Statistial Learning Notes

Mou Minghao, CUHK(SZ)

August 20, 2022

Abstract

This is my learning notes.

Contents

1 Perceptron

Definition 1.1 (Linear Separability)**.** *Given a dataset*

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

where $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{1, -1\}, i = 1, 2, ..., N$*, if there exists a hyperplane* S

$$w \cdot x + b = 0$$

that can separate the positive and negative samples completely to both sides of the hyperplane. Then T *is said to be linearly separable.*

Algorithm 1.1 (Perceptron)**.**
Input: *training data* $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$*, where* $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{1, -1\}, i = 1, 2, ..., N$*; learning rate* $0 < \eta \leq 1$
Output: w, b*; perceptron* $f(x) = sign(w \cdot x + b)$

1. *initialize* w_0, b_0
2. *pick* (x_i, y_i) *from* T
3. *if* $y_i(w \cdot x_i + b) \leq 0$

$$\begin{aligned} w &\leftarrow w + \eta y_i x_i \\ b &\leftarrow b + \eta y_i \end{aligned} \tag{1}$$

4. *return to (2) until there is no misclassified point*

Theorem 1.1 (Novikoff)**.** *Let the training data* $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ *be linearly separable, where* $x_i \in \mathcal{X} = \mathbb{R}^n$*,* $y_i \in \mathcal{Y} = \{1, -1\}, i = 1, 2, ..., N$*, then*

1. *there exists* $\|\hat{w}_{opt}\| = 1$ *such that the hyperplane* $\hat{w}_{opt} \cdot \hat{x} = w_{opt} \cdot x + b_{opt} = 0$ *separates the training data perfectly. Moreover, there exists a* $\gamma > 0$ *such that*

$$y_i(\hat{w}_{opt} \cdot \hat{x}_i) = y_i(w_{opt} \cdot x_i + b_{opt}) \geq \gamma \tag{2}$$

2. *Set* $R = \max_{1 \leq i \leq N} \|\hat{x}_i\|$*, then the misclassification* k *satisfies*

$$k \leq \left(\frac{R}{\gamma}\right)^2 \tag{3}$$

Algorithm 1.2 (Perceptron-Dual Form)**.**
Input: *traning data* $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$*, where* $x_i \in \mathbb{R}^n, y_i \in \{1, -1\}, i = 1, 2, ..., N$*; learning rate* $0 < \eta \leq 1$
Output: a, b*; perceptron* $f(x) = sign\left(\sum_{j=1}^{N} a_j y_j x_j \cdot x + b\right)$*, where* $a = (a_1, ..., a_N)^T$*.*

1. $a \leftarrow 0, b \leftarrow 0$;

2. *pick* (x_i, y_i) *from* T;

3. *if* $y_i \left(\sum_{j=1}^{N} a_j y_j x_j \cdot x_i + b \right) \leq 0$,

$$\begin{aligned} a_i &\leftarrow a_i + \eta \\ b &\leftarrow b + \eta y_i \end{aligned} \tag{4}$$

4. *return to (2) until no misclassfied point*

Definition 1.2 (Gram Matrix)**.**

$$\boldsymbol{G} := [x_i \cdot x_j]_{N \times N} \tag{5}$$

Example 1.1. ***Prove the theorem:*** *data T is linearly separable if and only if the convex hull*[1] *formed by positive samples does not intersect with the convex hull formed by negative samples*

Proof. For the forward direction, show that for any point x in the positive convex hull, we have $w \cdot x + b > 0$ and similarly for any point in the negaive convex hull. Conversely, it can be proved by notice that two disjoint convex sets can be separated by a hyperplane. □

2 K Nearest Neighbors

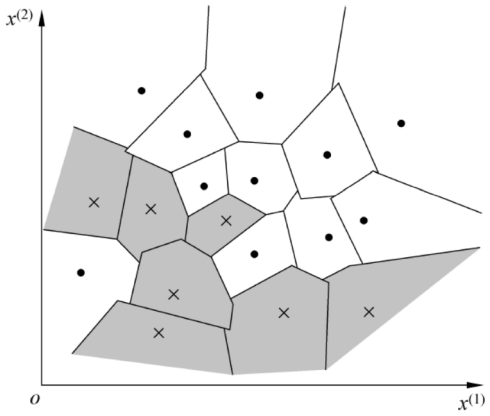

Figure 1: The corresponding division of space by K-nearest neibors model

3 Naive Bayes

Naive Bayes is a typical generative learning method. From the given data, learn the joint distribution $P(X, Y)$, then obtain the posterior $P(Y|X)$. Specifically, learn $P(X|Y)$ and $P(Y)$ from the data, and then get

$$P(X, Y) = P(X|Y)P(Y)$$

The basic assumption of Naive Bayes is conditional independence of features given the class label

$$P(X = x|Y = c_k) = \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_k) \tag{7}$$

[1] Let $S = \{x_1, x_2, ..., x_k\} \subset \mathbb{R}^n$.

$$conv(S) := \left\{ x = \sum_{i=1}^{k} \lambda_i x_i | \sum_{i=1}^{k} \lambda_i = 1, \lambda_i \geq 0, i = 1, 2, ..., k \right\} \tag{6}$$

This assumption is strong and hence the models are restrictive. However, it makes the learning and prediction easy. Moreover, Naive Bayes is efficient and easy to implement. *One disadvantage is the prediction accuracy might be low.*
Naive Bayes uses Bayes' theorem and the learned joint distribution to perform prediction

$$P(Y|X) = \frac{P(X,Y)}{P(X)} = \frac{P(Y)P(X|Y)}{\sum_Y P(Y)P(X|Y)} \tag{8}$$

then assign x to the y such that

$$y := arg\max_{c_k} P(Y = c_k) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_k) \tag{9}$$

Maximizing the posterior $P(Y|X = x)$ is equivalent to minimizing the expected risk of 0-1 loss function.

3.1 Maximum Likelihood Estimation

Algorithm 3.1 (Naive Bayes)**.**
Input: *training data* $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$*, where* $x_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(n)})^T$*,* $x_i^{(j)}$ *is the jth feature of the ith sample,* $x_i^{(j)} \in \left\{a_{j1}, a_{j2}, ..., a_{jS_j}\right\}$*,* a_{jl} *is the lth possible value of the jth feature,* $j = 1, 2, ..., n$*,* $l = 1, 2, ..., S_j$*,* $y_i \in \{c_1, c_2, ..., c_K\}$*; instance* x*;*
Output: *the label of* x

1. *calculate prior probability and conditional probability*

$$\begin{aligned} P(Y = c_k) &= \frac{\sum_{i=1}^{N} I(y_i = c_k)}{N}, k = 1, 2, ..., K \\ P(X^{(j)} = a_{jl}|Y = c_k) &= \frac{\sum_{i=1}^{N} I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^{N} I(y_i = c_k)}, j = 1, 2, ..., n, l = 1, 2, ..., S_j, k = 1, 2, ..., K \end{aligned} \tag{10}$$

2. *for the given instance* $x = (x^{(1)}, x^{(2)}, ..., x^{(n)})^T$*, calculate*

$$P(Y = c_k) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_k), k = 1, 2, ..., K \tag{11}$$

3. *determine the label of* x

$$y = arg\max_{c_k} P(Y = c_k) \prod_{j=1}^{n} P(X^{(j)} = x^{(j)}|Y = c_k) \tag{12}$$

3.2 Bayesian Estimation

One problem with MLE is that the probability might be 0. To solve this problem, we adopt *Bayesian Estimation.*

$$P_\lambda(X^{(j)} = a_{jl}|Y = c_k) = \frac{\sum_{i=1}^{N} I(x_i^{(j)} = a_{jl}, y_i = c_k) + \lambda}{\sum_{i=1}^{N} I(y_i = c_k) + S_j\lambda} \tag{13}$$

where $\lambda \geq 0$, when $\lambda = 0$, it is equivalent to MLE. Usually $\lambda = 1$, which is called *Laplacian Smoothing.* Obviously,

$$\begin{aligned} &P_\lambda(X^{(j)} = a_{jl}|Y = c_k) > 0 \\ &\sum_{l=1}^{S_j} P(X^{(j)} = a_{jl}|Y = c_k) = 1 \end{aligned}$$

Similarly, the Bayesian estimation of the prior probability is

$$P_\lambda(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k) + \lambda}{N + K\lambda} \tag{14}$$

4 Decision Tree

4.1 Feature Selection

Feature selection is to decide which features should be used to partition the feature space $\mathcal{X}$. The criterion could be *information gain* or *information gain ratio*.

Definition 4.1 (Entropy)**.** *the entropy of a random variable X is defined as*

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i \tag{15}$$

Remark 4.1.

- *entropy measures the uncertainty of a random variable*
- *the unit of entropy could be bit or nat*
- *$H(X)$ only depends on the distribution of X. Hence, we writr $H(X) = H(p)$*
- $0 \leq H(p) \leq \log n$

Definition 4.2 (Conditional Entropy)**.**

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{16}$$

Algorithm 4.1 (Information Gain)**.**
Input: *training data D and feature A*
Output: *the information gain $g(D, A)$ of feature A to data D*

1. *calculate empirical entropy*

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|} \tag{17}$$

2. *calculate the conditional empirical entropy*

$$H(D|A) = sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|} \tag{18}$$

3. *calculate the information gain*

$$g(D, A) = H(D) - H(D|A) \tag{19}$$

Definition 4.3 (Information Gain Ratio)**.** *the information gain ratio of feature A to data D, $g_R(D, A)$ is defined as*

$$g_R(D, A) := \frac{g(D, A)}{H_A(D)}. \tag{20}$$

where $H_A(D) := -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$

4.2 ID3

The core idea of ID3 algorithm is selecting features on each node by applying the *information gain criterion* and recursively generate a decision tree.

Algorithm 4.2 (ID3)**.**
Input: *training data D, threshold ϵ*
Output: *decision tree T*

1. *if all samples in D belong to the same C_k, then T has only one node. Set C_k as the class label of the node, return T*

2. *if $A = \emptyset$, then T has only one node. Set C_k, the most frequent label as the node label, return T*

3. *Otherwise, apply algorithm 4.1 to calculate information gain for each feature, choose the feature A_g with the highest information gain*

4. *if $g(D, A_g) < \epsilon$, set T to be an one-node tree, let C_k, the most frequent label as the node label, return T*

5. *Otherwise, for each possible value a_i of A_g, partition D into disjoint nonempty subset D_i, set the most frequent label as the node label. Return T*

6. *for the ith node, use D_i as the training data, $A - \{A_g\}$ as feature collection, recursively invoke 1 5*

4.3 C4.5

C4.5 improves ID3 by adopting the information gain ratio as feature selection criterion.

Algorithm 4.3 (C4.5).
Input: *training data D, threshold ϵ*
Output: *decision tree T*

1. *if all samples in D belong to the same C_k, then T has only one node. Set C_k as the class label of the node, return T*

2. *if $A = \emptyset$, then T has only one node. Set C_k, the most frequent label as the node label, return T*

3. *Otherwise, apply equation (20) to calculate information gain ratio for each feature, choose the feature A_g with the highest information gain ratio*

4. *if $g(D, A_g) < \epsilon$, set T to be an one-node tree, let C_k, the most frequent label as the node label, return T*

5. *Otherwise, for each possible value a_i of A_g, partition D into disjoint nonempty subset D_i, set the most frequent label as the node label. Return T*

6. *for the ith node, use D_i as the training data, $A - \{A_g\}$ as feature collection, recursively invoke 1 5*

4.4 Pruning

Applying generation algorithm to construct decision trees tends to produce overfitting. The operation to *simplifify the tree* is called *pruning*. Pruning is often realized by minimizing the loss function or cost function of the decision tree. Let $|T|$ be the number of nodes, t be the leaf node of T. There are N_t samples in t and N_{tk} samples belong to calss k. $H_t(T)$ is the empirical entropy of leaf node t. $\alpha \geq 0$ is a hyperparameter. The loss function could be defined as

$$C_\alpha(T) := \sum_{t=1}^{|T|} N_t H_T(t) + \alpha |T| \tag{21}$$

where

$$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

write

$$C(T) := \sum_{t=1}^{|T|} N_t H_T(t) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

then we have

$$C_\alpha(T) = C(T) + \alpha |T| \tag{22}$$

where $C(T)$ measures the fitting effect of the model and $\alpha|T|$ measures the complexity of the model.

Algorithm 4.4 (Pruning).
Input: *tree T, hyperparameter α*
Output: *pruned tree T_α*

1. *calculate the empirical entropy of each node*
2. *recursively retract from leaf node: Set the pre-retracted tree and post-retracted tree as T_B and T_A respectively. if*
$$C_\alpha(T_A) \leq C_\alpha(T_B) \tag{23}$$
then prune the tree.
3. *back to (2), until it stops to get the tree T_α with minimum loss*

4.5 CART

4.5.1 Least Square Regression Tree

Algorithm 4.5 (Least Square Regression Tree).
Input: *training data D*
Output: *regression tree $f(x)$*

1. *choose the best splitting variable j and point s: solve*
$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_1-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{24}$$
2. *use the chosen (j,s) to partition the space*
$$\begin{aligned} &R_1(j,s)=\left\{x|x^{(i)}\leq s\right\}, R_2(j,s)=\left\{x|x^{(j)}>s\right\} \\ &\hat{c}_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)} y_i, x\in R_m, m=1,2 \end{aligned} \tag{25}$$
3. *invoke (1),(2) for the two subspaces generated, until the stopping criterion is satisfied*
4. *partition the input space into M subspaces $R_1, R_2, ..., R_M$, generate the decision tree*
$$f(x)=\sum_{m=1}^{M}\hat{c}_m I(x\in R_m) \tag{26}$$

4.5.2 Classification Tree

classification tree uses *gini index* to select best features.

Algorithm 4.6.
Input: *training data D, stopping criterion*
Output: *CART decision tree*

1. *for each feature A and each possible value a of this feature, calculate $Gini(D,A)$. Partition D into D_1 and D_2*
2. *choose the feature A and the value a with the smallest gini index, generate two sub-nodes*
3. *recursively apply (1) (2) until the stopping criterion is satisfied*
4. *generate the CART decision tree*

Definition 4.4 (Gini Index). *suppose there are K classes, then the gini index of a distribution is*
$$Gini(p):=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2 \tag{27}$$

5 Logistic Regression and Maximum Entropy Model

5.1 Logistic Regression Model

Definition 5.1 (Logistic Distribution)**.** *Let X be a continuous random variable, X follows logistic distribution if it has the following distribution function and density function*

$$\begin{aligned} F(x) = P(X \leq x) &= \frac{1}{1 + e^{-(x-\mu)/\gamma}} \\ f(x) = F'(x) &= \frac{e^{-(x-\mu)/\gamma}}{\gamma(1 + e^{-(x-\mu)/\gamma})^2} \end{aligned} \tag{28}$$

5.2 Binomial Logistic Regression Model

Binomial logistic regression model is a kind of *classification* model, which is represented by $P(Y|X)$.

Definition 5.2 (Logistic Regression Model)**.**

$$\begin{aligned} P(Y = 1|x) &= \frac{exp(w \cdot x + b)}{1 + exp(w \cdot x + b)} \\ p(Y = 0|x) &= \frac{1}{1 + exp(w \cdot x + b)} \end{aligned} \tag{29}$$

where $x \in \mathbb{R}^n$ is the input data and $Y \in \{0, 1\}$ is the output, $w \in \mathbb{R}^n, b \in \mathbb{R}$ are parameters. For simplicity, we can write $w := (w, b)^T \in \mathbb{R}^{n+1}$

Definition 5.3.

$$logit(p) = \log \frac{p}{1 - p} \tag{30}$$

In terms of logistic regression, we have

$$\log \frac{P(Y = 1|x)}{1 - P(Y = 1|x)} = w \cdot x \tag{31}$$

5.3 Model Parameter Estimation

Given data $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}, x_i \in \mathbb{R}^n, y_i \in \{0, 1\}$, we can apply *Maximum Likelihood Estimation* to learn the model parameter. Set

$$P(Y = 1|x) = \pi(x), \quad P(Y = 0|x) = 1 - \pi(x)$$

the likelihood function is

$$\prod_{i=1}^{N} [\pi(x_i)]^{y_i} [1 - \pi(x_i)]^{1-y_i}$$

the log-likelihood is

$$L(\omega) = \sum_{i=1}^{N} [y_i(\omega \cdot x_i) - \log(1 + exp(\omega \cdot x_i))]$$

maximize the log-likehood, one can get the model parameter $\hat{\omega}$

5.4 Multi-nominal Logistic Regression Model

Definition 5.4 (Multi-nominal Logistic Regression Model)**.**

$$\begin{aligned} P(Y = k|x) &= \frac{exp(\omega_k \cdot x)}{1 + \sum_{k=1}^{K-1} exp(\omega_k \cdot x)}, k = 1, 2, ..., K - 1 \\ P(Y = K|x) &= \frac{1}{1 + \sum_{k=1}^{K-1} exp(\omega_k \cdot x)} \end{aligned} \tag{32}$$

where $x \in \mathbb{R}^{n+1}, \omega_k \in \mathbb{R}^{n+1}, k = 1, 2, ..., K$

The model parameter ω_k can be learned in a similar way.

5.5 Maximum Entropy Model

Definition 5.5 (Maximum Entropy Principle)**.** *Choose the model satisfying all the constraints with the largest entropy.*

Definition 5.6 (Maximum Entropy Model)**.** *Suppose the collection of all probabilistic models satisfying the constraints is*

$$\mathcal{C} = \{P \in \mathcal{P} | \mathbb{E}_P(f_i) = \mathbb{E}_{\tilde{P}}(f_i), i = 1, 2, ..., n\} \tag{33}$$

the conditional entropy defined on the conditional distribution $P(Y|X)$ is

$$H(P) = -\sum_{x,y} \tilde{P}(x) P(y|x) \log P(y|x) \tag{34}$$

then the model in $\mathcal{C}$ which has the largest entropy is called the maximum entropy model.

Suppose the classification model is a conditional distribution $P(Y|X)$, $X \in \mathcal{X} \subset \mathbb{R}^n, Y \in \mathcal{Y}$, given the training data $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$. The learning objective is to choose the best model according to the *maximum entropy principle*.

Definition 5.7 (Empirical Distribution)**.**

$$\tilde{P}(x, y) = \frac{v(x, y)}{N}; \quad \tilde{P}(x) = \frac{v(x)}{N} \tag{35}$$

where v counts the number of occurence

5.6 Improved Iterative Scaling

Improved Iterative Scaling (IIS) is a learning algorithm for maximimum entropy model. Recall that the maximum entropy model is

$$P_\omega(y|x) = \frac{1}{Z_\omega(x)} exp\left(\sum_{i=1}^{n} \omega_i f_i(x, y)\right) \tag{36}$$

where

$$Z_\omega(x) = \sum_y exp\left(\sum_{i=1}^{n} \omega_i f_i(x, y)\right) \tag{37}$$

the log-likelihood function is

$$L(\omega) = \sum_{x,y} \tilde{P}(x, y) \sum_{i=1}^{n} \omega_i f_i(x, y) - \sum_x \tilde{P} \log Z_\omega(x) \tag{38}$$

The objective is to learn the model parameter $\hat{\omega}$ by MLE.

Algorithm 5.1.
Input: *feature functions $f_1, f_2, ..., f_n$; empirical distribution $\tilde{P}(x, y)$, model $P_\omega(y|x)$;*
Output: *optimal parameter ω_i^*; optimal model P_{ω^*}*

1. *for $i = 1, 2, ..., n$, $\omega_i \leftarrow 0$*
2. *for $i = 1, 2, ..., n$,*
   (a) *solve for δ_i*
   $$\sum_{x,y} \tilde{P}(y) P(y|x) f_i(x, y) exp(\delta_i f^\#(x, y)) = \mathbb{E}_{\tilde{P}(f_i)} \tag{39}$$
   (b) *Update ω_i: $\omega_i \leftarrow \omega_i + \delta_i$*
3. *if not all ω_i converge, continue (2)*

6 Support Vector Machine

The basic idea of *Support Vector Machine* is to maximize the margin, which makes it different from perceptron. This idea could be formulated as a convex quadratic programming. The problem could be classified into several cases:

- linear support vector machine in linearly separable case
- linear support vector machine
- non-linear support vector machine

6.1 Linearly Separable Case

Definition 6.1 (Linearly Separable Support Vector Machine)**.** *Given linearly separable data, by maximizing the margin (equivalently solving corresponding convex quadratic programming), the hyperplane*

$$\omega^* \cdot x + b^* = 0 \tag{40}$$

and the decision function

$$f(x) = sign(\omega^* \cdot x + b^*) \tag{41}$$

is called the linearly separable support vector machine.

Definition 6.2 (Functional Margin)**.**

$$\hat{\gamma} = \min_{i=1,2,...,N} \hat{\gamma}_i \tag{42}$$

where $\hat{\gamma}_i = y_i(\omega \cdot x_i + b)$

Definition 6.3 (Geometric Margin)**.**

$$\gamma = \min_{i=1,...,N} \gamma_i \tag{43}$$

where $\gamma_i = y_i \left(\frac{\omega}{\|\omega\|} \cdot x_i + \frac{b}{\|\omega\|} \right)$

Theorem 6.1 (Existence and Uniqueness)**.** *If the training data T is linearly separable, then the hyperplane with largest margin exists and is unique.*

Algorithm 6.1 (Linearly Separable Support Vector Machine Learning - Margin Maximization)**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}$, *where* $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{-1, 1\}$
Output: *hyperplane and decision function*

1. *solve the constrained optimization problem*

$$\begin{aligned} &\min_{\omega,b} \frac{1}{2}\|\omega\|^2 \\ &s.t\ y_i(\omega \cdot x_i + b) - 1 \geq 0,\ i = 1, 2, ..., N \end{aligned} \tag{44}$$

   to get optimal ω^*, b^*

2. *obtain the hyperplane and decision function*

$$\begin{aligned} &\omega^* \cdot x + b^* = 0 \\ &f(x) = sign(\omega^* \cdot x + b^*) \end{aligned} \tag{45}$$

Algorithm 6.2 (Linearly Separable Support Vector Machine Learning - Margin Maximization (Dual Form))**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}$, *where* $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{-1, 1\}$
Output: *hyperplane and decision function*

1. *solve the constrained optimization problem*

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i \\ &\sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\alpha_i \geq 0, i = 1, 2, ..., N \end{aligned} \tag{46}$$

obtain the optimal solution $\alpha^* = (\alpha_1^*, ..., \alpha_N^*)^T$

2. *calculate*

$$\omega^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i \tag{47}$$

choose a component of $\alpha_j > 0$*, calculate*

$$b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i (x_i \cdot x_j) \tag{48}$$

3. *obtain the hyperplane and decision function*

$$\begin{aligned} &\omega^* \cdot x + b^* = 0 \\ &f(x) = sign(\omega^* \cdot x + b^*) \end{aligned} \tag{49}$$

It is easy to see that ω^* and b^* only depend on the sample (x_i, y_i) such that $\alpha_i^* > 0$. Those vectors are called *Support Vectors*.

6.2 Not Linear Separable Case

basic idea: introduce slack variables ξ_i. Learning for the *Linear Support Vector Machine* is equivalently to solve the following convex quadratic programming:

$$\begin{aligned} &\min_{\omega,b,\xi} \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t.\ y_i(\omega \cdot x_i + b) \geq 1 - \xi_i,\ i = 1, 2, ..., N \\ &\quad \xi_i \geq 0,\ i = 1, 2, ..., N \end{aligned} \tag{50}$$

Definition 6.4 (Linear Support Vector Machine)**.** *For given possibly not linear separable data, by solving* (50)*, the hyperplane*

$$\omega^* \cdot x + b^* = 0 \tag{51}$$

and the decision function

$$f(x) = sign(\omega^* \cdot x + b^*) \tag{52}$$

together is called the linear support vector machine.

Proposition 6.2 (Dual Problem)**.** *The dual problem of* (50) *is*

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N}\alpha_i \\ &s.t.\ \sum_{i=1}^{N}\alpha_i y_i = 0 \\ &\quad 0 \leq \alpha_i \leq C,\ i = 1, 2, ..., N \end{aligned} \tag{53}$$

Algorithm 6.3 (Linear Support Vector Machine)**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}$, *where* $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{-1, 1\}$
Output: *hyperplane and decision function*

1. *choose penalty parameter* $C \geq 0$, *solve*

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N} \alpha_i \\ &s.t. \ \sum_{i=1}^{N} \alpha_i y_i = 0 \\ &\quad 0 \leq \alpha_i \leq C, \ i = 1, 2, ..., N \end{aligned} \tag{54}$$

   obtain optimal $\boldsymbol{\alpha}^* = (\alpha_1^*, ..., \alpha_N^*)^T$

2. *calculate* $\omega^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$; *choose a* $0 < \alpha_j < C$ *and calculate* $b^* = y_j - \sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x_j)$

3. *obtain the hyperplane* $\omega^* \cdot x + b^*$ *and the decision function* $f(x) = sign(\omega^* \cdot x + b^*)$

Remark 6.1.
- *the vectors* x_i *such that* $0 < \alpha_i \leq C$ *is called support vectors. The distance from* x_i *to separate boundary is* $\frac{\xi_i}{\|\omega\|}$.
- *if* $\alpha_i^* < C$, *then* $\xi_i = 0$, *which means* x_i *sits on the boundary*
- $\alpha_i^* = C, 0 < \xi_i < 1$, *it is correctly classified;* $\alpha_i^* = C, \xi_i = 1$, *it sits on the hyperplane;* $\alpha_i^* = C, \xi_i > 1$, *it is incorrectly classified.*

Definition 6.5 (Hinge Loss Function)**.**

$$[z]_+ = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases} \tag{55}$$

Theorem 6.3. *The problem* (50) *is equivalent to*

$$\min_{\omega, b} \sum_{i=1}^{N} [1 - y_i(\omega \cdot x_i + b)]_+ + \lambda \|\omega\|^2 \tag{56}$$

(**Hint:** set $\xi_i = [1 - y_i(\omega \cdot x_i + b)]$)

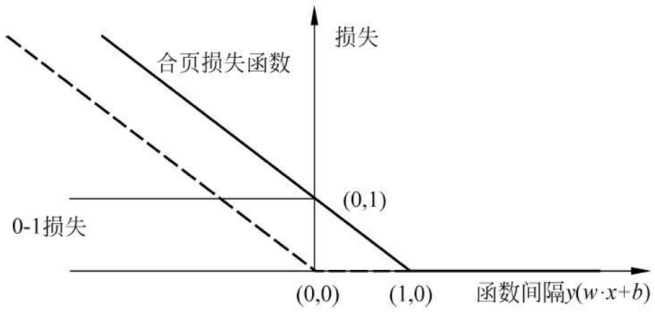

Figure 2: Hinge Loss Function

6.3 Non-linear Support Vector Machine and Kernel Function

6.3.1 Kernel Trick

Given training data $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}, x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{-1, +1\}, i = 1, 2, ..., N$. If one can separate data by a hypersurface in $\mathbb{R}^n$, then the problem is said to be a *nonlinear separable problem.*
The basic idea to solve such a problem is to define a nonlinear transformation so that the nonlinear problem could be converted into a linear problem. Solving the linear problem is thus equivalent to solve the original nonlinear problem. As shown in fig 3, the ellipse is transformed into a line. The

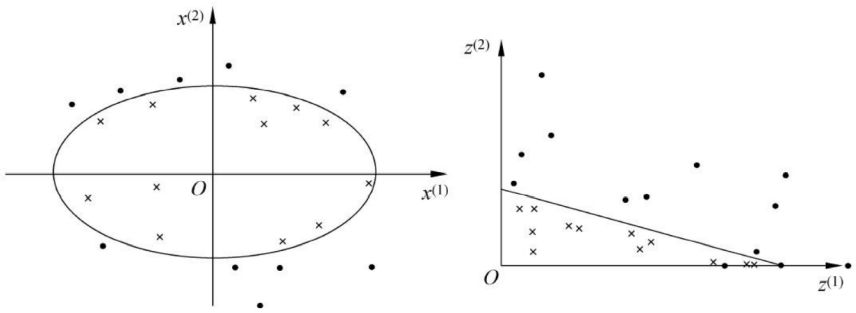

Figure 3: Nonlinear separable problem and kernel trick

example above shows that solving nonlinear classification problem by linear classification methods could be divided into two steps:

1. define a nonlinear transformation $\phi : \mathcal{X} \to \mathcal{H}$
2. apply linear classification methods in the feature space $\mathcal{H}$

Kernel trick is one specific kind of this method. The basic idea is learn a hyperplane in $\mathcal{H}$ which corresponds to a hypersurface in $\mathcal{X}$.

Definition 6.6 (Kernel Function)**.** *$\mathcal{X}$ is the input space (could be discrete sets or $\mathbb{R}^n$) and $\mathcal{H}$ is the transformed feature space (Hilbert space), if there exists a mapping*

$$\phi(x) : \mathcal{X} \to \mathcal{H} \tag{57}$$

such that for all $x, z \in \mathcal{X}$, function $K(x, z)$ satisfies

$$K(x, z) = \phi(x) \cdot_{\mathcal{H}} \phi(z) \tag{58}$$

then $K(x, z)$ is called the kernel function

In the learning process, one only defines the kernel function K, rather than explicitly define $\mathcal{H}$ and ϕ. This simplifies the process since usually the feature space has very high dimension (even has infinite dimension). Given K, the choice for ϕ and $\mathcal{H}$ is not unique.
Based on the discussion above, we can apply kernel trick in support vector machine. Use $k(x_i, x_j)$ in place of $x_i \cdot x_j$, the objective of dual problem (6.2) becomes

$$W(\alpha) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i \tag{59}$$

Similarly, the inner product in decision functions can also be replaced by kernel function.

6.3.2 Positive Definite Kernel Function

Theorem 6.4. *Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric function, then $K(x, z)$ is a positive definite kernel function if and only if for all $x_i \in \mathcal{X}, i = 1, 2, ..., m$, the correspondig gram matrix of $K(x, z)$*

$$K = [K(x_i, x_j)]_{m \times m} \tag{60}$$

is positive semi-definite

Definition 6.7 (Positive Definite Kernel Function)**.** *Let $\mathcal{X} \subset \mathbb{R}^n$, $K(x,z)$ is a symmetric function defined on $\mathcal{X} \times \mathcal{X}$, if for all $x_i \in \mathcal{X}, i = 1,2,...,m$, the gram matrix corresponding to $K(x,z)$*

$$K = [K(x_i, x_j)]_{m \times m} \tag{61}$$

is positive semi-definite, then $K(x,z)$ is a positive kernel function

Although we can check if a given $K(x,z)$ is a kernel function by applying the definition, in practise it is difficult to check the gram matrix is PSD. Hence, we tend to use some existing kernel functions.

6.3.3 Common Kernel Functions

Definition 6.8 (Common Kernal Functions)**.**

- ***Polynomial Kernel Function***

$$K(x,z) = (x \cdot z + 1)^p \tag{62}$$

- ***Guassian Kernel Function***

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) \tag{63}$$

- ***String Kernel Function***

$$k_n(s,t) = \sum_{u \in \Sigma_n} [\phi_n(s)]_u [\phi_n(t)]_u = \sum_{u \in \Sigma_n} \sum_{(i,j): s(i)=t(j)=u} \lambda^{l(i)} \lambda^{l(j)} \tag{64}$$

6.3.4 Non-linear Support Vector Machine

Definition 6.9 (Non-Linear Support Vector Machine)**.** *Given non-linear training data, maximize the soft margin*

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_i K(x, x_i) + b^*\right) \tag{65}$$

is called nonlinear support vector machine, $K(x,z)$ is positive definite kernel function

Algorithm 6.4 (Non-linear Support Vector Machine)**.**
Input: *training data $T = \{(x_1, y_1), ..., (x_N, y_N)\}$, where $x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{-1, +1\}$*
Output: *decision function*

1. *choose appropriate kernel $K(x,z)$ and parameter C, solve*

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} \alpha_i \\ &s.t. \ \sum_{i=1}^{N} \alpha_i y_i = 0 \\ &\quad 0 \le \alpha_i \le C, \ i = 1,2,...,N \end{aligned} \tag{66}$$

   to obtain optimal $\alpha^* = (\alpha_1^*, ..., \alpha_N^*)^T$

2. *choose a $0 < \alpha_j^* < C$, calculate*

$$b^* = y_j - \sum_{i=1}^{N} \alpha_i^* y_i K(x_i, x_j) \tag{67}$$

3. *decision function*

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i^* y_k K(x_i, x_j +) b^*\right) \tag{68}$$

when K is positive definite, the optimization problem (66) is convex and thus the solution exists.

6.3.5 Sequential Minimal Optimization

Algorithm 6.5 (Sequential Minimal Optimization).
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}$,$x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, ..., N, precision\epsilon$;
Output: *approximate solution to* (66), $\hat{\alpha}$

1. $\alpha^{(0)} \leftarrow 0$; *set* $k = 0$
2. *choose* $\alpha_i^{(k)}, \alpha_2^{(k)}$, *solve the subproblem, obtain* $\alpha_1^{(k+1)}, \alpha_2^{(k+1)}$, *update* $\alpha \leftarrow \alpha_i^{(k+1)}$
3. *if under precision* ϵ, *the stopping criterion is satisfied*

$$\sum_{i=1}^{N} \alpha_i y_i = 0, 0 \le \alpha_i \le C, i = 1, 2, ..., N$$
$$y_i \cdot g(x_i) \begin{cases} \ge 1, & \{x_i | \alpha_i = 0\} \\ = 1, & \{x_i | 0 < \alpha_i < C\} \\ \le 1, & \{x_i | \alpha_i = C\} \end{cases} \tag{69}$$

   go to (4), otherwise set $k \leftarrow k + 1$, *go to (2)*

4. $\hat{\alpha} \leftarrow \alpha^{(k+1)}$

7 Boosting

Boosting is a common statistical method which is popular and widely applied. In classification problem, by changing the weights of samples, it can learn multiple classifiers, the classification accuracy can be promoted by linearly combining weak classifiers.
If there exists a polynomial algorithm that can learn a concept with high accuracy, then the concept is said to be *strongly learnable*. If there exists a polynomial which learns the concept with an accuracy slightly better than random guess, the concept is said to be *weakly learnable*. In fact, Schapire proved that strongly leanable and weakly learnable are equivalent.

7.1 AdaBoost

Algorithm 7.1 (AdaBoost).
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}, x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \{+1, -1\}$; *weak learning algorithm;*
Output: *strong classifier* $G(x)$

1. *initialize the weights* $D_1 \leftarrow (\omega_{11}, ..., \omega_{1N}), \omega_{1i} \leftarrow \frac{1}{N}, i = 1, 2, ..., N$
2. *for* $m = 1, 2, ..., M$
   (a) *use the weighted data to learn a basic classifier* $G_m(x) : \mathcal{X} \to \mathcal{Y}$
   (b) *calculate the classification error*

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \ne y_i) = \sum_{i=1}^{N} \omega_{mi} I(G_m(x_i) \ne y_i) = \sum_{i: G_m(x_i) \ne y_i} w_{mi} \tag{70}$$

   (c) *calculate the coefficient of* $G_m(x)$:

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \tag{71}$$

   (d) *update the weights*

$$\begin{aligned} &D_{m+1} \leftarrow (\omega_{m+1,1}, ..., \omega_{m+1,N}) \\ &\omega_{m+1,i} = \frac{\omega_{m,i}}{Z_m} \exp\left(-\alpha_m y_i G_m(x_i)\right), i = 1, 2, ..., N \\ &Z_m = \sum_{i=1}^{N} \omega_{mi} \exp\left(-\alpha_m y_i G_m(x_i)\right) \end{aligned} \tag{72}$$

3.

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) \tag{73}$$

and obtain the classifier

$$G(x) = sign(f(x)) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{74}$$

7.2 Error Analysis

Theorem 7.1 (Training Error Bound of AdaBoost)**.** *The error bound of AdaBoost is*

$$\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i) \leq \frac{1}{N}\sum_{i} \exp\left(-y_i f(x_i)\right) = \prod_{m} Z_m \tag{75}$$

7.3 Additive Model and Forward Stagewise Algorithm

Algorithm 7.2 (Forward Stagewise Algorithm)**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}$*; loss funciton* $L(y, f(x)$*; basis function* $\{b(x; \gamma)\}$*;*
Output: *additive model* $f(x)$

1. *initialize* $f_0(x) = 0$
2. *for* $m = 1, 2, ..., M$*:*
   (a) *minimize the loss function*

$$(\beta_m, \gamma_m) = arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) \tag{76}$$

   (b) *update* $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$
3. *obtain additive model*

$$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{77}$$

Remark 7.1. *AdaBoost could be viewed as a special case of forward stagewise algorithm + addtive model where the loss function is chosen to be* $L(y, f(x)) = \exp[-yf(x)]$

7.4 Boosting Tree

Boosting tree uses *regression tree* or *classification tree* as basic classifier. Boosting tree model could be represented as the additive model of decision trees

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{78}$$

where $T(x; \Theta_m)$ is a decision tree, Θ_m is the parameter, M is the number of trees

Algorithm 7.3 (Boosting for Regression Tree)**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}, x_i \in \mathcal{X} = \mathbb{R}^n, y_i \in \mathcal{Y} \subset \mathbb{R}$*;*
Output: *boosting tree* $f_M(x)$

1. *initialize* $f_0(x) = 0$
2. *for* $m = 1, 2, ..., M$:

(a) *calculate the residual*

$$r_{mi} = y_i - f_{m-1}(x_i), i = 1, 2, ..., N \tag{79}$$

(b) *fit* r_{mi} *to learn a regression tree* $T(x; \Theta_m)$

(c) *update* $f_m(x) = f_{m-1}(x) + T(x; \Theta_m)$

3. *obtain the boosting tree for regression problem*

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{80}$$

7.5 Gradient Boosting

when the loss function is *exponential function* or *square loss function*, the optimization of each step $(\beta_m, \gamma_m) = arg \min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$ is easy. However, for general loss function, the optimizaiton might not be easy. Freidman proposed *gradient boosting*. The basic idea is to use $-\frac{\partial L(y, f(x_i))}{\partial f(x_i)}_{f(x)=f_{m-1}(x)}$ to approximate the residual r_{mi}.

Algorithm 7.4 (Gradient Boosting)**.**
Input: *training data* $T = \{(x_1, y_1), ..., (x_N, y_N)\}, x_i \in \mathcal{X} \subset \mathbb{R}^n, y_i \in \mathcal{Y} \subset \mathbb{R}$*, loss function* $L(y, f(x))$
Output: *regression tree* $\hat{f}(x)$

1. *initialize*

$$f_0(x) = arg \min_c \sum_{i=1}^{N} L(y_i, c) \tag{81}$$

2. *for* $m = 1, 2, ..., M$:

(a) *for* $i = 1, 2, ..., N$:*, calculate*

$$r_{mi} = -\left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)} \tag{82}$$

(b) *fit* r_{mi} *to learn a regression tree, obtain the leaf regions* $R_{mj}, j = 1, 2, ..., J$ *of the mth tree*

(c) *for* $j = 1, 2, ..., J$*, calculate*

$$c_{mj} = arg \min_c \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c) \tag{83}$$

(d) *update* $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$

3. *obtain the regression tree*

$$\hat{f}(x) = f_M(x) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_{mj}) \tag{84}$$

8 Expectation Maximization

EM algorithm is a kind of *iterative* algorithm. It is used in the maximum likelihood estimation of model containing hidden variables. If all the variables of a probabilistic model is observable, one can directly apply MLE or Bayesian estimation. However, if there are hidden variables, those methods could not be applied.

8.1 Introduction

8.1.1 Convergence

Theorem 8.1 (Monotonicity). *Let $P(Y|\theta)$ be the likelihood of observations, $\{\theta^{(i)}\}$ be the sequence obtained by EM algorithm, $P(Y|\theta^{(i)})$ is the corresponding sequence of likelihood. Then, $P(Y|\theta^{(i)})$ is monotonically increasing.*

$$P(Y|\theta^{(i+1)}) \geq P(Y|\theta^{(i)}) \tag{85}$$

Theorem 8.2. *Let $L(\theta) = \log P(Y|\theta)$ be the log-likelihood of observations, $\theta^{(i)}$ is the sequence obtained by EM, $L(\theta^{(i)})$ is the corresponding sequence of log-likelihood.*

1. *if $P(Y|\theta)$ is upper bounded, then $L(\theta^{(i)})$ converges to some L^**
2. *if $Q(\theta, \theta')$ and $L(\theta)$ satisfy some conditions, then the limit of $\theta^{(i)}$, θ^* is a stationary point of $L(\theta)$*

8.1.2 Application in Gaussian Mixture Model

9 Appendix

9.1 Bounds in Probability[Duc17]

Proposition 9.1 (Markov's Inequality). *Let $Z \geq 0$ be a nonnegative random variable. Then for all $t \geq 0$,*

$$\mathbb{P}(Z \geq t) \leq \frac{Z}{t} \tag{86}$$

Proposition 9.2. *Let Z be any random variable with $Var(Z) < \infty$. Then*

$$\mathbb{P}(|Z - \mathbb{E}[Z]| \geq t) \leq \frac{Var(Z)}{t^2}, \tag{87}$$

for $t \geq 0$

9.2 Moment Generating Function

Often, we would like sharper—even exponential—bounds on the probability that a random variable Z exceeds its expectation by much. With that in mind, we need a stronger condition than finite variance, for which moment generating functions are natural candidates. (Conveniently, they also play nicely with sums, as we will see.) Recall that for a random variable Z, the moment generating function of Z is the function

$$M_Z(\lambda) := \mathbb{E}[\exp(\lambda Z)], \tag{88}$$

which may be infinite for some λ.

9.2.1 Chernoff Bounds

Chernoff bounds use of moment generating functions in an essential way to give exponential deviation bounds.

Proposition 9.3 (Chernoff Bounds). *Let Z be any random variable. Then for any $t \geq 0$,*

$$\mathbb{P}[Z \geq \mathbb{E}[Z] + t] \leq \min_{\lambda \geq 0} \mathbb{E}[e^{\lambda(Z - \mathbb{E}[Z])}]e^{-\lambda t} = \min_{\lambda \geq 0} M_{Z-\mathbb{E}[Z]}(\lambda)e^{-\lambda t} \tag{89}$$

and

$$\mathbb{P}[Z \leq \mathbb{E}[Z] - t] \leq \min_{\lambda \geq 0} \mathbb{E}[e^{\lambda(\mathbb{E}[Z] - Z)}]e^{-\lambda t} = \min_{\lambda \geq 0} M_{\mathbb{E}[Z]-Z}(\lambda)e^{-\lambda t} \tag{90}$$

9.2.2 Moment generating function examples

Now we give several examples of moment generating functions, which enable us to give a few nice deviation inequalities as a result. For all of our examples, we will have very convienent bounds of the form

$$M_Z(\lambda) = \mathbb{E}[e^{\lambda Z}] \leq \exp\left(\frac{C^2\lambda^2}{2}\right), \ \forall \lambda \in \mathbb{R}$$

for some $C \in \mathbb{R}$ (which depends on the distribution of Z); this form is very nice for applying Chernoff bounds.
We begin with the classical normal distribution, where $Z \sim \mathcal{N}(0, \sigma^2)$. Then we have

$$\mathbb{E}[\exp(\lambda Z)] = \exp\left(\frac{\lambda^2\sigma^2}{2}\right)$$

A second example is known as a Rademacher random variable, or the random sign variable. Let $S = 1$ with probability $\frac{1}{2}$ and $S = -1$ with probability $\frac{1}{2}$. Then we claim that

$$\mathbb{E}[e^{\lambda S}] \leq \exp\left(\frac{\lambda^2}{2}\right), \ \forall \lambda \in \mathbb{R} \tag{91}$$

9.3 Hoeffding's Inequality

Hoeffding's inequality is a powerful technique—perhaps the most important inequality in learning theory—for bounding the probability that sums of bounded random variables are too large or too small. We will state the inequality, and then we will prove a weakened version of it based on our moment generating function calculations earlier.

Lemma 9.4 (Hoeffding's Lemma)**.** *Let Z be a bounded random variable with $Z \in [a, b]$. Then*

$$\mathbb{E}[\exp(\lambda(Z - \mathbb{E}[Z]))] \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right), \ \forall \lambda \in \mathbb{R} \tag{92}$$

Theorem 9.5. *Let $Z_1, Z_2, ..., Z_n$ be independent bounded random variables with $Z_i \in [a, b], \forall i$, where $-\infty < a \leq b < \infty$. Then*

$$\mathbb{P}(\frac{1}{n}\sum_{i=1}^{n}(Z_i - \mathbb{E}[Z_i]) \geq t) \leq \exp\left(-\frac{2nt^2}{(b-a)^2}\right) \tag{93}$$

and

$$\mathbb{P}(\frac{1}{n}\sum_{i=1}^{n}(Z_i - \mathbb{E}[Z_i]) \leq -t) \leq \exp\left(-\frac{2nt^2}{(b-a)^2}\right) \tag{94}$$

for all $t \geq 0$

We prove Theorem 4 by using a combination of (1) Chernoff bounds and (2) a classic lemma known as Hoeffding's lemma

References

[Duc17] John Duchi. Cs229 supplemental lecture notes hoeffding's inequality, 2017.