

Linear Regression

Instructor: Hui Guan

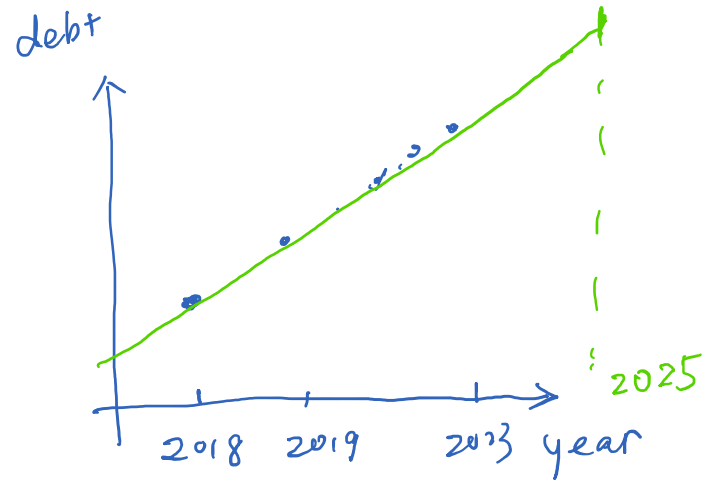
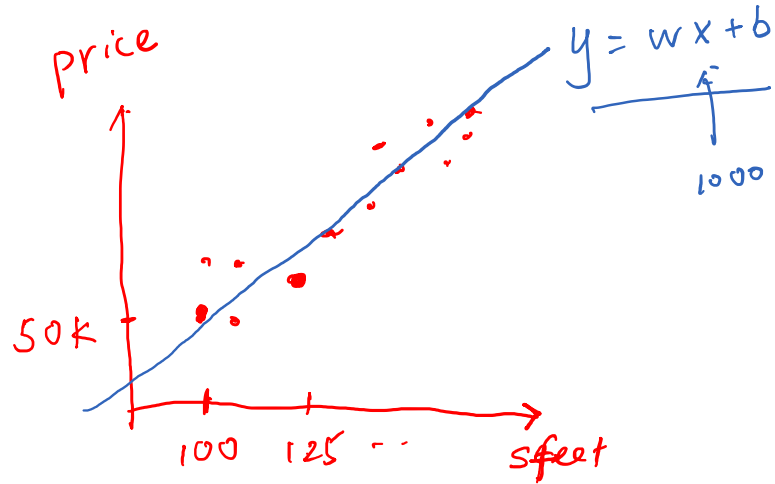
Slides adapted from: https://github.com/jermwatt/machine_learning_refined

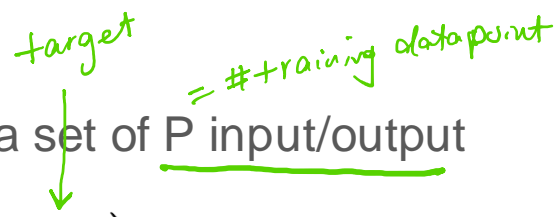

Outline

- Least Squares Linear Regression
- Least Absolute Deviations
- Regression Quality Metrics

5.2 Least Squares Linear Regression

- *linear regression* -- the fitting of a representative line (or hyperplane in higher dimensions) to a set of input/output data points.



- Data for regression problems comes in the form of a set of P input/output observation pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_P, y_P)$

- More compactly this is $\{(\mathbf{x}_p, y_p)\}_{p=1}^P$, where \mathbf{x}_p and y_p denote the p^{th} input and output respectively.


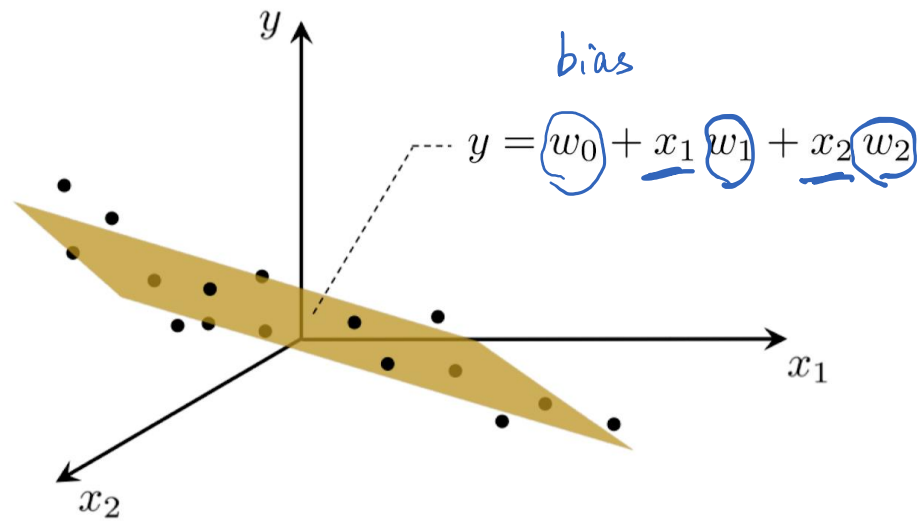
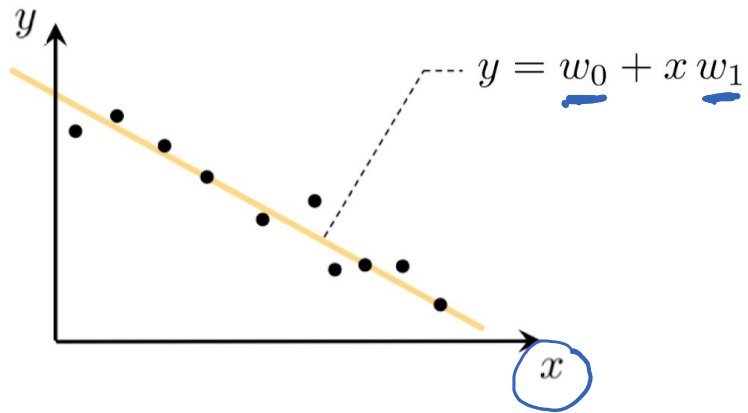
$$x \in \mathbb{R}^1 \quad y = wx + b \cdot 1 \quad [w, b] \in \mathbb{R}^2$$

$$y = \underbrace{w_1 x_1} + \underbrace{w_2 x_2} + b \cdot 1 \quad [w_1, w_2, b] \in \mathbb{R}^3$$

- In general each input \mathbf{x}_p may be a column vector of length N

$$\underline{\mathbf{x}_p} = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix} \sim p(\mathbf{x}) \quad \underbrace{\dot{\mathbf{x}} = [x_1, x_2, \dots, x_N, 1]^T}_{\in \mathbb{R}^{N+1}}$$

- In this case the linear regression problem is analogously one of fitting a hyperplane to a scatter of points in $N+1$ dimensional space.



- With scalar input we must determine appropriate vertical intercept w_0 and w_1 slope so that the following holds

$$\text{model } \boxed{w_0 + x_p w_1} \approx \overset{\text{target}}{\boxed{y_p}}, \quad p = 1, \dots, P.$$

- Notice that we have used the approximately equal sign because we cannot be sure that all data lies completely on a single line.
- More generally, when dealing with N dimensional input we have a bias and N associated slope weights to tune properly

$$\boxed{w_0 + x_{1,p} w_1 + x_{2,p} w_2 + \dots + x_{N,p} w_N} \approx y_p, \quad p = 1, \dots, P.$$

$\vec{x}_p^T \vec{w}$: inner product y_p

- Note: **each dimension of the input** is referred to in the jargon of machine learning as **a feature** or **input feature**.
- So we will often refer to $w_{1,p}, w_{2,p}, \dots, w_{N,p}$ as the **feature-touching weights** (the only weight *not* touching a feature is the **bias** w_0).

"weights" : { feature touching + bias ✓ ← "parameters"
 { feature touching

- For any \mathbf{N} we can write the above more compactly - in particular using the notation $\dot{\mathbf{x}}$ to denote an input \mathbf{x} with a 1 placed on top of it as

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \quad \dot{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

The diagram illustrates the compact notation for a weighted sum. The weight vector \mathbf{w} is shown as a column vector with elements $w_0, w_1, w_2, \dots, w_N$. The augmented input vector $\dot{\mathbf{x}}$ is shown as a column vector with elements $1, x_1, x_2, \dots, x_N$. A blue arrow points from the 1 in $\dot{\mathbf{x}}$ to the w_0 in \mathbf{w} , with the word "bias" written above it. A blue bracket groups the elements x_1, x_2, \dots, x_N in $\dot{\mathbf{x}}$.

- In particular, this means that we stack a **1** on top of each of our input points \mathbf{x}_p as

$$\boxed{\dot{\mathbf{x}}_p} = \begin{bmatrix} 1 \\ x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}, \quad p = 1, \dots, P$$

$$\min \underbrace{g(w) = \dots w \dots}_{\uparrow \uparrow}$$

- We may then write our desired linear relationships more compactly as

model target/output

Goal: $\boxed{\dot{\mathbf{x}}_p^T \mathbf{w}} \approx \boxed{y_p} \quad p = 1, \dots, P.$

$\checkmark \quad \underbrace{(\dot{\mathbf{x}}_p^T \mathbf{w} - y_p)^2}_{\checkmark}$

$\underbrace{|\dot{\mathbf{x}}_p^T \mathbf{w} - y_p|}_{\checkmark} \leq \checkmark$

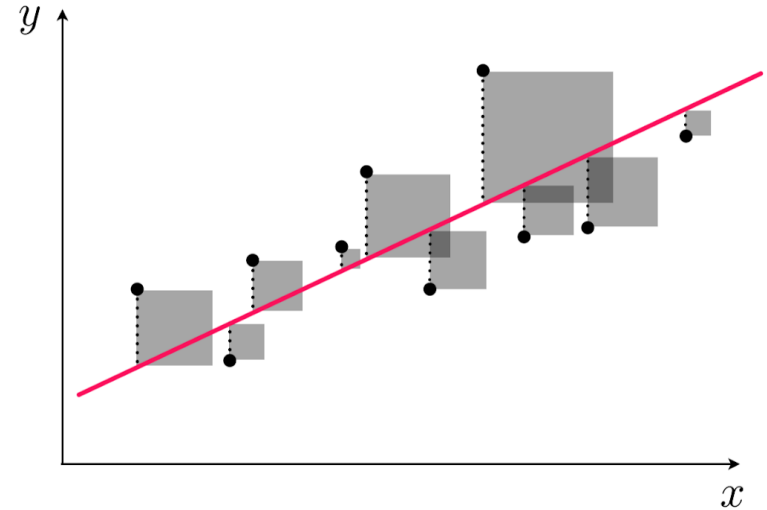
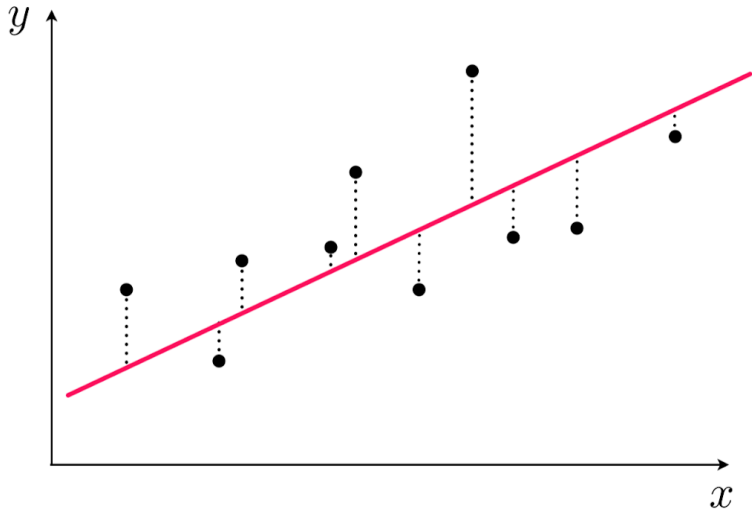
$\underbrace{(\dot{\mathbf{x}}_p^T \mathbf{w} - y_p)^4}_{\checkmark}$

Handwritten notes:

- An arrow points from the word "unknown" to the \mathbf{w} term in the goal equation.
- A green oval encloses a vertical sequence of three points labeled y , with a green letter d between the top and bottom points. A blue arrow points from the top point to the bottom point.
- Below the green oval is a blue vertical line with three points labeled y , with a blue arrow pointing from the top point to the bottom point.

The Least Squares cost function

- To find the parameters of the hyperplane which best fits a regression dataset, it is common practice to first form the *Least Squares cost function*.
- For a given set of parameters \mathbf{w} this cost function **computes the total squared error between the associated hyperplane and the data**.
- Naturally then the best fitting hyperplane is the one whose parameters minimize this error.



- The Least Square error: $J(w) = \sum_{p=1}^P J_p(w)$ P : # data point

$$J_p(\mathbf{w}) = (\mathbf{x}_p^T \mathbf{w} - y_p)^2.$$

- This squared error $J_p(\cdot)$ is one example of a **point-wise cost** that measures the error of a model (here a linear one) on the point $\{\mathbf{x}_p, y_p\}$

$$P = 1000$$

↑

\vec{x}_p : 200 times

20%

$\alpha_p \uparrow$ credibility \uparrow

$\alpha_p \downarrow$

- Since we want all P such values to be small we can take their average - forming a **Least Squares cost function** for linear regression

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P g_p(\mathbf{w}) = \left(\frac{1}{P} \right) \sum_{p=1}^P \alpha_p (\dot{\mathbf{x}}_p^T \mathbf{w} - y_p)^2$$

- Note that the Least Squares cost function is not just a function of the weights \mathbf{w} , but of the data as well.
- However, when we express the function in mathematical shorthand as $g(\mathbf{w})$ we only show dependency on the weights \mathbf{w} .

- We want to tune our parameters \mathbf{w} to *minimize* the Least Squares cost...

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{P} \sum_{p=1}^P \left(\dot{\mathbf{x}}_p^T \mathbf{w} - y_p \right)^2$$

$g(\mathbf{w})$ "ground truth / output / target"

```
def model( x )  
    ...  
    return  $\hat{y}$ 
```

```
def cost(  $\hat{y}$ , y )  
    ...  
    return cost
```

Implementing the Least Squares cost in Python

```
def optimizer( ... )  
    ...  
    return  $w^*$ 
```

$$\text{model}(\mathbf{x}_p, \mathbf{w}) = \dot{\mathbf{x}}_p^T \mathbf{w}.$$

```
# compute linear combination of input point  
def model(x_p,w):  
    # compute linear combination and return  
    a = w[0] + np.dot(x_p.T,w[1:])  
    return a.T
```

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P (\text{model}(\mathbf{x}_p, \mathbf{w}) - y_p)^2.$$

```
# a least squares function for linear regression
def least_squares(w,x,y):
    # loop over points and compute cost contribution from each input/output pair
    cost = 0
    for p in range(y.size):
        # get pth input/output pair
        x_p = x[:,p][:,np.newaxis]
        y_p = y[p]

        ## add to current cost
        cost += (model(x_p,w) - y_p)**2

    # return average least squares error
    return cost/float(y.size)
```

- However when such for loops can be equivalently written using `numpy` operations, this is typically far more efficient.
- *Explicit* `for` loops (including list comprehensions) written in `Python` are rather slow due to the very nature of the language (e.g., it being a dynamically typed interpreted language)

- It is easy to get around most of this inefficiency by replacing explicit `for` loops with numerically equivalent operations performed using operations from the `numpy` library.
- `numpy` is an API for some very efficient vector/matrix manipulation libraries written in `C`. In fact `Python` code, employing heavy use of `numpy` functions, can often execute almost as fast as a raw `C` implementation itself.

$$\text{model}(\mathbf{x}_p, \mathbf{w}) = \dot{\mathbf{x}}_p^T \mathbf{w}.$$

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P (\text{model}(\mathbf{x}_p, \mathbf{w}) - y_p)^2.$$

```
# compute linear combination of input points
```

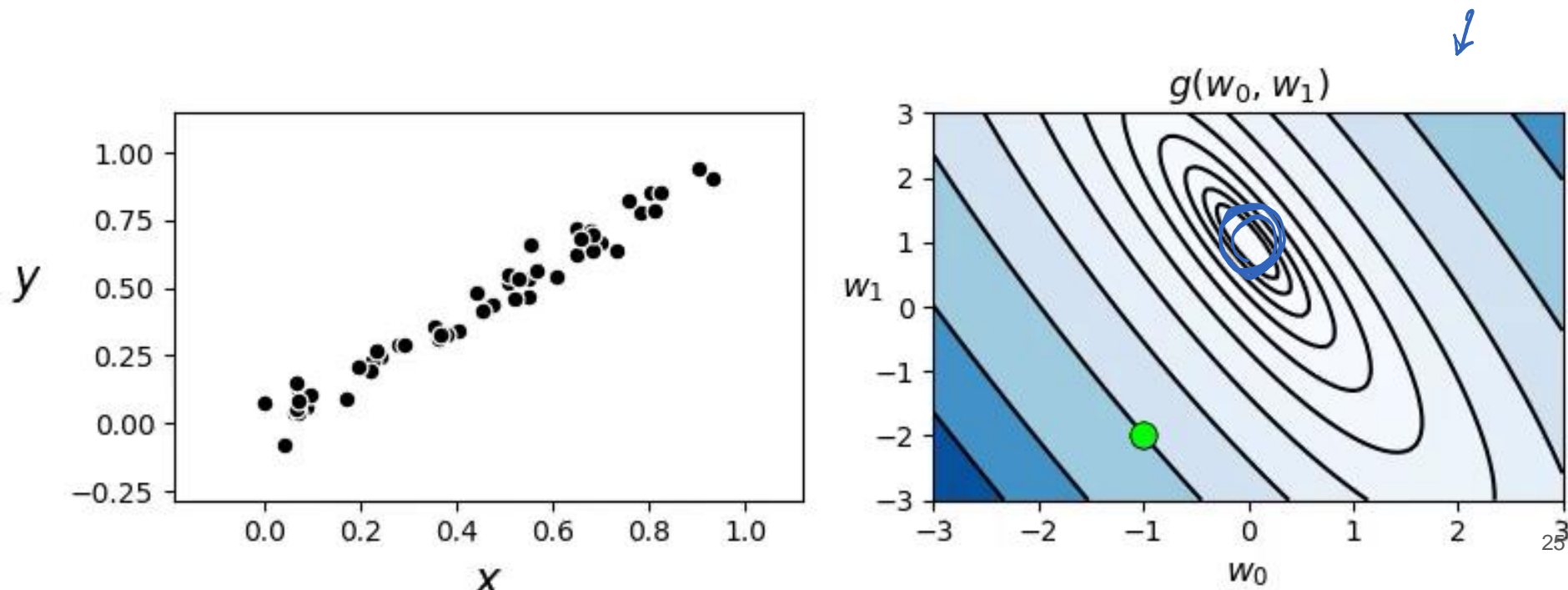
```
def model(x,w):  
    a = w[0] + np.dot(x.T,w[1:])  
    return a.T
```

```
# an implementation of the least squares cost function for linear regression
```

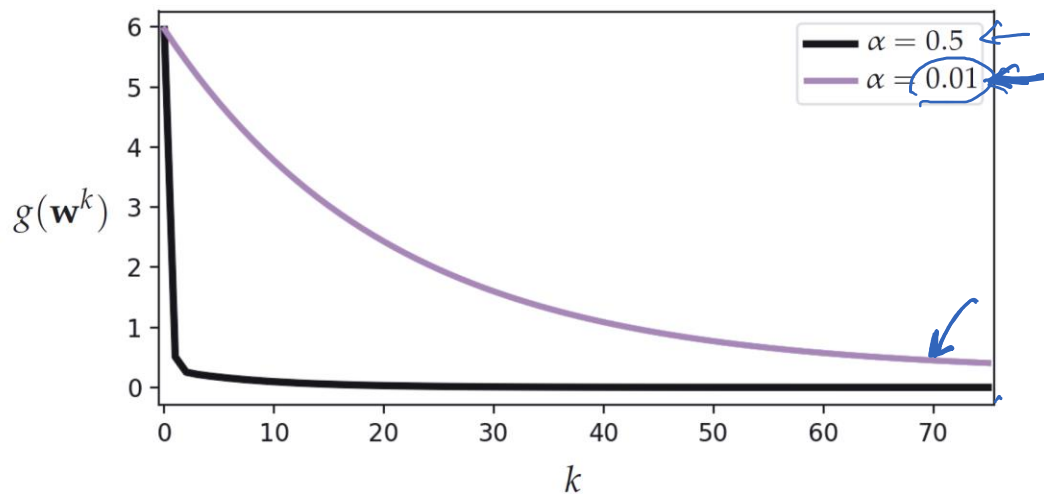
```
def least_squares(w):  
    # compute the least squares cost  
    cost = np.sum((model(x,w) - y)**2)  
    return cost/float(y.size)
```

Example: Using gradient descent to minimize the Least Squares cost on our toy dataset

- Below we minimize the Least Squares cost using a toy dataset.
- We use gradient descent and employ a fixed steplength value $\alpha = 0.5$ for all 75 steps until approximately reaching the minimum of the function.



- Whenever we use a local optimization method like gradient descent we must properly tune the steplength parameter α .
- Below we re-create those runs using $\alpha = 0.5, \alpha = 0.01$, showing the the cost function history plot for each steplength value choice. We can see from the plot that indeed the first steplength value works considerably better.



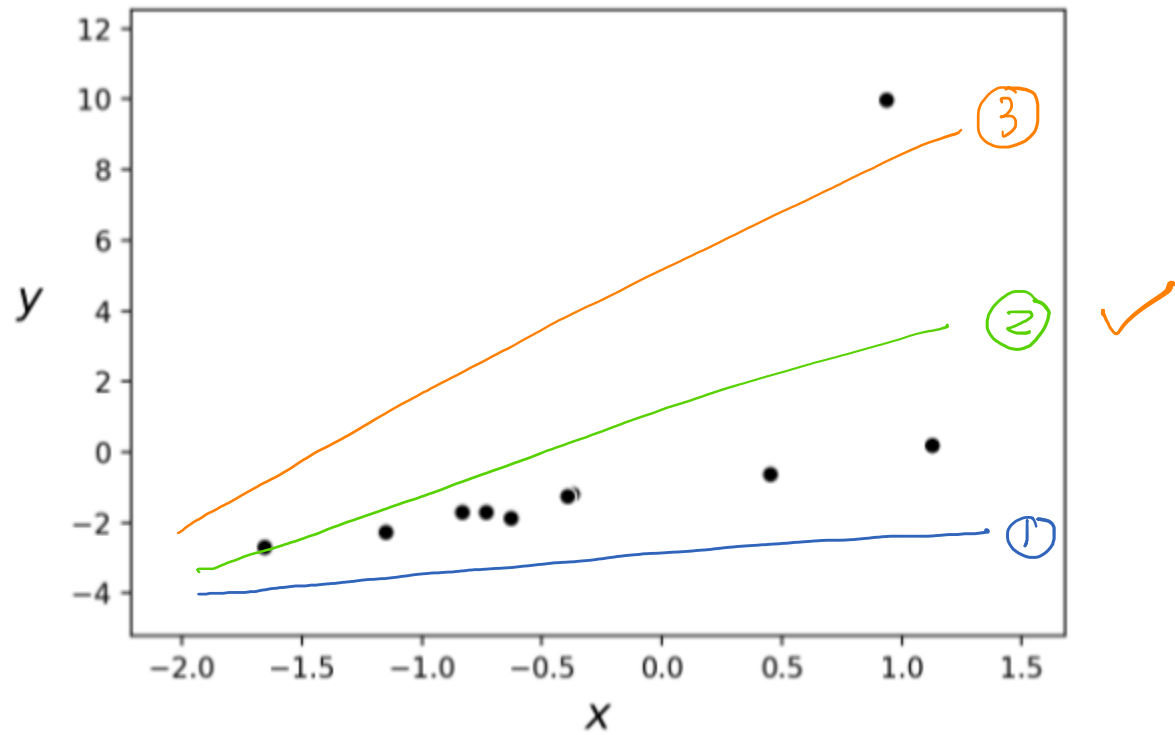
5.3 Least Absolute Deviations

- A slight twist on the derivation of the Least Squares cost function that leads to an alternative cost for linear regression called *Least Absolute Deviations*.
- This alternative cost function is much **more robust to outliers** in a dataset than the original Least Squares.

The susceptibility of the Least Squares cost to outliers

- One downside of using the squared error in the Least Squares cost is that ***squaring the error increases the importance of larger errors.***
- In particular squaring errors of length greater than **1** makes these values considerably larger.
- This forces weights learned via the Least Squares cost to produce a linear fit that is especially focused on trying to minimize these large errors.
- This is often due to *outliers* in a dataset.

Example: Least Squares overfits to outliers





Replacing squared error with absolute error

- How can we make our linear regression framework more robust to outliers?
- Remember our aim in learning a linear regressor is to learn a set of ideal weights so that

$$\dot{\mathbf{x}}_p^T \mathbf{w} \approx y_p \quad p = 1, \dots, P$$

- To learn these ideal weights the first step we took there was to square the difference between as

$$g_p(\mathbf{w}) = (\dot{\mathbf{x}}_p^T \mathbf{w} - y_p)^2 \quad p = 1, \dots, P.$$

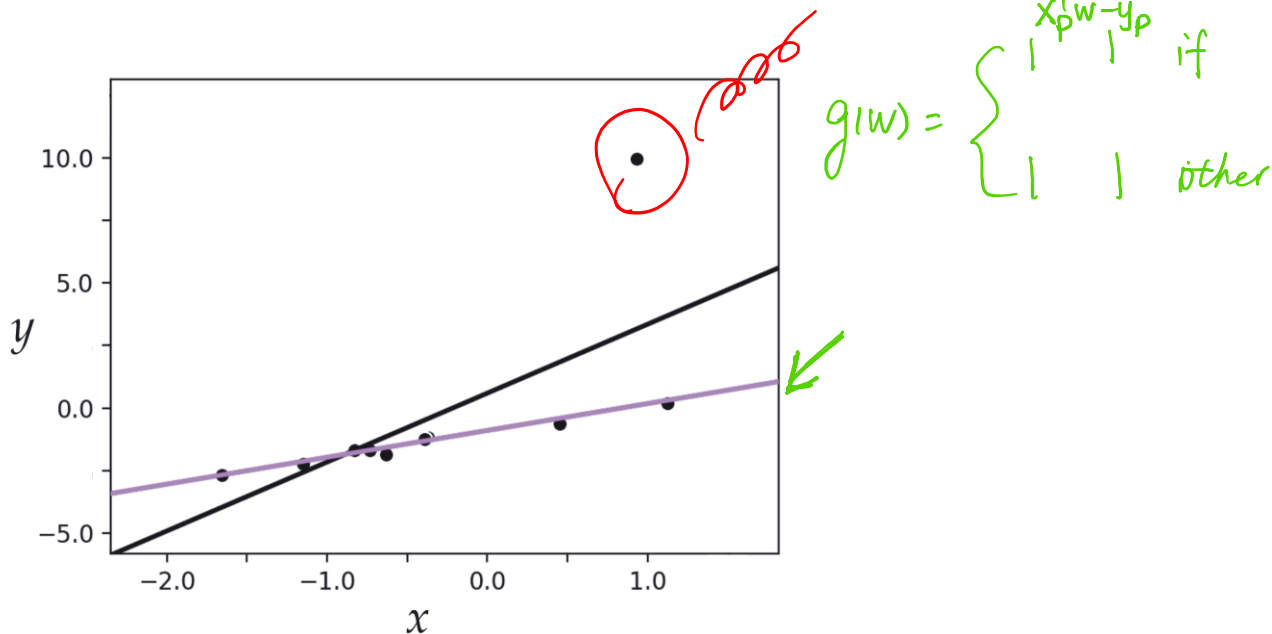
- As an alternative to using a *squared* error for our point-wise cost we can instead measure the *absolute error* as

$$g_p(\mathbf{w}) = |\dot{\mathbf{x}}_p^T \mathbf{w} - y_p| \quad p = 1, \dots, P.$$

- By using absolute error instead of the squared variety we still treat negative and positive errors equally.

- If we form the average of these absolute error point-wise costs we have the so-called *Least Absolute Deviations* cost function

$$g(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P g_p(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P |\mathbf{x}_p^T \mathbf{w} - y_p|.$$



5.4 Regression Quality Metrics

- Plug in our **trained model** and **dataset** into the Least Squares cost - giving the *Mean Squared Error* (or *MSE* for short) of our trained model

$$\text{MSE} = \frac{1}{P} \sum_{p=1}^P \left(\text{model}(\mathbf{x}_p, \mathbf{w}^*) - y_p \right)^2.$$

well-trained parameters

- The name for this quality measurement describes precisely what the Least Squares cost computes - the average (or *mean*) squared error.

- In the same way we can employ the Least Absolute Deviations cost to determine the quality of our trained model.
- Plugging in our trained model and dataset into this cost computes the **Mean Absolute Deviations** (or *MAD* for short).
- This is precisely what this cost function computes

$$\text{MAD} = \frac{1}{P} \sum_{p=1}^P \left| \text{model}(\mathbf{x}_p, \mathbf{w}^*) - y_p \right|.$$

- Of course in general the *lower* one can make a quality measures, by proper tuning of model weights, the *better* the quality of the corresponding trained model (and vice versa).
- However the threshold for what one considers 'good' or 'great' performance can depend on personal preference, an occupational or institutionally set benchmark, or some other problem-dependent concern.

Quiz

Top 30 Linear Regression Interview Questions & Answers for Data Scientists (Updated 2023)

<https://www.analyticsvidhya.com/blog/2017/07/30-questions-to-test-a-data-scientist-on-linear-regression/>

Q4. Which of the following methods do we use to find the best-fit line for data in Linear Regression?

- A) Least Square Error
- B) Maximum Likelihood
- C) Logarithmic Loss
- D) Both A and B

Q4. Which of the following methods do we use to find the best-fit line for data in Linear Regression?

- A) Least Square Error
- B) Maximum Likelihood
- C) Logarithmic Loss
- D) Both A and B

Solution: (A)

In linear regression, we try to minimize the least square errors of the model to identify the line of best fit.

Q8. Suppose we have N independent variables ($X_1, X_2 \dots X_n$) and Y 's dependent variable.

Now Imagine that you are applying linear regression by fitting the best-fit line using the least square error on this data. You found that the correlation coefficient for one of its variables (Say X_1) with Y is -0.95.

Which of the following is true for X_1 ?

↑
feature touching weight

- A) Relation between the X_1 and Y is weak
- ☒ B) Relation between the X_1 and Y is strong
- C) Relation between the X_1 and Y is neutral
- D) Correlation can't judge the relationship

Q8. Suppose we have N independent variables ($X_1, X_2 \dots X_n$) and Y 's dependent variable.

Now Imagine that you are applying linear regression by fitting the best-fit line using the least square error on this data. You found that the correlation coefficient for one of its variables (Say X_1) with Y is -0.95 .

Which of the following is true for X_1 ?

- A) Relation between the X_1 and Y is weak
- B) Relation between the X_1 and Y is strong
- C) Relation between the X_1 and Y is neutral
- D) Correlation can't judge the relationship

Solution: (B)

The absolute value of the correlation coefficient denotes the strength of the relationship. Since the absolute correlation is very high, we infer that the relationship is strong between X_1 and Y .

Q18. Which of the following statement is true about outliers in Linear regression?

- A) Linear regression is sensitive to outliers
- B) Linear regression is not sensitive to outliers
- C) Can't say
- D) None of these

Q18. Which of the following statement is true about outliers in Linear regression?

- A) Linear regression is sensitive to outliers
- B) Linear regression is not sensitive to outliers
- C) Can't say
- D) None of these

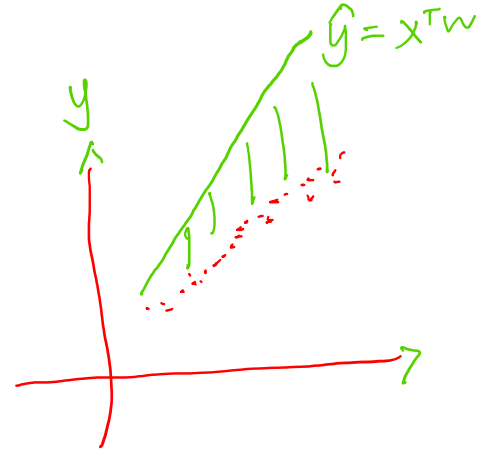
Solution: (A)

The slope of the regression line will change due to outliers in most cases. So Linear Regression is sensitive to outliers.

$$|x^T w - y| \quad \hat{y} = x^T w$$

Q19. Suppose you plotted a scatter plot between the residuals and predicted values in linear regression and found a relationship between them. Which of the following conclusion do you make about this situation?

- A) Since there is a relationship means our model is not good
- B) Since there is a relationship means our model is good
- C) Can't say
- D) None of these



Q19. Suppose you plotted a scatter plot between the residuals and predicted values in linear regression and found a relationship between them. Which of the following conclusion do you make about this situation?

- A) Since there is a relationship means our model is not good
- B) Since there is a relationship means our model is good
- C) Can't say
- D) None of these

Solution: (A)

There should not be any relationship between predicted values and residuals. If there exists any relationship between them, it means that the model has not perfectly captured the information in the data points.



given me an example where linear regression beats deep neural networks



While deep neural networks (DNNs) are often powerful for complex tasks, there are situations where a simple linear regression model can outperform DNNs. Here's an example:



Example: Predicting Housing Prices

Suppose you have a dataset of housing prices, and you want to predict the price of a house based on its size (in square feet). Here's how linear regression can be more effective in this specific scenario:

- ✓ 1. **Linearity of the Relationship:** In this case, it's reasonable to assume that there is a linear relationship between the size of a house and its price. As the size of the house increases, the price tends to increase proportionally. Linear regression is well-suited for modeling such linear relationships.
- ✓ 2. **Simplicity and Interpretability:** Linear regression provides a straightforward and interpretable model. The coefficients (weights) associated with the input features (e.g., size) directly indicate the impact of each feature on the target variable (price). This transparency can be valuable for understanding the relationship between the input and output.
- ✓ 3. **Data Availability:** If you have a relatively small dataset with limited examples, using a simple linear regression model can help avoid overfitting, which is a common issue with complex DNNs when data is scarce. Linear regression is less prone to overfitting compared to deep networks.
- ✓ 4. **Computational Efficiency:** Linear regression is computationally efficient and quick to train, making it a good choice when you want to build a predictive model rapidly.

Summary

- Linear regression
- Least squares cost vs. least absolute deviation costs
- MSE, MAD

- Concepts: feature/input feature, feature-touching weights, bias, outliers