EECS 445 FA 2014

HW 1

WU, Tongshuang 40782356

1.

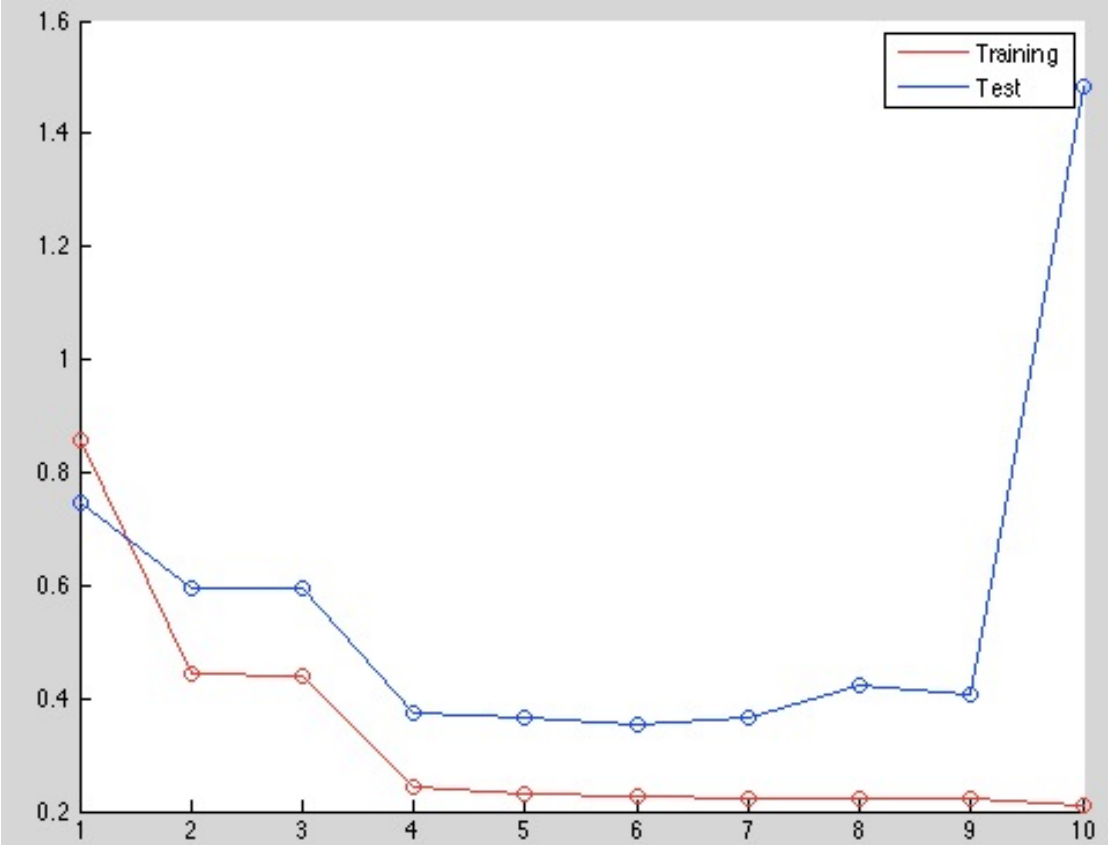
a)

i) The coefficient found by:

Batch gradient descent	$\begin{bmatrix} 2.446406 \\ -2.816353 \end{bmatrix}$	*1940*
Stochastic gradient descent	$\begin{bmatrix} 2.454460 \\ -2.809909 \end{bmatrix}$	*2294*
Newton's method	$\begin{bmatrix} 2.446406 \\ -2.816353 \end{bmatrix}$	*1*

ii) The count of iteration is used for measuring the rate of convergence, as shown in the column row above. While Newton's method takes only one iteration, it requires 0.015131s for completement.

b)

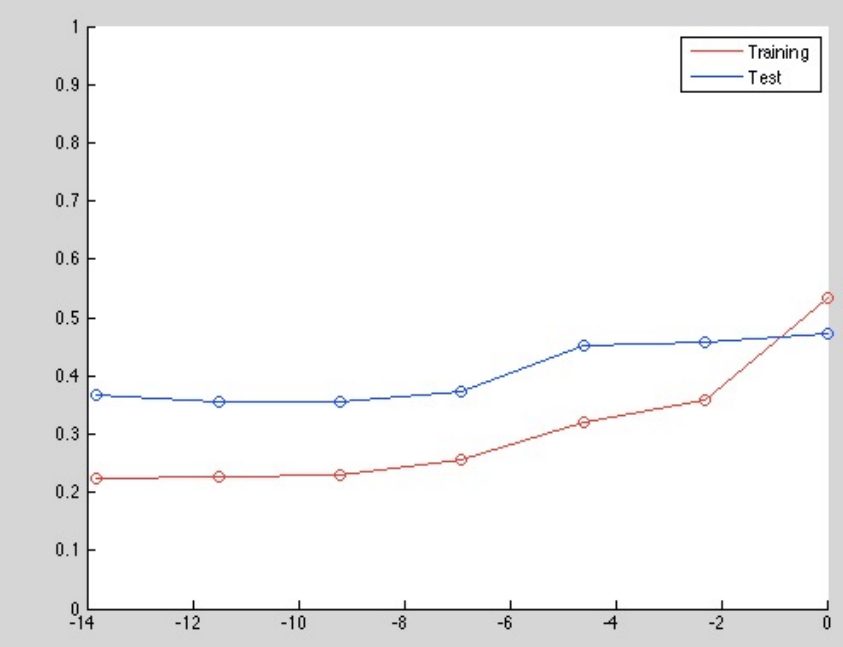


ii) The best fit degree of polynomial should be 6, where both training data set and test data set achieve the lowest RMS error. First degree of polynomial tends to

under-fitting the data, since both the training and test data set get high RMS error, and over-fitting occurs at ninth and tenth degree of polynomial, where the training set's RMS error nears zero, while the one for test set grows dramatically, meaning that the computed ***w*** fits the traninig set perfectly, but cannot be generated for other sets.

c)

i) The regenerated chart is as the following.



ii) The RMS error reachs the lowest point at $ln(\lambda) \approx -12$, i.e. $\lambda = 10^{-6}$ works the best.

2.

(a) For $E_D(w) = \frac{1}{2}\sum_{i=1}^{N} r^{(i)} \left(w^T x^{(i)} - t^{(i)}\right)^2$

$$= \frac{1}{2}\sum_{i=1}^{N} \left(w^T x^{(i)} - t^{(i)}\right) r^{(i)} \left(w^T x^{(i)} - t^{(i)}\right)$$

Also notice:

$$\phi w - T = \begin{bmatrix} w^T \phi(x)^{(1)} - t^{(1)} \\ w^T \phi(x)^{(2)} - t^{(2)} \\ w^T \phi(x)^{(3)} - t^{(3)} \\ \dots \\ w^T \phi(x)^{(n)} - t^{(n)} \end{bmatrix},$$

$$(\phi w - T)^T = [w^T \phi(x)^{(1)} - t^{(1)} \quad w^T \phi(x)^{(2)} - t^{(2)} \quad \dots \quad w^T \phi(x)^{(n)} - t^{(n)}]$$

$\therefore E_D(w) = (\phi w - T)^T R (\phi w - T)$, where

$$\mathrm{R} = \begin{bmatrix} r^{(1)} & 0 & \dots & 0 \\ 0 & r^{(2)} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & r^{(n)} \end{bmatrix}$$

(b) $\frac{\partial}{\partial w} E_D(w) = \frac{\partial}{\partial w} (\phi w - T)^T R (\phi w - T)$

$= \frac{\partial}{\partial w} (\phi w - T)^T R (\phi w - T)$

$= \frac{\partial}{\partial w} (w^T \phi^T - T^T) R (\phi w - T)$

$= \frac{\partial}{\partial w} (w^T \phi^T R \phi w - w^T \phi^T R T - T^T R \phi w + T^T R T)$

$= \phi^T R \phi w - \phi^T R T$

Set $\frac{\partial}{\partial w} E_D(w) = 0$,

$\therefore w = (\phi^T R \phi)^{-1} \phi^T R T$

(c) Likelihood function: $L(w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} exp\left(-\frac{\left(w^T x^{(i)} - t^{(i)}\right)^2}{2(\sigma^{(i)})^2}\right)$

$l(w) = log(L(w)) = log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} exp\left(-\frac{\left(w^T x^{(i)} - t^{(i)}\right)^2}{2(\sigma^{(i)})^2}\right)\right)$

$= \sum_{i=1}^{n} log\left(\frac{1}{\sqrt{2\pi}\sigma^{(i)}} exp\left(-\frac{\left(w^T x^{(i)} - t^{(i)}\right)^2}{2(\sigma^{(i)})^2}\right)\right)$

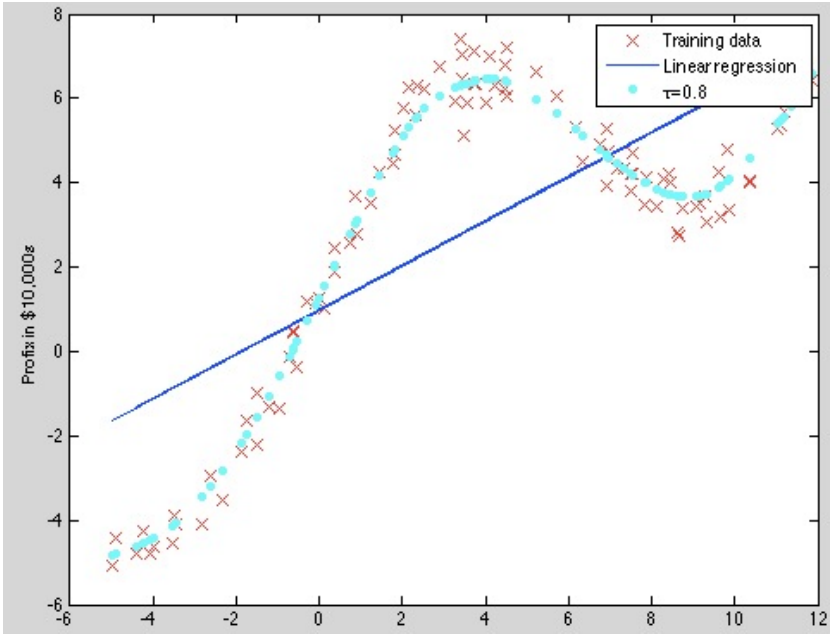
$= log \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} - \sum_{i=1}^{n} \frac{\left(w^T x^{(i)} - t^{(i)}\right)^2}{2(\sigma^{(i)})^2}$

Let $r^{(i)} = \frac{1}{(\sigma^{(i)})^2}$,

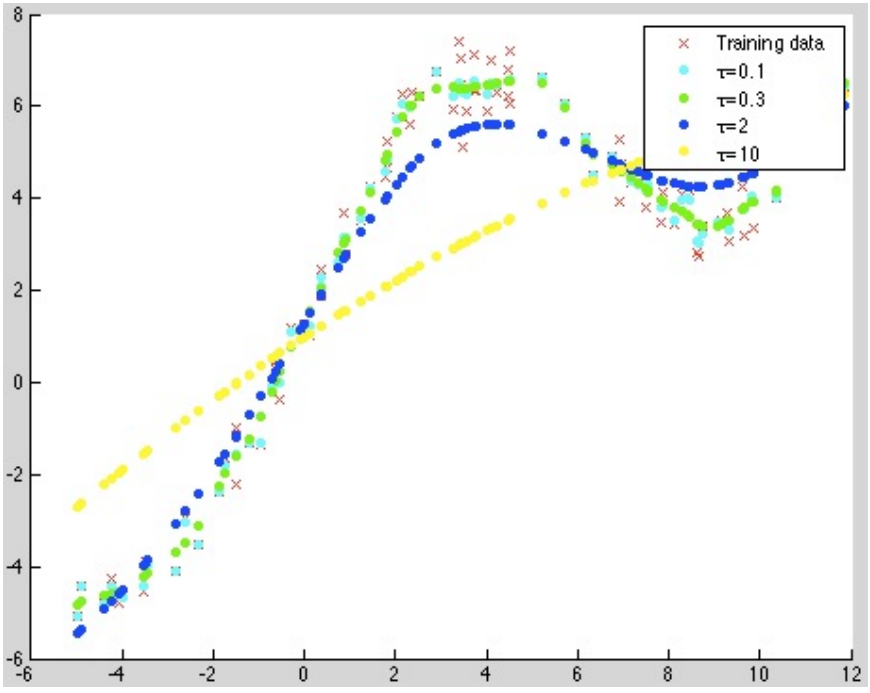
$= log \sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} - \frac{1}{2} \sum_{i=1}^{n} r^{(i)} \left(w^T x^{(i)} - t^{(i)}\right)^2$

Since *log* function is a strictly increasing function, L(w) and l(w) has the same maximum point, and finding l(w)'s max value is equivalent to finding $\frac{1}{2} \sum_{i=1}^{n} r^{(i)} \left(w^T x^{(i)} - t^{(i)}\right)^2$, or $E_D(w)$'s minimum value. Therefore, finding the maximum likelihood estimate of w reduces to solving a weighted linear regression problem.

(d) (i) (ii) Implement un-weighted ("--") and weighted ("o") linear regression



iii) The implementations are shown below.



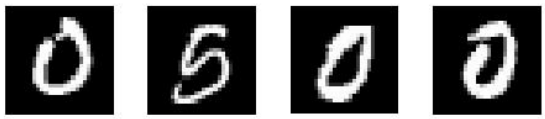
As discussed in class, too large a kernel includes points that degrade the fit; too small a kernel neglects points that increase confidence in the fit, which is confirmed by the above implementation.

3.

i)

Example 1:

Eight neighbors	neighbor =

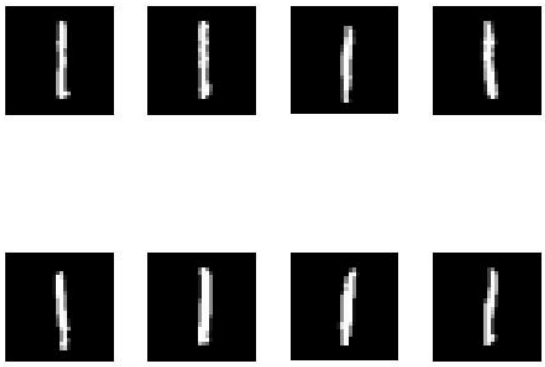
	953 0 691 5 321 0 211 0 193 0 837 5 82 0 543 0
Example image	
Nearest neighbors	 

Example 2:

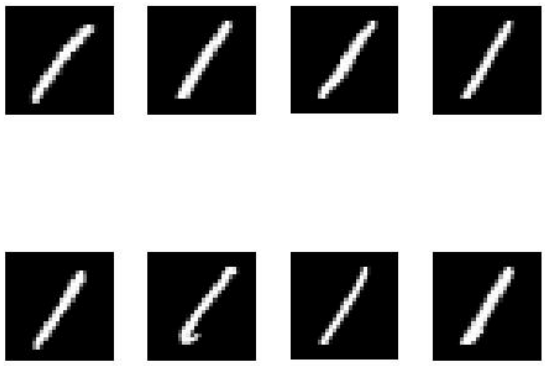
Eight neighbors	neighbor = 724 7 754 7 72 7 16 7 912 7 982 7 194 7 80 7
Example image	

Nearest neighbors	

Example 3:

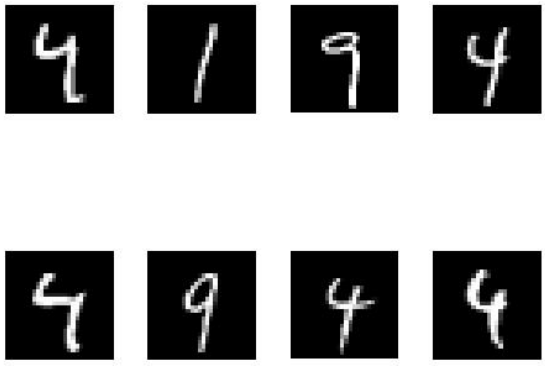
Eight neighbors	neighbor = 485 1 399 1 455 1 639 1 9 1 679 1 113 1 311 1
Example image	
Nearest neighbors	

Example 4:

Eight neighbors	neighbor = 588 1 952 1 594 1 962 1 476 1 638 1 994 1 920 1
Example image	
Nearest neighbors	

Example 5:

Eight neighbors	neighbor = 577 4 605 1 305 9 27 4 719 4 901 9 413 4 355 4

Example image	
Nearest neighbors	

(b) When *k* is 10, the accuracy is 85.7%. See the code for detailed implementation.

(c)

k	Accuracy (un-weighted Euclidean distance)	Accuracy (weighted Euclidean distance) **(for d)**
2	85.0%	88.3%
4	85.3%	88.4%
6	86.5%	87.0%
8	85.5%	86.1%
10	85.7%	85.9%
20	82.3%	82.9%
50	75.8%	76.9%
100	67.7%	69.3%

Advantages: KNN goes through every pixel of the test and the training example. This guarantees accuracy.

Disadvantage: in the cases where we have extreme large figures or too many training examples, the computation will be extremely expensive. Also if the training example is very biased, e.g. 1000 “4” and 10 “6”, it’s very likely that some neighbors will be off-topic.

From the experiment result, K needs to be small. When it’s very small the accuracy varies around 85%. 6-10 seems to be a reasonable range, since if it’s 2-5,

neighbors that could raise classification accuracy can potentially be ignored, and therefore finding the test example's class among them will lead to relatively unreliable result. As k keeps growing, some less similar neighbors will be included, which decreases the accuracy significantly (see accuracy for k = 20, 50 and 100)

However, k = 1 leads to a much higher accuracy. This should be because of 1-NN's accuracy. When browsing online, it is said KNN can only dominate 1NN in terms of frequency when the neighbor's size is extremely small comparing to the training size.

(d)

Improvement 1: Use dimension reduction (e.g. PCA) to narrow down the variables that needs to be checked, which can speed up the code (not implemented since this method cannot improve accuracy)

Improvement 2: weighted KNN: weighting every neighbor with $\frac{1}{d}$, where d is the distance.

$$y_{test} = argmax_t \sum_{(x't')\in KNN(X_{test})} \frac{1}{d}[t' = t]$$

As shown in the table, of each k, the weighted KNN can improve the accuracy, and it can perform better improvement when k is bigger.

4.

(a) $\frac{\partial}{\partial w}h(x) = \frac{\partial}{\partial w}\frac{1}{1+e^{-w^T x}}$

$= \frac{1}{(1+e^{-w^T x})^2}(x)(e^{-w^T x})$

$= (x)h(x)(1-h(x))$

As in the class:

$\frac{\partial}{\partial w_j}l(w) = \sum_{n=1}^{N}(t^{(n)} - h(x^{(n)}))\phi_{nj}$

$\therefore \frac{\partial^2}{\partial w_i \partial w_j}l(w) = \frac{\partial}{\partial w_i}\sum_{n=1}^{N}(t^{(n)} - h(x^{(n)}))\phi_{nj}$

$= -\frac{\partial}{\partial w_i}\sum_{n=1}^{N} h(x^{(n)})\phi_{nj}$

$= -\sum_{n=1}^{N}(1-h(x^{(n)}))h(x^{(n)})\phi_n \phi_n{}^T$

$= -\sum_{n=1}^{N}\phi_{ni}(1-h(x^{(n)}))h(x^{(n)})\phi_{nj}$

To show Hessian H is negative semi-definite:

$$\sum_i \sum_j z_j x_j x_i z_i$$
$$= \sum_i x_i z_i \sum_j z_j x_j$$
$$= (z^T x)^T x^T z$$
$$= (x^T z)^2$$
$$\geq 0$$

$\because \quad H = -\sum_{n=1}^{N} \phi_{ni}(1 - h(x^{(n)}))h(x^{(n)})\phi_{nj}$

where $-(1 - h(x^{(n)}))h(x^{(n)}) \leq 0$ is always the case, the coefficient of $x_j x_i$ is always negative, so that the Hessian is a sum of negative semi-definite matrices, and is thus negative semi-definite.

(b)

$$\begin{bmatrix} -1.419251 \\ -0.784840 \\ -0.199423 \end{bmatrix}$$

(c)