Linear Models for Regression and Classification

Adila A. Krisnadhi*, Siti Aminah, Aruni Y. Azizah,
Dina Chahyati, Fariz Darari

Faculty of Computer Science
Universitas Indonesia

CSGE603130 - Kecerdasan Artifisial dan Sains Data Dasar
2022-11-16

Outline

Outline

Regression problem

x di bold itu feature vector

- **Supervised learning**: Let $\mathcal{D}$ be a finite set of N training examples $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{N}, y^{N})\}$ such that:
  - each $\boldsymbol{x}^{(i)}$ is a **feature** vector of the ith example from the **input space** $\mathcal{X}$,
  - each $y^{(i)}$ is the ith **target** value from the output space $\mathcal{Y}$, and
  - $y^{(i)} = f(\boldsymbol{x}^{(i)})$ for each $i = 1, \ldots, N$ for some unknown (total) function $f \colon \mathcal{X} \to \mathcal{Y}$.

  Then, supervised learning aims to find a function $h \colon \mathcal{X} \to \mathcal{Y}$ that is a good approximation of f and able to predict the output $\hat{y} = f(\hat{\boldsymbol{x}})$ accurately from an unseen input $\hat{\boldsymbol{x}}$.
- The function h above is called a **hypothesis** or a **model**.
- If $\mathcal{Y}$ is discrete/categorical, then the task is called **classification**.
- If $\mathcal{Y}$ is continuous, then the task is called **regression**.
- We've learned in previous lectures:
  - Decision trees (CART) and random forest $\rightsquigarrow$ classification and regression (discriminative)
  - k-nearest neighbors $\rightsquigarrow$ classification and regression (discriminative)
  - Naive Bayes $\rightsquigarrow$ classification (generative)
- Here, we learn linear discriminative models for regression (i.e., linear regression) and classification (logistic regression)

Motivation: House pricing

Area (m^2)	Price (1000 USD)
158.9	208.5
117.2	181.5
165.9	223.5
159.5	140
204.2	250
126.5	143
157.4	307
194.2	200
164.8	129.9
100.1	118
96.6	129.5
215.9	345
84.7	144
138.8	279.5
116.4	157
79.3	132
93.3	149
120.4	90
103.5	159
124.4	139

- Task: based on the data, predict the house price given its size/area.
- Equivalently, we want to find a function f such that $f(\text{Area}) = \text{Price}$.
- Such a function can be seen as a

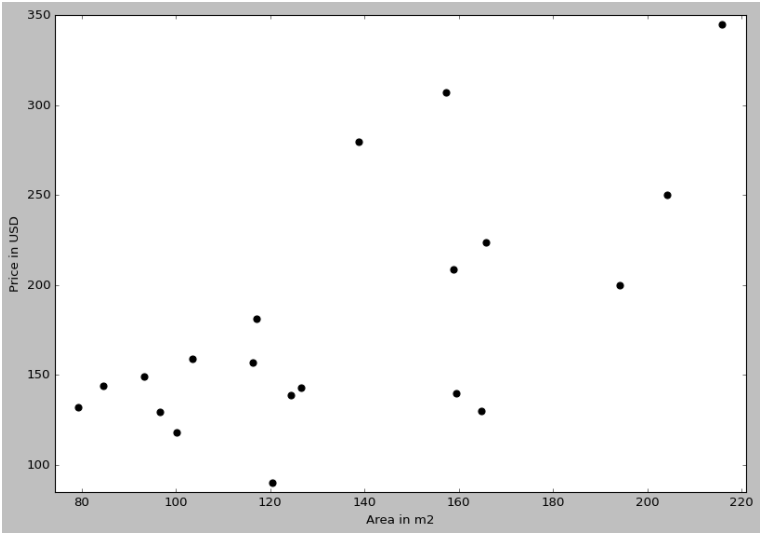
Motivation: House pricing

Area (m^2)	Price (1000 USD)
158.9	208.5
117.2	181.5
165.9	223.5
159.5	140
204.2	250
126.5	143
157.4	307
194.2	200
164.8	129.9
100.1	118
96.6	129.5
215.9	345
84.7	144
138.8	279.5
116.4	157
79.3	132
93.3	149
120.4	90
103.5	159
124.4	139

- Task: based on the data, predict the house price given its size/area.
- Equivalently, we want to find a function f such that $f(\text{Area}) = \text{Price}$.
- Such a function can be seen as a curve on a xy-plane.
- The task is thus also called **curve fitting**.
- **Linear regression** $\rightsquigarrow$ the curve is a straight line.

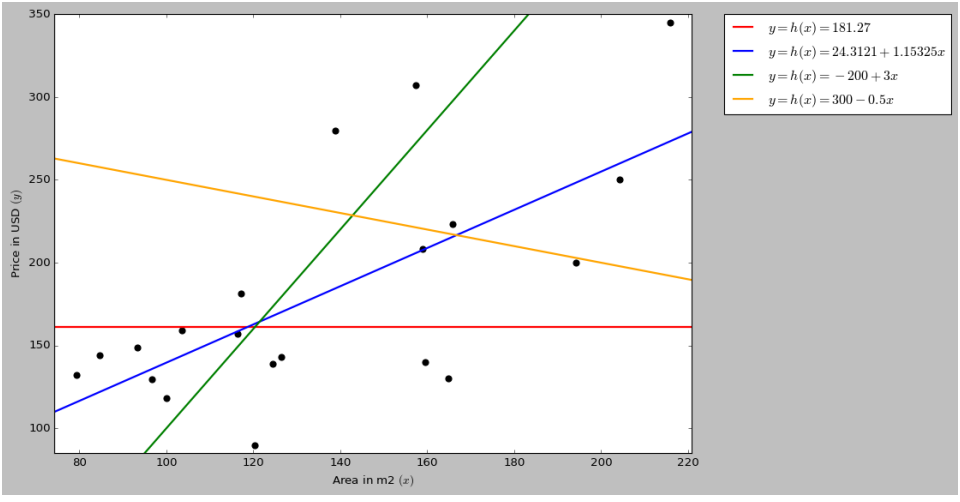
mencari kurva yang paling fit berdasarkan data yang ada saat ini

Let's look at the data ...



Hypothesis

Which linear curve gives the most reasonable hypothesis? Why?



Linear regression (1)

- Each hypothesis can be written as $y = h(x) = w_0 + w_1 x$ where $w_0, w_1 \in \mathbb{R}$.
- Generalizing for n features $(x_1, \ldots, x_n)$, such a hypothesis can be written as

$$h(x_1, \ldots, x_n) = w_0 + w_1 x_1 + \cdots + w_n x_n$$

  where the x_i's are the input features, w_i's are the **parameters** or **weights** of the hypothesis, and w_0 is called the **intercept** or **bias** term. (Note: the term "bias" here is different from that of "bias-variance tradeoff").
- Here, h is an n-**dimensional linear hyperplane** within $n+1$-dimensional space. If $n = 1$, we have a straight line.
  - Thus, h is a linear model and since the target value is a real number, we call it a **linear regression model**.
- The intercept corresponds to the point $(\mathbf{0}, w_0)$, the location at which the linear hyperplane intersects the axis that corresponds to the target variable.

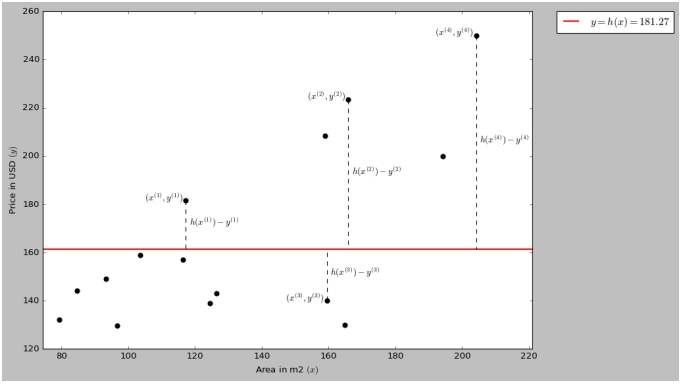
Linear regression (2)

- Using vector notation, let $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)^\mathsf{T}$ with $x_0 = 1$, and $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^\mathsf{T}$. Then, the hypotheses can be written as a matrix operation:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{i=0}^{n} w_i x_i = \begin{pmatrix} w_0 & w_1 & \ldots & w_n \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} = \boldsymbol{w}^\mathsf{T} \boldsymbol{x}$$

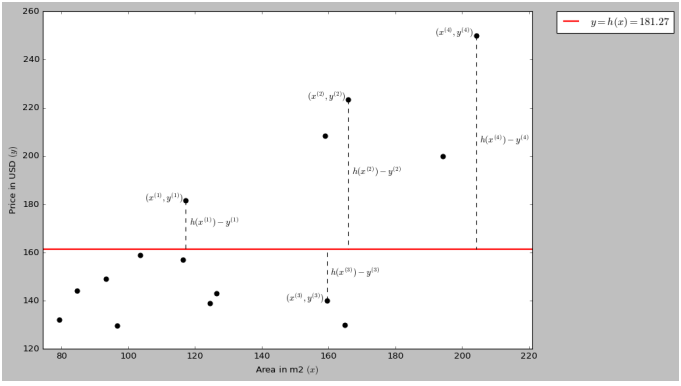
- We write $h_{\boldsymbol{w}}$ instead of just h to indicate that h is parameterized by $\boldsymbol{w}$ (different h would have different $\boldsymbol{w}$).
- **Linear regression** aims to find the appropriate parameter values $\boldsymbol{w}$ that best fit the data.
  - How do we choose the best parameter values, i.e., the best hypothesis?
  - $\rightsquigarrow$ Need a metric that measures the "fitness" of each hypothesis.

Errors/residuals



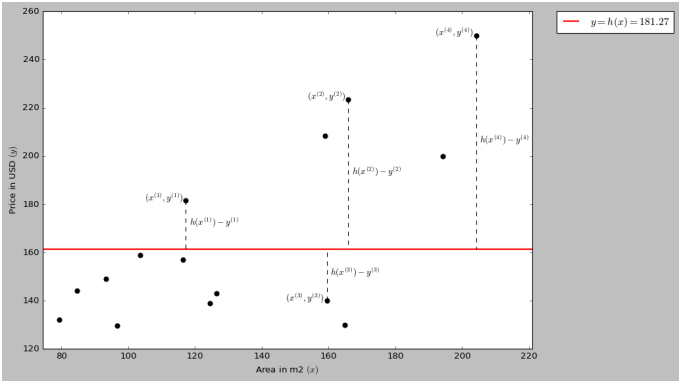
- **Error** or **residual** of each instance data $(x^{(i)}, y^{(i)})$ is the difference $h(x^{(i)}) - y^{(i)} = (w_0 + w_1 x_1^{(i)}) - y^{(i)}$.
  - The lower the residual the better fit h to the instance data.
- Fitness of a hypothesis h accounts for errors over ALL data, i.e., the sum of errors of h over all data.
  - Which sum of errors?

Sum of errors/residuals



- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?

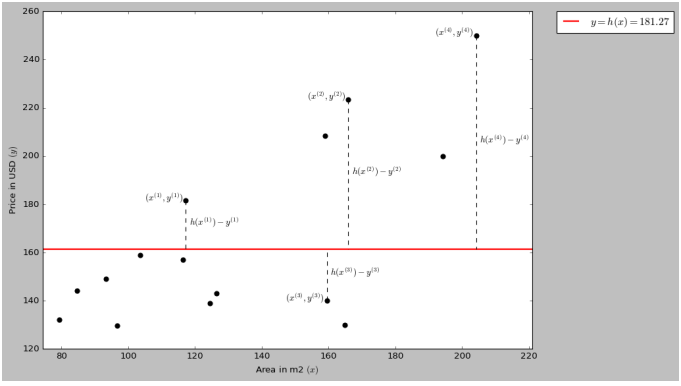
Sum of errors/residuals



- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?
- **Is this the one?**

$$J(w_0, w_1) = (w_0 + w_1 x_1^{(1)} - y^{(1)}) + \cdots + (w_0 + w_1 x_1^{(N)} - y^{(N)}) = \sum_{i=1}^{N} (h(x^{(i)}) - y^{(i)})$$

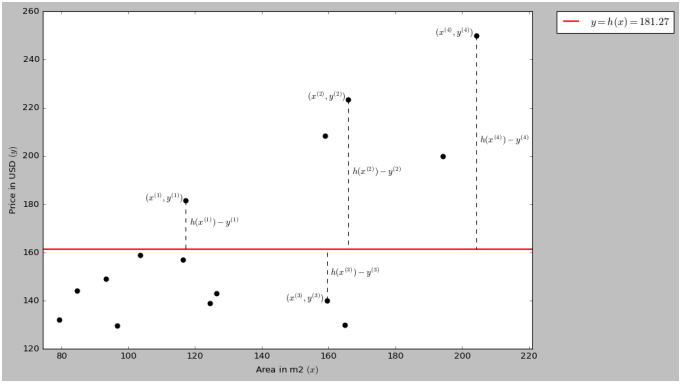
Sum of errors/residuals



- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?
- **Is this the one?** No. Because positive and negative residuals cancel out.

$$J(w_0, w_1) = (w_0 + w_1 x_1^{(1)} - y^{(1)}) + \cdots + (w_0 + w_1 x_1^{(N)} - y^{(N)}) = \sum_{i=1}^{N} (h(x^{(i)}) - y^{(i)})$$

Sum of errors/residuals

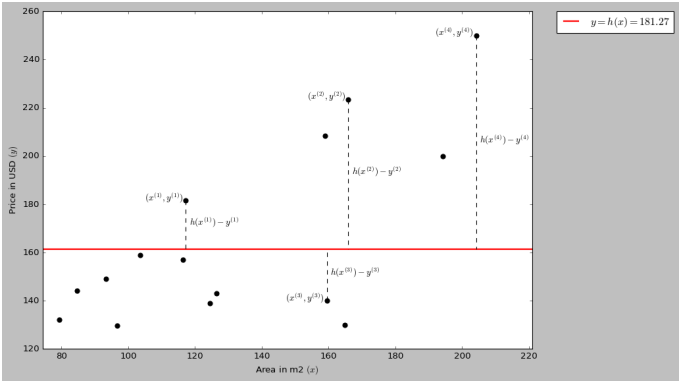


- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?
- **Or this one?**

mau menurunkan nilai min (kalau di kalkulus turunan)

$$J(w_0, w_1) = |w_0 + wx_1^{(1)} - y^{(1)}| + \cdots + |w_0 + wx_1^{(N)} - y^{(N)}| = \sum_{i=1}^{N} |h(x^{(i)}) - y^{(i)}|$$

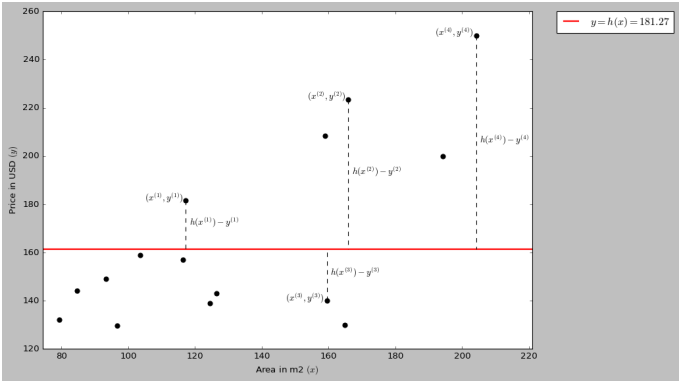
Sum of errors/residuals



- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?
- **Or this one?** Could be. But, finding minima with absolute function is complicated.

$$J(w_0, w_1) = |w_0 + wx_1^{(1)} - y^{(1)}| + \cdots + |w_0 + wx_1^{(N)} - y^{(N)}| = \sum_{i=1}^{N} |h(x^{(i)}) - y^{(i)}|$$

Sum of errors/residuals



- For a given dataset, the sum of errors of hypothesis h depends solely on the parameters (w_0, w_1) of h.
- So, let $J(w_0, w_1)$ be the sum of errors for a hypothesis h whose parameter vector is $(w_0, w_1)^\mathsf{T}$. Our aim is to **minimize** J, but how do we express J?
- **This one is better**: never negative, no cancelling out among residuals, finding minima is simpler.

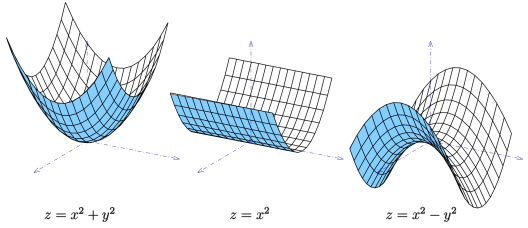
$$J(w_0, w_1) = (w_0 + w_1 x_1^{(1)} - y^{(1)})^2 + \cdots + (w_0 + w_1 x_1^{(N)} - y^{(N)})^2 = \sum_{i=1}^{N} (h(x^{(i)}) - y^{(i)})^2$$

Sum of squares error

- Training a linear regression model $h_{\boldsymbol{w}}(\boldsymbol{x})$ over dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$ aims to find parameter values $\boldsymbol{w}$ that minimizes the **sum of squares** error

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} (h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

- $J(\boldsymbol{w})$ is **paraboloid convex** $\rightsquigarrow$ always have a global minima.



Left, middle: paraboloid convex. Right: paraboloid, non-convex

- Why multiply by $\frac{1}{2}$?

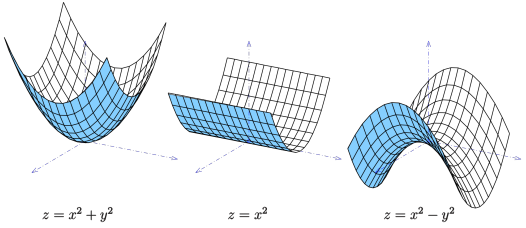
1/2 kalo di turunin ilang nanti

Sum of squares error

- Training a linear regression model $h_{\boldsymbol{w}}(\boldsymbol{x})$ over dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$ aims to find parameter values $\boldsymbol{w}$ that minimizes the **sum of squares** error

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} (h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

- $J(\boldsymbol{w})$ is **paraboloid convex** $\rightsquigarrow$ always have a global minima.



Left, middle: paraboloid convex. Right: paraboloid, non-convex

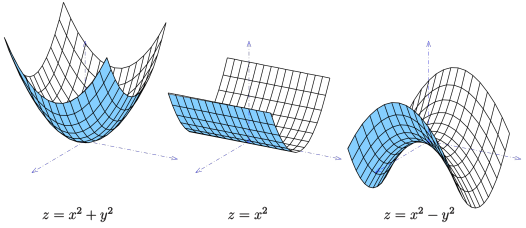
- Why multiply by $\frac{1}{2}$? Just for mathematical convenience when computing derivatives :-).

Sum of squares error

- Training a linear regression model $h_{\boldsymbol{w}}(\boldsymbol{x})$ over dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$ aims to find parameter values $\boldsymbol{w}$ that minimizes the **sum of squares** error

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} (h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

- $J(\boldsymbol{w})$ is **paraboloid convex** $\rightsquigarrow$ always have a global minima.



Left, middle: paraboloid convex. Right: paraboloid, non-convex

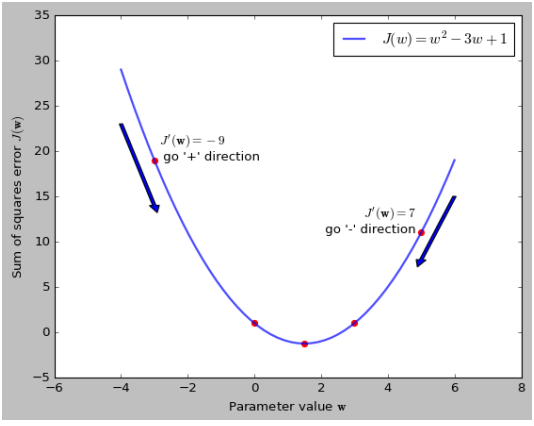
- Why multiply by $\frac{1}{2}$? Just for mathematical convenience when computing derivatives :-).
- $J(\boldsymbol{w})$ as above leads to the **ordinary least squares** regression model.

Gradient descent: Idea

- How do we learn $\boldsymbol{w}$? Suppose our current parameter value is $\boldsymbol{w}$, with error $J(\boldsymbol{w})$.
- To where do we move from $\boldsymbol{w}$ so that we get a maximum decrease in $J(\boldsymbol{w})$?

Gradient descent: Idea

- How do we learn $\boldsymbol{w}$? Suppose our current parameter value is $\boldsymbol{w}$, with error $J(\boldsymbol{w})$.
- To where do we move from $\boldsymbol{w}$ so that we get a maximum decrease in $J(\boldsymbol{w})$?
- Simplified example: J is quadratic, e.g., $J(w) = w^2 - 3w + 1$.

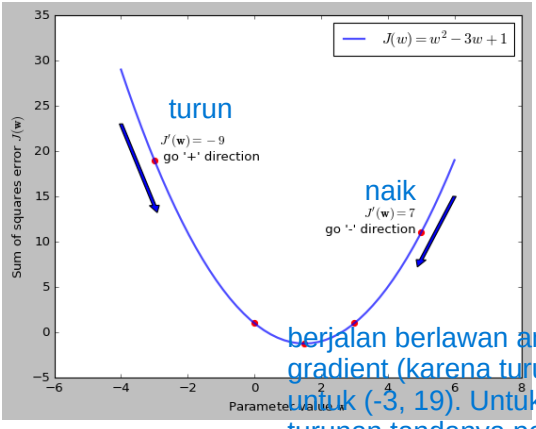


To reach the minima:

- At $(-3, 19)$, the derivative/gradient of the paraboloid is $J'(-3) = -9 < 0$.
  - We move in the positive direction.
- At $(5, 11)$, the gradient is $J'(5) = 7 > 0$
  - We move in the negative direction.
- So, at $\boldsymbol{w}$, we move in direction that is

Gradient descent: Idea

- How do we learn $\boldsymbol{w}$? Suppose our current parameter value is $\boldsymbol{w}$, with error $J(\boldsymbol{w})$.
- To where do we move from $\boldsymbol{w}$ so that we get a maximum decrease in $J(\boldsymbol{w})$?
- Simplified example: J is quadratic, e.g., $J(w) = w^2 - 3w + 1$.



To reach the minima:

- At $(-3, 19)$, the derivative/gradient of the paraboloid is $J'(-3) = -9 < 0$.
  - We move in the positive direction.
- At $(5, 11)$, the gradient is $J'(5) = 7 > 0$
  - We move in the negative direction.
- So, at $\boldsymbol{w}$, we move in direction that is opposite of the gradient sign.
- This approach is called **gradient descent**.

berjalan berlawan arah dari tanda gradient (karena turun -, kita pergi ke + untuk (-3, 19). Untuk (5,11), karena turunan tandanya positif, kita pergi ke

Gradient descent

Gradient descent updates the parameter vector $\boldsymbol{w}$ according to the following rule:

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \nabla J(\boldsymbol{w})$$

or, in the case of single feature data (two parameters):

$$w_0 \coloneqq w_0 - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1) \qquad w_1 \coloneqq w_1 - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1)$$

- $\nabla J(\boldsymbol{w})$ is the gradient of J with respect to $\boldsymbol{w} = (w_0, \ldots, w_n)$, i.e., the vector:

$$\nabla J(\boldsymbol{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} J(\boldsymbol{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} J(\boldsymbol{w}) \end{bmatrix}$$

turunan parsial

- α is the **step size** (i.e., **learning rate**); a hyperparameter whose value is usually small.

Batch gradient descent for linear regression

- For ordinary least squares regression, we can analytically compute the gradient of $J(\boldsymbol{w})$.
- **Exercise**: prove that each parameter w_j in an OLS linear regression model can be updated by gradient descent according to the following rule:

$$w_j \coloneqq w_j - \alpha \sum_{i=1}^{N} \big(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\big) x_j^{(i)} = w_j + \alpha \sum_{i=1}^{N} \big(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\big) x_j^{(i)}$$

The previous rule yields following **batch gradient descent** algorithm:

1. Start with an initial value of $\boldsymbol{w} = (w_0, \ldots, w_n)^{\mathsf{T}}$.
2. Repeat until convergence:
   - For every $j = 0, \ldots, n$: $w_j \coloneqq w_j + \alpha \sum_{i=1}^{N} \big(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\big) x_j^{(i)}$

Gradient descent contour

- Contours of the sum-of-squares J are in the form of ellipses.
- The trajectory of gradient descent goes towards the smallest ellipse in the contours.

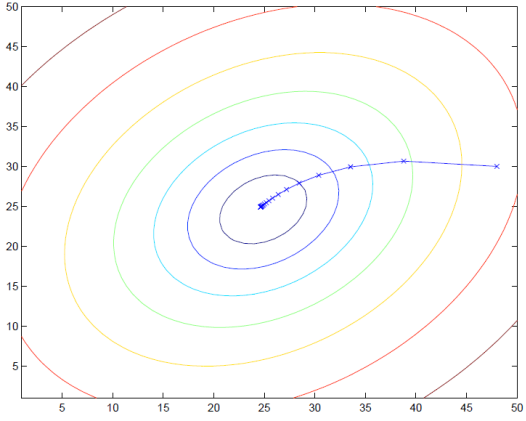


Image source: Andrew Ng, CS229 Lecture Notes for CS229 Fall 2018 course, Part I: Linear Regression, page 6.

Termination condition

Choices for termination condition (convergence):

- Fixed number of iterations: for a fixed step size and under certain conditions, if $\boldsymbol{w}^{(k)}$ is $\boldsymbol{w}$ at the k-th iteration and $\boldsymbol{w}^*$ is the optimal parameters, then $J(\boldsymbol{w}^{(k)}) - J(\boldsymbol{w}^*) \leqslant \epsilon$ after $O(1/\epsilon)$ iterations.
- $\|\boldsymbol{w}' - \boldsymbol{w}\|$ (change in parameter values) or $\|J(\boldsymbol{w}') - J(\boldsymbol{w})\|$ (change in error value) is small enough where $\boldsymbol{w}'$ is $\boldsymbol{w}$ after a single update.
- $\|\nabla J(\boldsymbol{w})\|$ is small enough: the closer it is to the minima, the smaller the gradient is.

error tolerance

Stochastic/incremental gradient descent

- Batch gradient descent must scan through the entire training set before making a single update.
  - May be expensive if the size of training set is very large.
- Alternative: **stochastic gradient descent** that performs update to the parameters according to the gradient of the error/loss at each single example.

1. Start with an initial value of $\boldsymbol{w} = (w_0, \ldots, w_n)^\mathsf{T}$.
2. Loop until convergence
   - Loop for every training example $\boldsymbol{x}^{(i)}$:
     - For every j: $w_j \coloneqq w_j + \alpha\big(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\big)x_j^{(i)}$

- SGD usually gets close to the minima much faster than batch gradient descent, but sometimes, the parameters may keep oscilating around the minima.
- Side note: in a more complicated setting, e.g., in deep learning, one does not compute the derivatives by hand. Instead, an automated differentiation algorithm is used (see Dive into Deep Learning book for more details).

Closed form solution to linear regression

- Besides using an iterative algorithm, we can find a closed-form solution for the optimal weight $\boldsymbol{w}$.
  - Most regression problem (esp. nonlinear ones) don't have a closed-form solution.
- Let X be the **design matrix** containing all training examples as its rows and $\boldsymbol{y}$ the vector containing all target values from the training set.

$$X = \begin{pmatrix} — \left(\boldsymbol{x}^{(1)}\right)^{\mathsf{T}} — \\ — \left(\boldsymbol{x}^{(2)}\right)^{\mathsf{T}} — \\ \vdots \\ — \left(\boldsymbol{x}^{(m)}\right)^{\mathsf{T}} — \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

- If $J(\boldsymbol{w})$ is the sum of squares cost function, then it can be shown that $\nabla J = X^{\mathsf{T}} X \boldsymbol{w} - X^{\mathsf{T}} \boldsymbol{y}$, which is minimized when $X^{\mathsf{T}} X \boldsymbol{w} = X^{\mathsf{T}} \boldsymbol{y}$
- Thus, the optimal $\boldsymbol{w}$ is $\boldsymbol{w} = (X^{\mathsf{T}} X)^{-1} X^{\mathsf{T}} \boldsymbol{y}$.

Why not just use the closed-form solution?

- If possible, it's better to use the closed-form solution.
- But, the closed-form solution may be impractical:
  - Suppose X is very large, sparse matrix (e.g., 10^6 examples with 10^5 features), but $X^\mathsf{T}X$ is fairly dense.
  - Note that the dimension of $X^\mathsf{T}X$ is $10^5 \times 10^5$. Hence, we have to store 10^{10} floating point numbers (at 8 bytes per number, takes roughly 80 gigabytes space) $\rightsquigarrow$ may be impractical.
- Matrix operations in the closed-form solution may have numerical accuracy issues (refer to the Numerical Analysis course).
  - Optimal $\boldsymbol{w}$ can be computed via singular value decomposition, which is more stable than directly computing the matrix inverse. This is an approach taken by `scikit-learn`.

On optimization

- Batch and stochastic gradient descent are **gradient-based** optimization approaches that employ **first derivatives**.
- In practice, plain/vanilla batch gradient descent is almost never used since it's extremely slow. Instead, one uses various forms of stochastic gradient descent.
  - Optimization used in neural networks/deep learning is usually some form of stochastic gradient descent.
- Alternatively, one can also use gradient-based optimization that relies on computing **second derivatives** such as Newton-Rhapson method, which can actually reach convergence faster. However, guaranteeing such a convergence is typically more difficult that stochastic gradient descent.

Example with scikit-learn (1)

```
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error

train_x = np.array([158.9, 117.2, 165.9, 159.5, 204.2, 126.5, 157.4, 194.2,
                    164.8, 100.1,  96.6, 215.9,  84.7, 138.8, 116.4,  79.3,
                     93.3, 120.4, 103.5, 124.4])
train_y = np.array([208.5, 181.5, 223.5, 140.0, 250.0, 143.0, 307.0, 200.0,
                    129.9, 118.0, 129.5, 345.0, 144.0, 279.5, 157.0, 132.0,
                    149.0,  90.0, 159.0, 139.0])
train_x = np.reshape(train_x, (20,1))
train_y = np.reshape(train_y, (20,1))
regr = linear_model.LinearRegression()
regr.fit(train_x, train_y)
print('The regression line equation is: y = %f + %f * x' %
      (regr.intercept_[0], regr.coef_[0]))

test_x = np.array([105,120]).reshape((2,1))
test_y = np.array([150,180]).reshape((2,1))
pred_y = regr.predict(test_x)
print('Test data:', test_x.reshape((2,)))
print('Prediction:', pred_y.reshape((2,)))
print('Ground truth:', test_y.reshape((2,)))
print('Test mean squared error: %.2f' % mean_squared_error(test_y, pred_y))
```

Training set

x	y
158.9	208.5
117.2	181.5
165.9	223.5
159.5	140
204.2	250
126.5	143
157.4	307
194.2	200
164.8	129.9
100.1	118
96.6	129.5
215.9	345
84.7	144
138.8	279.5
116.4	157
79.3	132
93.3	149
120.4	90
103.5	159
124.4	139

Testing set

x	y
105	150
120	180

- `LinearRegression` model can be obtained from the `linear_model` module of `scikit-learn`.
- Use the `fit(X,Y)` method of the model to fit a curve to features X and target values Y.
- The non-intercept parameters can be accessed via `coef_` array.
- The intercept can be accesed via `intercept_`
- To get a prediction over test data X, use `predict(X)` method of the model.
- Mean squared error of the test can be computed via `mean_squared_error` function.

Example with scikit-learn (2)

Visualization of the model.

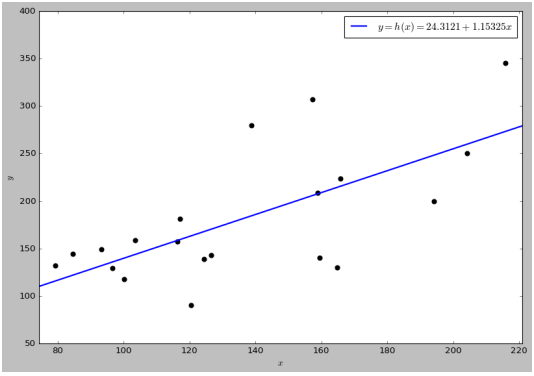
```
import matplotlib.pyplot as plt

plt.figure()
plt.figure(figsize=(12, 8))
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.style.use('classic')

plt.xlim(train_x.min()-5, train_x.max()+5)

plt.scatter(train_x,train_y,s=40, color='black')

x = np.linspace(train_x.min()-5, train_x.max()+5)
w0 = regr.intercept_[0]
w1 = regr.coef_[0]
y = w1*x + w0
label = r'$y = h(x) = %g + %gx $' % (w0,w1)
plt.plot(x,y, color='blue', linewidth=2, label=label)
plt.legend()
plt.show()
```



Gradient descent animation

See: https://towardsdatascience.com/gradient-descent-animation-1-simple-linear-regression-e49315b24672

Probabilistic perspective on linear regression

- Probabilistically, linear regression models the scalar output y given $(D+1)$-dimensional input $\boldsymbol{x} = (1, x_1, \ldots, x_D)^\mathsf{T}$ as a **random variable**: $y = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + \varepsilon$ where
  - $\boldsymbol{w} = (w_0, w_1, \ldots, w_D)^\mathsf{T}$ are the **weights** with w_0 called the **intercept**;
  - ε is a random variable satisfying $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, i.e., ε is normally distributed with mean 0 and variance σ^2;
  - y is "approximately" linear of $\boldsymbol{x}$ with noise whose mean is 0 and whose variance is σ^2.

Probabilistic perspective on linear regression

- Probabilistically, linear regression models the scalar output y given $(D+1)$-dimensional input $\boldsymbol{x} = (1, x_1, \ldots, x_D)^\mathsf{T}$ as a **random variable**: $y = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + \varepsilon$ where
  - $\boldsymbol{w} = (w_0, w_1, \ldots, w_D)^\mathsf{T}$ are the **weights** with w_0 called the **intercept**;
  - ε is a random variable satisfying $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, i.e., ε is normally distributed with mean 0 and variance σ^2;
  - y is "approximately" linear of $\boldsymbol{x}$ with noise whose mean is 0 and whose variance is σ^2.
- Characteristic of normal distribution: if $X \sim \mathcal{N}(\mu, \sigma^2)$, then for any real numbers a, b, $aX + b \sim \mathcal{N}(a\mu + b, |a|\sigma^2)$. Thus, if $\boldsymbol{x}$ is given/known, then

$$y \sim \mathcal{N}(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}, \sigma^2) \quad \text{i.e.,} \quad P(y \mid \boldsymbol{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \boldsymbol{w}^\mathsf{T}\boldsymbol{x})^2}{2\sigma^2}\right)$$

  - $\mathbb{E}[y \mid \boldsymbol{x}] = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ or "the expected value of the output is a linear function of the input"
  - $(\boldsymbol{w}, \sigma^2)$ are parameters of the model

Probabilistic perspective on linear regression

- Probabilistically, linear regression models the scalar output y given $(D+1)$-dimensional input $\boldsymbol{x} = (1, x_1, \ldots, x_D)^\mathsf{T}$ as a **random variable**: $y = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + \varepsilon$ where
  - $\boldsymbol{w} = (w_0, w_1, \ldots, w_D)^\mathsf{T}$ are the **weights** with w_0 called the **intercept**;
  - ε is a random variable satisfying $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, i.e., ε is normally distributed with mean 0 and variance σ^2;
  - y is "approximately" linear of $\boldsymbol{x}$ with noise whose mean is 0 and whose variance is σ^2.
- Characteristic of normal distribution: if $X \sim \mathcal{N}(\mu, \sigma^2)$, then for any real numbers a, b, $aX + b \sim \mathcal{N}(a\mu + b, |a|\sigma^2)$. Thus, if $\boldsymbol{x}$ is given/known, then

$$y \sim \mathcal{N}(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}, \sigma^2) \quad \text{i.e.,} \quad P(y \mid \boldsymbol{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \boldsymbol{w}^\mathsf{T}\boldsymbol{x})^2}{2\sigma^2}\right)$$

  - $\mathbb{E}[y \mid \boldsymbol{x}] = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ or "the expected value of the output is a linear function of the input"
  - $(\boldsymbol{w}, \sigma^2)$ are parameters of the model
- Aim: find $\boldsymbol{w}$ that maximizes the likelihood of the parameters given data (input, output).

Likelihood of parameters for simple linear regression

- Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) \mid i = 1, \ldots, N\}$ be the training data.
- We assume:
  1. all inputs $\boldsymbol{x}^{(i)}$ are independent;
  2. given input $\boldsymbol{x}^{(i)}$, the output $y^{(i)}$ is independent of the other input $\boldsymbol{x}^{(j)}$ with $j \neq i$;
  3. the outputs $y^{(i)}$'s are conditionally independent (given the inputs $\boldsymbol{x}^{(i)}$'s) and identically distributed.
- The likelihood of the parameter given the training data is:

Likelihood of parameters for simple linear regression

- Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) \mid i = 1, \ldots, N\}$ be the training data.
- We assume:
  1. all inputs $\boldsymbol{x}^{(i)}$ are independent;
  2. given input $\boldsymbol{x}^{(i)}$, the output $y^{(i)}$ is independent of the other input $\boldsymbol{x}^{(j)}$ with $j \neq i$;
  3. the outputs $y^{(i)}$'s are conditionally independent (given the inputs $\boldsymbol{x}^{(i)}$'s) and identically distributed.
- The likelihood of the parameter given the training data is:

$$
\begin{aligned}
L(\boldsymbol{w}, \sigma^2) &= P(y^{(1)}, \ldots, y^{(N)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}, \boldsymbol{w}, \sigma^2) \\
&= \prod_{i=1}^{N} P(y^{(i)} \mid \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}, \boldsymbol{w}, \sigma^2) && \text{(assumption 3)} \\
&= \prod_{i=1}^{N} P(y^{(i)} \mid \boldsymbol{x}^{(i)}, \boldsymbol{w}, \sigma^2) && \text{(assumption 1 and 2)} \\
&= \prod_{i=1}^{N} \left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}^{(i)})^2}{2\sigma^2} \right) && \text{(by pdf of } P\text{)}
\end{aligned}
$$

Negative likelihood of parameters for simple linear regression

- $\boldsymbol{w}$ maximizes $L(\boldsymbol{w}, \sigma^2)$ iff $\boldsymbol{w}$ maximizes $\log L(\boldsymbol{w}, \sigma^2)$ iff $\boldsymbol{w}$ minimizing $-\log L(\boldsymbol{w}, \sigma^2)$.

$$
\begin{aligned}
\mathrm{NLL}(\boldsymbol{w}, \sigma^2) &= -\log L(\boldsymbol{w}, \sigma^2) = -\log \prod_{i=1}^{N} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2}{2\sigma^2}\right) \\
&= -\sum_{i=1}^{N} \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2}{2\sigma^2}\right)\right] \\
&= -\sum_{i=1}^{N} \left[\log\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} + \log\exp\left(-\frac{(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2}{2\sigma^2}\right)\right] \\
&= -\sum_{i=1}^{N} \left[-\frac{1}{2}\log 2\pi\sigma^2 + \left(-\frac{(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2}{2\sigma^2}\right)\right] \\
&= \sum_{i=1}^{N} \frac{1}{2}\log 2\pi\sigma^2 + \sum_{i=1}^{N} \left(\frac{(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2}{2\sigma^2}\right) \\
&= \frac{N}{2}\log 2\pi\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y^{(i)} - \boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})^2
\end{aligned}
$$

Justification for the sum of squares error

- So, from a probabilistic perspective, linear regression aims to minimize the negative likelihood of parameters given N training data points:

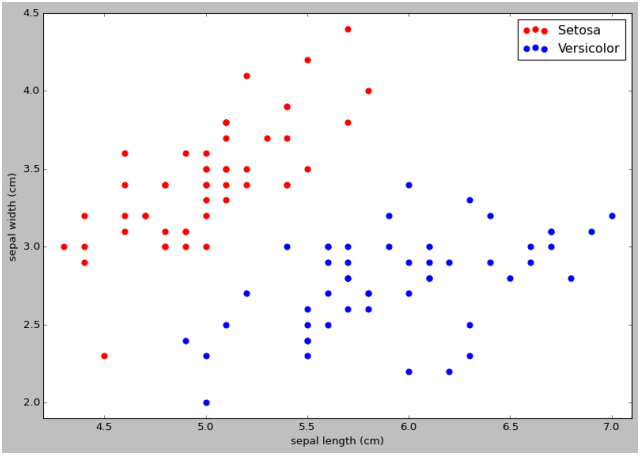
$$\mathrm{NLL}(\boldsymbol{w}, \sigma^2) = \frac{N}{2} \log 2\pi\sigma^2 + \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y^{(i)} - \boldsymbol{w}^\mathsf{T} \boldsymbol{x}^{(i)})^2$$

- If the training data is given/known, then the term $\frac{N}{2} \log 2\pi\sigma^2$ is fixed. Also, the term $\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y^{(i)} - \boldsymbol{w}^\mathsf{T} \boldsymbol{x}^{(i)})^2$ is proportional to $\frac{1}{2} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}^{(i)} - y^{(i)})^2$ because σ^2 does not depend on the data points and the residuals are squared.
- Hence, **the sum of squares error function is probabilistically justified**:

$$\mathrm{NLL}(\boldsymbol{w}, \sigma^2) \text{ is minimized} \quad \text{if and only if} \quad \frac{1}{2} \sum_{i=1}^{N} (\boldsymbol{w}^\mathsf{T} \boldsymbol{x}^{(i)} - y^{(i)})^2 = J(\boldsymbol{w}) \text{ is minimized}$$

Outline

Motivation



- Consider the problem of classifying two types of Iris flower (Setosa and Versicolor) based on the sepal length and width (data on the right).
- This is a binary classification problem, i.e., with target $y = \{0, 1\}$
- Is there a linear hyperplane separating the two classes?

Motivation

Let's consider a classification problem with target $y \in \{0, 1\}$

- Linear regression algorithms can be used, but it is quite simple to construct examples where the algorithm performs very poorly.
- Intuitively, it makes no sense to have target values larger than 1 or smaller than 0.
- Can we still use the same linear form, but ensure the output is neither below 0 nor above 1?

Motivation

Let's consider a classification problem with target $y \in \{0, 1\}$

- Linear regression algorithms can be used, but it is quite simple to construct examples where the algorithm performs very poorly.
- Intuitively, it makes no sense to have target values larger than 1 or smaller than 0.
- Can we still use the same linear form, but ensure the output is neither below 0 nor above 1?
  - Idea: scale the output of the linear form, which is $\mathbb{R}$, into the range $[0, 1]$ using some function (called the **link** function).
  - Intuition: the weights/parameters transform the input into a point in an intermediate space and the link function transform that point into a probability value that the input belongs to the class (given by the label).

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{e^z + 1}$$

What properties are satisfied by $g(z)$?

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why?

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{e^z + 1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.
- $g(z)$ is symmetric with respect to the point $(0, 0.5)$. Why?

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.
- $g(z)$ is symmetric with respect to the point $(0, 0.5)$. Why?
  For every $z \in \mathbb{R}$, $g(-z) = \frac{e^{-z}}{e^{-z}+1} = \frac{e^{-z}+1-1}{e^{-z}+1} = 1 - \frac{1}{e^{-z}+1} = 1 - g(z)$. So, let $h(z) = g(z) - 0.5$. Then, $h(-z) = g(-z) - 0.5 = 1 - g(z) - 0.5 = 0.5 - g(z) = -(g(z) - 0.5) = -h(z)$, i.e., $h(z)$ is an odd function, which always has a rotational symmety with respect to (0,0). Thus, $g(z) = h(z) + 0.5$ must have a rotational symmetry with respect to (0,0.5)

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.
- $g(z)$ is symmetric with respect to the point $(0, 0.5)$. Why?
  For every $z \in \mathbb{R}$, $g(-z) = \frac{e^{-z}}{e^{-z}+1} = \frac{e^{-z}+1-1}{e^{-z}+1} = 1 - \frac{1}{e^{-z}+1} = 1 - g(z)$. So, let $h(z) = g(z) - 0.5$. Then, $h(-z) = g(-z) - 0.5 = 1 - g(z) - 0.5 = 0.5 - g(z) = -(g(z) - 0.5) = -h(z)$, i.e., $h(z)$ is an odd function, which always has a rotational symmety with respect to (0,0). Thus, $g(z) = h(z) + 0.5$ must have a rotational symmetry with respect to (0,0.5)
- $g(z)$ is strictly increasing. Why?

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.
- $g(z)$ is symmetric with respect to the point $(0, 0.5)$. Why?
  For every $z \in \mathbb{R}$, $g(-z) = \frac{e^{-z}}{e^{-z}+1} = \frac{e^{-z}+1-1}{e^{-z}+1} = 1 - \frac{1}{e^{-z}+1} = 1 - g(z)$. So, let $h(z) = g(z) - 0.5$. Then, $h(-z) = g(-z) - 0.5 = 1 - g(z) - 0.5 = 0.5 - g(z) = -(g(z) - 0.5) = -h(z)$, i.e., $h(z)$ is an odd function, which always has a rotational symmety with respect to (0,0). Thus, $g(z) = h(z) + 0.5$ must have a rotational symmetry with respect to (0,0.5)
- $g(z)$ is strictly increasing. Why?
  For $z_1, z_2 \in \mathbb{R}$, if $z_1 < z_2$, then $e^{-z_1} > e^{-z_2}$. Thus, $g(z_1) = \frac{1}{1+e^{-z_1}} < \frac{1}{1+e^{-z_2}} = g(z_2)$.

Logistic function

Consider the following continuous function called **logistic function** or **sigmoid function**

$$g(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

What properties are satisfied by $g(z)$?

- $0 < g(z) < 1$ for every $z \in \mathbb{R}$. Why? Because $e^z > 0$ for all $z \in \mathbb{R}$.
- $g(z)$ is symmetric with respect to the point $(0, 0.5)$. Why?
  For every $z \in \mathbb{R}$, $g(-z) = \frac{e^{-z}}{e^{-z}+1} = \frac{e^{-z}+1-1}{e^{-z}+1} = 1 - \frac{1}{e^{-z}+1} = 1 - g(z)$. So, let $h(z) = g(z) - 0.5$. Then, $h(-z) = g(-z) - 0.5 = 1 - g(z) - 0.5 = 0.5 - g(z) = -(g(z) - 0.5) = -h(z)$, i.e., $h(z)$ is an odd function, which always has a rotational symmety with respect to (0,0). Thus, $g(z) = h(z) + 0.5$ must have a rotational symmetry with respect to (0,0.5)
- $g(z)$ is strictly increasing. Why?
  For $z_1, z_2 \in \mathbb{R}$, if $z_1 < z_2$, then $e^{-z_1} > e^{-z_2}$. Thus, $g(z_1) = \frac{1}{1+e^{-z_1}} < \frac{1}{1+e^{-z_2}} = g(z_2)$.
- The derivative $g'(z)$ of $g(z)$ satisfies the following: $g'(z) = g(z)(1 - g(z))$. (Prove it!)

Logistic function

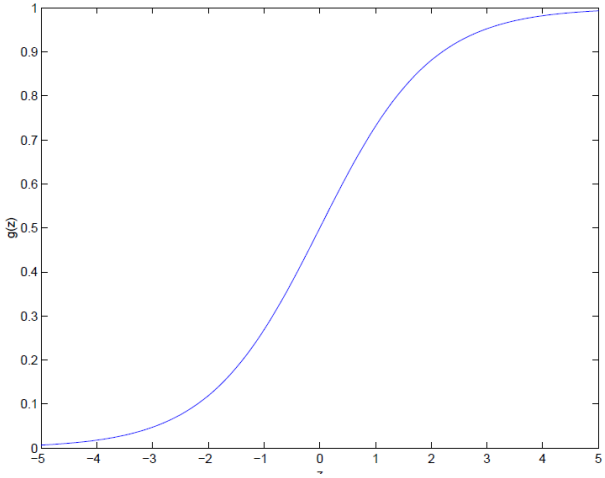


Image source: Andrew Ng, CS229 Lecture Notes for CS229 Fall 2018 course, Part II: Classification and Logistic Regression, page 17.

Logistic regression

- With input $\boldsymbol{x}$ of n features $(x_0, x_1, \ldots, x_n)$ where $x_0 = 1$, a hypothesis for logistic regression is of the form:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}}} = (1 + \exp(-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}))^{-1}$$

- The model parameter is $\boldsymbol{w}$, like in linear regression.
- Classification is then simply done by taking a threshold, e.g., if threshold is 0.5, then $\boldsymbol{x}$ is labeled 1 if $h_{\boldsymbol{w}}(\boldsymbol{x}) \geqslant 0.5$ and 0 otherwise.
- Logistic function is a smoothly increasing function and maps all real numbers into a real number between 0 and 1.
  - It can be shown that the choice of logistic function as a link function is a consequence of modeling class posterior $P(y|\boldsymbol{x})$ as Bernoulli distribution.

Why logistic function?

- Logistic regression assumes that $P(y = 1|\boldsymbol{x})$ follows Bernoulli distribution. Thus, $P(y = 1|\boldsymbol{x}) = p$ and $P(y = 0|\boldsymbol{x}) = 1 - p$ where p is the parameter of the distribution.
- Moreover, logistic regression employs logistic model that assumes a linear relationship between input features and the **log-odds** of the label $y = 1$.
  - **Odds** is the ratio of between the chance that the event occurs against the chance that the event does not occur. Thus the odds of $y = 1$ given $\boldsymbol{x}$ is $z = p/(1 - p)$
  - Odds is an expression of relative probabilities often used, e.g., in gambling.
  - The **logit** of p is the log-odds, i.e., $\log(p/(1 - p))$.
- The aforementioned linear relationship from the logistic model can be written:

$$\log\left(\frac{p}{1-p}\right) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$$

- **Exercise:** Show that $P(y = 1|\boldsymbol{x}) = p = (1 + \exp(-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}))^{-1}$, as given in the previous slide.
- **Exercise:** What is the mathematical relationship between the logit function $\text{logit}(x) = \log(x/1 - x)$ and the logistic function $g(x) = (1 + \exp(-x))^{-1}$?

Likelihood function for logistic regression

How do we learn $\boldsymbol{w}$? Or which value of $\boldsymbol{w}$ is best supported by the available data, i.e., the training examples?

Likelihood function for logistic regression

How do we learn $\boldsymbol{w}$? Or which value of $\boldsymbol{w}$ is best supported by the available data, i.e., the training examples?

- The data is given by the design matrix X (all training inputs as its roows) and the vector $\boldsymbol{y}$ (all target values in the training set).
- For a particular value of $\boldsymbol{w}$, the **likelihood** function $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ gives the probability (or probability density) of the data $X, \boldsymbol{y}$ to occur when the parameter of the model is $\boldsymbol{w}$.

Likelihood function for logistic regression

How do we learn $\boldsymbol{w}$? Or which value of $\boldsymbol{w}$ is best supported by the available data, i.e., the training examples?

- The data is given by the design matrix X (all training inputs as its roows) and the vector $\boldsymbol{y}$ (all target values in the training set).
- For a particular value of $\boldsymbol{w}$, the **likelihood** function $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ gives the probability (or probability density) of the data $X, \boldsymbol{y}$ to occur when the parameter of the model is $\boldsymbol{w}$.
- Our model represents the (probabilistic) relationship between the known input X and target $\boldsymbol{y}$. That is, the conditional distribution $p_{\boldsymbol{w}}(\boldsymbol{y}|X)$ of $\boldsymbol{y}$ given X, which is parameterized by $\boldsymbol{w}$.

Likelihood function for logistic regression

How do we learn $\boldsymbol{w}$? Or which value of $\boldsymbol{w}$ is best supported by the available data, i.e., the training examples?

- The data is given by the design matrix X (all training inputs as its roows) and the vector $\boldsymbol{y}$ (all target values in the training set).
- For a particular value of $\boldsymbol{w}$, the **likelihood** function $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ gives the probability (or probability density) of the data $X, \boldsymbol{y}$ to occur when the parameter of the model is $\boldsymbol{w}$.
- Our model represents the (probabilistic) relationship between the known input X and target $\boldsymbol{y}$. That is, the conditional distribution $p_{\boldsymbol{w}}(\boldsymbol{y}|X)$ of $\boldsymbol{y}$ given X, which is parameterized by $\boldsymbol{w}$.
- Thus, the likelihood of $\boldsymbol{w}$ given the data: $L_{X,\boldsymbol{y}}(\boldsymbol{w}) = p_{\boldsymbol{w}}(\boldsymbol{y}, X) = p_{\boldsymbol{w}}(\boldsymbol{y}|X)$ where the last equality is because in our model the input (for which X is a sample) is always known.

Likelihood function for logistic regression

How do we learn $\boldsymbol{w}$? Or which value of $\boldsymbol{w}$ is best supported by the available data, i.e., the training examples?

- The data is given by the design matrix X (all training inputs as its roows) and the vector $\boldsymbol{y}$ (all target values in the training set).
- For a particular value of $\boldsymbol{w}$, the **likelihood** function $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ gives the probability (or probability density) of the data $X, \boldsymbol{y}$ to occur when the parameter of the model is $\boldsymbol{w}$.
- Our model represents the (probabilistic) relationship between the known input X and target $\boldsymbol{y}$. That is, the conditional distribution $p_{\boldsymbol{w}}(\boldsymbol{y}|X)$ of $\boldsymbol{y}$ given X, which is parameterized by $\boldsymbol{w}$.
- Thus, the likelihood of $\boldsymbol{w}$ given the data: $L_{X,\boldsymbol{y}}(\boldsymbol{w}) = p_{\boldsymbol{w}}(\boldsymbol{y}, X) = p_{\boldsymbol{w}}(\boldsymbol{y}|X)$ where the last equality is because in our model the input (for which X is a sample) is always known.

Our aim is thus finding the value of $\boldsymbol{w}$ that maximizes the conditional probability above, i.e., **maximizes the likelihood**.

Maximizing likelihood for logistic regression

- As noted previously, for logistic regression, $P(y = 1|\boldsymbol{x})$ follows Bernoulli distribution with parameter $p = h_{\boldsymbol{w}}(\boldsymbol{x}) = (1 + \exp(-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}))^{-1}$. Thus,

$$p_{\boldsymbol{w}}(y{=}1|\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{x}) \qquad\qquad p_{\boldsymbol{w}}(y{=}0|\boldsymbol{x}) = 1 - h_{\boldsymbol{w}}(\boldsymbol{x})$$

  or more compactly as a single (mass) function:

Maximizing likelihood for logistic regression

- As noted previously, for logistic regression, $P(y = 1|\boldsymbol{x})$ follows Bernoulli distribution with parameter $p = h_{\boldsymbol{w}}(\boldsymbol{x}) = (1 + \exp(-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}))^{-1}$. Thus,

$$p_{\boldsymbol{w}}(y{=}1|\boldsymbol{x}) = h_{\boldsymbol{w}}(\boldsymbol{x}) \qquad p_{\boldsymbol{w}}(y{=}0|\boldsymbol{x}) = 1 - h_{\boldsymbol{w}}(\boldsymbol{x})$$

  or more compactly as a single (mass) function:

$$p_{\boldsymbol{w}}(y|\boldsymbol{x}) = (h_{\boldsymbol{w}}(\boldsymbol{x}))^{y}(1 - h_{\boldsymbol{w}}(\boldsymbol{x}))^{1-y}$$

- This corresponds to our earlier intuition that $\boldsymbol{w}$ brings $\boldsymbol{x}$ to $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$ and then the logistic function brings $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$ to the appropriate probability value.

y = 1 di cek aja yang (h_w(x)) = y = 1

y = 0 di evaluasi karena 1-y = 0, maka ((1-h_w(x))

Maximizing likelihood for logistic regression (2)

- Next, we assume that X and $\boldsymbol{y}$ give m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, each generated independently.
- Then, the likelihood $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ corresponds to joint distribution over all training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, m$.

$$L_{X,\boldsymbol{y}}(\boldsymbol{w}) =$$

Maximizing likelihood for logistic regression (2)

- Next, we assume that X and $\boldsymbol{y}$ give m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, each generated independently.
- Then, the likelihood $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ corresponds to joint distribution over all training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, m$.

$$L_{X,\boldsymbol{y}}(\boldsymbol{w}) = \prod_{i=1}^{m} p_{\boldsymbol{w}}(y^{(i)}|\boldsymbol{x}^{(i)}) = \prod_{i=1}^{m} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{1-y^{(i)}}$$

Maximizing likelihood for logistic regression (2)

- Next, we assume that X and $\boldsymbol{y}$ give m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, each generated independently.
- Then, the likelihood $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ corresponds to joint distribution over all training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, m$.

$$L_{X,\boldsymbol{y}}(\boldsymbol{w}) = \prod_{i=1}^{m} p_{\boldsymbol{w}}(y^{(i)}|\boldsymbol{x}^{(i)}) = \prod_{i=1}^{m} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{1-y^{(i)}}$$

- Maximimizing the likelihood is equivalent to maximizing the **log-likelihood** $\ell(\boldsymbol{w})$ whose form is easier for computing derivatives (the log is natural logarithm):

$$\ell(\boldsymbol{w}) = \log L(\boldsymbol{w}) =$$

Maximizing likelihood for logistic regression (2)

- Next, we assume that X and $\boldsymbol{y}$ give m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, each generated independently.
- Then, the likelihood $L_{X,\boldsymbol{y}}(\boldsymbol{w})$ corresponds to joint distribution over all training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, m$.

$$L_{X,\boldsymbol{y}}(\boldsymbol{w}) = \prod_{i=1}^{m} p_{\boldsymbol{w}}(y^{(i)}|\boldsymbol{x}^{(i)}) = \prod_{i=1}^{m} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{y^{(i)}} \left(1 - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\right)^{1-y^{(i)}}$$

- Maximimizing the likelihood is equivalent to maximizing the **log-likelihood** $\ell(\boldsymbol{w})$ whose form is easier for computing derivatives (the log is natural logarithm):

$$\ell(\boldsymbol{w}) = \log L(\boldsymbol{w}) = \sum_{i=1}^{m} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right)$$

Maximizing likelihood for logistic regression (3)

maximizing (cari nilai maksimal)

For maximization, we employ **gradient ascent** instead of gradient descent.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \alpha \nabla \ell(\boldsymbol{w})$$

berbeda di tanda (+ kalau gradient ascent)

Maximizing likelihood for logistic regression (3)

For maximization, we employ **gradient ascent** instead of gradient descent.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \alpha \nabla \ell(\boldsymbol{w})$$

We compute the partial derivative for each weight w_j:

$$\frac{\partial}{\partial w_j} \ell(\boldsymbol{w}) = \frac{\partial}{\partial w_j} \sum_{i=1}^{m} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right)$$

Maximizing likelihood for logistic regression (3)

For maximization, we employ **gradient ascent** instead of gradient descent.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \alpha \nabla \ell(\boldsymbol{w})$$

We compute the partial derivative for each weight w_j:

$$\begin{aligned}
\frac{\partial}{\partial w_j} \ell(\boldsymbol{w}) &= \frac{\partial}{\partial w_j} \sum_{i=1}^{m} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right) \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial w_j} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right)
\end{aligned}$$

Maximizing likelihood for logistic regression (3)

For maximization, we employ **gradient ascent** instead of gradient descent.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \alpha \nabla \ell(\boldsymbol{w})$$

We compute the partial derivative for each weight w_j:

$$\begin{aligned}
\frac{\partial}{\partial w_j} \ell(\boldsymbol{w}) &= \frac{\partial}{\partial w_j} \sum_{i=1}^{m} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right) \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial w_j} \left(y^{(i)} \log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{w}}(x^{(i)})) \right) \\
&= \sum_{i=1}^{m} \left(\frac{y^{(i)}}{h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})} - \frac{1 - y^{(i)}}{1 - h_{\boldsymbol{w}}(x^{(i)})} \right) \frac{\partial}{\partial w_j} h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) \qquad \text{(because } \tfrac{\partial}{\partial x} \log x = \tfrac{1}{x}\text{)}
\end{aligned}$$

Maximizing likelihood for logistic regression (3)

For maximization, we employ **gradient ascent** instead of gradient descent.

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \alpha \nabla \ell(\boldsymbol{w})$$

We compute the partial derivative for each weight w_j:

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(\boldsymbol{w}) &= \frac{\partial}{\partial w_j}\sum_{i=1}^{m}\left(y^{(i)}\log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1-y^{(i)})\log(1-h_{\boldsymbol{w}}(x^{(i)}))\right) \\
&= \sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y^{(i)}\log h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) + (1-y^{(i)})\log(1-h_{\boldsymbol{w}}(x^{(i)}))\right) \\
&= \sum_{i=1}^{m}\left(\frac{y^{(i)}}{h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-h_{\boldsymbol{w}}(x^{(i)})}\right)\frac{\partial}{\partial w_j}h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) \qquad \text{(because } \tfrac{\partial}{\partial x}\log x = \tfrac{1}{x}\text{)} \\
&= \sum_{i=1}^{m}\left(\frac{y^{(i)}}{g(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-g(\boldsymbol{w}^{\mathsf{T}}x^{(i)})}\right)\frac{\partial}{\partial w_j}g(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}^{(i)})
\end{aligned}$$

Maximizing likelihood for logistic regression (4)

Since g is the logistic function, using chain rule we obtain:

$$\frac{\partial}{\partial w_j}\ell(\boldsymbol{w})$$

Maximizing likelihood for logistic regression (4)

Since g is the logistic function, using chain rule we obtain:

$$\frac{\partial}{\partial w_j}\ell(\boldsymbol{w}) = \sum_{i=1}^{m}\left(\frac{y^{(i)}}{g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-g(\boldsymbol{w}^\mathsf{T}x^{(i)})}\right)g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))\frac{\partial}{\partial w_j}\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}$$

Maximizing likelihood for logistic regression (4)

Since g is the logistic function, using chain rule we obtain:

$$\begin{aligned}\frac{\partial}{\partial w_j}\ell(\boldsymbol{w}) &= \sum_{i=1}^{m}\left(\frac{y^{(i)}}{g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-g(\boldsymbol{w}^\mathsf{T}x^{(i)})}\right)g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))\frac{\partial}{\partial w_j}\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)} \\ &= \sum_{i=1}^{m}\big(y^{(i)}(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})) - (1-y^{(i)})g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})\big)x_j^{(i)}\end{aligned}$$

Maximizing likelihood for logistic regression (4)

Since g is the logistic function, using chain rule we obtain:

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(\boldsymbol{w}) &= \sum_{i=1}^{m}\left(\frac{y^{(i)}}{g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-g(\boldsymbol{w}^\mathsf{T}x^{(i)})}\right) g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))\frac{\partial}{\partial w_j}\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)} \\
&= \sum_{i=1}^{m}\big(y^{(i)}(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})) - (1-y^{(i)})g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})\big)x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)} - g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))x_j^{(i)}
\end{aligned}$$

Maximizing likelihood for logistic regression (4)

Since g is the logistic function, using chain rule we obtain:

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(\boldsymbol{w}) &= \sum_{i=1}^{m}\left(\frac{y^{(i)}}{g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})} - \frac{1-y^{(i)}}{1-g(\boldsymbol{w}^\mathsf{T}x^{(i)})}\right) g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))\frac{\partial}{\partial w_j}\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)} \\
&= \sum_{i=1}^{m}\big(y^{(i)}(1-g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})) - (1-y^{(i)})g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)})\big)x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)} - g(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}^{(i)}))x_j^{(i)} \\
&= \sum_{i=1}^{m}(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))x_j^{(i)}
\end{aligned}$$

Batch & stochastic gradient ascent for logistic regression

The gradient ascent update turned out to be the same as the gradient descent update in linear regression, except that $h_{\boldsymbol{w}}$ is no longer linear in $\boldsymbol{x}^{(i)}$. This is actually not coincidence – for deeper discussion, read more on the topic of Generalized Linear Model.

$$w_j \coloneqq w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}))x_j^{(i)}$$

Thus, analogous to linear regression we have two version of gradient ascents.

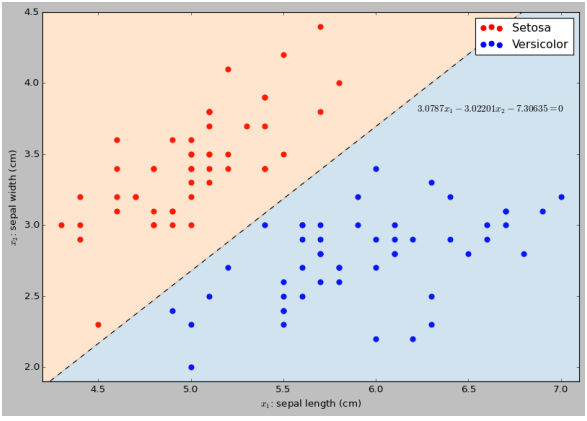
Batch gradient ascent from an initial value of $\boldsymbol{w} = (w_0, \ldots, w_n)^\mathsf{T}$.

- Repeat until convergence:
  - For every j: $w_j \coloneqq w_j + \alpha \sum_{i=1}^{m} \big(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\big)x_j^{(i)}$

Stochastic gradient ascent from an initial value of $\boldsymbol{w} = (w_0, \ldots, w_n)^\mathsf{T}$.

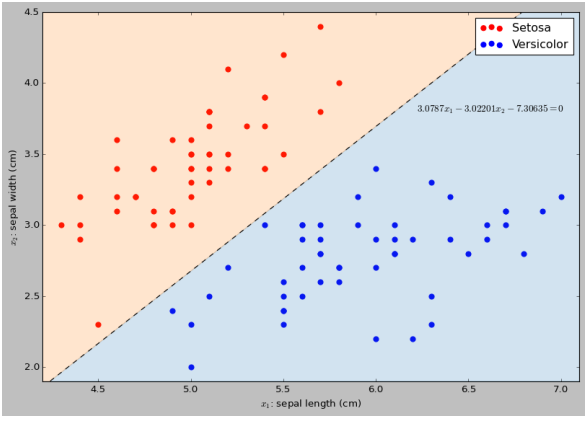
- Loop until convergence
  - Loop for every training example $\boldsymbol{x}^{(i)}$:
    - For every j: $w_j \coloneqq w_j + \alpha \big(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})\big)x_j^{(i)}$

Linear decision boundary (1)



- Although $h_{\boldsymbol{w}}(\boldsymbol{x})$ is nonlinear in logistic regression, it is still a **linear model** as the decision boundary is linear as seen in the figure.
- The linear hyperplane decision boundary satisfies the equation $z = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = 0$, i.e., $w_0 + w_1 x_1 + \cdots + w_n x_n = 0$ where z is the logit of $P(y = 1|\boldsymbol{x})$.
- Intuitively, z is in the "intermediate" space between the input space and the output probability space. The decision boundary satisfies $z = 0$, corresponding to $P(y = 1|\boldsymbol{x}) = 0.5$, or odds 1:1.

Linear decision boundary (2)

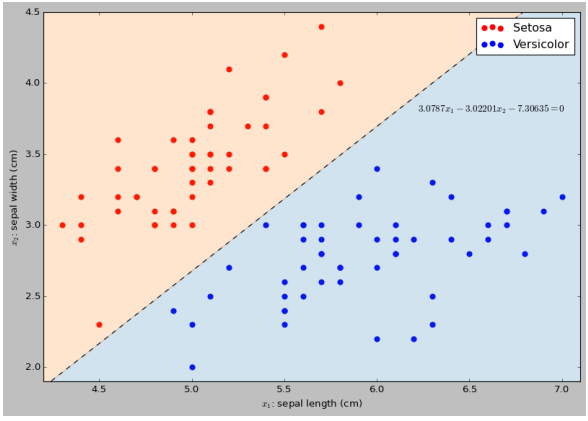



```python
# Getting original iris data
iris_data = load_iris(return_X_y=False)

# load only instance data whose target is 0 and 1 and
# only take the first two features
train_x = iris_data.data[iris_data.target < 2][:, :2]
train_y = iris_data.target[iris_data.target < 2]

# fitting the model and get parameters w0, w1, and w2
logreg = linear_model.LogisticRegression(random_state=0,max_iter=1000)
clf = logreg.fit(train_x, train_y)
w0 = clf.intercept_[0]
w1, w2 = clf.coef_.T
```

Linear decision boundary (3)



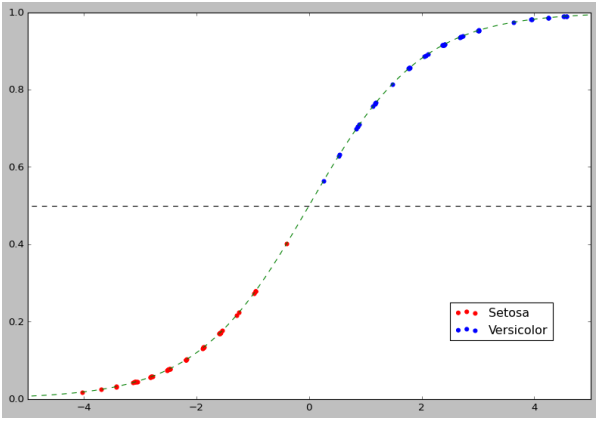

```
# prepare visualization
fig=plt.figure(figsize=(12, 8))
xs, ys = train_x[:,0], train_x[:,1]
xlim = (xs.min()-0.1, xs.max()+0.1)
plt.xlim(xlim)
ylim = (ys.min()-0.1, ys.max()+0.1)
plt.ylim(ylim)
plt.xlabel(iris_data.feature_names[0])
plt.ylabel(iris_data.feature_names[1])
plt.style.use('classic')

# visualize data points
plt.scatter(xs[train_y == 0], ys[train_y == 0], s=40,  color='red',
            label=iris_data.target_names[0])
plt.scatter(xs[train_y == 1], ys[train_y == 1], s=40,  color='blue',
            label=iris_data.target_names[1])

# visualize decision boundary
w0 = clf.intercept_[0]
w1, w2 = clf.coef_.T
xd = np.array([xlim[0], xlim[1]])
yd = (-w1/w2)*xd + (-w0/w2)
plt.plot(xd, yd, 'k', lw=1, ls='--')
plt.fill_between(xd, yd, ylim[0], color='tab:blue', alpha=0.2)
plt.fill_between(xd, yd, ylim[1], color='tab:orange', alpha=0.2)
label = '$%g_1  %g_2  %g=0$' % (w1,w2,w0)
plt.annotate(label, (6.2, 3.8), textcoords='offset points',
             xytext=(5,0), ha='left')

plt.legend()
plt.show()
```

Linear decision boundary (4)



- Probability that a training example belongs to class 1 (Versicolor) lies on the logistic function's curve.
- Since the classes are linearly separable, no example of Versicolor that lies under $y = 0.5$ and no example of Setosa that lies above $y = 0.5$.

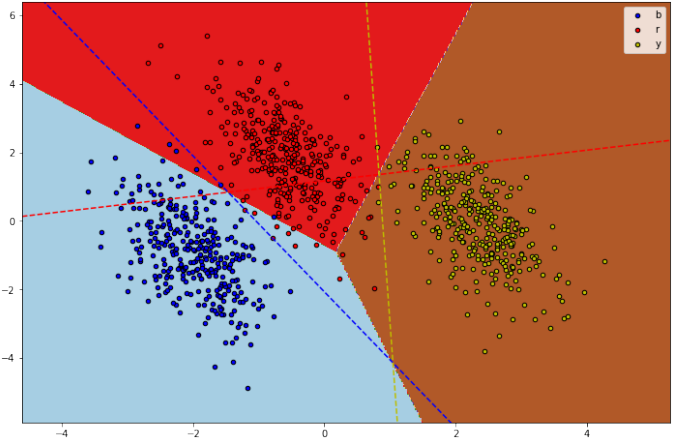
Outline

Multiclass classification

- Logistic regression introduced in the previous section is intended for binary classification.
- What if the number of classes is more than 2, i.e., (**multiclass classification**)?
  - Note: multiclass classification differs from **multilabel classification** where one datapoint can have multiple labels simultaneously.
- **One-vs-rest approach**: Let $K > 2$ be the number of classes and L be a binary classification algorithm that can return real-valued confidence or probability scores (such as logistic regression).
  - For each $k = 1, \ldots, K$, we train a binary classifier f_k to separate instances of class k from instances of other classes ("non-k").
  - Given an instance $\boldsymbol{x}$, its class is determined by finding k such that $f_k(\boldsymbol{x})$ returns the highest confidence or probability score.

  Loss function is applied independently between individual binary classifiers.
- A better choice is true **multinomial logistic regression**, a.k.a., **softmax regression** where loss function is applied across all classes.
  - Typically, the probability estimate is calibrated better.

Multiclass classification: One-vs-rest approach



- The dotted line represents the decision boundary of the individual binary classifiers.
- The solid line represents the decision boundary when accounting for the confidence/probability scores using one-versus-rest approach.

Softmax regression (1)

- Recall that in binary logistic regression, we have that

$$\log\left(\frac{P(y{=}1|\boldsymbol{x})}{P(y{=}0|\boldsymbol{x})}\right) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}$$

- If we have $K > 2$ classes, we can work as if the labels are $0, 1, 2, \ldots, K-1$.
- So, we can run $K-1$ independent binary logistic regression model between class $k \in \{1, 2, \ldots, K-1\}$ versus class 0. Assuming the logistic model, we have the following set of linear relationships:

$$\log\left(\frac{P(y{=}1|\boldsymbol{x})}{P(y{=}0|\boldsymbol{x})}\right) = \boldsymbol{w}_1^{\mathsf{T}}\boldsymbol{x}, \quad \ldots, \quad \log\left(\frac{P(y{=}K-1|\boldsymbol{x})}{P(y{=}0|\boldsymbol{x})}\right) = \boldsymbol{w}_{K-1}^{\mathsf{T}}\boldsymbol{x}$$

  where $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{K-1}$ are $K-1$ vectors of parameters, each associated to class k.
- Hence, we can write

$$P(y{=}1|\boldsymbol{x}) = P(y{=}0|\boldsymbol{x})\exp(\boldsymbol{w}_1^{\mathsf{T}}\boldsymbol{x}), \quad \ldots, \quad P(y{=}K-1|\boldsymbol{x}) = P(y{=}0|\boldsymbol{x})\exp(\boldsymbol{w}_{K-1}^{\mathsf{T}}\boldsymbol{x})$$

Softmax regression (2)

- Since all probabilities sum to 1, i.e., $\sum_{k=0}^{K-1} P(y{=}k|\boldsymbol{x}) = 1$, we obtain

$$P(y{=}0|\boldsymbol{x}) = 1 - \sum_{k=1}^{K-1} P(y{=}k|\boldsymbol{x}) = 1 - \sum_{k=1}^{K-1} P(y{=}0|\boldsymbol{x}) \exp(\boldsymbol{w}_k^\mathsf{T}|\boldsymbol{x})$$

- This implies that

$$P(y{=}0|\boldsymbol{x}) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})}$$

- Substituting it to the previous equations and setting $\boldsymbol{w}_0 = \mathbf{0}$, i.e., the zero vector, we have

$$P(y{=}1|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_1^\mathsf{T}\boldsymbol{x})}{1 + \sum_{k=1}^{K-1} \exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})}, \quad \ldots, \quad P(y{=}K-1|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_{K-1}^\mathsf{T}\boldsymbol{x})}{1 + \sum_{k=1}^{K-1} \exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})},$$

$$P(y{=}0|\boldsymbol{x}) = \frac{1}{1 + \sum_{k=1}^{K-1} \exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})} = \frac{\exp(\boldsymbol{w}_0^\mathsf{T}\boldsymbol{x})}{\exp(\boldsymbol{w}_0^\mathsf{T}\boldsymbol{x}) + \sum_{k=1}^{K-1} \exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})}$$

The latter is because $\exp(\boldsymbol{w}_0^\mathsf{T}\boldsymbol{x}) = \exp(0) = 1$.

Softmax regression (3)

- Thus, we can write for all class label $k = 0, \ldots, K-1$:

$$P(y{=}k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})}{\sum_{\ell=0}^{K-1}\exp(\boldsymbol{w}_\ell^\mathsf{T}\boldsymbol{x})}$$

- Note though the above formula still assumes that $\boldsymbol{w}_0 = \boldsymbol{0}$. Fortunately, the formula is invariant under translation by the same value in each dimension:

$$P(y{=}k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\mathsf{T}\boldsymbol{x})}{\sum_{\ell=0}^{K-1}\exp(\boldsymbol{w}_\ell^\mathsf{T}\boldsymbol{x})} = \frac{\exp((\boldsymbol{t}+\boldsymbol{w}_k)^\mathsf{T}\boldsymbol{x})}{\sum_{\ell=0}^{K-1}\exp((\boldsymbol{t}+\boldsymbol{w}_\ell)^\mathsf{T}\boldsymbol{x})}$$

  for any constant vector $\boldsymbol{t}$.

- So instead of fixing $\boldsymbol{w}_0 = \boldsymbol{0}$, we simply allow $\boldsymbol{w}_0$ as another (redundant) parameter vector and the right hand side of the above formulation is the **softmax** of $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$.
  - Softmax = softargmax.
  - It is a generalization of the logistic function to multiple dimensions.
- Finally, as usual, prediction is simply obtained by taking k such that $P(y = k|\boldsymbol{x})$ is the largest.

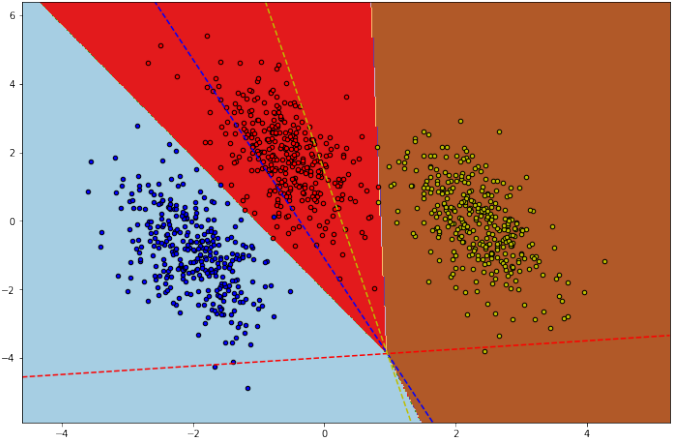
Finding the parameters of softmax regression

- As in logistic regression, parameter vectors $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{K-1}$ can be computed by maximizing the likelihood $L_{X,\boldsymbol{y}}(\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{K-1})$ corresponding to joint distribution over all training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$, $i = 1, \ldots, N$.

$$L_{X,\boldsymbol{y}}(\boldsymbol{w}_0, \ldots, \boldsymbol{w}_{K-1}) = \prod_{i=1}^{N} P(y{=}y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{w}_0, \ldots, \boldsymbol{w}_{K-1})$$

  Here, the probabilities inside the product use the softmax formulation defined previously.
- Any gradient-based optimization in principle can optimize this objective, although deriving derivatives by hand (as in the previous section) is beyond the scope of this lecture.
- In Scikit-learn, the module `LogisticRegression` can be used for multinomial/softmax case by passing an appropriate parameter – see the documentation in more detail.

Softmax regression: Class boundary



- The dotted line represents the decision boundary of the individual binary classifiers.
- The solid line represents the decision boundary when accounting for the confidence/probability scores using softmax regression.

Outline

Overfitting in linear regression

- Although linear regression can be considered a high-bias low-variance method (compared to e.g., polynomial regression), in practice, linear regression can still suffer from overfitting particularly when it is applied to high-dimensional data.
- Consider a linear regression model with large n:

$$h(\boldsymbol{x}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- One approach to avoid overfitting is to perform dimensionality reduction either manually (e.g., best-subset selection) or using a dimensionality reduction technique during pre-processing (e.g., PCA, t-SNE).
  - Intuition: discarded features are essentially given zero weight.
- Alternative to the above approach is to incorporate additional term(s) **inside** the loss function to produce a similar effect. This is called **regularization**, which can significantly reduce variance without increasing the bias too much.
- Two prominent regularization formulations for linear regression (and generally, polynomial regression) are:
  - ridge regression; and
  - lasso regression.

Ridge regression

- Recall the sum-of-squares loss function for linear regression:

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

- **Ridge regression** modifies the above loss function by adding a **regularization term** or **regularizer** resulting in the following loss :

$$J(\boldsymbol{w}, \lambda) = \frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2 + \lambda\|\boldsymbol{w}\|_2^2$$

  where $\|\boldsymbol{w}\|_2$ refers to **L2-norm** of vector $\boldsymbol{w}$ and λ is a **regularization parameter**. Thus, $\lambda\|\boldsymbol{w}\|_2^2$ is also called **L2 regularizer**.
- λ is **hyperparameter**: its value must be set before training and can be fine-tuned via cross-validation.

Ridge regression (2)

- Expanding the norm: $J(\boldsymbol{w}, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 + \lambda(w_0^2 + \cdots + w_n^2)$
- What would happen to the value of parameters w_i if the loss function is minimized?

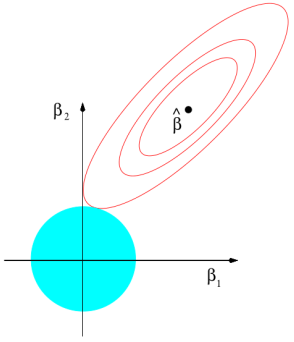
Ridge regression (2)

- Expanding the norm: $J(\boldsymbol{w}, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 + \lambda(w_0^2 + \cdots + w_n^2)$
- What would happen to the value of parameters w_i if the loss function is minimized?
  - If λ is large than all values of w_i's will get close to zero, hence may cause underfitting.
  - If $\lambda = 0$, the values would behave as in the usual OLS regression.
  - λ determines the penalty to the flexibility of the model.

Ridge regression (2)

- Expanding the norm: $J(\boldsymbol{w}, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 + \lambda(w_0^2 + \cdots + w_n^2)$
- What would happen to the value of parameters w_i if the loss function is minimized?
  - If λ is large than all values of w_i's will get close to zero, hence may cause underfitting.
  - If $\lambda = 0$, the values would behave as in the usual OLS regression.
  - λ determines the penalty to the flexibility of the model.
- For nonlinear regression, regularization term can yield a smoothing effect to the model.
  - E.g., in polynomial regression, the weight of unimportant features can be significantly reduced, causing the curve to be smoother. (See: `https://stats.stackexchange.com/questions/226553/why-use-regularisation-in-polynomial-regression-instead-of-lowering-the-degree`)
- Gradient-based optimization can still be used as usual, though closed-form solution for ridge (linear) regression is possible.
- OLS regression parameters are scale-invariant: if features are multiplied by a constant c, then weights can be multiplied by $1/c$ to obtain the same result. But, ridge regression is **not** scale invariant.
  - Features should be standardized prior to applying ridge regression.

Ridge regression (3)



cari koordinat error yang errornya seminimal mungkin dan memotong koordinat lingkaran (intersect)

Ellipses are countours of the original loss function (without regularization term). Blue circle is the region of values satisfying the constraint due to regularization.

Figure is from Hastie et al., "The Elements of Statistical Learning" 2nd ed.

- Ridge regression can be expressed as:
  minimize $\frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2$
  subject to $\sum_{j=0}^{n} w_j^2 \leqslant t$ for some parameter t.
- To satisfy the constraint, w_i's must be such that $\boldsymbol{w}$ lies inside the circle/sphere, i.e., optimal parameters must be at the intersection between the contours and the circle.
- Intersection between the contours and the circle will not generally occur at an axis.
  - Final model of ridge regression will include all features, including unimportant ones, albeit with (possibly) small weights.
  - Model is much more difficult to interpret.

Lasso regression

- Instead of L2-norm, one can also use L1-norm for the regularizer, hence yielding the following loss function.

$$J(\boldsymbol{w}, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 + \lambda \|\boldsymbol{w}\|_1$$

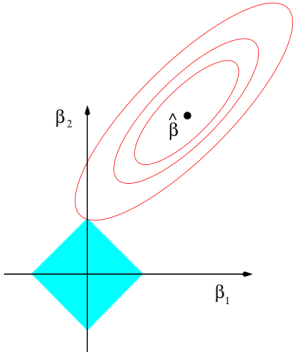
  where $\|\boldsymbol{w}\|_1$ refers to **L1-norm** of vector $\boldsymbol{w}$ and λ is a **regularization parameter**. Thus, $\lambda \|\boldsymbol{w}\|_1$ is also called **L1 regularizer**.

- Expanding the norm, we obtain

$$J(\boldsymbol{w}, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 + \lambda (|w_0| + \cdots + |w_n|)$$

- The resulting regression model is called **lasso regression** (lasso = least absolute shrinkage and selection operator)

Lasso regression (2)



Ellipses are countours of the original loss function (without regularization term). Blue diamond is the region of values satisfying the constraint due to regularization.

Figure is from Hastie et al., "The Elements of Statistical Learning" 2nd ed.

- Lasso regression can be expressed as:
  minimize $\frac{1}{2}\sum_{i=1}^{N}\left(h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2$
  subject to $\sum_{j=0}^{n}|w_j| \leqslant t$ for some parameter t.
- To satisfy the constraint, w_i's must be such that $\boldsymbol{w}$ lies inside the diamond, i.e., optimal parameters must be at the intersection between the contours and the diamond.
- The optimization becomes quadratic programming
- Intersection between the contours and the diamond often occurs at an axis.
  - Some features may be assigned zero weight, dropping the features from the model (especially in higher dimensional data).
  - Model is usually sparse.