Machine Learning Notes

Yu-Zhe Shi

May 16, 2020

Contents

Chapter 1

Evaluation Protocol

1.1 Error

Error is difference between prediction made by machine learning model and groundtruth of data. Consider a errors in m samples, then error rate is $E = \frac{a}{m}$ and accuracy is $1 - \frac{a}{m}$. The error between prediction and groundtruth of training set is **empirical error**, while the error on unseen testing data is **generalization error**. When a model fits training data well yet performance poor on novel testing data, the model is considered **overfitted**.

1.2 Evaluation

Testing data should be independent indentically distributed (i.i.d.) with training data and be novel to training data. Hence, it's reasonable to split a dataset into training set and testing set. In practise, we don't tune the model at testing set, but modify the parameters on an additional validation set separated from training set.

1.2.1 Hold-out

Simply split dataset D into training set S and testing set T where $D = S \cup T$ and $\frac{|S|}{m} \in [2/3, 4/5]$.

1.2.2 Cross Validation

Let $D = D_1 \cup D_2 \cup \cdots \cup D_k$, where $D_i \cap D_j = \phi$. For each $i \in [1, k]$ selected as testing set, all the other sets serve as training set. Hence, we obtain k training and testing experiments respectively and the final result is the mean of k results.

1.2.3 Bootstrapping

Sample d randomly from D to construct D' without $D \leftarrow D \setminus d$ until $|D'| = m$. Since the probability that a sample in D never be sampled is $(1 - \frac{1}{m})^m$, we obtain

$$\lim_{m \to \infty} (1 - \frac{1}{m})^m \approx \frac{1}{e} = 0.368 \tag{1.1}$$

which means that the proportion of testing data is 36.8%. Though changing the real data distribution slightly, Bootstrapping is appropriate for ensemble learning and training from few data.

1.3 Performance Measure

1.3.1 Mean Square Error

For regression from $D = \{(x_1, y_1), \dots, (x_m, y_m)\}$, the **Mean Square Error** is the mostly used indicator.

$$E(f; D) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{1.2}$$

For distribution $\mathcal{D}$ and density $p(\cdot)$, MSE is

$$E(f; \mathcal{D}) = \int_{x \sim \mathcal{D}} (f(x) - y)^2 p(x) dx \tag{1.3}$$

1.3.2 Accuracy

We can write error as

$$E(f; D) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}(f(x_i) \neq y_i) \tag{1.4}$$

where $\mathbb{I}(\cdot)$ is the unit-impulse function. Accuracy is

$$acc(f; D) = 1 - E(f; D) \tag{1.5}$$

Similarly, we can rewrite it from the perspective of probability

$$acc(f; \mathcal{D}) = 1 - \int_{x \sim \mathcal{D}} \mathbb{I}(f(x) \neq y) p(x) dx \tag{1.6}$$

1.3.3 Precision and Recall

The result space of the model can be splited into 4 sets: 1) True Positive (TP), positive data that predicted correctly as positive; 2) False Positive (FP), negative data that predicted wrongly as positive; 3) False Negative (FN), positive data that predicted wrongly as neative; 4) True Negative (TN), negative data that predicted correctly as negative. It is obvious that $|TP \cup FP \cup FN \cup TN| = m$In a word, T/F denotes the correctness of the prediction and P/N denotes the output of the model. As we are evaluating a discriminative model, we usually measure the precision and the recall of the output. Precision is

$$P = \frac{TP}{TP + FP} \tag{1.7}$$

, the propotion of true result in positive predictions. Recall is

$$R = \frac{TP}{TP + FN} \tag{1.8}$$

, the ratio of true result in positive data. The precision and recall is a trade-off of machine learning models. Intuitively, if we aim for a model that makes as few errors as possible, we learn a model with higher precision but lower recall, since the model is ”strict” with ambiguous instances. If we aim for a model that finds real data as much as possible, we learn a model with higher recall but lower precision, since the model is ”tolerant” with ambiguous examples.
F1 metric is the harmonic mean of P and R

$$\frac{1}{F1} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R}) \tag{1.9}$$

yet more often, we have a bias on precision or recall. Let β be the indicator where we bias on recall when $\beta > 1$ and we bias on precision when $\beta < 1$

$$F_\beta = \frac{(1 + \beta^2)PR}{\beta^2 P + R} \tag{1.10}$$

When there are multiple hypothesis sets, we use the sum or the mean of aforementioned variables similarly. For multiple-class classification, we have multiple confuse matrix, thus multiple pairs of precision and recall. Hence, we obtain the macro measurement

$$\begin{aligned} macroP &= \frac{1}{n}\sum_{i=1}^{n} P_i \\ macroR &= \frac{1}{n}\sum_{i=1}^{n} R_i \end{aligned} \tag{1.11}$$

and the micro measurement

$$\begin{aligned} microP &= \frac{\overline{TP}}{\overline{TP}+\overline{FP}} \\ microR &= \frac{\overline{TP}}{\overline{TP}+\overline{FN}} \end{aligned} \tag{1.12}$$

1.3.4 ROC and AUC

The coordinates of Receiver Operating Characteristic (ROC) curve is difined as

$$TPR = \frac{TP}{TP \cup FN},\ FPR = \frac{FP}{FP \cup TN} \tag{1.13}$$

true positive ratio (TPR) and false positive ratio (FPR) serves as horizontal axis and vertical axis respectively. Hence, we measure the area under curve (AUC) of ROC, which is increases monotonically as model performance increases. An approximation of AUC is

$$AUC = \frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1}-x_i)(y_{i+1}+y_i) \tag{1.14}$$

1.3.5 Weighted Cost

All aforementioned metrics regard each failure equivalently, yet it is not appropriate in real scenes. We can rewrite error as

$$E(f;D;cost) = \frac{1}{m}\left(\sum_{x_i \in D^+} \mathbb{I}(f(x_i) \neq y_i)cost_{FP} + \sum_{x_i \in D^-} \mathbb{I}(f(x_i) \neq y_i)cost_{FN}\right) \tag{1.15}$$

1.4 Multiple Comparison

1.4.1 Pair t-tests

Here we denote error as ε. For model A and B in k-fold testing, we calculate $\Delta\varepsilon_i = \varepsilon_i^A - \varepsilon_i^B$ to measure the performance difference of A and B. If

$$\tau_t = |\frac{\sqrt{k}\mu}{\sigma}| < t_{\alpha/2,k-1} \tag{1.16}$$

then we announce that A and B don't have significant performance difference on the dataset. To independent error at every iteration, we produce two-fold validation in 5 iterations, where we shuffle the dataset randomly in every iteration. We calculate mean error of the first iteration $\mu = (\Delta_1^1 + \Delta_1^2)/2$ and variance of all iterations $\sigma_i^2 = (\Delta_i^1 - (\Delta_i^1 + \Delta_i^2)/2)^2 + (\Delta_i^2 - (\Delta_i^1 + \Delta_i^2)/2)^2$.

$$\tau_t = \frac{\mu}{\sqrt{\frac{1}{5}\sum_{i=1}^{5}\sigma_i^2}} \tag{1.17}$$

1.4.2 Multiple Model Evaluation

Evaluate k algorithms on N datasets. Let r_i be the mean rank of algirithm i on all datasets, then use Friedman test

$$\begin{aligned}\tau_{\chi^2} &= \frac{12N}{k(k+1)}(\sum_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4}) \\ \tau_F &= \frac{(N-1)\tau_{\chi^2}}{N(k-1)-\tau_{\chi^2}}\end{aligned} \tag{1.18}$$

If the hypothesis that all algorithms performance equivalently is denied, we use post-hoc test to compare the algorithms pairwise

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{1.19}$$

1.4.3 Bias and Variance

Expectation of learning model is

$$\overline{f}(x) = \mathbb{E}_D[f(x;D)] \tag{1.20}$$

variance on diverse training sets is

$$var(x) = \mathbb{E}_D[\big(f(x;D) - \overline{f}(x)\big)^2] \tag{1.21}$$

while noise is

$$\varepsilon^2 = \mathbb{E}_D[(y_D - y)^2] \tag{1.22}$$

Different between expectation and input is

$$bias^2(x) = \big(\overline{f}(x) - y\big)^2 \tag{1.23}$$

Generalized error is the combination of bias, variance and noise.

$$E(f;D) = bias^2(x) + var(x) + \varepsilon^2 \tag{1.24}$$

Bias-variance dilemma: bias dominates generalized error when learning is inadequate, causing underfitting;

	bias	variance	noise
Measures	Fitting Power	Data Disturb	Learning Difficulty

Table 1.1: Essence

variance dominates generalized is adequate, causing overfitting.

Chapter 2

Mathmatical Background

2.1 Useful Inequalities

2.1.1 Jensen Inequality

For convex function $f(x)$,

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)] \tag{2.1}$$

2.1.2 Hoeffding Inequality

For indenpendent random variables $x_1, \ldots, x_m \in [0,1]$, $\forall \varepsilon > 0$,

$$\begin{aligned} P\left(\frac{1}{m}\sum_{i=1}^{m} x_i - \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}[x_i] \geq \varepsilon\right) \leq \exp(-2m\varepsilon^2) \\ P\left(\left|\frac{1}{m}\sum_{i=1}^{m} x_i - \frac{1}{m}{}_{i=1}^{m}\sum_{i=1}^{m}\mathbb{E}(x_i)\right| \geq \varepsilon\right) \leq \exp(-2m\varepsilon^2) \end{aligned} \tag{2.2}$$

2.1.3 McDiarmid Inequality

For indenpendent random variables $x_1, \ldots, x_m \in [0,1]$, if $\forall 1 \leq i \leq m$, f satisfies

$$\sup_{x_1,\ldots,x_m,x_i'} |f(x_1,\ldots,x_m) - f(x_1,\ldots,x_{i-1},x_i',\ldots,x_m)| \leq c_i \tag{2.3}$$

, $\forall \varepsilon > 0$,

$$\begin{aligned} P(f(x_1,\ldots,x_m) - \mathbb{E}[f(x_1,\ldots,x_m)] \geq \varepsilon) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_i c_i^2}\right) \\ P(|f(x_1,\ldots,x_m) - \mathbb{E}[f(x_1,\ldots,x_m)]| \geq \varepsilon) \leq \exp\left(-\frac{2\varepsilon^2}{\sum_i c_i^2}\right) \end{aligned} \tag{2.4}$$

2.2 Lagrange Multipliers

Lagrange Multipliers transform a optimization problem with d variables and k constraints to an unconstrained optimization problem with $d + k$ variables.

2.2.1 Equality Constraint

$$\begin{gathered} \min_x f(x) \\ \textit{subject to } g(x) = 0 \end{gathered} \tag{2.5}$$

where

$$\begin{aligned} \forall x \in g(x), \nabla g(x) \text{ is orthogonal with } g(x) = 0 \\ \exists x^*, \nabla f(x^*) \text{ is orthogonal with } g(x) = 0 \end{aligned} \tag{2.6}$$

Hence, at optimal x^*, the direction of $\nabla f(x)$ and $\nabla g(x)$ must be consistant or opponent.

$$\nabla f(x^*) + \lambda \nabla g(x^*) = 0 \tag{2.7}$$

where $\lambda \neq 0$ is Lagrange Multiplier. Then the constraint can be removed by defining Lagrange Function and set its partial derivative to zero

$$\begin{aligned} L(x, \lambda) = f(x) + \lambda g(x) \\ let \ \frac{\partial L(x, \lambda)}{\partial \lambda} = 0 \\ \Rightarrow g(x) = 0 \end{aligned} \tag{2.8}$$

2.2.2 Inequality Constraint

$$\begin{aligned} \min_x f(x) \\ subject\ to\ h_i(x) = 0 (i = 1, \dots, m) \\ g_j(x) \leq 0 (j = 1, \dots, n) \end{aligned} \tag{2.9}$$

The optimal value p^* of the problem is

$$p^* = \inf\{f(x) | h_i(x) = 0, i = 1, \dots, m, g_j(x) \leq 0, j = 1, \dots, n\} \tag{2.10}$$

Employ Lagrange Multipliers $\lambda = (\lambda_1, \dots, \lambda_m), \mu = (\mu_1, \dots, \mu_n)$,

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{n} \mu_j g_j(x) \tag{2.11}$$

According to Karush-Kuhn-Tucker (KKT), we obtain

$$\begin{cases} g_j(x) \leq 0 \\ \mu_j \geq 0 \\ \mu_j g_j(x) = 0 \end{cases} \tag{2.12}$$

Hence, we can consider the optimization problem into the primal problem and the dual problem. The dual function $\mathbb{R}^{m \times n} \to \mathbb{R}$ is

$$\begin{aligned} \Lambda(\lambda, \mu) &= \inf_{x \in \mathbb{D}} L(x, \lambda, \mu) \\ &= \inf_{x \in \mathbb{D}} \left(f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{j=1}^{n} \lambda_j g_j(x) \right) \end{aligned} \tag{2.13}$$

and it is concave no matter if the primal problem is convex. Suppose $\tilde{x}$ is a feasible point for If p^* is the optimal of primal problem, the dual function gives the lower-bound of primal problem. We obtain the convex optimization problem

$$\max_{\lambda, \mu} \Lambda(\lambda, \mu) \ s.t. \ \mu \succeq 0 \tag{2.14}$$

Note that it is convex no matter if the primal problem is convex.

2.3 Computational Learning Theory

2.3.1 Prerequisites

Given $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$, $x_i \in \mathcal{X}$, $y_i \in \{+1, -1\}$, where $\mathcal{X} \sim \mathcal{D}$. Let h be a mapping from $\mathcal{X}$ to $\mathcal{Y}$, the generalized error of h is

$$E(h; \mathcal{D}) = P_{\mathbf{x} \sim \mathcal{D}}(h(\mathbf{x} \neq y)) \tag{2.15}$$

the empirical error of h is

$$\hat{E}(h;\mathcal{D}) = \frac{1}{m} sum_{i=1}^{m} \mathbb{I}(h(\mathbf{x}_i) \neq y_i) \tag{2.16}$$

For mappings h_1, h_2, the disaggrement between them is

$$d(h_1, h_2) = P_{\mathbf{x}\in\mathcal{D}}(h_1(\mathbf{x}) \neq h_2(\mathbf{x})) \tag{2.17}$$

Let ϵ denote the error parameter, namely the upperbound of $E(h)$.

2.3.2 Probably Approximately Correct

Let $c : \mathcal{X} \to \mathcal{Y}$, if $\forall(\mathbf{x}, y) \models c(\mathbf{x}) = y$, c is the aimed concept. The union of all aimed concepts forms the concept class $\mathcal{C}$. Given machine learning algorithm $\mathcal{M}$, it searches in a space of concepts, namely the hypothesis space $\mathcal{H}$ and a hypothesis $h : \mathcal{X} \to \mathcal{Y} \in \mathcal{H}$. If $c \in \mathcal{H}$, the problem is separable for (or consistent with) $\mathcal{M}$; if $c \notin \mathcal{H}$, the problem is non-separable for $\mathcal{M}$. Let δ denote confidence, for $\epsilon > 0$, $\delta < 1$, if $\exists\mathcal{M}$,

$$P(E(h) \leq \epsilon) \geq 1 - \delta \tag{2.18}$$

$\mathcal{M}$ has the capacity to identify concept class C from hypothesis space $\mathcal{H}$. Furthermore, $\forall\mathcal{D}$, let m be the number of samples from $\mathcal{D}$, if $\exists\mathcal{M}, poly(\cdot,\cdot,\cdot,\cdot)$,

$$m \geq poly(1/\epsilon, 1/\delta, size(\mathbf{x}), size(c)) \tag{2.19}$$

concept class $\mathcal{C}$ is PAC learnable for $\mathcal{M}$ in hypothesis space $\mathcal{H}$. The lowerbound of m that satisfies the inequality is the complexity of the samples. For PAC learnable $\mathcal{M}$, if its running time is

$$poly(1/\epsilon, 1/\delta, size(\mathbf{x}), size(c)) \tag{2.20}$$

$\mathcal{M}$ is the PAC learning algorithm of concept class $\mathcal{C}$.

2.3.3 Finite Hypothesis Space

Assume that the generalized error of h is greater that ϵ, for $(\mathbf{x}, y)$ randomly sampled from $\mathcal{D}$,

$$P(h(\mathbf{x}) = y) < 1 - \epsilon \tag{2.21}$$

the propablilty that h is consistent with D is

$$P((h(\mathbf{x}_1) = y_1) \wedge \cdots \wedge (h(\mathbf{x}_m), y_m)) < (1 - \epsilon)^m \tag{2.22}$$

As h generated by $\mathcal{M}$ is unseen,

$$\begin{aligned} P(h \in \mathcal{H} : (E(h) > \epsilon) \wedge (\hat{E}(h) = 0) &< |\mathcal{H}|(1 - \epsilon)^m \\ &< |\mathcal{H}|e^{-m\epsilon} \end{aligned} \tag{2.23}$$

Hence,

$$\begin{aligned} |\mathcal{H}|e^{-m\epsilon} &\leq \delta \\ m &\geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right) \end{aligned} \tag{2.24}$$

This means that we need m samples to learn $\mathcal{H}$.
However, for problems that are non-separable for $\mathcal{M}$, namely $c \notin \mathcal{H}$, we exploit the empirical error to approximate generalization error when m is large enough,

$$P\big(|E(h) - \hat{E}(h)| \leq \sqrt{\frac{\ln|\mathcal{H}| + \ln(2/\delta)}{2m}}\big) \geq 1 - \delta \tag{2.25}$$

Though $\mathcal{M}$ can't learn the ϵ-approximataion of c when $c \notin \mathcal{H}$, it can find the hypothesis that minimizes the generalized error in $\mathcal{H}$

$$\arg\min_{h\in\mathcal{H}} E(h) \tag{2.26}$$

which is agnostic learning. Let $\epsilon > 0$, $\delta < 1$, $\forall m \geq poly(1/\epsilon, 1/\delta, size(\mathbf{x})), size(c)$, if $\mathcal{H}$ can generate h from $\mathcal{H}$ that satisfies

$$P\big(E(h) - \min_{h\in\mathcal{H}} E(h') \leq \epsilon\big) \geq 1-\delta \tag{2.27}$$

$\mathcal{M}$ is agnostic PAC learnable.

2.3.4 Vapnik-Chervonenkis Dimension

VC Dimension is a metric for the complexity of hypothesis space. The growth function of $\mathcal{H}$ is the maximum number of hypothesis h generated from $\mathcal{H}$

$$\Pi_{\mathcal{H}}(m) = \max_{\{\mathbf{x}_1,\ldots,\mathbf{x}_m\}\in\mathcal{X}} |\{(h(\mathbf{x}_1),\ldots,h(\mathbf{x}_m))|h\in\mathcal{H}\}| \tag{2.28}$$

If every $h \in \mathcal{H}$ can separate D into 2 classes, i.e. $\Pi_{\mathcal{H}}(m) = 2^m$, $\mathcal{H}$ can shatter D. VC Dimension is the maximum size of D that can be shattered by $\mathcal{H}$,

$$VC(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\} \tag{2.29}$$

We can know that VC Dimension is independent with the distribution of the samples $\mathcal{D}$. In essential, let the VC Dimension of $\mathcal{H}$ be d,

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \tag{2.30}$$

Hence,

$$\Pi_{\mathcal{H}}(m) \leq \big(\frac{e\times m}{d}\big)^d \tag{2.31}$$

We obtain the boundary of generalization error from the above two conclusions

$$P\left(E(h) - \hat{E}(h) \leq \sqrt{\frac{8d\ln(2em/d) + 8\ln(4/\delta)}{m}}\right) \geq 1-\delta \tag{2.32}$$

Hence, the boundary of generalization error of $\mathcal{H}$ only relies on m and is distribution-free and data-indenpendent. When h stands

$$\hat{E}(h) = \min_{h'\in\mathcal{H}} \hat{H}(h') \tag{2.33}$$

, $\mathcal{M}$ satisfies Empirical Risk Minimization. All $\mathcal{H}$ with finite VC Dimension is agnostic PAC learnable.

2.3.5 Rademacher Complexity

Rademacher complexity takes distribution of samples into consideration, which is in contrary to VC Dimension that is distribution-free. Hence, Rademacher Complexity is appropriate for infinite hypothesis space. Let σ_i denote the Rademacher random variable, $P(\sigma_i = 1) = P(\sigma_i = -1) = 0.5$ the hypothesis that minimizes the empirical error is

$$\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(\mathbf{x}_i) \tag{2.34}$$

Hence, the empirical Rademacher complexity of $\mathcal{H}$ on $\mathcal{X}$ is

$$\hat{R}_X(\mathcal{H}) = \mathbb{E}_\sigma\big[\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(\mathbf{x}_i)\big] \tag{2.35}$$

the Rademacher complexity of $\mathcal{H}$ corresponding to $\mathcal{D}$ is

$$R_m(\mathcal{H}) = \mathbb{E}_{X\subseteq\mathcal{X}:|X|=m}\big[\hat{R}_X(\mathcal{H})\big] \tag{2.36}$$

For regression task, $\forall h \in \mathcal{H}$, the following stands with the probability of $1-\delta$:

$$\begin{aligned}\mathbb{E}[h(\mathbf{x})] &\le \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i) + 2R_m(\mathcal{H}) + \sqrt{\frac{\ln(1/\delta)}{2m}}\\ \mathbb{H}[h(\mathbf{x})] &\le \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i) + 2\hat{R}_X(\mathcal{H}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}\end{aligned} \tag{2.37}$$

For binary classification task, the following stands with the probablility of $1-\delta$:

$$\begin{aligned}E(h) &\le \hat{E}(h) + R_m(\mathcal{H}) + \sqrt{\frac{\ln(1/\delta)}{2m}}\\ E(h) &\le \hat{E}(h) + \hat{R}_D(\mathcal{H}) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}\end{aligned} \tag{2.38}$$

For hypothesis space $\mathcal{H}$, if

$$R_m(\mathcal{H}) \le \sqrt{\frac{2\ln \Pi_{\mathcal{H}}(m)}{m}} \tag{2.39}$$

, then we can obtain the generalization error from the growth function and Rademacher complexity.

$$E(h) \le \hat{E}(h) + \sqrt{\frac{2d\ln(em/d)}{m}} + \sqrt{\frac{\ln(1/\delta)}{2m}} \tag{2.40}$$

2.3.6 Stability

For $D = \{\mathbf{z}_1 = (\mathbf{x}_1, y_1), \ldots, \mathbf{z}_m = (\mathbf{x}_m, y_m)\}$, the change of D can be

- $D^{\setminus i}$ (leave-one-out): removing sample $\mathbf{z}_i$ from D.
- D^i: substituting sample $\mathbf{z}_i$ with $\mathbf{z}'_i$.

The loss function $\mathcal{L}(\mathcal{M}_D, \mathbf{z})$ describes the difference between predicted label $\mathcal{M}_D$ and groundtruth y

- Generalization Loss:
$$\mathcal{L}(\mathcal{M}, \mathcal{D}) = \mathbb{E}_{\mathbf{x}\in\mathcal{X},\mathbf{z}=(\mathbf{x},y)}[\mathcal{L}(\mathcal{M}_D, \mathbf{z})] \tag{2.41}$$
- Empirical Loss:
$$\hat{\mathcal{L}}(\mathcal{M}, \mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\mathcal{M}_D, \mathbf{z}_i) \tag{2.42}$$
- Leave-one-out Loss:
$$\mathcal{L}_{loo}(\mathcal{M}, \mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\mathcal{M}_{D^{\setminus i}}, \mathbf{z}_i) \tag{2.43}$$

For $\mathcal{M}$, if its generalization loss and leave-one-out loss satisfy

$$|\mathcal{L}(\mathcal{M}_D, \mathbf{z}) - \mathcal{L}(\mathcal{M}_{D^{\setminus i}}, \mathbf{z})| \le \beta \tag{2.44}$$

$\mathcal{M}$ satisfies β-Stability. $\forall D$, $0 \le \mathcal{L}(\mathcal{M}_D, \mathbf{z}) \le M$, $\forall m \ge 1$, the following stands with the probability of $1-\delta$:

$$\begin{aligned}\mathcal{L}(\mathcal{M}_D, \mathbf{z}) &\le \hat{\mathcal{L}}(\mathcal{M}_D, \mathbf{z}) + 2\beta + (4m\beta + M)\sqrt{\frac{\ln(1/\delta)}{2m}}\\ \mathcal{L}(\mathcal{M}_D, \mathbf{z}) &\le \mathcal{L}_{loo}(\mathcal{M}_D, \mathbf{z}) + \beta + (4m\beta + M)\sqrt{\frac{\ln(1/\delta)}{2m}}\end{aligned} \tag{2.45}$$

Ultimately, we conclude that if $\mathcal{M}$ satisfies Empirical Risk Minimization and has stability, $\mathcal{H}$ is learnable.

Chapter 3

Clustering

3.1 Clustering

Data $D = \{x_1, \ldots, x_m\}$ without label, $D = \cup_{l=1}^{k} C_l$ where C_l are clusters without intersection, we denote cluster label as $\lambda_j = \{1, 2, \ldots, k\}$.

3.2 Performance Evaluation

Clustering algorithms should maximizes intra-cluster similarity and minimizes inter-cluster similarity. Clustering result $\mathbb{C} = \{C_1, C_2, \ldots, C_k\}$, let $dist(\cdot)$ be distance between 2 samples, the mean distance between samples in C is

$$avg(C) = \frac{2}{|C|(|C|-1)} \sum_{1 \le i < j \le |C|} dist(x_i, x_j) \tag{3.1}$$

and we obtain the maximum distance between samples in C

$$d_{\max}(C) = \max_{1 \le i < j \le |C|} dist(x_i, x_j) \tag{3.2}$$

Furthermore, we measure the minimum distance and centroid distance cross clusters

$$\begin{aligned} d_{\min}(C_i, C_j) &= \min_{x_i \in C_i, x_j \in C_j} dist(x_i, x_j) \\ d_{cen}(C_i, C_j) &= dist(\mu_i, \mu_j) \end{aligned} \tag{3.3}$$

where $\mu = \frac{1}{|C|} \sum_{i=1}^{|C|} x_i$ is the centroid of the cluster. The lower DBI and the higher DI is, the better the algorithm is

$$\begin{aligned} DBI &= \frac{1}{k} \sum_{i=1}^{k} \max_{i \ne j} \left(\frac{avg(C_i) + avg(C_j)}{d_{cen}(C_i, C_j)} \right) \\ DI &= \min_{1 \le i \le k} \left\{ \min_{i \ne j} \frac{d_{\min}(C_i, C_j)}{\max_{1 \le l \le k} d_{\max}(C_l)} \right\} \end{aligned} \tag{3.4}$$

3.3 Distance

Minkowski Distance (LP Norm):

$$dist_{mk}(x_i, x_j) = \left(\sum_{u=1}^{n} (x_{iu} - x_{ju})^p \right)^{\frac{1}{p}} \tag{3.5}$$

when $p = 1$, it is Manhattan Distance (L1 Norm) and $p = 2$ (L2 Norm) is Euclidean Distance. LP Norm is appropriate for ordinal attribute, while we use Value Difference Metric (VDM) to calculate distance of ordinal attribute. Let $m_{u,a}$ denote the sum of samples with attribute a, $m_{u,a,i}$ be samples with attribute a in cluster i:

$$VDM(a,b) = \sum_{i=1}^{k} \| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \|_p^p \tag{3.6}$$

3.4 Prototype-based Clustering

3.4.1 K-means Clustering

MSE is

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2^2 \tag{3.7}$$

where $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.

Algorithm 1: K-means Clustering

```
   Data: $D = \{x_1, \dots, x_m\}$, $k$
   Result: $C = \{C_1, \dots, C_k\}$
1  while $\exists \mu_i \neq \mu_i'$ do
2      $C_i \leftarrow \phi (1 \leq i \leq k)$
3      for $j \leftarrow 1$ to $m$ do
4          $d_{ij} \leftarrow \|x_j - \mu_i\|_2^2$
5          $\lambda_j \leftarrow \arg\min_{i \in \{1,\dots,k\}} d_{ij}$
6          $C_{\lambda_j} \leftarrow C_{\lambda_j} \cup \{x_j\}$
7      for $i \leftarrow 1$ to $k$ do
8          $\mu_i' = \frac{1}{|C_i|} \sum_{x \in C_i} x$
9          if $\mu_i' \neq \mu_i$ then
10             $\mu_i \leftarrow \mu_i'$
```

3.4.2 Learning Vector Quantilization

Sample x_j is consist of n feature vectors $(x_{j1}; \dots; x_{jn})$. LVQ aims to learn n-dimensional prototype vector $\{p_1, \dots, p_q\}$, each of which denotes a cluster t_i. Note that LVQ is supervised clustering. For every p_i corresponds to a cluster space R_i, every sample in R_i is closer to p_i than any other $p_j (j \neq i)$.

3.4.3 Mixture of Gaussian

For random vector x in n-dimensional sample space, Gaussian Distribution of x is

$$p(x) = \frac{1}{2\pi^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{3.8}$$

where Σ is covariance matrix by $n \times n$. Gaussian mixture distribution is

$$p_{\mathcal{M}}(x) = \sum_{i=1}^{k} \alpha_i p(x|\mu_i, \Sigma_i) \tag{3.9}$$

Algorithm 2: Learning Vector Quantilization

```
    Data: $D = \{(x_1, y_1), \dots, (x_m, y_m)\},\ \{t_1, \dots, t_q\},\ \eta$
    Result: $\{p_1, \dots, p_q\}$
1   init $\{p_1, \dots, p_q\}$ (e.g. random sampling)
2   while $p_{i^*} \neq p'$ do
3       random sample $(x_i, y_i)$
4       $d_{ij} \leftarrow \|x_i - p_i\|_2^2$
5       $i^* \leftarrow \arg\min_{i \in \{1,\dots,q\}} d_{ij}$
6       if $y_i = t_i$ then
7           $p' \leftarrow p_{i^*} + \eta(x_j - p_{i^*})$
8       else
9           $p' \leftarrow p_{i^*} - \eta(x_j - p_{i^*})$
10      $p_{i^*} \leftarrow p'$
```

with k mixtures and α_i denotes mixture coefficient. This is solved by maximum likelihood. Generate training set $D = \{x_1, \dots, x_m\}$, $z_j = \{1, \dots, k\}$ denote unknown mixture component and its posterior is

$$\gamma_{ji} = p_{\mathcal{M}}(z_j = i|x_j) = \frac{\alpha_i p(x_j|\mu_i, \Sigma_i)}{\sum_{l=1}^{k} \alpha_i p(x_j|\mu_l, \Sigma_l)} \tag{3.10}$$

Hence, the cluster label of each sample is determined by

$$\lambda_j = \arg_{i \in \{1,\dots,k\}} \max \gamma_{ji} \tag{3.11}$$

We can solve it by E-M algorithm and Lagrangian, obtain

$$\begin{aligned} \mu_i &= \frac{\sum_{j=1}^{m} \gamma_{ji} x_j}{\sum_{j=1}^{m} \gamma_{ji}} \\ \Sigma_i &= \frac{\sum_{j=1}^{m} \gamma_{ji} (x_j - \mu_i)^T (x_j - \mu_i)}{\sum_{j=1}^{m} \gamma_{ji}} \\ \alpha_i &= \frac{1}{m} \sum_{j=1}^{m} \gamma_{ji} \end{aligned} \tag{3.12}$$

In E-step we calculate teh posterior of z_j and in M-step we update the parameters of the model.

3.5 Density-based Clustering

- ε-Neighbourhood:
$$N_\varepsilon(x_j) = \{x_i \in D | dist(x_i, x_j) \leq \varepsilon\} \tag{3.13}$$
- x_j is Core-Object:
$$|N_\varepsilon(x_j)| \geq MinPts \tag{3.14}$$
- Directly density-reachable(x_i, x_j):
$$(x_j \in N_\varepsilon(x_j)) \wedge (x_i isCore - Object) \tag{3.15}$$
- Density-reachable(x_i, x_j):
$$(p_1 = x_i) \wedge (p_n = x_j) \wedge (p_i\ is\ density - reachabletop_{i+1}) \Rightarrow density - reachable(x_i, x_j) \tag{3.16}$$
- Density-connected(x_i, x_j):
$$density - reachable(x_i, x_k) \wedge density - reachable(x_k, x_j) \Rightarrow density - connected(x_i, x_j) \tag{3.17}$$

Algorithm 3: E-M Gaussian Mixture Model

```
Data: $D = \{x_1, \dots, x_m\},\ k$
Result: $C = \{C_1, \dots, C_k\}$
1  while not terminated do
2      for $j \leftarrow 1$ to $m$ do
3          $\gamma_{ji} = p_{\mathcal{M}}(z_j = i|x_j)$
4      for $i \leftarrow 1$ to $k$ do
5          $\mu_i' \leftarrow \frac{\sum_{j=1}^m \gamma_{ji} x_j}{\sum_{j=1}^m \gamma_{ji}}$
6          $\Sigma_i' \leftarrow \frac{\sum_{j=1}^m \gamma_{ji}(x_j - \mu_i)^T (x_j - \mu_i)}{\sum_{j=1}^m \gamma_{ji}}$
7          $\alpha_i' \leftarrow \frac{1}{m} \sum_{j=1}^m \gamma_{ji}$
8  $C_i \leftarrow \phi$
9  for $j \leftarrow 1$ to $m$ do
10     $\lambda_j = \arg_{i \in \{1,\dots,k\}} \max \gamma_{ji}$
11     $C_i \leftarrow C_i \cup \{x_j\}$
```

Algorithm 4: Density-based Spatial Clustering of Applications with Noise

```
Data: $D = \{x_1, \dots, x_m\},\ \varepsilon,\ MinPts$
Result: $C = \{C_1, \dots, C_k\}$
1  $\Omega \leftarrow \phi$ //Core Object
2  for $j \leftarrow 1$ to $m$ do
3      $N_\varepsilon(x_j)$
4      if $|N_\varepsilon(x_j)| \geq MinPts$ then
5          $\Omega \leftarrow \Omega \cup \{x_j\}$
6  $k \leftarrow 0$
7  $\Gamma \leftarrow D$ // Unvisited data
8  while $\Omega \neq \phi$ do
9      $\Gamma_{old} \leftarrow \Gamma$
10     $Q < o >, o \in \Omega$
11     $\Gamma \leftarrow \Gamma \setminus \{o\}$while $Q \neq \phi$ do
12         $q \leftarrow Q[0]$
13         if $|N_\varepsilon(q)| \geq MinPts$ then
14             $\Delta \leftarrow N_\varepsilon(q) \cap \Gamma$
15             $Q[|Q|] \leftarrow \Delta$
16             $\Gamma \leftarrow \Gamma \setminus \Delta$
17     $k \leftarrow k + 1$
18     $C_k \leftarrow \Gamma_{old} \setminus \Gamma$
19     $\Omega \leftarrow \Omega \setminus C_k$
```

3.6 Hierachical Clustering

For two different clusters, we can calculate their maximum inter-cluster distance d_{max}, minimum inter-cluster distance d_{min} and average inter-cluster distance

$$d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{w \in C_i, z \in C_j} dist(w, z) \tag{3.18}$$

Agglomerative Nesting iteratively clusters the samples by minimizing the inner-cluster distance. The implementation of AGNES depends on the genre of distance: 1) d_{max} complete linkage; 2) d_{min} single linkage; 3)

d_{avg} average linkage.

Algorithm 5: Agglomerative Nesting

```
   Data: $D = \{x_1, \ldots, x_m\}$, $d_{min/max/avg}$, $k$
   Result: $C = \{C_1, \ldots, C_k\}$
1  for $j \leftarrow 1$ to $m$ do
2      $C_j \leftarrow x_j$
3  for $i \leftarrow 1$ to $m$ do
4      for $j \leftarrow 1$ to $m$ do
5          $M[i][j]) \leftarrow d(C_i, C_j)$
6          $M[j][i] \leftarrow M[i][j]$
7  $q \leftarrow m$
8  while $q > k$ do
9      $(C_{i^*}, C_{j^*}) \leftarrow \arg_{C_{i^*} \in C, C_{j^*} \in C} \min_{i^* \neq j^*} d(C_{i^*}, C_{j^*})$
10     $C_{i^*} \leftarrow C_{i^*} \cup C_{j^*}$for $j \leftarrow j^* + 1$ to $q$ do
11         $C_j \leftarrow C_{j-1}$
12     for $j \leftarrow 1$ to $q - 1$ do
13         $M[i^*][j] \leftarrow d(C_{i^*}, C_j)$
14         $M[j][i^*] \leftarrow M[i^*][j]$
15     $q \leftarrow q - 1$
```

Chapter 4

Metric Learning

4.1 k-NN

k-Nearest Neighbor is not trained explicititly. In contrast to eager learning, k-NN is lazy learning. The probability of error is (i.e. x and its nearest neighbor z are classified into different classes)

$$P(error) = 1 - \sum_{x \in \mathcal{Y}} P(c|x)P(c|z) \tag{4.1}$$

Let $c^* = \arg\max_{c \in \mathcal{Y}} P(c|x)$ be the optimal performance of Bayesian classifier,

$$\begin{aligned} P(error) &= 1 - \sum_{x \in \mathcal{Y}} P(c|x)P(c|z) \\ &\simeq 1 - \sum_{x \in \mathcal{Y}} P^2(c|x) \\ &\leq 1 - P^2(c^*|x) \\ &= (1 + P(c^*|x))(1 - P(c^*|x)) \leq 2(1 - P(c^*|x)) \end{aligned} \tag{4.2}$$

Hence, we can prove that the error rate of k-NN is not larger than twice of the error rate of optimal Bayesian classifier.

4.2 Multiple Dimensional Scaling

Curse of dimensionality makes sample sparse and distance difficult to measure. Hence, we exploit dimension reduction to embed the high dimensional sample space into a subspace. The aim of MDS is

$$\mathbf{D} \in \mathbb{R}^{m \times m} \Rightarrow \mathbf{Z} \in \mathbb{R}^{d' \times m} \tag{4.3}$$

where $\|z_i - z_j\|_2^2 = dist_{ij}$. Let $\mathbf{B} = \mathbf{Z}^T\mathbf{Z} \in \mathbb{R}^{m\times m}$, $b_{ij} = z_i^T z_j$,

$$\begin{aligned}
dist_{ij}^2 &= b_{ii} + b_{jj} - 2b_{ij} \\
dist_{i.}^2 &= \frac{1}{m}\sum_{j=1}^{m} dist_{ij}^2 \\
dist_{.j}^2 &= \frac{1}{m}\sum_{i=1}^{m} dist_{ij}^2 \\
dist_{..}^2 &= \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} dist_{ij}^2 \\
b_{ij} &= -\frac{1}{2}(dist_{ij}^2 - dist_{i.}^2 - dist_{.j}^2 + dist_{..}^2) \\
\mathbf{B} &= \mathbf{V}^T\mathbf{\Lambda}\mathbf{V} \\
\tilde{\mathbf{\Lambda}} &= diag(\lambda_1, \ldots, \lambda_d') \; where \; d' << d \\
\mathbf{Z} &= \tilde{\mathbf{\Lambda}}^{1/2}\mathbf{V}^T
\end{aligned} \tag{4.4}$$

Hence, we denote the linear transform of high dimensional space as

$$\mathbf{Z} = \mathbf{W}^T\mathbf{X} \tag{4.5}$$

4.3 Principle Component Analysis

PCA outputs a hyperplane which minimizes the its distance to all samples and maximizes the variance of the projections of samples on it.

$$\begin{aligned}
&\max_{\mathbf{W}} tr(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) \\
&subject\ to\ \mathbf{W}^T\mathbf{W} = \mathbf{I} \\
&\Rightarrow \\
&\mathbf{X}\mathbf{X}^T\mathbf{W} = \lambda\mathbf{W}
\end{aligned} \tag{4.6}$$

Select the d' largest eigenvalues of the covariance matrix $\mathbf{X}\mathbf{X}^T$ and construct the projection matrix by their corresponding eigenvectors

$$\mathbf{W} = (w_1, w_2, \ldots, w_{d'}) \tag{4.7}$$

We can calculate the d' from the predifined reconstruction threshold by

$$\frac{\sum_{i=1}^{d'} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq t \tag{4.8}$$

PCA densifies and denoises the samples.

4.4 Kernelized PCA

Linera dimensional reduction partially destroys the original lower-dimensional space, while Kernelized PCA retrieves its intrinsic space.

$$\begin{aligned}
\left(\sum_{i=1}^{m} z_i z_i^T\right)\mathbf{W} &= \lambda\mathbf{W} \\
\mathbf{W} &= \sum_{i=1}^{m} z_i \frac{z_i^T\mathbf{W}}{\lambda} \\
&= \sum_{i=1}^{m} z_i \alpha_i
\end{aligned} \tag{4.9}$$

Define the kernel matrix $\mathbf{K}$ with kernel functions as elements

$$\kappa(x_i, x_j) = \phi(x_i)^T \phi(x_j) \tag{4.10}$$

the result is d' largest eigenvalue of $\mathbf{K}$. The projection of new sample x is

$$z_j = w_j^T \phi(x) = \sum_{i=1}^{m} \alpha_i^j \kappa(x_i, x) \tag{4.11}$$

4.5 Manifold Learning

Manifold partially shares the properties of Euclidean space, i.e. it is locally homeomorphic with Euclidean space.

4.5.1 Isometric Mapping

Distance between two points in manifold is the distance of geodesic. Since Manifold is partially homeomorphic with Euclidean space, distance between nearest neighbors is L2 distance. Hence, we construct a adjacency graph of neighbors and geodesic between two points in manifold is shortest path of the graph.

4.5.2 Locally Linear Embedding

LLE maintains the linear relationship locally, as x_i can be reconstructed by the linear combination of its neighbor samples

$$x_i = w_{ij}x_j + w_{ij}x_k + w_{ij}x_l \tag{4.12}$$

Let Q_i be the label set of neighbor samples of x_i, z_i be the corresponding coordinates in lower dimension,

$$\begin{aligned} &\min_{w_1,\dots,w_m} \sum_{i=1}^{m} \|x_i - \sum_{j\in Q_i} w_{ij}x_j\|_2^2 \\ &\textit{subject to} \sum_{j\in Q_i} w_{ij} = 1 \\ &\min_{z_1,\dots,z_m} \|z_i - \sum_{j\in Q_i} w_{ij}z_j\|_2^2 \end{aligned} \tag{4.13}$$

Let $\mathbf{Z} = (z_1, \dots, z_m) \in \mathbb{R}^{d' \times m}$, the optimization problem can be rewritten as

$$\begin{aligned} &\mathbf{M} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W}) \\ &\min_{\mathbf{Z}} tr(\mathbf{Z}\mathbf{M}\mathbf{Z}^T) \\ &\textit{subject to } \mathbf{Z}\mathbf{Z}^T = \mathbf{I} \end{aligned} \tag{4.14}$$

4.6 Metric Learning

Euclidean distance regards feature subspaces of samples are orthogonal, metric learning aims to learn the relationship between feature subspaces. Mahalanobis distance is dimension-weighted Euclidean distance in essential (i.e. dimension with more significance contributes more to the distance)

$$dist_{mah}^2(x_i, x_j) = (x_i - x_j)^T \mathbf{M}(x_i - x_j) = \|x_i - x_j\|_M^2 \tag{4.15}$$

where $\mathbf{M}$ is the weight matrix to learn.

Algorithm 6: Locally Linear Embedding

```
   Data: $D = \{x_1, \dots, x_m\},\ k,\ d'$
   Result: $\mathbf{Z} = (z_1, \dots, z_m) \in \mathbb{R}^{d' \times m}$
 1 for $i \leftarrow 1$ to $m$ do
 2     $Q_i \leftarrow N_k(x_i)$
 3     if $j \in Q_i$ then
 4         solve $w_{ij}$
 5     else
 6         $w_{ij} \leftarrow 0$
 7 $\mathbf{M} = (\mathbf{I} - \mathbf{W})^T(\mathbf{I} - \mathbf{W})$
 8 solve $\mathbf{Z}$
```

4.6.1 Neighbourhood Component Analysis

The classification result of x_i is determined by its neighbor x_j

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|_M^2\right)}{\sum_l \exp\left(-\|x_i - x_j\|_M^2\right)} \tag{4.16}$$

The correct ratio of Left-One-Out on entire data set is

$$\sum_{i=1}^{m} p_{ij} = \sum_{i=1}^{m} \sum_{j \in \Omega_i} p_{ij} \tag{4.17}$$

where Ω_i denotes the sample set of same class with x_i. Let $\mathbf{M} = \mathbf{P}\mathbf{P}^T$, $\mathbf{M}$ can be solved by

$$\min_{\mathbf{P}}\ 1 - \sum_{i=1}^{m} \sum_{j \in \Omega_i} \frac{\exp\left(-\|\mathbf{P}^T x_i - \mathbf{P}^T x_j\|_M^2\right)}{\sum_l \exp\left(-\|\mathbf{P}^T x_i - \mathbf{P}^T x_j\|_M^2\right)} \tag{4.18}$$

4.6.2 Background Knowledge Embedding

Employ must-link constraint set $\mathcal{M}$ for samples with high similarity and cannot-link constraint set $\mathcal{C}$ for samples with low similarity, we can solve $\mathbf{M}$ by optimizing

$$\begin{aligned} \min_{\mathbf{M}} & \sum_{x_i, x_j \in \mathcal{M}} \|x_i - x_j\|_M^2 \\ \textit{subject to} & \sum_{x_i, x_j \in \mathcal{C}} \|x_i - x_j\|_M^2 \geq 1 \\ & \mathbf{M} \succeq 0 \end{aligned} \tag{4.19}$$

Chapter 5

Linear Model

Consider sample x with attributes $(x^1; x^2; \dots; x^d)$, linear model learns the weights of each attributes.

$$f(x) = w^1x^1 + w^2x^2 + \cdots + w^dx^d + b = \sum_{i=1}^{d} w^ix^i + b \tag{5.1}$$

Note that each attribute of each sample should be wrote as x_m^d and we omit the underscore here for succinctness. The weight vector $\mathbf{w} = (w^1; w^2; \dots; w^d)$ has the comprehensibility that feature corresponding to larger weight contriutes more to the entire property of the sample.

5.1 Linear Regression

Trained by $D = \{(x_1, y_1), \dots, (x_m, y_m)\}$, linear regression learns a function that

$$f(x_i) = wx_i + b \simeq y_i \tag{5.2}$$

and it is learned by MSE

$$(w^*, b^*) = \arg\min_{(w,b)} \sum_{i=1}^{m} \big(f(x_i) - y_i\big)^2 \tag{5.3}$$

We can solve the MSE by least square method that set partial derivative of w and b respectively, obtaining close form solution

$$\begin{aligned} w &= \frac{\sum_{i=1}^m y_i(x_i - \frac{1}{m}\sum_{i=1}^m x_i)}{\sum_{i=1}^m x_i^2 - (\frac{1}{m}\sum_{i=1}^m x_i)^2} \\ b &= \frac{1}{m}\sum_{i=1}^{m}(y_i - \mathbf{w}x_i) \end{aligned} \tag{5.4}$$

for multivariate linear regression, we can rewrite w, x, y in vector form and obtain

$$\hat{\mathbf{w}}^* = \arg\min_{\hat{\mathbf{w}}^*} (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^*)^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^*) \tag{5.5}$$

where $\mathbf{M}$ is a matrix with m rows whose rows are feature vectors concatenated by 1 $(x_1, \dots, x_d, 1)$.

$$w^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^Ty \tag{5.6}$$

when $\mathbf{X}^T\mathbf{X}$ is full-ranked. However, in most cases this cannot be satisfied and there are multiple solutions of $\hat{\mathbf{w}}$. Here we employ **Regularization Term** to introduce bias to select the solution

$$(w^*, b^*, \lambda) = \arg\min_{(w,b)} \sum_{i=1}^{m} \big(f(x_i) - y_i\big)^2 + \lambda\|w\|_2^2 \tag{5.7}$$

where λ is the regularization parameter.

5.2 Logistic Regression

Sigmoid function is a surrogate function for unit-step function in binary classification task. Note that logistic regression is not a regression model, but a classification model.

$$y(z) = \frac{1}{1+e^{-z}} \tag{5.8}$$

Hence

$$y(x) = \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{5.9}$$

We estimate w, b by log-maximum likelihood. Let $\hat{x_i}$ be $(x_i, 1)$,

$$\begin{aligned}
\mathcal{L}(\mathbf{w}, b) &= \sum_{i=1}^{m} \ln p(y_i|x_i; \mathbf{w}, b) \\
\beta &= \mathbf{w}^T x + b \\
\mathcal{L}(\beta) &= \sum_{i=1}^{m} \Big(-y_i \beta^T \hat{x_i} + \ln\big(1 + e^{\beta^T \hat{x_i}}\big) \Big) \\
\beta^* &= \arg\min_{\beta} \mathcal{L}(\beta)
\end{aligned} \tag{5.10}$$

This can be solved by convex optimization like Gauss-Newton, Gradient Descent. GN at $t+1$ iteration is

$$\begin{aligned}
\beta^{t+1} &= \beta^t = (\frac{\partial^2 \mathcal{L}(\beta)}{\partial\beta\partial\beta^T})^{-1} \frac{\partial \mathcal{L}(\beta)}{\partial \beta} \\
\frac{\partial \mathcal{L}(\beta)}{\partial \beta} &= -\sum_{i=1}^{m} \hat{x_i}(y_i - p_1(\hat{x_i}; \beta)) \\
\frac{\partial^2 \mathcal{L}(\beta)}{\partial\beta\partial\beta^T} &= \sum_{i=1}^{m} \hat{x_i}\hat{x_i}^T p_1(x_i; \beta)(1 - p_1(x_i; \beta))
\end{aligned} \tag{5.11}$$

where $p_1(\hat{x_i}; \beta) = p(y = 1|\hat{x_i}, \beta)$ is the posterior of class label $y = 1$.

5.3 Linear Discriminant Analysis

The intuition of LDA is maximizing the inter-class projection distance and minimizing the intra-class projection distance on a linear plane. This correspond to minimizing the covariance of samples of same class and maximizing the difference of mean of different classes.

$$J = \frac{\|\mathbf{w}^T \mu_{\mathbf{0}} - \mathbf{w}^T \mu_{\mathbf{1}}\|_2^2}{\mathbf{w}^T \mathbf{\Sigma}_0 \mathbf{w} + \mathbf{w}^T \mathbf{\Sigma}_0 \mathbf{w}} \tag{5.12}$$

Within-class scatter matrix is the combination of covariance matrces with respect to class 0 and 1

$$\mathbf{S}_w = \mathbf{\Sigma}_0 + \mathbf{\Sigma}_1 = \sum_{x \in X_0} (\mathbf{x} - \mu_{\mathbf{0}})(\mathbf{x} - \mu_{\mathbf{0}})^T + \sum_{x \in X_1} (\mathbf{x} - \mu_{\mathbf{1}})(\mathbf{x} - \mu_{\mathbf{1}})^T \tag{5.13}$$

Between-class scatter matrix is

$$\mathbf{S}_b = (\mathbf{x} - \mu_{\mathbf{0}})(\mathbf{x} - \mu_{\mathbf{1}})^T \tag{5.14}$$

Hence

$$\begin{aligned}
&\min_{w} \mathbf{w}^T \mathbf{S}_b \mathbf{w} \\
&\textit{subject to } \mathbf{w}^T \mathbf{S}_w \mathbf{w} = 1 \\
&\mathbf{w} = \mathbf{S}_w^{-1}(\mu_{\mathbf{0}} - \mu_{\mathbf{1}})
\end{aligned} \tag{5.15}$$

We can generalize the conclusions to N multi-classification.

$$\mathbf{S}_w = \sum_{i=1}^{N} \mathbf{S}_{w_i}, \mathbf{S}_b = \sum_{i=1}^{N} \mathbf{S}_{b_i} \tag{5.16}$$

The solution is optimizing

$$\max_{\mathbf{W}} \frac{tr(\mathbf{W}^T \mathbf{S}_b \mathbf{W})}{tr(\mathbf{W}^T \mathbf{S}_w \mathbf{W})} \tag{5.17}$$

5.4 Multiple Classification

We divide multiple classification into series of binary classification problems.

- One vs. One: Binary Classification between every two objective classes, $\frac{N(N-1)}{2}$ tasks. The final label is highest voted classifier from each task.
- One vs. Rest: Use C_i as positive samples and all other classes as negative samples, N tasks. The final label is positive output from all tasks.
- Many vs. Many: Generalized form of OvO and OvR. We split entire dataset into M sets and train M classifiers. Error Correcting Output Codes (ECOC) tries to make error-tolerant encoding of each class and predicted labels of each classifier. The final result is the class whose encoding has the highest similarity with the output encoding.

5.5 Class-inbalance

It is common that negative samples is far more than positive samples. We exploit rescaling to handle this problem

- Undersampling: discard some negative samples to obtain $m^+ = m^-$. This may cause the lose of significant knowledge. It is appropriate for ensemble learning since the global information can be maintained by integrating all the learners.
- Oversampling: augment positive samples by interpolation or data augmentation techinques such as adding disturbution or noise.
- Threshold-moving:

$$y \ is \ positive \ when \ \frac{y'}{1-y'} = \frac{y}{1-y} \times \frac{m^-}{m^+} > 1 \tag{5.18}$$

Chapter 6

Decision Tree

Decision Tree make binary classification on each property at each node. The generation of decision tree is recursive, where there are 3 return conditions: 1) $\forall x_i, x_j \in D_v, C(x_i) = C_(x_j)$; 2) $A = \phi \vee (\forall x_i, x_j \in D_v, A(x_i) = A(x_j))$; 3) $D_v = \phi$. Note that 1) exploits posterior of v while 2) uses prior of the super node of v.

Algorithm 7: Generating Decision Tree

```
Data: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_m, y_m)\},\ A = \{a^1, \dots, a^d\}$
Result: Decision Tree with node
function DecisionTree():
new node
if $\forall x_i, x_j \in D, C(x_i) = C_(x_j)$ then
    $label_{node} \leftarrow C$
    return
if $A = \phi \vee (\forall x_i, x_j \in D, A(x_i) = A(x_j))$ then
    $label_{node} \leftarrow \arg_C \max_{C \in D} label_C$
    return
$a^* \leftarrow \arg_{a \in A} \min Gini_{index}(D, a)$
for each $a^*_v$ in $a^*$ do
    $node \leftarrow node - > branch$
    $D_v \leftarrow \{x_i | x_i \in D(i = 1, \dots, m), a^*(x_i)\}$ if $D_v = \phi$ then
        $label_{branch} \leftarrow \arg_C \max_{C \in D} label_C$
        return
    $branch \leftarrow DecisionTree(D_v, A \setminus \{a^*\})$
```

6.1 How to find a^*_v

6.1.1 Information Entropy

Let p_k be the proportion of samples within class $k \in \{1, \dots, |\mathcal{Y}|\}$ in dataset D. Then information entropy of D is

$$Ent(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k \tag{6.1}$$

where $0 \leq Ent(D) \leq \log_2 |\mathcal{Y}|$. Consider attribute a with V assignments, if we split the dataset by attribute, the attribute containing more samples contributes more, namely we can get higher information purity if we split data with the attribute. This is defined by infromation gain of attribute a in dataset D

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v) \tag{6.2}$$

The gain ratio corresponds to the intrinsic IV of the attribute

$$\begin{aligned} IV(a) &= -\sum_{v=1}^{V} \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|} \\ Gain_{ratio}(D,a) &= \frac{Gain(D,a)}{IV(a)} \end{aligned} \tag{6.3}$$

Gini index measures the purity of data as well

$$Gini(D) = \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2 \tag{6.4}$$

Gini ratio is

$$Gini_{ratio} = \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v) \tag{6.5}$$

6.2 Prune

- Pre-pruning: In the process of generating the tree, estimate the generalized performance of the branch, if it doesn't contribute much then prune the branch.
- Post-pruning: After the tree is generated, we test the precision of each branch and prune the branches that lowers the precision of the subtree.

Chapter 7

Neural Network

7.1 Neuron

A neuron takes input from other n neurons

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \tag{7.1}$$

where w_i is the connection weight between this neuron and neuron i and x_i is the output of neuron i, θ is the activation threshold and can be regarded as the connection weight of a node outputting "-1". Hence, neural network is a model with many weighted linear combination.

7.2 Perceptron

Perceptron gets signal via input layer and output via threshold logic unit. Only the neurons in output layer can be activated. The weight is learned by the error of output $\hat{y}$

$$\begin{aligned} w_i &\leftarrow w_i + \Delta w_i \\ \Delta w_i &= \eta(y - \hat{y})x_i \end{aligned} \tag{7.2}$$

where η is learning rate. Multi-Layer Perceptron has hidden layers between input layer and output layer.

7.3 Back Propagation

Consider $D\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m)\}, \mathbf{x_i} \in \mathbb{R}^d, \mathbf{y_i} \in \mathbb{R}^l$, the neural network must have d input neurons, l output neurons and q hidden neurons. The input x_i from neuron i at input layer for neuron h at hidden layer is

$$\alpha_h = \sum_{i=1}^{d} w_{ih} x_i \tag{7.3}$$

The input b_h from neuron h at hidden layer with threshold γ_h for neuron j at output layer is

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h \tag{7.4}$$

The output for $(\mathbf{x_k}, \mathbf{y_k})$ at output neuron j is

$$\hat{y}_j^k = f(\beta_j - \theta_j) \tag{7.5}$$

MSE is

$$E_k = \frac{1}{2}\sum_{j=1}^{l}\left(\hat{y}_k - y_k\right)^2 \tag{7.6}$$

Hence, the network propagates the error back to neuron h by

$$\Delta w_{hj} = -\eta\frac{\partial E_k}{\partial w_{ij}} \tag{7.7}$$

the partial derivative is obtained by chain rule

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \times \frac{\partial \hat{y}_j^k}{\partial \beta_j} \times \frac{\partial \beta_j}{\partial w_{ij}} \tag{7.8}$$

As

$$\begin{aligned}\frac{\partial \beta_j}{\partial w_{ij}} &= b_h \\ g_j &= -frac\partial E_k \partial \hat{y}_j^k \times \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= \hat{y}_j^k(1-\hat{y}_j^k)(y_j^k - \hat{y}_j^k) \\ e_h &= b_h(1-b_h)\sum_{j=1}^{l} w_{hj}g_j\end{aligned} \tag{7.9}$$

Similarly,

$$\begin{aligned}\Delta\theta_j &= -\eta g_j \\ \Delta w_{ih} &= \eta e_h x_i \\ \gamma_h &= -\eta e_h\end{aligned} \tag{7.10}$$

Note that the training objective function is minimizing

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k \tag{7.11}$$

where E is accumulated error. There are two ways to alleviate overfitting

1. Early-Stop: Tune the parameters on training set and estimate the error on validation set. If the error on training set decreases yet error on validation set increases, we should stop training and set the parameters corresponding to the iteration that minimizes the error on validation set.

2. Regularization: Add a term that decribes the complexity of network w_i^2 (connection weight and threshold)

$$E = \lambda\frac{1}{m}\sum_{k=1}^{m} E_k + (1-\lambda)\sum_i w_i^2 \tag{7.12}$$

7.4 Local Minima

If

$$\exists\varepsilon > 0 \forall(\mathbf{w};\theta) \in \{(\mathbf{w};\theta) | \|(\mathbf{w};\theta) - (\mathbf{w}^*;\theta^*)\|_2^2 \leq \varepsilon\} \models E(\mathbf{w};\theta) \geq E(\mathbf{w}^*;\theta^*) \tag{7.13}$$

then $E(\mathbf{w}^*;\theta^*)$ is the local minima. Only if

$$\forall(\mathbf{w};\theta) \in \{(\mathbf{w};\theta)\} \models E(\mathbf{w};\theta) \geq E(\mathbf{w}^*;\theta^*) \tag{7.14}$$

then $E(\mathbf{w}^*;\theta^*)$ is the global minima.

Chapter 8

Support Vector Machine

8.1 Formulation

We aim to find a hyperplane that not only divides different classes of samples, but also has the best generalization on unseen data.

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{8.1}$$

where $\mathbf{w} = (w_1; \ldots; w_d)$ is the normal vector of the hyperplane, b is the distance between the hyperplane and original point. A hyper plane is denoted as $(\mathbf{w}, b)$ and the distance between an attribute point and $(\mathbf{w}, b)$ is

$$r = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|} \tag{8.2}$$

Let

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq +1, \, y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq +1, \, y_i = -1 \end{cases} \tag{8.3}$$

where $i \in \{1, \ldots, m\}$The samples closest to the hyperplane satisfies the equality, which are support vectors. The sum of distance between two inter-class vectors is

$$\gamma = \frac{2}{\|\mathbf{w}\|} \tag{8.4}$$

and γ is the margin. Hence, the desired hyperplane comes with the maximum margin

$$\begin{gathered} \max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \\ \textit{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{gathered} \tag{8.5}$$

The optimization can be rewritten as

$$\begin{gathered} \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \\ \textit{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{gathered} \tag{8.6}$$

8.2 Solve SVM

SVM can be solved as a convex quadratic programming. Firtly we obtain the dual form of primal problem

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i \Big(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\Big) \tag{8.7}$$

where $\alpha = (\alpha_1, \dots, \alpha_m)$. Set the partial of L to $\mathbf{w}$ and b respectively and obtain

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \\ 0 &= \sum_{i=1}^{m} \alpha_i y_i \\ b &= \frac{1}{|S|} \sum_{s \in S} \Big(y_s - \sum_{s \in S} y_i \alpha_i \mathbf{x}_i^T \mathbf{x}_s \Big) \end{aligned} \tag{8.8}$$

where $S = \{i | \alpha_i > 0, i = 1, 2, \dots, m\}$ is the set of all support vectors. Hence, we obtain the dual problem

$$\begin{aligned} \max_{\alpha} & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \textit{subject to} & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned} \tag{8.9}$$

Finally

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b \tag{8.10}$$

Note that the KKT is

$$\begin{cases} \alpha_i \geq 0 \\ y_i(f(\mathbf{x}_i) - 1) \geq 0 \\ \alpha_i(y_i f(\mathbf{x}_i) - 1) = 0 \end{cases} \tag{8.11}$$

8.3 Kernelization

Since some sample spaces are not linearly separatible, we map the samples into a higher dimensional feature space to obtain linearly separatible. Let $\phi(\cdot)$ be the mapping function

$$\begin{aligned} & f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \\ & \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \\ \textit{subject to } & y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 \\ & \max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \\ \textit{subject to } & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \end{aligned} \tag{8.12}$$

Denote $\kappa(\cdot, \cdot)$ in original sample space as the operation that is equivalent to the inner product of two vectors in the feature space

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{8.13}$$

We can substitute $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$, then the support vector expansion is

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \kappa(\mathbf{x}, \mathbf{x}_i) + b \tag{8.14}$$

The kernel matrix is a symmetric matrix whose elements are kernel functions, retrieving the symmetric property of kernel functions

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \dots & \kappa(x_1, x_m) \\ \vdots & \ddots & \vdots \\ \kappa(x_m, x_1) & \dots & \kappa(x_m, x_m) \end{pmatrix}$$

Hence, selection of kernel functions determines the performance of SVM. There are some common kernels

- Linear Kernel:
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \tag{8.15}$$
- Polynomial Kernel:
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d \tag{8.16}$$
- Gaussian Kernel:
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \quad \sigma > 0 \tag{8.17}$$
- Laplacian Kernel:
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\sigma}\right) \quad \sigma > 0 \tag{8.18}$$
- tanh Kernel:
$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \beta \mathbf{x}_i^T \mathbf{x}_j + \theta \quad \beta > 0,\ \theta < 0 \tag{8.19}$$

Let κ_1, κ_2 be kernel functions then

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{z}) &= \gamma_1 \kappa_1(\mathbf{x}, \mathbf{z}) + \gamma_2 \kappa_2(\mathbf{x}, \mathbf{z}) \quad \gamma_1 > 0,\ \gamma_2 > 0 \\ \kappa_1 \otimes \kappa_2(\mathbf{x}, \mathbf{z}) &= \kappa_1(\mathbf{x}, \mathbf{z}) \kappa_2(\mathbf{x}, \mathbf{z}) \\ \kappa(\mathbf{x}, \mathbf{z}) &= g(\mathbf{x}) \kappa_1(\mathbf{x}, \mathbf{z}) g(\mathbf{z}) \end{aligned} \tag{8.20}$$

are kernel functions.

8.4 Soft Margin

Soft margin is tolerant to some samples that don't satisfy the constraint

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \tag{8.21}$$

Since 0/1 loss is neither convex nor continuous, usually we use surrogate loss

$$\begin{aligned} \mathcal{L}_{hinge}(z) &= \max(0, 1 - z) \\ \mathcal{L}_{exp}(z) &= \exp(-z) \\ \mathcal{L}_{log}(z) &= \log(1 + \exp(-z)) \end{aligned} \tag{8.22}$$

We introduce slack variable $\xi \geq 0$ and the objective function is rewritten as

$$\begin{aligned} \min_{\mathbf{w}, b} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \mathcal{L}(\mathbf{w}^T \mathbf{x}_i + b) &= \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \\ \textit{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) &\geq 1 - \xi_i \\ \xi &\geq 0 \end{aligned} \tag{8.23}$$

the Lagrangian of objective function is

$$L(\mathbf{w}, b, \alpha, \mu, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i + \sum_{i=1}^{m} \alpha_i (1 - \xi_i - y_i(\mathbf{w}^T \mathbf{x}_i + b)) + \sum_{i=1}^{m} \mu_i \xi_i \tag{8.24}$$

set the partial derivatives to zero

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \\ 0 &= \sum_{i=1}^{m} \alpha_i y_i \\ C &= \alpha_i + \mu_i \\ \max_{\alpha} & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \textit{subject to} & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C \end{aligned} \tag{8.25}$$

KKT is

$$\begin{cases} \alpha_i \ge 0 \\ 1 - \xi_i - y_i(\mathbf{w}^T \mathbf{x}_i + b) \le 0 \\ \alpha_i(1 - \xi_i - y_i(\mathbf{w}^T \mathbf{x}_i + b)) = 0 \\ \mu_i \ge 0 \\ -\xi_i \le 0 \\ \mu_i \xi_i = 0 \end{cases} \tag{8.26}$$

Note that we can write the objective function in a more general form

$$\min_f \ \Omega(f) + C \sum_{i=1}^{m} \mathcal{L}(f(\mathbf{x}_i), y_i) \tag{8.27}$$

where $\Omega(f)$ is structural risk corresponding to the complexity of the model, which is the regularization term. The regularization term can be L2, L1 and L0 norm, among which the first one makes **w** dense while the other two make **w** sparse.

8.5 Support Vector Regression

The intuition of SVR is we can be tolerant for the output with some error ε. The loss function is ε-insensitive loss

$$\mathcal{L}_\varepsilon(z) = \max(0, |z| - \varepsilon) \tag{8.28}$$

with slack varibles $\xi_i, \hat{\xi}_i$, the objective is

$$\begin{aligned} \min_{\mathbf{w}, b, \xi_i, \hat{\xi}_i} & \ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} (\xi_i + \hat{\xi}_i) \\ \textit{subject to} & \ f(\mathbf{x}_i) - y_i \le \xi_i + \varepsilon \\ & f(\mathbf{x}_i) - y_i \ge \hat{\xi}_i + \varepsilon \\ & \xi_i \ge 0, \hat{\xi}_i \ge 0 \end{aligned} \tag{8.29}$$

The objective can be solved by Lagrange Dual. Solution omitted.

Chapter 9

Bayesian Classifier

9.1 Bayesian Decision Theory

Consider $\mathcal{Y} = \{c_1, \ldots, c_N\}$ classes, λ_{ij} the loss of misclassifying sample c_j to c_i, then we obtain the expectation loss $R(c_i|\mathbf{x})$ (i.e. conditional risk) for classifying $\mathbf{x}$ to c_i:

$$R(c_i|\mathbf{x}) = \sum_{j=1}^{N} \lambda_{ij} P(c_j|\mathbf{x}) \tag{9.1}$$

where $P(c_j|\mathbf{x})$ is posterior. Let h be the discriminate criteria that minimizes global risk (i.e. all conditional risks)

$$R(h) = \mathbb{E}_{\mathbf{x}}[R(h(\mathbf{x})|\mathbf{x})] \tag{9.2}$$

Bayesian decision rule is selecting the label of sample $\mathbf{x}$ that minimizes the conditional risk

$$h^* = \arg_{c\in\mathcal{Y}} \min R(c|\mathbf{x}) \tag{9.3}$$

where h^* is the Bayes Optimal Classifier and its corresponding globa risk $R(h^*)$ is Bayes risk. Since

$$R(c) = 1 - P(c|\mathbf{x}) \tag{9.4}$$

the classifier with minimal classification error is

$$h^*(x) = \arg_{c\in\mathcal{Y}} P(c|\mathbf{x}) \tag{9.5}$$

9.2 Naive Bayes Classifier

The Bayes Theory bridges posterior $P(c|\mathbf{x})$ with prior $P(c)$

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^{d} P(\mathbf{x}_i|c) \tag{9.6}$$

where $P(\mathbf{x}_i|c)$ is class-conditional probability on every property of $\mathbf{x}$, namely likelihood between $\mathbf{x}_i$ and c. The discriminate criteria of Naive Bayes is

$$h_{nb}(\mathbf{x}) = \arg_{c\in\mathcal{Y}} \max \prod_{i=1}^{d} P(x_i|c) \tag{9.7}$$

where

$$P(c) = \frac{|D_c|}{|D|} \tag{9.8}$$

for properties with discrete value,

$$P(x_i|c) = \frac{|D_{c,i}|}{|D_c|} \tag{9.9}$$

for properties with continuous value, let $P(x_i|c) \sim \mathcal{N}(\mu_{c,i}, \sigma^2_{c,i})$,

$$P(x_i|c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma^2_{c,i}}\right) \tag{9.10}$$

We usually employ Laplacian Correction to smooth the prior and evidence

$$\begin{aligned}\hat{P}(c) &= \frac{|D_c| + 1}{|D| + N} \\ \hat{P}(x_i|c) &= \frac{|D_{c,i}| + 1}{|D_c| + N_i}\end{aligned} \tag{9.11}$$

where N is the sum of potential classes and N_i is the number of possible assignments of property i.

9.3 Semi-naive Bayes Classifier

In contrary to considering mutual dependence between any two properties in Naive Bayes, Semi-naive Bayes Classifier only estimates the dependence of one other property for any property (One-Dependence Estimator, ODE).

$$P(c|\mathbf{x}) = P(c)\prod_{i=1}^{d} P(x_i|c, pa_i) \tag{9.12}$$

where property x_i in dependent on pa_i. A straightforward idea is assuming a all properties rely on a property "super-parent" (SPODE), and make sure the property of super-parent by cross validation. A more sophiscated method is Average One-dependence Estimator, which tries every property as the super-parent and the SPODE with sufficient training data

$$P(c|\mathbf{x}) = \sum_{i=1, |D_{x_i}| \geq m'}^{d} P(c, x_i) \prod_{j=1}^{d} P(x_j|c, x_i) \tag{9.13}$$

where

$$\begin{aligned}\hat{P}(c, x_i) &= \frac{|D_{c,x_i}| + 1}{|D| + N_i} \\ \hat{P}(x_j|c, x_i) &= \frac{|D_{c,x_i,x_j|+1}}{|D_{c,x_i}| + N_j}\end{aligned} \tag{9.14}$$

9.4 Bayesian Network

9.4.1 Introduction

A Bayesian Network $B = (G, \Theta)$, where G is a directed acylic graph that $\pi_i \to x_i$ and Θ is the set of $\theta_i = P(x_i|\pi_i)$. Assuming every attribute is independent with its non-descent attributes, the joint distribution of attribute $x_1, \ldots, x_d$ is

$$P_B(x_1, \ldots, x_d) = \prod i = 1^m P(x_i|\pi_i) = \prod i = 1^m \theta_{x_i|\pi_i} \tag{9.15}$$

V-structure decribes the scene that two independent attributes x_a, x_b are both the dependence of attribute x_c. When $P(x_c)$ is given, x_a, x_b are dependent yet if $P(x_c)$ stays unknown, x_a, x_b are indenpendent and this is marginal independence. In D-separation, we find all V-structures, add an undirected edge between two dependence nodes and substitute all edges with undirected edges. This operation is moralization and the generated undirected graph is moral-graph.

9.4.2 Training

Bayesian Network exploits Minimal Description Length criteria to determine the score function. Given training set D, score function is

$$\begin{aligned} s(B|D) &= f(\theta)|B| - LL(B|D) \\ LL(B|D) &= \sum_{i=1}^{m} \log P_B(\mathbf{x_i}) \end{aligned} \tag{9.16}$$

where $f(\theta)$ is describes the bits for encoding parameter θ. When G is fixed, minimizing $s(B|D)$ is equivalent to the maximum likelihood estimation of Θ,

$$\theta_{x_i|\pi_i} = \hat{P}_D(x_i|\pi_i) \tag{9.17}$$

9.4.3 Inference

Let $Q = \{Q_1 = q_1, \dots, Q_n = q_n\}$ denoted the querying variables, $E = \{E_1 = e_1, \dots, E_k = e_k\}$ are evidence variables and the objective of the query is calculating $P(Q = q|E = e)$. Inference is done by Gibbs sampling.

$$P(Q = q|E = e) \simeq \frac{n_q}{T} \tag{9.18}$$

the prior is the proportion of observations equal to query in sampling iteration t. Gibbs sampling is a random walk in a subspace of the joint distriution of all variables which is consistent to $E = e$.

Algorithm 8: Gibbs Sampling

```
Data: $B = (G, \Theta)$, $T$, $E(e)$, $Q(q)$
Result: $P(Q = q|E = e) \simeq \frac{n_q}{T}$
1  $n_q \leftarrow 0$
2  $q^0 \leftarrow random()$
3  for $t \leftarrow 1$ to $T$ do
4      for $Q_i \in Q$ do
5          $Z \leftarrow E \cup Q \setminus \{Q_i\}$
6          $z \leftarrow e \cup q^{t-1} \setminus \{q_i\}$
7          $P_B(Q_i|Z = z)$
8          $q_i^t \leftarrow assignment\ for\ Q_i\ in\ P_B(Q_i|Z = z)$
9          $q^t \leftarrow q^{t-1} \cup q_i^t \setminus q_i^{t-1}$
10     if $q^t = q$ then
11         $n_q \leftarrow n_q + 1$
```

9.5 Expectation-Maximization Algorithm

Let $\mathbf{X}$ be observed variables, $\mathbf{Z}$ be unobserved variables (i.e. latent variables), maximizing the log-likelihood is maximizing

$$LL(\mathbf{\Theta}|\mathbf{X}, \mathbf{Z}) = \ln P(\mathbf{X}, \mathbf{Z}|\mathbf{\Theta}) \tag{9.19}$$

Since $\mathbf{Z}$ are latent variables, we can maximize the marginal likelihood by calculating the expectation of $\mathbf{Z}$

$$LL(\mathbf{\Theta}|\mathbf{X}) = \ln P(\mathbf{X}|\mathbf{\Theta}) = \ln \sum_{\mathbf{Z}} P(\mathbf{X}, \mathbf{Z}|\mathbf{\Theta}) \tag{9.20}$$

For iteration t of the E-M algorithm,

- Expectation step: inference the distribution of $\mathbf{Z}$ by $\mathbf{\Theta}^t$, calculate the expectation of log-likelihood on the latent variables

$$Q(\mathbf{\Theta}|\mathbf{\Theta}^t) = \mathbb{E}_{\mathbf{Z}|\mathbf{X},\mathbf{\Theta}^t} LL(\mathbf{\Theta}|\mathbf{X}, \mathbf{Z}) \tag{9.21}$$

- Maximization step:
$$\mathbf{\Theta}^{t+1} = \arg_{\mathbf{\Theta}} \max Q(\Theta|\Theta)^t \tag{9.22}$$
the E-M algorithms uses current parameters to inference latent variables and updates the parameters to maximize likelihood iteratively until converge to a local optima.

Chapter 10

Ensemble Learning

Ensemble Learning is a collection of some individual learners, obtaining better generalization than weak classifiers. Consider a binary classification task and the error rate of a component learner is ε, h is hypothesis and f is groundtruth

$$P(h_i(\mathbf{x}) \neq f(\mathbf{x})) = \varepsilon \tag{10.1}$$

assume that the error rates of individual learners are independent, then according to Hoeffding Inequality,

$$\begin{aligned} P(H(\mathbf{x}) \neq f(\mathbf{x})) &= \sum_{k=0}^{\lfloor \frac{T}{2} \rfloor} \binom{T}{k} (1-\varepsilon)^k \varepsilon^{T-k} \\ &\leq \exp\Big(-\frac{1}{2}T(1-2\varepsilon)^2\Big) \\ H(\mathbf{x}) &= sign\Big(\sum_{i=1}^{T} h_i(\mathbf{x})\Big) \end{aligned} \tag{10.2}$$

this means that the error ratio of ensemble learner decreases exponentially as the number of individual learners increases.

10.1 Boosting

Boosting learns a base learner from initial data and shifs the data distribution according to the testing results, then learns the next learner on the modified data, which enhances the samples that lead the former learner to error. This procedure performs iteratively until it learns T individual learners. In essential the ensemble learner is a linear combination of weak learners

$$H(\mathbf{x}) = \sum_{t=1}^{T} a_t h_t(\mathbf{x}) \tag{10.3}$$

to minimize the exponential loss function

$$\mathcal{L}_{exp}(H|\mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}} \exp\Big(-f(\mathbf{x})H(\mathbf{x})\Big) \tag{10.4}$$

Set the partial derivative of H to zero, we obtain

$$H(\mathbf{x}) = \frac{1}{2} \ln \frac{P(f(x)=1)|\mathbf{x}}{P(f(x)=-1)|\mathbf{x}} \tag{10.5}$$

Hence

$$sign(H(\mathbf{x})) = \arg_{y\in\{-1,1\}} \max P(f(x)=y|\mathbf{x}) \tag{10.6}$$

and it approaches the error rate of Bayes Optimal Classifier. When the former classifier h_t based on $\mathcal{D}_t$ is generated, its weight α_t minimizes the loss function

$$\mathcal{L}_{exp}(\alpha_t h_t|\mathcal{D}_t) = \mathbb{E}_{x\sim\mathcal{D}_t} \exp\Big(-f(\mathbf{x})\alpha_t h_t(\mathbf{x})\Big) = (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t} \tag{10.7}$$

Set the partial derivative of α_t to zero, we obtain

$$\alpha_t = \frac{1}{2}\ln\Big(\frac{1-\varepsilon_t}{\varepsilon}\Big) \tag{10.8}$$

An ideal individual learner h_t has the capacity to correct all the errors made by H_{t-1}

$$\begin{aligned} \mathcal{L}_{exp}(H_{t-1}+h_t|\mathcal{D}) &= \mathbb{E}_{x\sim\mathcal{D}} \exp\Big(-f(\mathbf{x})(H_{t-1}(\mathbf{x})+h_t(\mathbf{x}))\Big) \\ h_t(\mathbf{x}) &= \arg_h \min \mathcal{L}_{exp}(H_{t-1}+h|\mathcal{D}) \\ &= \arg_h \min \mathbb{E}_{x\sim\mathcal{D}_\sqcup}[\mathbb{I}(f(\mathbf{x})\neq h(\mathbf{x}))] \end{aligned} \tag{10.9}$$

Hence, an ideal individual learner minimizes its error on $\mathcal{D}_t$, and the training data disturbution for next iteration is its residual approximation

$$\mathcal{D}_{t+1} = \mathcal{D}_t \exp\Big(-f(\mathbf{x}\alpha_t h_t(\mathbf{x}))\Big)\frac{\mathbb{E}_{x\sim\mathcal{D}}\exp\Big(-f(\mathbf{x})H_{t-1}(\mathbf{x})\Big)}{\mathbb{E}_{x\sim\mathcal{D}}\exp\Big(-f(\mathbf{x})H_t(\mathbf{x})\Big)} \tag{10.10}$$

Since, Boosting enhances decreasing bias instead of variance, it is appropriate for improving performance of

Algorithm 9: AdaBoost

```
  Data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, base learner $\mathcal{M}, T$
  Result: $H(\mathbf{x}) = sign\Big(\sum_{t=1}^{T} a_t h_t(\mathbf{x})\Big)$
1 $\mathcal{D}_1(\mathbf{x}) \leftarrow \frac{1}{m}$
2 for $t \leftarrow 1$ to $T$ do
3     $h_t \leftarrow \mathcal{M}(\mathcal{D}, \mathcal{D})_t$
4     $\varepsilon \leftarrow P_{\mathbf{x}\sim\mathcal{D}_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$
5     if $\varepsilon > 0.5$ then
6         terminate
7     $\alpha_t \leftarrow \ln\frac{1-\varepsilon_t}{\varepsilon_t}$
8     $\mathcal{D}_{t+1} \leftarrow \frac{1}{\mathbf{Z}_t}\mathcal{D}_t(\mathbf{x})\exp\Big(-f(\mathbf{x})\alpha_t h(\mathbf{x})\Big)$
```

base learner with poor generalization.

10.2 Random Forest

10.2.1 Bagging

Remind Bootstrap Sampling aforementioned, we obtain T training sets with the proportion of 63.2% samples in entire dataset. Hence, the remaining 36.8% samples are used for out-of-bag (oob) estimate to test generalization, let H^{oob} denote the estimation on $\mathbf{x}$ of model didn't trained on $\mathbf{x}$

$$H^{oob}(\mathbf{x}) = \arg_{y\in\mathcal{Y}} \max \sum_{t=1}^{T} \mathbb{I}(h_t(\mathbf{x}) = y)\mathbb{I}(x \notin D_t) \tag{10.11}$$

the generalized error of oob is

$$\varepsilon^{oob} = \frac{1}{|D|} \sum_{\mathbf{x}, y \in D} \mathbb{H}^{oob}(\mathbf{x}) \neq y \tag{10.12}$$

In contrary to Adaboost that enhances minimizing bias, Bagging focus on minimizing variance.

```
Algorithm 10: Bagging
  Data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, base learner $\mathcal{M}, T$
  Result: $H(\mathbf{x}) = \arg_{y \in \mathcal{Y}} \max \sum_{t=1}^{T} \mathbb{I}(h_t(\mathbf{x}) = y)$
1 for $t \leftarrow 1$ to $T$ do
2     $h_t \leftarrow \mathcal{M}(D, \mathcal{D}_{Bootstrap})$
```

10.2.2 Random Forest

Random Forest is an extension of Bagging that uses decision tree as base learner. Consider $a \in \{a_1, \ldots, a_d\}$ is the property of current node, convention decision tree selects the most appropriate property for splitting while RF selects a subset with k properties randomly and select the best attribute in the subset. Normally $k = \log_2 d$. RF not only add pertubation into samples, but into attributes, which improve the generalization of Bagging.

10.3 Combination of Individual Learners

We aim to ensemble T individual learners $\{h_1, \ldots, h_T\}$, we can combine them via

10.3.1 Average

- Average:

$$H(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^{T} h_i(\mathbf{x}) \tag{10.13}$$

- Weighted Average:

$$H(\mathbf{x}) = \sum_{i=1}^{T} w_i h_i(\mathbf{x}) \tag{10.14}$$

10.3.2 Voting

Let $(h_i^1(\mathbf{x}); \ldots; h_i^N(\mathbf{x}))$ be the output of the model on sample $\mathbf{x}$, where N is the number of class labels $\{c_1, \ldots, c_N\}$ for the classification task.

- Mojority Voting:

$$H(\mathbf{x}) = \begin{cases} c_j, & if \ \sum_{i=1}^{T} h_i^j(\mathbf{x}) > \frac{1}{2} \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(\mathbf{x}) \\ reject, & otherwise \end{cases} \tag{10.15}$$

- Plurality Voting:

$$H(\mathbf{x}) = c_{\arg_j \max \sum_{i=1}^{T} h_i^j(\mathbf{x})} \tag{10.16}$$

- Weighted Voting:

$$H(\mathbf{x}) = c_{\arg_j \max \sum_{i=1}^{T} w_i h_i^j(\mathbf{x})} \tag{10.17}$$

10.3.3 Meta-Learning

In contrary to learn a model directly predicts the distribution of data, a meta-learner learns how to combine the individual learners, thus a meta-learner is a secondary learner. Meta-learner is trained on the output of weak learners with the supervision of initial labels. Meta Learning aims to enable the model "learning to learn". In another word, meta learning doesn't learn the relationships in training data, but learns the metric to set appropriate configuration for specific learning task. As Stacking exploits a series of individual learning

Algorithm 11: Stacking

```
  Data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, base learner $\mathcal{M}_1, \ldots, \mathcal{M}_T$, meta − learner $\mathcal{M}$
  Result: $H(\mathbf{x}) = h'(h_1(\mathbf{x}), \ldots, h_t(\mathbf{x}))$
1 for $t \leftarrow 1$ to $T$ do
2     $h_t \leftarrow \mathcal{M}_t(D)$
3 $D' \leftarrow \phi$
4 for $i \leftarrow 1$ to $m$ do
5     for $t \leftarrow 1$ to $T$ do
6         $z_{it} \leftarrow h_t(\mathbf{x}_i)$
7     $D' \leftarrow D' \cup \{(z_{i1}, \ldots, z_{iT}), y_i\}$
8 $h' = \mathcal{M}(D')$
```

models, Model-Agnostic Meta Learning (MAML) is a technique that makes deep nerual networks gain fast adaption to new testing tasks with only a few training samples. We train the model on diverse tasks with a few examples in each task. MAML doesn't learn the best model to fit existing tasks, but learns the model easily to achieve convergence on unseen tasks. Let θ be parameters of the model and $\mathcal{T}_\rangle$ denote task i, the updating of θ over examples in task i is

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{10.18}$$

where $\mathcal{L}$ is the loss function for the task. After evaluating all examples in a task, MAML updates the parameter globally.

$$\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{10.19}$$

Note that the loss we calculate this time uses the parameters tuned on the corresponding task, thus distinct MAML from conventional gradient-based training. In essential, MAML uses the gradient on diverse tasks to update the parameters of the learner.

10.4 Diversity

10.4.1 Ambiguity

For sample $\mathbf{x}$, the Ambiguity of learner h_i is

$$A(h_i|\mathbf{x}) = \big(h_i(\mathbf{x}) - H(\mathbf{x})\big)^2 \tag{10.20}$$

and the ambiguity of the ensemble learner is

$$\overline{A}(h|\mathbf{x}) = \sum_{i=1}^{T} w_i \big(h_i(\mathbf{x}) - H(\mathbf{x})\big)^2 \tag{10.21}$$

Ambiguity decribes the inconsistency of individual learners on $\mathbf{x}$. The MSEs are

$$\begin{aligned} E(h_i|\mathbf{x}) &= \big(f(\mathbf{x}) - h_i(\mathbf{x})\big)^2 \\ E(H|\mathbf{x}) &= \big(f(\mathbf{x}) - H(\mathbf{x})\big)^2 \end{aligned} \tag{10.22}$$

where f is groundtruth function of the regression. The weighted average error of individual learner is

$$\overline{E}(h_i|\mathbf{x}) = \sum_{i=1}^{T} w_i E(h_i|\mathbf{x}) \tag{10.23}$$

Hence

$$\overline{A}(h|\mathbf{x}) = \overline{E}(h|\mathbf{x}) - E(h|\mathbf{x}) \tag{10.24}$$

Let $p(x)$ be the density of $\mathbf{x}$, the generalized error and ambiguity on entire data set are

$$\begin{aligned} E_i &= \int E(h_i|\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ A_i &= \int A(h_i|\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ E &= \int E(H|\mathbf{x})p(\mathbf{x})d\mathbf{x} \end{aligned} \tag{10.25}$$

Hence

$$\begin{aligned} \overline{E} &= \sum_{i=1}^{T} w_i E_i \\ \overline{A} &= \sum_{i=1}^{T} w_i A_i \\ \overline{A} &= \overline{E} - E \end{aligned} \tag{10.26}$$

this is error-ambiguity decomposition.

10.4.2 Diversity Measure

Consider two individual learners h_i, h_j, there are $P(2,2) = 4$ assignments of the result that where a, b, c, d

	$h_i = +1$	$h_i = -1$
$h_j = +1$	a	c
$h_j = -1$	b	c

denote the number of the qualified samples. There are several measures of diversity

- Disagreement measure:
$$dis_{ij} = \frac{b+c}{m} \tag{10.27}$$
- Correlation coefficient:
$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a+b)(a+d)(c+d)(b+d)}} \tag{10.28}$$
- Q-statistic:
$$Q_{ij} = \frac{ad - bc}{ad + bc} \tag{10.29}$$
- κ-statistic: p_1 is the probability that the two learners is consistent with each other and p_2 is the probability that the two learners make same assumption accidently
$$\begin{aligned} \kappa &= \frac{p_1 - p_2}{1 - p_2} \\ p_1 &= \frac{a+d}{m} \\ p_2 &= \frac{(a+b)(a+c) + (c+d)(b+d)}{m^2} \end{aligned} \tag{10.30}$$

10.4.3 Enhance Diversity

- Pertubation on sampling.
- Pertubation on attributes. Random Subspace algorithms selects several subspaces of attributes from sample space and trainings each learner on each subspace of attribute. This operation prunes the redundancy attributes.

```
Data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, base learner $\mathcal{M}$, $T$, number of subspaces $d'$
Result: $H(\mathbf{x}) = \arg_{y \in \mathcal{Y}} \max \mathbb{I}(h_t(Map_{\mathcal{F}_t}(\mathbf{x})) = y)$
1 for $t \leftarrow 1$ to $T$ do
2     $\mathcal{F}_t \leftarrow RandomlySelect(D, d')$
3     $D_t \leftarrow Map_{\mathcal{F}_t}(D)$
4     $h_t \leftarrow \mathcal{M}(D_t)$
```

- Pertubation on output representation.
- Pertubation on parameters of the model.

Chapter 11

Sparse Representation

11.1 Feature Selection

11.1.1 Filtering

Relief algorithm exploits correlative statistic to measure the significance of the subspace of feature space. Consider there are $|\mathcal{Y}|$ classes in D, sample x_i belongs to class k, then it finds the sample $x_{i,nh}$ (nearest-hit) closest to x_i in the same class and the sample $\mathbf{x}_{i,l,nm}(l = 1, \ldots, |\mathcal{Y}| \wedge (l \neq k))$ (nearest-miss) closest to x_i in every other class. The correlative statistic corresponding to attribute j is

$$\delta^j = \sum_i -(x_i^j - x_{i,nh}^j)^2 + \sum_{l \neq k} \big(p_l (x_i^j - x_{i,l,nm}^j)^2\big) \tag{11.1}$$

where p_l is the proportion of samples in class l in entire datas.

11.1.2 Las Vegas Wrapper

Las Vegas Wrapper takes the performance of the learner as evaluation metric. Note that LVW calculates the error only considering feature subset A' on entire dataset, and we want it to decrease the error on A or to smallerize the size of feature subset.

Algorithm 12: Las Vegas Wrapper

```
   Data: $D$, attribute set $A$, learner $\mathcal{M}$, maximum iteration $T$
   Result: feature subset $A^*$
 1 $E \leftarrow \infty$
 2 $A^* \leftarrow |A|$
 3 $d \leftarrow A$
 4 $t \leftarrow 0$
 5 while $t < T$ do
 6     randomly generate feature subset $A'$
 7     $d' \leftarrow |A'|$
 8     $E' \leftarrow CrossValidation(\mathcal{M}(D^{A'}))$
 9     if $(E' < E) \vee ((E' = E) \wedge (d' < d))$ then
10         $t = 0$
11         $E \leftarrow E'$
12         $A^* \leftarrow A'$
13         $d \leftarrow d'$
14     else
15         $t \leftarrow t + 1$
```

11.1.3 L1 Regularization

As aforementioned in Regression, we add an L2 norm to the MSE loss function to alleviate overfitting

$$\min_{\mathbf{w}} \sum_{i=1}^{m} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2 \tag{11.2}$$

which is Ridge Regression (or Tikhonov Regression). We can also add an L1 norm

$$\min_{\mathbf{w}} \sum_{i=1}^{m} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_1 \tag{11.3}$$

this is Least Absolute Shrinkage and Selection Operator (LASSO), obtaining a more sparse solution (i.e. less non-zero elements). Though L0 norm generates most sparse solution, it is discrete and hard to optimize. We solve LASSO

$$\min_{\mathbf{x}} f(\mathbf{x}) + \lambda \|\mathbf{x}\|_1 \tag{11.4}$$

via Proximal Gradient Descent, if $f(\mathbf{x})$ is derivative and satisfies L-Lipschitz that exists an constant $L > 0$

$$\|\nabla f(\mathbf{x}' - \mathbf{x})\|_2^2 \leq L \|\mathbf{x}' - \mathbf{x}\|_2^2 \; (\forall \mathbf{x}', \mathbf{x}) \tag{11.5}$$

the second-order taylor approximation of $\mathbf{x}_k$

$$\hat{f}(\mathbf{x}) \simeq \frac{L}{2} \left\| \mathbf{x} - \left(\mathbf{x}_k - \frac{1}{L} \nabla f(\mathbf{x}_k) \right) \right\|_2^2 + const \tag{11.6}$$

hense, we both descent $f(\mathbf{x})$ via gradient and minimizes L1 norm simultaneously,

$$\begin{aligned} \mathbf{x}_{k+1} &= \arg_{\mathbf{x}} \min \left\| \mathbf{x} - \left(\mathbf{x}_k - \frac{1}{L} \nabla f(\mathbf{x}_k) \right) \right\|_2^2 + \lambda \|\mathbf{x}\|_1 \\ \mathbf{z} &= \mathbf{x} - \frac{1}{L} \nabla f(\mathbf{x})_k \\ \mathbf{x}_{k+1} &= \arg_{\mathbf{x}} \min \|\mathbf{x} - \mathbf{z}\|_2^2 + \lambda \|\mathbf{x}\|_1 \end{aligned} \tag{11.7}$$

since each element of x is orthogonal, we have close solution

$$x_{k+1}^i = \begin{cases} z^i - \frac{\lambda}{L}, & z^i > \frac{\lambda}{L} \\ 0, & |z^i| < \frac{\lambda}{L} \\ z^i + \frac{\lambda}{L}, & z^i < -\frac{\lambda}{L} \end{cases} \tag{11.8}$$

11.2 Dictionary Learning

Dictionary Learning is consist of a procedure of sparse encoding. Let $\mathbf{B} \in \mathbb{R}^{d \times k}$ be the dictionary matrix and k is the vocabulary of the dictionary, $\alpha_i \in \mathbb{R}^k$ is the sparse representation of sample $\mathbf{x}_i \in \mathbb{R}^d$, dictionary learning is

$$\min_{\mathbf{B}, \alpha_i} \sum_{i=1}^{m} \|\mathbf{x}_i - \mathbf{B}\alpha_i\|_2^2 + \lambda \sum_{i=1}^{m} \|\alpha_i\|_1 \tag{11.9}$$

we follow the idea solving LASSO to solve it

$$\min_{\alpha_i} \|\mathbf{x}_i - \mathbf{B}\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \tag{11.10}$$

then we fix α_i to update $\mathbf{B}$.

$$\min_{\mathbf{B}} \|\mathbf{X} - \mathbf{B}\mathbf{A}\|_F^2 \tag{11.11}$$

where $\mathbf{X} \in \mathbb{R}^{d \times m}$ and $\mathbf{A} \in \mathbb{R}^{k \times m}$. $\|\cdot\|_F$ is the Frobenius norm, namely the matrix form of L2 norm.

11.3 Compressed sensing

Consider discrete signal $\mathbf{x}$ of length m, we sample by Nyquist and obtain $\mathbf{y}$ of length n where $n << m$, $\Phi \in \mathbb{R}^{n\times m}$ is sampling matrix,

$$\mathbf{y} = \mathbf{\Phi x} \tag{11.12}$$

Let $\mathbf{\Psi} \in \mathbb{R}^{m\times m}$ that $\mathbf{x} = \mathbf{\Psi s}$, then

$$\mathbf{y} = \mathbf{\Phi x \Psi s} = \mathbf{As} \tag{11.13}$$

where $\mathbf{A} \in \mathbb{R}^{n\times m}$ is a dictionary that transforms $\mathbf{x}$ to a sparse representation s. The core of compressed sensing is reconstruction. Restricted Isometry Property describes that for $\mathbf{A} \in \mathbb{R}^{n\times m}$ where $n << m$, if

$$\forall \mathbf{A}_k \in \mathbb{R}^{n\times k},\ \mathbf{s}\ \exists \delta_k \in (0,1)\ \models\ (1-\delta_k)\|\mathbf{s}\|_2^2 \leq \|\mathbf{A}_k\mathbf{s}\|_2^2 \leq (1+\delta_k)\|\mathbf{s}\|_2^2 \tag{11.14}$$

then $\mathbf{A}$ satisfies k-RIP. We can retrive $\mathbf{s}$ then $\mathbf{x}$ from $\mathbf{y}$ by optimizing

$$\begin{aligned} &\min_{\mathbf{s}} \|\mathbf{s}\|_0 \\ &\textit{subject to } \mathbf{y} = \mathbf{As} \end{aligned} \tag{11.15}$$

since L0 norm is not continuous, it can be relaxed to L1 norm and solved via the idea of solving LASSO.

Chapter 12

Semi-supervised Learning

12.1 Generative Models

This method is based on the assumption that all data are generated by a single potential model. Consider sample $\mathbf{x}$ and its groundtruth label $y \in \mathcal{Y} = \{1, \ldots, N\}$, the data is generated by the distribution

$$p(\mathbf{x}) = \sum_{i=1}^{N} p(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i) \tag{12.1}$$

where $p(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i)$ is the probability of $\mathbf{x}$ in in gaussian mixture model i. Denote $f(\mathbf{x}) \in \mathcal{Y}$ as the prediction of f on $\mathbf{x}$, $\mathbf{\Theta} \in \{1, \ldots, N\}$ is the gaussian mixture component of $\mathbf{x}$. The maximum prior of $\mathbf{x}$ is

$$f(\mathbf{x}) = \arg_{j \in \mathcal{Y}} \sum_{i=1}^{N} p(y = j|\mathbf{\Theta} = i, \mathbf{x}) p(\mathbf{\Theta} = i|\mathbf{x}_i) \tag{12.2}$$

where

$$p(\mathbf{\Theta} = i|\mathbf{x}) = \frac{p(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i)}{\sum_{i=1}^{N} p(\mathbf{x}|\mu_i, \mathbf{\Sigma}_i)} \tag{12.3}$$

is the probability of $\mathbf{x}$ generated by gaussian mixture model i. We calculate the probabilities of unlabeled sample belonging to all gaussian mixture components at E-step and update the parameters at M-step iteratively until convergence. The we use the learned parameters to pridect the classes of unlabeled samples. However, generative models come with it's fair share of drawbacks that they are based on the assumption that the distribution of model is equivalent to the groundtruth distribution, which is rarely seen in real scenes.

12.2 Semi-Supervised Support Vector Machine

S3VM tries to find a hyperplane that both separates labelled samples and goes through the space with low density. Transductive SSVM assigns labels on all unlabelled samples and searches for a hyperplane with maximum margin. The prediction is the final assignment of unlabelled data. Given $D_l = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$, $D_u = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_m\}$, the aim of TSVM is giving predicted label $\hat{\mathbf{y}} = \{\hat{y}_{l+1}, \ldots, \hat{y}_m\}$

$$\begin{aligned}
&\min_{\mathbf{w}, \mathbf{y}, \hat{\mathbf{y}}, \xi} \frac{1}{2}\|\mathbf{w}\|^2 + C_l \sum_{i=1}^{l} \xi_i + C_u \sum_{i=l+1}^{m} \xi_i \\
&\textit{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\
&\qquad\qquad \hat{y}_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\
&\qquad\qquad \xi_i \geq 0
\end{aligned} \tag{12.4}$$

Since make assinment for all unlabelled samples is computing consuming, TSVM search locally and optimizes iteratively. It initially trains a SVM with labelled data and assigns pseudo-labels to entire dataset. Note that only the separation of labelled data is reliable in the case, we set $C_l >> C_u$. Then TSVM searches for samples classfied in different classes and have the lowest confidence (i.e. sum of their slack variables is larger than 2), then shifts their labels and solves the optimization. Finally it increases C_u and shifts label in the next iteration until $C_u = C_l$. when there is severe class imbalance, we can exploits two constants C_u^+, C_u^-

Algorithm 13: Transductive SVM

```
   Data: $D_l = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l)\},;$
   $D_u = \{\mathbf{x}_{l+1}, \dots, \mathbf{x}_{l+u}\},\ C_l,\ C_u$
   Result: $\hat{\mathbf{y}} = \{\hat{y}_{l+1}, \dots, \hat{y}_{l+u}\}$
1  $SVM_l \leftarrow SVM(D_l)$
2  $\mathbf{y} = \{\hat{y}_{l+1}, \dots, \hat{y}_{l+u}\} \leftarrow SVM_l(D_u)$
3  $C_u \leftarrow \{C_u | C_u >> C_l\}$
4  while $C_u < C_l$ do
5      $\mathbf{w}, b, \xi \leftarrow$ solve optimization
6      while $\exists\{i, j | (\hat{y}_i \hat{y}_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2)\}$ do
7          $\hat{y}_i \leftarrow -\hat{y}_i$
8          $\hat{y}_j \leftarrow -\hat{y}_j$
9          $\mathbf{w}, b, \xi \leftarrow$ solve optimization
10     $C_u \leftarrow \{2C_u, C_l\}$
```

instead of C_u and let

$$C_u^+ = \frac{u^-}{u^+} C_u^- \tag{12.5}$$

12.3 Graph Semi-Supervised Learning

Given $D_l = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l)\}$, $D_u = \{\mathbf{x}_{l+1}, \dots, \mathbf{x}_m\}$, $G = (V, E)$ where $V = \{x_1, \dots, x_m\}$ and the graph can be represented as

$$\mathbf{W}_{ij} = \begin{cases} \exp\left(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right), & i \neq j \\ 0, & otherwise \end{cases} \tag{12.6}$$

G learns $f : V \to \mathbb{R}$, the energy function of $\mathbf{f}$ is

$$\begin{aligned} E(f) &= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{W}_{ij} \big(f(x_i) - f(x_j)\big)^2 \\ &= \mathbf{f}^T (\mathbf{D} - \mathbf{W}) \mathbf{f} \end{aligned} \tag{12.7}$$

where $\mathbf{f} = (\mathbf{f}_l^T \mathbf{f}_u^T)^T$, $\mathbf{f}_l, \mathbf{f}_u$ denotes the prediction on labelled data and on unlabelled data respectively, $\mathbf{D} = diag(d_1, \dots, d_m)$, $d_i = \sum_{j=1}^m \mathbf{W}_{ij}$. Let

$$\begin{aligned} \mathbf{W} &= \begin{pmatrix} \mathbf{W}_{ll} & \mathbf{W}_{lu} \\ \mathbf{W}_{ul} & \mathbf{W}_{uu} \end{pmatrix} \\ \mathbf{D} &= \begin{pmatrix} \mathbf{D}_{ll} & \mathbf{0}_{lu} \\ \mathbf{0}_{ul} & \mathbf{W}_{uu} \end{pmatrix} \\ \mathbf{P} &= \mathbf{D}^{-1}\mathbf{W} \begin{pmatrix} \mathbf{D}_{ll}^{-1}\mathbf{W}_{ll} & \mathbf{D}_{ll}^{-1}\mathbf{W}_{lu} \\ \mathbf{D}_{uu}^{-1}\mathbf{W}_{ul} & \mathbf{D}_{uu}^{-1}\mathbf{W}_{uu} \end{pmatrix} \end{aligned} \tag{12.8}$$

Set the partial derivative of $E(\mathbf{f})$ to $\mathbf{f}_u$ to zero,

$$\mathbf{f}_u = (\mathbf{I} - \mathbf{P}_{uu})^{-1} \mathbf{P}_{ul} \mathbf{f}_l \tag{12.9}$$

When it comes to multi-classification $y_i \in \mathcal{Y}$, denote $\mathbf{F} \in \mathbb{R}^{m \times |\mathcal{Y}}|$ as label matrix and its *ith* row is the label vector of $\mathbf{x}_1$ where

$$y_i = \arg \max_{1 \leq j \leq \mathcal{Y}} (\mathcal{F})_{ij} \tag{12.10}$$

$$\mathcal{F}(0) = \mathcal{Y}_{ij} = \begin{cases} 1, & (1 \leq i \leq l) \wedge (y_i = j) \\ 0, & otherwise \end{cases} \tag{12.11}$$

Our optimization objective is

$$\min_{\mathbf{F}} \frac{1}{2}\Big(\sum_{i,j=1}^{m} \mathbf{W}_{ij} \Big\| \frac{1}{\sqrt{d_i}}\mathbf{F}_i - \frac{1}{\sqrt{d_j}}\mathbf{F}_j \Big\|_2^2\Big) + \mu \sum_{i=1}^{l} \|\mathbf{F}_i - \mathbf{Y}_i\|_2^2 \tag{12.12}$$

the regularization term constraints that the learned model is not far away from initial model. We construct label propagation matrix to solve it

$$\mathbf{S} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}} \tag{12.13}$$

Hence,

$$\mathbf{F}(t+1) = \alpha\mathbf{S}\mathbf{F}(t) + (1-\alpha)\mathbf{Y} \tag{12.14}$$

and the iteration can converge to

$$\mathbf{F}^* = \lim_{t\to\infty} \mathbf{F}(t) = (1-\alpha)(\mathbf{I} - \alpha\mathbf{S})^{-1}\mathbf{Y} \tag{12.15}$$

Algorithm 14: Label Propagation

```
  Data: $D_l = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$,;
  $D_u = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$, $\sigma$, $\alpha$
  Result: $\hat{\mathbf{y}} = \{\hat{y}_{l+1}, \ldots, \hat{y}_{l+u}\}$
1 $\mathbf{W}$
2 $\mathbf{S} \leftarrow \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$
3 $\mathbf{F}(0)$
4 $t \leftarrow 0$
5 while not converged to $F^*$ do
6     $\mathbf{F}(t+1) \leftarrow \alpha\mathbf{S}\mathbf{F}(t) + (1-\alpha)\mathbf{Y}$
7     $t \leftarrow t+1$
8 for $i \leftarrow l+1$ to $l+u$ do
9     $y_i \leftarrow \arg\max_{1\leq j\leq \mathcal{Y}}(\mathcal{F}^*)_{ij}$
```

12.4 Disagreement-based Methods

Consider a sample with multiple attribute sets, namely each attribute sets denotes a perspective to describe the samples. This is appropriate for co-training, i.e. train a learner on each attribute set with labelled samples, then let the learner assign pseudo labels with highest confidence and send the samples to other learners as labelled samples to train on. The iteration terminates when every learner converges. Note that we only consider two attribute sets in this case.

12.5 Semi-Supervised Clustering

12.5.1 Constrain

Two kinds of constraints on connectivity: ”must-link” samples $(x_i, x_j) \in M$ must belong to the same cluster while ”cannot-link” samples $(x_i, x_j) \in C$ must belong to different clusters.

12.5.2 Labelled Samples

Exploiting labelled samples in k-means is relative easy that we use them as the initial centriods clusters.

Algorithm 15: co-training

```
Data: $D_l = \{(\langle \mathbf{x}_1^1, \mathbf{x}_1^2\rangle, y_1), \dots, (\langle \mathbf{x}_l^1, \mathbf{x}_1^2\rangle, y_l)\}$,;
$D_u = \{\langle \mathbf{x}_{l+1}^1, \mathbf{x}_{l+1}^2\rangle, \dots, \langle \mathbf{x}_{l+u}^1, \mathbf{x}_{l+u}^2\rangle\},\ s,\ p,\ n,\ T,\ \mathcal{M}\}$
Result: $h_1, h_2$
1  $D_s \leftarrow \phi \cup \{D_t | (D_t \subset D_u) \wedge (|D_t| = s)\}$
2  $D_u \leftarrow D_u \setminus D_s$
3  for $j$ in $\{1,2\}$ do
4      $D_l \leftarrow \{(\mathbf{x}_i^j, y_i) | (\langle \mathbf{x}_i^j, \mathbf{x}_i^{3-j}\rangle, y_i) \in D_l\}$
5  for $t \leftarrow 1$ to $T$ do
6      for $j$ in $\{1,2\}$ do
7          $h_j \leftarrow \mathcal{M}(D_l^j)$
8          $D_p, D_n \leftarrow D_s$ with highest confidence
9          $\tilde{D}_p \leftarrow \{(x_i^{3-j}, +1) | x_i^j \in D_p\}$
10         $\tilde{D}_n \leftarrow \{(x_i^{3-j}, -1) | x_i^j \in D_n\}$
11         $D_s \leftarrow D_s \setminus (D_p \cup D_n)$
12     if $(h_1 = h_1') \wedge (h_2 = h_2')$ then
13         terminate
14     else
15         for $j$ in $\{1,2\}$ do
16             $D_l^j \leftarrow D_l^j \cup (\hat{D}_p^j \cup \hat{D}_n^j$
17         $D_s \leftarrow D_s \cup \{D_t | (D_t \subset D_u) \wedge (|D_t| = 2p + 2n)\}$
```

Algorithm 16: Constrained k-means

```
Data: $D = \{\mathbf{x}_1, \dots, \mathbf{x}_m\},\ M,\ C,\ k$
Result: $\{C_1, \dots, C_k\}$
1  randomly select $k$ samples as initial mean vector$\{\mu_1, \dots, \mu_k\}$
2  while $\exists \mu_j \neq \mu_j'$ do
3      $C_j \leftarrow \phi$
4      for $i \leftarrow 1$ to $m$ do
5          $D_{ij} \leftarrow \|\mathbf{x}_i - \mu_j\|_2^2$
6          $\mathcal{K} \leftarrow \{1, 2, \dots, k\}$
7          is_merged$\leftarrow$ false
8          while $\neg$is_merged do
9              $r \leftarrow \arg\min_{j \in \mathcal{K}} d_{ij}$
10             if $(x_i \in C_r) \wedge C \wedge K \models \phi$ then
11                 $C_r \leftarrow C_r \cup \{x_i\}$
12                 is_merged$\leftarrow$ true
13             else
14                 $\mathcal{K} \leftarrow \mathcal{K} \setminus \{r\}$if $\mathcal{K} = \phi$ then
15                     terminate, raise error
16     for $j \leftarrow 1$ to $k$ do
17         $\mu_j' \leftarrow \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$
```

Algorithm 17: Semi-supervised k-means

```
Data: $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\},$ labelled samples $S = \cup_{j=1}^{k} S_j,\ k$
Result: $\{C_1, \ldots, C_k\}$
1  for $j \leftarrow 1$ to $k$ do
2      $\mu_j \leftarrow \frac{1}{|S_j|} \sum_{\mathbf{x} \in \S_j} \mathbf{x}$
3  while $\exists \mu_j \neq \mu'_j$ do
4      for $j \leftarrow 1$ to $k$ do
5          for $\mathbf{x}$ in $S_j$ do
6              $C_j \leftarrow C_j \cup \{\mathbf{x}\}$
7      for $\mathbf{x}$ in $D \setminus S$ do
8          $d_{ij} \leftarrow \|\mathbf{x}_i - \mu_j\|_2^2$
9          $r \leftarrow \arg\min_{j \in \{1,\ldots,k\}} d_{ij}$
10         $C_r \leftarrow C_r \cup \{\mathbf{x}_i\}$
11     for $j \leftarrow 1$ to $k$ do
12         $\mu'_j \leftarrow \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$
```

Chapter 13

Probabilistic Graphic Model

13.1 Hidden Markov Model

The variables in HMM can be divided into 2 groups, one is state variables $y_i \in \mathcal{Y} = \{s_1, \ldots, s_N\}$ and the other is observed variables $x_i \in \mathcal{X} = \{o_1, \ldots, o_M\}$. Since state variables are usually unobserved, they are latent variables. At attribute moment, observed variable x_t only relies on its corresponding state variable y_t, regardless of former states. Hence, the joint distribution of all variables is

$$P(x_1, y_1, \ldots, x_n, y_n) = P(y_1)P(x_1|y_1)\prod_{i=1}^{n} P(y_i|y_{i-1})P(x_i|y_i) \tag{13.1}$$

from the equation, we can find three genres of parameters:

- Probability of state transformation $P(y_i|y_{i-1})$:
$$a_{ij} = P(y_{t+1} = s_j|y_t = s_i),\ 1 \le i, j \le N \tag{13.2}$$
- Probability of observation $P(x_i|y_i)$:
$$b_{ij} = P(x_t = o_j|y_t = s_i),\ 1 \le j \le M \tag{13.3}$$
- Probability of initial state $\pi = \{\pi_1, \ldots, \pi_N\}$:
$$\pi_i = P(y_1 = s_1),\ 1 \le i \le N \tag{13.4}$$

13.2 Markov Random Field

MRF is an undirected graph model. For a node subset in the graph, if there's an egde between attribute two nodes, the subset is a clique. If the clique can't preserve this property by adding another node, the clique is a maximal clique. For variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and the set $\mathcal{C}$ for all cliques, the joint distribution is

$$P(\mathbf{x}) = \frac{1}{Z}\prod_{Q\in\mathcal{C}} \phi_Q(\mathbf{x}_Q) \tag{13.5}$$

where ϕ_Q is the potential function corresponding to clique Q and $Z = \sum_{\mathbf{x}} \prod_{Q\in\mathcal{C}} \phi_Q(\mathbf{x}_Q)$ is the normalization factor. The problem can be reduced to calculating joint distribution only on maximal cliques

$$P(\mathbf{x}) = \frac{1}{Z^*}\prod_{Q\in\mathcal{C}^*} \phi_Q(\mathbf{x}_Q) \tag{13.6}$$

where $\mathcal{C}^*$ is the set of maximal cliques.
Consider node sets A, B, C, if the nodes in C separates the nodes in A from B (i.e. make the two sets

unconnected with each other), C is the separating set. According to Global Markov Property, given C, the variables in A and B are indenpendent with each other.

$$\mathbf{x}_A \perp \mathbf{x}_B|\mathbf{x}_c \tag{13.7}$$

Hence,

$$P(x_A, x_B|x_C) = P(x_A|x_C)P(x_B|x_C) \tag{13.8}$$

The Local Markov Property derives from the global markov property that given the adjacent nodes of an attribute node, it is independent from all other nodes.

$$n^*(v) = n(v) \cup \{v\},\ \mathbf{x}_V \perp \mathbf{x}_{V\backslash n^*(v)}|\mathbf{x}_{n(v)} \tag{13.9}$$

This can be extended to Pairwise Markov Propety, namely the two nonadjacent nodes are independent with each other

$$(u, v \in V) \wedge (< u, v > \notin E) \models \mathbf{x}_u \perp \mathbf{x}_v|\mathbf{x}_{V\backslash <u,v>} \tag{13.10}$$

Finally we obtain the potential function of Markov Random Field

$$\begin{aligned} H(Q)(\mathbf{x}_Q) &= \sum_{u,v\in Q, u\neq v} \alpha_{uv} x_u x_v + \sum_{v\in Q} \beta_v x_v \\ \phi_Q(X_Q) &= e^{-H_Q(\mathbf{x}_Q)} \end{aligned} \tag{13.11}$$

13.3 Conditional Random Field

In contrary with Hidden Markov Model and Markov Random Field, which are generative models, Conditional Random Field is discriminative model. Let $\mathbf{x}$ be observed sequence and $\mathbf{y}$ be label sequence, for $G = (V, E)$ and nodes of the graph are elements of $\mathbf{y}$, if

$$P(y_v|\mathbf{x}_v, \mathbf{x}_{V\backslash n(v)}) = P(y_v|\mathbf{x}_v, \mathbf{x}_{n(v)}) \tag{13.12}$$

stands, $< \mathbf{y}, \mathbf{x} >$ constructs a CRF. Hence,

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\exp\Bigg(\sum_j \sum_{i=1}^{n-1} \lambda_j t_j(y_{i+1}, y_i, \mathbf{x}, i) + \sum_k \sum_{i=1}^{n} \mu_k s_k(y_i, \mathbf{x}, i)\Bigg) \tag{13.13}$$

where $t_j(y_{i+1}, y_i, \mathbf{x}, i)$ is the transition feature function on adjacent labels and $s_k(y_i, \mathbf{x}, i)$ is the status feature function on label i.

13.4 Marginalization

Consider $\mathbf{x}_F, \mathbf{x}_E$ are two unintersected subsets of variables $\mathbf{x}$,

$$\begin{aligned} P(\mathbf{x}_F|\mathbf{x}_E) &= \frac{P(\mathbf{x}_E, \mathbf{x}_F)}{\sum_{\mathbf{x}_F} P(\mathbf{x}_E, \mathbf{x}_F)} \\ P(\mathbf{x}_E) &= \sum_{\mathbf{x}_F} P(\mathbf{x}_E, \mathbf{x}_F) \end{aligned} \tag{13.14}$$

Belief Propagation reduces the redundance in calculating margin distribution

$$m_{ij}(x_j) = \prod_{x_i} \phi(x_i, x_j) \prod_{k\in n(i)\backslash j} m_{ki}(x_i) \tag{13.15}$$

the belief passes message m_{ij} from x_i to x_j, thus the operation only relies on x_i and its adjacent nodes.

$$P(x_i) = \prod_{k\in n(i)} m_{ki}(x_i) \tag{13.16}$$

Firstly we select a root and propagate the messages to all other nodes until the messages are received by all leaves, then the leaves propagate back until all the messages are received by root.

13.5 Approximate Inference

13.5.1 Markov Chain Monte Carlo Sampling

When we are interested in expectation instead of distribution,

$$E_p[f] = \int f(x)p(x)dx \tag{13.17}$$

we can sample $\{x_1, \dots, x_N\}$ according to $p(x)$,

$$\hat{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{13.18}$$

when the samples are indenpendent from each other, the sampling obtains high precision. When $\mathbf{x}$ is in high demensional, we use MCMC method. The intuition is that at time t the Markov Chain is stable with

$$p(\mathbf{x}^t)T(\mathbf{x}^{t-1}|\mathbf{x}^t) = p(\mathbf{x}^{t-1})T(\mathbf{x}^t|\mathbf{x}^{t-1}) \tag{13.19}$$

where T is the status transformation probability, then $p(\mathbf{x})$ is the stable distribution of the Markov Chain. Metrobolis-Hastings algorithm exploits reject sampling to approximate $p(\mathbf{x})$, i.e. there's chance that the candidate sample will be rejected. Let $Q(\mathbf{x}^*|\mathbf{x}^{t-1})$ be the prior, $A(\mathbf{x}^*|\mathbf{x}^{t-1})$ be the acceptance probability,

$$p(\mathbf{x}^{t-1})Q(\mathbf{x}^*|\mathbf{x}^{t-1})A(\mathbf{x}^*|\mathbf{x}^{t-1}) = p(\mathbf{x}^*)Q(\mathbf{x}^{t-1}|\mathbf{x}^*)A(\mathbf{x}^{t-1}|\mathbf{x}^*) \tag{13.20}$$

Hence,

$$A(\mathbf{x}^*|\mathbf{x}^{t-1}) = \min\left(1, \frac{p(\mathbf{x}^*)Q(\mathbf{x}^{t-1}|\mathbf{x}^*)}{p(\mathbf{x}^{t-1})Q(\mathbf{x}^*|\mathbf{x}^{t-1})}\right) \tag{13.21}$$

13.5.2 Variational Inference

Variational Inference approximates posterior of local optimal with deterministic solution. Consider observable variables $\{x_1, \dots, x_N\}$ rely on latent variable $\mathbf{z}$, the joint distribution of x is

$$p(\mathbf{x}|\mathbf{\Theta}) = \prod_{i=1}^{N}\sum_{\mathbf{x}} p(x_i; \mathbf{z}|\mathbf{\Theta}) \tag{13.22}$$

We use E-M algorithm to inference the distribution of latent variable $p(\mathbf{z}|\mathbf{x}, \mathbf{\Theta})$:

- Expectation step: inference $P(\mathbf{z}|\mathbf{x}, \mathbf{\Theta}^t)$ by $\mathbf{\Theta}^t$.
- Maximization step:

$$\begin{aligned}\mathbf{\Theta}^{t+1} &= \arg_{\mathbf{\Theta}} \max \mathcal{Q}(\mathbf{\Theta}; \mathbf{\Theta}^t) \\ &= \arg_{\mathbf{\Theta}} \max \sum_{\mathbf{x}} p(\mathbf{z}|\mathbf{x}, \mathbf{\Theta}^t) \ln p(\mathbf{x}, \mathbf{z}|\mathbf{\Theta})\end{aligned} \tag{13.23}$$

Hence, the approximate distribution $q(\mathbf{z})$ is

$$\begin{aligned} q(\mathbf{z}) &= \mathcal{L}(q) + KL(q\|p) \\ \mathcal{L}(q) &= \int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} d\mathbf{z} \\ KL(q\|p) &= -\int q(\mathbf{z}) \ln \frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})} d\mathbf{z}\end{aligned} \tag{13.24}$$

Assume $\mathbf{z}$ is the ensemble of independent variables z_i

$$q(\mathbf{z}) = \prod_{i=1}^{M} q_i(\mathbf{z}_i) \tag{13.25}$$

the optimal $q_j^*(z_j)$ is obtained by the Mean Field method

$$q_j^*(\mathbf{z}_j) = \frac{\exp \mathbb{E}_{i\neq j}[\ln p(\mathbf{x}, \mathbf{z})]}{\int \exp \mathbb{E}_{i\neq j}[\ln p(\mathbf{x}, \mathbf{z})] d\mathbf{z}_j} \tag{13.26}$$

Chapter 14

Logical Rule Learning

A rule is

$$A \leftarrow f_1 \wedge f_2 \wedge \cdots \wedge f_L \tag{14.1}$$

where f are literals and A is the head. A literal can be an atomic formula and its negation.

14.1 Sequential Convering

- Top-Down: Generate-then-test, generalize a general rule to its specified case. It starts with a wide-ranged cover and contract the cover by adding new literals. This is appropriate for learning propositional rules (i.e. zero-order rule).
- Bottom-Up: Data-driven, learn specified rules in a limited cover and relax the cover by removing literals. This is appropriate for first-order rule.

14.2 Prunning

CN2 for pre-prunning exploits Likelihood Ratio Statistics to measure the difference between empirical distribution of training set $\{m_+, m_-\}$ and the set of rules $\{\hat{m}_+, \hat{m}_-\}$

$$LRS = 2\left(\hat{m}_+ \log \frac{\frac{\hat{m}_+}{\hat{m}_+ + \hat{m}_-}}{\frac{m_+}{m_+ + m_-}} + \hat{m}_- \log \frac{\frac{\hat{m}_-}{\hat{m}_+ + \hat{m}_-}}{\frac{m_-}{m_+ + m_-}}\right) \tag{14.2}$$

the desired LRS would be large amd close to 1. Reduced Error Prunning for post-prunning exploits the replacement of samples. Incremental REP splits the dataset into training set and validation set, generates a rule $\mathbf{r}$ on training set and prunes it to $\mathbf{r}'$ by precision on validation set, ultimately removes the samples covered by $\mathbf{r}'$. Repeated Incremental Prunning to Produce Error Reduction (RIPPER) algorithm set

$$\frac{\hat{m}_+ + (m_- - \hat{m}_-)}{m_+ + m_-} \tag{14.3}$$

as the metric of rule set instead of precision. For $\mathbf{r}_i \in \mathcal{R}$, RIPPER generates $\mathbf{r}'$ by IREP and specialize $\mathbf{r}_i$ to $\mathbf{r}''$.

14.3 First-Order Inductive Learner

First-order rule is learned from relational data, which is consists of atomic formulae of background knowledge describing the attribute of samples and atomic formulae observed from examples.

$$\forall A \forall B (f(A, B) \leftarrow g(A, B) \wedge h(A, B)) \tag{14.4}$$

FOIL exploits FOIL grain to select literals

$$FOIL_{gain} = \hat{m}_+ \left(\log \frac{\hat{m}_+}{\hat{m}_+ + \hat{m}_-} - \log \frac{m_+}{m_+ + m_-} \right) \tag{14.5}$$

that enhances positive samples.

14.4 Inductive Logic Programming

ILP introduces function expression to embed with predicates. Let B denote background knowledge, H be the hypothesis model, $E = E^+ \cup E^-$ denote the training examples, the aim of ILP is

$$B \cup H \models E \tag{14.6}$$

The dataset of ILP is instances with mutual relationships instead of features or attributes in datasets for statistical learning. Induction is the inverse operation of deduction. The former generalizes from specialized examples while the latter specializes from a general rule. In other words, induction generates extension knowledge while deduction only exploits intrinsic knowledge.

14.4.1 Most General Unifier

We can substitute some variables to another

$$\begin{aligned} \theta &= \{X_1/X_0, Y_1, Y_0\} \\ C &= f(X_0, Y_0) \wedge g(X_0, Y_0) \\ C' &= C\theta = f(X_1, Y_1) \wedge g(X_1, Y_1) \end{aligned} \tag{14.7}$$

Substitution has its composition $\theta \circ \lambda$ and inverse θ^{-1}. Unification is the operation that make some clauses equivalent with each other via a substitution.

$$\exists \theta, C_1\theta = C_2\theta = C_3\theta \tag{14.8}$$

then the clauses are unifiable and θ is the unifier. For a set of clauses W,

$$\forall \theta, \exists \delta, \theta = \delta \circ \lambda \tag{14.9}$$

then δ is the only Most General Unifier (MGU) of W. For $C_1 = A \vee L_1, C_2 = B \vee L_2$, if

$$\exists \theta,\ L_1\theta = \neg L_2\theta \tag{14.10}$$

then

$$C = (C_1 - \{L_1\})\theta \vee (C_2 - \{L_2\})\theta \tag{14.11}$$

$C_1 = C/C_2$, $C_2 = C/C_1$ are resolution quotients. Hence, the aim of inverse resolution is obtaining C_2 given C, C_1. For $L_1 \in C_1$,

$$\exists \phi_1, (C_1 - \{L_1\})\phi_1 \quad \models \quad C \tag{14.12}$$

then ϕ_1 is the unifier of C_1 to C. Let $\phi_2 \in Domain(L_1) - Domain(C_1 - \{L_1\})$, $\theta_2 \in Domain(L_2)$,

$$\neq L_1\phi_1 \circ \phi_2 = L_2\theta_2 \tag{14.13}$$

Let $\theta_1 = \phi_1 \circ \phi_2$, first-order inverse resolution is

$$C_2 = (C - (C_1 - \{L_1\})\theta_1 \vee \{\neg L_1\theta_1\})\theta_2^{-1} \tag{14.14}$$

14.4.2 Least General Generalization

ILP exploits bottom-up strategy to generate a set of general rules from specialized groundtruth rules.As the concept in the domain of logical induction, LGG is the inverse operation of MGU, which is the concept in the domain of logical deduction.

Algorithm 18: Least General Generalization

```
  Input: first-order formulae $\mathbf{r}_1, \mathbf{r}_2$
  Output: generalized fist-order formula $\mathbf{r}'$
1 while $\mathbf{r}_n \neq \mathbf{r}'$ do
2     for $t \leftarrow 1$ to $L$ do
3         if $\mathbf{r}_1[t] = \mathbf{r}_2[t]$ then
4             $\mathbf{r}_n[t] \leftarrow \mathbf{r}_1[t]$
5         else
6             $\mathbf{r}_n[t] \leftarrow V$
7     $L \leftarrow |\mathbf{r}_n|$
8     $\mathbf{r}' \leftarrow \mathbf{r}_n$
```

14.4.3 Inverse Resolution

Resolution principle can be discribed as

$$\exists\{L\} \subset C_1, \exists\{\neg L\} \subset C_2, C = (C_1 - \{L\}) \vee (C_2 - \{\neg L\}) \tag{14.15}$$

Hence, given C, C_1 inverse resolution is solving C_2

$$C_2 = (C - (C_1 - \{L\})) \vee \{\neg L\} \tag{14.16}$$

There are 4 techniques of inverse resolution, let capital be conjunctive clauses

- Absorption:
$$p \leftarrow A \wedge B, q \leftarrow A \quad \models \quad p \leftarrow q \wedge B, q \leftarrow A \tag{14.17}$$
- Identification:
$$p \leftarrow A \wedge B, p \leftarrow A \wedge q \quad \models \quad q \leftarrow B, p \leftarrow A \wedge q \tag{14.18}$$
- Intra-construction:
$$p \leftarrow A \wedge B, p \leftarrow A \wedge C \quad \models \quad q \leftarrow B, p \leftarrow A \wedge q, q \leftarrow C \tag{14.19}$$
- Inter-construction:
$$p \leftarrow A \wedge B, q \leftarrow A \wedge C \quad \models \quad p \leftarrow r \wedge B, r \leftarrow A, p \leftarrow r \wedge C \tag{14.20}$$

In esssential, intra-construction and inter-construction implement predicate invention by outputing new atomic formulae.

14.4.4 Inverse Implication

Since the generalization of First-Order Logic is not equivalent with implication, inverse resolution is incomplete. For instance,

$$\begin{aligned} P &\equiv p(f(X)) \leftarrow p(X) \\ Q &\equiv p(f(f(X))) \leftarrow p(X) \\ \mathit{substitute\ X\ with}\ f(x)&, \\ R &\equiv p(f(f(x))) \leftarrow p(X) \end{aligned} \tag{14.21}$$

we can prove $P \rightarrow Q$ yet we couldn't operate P with any substitution to make it equivalent with Q. In another word, we can't obtain P by generalizing Q. In contrast, the derivation from P to Q only exploits the rule of P and this is self-resolution. Let $P\theta \subseteq Q$ denote that P is the Least General Generalization of Q, $P \rightarrow Q$ denote that P implicates Q, the complete implication is consist of the 3 following cases:
if $P \rightarrow Q$,

- $Q \equiv 1$
- $P\theta \subseteq Q$
- $\exists R(R\theta \subseteq Q)$ and R can be obtained by inverse implication of P.

14.4.5 Inverse Entailment

Since a proposition is equivalent to its converse-negative proposition, we write $B \cup H \models E$ as

$$B \cup \neg E \models \neg H \tag{14.22}$$

This approach converses solving H by induction to solving $\neg H$ by deduction. Since the $\neg H$ we obtain is a specialized hypothesis based on one simgle example, we generalize it by

$$\begin{aligned} B \cup \neg E &\models \perp \\ (\perp \models \neg H) \wedge (H &\models \neg \perp) \end{aligned} \tag{14.23}$$

14.4.6 Propositionalize

Propopsitionalize-based converses relational dataset to attribute-value dataset. For instance, converse all the relationships into propositions and assignment them with `True` or `False`.

14.5 Meta-Interpretive Learning

Meta-Interpretive Learning (MIL) is a variant of ILP. Similar to ILP, it takes background knowledge B and examples E as input, generating hypothesis H. The B of MIL is $B_M \cup B_A$, where B_M is a definite logic program that serves as meta interpreter, B_A and H are ground definite higher-order logical programs. $E = \langle E^+, E^- \rangle$ and they are positive and negative unit clauses respectively.

$$\begin{aligned} B \cup H &\models E^+ \\ B \cup \neg E^+ &\models \neg H \end{aligned} \tag{14.24}$$

Let $\mathcal{H}_{B,E}$ (i.e. the Hibrand Space for B, E) be the complete set of abductive hypothesis H. Algorithm A is a **Meta Interpretive Learner** iff

$$\forall B, E,\ H = A(B, E),\ E \in \mathcal{H}_{B,E} \tag{14.25}$$

This kind abduction can produce **Predicate Invention** by introducing Skolem constants representing new predicates. In contrary of the inverse resolution in ILP for predicate invention using a single clause as example, predicate invention by abduction in Metagol exploits several positive and negative examples. Meta-rule, rule of the rules in another word, is in the form of

```
metarule(Name, Substitutions, Head, Body)
```

where Substitutions contains all the predicates. The following are 5 commonly used metarules:

```
metarule(identity,[P,Q],[P,A,B],[[Q,A,B]]). % rule of identity
metarule(inverse[P,Q],[P,A,B],[[Q,B,A]]). % rule of inverse
metarule(precon,[P,Q,R],[P,A,B],[[Q,A],[R,A,B]]). % rule of precon
metarule(postcon,[P,Q,R],[P,A,B],[[Q,A,B],[R,B]]). % rule of postcon
metarule(chain,[P,Q,R],[P,A,B],[[Q,A,C],[R,C,B]]). % rule of chain
```

We can also put predicates in head into body to learn recursive rules

```
metarule(chain_recursive,[P,Q],[P,A,B],[[Q,A,C],[P,C,B]])
```

Here we explore the learning procedure of Metagol with code:

```
learn(Pos,Neg,Prog):-
    prove(Pos,[],Prog),
    \+ prove(Neg,Prog,Prog).
prove([],Prog,Prog).
prove([Atom|Atoms],Prog,Prog):-
    prove_aux(Atom,Prog1,Prog3),
    prove(Atoms,Prog3,Prog2).
prove_aux(Atom,Prog,Prog):-
    prim(Atom),!,
    call(Atom).
prove_aux(Atom,Prog1,Prog2):-
    member(sub(Name,Subs),Prog1),
    metarule(Name,Subs,(Atom:-Body)),
    prove(Body,Prog1,Prog2).
prove_aux(Atom,Prog1,Prog2):-
    metarule(Name,Subs,(Atom:-Body)),
    prove(Body,[sub(Name,Subs)|Prog1],Prog2).
```

Metagol firstly tries to deductively prove an atom using complied background knowledge by delegating the proof to Prolog `call(Atom)`. If this deduction fails, Metagol tries to unify the atom with the head of a metarule and tries to bind the higher-order variables in a metarule to symbols by substitution. Metagol saves the resulting substitutions and tries to prove the body of the metarule. After proving all atoms, Metagol generates the programs by projecting the substitutions onto their corresponding metarules. Afterwards, Metagol checks the consistency of learned programs with negative samples. If it is inconsistent, Meteagol backtracks to another branch of the SLD-tree.

14.6 Learning Higher-Order Logic

As aforementioned, a variable is first-order if it can be bound to a constant symbol or another first-order variable. Then we have a variable is **higher-order** if it can be bound to a predicate symbol or another higher-order variable. Let V_1, V_2 denote the set of first-order variables and higher-order variables respectively, a metarule is a higher-order formula of the form

$$\exists\pi\forall\mu \leftarrow l_1, \ldots, l_m \tag{14.26}$$

where $\pi \subseteq V_1 \cup V_2,\ \mu \subseteq V_1 \cup V_2,\ \pi \cap \mu = \phi$. The definition of abstract is

$$f(A,B) \leftarrow map(A,B,succ) \tag{14.27}$$

where map is

$$\begin{aligned} &map([],[],F) \leftarrow \\ &map([A|As],[B|Bs],F) \leftarrow F(A,B), map(As,Bs) \end{aligned} \tag{14.28}$$

where F is a universally quantified higher-order variable. The input of abstracted MIL is $B = B_C \cup B_I,\ E^+,\ E^-,\ M$, where B_C is the set of first-order horn clauses and B_I is the set of higher-order definitions. The procedural distinction between them is that the former is proved deductively while the latter is proved by meta interpretion. Hence, the returned hypothesis is

- $\forall h \in H,\ \exists m \in M$ such that $h = m\theta$ where θ is a substitution that grounds all the existentially quantified variables in m,
- $H \cup B \models E^+$
- $H \cup B| \neq E^-$

An invention is a predicate p/a that is in the predicate signature of H and not in the predicate signature of $B \cup E^+ \cup^-$. In other words, the invented predicate appears neither in background knowledge nor in the training data. The following codes show the the commons and the distinctions between background knowledge B_C, interpreted background knowledge B_I and metarule M respectively.

- Complied Background Knowledge B_C:

```
empty([]).
head([H|_],H).
tail([_|T],T).
last([A],A):-!.
last([_|T],A):-last(T,A).
```

- Interpreted Background Knowledge B_I:

```
ibk(([map,[],[],F]:-[]))
ibk(([map,[A|As],[B|Bs],F]:-[[F,A,B],[map,As,Bs,F]])).
ibk(([fold,[],Acc,Acc,F]:-[])).
ibk(([fold,[A|As],Acc1,B,F]:-[[F,A,Acc1,Acc2],[fold,As,Acc2,B,F]])).
ibk([ifthenelse,A,B,Cond,Then,_],[[Cond,A],[Then,A,B]]).
ibk([ifthenelse,A,B,Cond,_,Else],[[not,Cond,A],[else,A,B]]).
```

- Metarules M:

```
metarule(monadic,[P,Q],([P,A,A]:-[[Q,A]])).
metarule(identity,[P,Q],([P,A,B]:-[[Q,A,B]])).
metarule(inverse,[P,Q],([P,A,B]:-[[Q,B,A]])).
metarule(didentity,[P,Q],([P,A,B]:-[[Q,A,B],[R,A,B]])).
metarule(precon,[P,Q,R],([P,A,B]:-[[Q,A],[R,A,B]])).
metarule(postcon,[P,Q,R],([P,A,B]:-[[Q,A,B],[R,B]])).
metarule(curry1,[P,Q,R],([P,A,B]:-[[Q,A,B,R]])).
metarule(curry2,[P,Q,R,S],([P,A,B]:-[[Q,A,B,R,S]])).
metarule(curry3,[P,Q,R,S,T],([P,A,B]:-[[Q,A,B,R,S,T]])).
metarule(chain,[P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).
metarule(tailrec,[P,Q],([P,A,B]:-[[Q,A,C],[P,C,B]])).
```

A metarule is in the fragment M^i_j if it has at most j literals in the body and each literal has at most i arities. Given p predicate symbols and m metarules in M^i_j, the number of programs expressible with n clauses is at most $O\big((mp^{j+1})^n\big)$. For abstracted MIL programs, namely higher-order metarules with at most k exsistentially quantified higher-order variables, the number of abstracted M^i_j with n clauses is at most $O\big((mp^{j+1+k})^n\big)$. Given a maximum program size n_u, MIL has sample complexity (i.e. the number of training samples required to achieve error less than ϵ with probability at least $1-\delta$):

$$s_u \geq \frac{1}{\epsilon}\big(n_u \ln m + (j+1)n_u \ln p + \ln(\frac{1}{\delta})\big) \tag{14.29}$$

Similarly, given a maximum program size n_a the abstracted MIL has sample complexity:

$$s_a \geq \frac{1}{\epsilon}\big(n_a \ln m + (j+1+k)n_a \ln p + \ln(\frac{1}{\delta})\big) \tag{14.30}$$

Hence, we have $s_a < s_u$ when

$$n_u - n_a > \frac{k}{j+1} n_a \tag{14.31}$$

The proof procedure of Metagol-ho is silghtly different from that of Metagol:

```
learn(Pos,Neg,Prog):-
    prove(Pos,[],Prog),
    \+ prove(Neg,Prog,Prog).
prove([],Prog,Prog).
prove([Atom|Atoms],Prog,Prog):-
    prove_aux(Atom,Prog1,Prog3),
    prove(Atoms,Prog3,Prog2).
prove_aux(Atom,Prog,Prog):-
    prim(Atom),!,
    call(Atom).
prove_aux(Atom,Prog1,Prog2):-
    ibk((Atom:-Body)),
    prove(Body,Prog1,Prog2).
prove_aux(Atom,Prog1,Prog2):-
    member(sub(Name,Subs),Prog1),
    metarule(Name,Subs,(Atom:-Body)),
    prove(Body,Prog1,Prog2).
prove_aux(Atom,Prog1,Prog2):-
    metarule(Name,Subs,(Atom:-Body)),
    prove(Body,[sub(Name,Subs)|Prog1],Prog2).
```

Metagol-ho works in the way same to Metagol except for the use of IBK. It firstly proves the atoms deductively and if it fails, Metagol-ho tries to unify the atom with a head in the clause of IBK `ibk((Atom:-Body))` and tries to prove the body of the matched definition. Failing this, Metagol-ho continues to work in the way same as Metagol.

Chapter 15

Reinforcement Learning

15.1 Markov Decision Process

Recall Markov Chain, it can be interpreted as a decision process in essential. Let $x \in \mathcal{X}$ be status variable, $a \in \mathcal{A}$ be action variable, $\mathcal{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \mapsto \mathbb{R}$ be transformation pobability, $\mathcal{R} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \mapsto \mathbb{R}$ be transformation reward,

$$\mathcal{M} = \langle \mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle \tag{15.1}$$

is the form of markov decision. Reinforcement Learning aims to learn a policy π, then we obtain

$$a = \pi(x) \tag{15.2}$$

Policy can be deterministic $\pi : \mathcal{X} \mapsto \mathcal{A}$ or probabilistic $\pi : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$ and

$$\sum_a p(\pi, a) = 1 \tag{15.3}$$

The desired policy maximizes the expectation of reward, the accumulated reward at T step is

$$\mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t] \tag{15.4}$$

or the γ-fold accumulated reward

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}] \tag{15.5}$$

15.2 Model-based Learning

If $\mathcal{M}$ is given, we obtain a model-based learning task.

15.2.1 Policy Estimation

Let $V^{\phi}(x)$ (state value function) be the accumulated reward under policy π, $Q^{\pi}(x, a)$ (state-action value function) be the accumulated reward with action a then use policy π. Hence, the accumulated reward after T steps is

$$V_T^{\pi}(x) = \sum_{a \in A} \pi(x, a) \sum_{x' \in \mathcal{X}} P_{x \to x'}^{a} \left(\frac{1}{T} R_{x \to x'}^{a} + \frac{T-1}{T} V_{T-1}^{\pi}(x') \right) \tag{15.6}$$

this is the Bellman Equation. The accumulated γ-fold reward is

$$V_{\gamma}^{\pi}(x) = \sum_{a \in \mathcal{A}} \pi(x, a) \sum_{x' \in \mathcal{X}} P_{x \to x'}^{a} \left(R_{x \to x'}^{a} + \gamma V_{\gamma}^{\pi}(x') \right) \tag{15.7}$$

Hence, status-action value function is

$$\begin{aligned} Q_T^\pi(x,a) &= \sum_{x'\in\mathcal{X}} P_{x\to x'}^a \left(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}V_{T-1}^\pi(x')\right) \\ Q_\gamma^\pi(x,a) &= sum_{x\in\mathcal{X}} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma V_\gamma^a(x')\right) \end{aligned} \tag{15.8}$$

15.2.2 Policy Updating

If the current policy is suboptimal, we update it by

$$\pi^* = \arg_\pi \max V^\pi(x) \tag{15.9}$$

Hence,

$$\begin{aligned} V_T^*(x) &= \max_{a\in\mathcal{A}} \sum_{x'\in\mathcal{X}} P_{x\to x'}^a \left(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}V_{T-1}^*(x')\right) \\ V_\gamma^*(x) &= \max_{a\in\mathcal{A}} \sum_{x'\in\mathcal{X}} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma V_\gamma^*(x')\right) \\ V^*(x) &= \max_{a\in\mathcal{A}} Q^{\pi^*}(x,a) \\ Q_T^*(x,a) &= \sum_{x'\in\mathcal{X}} P_{x\to x'}^a \left(\frac{1}{T}R_{x\to x'}^a + \frac{T-1}{T}\max_{a'\in\mathcal{A}} Q_T^*(x',a')\right) \\ Q_\gamma^*(x,a) &= \sum_{x'\in\mathcal{X}} P_{x\to x'}^a \left(R_{x\to x'}^a + \gamma \max_{a'\in\mathcal{A}} Q_\gamma^*(x',a')\right) \end{aligned} \tag{15.10}$$

and these are Optimal Bellman equations. Ultimately, we update the policy by

$$\pi'(x) = \arg_{a\in\mathcal{A}} \max Q^\pi(x,a) \tag{15.11}$$

Model-based reinforcement learning doesn't have generalization capacity since it only finds the optimal policy for current state.

15.3 Model-free Reinforcement Learning

In most real scenes V is unknown.

15.3.1 Monte Carlo Reinforcement Learning

An alternative of the known model is sampling repeatedly to approximate status-action value function. We expolit ϵ-greedy strategy that

$$\pi^\epsilon(x) = (1-\epsilon)\pi(x) + \epsilon Uniform(a\in\mathcal{A}) \tag{15.12}$$

Hence, we obtain the on-policy algorithmOn-policy algorithm relies on the initial policy, we can drag two trajectories of π, π'. We can estimate the trajectory of π from that of π' by importance sampling, namely weight the reward via the propobabilities generating the ith trajectory

$$Q(x,a) = \frac{1}{m}\sum_{i=1}^m \frac{P_i^\pi}{P_i^{\pi'}} r_i \tag{15.13}$$

The probability of generating a specialized trajectory under policy π is

$$P^\pi = \prod_{i=0}^T \pi(a_i, x_i) P_{x_i\to x_i+1}^{a_i} \tag{15.14}$$

Algorithm 19: On-Policy

```
   Input: $\mathcal{A}$, $x_0$, $T$
   Output: $\pi$
1  $Q(x,a) \leftarrow 0,\ count(x,a) \leftarrow 0,\ \pi(x,a) \leftarrow \frac{1}{|\mathcal{A}|}$
2  for $s$ do
3      generate $\langle x_0, a_0, r_1, \ldots, a_{T-1}, r_T, x_T \rangle$
4      for $t \leftarrow 1$ to $T-1$ do
5          $R \leftarrow \frac{1}{T-t} sum_{i=t+1}^{T} r_i$
6          $Q(x_t, a_t) \leftarrow \frac{Q(x_t,a_t) \times count(x_t,a_t) + R}{count(x_t,a_t)+1}$
7          $count(x_t, a_t) \leftarrow count(x_t, a_t) + 1$
8      $\pi(x,a) \leftarrow (1-\epsilon) \arg\max_{a'} Q(x,a') + \epsilon Uniform(a \in \mathcal{A})$
```

in fact, the transformation probability is not essential

$$\frac{P^{\pi}}{P^{\pi'}} = \prod_{i=1}^{T-0} \frac{\pi(x_i, a_i)}{\pi'(x_i, a_i)} \tag{15.15}$$

Algorithm 20: Off-Policy

```
   Input: $\mathcal{A}$, $x_0$, $T$
   Output: $\pi$
1  $Q(x,a) \leftarrow 0,\ count(x,a) \leftarrow 0,\ \pi(x,a) \leftarrow \frac{1}{|\mathcal{A}|}$
2  for $s$ do
3      generate $\langle x_0, a_0, r_1, \ldots, a_{T-1}, r_T, x_T \rangle$
4      if $a_i = \pi(x)$ then
5          $p_i \leftarrow 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}$
6      else
7          $p_i \leftarrow \frac{\epsilon}{|\mathcal{A}|}$
8      for $t \leftarrow 1$ to $T$ do
9          $R \leftarrow \frac{1}{T-t} \sum_{i=t+1}^{T} r_i \prod_{j=i}^{T-1} \frac{1}{p_j}$
10         $Q(x_t, a_t) \leftarrow \frac{Q(x_T,a_t) count(x_t,a_t) + R}{count(x_t,a_t)+1}$
11         $count(x_t, a_t) \leftarrow count(x_t, a_t) + 1$
12     $\pi \leftarrow \arg\max_{a'} Q(x,a')$
```

15.3.2 Temporal Difference Learning

Temporal Difference algorithm updates the status-action reward function after sampling an entire trajectory.

$$Q_{t+1}^{\pi}(x,a) = Q_t^{\pi}(x,a) + \frac{1}{t+1}\big(r_{t+1} - Q_t^{\pi}(x,a)\big) \tag{15.16}$$

Hence,

$$Q_{t+1}^{\pi}(x,a) = Q_t^{\pi}(x,a) + \alpha\big(R_{x\to x'}^{\pi} + \gamma Q_t^{\pi}(x',a') - Q_t^{\pi}(x,a)\big) \tag{15.17}$$

15.3.3 Value Function Approximation

Given status space $\mathcal{X} = \mathbb{R}^n$, the linear function of status is

$$V_\theta(\mathbf{x}) = \theta^T \mathbf{x} \tag{15.18}$$

We minimize the MSE between V_θ and V^π via gradient descent and obtain

$$\theta = \theta + \alpha\big(V^\pi(\mathbf{x}) - V_\theta(\mathbf{x})\big)\mathbf{x} \tag{15.19}$$

Hence,

$$\theta = \theta + \alpha(r + \theta^T\mathbf{x}' - \theta^T\mathbf{x})\mathbf{x} \tag{15.20}$$

15.4 Imitation Learning

15.4.1 Behavior Cloning

Given a set of trajectories made by the expert, where

$$\tau_i = \langle s_1^i, a_1^i, \dots, s_{n_i+1}^i \rangle \tag{15.21}$$

we extract status-action pairs from all trajectories to construct a new dataset and learn the policy from it directly. However, this simple method prones to overfitting.

15.4.2 Inverse Reinforcement Learning

Given a set of trajectories made by the expert, inverse reinforcement learning aims to find a value function that optimally matches the dataset. Let $R(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the accumulated reward is

$$\rho^\pi = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{w}^T\mathbf{x} | \pi\Big] \tag{15.22}$$

We sample the optimal policy $\mathbf{w}^*$ from dataset and calculate the mean of all status on each trajectory $\overline{\mathbf{x}}$,

$$\mathbf{w}^{*T}(\overline{\mathbf{x}}^* - \overline{\mathbf{x}}^\pi) \geq 0 \tag{15.23}$$

Ultimately we obtain

$$\begin{gathered} \mathbf{w}^* = \arg_{\mathbf{w}} \max \ \min_{\pi} \mathbf{w}^*(\overline{\mathbf{x}}^* - \overline{\mathbf{x}}^\pi) \\ \textit{subject to } \|\mathbf{w}\| \leq 1 \end{gathered} \tag{15.24}$$

Algorithm 21: Inverse Reinforcement Learning

```
  Input: $\mathcal{A}$, $x_0$, $T$, $D = \{\tau_1, \dots, \tau_m\}$
  Output: $\mathbf{w}^{*T}\mathbf{x}$, $\pi$
1 $\overline{\mathbf{x}}^* \leftarrow Mean(\mathcal{X})$
2 $\pi \leftarrow Random(\pi)$
3 for $t$ do
4     $\overline{\mathbf{x}}^\pi \leftarrow Mean(Sample(\mathcal{X}))$
5     $\mathbf{w}^* \leftarrow \arg_{\mathbf{w}} \max \ \min_\pi \mathbf{w}^*(\overline{\mathbf{x}}^* - \overline{\mathbf{x}}^\pi)$   $\textit{subject to } \|\mathbf{w}\| \leq 1$
6     $\pi \leftarrow Optimal\langle \mathcal{X}, \mathcal{A}, \mathbf{w}^{*T}\mathbf{x} \rangle$
```

Bibliography

[1] Zhi-Hua Zhou, *Machine Learning*, THU Express, 2016.

[2] Kevin P. Murphy, *Machine Learning A Probabilistic Perspective* , the MIT Express, 2012.

[3] Wang-Zhou Dai, Zhi-Hua Zhou, *A Survey on Inductive Logic Programming*, Computer Research and Development, 2019, 56(1).