Contents

HW 1 Linear Regression

1.1 Derivation and Proof

1.1.1 Derive the solution for w_0 and w_1 in linear regression

Consider the linear regression problem for 1D data, where we would like to learn a function $h(x) = w_1 x + w_0$ with parameters w_0 and w_1 to minimize the sum squared error:

$$L = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - h(x^{(i)}))^2$$

for N pairs of data samples $(x^{(i)}, y^{(i)})$. Derive the solution for w_0 and w_1 for this 1D case of linear regression. Show the derivation to get the solution:

$$w_0 = Y - w_1 X,$$

$$w_1 = \frac{\frac{1}{N}\sum_{i=1}^{N} x^{(i)} y^{(i)} - YX}{\frac{1}{N}\sum_{i=1}^{N}(x^{(i)})^2 - X^2},$$

where X is the mean of $\{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\}$ and Y is the mean of $\{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\}$.

Proof (As we know, this is a convex function of w and the critical point for a convex function is a global min. So we) We first derive the optimal for w_0:

$$\frac{\partial L}{\partial w_0} = \frac{\partial}{\partial w_0}\left[\tfrac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - w_1 x^{(i)} - w_0\right)^2\right]$$

For each i, we have (by chain rule)

$$\frac{\partial}{\partial w_0}\left(y^{(i)} - w_1 x^{(i)} - w_0\right)^2 = 2\left(y^{(i)} - w_1 x^{(i)} - w_0\right)(-1)$$

Setting the partial to 0:

$$\frac{\partial L}{\partial w_0} = -\sum_{i=1}^{N}\left[y^{(i)} - (w_1 x^{(i)} + w_0)\right] = 0 \tag{1.1}$$

$$\sum_{i=1}^{N} y^{(i)} = \sum_{i=1}^{N}\left(w_1 x^{(i)} + w_0\right) \tag{1.2}$$

$$= w_1 \sum_{i=1}^{N} x^{(i)} + N\, w_0 \tag{1.3}$$

Then dividing by $1/N$ in both sides:

$$\frac{1}{N}\sum_{i=1}^{N} y^{(i)} = w_1\left(\frac{1}{N}\sum_{i=1}^{N} x^{(i)}\right) + w_0 \tag{1.4}$$

$$\overline{Y} = w_1 \overline{X} + w_0 \tag{1.5}$$

Thus we have

$$w_0 = \overline{Y} - w_1 \overline{X} \tag{1}$$

Next, we derive the optimal value of w_1.

$$\frac{\partial L}{\partial w_1} = \frac{\partial}{\partial w_1}\Big[\tfrac{1}{2}\sum_{i=1}^{N}\big(y^{(i)} - w_1 x^{(i)} - w_0\big)^2\Big].$$

for each i, we have (by chain rule):

$$\frac{\partial}{\partial w_1}\big(y^{(i)} - w_1 x^{(i)} - w_0\big)^2 = 2\left(y^{(i)} - w_1 x^{(i)} - w_0\right)(-x^{(i)}).$$

So by setting the partial to 0:

$$\frac{\partial L}{\partial w_1} = -\sum_{i=1}^{N}\left[y^{(i)} - (w_1 x^{(i)} + w_0)\right] x^{(i)} = 0 \tag{1.6}$$

$$\sum_{i=1}^{N} y^{(i)} x^{(i)} = \sum_{i=1}^{N}(w_1 x^{(i)} + w_0)\, x^{(i)} \tag{1.7}$$

$$\sum_{i=1}^{N} y^{(i)} x^{(i)} = w_1 \sum_{i=1}^{N}(x^{(i)})^2 \;+\; w_0 \sum_{i=1}^{N} x^{(i)} \tag{1.8}$$

Using (1) for w_0, and dividing by $1/N$ on both sides:

$$\frac{1}{N}\sum_{i=1}^{N} x^{(i)} y^{(i)} = w_1 \frac{1}{N}\sum_{i=1}^{N}(x^{(i)})^2 \;+\; (\overline{Y} - w_1\overline{X})\overline{X} \tag{1.9}$$

$$= w_1\left[\frac{1}{N}\sum_{i=1}^{N}(x^{(i)})^2 - X^2\right] + \overline{YX} \tag{1.10}$$

Resolving for w_1:

$$w_1 = \frac{\frac{1}{N}\sum_{i=1}^{N} x^{(i)} y^{(i)} - \overline{XY}}{\frac{1}{N}\sum_{i=1}^{N}(x^{(i)})^2 - \overline{X}^2} \tag{2}$$

This completes the solution for both w_0 and w_1.

1.1.2 positive definite matrices

Recall the definition and property of positive (semi-)definite matrix. Let A be a real, symmetric $d \times d$ matrix. A is positive semi-definite (PSD) if, for all $z \in \mathbb{R}^d$, $z^\top A z \geq 0$. A is positive definite (PD) if, for all $z \neq 0$, $z^\top A z > 0$. We write $A \succeq 0$ when A is PSD, and $A \succ 0$ when A is PD.

It is known that every real symmetric matrix A can be factorized via the eigenvalue decomposition: $A = U\Lambda U^\top$, where U is a $d \times d$ matrix such that $UU^\top = U^\top U = I$ and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$. Multiplying on the right by U, we see that $AU = U\Lambda$. If we let u_i denote the i-th column of U, we have $Au_i = \lambda_i u_i$ for each i; λ_i are eigenvalues of A, and the corresponding columns of U are eigenvectors associated with λ_i. The eigenvalues constitute the "spectrum" of A.

i . Prove A is PD if and only if $\lambda_i > 0$ for each i. (Note: "if and only if" means proving both directions.)

Proof Let A be a real $d \times d$, symmetric matrix, with eigen-decomposition

$$A = U\Lambda U^\top$$

where U is an orthogonal matrix and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$ is a diagonal matrix of the eigenvalues.

forward direction $\Rightarrow$: Assume A is positive definite. Let u_i be an eigenvector of A with eigenvalue λ_i. Then

$$u_i^\top A\, u_i = u_i^\top \left(\lambda_i u_i\right) = \lambda_i\, u_i^\top u_i = \lambda_i \left\|u_i\right\|^2 > 0$$

Since $u_i \neq 0$, must have $\|u_i\|^2 > 0$, and therefore $\lambda_i > 0$. This holds for each i, so all eigenvalues are positive.

backward direction ($\Leftarrow$): Conversely, assume $\lambda_i > 0$ for every i. Let $z \neq 0 \in \mathbb{R}^d$ be arbitrary vector, then

$$z^\top A\, z = z^\top (U\, \Lambda\, U^\top)\, z = (U^\top z)^\top \Lambda\, (U^\top z)$$

Let $w = U^\top z$. Note that $w \neq 0$ whenever $z \neq 0$ (since U is invertible). Then:

$$z^\top A\, z = w^\top \Lambda\, w = \sum_{i=1}^{d} \lambda_i\, w_i^2.$$

Since $\lambda_i > 0$ and at least one $w_i^2 \geq 0$ (since $w \neq 0$), we get $\sum_{i=1}^{d} \lambda_i w_i^2 > 0$.

Therefore $z^\top A\, z > 0$. Since z is arbitrary, finishing the proof that $A \succ 0$.

Thus, $A \succ 0 \iff$ all eigenvalues $\lambda_i > 0$.

ii Consider the linear regression problem where Φ and y are as defined in class. The closed-form solution becomes $(\Phi^\top \Phi)^{-1} \Phi^\top y$.

Now consider a ridge regression problem with the regularization term $\frac{1}{2\beta}\|w\|_2^2$. The symmetric matrix in the closed-form solution is $\Phi^\top \Phi + \beta I$. Derive the eigenvalues and eigenvectors for $\Phi^\top \Phi + \beta I$ with respect to the eigenvalues and eigenvectors of $\Phi^\top \Phi$, denoted as λ_i and u_i. Prove that the matrix $(\Phi^\top \Phi + \beta I)$ is PD for any $\beta > 0$.

Proof (1) Eigenvalues/eigenvectors of $\Phi^\top \Phi + \beta I$: Since $\Phi^T \Phi$ is real symmetric, suppose $\Phi^\top \Phi$ eigen-decomposes as:

$$\Phi^\top \Phi = U\, \Lambda\, U^\top$$

Then

$$\Phi^\top \Phi + \beta I = U\, \Lambda\, U^\top + \beta\, I$$

We can rewrite $\beta\, I = \beta\, U\, U^\top$ since $U\, U^\top = I$. Hence,

$$\Phi^\top \Phi + \beta I = U\, \Lambda\, U^\top + \beta\, U\, U^\top = U\, (\Lambda + \beta I)\, U^\top.$$

Thus the eigenvectors of $\Phi^\top \Phi + \beta I$ are the same as those of $\Phi^\top \Phi$ (the columns of U), and the eigenvalues are $\lambda_i + \beta$.

(2) Positivity of $\Phi^\top \Phi + \beta I$:

Claim: for any matrix A, $A^T A$ is potisive semidefinite.

Proof of claim: let x be arbitary nonzero input to $A^T A$, then

$$x^T (A^T A) x = (Ax)^T (Ax) = ||Ax||_2 0$$

Therefore $\Phi^\top\Phi$ is positive semidefinite, so its eigenvalues $\lambda_i \geq 0$ (for the same reasoning as the proof in in(i).) Adding βI shifts each eigenvalue by $\beta > 0$. Hence each eigenvalue of $\Phi^\top\Phi + \beta I$ is $\lambda_i + \beta > 0$. Therefore, $\Phi^\top\Phi + \beta I$ is positive definite for any $\beta > 0$, by (i).

1.1.3 Maximizing log-likelihood in logistic regression

In this sub-problem, logistic regression is used to predict the class label $y \in \{-1, +1\}$ instead of $y \in \{0, 1\}$. Show that maximizing the log-likelihood of logistic regression,

$$\sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}),$$

is equivalent to minimizing the following loss function:

$$\sum_{n=1}^{N} \log\left(1 + \exp(-y^{(n)} \cdot w^\top \phi(x^{(n)}))\right).$$

[Hint: You can expand the log-likelihood as follows:

$$logP(y^{(n)}|x^{(n)}) = I(y^{(n)} = 1)logP(y^{(n)} = 1|x^{(n)}) + I(y^{(n)} = -1)logP(y^{(n)} = -1|x^{(n)})$$

and then plug in the class posterior probability of the logistic regression model.]

Proof The log-likelihood for data $\{(x^{(n)}, y^{(n)})\}_{n=1}^N$ is

$$\log \prod_{n=1}^{N} P\big(y^{(n)} \mid x^{(n)}\big) \;=\; \sum_{n=1}^{N} \log P\big(y^{(n)} \mid x^{(n)}\big)$$

Since

$$P(y = +1 \mid x) = \sigma\big(w^\top \phi(x)\big)$$

$$P(y = -1 \mid x) = 1 - \sigma\big(w^\top \phi(x)\big)$$

and $\sigma(-a) = 1 - \sigma(a)$, so we can write:

$$P\big(y^{(n)} \mid x^{(n)}\big) = \sigma\big(y^{(n)}\, w^\top \phi(x^{(n)})\big)$$

Hence the log-likelihood is:

$$\sum_{n=1}^{N} \log \sigma\big(y^{(n)}\, w^\top \phi(x^{(n)})\big)$$

Since $\sigma(z) = 1/[1 + \exp(-z)]$, $\log \sigma(z) = -\log\big(1 + \exp(-z)\big)$, we have

$$\log \sigma\big(y^{(n)}\, w^\top \phi(x^{(n)})\big) = -\log\big[1 + \exp\big(-\, y^{(n)}\, w^\top \phi(x^{(n)})\big)\big]$$

Thus,

$$\sum_{n=1}^{N} \log P\big(y^{(n)} \mid x^{(n)}\big) = \sum_{n=1}^{N} \log \sigma\big(y^{(n)}\, w^\top \phi(x^{(n)})\big) = -\sum_{n=1}^{N} \log\Big[1 + \exp\big(-\, y^{(n)}\, w^\top \phi(x^{(n)})\big)\Big]$$

This finishes the proof that

$$\text{maximizing} \sum_{n=1}^{N} \log P\big(y^{(n)} \mid x^{(n)}\big) \iff \text{minimizing} \sum_{n=1}^{N} \log\Big[1 + \exp\big(-\, y^{(n)}\, w^\top \phi(x^{(n)})\big)\Big].$$

1.2 Linear Regression on a Polynomial

In this problem, you will implement linear regression on a polynomial. Please have a look at the accompanied starter code linear regression.py and notebook linear regression.ipynb for instructions first. Please note that all the sub-questions without (Autograder) need to be answered in your writeup.

Sample data: The files 'q2xTrain.npy', 'q2xTest.npy', 'q2yTrain.npy' and 'q2yTest.npy' specify a linear regression problem for a polynomial. 'q2xTrain.npy' represent the inputs $(x^{(i)}\mathbb{R})$ and 'q2yTrain.npy' represents the outputs $(y^{(i)}\mathbb{R})$ of the training set, with one training example per row.

1.2.1 GD and SGD

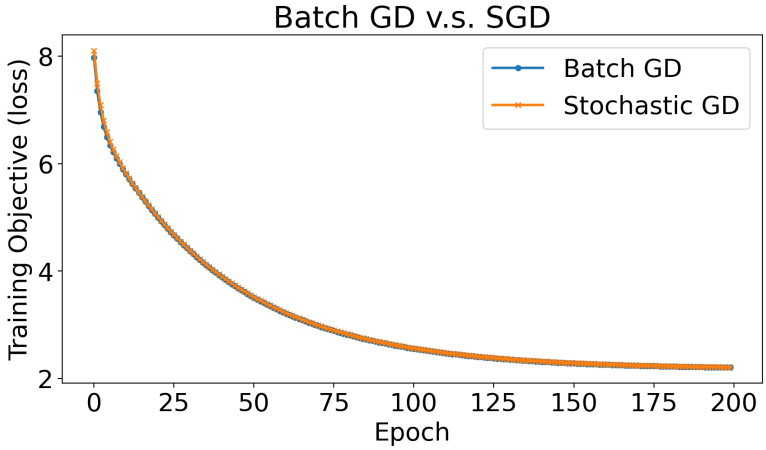
You will compare the following two optimization methods, in finding the coefficients of a polynomial of degree one (i.e. slope and intercept) that minimize the training loss. • Batch gradient descent (GD) • Stochastic gradient descent (SGD) Here, as we seen in the class, the training objective is defined as:

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(\sum_{j=0}^{M-1} w_j\phi(x^{(i)}) - y^{(i)})^2 = \frac{1}{2}(w^T\phi(x^{(i)}) - y^{(i)})^2$$

(a) [12 points] (Autograder) Implement the GD and SGD optimization methods. For all the implementation details (e.g., function signature, initialization of weight vectors, etc.), follow the instruction given in the code files. Your score for this question will be graded by the correctness of the implementation.

(b) [3 points] Please share the plot generated from the section 2(b) of your .ipynb file in your write-up, and then compare two optimization methods, GD and SGD. Which one takes less time and which one shows lower test objective $E(w_{test})$?

Sol.



"GD version took 0.00 seconds

GD Test objective = 2.7017

SGD version took 0.06 seconds

SGD Test objective = 2.6796"

In our example, GD takes shorter time than SGD. One reason it that we have extremely small scale of 20 data points, so we can make full use of parallel computing. Through vectorization, for the 200 epoches, we takes about 200 computation. But for SGD, we iterate through the whole data set every time, so we takes about 200*20

= 4000 computations. But for larger sample scale and more schochastic choice of data points each epoch (instead of going through the whole dataset), SDG could take shorter time.
SGD performs a little bit better on test objective than GD. It could help escape local minima and more possibly lead to better global solution.

1.2.2 Over-fitting Study

Next, you will investigate the problem of over-fitting. Recall the figure from lecture that explored over-fitting as a function of the number of features (e.g., M, the number of terms in a $(M-1)$-degree polynomial). To evaluate this behavior, we examine the Root-Mean-Square (RMS) Error defined below. (Note: we use the RMS error just for evaluation purposes, NOT as a training objective.)

$$E_{\text{RMS}} = \sqrt{\frac{2E(w^*)}{N}}$$

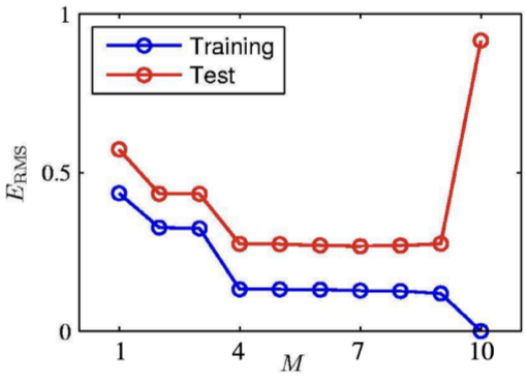
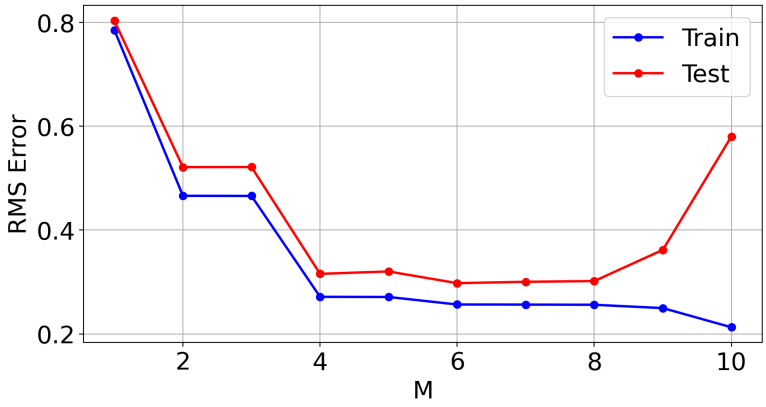


Figure 1: (Sample plot) Overfitting study: Train-Test accuracy.

(a) [8 points] (Autograder) Implement the closed-form solution of linear regression (assuming all conditions are met) instead of iterative optimization methods. (Hint: we recommend using `np.linalg.inv` to compute the inverse of a matrix.)
(b) [2 points] Regenerate the plot with the provided data. The sample training data can be generated with $(M-1)$-degree polynomial features (for $M = 1, 2, \ldots, 10$) from `q2xTrain.npy` and `q2yTrain.npy`. We assume the feature vector is:

$$\phi(x^{(i)}) = (1, x^{(i)}, (x^{(i)})^2, \ldots, (x^{(i)})^{M-1})$$

for any value of M. For the test curve, use the data in `q2xTest.npy` and `q2yTest.npy`. Note that the trend of your curve is not necessarily the same as the sample plot. Attach your plot to the **write-up**.
Sol.



(c) [2 points] In the **write-up**, discuss: Which degree polynomial would you say best fits the data? Was there evidence of under/over-fitting the data? Use your generated plots to justify your answer.

Sol. As M become larger, the training and testing E_{RMS} both went down before $M = 6$, stayed still for $M = 6$ through $M = 8$, and after M went beyong 8, the training E_{RMS} kept on going down, but the testing E_{RMS} suddenly went up. This is probably the point where over-fitting began.

I would say $M = 6$, i.e. the 5 degree polynomial best fits the data. This is because $M = 6$ reaches both the lowest training and testing E_{RMS}.

1.2.3 Regularization (Ridge Regression)

Finally, you will explore the role of regularization. Recall the image from lecture that explored the effect of the regularization factor λ.

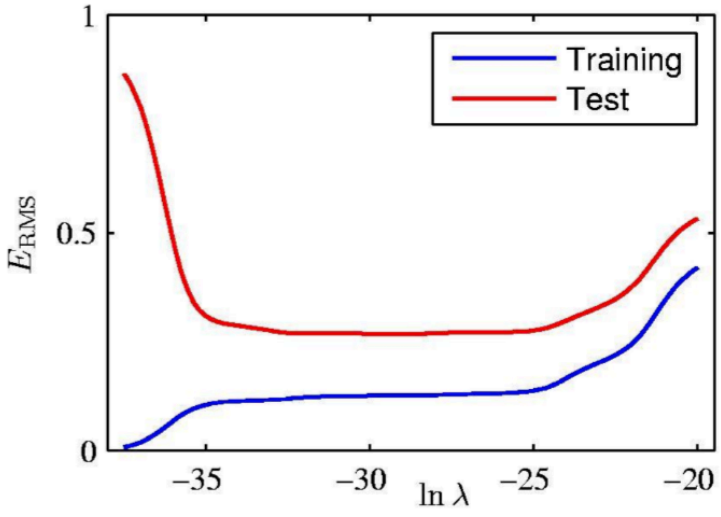


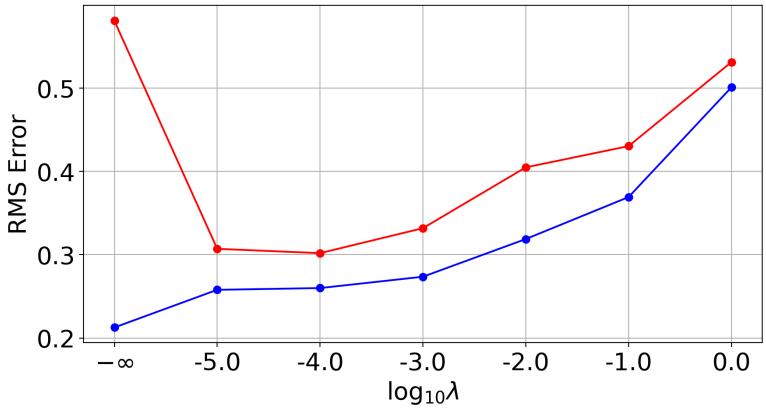
Figure 2: (Sample plot) effects of the regularization factor λ.

(a) [8 points] (Autograder) Implement the closed-form solution of ridge regression. Specifically, use the following regularized objective function:

$$\frac{1}{2}\sum_{i=1}^{N}(w^\top \phi(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\|w\|_2^2$$

to optimize the parameters w.

(b) [2 points] For the sample data, regenerate the plot (Figure 2) with $\lambda \in \{10^{-5}, 10^{-4}, \ldots, 10^{-1}, 10^{0}(=1)\}$. First, compute the E_{RMS} over the training data specified in `q2xTrain.npy` and `q2yTrain.npy` with λ, and then measure the test error using `q2xTest.npy` and `q2yTest.npy`. Attach your (unregularized) E_{RMS} plot of both the training and test data obtained from 2(g) to the **write-up**. Note that the trend of your curve is not necessarily the same as the sample plot.



(c) [2 points] Discuss: Which λ value seemed to work best for a ninth-degree polynomial ($M = 10$)? Use your generated plots to justify your answer. Provide your answer in the **write-up**.

Sol. I think $\lambda = 0.0001$, i.e. $\log_{10}\lambda = -4$ seemed to work best for a ninth-degree polynomial. As shown in the plot, when fixing $M = 10$, $\log_{10}\lambda = -4$ has the lowest testing error, and the training error is stable around the point. It shows that this regularization coefficient both can prevent over-fitting and keep descent expressibility of the model.

1.3 Locally Weighted Linear Regression

Consider a linear regression problem in which we want to weight different training examples differently. Specifically, suppose we want to minimize:

$$E_D(w) = \frac{1}{2}\sum_{i=1}^{N} r^{(i)}(w^\top x^{(i)} - y^{(i)})^2,$$

where $r^{(i)} \in \mathbb{R}$ is the "local" weight for the sample $(x^{(i)}, y^{(i)})$. In class, we worked on a special case where all the weights $r^{(i)}$ are equal. In this problem, we generalize these ideas to the weighted setting and implement the locally weighted linear regression algorithm.

Notes: 1. The weight $r^{(i)}$ can be different for each of the data points in the training data. 2. For a 1-dimensional input x (provided in this problem), the model can be written as $w_0 + w_1 x$, where w_0 acts as the intercept term. This is naturally incorporated by extending x to include a constant term, such as $x = [1, x]^\top$, in the formulation of the linear model.

(a) [3 points] Show that $E_D(w)$ can also be written as:

$$E_D(w) = (w^\top X - y^\top)R(w^\top X - y^\top)^\top,$$

for an appropriate diagonal matrix R, where $X \in \mathbb{R}^{D\times N}$ is a matrix whose i-th column is $x^{(i)} \in \mathbb{R}^{D\times 1}$, and $y \in \mathbb{R}^{N\times 1}$ is the vector whose i-th entry is $y^{(i)}$. Here, in locally weighted linear regression, we use raw data directly without mapping to high-dimensional features (i.e., each input is represented as a D-dimensional input vector, not an M-dimensional feature vector). Hence, we use the notation X instead of Φ. Clearly state what the R matrix is.

Proof

Let $R_0 \in \mathbb{R}^{N\times N}$ be the diagonal matrix with $r^{(i)}$ as the i-th diagonal element. Note that the residual vector is:

$$(w^\top X - y^\top)^\top = X^\top w - y$$

Then we have:

$$R_0(X^\top w - y) = \begin{bmatrix} r^{(1)}(w^\top x^{(1)} - y^{(1)}) \\ \cdots \\ r^{(N)}(w^\top x^{(N)} - y^{(N)}) \end{bmatrix}$$

Therefore the error function is:

$$\begin{aligned} E_D(w) &= \frac{1}{2}\sum_{i=1}^{N} r^{(i)}(w^\top x^{(i)} - y^{(i)})^2 \\ &= \frac{1}{2}\begin{bmatrix} (w^\top x^{(1)} - y^{(1)}) \\ \cdots \\ (w^\top x^{(N)} - y^{(N)}) \end{bmatrix}^T \begin{bmatrix} r^{(1)}(w^\top x^{(1)} - y^{(1)}) \\ \cdots \\ r^{(N)}(w^\top x^{(N)} - y^{(N)}) \end{bmatrix} \\ &= \frac{1}{2}(X^\top w - y)^T(R_0(X^\top w - y)) \\ &= \frac{1}{2}(w^\top X - y^\top)R_0(w^\top X - y^\top)^T \end{aligned}$$

We let $R := \frac{1}{2}R_0$, then we have

$$E_D(w) = (w^\top X - y^\top)R(w^\top X - y^\top)^T$$

(b) [7 points] If all $r^{(i)}$'s equal 1, the normal equation for $w \in \mathbb{R}^{D\times 1}$ becomes:

$$XX^\top w = Xy,$$

and the value of w^* that minimizes $E_D(w)$ is given by:

$$w^* = (XX^\top)^{-1}Xy.$$

Now, by finding the derivative $\nabla_w E_D(w)$ from part (a) and setting it to zero, generalize the normal equation and the closed-form solution to the locally weighted setting. Provide the new value of w^* that minimizes $E_D(w)$ in a closed form as a function of X, R, and y. (Hint: $\nabla_w(RX^\top w) = XR^\top = XR$.)

Sol.

Define

$$z := X^T w - y$$

Notice we have:

$$z = (w^\top X - y^\top)^\top$$

And we define $F(z) := z^T R z$, then we have $E(w) = F(z)$.
Here we denote the derivative of a function: $f : \mathbb{R}^n \to \mathbb{R}^m$ by $D(f)$.
By chain rule of differentiation:

$$\begin{aligned} D(E(w)) &= D(F(z))D(z(w)) && (1.11)\\ &= D_z(z^T R z)D_w(X^T w - y) && (1.12)\\ &= (2Rz)^T X^T && (1.13)\\ &= (2XRz)^T && (1.14) \end{aligned}$$

Thus

$$\nabla E(w) = D(E(w))^T = 2XRz = 2XR(X^T w - y)$$

Then setting $\nabla E(u) = 0$, we have $2XRX^T w = XRy$, so

$$w_* = \frac{1}{2}(XRX^T)^{-1} XRy$$

This is the solution.

(c) [8 points] Suppose we have a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, N\}$ of N independent examples, where the $y^{(i)}$'s are observed with differing variances. Specifically, suppose:

$$p(y^{(i)}|x^{(i)}; w) = \frac{1}{\sqrt{2\pi\sigma^{(i)}}} \exp\left(-\frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}\right),$$

i.e., $y^{(i)}$ is a Gaussian random variable with mean $w^\top x^{(i)}$ and variance $(\sigma^{(i)})^2$, where the $\sigma^{(i)}$'s are fixed, known constants. Show that finding the maximum likelihood estimate (MLE) of w reduces to solving a weighted linear regression problem $E_D(w)$. Clearly state what the $r^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.

Proof The log-likelihood for all data points is:

$$\begin{aligned} \log p(y|X; w) &= \log \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^{(i)}}} \exp\left(-\frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}\right)\\ &= \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^{(i)}}}\right) - \sum_{i=1}^{N} \frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}\\ &= -\frac{1}{2}\sum_{i=1}^{N} \log\left(2\pi\sigma^{(i)}\right) - \sum_{i=1}^{N} \frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2} \end{aligned}$$

Since the left part is constant, we have:

$$\nabla_w \log p(y|X; w) = -\nabla_w \sum_{i=1}^{N} \frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}$$

Therefore, finding the MLE of w reduces to minimizing $\nabla_w \sum_{i=1}^{N} \frac{(y^{(i)} - w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}$

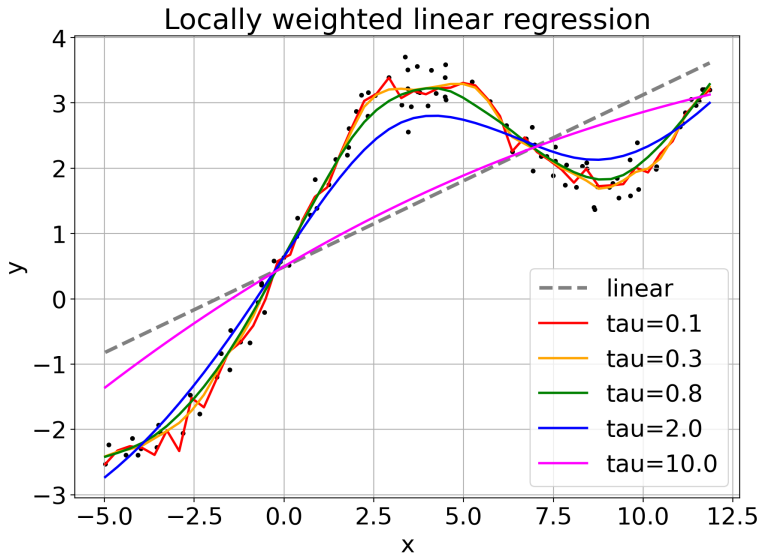
Notice that $\sum_{i=1}^{N} \frac{(y^{(i)}-w^\top x^{(i)})^2}{2(\sigma^{(i)})^2}$ is just the $E_D(w)$ in (a),(b), with each $r^{(i)} = \frac{1}{2(\sigma^{(i)})^2}$.
This reduces the problem to solving a weighted linear regression problem $E_D(w)$.

(d) [12 points, Programming Assignment] Use the files `q3x.npy`, which contains the inputs $x^{(i)}$ ($i = 1, \ldots, N$), and `q3y.npy`, which contains the outputs $y^{(i)}$ for a linear regression problem (one training example per row).

- **(i) [8 points] (Autograder)** Implement the closed-form solution for locally weighted linear regression (see the accompanied code).
- **(ii) [2 points]** Use the implemented locally weighted linear regression solver on this dataset (using the weighted normal equations derived in part (b)) and plot the data and the curve resulting from your fit. When evaluating local regression at a query point x (real-valued in this problem), use weights:

$$r^{(i)} = \exp\left(-\frac{(x - x^{(i)})^2}{2\tau^2}\right),$$

with a bandwidth parameter $\tau \in \{0.1, 0.3, 0.8, 2, 10\}$. Attach the plots generated in part (d)(ii) to your write-up.



- **(iii) [2 points]** Discuss and comment briefly on what happens to the fit when τ is too small or too large.

Sol. When τ is too small (e.g., $\tau = 0.1$), the weights become extremely localized. So the nearest training points to x contribute significantly. As a result, the model becomes overly sensitive to local variations and noise, leading to a highly fluctuating model, causing overfit.

When τ is too large (e.g., $\tau = 10.0$), the weights become almost uniform across all data points, since the exponential term decays very slowly. So the model almost reduces to becomes global linear regression. The weights then does not take effect.