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	1. Offline Learning
2. Unsupervised Learning	2. Online Learning
3. Reinforcement Learning	
4. Semi-supervised 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. Mathmatical Foundation

2.1. Derivative

- For a function $f : \mathbb{R}^d \mapsto \mathbb{R}$, the derivative of f with respect to $\boldsymbol{x}$ is:

$$\frac{df}{d\boldsymbol{x}} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \frac{\mathrm{df}}{dx_1} & \frac{\mathrm{df}}{dx_2} & \cdots & \frac{\mathrm{df}}{dx_d} \end{bmatrix}^T$$

- For a fixed vector $\boldsymbol{b} \in \mathbb{R}^d$, consider $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{b}$ (dot product), then the derivative of $f(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{b}$$

- For a matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$, consider $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$ (quadratic form), then the derivative of $f(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x}$$

  In most cases, the matrix $\boldsymbol{A}$ is symmetric, so the derivative of $f(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 2\boldsymbol{A}\boldsymbol{x}$$

2.2. Maximum Likelihood Estimation

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

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

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.

2.3. Solution to Linear System

- Rouche-capelle theorem: For a linear system $\boldsymbol{X}\boldsymbol{w}=\boldsymbol{y}$ where $\boldsymbol{X}\in\mathbb{R}^d$ and the augmented matrix $\widetilde{\boldsymbol{X}}=[\boldsymbol{X}\ \ \boldsymbol{y}]$
  1. The system has a unique solution, iff. $\text{rank}(\boldsymbol{X})=\text{rank}\left(\widetilde{\boldsymbol{X}}\right)=d$.
  2. The system has no solution, iff. $\text{rank}(\boldsymbol{X})<\text{rank}\left(\widetilde{\boldsymbol{X}}\right)$.
  3. The system has infinitely many solutions, iff. $\text{rank}(\boldsymbol{X})=\text{rank}\left(\widetilde{\boldsymbol{X}}\right)<d$.

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 $\boldsymbol{y}_{\text{new}}={\boldsymbol{x}_{\text{new}}}^T\boldsymbol{w}$.

Obtaining the LSE Solution

We have loss function:

$$L(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m} e_i^2 = \frac{1}{m}\sum_{i=1}^{m} \left(f_{\boldsymbol{w},b}(\boldsymbol{x}_i) - y_i\right)^2 = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

Then we open the bracket and get:

$$L(\boldsymbol{w}, b) = \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}^T\boldsymbol{y}$$

The middle two terms are symmetric and since they are scalar, they actually equal to each other, so we have:

$$L(\boldsymbol{w}, b) = \boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y}$$

Then we take the derivative with respect to $\boldsymbol{w}$ and set it to zero:

$$2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{X}^T\boldsymbol{y} = 0$$

Then we get:

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$

If $\boldsymbol{X}^T\boldsymbol{X}$ is invertible, we have:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

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 $\boldsymbol{W} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$, 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 then have

$$\sum_{i=1}^{n} (y_i - \boldsymbol{w}^T \boldsymbol{x_i}) \boldsymbol{x}_i = 0 \Rightarrow \boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{w}^T \boldsymbol{x}) = 0$$

4. 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 deduction 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:

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

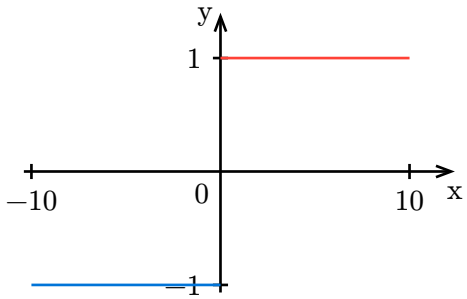

Figure 1: Sign function.

5.1. Python Demo

```
1  import numpy as np
2  from numpy.linalg import inv
3
```

```
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

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

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:

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

  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

```
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 $f_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$, then it has d variable and 1 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 $\boldsymbol{p}_i = \begin{bmatrix} 1 & m_1 & m_2 & \ldots & m_k \end{bmatrix}^T$, 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

- $y_{\text{new}} = \boldsymbol{p}_{\text{new}}^T \boldsymbol{w}$

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

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:

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

and the result is

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

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 $\mathbb{R}^{m \times m}$.

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

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

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

Primal-Dual Equivalence Proof:

Starting with dual form solution:

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

Develop equivalence through Woodbury identity $(\boldsymbol{I}+\boldsymbol{U}\boldsymbol{V})^{-1} = \boldsymbol{I} - \boldsymbol{U}(\boldsymbol{I}+\boldsymbol{U}\boldsymbol{V})^{-1}\boldsymbol{V}$:

$$\begin{aligned}
&\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} \\
&= \lambda^{-1}\boldsymbol{X}^T(\boldsymbol{I} + \lambda^{-1}\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{y} \\
&= \lambda^{-1}\boldsymbol{X}^T\left[\boldsymbol{I} - \lambda^{-1}\boldsymbol{X}(\boldsymbol{I} + \lambda^{-1}\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\right]\boldsymbol{y} \\
&= \lambda^{-1}\left[\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}\right] \\
&= \lambda^{-1}\left[\boldsymbol{I} - \boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\right]\boldsymbol{X}^T\boldsymbol{y} \\
&= \lambda^{-1}\left[\boldsymbol{I} - (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1} + \lambda\boldsymbol{I}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\right]\boldsymbol{X}^T\boldsymbol{y} \\
&= (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y})
\end{aligned}$$

Here $U \equiv \lambda^{-1}\boldsymbol{X}, V \equiv \boldsymbol{X}^T$ Final form matches primal solution. QED.

7.3. Python Demo

Refer to Python-Demo-4-Algorithm

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 $\boldsymbol{w} = [w_1 \ \cdots \ w_d]^T$
- Gradient of $\boldsymbol{w}$: $\nabla_{\boldsymbol{w}} C(\boldsymbol{w}) = \left[\frac{\sigma C}{\sigma w_1} \ \frac{\sigma C}{\sigma w_2} \ \cdots \ \frac{\sigma C}{\sigma w_d}\right]^T$. 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 shrink 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 $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = a$, $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0 = b$, 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 $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = \frac{1}{1+e^{-(\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0)}}$

- $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0 = 0$ 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.

  How to derive this: If a data lies on $\langle\boldsymbol{\theta},\boldsymbol{x}\rangle+\theta_0 = 0$, then we are not sure about the class of the data. So we simply put $\Pr(y=1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = \Pr(y=-1|\boldsymbol{x},\boldsymbol{\theta},\theta_0) = \frac{1}{2}$ here.

For how do we measure the performance, we could use likelihood function:

- For a single input $(\boldsymbol{x}_t, y_t)$, the likelihood function is

  $L(\boldsymbol{\theta},\theta_0|(\boldsymbol{x}_t,y_t) := \Pr(y_t|\boldsymbol{x}_t,\boldsymbol{\theta},\theta_0) = g(y_t(\langle\boldsymbol{x}_t,\boldsymbol{\theta}\rangle+\theta_0))$

- For the whole dataset, the likelihood function is

$$L(\boldsymbol{\theta},\theta_0|\boldsymbol{X},\boldsymbol{y}) = \prod_{t=1}^{m} g(y_t(\langle\boldsymbol{x}_t,\boldsymbol{\theta}\rangle+\theta_0))$$

Through maximizing this likelihood function, we could get the optimal parameter $\boldsymbol{\theta}$ and θ_0. Our strategy is stochastic gradient descent.

Procedure:

1. Goal

$$\max \prod_{t=1}^{m} g(y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0))$$
$$= \max \sum_{t=1}^{m} \log g(y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0))$$

2. We tranform it into the minimization problem for better generality: Minimization and Maximization in ML:

$$\min - \sum_{t=1}^{m} \log g(y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0))$$
$$= \min \sum_{t=1}^{m} \log\left(1 + e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}\right)$$

3. Take the gradients w.r.t. $\theta_0, \boldsymbol{\theta}$ of the function:

$$\frac{d}{d\theta_0} \log\left(1 + e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}\right) = -y_t \frac{e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}}{1 + e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}} = -y_t[1 - \Pr(y_t|\boldsymbol{x}_t, \boldsymbol{\theta}, \theta_0)]$$
$$\frac{d}{d\boldsymbol{\theta}} \log\left(1 + e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}\right) = -y_t[1 - \Pr(y_t|\boldsymbol{x}_t, \boldsymbol{\theta}, \theta_0)]\boldsymbol{x}_t$$

4. Set the gradient to zero: $\sum_{t=1}^{m}(-y_t[1 - \Pr(y_t|\boldsymbol{x}_t, \boldsymbol{\theta}, \theta_0)]) = 0$ and $\sum_{t=1}^{m}(-y_t[1 - \Pr(y_t|\boldsymbol{x}_t, \boldsymbol{\theta}, \theta_0)])\boldsymbol{x}_t = \boldsymbol{0}$

Note:

1. SGD has no significant change when the gradient of the full objective is zero.
2. We could add the regularization term into the function we concern:

$$\min \sum_{t=1}^{m} \log\left(1 + e^{-y_t(\langle \boldsymbol{x}_t, \boldsymbol{\theta} \rangle + \theta_0)}\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$$

9.3. Multi-Class Logistic Regression

For multi-class classification, we just replace

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

with softmax function:

$$\Pr(y = k|\boldsymbol{\theta}, \theta_0, \boldsymbol{x}) = \frac{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_c \rangle + \theta_{0,c}}}{\sum_{j=1}^{C} e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_j \rangle + \theta_{0,j}}}$$

where C is the number of classes.

Claim:

Two-class logistic regression is a special case of multi-class logistic regression.

Proof:

Without loss of generality, we assume $\boldsymbol{\theta}_1 = \mathbf{0}, \theta_{0,1} = 0$, using the equation defined for multi-class logistic regression, we have:

$$\begin{aligned}
\Pr(y=1|\boldsymbol{x}, \boldsymbol{\theta}_1, \theta_{0,1}) &= \frac{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_1 \rangle + \theta_0}}{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_1 \rangle + \theta_{0,1}} + e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2}}} \\
\Pr(y=2|\boldsymbol{x}, \boldsymbol{\theta}_2, \theta_{0,2}) &= \frac{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2}}}{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_1 \rangle + \theta_{0,1}} + e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2}}} \\
&= \frac{e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2}}}{1 + e^{\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2}}} \\
&= \frac{1}{1 + e^{-(\langle \boldsymbol{x}, \boldsymbol{\theta}_2 \rangle + \theta_{0,2})}}
\end{aligned}$$

The equation below appears in the same form of two-class logistic regression. QED.

9.4. Python Demo

Refer to Python-Demo-4-Algorithm

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

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

Here our class is $\{0, 1\}$. 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. Task

Given a dataset $\boldsymbol{D} = \{(\boldsymbol{x}_t, y_t) \in \mathbb{R}^d \times \{-1, +1\} : t = 1, 2, ..., n\}$, our goal is to learn from the data and predict the class of a new input $\boldsymbol{x}_{\text{new}}$.

10.3. Idea

10.3.1. Preliminary

- Margin of classifier: We have sign function

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

For a correct classification, we have $\text{sign}(\boldsymbol{\theta}^T\boldsymbol{x}) = y_t$. So we have $y_t\boldsymbol{\theta}^T\boldsymbol{x} > 0$. We define the margin of the classifier as:

$$y_t\boldsymbol{\theta}^T\boldsymbol{x}$$

If it's positive, then the sample is correctly classified and vice versa. What's more, the larger the margin is, the better the classifier is (we could infer this from the graph of sign function).

- Linearly separable: A dataset $\boldsymbol{D} = \{(\boldsymbol{x}_t, y_t) \in \mathbb{R}^d \times \{-1, +1\} : t = 1, 2, ..., n\}$ is linearly separable if

$$\exists\boldsymbol{\theta}, \forall t, y_t\boldsymbol{\theta}^T\boldsymbol{x} > 0$$

- Minimum margin:

$$\gamma = \min_t y_t\boldsymbol{\theta}^T\boldsymbol{x} > 0$$

- γ-linearly separable: A dataset $\boldsymbol{D}$ is γ-linearly separable if for some γ,

$$\exists\boldsymbol{\theta}, \forall t, y_t\boldsymbol{\theta}^T\boldsymbol{x} \geq \gamma$$

- Geometric margin: the smallest distance over all samples to the decision boundary

$$\gamma_{\text{geom}} = \frac{\gamma}{\|\boldsymbol{\theta}\|} = \frac{\min_t y_t\boldsymbol{\theta}^T\boldsymbol{x}}{\|\boldsymbol{\theta}\|}$$

The inverse of the margin $\gamma_{\text{geom}}^{-1}$ could represent the difficulty of the classification problem.

10.3.2. Optimization

For a γ-linearly separable dataset, we have the following optimization problem:

$$\max_{\boldsymbol{\theta}' \in \mathbb{R}^d} \frac{\gamma}{\|\boldsymbol{\theta}'\|} \; s.t. \; \forall t, y_t\boldsymbol{x}^T\boldsymbol{\theta}' \geq \gamma \Leftrightarrow \min_{\boldsymbol{\theta}' \in \mathbb{R}^d} \frac{\|\boldsymbol{\theta}'\|}{\gamma} \; s.t. \; \forall t, y_t\boldsymbol{x}^T\boldsymbol{\theta}' \geq \gamma$$

We could transform the term we optimize to be $\left\|\frac{\boldsymbol{\theta}'}{\gamma}\right\|$, which is the inverse of the geometric margin.

$$\min_{\boldsymbol{\theta}' \in \mathbb{R}^d} \left\|\frac{\boldsymbol{\theta}'}{\gamma}\right\| \; s.t. \; \forall t, y_t\boldsymbol{x}^T\frac{\boldsymbol{\theta}'}{\gamma} \geq 1$$

Since scaling the $\boldsymbol{\theta}'$ by constant does not change the decision boundary, the goal here simply put:

$$\min_{\boldsymbol{\theta}' \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{\theta}'\|^2 \; s.t. \; \forall t, y_t\boldsymbol{x}^T\boldsymbol{\theta}' \geq 1$$

It is so called the primal SVM problem.

10.3.3. Proof of Uniqueness

Claim:

The primal SVM problem has a unique solution.

Proof:

1. Suppose we have two distinct solutions $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$
2. Then $\|\boldsymbol{\theta}_1\| = \|\boldsymbol{\theta}_2\|$
3. Let $\overline{\boldsymbol{\theta}} = \frac{\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2}{2}$
4. By the linearity of inner product, we have

$$y_t \boldsymbol{x}^T \overline{\boldsymbol{\theta}} = y_t \boldsymbol{x}^T \left(\frac{\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2}{2} \right) = \frac{1}{2} y_t \boldsymbol{x}^T \boldsymbol{\theta}_1 + \frac{1}{2} y_t \boldsymbol{x}^T \boldsymbol{\theta}_2 \geq 1$$

5. By the triangle inequality, we have

$$\left\| \overline{\boldsymbol{\theta}} \right\| = \left\| \frac{1}{2} (\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2) \right\| \leq \frac{1}{2} \|\boldsymbol{\theta}_1\| + \frac{1}{2} \|\boldsymbol{\theta}_2\| = \|\boldsymbol{\theta}_1\| = \|\boldsymbol{\theta}_2\|$$

   If $<$ hold, that's a contradiction to the assumption that $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ are solutions. If $=$ hold, then $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ are the same solution.
6. Then the primal SVM problem has a unique solution.

10.3.4. SVM with offset

Sometimes the datas do not "centered around zero", so we could add a constant term to the decision boundary:

$$\boldsymbol{\theta}^T \boldsymbol{x} + \theta_0 = 0$$

Then the optimization goal becomes:

$$\min_{(\boldsymbol{\theta}, \theta_0) \in \mathbb{R}^d \times \mathbb{R}} \frac{1}{2} \|\boldsymbol{\theta}\|^2 \ s.t. \ \forall t, y_t (\boldsymbol{x}^T \boldsymbol{\theta} + \theta_0) \geq 1$$

Remarks: θ_0 only appears in the constraint. In that case, θ_0 is not in the vector that we optimize and it has no constraint to the "place" of the decision boundary, compared to $\widetilde{\boldsymbol{x}} = [\boldsymbol{x}, 1]$

10.3.5. Evaluation

The correctness of the classification model could be:

$$\frac{\text{correct predictions}}{\text{all predictions}} = \frac{1}{n} \sum_1^n \boldsymbol{I}\{y_i, f(\boldsymbol{x}, (\boldsymbol{\theta}, \theta_0))\}$$

$\boldsymbol{I}$ returns 1 if the parameters are the same and 0 otherwise.

10.3.5.1. Leave-one-out Cross Validation

To access the robustness of the model:

$$\text{LOOCV} = \frac{1}{n} \sum_1^n \boldsymbol{I}\{y_t, f(\boldsymbol{x}, (\boldsymbol{\theta}^{-t}, \theta_0^{-t}))\}$$

The parameters $\boldsymbol{\theta}^{-t}, \theta_0^{-t}$ are obtained training the model with the dataset that removes the t-th sample. We have the proposition:

Proposition:

$$\text{LOOCV} \leq \frac{N}{n}$$

where N is the number of support vectors and n is the size of dataset.

Proof:

1. for all datapoint that is not a support vector, we have $y_t f(\boldsymbol{x}, (\boldsymbol{\theta}^{-t}, \theta_0^{-t})) = 0$, so the term is not counted.
2. for all datapoint that is a support vector, we have $y_t f(\boldsymbol{x}, (\boldsymbol{\theta}^{-t}, \theta_0^{-t})) \leq 1$, so the term is counted.
3. Then LOOCV <= $\frac{N}{n}$ by sum over all datapoints.

If the support vectors are few, then the loss is small and the model generalizes well.

10.3.5.2. Allowing misclassification

The primal form of SVM with offset and slack:

$$\min_{\boldsymbol{\theta}, \theta_0, \boldsymbol{\varepsilon} \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}_+^n} \frac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_1^n \varepsilon_i \ s.t. \ \forall t, y_t(\boldsymbol{x}^T \boldsymbol{\theta} + \theta_0) \geq 1 - \varepsilon_t$$

We could set the slack variable to be large if we want to allow more misclassification for specific datapoint and here will still be penalty since we have sum over the slack variable. For the constant,

$$C \to \infty \Leftrightarrow \text{No misclassification}, C \to 0 \Leftrightarrow \text{Allow misclassification}$$

11. Feature Engineering

Machine learning algorithms prefer numerical data and fixed length feature vectors. Before, in computer vision, people need to manually extract the features from the image then put the features to linear model like SVM or other models for training. In that case, the feature extraction is the most important part since in the model training part, all we could do is to tune the parameters. Later in deep learning, the feature extraction part is done by neural network, which is more efficient and flexible.

11.1. Tabular Data

- Int/float: Could be used directly.
- Categorical: One-hot encoding. When there is long tail, we could put the rare category into a new category-unknown.
- Date/time: A feature list such as: [year,month,day,day_of_year,day_of_month,day_of_week].
- Feature Combination: Put the cartesian product of the features into the feature vector, we could obtain the relevance between the features.

11.2. Text Data

- Represent as token features

1. Bag of words: Count the number of times each word appears in the document.
2. Word-Embedding: Vectorizing the words such that similar words's distance is small (could be represented by cosine similarity).

- Pre-trained LLM: Use the pre-trained LLM to extract the features from the text.

11.3. Image Data

- Manually: SIFT, color feature, edge feature, etc.
- Pre-trained deep neural network: Put the image into the pre-trained deep neural network and extract the features from the second last layer (last layer is the classification layer).

12. Training Set & Model Complexity

This section is the summary of lecture 2 of Li Hongyi's course: What to do if my network fails to train

12.1. Context

For a classification problem, we could define the prediction function as

$$f_{h(x)} = \begin{cases} 1 \text{ if } e(x) \geq h \\ 0 \text{ otherwise} \end{cases}$$

Here, the $e(x)$ is a function that return specific features of the input x. The h is a threshold that we set. So here the parameter of the model is h.

- Model Complexity: We could measure the complexity of the model by the size of the parameter set, that is $|\mathcal{H}|$.
- Loss function (Given dataset D and parameter h):

$$L(h) = \sum_{i=1}^{m} \boldsymbol{I}\left\{y_i, f_{h(x_i)}\right\}$$

So then we could define the global optimal parameter $h^{\text{all}} = \text{argmin } L\left(h, D^{\text{all}}\right)$ and the local optimal parameter $h^{\text{train}} = \text{argmin } L\left(h, D^{\text{train}}\right)$.

12.2. Goal

Goal: We want the difference between $L\left(h^{\text{train}}, D^{\text{all}}\right), L\left(h^{\text{all}}, D^{\text{all}}\right)$ to be small, which means the model generalizes well. We could put a small constant to bound it:

$$L\left(h^{\text{train}}, D^{\text{all}}\right) - L\left(h^{\text{all}}, D^{\text{all}}\right) \leq \delta$$

Claim: $\forall h \in \mathcal{H}, \left|L\left(h, D^{\text{all}}\right) - L\left(h, D^{\text{all}}\right)\right| \leq \frac{\delta}{2} \Rightarrow L\left(h^{\text{train}}, D^{\text{all}}\right) - L\left(h^{\text{all}}, D^{\text{all}}\right) \leq \delta$

In the claim above, we call a training dataset D^{set} a good dataset if it satisfies the condition

$$\forall h \in \mathcal{H}, \left|L\left(h, D^{\text{all}}\right) - L\left(h, D^{\text{all}}\right)\right| \leq \frac{\delta}{2}$$

We want to know the probability of we using a bad dataset.

1. For a bad dataset, $\exists h \in \mathcal{H}, L\left(h^{\text{train}}, D^{\text{all}}\right) - L\left(h^{\text{all}}, D^{\text{all}}\right) > \frac{\delta}{2} = \varepsilon$
2. So for different h, there maybe some datasets that is bad because the difference between $L\left(h^{\text{train}}, D^{\text{all}}\right), L\left(h^{\text{all}}, D^{\text{all}}\right)$ is larger than ε.

3. Therefore,

$$P(\text{Bad Training Set}) = \bigcup P(\text{Bad Training Set}|h) \leq \sum_{h\in\mathcal{H}} P(\text{Bad Training Set}|h)$$

4. We introduce Hoeffding's inequality:

$$P(\text{Bad Training Set} \mid h) \leq 2\exp(-2N\varepsilon^2)$$

where N is the size of dataset. Then we have

$$P(\text{Bad Training Set}) \leq 2|\mathcal{H}|\exp(-2N\varepsilon^2) \ (1)$$

By the inequality (1), we could deduce, firstly, what is the probability of a bad dataset? Secondly, what size of dataset is required to achieve a certain probability of good dataset?

12.3. Trade-off

From (1), we could obtain the size of "credible" dataset $N \geq \frac{\log\left(\frac{2|\mathcal{H}|}{\delta}\right)}{2\varepsilon^2}$, that means, if we want a small N, then we need small $|\mathcal{H}|$ or large δ.

- Small $|\mathcal{H}|$: The model is simple, so the difference between $L\left(h^{\text{train}}, D^{\text{all}}\right), L\left(h^{\text{all}}, D^{\text{all}}\right)$ is small. However, the model could be bad because the model could not fit the reality well. Though the difference is small, it is of no use.
- Large δ: Large δ means we are willing to accept larger difference between our perception and reality. However, if the difference is large, the perception (model) is bad.

The solution for the trade-off is deep learning.

13. Model Selection

13.1. Generalization of Supervised Learning Algorithms

A supervised learning algorithm contains the following key elements:

- Unknown target function $f : \mathcal{X} \mapsto \mathcal{Y}$
- Training examples $\mathcal{D} = \{(x_i, y_i); i \in \{1, 2, ..., m\}\}$
- Hypothesis set $\mathcal{H}$
- Learning algorithm $\mathcal{A}$
- Final hypothesis $\mathcal{H}$

Usually, we do not know the performance of the model on new data and we need to evaluate it. For infinite data space, we just evaluate the model on new data. However, in practice, we only have a finite dataset $\mathcal{D}$. So we need to split the dataset into training set and test set.

- Data Partition:
  1. Training set, used to fit the model
  2. Validation set, used to fit the hyperparameters
  3. Test set, used to evaluate the model

13.2. Error Metric & Regularizer

13.2.1. Error Metric

Generally speaking, an error metric is a function that add penalty to the model if it classify (predict) the data wrong, denoted by:

$$E : \mathcal{y} \times \mathcal{y} \to \mathbb{R}$$

Here we have y as the ground truth and y' as the prediction.

- Regression
  - Square error:

    $e(y, y') = (y - y')^2$
  - Absolute error:

    $e(y, y') = |y - y'|$
- Classification
  - Zero-one error:

    $$e(y, y') = \boldsymbol{I}\{y \neq y'\}$$
  - Weighted Misclassification error: Used when we want to give different penalty to different types of errors, such as false positive is more serious than false negative

    $$e_{\beta(y,y')} = (1 - \beta)\boldsymbol{I}\{y = 1, y' = 0\} + \beta\boldsymbol{I}\{y = 0, y' = 1\}, \beta > .5$$
  - Balanced Error Rate:

    $$e(y, y') = \frac{1}{2}\left(\frac{\boldsymbol{I}\{y = 1, y' = 0\}}{n\{y = 1\}} + \frac{\boldsymbol{I}\{y = 0, y' = 1\}}{n\{y = 0\}}\right)$$
- Confusion Matrix: A table that shows the number of true positives, false positives, true negatives, and false negatives.

  1. True Positive: $\boldsymbol{I}\{y = 1, y' = 1\}$, False Positive: $\boldsymbol{I}\{y = 0, y' = 1\}$, True Negative: $\boldsymbol{I}\{y = 0, y' = 0\}$, False Negative: $\boldsymbol{I}\{y = 1, y' = 0\}$
  2. Precision: $\frac{\text{TP}}{\text{TP+FP}}$
  3. Recall: $\frac{\text{TP}}{\text{TP+FN}}$
  4. F1-Score: $\frac{2}{\text{Precision}^{-1}+\text{Recall}^{-1}}$, useful when exists the trade off between precision and recall.
  5. Accuracy: $\text{TP} + \frac{\text{TN}}{\text{TP+TN+FP+FN}}$
  6. AUC, ROC: ROC is a plot of the true positive rate against the false positive rate. AUC is the area under the ROC curve. When AUC is 1, the model is perfect. When AUC is 0.5, the model is random. When AUC is larger than 0.5, the model is better than random.

  This metric is not sensitive to **imbalanced dataset**, for example, some diseases are not common so the number of patients is small and comprise a very small part of the dataset, so if the model predict negative for all datapoint, it could perform well when using traditional accuracy metrics, which is not the case. Besides, AUC could reflect the overall performance of the model.

13.3. Generalizing & Overfitting

13.4. Data Partition: Train/Validation/Test

13.4.1. LI Hungyi: Why validation set → still overfitting?

The context here: We have 3 different models $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and obtain their optimal parameters h_1^*, h_2^*, h_3^* on the training set. Then we put the optimal parameters h_1^*, h_2^*, h_3^* to the validation set to evaluate the performance of the model and get the best model

$$h^* = \underset{h \in \{h_1^*, h_2^*, h_3^*\}}{\operatorname{argmin}} L(h, D_{\text{val}})$$

Then we put the best model h^* to the test set to evaluate the performance of the model. However, sometimes the model would still overfit the data and perform badly on the test set.

The process of model selection is actually another training process. So there is probability that the validation set is not a good dataset. Recall that:

$$P(D_{\text{val}} \text{ is bad}) \leq 2|\mathcal{H}| \exp(-2N\varepsilon^2)$$

where N is the size of dataset. If the number of parameters is too large, then the upper bound is large. So the probability of the validation set is bad could large, leading to a bad generalized model.

14. Ensemble Learning

14.1. Context

For a decision tree model, usually the prediction performance is bad and the model does not generalize well. But a decision tree model is simple and easy to interpret. So we could combine multiple decision tree models to get a better prediction performance.

14.2. Ensemble Learning

- The base learner used for ensemble should be:
  1. Comparatively accurate
  2. Diverse
- Error rate of a majority vote ensemble: Assume we have N independent base learners, each of them has error rate ε. Then the error rate of a majority vote ensemble is:

$$P(\text{Error}) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor} \binom{N}{k} (1-\varepsilon)^k \varepsilon^{N-k} \leq \exp\left(-\frac{N(1-2\varepsilon)^2}{2}\right)$$

  When $\varepsilon < 0.5$, the upper bound of the error rate would converge to 0 as N(number of base learners) increases.

Upper Bound for Error Rate

First, we introduce the Hoeffding's inequality. Let $\{X_i | 1 \leq i \leq n, \forall i, a \leq X_i \leq b\}$ be a set of bounded random varibles, and $-\infty < a \leq b < \infty$. Then we have:

$$P\left(\frac{1}{n}\sum X_i - \frac{1}{n}\sum E(X_i) \le -\delta\right) \le \exp\left(-\frac{2N\delta^2}{(b-a)^2}\right)$$

Now assume the set of random variables is our base learner and if they give correct prediction then the value is 1. $X_i \in [0,1] \Rightarrow a = 0, b = 1$ and $E[X_i] = 1 - \varepsilon$

$$P\left(\frac{1}{n}\sum X_i < \frac{1}{2}\right) = P\left(\frac{1}{n}\sum X_i - (1-\varepsilon) < \frac{1}{2} - (1-\varepsilon)\right)$$

$$\Rightarrow \delta = \varepsilon - \frac{1}{2}, (b-a)^2 = 1$$

Therefore,

$$P(\text{Error}) = P\left(\frac{1}{n}\sum X_i < \frac{1}{2}\right) \le \exp\left(-\frac{2N\left(\varepsilon - \frac{1}{2}\right)^2}{1}\right) = \exp\left(-\frac{N(1-2\varepsilon)^2}{2}\right)$$

14.3. Bagging

Goal: Obtain a set of diverse and high-performance base learners.

Methods: Different models, different hyperparameters, different training Algorithms

14.3.1. How to bagging

1. Sampling: The method is bootstrap sampling, which means we sample with replacement from the dataset. Each sample are of same size with slight variation.
2. Training: Train each base learner with the sampled dataset.
3. Aggregation: Majority vote for classification problem, average for regression problem.

Each bootstrap sampling process could only cover roughly 63.2% of the dataset. So we could use the remaining samples to evaluate the performance of the model. This is called out-of-bag (OOB) error. OOB error could help for hyperparameter tuning, early stopping and cross-validation.

Assume we want to learn the function g, and trained result is f', the expected model is $E[f']$. The error is:

$$\begin{aligned} E\left[(g - f')^2\right] &= E\left[(g - E[f'] + E[f'] - f')^2\right] \\ &= (g - f')^2 + E\left[(f' - E[f'])^2\right] \end{aligned}$$

The first term is the bias of the model and the second term is the variance of the model. The following are several conditions we have:

1. The expectation of bagged tree is equal to the expectation of the individual tree.

$$E[f'] = E\left[\frac{1}{B}\sum_{b=1}^{B} f'_b\right] = \frac{1}{B}\sum_{b=1}^{B} E[f'_b] = E[f'_b]$$

2. Bagged tree has the same bias as the individual tree.
3. Each tree is identically distributed but might not be independent identically distributed.

Therefore, we have the following:

Claim:

Correlated learners could not effectively reduce the variance.

Proof:

If we have a size B bagged model, the change of variance has 2 cases. For i.i.d random variables $X_1, X_2, ..., X_n$, we have:

$$\mathrm{Var}\left(\overline{X}\right) = \mathrm{Var}\left(\frac{1}{B}\sum_i X_i\right) = \frac{1}{B^2}\,\mathrm{Var}\left(\sum_i X_i\right) = \frac{1}{B^2}B\,\mathrm{Var}(X_i) = \frac{1}{B}\,\mathrm{Var}(X_i) = \frac{\sigma^2}{B}$$

For i.d random variables $X_1, X_2, ..., X_n$, let say the pairwise correlation between the base learners is ρ. The variance is:

$$\mathrm{Var}\left(\overline{X}\right) = \frac{1}{B^2}\,\mathrm{Var}\left(\sum_i X_i\right) = \frac{\sum_i \mathrm{Var}(X_i) + \sum_{i\neq j}\mathrm{Cov}\left(X_i, X_j\right)}{B^2} = \left(\rho + \frac{1-\rho}{B}\right)\sigma^2$$

Therefore, when $\rho \to 0, \sigma^2 \to \frac{\sigma^2}{B}$. When $\rho \to 1, \sigma^2 \to \sigma^2$. When the correlation between the base learners is large,the variance does not decrease much.

14.4. Random Forest

14.5. Boosting

14.6. Feature Selection

15. Multi Layer Perceptron

15.1. Perceptron

- Output

  It's similar to SVM. Given an input $\boldsymbol{x}$, weight $\boldsymbol{w}$ and bias b, the model output:

  $$o = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b), \text{where } \sigma(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$$

  We could also output the probability using softmax function.

- Training We have algorithm to train the model:

```
1                                                                    Algo
2  initialize w = 0, b = 0
3  repeat
4    if y_i (< w,x_i > + b) <= 0
5      w <- w + y_i x_i, b <- b + y_i
6    end if
7  until all classified correctly
```

The loss function is $\ell(\boldsymbol{x}, y, \boldsymbol{w}) = \max(0, -y_i \langle \boldsymbol{x}, \boldsymbol{w} \rangle)$ and the operation in loop is actually a gradient descent using 1 data each time.

Convergence Theorem

1. The data lies in an area of raidus R
2. $\exists \rho, s.t. y(\boldsymbol{w}^T \boldsymbol{x} + b) \geq \rho$, where $\|\boldsymbol{w}\|^2 + b^2 \leq 1$
3. The model converges after $\frac{R^2+1}{\rho^2}$ iterations.

The 1 layer perceptron is a linear classifier and could not be used to solve non-linear problems, such as XOR problem.

15.2. Multi Layer Perceptron

First we consider the situation of only 1 hidden layer.

1. Input layer: $\boldsymbol{x} \in \mathbb{R}^m$
2. Hidden layer: $\boldsymbol{W}_1 \in \mathbb{R}^{m \times n}, b_1 \in \mathbb{R}^n$
3. Output layer: $\boldsymbol{w}_2 \in \mathbb{R}^{n \times 1}, b_2 \in \mathbb{R}$

Procedure:

1. $\boldsymbol{h} = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + b_1)$
2. $o = \boldsymbol{w}_2^T \boldsymbol{h} + b_2$

σ is the activation function. We could use sigmoid function, tanh function, relu function, etc.

1. Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
2. Tanh function: $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
3. ReLU function: $\text{relu}(x) = \max(0, x)$

Secondely, to do task of multi-class classification, we could use softmax function.

1. Input layer: $\boldsymbol{x} \in \mathbb{R}^m$
2. Hidden layer: $\boldsymbol{W}_1 \in \mathbb{R}^{m \times n}, b_1 \in \mathbb{R}^n$ (we could have many layers here, the number of layers and the size of each layer are hyperparameters)
3. Output layer: $\boldsymbol{W}_2 \in \mathbb{R}^{n \times k}, \boldsymbol{b}_2 \in \mathbb{R}^k$

Procedure:

1. $\boldsymbol{h} = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + b_1)$
2. $\boldsymbol{O} = \boldsymbol{W}_2^T \boldsymbol{h} + b_2$
3. $\boldsymbol{y} = \text{softmax}(\boldsymbol{O})$

When desiging a MLP, the size of layers are important. The size of layers could be determined by the following rules:

1. The size of first hidden layer could be larger to track more complex features.
2. The size of following hidden layers could be reduced step by step to get the same size of expected output.