CPEN 400D: Deep Learning

Lecture 1 (II): ML Basics & Linear Models

Renjie Liao

University of British Columbia

Winter, Term 2, 2022

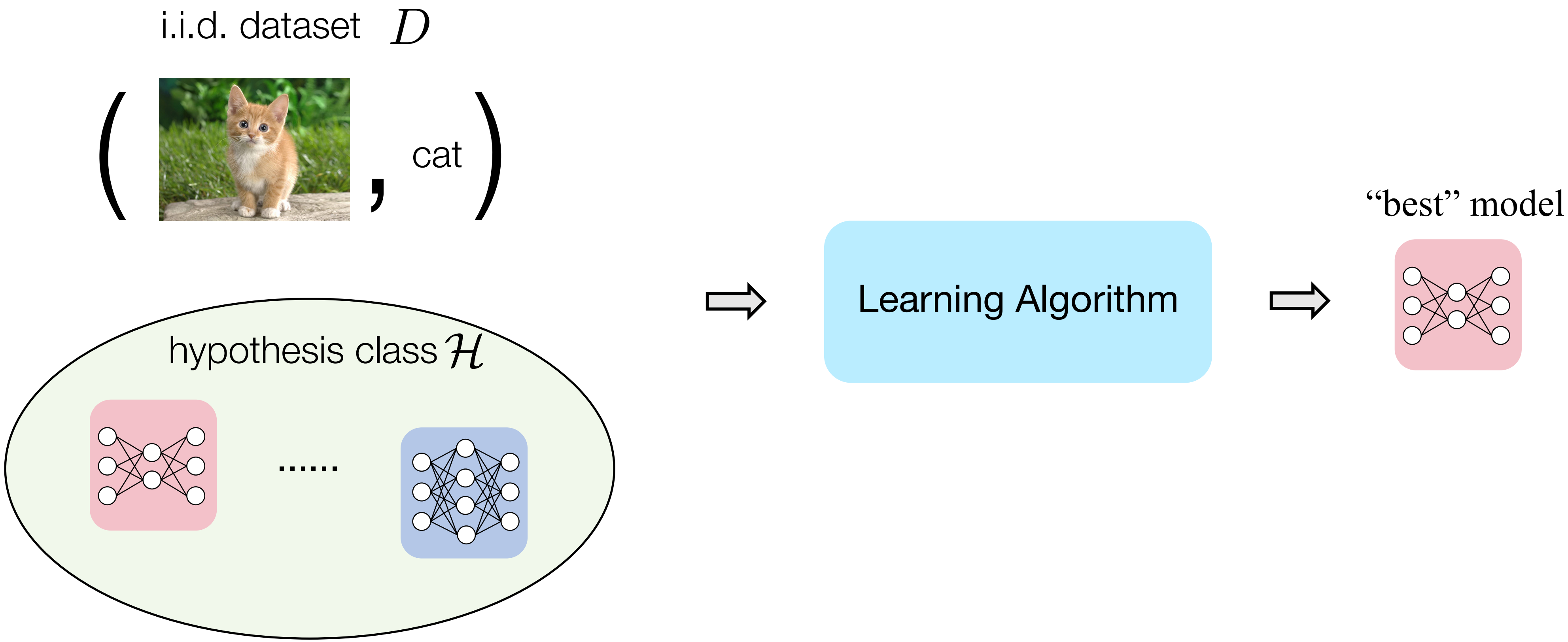
Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Outline

- **Statistical Learning Setup**
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Statistical Learning Setup



Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

1) Assumptions of IID sampling and unknown data distribution

Training data are sampled from an unknown data distribution in an i.i.d. (independent and identically distributed) fashion

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Therefore, the training dataset

$$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

1) Assumptions of IID sampling and unknown data distribution

Training data are sampled from an unknown data distribution in an i.i.d. (independent and identically distributed) fashion

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Therefore, the training dataset

$$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Either input or output could be continuous or discrete scalars, vectors, tensors, sets, sequences, graphs, …

E.g. regression $x_n \in \mathbb{R}^2 \qquad y_n \in \mathbb{R}$

classification $x_n \in \mathbb{R}^2 \qquad y_n \in \{1, 2, \ldots, K\}$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

2) Model and Loss

We introduce a *model* (a.k.a., *hypothesis*) $f(x, w)$ with learnable parameters w

N.B.: *hyperparameters* are fixed and not learnable

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

2) Model and Loss

We introduce a *model* (a.k.a., *hypothesis*) $f(x, w)$ with learnable parameters w

N.B.: *hyperparameters* are fixed and not learnable

It belongs to a hypothesis class $f(x, w) \in \mathcal{H}$

E.g. all linear models with weight norm no larger than 1 $\mathcal{H} = \{f(x, w) | f(x, w) = w^\top x, \|w\| \leq 1\}$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

2) Model and Loss

We introduce a *model* (a.k.a., *hypothesis*) $f(x, w)$ with learnable parameters w

N.B.: *hyperparameters* are fixed and not learnable

It belongs to a hypothesis class $f(x, w) \in \mathcal{H}$

E.g. all linear models with weight norm no larger than 1 $\mathcal{H} = \{f(x, w) | f(x, w) = w^\top x, \|w\| \leq 1\}$

Loss is denoted as $\ell(y, f(x, w))$

Generalization error (a.k.a., *risk or expected loss*) is $\mathbb{E}_{\mathbb{P}_{\mathrm{data}(x,y)}} [\ell(y, f(x, w))]$

Training error (a.k.a., *empirical risk* or *training loss*) is $\frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n, w))$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

2) Model and Loss

We introduce a *model* (a.k.a., *hypothesis*) $f(x, w)$ with learnable parameters w

N.B.: *hyperparameters* are fixed and not learnable

It belongs to a hypothesis class $f(x, w) \in \mathcal{H}$

E.g. all linear models with weight norm no larger than 1 $\mathcal{H} = \{f(x, w) | f(x, w) = w^\top x, \|w\| \leq 1\}$

Loss is denoted as $\ell(y, f(x, w))$

Generalization error (a.k.a., *risk or expected loss*) is $\mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)} [\ell(y, f(x, w))]$

Training error (a.k.a., *empirical risk* or *training loss*) is $\frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n, w))$

Monte Carlo Estimation

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

3) Learning

Ideally, we want to find a model in the hypothesis class that minimizes the risk:

$$\min_{f \in \mathcal{H}} \quad \mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)} \left[\ell(y, f(x, w)) \right]$$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

3) Learning

Ideally, we want to find a model in the hypothesis class that minimizes the risk:

$$\min_{f \in \mathcal{H}} \quad \mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)} \left[\ell(y, f(x, w)) \right]$$

But since risk is incomputable (why?), we can approximate it via

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n, w))$$

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

3) Learning

Ideally, we want to find a model in the hypothesis class that minimizes the risk:

$$\min_{f \in \mathcal{H}} \quad \mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)} \left[\ell(y, f(x, w)) \right]$$

But since risk is incomputable (why?), we can approximate it via

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n, w))$$

This learning framework is called *empirical risk minimization (ERM)*!

Statistical Learning Setup

Let us review some key concepts and assumptions (mainly about supervised learning like classification and regression) in statistical learning theory, which was initially developed by Vladimir Vapnik, e.g., [1].

3) Learning

Ideally, we want to find a model in the hypothesis class that minimizes the risk:

$$\min_{f \in \mathcal{H}} \quad \mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)} \left[\ell(y, f(x, w)) \right]$$

But since risk is incomputable (why?), we can approximate it via

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(x_n, w))$$

A learning algorithm can be viewed as a mapping that maps a training dataset, initial parameters, and hyperparameters to "optimal" parameters:

$$w^* = \mathcal{A}(D, w^0)$$

This learning framework is called *empirical risk minimization (ERM)*!

Outline

- Statistical Learning Setup
- **Linear Regression**
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

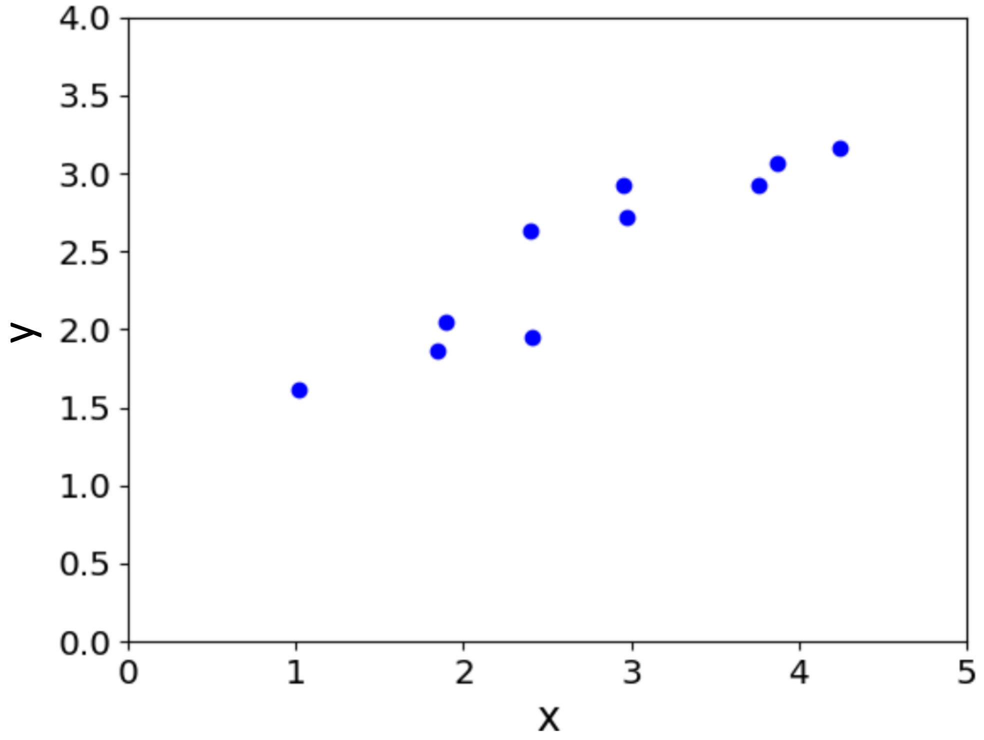


Image Credit: UofT CSC413 Lecture 1

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

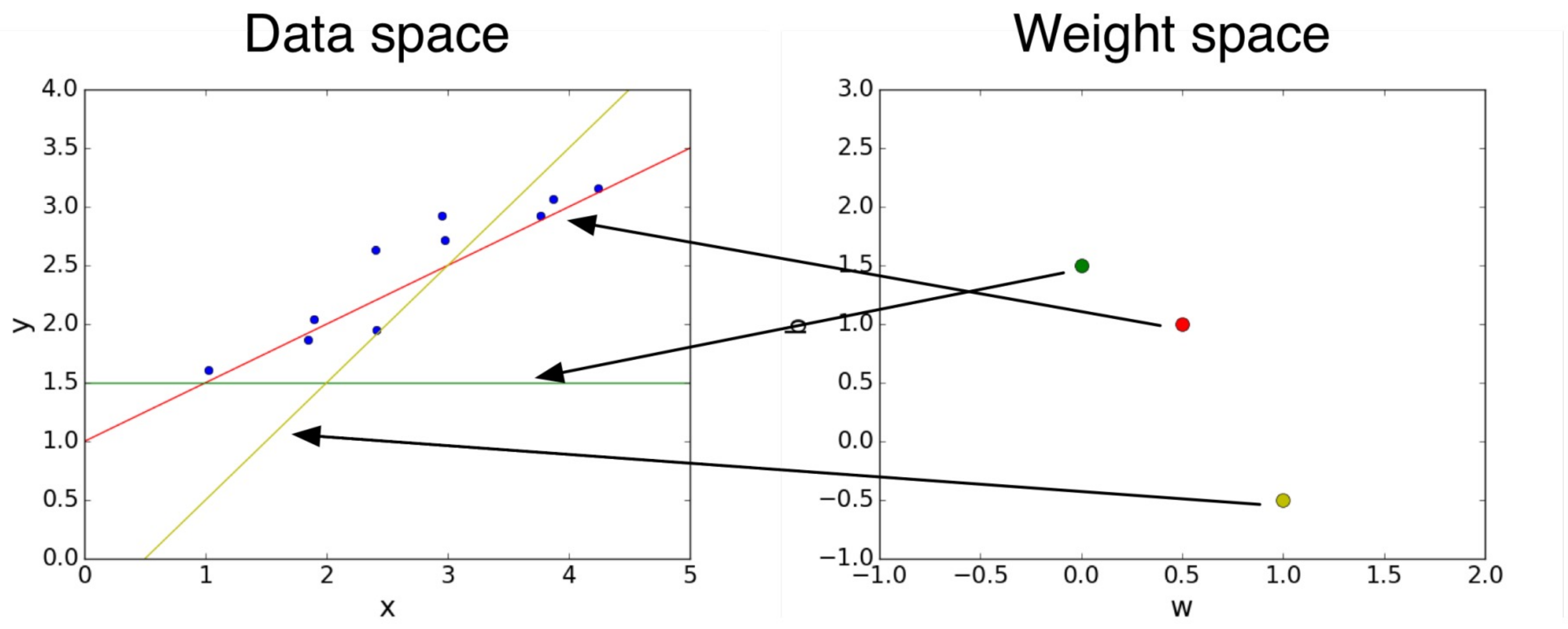


Image Credit: UofT CSC413 Lecture 1

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) Mean squared error (MSE), a.k.a., L2 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

  Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) Mean squared error (MSE), a.k.a., L2 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$
2) L1 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$

Linear Regression

- Problem Specification (1D-Regression)

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

$$x_n \in \mathbb{R} \qquad y_n \in \mathbb{R}$$

- Model Design

  Linear model (or a linear layer) $\hat{y} = w^\top x + b$

- Loss Design

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) Mean squared error (MSE), a.k.a., L2 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$
2) L1 loss $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$
3) Smooth L1 loss (similar to Huber loss used in robust statistics)

$$\ell(\hat{y}_n, y_n) = \begin{cases} \frac{1}{2\beta} \|\hat{y}_n - y_n\|_2^2 & \text{if } \|\hat{y}_n - y_n\|_1 < \beta \\ \|\hat{y}_n - y_n\|_1 - \frac{1}{2\beta} & \text{otherwise} \end{cases}$$

Linear Regression

- Loss Design

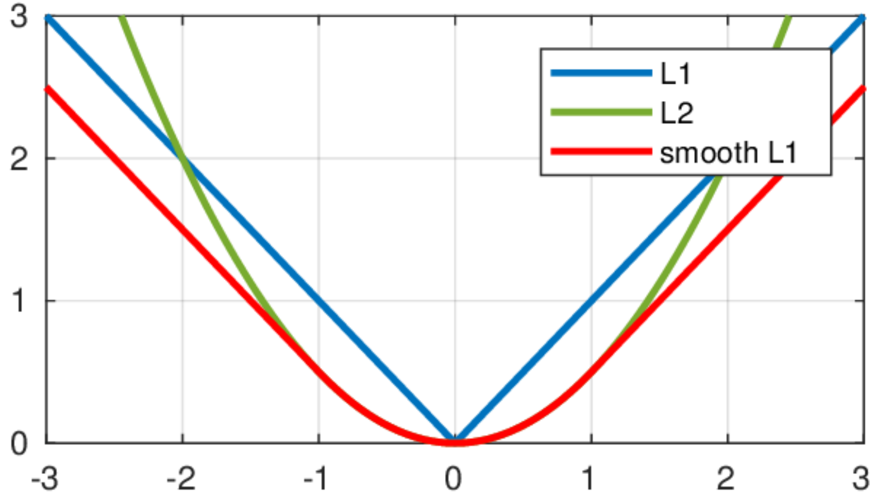
$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N}\sum_{n=1}^{N} \ell(\hat{y}_n, y_n)$$

1) Mean squared error (MSE), a.k.a., L2 loss $\quad \ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$

2) L1 loss $\quad \ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$

3) Smooth L1 loss (similar to Huber loss used in robust statistics)

$$\ell(\hat{y}_n, y_n) = \begin{cases} \frac{1}{2\beta}\|\hat{y}_n - y_n\|_2^2 & \text{if } \|\hat{y}_n - y_n\|_1 < \beta \\ \|\hat{y}_n - y_n\|_1 - \frac{1}{2\beta} & \text{otherwise} \end{cases}$$



Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - **Inference Algorithm**
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Linear Regression

- Inference Algorithm

  The term "*inference*" has been used in many contexts, hence being very confusing.

Linear Regression

- Inference Algorithm

  The term “*inference*” has been used in many contexts, hence being very confusing.

  1) “*Inference*” in DL and many ML areas:

     It typically means the computational process from input to output, e.g., the forward pass of a feedforward neural network

Linear Regression

- Inference Algorithm

The term "*inference*" has been used in many contexts, hence being very confusing.

1) "*Inference*" in DL and many ML areas:

   It typically means the computational process from input to output, e.g., the forward pass of a feedforward neural network

2) "*Inference*" or "*probabilistic inference*" in graphical models:

   It typically means computing the marginal probability or the maximum a posterior (MAP) estimation

Linear Regression

- Inference Algorithm

The term "*inference*" has been used in many contexts, hence being very confusing.

1) "*Inference*" in DL and many ML areas:

   It typically means the computational process from input to output, e.g., the forward pass of a feedforward neural network

2) "*Inference*" or "*probabilistic inference*" in graphical models:

   It typically means computing the marginal probability or the maximum a posterior (MAP) estimation

3) Statistical inference in statistics:

   It typically means estimating the parameters of the model, which is called learning/training in DL/ML

Linear Regression

- Inference Algorithm

The term “*inference*” has been used in many contexts, hence being very confusing.

1) “*Inference*” in DL and many ML areas:

   It typically means the computational process from input to output, e.g., the forward pass of a feedforward neural network

2) “*Inference*” or “*probabilistic inference*” in graphical models:

   It typically means computing the marginal probability or the maximum a posterior (MAP) estimation

3) Statistical inference in statistics:

   It typically means estimating the parameters of the model, which is called learning/training in DL/ML

For our linear models, inference is just: $\hat{y} = w^\top x + b$

For other models in DL/ML, e.g., deep energy based models, one may need both of first two!

Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - **Learning/Training Algorithm (Gradient Descent)**
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

Since learning is an optimization problem, a learning algorithm is just an optimization algorithm.

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

Since learning is an optimization problem, a learning algorithm is just an optimization algorithm.

1) Gradient-based learning algorithms:

2) Gradient-free learning algorithms:

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

Since learning is an optimization problem, a learning algorithm is just an optimization algorithm.

1) Gradient-based learning algorithms:
   - 1st order gradient method, e.g., stochastic gradient descent (SGD)
   - 2nd order gradient method, e.g., Newton's method

   ……
2) Gradient-free learning algorithms:
   - Genetic algorithms
   - Random search, e.g., simulated annealing

   ……

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

In linear regression, we have

$$f(x, w, b) = \hat{y} = w^\top x + b$$

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

In linear regression, we have $f(x, w, b) = \hat{y} = w^\top x + b$

Therefore, with MSE loss, the learning problem is

$$\min_{w,b} \quad L(w, b) = \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n, w, b), y_n) = \frac{1}{N} \sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

Linear Regression

- Learning Algorithm

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n), y_n)$$

In linear regression, we have

$$f(x, w, b) = \hat{y} = w^\top x + b$$

Therefore, with MSE loss, the learning problem is

$$\min_{w,b} \quad L(w, b) = \frac{1}{N} \sum_{n=1}^{N} \ell(f(x_n, w, b), y_n) = \frac{1}{N} \sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

We can equivalently (why?) rewrite it as

$$\min_{w,b} \quad L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

What is a gradient?

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

What is a gradient?

Loss is typically a scalar, parameters can be viewed as a vector, the gradient is defined as

$$\frac{\partial L}{\partial w[i]} = \lim_{\epsilon \to 0} \frac{L(w + \epsilon e_i, b) - L(w,b)}{\epsilon}$$

where $w[i]$ is the i-th element (scalar) of weight and e_i is a zero vector except that the i-th element is 1.

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

What is a gradient?

Loss is typically a scalar, parameters can be viewed as a vector, the gradient is defined as

$$\frac{\partial L}{\partial w[i]} = \lim_{\epsilon \to 0} \frac{L(w + \epsilon e_i, b) - L(w,b)}{\epsilon}$$

where $w[i]$ is the i-th element (scalar) of weight and e_i is a zero vector except that the i-th element is 1.

This definition (central difference version) is useful for checking the correctness of gradient implementation!

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

The definition does not help much in getting the analytical form of the gradient.

Linear Regression

- Learning Algorithm

$$\min_{w,b} \quad L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \|w^\top x_n + b - y_n\|_2^2$$

To use gradient descent (GD) or stochastic gradient descent (SGD), we first need to derive the gradient

The definition does not help much in getting the analytical form of the gradient.

We learn from calculus about how to derive gradient via basic derivatives and their rules:

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2$$

$$= \frac{1}{2N}\sum_{n=1}^{N} \left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n \right)^2$$

Linear Regression

- Learning Algorithm

$$L(w, b) = \frac{1}{2N} \sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} w[d] x_n[d] + b - y_n \right)^2$$

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

$$\begin{aligned}
\frac{\partial L(w,b)}{\partial w[i]} &= \sum_{n=1}^{N} \frac{\partial L(w,b)}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \sum_{n=1}^{N} \frac{\partial \frac{1}{2N}\sum_{n=1}^{N} l_n^2}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{2N}\sum_{n=1}^{N} 2l_n \frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{N}\sum_{n=1}^{N} l_n \frac{\partial \left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)}{\partial w[i]} \\
&= \frac{1}{N}\sum_{n=1}^{N} l_n x_n[i]
\end{aligned}$$

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

$$\begin{aligned}
\frac{\partial L(w,b)}{\partial w[i]} &= \sum_{n=1}^{N} \frac{\partial L(w,b)}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \sum_{n=1}^{N} \frac{\partial \frac{1}{2N}\sum_{n=1}^{N} l_n^2}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{2N}\sum_{n=1}^{N} 2l_n \frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{N}\sum_{n=1}^{N} l_n \frac{\partial \left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)}{\partial w[i]} \\
&= \frac{1}{N}\sum_{n=1}^{N} l_n x_n[i]
\end{aligned}$$

Write in a compact vector form:

$$\frac{\partial L(w,b)}{\partial w} = \begin{bmatrix} \frac{\partial L(w,b)}{\partial w[1]} \\ \vdots \\ \frac{\partial L(w,b)}{\partial w[D]} \end{bmatrix}$$

$$= \frac{1}{N}\sum_{n=1}^{N} l_n x_n$$

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

$$\begin{aligned}
\frac{\partial L(w,b)}{\partial w[i]} &= \sum_{n=1}^{N} \frac{\partial L(w,b)}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \sum_{n=1}^{N} \frac{\partial \frac{1}{2N}\sum_{n=1}^{N} l_n^2}{\partial l_n}\frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{2N}\sum_{n=1}^{N} 2l_n \frac{\partial l_n}{\partial w[i]} \\
&= \frac{1}{N}\sum_{n=1}^{N} l_n \frac{\partial \left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)}{\partial w[i]} \\
&= \boxed{\frac{1}{N}\sum_{n=1}^{N} l_n x_n[i]}
\end{aligned}$$

Write in a compact vector form:

$$\frac{\partial L(w,b)}{\partial w} = \begin{bmatrix} \frac{\partial L(w,b)}{\partial w[1]} \\ \vdots \\ \frac{\partial L(w,b)}{\partial w[D]} \end{bmatrix} = \frac{1}{N}\sum_{n=1}^{N} l_n x_n$$

If N is too large, we can randomly sample a smaller subset (a.k.a. mini-batch) to compute the gradient.

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Linear Regression

- Learning Algorithm

$$L(w,b)=\frac{1}{2N}\sum_{n=1}^{N}\|\underbrace{w^\top x_n+b-y_n}_{l_n}\|_2^2=\frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D}w[d]x_n[d]+b-y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Then we can perform the gradient descent algorithms

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t=1,\cdots,T$
3: $\quad w^t=w^{t-1}-\eta\frac{\partial L(w^{t-1},b^{t-1})}{\partial w}$
4: $\quad b^t=b^{t-1}-\eta\frac{\partial L(w^{t-1},b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Then we can perform the gradient descent algorithms

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)

2: **For** $t = 1, \cdots, T$

3: $\quad w^t = w^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$

4: $\quad b^t = b^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$

5: **Return** (w^T, b^T)

learning rate / step size

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Then we can perform the gradient descent algorithms

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t = 1, \cdots, T$
3: $\quad w^t = w^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$
4: $\quad b^t = b^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

learning rate / step size

- If we use full training dataset to compute the gradient per step, then it is called *(batch) gradient descent*

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Then we can perform the gradient descent algorithms

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t = 1, \cdots, T$
3: $\quad w^t = w^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$
4: $\quad b^t = b^{t-1} - \eta \frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

learning rate / step size

- If we use full training dataset to compute the gradient per step, then it is called *(batch) gradient descent*
- If we use random mini-batch data to compute the gradient per step, then it is called *stochastic gradient descent*

Linear Regression

- Learning Algorithm

$$L(w,b) = \frac{1}{2N}\sum_{n=1}^{N} \| \underbrace{w^\top x_n + b - y_n}_{l_n} \|_2^2 = \frac{1}{2N}\sum_{n=1}^{N}\left(\sum_{d=1}^{D} w[d]x_n[d] + b - y_n\right)^2$$

Similarly, we can obtain the partial derivative $\frac{\partial L(w,b)}{\partial b}$ (do it by yourself)

Then we can perform the gradient descent algorithms

Algorithm 1 GD Learning Algorithm

1: **Input**: GD step T, learning Rate η, initial (w^0, b^0)
2: **For** $t = 1, \cdots, T$
3: $\quad w^t = w^{t-1} - \eta\frac{\partial L(w^{t-1}, b^{t-1})}{\partial w}$
4: $\quad b^t = b^{t-1} - \eta\frac{\partial L(w^{t-1}, b^{t-1})}{\partial b}$
5: **Return** (w^T, b^T)

Is the model at the last step necessarily the best?

Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - **Validation and Testing** (**Overfitting vs. Underfitting**, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

Linear Regression

- Validation and Testing

The goal of ML is to learn a model on observed data so that it can generalize well to unseen data.

Linear Regression

- Validation and Testing

The goal of ML is to learn a model on observed data so that it can generalize well to unseen data.

To facilitate this goal, we typically split a dataset into **train**/**validation**(a.k.a. develop)/**test** subsets

1) During training, you can use training set to train your model, e.g., *GD to train linear regression*

Linear Regression

- Validation and Testing

The goal of ML is to learn a model on observed data so that it can generalize well to unseen data.

To facilitate this goal, we typically split a dataset into **train**/**validation**(a.k.a. develop)/**test** subsets

1) During training, you can use training set to train your model, e.g., *GD to train linear regression*
2) We can tune hyperparameters and select the best model based on the validation performance, e.g., *we can evaluate models on the validation set every 100 steps and return the model with the best validation metric.*

Linear Regression

- Validation and Testing

The goal of ML is to learn a model on observed data so that it can generalize well to unseen data.

To facilitate this goal, we typically split a dataset into **train**/**validation**(a.k.a. develop)/**test** subsets

1) During training, you can use training set to train your model, e.g., *GD to train linear regression*
2) We can tune hyperparameters and select the best model based on the validation performance, e.g., *we can evaluate models on the validation set every 100 steps and return the model with the best validation metric.*
3) We should never use test set to select the model since it is cheating!

Linear Regression

- Validation and Testing

The goal of ML is to learn a model on observed data so that it can generalize well to unseen data.

To facilitate this goal, we typically split a dataset into **train**/**validation**(a.k.a. develop)/**test** subsets

1) During training, you can use training set to train your model, e.g., *GD to train linear regression*
2) We can tune hyperparameters and select the best model based on the validation performance, e.g., *we can evaluate models on the validation set every 100 steps and return the model with the best validation metric.*
3) We should never use test set to select the model since it is cheating!

If your dataset is of a small size, then you can use k-fold cross-validation.

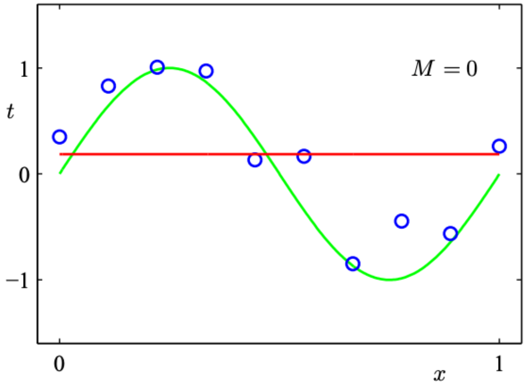
Linear Regression

- Validation and Testing: Overfitting vs. Underfitting

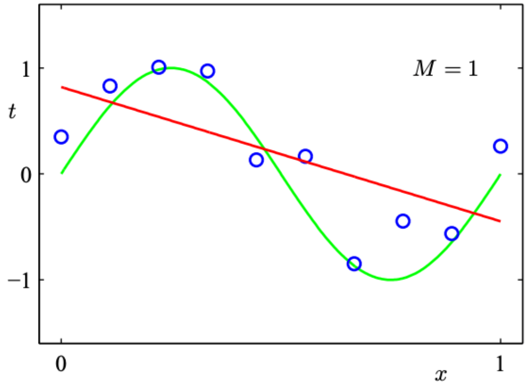
Linear Regression

- Validation and Testing: Overfitting vs. Underfitting

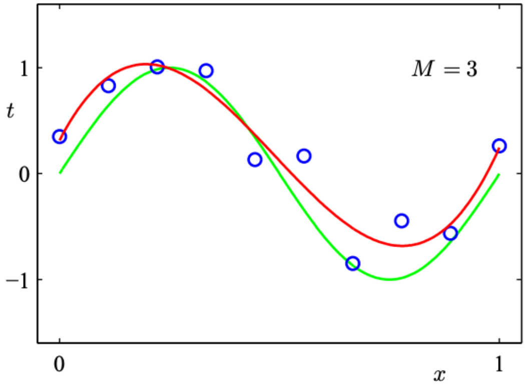
$$y = w_0$$



$$y = w_0 + w_1 x$$



$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$



$$y = w_0 + w_1 x + \cdots + w_9 x^9$$

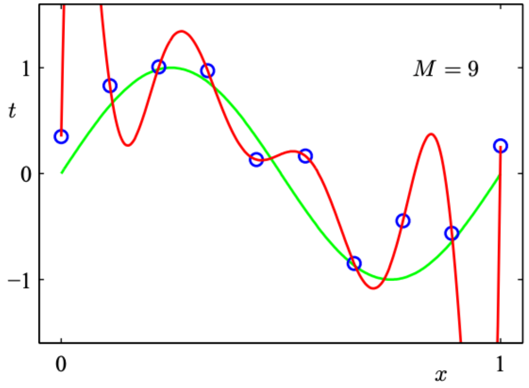
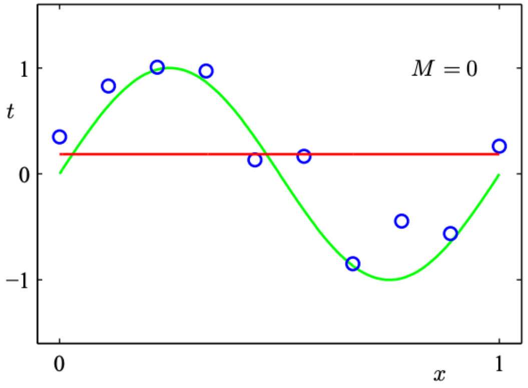


Image Credit: [2]

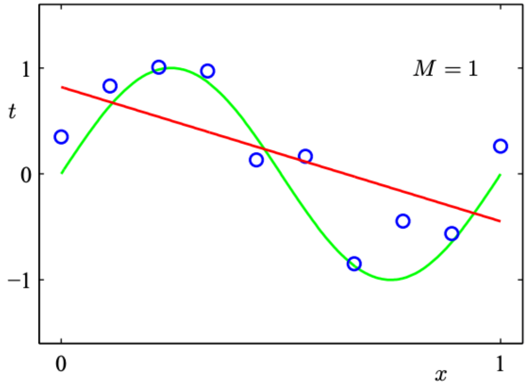
Linear Regression

- Validation and Testing: Overfitting vs. Underfitting

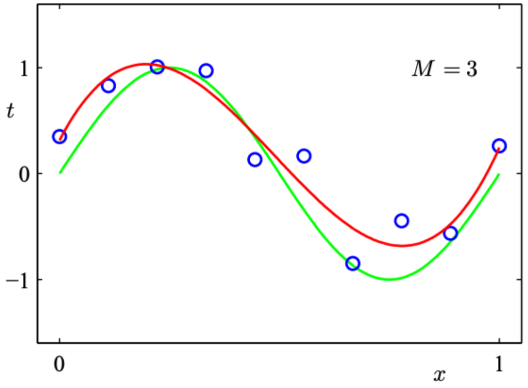
$y = w_0$



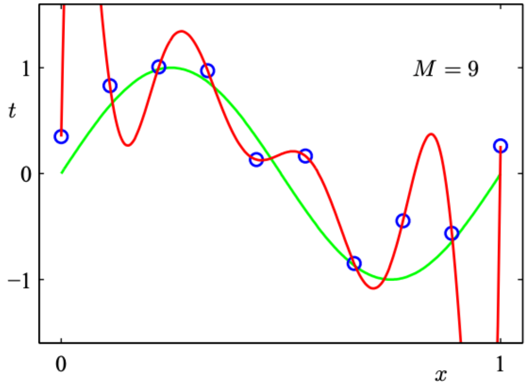
$y = w_0 + w_1 x$



$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$



$y = w_0 + w_1 x + \cdots + w_9 x^9$



Underfitting:

Model is too simple to fit the data

Overfitting:

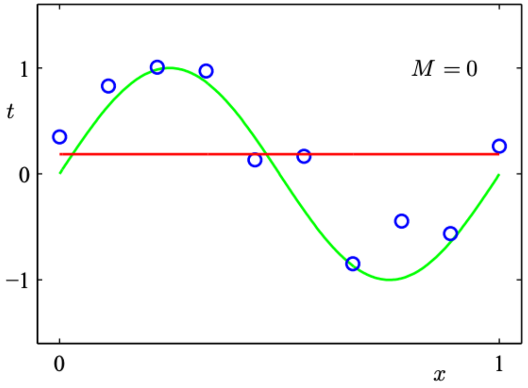
Model is too complicate, perfectly fits the data, but does not generalize

Image Credit: [2]

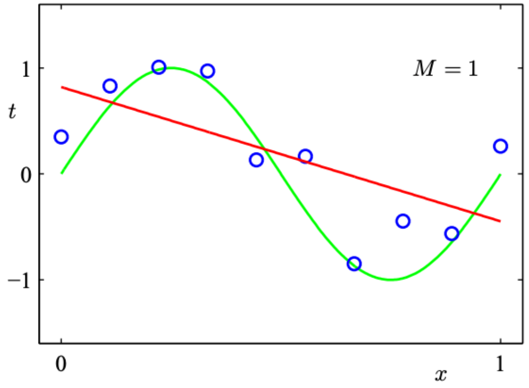
Linear Regression

- Validation and Testing: Overfitting vs. Underfitting

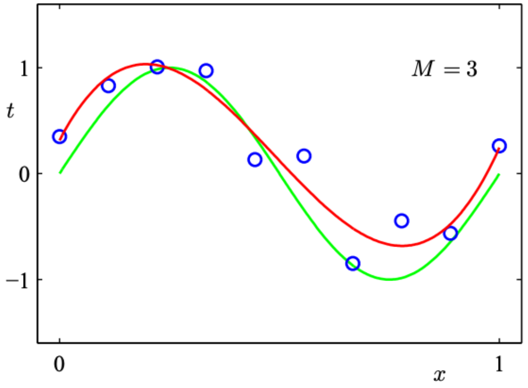
$y = w_0$



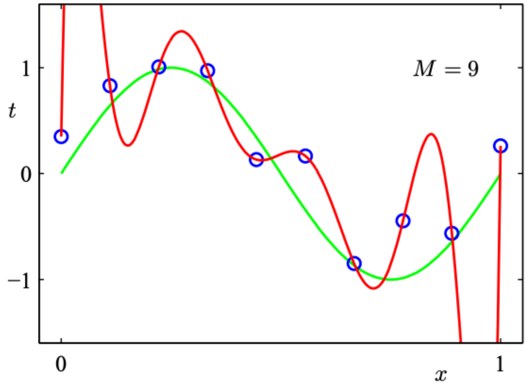
$y = w_0 + w_1 x$



$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$



$y = w_0 + w_1 x + \cdots + w_9 x^9$



Underfitting:

Model is too simple to fit the data

Overfitting:

Model is too complicate, perfectly fits the data, but does not generalize

There exists *benign overfitting* (i.e., complicated models perfectly fit and generalize well) in deep learning (cf. [32])!

Image Credit: [2]

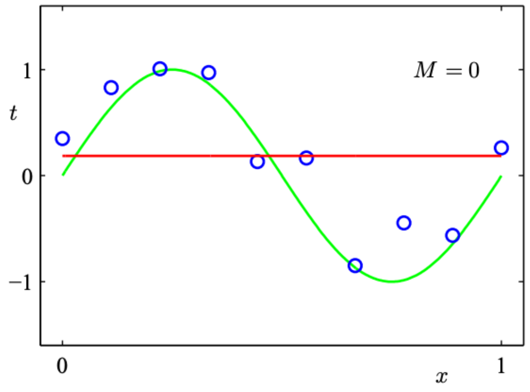
Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - **Validation and Testing** (Overfitting vs. Underfitting, **Bias Variance Tradeoff**)
- Linear Classification
  - Logistic Regression
  - Multiclass Logistic Regression

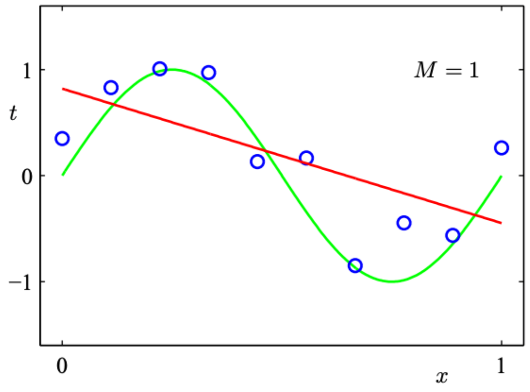
Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

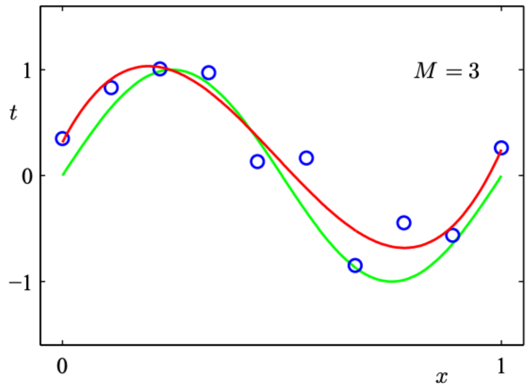
$y = w_0$



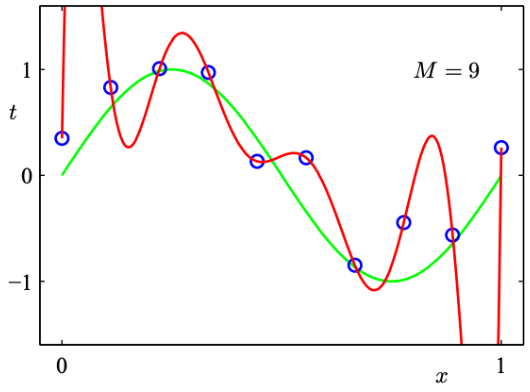
$y = w_0 + w_1 x$



$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$



$y = w_0 + w_1 x + \cdots + w_9 x^9$



As the degree (complexity) increases, the variance of the model tends to increase, and the bias tends to decrease!

Image Credit: [2]

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Recall

$$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Our training dataset

$$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Expected label/output

$$\bar{y}(x) = \mathbb{E}_{\mathbb{P}_{\text{data}}(y|x)}[y] = \int_y y \mathbb{P}_{\text{data}}(y|x) dy$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Recall $$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Our training dataset $$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Expected label/output $$\bar{y}(x) = \mathbb{E}_{\mathbb{P}_{\text{data}}(y|x)}[y] = \int_y y \mathbb{P}_{\text{data}}(y|x) dy$$

Our learned model $$f(x, w^*) = f(x, \mathcal{A}(D, w^0)) \equiv f_D(x)$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Recall $$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Our training dataset $$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Expected label/output $$\bar{y}(x) = \mathbb{E}_{\mathbb{P}_{\text{data}}(y|x)}[y] = \int_y y \mathbb{P}_{\text{data}}(y|x) dy$$

Our learned model $$f(x, w^*) = f(x, \boxed{\mathcal{A}(D, w^0)}) \equiv f_D(x)$$

Learning algorithm depends on training dataset, initial parameters, and hyperparameters

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Recall $$(x_n, y_n) \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y) \qquad n = 1, \ldots, N$$

Our training dataset $$D = \{(x_n, y_n) | n = 1, \ldots, N\} \sim \mathbb{P}_{\text{data}}(x, y)^N$$

Expected label/output $$\bar{y}(x) = \mathbb{E}_{\mathbb{P}_{\text{data}}(y|x)}[y] = \int_y y \mathbb{P}_{\text{data}}(y|x) dy$$

Our learned model $$f(x, w^*) = f(x, \boxed{\mathcal{A}(D, w^0)}) \equiv f_D(x)$$

Learning algorithm depends on training dataset, initial parameters, and hyperparameters

Expected learned model $$\bar{f}(x) = \mathbb{E}_{D \sim \mathbb{P}_{\text{data}}(x,y)^N}[f_D(x)]$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Generalization error $\mathbb{E}_{\mathbb{P}_{\mathrm{data}}(x,y)}\left[(y - f_D(x))^2\right]$

Expected generalization error $\mathbb{E}_{D\sim\mathbb{P}_{\mathrm{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\mathrm{data}}(x,y)}\left[(y - f_D(x))^2\right]$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Generalization error $\mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)}\left[(y - f_D(x))^2\right]$

Expected generalization error $\mathbb{E}_{D\sim\mathbb{P}_{\text{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\text{data}}(x,y)}\left[(y - f_D(x))^2\right]$

This is what we really care in comparing different learning systems as it considers all possible training sets!

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Generalization error $\mathbb{E}_{\mathbb{P}_{\mathrm{data}}(x,y)}\left[(y - f_D(x))^2\right]$

Expected generalization error $\mathbb{E}_{D\sim\mathbb{P}_{\mathrm{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\mathrm{data}}(x,y)}\left[(y - f_D(x))^2\right]$

This is what we really care in comparing different learning systems as it considers all possible training sets!

Let us decompose it

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] \equiv \mathbb{E}_{D\sim\mathbb{P}_{\mathrm{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\mathrm{data}}(x,y)}\left[(y - f_D(x))^2\right]$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Generalization error $\mathbb{E}_{\mathbb{P}_{\text{data}}(x,y)}\left[(y - f_D(x))^2\right]$

Expected generalization error $\mathbb{E}_{D\sim\mathbb{P}_{\text{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\text{data}}(x,y)}\left[(y - f_D(x))^2\right]$

This is what we really care in comparing different learning systems as it considers all possible training sets!

Let us decompose it

$$
\begin{aligned}
\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] &\equiv \mathbb{E}_{D\sim\mathbb{P}_{\text{data}}(x,y)^N,(x,y)\sim\mathbb{P}_{\text{data}}(x,y)}\left[(y - f_D(x))^2\right] \\
&= \mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x) + \bar{f}(x) - y)^2\right] \\
&= \mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{D,x,y}\left[(\bar{f}(x) - y)^2\right] \\
&= \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]
\end{aligned}
$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

$$\begin{aligned}
\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] &= \mathbb{E}_{x,y}\left[\mathbb{E}_D\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right]\right] \\
&= \mathbb{E}_{x,y}\left[\mathbb{E}_D\left[(f_D(x) - \bar{f}(x))\right](\bar{f}(x) - y)\right] \\
&= \mathbb{E}_{x,y}\left[(\mathbb{E}_D\left[f_D(x)\right] - \bar{f}(x))(\bar{f}(x) - y)\right] \\
&= \mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{f}(x))(\bar{f}(x) - y)\right] \\
&= 0
\end{aligned}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \boxed{\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \boxed{\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]}$$

$$\begin{aligned}\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right] &= \mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x) + \bar{y}(x) - y)^2\right] \\ &= \mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x))^2\right] + 2\mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x))(\bar{y}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]\end{aligned}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y-f_D(x))^2\right]=\mathbb{E}_{D,x}\left[(f_D(x)-\bar{f}(x))^2\right]+2\mathbb{E}_{D,x,y}\left[(f_D(x)-\bar{f}(x))(\bar{f}(x)-y)\right]+\mathbb{E}_{x,y}\left[(\bar{f}(x)-y)^2\right]$$

$$\begin{aligned}\mathbb{E}_{x,y}\left[(\bar{f}(x)-y)^2\right]&=\mathbb{E}_{x,y}\left[(\bar{f}(x)-\bar{y}(x)+\bar{y}(x)-y)^2\right]\\&=\mathbb{E}_{x,y}\left[(\bar{f}(x)-\bar{y}(x))^2\right]+2\mathbb{E}_{x,y}\left[(\bar{f}(x)-\bar{y}(x))(\bar{y}(x)-y)\right]+\mathbb{E}_{x,y}\left[(\bar{y}(x)-y)^2\right]\end{aligned}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

$$\begin{aligned}\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right] &= \mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x) + \bar{y}(x) - y)^2\right] \\ &= \mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x))^2\right] + 2\mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x))(\bar{y}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{x,y}\left[(\bar{f}(x) - \bar{y}(x))(\bar{y}(x) - y)\right] &= \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))\mathbb{E}_{y|x}\left[(\bar{y}(x) - y)\right]\right] \\ &= \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))(\bar{y}(x) - \mathbb{E}_{y|x}\left[y\right])\right] \\ &= \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))(\bar{y}(x) - \bar{y}(x))\right] \\ &= 0\end{aligned}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

So far, we have

$$\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] = 0$$

$$\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right] = \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right] + \mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

So far, we have

$$\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] = 0$$

$$\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right] = \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right] + \mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]$$

Putting together

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \underbrace{\mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right]}_{\text{variance}} + \underbrace{\mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\text{Noise}}$$

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right] + 2\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] + \mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right]$$

So far, we have

$$\mathbb{E}_{D,x,y}\left[(f_D(x) - \bar{f}(x))(\bar{f}(x) - y)\right] = 0$$

$$\mathbb{E}_{x,y}\left[(\bar{f}(x) - y)^2\right] = \mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right] + \mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]$$

Putting together

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \underbrace{\mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right]}_{\text{variance}} + \underbrace{\mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\text{Noise}}$$

Variance: Captures how much your classifier changes if you train on a different training set. How "over-specialized" is your classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?

Bias: What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model.

Noise: How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data.

Credit & More Info: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \underbrace{\mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right]}_{\text{variance}} + \underbrace{\mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\text{Noise}}$$

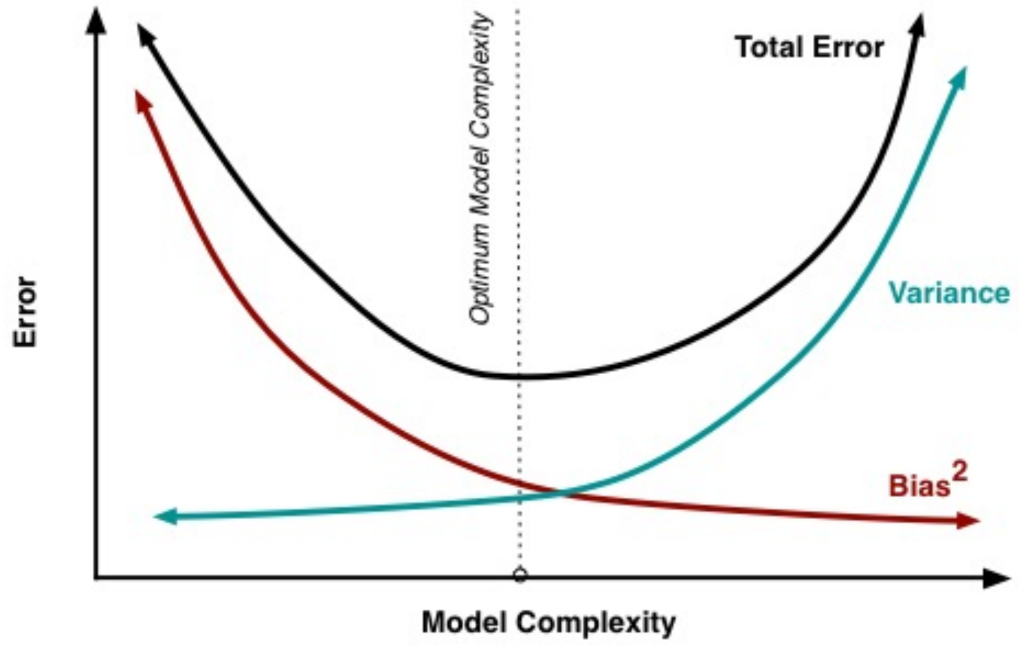


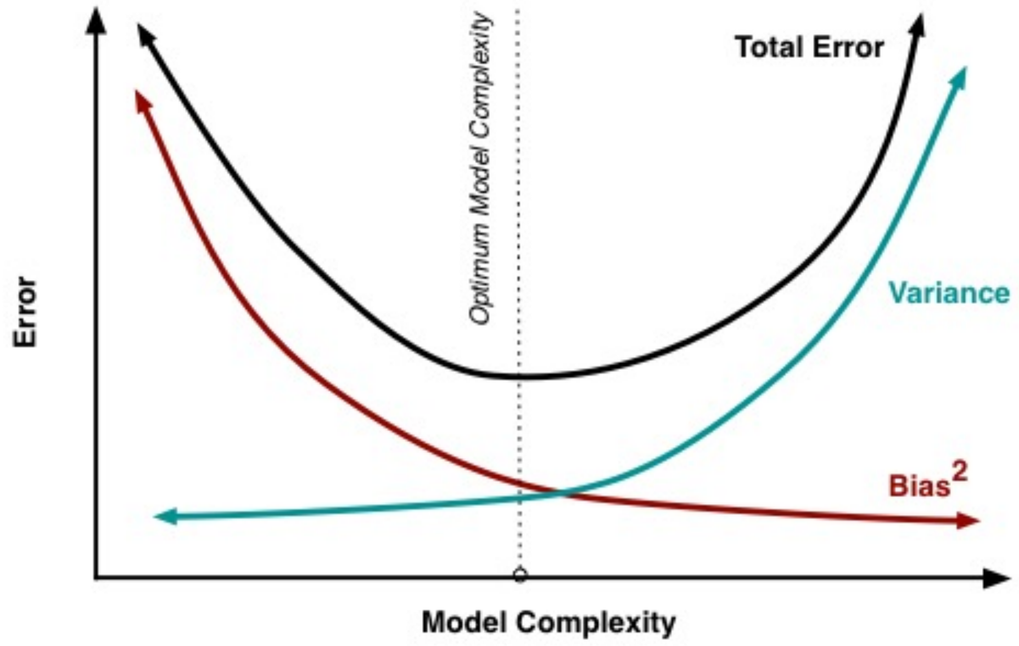
Image Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html

Linear Regression

- Validation and Testing: Bias vs. Variance Tradeoff

Decomposition of expected generalization error:

$$\mathbb{E}_{D,x,y}\left[(y - f_D(x))^2\right] = \underbrace{\mathbb{E}_{D,x}\left[(f_D(x) - \bar{f}(x))^2\right]}_{\text{variance}} + \underbrace{\mathbb{E}_{x}\left[(\bar{f}(x) - \bar{y}(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_{x,y}\left[(\bar{y}(x) - y)^2\right]}_{\text{Noise}}$$



This classic bias-variance tradeoff can not explain deep learning as the model complexity measure is hard to find, e.g., #parameters is clearly not the right one!

Image Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html

Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - **Logistic Regression**
  - Multiclass Logistic Regression

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification accuracy (can be rewritten using 0-1 loss) is

$$\bar{\ell} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n = y_n\right]$$

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification accuracy (can be rewritten using 0-1 loss) is

$$\bar{\ell} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n = y_n\right]$$

How can we perform gradient descent to learn the model?

Linear Models for Classification

Suppose we'd like to do a binary classification with a linear model

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{0, 1\}$$

$$f(x, w, b) = w^\top x + b$$

We can construct a threshold classifier (a discontinuous Heaviside step function) as

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification accuracy (can be rewritten using 0-1 loss) is

$$\bar{\ell} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n = y_n\right]$$

How can we perform gradient descent to learn the model?

Answer: continuous approximation!

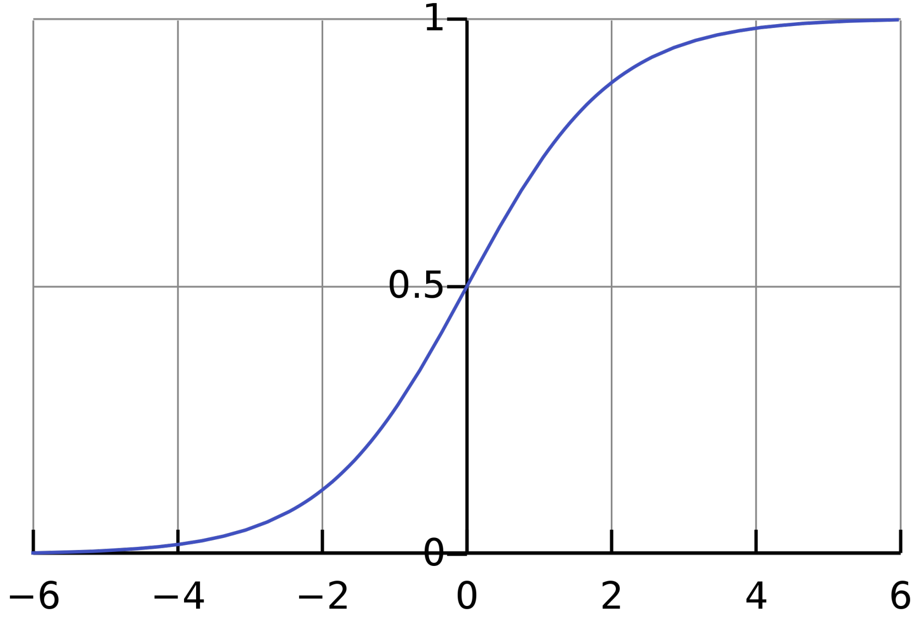
Linear Models for Classification

For the threshold classifier (a discontinuous Heaviside step function),

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we can approximate it with a logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Linear Models for Classification

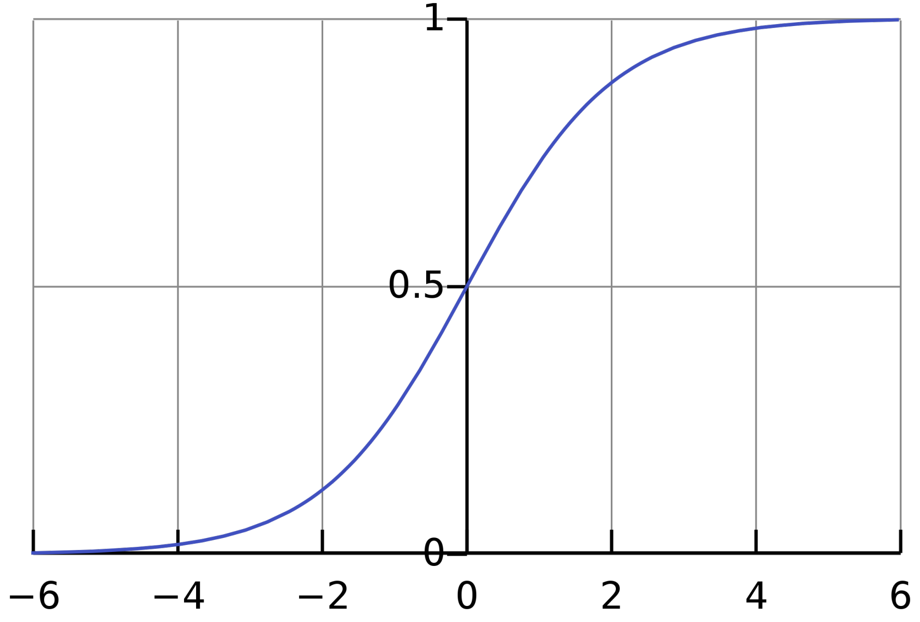
For the threshold classifier (a discontinuous Heaviside step function),

$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we can approximate it with a logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\hat{y} = \sigma(w^\top x + b)$$



Linear Models for Classification

For the threshold classifier (a discontinuous Heaviside step function),

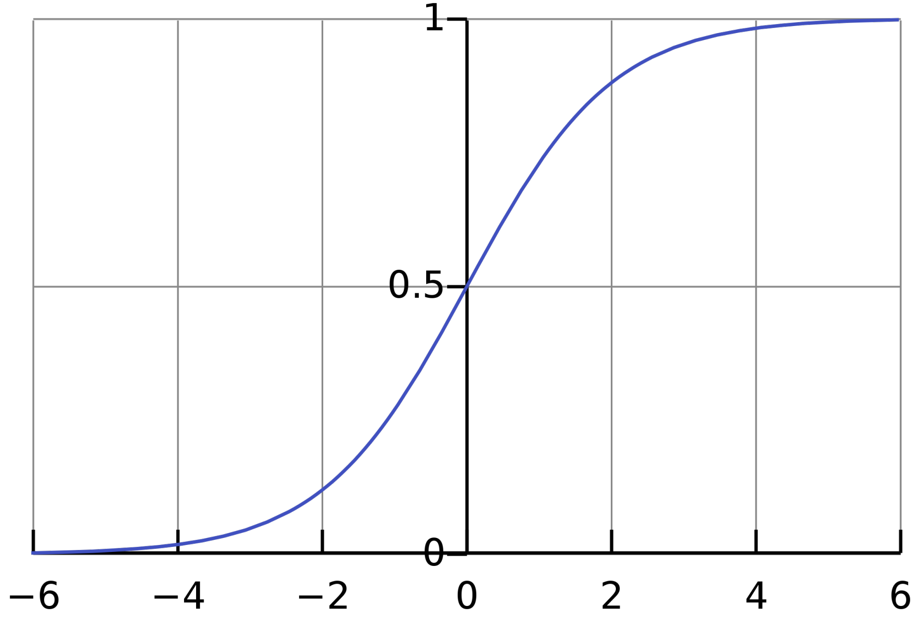
$$\hat{y} = \begin{cases} 1 & \text{if } f(x, w, b) > 0 \\ 0 & \text{otherwise} \end{cases}$$

we can approximate it with a logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\hat{y} = \sigma(w^\top x + b)$$

This outputs a probability!



Linear Models for Classification

The non-differentiable 0-1 loss for classification is,

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

Linear Models for Classification

The non-differentiable 0-1 loss for classification is,

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} [\hat{y}_n \neq y_n]$$

Since the sigmoid outputs a probability, we can use cross-entropy (CE) to approximate the 0-1 loss.

In particular, for two distributions (p, q) of a categorical (discrete) random variable (RV) with K states, CE is defined as,

$$\text{CE}(p, q) = -\sum_{i=1}^{K} p[i] \log q[i]$$

Linear Models for Classification

The non-differentiable 0-1 loss for classification is,

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

Since the sigmoid outputs a probability, we can use cross-entropy (CE) to approximate the 0-1 loss.

In particular, for two distributions (p, q) of a categorical (discrete) random variable (RV) with K states, CE is defined as,

$$\mathrm{CE}(p, q) = -\sum_{i=1}^{K} p[i] \log q[i]$$

For discrete RVs, it is non-negative and becomes smaller when p and q are closer.

Compared to 0-1 loss, it provides a finer measure, e.g., a 60% wrong answer is better than than a 90% wrong answer.

Linear Models for Classification

The non-differentiable 0-1 loss for classification is,

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

Since the sigmoid outputs a probability, we can use cross-entropy (CE) to approximate the 0-1 loss.

In particular, for two distributions (p, q) of a categorical (discrete) random variable (RV) with K states, CE is defined as,

$$\mathrm{CE}(p, q) = -\sum_{i=1}^{K} p[i] \log q[i]$$

For discrete RVs, it is non-negative and becomes smaller when p and q are closer.

Compared to 0-1 loss, it provides a finer measure, e.g., a 60% wrong answer is better than than a 90% wrong answer.

Since we have binary states, the CE loss reduces to

$$L(\{\hat{y}_n\}, \{y_n\}) = -\frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{1}\left[y_n = 1\right] \log \hat{y}_n + \mathbf{1}\left[y_n = 0\right] \log(1 - \hat{y}_n)\right)$$

$$\hat{y}_n = \sigma(w^\top x_n + b)$$

Linear Models for Classification

The non-differentiable 0-1 loss for classification is,

$$L(\{\hat{y}_n\}, \{y_n\}) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[\hat{y}_n \neq y_n\right]$$

Since the sigmoid outputs a probability, we can use cross-entropy (CE) to approximate the 0-1 loss.

In particular, for two distributions (p, q) of a categorical (discrete) random variable (RV) with K states, CE is defined as,

$$\mathrm{CE}(p, q) = -\sum_{i=1}^{K} p[i] \log q[i]$$

For discrete RVs, it is non-negative and becomes smaller when p and q are closer.

Compared to 0-1 loss, it provides a finer measure, e.g., a 60% wrong answer is better than than a 90% wrong answer.

Since we have binary states, the CE loss reduces to

$$L(\{\hat{y}_n\}, \{y_n\}) = -\frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{1}\left[y_n = 1\right] \log \hat{y}_n + \mathbf{1}\left[y_n = 0\right] \log(1 - \hat{y}_n)\right)$$

$$\hat{y}_n = \sigma(\boxed{w^\top x_n + b})$$

It is called *logits* and the whole model is called *logistic regression*

Outline

- Statistical Learning Setup
- Linear Regression
  - Problem Specification
  - Model Design
  - Loss Design
  - Inference Algorithm
  - Learning/Training Algorithm (Gradient Descent)
  - Validation and Testing (Overfitting vs. Underfitting, Bias Variance Tradeoff)
- Linear Classification
  - Logistic Regression
  - **Multiclass Logistic Regression**

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, w, b) = Wx + b \quad W \in \mathbb{R}^{K \times D} \quad b \in \mathbb{R}^{K \times 1}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n) | n = 1, \dots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \dots, K\}$$

$$f(x, w, b) = Wx + b \quad W \in \mathbb{R}^{K \times D} \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \dots, 0, 1, 0, \dots, 0]}_{\text{k-th entry is 1}}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n)|n = 1, \dots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \dots, K\}$$

$$f(x, w, b) = Wx + b \quad W \in \mathbb{R}^{K \times D} \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \dots, 0, 1, 0, \dots, 0]}_{\text{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

But we can not use sigmoid as the output anymore. Why?

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n) | n = 1, \ldots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \ldots, K\}$$

$$f(x, w, b) = Wx + b \quad W \in \mathbb{R}^{K \times D} \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

But we can not use sigmoid as the output anymore. Why?

Instead, we can use the softmax function, which outputs a valid probability distribution of a categorical RV with K states,

$$\text{softmax}(x)[i] = \frac{\exp(x[i])}{\sum_{k=1}^{K} \exp(x[k])}$$

Linear Models for Classification

What if we'd like to do multiclass classification:

$$\{(x_n, y_n) | n = 1, \dots, N\} \overset{\text{iid}}{\sim} \mathbb{P}_{\text{data}}(x, y)$$

$$x_n \in \mathbb{R}^D \qquad y_n \in \{1, \dots, K\}$$

$$f(x, w, b) = Wx + b \quad W \in \mathbb{R}^{K \times D} \quad b \in \mathbb{R}^{K \times 1}$$

We typically use 1-of-K encoding for the output:

$$y_n = k \quad \Leftrightarrow \quad y_n = \underbrace{[0, \dots, 0, 1, 0, \dots, 0]}_{\texttt{k-th entry is 1}}$$

By doing so, we can conveniently use the cross-entropy as the loss.

But we can not use sigmoid as the output anymore. Why?

Instead, we can use the softmax function, which outputs a valid probability distribution of a categorical RV with K states,

$$\text{softmax}(x)[i] = \frac{\exp(\boxed{x[i]})}{\sum_{k=1}^{K} \exp(x[k])}$$

It is called *logits* and the whole model is called *multiclass logistic regression*

Linear Models for Classification

We can write the cross-entropy as

$$
\begin{aligned}
L(\{\hat{y}_n\}, \{y_n\}) &= -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}\left[y_n = k\right] \log \hat{y}_n[k] \\
&= -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}\left[y_n = k\right] \log \operatorname{softmax}(f(x_n, w, b))[k] \\
&= -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}\left[y_n = k\right] \left(f(x_n, w, b)[k] - \log \sum_{j=1}^{K} \exp(f(x_n, w, b)[j]) \right)
\end{aligned}
$$

Linear Models for Classification

We can write the cross-entropy as

$$L(\{\hat{y}_n\}, \{y_n\}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}[y_n = k] \log \hat{y}_n[k]$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}[y_n = k] \log \mathrm{softmax}(f(x_n, w, b))[k]$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{1}[y_n = k] \left(f(x_n, w, b)[k] - \log \sum_{j=1}^{K} \exp(f(x_n, w, b)[j]) \right)$$

This is the *log-sum-exp* operator!
It approximates *maximum* operator.

Linear Models for Classification

We can write the cross-entropy as

$$L(\{\hat{y}_n\}, \{y_n\}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\log \hat{y}_n[k]$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\log \mathrm{softmax}(f(x_n, w, b))[k]$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\left(f(x_n, w, b)[k] - \boxed{\log\sum_{j=1}^{K}\exp}(f(x_n, w, b)[j])\right)$$

This is the *log-sum-exp* operator!
It approximates *maximum* operator.

log-sum-exp permits numerically-efficient (avoid overflow/underflow) implementation since

$$\log\sum_{i=1}^{K}\exp(x_i) = x^* - \log\exp(x^*) + \log\sum_{i=1}^{K}\exp(x_i) = x^* + \log\sum_{i=1}^{K}\exp(x_i - x^*) \qquad x^* = \max\{x_1, \ldots, x_K\}$$

Linear Models for Classification

We can write the cross-entropy as

$$
\begin{aligned}
L(\{\hat{y}_n\},\{y_n\}) &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\log \hat{y}_n[k] \\
&= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\log \mathrm{softmax}(f(x_n,w,b))[k] \\
&= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbf{1}\left[y_n = k\right]\left(f(x_n,w,b)[k] - \boxed{\log\sum_{j=1}^{K}\exp}(f(x_n,w,b)[j])\right)
\end{aligned}
$$

This is the *log-sum-exp* operator!
It approximates *maximum* operator.

log-sum-exp permits numerically-efficient (avoid overflow/underflow) implementation since

$$
\log\sum_{i=1}^{K}\exp(x_i) = x^* - \log\exp(x^*) + \log\sum_{i=1}^{K}\exp(x_i) = x^* + \log\sum_{i=1}^{K}\exp(x_i - x^*) \qquad x^* = \max\{x_1,\ldots,x_K\}
$$

In practice, the *softmax+cross-entropy* is implemented via this log-sum-exp trick!

More About Softmax

Softmax is an approximation to the argmax.

More About Softmax

Softmax is an approximation to the argmax.

To see this, we can change the base of the power $e \to e^{\frac{1}{\beta}}$

Then softmax becomes

$$\mathrm{softmax}(x)[i] = \frac{\exp(\frac{1}{\beta}x[i])}{\sum_{k=1}^{K}\exp(\frac{1}{\beta}x[k])}$$

More About Softmax

Softmax is an approximation to the argmax.

To see this, we can change the base of the power $e \to e^{\frac{1}{\beta}}$

Then softmax becomes

$$\text{softmax}(x)[i] = \frac{\exp(\frac{1}{\beta}x[i])}{\sum_{k=1}^{K} \exp(\frac{1}{\beta}x[k])}$$

We have

$$\lim_{\beta \to 0} \text{softmax}(x) = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1 and } k = \text{argmax}_i x[i]}$$

Why?

More About Softmax

Softmax is an approximation to the argmax.

To see this, we can change the base of the power $e \to e^{\frac{1}{\beta}}$

Then softmax becomes

$$\text{softmax}(x)[i] = \frac{\exp(\frac{1}{\beta}x[i])}{\sum_{k=1}^{K} \exp(\frac{1}{\beta}x[k])}$$

We have

$$\lim_{\beta \to 0} \text{softmax}(x) = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]}_{\text{k-th entry is 1 and } k = \text{argmax}_i x[i]}$$

Why? Also,

$$\lim_{\beta \to \infty} \text{softmax}(x) = [\frac{1}{K}, \ldots, \frac{1}{K}]$$

β is often called as *temperature*.

References

[1] Vapnik, V. (1999). The nature of statistical learning theory. Springer science & business media.

[2] Bishop, C. M., & Nasrabadi, N. M. (2006). Pattern recognition and machine learning (Vol. 4, No. 4, p. 738). New York: springer.

[3] Bartlett, P. L., Long, P. M., Lugosi, G., & Tsigler, A. (2020). Benign overfitting in linear regression. Proceedings of the National Academy of Sciences, 117(48), 30063-30070.

Questions?