SME3006 Machine Learning – 2025 Fall

Linear and Logistic Regression

INHA UNIVERSITY

Supervised learning

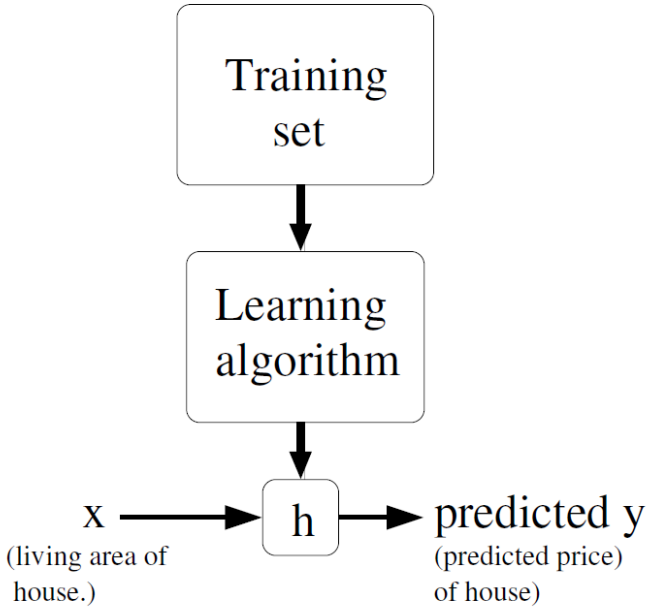
- House size vs Price
  - We want to predict the prices of other houses as a function of the size

Size	Price
2480	500
560	110
1400	290
1033	200
600	100
...	...
3456	600

Supervised learning

- Given a training set, to learn a function $h: X \rightarrow Y$
  - h is a good predictor (hypothesis)

- When the target value is continuous, we call the problem a regression (*when y can take on a small number of discrete values → classification)



Linear regression

> To make our housing example more interesting, let's consider a richer dataset in which we also know the number of bedrooms

Living area	#bedrooms	Prices
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
…	…	…

> Approximate y as a linear function of x

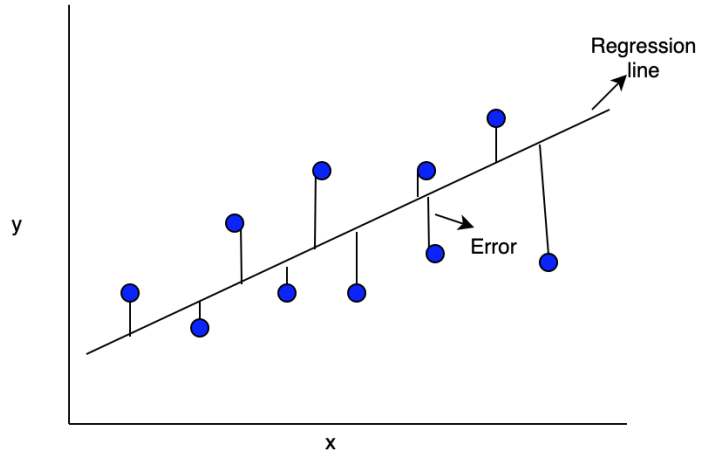
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- θ_i are the parameters (weights)

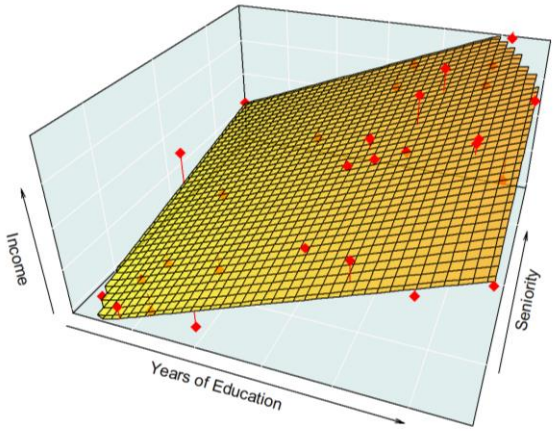
Linear regression

- In general, $h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^\top x$ where $x_0 = 1$

- To learn the parameters θ, we define a function that measures, for each value of θ, how close the h is to the corresponding y
  - We define the cost function
  - $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

  - We want to find θ that minimizes the cost function





The normal equations

- Matrix form

$$
\begin{aligned}
&x_{11}\theta_1 + \cdots + x_{1d}\theta_d = y_1 \\
&x_{21}\theta_1 + \cdots + x_{2d}\theta_d = y_2 \\
&\cdots \\
&x_{n1}\theta_1 + \cdots + x_{nd}\theta_d = y_n
\end{aligned}
$$

- $X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} = \begin{bmatrix} -(x^{(1)})^\top - \\ \vdots \\ -(x^{(n)})^\top - \end{bmatrix}$, $y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$, $\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

- Then, $X\theta - y = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}$

- $\frac{1}{2}\|X\theta - y\|^2 = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = J(\theta)$

The normal equations

- minimize $J = \frac{1}{2}\|X\theta - y\|^2 = \frac{1}{2}(X\theta - y)^\top (X\theta - y)$

- $\frac{d}{d\theta}\frac{1}{2}(X\theta - y)^\top (X\theta - y) = \frac{d}{d\theta}(\theta^\top X^\top X\theta - \theta^\top X^\top y - y^\top X\theta + y^\top y)$

$$= \frac{d}{d\theta}\frac{1}{2}(\theta^\top X^\top X\theta - 2y^\top X\theta)$$

$$= \frac{1}{2}(2X^\top X\theta - 2X^\top y)$$

$$= X^\top (X\theta - y) = 0$$

- Normal equation: $X^\top X\theta = X^\top y$
  - $\theta = (\mathrm{X}^\top \mathrm{X})^{-1} X^\top y$ (pseudo-inverse)

Linear regression

- To minimize $J(\theta)$, we use search algorithm
- Start with some 'initial guess' and repeatedly changes θ to make $J(\theta)$ smaller

- Gradient descent:

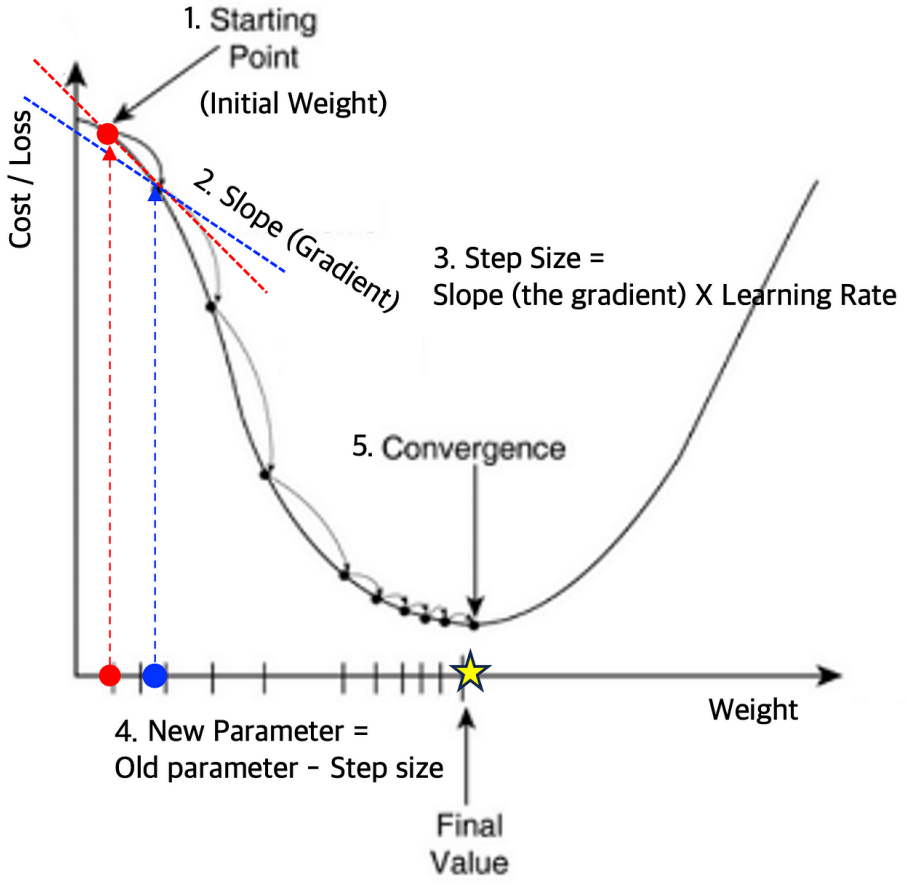
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \quad \alpha \text{ is the learning rate}$$

- $$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \left[\frac{1}{2}(h_\theta(x) - y)\right]^2$$
$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y)$$
$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (\theta^\top x - y)$$
$$= (h_\theta(x) - y) x_j$$

Gradient descent

- Gradient descent:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

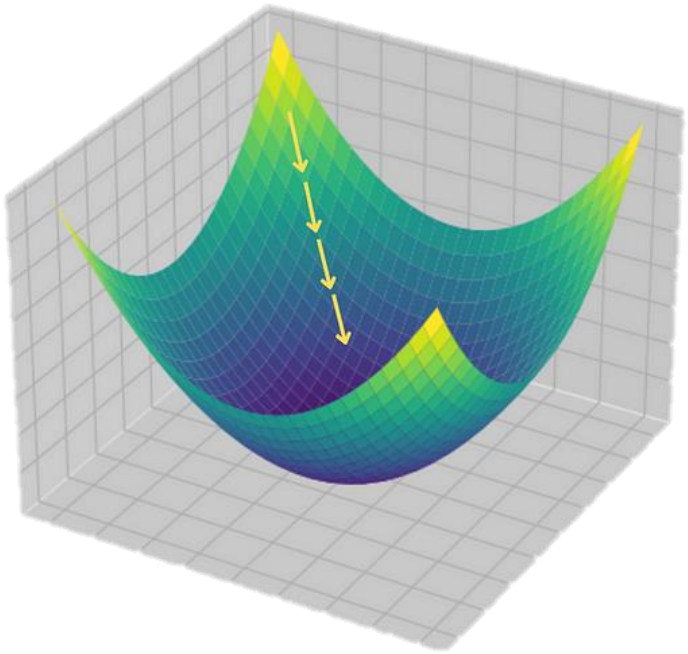


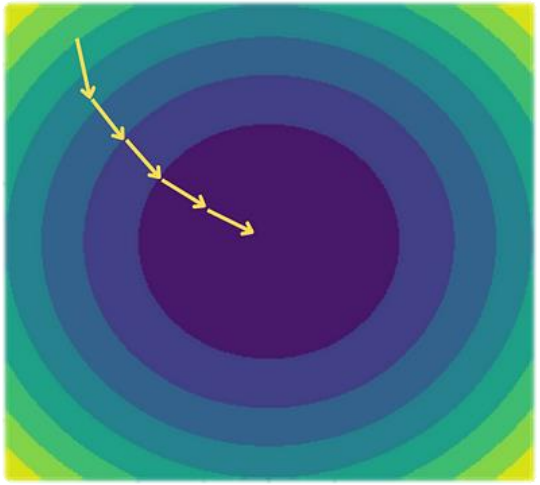
Gradient descent

- Gradient descent:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

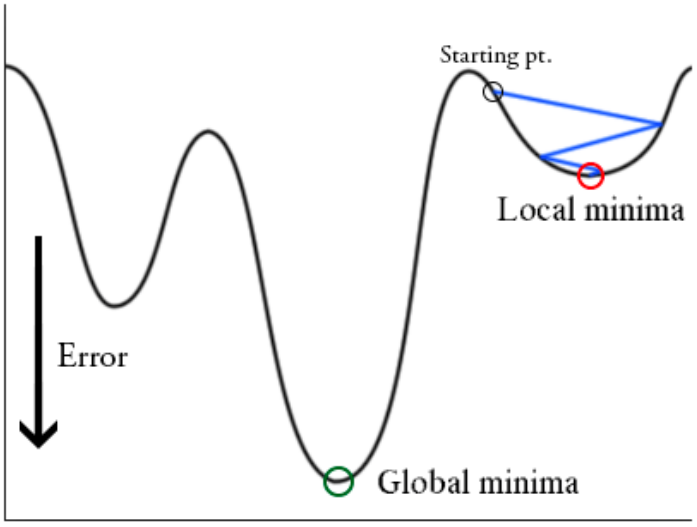
update is simultaneously performed for all values of $j = 0, \ldots, d$





Gradient descent

- Gradient descent is susceptible to local minima in general



- For a quadratic cost function, which is convex, gradient descent always converges to the global minimum (assuming the learning rate is not too large)

Gradient descent

- Batch gradient

$$\theta \leftarrow \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right) x^{(i)}$$

- Stochastic gradient

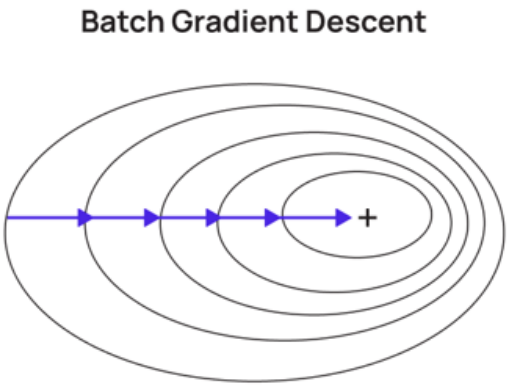
for i = 1 to n, {

$$\theta \leftarrow \theta + \alpha \left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right) x^{(i)}$$

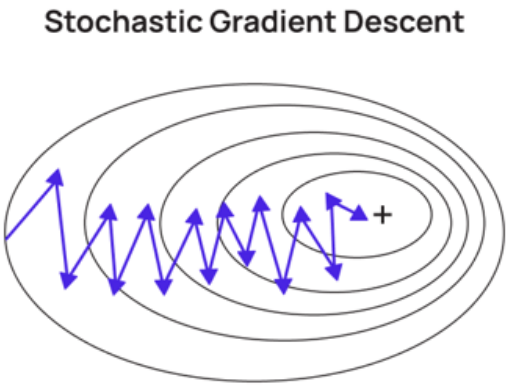
}

- Whereas batch gradient descent has to scan through the entire training set, stochastic gradient descent can start making progress right away
  - However, SGD may never converge to the minimum, and the parameters θ will keep oscillating around the minimum of $J(\theta)$

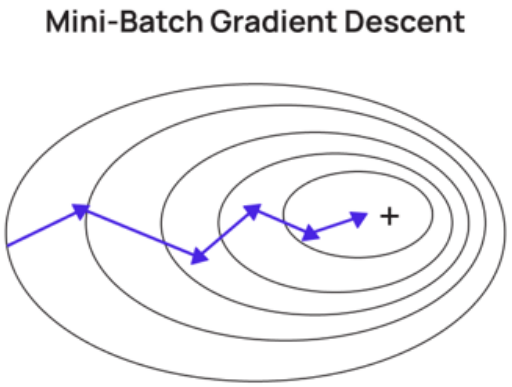
Gradient descent



All training samples



one training sample



part of training sample

Code

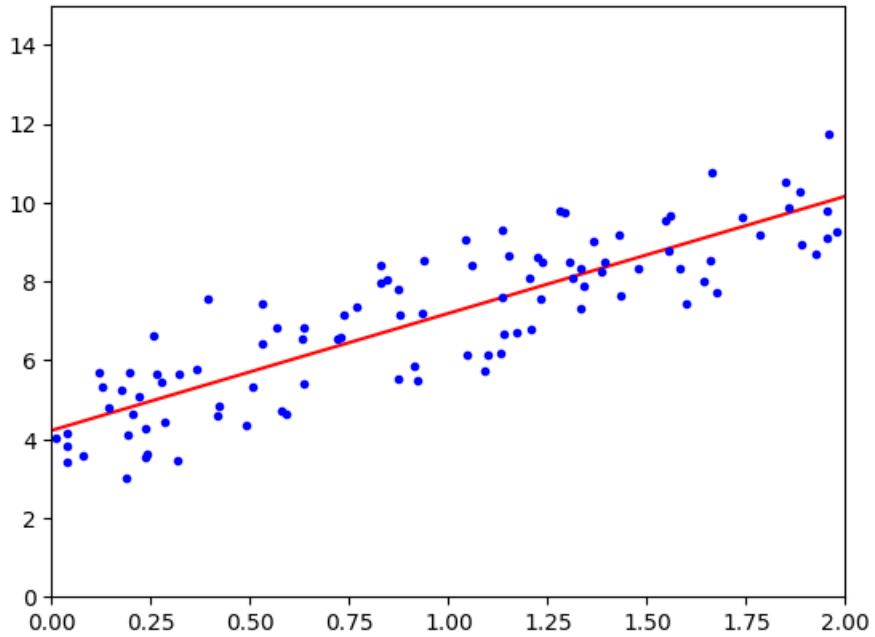
> Dataset generation

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

X = 2*np.random.rand(100,1)
y = 4 + 3*X+np.random.randn(100,1)
```

Code

- Normal equation

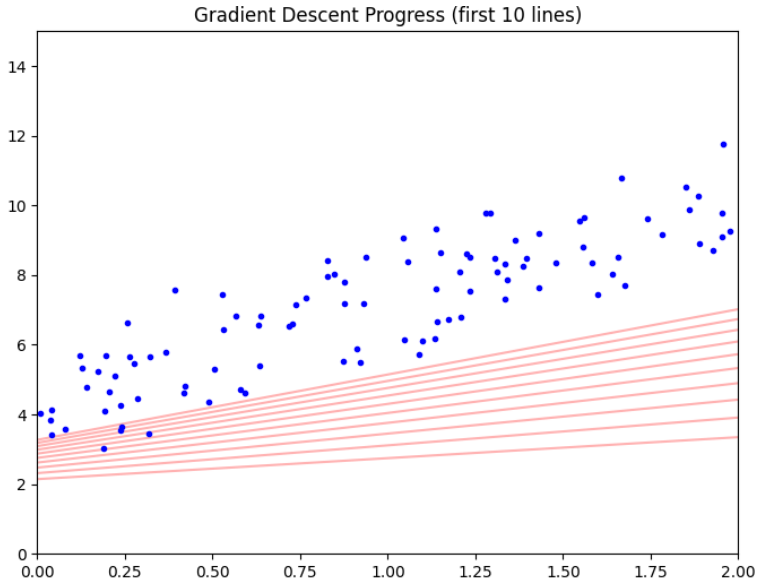
```
X_b = np.c_[np.ones((100,1)), X]
theta_best = np.linalg.inv(X_b.T@(X_b))@(X_b.T)@(y)

X_new = np.array([[0], [2]])
X_new_b = np.hstack((np.ones((2,1)), X_new))
y_predict = X_new_b.dot(theta_best)
```



Code

> Gradient descent

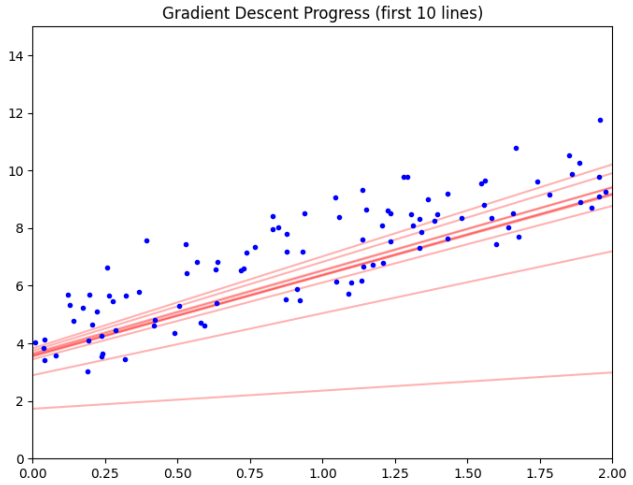
```
alpha = 0.02 # learning rate # test 0.02, 0.1, 0.4
n_iterations = 1000
m = 100
theta= np.random.randn(2,1)
for iteration in range(n_iterations):
  gradients = 1/m * X_b.T@(X_b@theta-y)
  theta = theta - alpha * gradients
```



Code

> Stochastic gradient descent

```
alpha = 0.02; n_epochs = 50
theta = np.random.randn(2,1) # random initialization
for epoch in range(n_epochs):
  for i in range(m):
    random_index = np.random.randint(m)
    xi = X_b[random_index:random_index+1]
    yi = y[random_index:random_index+1]
    gradients = 2*xi.T@(xi@theta -yi)
    theta = theta - alpha * gradients
```



Probabilistic interpretation

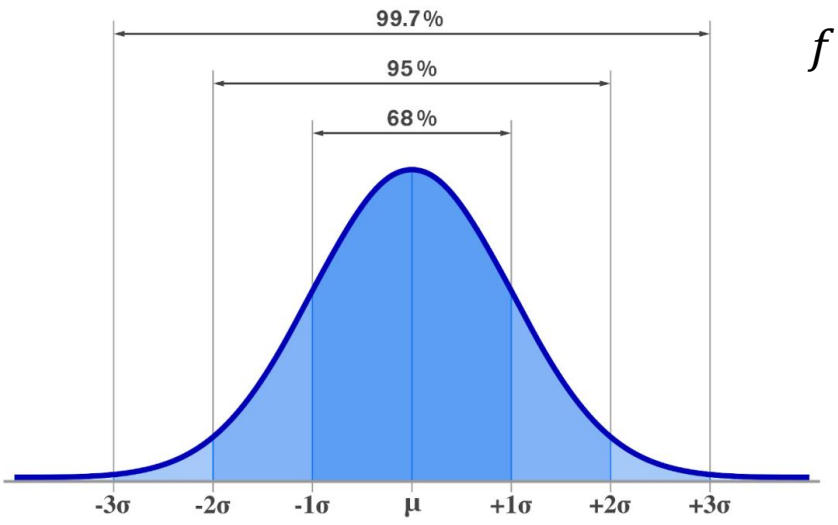
- Why linear regression with the least-squares cost function?

- In perspective of probability, assume that
  - $y^{(i)} = \theta^{\top} x^{(i)} + \epsilon^{(i)}$
  - where ϵ is an error that captures unmodeled effects or random noise
  - we assume that ϵ is i.i.d. Gaussian
  - $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$

- Gaussian distribution and its density function
  - $p(\epsilon^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(\epsilon^{(i)})^2}{2\sigma^2})$

Probabilistic interpretation

> Gaussian (Normal) distribution



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

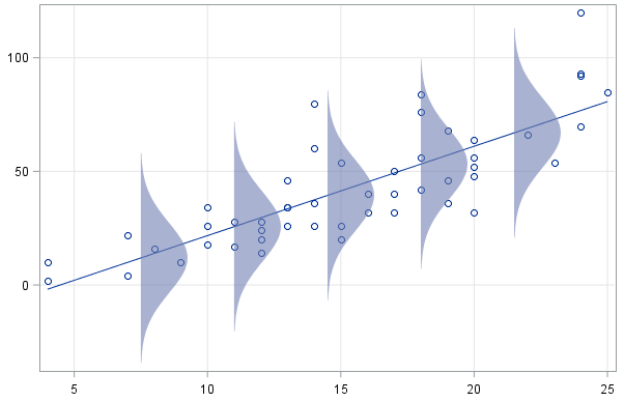
$$p(D|\theta) \propto p(\theta|D)p(D)$$

posterior likelihood prior

$$f(\mu|x = a) = \mathcal{N}(a, \sigma^2)$$

- Why Gaussian?
  - Central limit theorem
  - tractability (linear combination, conditional distribution, marginal distribution)
  - maximum entropy

Probabilistic interpretation



- $p\left(\epsilon^{(i)}\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\left(\epsilon^{(i)}\right)^2}{2\sigma^2}\right)$
- $p\left(y^{(i)}|x^{(i)};\theta\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\left(y^{(i)}-\theta^\top x^{(i)}\right)^2}{2\sigma^2}\right)$
  - distribution of $y^{(i)}$ given $x^{(i)}$ and θ.
  - we call it likelihood $L(\theta) = L(\theta; X, y) = p(y|X;\theta)$

- By the assumption of independence of ϵ, $L(\theta) = \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta)$
- Given this probabilistic model, what is a reasonable way of choosing θ?
- The principal of **maximum likelihood** says that we should choose θ so as to make the data as high probability as possible (i.e., we should maximize $L(\theta)$)

Probabilistic interpretation

- Instead of maximizing $L(\theta)$, we maximize $\log(L(\theta))$ – log likelihood
  - why? because it make the problem simple

- $$\begin{aligned} l(\theta) &= \log\big(L(\theta)\big) \\ &= \log \textstyle\prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) \\ &= \log \textstyle\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\left(y^{(i)}-\theta^{\top}x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= \log \textstyle\sum_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\left(y^{(i)}-\theta^{\top}x^{(i)}\right)^2}{2\sigma^2}\right) \\ &= \underbrace{n \log \frac{1}{\sigma\sqrt{2\pi}}}_{\text{constant}} - \underbrace{\frac{1}{2\sigma^2} \textstyle\sum_{i=1}^{n} \left(y^{(i)} - \theta^{\top}x^{(i)}\right)^2}_{\text{Least squares cost function}} \end{aligned}$$

- Therefore, maximizing $l(\theta)$ is equivalent to minimizing LS cost function

Logistic regression

- Regression can be used for classification
  - binary classification using logistic regression
  - use linear regression algorithm to predict y given x
  - It will generate values larger than 1 or smaller than 0 while $y \in \{0,1\}$

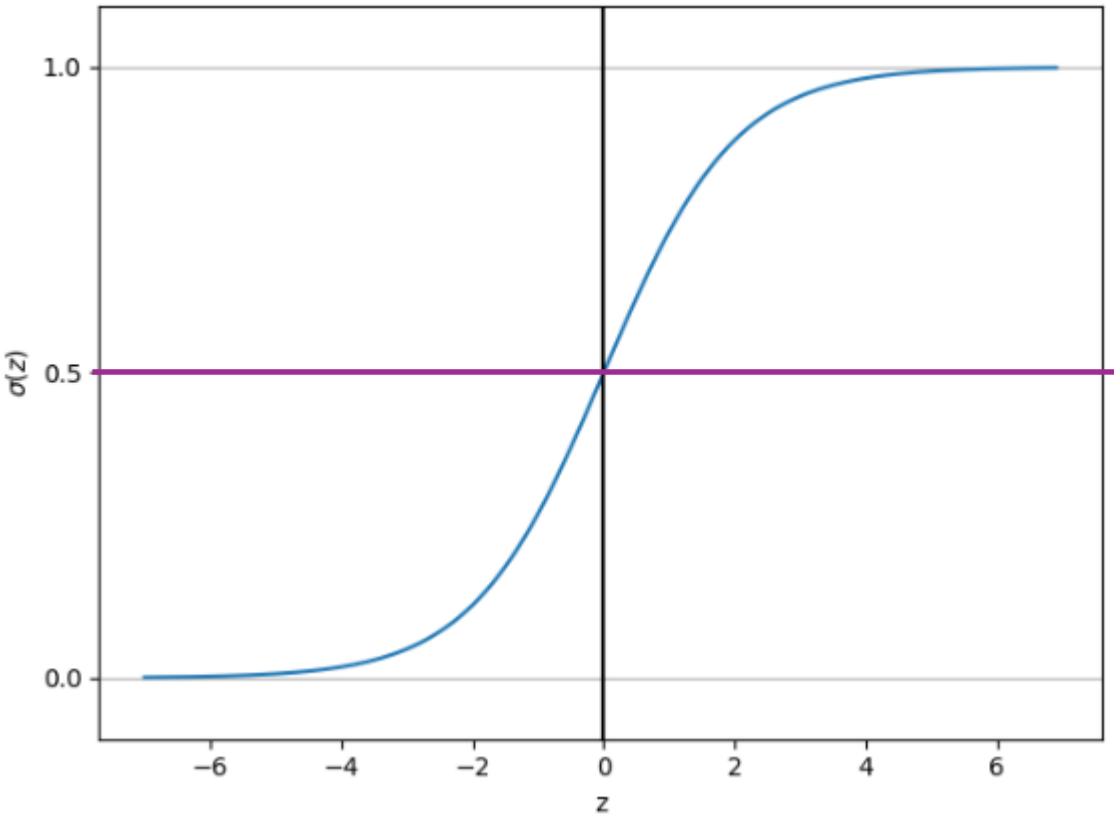
  - To fix this, we change the form for our hypotheses

$$h_\theta(x) = g(\theta^\top x) = \frac{1}{1 + e^{-\theta^\top x}}$$

  which is called the logistic function or the sigmoid function

Logistic regression

- Regression can be used for classification
  - binary classification using logistic regression



$$\frac{1}{1+e^{-z}}$$

positive class

threshold

negative class

Logistic regression

- Logistic function $g(z) = \frac{1}{1+e^{-z}}$
  - $g(z) \to 1 \;\; as \;\; z \to \infty$
  - $g(z) \to 0 \;\; as \;\; z \to -\infty$

- Other functions that smoothly increase from 0 to 1 can also be used, but the choice of the logistic function is fairly nature

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

$$= \frac{1}{(1+e^{-z})^2}(e^{-z})$$

$$= \frac{1}{(1+e^{-z})^2}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)g(1-z)$$

Logistic regression

- Likewise, fit θ for the logistic regression model
  - $p(y = 1|x; \theta) = h_\theta(x)$
    $p(y = 0|x; \theta) = 1 - h_\theta(x)$
  - $p(y|x; \theta) = \big(h_\theta(x)\big)^y \big(1 - h_\theta(x)\big)^{1-y}$
- Assume that the n training examples were generated independently,
  - $L(\theta) = p(y|X; \theta)$
    $= \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}; \theta)$
    $= \prod_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$
  - $l(\theta) = \log(L(\theta)) = \sum_{i=1}^{n} y^{(i)} \log h\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - h\left(x^{(i)}\right))$

Logistic regression

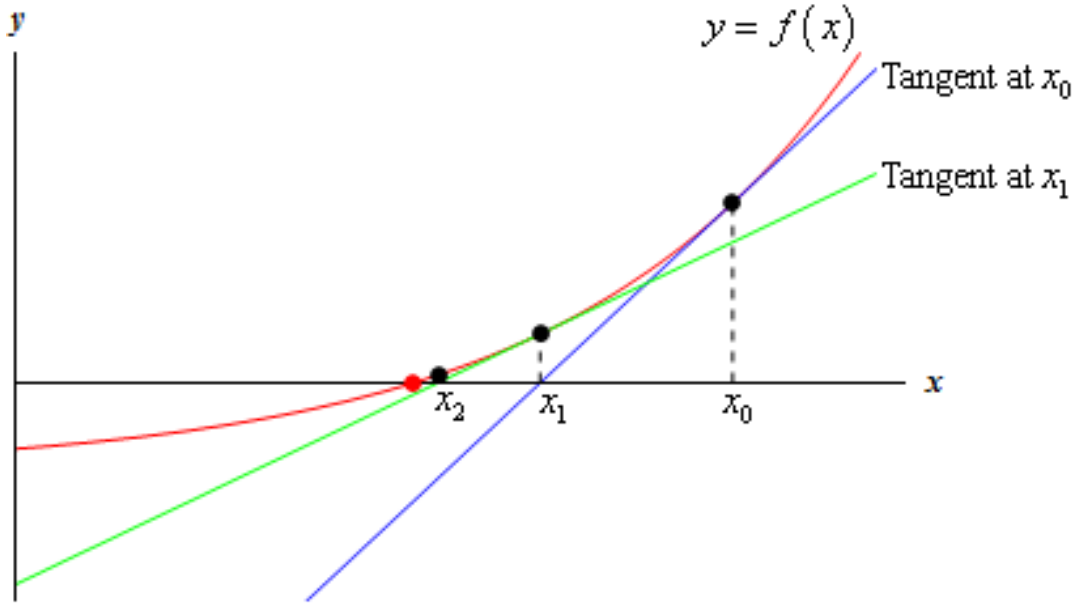
- To do gradient ascent, we take the derivative of $l(\theta)$
  - $$\begin{aligned}\frac{\partial}{\partial\theta_j}l(\theta) &= \left(y\frac{1}{g(\theta^\top x)} - (1-y)\frac{1}{1-g(\theta^\top x)}\right)\frac{\partial}{\partial\theta_j}g(\theta^\top x)\\ &= \left(y\frac{1}{g(\theta^\top x)} - (1-y)\frac{1}{1-g(\theta^\top x)}\right)g(\theta^\top x)\left(1-g(\theta^\top x)\right)\frac{\partial}{\partial\theta_j}\theta^\top x\\ &= \left(y\left(1-g(\theta^\top x)\right) - (1-y)g(\theta^\top x)\right)x_j\\ &= \left(y-h_\theta(x)\right)x_j\end{aligned}$$

  - $\theta_j \leftarrow \theta_j + \alpha\nabla_\theta l(\theta)$
    $\theta_j \leftarrow \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}$

  - If we compare this to the LMS update rule, it looks identical;
    but this is not the same algorithm because h_θ is a nonlinear function

Another method: Newton's method

- Our objective: find θ that maximizes $l(\theta)$
- Newton's method: finding a zero of a function (numerical optimization)
  - To find $f(x) = 0$, $x \leftarrow x - \frac{f(x)}{f'(x)}$
  - Approximating the function f via a linear function



Another method: Newton's method

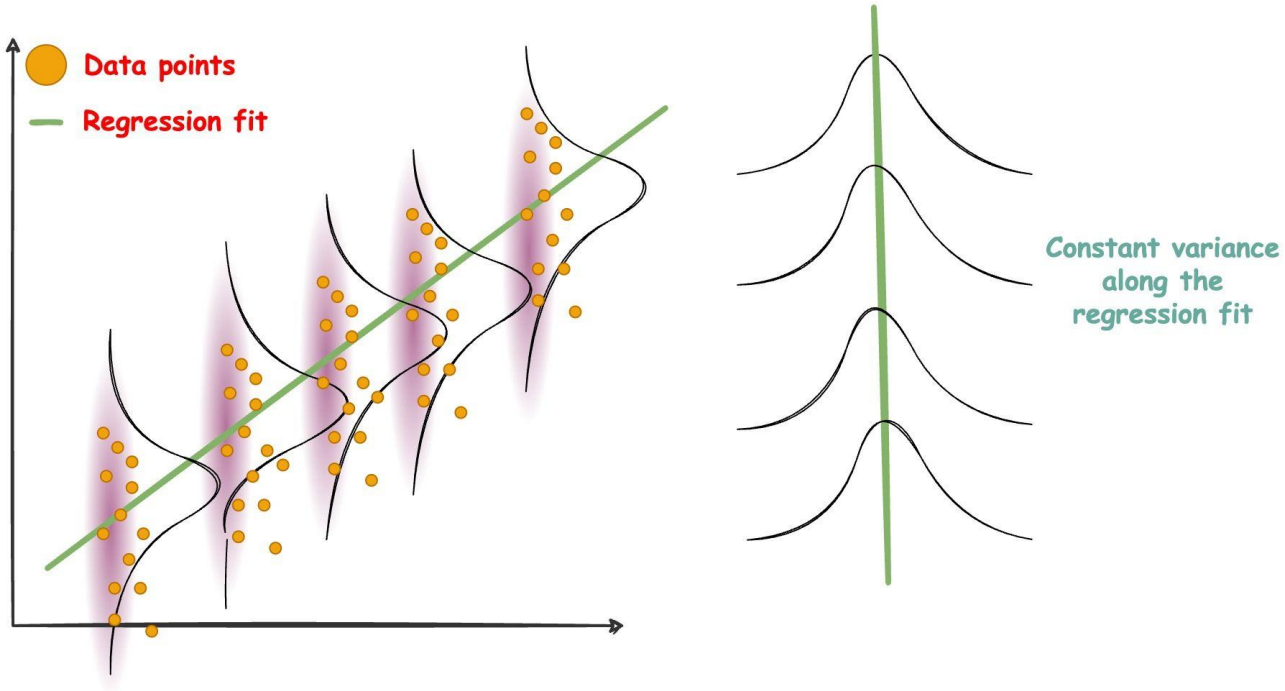
- Our objective: find θ that maximizes $l(\theta)$
  - The maxima of l corresponds to points where its first derivate $l'(\theta)$ is 0
  - To find $l'(\theta) = 0$, $\theta \leftarrow \theta - \frac{l'(x)}{l''(x)}$
  - In multi-dimensional setting $\theta \leftarrow \theta - H^{-1}\nabla_\theta l(\theta)$,

    where H is hessian, $H = \frac{\partial^2 l(\theta)}{\partial\theta_i \partial\theta_j}$

- Newton's method typically converges faster than gradient descent
  - however, one iteration is more expensive
  - still, it is usually much faster overall
  - When Newton's method is applied to maximize the logistic regression log likelihood function $l(\theta)$, the resulting method is called Fisher scoring

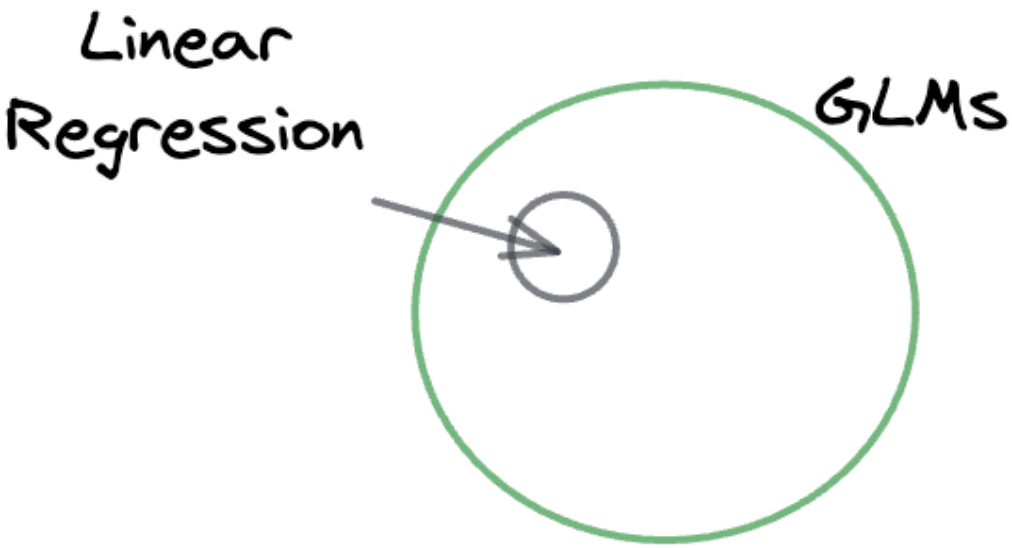
Generalized linear models

- Linear regression $p(y|X) \sim \mathcal{N}(\theta^\top X, \sigma^2)$
  - variance is constant
  - mean is a linear combination of features
  - distribution is Gaussian



Generalized linear models

- $p(y|X) \sim \mathcal{N}(\theta^\top X, \sigma^2)$
  - What if the distribution is not normal, but some other distribution?
  - What if X has a more sophisticated relationship with the mean?
  - What if the variance varies with X?



Generalized linear models

- $p(y|X) \sim \mathcal{N}(\theta^\top X, \sigma^2)$
  - What if the distribution is not normal, but some other distribution?
    - change the normal distribution to some other distribution from the exponential family of distributions

Exponential Family of Distributions

Gaussian	$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\| x-\mu \|^2/(2\sigma^2)}$	$x \in \mathbb{R}$
Bernoulli	$p(x) = \alpha^x (1-\alpha)^{1-x}$	$x \in \{0,1\}$
Binomial	$p(x) = \binom{n}{x} \alpha^x (1-\alpha)^{n-x}$	$x \in \{0,1,2,\ldots,n\}$
Multinomial	$p(x) = \frac{n!}{x_1!x_2!\ldots x_n!} \prod_{i=1}^{n} \alpha_i^{x_i}$	$x_i \in \{0,1,2,\ldots,n\},\ \sum_i x_i = n$
Exponential	$p(x) = \lambda\, e^{-\lambda x}$	$x \in \mathbb{R}^+$
Poisson	$p(x) = \frac{e^{-\lambda}}{x!}\, \lambda^x$	$x \in \{0,1,2,\ldots\}$

Generalized linear models

- $p(y|X) \sim \mathcal{N}(\theta^\top X, \sigma^2)$
  - What if X has a more sophisticated relationship with the mean?
    - $\mu(x) = \sum_{j=1}^{d} \theta_d x_d \rightarrow g\big(\mu(x)\big) = \sum_{j=1}^{d} \theta_d x_d$
    - g is called the link function (nonlinear)

Generalized linear models

- $p(y|X) \sim \mathcal{N}(\theta^\top X, \sigma^2)$
  - What if the variance varies with X?

    - variance depends on the mean
      $var(y) = \phi V(\mu)$ where ϕ is a constant

    - variance function is determined by the choice of distribution
      - Gaussian: $V(\mu) = 1$
      - Binomial (logistic regression): $V(\mu) = \mu(1-\mu)$
      - Poisson: $V(\mu) = \mu$

Generalized linear models

- Exponential family
  - it includes many of the most common distributions (normal, gamma, beta, Bernoulli, Poisson, exponential, chi-squared, Dirichlet, categorical, …)
  - $p(y;\eta) = b(y)\exp\left(\eta^{\top}T(y) - a(\eta)\right)$
  - $\eta = \theta^{\top}x$

  - η is a natural parameter (canonical parameter)
  - $T(y)$ is the sufficient statistics (in many times, $T(y) = y$)
  - $a(\eta)$ is the log partition function (essentially, normalization constant)
  - $\mathrm{b}(\mathrm{y})$ is the base measure

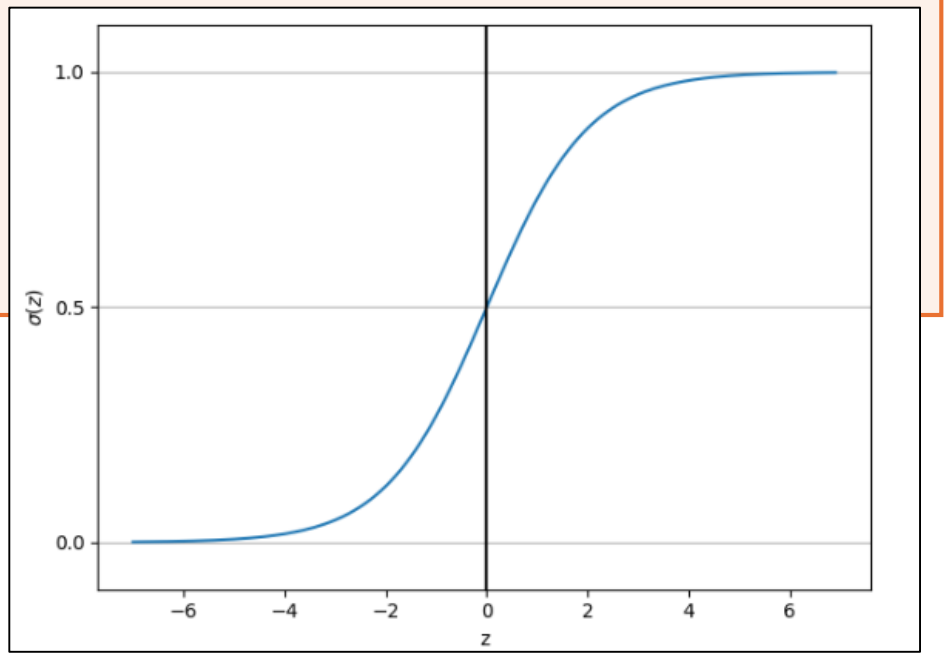
Generalized linear models

- Exponential family
  - log-likelihood: $l(\theta) = \log(\prod_{i=1}^{n} b(y) \exp(\eta^\top T(y) - a(\eta)))$
    $= \sum_{i=1}^{n} \log(b(y) \exp(\eta^\top T(y) - a(\eta)))$
  - log-likelihood gradient: $\frac{\partial}{\partial \theta_j} l(\theta) = \frac{\partial}{\partial \theta_j} \sum_{i=1}^{n} \eta^\top T(y) - a(\eta)$
    $= \sum_{i=1}^{n} T(y) \frac{d\eta}{d\theta} - a'(\eta) \frac{d\eta}{d\theta}$
    $= \sum_{i=1}^{n} \underbrace{(T(y) - a'(\eta))}_{\text{residual}} \underbrace{x}_{\text{input}}$

Code

> Sigmoid function

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def sigmoid(z):
  return 1.0 / (1.0 + np.exp(-z))

z = np.arange(-7, 7, 0.1)
sigma_z = sigmoid(z)
plt.plot(z, sigma_z)
plt.show()
```



Code

> Logistic regression -1

```
class LogisticRegressionGD:
  def __init__(self, alpha = 0.01, n_iter = 50, random_state = 1):
    self.alpha = alpha
    self.n_iter = n_iter
    self.random_state = random_state

  def net_input(self, X):
    return X@self.w_ + self.b_

  def activation(self, z):
    return 1./(1. + np.exp(-np.clip(z, -250, 250)))

  def predict(self, X):
    return np.where(self.activation(self.net_input(X)) >= 0.5, 1, 0)
```

Code

> Logistic regression -2

```
def fit(self, X, y):
  rgen = np.random.RandomState(self.random_state)
  self.w_= rgen.normal(loc=0.0, scale = 0.01, size = X.shape[1])
  self.b_ = np.float64(0.)
  self.losses_ = []
  for i in range(self.n_iter):
    net_input = self.net_input(X)
    output = self.activation(net_input)
    errors = (y-output)
    self.w_ += self.alpha *2.0 * X.T@errors / X.shape[0]
    self.b_ += self.alpha *2.0 *errors.mean()
    loss = (-y@np.log(output))- ((1-y)@np.log(1-output))/X.shape[0]
    self.losses_.append(loss)
  return self
```

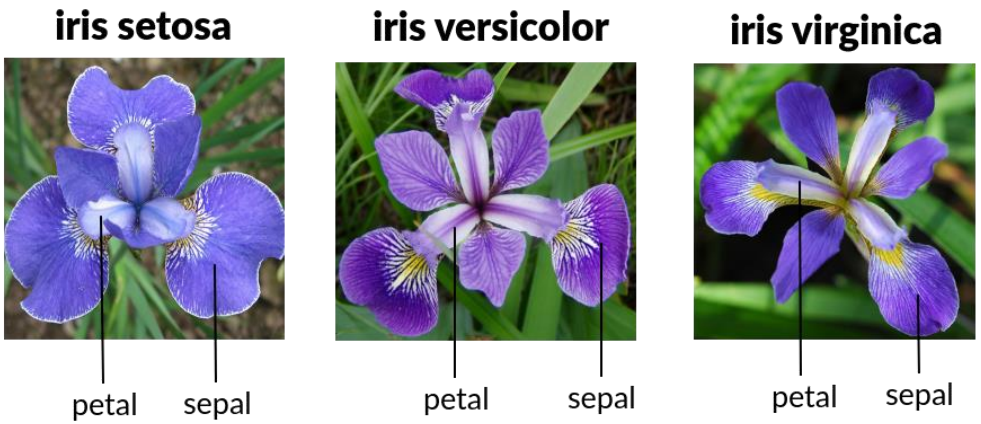
Code

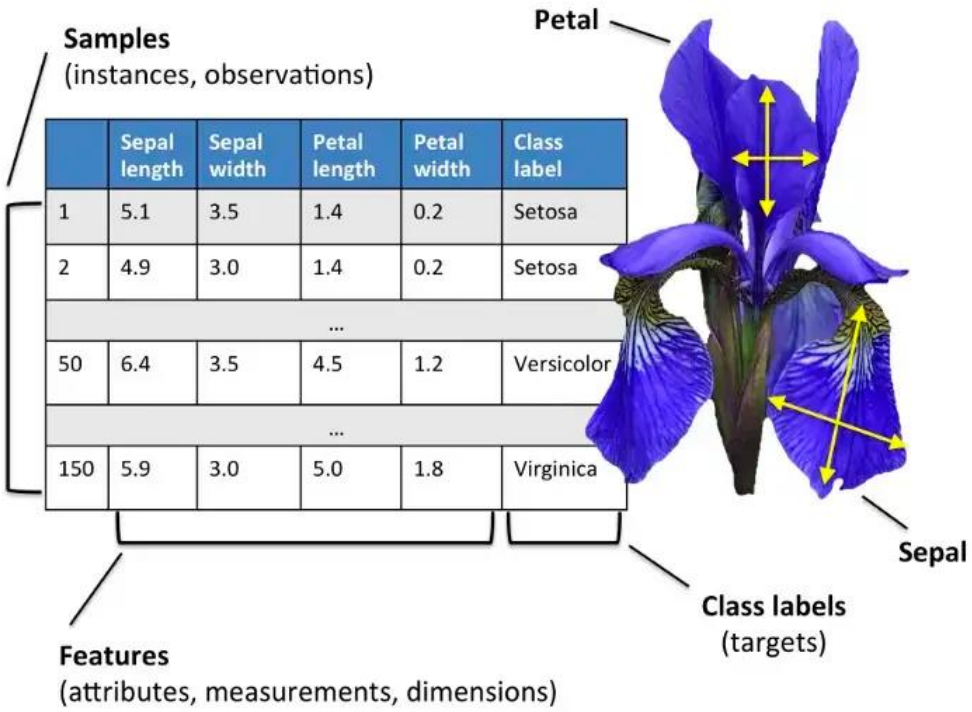
- Load iris dataset

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()
X = iris.data[:, [2,3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state = 1, stratify = y)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Code

- Load iris dataset





	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

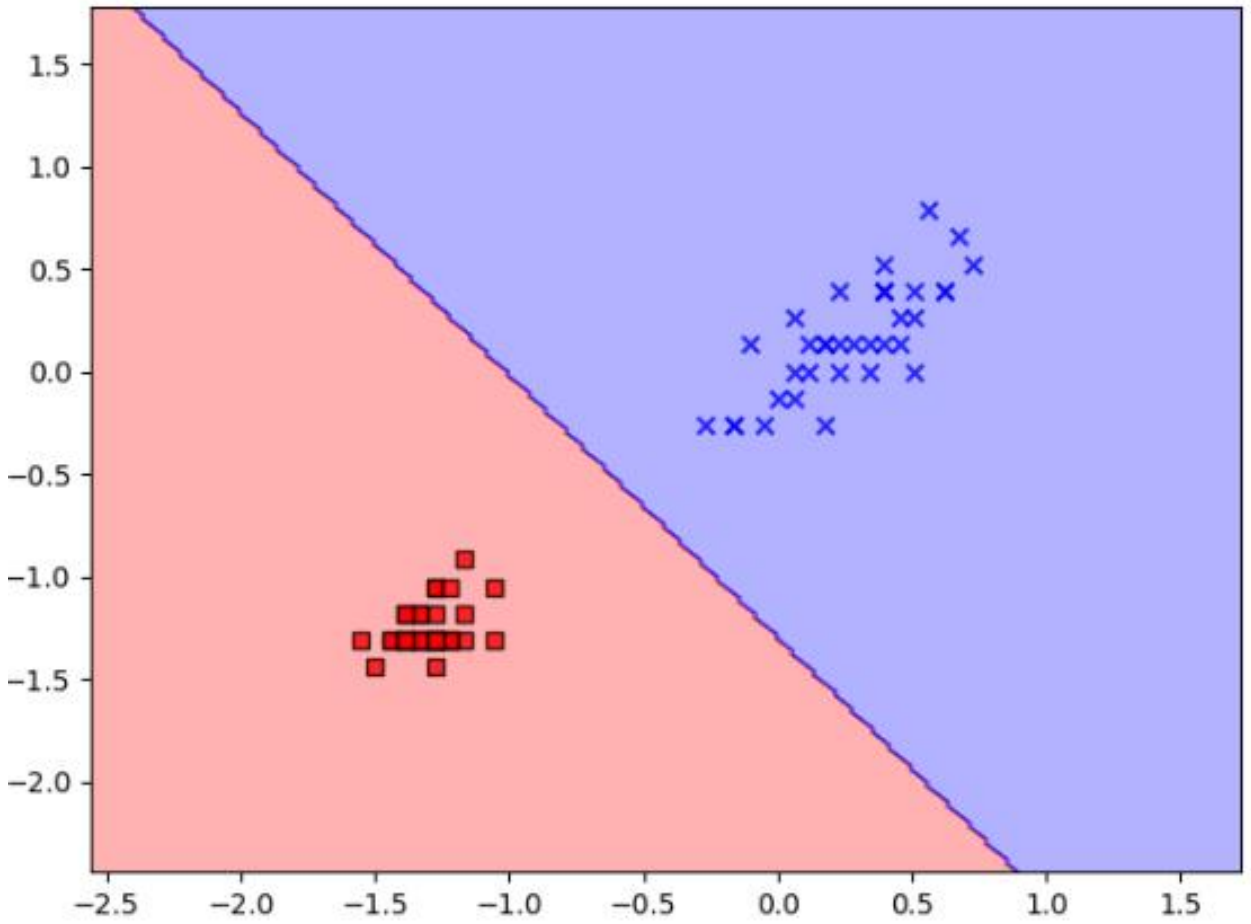
Code

- Perform the logistic regression

```
X_train_01_subset = X_train_std[(y_train == 0) | (y_train==1)]
y_train_01_subset = y_train[(y_train == 0) | (y_train == 1)]
lrgd = LogisticRegressionGD(alpha = 0.3, n_iter = 1000, random_state = 1)
lrgd.fit(X_train_01_subset, y_train_01_subset)
plot_decision_regions(X = X_train_01_subset, y = y_train_01_subset,
classifier = lrgd)
plt.tight_layout()
plt.show()
```

Code

> Perform the logistic regression



Reference

- CS229 Lecture Notes, Stanford
- Generalized linear model
  - https://www.dailydoseofds.com/generalized-linear-models-glms-the-supercharged-linear-regression/
  - Heather Turner lecture note, University of Warwick
- Linear regression code: 핸즈온 머신러닝 Ch.4
- logistic regression code: 머신러닝 교과서 파이토치편 Ch.3

- Further reading
  - https://bookdown.org/roback/bookdown-BeyondMLR/ch-poissonreg.html