Machine Learning

Linear Regression & Gradient Descent

Kien C Nguyen

August 21, 2024

Supervised learning

Recall from Lecture 1 that, in the predictive or **supervised learning** approach, given an input vector $\mathbf{x}$, we have to predict an output y, where y can be categorical or numerical.

- The goal is to learn a mapping from inputs $\mathbf{x}$ to outputs y, given a labeled set of input-output pairs $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$.
- Here D is called the training set, and N is the number of training examples.
- when y_i is categorical, the problem is known as classification or pattern recognition. For example, the problem of classifying emails into 'spam' and 'not spam'.
- **When y_i is real-valued (numerical), the problem is called regression. For example, the problem of predicting the income level.**
- Another variant, known as ordinal regression, occurs where label space Y has some natural ordering, such as grades A–F.

Some example applications

- Predict the cost (in USD) of a taxi ride in New York City given the features: pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count (Lecture 1)
- Predict tomorrow's closed price of a stock given current market conditions, fundamental indicators, and technical indicators of the stock price.
- Predict the salary of a graduate given the major, the GPA, social activities, and demographic features (such as gender, age, and location).
- Predict the price of a house given the location, area, frontage, number of bedrooms, and the number of bathrooms.
- In time series analysis, the value of the time series at time t, V_t can be written as a linear combination of the values in previous time points ($V_{t-1}, V_{t-2}, \ldots$).

Advertising dataset

The Advertising dataset consists of the sales of a particular product in 200 different markets, and advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper. Everything is given in units of $1000.

TV	radio	newspaper	sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

Figure: Advertising dataset ("An Introduction to Statistical Learning, with applications in R" (Springer, 2013))

Problem formulation

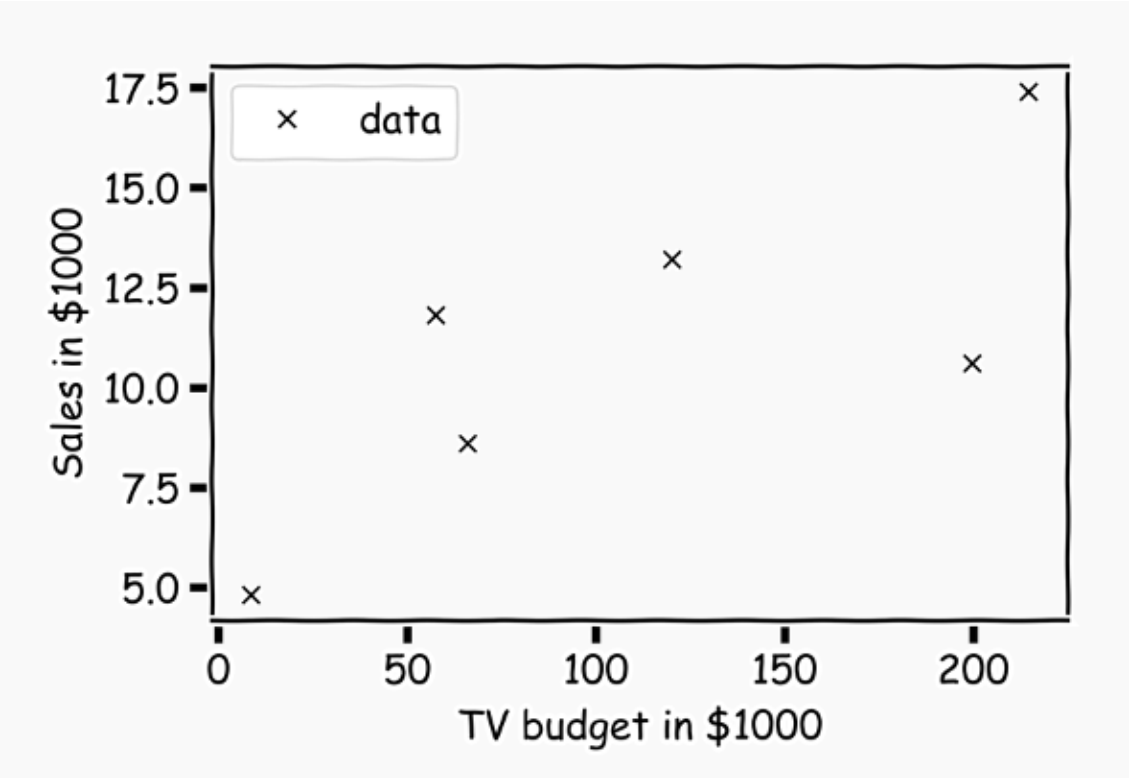
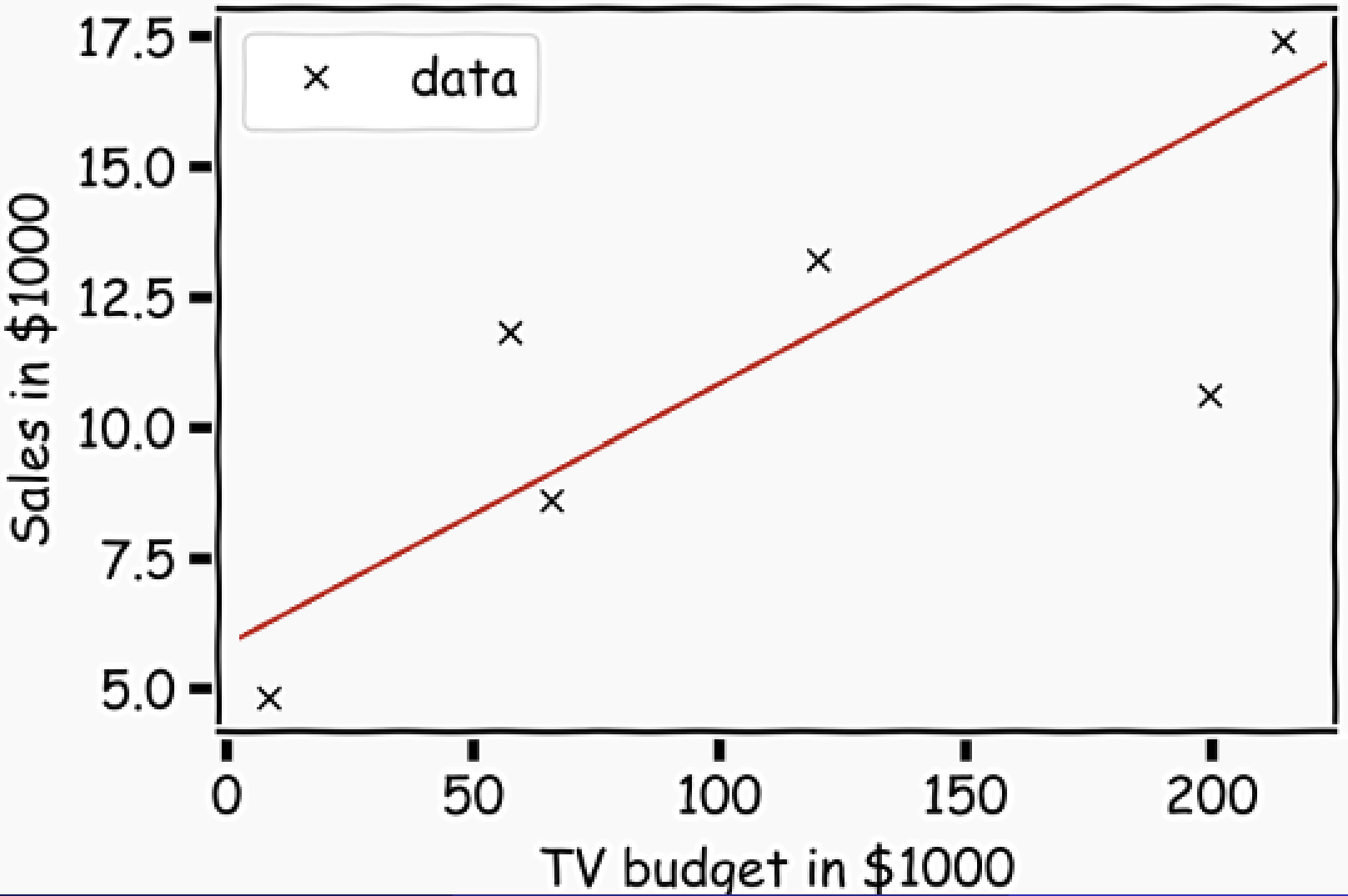


Figure: Annual sales vs TV advertisement budget

- Suppose we plot a company's sales versus its TV advertisement budget in previous year
- We want to be able to estimate the sales given a TV advertisement budget
- We want to build a model $y = f(x)$.

One possible model

If we choose y to be a linear function of x, one possible function is shown below



Another possible model

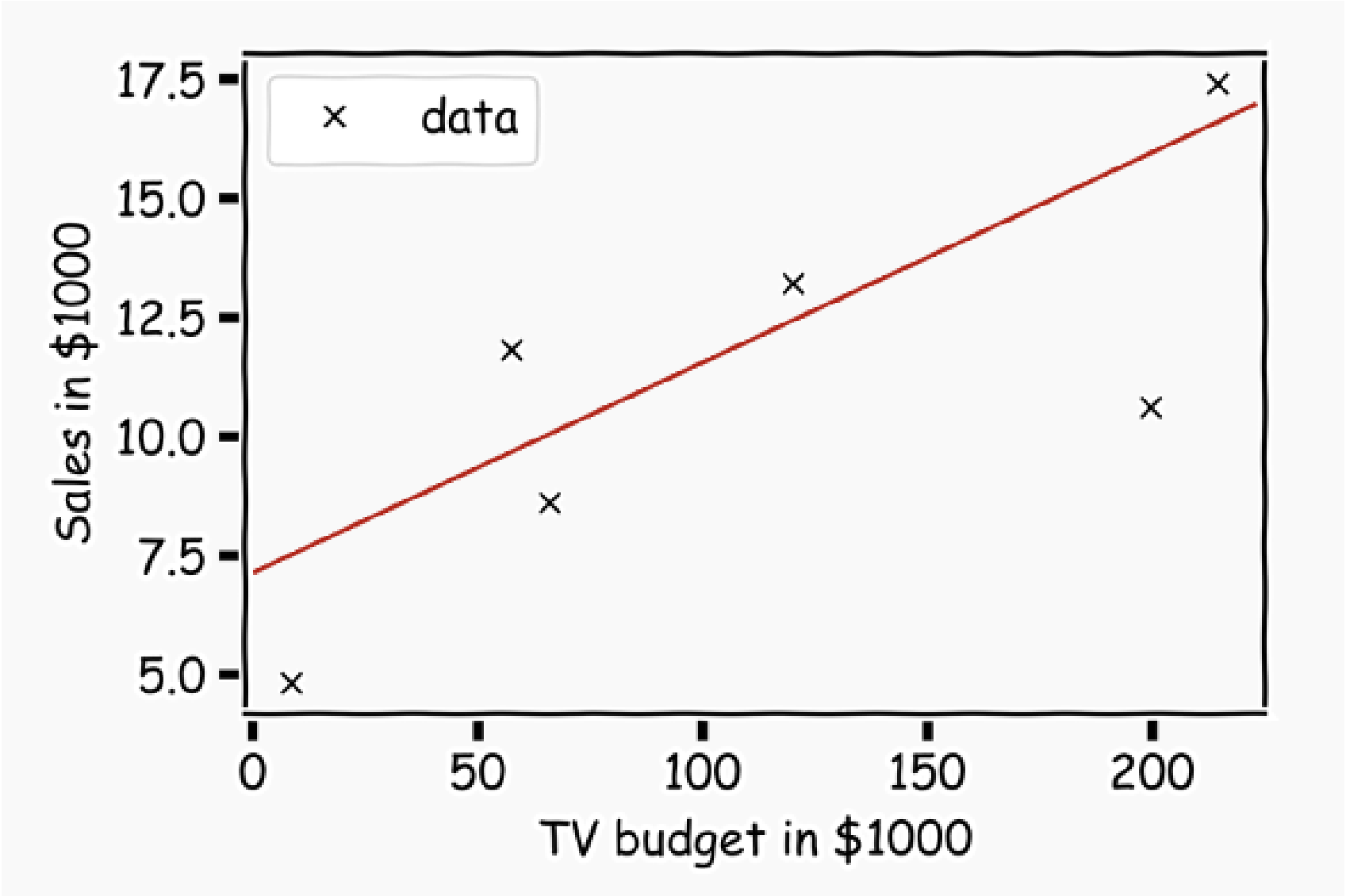


Figure: Another possible model

Yet another possible model

Figure: Yet another possible model

Residuals/ errors aggregation

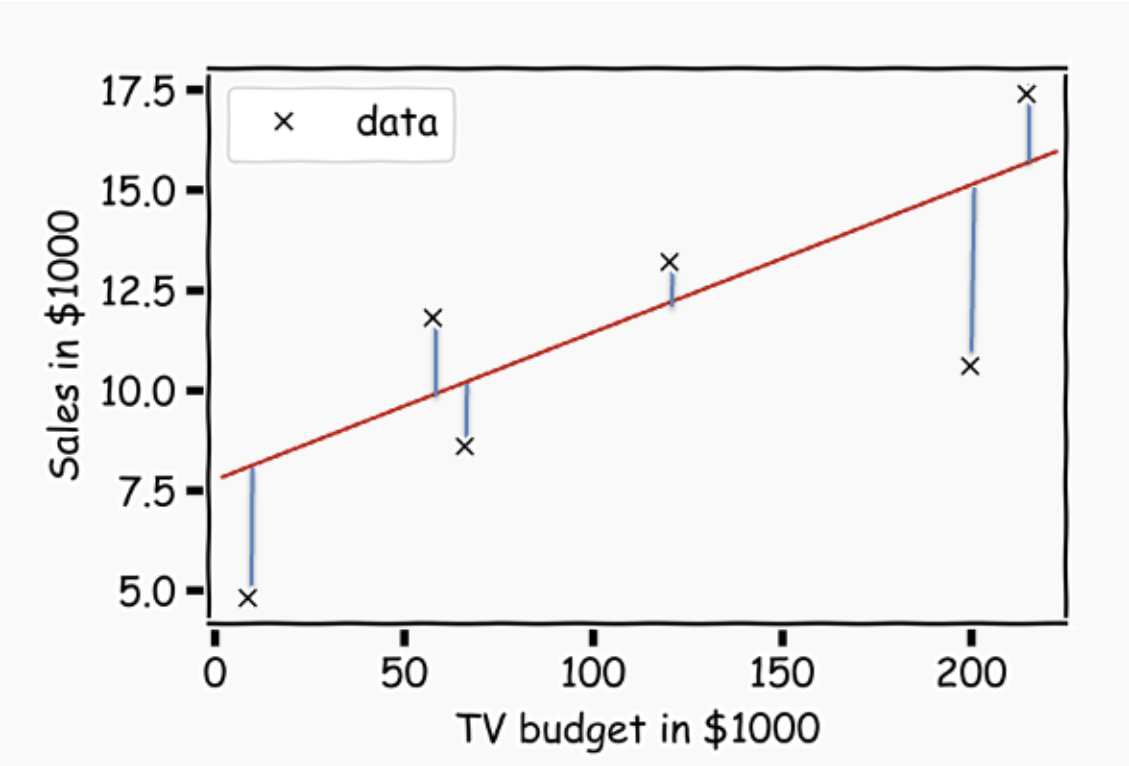


Figure: Residual aggregation

How do we aggregate the residuals/ errors?

- Max Absolute Error
- Mean Absolute Error
- Mean Squared Error
- Mean Absolute Percentage Error

Linear model for regression

We can build a model by first assuming a simple form of f :

$$f(x) = \beta_0 + \beta_1 X$$

We then estimate $\widehat{Y} = \widehat{f}(X) = \widehat{\beta}_1 X + \widehat{\beta}_0$ where $\hat{\beta}_1$ and $\hat{\beta}_0$ are estimates of β_1 and β_0 respectively, that we compute using observations.

Linear model for regression

Linear model for regression is a linear combination of the input variables. It assumes the dependency of the response variable y on the explanatory variables $\mathbf{x}$ is linear.

Formula

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + ... + w_D x_D = w_0 + \sum_{j=1}^{D} w_j x_j$$

where

- $y \in \mathbf{R}$: response variable, dependent variable, outcome.
- D: number of dimensions of the input vector $\mathbf{x}$.
- $\mathbf{x} = (x_1, ..., x_D)^T$: input vector (explanatory variable, independent variable, features).
- $\mathbf{w} = (w_0, ..., w_D)$: parameters.
- $D + 1$: total number of parameters.

Loss function

Given the features $\mathbf{x}$, the predicted value of y, $\hat{y}$, is given by $\hat{y} = f(\mathbf{x}) = w_0 + \sum_{j=1}^{D} w_j x_j$

Loss function

A loss function is a measure of how good a prediction model does in terms of being able to predict the expected outcome. If we use Mean Squared Error with the constant changed from $1/N$ to $1/2$ for easier manipulation later on.

$$L(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where N is the number of training examples.

Our goal: find parameters $\mathbf{w}$ that minimize the loss function. How?

Ordinary Least Squares

We have

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

where we let $x_0^{(i)} = 1$ to simplify the notation.

Our goal is to find $\hat{\mathbf{w}}$:

$$\hat{\mathbf{w}} = argmin_{\mathbf{w}} L(\mathbf{w}) = argmin_{\mathbf{w}} \left(\frac{1}{2}||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2\right)$$

Explanation of matrices & matrix equations

$$\mathbf{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,D} \\ 1 & x_{2,1} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \dots & x_{n,D} \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

$$\hat{\mathbf{Y}} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,D} \\ 1 & x_{2,1} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \dots & x_{n,D} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{pmatrix}$$

Ordinary Least Squares (OLS)

$$
\begin{aligned}
L(\mathbf{w}) &= \frac{1}{2}(y - \mathbf{X}\mathbf{w})^T(y - \mathbf{X}\mathbf{w}) \\
&= \frac{1}{2}\left(y^T y - 2\mathbf{w}^T\mathbf{X}^T y + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}\right).
\end{aligned}
$$

Setting the gradient to 0:

$$
\begin{aligned}
\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} &= -\mathbf{X}^T y + \mathbf{X}^T\mathbf{X}\mathbf{w} = 0 \\
&\iff \mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T y \\
&\iff \mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y.
\end{aligned}
$$

Notes:

- The Hessian in this case is $\mathbf{X}^T\mathbf{X}$, which is a positive semidefinite matrix.
- The matrix $\mathbf{X}^T\mathbf{X}$ must be invertible and difficult to scale with high dimension input vector.
- The case in which $\mathbf{X}^T\mathbf{X}$ is non-invertible will be addressed later.

Gradient Descent algorithm

Gradient descent algorithm

Initialize $\mathbf{w} = [0, ..., 0]$;

for $t = 1, ..., T$ **do**

$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$

end

- η: learning rate
- $\nabla L(\mathbf{w})$: gradient

Cons: requires the entire set of data samples to be loaded in memory, since it operates on all of them at the same time

Stochastic Gradient Descent algorithm

Stochastic Gradient Descent algorithm

Initialize $\mathbf{w} = [0, ..., 0]$;
for $t = 1, ..., T$ **do**
  for $(x, y) \in D_{train}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(x, y, \mathbf{w})$
  end
end

- Pros: during learning, compute $L(x, y, \mathbf{w})$ before updating $\mathbf{w}$, so require less memory.
- Cons: requires a number of hyperparameters such as the regularization parameter and the number of iterations.

Gradient Descent (GD)

Pros:

1. Stable Convergence: GD tends to converge smoothly towards the minimum since it uses the entire dataset to compute the gradient.
2. Deterministic: Given the same starting point, it will always produce the same path to the minimum because the gradient is calculated over the whole dataset.
3. Batch Processing: Takes advantage of matrix operations which can be optimized efficiently with certain hardware accelerations.

Cons:

1. Computationally Expensive: Computing the gradient over the entire dataset can be very slow and computationally expensive, especially with large datasets.
2. Memory Intensive: Requires loading the entire dataset into memory for each iteration, which can be impractical with large datasets.
3. Scalability Issues: Not suitable for very large datasets due to its high computational cost and memory requirements.

Stochastic Gradient Descent (SGD) - Pros

1. Faster Iterations: Since SGD updates the model parameters more frequently with each training example, it can be significantly faster in terms of the number of iterations.
2. Lower Memory Usage: Requires only a single training example (or a small batch) for each iteration, making it more memory-efficient.
3. Better for Large Datasets: Scales well with large datasets because it processes one sample (or a small batch) at a time.
4. Avoids Local Minima: The stochastic nature helps in escaping local minima, potentially finding a better global minimum.

Stochastic Gradient Descent (SGD) - Cons

1. Noisy Convergence: The updates are noisier and less stable, leading to more erratic paths towards the minimum. This can result in longer convergence times.
2. Hyperparameter Sensitivity: Requires careful tuning of hyperparameters like the learning rate and batch size to ensure convergence.
3. Less Accurate Gradient: Since it uses only one sample (or a small batch), the gradient estimate is less accurate compared to GD.
4. More Hyperparameter Tuning: Often requires more effort in tuning and sometimes additional mechanisms like learning rate schedules or momentum to ensure stable convergence.

Mini-Batch Gradient Descent (a compromise)

Mini-Batch Gradient Descent combines aspects of both GD and SGD. It updates the model parameters based on a small batch of training examples.

Pros:

1. Balanced Trade-off: Offers a balance between the fast convergence of SGD and the stable updates of GD.
2. Efficient Computation: Can take advantage of matrix operations while being faster and less memory-intensive than GD.
3. Reduced Variance: Reduces the noise in parameter updates compared to pure SGD.

Cons:

1. Still Requires Tuning: Requires tuning of the batch size and learning rate.
2. Complexity: Adds some complexity in implementation compared to pure GD or SGD.

Gradient Descent

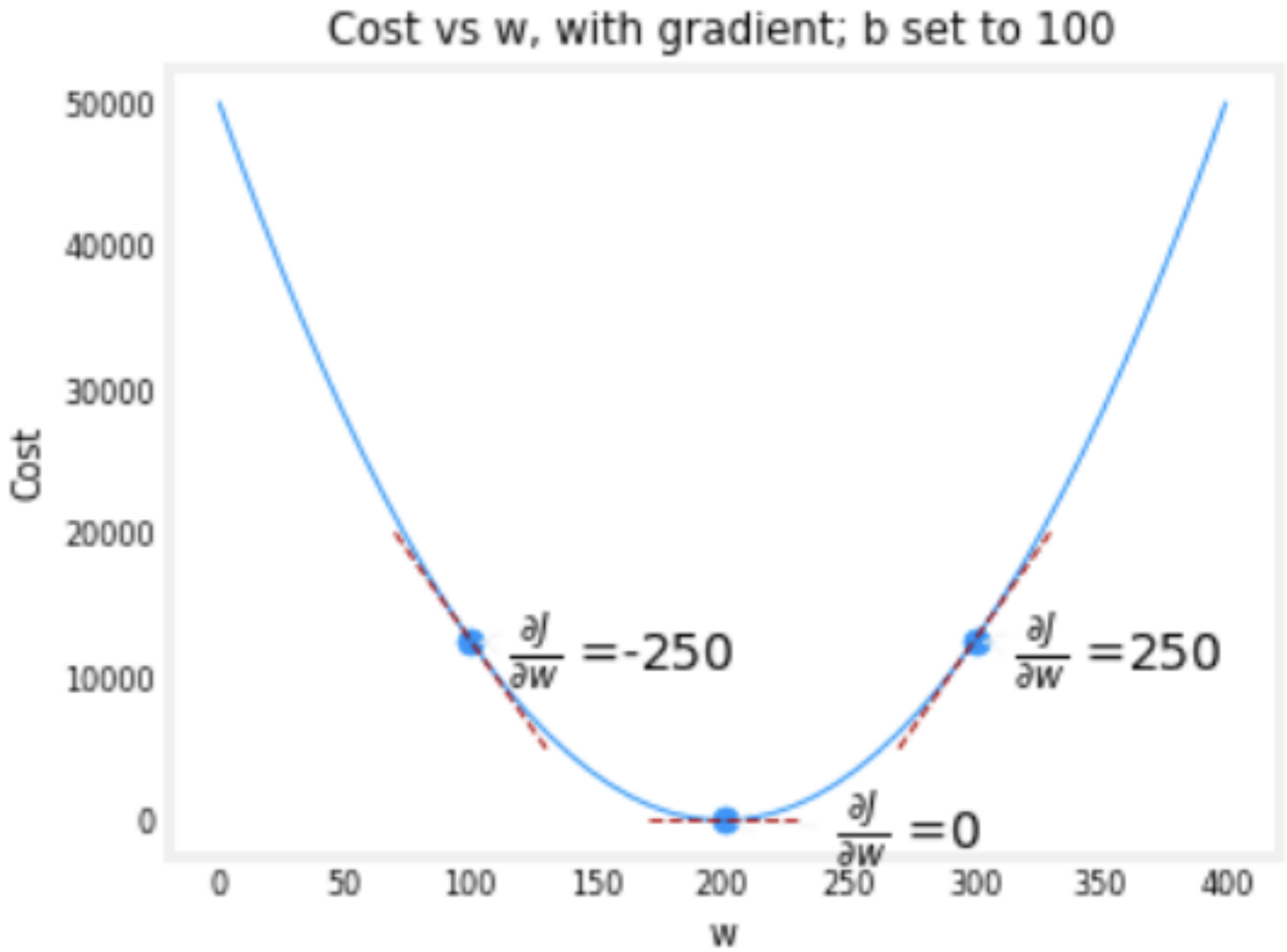


Figure: Gradient Descent (Coursera - Machine Learning Specialization)

Gradient Quiver Plot

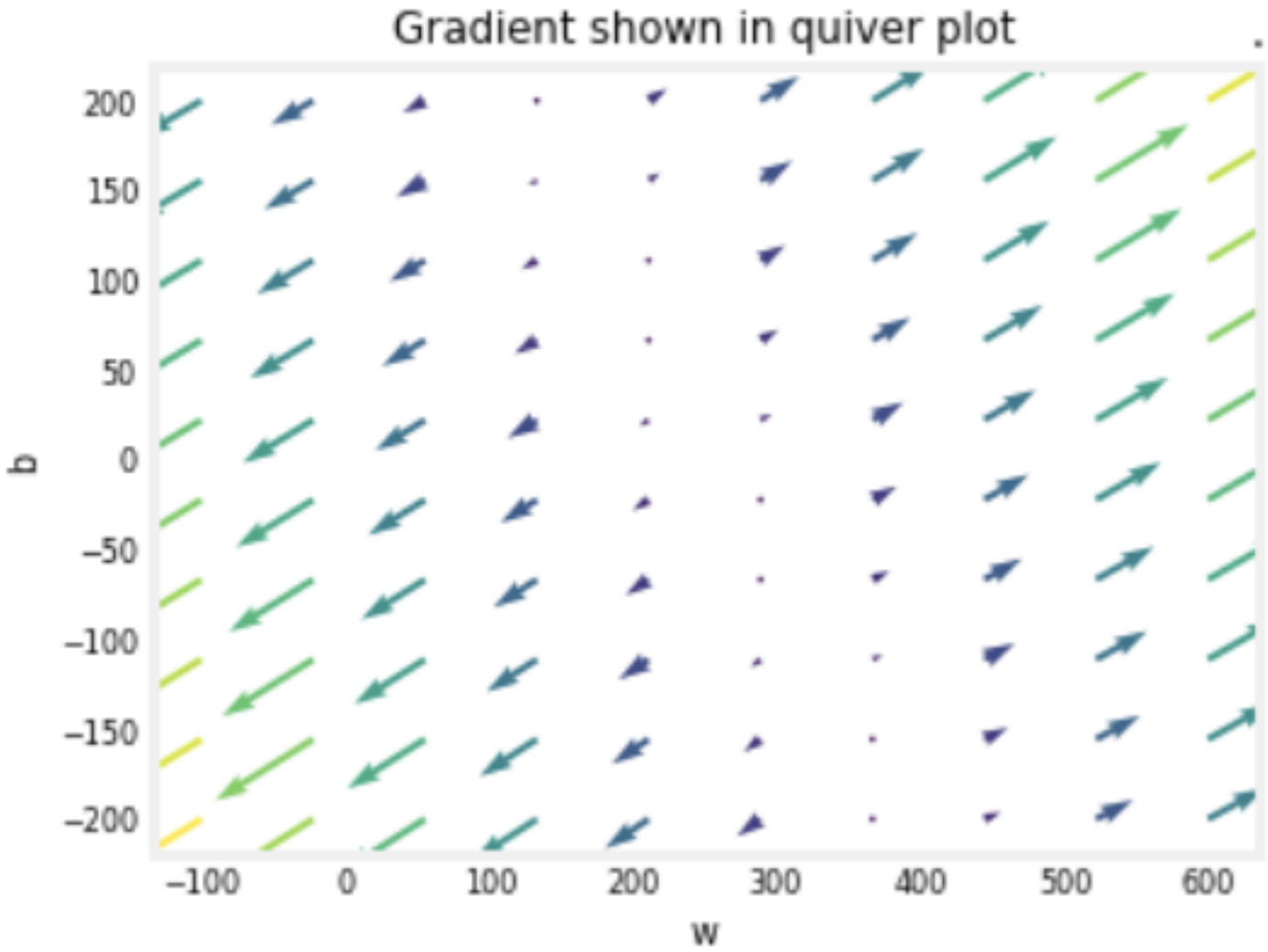


Figure: Gradient Quiver Plot (Coursera - Machine Learning Specialization)

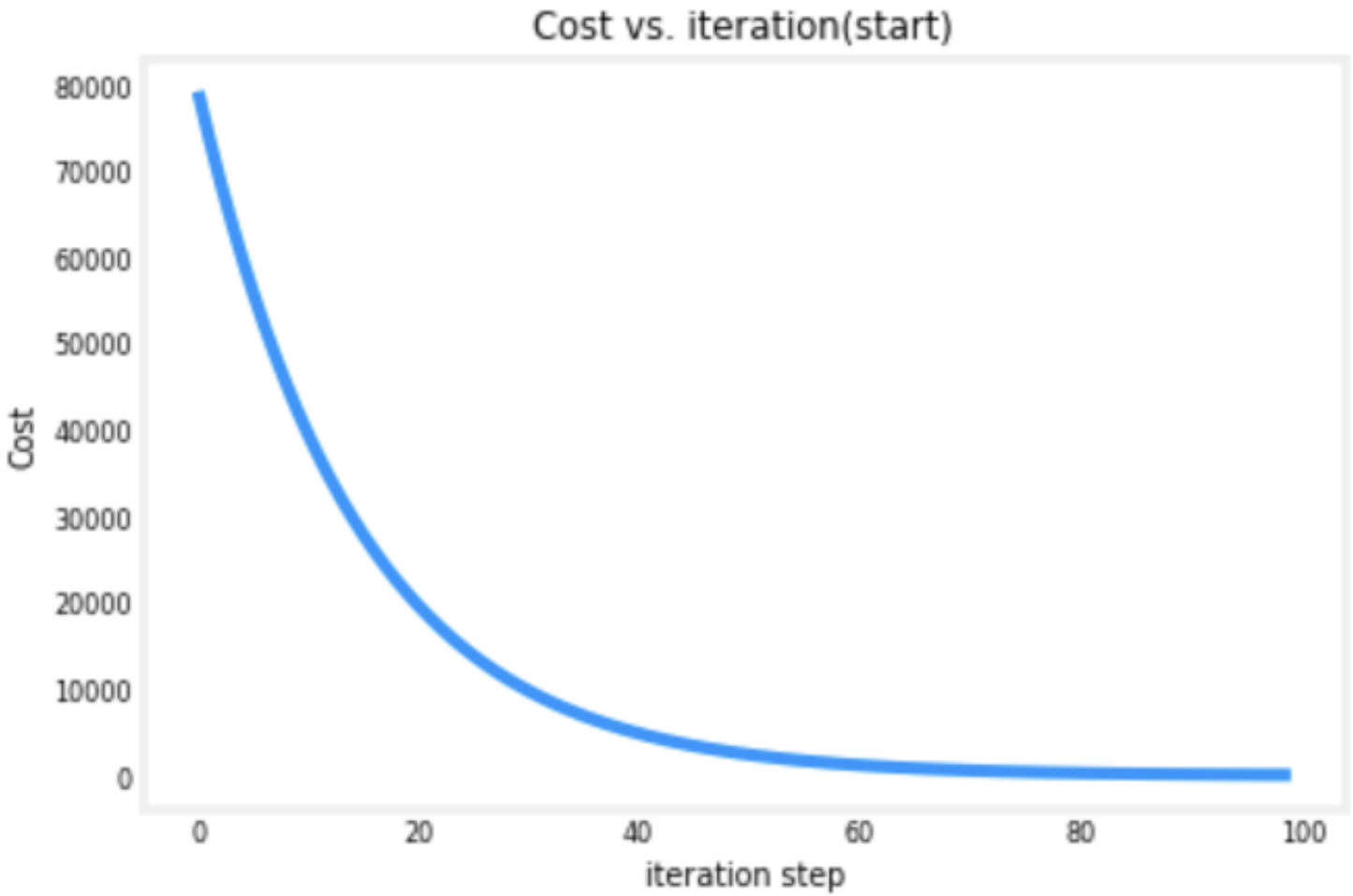


Figure: Cost vs Iteration in the first 100 iterations (Coursera - Machine Learning Specialization)

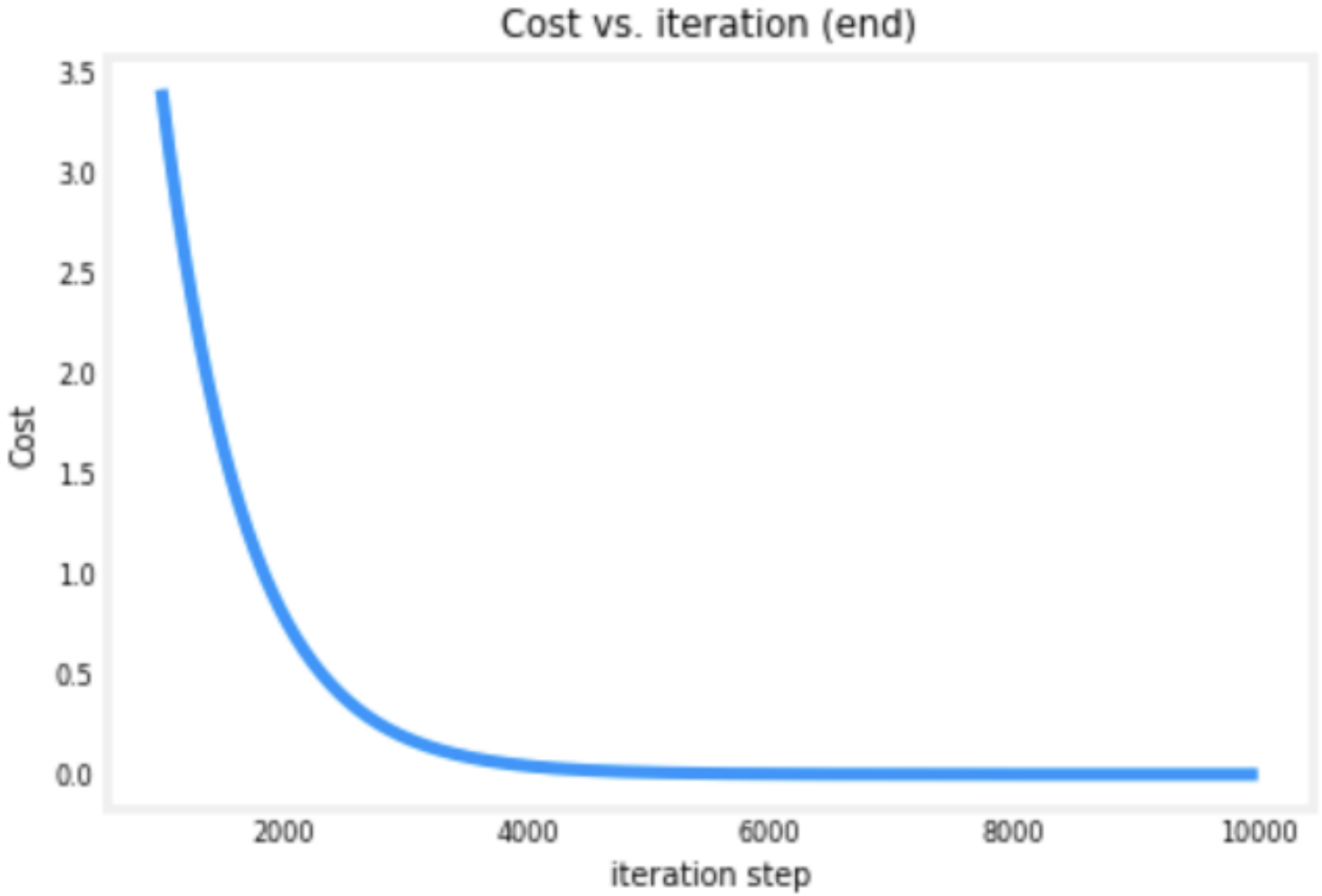


Figure: Cost vs Iteration in the remaining iterations (Coursera - Machine Learning Specialization)

Contour plot of the loss function

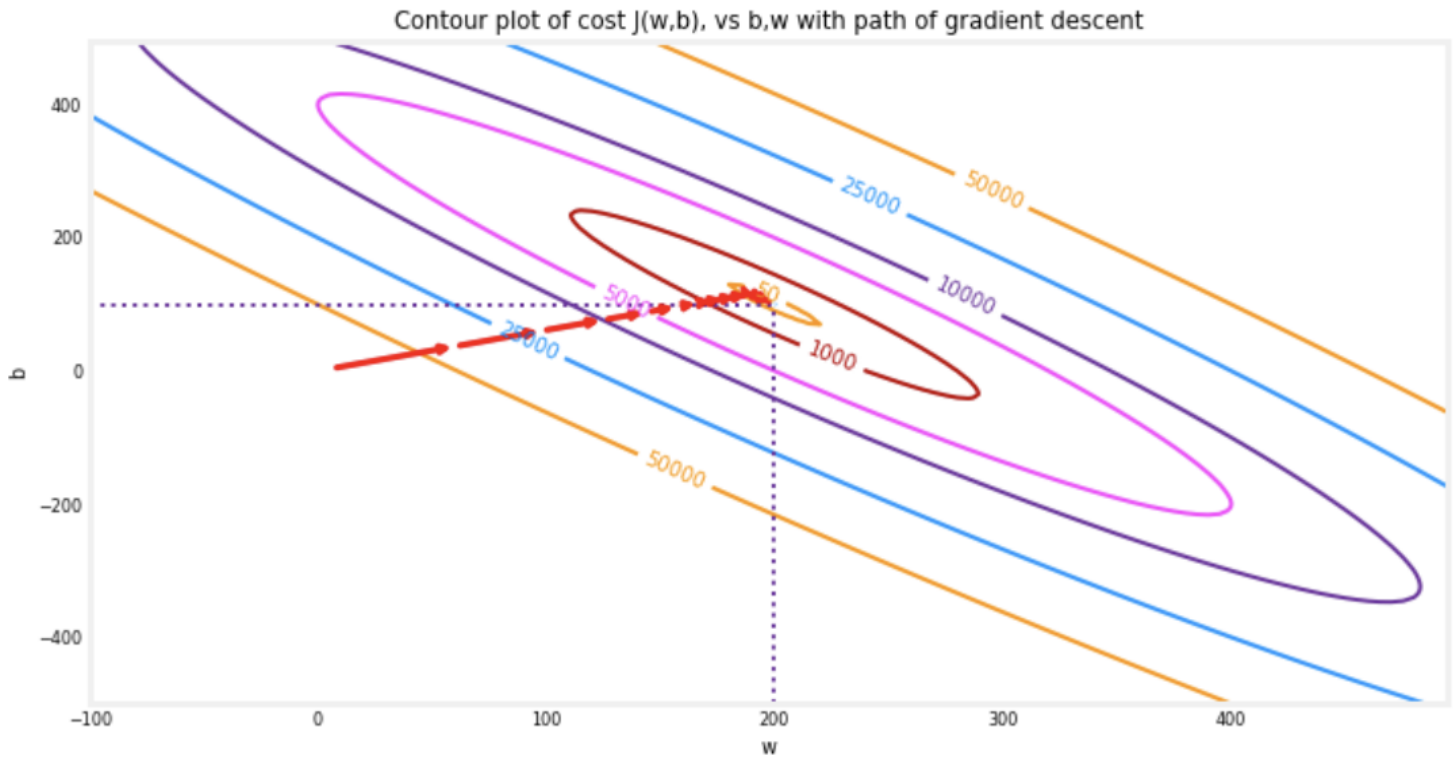


Figure: Contour plot of the loss function (Coursera - Machine Learning Specialization)

Cost escalation due to large learning rates

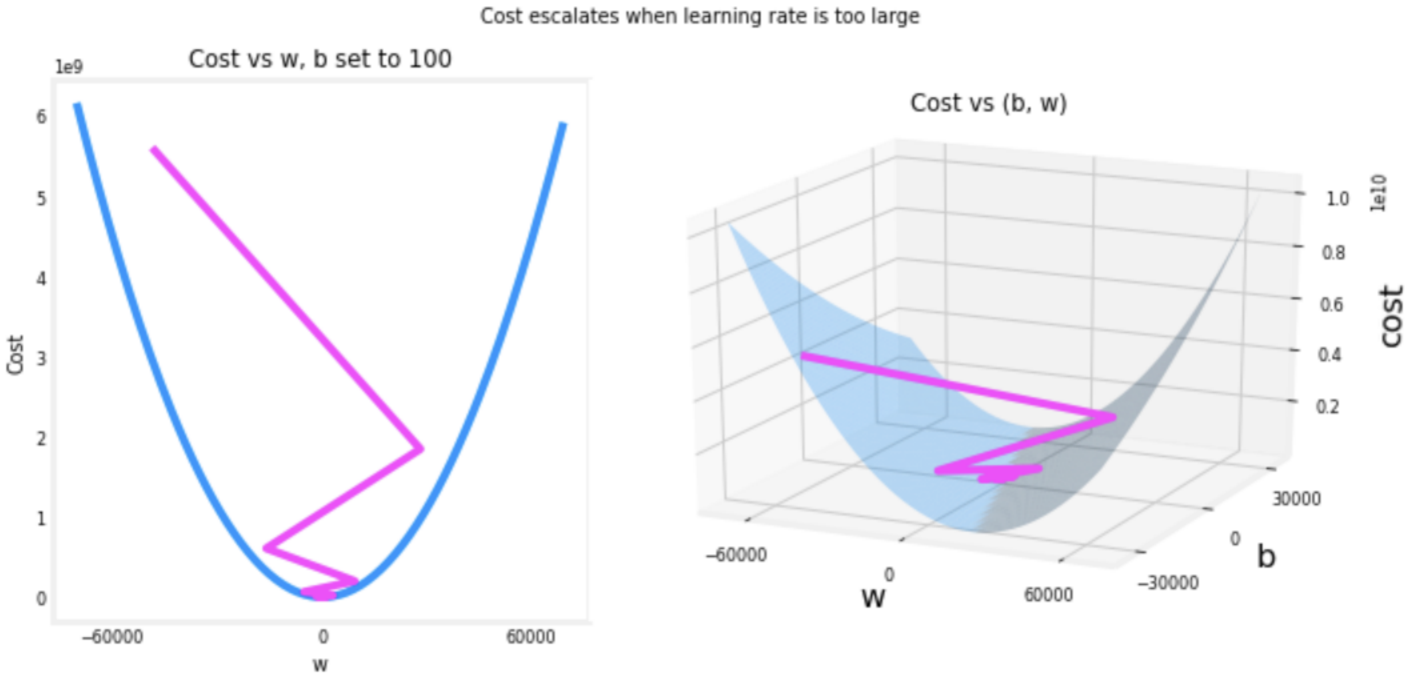


Figure: Cost escalation due to large learning rates (Coursera - Machine Learning Specialization)

How about datasets with nonlinear relationships?

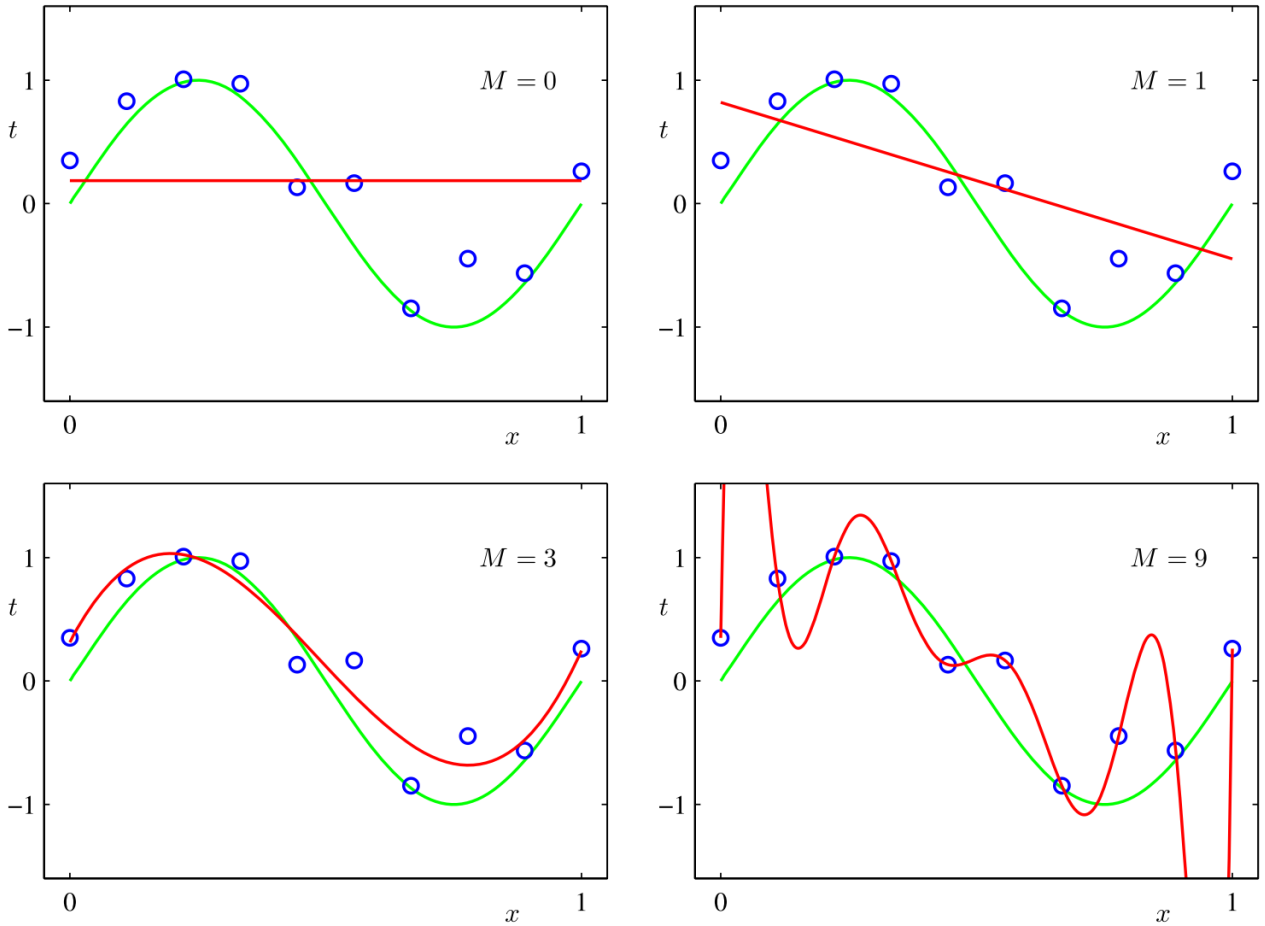


Figure 1.4 Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

Figure: Plots of polynomials having various orders M, shown as red curves, fitted to the dataset (Bishop, Pattern Recognition and Machine Learning)

Basis function

Extend the class of models by considering linear combinations of fixed **nonlinear functions** of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where $\phi_j(\mathbf{x})$ are known as basis functions. Identity "basis function" is $\phi(\mathbf{x}) = \mathbf{x}$.

Some basis functions

Polynomial basis function

$$\phi_j(x) = x^j$$

Gaussian basis function

$$\phi_j(x) = exp\{-\frac{(x-\mu_j)^2}{2s^2}\}$$

Sigmoidal basis function

$$\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$$

where $\sigma(a)$ is the logistic sigmoid function defined by

$$\sigma(a) = \frac{1}{1+exp(-a)}$$

Some basis functions

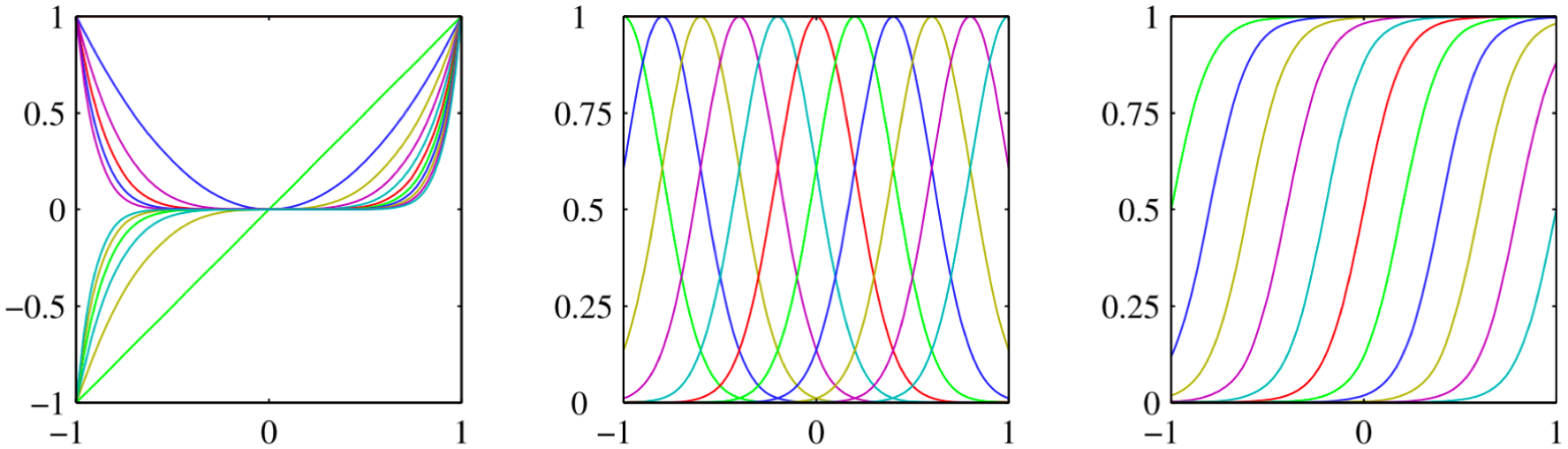


Figure 3.1 Examples of basis functions, showing polynomials on the left, Gaussians of the form (3.4) in the centre, and sigmoidal of the form (3.5) on the right.

Figure: Examples of basis functions, showing polynomials on the left, Gaussians in the centre, and sigmoidal on the right (Bishop, Pattern Recognition and Machine Learning)

Assumptions of the (Multiple) Linear Regression Model

Formula

$$y^{(i)}(\mathbf{x}^{(i)}, \mathbf{w}) = w_0 + \sum_{j=1}^{D} w_j x_j^{(i)} + \epsilon^{(i)}$$

- The relationship between the dependent variable (y) and the independent variables ($x_j,\ j = 1, \ldots, D$) is linear.
- The independent variables ($x_j,\ j = 1, \ldots, D$) are not random. There is no exact linear relation between two or more of the independent variables (multi-collinearity).
- The expected value of the error term, conditioned on the independent variables, is 0. $E\left[\epsilon^{(i)} | x^{(i)}\right] = 0$
- The variance of the error term is the same for all observations $E\left[\left(\epsilon^{(i)}\right)^2\right] = \sigma_\epsilon^2$ (homoscedasiticity).
- The error term is uncorrelated across observations $E\left[\epsilon^{(i)}\epsilon^{(j)}\right] = 0,\ i \neq j$
- The error term is normally distributed.

LASSO, Ridge, and Elastic Net

In order to prevent overfitting for Linear Regression models, we can use various regularization techniques:

- Purpose: Prevent overfitting by adding a penalty to the model complexity.
- Key Methods:
  - LASSO (Least Absolute Shrinkage and Selection Operator):
    - Adds a penalty equal to the absolute value of the coefficients.
    - Can set some coefficients to zero, performing feature selection.
  - Ridge Regression:
    - Adds a penalty equal to the square of the coefficients.
    - Shrinks coefficients but retains all predictors.
  - Elastic Net:
    - Combines LASSO and Ridge penalties.
    - Balances between feature selection and coefficient shrinkage.

Least Absolute Shrinkage and Selection Operator (LASSO)

- Penalty: $\lambda \sum_{j=1}^{D} |\beta_j|$
- Effect: Can reduce some coefficients to zero, effectively performing variable selection.
- Formula:

$$\text{Minimize} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{D} |w_j| \right\}$$

- Use Case: When you expect only a few predictors to be relevant.

Ridge Regression

- Penalty: $\lambda \sum_{j=1}^{D} \beta_j^2$
- Effect: Shrinks coefficients, but none are exactly zero, thus retaining all predictors.
- Formula:

$$\text{Minimize} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{D} w_j^2 \right\}$$

- Use Case: When you expect all predictors to have a small effect.

Elastic Net

- Penalty: $\lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$
- Effect: Combines LASSO and Ridge penalties to balance between variable selection and coefficient shrinkage.
- Formula:

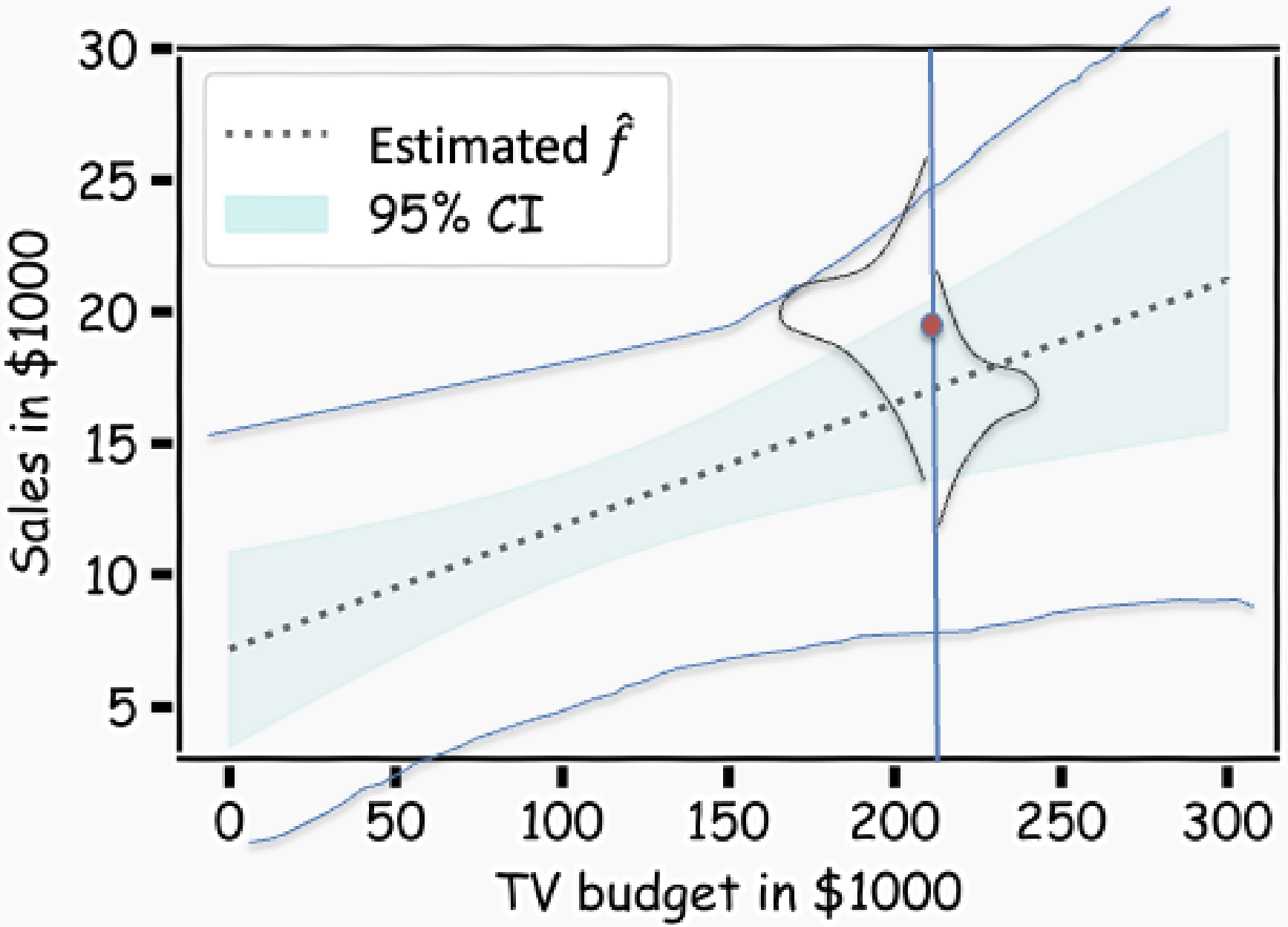
$$\text{Minimize} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \right\}$$

- Use Case: When you have multiple correlated predictors.

Confidence Interval in predicting $\hat{y}$

- If we consider the training dataset to be just a sample of a larger population of training examples, the Linear Regression model will become an estimation problem where we estimate the weights $w_i, i = 0, \ldots D$ of the model.
- Larger numbers of training examples will reduce the standard errors of the estimations.
- We can also calculate the Confidence Interval (such as 95% Confidence Interval) of each weight.
- Detailed treatments of this topic can be found in the references.

Confidence Interval in predicting $\hat{y}$



References

[1] Bishop, C. M. (2013). Pattern Recognition and Machine Learning. Journal of Chemical Information and Modeling (Vol. 53).
[2] Wikipedia. Gradient descent, Ordinary least squares, Stochastic gradient descent.
[3] Andrew Ng – Machine Learning (Coursera) – https://www.coursera.org/learn/machine-learning
[4] Havard University CS109a: Introduction to Data Science