 0$$

$$= \boxed{\beta}$$

sidenote

$$(X^T X)^{-1} (X^T (X\beta + \varepsilon))$$
$$= (X^T X)^{-1} (X^T X \beta + X^T \varepsilon)$$
$$= (X^T X)^{-1} (X^T X)\beta + (X^T X)^{-1} X^T \varepsilon$$
$$= \beta + X^{-1} (X^T)^{-1} X^T \varepsilon$$
$$= \beta + X^{-1} \varepsilon$$

$$\mathrm{Cov}(\hat{\beta}_{OLS}) = E\left[ (\hat{\beta}_{OLS} - E(\hat{\beta}_{OLS}))(\hat{\beta}_{OLS} - E(\hat{\beta}_{OLS}))^T \right]$$

$$= E\left( ((\beta + X^{-1}\varepsilon) - \beta)((\beta + X^{-1}\varepsilon) - \beta)^T \right)$$

$$= E\left( (X^{-1}\varepsilon)(X^{-1}\varepsilon)^T \right)$$

$$= E\left( X^{-1} \varepsilon \varepsilon^T (X^{-1})^T \right)$$

$$= X^{-1} E(\varepsilon^2)(X^{-1})^T$$

$$= \sigma^2 X^{-1} (X^T)^{-1}$$

$$= \sigma^2 (X^T X)^{-1}$$

from above, $E(\hat{\beta}_{OLS}) = \beta$ and $\hat{\beta}_{OLS} = \beta + X^{-1}\varepsilon$

$\Leftarrow$ since $E(\varepsilon) = 0$. $\mathrm{Cov}(\varepsilon) = E(\varepsilon^2) - E(\varepsilon)^2$ $\Rightarrow$ $\sigma^2 I = E(\varepsilon^2)$

(c) Show that the OLS predictions, $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}_{OLS}$, can be written as $\hat{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{y}$ for some matrix $\boldsymbol{H}$.

$$\hat{y} = X\hat{\beta}_{OLS}$$

$$\hat{y} = X(X^TX)^{-1}(X^Ty)$$

$H = X(X^TX)^{-1}X^T$ satisfies the equation

---

*Matrix Analysis:*

2. Answer the following:

   (a) Let $\boldsymbol{x} \in \mathbb{R}^p$ be a random vector with known mean $\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu}$ and known covariance $\mathrm{cov}(\boldsymbol{x}) = \boldsymbol{\Sigma}$, where $\boldsymbol{\Sigma}$ is positive definite. Find an affine transformation of $\boldsymbol{x}$ such that the transformed variable, $\tilde{\boldsymbol{x}}$, is such that $\mathrm{E}(\tilde{\boldsymbol{x}}) = \boldsymbol{0}$ and $\mathrm{cov}(\tilde{\boldsymbol{x}}) = \boldsymbol{I}$

$$\tilde{x} = Ax + b$$

- $E(\tilde{x}) = E(Ax+b) = AE[x] + b = 0 \quad \Rightarrow \quad b = -A\mu$
- $\mathrm{Cov}(\tilde{x}) = E\left((\tilde{x} - E(\tilde{x}))(\tilde{x} - E(\tilde{x}))^T\right)$

$$= E((\tilde{x})(\tilde{x})^T)$$
$$= E((Ax+b)(Ax+b)^T)$$
$$= E((Ax - A\mu)(Ax - A\mu)^T)$$
$$= E(A(x-\mu)(x-\mu)^TA^T)$$
$$I = AE((x-\mu)(x-\mu)^T)A^T$$
$$I = A\Sigma A^T$$

substitute: $\mathrm{cov}(\tilde{x}) = E((x-\mu)(x-\mu)^T) = \Sigma$

$\Sigma$ is positive definite $\exists$ matrix $R \in \mathbb{R}^{p \times p}$ such that $\Sigma = R^TR$

$$I = AR^TRA^T$$
$$(A^TA)^{-1} = R^TR$$
$$A^TA = R^{-1}(R^T)^{-1} = R^{-1}(R^{-1})^T$$
$$\Rightarrow \quad A^T = R^{-1} \quad \Rightarrow \quad A = (R^T)^{-1}$$

$\therefore$ Let $A = (R^T)^{-1}$ and $b = -(R^T)^{-1}\mu$

$$\tilde{x} = Ax + b$$

(b) Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Find the eigenvalues and eigenvectors of $\boldsymbol{xy}^T$.

$\underset{n\times 1}{x}\underset{1\times n}{y^T}\underset{n\times 1}{v} = \lambda \underset{n\times 1}{v}$

notice how $y^T x \in \mathbb{R}$ is a scalar.

We see that $(xy^T)x = x(y^T x) = (y^T x)x$ since $y^T x \in \mathbb{R}$

$\Rightarrow \lambda_1 = y^T x$ is an eigenvalue of $xy^T$ with corresponding eigenvector $x$.

Moreover, it's the only eigenvalue since if $\lambda_2$ with $v_2$ s.t $xy^T v_2 = \lambda_2 v_2$

then, $x(y^T v_2) = (y^T v_2)x = \underbrace{\lambda_2}_{\in\mathbb{R}} v_2$ $\quad v_2 = cx$ for some $c \in \mathbb{R}$
$\in \mathbb{R}$

$\Rightarrow v_2 = cx$ for some $c \in \mathbb{R}$ $\Rightarrow$ $x$ and $v_2$ correspond to the same eigenvalue
$\Rightarrow \lambda_1 = \lambda_2$.

$\therefore$ $y^T x$ is the only eigenvalue of $xy^T$ with corresponding vector $x$.

(c) Let $\boldsymbol{A}$ be a real symmetric $p \times p$ matrix with eigenvalues $\lambda_1, \ldots, \lambda_p$. Show that $\sum_{i=1}^p \sum_{j=1}^p a_{ij}^2 = \sum_{i=1}^p \lambda_i^2$.

Note how:
$(A^2)_{ii} = a_{1i} \cdot a_{i1} + a_{2i} \cdot a_{i2} + \ldots + a_{ni} \cdot a_{in}$
$= a_{1i}^2 + a_{2i}^2 + \ldots + a_{ni}^2$ since $A$ is symmetric

(a) $\operatorname{tr}(A^2) = \sum_{i=1}^n (A^2)_{ii}$

$= a_{11}^2 + a_{21}^2 + \ldots + a_{n1}^2$
$+ a_{12}^2 + a_{22}^2 + \ldots + a_{n2}^2$
$\vdots$
$+ a_{1n}^2 + a_{2n}^2 + \ldots + a_{nn}^2$
$= \sum_{i=1}^p \sum_{j=1}^p a_{ij}^2$

(b) Since $A$ is symmetric $\Rightarrow \exists\, U \in \mathbb{R}^n$ s.t $U \Lambda U^T = A$

$\operatorname{tr}(A^2) = \operatorname{tr}(U \Lambda U^T U \Lambda U^T)$
$= \operatorname{tr}(U \Lambda \Lambda U^T)$
$= \operatorname{tr}(\Lambda^2)$
$= \sum_{i=1}^p \lambda_i^2$ since $\Lambda$ is diagonal

Combining (a) & (b) $\Rightarrow$ $\sum_{i=1}^p \sum_{j=1}^p a_{ij}^2 = \sum_{i=1}^p \lambda_i^2$ ∎

*Multivariate Statistics:*

3. Suppose $\{\boldsymbol{x}_i\}_{i=1}^n$ are $n$ IID samples from a $p$-variate normal distribution with known mean $\mathbf{0}$ and unknown covariance $\Sigma$ (positive definite). Show that

$$\hat{\Sigma} = \frac{\boldsymbol{X}^T\boldsymbol{X}}{n}$$

is the maximum likelihood estimate of $\Sigma$, where

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{pmatrix}$$

is the *data matrix*.

*Here are some suggested steps to get you started:*

*(i) Write down the (joint) likelihood for this problem.*

*(ii) Calculate the log-likelihood (dropping constant terms as appropriate).*

*(iii) Take the gradient of the log-likelihood to get the* score equations*.*

*(iv) Solve for the maximum likelihood estimate.*

Hint: *It may be easier to parametrize the problem in terms of the precision (inverse covariance) matrix* $\Theta = \Sigma^{-1}$*, find* $\hat{\Theta}$*, and then use the* invariance of the maximum likelihood estimate *to find* $\hat{\Sigma} = (\hat{\Theta})^{-1}$.

(i) $x_i \sim N_p(0, \Sigma)$ Since each $x^{(i)}$ are IID, the joint liklihood:

$$L(\Sigma \mid X) = \prod_{i=1}^{n} f_{X_i}(x_i) = \prod_{i=1}^{n} \underbrace{\left( \frac{1}{(2\pi)^{p/2} \cdot |\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(x_i)^T \Sigma^{-1} x_i} \right)}_{f_{X_i}(x_i)}$$

(ii) log-liklihood:

$$\begin{aligned} \ell(\Sigma \mid X) &= \log(L(\Sigma \mid X)) \\ &= \sum_{i=1}^{n} \log\left( \frac{1}{(2\pi)^{p/2} \cdot |\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(x_i)^T \Sigma^{-1} x_i} \right) \qquad \text{properties of logs} \\ &= \sum_{i=1}^{n} \log\left(2\pi^{-\frac{p}{2}}\right) + \sum_{i=1}^{n} \log|\Sigma|^{-\frac{1}{2}} + \sum_{i=1}^{n} -\frac{1}{2}\left(x_i^T \Sigma^{-1} x_i\right) \\ &= C + \frac{n}{2}\log\left(\frac{1}{|\Sigma|}\right) - \frac{1}{2}\sum_{i=1}^{n}\left(x_i^T \Sigma^{-1} x_i\right) \qquad \text{where } C \text{ is a constant WRT. } \Sigma \end{aligned}$$

(iii) Following the hint, take the gradient WRT $\Theta = \Sigma^{-1}$

$$\begin{aligned} \nabla_\Theta \ell(\Theta \mid X) &= \nabla_\Theta \frac{n}{2}\log(|\Theta|) + \nabla_\Theta -\frac{1}{2}\sum_{i=1}^{n}\left(x_i^T \Theta x_i\right) \\ &= \frac{n}{2}\left(\Theta^{-1}\right)^T - \frac{1}{2}\sum_{i=1}^{n} x_i x_i^T \qquad (X^TX)^T = X^TX \\ 0 &= \frac{n}{2}\left(\hat{\Theta}^{-1}\right)^T - \frac{1}{2}\sum_{i=1}^{n} x_i x_i^T \end{aligned}$$

$\Rightarrow \quad (\hat{\Theta}^{-1})^T = \frac{1}{n} X^T X \Rightarrow \hat{\Theta}^{-1} = \frac{1}{2} X^T X$

$\Rightarrow \quad \hat{\Sigma} = (\hat{\theta})^{-1} \Rightarrow \boxed{\hat{\Sigma} = \frac{X^T X}{n}}$

---

*Convex Optimization:*

4. *For each of the following, is the function convex in* $\boldsymbol{x} \in \mathbb{R}^n$*? Strictly convex? Concave? None of the above? Briefly justify your answer.*

   (a) $\max(0, 1 - y * \alpha^T \boldsymbol{x})$ *for* $y \in \{\pm 1\}$. – convex : a hyperplane that becomes 0

   (b) $\log(1 + e^{\alpha^T \boldsymbol{x}})$ – strictly convex :

   (c) $-y \log(\alpha^T \boldsymbol{x}) - (1 - y) \log(1 - \alpha^T \boldsymbol{x})$ *for* $y \in \{0, 1\}$ - equation of a line ?

   (d) $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_1$ – convex : visualizing the graph, there are constant slope lines, so not strictly convex

   (e) $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \|\boldsymbol{x}\|_2^2$ – convex : both $\|y - Ax\|_2^2$ and $\|x\|_2^2$ are convex $\Rightarrow$ their sum is convex
   (we optimize this form in Ridge Regression, so makes sense that it is convex)

   *In each of the above,* $\alpha \in \mathbb{R}^n$ *is a fixed vector,* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ *is a full-rank matrix with* $m < n$.

5. ~~*For parts (c) and (e) above, calculate the gradient of the function and write out a gradient descent (or ascent) method in pseudo-code to minimize (or maximize) the function.*~~

didn't got to this