Machine Learning Notes

DeepWalter*

*Email: deepwalter.cn@gmail.com

Contents

Part I

Basics

Chapter 1

Model Assessment and Selection

1.1 Empirical Error and Overfitting

1.2 Assessment

1.3 Performance Measurement

1.3.1 Error Rate and Accuracy

Error rate is the most direct measurement of performance. It is defined as the ratio of misclassified examples, i.e.

$$err(f; D) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(f(\boldsymbol{x}_i) \neq y_i)$$

where m is the total number of examples. The accuracy is just the complement of error rate:

$$acc(f; D) = 1 - err(f; D)$$

Generally, for a distribution $\mathcal{D}$ over a data set D with pdf (pmf) p, the error rate is defined as

$$err(f; \mathcal{D}) = \int_{\boldsymbol{x}\sim\mathcal{D}} \mathbb{I}(f(\boldsymbol{x}) \neq y)p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \tag{1.1}$$

1.3.2 Precision, Recall and F1 Score

Fact \ Prediction	Positive	Negative
True	TP	FN
False	FP	TN

Table 1.1: Confusion Matrix

Let T and F be the number of **true** and **false** examples respectively. Let P and N be the number of **positive** and **negative** examples respectively. Then we have $m = P + N = T + F$ where m is the number of total examples, and

Look at the columns of the confusion matrix.

$$\begin{cases} P &= TP + FP \\ N &= FN + TN \end{cases} \tag{1.2}$$

and

Look at the rows of the confusion matrix.

$$\begin{cases} T &= TP + FN \\ F &= FP + TN \end{cases} \tag{1.3}$$

Of those positive examples, how many are truly positive?

The **precision** is defined as

$$p = \frac{TP}{P} = \frac{TP}{TP + FP} \tag{1.4}$$

The **recall** is defined as

How many truly positive examples have been picked out?

$$r = \frac{TP}{T} = \frac{TP}{TP + FN} \tag{1.5}$$

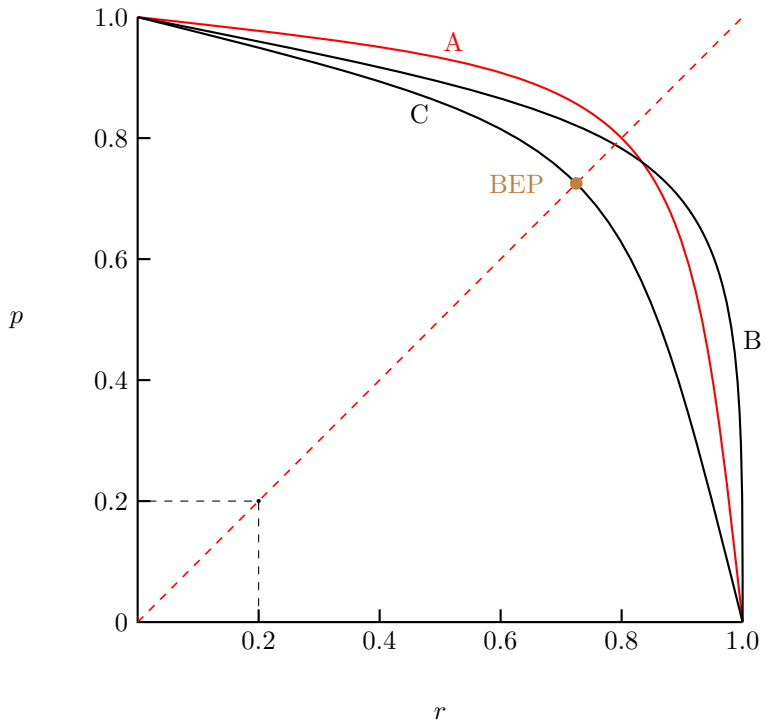

Figure 1.1: Precision-Recall Curve

In many algorithms, for example the logistic regression, we first compute a value from the given feature vector of an example, then we compare it with a threshold value to determine whether it is positive or negative. In order to get high recall, we can mark all examples as positive (that is, set the threshold value to a very small value), which makes $FN = 0$; but this will make the precision low since all false examples are classified as positive too. Similarly, in order to achieve high precision, we can set the threshold value large (that is, we only classify an example as positive if we are confident enough), but this will leave many examples as negative, hence FN is large, which causes low recall. In general, precision and recall can not both increase at the same time. See figure 1.1. If one learner's P-R curve is enclosed by anothers', then the latter has a better performance. In other situations, can use the **B**reak **E**vent **P**oint to compare two learners' performance. Here, the BEP is just the point where

For example, B performs better than C in figure 1.1.

precision equals recall. See figure 1.1 for an example.

The F1 score is defined as

$$F1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} = \frac{2pr}{p + r} \tag{1.6}$$

1.3.3 ROC and AUC

True/False Positive Rate is defined as

$$\begin{cases} TPR &= \frac{TP}{T} = \frac{TP}{TP + FN} \\ FPR &= \frac{FP}{F} = \frac{FP}{FP + TN} \end{cases} \tag{1.7}$$

The **R**eceiver **O**perating **C**haracteristic curve uses true positive rate as vertical axis and false positive rate as horizontal axis. Below is a ROC curve and the gray area is the corresponding **A**era **U**nder **C**urve.

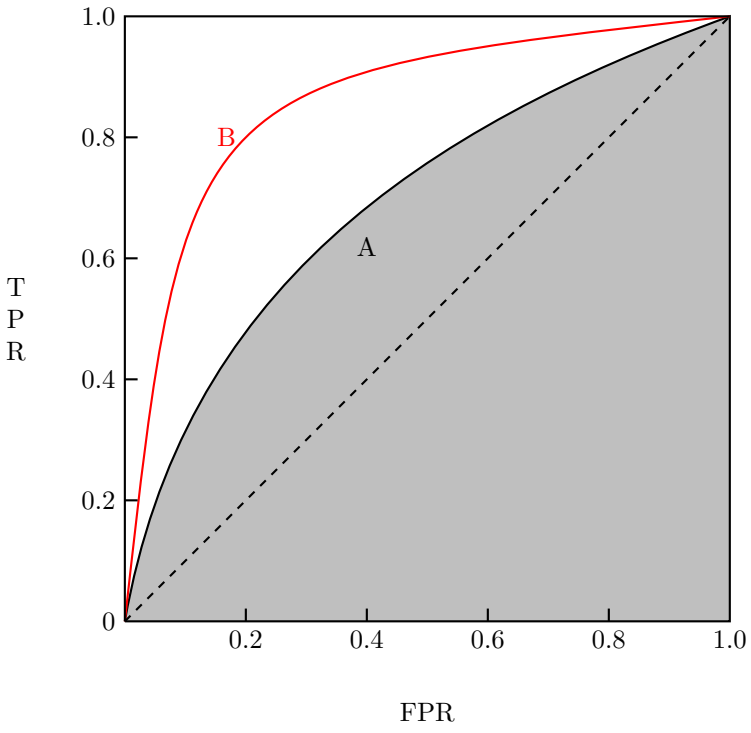

Figure 1.2: ROC Curve

When $TPR = FPR$, we have

$$\frac{TP}{T} = \frac{FP}{F}$$

Notice that $T = TP + FN$ and $F = FP + TN$, we have

$$\frac{FN}{T} = \frac{TN}{F}$$

That is, for any example, we predict it to be positive with a given probability. So in figure 1.2, the dashed diagonal corresponds to the random guess model (with a fixed probability). When $(FPR, TPR) = (1, 0)$, we have

$$\begin{cases} TP &= 0 \\ TN &= 0 \end{cases}$$

that is, $acc = 0$. When $(FPR, TPR) = (0, 1)$, we have

$$\begin{cases} FP &= 0 \\ FN &= 0 \end{cases}$$

that is, $acc = 1$. When $(FPR, TPR) = (0, 0)$, we have

$$\begin{cases} FP &= 0 \\ TP &= 0 \end{cases}$$

that is, the predictor marks all examples as negative. When $(FPR, TPR) = (1, 1)$, we have

$$\begin{cases} FN &= 0 \\ TN &= 0 \end{cases}$$

that is, the predictor marks all example as positive.

Of course, if a learner's ROC is fully enclosed by anothers', then the latter has a better performance. For example, in figure 1.2, the learner with ROC B performs better than the one with ROC A. If two learners' ROCs cross with each other, then we may compare their AUCs and again we prefer the one with larger AUC.

1.4 Bias and Variance

Chapter 2

Linear Model

In a linear model, the learner takes the form of a (affine) linear function

$$y = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = \langle \hat{\boldsymbol{w}}, \hat{\boldsymbol{x}} \rangle$$

where $\hat{\boldsymbol{w}} = (\boldsymbol{w}, b),\ \hat{\boldsymbol{x}} = (\boldsymbol{x}, 1)$.

2.1 Linear Regression

2.2 Logistic Regression

2.3 Linear Discriminant Analysis

The idea of linear discriminant analysis is to find a line with direction $\boldsymbol{w}$ s.t. when projected to this line, examples with the same label will stay close while examples with different labels will be far away. When predicting, we first project the example onto the line, then assign it the label of the group of examples which is closest to it.

$n_0+n_1 = m$ and $\boldsymbol{\mu}_i$ is the center of examples with label i.

Let $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ be the data set, $\mathcal{Y} = \{0, 1\}$ the label space. Let $X_i \in \mathbb{R}^{n_i \times n}$ be the matrix of all examples with label i and $\boldsymbol{\mu}_i \in \mathbb{R}^{1 \times n}$ the mean vector of all such examples. Let

$$\Sigma_i = (X_i - \boldsymbol{\mu}_i)^\mathsf{T}(X_i - \boldsymbol{\mu}_i)$$

be the covariance matrix of all examples with label i. After projecting onto the line $\boldsymbol{w}$, X_i becomes $X_i\boldsymbol{w}^\mathsf{T}$, and $\boldsymbol{\mu}_i$ becomes $\boldsymbol{\mu}_i\boldsymbol{w}^\mathsf{T}$. Hence the covariance matrix becomes $\boldsymbol{w}\Sigma_i\boldsymbol{w}^\mathsf{T}$, which is a real number. To make examples with the same label stay close and examples with different labels stay away, we want small variances $\boldsymbol{w}\Sigma_i\boldsymbol{w}^\mathsf{T}$ and a large distance $||\boldsymbol{\mu}_0\boldsymbol{w}^\mathsf{T} - \boldsymbol{\mu}_1\boldsymbol{w}^\mathsf{T}||_2^2$. That is, we want to maximize

Notice that J is independent of the module of $\boldsymbol{w}$.

$$J = \frac{||\boldsymbol{\mu}_0\boldsymbol{w}^\mathsf{T} - \boldsymbol{\mu}_1\boldsymbol{w}^\mathsf{T}||_2^2}{\boldsymbol{w}\Sigma_0\boldsymbol{w}^\mathsf{T} + \boldsymbol{w}\Sigma_1\boldsymbol{w}^\mathsf{T}} \tag{2.1}$$

If we define the **within-class scatter matrix** as

$$S_w = \sum_{\boldsymbol{x} \in X_0} (\boldsymbol{x} - \boldsymbol{\mu}_0)^\mathsf{T}(\boldsymbol{x} - \boldsymbol{\mu}_0) + \sum_{\boldsymbol{x} \in X_1} (\boldsymbol{x} - \boldsymbol{\mu}_1)^\mathsf{T}(\boldsymbol{x} - \boldsymbol{\mu}_1) \tag{2.2}$$

and **between-class scatter matrix** as

$$S_b = (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^\mathsf{T}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) \tag{2.3}$$

then equation (2.1) becomes

$$J = \frac{\boldsymbol{w} S_b \boldsymbol{w}^\mathsf{T}}{\boldsymbol{w} S_w \boldsymbol{w}^\mathsf{T}} \tag{2.4}$$

which is the **generalized Rayleigh quotient** of S_b and S_w.

To maximize the generalized Rayleigh quotient, we solve the equivalent problem

$$\underset{\boldsymbol{w}}{\operatorname{argmax}}\, \boldsymbol{w} S_b \boldsymbol{w}^\mathsf{T} \quad \text{s.t. } \boldsymbol{w} S_w \boldsymbol{w}^\mathsf{T} = 1$$

Let $f(\boldsymbol{w}, \lambda) = \boldsymbol{w} S_b \boldsymbol{w}^\mathsf{T} + \lambda(1 - \boldsymbol{w} S_w \boldsymbol{w}^\mathsf{T})$ be the Lagrangian, then $\frac{\partial f}{\partial \boldsymbol{w}} = 0$ implies

$$\boldsymbol{w} S_b = \lambda \boldsymbol{w} S_w \tag{2.5}$$

Since $\boldsymbol{w} S_b = \alpha(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$ for some α, we have

$$\boldsymbol{w} = \frac{\alpha}{\lambda}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) S_w^{-1} \tag{2.6}$$

Notice that we have the restriction $\boldsymbol{w} S_w \boldsymbol{w}^\mathsf{T} = 1$, which gives

$$\frac{\lambda}{\alpha} = \sqrt{(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) S_w^{-1} (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^\mathsf{T}} \tag{2.7}$$

Combine equation (2.6) and (2.7), we have

$$\boldsymbol{w} = \frac{(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) S_w^{-1}}{\sqrt{(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) S_w^{-1} (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^\mathsf{T}}} \tag{2.8}$$

Chapter 3

Decision Tree

3.1 Classification Tree

In this section, we focus on classification trees. First we assume that the features are discrete. We want to find a tree that can determine to which class an example belongs given any combination of features.

3.1.1 General Algorithm

The general algorithm for generating a classification tree is very simple. It recursively splits the node into subtrees according to a given rule until some stop conditions occur.

Algorithm 1 Decision Tree

Input: training set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$; attribute set $A = \{a_1, a_2, \ldots, a_d\}$.
Output: a decision tree.

procedure DT(D, A)
  Generate a node.
  if $y_i = c, \forall\ i = 1, \ldots, m$ **then** ▷ All samples have the same label
    mark the node as a leaf with label c.
    return
  end if
  if $A = \emptyset$ **OR** samples of D take the same value on each attribute from A **then** ▷ attributes from A cannot distinguish elements of D
    mark the node as a leaf with the majority label of D.
    return
  end if
  Choose the *best* attribute a_* from A.
  if NOT worthSplitting(D, a_*) **then** ▷ Not worth splitting.
    mark the node as a leaf with the majority label of D.
    return
  else
    for all possible value a_*^v of a_* **do**
      generate a branch from the node.
      let D_v be all samples that take value a_*^v on attribute a_*.
      if $D_v = \emptyset$ **then** ▷ no sample of D takes value a_*^v on a_*
        mark the branch as a leaf with the majority label of D.
        return
      else
        set the branch to DT($D_v, A - \{a_*\}$).
      end if
    end for
  end if
end procedure

3.1.2 Measurement

The line 1.11 and 1.12 are the key points of the algorithm. Namely, we need some reasonable measurement to choose the best feature and decide whether it is necessary to do the split. As ususal, we find some loss function L which is defined on a single node. For any feature a, let $\{D^v\}_v$ be a splitting of D according to it. Then L can be extended to $\{D^v\}_v$, the tree after splitting, in a natural way:

$$L(D, a) = \sum_v \frac{|D^v|}{|D|} L(D^v) \tag{3.1}$$

Our goal is to find a feature that minimize the above loss. That is, the best feature a_* is given by:

$$a_* = \operatorname*{argmin}_a L(D, a)$$

Notice that before splitting, the loss on the single node is $L(D)$. Hence we can define the gain of a splitting* as

$$\text{Gain}(D, a) = L(D) - L(D, a) \tag{3.2}$$

If the gain is too small, for example, less than a preset threshold $\varepsilon \geqslant 0$, then we decide that it is not worth splitting. And also notice that

$$\operatorname*{argmax}_a \text{Gain}(D, a) = \operatorname*{argmin}_a L(D, a)$$

We can combine the above two metrics into a single one.

Information Gain

Definition 3.1 (Information Entropy) The **information entropy** of a discrete random variable X is defined as:

$$H(X) = \mathbb{E}[-\log_2(\mathbb{P}(X))] \tag{3.3}$$

If $X \in \{x_1, \ldots, x_n\}$ and $\mathbb{P}(x_i) = p_i$, then the above entropy can be written as:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Remark 3.2 It is easy to see that:

1. $H(X) \geqslant 0$
2. H takes the minimal value when $p_1 = 1$.

For any training dataset $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$, let $\mathcal{Y}$ be the label space and $Y \in \mathcal{Y}$ the random variable determined by D. We can define the

*i.e. the amount of loss reduced by the splitting.

entropy of D as the entropy of Y:

$$\text{Entropy}(D) = -\sum_{i=1}^{|\mathcal{Y}|} p_i \log_2 p_i \tag{3.4}$$

where p_i is the probability of the i-th label occuring in D.

Similarly, we have the conditional entropy of two random variables X, Y:

Definition 3.3 (Conditional Entropy) The entropy of X given Y is defined as

$$\begin{aligned} H(X|Y) &= \sum_y p(y) H(X|Y=y) \\ &= -\sum_y p(y) \sum_x p(X=x|Y=y) \log_2 p(X=x|Y=y) \\ &= -\sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(y)} \end{aligned} \tag{3.5}$$

Notice that the conditional entropy has the same form as the loss in (3.1). That is, for any possible value a^v of a, let $D^v = \{\boldsymbol{x}_i | y_i = a^v\}$, i.e. D^v is the collection of those example with label a^v. Then the conditional entropy of D w.r.t. a can be written as

$$\text{Entropy}(D, a) = \sum_v \frac{|D^v|}{|D|} \text{Entropy}(D^v) \tag{3.6}$$

Remember we want to reduce the entropy of the tree.

Hence the **information gain** of the splitting according to a is

$$\begin{aligned} \text{Gain}(D, a) &= \text{Entropy}(D) - \text{Entropy}(D, a) \\ &= \text{Entropy}(D) - \sum_v \frac{|D^v|}{|D|} \text{Entropy}(D^v) \end{aligned} \tag{3.7}$$

Of course, we want to choose $a_* = \max_a \text{Gain}(D, a)$ as the best feature in line 1.11. And it is worth splitting the tree only if $\text{Gain}(D, a_*) \geqslant \varepsilon$ for some preset threshold $\varepsilon \geqslant 0$. This kind of measurement is adopted by the ID3 algorithm.

Information Gain Ratio

When all samples in D have the same label, then it's easy to see that

$$\text{Entropy}(D) = -1 \cdot \log_2 1 = 0$$

Hence if some feature takes different values on each example, then the conditional entropy is 0, and the information gain is maximal. That is, information

gain favours those features with more possible values. To reduce this bias, we introduce the information gain ratio. Let

$$\mathrm{IV}(a) = -\sum_{v} \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

be the **intrinsic value** of a, then the information gain ratio w.r.t. a is defined as:

$$\mathrm{GainRatio}(D, a) = \frac{\mathrm{Gain}(D, a)}{\mathrm{IV}(a)} \tag{3.8}$$

Notice that when a only takes one value, then $\mathrm{IV}(a) = 0$. Hence the information gain ratio favours those features with less values. In order to balance between gain and gain ratio, we can first choose those features with information gain above average, then choose the one with highest gain ratio from them, as in C4.5 algorithm. As ususal, we can compare the gain ratio with the threshold ε to determine whether to split the tree or not.

Gini Index

The Gini index of the training set D is

$$\begin{aligned} \mathrm{Gini}(D) &= \sum_{i} \sum_{i'} p_i p_{i'} \\ &= 1 - \sum_{i=1}^{|\mathcal{Y}|} p_i^2 \end{aligned} \tag{3.9}$$

Namely, it is the probability that you get two different labels when you randomly pick two examples from D. That is, it reflects the purity of the node. As usual, the Gini index of D w.r.t. a can be defined as:

$$\mathrm{Gini}(D, a) = \sum_{v} \frac{|D^v|}{|D|} \mathrm{Gini}(D^v) \tag{3.10}$$

which measures the purity of the tree after splitting. Then we can use Gini index as the loss function in the measurement.

3.1.3 Pruning

3.2 Classification and Regression Tree

Chapter 4

Support Vector Machine

Let $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ be the training set where $y_i \in \{-1, +1\}$. The SVM is an algorithm that tries to find a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ which separate the positive examples from the negative ones. The corresponding predictor is $f(\boldsymbol{x}) = \text{sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)$.

4.1 Hard SVM

In the case of hard SVM, we assume that the training set is linear separable. Before going any further about the ideas of SVM, let's first introduce the concept of the margin of a hyperplane:

Definition 4.1 (Margin) The margin of a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ w.r.t. a training set D is defined as

$$\min_i \frac{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)}{||\boldsymbol{w}||}$$

Obviously, positive margin means the hyperplane classifies D correctly while negative margin means there is at least one point on the wrong side. When D is correctly classified by the hyperplane, the margin is just the geometric distance between the training set and the hyperplane.

The core idea of (hard) SVM is to find a hyperplane which separates the training set *with the largest margin*. That is, we hope to solve the following:

Obviously, for any positive λ, $(\boldsymbol{w}, b)$ and $(\lambda\boldsymbol{w}, \lambda b)$ specify the same hyperplane with the same "left" and "right" sides.

$$\underset{\boldsymbol{w},b}{\text{argmax}} \min_i \frac{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)}{||\boldsymbol{w}||} \tag{4.1}$$

Let

$$\gamma = \min_i \frac{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)}{||\boldsymbol{w}||}$$

then the problem (4.1) becomes:

$$\underset{\boldsymbol{w},b}{\text{argmax}}\, \gamma\ , \quad \text{s.t. } \frac{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)}{||\boldsymbol{w}||} \geqslant \gamma \quad \forall\ i \tag{4.2}$$

Let $\gamma \leftarrow ||\boldsymbol{w}||\gamma$, the above problem becomes:

$$\underset{\boldsymbol{w},b}{\text{argmax}} \frac{\gamma}{||\boldsymbol{w}||}\ , \quad \text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant \gamma \quad \forall\ i \tag{4.3}$$

Using the linear separable assumption, we know that $\gamma > 0$. Let $\boldsymbol{w} \leftarrow \frac{\boldsymbol{w}}{\gamma}$ and $b \leftarrow \frac{b}{\gamma}$, then the above becomes:

$$\underset{\boldsymbol{w},b}{\text{argmax}} \frac{1}{||\boldsymbol{w}||}\ , \quad \text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant 1 \quad \forall\ i \tag{4.4}$$

which is obviously equivalent to:

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{||\boldsymbol{w}||^2}{2}, \quad \text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant 1 \quad \forall\, i \tag{4.5}$$

Theorem 4.2 Assume the training set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ is linear separable, then there exists a unique hyperplane separating the dataset with the largest margin, which is given by the solution of the above problem (4.5).

Proof The existence part is immediately from the separability assumption.
TODO □

Remark 4.3 From the problem (4.4), it is clear that if $(\boldsymbol{w}, b)$ is the separating hyperplane, then $\exists\ i$ s.t. $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) = 1$ and the (largest) margin is $\frac{1}{||\boldsymbol{w}||}$. Since the hyperplane is totally determined* by those $\boldsymbol{x}_i$ s, they are called the **supporting vectors** of the hyperplane.

4.2 Soft SVM

The hard SVM works well when the data set is linear separable, but it behaves poorly on the sets which are not linear separable since all the restrictions cannot be satisfied at the same time. In order to adapt to the non-separable case, we can allow some points to violate the restriction, and penalize them in the optimization target. That is, we can consider the following problem:

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{||\boldsymbol{w}||^2}{2} + C \sum_{i=1}^{m} \max\left(0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)\right) \tag{4.6}$$

Here, if $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) < 1$, we add the penalization $1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)$ to the optimization target; otherwise, no penalization is added. The constant $C > 0$ is the weight that determines how much the penalization matters (or how much you can violate the restrictions). For example, if $C = 0$, then the penalization dosen't matter at all and you can violate all the restrictions; if $C = +\infty$, then the penalization matters most and you cannot violate any single restriction.

If we introduce the slack variabels $\xi_i = \max\left(0, 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)\right)$, then the problem (4.6) becomes:

$$\underset{\boldsymbol{w},b,\boldsymbol{\xi}}{\operatorname{argmin}} \frac{||\boldsymbol{w}||^2}{2} + C \sum_{i=1}^{m} \xi_i \quad \text{s.t.} \begin{cases} y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant 1 - \xi_i \\ \xi_i \geqslant 0 \end{cases} \quad \forall\, i \tag{4.7}$$

this is what we called the soft SVM.

*that is, you can remove other points without affecting the separating hyperplane.

4.3 Duality

Let $h_i(\boldsymbol{w}, b) = 1 - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)$, then the hard SVM (4.5) becomes

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 \quad \text{s.t. } h_i(\boldsymbol{w}, b) \leqslant 0 \quad \forall\, i \tag{4.8}$$

Its Lagrangian is

$$L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||^2 + \sum_i \alpha_i h_i(\boldsymbol{w}, b) \tag{4.9}$$

The orignal problem (4.8) above is equivalent to

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \max_{\boldsymbol{\alpha}:\alpha_i \geqslant 0} L(\boldsymbol{w}, b; \boldsymbol{\alpha})$$

Let $\theta_D(\boldsymbol{\alpha}) = \min\limits_{\boldsymbol{w},b} L(\boldsymbol{w}, b; \boldsymbol{\alpha})$, then $\nabla_{\boldsymbol{w},b} L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = 0$ gives

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \tag{4.10a}$$

$$\sum_i y_i \alpha_i = 0 \tag{4.10b}$$

Substitute equation (4.10a) into the Lagrangian (4.9), we have

$$\theta_D(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{4.11}$$

Hence the dual problem $\max\limits_{\boldsymbol{\alpha}:\alpha_i \geqslant 0} \operatorname{argmin}\limits_{\boldsymbol{w},b} L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \max\limits_{\boldsymbol{\alpha}:\alpha_i \geqslant 0} \theta_D(\boldsymbol{\alpha})$ becomes

$$\max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \quad \text{s.t.} \begin{cases} \sum\limits_i y_i \alpha_i = 0 \\ \alpha_i \geqslant 0 \quad \forall\, i \end{cases}$$

or equivalently

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_i \alpha_i \quad \text{s.t.} \begin{cases} \sum\limits_i y_i \alpha_i = 0 \\ \alpha_i \geqslant 0 \quad \forall\, i \end{cases} \tag{4.12}$$

Notice that $\frac{1}{2}||\boldsymbol{w}||^2$ is convex and h_i are affine linear. And the linear separable assumption guarantees that there is some $(\boldsymbol{w}, b)$ s.t. $h_i(\boldsymbol{w}, b) < 0\ \forall\ i$, i.e. the inequality restrictions are strict. Hence there is a solution $(\boldsymbol{w}^*, b^*)$ for the hard SVM (4.5), and a solution $\boldsymbol{\alpha}^*$ for the dual problem (4.12). Moreover, they

satisfy the KKT condition:

$$\begin{cases} \boldsymbol{w}^* - \sum_i \alpha_i^* y_i \boldsymbol{x}_i = 0 \\ \sum_i y_i \alpha_i^* = 0 \\ \alpha_i^* \geqslant 0 \\ h_i(\boldsymbol{w}^*, b^*) \leqslant 0 \\ \alpha_i^* h_i(\boldsymbol{w}^*, b^*) = 0 \end{cases} \tag{4.13}$$

Notice that not all α_i^* could be 0 (otherwise $\boldsymbol{w}^* = 0$, which is not a solution of the hard SVM). Let $\alpha_{i_0}^* > 0$. Then $\alpha_{i_0}^* h_{i_0}(\boldsymbol{w}^*, b^*) = 0$ implies

$$b^* = y_{i_0} - \sum_i \alpha_i^* y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_{i_0} \rangle$$

In summary, we have

$$\begin{cases} \boldsymbol{w}^* = \sum_i \alpha_i^* y_i \boldsymbol{x}_i \\ b^* = y_{i_0} - \sum_i \alpha_i^* y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_{i_0} \rangle \end{cases} \tag{4.14}$$

Remark 4.4 For those indices i s.t. $\alpha_i^* > 0$, the corresponding $\boldsymbol{x}_i$ are the supporting vectors since $h_i(\boldsymbol{w}^*, b^*) = 0$. And it is easy to see that those $\boldsymbol{x}_i$ with $\alpha_i^* = 0$ do not affect the hyperplane.

Similarly, let $h_i(\boldsymbol{w}, b, \xi_i) = 1 - \xi_i - y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)$. Then the Lagrangian of the soft SVM (4.7) is

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^m \xi_i + \sum_{i=1}^m \alpha_i h_i(\boldsymbol{w}, b, \xi_i) + \sum_{i=1}^m \beta_i(-\xi_i)$$

The soft SVM (4.7) is equivalent to

$$\underset{\boldsymbol{w}, b, \boldsymbol{\xi}}{\operatorname{argmin}} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i, \beta_i \geqslant 0} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta})$$

Let $\theta_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta})$, then $\nabla_{\boldsymbol{w}, b, \boldsymbol{\xi}} L = 0$ implies

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \tag{4.15a}$$

$$\sum_i y_i \alpha_i = 0 \tag{4.15b}$$

$$\alpha_i + \beta_i = C \tag{4.15c}$$

Hence we have:

$$\theta_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{4.16}$$

which is exactly the same as before. Hence the dual problem of soft SVM (4.7) is

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_i \alpha_i \quad \text{s.t.} \begin{cases} \sum_i \alpha_i y_i = 0 \\ 0 \leqslant \alpha_i \leqslant C \quad \forall\ i \end{cases} \tag{4.17}$$

Let $\boldsymbol{w}^*, b^*, \boldsymbol{\xi}^*$ be the solution to the original soft SVM (4.6) and $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ the solution to the dual problem (4.17), then they satifies the KKT conditions:

$$\begin{cases} \boldsymbol{w}^* - \sum_i \alpha_i^* y_i \boldsymbol{x}_i = 0 \\ \sum_i y_i \alpha_i^* = 0 \\ a_i^* + \beta_i^* = C \\ \alpha_i^* \geqslant 0 \\ \beta_i^* \geqslant 0 \\ \xi_i^* \geqslant 0 \\ h_i(\boldsymbol{w}^*, b^*, \xi_i^*) \leqslant 0 \\ \alpha_i^* h_i(\boldsymbol{w}^*, b^*, \xi_i^*) = 0 \\ \beta_i^* \xi_i^* = 0 \end{cases} \tag{4.18}$$

If $\alpha_i^* > 0$, then $h_i(\boldsymbol{w}^*, b^*, \xi_i^*) = 0$, i.e. $y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) = 1 - \xi_i^*$. Those $\boldsymbol{x}_i$ are called supporting vectors. Moreover, if $\alpha_i^* < C$, then $\beta_i^* > 0$, hence $\xi_i^* = 0$ and $y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) = 1$, thus those supporting vectors locate on the decision boundaries; if $\alpha_i^* = C$, then $\beta_i^* = 0$, thus those supporting vectors locate between the decision boundaries if $\xi_i^* \leqslant 1$ and they are mis-classified if $\xi_i^* > 1$. If $\alpha_i^* = 0$, then $\beta_i^* = C$ and $\xi_i^* = 0$, thus $1 \leqslant y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*)$, that is they are correctly classified*.

If there is some α_{i_0} such that $0 < \alpha_{i_0}^* < C$, then the solution of the soft SVM (4.7) can be written as:

$$\begin{cases} \boldsymbol{w}^* = \sum_i y_i \alpha_i^* \boldsymbol{x}_i \\ b^* = y_{i_0} - \sum_i y_i \alpha_i^* \langle \boldsymbol{x}_i, \boldsymbol{x}_{i_0} \rangle \end{cases} \tag{4.19}$$

*but they don't affect the separating hyperplane.

4.4 Kernel Method

The general idea of kernel method is that after mapping the original feature space into an Hilbert space via a map ϕ, if we need to compute the inner product $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ which usually is difficult, we hope that there is a kernel function κ s.t. $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and the latter is easier to compute. Following this idea, the dual problem (4.12) in Hilbert space

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle - \sum_i \alpha_i \quad \text{s.t.} \begin{cases} \sum_i y_i \alpha_i = 0 \\ \alpha_i \geqslant 0 \quad \forall\ i \end{cases} \tag{4.20}$$

becomes

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_i \alpha_i \quad \text{s.t.} \begin{cases} \sum_i y_i \alpha_i = 0 \\ \alpha_i \geqslant 0 \quad \forall\ i \end{cases} \tag{4.21}$$

and its solution satifies

$$\begin{cases} \boldsymbol{w}^* = \sum_i \alpha_i^* y_i \phi(\boldsymbol{x}_i) \\ b^* = y_{i_0} - \sum_i \alpha_i^* y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_{i_0}) \end{cases} \tag{4.22}$$

Hence the predictor is

$$y = \operatorname{sign}\left(\sum_i \alpha_i^* y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + y_{i_0} - \sum_i \alpha_i^* y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_{i_0}) \right)$$

Theorem 4.5 (Kernel function) Let $\mathcal{X}$ be a feature space, $\kappa(\cdot, \cdot)$ is some symmetric function on $\mathcal{X} \times \mathcal{X}$, then κ is a kernel function if and only if for any data set $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$, the matrix

$$\begin{bmatrix} \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_i, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_i, \boldsymbol{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\boldsymbol{x}_m, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_m, \boldsymbol{x}_j) & \cdots & \kappa(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{bmatrix}$$

is semi-positive.

Proposition 4.6 Let κ_1 and κ_2 be any kernel functions. Then

1. For any $\gamma_1, \gamma_2 > 0$, $\gamma_1 \kappa_1 + \gamma_2 \kappa_2$ is a kernel function.
2. $\kappa_1 \cdot \kappa_2$ is a kernel function.
3. For any function g, $\kappa(\boldsymbol{x}, \boldsymbol{y}) := g(\boldsymbol{x}) \kappa_1(\boldsymbol{x}, \boldsymbol{y}) g(\boldsymbol{y})$ is a kernel function.

Some useful kernels are:

1. Linear kernel: $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$.
2. Polynomial kernel: $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\beta \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + \theta)^d$ where $\beta, \theta > 0\,, d \geqslant 1$.
3. Guassian kernel: $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{2\sigma^2})$ where $\sigma > 0$.
4. Laplacian kernel: $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||}{\sigma})$ where $\sigma > 0$.
5. Sigmoid kernel: $\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \tanh(\beta \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + \theta)$ where $\beta > 0,\ \theta < 0$.

4.5 Support Vector Regression

Although SVM is used for binary classification, we can apply its idea to linear regression. Namely, we can tolerate those examples that are close enough to the predictor and only penalize those that are far away. More specifically, we want to find the predictor $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ by minimizing the loss

L is just a regularized loss.

$$L(\boldsymbol{w}, b) := \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{m} l_\varepsilon(\hat{y}_i - y_i) \tag{4.23}$$

where l_ε is the ε-insensitive loss:

$$l_\varepsilon(z) = \begin{cases} 0, & \text{if } |z| < \varepsilon \\ |z| - \varepsilon, & \text{otherwise} \end{cases} \tag{4.24}$$

Notice that $l_\varepsilon(z) = \max(|z| - \varepsilon, 0) = \max(z - \varepsilon, -z - \varepsilon, 0)$, if we introduce the slack variabels $\xi_i = l_\varepsilon(\hat{y}_i - y_i)$, the problem becomes

$$\underset{\boldsymbol{w}, b, \boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{m} \xi_i \quad \text{s.t.} \quad \begin{cases} \xi_i \geqslant 0 \\ \xi_i \geqslant \hat{y}_i - y_i - \varepsilon \quad \forall\ i \\ \xi_i \geqslant -\hat{y}_i + y_i - \varepsilon \end{cases} \tag{4.25}$$

Its Lagrangian is:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{m} \xi_i + \sum_{i=1}^{m} \alpha_i(-\xi_i) + \sum_{i=1}^{m} \beta_i(\hat{y}_i - y_i - \xi_i - \varepsilon) \\ + \sum_{i=1}^{m} \gamma_i(-\hat{y}_i + y_i - \xi_i - \varepsilon) \end{aligned} \tag{4.26}$$

Its dual problem is

$$\max_{\substack{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \\ \alpha_i, \beta_i, \gamma_i \geqslant 0}} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$$

For the inner minimization problem,

$$\nabla_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = 0$$

gives

$$\boldsymbol{w} = \sum_i (\gamma_i - \beta_i)\boldsymbol{x}_i \tag{4.27a}$$

$$\sum_i (\beta_i - \gamma_i) = 0 \tag{4.27b}$$

$$C = \alpha_i + \beta_i + \gamma_i \tag{4.27c}$$

Substitute those equations into the dual problem, we get

$$\max_{\substack{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma} \\ \alpha_i,\beta_i,\gamma_i \geqslant 0}} -\frac{1}{2}\sum_{i,j}(\beta_i - \gamma_i)(\beta_j - \gamma_j)\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_i \big(y_i(\gamma_i - \beta_i) - \varepsilon(\gamma_i + \beta_i)\big)$$

which is equivalent to

$$\min_{\substack{\boldsymbol{\beta},\boldsymbol{\gamma} \\ \beta_i,\gamma_i \geqslant 0}} \frac{1}{2}\sum_{i,j}(\beta_i - \gamma_i)(\beta_j - \gamma_j)\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_i \big(\varepsilon(\beta_i + \gamma_i) + y_i(\beta_i - \gamma_i)\big) \tag{4.28}$$

subjecting to the conditions

$$\begin{cases} \sum_i (\beta_i - \gamma_i) = 0 \\ \beta_i + \gamma_i \leqslant C \\ \beta_i \geqslant 0 \\ \gamma_i \geqslant 0 \end{cases}$$

Moreover, the KKT condition is

$$\begin{cases} \boldsymbol{w} = \sum_i (\gamma_i - \beta_i)\boldsymbol{x}_i \\ \sum_i (\beta_i - \gamma_i) = 0 \\ C = \alpha_i + \beta_i + \gamma_i \\ \alpha_i, \beta_i, \gamma_i \geqslant 0 \\ \xi_i \geqslant 0 \\ \hat{y}_i - y_i - \xi_i - \varepsilon \leqslant 0 \\ -\hat{y}_i + y_i - \xi_i - \varepsilon \leqslant 0 \\ \alpha_i \xi_i = 0 \\ \beta_i(\hat{y}_i - y_i - \xi_i - \varepsilon) = 0 \\ \gamma_i(-\hat{y}_i + y_i - \xi_i - \varepsilon) = 0 \end{cases} \tag{4.29}$$

If $\beta_i > 0$, then $\hat{y}_i - y_i - \xi_i - \varepsilon = 0$, from this we can get

$$b = y_i + \xi_i + \varepsilon - \sum_{j=1}^{m}(\gamma_j - \beta_j)\langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle \tag{4.30}$$

Similarly, if $\gamma_i > 0$, then $-\hat{y}_i + y_i - \xi_i - \varepsilon = 0$, which implies

$$b = y_i - \xi_i - \varepsilon - \sum_{i=1}^{m} (\gamma_j - \beta_j) \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle \tag{4.31}$$

To get the bias b of the predictor, we can first calculate all bs by the above two equations, then use their average as the bias.

Chapter 5

Neuron Networks

5.1 Fully Connected Neuron Networks

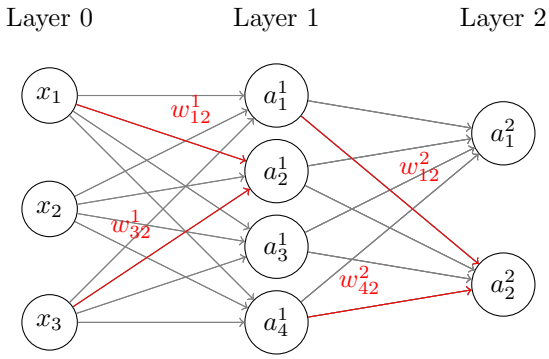

Figure 5.1: A 2 Layer Neuron Network

5.1.1 Notations

L output layer.

n_l number of neurons in layer l. In particular, $n_0 = n$.

w_{ij}^l weight from the i-th neuron in the layer $l-1$ to the j-th neuron in the layer l.

b_j^l bias of the j-th neuron in layer l.

a_j^l activation of the j-th neuron in layer l.

z_j^l raw output of the j-th neuron in layer l.

σ_j^l activation function of the j-th neuron in layer l.

$\boldsymbol{W}^l$ weight matrix connecting layer $l-1$ to layer l, i.e. (w_{ij}^l), of dimension $(n_{l-1},\ n_l)$.

$\boldsymbol{b}^l$ bias vector of layer l, i.e. (b_j^l), of dimension $(1, n_l)$.

$\boldsymbol{z}^l$ raw output vector of layer l, of dimension $(1, n_l)$.

$\boldsymbol{a}^l$ activation vector of layer l, of dimension $(1, n_l)$.

$\boldsymbol{\sigma}^l$ vector of activation functions of layer l, of dimension $(1, n_l)$.

Some basic equations:

$$z_j^0 = a_j^0 = x_j \tag{5.1}$$
$$z_j^l = \sum_k a_k^{l-1} w_{kj}^l + b_j^l \quad \forall\, l \geq 1 \tag{5.2}$$
$$a_j^l = \sigma_j^l(z_j^l) \tag{5.3}$$

The corresponding matrix forms are:

$$\boldsymbol{z}^0 = \boldsymbol{a}^0 = \boldsymbol{x} = (x_1, x_2, \ldots, x_n) \tag{5.4}$$
$$\boldsymbol{z}^l = \boldsymbol{a}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l \tag{5.5}$$
$$\boldsymbol{a}^l = \boldsymbol{\sigma}^l(\boldsymbol{z}^l) \tag{5.6}$$

For a single input example $\boldsymbol{x}$, the cost function C should only directly depend on the output layer L, for example C is the square loss function:

$$C = \frac{1}{2}||\boldsymbol{a}^L - \boldsymbol{y}||_2^2 \tag{5.7}$$

For a collection of examples, the cost function is the average cost on those examples:

$$C = \frac{1}{m}\sum_{i=1}^{m} C(\boldsymbol{x}^i) \tag{5.8}$$

5.1.2 Backpropagation

First let's consider the case with a single input example. Let δ_j^l be the error in the j-th neuron in the l-th layer, i.e.

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{5.9}$$

For the output layer L, by definition we have:

$$\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L}\frac{\partial \sigma_j^L(z_j^L)}{\partial z_j^L} \\
&= \frac{\partial C}{\partial a_j^L} \cdot (\sigma_j^L)'(z_j^L)
\end{aligned}$$

Let $\boldsymbol{\delta}^l = (\delta_j^l)_{1\times n_l}$, then the above equation can be written as:

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}^L} C \odot (\boldsymbol{\sigma}^L)'(\boldsymbol{z}^L) \tag{5.10}$$

For example, when C is the square loss (5.7), $\nabla_{\boldsymbol{a}^L} C = \boldsymbol{a}^L - \boldsymbol{y}$.

We can write $\boldsymbol{\delta}^l$ in terms of $\boldsymbol{\delta}^{l+1}$ as following:

$$\begin{aligned}
\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\
&= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\
&= \sum_k \delta_k^{l+1} \sum_r \frac{\partial a_r^l}{\partial z_j^l} w_{rk}^{l+1} \\
&= \sum_k \delta_k^{l+1} (\sigma_j^l)'(z_j^l) w_{jk}^{l+1} \\
&= \left(\boldsymbol{\delta}^{l+1} (\boldsymbol{W}^{l+1})^{\mathsf{T}}\right)_j (\sigma_j^l)'(z_j^l)
\end{aligned}$$

Its corresponding matrix form is:

$$\boldsymbol{\delta}^l = \left(\boldsymbol{\delta}^{l+1} (\boldsymbol{W}^{l+1})^{\mathsf{T}}\right) \odot (\boldsymbol{\sigma}^l)'(\boldsymbol{z}^l) \tag{5.11}$$

Now, let's compute $\frac{\partial C}{\partial b_j^l}$:

$$\begin{aligned}
\frac{\partial C}{\partial b_j^l} &= \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} \\
&= \sum_k \delta_k^l \frac{\partial b_k^l}{\partial b_j^l} \\
&= \delta_j^l
\end{aligned}$$

In shorthand, it can be rewritten as:

$$\frac{\partial C}{\partial \boldsymbol{b}} = \boldsymbol{\delta} \tag{5.12}$$

Similarly, we can compute $\frac{\partial C}{\partial w_{ij}^l}$:

$$\begin{aligned}\frac{\partial C}{\partial w_{ij}^l} &= \sum_k \frac{\partial C}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{ij}^l} \\ &= \sum_k \delta_k^l \sum_r \frac{\partial (a_r^{l-1} w_{rk}^l + b_k^l)}{\partial w_{ij}^l} \\ &= \sum_k \delta_k^l a_i^{l-1} \delta_{kj} \\ &= a_i^{l-1}\delta_j^l\end{aligned}$$

In shorthand, it can be rewritten as:

$$\frac{\partial C}{\partial \boldsymbol{W}} = \boldsymbol{a}_{\text{in}}\boldsymbol{\delta}_{\text{out}} \tag{5.13}$$

For simplicity, let's assume that all the activation functions are the same, i.e. $\sigma_i^l = \sigma$, then we can write the pseudocode of backpropagation algorithm easily as the following:

Algorithm 2 Backpropagation

Input: $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
1: **for** $l = 1$ **to** L **do**
2: Compute $\boldsymbol{z}^l = \boldsymbol{a}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l$ and $\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$.
3: **end for**
4: Compute $\boldsymbol{\delta}^l = \nabla_{\boldsymbol{a}^L} C \odot \boldsymbol{\sigma}'(\boldsymbol{z}^L)$ ▷ $\nabla_{\boldsymbol{a}^L} C = \boldsymbol{a}^L - \boldsymbol{y}$ if C is square loss.
5: **for** $l = L-1$ **to** 1 **do**
6: Compute $\boldsymbol{\delta}^l = \left(\boldsymbol{\delta}^{l+1}(\boldsymbol{W}^{l+1})^{\mathsf{T}}\right) \odot \sigma'(\boldsymbol{z}^l)$
7: **end for**
Output: $\frac{\partial C}{\partial w_{ij}^l} = a_i^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

We are now ready to do the vectorization. Let $\boldsymbol{x}^i$ and $\boldsymbol{y}^i$ be the i-th example and its output respectively, let

$$\begin{aligned}\boldsymbol{X} &= (\boldsymbol{x}^1; \ldots; \boldsymbol{x}^m)_{m\times n} \\ \boldsymbol{Y} &= (\boldsymbol{y}^1; \ldots; \boldsymbol{y}^m)_{m\times n_L}\end{aligned}$$

the input matrix. Let $\boldsymbol{z}^{i,l}, \boldsymbol{a}^{i,l}, \boldsymbol{\delta}^{i,l}$ be the raw output, output, error vectors w.r.t. the i-th example respectively. Let

$$\begin{aligned}\boldsymbol{Z}^l &= (\boldsymbol{z}^{1,l}; \ldots; \boldsymbol{z}^{m,l})_{m\times n_l} \\ \boldsymbol{A}^l &= (\boldsymbol{a}^{1,l}; \ldots; \boldsymbol{a}^{m,l})_{m\times n_l} \\ \boldsymbol{\Delta}^l &= (\boldsymbol{\delta}^{1,l}; \ldots; \boldsymbol{\delta}^{m,l})_{m\times n_l}\end{aligned}$$

then we have:

$$\boldsymbol{Z}^l = \boldsymbol{A}^{l-1}\boldsymbol{W}^l + \boldsymbol{b}^l \tag{5.14}$$

$$\boldsymbol{A}^l = \sigma(\boldsymbol{Z}^l) \tag{5.15}$$

$$\boldsymbol{\Delta}^L = \nabla_{\boldsymbol{A}^L} C \odot \sigma'(\boldsymbol{Z}^L) \tag{5.16}$$

$$\boldsymbol{\Delta}^l = \left(\boldsymbol{\Delta}^{l+1}(\boldsymbol{W}^{l+1})^{\mathsf{T}}\right) \odot \sigma'(\boldsymbol{Z}^l) \tag{5.17}$$

If $C = \frac{1}{m}\sum_{i=1}^{m} C(\boldsymbol{x}^i)$, then

$$\frac{\partial C}{\partial b_j^l} = \text{reduce\_mean}(\text{col}_j(\boldsymbol{\Delta}^l))$$

and

$$\frac{\partial C}{\partial w_{ij}^l} = \text{reduce\_mean}(\text{col}_i(\boldsymbol{A}^{l-1}) \odot \text{col}_j(\boldsymbol{\Delta}^l))$$

5.1.3 Cost Functions

5.1.4 Regularizations

L^1 and L^2 Regularizations

Dropout

5.2 Convolutional Neuron Networks

5.2.1 Padding

Chapter 6

Bayesian

Chapter 7

Clustering

7.1 Clustering

Chapter 8

Expectation Maximalization

Chapter 9

Ensemble Learning

9.1 AdaBoost

AdaBoost combines a family of binary classification base (weak) learners from an algorithm linearly to get a stronger (more powerful) one. It takes advantage of a former learner and use its output to force the latter learner to focus more on those data on which the former learner performs poorly. In this way, it hopes to make the aggregate learner performs well on all training data. Thus AdaBoost focus more on reducing the bias.

Algorithm 3 AdaBoost

Input: training set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$; base learner $\mathcal{L}$; training round T.

1: $\mathcal{D}_1(\boldsymbol{x}) = \frac{1}{m}$. ▷ $\mathcal{D}$ is a distribution over the input examples.
2: **for** $t = 1$ **to** T **do**
3: $h_t = \mathcal{L}(D, \mathcal{D}_t)$;
4: $\varepsilon_t = \mathbb{P}_{\boldsymbol{x} \sim \mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$; ▷ ε_t is the error rate of h_t w.r.t $\mathcal{D}_t$.
5: **if** $\varepsilon_t > 0.5$ **then break** ▷ Base learner must be above average.
6: $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$;
7: $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$. ▷ Z_t is the normalization constant.
8: **end for**

Output: $H(\boldsymbol{x}) = \operatorname{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

Consider the exponential loss function

$$L_{\exp}(H|\mathcal{D}) = \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathcal{D}}[e^{-f(\boldsymbol{x})H(\boldsymbol{x})}] \tag{9.1}$$

We want to find a linear combination of base learners

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$$

This is the so called greedy method.

that minimize it. Although it is difficult to find such H as a whole, we can approach this problem step by step. That is, we first find some h_1 and α_1 s.t. $H_1 = \alpha_1 h_1$ minimize the exponential loss, then we find some h_2 and α_2 s.t. $H_2 = H_1 + \alpha_2 h_2$ minimize the exponential loss too. And we carray on this process until $t = T$. Assume we have H_{t-1} and want to find the next learner and its weight, that is, we want to find α_t and h_t s.t. $H_t = H_{t-1} + \alpha_t h_t$ minimize

the exponential loss. Since

$$\begin{aligned} L_{\exp}(H_t|\mathcal{D}) &= L_{\exp}(H_{t-1} + \alpha_t h_t|\mathcal{D}) \\ &= \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})} \cdot e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}] \\ &= C_{t-1} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{C_{t-1}} e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}] \\ &= C_{t-1} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}_t}[e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}] \end{aligned}$$

where $C_{t-1} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}]$ is a constant and $\mathcal{D}_t = \frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{C_{t-1}}\mathcal{D}$ is a distribution, we have

$$\begin{aligned} L_{\exp}(H_t|\mathcal{D}) &= C_{t-1}\left\{e^{-\alpha_t} \mathop{\mathbb{P}}_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x}) = h_t(\boldsymbol{x})) + e^{\alpha_t} \mathop{\mathbb{P}}_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x}) \neq h_t(\boldsymbol{x}))\right\} \\ &= C_{t-1}\{e^{-\alpha_t}(1-\varepsilon_t) + e^{\alpha_t}\varepsilon_t\} \end{aligned} \tag{9.2}$$

where ε_t is the error rate of h_t w.r.t. the distribution $\mathcal{D}_t$. Hence

$$\frac{\partial}{\partial\alpha_t} L_{\exp}(H_t|\mathcal{D}) = 0$$

implies

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t}) \tag{9.3}$$

which justifies the choice of α_t in line 3.6. Moreover,

$$\mathcal{D}_t = \frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{C_{t-1}}\mathcal{D}$$

implies

$$\begin{aligned} \mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})} C_{t-1}\mathcal{D}_t(\boldsymbol{x})}{C_t} \\ &= \frac{e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\mathcal{D}_t(\boldsymbol{x})}{Z_t} \end{aligned} \tag{9.4}$$

which justifies the choice of $\mathcal{D}_t$ in line 3.7. If we plug equation (9.3) into (9.2), we get

$$L_{\exp}(H_t|\mathcal{D}) = 2C_{t-1} \cdot \sqrt{\varepsilon_t(1-\varepsilon_t)} \tag{9.5}$$

Since the performance of the base learner is slightly above the average, that is, $\varepsilon_t < \frac{1}{2}$, the minimum of the exponential loss is achieved at the h_t which has the minimal error rate w.r.t. $\mathcal{D}_t$. That is, h_t is the output of the algorithm w.r.t. the distribution $\mathcal{D}_t$ over the training data.

Moreover, notice that

$$C_{t-1} = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}] = L_{\exp}(H_{t-1}|\mathcal{D})$$

equation (9.5) implies that

$$\begin{aligned} L_{\exp}(H|\mathcal{D}) &= \prod_{t=1}^{T} \sqrt{4\varepsilon_t(1-\varepsilon_t)} \\ &= \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \end{aligned} \tag{9.6}$$

where $\gamma_t = \frac{1}{2} - \varepsilon_t$ and we use the fact that $L_{\exp}(0|\mathcal{D}) = 0$. Since

$$1 + x \leqslant e^x \quad \forall\ x \in \mathbb{R}$$

we have

$$\sqrt{1-4\gamma_t^2} \leqslant \sqrt{e^{-4\gamma_t^2}} = e^{-2\gamma_t^2}$$

which implies

$$L_{\exp}(H|\mathcal{D}) \leqslant e^{-2\sum_{t=1}^{T}\gamma_t^2} \tag{9.7}$$

If there is some $\varepsilon \in (0, \frac{1}{2})$ s.t. $\varepsilon_t \leqslant \varepsilon\ \forall\ t$, then we have $\gamma_t \geqslant \gamma = \frac{1}{2} - \varepsilon$, hence

$$L_{\exp}(H|\mathcal{D}) \leqslant e^{-2\gamma^2 T} \tag{9.8}$$

That is, the loss function $L_{\exp}$ decreases exponentially in T.

9.2 Bagging

Bagging is short for bootstrap aggregating, namely we use bootstrap sampling to sample T groups of training data, and train T base learners using those groups, then we aggregate those T base learners into one via plurality vote (for classification) or averaging (for regression). In this way, bagging mainly focus on reducing the variance.

Algorithm 4 Bagging

Input: training set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$; base learner $\mathcal{L}$; training round T.

1: **for** $t = 1$ **to** T **do**
2: $\quad h_t = \mathcal{L}(D, \mathcal{D}_{bs})$ ▷ bs stands for bootstrap sampling.
3: **end for**

Output: $H(\boldsymbol{x}) = \underset{y\in\mathcal{Y}}{\operatorname{argmax}} \sum_{t=1}^{T} 1(h_t(\boldsymbol{x}) = y)$ ▷ H is the plurality vote of h_t.

Unlike the AdaBoost in the preceding section, bagging can be applied to multiclass classification and regression without any modifications. Morover, due to bootstrap sampling, we have around 36.8% samples untouched in each training, which can be used to do the out-of-bag estimation for testing the generalization ability.

9.3 Random Forest

A forest is a collection of trees. When applying bagging to decision tree, we get a forest. A random forest is obtained this way with randomness. Namely, when deciding the best feature for an internal node, we restrict the candidates to an randomly chosen subset from all possible features. Usually, the size of subset is $\log_2 d$ where d is the total number of all possible features.

9.4 Gradient Boost

9.5 Ensemble Strategy

Chapter 10

Dimension Reduction

10.1 Low-dimensional embedding

Let the feature space $\mathcal{X} = \mathbb{R}^d$. Our goal is to find an embedding of all samples into a low dimension space $\mathbb{R}^{d'}$, here $d' \leq d$. That is, we want to find a map $e : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}$ s.t. for any samples $\boldsymbol{x}_i, \boldsymbol{x}_j$, we have

$$\mathrm{dist}(e(\boldsymbol{x}_i), e(\boldsymbol{x}_j)) = \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Thus $\{e(\boldsymbol{x}_i)\}_i$ is a low-dimensional embedding of the original samples.

Let $\boldsymbol{D} = (\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j))_{m\times m}$ be the distance matrix of the samples $\{\boldsymbol{x}_i\}_i$, we want to find the embedding matrix $\boldsymbol{Z} \in \mathbb{R}^{m\times d'}$, s.t. $||\boldsymbol{z}_i - \boldsymbol{z}_j|| = \boldsymbol{D}_{ij}$, where $\boldsymbol{Z} = (\boldsymbol{z}_1; \ldots; \boldsymbol{z}_m)$ and $\boldsymbol{D}_{ij} = \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Hence we have

$$\boldsymbol{D}_{ij}^2 = ||\boldsymbol{z}_i||^2 + ||\boldsymbol{z}_j||^2 - 2\langle \boldsymbol{z}_i, \boldsymbol{z}_j\rangle$$

Let $\boldsymbol{B} = (b_{ij})_{m\times m} = \boldsymbol{Z}\boldsymbol{Z}^\mathsf{T}$ where $b_{ij} = \langle \boldsymbol{z}_i, \boldsymbol{z}_j\rangle$, we have

$$\boldsymbol{D}_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij} \tag{10.1}$$

Let $\boldsymbol{c} = \frac{1}{m}\sum_i \boldsymbol{z}_i$, then $\{\boldsymbol{z}_i - \boldsymbol{c}\}_i$ is an embedding with the property that $\sum_i(\boldsymbol{z}_i - \boldsymbol{c}) = 0$. Hence we can require that $\sum_i \boldsymbol{z}_i = 0$ in the above discussion. Then from equation (10.1), we know that:

$$\sum_i \boldsymbol{D}_{ij}^2 = \mathrm{tr}(\boldsymbol{B}) + mb_{jj} \tag{10.2}$$

$$\sum_j \boldsymbol{D}_{ij}^2 = \mathrm{tr}(\boldsymbol{B}) + mb_{ii} \tag{10.3}$$

$$\sum_{i,j} \boldsymbol{D}_{ij}^2 = 2m\,\mathrm{tr}(\boldsymbol{B}) \tag{10.4}$$

From the above equations, we have

$$b_{ij} = -\frac{1}{2}\left(\boldsymbol{D}_{ij}^2 - \frac{1}{m}\sum_i \boldsymbol{D}_{ij}^2 - \frac{1}{m}\sum_j \boldsymbol{D}_{ij}^2 + \frac{1}{m^2}\sum_{i,j} \boldsymbol{D}_{ij}^2\right) \tag{10.5}$$

That is, $\boldsymbol{B}$ is totally determined by $\boldsymbol{D}$. Let $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}$ be the eigenvalue decomposition of $\boldsymbol{B}$. Since $\boldsymbol{B}$ is semi-positive, $\boldsymbol{\Lambda}$ is a diagonal matrix with non-negative diagonals. Let $\boldsymbol{\Lambda}_*$ be the diagonal matrix obtained by removing the zero eigenvalues from $\boldsymbol{\Lambda}$ and $\boldsymbol{V}_*$ the matrix by removing the corresponding columns of $\boldsymbol{V}$. Then it's easy to conclude that

$$\boldsymbol{Z} = \boldsymbol{V}_*\boldsymbol{\Lambda}_*^{1/2}$$

is what we want. Hence the dimension d' is totally determined by the distance

matrix $\boldsymbol{D}$.

In practise, we often fix some $d' \ll d$ at first, then obtain $\boldsymbol{\Lambda}_*$ by keeping the d' largest eigenvalues and remove the rest, and $\boldsymbol{V}_*$ the corresponding matrix. By this way, we may lose some precision in keeping the pairwise distance, but we can greatly reduce the dimension.

Algorithm 5 Multiple Dimensional Scaling

Input: the distance matrix $\boldsymbol{D}_{m\times m}$; dimension d'.

1: Compute the matrix $\boldsymbol{B}$ according to equation (10.5).
2: Eigenvalue decomposition: $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}$.
3: Let $\boldsymbol{\Lambda}_*$ be the diagonal matrix with the d' largest eigenvalues and $\boldsymbol{V}_*$ the matrix with the corresponding eigenvectors.

Output: Low-dimensional embedding: $\boldsymbol{Z} = \left(\boldsymbol{V}_*\boldsymbol{\Lambda}_*^{1/2}\right)_{m\times d'}$

10.2 Linear Dimension Reduction

The simplest way to reduce dimension is by dropping some coordinates, that is, via projection. This keeps the linear structure of the samples. A more generalized way is via linear transformation: $A : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}$. In matrix form, we want to find a transformation matrix $\boldsymbol{W} \in \mathbb{R}^{d\times d'}$ s.t. for any row vector $\boldsymbol{x} \in \mathbb{R}^d$, we have $A(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W}$. If we write $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d'})$, then we have $A(\boldsymbol{x}) = (\langle \boldsymbol{x}, \boldsymbol{w}_1\rangle, \ldots, \langle \boldsymbol{x}, \boldsymbol{w}_{d'}\rangle)$. If $\{\boldsymbol{w}_i\}_{i=1}^{d'}$ is orthonormal, then $A(\boldsymbol{x})$ is the projection of $\boldsymbol{x}$ into the subspace spanned by $\{\boldsymbol{w}_i\}_{i=1}^{d'}$. It's easy to see that if $\boldsymbol{X} = (\boldsymbol{x}_1; \ldots; \boldsymbol{x}_m)_{m\times d}$ is the feature matrix of the samples, then $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W} \in \mathbb{R}^{m\times d'}$ is the representation matrix desired.

10.2.1 Principle Component Analysis

The naive projection method may fail because of the huge possibility of dropping those components that really matters while keeping those that have little information about the dataset. To overcome this, we need to determine what is it mean for a component to be principle. A natrual measurement is the variance. Namely, the component with largest variance and lowest covariance w.r.t. others counts most. We only drop out those with small variances.

Let $\boldsymbol{X} = (\boldsymbol{x}_1; \ldots; \boldsymbol{x}_m)$ be the feature matrix as above. Without loss of generality, we may assume that $\sum_i \boldsymbol{x}_i = 0$. Then $\boldsymbol{X}^\mathsf{T}\boldsymbol{X} \in \mathbb{R}^{d\times d}$ is the covariance matrix of $\boldsymbol{X}$. If $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is diagonal, then the largest eigenvalue corresponds to the principle component. Hence we may keep the components corresponding to the d' largest eigenvalues. If $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ is not diagonal, let $\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}$ be its eigenvalue decomposition. Then we have $\boldsymbol{V}^\mathsf{T}\boldsymbol{X}^\mathsf{T}\boldsymbol{X}\boldsymbol{V} = \boldsymbol{\Lambda}$. The matrix $(\boldsymbol{V}_*)_{d\times d'}$ consists of the eigenvectors corresponding to the d' largest eigenvalues

of $\mathbf{\Lambda}$ is the desired transformation matrix. And $\boldsymbol{X}\boldsymbol{V}_*$ is the representation of the original samples in low dimensional space $\mathbb{R}^{d'}$.

Algorithm 6 Principle Component Analysis

Input: dataset $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$; dimension d'.

1: Compute the center of samples: $\boldsymbol{c} = \frac{1}{m}\sum_i \boldsymbol{x}_i$
2: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \boldsymbol{c}$
3: Compute the covariance matrix $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ ▷ $\boldsymbol{X} = (\boldsymbol{x}_1; \ldots; \boldsymbol{x}_m)$
4: Eigenvalue decomposition: $\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\mathbf{\Lambda}\boldsymbol{V}^\mathsf{T}$
5: Fetch the (column) eigenvectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d'}$ from $\boldsymbol{V}$ corresponding to the d' largest eigenvalues of $\mathbf{\Lambda}$.

Output: transformation matrix $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{d'})$

Remark 10.1 The PCA method actually consists of two steps:

1. Centralization: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{c} \quad \forall\, \boldsymbol{x}$ where the center $\boldsymbol{c}$ is computed from the the given data set.
2. Linear transformation: $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W}$ where $\boldsymbol{z}, \boldsymbol{x}$ are both row vectors and $\boldsymbol{z}$ is the desired representation in low dimensional space.

Of course, those two steps should also apply to unseen data.

Remark 10.2 In the above PCA algorithm 6, we can use the **singular value decomposition** instead of the eigenvalue decomposition. Let $\boldsymbol{X} = \boldsymbol{U}\mathbf{\Sigma}\boldsymbol{V}^\mathsf{T}$ be the singular value decomposition of $\boldsymbol{X}$, then we have $\boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\mathbf{\Sigma}^\mathsf{T}\mathbf{\Sigma}\boldsymbol{V}^\mathsf{T}$. Notice that $\mathbf{\Sigma}^\mathsf{T}\mathbf{\Sigma}$ is a diagonal. We can proceed as in the above algorithm to get the transformation matrix.

There are other approaches to derive the PCA algorithm.

Chapter 11

Computational Learning Theory

11.1 Computational Learning Theory

In this section, we mainly consider supervised learning. Let $\mathcal{X}$ be the instance space, $\mathcal{Y}$ the label set, $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ the training set. Assume $\mathcal{D}$ is the distribution on $\mathcal{X}$, and all instances of D are sampled i.i.d. according to $\mathcal{D}$. Let $f : \mathcal{X} \longrightarrow \mathcal{Y}$ be the underlying labeling function and $h : \mathcal{X} \longrightarrow \mathcal{Y}$ any prediction function, then the **true (generalization) loss (error)** is defined as:

$$L_{\mathcal{D},f}(h) := \mathop{\mathbb{P}}_{\boldsymbol{x}\sim\mathcal{X}}(h(\boldsymbol{x}) \neq f(\boldsymbol{x})) := \mathcal{D}\left(\{\boldsymbol{x} : h(\boldsymbol{x}) \neq f(\boldsymbol{x})\}\right)$$

the **empirical risk (error, loss)** is defined as:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{I}(h(\boldsymbol{x}) \neq f(\boldsymbol{x}))$$

11.1.1 Probably Approximately Correct Learning

Definition 11.1 (Concept class) A (target) concept is just a true labeling function $c : \mathcal{X} \longrightarrow \mathcal{Y}$, that is, for any instace $(\boldsymbol{x}, y)$ (assuming sampling process is noise free) we have $c(\boldsymbol{x}) = y$. The collection $\mathcal{C}$ of all target concepts is called the concept class.

Definition 11.2 (Hypothesis space) The collection of all labeling functions $f : \mathcal{X} \longrightarrow \mathcal{Y}$ a learner $\mathcal{L}$ can return is called the hypothesis space (w.r.t. $\mathcal{L}$). We denote it as $\mathcal{H}$.

Remark 11.3 (Inductive bias) By restricting our learner to the hypothesis space instead of arbitrary predictors, we bias it toward a particular set of predictors. Such restrictions are called **inductive bias**.

Definition 11.4 (Realizability Assumption) The realizable assumption asserts that there is a $h^* \in \mathcal{H}$ s.t. $L_{\mathcal{D},f}(h^*) = 0$.

Remark 11.5 The realizable assumption implies that with probability 1 over i.i.d. samples D, we have $L_D(h^*) = 0$. That is, $\mathcal{D}^m(\{D : L_D(h^*) = 0\}) = 1$.

Definition 11.6 (PAC learnability) The concept class $\mathcal{C}$ is PAC learnable w.r.t. a hypothesis space $\mathcal{H}$ if there exist

1. a function $m : (0,1)^2 \longrightarrow \mathbb{N}$;
2. a learner $\mathcal{L}$.

s.t. for any $\varepsilon, \delta \in (0,1)$, for any distribution $\mathcal{D}$ over $\mathcal{X}$, and for any concept $c : \mathcal{X} \longrightarrow \mathcal{Y}$, if the realizable assumption holds w.r.t. $\mathcal{H}, \mathcal{D}, c$, then when applying the learner $\mathcal{L}$ to $m \geq m(\varepsilon, \delta)$ i.i.d. samples generated by $\mathcal{D}$ and labeled by c,

the learner returns a hypothesis h s.t. with probability at least $1 - \delta$ (over the choice of the samples), we have $L_{\mathcal{D},c}(h) \leq \varepsilon$. That is,

$$\mathcal{D}^m(\{D : L_{\mathcal{D},c}(h) \leq \varepsilon\}) \geq 1 - \delta$$

Definition 11.7 (Sample complexity) The sample complexity of a learner is the minimal number of examples needed for the learner to produce a PAC solution on any i.i.d. data sets with that many samples. That is, it is the minimum of all $m(\varepsilon, \delta)$ where m satisfies the requirements in definition 11.6.

Chapter 12

Reinforcement Learning

Let X be the state space, A the action space. Reinforcement learning can be described as a **Markov Decision Process**: system changes from a state to another under the actions from A with a rewarding function grading the change. That is, a reinforcement learning is a quadruple $E = \langle X, A, P, R \rangle$, where $P : X \times A \times X \longrightarrow \mathbb{R}$ describes the probability of a state changed into another via an action from A and $R : X \times A \times X \longrightarrow \mathbb{R}$ describes the reward of that change. Sometimes the rewarding function only depends on states, that is $R : X \times X \longrightarrow \mathbb{R}$. Note that given a state x and an action a that can act on it, the resulting state x' may not be unique, but the identity $\sum_{x' \in X} P^a_{x \to x'} = 1$ always holds true.

The goal of reinforcement leanring is to find a policy π dictating which action to take given a state x. π can be described in a determinate form $\pi : X \longrightarrow A$, which dictates that action $\pi(x)$ must be performed on state x. It can also be described in a probability form: $\pi : X \times A \longrightarrow \mathbb{R}$ where $\pi(x, a)$ is the probability of performing action a on x. Obviously, the determinate form is a special case of the probability form. The performance measurement of the policy is the accumulated reward gained from performing this policy for a reasonably long time. The T **steps accumulated reward** $\mathbb{E}[\frac{1}{T} \sum_{t=1}^{T} r_t]$ and γ **discount accumulated reward** $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}]$ are two commonly used accumulated rewards, where r_t is the reward of step t.

12.1 K-armed Bandit

The K-armed bandit is a model of one step reinforcement learning. Every time you pull one arm of the machine, it gives you some rewards with certain probability. The goal is to acquire as many rewards as possible.

12.1.1 Exploration and Exploitation

The **exploration-only** method tries to figure out the expectation reward of each arm. It divides the exploration chance evenly on each arm, and use the average reward of each arm as the expectation reward. This method may do a good job in estimating each arm's reward, but it may miss the opportunity to get the most rewards. The **exploitation-only** method, on the other hand, only pulls the arm that gives the most average rewards so far. Since it dosen't care about the expectation reward of each arm, it may also miss the opportunity to get the most rewards.

12.1.2 ε-greedy

The ε-greedy is a compromise between exploration-only and exploitation-only method. At each try, it performs exploration-only method with a probability ε

and exploitation-only method the other $1 - \varepsilon$.

Let $Q_n(k)$ denote the average reward of arm k with n tries and v_n^k its n-th reward, then it is clearly that

$$Q_n(k) = \frac{1}{n}((n-1)Q_{n-1}(k) + v_n^k)$$

This is the updating rules for the average reward.

Algorithm 7 ε-greedy

Input: number of arms K; rewarding function R; number of tries T; exploration threshold ε.

$r = 0$ ▷ the accumulated reward.
$Q(i) = 0, \quad count(i) = 0 \quad \forall\ i = 1, \ldots, K$ ▷ Initialization.
for $t = 1$ **to** T **do**
  if $rand() < \varepsilon$ **then**
    $k = rand(\{1, \ldots, K\})$ ▷ choose with equal probability
  else
    $k = \text{argmax}_i\, Q(i)$
  end if
  $v = R(k)$
  $r \leftarrow r + v$
  $Q(k) \leftarrow \frac{count(k) \cdot Q(k) + v}{count(k)+1}$
  $count(k) \leftarrow count(k) + 1$
end for

Output: the accumulated reward r.

12.1.3 Softmax

Softmax algorithm is another way of compromising between the exploration-only and exploitation-only methods. Unlike the ε-greedy algorithm which uses a threshold to determine how to choose an arm, softmax attach each arm with a probability to be chosen base on its current average reward. The probability distribution among the arms is a Boltzmann distribution, namely:

$$P(k) = \frac{e^{\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}} \quad \forall\ k = 1, \ldots, K \tag{12.1}$$

where the parameter $\tau > 0$. Apparently, if τ is close to 0, the softmax favours the arm with the highest average reward, which is the case of exploitation-only method; if τ is very large (close to $+\infty$), the softmax degenerates to uniform distribution, which is the case of exploration-only method.

Algorithm 8 Softmax

Input: number of arms K; rewarding function R; number of tries T; parameter τ.

1: $r = 0$
2: $Q(i) = 0, \quad count(i) = 0 \quad \forall\, i = 1, \ldots, K$
3: **for** $t = 1$ **to** T **do**
4: choose k according to the distribution given by equation (12.1).
5: $v = R(k)$
6: $r \leftarrow r + v$
7: $Q(k) \leftarrow \frac{count(k) \cdot Q(k)}{count(k)+1}$
8: $count(k) \leftarrow count(k) + 1$
9: **end for**

Output: the accumulated reward r.

12.2 Model-based Learning

In model-based learning, the quadruple $E = \langle X, A, P, R \rangle$ are known to us. That is, we know all the possible states, all the possible actions that may act on them, the transition function which describes the probability of one state changing into another under some action, the rewarding function which grades the change. In the following discussion, we assume that the state space X and the action space A are finite.

12.2.1 Policy Measurement

Let $V^\pi(x)$ be the expected reward gained from applying policy π on the starting state x, $Q^\pi(x, a)$ the expected reward gained from first applying action a then policy π on the starting state x. V and Q are called **state value function** and **state-action value function** respectively.

By definition, we can write the state value function as:

$$V_T^\pi(x) = \mathop{\mathbb{E}}_\pi \left[\frac{1}{T} \sum_{t=1}^{T} r_t | x_0 = x \right], \qquad T \text{ steps} \tag{12.2}$$

$$V_\gamma^\pi(x) = \mathop{\mathbb{E}}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | x_0 = x \right], \qquad \gamma \text{ discount} \tag{12.3}$$

Similarly, we can write the state-action value function as:

$$Q_T^\pi(x, a) = \mathop{\mathbb{E}}_\pi \left[\frac{1}{T} \sum_{t=1}^{T} r_t | x_0 = x, a_0 = a \right], \qquad T \text{ steps} \tag{12.4}$$

$$Q_\gamma^\pi(x, a) = \mathop{\mathbb{E}}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | x_0 = x, a_0 = a \right] \qquad \gamma \text{ discount} \tag{12.5}$$

Since the next state of the system only depends on the current state, those

value functions can be written in a recursive form. For example, the T steps accumulated state value function (12.2) can be written as:

$$\begin{aligned} V_T^\pi(x) &= \mathop{\mathbb{E}}_\pi\left[\frac{1}{T}\sum_{t=1}^{T} r_t | x_0 = x\right] \\ &= \mathop{\mathbb{E}}_\pi\left[\frac{1}{T} r_1 + \frac{T-1}{T}\frac{1}{T-1}\sum_{t=2}^{T} r_t | x_0 = x\right] \\ &= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P^a_{x\to x'}\left(\frac{1}{T}R^a_{x\to x'} + \frac{T-1}{T}\mathop{\mathbb{E}}_\pi\left[\frac{1}{T-1}\sum_{t=1}^{T-1} r_t | x_0 = x'\right]\right) \\ &= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P^a_{x\to x'}\left(\frac{1}{T}R^a_{x\to x'} + \frac{T-1}{T}V^\pi_{T-1}(x')\right) \end{aligned} \tag{12.6}$$

Algorithm 9 T steps accumulated state value function

Input: $E = \langle X, A, P, R\rangle$; policy π; steps T.

1: $V(x) = 0 \quad \forall\, x \in X$
2: **for** $t = 1$ **to** T **do**
3: $\quad V'(x) = \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P^a_{x\to x'}\left(\frac{1}{t}R^a_{x\to x'} + \frac{t-1}{t}V(x')\right) \quad \forall\, x \in X$
4: $\quad$ **if** $t = T+1$ **then**
5: $\quad\quad$ **break**
6: $\quad$ **else**
7: $\quad\quad V \leftarrow V'$
8: $\quad$ **end if**
9: **end for**

Output: state value function V.

Similarly, the γ discount accumulated state value function (12.3) can be written as:

$$\begin{aligned} V^\pi_\gamma(x) &= \mathop{\mathbb{E}}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1} | x_0 = x\right] \\ &= \mathop{\mathbb{E}}_\pi\left[r_0 + \gamma\sum_{t=1}^{\infty}\gamma^{t-1} r_{t+1} | x_0 = x\right] \\ &= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P^a_{x\to x'}\left(R^a_{x\to x'} + \gamma\mathop{\mathbb{E}}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1} | x_0 = x'\right]\right) \\ &= \sum_{a\in A}\pi(x,a)\sum_{x'\in X} P^a_{x\to x'}\left(R^a_{x\to x'} + \gamma V^\pi_\gamma(x')\right) \end{aligned} \tag{12.7}$$

Since the state-action value functions are just the state value functions with the given first action, we can easily write the state-action value functions in

Algorithm 10 γ discount accumulated state value function

Input: $E = \langle X, A, P, R \rangle$; policy π; parameter γ; threshold θ.

$V(x) = 0 \quad \forall\, x \in X$
loop
  $V'(x) = \sum_{a \in A} \pi(x,a) \sum_{x' \in X} P^a_{x \to x'} \left(R^a_{x \to x'} + \gamma V(x') \right) \quad \forall\, x \in X$
  if $\max_{x \in X} |V(x) - V'(x)| < \theta$ **then**
    break
  else
    $V \leftarrow V'$
  end if
end loop

Output: state value function V.

terms of state value functions:

$$Q^\pi_T(x,a) = \sum_{x' \in X} P^a_{x \to x'} \left(\frac{1}{T} R^a_{x \to x'} + \frac{T-1}{T} V^\pi_{T-1}(x') \right) \tag{12.8}$$

$$Q^\pi_\gamma(x,a) = \sum_{x' \in X} P^a_{x \to x'} \left(R^a_{x \to x'} + \gamma V^\pi_\gamma(x') \right) \tag{12.9}$$

Hence it is obvious that:

$$V^\pi(x) = \sum_{a \in A} \pi(x,a) Q^\pi(x,a) \tag{12.10}$$

12.2.2 Policy Improvement

Of course, the best policy is the one that maximize the state value function, i.e.

$$\pi^* = \operatorname*{argmax}_\pi \sum_{x \in X} V(x)$$

Due to the recursive nature of the state value functions (12.6) and (12.7), we can conclude that if there is no restriction on the policies that we can choose, the optimal state value functions satisfy the following equations:

$$V^*_T(x) = \max_{a \in A} \sum_{x' \in X} P^a_{x \to x'} \left(\frac{1}{T} R^a_{x \to x'} + \frac{T-1}{T} V^*_{T-1}(x') \right) \tag{12.11}$$

$$V^*_\gamma(x) = \max_{a \in A} \sum_{x' \in X} P^a_{x \to x'} \left(R^a_{x \to x'} + \gamma V^*_\gamma(x') \right) \tag{12.12}$$

That is,

$$V^*(x) = \max_{a \in A} Q^*(x,a) \tag{12.13}$$

This in turn gives us the expression of $Q^*(x,a)$:

$$Q_T^*(x,a) = \sum_{x' \in X} P_{x \to x'}^a \left(\frac{1}{T} R_{x \to x'}^a + \frac{T-1}{T} \max_{a' \in A} Q_{T-1}^*(x', a') \right) \tag{12.14}$$

$$Q_\gamma^*(x,a) = \sum_{x' \in X} P_{x \to x'}^a \left(R_{x \to x'}^a + \gamma \max_{a' \in A} Q_\gamma^*(x', a') \right) \tag{12.15}$$

The relation between optimal state value function and optimal state-action value function (12.13) shed light on how to improve the current policy: always choose the current best action for the state. That is, let

$$\pi'(x) = \operatorname*{argmax}_{a \in A} Q^\pi(x,a) \quad \forall\ x \in X \tag{12.16}$$

Then by relation (12.10), we have

$$Q^\pi(x, \pi'(x)) \geqslant V^\pi(x)$$

Since $\pi'(x)$ is the action that should be taken on the state x, by relation (12.10) again, we have

$$\begin{aligned} V^{\pi'}(x) &= \sum_{a \in A} \pi'(x,a) Q^{\pi'}(x,a) \\ &= Q^{\pi'}(x, \pi'(x)) \end{aligned}$$

This gives us the following estimate:

$$\begin{aligned} V^\pi(x) - V^{\pi'}(x) &\leqslant Q^\pi(x, \pi'(x)) - Q^{\pi'}(x, \pi'(x)) \\ &\leqslant \sum_{x' \in X} P_{x \to x'}^{\pi'(x)} \cdot \gamma \left(V^\pi(x') - V^{\pi'}(x') \right) \\ &\leqslant \gamma \max_{x' \in X} \left(V^\pi(x') - V^{\pi'}(x') \right) \sum_{x' \in X} P_{x \to x'}^{\pi'(x)} \\ &\leqslant \gamma \max_{x' \in X} \left(V^\pi(x') - V^{\pi'}(x') \right) \end{aligned}$$

Here, we use the fact that $P \geq 0$ and $\sum_{x' \in X} P_{x \to x'}^{\pi'(x)} = 1$. Since $\gamma \in (0,1)$, we conclude that

$$\max_{x \in X} \left(V^\pi(x) - V^{\pi'}(x) \right) \leqslant 0$$

That is

$$V^\pi(x) \leqslant V^{\pi'}(x) \quad \forall\ x \in X$$

This justifies the choice of $\pi'(x)$.

Remark 12.1 Note that the above discussion is also valid in the case of T-steps accumulated state value function with almost the same argument.

12.3 Model-free Learning

Part II

Selected Papers

Chapter 13

Neuron Networks

13.1 Multilayer Feedforward Networks are Universal Approximators

This paper [2] proves that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measureble function, provided sufficiently many hidden units are available.

13.1.1 Terminology and Notation

Let $\boldsymbol{A}^r = \{A : \mathbb{R}^r \longrightarrow \mathbb{R} | A(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\}$ be the collection of all affine functions, $\boldsymbol{M}^r$ the collection of all Borel measureble functions, $\boldsymbol{C}^r$ the collection of all continuous functions and $\boldsymbol{B}^r$ the collection of all Borel sets, on $\mathbb{R}^r$.

Definition 13.1 (Squashing function) $\phi : \mathbb{R} \longrightarrow [0, 1]$ is a squashing function if and only if it is non-decreasing, satifies $\lim_{\lambda \to \infty} \phi(\lambda) = 1$ and $\lim_{\lambda \to -\infty} \phi(\lambda) = 0$.

Definition 13.2 Let $G : \mathbb{R} \longrightarrow \mathbb{R}$ be a Borel measureble function. Then $\boldsymbol{\Sigma}^r(G)$ is defined as:

$$\boldsymbol{\Sigma}^r(G) = \left\{ f : \mathbb{R}^r \longrightarrow \mathbb{R} \middle| f(\boldsymbol{x}) = \sum_{j=1}^{q} \beta_j G(A_j(\boldsymbol{x})),\ q \in \mathbb{N}, \beta_j \in \mathbb{R}, A_j \in \boldsymbol{A}^r \right\}$$

Obviously, functions in $\boldsymbol{\Sigma}^r(G)$ are the outputs of neuron networks which have a single hidden layer with activation G (no activation in the output layer).

Definition 13.3 Let $G : \mathbb{R} \longrightarrow \mathbb{R}$ be a Borel measureble function. Then $\boldsymbol{\Sigma\Pi}^r(G)$ is defined as:

$$\boldsymbol{\Sigma\Pi}^r(G) = \left\{ f : \mathbb{R}^r \longrightarrow \mathbb{R} \middle| f(\boldsymbol{x}) = \sum_{j=1}^{q} \beta_j \prod_{i}^{l_j} G(A_i(\boldsymbol{x})) \right\}$$

Obviously, $\boldsymbol{\Sigma\Pi}^r(G)$ is the algebra generated by $\boldsymbol{\Sigma}^r(G)$.

Definition 13.4 Let (X, ρ) be a metric space, $S \subseteq T \subseteq X$. Then S is said to be ρ-dense in T if $\forall\ \varepsilon > 0, \forall\ t \in T, \exists\ s \in S$, s.t. $\rho(s, t) < \varepsilon$.

Definition 13.5 $S \subseteq \boldsymbol{C}^r$ is **uniformly dense on compacta** in $\boldsymbol{C}^r$ if $\forall\ K \subseteq \mathbb{R}^r$ which is compact, S is ρ_K-dense in $\boldsymbol{C}^r$. A sequence $\{f_n\}$ converges to f uniformly on compacta if $\forall\ K \subseteq \mathbb{R}^r$ compact, $\lim_{n \to \infty} \rho_K(f_n, f) = 0$. Here the metric ρ_K is defined as:

$$\rho_K(f, g) = \sup_{\boldsymbol{x} \in K} |f(\boldsymbol{x}) - g(\boldsymbol{x})| \quad \forall\ f, g \in \boldsymbol{C}^r$$

13.2 Approximation by Superpositions of a Sigmoidal Function

This paper [1] proves that the class of single hidden layer neuron networks can approximate any continuous function with support in the unit hypercube; only mild conditions are imposed on the activation function.

13.2.1 Terminology and Notation

Let $I_n = [0,1]^n$. Let $\boldsymbol{C}(I_n)$ denote the collection of all continuous function on I_n and $||f||$ the supremum norm of $f \in \boldsymbol{C}(I_n)$. Let $M(I_n)$ denote the space of finite, signed regular Borel measures on I_n.

Definition 13.6 (Discriminatory function) $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ is **discriminatory** if for a measure $\mu \in M(I_n)$

$$\int_{I_n} \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b) d\mu(\boldsymbol{x}) = 0$$

for all $\boldsymbol{w}, b$ implies that $\mu = 0$.

Definition 13.7 (Sigmoidal function) $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ is sigmoidal if

$$\sigma(t) \to \begin{cases} 1 & \text{as } t \to \infty \\ 0 & \text{as } t \to -\infty \end{cases}$$

13.2.2 Main Results

Theorem 13.8 (THM 1) Let σ be any continuous discriminatory function. Then the finite sums of the form

$$G(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j \sigma(\langle \boldsymbol{w_j}, \boldsymbol{x} \rangle + b) \tag{13.1}$$

are dense in $\boldsymbol{C}(I_n)$.

Proof Let $S \subseteq \boldsymbol{C}(I_n)$ be the set of all such function G. If $\overline{S} \neq \boldsymbol{C}(I_n)$, then there is a $f_0 \in \boldsymbol{C}(I_n)$ s.t. $f_0 \notin \overline{S}$. Let $d_0 = \text{dist}(f_0, \overline{S}) = \inf_{s \in \overline{S}} ||f_0 - s||$, then we have $d_0 > 0$. Let L be the linear functional on $\textbf{span}\left\{\overline{S}, f_0\right\}$ defined as:

$$L(s + \alpha f_0) = \alpha$$

It is easy to check that L is a well defined linear functional on $\textbf{span}\left\{\overline{S}, f_0\right\}$ and

$L(s) = 0 \ \forall \ s \in \overline{S}$. Moreover, if $\alpha \neq 0$, we have

$$\begin{aligned} ||s + \alpha f_0|| &\geqslant |\alpha| \cdot || - \tfrac{1}{\alpha} s - f_0|| \\ &\geqslant |\alpha| d_0 \\ &\geqslant d_0 \cdot |L(s + \alpha f_0)| \end{aligned}$$

That is $|L(s + \alpha f_0)| \leqslant \frac{1}{d_0} \cdot ||s + \alpha f_0||$. And this also holds true when $\alpha = 0$ since $L(s) = 0$. Since the supremum norm $|| \cdot ||$ is positive homogenous and subadditive on $\boldsymbol{C}(I_n)$, the Hahn-Banach theorem implies that L can be extended to the whole $\boldsymbol{C}(I_n)$ s.t. $|L(f)| \leqslant \frac{1}{d_0}||f||$. By the Riesz representation theorem, there is some $\mu \in M(I_n)$ s.t.

$$L(h) = \int_{I_n} h(\boldsymbol{x}) d\mu(\boldsymbol{x}) \quad \forall \ h \in \boldsymbol{C}(I_n)$$

Since $\sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b) \in \overline{S}$, we have

$$\int_{I_n} \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b) d\mu(\boldsymbol{x}) = 0$$

for any $\boldsymbol{w}, b$. Hence $\mu = 0$, which is a contradiction with $L(f_0) = 1$. □

Lemma 13.9 (LEMMA 1) Any bounded, measurable sigmoidal function is discriminatory. In particular, any continuous sigmoidal function is discriminatory.

Proof For any $\boldsymbol{x}, \boldsymbol{y}, \theta, \varphi$, we have

$$\sigma_\lambda(\boldsymbol{x}) := \sigma(\lambda(\langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta) + \varphi) \begin{cases} \to 1 & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta > 0 \text{ as } \lambda \to +\infty \\ \to 0 & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta < 0 \text{ as } \lambda \to +\infty \\ = \sigma(\varphi) & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta = 0 \text{ for all } \lambda \end{cases}$$

That is,

$$\lim_{\lambda \to +\infty} \sigma_\lambda(\boldsymbol{x}) = \begin{cases} 1 & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta > 0 \\ 0 & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta < 0 \\ \sigma(\varphi) & \text{for } \langle \boldsymbol{y}, \boldsymbol{x} \rangle + \theta = 0 \end{cases}$$

Since σ is bounded, we have

$$\begin{aligned} 0 &= \lim_{\lambda \to +\infty} \int_{I_n} \sigma_\lambda(\boldsymbol{x}) d\mu(\boldsymbol{x}) \\ &= \int_{I_n} \lim_{\lambda \to +\infty} \sigma_\lambda(\boldsymbol{x}) d\mu(\boldsymbol{x}) \\ &= \sigma(\varphi) \mu(\Pi_{\boldsymbol{y}, \theta}) + \mu(H_{\boldsymbol{y}, \theta}) \end{aligned}$$

where $\Pi_{\boldsymbol{y},\theta} = \{\boldsymbol{x}|\langle \boldsymbol{y}, \boldsymbol{x}\rangle + \theta = 0\}$ and $H_{\boldsymbol{y},\theta} = \{\boldsymbol{x}|\langle \boldsymbol{y}, \boldsymbol{x}\rangle + \theta > 0\}$. Fix $\boldsymbol{y}$, let F be the linear functional

$$F(h) = \int_{I_n} h(\langle \boldsymbol{y}, \boldsymbol{x}\rangle) d\mu(\boldsymbol{x})$$

It is bounded on $L^\infty(\mathbb{R})$. If $h = 1_{x \geqslant \theta}$, then

$$F(h) = \mu(\Pi_{\boldsymbol{y},-\theta}) + \mu(H_{\boldsymbol{y},-\theta}) = 0$$

If $h = 1_{x>\theta}$, similarly, we have

$$F(h) = \mu(H_{\boldsymbol{y},-\theta}) = 0$$

Hence F is zero on simple fucntions, which implies $F = 0$ on $L^\infty(\mathbb{R})$. In particular,

$$F(\sin(\langle \boldsymbol{y}, \boldsymbol{x}\rangle) + i\cos(\langle \boldsymbol{y}, \boldsymbol{x}\rangle)) = \int_{I_n} e^{i\langle \boldsymbol{y}, \boldsymbol{x}\rangle} d\mu(\boldsymbol{x}) = 0$$

for all $\boldsymbol{y}$, which implies that $\mu = 0$. □

Given the above lemma, it is obvious that:

Theorem 13.10 Let σ be any continuous sigmoidal function. Then the finite sums of the form

$$G(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j \sigma(\langle \boldsymbol{w_j}, \boldsymbol{x}\rangle + b) \tag{13.2}$$

are dense in $\boldsymbol{C}(I_n)$.

Bibliography

[1] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control Signals & Systems **9** (1993), no. 3, 17–28. 69

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), no. 5, 359–366. 68