Machine Learning

1. Introduction

Definition: If a machine's performance,denoted by P, on the task T improves with experience E, then the machine is said to learn from E with respect to T and P.

Taxonomy of ML:

data type	learning type
1. Supervised Learning 2. Unsupervised Learning 3. Reinforcement Learning 4. Semi-supervised Learning	1. Offline Learning 2. Online Learning

Supervised Learning: It could help us with two tasks in general. The first if regression and the second is classification. In regression, the output is a continuous value, while in classification, the output is a discrete value.

Unsupervised Learning: It is used to find the hidden patterns in the data. It is used in clustering and association.

Reinforcement Learning: The agent learns from the feedback it receives from the environment after each action it takes.

2. Maximum Likelihood Estimation

For an example of multiple cases from the same probability distribution, parameterized by θ, there exists a θ that maximized the likelihood function of the joint distribution of all the cases, written:

$$\begin{aligned}\hat{\underset{\text{ML}}{\theta}} &= \underset{\theta\in(0,1)}{\operatorname{argmax}}\ p_{X_1,\ldots,X_m}(X_1, X_2, \ldots, X_m; \theta) \\ &= \underset{\theta\in(0,1)}{\operatorname{argmax}}\ \log p_{X_1,\ldots,X_m}(X_1, X_2, \ldots, X_m; \theta)\end{aligned}$$

Property:

- Consistent: As the number of samples increases, the estimated parameter converges to the true parameter.
- Asymptotically Normal: The estimated parameter is normally distributed as the number of samples increases, and the variance of the distribution decreases as the number of samples increases.

3. Linear Regression(With offset)

We define:

- m: number of training examples
- d: number of features
- $\boldsymbol{X} \in \mathbb{R}^{m\times(d+1)}$: input matrix, put all the data vector into a matrix
- $\boldsymbol{y} \in \mathbb{R}^{m}$: target vector, each entry corresponds to the output of one input vector

- $\boldsymbol{w} \in \mathbb{R}^{d+1}$: weight vector
- $\boldsymbol{b} \in \mathbb{R}$: offset number

3.1. Task

Train the model(defined by the parameter, here is a vector) to predict the output vector $\boldsymbol{y}$ given a new input vector $\boldsymbol{x}$.

3.2. Procedure

Here we just use the offset version

1. We simply want to find the $\boldsymbol{w}$ that satisfied: $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$ In this step, we may decide whether the linear system has a solution or not. If not, we may use the least square solution.
2. Define notation $f_{\boldsymbol{w},b}(\boldsymbol{x}_i) = \boldsymbol{w}^T\boldsymbol{x} + b$, and define the loss for each example

$$e_i = f_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i$$

   Then the loss function is $L(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m} e_i^2 = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i\right)^2$
3. The least square solution that minimize the loss function is $\boldsymbol{w} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$, (without offset, just use the original input matrix $\boldsymbol{X}$). Be careful whether the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible or not, i.e., $\boldsymbol{X}$ is full column rank.
4. Then we could predict new input $\boldsymbol{y}_{\mathbf{new}} = {\boldsymbol{x}_{\mathbf{new}}}^T\boldsymbol{w}$.

4. Linear Regression with Multiple Outputs

We define:

- m: number of training examples
- d: number of features
- h: number of outputs features
- $\boldsymbol{X} \in \mathbb{R}^{m\times(d+1)}$: input matrix, $\boldsymbol{X} = [\mathbf{1} \;\; \boldsymbol{X}']$
- $\boldsymbol{Y} \in \mathbb{R}^{m\times h}$: output matrix
- $\boldsymbol{W} \in \mathbb{R}^{(d+1)\times h}$: design matrix, $\boldsymbol{W} = \begin{bmatrix}\boldsymbol{b}^T \\ \boldsymbol{W}'\end{bmatrix}$
- $\boldsymbol{b} \in \mathbb{R}^h$: offset vector, we ignore it and put it into the design matrix

4.1. Task

Train the model(defined by the parameter, here is a matrix) to predict the output vector $\boldsymbol{y}$ given a new input vector $\boldsymbol{x}$.

4.2. Procedure

The procedure is similar to the single output case. For the first step, we could devide the output matrix into h single output vectors, try to solve the linear system for each output vector. If the matrix $\boldsymbol{X}$ is full column rank, we may get the least square solution.

1. Define loss function: $\text{Loss}(\boldsymbol{W}) = \text{Loss}(\boldsymbol{W}, \boldsymbol{b}) = \sum_{k=1}^{h}\left(\boldsymbol{X}\boldsymbol{w}_k - \boldsymbol{y}^k\right)^T\left(\boldsymbol{X}\boldsymbol{w}_k - \boldsymbol{y}^k\right)$
2. The least square solution is $\boldsymbol{W} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{Y}$, where $\boldsymbol{X} = [\mathbf{1} \;\; \boldsymbol{X}']$.

4.3. MLE and Linear Regression

We have $y_i = \boldsymbol{w}^T\boldsymbol{x}_i + e_i$ (b is in the vector $\boldsymbol{w}$). The error term e_i is of gaussian distribution and therefore $y_i|\boldsymbol{x}_i; \boldsymbol{w}, \sigma^2 \sim N(\boldsymbol{w}^T\boldsymbol{x}_i, \sigma^2)$, where σ^2 is the variance of the error term. We could write the likelihood function as:

$$L(\boldsymbol{w}, \sigma^2 \mid \{y_i, \boldsymbol{x}_i\}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2}{2\sigma^2}\right)$$

Solve the MLE problem

1. $\log(L(\boldsymbol{w}, \sigma^2)|\{y_i, \boldsymbol{x}_i\}) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2$
2. use the derivative to find the maximum likelihood estimator:

$$\frac{\sigma}{\sigma \boldsymbol{w}} \log(L(\boldsymbol{w}, \sigma^2)|\{y_i, \boldsymbol{x}_i\}) = \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)\boldsymbol{x}_i = 0$$

3. we could get the least square solution: $\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$. In fact, the least square solution is the MLE solution. (The decution process is ignored here)

5. Linear Classification

Main idea: To treat binary classification as linear regression in which the output y_i is binary. $y_i \in \{-1, +1\}$.

- Learning and training part: Similar procedure, obtain the weight vector $\boldsymbol{w}$.
- Prediction part: $y_{\text{new}} = \text{sign}(\boldsymbol{x}_{\text{new}}^T \boldsymbol{w}) = \text{sign}\left(\begin{bmatrix}1\\ \boldsymbol{x}'_{\text{new}}\end{bmatrix}^T \boldsymbol{w}\right) \in \{-1, +1\}$

The sign function is defined as:

$$\text{sign}(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ -1 \text{ otherwise} \end{cases}$$

5.1. Python Demo

```python
import numpy as np
from numpy.linalg import inv

X = np.array([[1,-7],[1,-5],[1,1],[1,5]])
y = np.array([[-1],[-1],[1],[1]])
#linear regression for classification
w = inv(X.T @ (X)) @ (X.T) @ (y)
print(w,"\n")

#predict
X_new = np.array([[1,2]])
y_predict_new = np.sign(X_new @ w)
print(y_predict_new)
# expected output: [[-1.]]
```

5.2. Multi-class Classification

Idea: one-hot encoding, for classes $\{1, 2, ..., C\}$, where $C > 2$ is the number of classes. The correspoding label vector is

$$\begin{aligned}\boldsymbol{y}_{c1} &= [1\ \ 0\ \ 0\ \ \ldots\ \ 0]\\ \boldsymbol{y}_{c2} &= [0\ \ 1\ \ 0\ \ \ldots\ \ 0]\\ &\vdots\\ \boldsymbol{y}_{cC} &= [0\ \ 0\ \ 0\ \ \ldots\ \ 1]\end{aligned}$$

We store the class vectors of datasets into a label matrix $\boldsymbol{Y} \in \mathbb{R}^{m\times C}$, where m is the number of training examples. It is a binary matrix. Essentially we are doing C separate linear classification problems with class k and other classes as a single class.

- Learning and training part: Similar procedure, obtain the weight matrix $\boldsymbol{W} \in \mathbb{R}^{(d+1)\times C}$.
- Prediction part:

$$\boldsymbol{y}_{\text{new}} = \underset{k\in\{1,2,\ldots,C\}}{\operatorname{argmax}} \left(\boldsymbol{x}_{\text{new}}^T \boldsymbol{W}[:,\boldsymbol{k}]\right)$$

  in which the $\boldsymbol{W}[:,k] \in \mathbb{R}^{d+1}$ is the k-th column of the weight matrix.

5.2.1. Python Demo

```python
import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import OneHotEncoder
# manually encode the label matrix
X = np.array([[1,1,1],[1,-1,1],[1,1,3],[1,1,0]])
Y_class = np.array([[1],[2],[1],[3]])
Y = np.array([[1,0,0],[0,1,0],[1,0,0],[0,0,1]])

# one-hot encoding function
Y_onehot = OneHotEncoder().fit_transform(Y_class).toarray()

# learning
W = inv(X.T @ X) @ X.T @ Y # could use Y_onehot instead of Y

# predicting
X_new = np.array([[1,0,-1]])
Y_predict = X_new @ W
print(np.argmax(Y_predict)+1)
```

6. Polynomial Regression

6.1. Motivation

- The data may come from a polynomial function, but the linear model may not be able to fit the data well.

- For classification, the $\oplus$ dataset is not linearly separable, and polynomial function could separate the data.

6.2. Idea

- We need some knowledge about polynomial functions.
  1. Let $f_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + ... + w_d x_d$, then it has 1 variable and d degree. Let $f_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$, then it has 2 variables and 2 degree. each term is called monomial.
  2. For a polynomial function with d degree and n variables, the number of monomials is $\binom{d+n}{n}$

Training

- For m data, n order and d variables, the polynomial matrix is

$$P = \begin{bmatrix} \boldsymbol{p}_1^T \\ \boldsymbol{p}_2^T \\ \vdots \\ \boldsymbol{p}_m^T \end{bmatrix} \in \mathbb{R}^{m \times \binom{d+n}{n}}$$

  and $\boldsymbol{p}_i = \begin{bmatrix} 1 & m_1 & m_2 & \dots & m_k \end{bmatrix}^T$, m_i is monomial and $k = \binom{d+n}{n}$
- weight vector $\boldsymbol{w} = P^T \left(PP^T\right)^{-1} \boldsymbol{y}$ (Dual form of the least square solution before)

Predicting

- $y_{\text{new}} = P_{\text{new}} \boldsymbol{w}$

6.3. Polynomial Classification

- Similar to the linear Classification
- For simple output, $y_{\text{new}} = \text{sign}(\boldsymbol{p}_{\text{new}}^T \boldsymbol{w})$
- For multiple outputs, $\boldsymbol{y}_{\text{new}} = \underset{k \in \{1,2,\dots,c\}}{\text{argmax}} \left(\boldsymbol{p}_{\text{new}}^T \boldsymbol{W}[:, \boldsymbol{k}]\right)$

7. Ridge Regression

7.1. Motivation

In real life, the features d is large and the number of examples m is small, so the matrix $\boldsymbol{X^T X} \in \mathbb{R}^{\boldsymbol{d \times d}}$ is hard to invert. So we need to stabilize and robustify the solution.

7.2. Idea

Recall the loss function of linear regression and introduce the regularization term(ridge regression version):

$$\begin{aligned} L(\boldsymbol{w}, b) &= \frac{1}{m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i\right)^2 = (\boldsymbol{Xw} - \boldsymbol{y})^T (\boldsymbol{Xw} - \boldsymbol{y}) \\ J(\boldsymbol{w}, b) &= L(\boldsymbol{w}, b) + \lambda \left\|\boldsymbol{w}\right\|_2^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i\right)^2 + \lambda \left\|\boldsymbol{w}\right\|_2^2 \\ &= (\boldsymbol{Xw} - \boldsymbol{y})^T (\boldsymbol{Xw} - \boldsymbol{y}) + \lambda \boldsymbol{w}^T \boldsymbol{w} \end{aligned}$$

where λ is the regularization parameter. We need to minimize the loss function, that is to find:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}((\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w}-\boldsymbol{y}) + \lambda \boldsymbol{w}^T\boldsymbol{w})$$

and the result is

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_{d+1})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

The term $\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}$ is always invertible because it is positive definite. The predicting part is the same as the linear regression.

To prove $\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}$ is invertible, we need to prove it is positive definite. By definition of positive definite, we need to prove: $\forall \boldsymbol{x}, \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leftrightarrow \boldsymbol{A}$ is positive definite. We have

$$\begin{aligned}\boldsymbol{x}^T(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I})\boldsymbol{x} &= \boldsymbol{x}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{x} + \boldsymbol{x}^T\lambda \boldsymbol{I}\boldsymbol{x} \\ &= (\boldsymbol{X}\boldsymbol{x})^T(\boldsymbol{X}\boldsymbol{x}) + \boldsymbol{x}^T\boldsymbol{x}\end{aligned}$$

Since every term is positive, then the result is larger or equal to zero. $\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}$ is positive definite and therefore always invertible.

However, this term is in $\mathbb{R}^{(d+1)\times(d+1)}$ so it is hard to find the inverse of it. So we could use the dual form of the solution which needs to find the inverse of a matrix in $\mathbb{R}^{m\times m}$.

$$\boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda \boldsymbol{I}_m)^{-1}\boldsymbol{y}$$

The proof of the dual form need to use Woodbury formula:

$$(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{V})^{-1} = \boldsymbol{I} - \boldsymbol{U}(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{V})^{-1}\boldsymbol{V}$$

7.3. Python Demo

Refer to DSAA-2011-Demo

8. Gradient Descent

8.1. Motivation

In linear regression and the other few models mentioned above, the optimal solution could be simply found by solving the equation. However, in many cases, to minimize the loss function with respect to parameter w is hard. So we want an algorithm that could iteratively find the optimal solution, that is the gradient descent algorithm.

8.2. Idea

- Task: Minimize the loss function $C(\boldsymbol{w})$ in which $\boldsymbol{w} = \begin{bmatrix} w_1 & \cdots & w_d \end{bmatrix}^T$
- Gradient of $\boldsymbol{w}$: $\nabla_{\boldsymbol{w}} C(\boldsymbol{w}) = \begin{bmatrix} \frac{\sigma C}{\sigma w_1} & \frac{\sigma C}{\sigma w_2} & \cdots & \frac{\sigma C}{\sigma w_d} \end{bmatrix}^T$. It's a function or say a vector of $\boldsymbol{w}$. And the direction of the gradient is the direction of the fastest **increase** of the function, while the opposite direction is the direction of the fastest **decrease** of the function.
- Algorithm
  1. Initial $\boldsymbol{w}$ with learning rate $\eta > 0$
  2. $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_k)$
  3. Repeat the above step until the convergence condition is satisfied.
- Convergence Criterias:

1. The absolute or percentage change of the loss function is smaller than a threshold.
2. The absolute or percentage change of parameter $\boldsymbol{w}$ is smaller than a threshold.
3. Set a maximum number of iterations.

- Notice that according to multivariate calculus, if η is not too large,$C(\boldsymbol{w}_{k+1}) < C(\boldsymbol{w}_k)$. But GD could only find the local minimum.

8.3. Variation of GD

8.3.1. Change the learning rate

1. Decreasing learning rate

$$\eta = \frac{\eta_0}{1+k}$$

   where k is the number of iterations or other form of changes. It could help the algorithm to converge faster at the beginning and avoid oscillation at the end.
2. Adaptive learning rate

$$\boldsymbol{w}_{k+1} = \boldsymbol{w_k} - \frac{\eta}{\sqrt{G_k + \varepsilon}} \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_k)$$

   where $G_k = \sum_{i=0}^{k} \left\| \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_i) \right\|_2^2$ and ε is a small positive number to avoid zero denominator. This method gives larger update to smaller gradient and vice versa , adjusting the learning rate according to the history information. But the learning rate may shirnk too fast.

8.3.2. Different gradient

1. Momentum-based GD

$$\begin{aligned} \boldsymbol{v}_k &= \beta \boldsymbol{v}_{k-1} + (1-\beta) \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_{k-1}) \\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \eta \boldsymbol{v}_k \end{aligned}$$

   where $\beta \in (0,1)$ is the momentum parameter and $\boldsymbol{v}_0 = \nabla_{\boldsymbol{w}} C(\boldsymbol{w}_0)$. It converges fast but may overshoot the optimal point.
2. Nesterov Accelerated Gradient (NAG)

$$\begin{aligned} \boldsymbol{v}_k &= \beta \boldsymbol{v}_{k-1} + \eta \nabla_{\boldsymbol{w}} C(\boldsymbol{w} + \beta \boldsymbol{w}_{k-1}) \\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \boldsymbol{v}_k \end{aligned}$$

   where $\boldsymbol{v}_0 = 0$.It works by anticipating the next direction of the optimizer. It is fast but complex.

8.3.3. Design of loss function

The idea is only calculate the loss of some sample from the dataset, reduce the computation cost.

1. Batch GD: Use all the data to calculate the gradient.
2. Stochastic GD: Use one randomly chosen data to calculate the gradient.
3. Mini-batch GD: Use a small batch of randomly chosen data to calculate the gradient.

9. Logistic Regression

9.1. Motivation

There are some possible issues for classification problems:

1. Noises: Lead to unseparable data.
2. Mediocre generalization: Only find barely boundary.
3. Overfitting: The model is too complex and fits the noise in the data.

In that case, we want a model that output our confidence of the prediction and that's why we need logistic regression.

9.2. Idea

- Logistic function:

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

- Logistic Regression: In binary classification task, we set our prediction

$$\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = g(\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0) = \frac{1}{1+e^{-(\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0)}}$$

  This is known as class conditional probability.

How could we derive the form of the class conditional probability? We first define the log-odds function of both classes as a affine function of the input vector $\boldsymbol{x}$:

$$\log\frac{\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0)}{\Pr(y=-1|\boldsymbol{x},\boldsymbol{\theta},\theta_0)} = \boldsymbol{\theta}\boldsymbol{x}+\theta_0$$

Set $\Pr(y=-1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = 1-\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0)$, then:

$$\log\frac{\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0)}{1-\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0)}$$

Let $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = a$, $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle = b$, then it is equivalent to:

$$\log\frac{a}{1-a} = b \Rightarrow \frac{a}{1-a} = e^b \Rightarrow a = (1-a)e^b \Rightarrow a = \frac{e^b}{1+e^b} \Rightarrow a = \frac{1}{1+e^{-b}}$$

So $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = \frac{1}{1+e^{-\langle\boldsymbol{\theta},\boldsymbol{x}\rangle}}$

- $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0 = 0$ is the decision boundary. If $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0 > 0$, then $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) > 0.5$ and vice versa.

10. Support Vector Machine

10.1. Get familiar with Logistic Regression

Logistic regression is a binary classification model.

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}$$

The loss function in logistic regression is

$$L(\boldsymbol{w}) = -\sum_{i=1}^{m} y_i \log(f_{\boldsymbol{w}}(\boldsymbol{x}_i)) + (1-y_i)\log(1-f_{\boldsymbol{w}}(\boldsymbol{x}_i)) + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

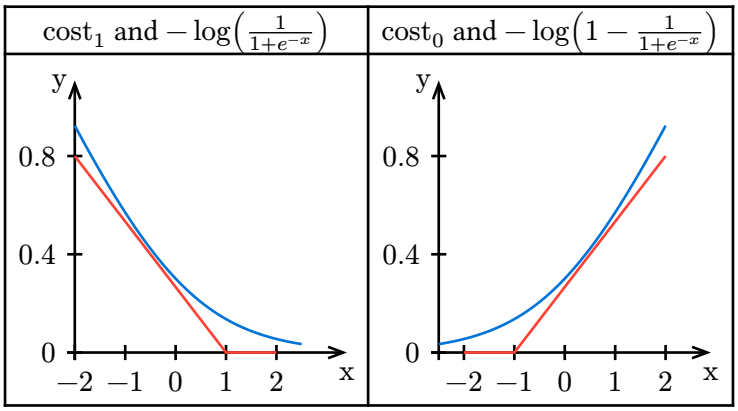
The loss function shows that if the label is 1, then the loss is $-\log(f_{\boldsymbol{w}}(\boldsymbol{x}_i))$, meaning that $f_{\boldsymbol{w}}(\boldsymbol{x}_i)$ or say $-\boldsymbol{w}^T\boldsymbol{x}$ should be as large as possible and vice versa. Note that the λ could be understood as how much we want to penalize the large $\boldsymbol{w}$, if we care more about the loss of each example, we just set λ small.

10.2. Idea

We set the term $-\log\left(\frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}\right)$ as $\text{cost}_1(\boldsymbol{w}^T\boldsymbol{x})$ and the term $-\log\left(1-\frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}\right)$ as $\text{cost}_0(\boldsymbol{w}^T\boldsymbol{x})$. The cost function for SVM is then:

$$L(\boldsymbol{w}) = C\sum_{i=1}^{m} \text{cost}_{y_i}(\boldsymbol{w}^T\boldsymbol{x}_i) + \frac{1}{2}\sum_{j=1}^{n} w_j^2$$

Constrast to the loss function of logistic regression, we change the cost term and the constant parameter before each term.



10.3. Nickname: Large Margin Classifier

When classify the data, classifier would figure out a boundary that could separate the data. The margin is defined as the distance between the boundary and the nearest point of the data. The boundary SVM figure out are the one that has the largest margin.

When the parameter of example cost C is really large, the classifier would be snesitive to the outliers, which may lead to overfitting. The reason why SVM would act like this lies in the optimization problem of the cost function(SVM tend to set the cost really small when C is large).