Regression and Classification

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2020

Table of Contents

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?
    - 1×1 (inner product)
  - What is the dimension of $\mathbf{v}\mathbf{v}^T$?

Some notations and simple linear algebra

- A scalar s is lower-case and normal font
- A vector $\mathbf{v}$ is lower-case and bold
  - By convention, we always stick with column vectors
- A matrix M is upper-case
- M^T is the transpose of the matrix M
  - If $B = A^T$, $b_{ij} = a_{ji}$
- Quiz: for a n-dim vector $\mathbf{v}$,
  - What is the dimension of $\mathbf{v}^T\mathbf{v}$?
    - 1×1 (inner product)
  - What is the dimension of $\mathbf{v}\mathbf{v}^T$?
    - $n \times n$ (outer product)

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right)$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} x_2^2 x_3 \\ 2(x_1 + 2)x_2 x_3 \\ (x_1 + 2)x_2^2 \end{pmatrix}$$

and $\nabla f(\mathbf{x})|_{(0,1,0)}$?

A quick review of gradient

For a vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)^T$, the gradient of a scalar multivariate function $f(\mathbf{x})$ is denoted by $\nabla f(\mathbf{x})$

- Note that $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$
- $\nabla f(\mathbf{x})$ is a vector pointing to the steepest (ascending) direction
- The magnitude $\|\nabla f(\mathbf{x})\|$ is the slope along that steepest direction

E.g., $f((x_1, x_2, x_3)) = (x_1 + 2)x_2^2 x_3$

$$\nabla f(\mathbf{x}) = \begin{pmatrix} x_2^2 x_3 \\ 2(x_1 + 2)x_2 x_3 \\ (x_1 + 2)x_2^2 \end{pmatrix}$$

and $\nabla f(\mathbf{x})|_{(0,1,0)}$?

$$\nabla f(\mathbf{x})|_{(0,1,0)} = (0, 0, 2)^T$$

Empirical risk minimization

- The goal of supervised learning is to minimize generalization error
- If we know the data distribution p_{data}, we can train and select the optimal model parameter $\hat{\theta}$ by simply minimizing a risk (cost)

$$E_{(\mathbf{x},y)\sim p_{data}}[L(f(\mathbf{x};\theta),y)]$$

- Note that however we don't know p_{data} in general, instead we typically are only given some training data $(\mathbf{x}^{(1)},y^{(1)}),(\mathbf{x}^{(2)},y^{(2)}),\cdots,(\mathbf{x}^{(N)},y^{(N)})$. So instead, we may minimize the empirical risk

$$\frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}^{(i)};\theta),y^{(i)})$$

Loss function for regression

Let us start with the regression problem. Recall from previously that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective, risk) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

Loss function for regression

Let us start with the regression problem. Recall from previously that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective, risk) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

- During training, a reasonable goal will simply be to

$$\min_W \sum_i L(f(x_i; W), y_i),$$

where in the objective function, we are summing the corresponding loss over all pair of training data

Loss function for regression

Let us start with the regression problem. Recall from previously that

- We are trying to learn a function $f(x; W)$ such that for training input x_i and desired output y_i, $f(x_i; W) \sim y_i$

We can define a loss (aka cost, objective, risk) function $L(\cdot, \cdot)$ to measure the discrepancy between the desired output and the actual output

- During training, a reasonable goal will simply be to

$$\min_W \sum_i L(f(x_i; W), y_i),$$

  where in the objective function, we are summing the corresponding loss over all pair of training data

- For regression, it is common to use mean square error for loss function, i.e., $l(f(x_i; W), y_i) = (f(x_i; W) - y_i)^2$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$
- Write $X_{train} = \left(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\right)$ and $\mathbf{y}_{train} = (y_1, y_2, \cdots, y_N)^T$, we want

$$\mathbf{y}_{train} \sim X_{train}^T \mathbf{w}$$

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

$$L(\mathbf{w})$$

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

$$
\begin{aligned}
& L(\mathbf{w}) \\
=& \frac{1}{2}(\mathbf{y}_{train} - X_{train}^T \mathbf{w})^T (\mathbf{y}_{train} - X_{train}^T \mathbf{w})
\end{aligned}
$$

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

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

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train} X_{train}^T)^{-1} X_{train}}\, \mathbf{y}_{train}$$

Linear regression – analytical solution

Assume that the mean square loss is used. We want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train} X_{train}^T)^{-1} X_{train}}_{(X_{train}^T)^\dagger} \mathbf{y}_{train}$$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$

Linear regression

For example, try to predict the mass (weight) of a man based on his height, bmi, and his age (assuming we don't know what bmi is here)

- E.g., height = 1.8 m, bmi = 23, age = 29, what is his mass?
- For linear regression, we assume $y \sim \mathbf{x}^T\mathbf{w}$
  - $\mathbf{x} = (1.8, 23, 29, 1)^T$
  - $\mathbf{w} = (w_1, w_2, w_3, b)^T$ is an unknown weight vector
  - N.B. we append the feature vector by 1 to make the expression more compact. b is a bias weight
- Given training data, we need to find $\mathbf{w}$
  - $\mathbf{x}_1 = (1.68, 31.80, 43.34, 1)^T$, $y_1 = 87.50$
  - $\mathbf{x}_2 = (1.80, 33.11, 16.69, 1)^T$, $y_2 = 110.06$
  - $\cdots$
  - $\mathbf{x}_N = (1.83, 33.79, 43.30, 1)^T$, $y_N = 112.33$
- Write $X_{train} = \left(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\right)$ and $\mathbf{y}_{train} = (y_1, y_2, \cdots, y_N)^T$, we want

$$\mathbf{y}_{train} \sim X_{train}^T\mathbf{w}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$L(\mathbf{w})$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

$$
\begin{aligned}
& L(\mathbf{w}) \\
=& \frac{1}{2}(\mathbf{y}_{train} - X_{train}^T \mathbf{w})^T (\mathbf{y}_{train} - X_{train}^T \mathbf{w})
\end{aligned}
$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

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

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train} X_{train}^T)^{-1} X_{train}}\, \mathbf{y}_{train}$$

Linear regression – analytical solution

Assume that mean square loss is used, we want to minimize

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

- Let's compute the gradient (try it out at home),

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X_{train} \mathbf{y}_{train} + X_{train} X_{train}^T \mathbf{w}$$

- At the optimum, we expect $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$. Thus

$$\mathbf{w} = \underbrace{(X_{train} X_{train}^T)^{-1} X_{train}}_{(X_{train}^T)^\dagger} \mathbf{y}_{train}$$

Experiment

- $\text{mass} = \text{bmi} \times \text{height}^2$

Experiment

- mass $=$ bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg

Experiment

- mass = bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)

Experiment

- mass = bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)
- The weights are quite reasonable
  - mass should not really depend on age
  - height should have a stronger effect to mass than bmi

Experiment

- mass = bmi $\times$ height^2
- We generated 30 training data points and wiggled the masses with Gaussian noises of a standard deviation of 3 kg
- Trained weights: (1.17e+02, 3.11, 8.97e-03, -2.05e+02) # (height,bmi,age,1)
- The weights are quite reasonable
  - mass should not really depend on age
  - height should have a stronger effect to mass than bmi
- MSE: 6.63. It is a bit high, let's try to reduce it

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$$

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$$

- We can do linear regression just as before, just the number of weights increases from 4 to 10

Expanding features...

- Let's include some higher "order" features. For the raw feature x_1, x_2, x_3, we can also include products of them as a feature. So a new feature vector becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$$

- We can do linear regression just as before, just the number of weights increases from 4 to 10
- MSE: 1.01. Nice!

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1x_2x_3$ and $x_1^2x_2$. So the new feature vector now becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, \cdots)$$

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1x_2x_3$ and $x_1^2x_2$. So the new feature vector now becomes

$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, \cdots)$$

- Again we will do linear regression as before, the number of weights now increases from to 25

Expanding features (con't)...

- Let's go even higher order and also include products like $x_1x_2x_3$ and $x_1^2x_2$. So the new feature vector now becomes

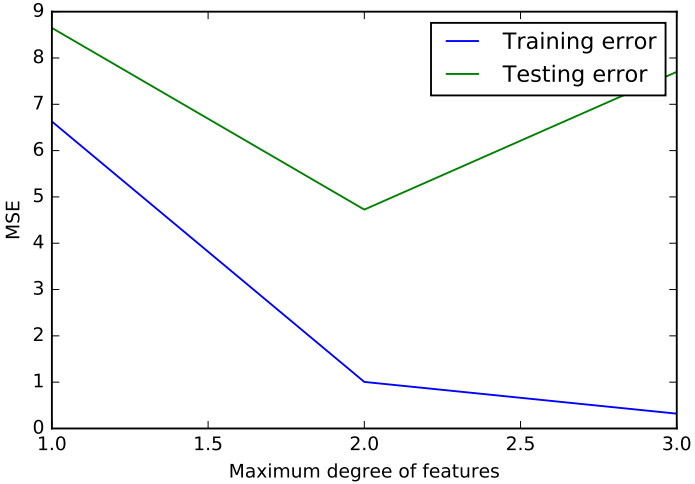
$$(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, \cdots)$$

- Again we will do linear regression as before, the number of weights now increases from to 25
- MSE: 0.32...

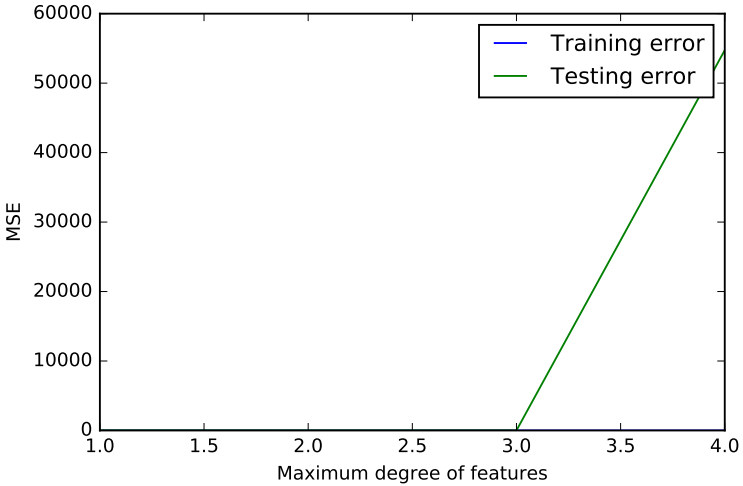
Expanding features (con't)...

- We can go further to the 4-th order and the number of weights now increases to 70
- MSE: 1.13e-12. Wow!

Wait, how about testing error?



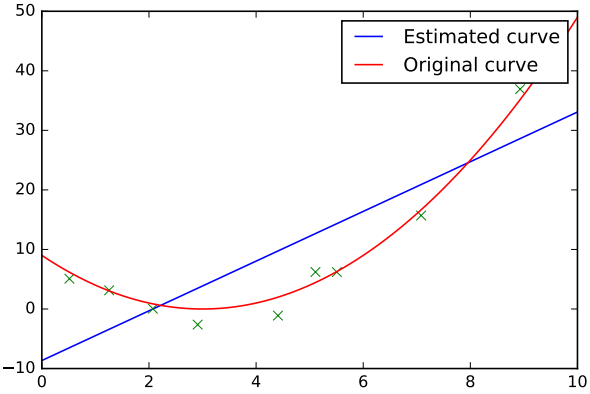
Wait, how about testing error...? Oops



Curve fitting

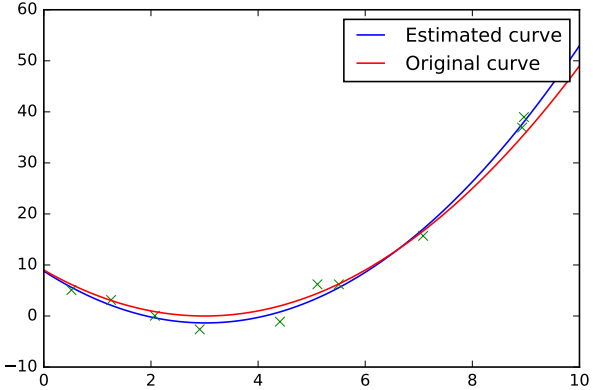
Why is it so bad for testing? Let's visit another even simpler example

- Let's try to fit a quadratic curve $y = (x-3)^2$ with linear regression. And again our training data will be wiggled a little bit by a Gaussian noise



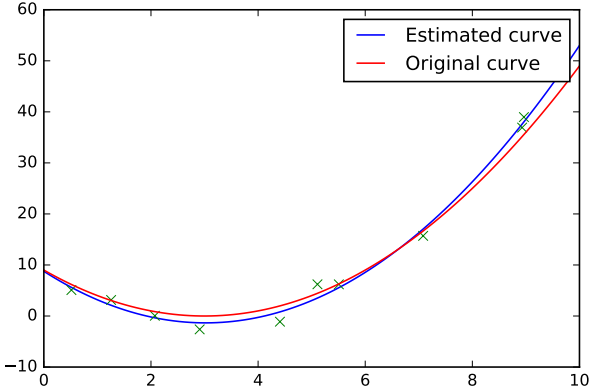
Curve fitting (2nd order)

Let's include higher order feature just as before. Take $(1, x, x^2)$ as feature by including x^2



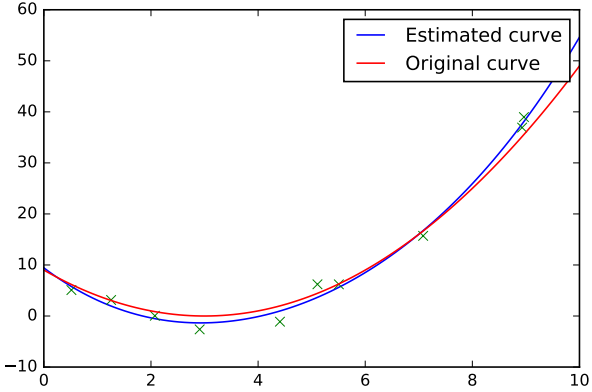
Curve fitting (3rd order)

$(1, x, x^2, x^3)$



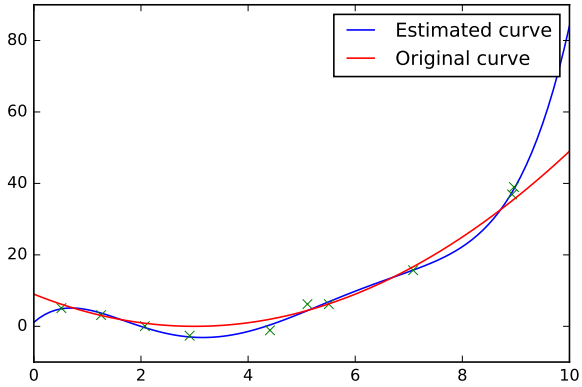
Curve fitting (4th order)

$(1, x, x^2, x^3, x^4)$



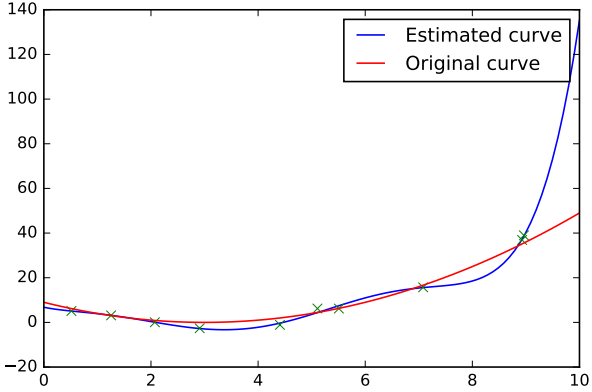
Curve fitting (5th order)

$(1, x, x^2, x^3, x^4, x^5)$



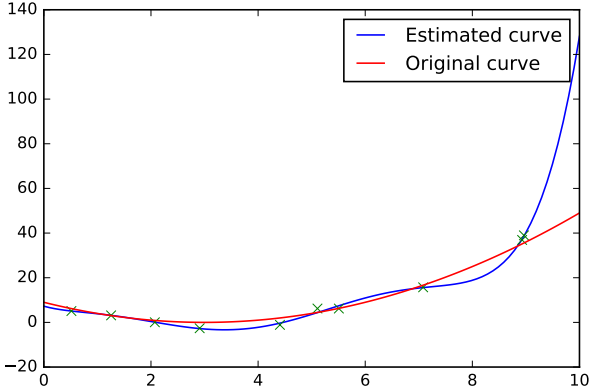
Curve fitting (6rd order)

$(1, x, x^2, x^3, x^4, x^5, x^6)$



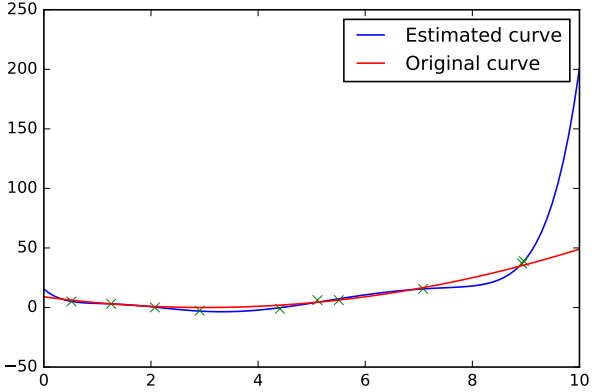
Curve fitting (7rd order)

$(1, x, x^2, x^3, x^4, x^5, x^6, x^7)$



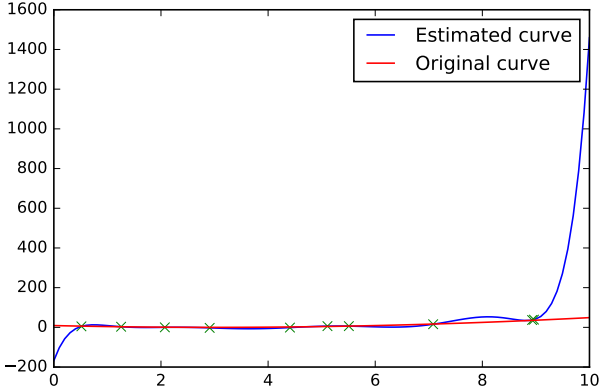
Curve fitting (8th order)

$(1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8)$

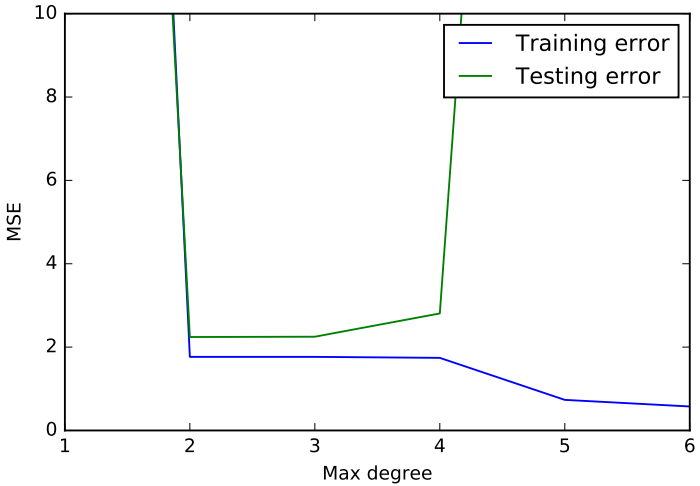


Curve fitting (9th order)

$(1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9)$



Overfitting vs underfitting



Lesson learned

- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts

Lesson learned

- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...

Lesson learned

- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don’t actually know the true objective function
  - Loss function is just an approximated goal

Lesson learned

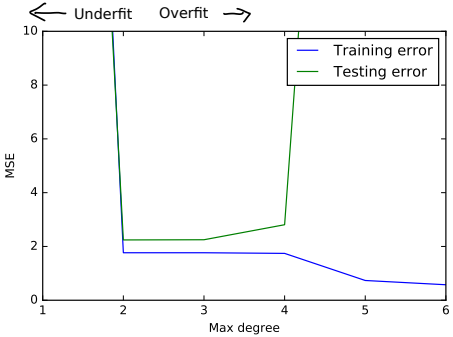
- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don’t actually know the true objective function
  - Loss function is just an approximated goal
- Should try to avoid neither overfitting nor underfitting

Lesson learned

- Given sufficiently complex model, we can learn “anything”, but ...
  - Machine learning is all about generalization
  - It is testing error but not training error that actually counts
- Machine learning is very similar to optimization, we just try to find our best model by minimizing a loss function, but...
  - Unlike optimization, we don’t actually know the true objective function
  - Loss function is just an approximated goal
- Should try to avoid neither overfitting nor underfitting
  - Everything should be made as simple as possible, but not simpler – Albert Einstein
  - Occam’s razor: overly complex model is not a good thing (if you don’t have sufficient data to fit the model)

High-bias vs high-variance

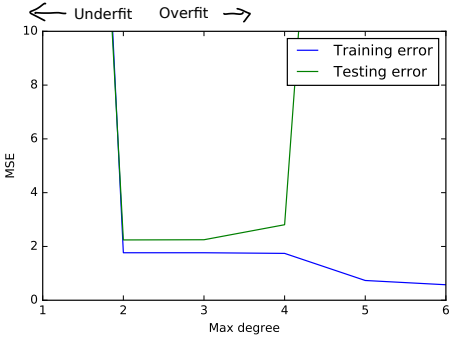
Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias



High-bias vs high-variance

Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias

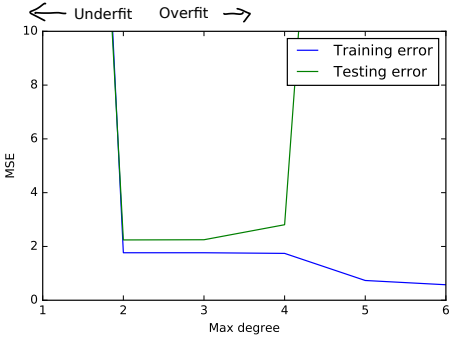
- High-bias: model is too rigid to learn (thus biased) and it cannot adapt to the data



High-bias vs high-variance

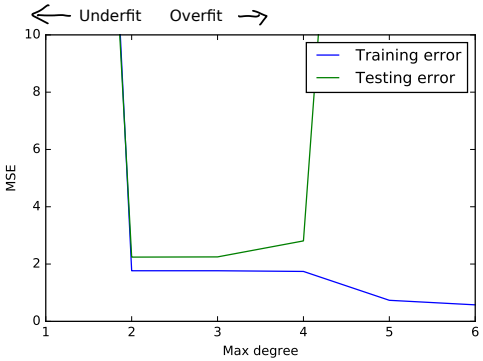
Sometimes we also refer to overfitting and underfitting roughly as high-variance and high-bias

- High-bias: model is too rigid to learn (thus biased) and it cannot adapt to the data
- High-variance: model is too elastic and can fit any arbitrary data. When fitted with different training data, the weights just converge to totally different values (thus high variance)



More on overfitting (high-variance)

- In the high-variance domain, the model is essentially learning the training data noise. That's why weights converge to different values for different training data
- Model complexity is relative. If more training data are available, the model used to be overfitted may not be overfitted anymore. So should we change a model every time we added new data?!



Regularization

Rather than using a simple model, we could restrain a more complex model from running wild with additional constraints. This process is commonly known as regularization

- As regularization can mitigate the overfitting problem, we can use a more expressive model even when we have only few data. And the same model can be used as data size increases
- A regularized complex model often outperforms an unregularized simple model

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$ in the loss objective function. Thus

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$$

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$ in the loss objective function. Thus

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

- And the gradient is

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X\mathbf{y} + [XX^T + \lambda I]\ \mathbf{w}$$

Ridge regression

A most common type of regularization is by restraining the magnitudes of the weights

- For example, in ridge regression, we try to achieve this by simply including $\frac{1}{2}\lambda \mathbf{w}^T\mathbf{w}$ in the loss objective function. Thus

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

- And the gradient is

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = -X\mathbf{y} + [XX^T + \lambda I]\ \mathbf{w}$$

- As before, if we set $\nabla_{\mathbf{w}} L(\mathbf{w}) = 0$, we have

$$\mathbf{w} = [XX^T + \lambda I]^{-1} X\mathbf{y}$$

Lasso

- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_1$ is added to the loss objective function Thus, we want to

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda \|\mathbf{w}\|_1,$$

where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$

Lasso

- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_1$ is added to the loss objective function Thus, we want to

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda\|\mathbf{w}\|_1,$$

  where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$
- Unlike ridge regression, one cannot write the close form solution directly though
  - But a local optimum can be found with iterative soft-thresholding
  - For the next several slides, I just used sciki-learn libraryin Python

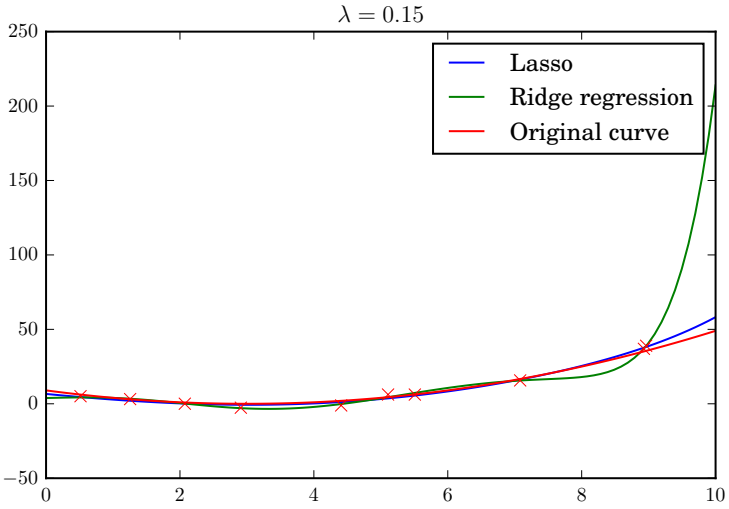
Lasso

- Another common regularization is lasso. Instead of $\lambda \mathbf{w}^T \mathbf{w}$, the scaled l_1-norm of $\mathbf{w}$, $\lambda \|\mathbf{w}\|_1$ is added to the loss objective function Thus, we want to

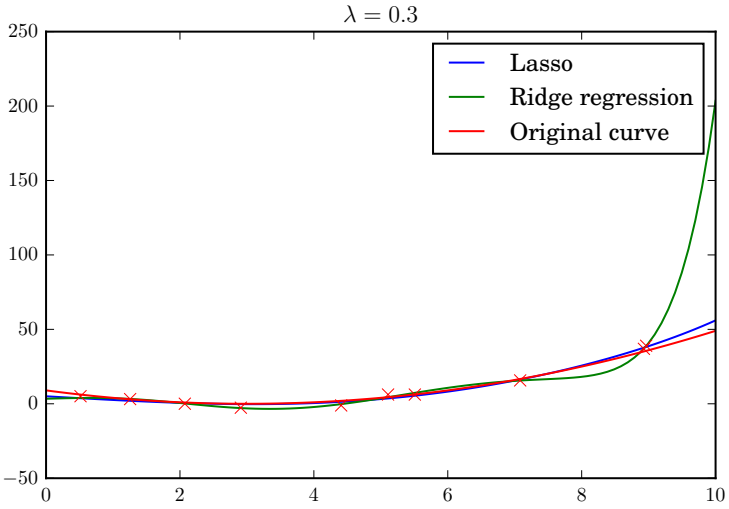
$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}) + \lambda\|\mathbf{w}\|_1,$$

  where $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \cdots + |w_D|$
- Unlike ridge regression, one cannot write the close form solution directly though
  - But a local optimum can be found with iterative soft-thresholding
  - For the next several slides, I just used sciki-learn libraryin Python
- Lasso tends to enforce a sparse weight solution. It was popular several years ago because of compressed sensing

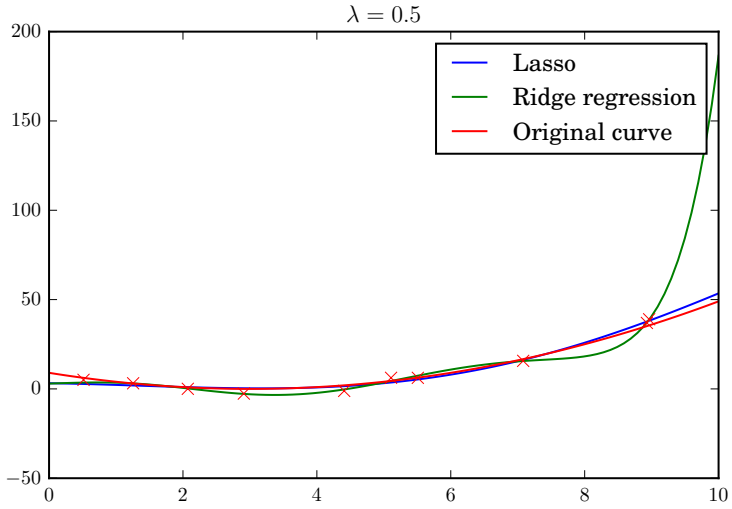
Curve fitting with Lasso and ridge regression (degree=9)



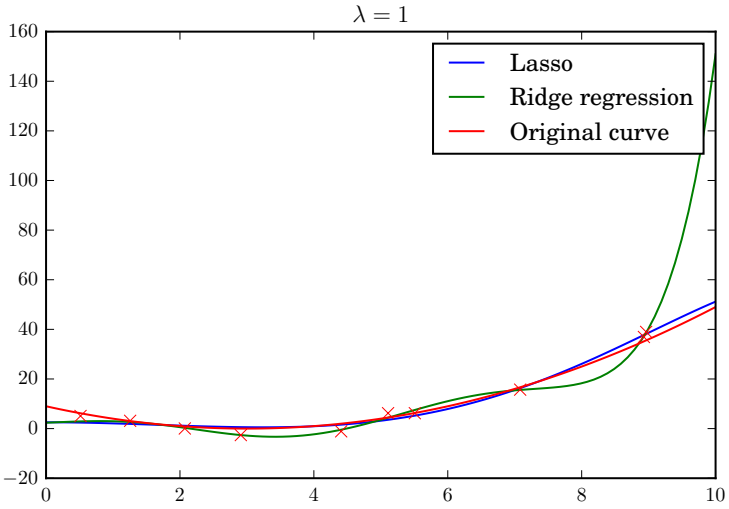
Curve fitting with Lasso and ridge regression (degree=9)



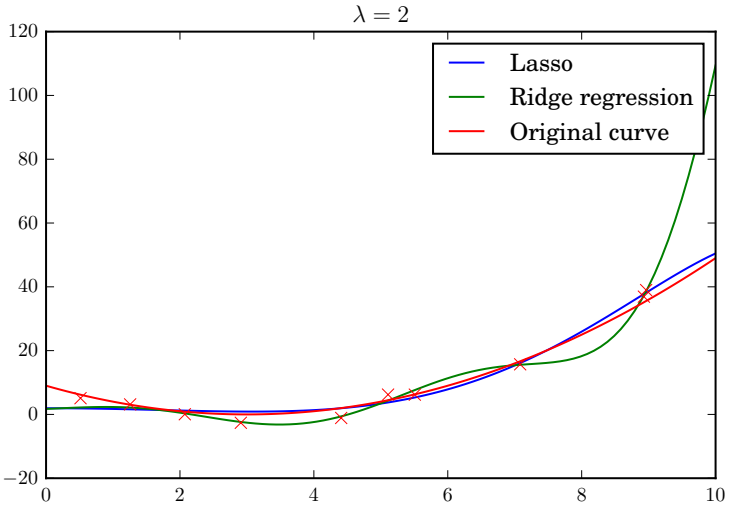
Curve fitting with Lasso and ridge regression (degree=9)



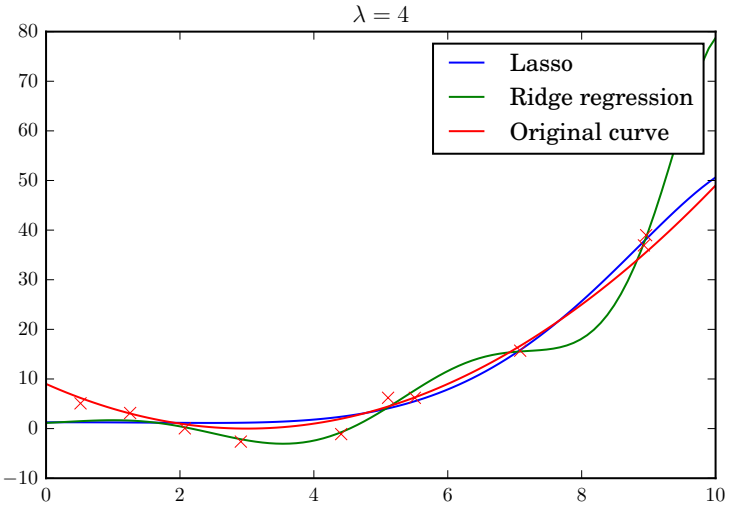
Curve fitting with Lasso and ridge regression (degree=9)



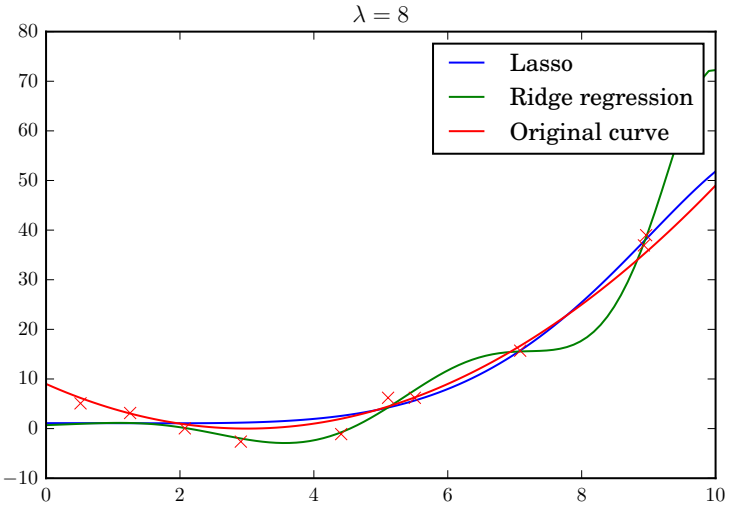
Curve fitting with Lasso and ridge regression (degree=9)



Curve fitting with Lasso and ridge regression (degree=9)



Curve fitting with Lasso and ridge regression (degree=9)



Conclusion

- Machine learning is all about generalization (from data)
- One can decrease the training error to arbitrarily small (by increasing model complexity)
- On the other hand, we really only care about test error, which is composed of
  - Bias: High bias when model is too rigid (model complexity is too low) to adapt to the training data
  - Variance: High variance when model is too flexible (model complexity is too high) that different sets of training data will converge to completely different weight parameters
- Occam's razor: a good explanation should be minimal

Conclusion

- For supervised learning systems (both classification and regression), we can typically reduce it to an optimization problem of minimizing a loss function (instead of training error) w.r.t. some weights
- Regularization terms can typically be incorporated in the loss function to keep the weights from running wild
- It is almost always better to use a more complex but regularized model than a simple model when one has sufficient training data
  - Provided that one regularized wisely
  - That is why deep neural networks typically work better

Conclusion

- For supervised learning systems (both classification and regression), we can typically reduce it to an optimization problem of minimizing a loss function (instead of training error) w.r.t. some weights
- Regularization terms can typically be incorporated in the loss function to keep the weights from running wild
- It is almost always better to use a more complex but regularized model than a simple model when one has sufficient training data
  - Provided that one regularized wisely
  - That is why deep neural networks typically work better
    - Actually with sufficient data, we don't need to worry about overfitting

Conclusion

- For supervised learning systems (both classification and regression), we can typically reduce it to an optimization problem of minimizing a loss function (instead of training error) w.r.t. some weights
- Regularization terms can typically be incorporated in the loss function to keep the weights from running wild
- It is almost always better to use a more complex but regularized model than a simple model when one has sufficient training data
  - Provided that one regularized wisely
  - That is why deep neural networks typically work better
    - Actually with sufficient data, we don't need to worry about overfitting
    - Furthermore, sometimes you may even want to overfit a small training set (attain 0 training error but large testing error) just to make sure your model is correct

New perspective?!

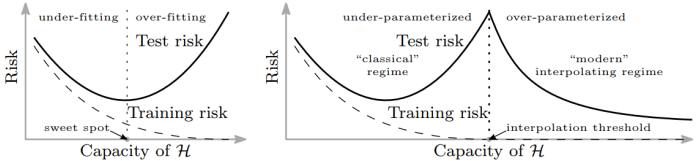


Figure 1: **Curves for training risk (dashed line) and test risk (solid line).** (**a**) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (**b**) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)
  - E.g., for a very small 2×2 image patch $\begin{pmatrix} 10 & 25 \\ 36 & 90 \end{pmatrix}$, it will be converted to

$$\mathbf{x} = (10, 25, 36, 90, 1)^T$$

Linear classification

The same linear regression idea can be transferred to classification problems

- Consider binary classification whether an image contains a cat or not
  - We can first vectorize the input image into a column vector $\mathbf{x}$ (with an extra 1 appended to account for bias)
  - E.g., for a very small 2×2 image patch $\begin{pmatrix} 10 & 25 \\ 36 & 90 \end{pmatrix}$, it will be converted to

$$\mathbf{x} = (10, 25, 36, 90, 1)^T$$

  - We will decide if the image contains a cat of not by verifying if

$$\mathbf{x}^T \mathbf{w} \lessgtr 0,$$

    where we will need to obtain the weight $\mathbf{w}$ through training (more later)

Logistic regression

- We can introduce a scoring function

$$f(\mathbf{x}; \mathbf{w}) = H(\mathbf{x}^T \mathbf{w}),$$

where $H(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$ is a step function and we have a cat if $f(\mathbf{x}; \mathbf{w}) = 1$ and no cat if $f(\mathbf{x}; \mathbf{w}) = 0$

Logistic regression

- We can introduce a scoring function

$$f(\mathbf{x}; \mathbf{w}) = H(\mathbf{x}^T \mathbf{w}),$$

where $H(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$ is a step function and we have a cat if $f(\mathbf{x}; \mathbf{w}) = 1$ and no cat if $f(\mathbf{x}; \mathbf{w}) = 0$

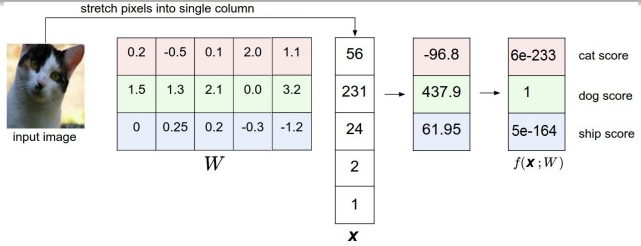
- Note that $f(\mathbf{x}; \mathbf{w})$ essentially is a perceptron model and is difficult to train because of the discontinuity of $H(\cdot)$. Instead, we could replace $H(\cdot)$ by the sigmoid (or logistic) function $S(t) = \frac{1}{1+e^{-t}}$
  - Hence, known as logistic regression

Loss function of logistic regression

Another advantage of using $S(\cdot)$ is that we can interpret the output as probability and then the loss function can be specified by a "cross-entropy loss" as follows (will explain next)

$$L(\mathbf{w};\mathbf{x}) = \begin{cases} -\log f(\mathbf{x};\mathbf{w}), & \text{if the image is a cat} \\ -\log(1 - f(\mathbf{x};\mathbf{w})), & \text{otherwise} \end{cases}$$

Softmax classifier

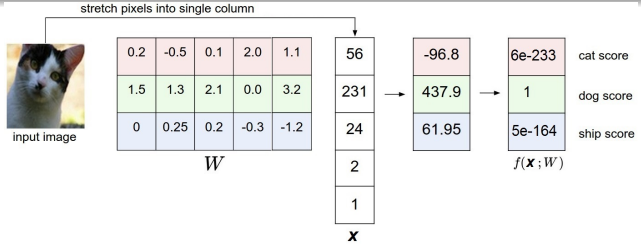


- For multiclass problem, we can extend the logistic scoring function to

$$f_i(\mathbf{x}; W) = \sigma_i(W\mathbf{x}),$$

where $\sigma_i(\mathbf{y}) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$ is known as a softmax function and is really just a normalized exponential function

Softmax classifier



- For multiclass problem, we can extend the logistic scoring function to

$$f_i(\mathbf{x}; W) = \sigma_i(W\mathbf{x}),$$

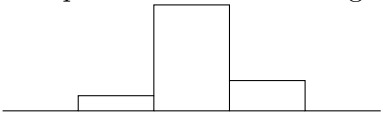
where $\sigma_i(\mathbf{y}) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$ is known as a softmax function and is really just a normalized exponential function

- Again, we can interpret $f_i(\mathbf{x}; W)$ as the estimated probability of $\mathbf{x}$ belong to class i
  - E.g., $p(cat; \mathbf{x}, W) = f_{cat}(\mathbf{x}; W)$

Surrogate loss function

- Both classifiers below will result in zero prediction error if the ground truth is dog

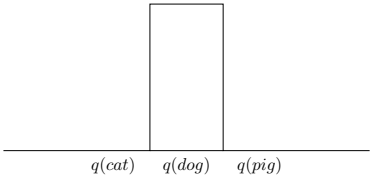
Classifier A



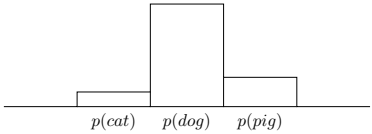
Classifier B

- However, Classifier B is apparently better than Classifier A. Using zero-one loss will not be able to distinguish them though.
- A surrogate loss function should be used instead. The most common one is the cross-entropy loss function

Cross entropy loss function



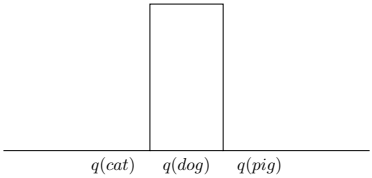
Actual



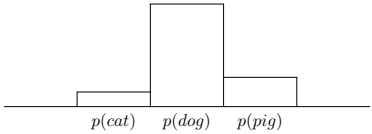
Estimate

- Let say the image is actually a dog. We can express this as a distribution as shown on the left

Cross entropy loss function



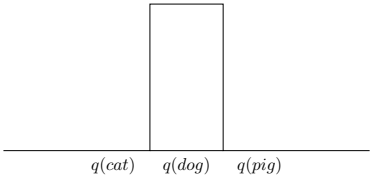
Actual



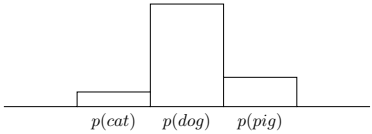
Estimate

- Let say the image is actually a dog. We can express this as a distribution as shown on the left
- Ideally we would like the estimated probability distribution matches the actual one

Cross entropy loss function



Actual



Estimate

- Let say the image is actually a dog. We can express this as a distribution as shown on the left
- Ideally we would like the estimated probability distribution matches the actual one
- We can measure the difference between two distributions with KL-divergence given by

$$KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i}$$

KL-divergence is non-negative

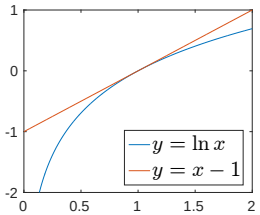
$$
\begin{aligned}
KL(p\|q) &= \sum_i p_i \log_2 \frac{p_i}{q_i} \\
&= -\sum_i p_i \log_2 \frac{q_i}{p_i}
\end{aligned}
$$

KL-divergence is non-negative

$$
\begin{aligned}
KL(p\|q) &= \sum_i p_i \log_2 \frac{p_i}{q_i} \\
&= -\sum_i p_i \log_2 \frac{q_i}{p_i} \\
&= -\sum_i \frac{p_i}{\ln 2} \ln \frac{q_i}{p_i}
\end{aligned}
$$

KL-divergence is non-negative

$$
\begin{aligned}
KL(p\|q) &= \sum_i p_i \log_2 \frac{p_i}{q_i} \\
&= -\sum_i p_i \log_2 \frac{q_i}{p_i} \\
&= -\sum_i \frac{p_i}{\ln 2} \ln \frac{q_i}{p_i}
\end{aligned}
$$

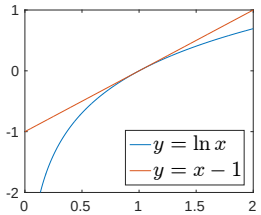


Fact

For any real x, $\ln(x) \leq x - 1$. Moreover, the equality only holds when $x = 1$

KL-divergence is non-negative

$$
\begin{aligned}
KL(p\|q) &= \sum_i p_i \log_2 \frac{p_i}{q_i} \\
&= -\sum_i p_i \log_2 \frac{q_i}{p_i} \\
&= -\sum_i \frac{p_i}{\ln 2} \ln \frac{q_i}{p_i} \\
&\geq -\sum_i \frac{p_i}{\ln 2} \left(\frac{q_i}{p_i} - 1 \right)
\end{aligned}
$$

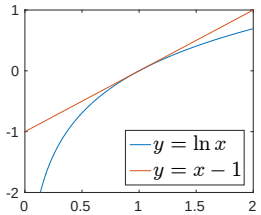


Fact

For any real x, $\ln(x) \leq x - 1$. Moreover, the equality only holds when $x = 1$

KL-divergence is non-negative

$$
\begin{aligned}
KL(p\|q) &= \sum_i p_i \log_2 \frac{p_i}{q_i} \\
&= -\sum_i p_i \log_2 \frac{q_i}{p_i} \\
&= -\sum_i \frac{p_i}{\ln 2} \ln \frac{q_i}{p_i} \\
&\geq -\sum_i \frac{p_i}{\ln 2} \left(\frac{q_i}{p_i} - 1 \right) \\
&= \frac{1}{\ln 2} \left(\sum_i p_i - \sum_i q_i \right) = 0
\end{aligned}
$$



Fact

For any real x, $\ln(x) \leq x - 1$. Moreover, the equality only holds when $x = 1$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$
  - It is not an actual distant measure: $KL(q||p) \neq KL(p||q)$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$
  - It is not an actual distant measure: $KL(q||p) \neq KL(p||q)$
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W; \mathbf{x}) = KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i} = \underbrace{-\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i$$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$
  - It is not an actual distant measure: $KL(q||p) \neq KL(p||q)$
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W; \mathbf{x}) = KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i} = \underbrace{-\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})}$$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$
  - It is not an actual distant measure: $KL(q||p) \neq KL(p||q)$
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i} = \underbrace{-\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})} = -\log f_{j(\mathbf{x})}(\mathbf{x};W) = -\log \sigma_{j(\mathbf{x})}(W\mathbf{x}),$$

where $j(\mathbf{x})$ is the actual class index of $\mathbf{x}$

Cross entropy loss function (con't)

- KL-divergence is a way to estimate the difference between two distribution
  - $KL(q\|p) \geq 0$ and $KL(q||p) = 0$ if and only if $q \equiv p$
  - It is not an actual distant measure: $KL(q||p) \neq KL(p||q)$
- We can pick $KL(q\|p)$ as the loss function, then

$$L(W;\mathbf{x}) = KL(q\|p) = \sum_i q_i \log \frac{q_i}{p_i} = \underbrace{-\left[-\sum_i q_i \log q_i\right]}_{entropy} + \underbrace{\left[-\sum_i q_i \log p_i\right]}_{cross-entropy}$$

$$= -\sum_i q_i \log p_i = -\log p_{j(\mathbf{x})} = -\log f_{j(\mathbf{x})}(\mathbf{x};W) = -\log \sigma_{j(\mathbf{x})}(W\mathbf{x}),$$

  where $j(\mathbf{x})$ is the actual class index of $\mathbf{x}$
- The total loss is just sum over all training $\mathbf{x}$: $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x})$

Optimization

- For linear regression and ridge regression, we have a close form solution for minimizing the loss function but in most other models, we do not

Optimization

- For linear regression and ridge regression, we have a close form solution for minimizing the loss function but in most other models, we do not
- In practice, to minimize the loss function w.r.t. the weight W, we can use simple steepest descent. That is,

$$W = W - \Delta W \quad \text{with} \quad \Delta W = \epsilon \nabla_W L(W),$$

where ϵ is the learning rate and suppose to be small. It is often just set heuristically. We may talk more about it later in this course

Optimization

- For linear regression and ridge regression, we have a close form solution for minimizing the loss function but in most other models, we do not
- In practice, to minimize the loss function w.r.t. the weight W, we can use simple steepest descent. That is,

$$W = W - \Delta W \quad \text{with} \quad \Delta W = \epsilon \nabla_W L(W),$$

  where ϵ is the learning rate and suppose to be small. It is often just set heuristically. We may talk more about it later in this course
- So to optimize, we need to find the gradient of L wrt W

Derivative of softmax loss

- Recall that $L(W) = \sum_{\mathbf{x}} L(W; \mathbf{x}) = -\sum_{\mathbf{x}} \sum_l q_l^{(\mathbf{x})} \log \sigma_l(W\mathbf{x})$, where $q_j^{(\mathbf{x})}$ is non-zero $(= 1)$ only when j is the true label of $\mathbf{x}$

Derivative of softmax loss

- Recall that $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x}) = -\sum_{\mathbf{x}} \sum_l q_l^{(\mathbf{x})} \log \sigma_l(W\mathbf{x})$, where $q_j^{(\mathbf{x})}$ is non-zero $(=1)$ only when j is the true label of $\mathbf{x}$
- $\nabla L(W) = \sum_{\mathbf{x}} \nabla L(W;\mathbf{x})$. Let's focus on computing the individual gradient $\nabla L(W;\mathbf{x})$

Derivative of softmax loss

- Recall that $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x}) = -\sum_{\mathbf{x}} \sum_l q_l^{(\mathbf{x})} \log \sigma_l(W\mathbf{x})$, where $q_j^{(\mathbf{x})}$ is non-zero $(=1)$ only when j is the true label of $\mathbf{x}$
- $\nabla L(W) = \sum_{\mathbf{x}} \nabla L(W;\mathbf{x})$. Let's focus on computing the individual gradient $\nabla L(W;\mathbf{x})$
- Write $L(W;\mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$, where $\mathbf{o} = W\mathbf{x}$. And we drop the superscript $(\mathbf{x})$ for clarity

Derivative of softmax loss

- Recall that $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x}) = -\sum_{\mathbf{x}} \sum_l q_l^{(\mathbf{x})} \log \sigma_l(W\mathbf{x})$, where $q_j^{(\mathbf{x})}$ is non-zero $(=1)$ only when j is the true label of $\mathbf{x}$
- $\nabla L(W) = \sum_{\mathbf{x}} \nabla L(W;\mathbf{x})$. Let's focus on computing the individual gradient $\nabla L(W;\mathbf{x})$
- Write $L(W;\mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$, where $\mathbf{o} = W\mathbf{x}$. And we drop the superscript $(\mathbf{x})$ for clarity
  - Using chain rule,

$$\frac{\partial}{\partial w_{i,j}} L(W;\mathbf{x}) = \sum_k \frac{\partial}{\partial o_k} L(W;\mathbf{x}) \frac{\partial o_k}{\partial w_{i,j}} = x_j \frac{\partial}{\partial o_i} L(W;\mathbf{x})$$

Derivative of softmax loss

- Recall that $L(W) = \sum_{\mathbf{x}} L(W;\mathbf{x}) = -\sum_{\mathbf{x}} \sum_l q_l^{(\mathbf{x})} \log \sigma_l(W\mathbf{x})$, where $q_j^{(\mathbf{x})}$ is non-zero $(=1)$ only when j is the true label of $\mathbf{x}$
- $\nabla L(W) = \sum_{\mathbf{x}} \nabla L(W;\mathbf{x})$. Let's focus on computing the individual gradient $\nabla L(W;\mathbf{x})$
- Write $L(W;\mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$, where $\mathbf{o} = W\mathbf{x}$. And we drop the superscript $(\mathbf{x})$ for clarity
  - Using chain rule,

$$\frac{\partial}{\partial w_{i,j}} L(W;\mathbf{x}) = \sum_k \frac{\partial}{\partial o_k} L(W;\mathbf{x}) \frac{\partial o_k}{\partial w_{i,j}} = x_j \frac{\partial}{\partial o_i} L(W;\mathbf{x})$$

  - We need to find $\frac{\partial}{\partial o_i} L(W;\mathbf{x})$

$\frac{\partial}{\partial o_i} L(W; \mathbf{x})$

- Recall $L(W; \mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$ and[1] $\sigma_l(\mathbf{o}) = \frac{\exp(o_l)}{\sum_k \exp(o_k)}$. It is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1 - \sigma_i(\mathbf{o}))$.

[1] $\mathbf{o} = W\mathbf{x}$

$\frac{\partial}{\partial o_i} L(W; \mathbf{x})$

- Recall $L(W; \mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$ and[1] $\sigma_l(\mathbf{o}) = \frac{\exp(o_l)}{\sum_k \exp(o_k)}$. It is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1 - \sigma_i(\mathbf{o}))$. Thus,

$$\begin{aligned}
\frac{\partial}{\partial o_i} L(W; \mathbf{x}) &= -\frac{\partial}{\partial o_i} \sum_l q_l \log \sigma_l(\mathbf{o}) \\
&= \frac{q_i}{\sigma_i}(\sigma_i)(1 - \sigma_i) - \sum_{l \neq i} \frac{q_l}{\sigma_l}\sigma_i\sigma_l = q_i - \sum_l q_l \sigma_i \\
&= q_i - \sigma_i
\end{aligned}$$

[1] $\mathbf{o} = W\mathbf{x}$

$\frac{\partial}{\partial o_i} L(W; \mathbf{x})$

- Recall $L(W; \mathbf{x}) = \sum_l q_l \log \sigma_l(\mathbf{o})$ and[1] $\sigma_l(\mathbf{o}) = \frac{\exp(o_l)}{\sum_k \exp(o_k)}$. It is easy to verify that $\frac{\partial}{\partial o_i}\sigma_j(\mathbf{o}) = -\sigma_i(\mathbf{o})\sigma_j(\mathbf{o})$ and $\frac{\partial}{\partial o_i}\sigma_i(\mathbf{o}) = \sigma_i(\mathbf{o})(1 - \sigma_i(\mathbf{o}))$. Thus,

$$\begin{aligned}
\frac{\partial}{\partial o_i} L(W; \mathbf{x}) &= -\frac{\partial}{\partial o_i} \sum_l q_l \log \sigma_l(\mathbf{o}) \\
&= \frac{q_i}{\sigma_i}(\sigma_i)(1 - \sigma_i) - \sum_{l \neq i} \frac{q_l}{\sigma_l}\sigma_i\sigma_l = q_i - \sum_l q_l \sigma_i \\
&= q_i - \sigma_i
\end{aligned}$$

- Using chain rule

$$\frac{\partial}{\partial w_{i,j}} L(W; \mathbf{x}) = \sum_k \frac{\partial}{\partial o_k} L(W; \mathbf{x}) \frac{\partial o_k}{\partial w_{i,j}} = \frac{\partial}{\partial o_i} L(W; \mathbf{x}) x_j = (q_i - \sigma_i) x_j$$

[1] $\mathbf{o} = W\mathbf{x}$

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
  - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$

Stochastic gradient descent

- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
  - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$
  - But $L(W)$ is really just an approximate as any training set is stochastic in natural in any case. Why not just approximate $L(W)$ not as refined with few data? That is, just pick a subset $\mathcal{X}_i$ from the training set and use

$$L_i(W) = \sum_{\mathbf{x} \in \mathcal{X}_i} L(W; \mathbf{x})$$

  instead. And this is known as the mini-batch gradient descent

Stochastic gradient descent

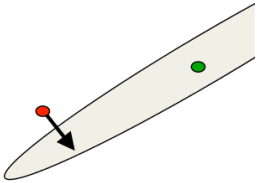
- An immediate issue that one will come across is that the original "full-batch" gradient descent is too slow
  - Recall that $L(W)$ supposes to a sum over individual loss of all training data $L(W; \mathbf{x})$
  - But $L(W)$ is really just an approximate as any training set is stochastic in natural in any case. Why not just approximate $L(W)$ not as refined with few data? That is, just pick a subset $\mathcal{X}_i$ from the training set and use

$$L_i(W) = \sum_{\mathbf{x} \in \mathcal{X}_i} L(W; \mathbf{x})$$

    instead. And this is known as the mini-batch gradient descent
- One may go to the extreme and only pick one $\mathbf{x}$ to estimate the gradient. This formally is known as the stochastic gradient descent. But in practice, no one uses it. But people often say stochastic gradient descent when they actually mean mini-batch gradient descent

Gradient descent with moment

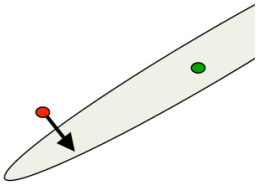
- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle



[1]Slide borrowed from Hinton's coursera course

Gradient descent with moment

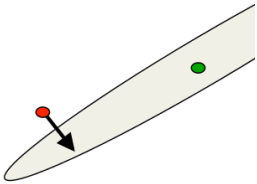
- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle
  - The gradient is big in the direction in which we only want to travel a small distance
  - The gradient is small in the direction in which we want to travel a large distance



[1]Slide borrowed from Hinton's coursera course

Gradient descent with moment

- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle
  - The gradient is big in the direction in which we only want to travel a small distance
  - The gradient is small in the direction in which we want to travel a large distance
- A simple solution is to introduce "momentum" to the change of W. That is,
  $\Delta W = \lambda(\epsilon \nabla_W L(W)) + (1 - \lambda)\Delta W^{(old)}$
- Will talk more about optimization methods later. So much for today



[1]Slide borrowed from Hinton's coursera course

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also
- But gradient of $L(W)$ can easily be computed numerically. For example, say $W = \begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}$,

$$\frac{\partial}{\partial W_{1,1}} L(W) \approx \frac{1}{h}\left[L\left(\begin{pmatrix} 4.1+h & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right) - L\left(\begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right)\right]$$

Remark on computing gradient

- For the previous discussion, we always assume that the gradient can be found analytically. In practice, it may not be true also
- But gradient of $L(W)$ can easily be computed numerically. For example, say $W = \begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}$,

$$\frac{\partial}{\partial W_{1,1}} L(W) \approx \frac{1}{h}\left[L\left(\begin{pmatrix} 4.1+h & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right) - L\left(\begin{pmatrix} 4.1 & 3.3 \\ -1.2 & 2.1 \end{pmatrix}\right)\right]$$

- Actually, the numerical gradient is useful even if an analytical gradient exists. It at least provides a mean to debug your system
  - And luckily, for some packages such as Theano, they automatically find the analytical gradient for you

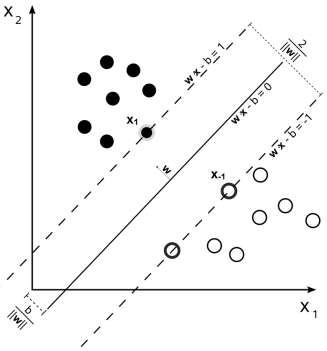
Conclusion

- For classification, we can feed the output of a linear regressor to a logistic function or softmax function to form a linear classifier
  - For only two classes, we have the logistic "regression" classifier
  - For multi-class cases, we have the softmax classifiers

Conclusion

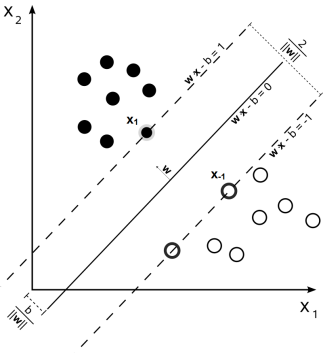
- For classification, we can feed the output of a linear regressor to a logistic function or softmax function to form a linear classifier
  - For only two classes, we have the logistic "regression" classifier
  - For multi-class cases, we have the softmax classifiers
- For finding the optimal weights, we may not be able to get the solution right away analytically (possible though for linear regression and ridge regression)
  - Can optimize iteratively with gradient descent
  - Can speed up gradient descent by using mini-batch instead of full batch
  - Momentum is a common trick to improve optimization efficiency also

SVM



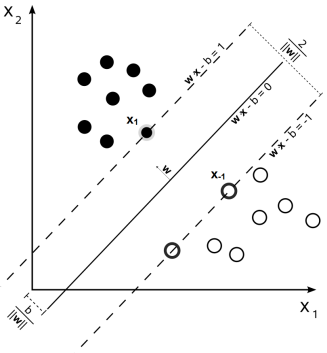
- Denote $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$, $\hat{\mathbf{w}} \cdot \mathbf{x}_1$ $(\hat{\mathbf{w}} \cdot \mathbf{x}_{-1})$ is the distance of the boundary line of $\mathbf{x}_1$ $(\mathbf{x}_{-1})$ from the origin

SVM



- Denote $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$, $\hat{\mathbf{w}} \cdot \mathbf{x}_1$ $(\hat{\mathbf{w}} \cdot \mathbf{x}_{-1})$ is the distance of the boundary line of $\mathbf{x}_1$ $(\mathbf{x}_{-1})$ from the origin
- Thus, the distance between the two boundary lines is $\hat{\mathbf{w}} \cdot (\mathbf{x}_1 - \mathbf{x}_{-1}) = \frac{2}{\|\mathbf{w}\|}$

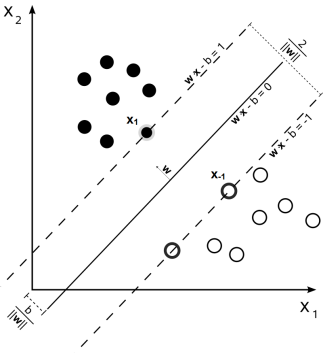
SVM



- Denote $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$, $\hat{\mathbf{w}} \cdot \mathbf{x}_1$ $(\hat{\mathbf{w}} \cdot \mathbf{x}_{-1})$ is the distance of the boundary line of $\mathbf{x}_1$ $(\mathbf{x}_{-1})$ from the origin
- Thus, the distance between the two boundary lines is $\hat{\mathbf{w}} \cdot (\mathbf{x}_1 - \mathbf{x}_{-1}) = \frac{2}{\|\mathbf{w}\|}$
- SVM: for all $\mathbf{x}^{(i)}$

$$\max \frac{2}{\|\mathbf{w}\|} \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1$$

SVM



- Denote $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$, $\hat{\mathbf{w}} \cdot \mathbf{x}_1$ $(\hat{\mathbf{w}} \cdot \mathbf{x}_{-1})$ is the distance of the boundary line of $\mathbf{x}_1$ $(\mathbf{x}_{-1})$ from the origin
- Thus, the distance between the two boundary lines is $\hat{\mathbf{w}} \cdot (\mathbf{x}_1 - \mathbf{x}_{-1}) = \frac{2}{\|\mathbf{w}\|}$
- SVM: for all $\mathbf{x}^{(i)}$

$$\max \frac{2}{\|\mathbf{w}\|} \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1$$

Equivalently,

$$\min \|\mathbf{w}\|^2 \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1$$

KKT conditions

We can absorb the constraint using Lagrange multiplier and rewrite the optimization problem (why?) as

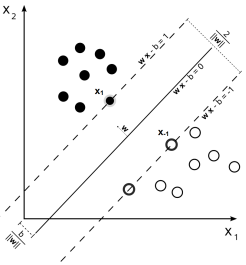
$$\min_{\mathbf{w},b} \max_{\alpha_i \geq 0} \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i (y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) - 1)$$

Consider a slightly modified problem

$$\max_{\alpha_i \geq 0} \underbrace{\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i (y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) - 1)}_{L}$$

Generally speaking, the solution of the dual problem will be smaller. However, when the two solutions are the same the complementary slackness conditions $\alpha_i^*(y^{(i)}(\mathbf{w}^* \cdot \mathbf{x}^{(i)} - b^*) - 1) = 0$ have to be satisfied. Together with $y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1, \alpha_i \geq 0, \nabla_{\mathbf{w}} L = 0$, these are known as the KKT conditions, which are necessary condition for optimality

Dual problem



Let's try to minimize L w.r.t. $\mathbf{w}$ and b

- $\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_i \alpha_i y^{(i)} \mathbf{x}^{(i)}$
- $\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_i \alpha_i y^{(i)} = 0$

Therefore the dual problem can now be rewritten as

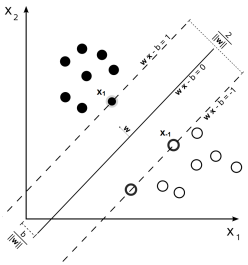
$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$

such that

$$\sum_i \alpha_i y^{(i)} = 0$$

Note that if we let all α fixed except two of them, the above is just a quadratic function that can be solved analytically

Support vectors



Say after solving the dual problem, we have

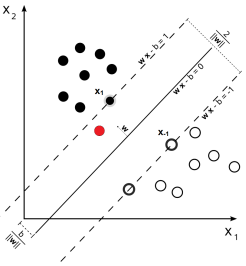
$$\mathbf{w} = \sum_i \alpha_i^* y^{(i)} \mathbf{x}^{(i)}$$

Evaluating a new input $\mathbf{x}$ is simply computing the sign of

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i^* y^{(i)} \mathbf{x}^{(i)} \cdot \mathbf{x} + b$$

Now, recall the complementary slackness condition $\alpha_i^*(y^{(i)}(\mathbf{w}^* \cdot \mathbf{x}^{(i)} - b^*) - 1) = 0$, actually most α_i^* will be equal to 0 except those with corresponding $\mathbf{x}^{(i)}$ "touching" the boundary, which are the support vectors

Soft-margin SVM and hinge loss



- Hard-margin SVM

$$\min \|\mathbf{w}\|^2 \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1$$

- Soft-margin SVM (allow constrain to be violate)

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

such that $y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} - b) \geq 1 - \xi_i, \xi_i \geq 0$

Soft-margin SVM

$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$ such that
$y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)} - b) \geq 1 - \xi_i, \xi_i \geq 0$
For the dual problem, write

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i[y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)} - b) - 1 + \xi_i] - \sum_i r_i\xi_i$$

We should minimize L w.r.t. $\mathbf{w}$, b, and ξ_i. This gives us $\mathbf{w} = \sum_i \alpha_i y^{(i)}\mathbf{x}^{(i)}, \sum_i \alpha_i y^{(i)} = 0$, and $C - \alpha_i - r_i = 0$. So the dual problem can be rewritten as

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_i y^{(i)}y^{(j)}\alpha_i\alpha_j\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}$$

such that $0 \leq \alpha_i \leq C$, and $\sum_i \alpha_i y^{(i)} = 0$

Complimentary slackness conditions

Note that we have the conditions $\alpha_i[y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)-1+\xi_i]=0$ and $r_i\xi_i=0$. Also $C-\alpha_i-r_i=0$ as we shown earlier, therefore

- If $0<r_i<C \Rightarrow 0<\alpha_i<C$, $y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)-1+\xi_i=0$ and since $\xi_i=0$,

$$y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)=1$$

- If $r_i=0$, $\alpha_i=C$, $y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)-1+\xi_i=0$ but $\xi_i\geq 0$, therefore

$$y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)\leq 1$$

- If $r_i=C$, $\alpha_i=0$, $y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)-1+\xi_i\geq 0$ and since $\xi_i=0$,

$$y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}-b)\geq 1$$

Sequential minimal optimization

- A major reason that SVM was so popular is that there are efficient methods in solving the optimization problem for training
- One popular method is SMO due to John Platt, the key idea is to select heuristically two α at a time and fix the rest
  - Pick one of the α that violates the KKT conditions. Pick the second α that maximizes the optimization step
  - The remaining problem will be a simple quadratic optimization problem with closed form solution

Kernel trick

Note that during both evaluating and testing. We just need to manipulate the inner products among training features $\mathbf{x}^{(i)}$ and with a new input $\mathbf{x}$

- Potentially, we can increase the model complexity by evaluating these inner product projected to a higher dimensional space (including higher order monomials) without actually projection
- E.g., $\mathbf{x} = [x_1, x_2], \phi(\mathbf{x}) = [x_1x_1, x_1x_2, x_2x_1, x_2x_2]^\top$, $\phi(\mathbf{x})^\top \phi(\mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2 \triangleq K(\mathbf{x}, \mathbf{z})$
- More generally, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + c)^d$ corresponds to inner product of $\phi(\mathbf{x})$ including all monomials up to order d
- Generally, inner product can also be interpreted as the similarity between two vectors. One may think a reasonable (so-called Gaussian) kernel will be

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right),$$

which actually corresponds to features projected to infinite dimensional space

Valid kernel (Mercer)

For any m vectors, $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}$, we can define a "kernel matrix" $\mathcal{K}$ with $\mathcal{K}_{i,j} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. It is easy to verify that $\mathcal{K}$ is symmetric (trivial) and positive semi-definite

- for any $\mathbf{z} = [z_1, \cdots, z_m]^\top$, $\mathbf{z}^\top \mathcal{K} \mathbf{z} = \sum_{i,j} z_i \phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)}) z_j = \sum_{i,j,k} z_i \phi_k(\mathbf{x}^{(i)}) \phi_k(\mathbf{x}^{(j)}) z_j = \sum_k \left(\sum_i z_i \phi_k(\mathbf{x}^{(i)}) \right)^2 \geq 0$

Kernel SVM

- Note that for both solving the dual problem and evaluating a new input only involve inner product of input and training vectors. So we can apply the kernel trick. The dual problem will be modified as

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_i y^{(i)} y^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- After solving for α, an input $\mathbf{x}$ can be evaluated with

$$\phi(\mathbf{w}) \cdot \phi(\mathbf{x}) + b = \sum_i \alpha_i^* y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + b$$

Multi-class SVM

- We can easily extend soft-margin SVM to multi-class case. Let $s_l(\mathbf{x}) = \mathbf{w_l}^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$ be the score for class l.

Multi-class SVM

- We can easily extend soft-margin SVM to multi-class case. Let $s_l(\mathbf{x}) = \mathbf{w_l}^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$ be the score for class l. We can define the hinge loss for sample $\mathbf{x}$ as

$$\sum_{l \neq j} h(s_l(\mathbf{x}) - s_j(\mathbf{x}) + \Delta) = \sum_{l \neq j} \max(0, s_l(\mathbf{x}) - s_j(\mathbf{x}) + \Delta),$$

where j is the true label of $\mathbf{x}$ and Δ contributes a margin ensuring that the true label score has to be at least Δ more than the rest to be penalty free

Multi-class SVM

- We can easily extend soft-margin SVM to multi-class case. Let $s_l(\mathbf{x}) = \mathbf{w_l}^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$ be the score for class l. We can define the hinge loss for sample $\mathbf{x}$ as

$$\sum_{l \neq j} h(s_l(\mathbf{x}) - s_j(\mathbf{x}) + \Delta) = \sum_{l \neq j} \max(0, s_l(\mathbf{x}) - s_j(\mathbf{x}) + \Delta),$$

  where j is the true label of $\mathbf{x}$ and Δ contributes a margin ensuring that the true label score has to be at least Δ more than the rest to be penalty free
- Multi-class SVM:

$$\min \|\mathbf{w}\|^2 + C \sum_i \sum_{l \neq j(\mathbf{x}_i)} h(s_l(\mathbf{x}_i) - s_{j(\mathbf{x}_i)}(\mathbf{x}_i) + \Delta)$$

Support vector regression

Reference: A Tutorial on Support Vector Regression

- Hard-margin

$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \begin{cases} y^{(i)} - \langle \mathbf{w}, \mathbf{x}^{(i)} \rangle - b \leq \epsilon \\ \langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b - y^{(i)} \leq \epsilon \end{cases}$$

- Soft-margin

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i (\xi_i + \xi_i^*)$$

$$\text{s.t.} \begin{cases} y^{(i)} - \langle \mathbf{w}, \mathbf{x}^{(i)} \rangle - b \leq \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b - y^{(i)} \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

Dual problem of SVR

$$
\begin{aligned}
L \triangleq \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i(\xi_i + \xi_i^*) - \sum_i(\eta_i\xi_i + \eta_i^*\xi_i^*) \\
- \sum_i \alpha_i(\epsilon + \xi_i - y^{(i)} + \langle \mathbf{w}, \mathbf{x}^{(i)}\rangle + b) \\
- \sum_i \alpha_i^*(\epsilon + \xi_i^* + y^{(i)} - \langle \mathbf{w}, \mathbf{x}^{(i)}\rangle - b)
\end{aligned}
$$

We can reformulate the problem to $\min_{\mathbf{w},\xi_i,\xi_i^*} \max_{\alpha_i,\alpha_i^*,\eta_i,\eta_i^*} L$ and this leads to

$$
\max \begin{cases} \frac{1}{2}\sum_{i,j}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)}\rangle \\ -\epsilon\sum_i(\alpha_i + \alpha_i^*) + \sum_i y^{(i)}(\alpha_i - \alpha_i^*) \end{cases}
$$

s.t. $\sum_i(\alpha_i - \alpha_i^*) = 0$ and $\alpha_i, \alpha_i^* \in [0, C]$

$$
\Rightarrow \mathbf{w} = \sum_i(\alpha_i - \alpha_i^*)\mathbf{x}^{(i)}
$$

Kernel PCA

- Principal component analysis (PCA) is a very common technique for dimension reduction. Consider data in high dimension, often data only vary along several dimensions and so we can keep dimensions for data with the highest variations and discard the rest
- The problem of PCA is that the analysis is linear. So for data like below, they are not separable

Kernel PCA

Consider the N d-dimensional data points as $x^{(1)}, \cdots, x^{(N)}$. Assuming the project vectors in high dimensional are zero-mean (will come back to that later), the covariance matrix C at the high dimension can then be approximate by

$$C = \frac{1}{N} \sum_{i=1}^{N} \phi(x^{(i)}) \phi(x^{(i)})^{\top}$$

If we want to apply PCA at this high dimension, we need to eigen-decompose C. That is, we want to find v such that $Cv = \lambda v$. Amazingly, we have the following theorem regarding v

Eigenvectors of projected space

Theorem (Eigenvectors)

Eigenvectors of C can be represented as weighted sum of $\phi(x^{(i)})$. That is, $v = \sum_{i=1}^{N} \alpha_i \phi(x^{(i)})$

Proof.

Assume that $Cv = \lambda v$, thus

$$Cv = \frac{1}{N} \sum_{i=1}^{N} \phi\left(x^{(i)}\right) \phi\left(x^{(i)}\right)^{\top} v = \lambda v$$

$$\Rightarrow v = \sum_{i=1}^{N} \underbrace{\frac{\phi\left(x^{(i)}\right)^{\top} v}{N\lambda}}_{\alpha_i} \phi\left(x^{(i)}\right)$$

Gram matrix

The previous theorem gives us some ideas what eigenvectors in the high dimensional space are like. Let's substitute $v = \sum_{i=1}^{N} \alpha_i \phi(x^{(i)})$ into $Cv = \lambda v$. We have,

$$\lambda \sum_{j=1}^{N} \alpha_j \phi\left(x^{(j)}\right) = \lambda v = Cv = \frac{1}{N} \sum_{i=1}^{N} \phi\left(x^{(i)}\right) \phi\left(x^{(i)}\right)^{\top} \left(\sum_{j=1}^{N} \alpha_j \phi\left(x^{(j)}\right) \right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \phi\left(x^{(i)}\right) \left(\sum_{j=1}^{N} \alpha_j \phi\left(x^{(i)}\right)^{\top} \phi\left(x^{(j)}\right) \right)$$

Now let's define the Gram matrix G with its i,j element given by

$$G_{i,j} = \langle \phi\left(x^{(i)}\right), \phi\left(x^{(j)}\right) \rangle = \phi\left(x^{(i)}\right)^{\top} \phi\left(x^{(j)}\right) \triangleq K(x^{(i)}, x^{(j)}),$$

where $K(\cdot, \cdot)$ is the kernel function. For example, we can have the Gaussian kernel with $K(x, y) = \exp(-\|x - y\|^2/c)$

Solving for v

$$\lambda \sum_{j=1}^{N} \alpha_j \phi\left(x^{(k)}\right)^\top \phi\left(x^{(j)}\right) = \frac{1}{N} \sum_{i=1}^{N} \phi\left(x^{(k)}\right)^\top \phi\left(x^{(i)}\right) \sum_{j=1}^{N} \alpha_j \phi\left(x^{(i)}\right)^\top \phi\left(x^{(j)}\right)$$

$$\Rightarrow \lambda \sum_{j=1}^{N} \alpha_j G(k,j) = \frac{1}{N} \sum_{i=1}^{N} G(k,i) \sum_{j=1}^{N} \alpha_j G(i,j) \Rightarrow \lambda (G\alpha)_k = \frac{1}{N} \sum_{j=1}^{N} \alpha_j (G^2)_{k,j}$$

$$\Rightarrow \lambda (G\alpha)_k = \frac{1}{N} (G^2\alpha)_k \Rightarrow \lambda G\alpha = \frac{1}{N} G^2\alpha \Rightarrow \lambda N\alpha = G\alpha$$

- Thus, α is actually an eigenvector of G with eigenvalue λN
- Similar to the original PCA, we can sort the eigenvalues. And given α, the eigenvector in ϕ-space is $v = \sum_{i=1}^{N} \alpha_i \phi(x^{(i)})$
- When receive a new input x, we can project to v as $\langle \phi(x), v \rangle = \sum_{i=1}^{N} \alpha_i \langle \phi(x), \phi(x^{(i)}) \rangle = \sum_{i=1}^{N} \alpha_i K(x, x^{(i)})$

Centering $\phi(x^{(i)})$

We mentioned earlier that we have assumed $\phi(x^{(i)})$ are zero-mean. In general, this is not true but can be easily fixed below. If $\phi(x^{(i)})$ are not zero-mean, $\phi(x^{(i)})$ should be replaced by $\phi(x^{(i)}) - \frac{1}{N}\sum_{k=1}^{N}\phi(x^{(k)})$ instead. Thus we should have the correct Gram matrix

$$
\begin{aligned}
&\tilde{G} \\
&= \left[I_N - \frac{1}{N}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \right]^\top \begin{pmatrix} \phi(x^{(1)})^\top \\ \vdots \\ \phi(x^{(N)})^\top \end{pmatrix} [\phi(x^{(1)}), \cdots, \phi(x^{(N)})] \left[I_N - \frac{1}{N}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \right] \\
&= \left[I_N - \frac{1}{N}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \right]^\top G \left[I_N - \frac{1}{N}\begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \right] \\
&= G - \frac{1}{N}1_N G - \frac{1}{N} G 1_N + \frac{1}{N^2} 1_N G 1_N,
\end{aligned}
$$

where 1_N is $N \times N$ matrix with all ones

Summary of Kernel PCA

- Decide a kernel and compute the normalized Gram matrix
- Eigen-decompose the normalized Gram matrix
- Sort the eigenvalues in the descending order. An eigenvector composes of the weights α for constructing the corresponding principal component in the ϕ-space
- Given an input x, the projection to a principal component with weight α is given by $\sum_{i=1}^{N} \alpha_i K(x, x^{(i)})$

Denoising with Kernel PCA

Now, consider K principal components $v_1, \cdots, v_K$ in (ϕ-space) with

$$v_k = \sum_{i=1}^{N} \alpha_i^{(k)} \phi(x^{(i)}) \text{ for } k = 1, \cdots, K$$

We have a denoised version of x (let's call z here) if we only keep projection of x onto the K principal components in the ϕ-space. That is,

$$\phi(z) = \sum_{k=1}^{K} \langle \phi(x), v_k \rangle v_k.$$

The problem is that it is not immediately clear how to find z to satisfy the above. So instead, let's try to minimize

$$L = \|\phi(z) - \sum_{k=1}^{K} \langle \phi(x), v_k \rangle v_k\|$$

Minimizing L

$$
\begin{aligned}
L &= \|\phi(z) - \sum_{k=1}^{K} \langle \phi(x), v_k \rangle v_k \| \\
&= \langle \phi(z), \phi(z) \rangle - 2 \sum_{k=1}^{K} \langle \phi(x), v_k \rangle \langle v_k, \phi(z) \rangle + \Omega \\
&= K(z,z) - 2 \sum_{i=1}^{N} \sum_{k=1}^{K} \langle \phi(x), v_k \rangle \alpha_i^{(k)} \langle \phi(x^{(i)}), \phi(z) \rangle + \Omega \\
&= K(z,z) - 2 \sum_{i=1}^{N} \underbrace{\sum_{k=1}^{K} \langle \phi(x), v_k \rangle \alpha_i^{(k)}}_{\gamma_i} K(x^{(i)}, z) + \Omega \\
&= K(z,z) - 2 \sum_{i=1}^{N} \gamma_i K(x^{(i)}, z) + \Omega
\end{aligned}
$$

Note that Ω does not depend on z and hence can be ignored.

Maximizing Λ

Now, if we focus on kernel with the form $K(x, y) = K(\|x - y\|)$, the first term $K(z, z)$ is a constant and can be ignored as well. So minimizing L is the same as maximizing

$$\Lambda = \sum_{i=1}^{N} \gamma_i K(x^{(i)}, z)$$

Let's maximize Λ by setting $\nabla_z \Lambda$ to 0,

$$\nabla_z \Lambda = 2 \sum_{i=1}^{N} \gamma_i K'(\|x^{(i)} - z\|^2)(x^{(i)} - z) = 0$$

$$\Rightarrow z = \frac{\sum_{i=1}^{N} \gamma_i K'(\|x^{(i)} - z\|^2) x^{(i)}}{\sum_{i=1}^{N} \gamma_i K'(\|x^{(i)} - z\|^2)} = \frac{\sum_{i=1}^{N} \gamma_i e^{-\frac{\|x^{(i)} - z\|^2}{c}} x^{(i)}}{\sum_{i=1}^{N} \gamma_i e^{-\frac{\|x^{(i)} - z\|^2}{c}}}$$

Thus, we can iteratively update

$$z^{(m)} = \frac{\sum_{i=1}^{N} \gamma_i e^{-\frac{\|x^{(i)} - z^{(m-1)}\|^2}{c}} x^{(i)}}{\sum_{i=1}^{N} \gamma_i e^{-\frac{\|x^{(i)} - z^{(m-1)}\|^2}{c}}}$$