Machine Learning

Contents

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:

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

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

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:

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

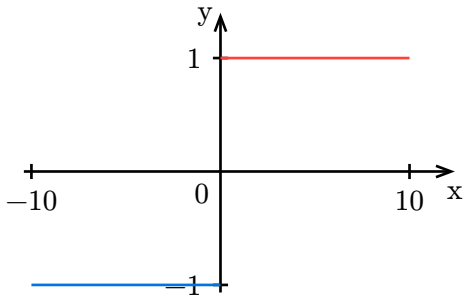


Figure 1: Sign function.

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:

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

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 $\boldsymbol{w} = [w_1 \ \dots \ w_d]^T$
- Gradient of $\boldsymbol{w}$: $\nabla_{\boldsymbol{w}}C(\boldsymbol{w}) = \left[\frac{\sigma C}{\sigma w_1} \ \frac{\sigma C}{\sigma w_2} \ \dots \ \frac{\sigma C}{\sigma w_d}\right]^T$. 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:

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

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

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

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:

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

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:

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

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. Multi Layer Perceptron

11.1. 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
```

```
    end if
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.

11.2. 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.

12. 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.

12.1. 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.

12.2. 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.

12.3. 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).

13. 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

13.1. 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 $h^{\text{train}} = \text{argmin } L(h, D^{\text{train}})$.

13.2. 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?

13.3. 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.

14. Model Selection

14.1. 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

14.2. Error Metric & Regularizer

14.2.1. 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.

14.2.2. Regularizer

In ridge regression, we use l_2 norm regularization to penalize the large $\boldsymbol{w}$. From this example, we could get the insight: Objective function = Error + Regularizer.

- Why regularization?
  1. To avoid overfitting
  2. Impose prior structral knowledge
  3. Reduce the variance of the model
  4. Improve interpretability: simpler model, automatic feature selection
- Why not?
  1. Gauss-Markov Theorem: Least square estimation is best linear unbiased estimator (BLUE) in some cases.
  2. Introduce regularization would also introduce bias.
- expected prediction error = bias^2 + variance + noise
- No free lunch:If the bias is low, we fit the data well but the variance is high, which means the model is overfitting. If the bias is high, we fit the data poorly but the variance is low, which means the model is underfitting.
- l_1 regularization (used in Lasso Regression):

$$L(h) = \text{Mean Square Loss} + \lambda \sum_{j=1}^{n} |w_j|$$

It penalize less than l_2 regularization. It is used when we want to perform feature selection. However, absolution function is not differentiable, so there is no closed form solution.

14.3. Generalizing & Overfitting

	Low test set error	High test set error
Low training set error	Generalize	Overfit
High training set error	Error	Underfit

- Fixing Underfit: Increase the model complexity(more features, more complex model), more data.

- Fixing Overfit: Reduce the model complexity(reduce features, less complex model), more data.

Genearally, it is not possible to overfit and underfit at the same time.

14.4. Data Partition: Train/Validation/Test

14.4.1. Validation Set

Procedure

1. Split the dataset into training set and validation set: $\mathcal{D} = \mathcal{D}_{\text{train}} + \mathcal{D}_{\text{val}}$
2. Pick m different models $\phi : \phi_1, \phi_2, \phi_3$
3. Train the model on the training set and get hypothesis for each ϕ_i:

$$\mathcal{H} = \{h_i = \text{argmin } L(h, D_{\text{train}}) |\ i \in [1, m]\}$$

   Compute error of each hypothesis on validation set D_{val} and choose the lowest:

$$\not{h}^* = \underset{h \in \mathcal{H}}{\text{argmin}}\ L(h, D_{\text{val}})$$

4. Retrain the model on the whole data set to obtain useful information)

14.4.2. Cross-Validation

Procedure

1. Split the dataset into k folds: $\mathcal{D} = \mathcal{D}_1 + \mathcal{D}_2 + ... + \mathcal{D}_k$, we now have k pairs of training set and validation set: $\{(\mathcal{D}_1, \mathcal{D}_1'), (\mathcal{D}_2, \mathcal{D}_2'), ..., (\mathcal{D}_k, \mathcal{D}_k')\}$
2. Pick m different models $\phi : \phi_1, \phi_2 ...$
3. For each ϕ_i, k
   1. Train the model on the training set and get hypothesis $h_{\phi_i, \mathcal{D}_k}$
   2. Compute error of the hypothesis on validation set $\mathcal{D}_k'$
4. Choose the best model $\phi^* = \underset{\phi \in \phi}{\text{argmin}}\ \frac{1}{k} \sum_{j=1}^{k} E_{\mathcal{D}_j'} \left(h_{\phi_i, \mathcal{D}_j} \right)$
5. Retrain the model on the whole data set to obtain useful information

The technique for folds is called **k-fold cross-validation**. Besides, we could also use **leave-one-out cross-validation** to split the dataset.

14.4.3. 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

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

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.

15. Ensemble Learning

15.1. 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.

15.2. Decision Tree

Definition: A decision tree is a tree-like model where each internal node represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a regression value.

15.2.1. Elements

- Node: A test on a feature x_i
- Branch: The outcome of the test, usually binary split for continuous feature or multi-way split for categorical feature
- Leaf: The prediction value y
- Path: A sequence of nodes from root to leaf, representing the decision rules

15.2.2. Training Process

The goal is to find a tree structure that minimizes the impurity of the leaf nodes. For classification tasks, we use information gain or Gini impurity as the metric:

Information Gain:

$$\text{IG}(\boldsymbol{\mathcal{D}}, a) = H(\boldsymbol{\mathcal{D}}) - \sum_{v \in \text{ Values}(a)} \left(\frac{|\boldsymbol{\mathcal{D}}_v|}{|\boldsymbol{\mathcal{D}}|} \right) H(\boldsymbol{\mathcal{D}}_v)$$

where $H(\boldsymbol{\mathcal{D}})$ is the entropy of dataset $\boldsymbol{\mathcal{D}}$: $H(\boldsymbol{\mathcal{D}}) = -\sum_{i=1}^{c} p_i \log_2(p_i)$

Gini Impurity:

$$\text{Gini}(\boldsymbol{\mathcal{D}}) = \sum_{i=1}^{c} p_{i(1-p_i)} = 1 - \sum_{i=1}^{c} p_i^2$$

The training procedure:

1. Start from root node with all training data
2. For each feature a:
   - Find the best split point that maximizes information gain or minimizes Gini impurity
   - Split the data into subsets according to the split point
3. Recursively build the tree for each subset until stopping criteria met:
   - Maximum depth reached

- Minimum samples in node reached
- No improvement in impurity

15.2.3. Advantages and Disadvantages

Advantages:

- Easy to understand and interpret
- Can handle both numerical and categorical data
- Requires little data preprocessing
- Can handle missing values

Disadvantages:

- Can create overly complex trees that do not generalize well
- Unstable: small changes in data can result in very different trees
- May create biased trees if classes are imbalanced

15.3. 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:

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

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,

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

15.4. Bagging

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

Methods: Different models, different hyperparameters, different training Algorithms

15.4.1. 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:

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

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.

In bagging, we usually we do not need prunning trees. Firstly, decision tree has the properties of low bias and high variance. And bagging techniques could reduce the variance. If we prun the decision trees, then the base learners are more likely to be correlated.

15.5. Random Forest

- **Goal** of Random Forest is to further reduce the correlation between the base learners.
- Procedure:
  1. Sampling: Choose B bootstrap split from the dataset.
  2. Training: Train B decision trees with the different bootstrap split. Note that each tree is trained with a random subset of features. If we have m features, we usually train the trees with $k < m$ features. For classification, we could use random forest with $k = \sqrt{m}$ features. For regression, we could use random forest with $k = \frac{m}{3}$ features.
  3. Aggregation: Majority vote for classification problem, average for regression problem. Ratio of being predicted of each class could be used as probability.
- Random forest could be tuned using OOB error and it could be done along the way of training.
- The number of features k and the minimum leaf size are hyperparameters. The number of trees B is also a hyperparameter.

15.6. Boosting

Besides bagging, we have another way to address the shortcoming of single decision tree, that is boosting.

15.6.1. Gradient Boosting

The idea is to train the base learners sequentially and each base learner is trained to correct the error of the previous base learner.

Procedure

1. Fit a simple model $T = T^0$ on the training data $\{(x_1, y_1), ..., (x_N, y_N)\}$
2. Compute the residuals $\{r_1, ..., r_N\}$ for model T
3. Fit a simple model T^1 to the current residuals, i.e., train using $\{(x_1, r_1), ..., (x_N, r_N)\}$
4. Build a new model $T \leftarrow T + \lambda T^1$
5. Compute the updated residuals $r_n \leftarrow r_n - \lambda T^1(x_n)$
6. Repeated steps 2-5 until the stopping condition met

Remarks

1. The learning rate λ is a hyperparameter that controls how much we trust the new model
2. The stopping condition could be:
   - Maximum number of iterations reached
   - The improvement of loss function is smaller than a threshold
   - The residuals are small enough
3. Each iteration, we train a new model to fit the residuals, which represents the error of our current model
4. The final model is a weighted sum of all the base models

Tuning Parameters There are three key parameters in Gradient Boosting:

1. Number of trees (B)
   - Total number of trees to be trained
   - Unlike Bagging/Random Forests, Boosting can overfit
   - Use cross-validation to determine optimal B
2. Shrinkage parameter (λ)
   - Small positive number (typically 0.01-0.1) that sets the learning rate
   - Controls how much we trust each new tree
   - Smaller λ requires more trees but often gives better generalization
3. Number of splits in each tree (d)
   - Usually just choose $d = 1$, i.e., tree stumps work well
   - Simple base learners are sufficient as the power comes from combining many models

Trade-offs:
- Smaller λ requires larger B
- Best parameter combination should be determined through cross-validation

15.6.2. AdaBoost

For classification task, the loss function is

$$\mathcal{L} = \frac{1}{N} \sum_n \mathbf{1}(y_n \neq \hat{y_n})$$

This is not differentiable w.r.t. predictions $\hat{y}$ So we use exponential loss function

$$\mathcal{L} = \frac{1}{N} \sum_n \exp(-y_n \hat{y_n})$$

The gradient of it is

$$\nabla \mathcal{L} = \left[-y_1 \exp(-y_1 \hat{y_1}) \;\; -y_2 \exp(-y_2 \hat{y_2}) \;\; \ldots \;\; -y_N \exp(-y_N \hat{y_N})\right]$$

The advantages of it are, firstly, just a reweighting for to the target values. Secondly, when the prediction is right, the gradient is small and vice versa.

So the **core idea** of adaboost: **Adaptively reweights samples**
- Put more weight on hard to classify instances and less on those already good
- New weak learners are added sequentially that focus their training on more difficult patterns

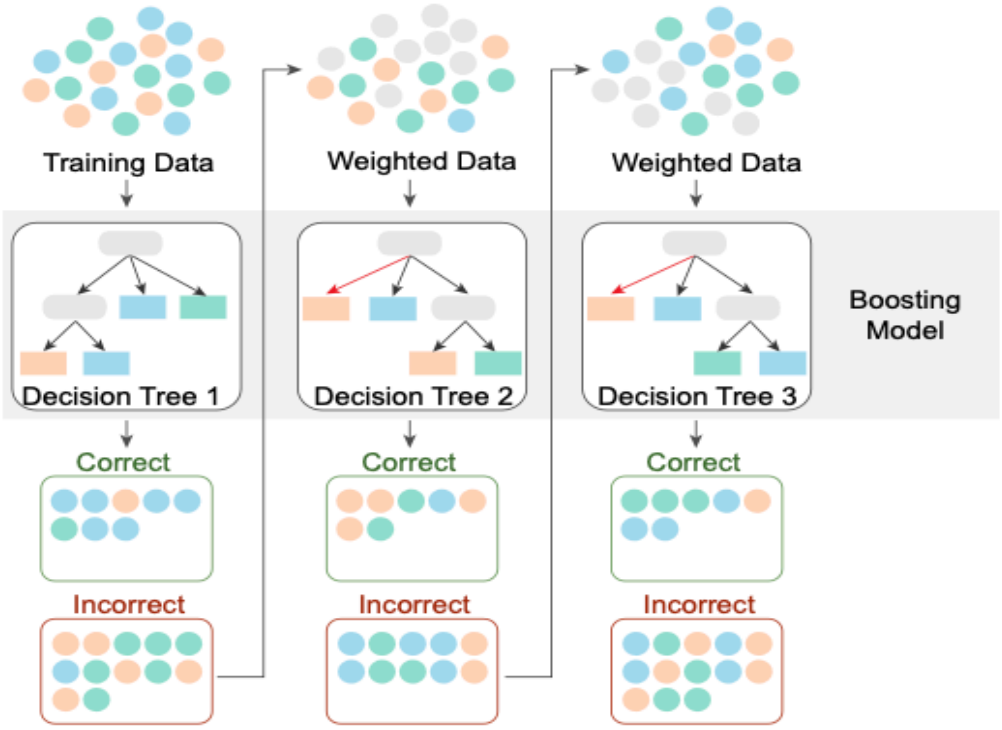


Figure 2: illustration of adaboost

AdaBoost

Procedure:

1. Initialize weights $w_i = \frac{1}{N}$ for all training samples
2. For each iteration:
   1. Train a week learner $T^{(t)}$ on the weighted training data $\{(x_i, w_i y_i)\}_{\{i=1\}}^{N}$ and get the error rate ε^t
   2. If $\varepsilon^t > 0.5$, stop the process.
   3. Compute the classifier weight $\lambda^t = \frac{1}{2}\ln\left(\frac{1-\varepsilon^t}{\varepsilon^t}\right)$
   4. Update the weights $w_i \leftarrow w_i \exp\left(\lambda^t \mathbf{1}\left(y_i \neq T^{(t)}(x_i)\right)\right)$
   5. Update the model with new weak learner (optional)
3. Combine all weak learners $T = \sum_t \lambda^t T^{(t)}$

How do we derive the classifier weight λ^t:

- When error ε^t is small, λ^t is large.
- λ^t should minimize exponential loss of learner $T^{(t)}$

$$\frac{\partial \mathcal{L}_{\text{exp}}}{\partial \lambda^{(t)}} = \frac{\partial \sum_i \exp(-y_i \lambda^{(t)} \hat{y}_i^{(t)})}{\partial \lambda^{(t)}} = -N_{\text{correct}} \exp(-\lambda^{(t)}) + N_{\text{wrong}} \exp(\lambda^{(t)}) = 0$$

$$\lambda^{(t)} = \frac{1}{2} \ln \left(\frac{N_{\text{correct}}}{N_{\text{wrong}}} \right) = \frac{1}{2} \ln \left(\frac{N_{\text{correct}}/(N_{\text{correct}} + N_{\text{wrong}})}{N_{\text{wrong}}/(N_{\text{correct}} + N_{\text{wrong}})} \right) = \frac{1}{2} \ln \left(\frac{1 - \epsilon^{(t)}}{\epsilon^{(t)}} \right)$$

Figure 3: Use gradient to derive the classifier weight of specific error rate

- Advantages of Boosting methods:
  - Accuracy
    1. Adaboost: turns weak learners into strong classifiers
    2. Gradientboosting: often beat other methods (XGBoost)
  - Flexibility
    1. Adaboost: simple and fast for binary tasks
    2. Gradientboosting: handles any loss function - regression, classification, ranking, etc.
  - Bonus: both can rank features by importance (gradientboosting especially)
- Limitations of Boosting methods:
  - Overfitting:
    1. Adaboost: sensitive to noisy data
    2. Gradientboosting: too many trees or poor tuning can overfit
  - Computational Cost:
    - Both: sequential training
    - Gradientboosting: more trees, more complex

15.7. Feature Selection

As the name says, feature selection is to choose a subset of relevant features from dataset. It could improve accuracy, reduce overfitting, reduce training time.

Types of feature selection:

- Filter: rank features(correlation) by importance, before training, use correlation, chi-squared, mutual information stuffs to compute the relationship of features and target. The advantage is fast and easy to implement. But may misses feature interaction
- Wrapper: test feature subsets with a model. Forward selection (each time add the feature that improves the models most), backward elimination (remove feature). The advantage is model-specific, consider interactions. But it is computationally expensive.
- Embedded: selection during training (often used in bagging and boosting). Such as lasso regression directly introduce penalty for some features. And some tree models would split on key features. The advantage is efficient and tailored to model. But it depends on model choice.

There are two main idea to evaluate the importance of features:

- Impurity/Gain: How much a feature improves splits or reduces error
- Usage: How often a feature is picked

For gradient boosting, use sum loss reduction perfeature. For Adaboost, use high frequency of features in high lambda learners. For bagging, use average reduction in impurity per feature.

16. Clustering

For a dataset $\mathcal{D}$ consist of n data points, we want to separate the data into k clusters。 That is to find the $\Delta = C_1, C_2, ..., C_k$, a partition of $\mathcal{D}$. And we have a $\mathcal{L}(\Delta)$ for the partition Δ. We also define $k(i)$ as the label of the i-th data point.

16.1. What is a good clustering?

We have some numerical measures.

- External
  - External information: The category of images
  - Measure how well the clustering match the groundtruth.
- Internal
  - Do not rely on external information
  - Assess the performance based on the intrinsic properties of the data
  - Compactness of clusters, separation between clusters, etc.

16.1.1. External Metrics

- **Purity**: Measure the extent to which clusters contain a single class.

$$\text{Purity} = \frac{1}{N}\sum_{i=1}^{k}\max_{k}|C_i \cap L_k|$$

  The constraint is purity metric do not penalize over-clustering.

- **Rand Index (RI)**: We first have four cases

True Positive	False Positive	False Negative	True Negative
$c_i = c_j, I_i = I_j$	$c_i = c_j, I_i \neq I_j$	$c_i \neq c_j, I_i = I_j$	$c_i \neq c_j, I_i \neq I_j$

  where c_i is the cluster label and I_i is the true label. Then we have, for all i, j,

$$\text{RI} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{T}} = 2\frac{\text{TP} + \text{TN}}{n(n-1)} \in (0, 1)$$

  If RI is close to 1, the clustering is good. The limitation is RI does not consider the chance agreement between the clusters

- **Adjusted Rand Index (ARI)**: It is adjusted to handle random clustering.

$$\text{ARI} = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]}$$

- **Fowlkes-Mallows Index (FMI)**: Measures the geometric mean of precision and recall.

$$\text{FMI} = \sqrt{\text{Precision} \times \text{Recall}} = \sqrt{\frac{\text{TP}}{\text{TP} + \text{FP}} \times \frac{\text{TP}}{\text{TP} + \text{FN}}}$$

- **Normalized Mutual Information (NMI)**: Measures the mutual information between the clustering and the true labels.
  - **Entropy of random variable**: It quantifies the information content of a random variable.

$$H(X) = E[-\ln(P(x))] = -\sum_{x \in X} P(x)\ln(P(x))$$

- **Conditional Entropy**: It measures the uncertainty of a random variable given another random variable.

$$H(X|Y) = E[-\ln(P(x|y))] = -\sum_{x\in X, y\in Y} P(x,y)\ln(P(x|y))$$

  $0 \leq H(X|Y) \leq H(X), H(X|Y) = 0$ iff X and Y are independent, $H(X|Y) = H(X)$ iff Y determines X completely, $H(X|Y) = H(X,Y) - H(Y)$
- **Mutual Information**: It measures the amount of information shared between two random variables.

$$\begin{aligned} I(X,Y) = I(Y,X) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X,Y) \\ &= H(X,Y) - H(X|Y) - H(Y|X) \end{aligned}$$

  $\text{NMI} = \frac{2I(C,L)}{H(C)+H(L)} \in (0,1)$ The higher the better.

- **V-measure**: it is the harmonic mean of homogeneity and completeness. The homogeneity measures the purity of each cluster, and the completeness measures how well all data in one class got into the same cluster.

$$\text{homogeneity} = 1 - \frac{H(C|L)}{H(C)}$$

$$\text{completeness} = 1 - \frac{H(L|C)}{H(L)}$$

$$\text{V} = 2\,\frac{\text{homogeneity completeness}}{\text{homogeneity} + \text{completeness}} \in (0,1)$$

  The higher the better.

16.2. Internal Metrics

- **Silhouette Score**: It measures the average similarity of a data point to its own cluster compared to other clusters.

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1]$$

  $a(i)$ is the average distance of i to other data points in the same cluster, $b(i)$ is the minimum average distance of i to other clusters. We take the average of all the points' $s(i)$ to get the final score. 1 is well clusterd, 0 is on the border, -1 is misclusterd.

- **Calinski-Harabasz Index**: It measures the ratio of the between-cluster variance to the within-cluster variance.

$$\text{CH} = \left(\frac{B}{W}\right) \times \frac{N-k}{k-1}$$

  B is the between-cluster dispersion and W is the within-cluster dispersion. N is the number of points and k is the number of clusters. $B = \sum_{i=1}^{k} n_i \|c_i - c\|^2$. c_i is the center of

the i-th cluster and c is the center of all the points. $W = \sum_{i=1}^{k} \sum_{j \in C_i} \|x_j - c_i\|^2$. It calculates the dispersion of each points and corresponding cluster center. The larger the better.

- **Davies-Bouldin Index**: It measures the average similarity of a data point to its own cluster compared to other clusters. Lower values mean better clustering.

$$\mathrm{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \frac{S_i + S_j}{d(c_i, c_j)}$$

- **Dunn Index**: It measures the ratio of the minimum distance between clusters to the maximum diameter of any cluster.

$$\mathrm{DI} = \frac{\min_{1 \leq i \leq j \leq k} d(C_i, C_j)}{\max_{1 \leq I \leq k} \mathrm{diameter}(C_I)}$$

It's obvious that the larger the better.

16.3. How to find a good clustering?

In optimization process, it is hard to operate on those metrics since they are mostly indifferentiable. So we just use proxy optimization object but use the metrics when we evaluate the quality of clustering.

We have three types of **hard clustering algorithms**:

- Prototype-based: k-means, k-mendoids
- Density-based: DBSCAN
- Hierarchical methods

16.3.1. Prototype-based Algo

The definition of this kind of algorithms is clustering that represent each cluster by a prototype (centorid, medoid) that summarize the group. The key idea is to assign the data to the nearest prototype and optimize a distance based objective function.

This kind of algo is simple, scalable and effective for **spherical** clusters, and they are sensitive to initialization and cluster shape.

16.3.1.1. K means

Algorithm

k-means

Input: Dataset $X = \{x_i\}_1^n$ and the number of clusters k

1 Initialize centers $\{\mu_i\}_1^K$ (e.g., random points from X)

2 Repeat

Assign step: Assign each point x_i to the nearest center

$$r_i = \mathrm{argmin}_k \|x_i - \mu_k\|$$

Update step: Update the centers to the mean of the assigned points

k-means

$$\mu_k = \frac{1}{n_k} \sum_{i=1}^{n_k} x_i$$

Until: Maximum iterations reached or assignments no longer change, or the cost function (qudratic distortion) coverges.

3

$$\mathcal{L}(\Delta) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

Proposition: The k-means algorithm decreases the cost function monotonically.

We could use elbow plot to find the optimal number of clusters. Set a range for k and compute the within-cluster sum of squares (WCSS) for each k. The plot of WCSS vs k is called elbow plot. The optimal k is the one where the elbow occurs.

- Pros: Simple and fast, scalable, effective for spherical clusters.
- Cons: Sensitive to initialization, assume similar size and density of clusters.

We may have some problems about initialization, if the realistic clusters are numerous, then we are impossible to start with k centers that captures all the clusters. We have some solutions:

1. Fastest First Traversal: each time choose the farthest point from the current centers as the next center.
2. K-means++: each time compute the probability of choosing each point as the next center.
3. K-log k

k-means++ Algorithm

1. Choose the first center randomly
2. For each point, compute the squared distance to the nearest center
3. Select the next center with probablity proportioanl to that squared distance
4. Repeat until k centers are chosen

It reduce the sensitivity of bad initialization and guarantees $O(\log K)$ solution.
$E(\varphi_{\text{kmeans++}}) \leq 8(\ln K + 2)\varphi_{\text{OPT}}$. That is the loss function of k-means++ is most $8(\ln K + 2)$ times the optimal loss function values.

16.3.1.2. K-Medoids

Similar to k-means but uses actual data points as centers.

Steps:

1. Initialize k medoids randomly
2. Assign each point to the nearest medoid
3. Update mediods to minimize the total cost
4. Repeat 2 and 3 until convergence

- Cost: $\mathcal{L}(\Delta) = \sum_k \sum_{i \in C_k} \|x_i - \mu_k\|$

16.3.1.3. Non-linear Data

Real world data are not linearly separable in original feature space but could be separable in a higher dimensional space. So we need a mapping from original feature space to a higher dimensional space.

We use kernel tricks: use low dimensional function to represent the high dimensional function, implicitly map the data to a higher dimensional space.

1. Linear Kernel: $K(x,y) = x^T y$
2. Polynomial Kernel: $K(x,y) = (x^T y + c)^d$
3. Gaussian Kernel: $K(x,y) = \exp(-\gamma\|x-y\|^2)$
4. Sigmoid Kernel: $K(x,y) = \tanh(\alpha x^T y + c)$

Algorithm

Kernel k-means

1 Initialize: randomly assign points to k clusters
Assign: for each point, assign it to the cluster with the nearest center

2 $$\|\varphi(x_i) - \varphi(m_k)\|^2 = 1 - \frac{1}{|C_k|}\sum_{x_j \in C_k} K(x_i, x_j) + \frac{1}{|C_k|^2}\sum_{x_j, x_l \in C_k} K_{jl}$$

3 Update: update the centers to the mean of the assigned points

4 Repeat until the assignments no longer change

16.3.2. Density-based Clustering

Motivations (limitations of prototype clustering):

1. Assume spherical, evenly sized clusters
2. Require pre-specifying K
3. struggle with noise and non-convex shapes

The key idea is to seperate the high density regions with low density regions. It could handle multiple shapes of clusters and naturally deal with noise.

16.3.2.1. DBSCAN

For DBSCAN, we first need to define some concepts: For a datasets $X = \{x_i\}_1^n$

- ε-neighborhood: $N_\varepsilon(x_i) = \{x_j \in X : d(x_i, x_j) \le \varepsilon\}$
- Core point: x_i is a core point if $|N_\varepsilon(x_i)| \ge$ minPts where minPts is a pre-specified threshold.
- Directly density-reachable: x_j is directly density-reachable from x_i if $x_j \in N_\varepsilon(x_i)$ and x_i is a core point.
- Density-reachable: x_j is density-reachable from x_i if there is a chain of points $x_i, x_{i_1}, ..., x_{i_k}, x_j$ such that x_{i_i} is directly density-reachable from $x_{i_{i-1}}$ for $i = 1, 2, ..., k$.
- Density-connected: x_j is density-connected to x_i if there is a point x_k that is density-reachable from both x_i and x_j.

The clusters C in DBSCAN is defined by the two rules:

1. Connectivity: $\forall x_i, x_j \in C, x_j$ is density-connected from x_i
2. Maximallity: $\forall x_j$, if $(x_i \in C) \wedge x_j$ is density reachable from $x_i \Rightarrow x_j \in C$

DBSCAN

Require: Dataset D, radius ε, minimum points minPts

initialize all points as unvisited
for each point $p \in D$ **do**
  mark p as visited
  $N \leftarrow$ getNeighbors(p,ε)
  if |N| < "minPts" **then**
    Label p as noise
  else
    create a new cluster C, add p to C
    expand cluster C
  endif
endfor

Expand the cluster C

Require: Initial cluster $C = \{p\}$, radius ε, minimum points MinPts

for each point $q \in C$ **do**
  if q is unvisited **then**
    Mark q as visited
    $N \leftarrow$ getNeighbors(q, ε)
    if $|N| \geq$ MinPts **then**
      $C \leftarrow C \cup N$
    endif
  endif
  if q is not yet assigned to any cluster **then**
    Add q to cluster C
  endif
endfor

Some **properties** of DBSCAN:

- Core points are deterministically assigned
  - Two core points p and q belong to same cluster if there exists a chain of core points
  - The relation $p \sim q$ is an equivalence relation because if reflexive, symmetric and transitive.
  - Then we obtain a partition induced by this equivalence relation
- Noise points are deterministically assigned
  - Noise point is not a core point and not within the ε-neighborhood of any core point
- Border points are non-deterministically assigned
  - Border points is not a core point but lies within the ε-neighborhood of a core point
  - The assignment depends on the order of core points being processed.

Here are the **pros and cons** analysis:

- Pros: no k is needed, can identify clusters of different shapes and sizes, handle noise and outliers.
- Cons: sensitive to epsilon and minPts, degrades with high-dimensional data

16.3.2.2. Curse of Dimensionality

When we deal with high dimensional data, we have some properties that is bad for clustering:

- The distance of any two points in the space are almost the same.
- The volume of the data concentrate near its surface $\rightarrow$ local density is hard to estimate.
- All vectors in high dimensional space are almost orthogonal $\rightarrow$ hard to estimate the similarity

DBSCAN is a method base on density and distance, with this bad things happen, DBSCAN is not good at clustering.

16.3.2.3. OPTICS

Motivation: DBSCAN could only set on ε and may lose some details when it is too large and may not be unable to find correct clustering if it is too small. So we need automatic ε for different density regions.

Key concepts in OPTICS:

- hyperparameters
  - ε: maximum radius, larger $\rightarrow$ cover more sparse clusters, smaller $\rightarrow$ faster convergence
  - minPts: minimum points in a clusters
- key concepts to achieve adaptive ε:
  - **Core Distance** $\mathrm{cd}(p) = \begin{cases} \text{UND} & |N_\varepsilon(p)| < \text{minPts} \\ d\left(p, N_\varepsilon^{\{\text{minPts}\}}(p)\right) & |N_\varepsilon(p)| \geq \text{minPts} \end{cases}$
    1. Minimum distance to make a point p to be a core point.
    2. "Local density"
  - **Reachability Distance** $\mathrm{rd}(q,p) = \begin{cases} \text{UND} & |N_\varepsilon(p)| < \text{minPts} \\ \max(\mathrm{cd}(p), d(p,q)) & |N_\varepsilon(p)| \geq \text{minPts} \end{cases}$
    1. Minimum distance to make p to be a core point and q is directly reachable from p
- Process:
  - Always do the points with minimum reachability distance first.
  - keep updating reachability distances for all unprocessed points.

Pros and Cons:

- Pros:
  1. Flexible ε
  2. Hierarchical: Produce reachability plot for multiple resolutions
  3. Robust with parameters
  4. Effectively identifies the outliers
- Cons:
  1. High computational cost
  2. Parameter Tuning is hard
  3. Interpretation of reachability plot is complexa

16.3.3. Hierarchical Clustering

Two types of hierarchical clustering:

1. Agglomerative: bottom-up, start with each point as a cluster and merge the closest clusters until all points are in one cluster.
2. Divisive: top-down, start with all points in one cluster and split the cluster into two clusters until each point is in a cluster. It is more computational intensive.

Linkage Method	Distance Measure	Characteristics
Single Linkage	Minimum pairwise distance	Tends to form elongated clusters; Sensitive to noise and outliers.
Complete Linkage	Maximum pairwise distance	Forms compact, spherical clusters; Less sensitive to noise compared to single linkage.
Average Linkage	Average of all pairwise distances	Balances between single and complete linkage; Produces more balanced clusters.
Ward's Method	Increase in total within-cluster variance when merged	Produces compact clusters with minimum variance; Efficient in minimizing within-cluster sum of squares.

Table: Comparison of Common Linkage Methods

Figure 4: Different distance measure ways in hierarchical clustering

- Ward's: Cost of merging cluster A and B

$$\Delta(A,B) = \sum_{x\in A\cup B} \|x-\mu_{A\cup B}\|^2 - \sum_{x\in A}\|x-\mu_A\|^2 - \sum_{x\in B}\|x-\mu_B\|^2 = \frac{2n_An_B}{n_A+n_B}\|\mu_A-\mu_B\|^2$$

Hierarchical clustering is a tree-based clustering method. The tree is called dendrogram. It do not need to specify the number of clusters and could provide good visualization. But it is computationally expensive and sensitive to noise.

- HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise): builds a hierarchy of clusters based on the density of the data.

16.3.4. Guassian Mixture Model (Soft Clustering)

The real world data are messy and we need some probabilistic model to describe the datas.

Recap of gaussian:

- Univariate gaussian: $P(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

  Maximum likelihood estimation:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2$$

- Multivariate gaussian: $P(x) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu)^T\Sigma^{-1}(\boldsymbol{x}-\mu)\right)$

  Maximum likelihood estimation:

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i, \Sigma = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\mu)(\boldsymbol{x}_i-\mu)^T$$

If we use a single gaussian distribution to model the data, it is not flexible enough to model data from different subdistributions such as two different gaussian distributions. We use **latent variables** to introduce the hidden sub-distributions.

For exaple, we have heights $\{x_i\}_1^n$ for a class, but there are different genders and different height distribution, then we introduce latent variable z_i to represent the gender of the i-th person. So the original distribution $P(x|\theta)$ change into $P(x,z|\theta) = P(x|z,\theta)P(z|\theta)$, where the first terms is the distribution of height given the gender and the second terms is the distribution of gender. If we know the distribution of heights, we could use bayesian to deduce the distribution of gender. $P(z=k|x,\theta) = \frac{P(x|z=k,\theta)P(z=k|\theta)}{\sum_{k=1}^K P(x|z=k,\theta)P(z=k|\theta)}$

GMM

The distribution of GMM is:

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

and

$$\sum_1^K \pi_k = 1$$

In the view of latent variable, we have:
1. $\theta = \{(\pi_i, \boldsymbol{\mu}_i, \Sigma_i)\}_1^K$
2. $p(x_i|z_i = k, \theta) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k)$
3. $p(z_i = k|\theta) = \pi_k$
4. $p(x_i|\theta) = \sum_{k=1}^K p(x_i|z_i = k, \theta)p(z_i = k|\theta) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k)$

For fitting of GMM, what we train is the parameters θ. For a dataset $X = \{x_i\}_1^n$, the log-likelihood is:

$$\log(L(\theta)) = \sum_{i=1}^n \log\left(\sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k)\right)$$

So it is hard to deriviate over the parameters since there is a sum term in log. So we use EM algorithm to maximize the log-likelihood.

EM algorithm

Goal: Iteratively estimate $\theta = \{\pi_k, \mu_k, \sigma_k\}_1^K$ using unlabled data.

1 Randomly Initialize θ^0
2 **for** $t = 1, 2, \ldots$ **do**
3 E-step (Compute the responsibilities): $r_{ik} = \frac{\pi_k \mathcal{N}(x_i|\mu_k, \sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(x_i|\mu_j, \sigma_j)}$
M-step (update parameters):
4

$$\pi_k^{t+1} = \frac{\sum_i r_{ik}}{n}$$

$$\mu_k^{t+1} = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}}$$

$$\sigma_k^{t+1} = \frac{\sum_i r_{ik} (x_i - \mu_k)^2}{\sum_i r_{ik}}$$

5 Until: θ^{t+1} converges or max iterations reached

- Why does EM works? (TBD)

Pros and Cons:

- Pros:
  1. Easy to implement
  2. likelyhood increase each iteration
  3. Converge to a local maximum
  4. Faster and more stable than GD
  5. Handles covariance constraints well
- Cons:
  1. Sensitive to initialization
  2. Risk of sigularities ($\sigma \to 0$)
  3. May overfit without validation

17. Dimensionality Reduction

Motivation:

- Curse of Dimensionality: High-dimensional data is sparse and hard to analyze.
- Goals: Visulization, Noise reduction, Feature extraction, Computational efficiency

17.1. Principal Component Analysis

PCA is a linear dimensionality reduction technique. It finds new axes(principal components, linear combinations of original features) that capture most important patterns in the data.

Important patterns:

- Maximize variance
- Minimize reconstruction error

PCA

Input: $X \in \mathbb{R}^{n \times p}$, with n samples and p features.

Procedure:

1. Standardize the data: $X_i' = \frac{X_i - \mu_i}{\sigma_i}$ or $X' = \left(V^{\frac{1}{2}}\right)^{-1}(X - \mu)$ Data is now zero mean and unit variance.
2. Compute the covariance matrix: $\Sigma = \frac{1}{n} X'^T X'$. That is the covariance between each pair of features.
3. Do Eigen-decomposition: $\Sigma = V \Lambda V^T$
4. Select the top k eigenvectors: $V_k = V_{:,1:k}$ (first k columns of V)

5. Project the data onto the new axes: $Y = X'V_k, X' \in \mathbb{R}^{n\times p}, V_k \in \mathbb{R}^{p\times k}$
6. Output: $Y \in \mathbb{R}^{n\times k}$

Variance:

$$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{p}\lambda_i}$$

Reconstruction error: For the projection matrix V_k, the reconstruction matrix is $\hat{X} = YV_k^T \in \mathbb{R}^{n\times p}$, then the error defined by square error is

$$\sum_{i=1}^{n}\left\|x_i - \hat{x_i}\right\|^2 = \sum_{i=1}^{n}\|x_i\|^2 - \sum_{i=1}^{n}\left\|\hat{x_i}\right\|^2$$

The latter term is the variance of the projected data and the former term is a constant. So maximize the variance is equivalent to minimize the reconstruction error.

The reason why we do standardization is to make the features have the same scale. If not, the features with larger scale would dominate the covariance matrix. Besides, centering the data ensure the principal components not relate to the mean.

For visualization, we could use $k = 2, k = 3$. For other applications, use elbow method in scree plot to determine k.

Pros and Cons:

- Pros:
  - Linear method, simple to implement
  - Computationally efficient
  - Reduces multi-collinearity
- Cons:
  - Assumes linear relationships between features
  - Less effective for preserving local structure
  - May lose critical information
  - Sensitive to outliers

17.2. Multidimensional Scaling(MDS)

The **proximity matrix** captures the pairwise distances (relationships) between data points. The goal of MDS is to find a low-dimensional representation of the data that preserves structure of proximity matrix.

17.2.1. Classical MDS

Based on **Euclidean distances**, assumes data is **metric**.

Classical MDS

Input: $D \in \mathbb{R}^{n\times n}$, where D_{ij} is the distance between samples i and j.

Goal: Find $X \in \mathbb{R}^{n\times k}, s.t. \left\|x_i - x_j\right\| = D_{ij}$

Procedure:

1. $D \to B$: Inner product matrix B could be calculated from D by double centering.

$$B = -\frac{1}{2} H D^{(2)} H$$

where $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ and $D_{ij}^{(2)} = \left(D_{ij}\right)^2$

2. $B \to E\Lambda E^T \approx XX^T$: The coordinate matrix X could be calculated from B by Eigen-decomposition.
3. $X = E_k \Lambda_k^{\left(\frac{1}{2}\right)} \in \mathbb{R}^{n \times k}$, where E_k is the first k columns of E and Λ_k is the first k columns and rows of Λ. $\lambda^{\left(\frac{1}{2}\right)} = \text{diag}\left\{\sqrt{\lambda_1}, \sqrt{\lambda_2}, ..., \sqrt{\lambda_k}\right\}$

We could use strain function to evaluate the quality of MDS.

$$\text{Strain}(X_1, ..., X_k) = \frac{\sum_{i<j}\left(b_{ij} - X_i^T X_j\right)^2}{\sum_{i<j} b_{ij}^2}$$

Lower value of strain function means better preservation of the original distances. And classical MDS provides a closed form solution for minimizing strain.

- Metric MDS: Classical MDS use Euclidean distance proximity matrix. But sometime sthe distance or relationship is not Euclidean, such as using dissimilarity matrix. So we use metric MDS in this case.
- Non-metric MDS: For reseach that only focus on the ordering of the distances, we use non-metric MDS.

Pros and Cons of MDS

- Pros:
  - Preserves pairwise distances
  - Flexible, could work with different distance metrics
- Cons:
  - Assume linear relationships, fail to capture non-linear structure
  - Computationally expensive
  - Sensitive to outliers in dissimilarity matrix
  - Results depend on the choice of metrics

17.3. Isomap

MDS use Euclidean distance which could be not suitable for samples on a manifold. Isomap use geodesic distance to preserve the local structure of the data.

Isomap

Procedure:

1. Compute the k-nearest neighbors graph woth radius ε
2. Geodesic distance: Compute the shortest path between each pair of points
3. Apply classical MDS to the shortest path matrix
4. Output low-dimensional representation

Pros and Cons of Isomap

- Pros:
  - Preserves local structure via geodesic distance
  - Handles non-linear relationships
- Cons:

- Computationally expensive
- Sensitive to the choice of ε and k
- Assume data lies on a single, well-behaved manifold

17.4. Linear Discriminant Analysis(LDA)

Use the **class label information** to project data to low-dimensional space (so it is supervised), **maximizing the between-class variance** and **minimizing the within-class variance**.

- Within-class scatter matrix: $S_w = \sum_{j=1}^{C} \sum_{i \in C_j} (x_i - m_j)(x_i - m_j)^T$
- Between-class scatter matrix: $S_b = \sum_{j=1}^{C} n_j (m_j - m)(m_j - m)^T$
- The problem is

$$\max_v \frac{v^T S_b v}{v^T S_w v}$$

- The solution is the eigenvectors of $S_w^{-1} S_b$ corresponding to the largest eigenvalues.
- The solution is $W = [w_1, w_2, ..., w_k]$, project the data to the new axes: $Y = XW \in \mathbb{R}^{n \times k}$

Pros and Cons of LDA

- Pros:
  - Effectively leverage label infromation to maximize class separation
  - Effecitive for classification tasks
- Cons:
  - **At most C-1 feature projections**
  - Assume linear class boundaries
  - **Assume Gaussian distributed data with equal covariance matrix**
  - Assumes different classes could be identifiable through the mean
- QDA: Quadratic Discriminant Analysis, assume **different classes have different covariance matrix**, allowing quadratic class boundaries. It models each class as a multivariate gaussian distribution, then assigns points based on maximum posterior probability.

17.5. Self-Organizing Map(SOM)

SOM is a unsupervised learning technique. It aims at mapping high-dimensional data to a low-dimensional space, and preserving the **topological structure** of the data.

SOM Algorithm Procedure

Input: $X \in \mathbb{R}^{n \times p}$

1. Randomly initialize weight vectors $w \in \mathbb{R}^p$ for each neuron on the grid.
2. Repeat until convergence (gradually reducing the learning rate and neighborhood size):
   1. Randomly select an input vector x from the dataset.
   2. **Find Best Matching Unit (BMU):** Identify the neuron whose weight vector is closest to x: $i^* = \text{argmin}_i \|x - w_i\|$.
   3. **Update Weights:** Adjust the weight vectors of the BMU and its neighbors to become closer to x.

$$w_{i(t+1)} = w_{i(t)} + \alpha(t)h_{i,i^*}(t)(x - w_i(t))$$

We set the neighborhood function $h(i,j) = \exp\left(-\frac{\|r_i - r_j\|^2}{2\sigma^2(t)}\right)$ to control the update. The closer to BMU, the larger the update. Besides, the neighborhood function determined by a dynamic radius $\sigma(t) = \sigma_0 \exp(-\frac{t}{\tau})$. It would enhance the speed at the beginning and slow down at the end.

Pros and Cons of SOM

- Pros:
  - Preserves topological structure
  - Unsupervised learning
  - Good for visualization
- Cons:
  - Computationally expensive
  - Sensitive to the choice of grid size and learning rate
  - May not converge

17.6. SNE, t-SNE and UMAP

17.6.1. SNE

Goal: preserve the neighborhood information of high dimensional data in low dimensional embedding.

The difference between SNE and SOM is, SNE use conditional probability distribution and KL divergence for training process. SOM use the distance between the input and the weight vector to update the weight vector, winner takes all.

SNE

Procedure:

1. Compute conditional probabilities $p_{j|i}$ in high dimensional space (jump from j to i in original space)

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}$$

   $p_{i|i} = 0, \sigma$ is the size of neighborhood
2. Define similar probabilities $q_{j|i}$ in low dimensional space

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}$$

   $q_{i|i} = 0, \sigma := \frac{1}{\sqrt{2}}$
3. Minimize the KL divergence: $\mathrm{KL}(P \| Q)$

Pros and Cons of SNE

- Pros:

- Preserves local structure
- Flexible framework for extensions

- Cons:
  - Crowding problem: Points clump in low dimensional space
  - Computationally expensive
  - Sensitive to the choice of perplexity, high perplexity means more neighbors are considered, but it could lead to overfitting.

- Crowding problem: The geometric structure of high dimensional space is not preserved in low dimensional space. And the points would be crowded in low dimensional space.

17.6.2. t-SNE

- Key difference with SNE: use Student-t distribution instead of Gaussian distribution to compute the conditional probability. t-distribution has heavier tails, so the probability of far away points would not be underestimated. If they are underestimated, the points would be crowded in low dimensional space. And it symmetrizes the objective function.
- Key definition:

$$p_{j|i} = \frac{p_{j|i} + p_{i|j}}{2n}$$

t-SNE

Procedure:

1. Compute conditional probabilities p_{ij} in high dimensional space

$$p_{ij} = \frac{p(j|i) + p(i|j)}{2n}$$

2. Define similar probabilities q_{ij} in low dimensional space

$$q_{ij} = \frac{\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \left\|y_i - y_k\right\|^2\right)^{-1}}$$

3. Minimize the KL divergence:

$$\mathrm{KL}(P \| Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

Pros and Cons of t-SNE

- Pros:
  - Excellent for visualizing high-dimensional data
  - Mitigates crowding problem
  - Robust for different datasets
- Cons:
  - Non-convex optimization problem
  - Do not preserve global structure well
  - Computationally expensive
  - Not easy to embed new points

17.6.3. UMAP

Deal with manifold structure.

- Key features:
  1. Faster and more scalable
  2. Preserves more global structure
  3. Flexible for various distance metrics
  4. Can embed new points

Procedure:

1. Find k-nearest neighbors, predefine the k
2. Calculate the weight for each neighbor
3. Use weighted average on low dimensional space $\frac{\sum w_i y_i}{\sum w_i}$
4. Use Cross-Entropy to optimize the low dimensional space representation

t-SNE could not deal with new points because it need to calculate the conditional probability of each point and use it as a normalization factor.

Pros and Cons of UMAP

- Pros:
  - Faster and more scalable
  - Preserves more global and local structure
- Cons:
  - Sensitive to hyperparameters (n_neighbors determine the local structure in high dimensional space, min_dist determine the global structure in low dimensional space)
  - May overfit the noise

18. Probabilistic Graphical Model

18.1. Introduction

Components of PGMs:

- Random Variables
- Edges: Probablistic dependencies
- Factors: Conditional probabilities or potentials

Types of PGMs:

- Bayesian Network: Directed Acyclic Graph (DAG), conditional dependencies, directed edges
- Markov Random Field: Undirected Graph, potential functions over cliques, for cyclic or mutual dependencies
- Factor Graph: Undirected Graph

18.2. Bayesian Network

- Nodes set: $X_{1:d} = \{X_1, X_2, ..., X_d\}$
- Joint distribution: $P(X_{1:d}) = \prod_{i=1}^{d} P(X_i \mid \text{pa}(X_i))$

Static Bayesian Network has 3 basic structures:

- Chain: $A \to B \to C \Rightarrow P(A, B, C) = P(A)P(B|A)P(C|B)$

  We have A is independent to C given B. And A would be dependent to C when B is not observed. Besides, the chain is markovian: $P(X_3|X_{1:2}) = P(X_3 \mid X_2)$.

- Fork: $A \leftarrow B \rightarrow C \Rightarrow P(A,B,C) = P(B)P(A|B)P(C|B)$

  We have given B, A is independent to C. And A may be marginally dependent to C.

- Sink: $A \rightarrow B \leftarrow C \Rightarrow P(A,B,C) = P(A)P(B|A,C)P(C)$

  In this case, A and C are marginally independent. And A would be dependent to C when A is observed.

- Independence deduction:

d-separation

- A, B, C are arbitrary non-intersecting sets of nodes. C can be empty
- We wish to know whether $A \perp\!\!\!\perp B \mid C$ holds or not.
- Construct all possible (undirected) paths from any node in A to any node in B.
- A path is blocked by C if
  - The path contains a chain $i \rightarrow j \rightarrow k$ or a fork $i \leftarrow j \rightarrow k$ such that the middle node j is in C
  - The path contains a v-structure $i \rightarrow j \leftarrow k$ such that the j (and all the descendants) is not in C
- If all paths from A to B are blocked, then A is d-separated from B by C

Theorem 1

(d-separation) If A is d-separated from B by C, then the joint distribution over all the variables in the graph will satisfy $A \perp\!\!\!\perp B \mid C$

- Linear regression model using Bayesian Network:

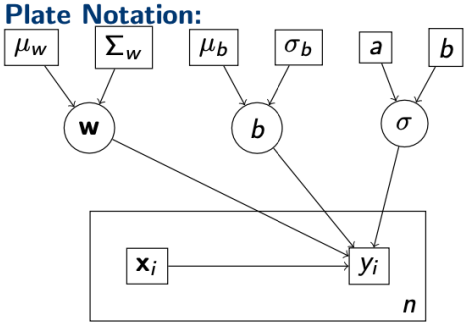
Static Bayesian Network: Bayesian Linear Regression

- Bayesian linear regression model

$$y_i \sim \mathcal{N}(\mathbf{w}^T \mathbf{x}_i + b, \sigma^2)$$
$$\mathbf{w} \sim \mathcal{N}(\mu_w, \Sigma_w)$$
$$b \sim \mathcal{N}(\mu_b, \sigma_b)$$
$$\sigma^2 \sim \text{Inv-Gamma}(a, b)$$

- Variables:
  - $\mathbf{w}$: Weight vector (latent)
  - b: Bias term (latent)
  - σ^2: Variance (latent)
  - $\mathbf{x}_i$: Input features (observed)
  - y_i: Target variable (observed)

Plate Notation:



- Limitations of static Bayesian Network:
  1. Random variables' relationship are at single time point
  2. Cannot capture temporal dynamics or evolving dependencies

- Dynamic Bayesian Networks (DBNS):
  1. Extend Beyesian networks to model temporal dynamics

2. Represent variables and their relationships over time
3. Enable efficient inference and prediction in dynamic systems

18.3. Markov Random Field

- Markov Chain: model for systems that transition between states. It consists of a set of states and a transition probability between them.
- Key properties: the future state is only dependent on the current state, not the past states.

$$P\big(X_{t+1} = s_j | X_t = s_i, X_{t-1}, ..., X_0\big) = P\big(X_{t+1} = s_j \mid X_t = s_i\big)$$

- Components of a Markovw Chain
  - States: A finite set of possible states
  - Transition Probabilities: $P\big(s_i \to s_j\big)$
  - Initial State Distribution: $\pi_0 = \{P(X_0 = s_i)\}_{i=1}^n$
  - Stationary Distribution π^*: Probability that satisfies $\pi^* = \pi^* P$

18.3.1. Dynamic Bayesian Network: Hidden Markov Model

$$P(X_{1:T}, Z_{1:T}) = P(X_1) \prod_{t=2}^{T} P(X_t \mid X_{t-1}) \prod_{t=1}^{T} P(Z_t \mid X_t)$$

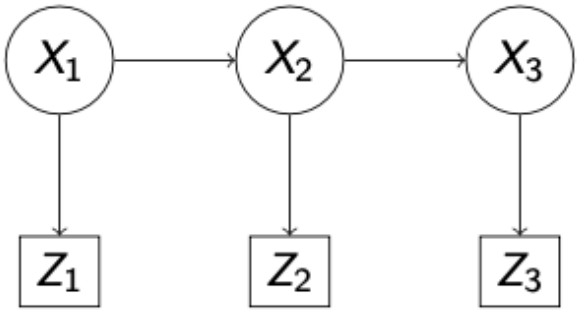


Figure 7: illustration of HMM

Here we have $X = \{X_i\}_1^n$ is the hidden states, $Z = \{Z_i\}_1^n$ is the observed states. X_t is the hidden state at time t, Z_t is the observed state at time t.

- Transition Probabilities: $P\big(X_t = s_j \mid X_{t-1} = s_i\big)$ s_i is an element from a set of hidden states.
- Emission Probabilities: $P\big(Z_t = o_k \mid X_t = s_j\big)$ where o_k is an element from a set of observalble values.
- Initial State Distribution: $P(X_0 = s_i)$

HMM could be used to:

1. Likelihood Estimation: calculate $P(O|\lambda)$, where λ is the parameter and O is the observed sequence.
2. Decoding: infer hidden state sequence S given observed sequence O and parameter λ.
3. Learning, given observations O, learn the model parameters λ that maximize $P(O|\lambda)$.

18.3.2. Markow Random Field

There maybe undirected influence between variables, such as the weather of today is influenced by the weather of yesterday.

Markov random field is a model for representing joint distribution over a set of variables. It is defined on an undirected graph, where nodes represent variables and edges represent mutual dependencies.

Markov Random Field

Key components of an MRF:

- Graph structure: $G = (V, E)$
- Cliques: the set of nodes of subgraph that is fully connected
- Maximal cliques: A clique that is not a subset of any other clique
- Potential functions: $\varphi_{C(X_C)}$, it assigns non-negative values to configurations of clique C. We usually use exponential function to make it non-negative:

$$\varphi_{C(X_C)} = \exp\left(-H_Q\left(X_Q\right)\right)$$
$$H_Q\left(X_Q\right) = \sum_{i,j \in Q} w_{ij} x_i x_j + \sum_{i \in Q} b_i x_i$$

- Joint probability:

$$P(X) = \frac{1}{Z} \prod_{C \in \text{ max cliques}} \varphi_{C(X_C)}$$

where Z is the partition function.

$$Z = \sum_{x} \prod_{C \in \text{ cliques}} \varphi_{C(X_C)}$$

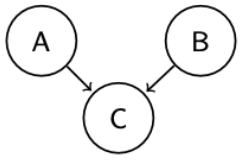
- Lemma: In MRF, if you could wirte $P(X, Y, Z) = f_1(X, Z) f_2(Y, Z)$, then X and Y are conditionally independent given Z.

Markov property in MRFS:

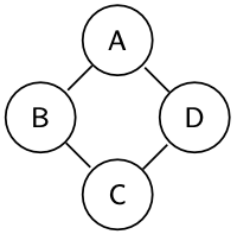
- Local Markov property: A variable is conditionally independent of all other variables given its neighbors.
- Global Markov property: Variables in non-adjacent nodes are independent given a separating set(all path from A to B need to pass through separating set C).
- Pairwise Markov property: Non-adjacent variables are independent given all others,
- Conditional Random Field: MRF is a generative model the models the joint distribution of all variables, while CRF is a discriminative model that models the conditional distribution of a variable given the others.

Non-equivalence of Directed / Undirected Graphical Models

- No undirected graphical model can capture the conditional independence assumptions of this directed graphical model



- No directed graphical model can capture the conditional independence assumptions of this undirected graphical model



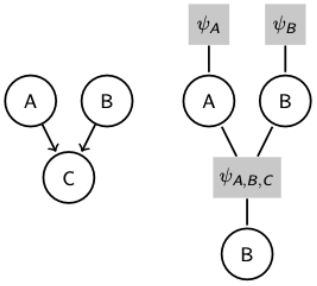
18.4. Factor Graph

A bipartite graph representig the factorization of a function, can describe both bayesian network and markov random field. Key components:

- variable nodes (circles)
- Factor nodes (squares)
- Edges: connect variable nodes to factor nodes

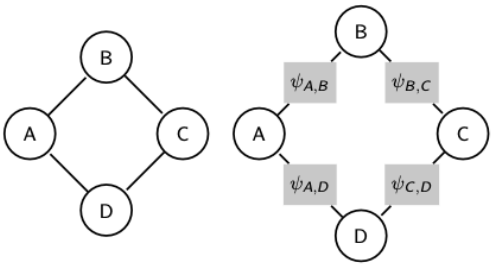
Transform BN/MRF to Factor Graph

- Each conditional and marginal distribution in a Bayesian Network becomes a factor



- $P(X) = P(A)P(B)P(C|A,B)$
- $P(X) = \psi_A \psi_B \psi_{A,B,C}$

- Each clique in a Markov Random Field becomes a factor



- $P(X) = \frac{1}{Z}\psi_{A,B}\psi_{B,C}\psi_{C,D}\psi_{D,A}$

Figure 9: illustration of transformation from BN and MRF to factor graph

19. Active Learning

There are a lot of unlabeled data and it's expensive to label them all. Active learning is a technique that help single out the most informative data to label.

Passive learning is a one-time process that train the model with all the available labeled data. In contrast to passive learning, active learning is an iterative process that involves selecting the most informative samples for labeling. Besides, active learning is less expensive in labelling process but more complicated in model design.

19.1. Two Scenarios

- Pool-based active learning:
  Assumption: a pool of unlabeled data is available.
  Process:
  1. Rank the unlabeled data based on the informativeness.
  2. Query the most informative data and label them.
- Stream-based active learning:
  Assumption: unlabeled data arrives sequentially.
  Process:
  1. First sample an unlabeled data
  2. Decide whether to label the data: Informative strategy or Random strategy

In real world applications, pool-based active learning is more common, but in case of limited budget, stream-based active learning is more suitable.

19.2. Query Strategy

- Random sampling: Randomly select unlabeled data to label.
- Query-by-Committee: Select the data that the committee is most uncertain about.
- Uncertainty sampling: Select the data that the model is most uncertain about.
- Expected model change: Select the data that the model is most likely to change.
- Density-based sampling: Select the data that is in the dense region.

19.2.1. Uncertainty Sampling

Query the most uncertain data, usually used in probabilistic model.

- Least confidence:

$$x_{\text{LC}} = \underset{x}{\operatorname{argmax}}\ 1 - P_\theta(y^*|x)$$

  where $y^* = \underset{y}{\operatorname{argmax}} P_\theta(y|x)$
- Smallest margin:

$$x_{\text{SM}} = \underset{x}{\operatorname{argmin}}\ P_\theta(y_1|x) - P_\theta(y_2|x)$$

  where y_1 and y_2 are the two most confident predictions for x.
- Max entropy:

$$x_{\text{ME}} = \underset{x}{\operatorname{argmax}}\ H(y|x)$$

19.2.2. Query-by-Committee

Use an ensemble of models to make the decision. And choose the sample with the highest disagreement.

- Vote Entropy: let $V(y)$ be the number of votes for y and we use K models. Then the vote entropy is

$$x_{\text{VE}} = \underset{x}{\operatorname{argmax}} \sum_{y \in Y} \frac{V(y)}{K} \log \frac{V(y)}{K}$$

- KL-Divergence: Let P_{θ_i} be the prediction of model θ_i and P_C be the prediction of the committee. Then the KL-Divergence is

$$x_{\text{KL}} = \underset{x}{\text{argmax}} \frac{1}{C} \sum_{i=1}^{C} \text{KL}\big(P_{\theta_i} \| P_C\big)$$

19.2.3. Effects of Outliers

Sometimes an instance is uncertain if it is an outlier. But including them is not useful for the model. We have some robust way to deal with singling out outliers.

- Density Weighting: weightiing the uncertainty of an instance by its density w.r.t the pool U.

$$\text{uncertainty} * \sum_{u} \frac{\text{sim}(x, u)}{|U|}$$

- Estimate model performance: The idea is choose the data that could minimize the model uncertainty over other unlabeled data in the pool.

$$x^* = \underset{x}{\text{argmin}} \sum_{u \in U} \sum_{y} P_\theta(y|x)(1 - P_{\theta^{+\langle x, y \rangle}}(y^*|u))$$

where $\theta^{+\langle x,u \rangle}$ is the model trained with dataset added $\langle x, y \rangle$ added.

19.2.4. Expected Model Change

Choose the data that could maximize the expected model change. Let $\nabla \mathcal{L}_{\theta(\langle x, y_i \rangle)}$ be the gradient after training the model with $\langle x, y_i \rangle$.

$$x^* = \underset{x}{\text{argmax}} \sum_{y} P_\theta(y|x) \| \nabla \mathcal{L}_\theta(\langle x, y_i \rangle) \|$$

19.2.5. Expected Error Reduction

Estimate the error reduction of the model after labeling the data. But it is costly to do so. There are some solutions:

- use models that naturally support incremental training
- use sampling to reduce $|U|$ when estimating the error reduction
- approximate training techniques

19.2.6. Variance Reduction

Select the data that could reduce the most in the variance of the model.

- $\mathbb{E}\Big[(y^* - y)^2 |\ x\Big] = \text{bias}^2 + \text{variance} + \text{noise}$
- Maximize the fisher information: a lower bound of the variance of the model (Cramer-Rao inequality)

19.2.7. Density-based Sampling

We aim to identify the core-set that well represent the distribution of universe, known as coreset selection, prototype learning. Some ideas:

- Reduce probability of selecting neighbours
- Minimize reconstruction error
- Distance-based weighting

19.3. Reinforcement Learning

Supervised learning learns from labeled data. Reinforcement learning learns from the environment to maximize the reward.

There are 5 key components in reinforcement learning:

- Agent: The learner
- Environment: The world
- Action: a_t, The action the agent takes
- Reward: r_t, The reward the agent receives
- State: s_t, The state the agent is in

19.3.1. Multi-Armed Bandit

It's a single state RL problem.The agent faces several decision available and would lead to different rewards. The goal is to learn the best action to maximize the expected reward.

There are 3 fundamental aims in bandits problem:
- Cumulative regret minimization: Regret = Maximum possible reward - Actual reward
- Best arm identification/Pure exploration: Fixed confidence level or fixed budget
- Simple regret minimization: Minimize one-step regret

19.3.2. Markov Decision Process

Compare to the bandits problem in which the environment would only give reward feedback, in MDP, the environment would give reward feedback and state feedback.

An MDP is defined by:
- A set of states $\mathcal{S} : s \in \mathcal{S}$
- A set of actions $\mathcal{A} : a \in \mathcal{A}$
- A transition function $T(s, a, s') = P(s'|s, a)$. It is actually the probability of transitioning from state s to state s' when action a is taken.
- A reward function $R(s, a, s')$. Sometimes just $R(s), R(s, a), R(s')$ is used.
- A start state s_0.
- A terminal state s_T (Maybe).

- Policy: Policy is a mapping from states to actions. $\pi : \mathcal{S} \to \mathcal{A}$ For a single agent, deterministic search problems, we want an optimal policy that maximizes the expected reward if we follow it, denoted by $\pi^* : \mathcal{S} \to \mathcal{A}$.
- Utility: The sum of (discounted) rewards.

How could we define a policy better over another policy? We have some considerations, such as total reward in the end, and people prefer to get reward sooner. For the second one, we introduce the discount factor $\gamma \in (0, 1)$. For a reward sequence $[r_1, r_2, ..., r_T]$, the discounted reward is $\sum_{t=1}^{T} \gamma^{t-1} r_t$.

A example is for our algorithm, we want to compare reward sequences $[1, 2, 3]$ and $[3, 2, 1]$. They are the same in the end, but if we consider the discount factor $\gamma = 0.5$, the discounted reward of $[1, 2, 3]$ is $1 + 0.5 \times 2 + 0.5^2 \times 3 = 2.75$, and the discounted reward of $[3, 2, 1]$ is $3 + 0.5 \times 2 + 0.5^2 \times 1 = 4.25$. So we prefer the second sequence.

Another thing is **Infinite Utilities** stuff, if the games lasts forever, the rewards would be infinite, then it's no use to compare each policy. In that case, we have 3 solutions:
- Finite horizon:
  1. Terminate episodes after a fixed T steps.
  2. Gives nonstationary policies.
- Discounting: $U([r_1, r_2, ..., r_T]) = \sum_{t=1}^{T} \gamma^{t-1} r_t \leq \frac{R_{\max}}{1-\gamma}$. Smaller γ means shorter term focus.

- Absorbing state: guarantee each policy would reach a terminal state.

19.3.3. Solving MDPs

We want to find the optimal policy π^* that maximizes the expected reward.

- optimal state value function:

$$V^* = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right]$$

- optimal action value function:

$$Q^* = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a)$$

For a problem, we could compute π^* and follow this optimal policy. But it is hard to compute π^* directly. Or we could just fix the π, it could simplyfy the tree graph, but the reward would be worse.

In RL, we have 2 aims:
- Minimize the regret
- Identify the best policy