第5章 量子分类

分类算法是数据挖掘、机器学习和模式识别中一个重要的研究领域。它先根据已知分类标签的训练集分析出分类模式，再通过该模式对新输入数据的类别进行预测。分类算法主要包括决策树算法、贝叶斯算法、人工神经网络算法、K 近邻算法、支持向量机算法等。

分类算法被广泛应用，随着信息技术的发展以及大数据时代的到来，目前需要处理的数据量越来越大、数据维度越来越高，运行算法所需要的代价越来越大。因此，迫切需要设计高效、快速的机器学习算法以满足各种分类任务的需求。

量子分类算法利用量子存储和量子并行性加速经典分类算法，相比于其他量子机器学习算法，量子分类算法的研究成果相对较多，本节主要介绍量子支持向量机、量子 K 近邻和量子决策树三种分类算法。

5.1 量子支持向量机

支持向量机(Support Vector Machine，SVM)算法是一种常用的有监督机器学习算法，该算法在给定训练样本的基础上，找到一个超平面，将不同类别的样本分开。2014 年 Rebentrost 等提出了能够实现指数加速的量子支持向量机(Quantum SVM，QSVM)算法，它是最小二乘支持向量机的量子版本。QSVM 主要通过矩阵的指数化和相位估计等算法完成训练，最后进行分类。本节首先介绍支持向量机原理，然后介绍相应的量子算法。

5.1.1 支持向量机原理

假设 N 个训练样本组成的数据集为

$$\{(\boldsymbol{x}_i, y_i): \boldsymbol{x}_i \in \mathbf{R}^M, \quad y_i \in \{1, -1\}\}_{i=0,1,\cdots,N-1} \tag{5.1.1}$$

式中：$\boldsymbol{x}_i$ 为单个样本的 M 维特征向量，$\boldsymbol{x}_i = (x_{0i} \quad x_{1i} \quad \cdots \quad x_{(M-1)i})^{\mathrm{T}}$ $(i=0,1,\cdots,N-1)$；y_i 为第 i 个样本的类别。

分类的原理是找到一个超平面，将分属不同类别的样本分开。如图 5.1 所示，有很多的超平面(图 5.1 中的细实线)可以将两个类区分开。支持向量机的基本思想是从这许多的超平面中找到一个最优超平面 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$(图 5.1 中的粗实线)，使得两个类的最

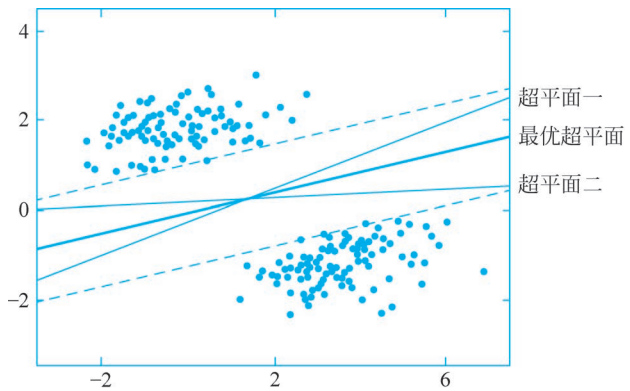


图 5.1 支持向量机原理示意图

前沿(图 5.1 中的虚线)到超平面的距离$\frac{2}{\|\boldsymbol{w}\|}$最大。正好处在最前沿的样本称为支持向量。对所有标签 $y_i=1$ 的样本,$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b>0$;对所有标签 $y_i=-1$ 的样本,$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b<0$。

因此,SVM 的数学表达式为

$$\max_{\boldsymbol{w},b}\ \frac{2}{\|\boldsymbol{w}\|}$$
$$\text{s.t.}\ \ y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)\geqslant 1,\quad i=0,1,\cdots,N-1 \tag{5.1.2}$$

其等价形式为

$$\min_{\boldsymbol{w},b}\ \frac{\|\boldsymbol{w}\|^2}{2}$$
$$\text{s.t.}\ \ y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)\geqslant 1,i=0,1,\cdots,N-1 \tag{5.1.3}$$

式中:$\boldsymbol{w}$ 决定了超平面 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$ 的方向;b 是位移,决定了超平面到原点的距离。

最小二乘支持向量机通过引入非负参数 ξ_i,将式(5.1.3)转换为下列形式:

$$\min_{\boldsymbol{w},b,\xi_i}\ \frac{1}{2}\|\boldsymbol{w}\|^2+\gamma\frac{1}{2}\sum_{i=0}^{N-1}\xi_i^2$$
$$\text{s.t.}\ \ (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+b)=y_i-y_i\xi_i,i=0,1,\cdots,N-1 \tag{5.1.4}$$

式中:γ 为正则化参数;ξ_i 为松弛变量或者软边距,其作用是在一定程度上放松对分类样本的要求,允许部分样本点分类错误来换取模型更强的泛化能力。

由拉格朗日乘子法可得式(5.1.4)的对偶问题为

$$\begin{pmatrix}0 & \boldsymbol{e}^{\mathrm{T}}\\ \boldsymbol{e} & \boldsymbol{K}+\gamma^{-1}\boldsymbol{I}\end{pmatrix}\begin{pmatrix}b\\ \boldsymbol{\alpha}\end{pmatrix}=\begin{pmatrix}0\\ \boldsymbol{y}\end{pmatrix} \tag{5.1.5}$$

式中:$\boldsymbol{\alpha}$ 为拉格朗日乘子,$\boldsymbol{\alpha}=(\alpha_0\quad\alpha_1\quad\cdots\quad\alpha_{N-1})^{\mathrm{T}}$;$\boldsymbol{e}=(1\quad 1\quad\cdots\quad 1)^{\mathrm{T}}$;$\boldsymbol{I}$ 为单位矩阵;$\boldsymbol{y}=(y_0\quad y_1\quad\cdots\quad y_{N-1})^{\mathrm{T}}$;$\boldsymbol{K}$ 中包含所有的训练样本,其元素的形式不唯一,由核函数决定。$\boldsymbol{K}_{ij}=\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j$ 为最简单的线性核,在后续的章节中,将介绍其他形式的核函数。

通过求解式(5.1.5)得到的 b 和$\boldsymbol{\alpha}$ 构成 SVM 的超平面。此时,对于待分类样本 $\boldsymbol{x}$ 来说,可以使用下式实现分类:

$$y(\boldsymbol{x})=\mathrm{sign}\Big(\sum_{i=0}^{N-1}\alpha_i\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}+b\Big) \tag{5.1.6}$$

5.1.2 量子支持向量机算法

现在主流的量子支持向量机算法是基于最小二乘形式的量子支持向量机,主要任务是使用 HHL 算法求解式(5.1.5)中的线性方程组,得到描述超平面的量子态$|b,\alpha\rangle$,然后构造待分类样本 $\boldsymbol{x}$ 的量子态,并对其进行分类。本节介绍的量子支持向量机主要分为训练和分类两个部分,并给出复杂度分析。

1. 训练

训练量子支持向量机的线路图如图 5.2 所示。由图可以看到,其训练线路图与

HHL 算法的线路图基本相同。

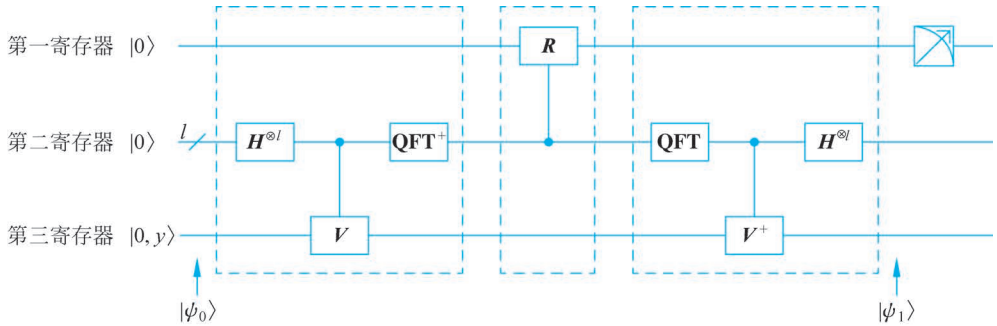


图 5.2 训练量子支持向量机的线路图

为了使用 HHL 算法求解式(5.1.5),需要按照 HHL 算法的要求,将矩阵 $\boldsymbol{F}=\begin{pmatrix}0 & \boldsymbol{e}^{\mathrm{T}}\\ \boldsymbol{e} & \boldsymbol{K}+\gamma^{-1}\boldsymbol{I}\end{pmatrix}$ 和$(0,\boldsymbol{y}^{\mathrm{T}})^{\mathrm{T}}$ 制备好。$\boldsymbol{F}$ 以指数的形式模拟,即 $V=\mathrm{e}^{\mathrm{i}\boldsymbol{F}t}$。向量$(0,\boldsymbol{y}^{\mathrm{T}})^{\mathrm{T}}$ 被制备进量子态中,再加上辅助量子比特组成量子态$|\psi_0\rangle$,$|\psi_0\rangle$为

$$|\psi_0\rangle=|0\rangle|0\rangle^{\otimes l}\sum_{i=0}^{N}k_i|i\rangle \tag{5.1.7}$$

式中:k_i 为向量$(0,\boldsymbol{y}^{\mathrm{T}})^{\mathrm{T}}$ 的第 i 个元素;l 取决于相位估计的精度和成功率。

经过 HHL 算法之后,其输出为$|\psi_1\rangle$,即

$$\begin{aligned}|\psi_1\rangle&=\sum_{i=0}^{N}\left(\sqrt{1-\frac{C_1^2}{\lambda_i^2}}|0\rangle+\frac{C_1}{\lambda_i}|1\rangle\right)\langle u_i|k\rangle|0\rangle^{\otimes l}|u_i\rangle\\&=\sum_{i=0}^{N}\sqrt{1-\frac{C_1^2}{\lambda_i^2}}\langle u_i|k\rangle|0\rangle|0\rangle^{\otimes l}|u_i\rangle+\sum_{i=0}^{N}\frac{C_1}{\lambda_i}\langle u_i|k\rangle|1\rangle|0\rangle^{\otimes l}|u_i\rangle\end{aligned} \tag{5.1.8}$$

式中:$|k\rangle=\sum_{i=0}^{N}k_i|i\rangle$,$|k\rangle$中存储的就是$(0,\boldsymbol{y}^{\mathrm{T}})^{\mathrm{T}}$;$C_1$ 为归一化常数。

最后对存储在第一寄存器中的辅助量子比特进行测量,当量子比特为 1 时,第三寄存器的量子态为

$$|\psi_2\rangle=C_1\sum_{i=0}^{N}\frac{\langle u_i|k\rangle}{\lambda_i}|u_i\rangle \tag{5.1.9}$$

由 HHL 算法可知,除去归一化因子 C_1,式(5.1.9)中的量子态就是式(5.1.5)的解。将$|u_i\rangle$写成标准正交基线性组合的形式,则式(5.1.9)可以写为

$$|b,\alpha\rangle=\frac{1}{\sqrt{C}}\left(b|0\rangle+\sum_{i=0}^{N-1}\alpha_i|i+1\rangle\right) \tag{5.1.10}$$

式中:C 为归一化因子。

2. 分类

上面得到了支持向量机超平面的量子态$|b,\alpha\rangle$,下面对待分类样本 $\boldsymbol{x}$ 进行分类,其对

应的量子态为$|x\rangle$。由式(5.1.6)可知,对样本进行分类,需要构造新的量子态得到$b+\sum_{i=0}^{N-1}\alpha_i\|\boldsymbol{x}_i\|\|\boldsymbol{x}\|\langle x_i\,|\,x\rangle$的符号。

根据超平面量子态$|b,\alpha\rangle$和待分类样本$|x\rangle$分别构造下述量子态:

$$|\tilde{u}\rangle=\frac{1}{\sqrt{N_{\tilde{u}}}}\left(b\,|\,0\rangle\,|\,0\rangle+\sum_{i=0}^{N-1}\alpha_i\|\boldsymbol{x}_i\|i+1\rangle\,|\,x_i\rangle\right)\tag{5.1.11}$$

$$|\tilde{x}\rangle=\frac{1}{\sqrt{N_{\tilde{x}}}}\left(|\,0\rangle\,|\,0\rangle+\sum_{i=0}^{N-1}\|\boldsymbol{x}\|i+1\rangle\,|\,x\rangle\right)\tag{5.1.12}$$

式中:$N_{\tilde{u}}$和$N_{\tilde{x}}$为归一化因子,且有

$$N_{\tilde{u}}=b^2+\sum_{i=0}^{N-1}\alpha_i^2\|\boldsymbol{x}_i\|^2,\quad N_{\tilde{x}}=1+N\,|\,\boldsymbol{x}\,|^2$$

使用哈达玛测试可得$\langle\tilde{u}\,|\tilde{x}\rangle$,即

$$\langle\tilde{u}\,|\tilde{x}\rangle=\frac{1}{\sqrt{N_{\tilde{x}}N_{\tilde{u}}}}\left(b+\sum_{i=0}^{N-1}\alpha_i\|\boldsymbol{x}_i\|\|\boldsymbol{x}\|\langle x_i\,|\,x\rangle\right)\tag{5.1.13}$$

由于对辅助比特进行测量得到$|1\rangle$的概率$P(1)=\frac{1}{2}(1-\langle\tilde{u}\,|\tilde{x}\rangle)$,因此当$P(1)<\frac{1}{2}$时,内积$\langle\tilde{u}\,|\tilde{x}\rangle>0$,可将$\boldsymbol{x}$分类为+1类;否则,分类为−1类。

3. 复杂度分析

在量子支持向量机中,HHL算法是主要组成部分,HHL算法的复杂度为$O(\mathrm{poly}(\log N))$。此外,使用哈达玛测试求内积时需要运行$O\left(\frac{1}{\varepsilon}\right)$次,才能以误差$\varepsilon$得到概率$P(1)$。因此,量子支持向量机的时间复杂度为$O\left((\mathrm{poly}(\log N))\frac{1}{\varepsilon}\right)$。

5.1.3 量子核函数

线性核对于线性不可分问题的分类效果不佳。在经典算法中,解决这个问题的方法是使用核函数。核函数能够将样本从原始空间映射到一个更高维的空间,使得样本在这个高维空间中线性可分。在量子支持向量机中,量子核函数也起着同样的作用。本节首先介绍核函数原理,然后介绍量子核函数。

1. 核函数原理

式(5.1.5)中矩阵$\boldsymbol{K}$的元素为$K_{ij}=\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j$,这是最简单的线性核。事实上,$\boldsymbol{K}$的元素有很多种取法,下式为常见的多项式核:

$$K_{ij}=(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)^d\tag{5.1.14}$$

式中:d为多项式的次数。

此外,还有拉普拉斯核、高斯核等形式,具体的可参见文献[1]。

2. 量子核函数

本节介绍多项式核的量子形式。令$|\varphi(x_i)\rangle=|x_i\rangle\otimes\cdots\otimes|x_i\rangle$,则量子多项式核可

以表示为

$$\begin{aligned} K_{ij} &= \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_j) \\ &= \langle \varphi(x_i) \mid \varphi(x_j) \rangle = \langle x_i \mid \otimes \cdots \otimes \langle x_i \mid x_j \rangle \otimes \cdots \otimes \mid x_j \rangle \\ &= \langle x_i \mid x_j \rangle \otimes \cdots \otimes \langle x_i \mid x_j \rangle = \langle x_i \mid x_j \rangle^d \end{aligned} \tag{5.1.15}$$

利用式(5.1.15)的技巧可以构造任意多项式核。

5.1.4 实现

本实验利用量子支持向量机来对手写数字 6 和 9 进行分类。训练集为一个 6 和一个 9，它们对应的二维特征向量分别为 $\boldsymbol{x}_1=(0.987\ 0.159)^{\mathrm{T}}$ 和 $\boldsymbol{x}_2=(0.354\ 0.935)^{\mathrm{T}}$，满足归一化，标签分别为 $y_1=+1$ 和 $y_2=-1$。测试集 6 和 9 的特征向量分别为 $\boldsymbol{x}_3=(0.987\ 0.160)^{\mathrm{T}}$ 和 $\boldsymbol{x}_4=(0.352\ 0.936)^{\mathrm{T}}$。

在本实验中，考虑偏置 $b=0$ 的情况，令 $\gamma=2$，则式(5.1.5)退化为

$$(\boldsymbol{K}+2^{-1}\boldsymbol{I})\,\boldsymbol{\alpha}=\boldsymbol{y} \tag{5.1.16}$$

由于训练集为 $\boldsymbol{x}_1=(0.987\quad 0.159)^{\mathrm{T}}$ 和 $\boldsymbol{x}_2=(0.354\quad 0.935)^{\mathrm{T}}$，则 $\boldsymbol{K}=\begin{pmatrix}1 & 0.498\\ 0.498 & 1\end{pmatrix}$，因此 $\boldsymbol{F}=(\boldsymbol{K}+2^{-1}\boldsymbol{I})=\begin{pmatrix}1.5 & 0.498\\ 0.498 & 1.5\end{pmatrix}$，约等于 $\begin{pmatrix}1.5 & 0.5\\ 0.5 & 1.5\end{pmatrix}$；对应 $\boldsymbol{y}=\begin{pmatrix}1\\ -1\end{pmatrix}$，归一化之后 $\tilde{\boldsymbol{y}}=\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix}$。如图 5.3 所示，U_gate 实现了 HHL 算法以及训练样本的构造(式(5.1.11))，X_gate 用于构造待分类样本(式(5.1.12))。U_gate 包括三部分：一是 HHL 算法初始输入态，即 $|\tilde{y}\rangle=\frac{|0\rangle-|1\rangle}{\sqrt{2}}$；二是利用 HHL 算法求得 $(\boldsymbol{K}+2^{-1}\boldsymbol{I})\boldsymbol{\alpha}=\boldsymbol{y}$ 的解，在此过程中要模拟 $\mathrm{e}^{\mathrm{i}\frac{\boldsymbol{F}}{4}t}\mathrm{e}^{\mathrm{i}\frac{\boldsymbol{F}}{2}t}$，过程和 HHL 算法一样，具体过程可参考 HHL 算法，这里不再赘述；三是将训练集数据制备到量子线路中，实现式(5.1.11)。

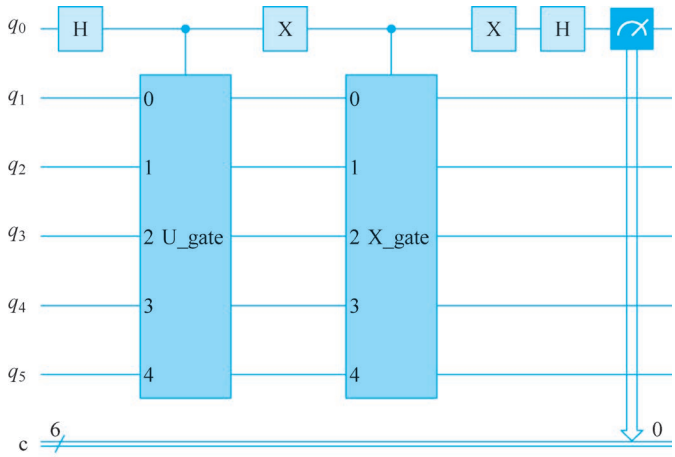


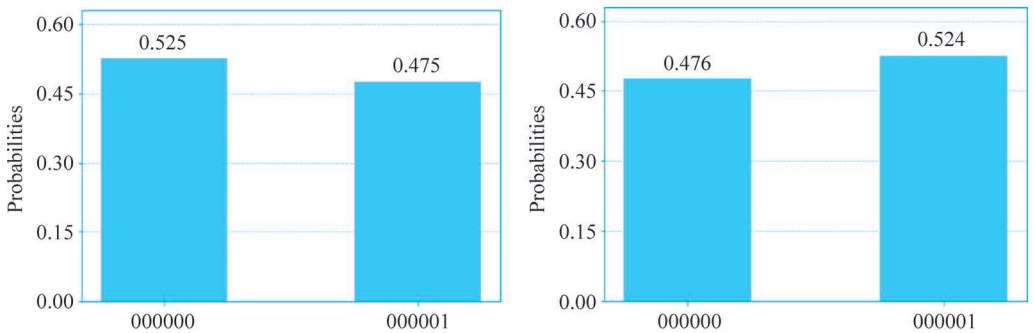
图 5.3 量子支持向量机线路图

值得注意的是，由于 HHL 算法的最后没有进行测量，得到的量子态为 $|1\rangle|b,\alpha\rangle+|0\rangle|G_1\rangle$，其中 $|G_1\rangle$ 为式(5.1.8)中的量子态 $\sum_{i=0}^{N}\sqrt{1-\frac{C_1^2}{\lambda_i^2}}\langle u_i|y\rangle|u_i\rangle$，记为垃圾态。因此，式(5.1.11)变为 $|1\rangle|\tilde{u}\rangle+|0\rangle|G_2\rangle$，$|G_2\rangle$ 为垃圾态，此时量子态 $|\tilde{u}\rangle$ 和 $|G_2\rangle$ 都在量子比特 q_4 和 q_5 中，$|1\rangle$ 和 $|0\rangle$ 都在量子比特 q_1 中。由于

$$(\langle 1|\langle\tilde{u}|+\langle 0|\langle G_2|)(|1\rangle|\tilde{x}\rangle)=\langle\tilde{u}|\tilde{x}\rangle \tag{5.1.17}$$

因此，为了得到 $\langle\tilde{u}|\tilde{x}\rangle$，在量子比特 q_4 和 q_5 上使用 X_gate 制备量子态 $|\tilde{x}\rangle$ 时，需要将 q_1 的量子态制备成 $|1\rangle$。

当输入待测数据为 6($\boldsymbol{x}_3=(0.987\quad 0.160)^{\mathrm{T}}$)时，由图 5.4(a)可知，测量 q_0 得到 $|1\rangle$ 的概率为 $0.475<\frac{1}{2}$，分类结果为+1，即分类为“6”，因此分类正确；当输入待测数据为 9($\boldsymbol{x}_4=(0.352\quad 0.936)^{\mathrm{T}}$)时，由 5.4(b)可知，测量 q_0 得到 $|1\rangle$ 的概率为 $0.524>\frac{1}{2}$，分类结果为−1，即分类为“9”，因此分类也是正确的。



(a) 测试数据为6　　(b) 测试数据为9

图 5.4　量子支持向量实验结果

量子支持向量机的代码如下：

```
%matplotlib inline
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import Aer
from qiskit import IBMQ
from math import pi
from qiskit.tools.visualization import plot_histogram

circuit = QuantumCircuit(6,6)

#自定义 U_gate
def U_gate():
    circuit = QuantumCircuit(5)
    #量子态制备
    circuit.x(3)
```

```
    circuit.h(3)
    # hhl 算法
    circuit.h(1)
    circuit.h(2)
    circuit.cu3( - pi/2, - pi/2,pi/2,1,3)
    circuit.u1(3 * pi/4,1)
    circuit.cx(2, 3)
    circuit.swap(1, 2)
    circuit.h(2) # iQFT
    circuit.cu1( - pi/2, 1, 2)
    circuit.h(1)
    circuit.swap(1,2)
    circuit.cu3(pi/8, 0, 0, 1, 0)
    circuit.cu3(pi/16, 0, 0, 2, 0)
    circuit.swap(1,2)
    circuit.h(1)
    circuit.cu1(pi/2, 1, 2)
    circuit.h(2)
    circuit.swap(1, 2)
    circuit.cx(2, 3)
    circuit.u1( - 3 * pi/4,1)
    circuit.cu3( - pi/2,pi/2, - pi/2,1,3)
    circuit.h(2)
    circuit.h(1)
    # 添加训练数据
    circuit.cry(0.32,3,4)
    circuit.x(3)
    circuit.cry(2.82,3,4)
    circuit.x(3)
    circuit = circuit.to_gate()
    circuit.name = "U_gate"
    c_U = circuit.control()
    return c_U

circuit.h(0)
# 使用 U_gate
circuit.append(U_gate(),[0] + [i + 1 for i in range(5)])

# 自定义 X_gate
def X_gate():
    circuit = QuantumCircuit(5)
    circuit.h(3)
    circuit.x(0)
    circuit.ry(2.422,4)
    circuit = circuit.to_gate()
    circuit.name = "X_gate"
    c_U = circuit.control()
    return c_U

circuit.x(0)
```

```
66.  #使用 X_gate
67.  circuit.append(X_gate(),[0]+[i+1 for i in range(5)])
68.  circuit.x(0)
69.  circuit.h(0)
70.
71.  #测量
72.  circuit.measure(0,0)
73.
74.  #绘制线路图
75.  circuit.draw(output='mpl')
76.  backend=Aer.get_backend('qasm_simulator')
77.  job_sim=execute(circuit,backend,shots=20000)
78.  sim_result=job_sim.result()
79.
80.  #绘制结果图
81.  measurement_result=sim_result.get_counts(circuit)
82.  plot_histogram(measurement_result)
```

5.2 量子 K 近邻

K 近邻是一种简单有效的有监督机器学习算法。K 近邻从一个给定训练集中找到与待分类样本最邻近的 K 个样本，这 K 个样本中的多数属于哪个类，则待分类样本就划分到哪个类中。2020 年，Basheer 等给出了量子 K 近邻算法，原理与经典 K 近邻相同，不同之处主要是先利用量子算法计算样本之间的距离，再利用最大值搜索算法找到距离最近的 K 个样本。

5.2.1 K 近邻基本原理

图 5.5 给出 K 近邻原理示意图。训练样本有两个类别，分别用三角形和正方形表示。对于新输入的待分类样本(用圆形表示)，计算所有训练样本与待分类样本的距离，找到与待分类样本最近的 K 个邻居。当 $K=3$ 时，邻居中包括 2 个三角形和 1 个正方形，因此分类为三角形；当 $K=5$ 时，邻居中包括 2 个三角形和 3 个正方形，因此分类为正方形。虽然 K 近邻是有监督学习方法，但是它不需要训练过程，直接根据待分类样本与所有训练样本的距离确定分类结果。

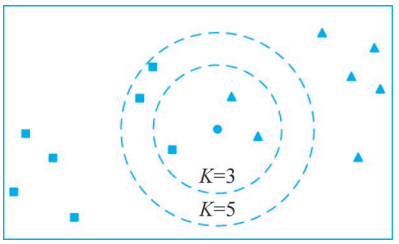


图 5.5 K 近邻原理示意图

在 K 近邻算法中，当训练集、K 值以及距离度量确定后，任何一个新输入的样本都会对应一个唯一的类别。其中两个样本之间的距离计算是量子 K 近邻中最重要的部分，此外使用量子算法寻找 K 个最近邻样本也是一项重要内容。

5.2.2 量子距离

在特征空间中，两个样本点的距离是两个样本相似程度的度量。在经典计算方法中，常用的相似度计算方法包括欧几里得距离、汉明距离、向量内积以及余弦距离等形式。在量子计算中，样本信息存储在量子态中。两个量子态之间的距离度量方式有迹距离和保真度两种方式。迹距离并不常用，这里不再介绍。保真度是衡量两个量子态相似程度的常用方法，保真度越大，两个量子态越相似。

【定义 5.2.1】 对于两个量子态 $|u\rangle$ 和 $|v\rangle$ 来说，其保真度定义为

$$F(|u\rangle,|v\rangle)=|\langle u|v\rangle|^2 \tag{5.2.1}$$

即保真度是 $|u\rangle$ 和 $|v\rangle$ 内积的模的平方，因此交换测试是计算两个量子态保真度的有效方法。

5.2.3 量子最大值搜索

上一节描述了两个量子态之间的距离度量方法，如何使用量子算法找到和待分类样本最近的 K 个样本呢？由于保真度越大，两个量子态越相似，因此需要找到距离中最大的 K 个值。本节给出搜索最大值的量子算法。

令 $T=\{T_0,T_1,\cdots,T_{N-1}\}$ 为含有 N 个元素的无序数据库，元素的下标为 $\{0,1,\cdots,N-1\}$。最大值搜索算法的目的就是找到下标 $i(i\in\{0,1,\cdots,N-1\})$，使得 $T_i=\max\{T_0,T_1,\cdots,T_{N-1}\}$。具体算法如下：

第一步：从 $\{0,1,\cdots,N-1\}$ 中随机选取一个下标 z_1。

第二步：重复下列步骤 $O(\sqrt{N})$ 次：

(1) 定义函数：

$$f_{z_1}(z_2)=\begin{cases}1, & T_{z_2}>T_{z_1}\\ 0, & \text{其他}\end{cases} \tag{5.2.2}$$

(2) 应用 Grover 搜索算法找到满足式 $f_{z_1}(z_2)=1$ 的下标 z_2。

(3) 如果测量能够得到 z_2，则用 z_2 代替 z_1；否则，不做任何改变。

第三步：最终的 z_1 就是使得 T_i 为最大值的下标 i。

在该算法中，第二步要重复 $O(\sqrt{N})$ 次，而第二步(2)中使用的 Grover 算法的复杂度为 $O(\sqrt{N})$，因此该算法总的复杂度为 $O(N)$。

5.2.4 量子 K 近邻算法

量子 K 近邻算法需要计算待分类样本与所有训练样本之间的保真度，并找到最近邻的 K 个训练样本。本节给出量子 K 近邻的具体算法。

假设 N 个训练样本组成的数据集如式(5.1.1)所示。令$\{|x_i\rangle:i\in\{0,1,\cdots,N-1\}\}$是训练样本所对应的量子态，$|x\rangle$是待分类的样本。量子 K 近邻算法先使用K 次最大值搜索算法找到和待分类样本最近邻的 K 个样本$\{|x_{i_0}\rangle,|x_{i_1}\rangle,\cdots,|x_{i_{K-1}}\rangle\}$，再采用经典的多数表决规则对$|x\rangle$进行分类。

经典的多数表决规则这里不再单独介绍。下面主要介绍量子实现过程，也就是找到和待分类样本最近邻的 K 个训练样本。为此首先计算待分类样本$|x\rangle$和所有训练样本$\{|x_i\rangle:i\in\{0,1,\cdots,N-1\}\}$之间的保真度，然后找到和待分类样本最近邻的 K 个训练样本。

1. 计算保真度

在上述数据集中每个样本有 M 个特征(令 $M=2^m$)。如图 5.6 所示的过程能够将训练样本$|x_i\rangle$与待分类样本$|x\rangle$之间的保真度存储在基态中。

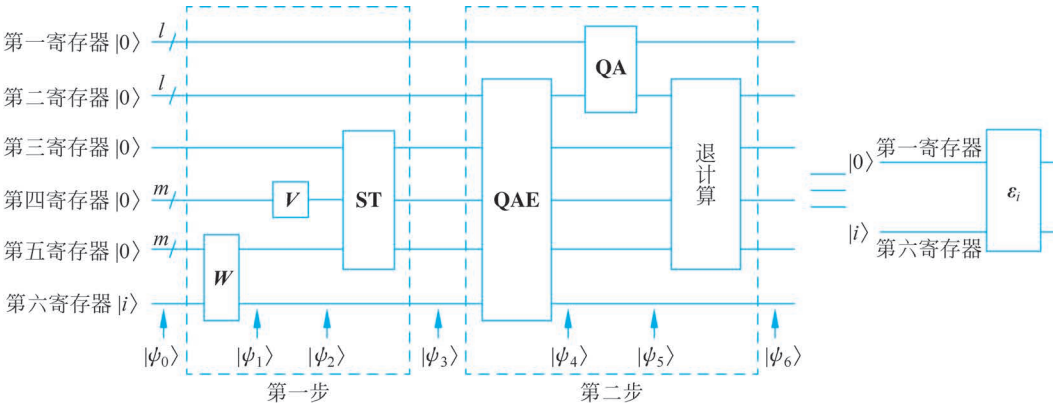


图 5.6　将保真度存储在基态中的量子线路图

主要分为两步：

第一步：使用交换测试将保真度相关信息存储到振幅中。

首先制备初始态

$$|\psi_0\rangle=|0\rangle^{\otimes l}\,|0\rangle^{\otimes l}\,|0\rangle\,|0\rangle^{\otimes m}\,|0\rangle^{\otimes m}\,|i\rangle \tag{5.2.3}$$

使用黑箱 $\boldsymbol{W}$ 作用于第五寄存器和第六寄存器得到量子态

$$|\psi_1\rangle=|0\rangle^{\otimes l}\,|0\rangle^{\otimes l}\,|0\rangle\,|0\rangle^{\otimes m}\,|x_i\rangle\,|i\rangle \tag{5.2.4}$$

使用黑箱 $\boldsymbol{V}$ 作用于第四寄存器得到量子态

$$|\psi_2\rangle=|0\rangle^{\otimes l}\,|0\rangle^{\otimes l}\,|0\rangle\,|x\rangle\,|x_i\rangle\,|i\rangle \tag{5.2.5}$$

使用交换测试 **ST** 作用于第三寄存器、第四寄存器和第五寄存器可得

$$\begin{aligned}|\psi_3\rangle&=|0\rangle^{\otimes l}\,|0\rangle^{\otimes l}\,\frac{1}{2}[|0\rangle(|x\rangle\,|x_i\rangle+|x_i\rangle\,|x\rangle)+|1\rangle(|x\rangle\,|x_i\rangle-|x_i\rangle\,|x\rangle)]\,|i\rangle\\&=|0\rangle^{\otimes l}\,|0\rangle^{\otimes l}\,|\varphi_i\rangle\,|i\rangle\end{aligned} \tag{5.2.6}$$

式中

$$|\varphi_i\rangle=\frac{1}{2}[|0\rangle(|x\rangle\,|x_i\rangle+|x_i\rangle\,|x\rangle)+|1\rangle(|x\rangle\,|x_i\rangle-|x_i\rangle\,|x\rangle)]$$

令

$$|\varphi_{i0}\rangle=|x\rangle|x_i\rangle+|x_i\rangle|x\rangle \tag{5.2.7}$$

$$|\varphi_{i1}\rangle=|x\rangle|x_i\rangle-|x_i\rangle|x\rangle \tag{5.2.8}$$

则$|\varphi_i\rangle$可以表示成量子振幅估计算法中式(3.5.1)的形式，即

$$|\varphi_i\rangle=\sqrt{\frac{(1+F_i)}{2}}|0\rangle|\varphi_{i0}\rangle+\sqrt{\frac{(1-F_i)}{2}}|1\rangle|\varphi_{i1}\rangle \tag{5.2.9}$$

式中：$F_i=F(x,x_i)$。

这时保真度相关信息存储在振幅 $\cos(\pi\theta_i)\sqrt{\frac{1+F_i}{2}}$ 和 $\sin(\pi\theta_i)\sqrt{\frac{1-F_i}{2}}$ 中，接下来要把存储在振幅中的信息转移到基态中。

第二步：利用量子振幅估计算法将保真度信息存储到基态中。

令 $\boldsymbol{U}$ 为第一步算子的组合，构造振幅放大算子

$$\boldsymbol{G}_i=\boldsymbol{U}\boldsymbol{S}_i\boldsymbol{U}^{-1}\boldsymbol{Z} \tag{5.2.10}$$

式中：$\boldsymbol{S}_i$ 只改变初始态$|0\rangle|0\rangle^{\otimes m}|0\rangle^{\otimes m}$ 的符号；$\boldsymbol{Z}$ 作用于第三寄存器用于改变式(5.2.9)中$|1\rangle$的符号。

执行量子振幅估计，将存储在振幅中的 θ_i 转移到第二寄存器的基态中，即

$$\begin{aligned}|\psi_4\rangle&=|0\rangle^{\otimes l}\frac{1}{\sqrt{2}}(\mathrm{e}^{-\mathrm{i}\pi\theta_i}|2^m\theta_i\rangle|\delta_{i+}\rangle+\mathrm{e}^{\mathrm{i}\pi\theta_i}|2^m(1-\theta_i)\rangle|\delta_{i-}\rangle)|i\rangle\\&=|0\rangle^{\otimes l}|\delta_i\rangle|i\rangle\end{aligned} \tag{5.2.11}$$

式中：$|\delta_i\rangle=\frac{1}{\sqrt{2}}(\mathrm{e}^{-\mathrm{i}\pi\theta_i}|2^m\theta_i\rangle|\delta_{i+}\rangle+\mathrm{e}^{\mathrm{i}\pi\theta_i}|2^m(1-\theta_i)\rangle|\delta_{i-}\rangle)$，包含第二寄存器、第三寄存器、第四寄存器和第五寄存器，$-\mathrm{e}^{\pm 2\mathrm{i}\theta_i}$ 为 G_i 的特征值，$|\delta_{i\pm}\rangle$为对应的特征向量。

三角函数等一些基本的函数可以用量子线路来实现(见附录 B)，因此总可以实现一个算子 **QA**，能使存储在第二寄存器中的函数 $F_i=1-2\sin^2(\pi\theta_i)$存储在第一寄存器中，第一寄存器存储的量子态为$|F_i\rangle=|1-2\sin^2(\pi\theta_i)\rangle$。因此经过算子 **QA** 之后式(5.2.11)演化为

$$|\psi_5\rangle=|F_i\rangle|\delta_i\rangle|i\rangle \tag{5.2.12}$$

对第二寄存器、第三寄存器、第四寄存器和第五寄存器实施退计算，可得

$$|\psi_6\rangle=|F_i\rangle|i\rangle \tag{5.2.13}$$

至此，保真度被存储于基态中。也就是说，如果记上述量子线路为$\boldsymbol{\varepsilon}_i$，经过$\boldsymbol{\varepsilon}_i$ 之后，第一寄存器和第六寄存器的量子态由$|0\rangle^{\otimes l}|i\rangle$演化为$|F_i\rangle|i\rangle$，第二寄存器至第五寄存器为辅助寄存器，输入和输出全为$|0\rangle$。因此，在后续的描述的中记$\boldsymbol{\varepsilon}_i$ 的输出为$|F_i\rangle|i\rangle$。图 5.6 中，$\boldsymbol{\varepsilon}_i$ 的输入和输出只画出了第一寄存器和第六寄存器，其余辅助寄存器没有再体现，可以理解为$\boldsymbol{\varepsilon}_i$ 内部的寄存器。

2. 寻找和待分类样本最近邻的 K 个样本

下面介绍如何得到距离最近的 K 个样本，也就是得到 K 个最大的保真度。要找 K

个最大值，需要先随机选取 K 个下标，记为 $A=\{i_0,i_1,\cdots,i_{K-1}\}$，对于 A 中的每个元素都执行最大值搜索算法中的第二步。使用黑箱算子 $\boldsymbol{O}_{i',A}$，使其能够满足

$$\boldsymbol{O}_{i',A}\,|0\rangle\,|i\rangle=|f_{i',A}(i)\rangle\,|i\rangle \tag{5.2.14}$$

式中

$$f_{i',A}(i)=\begin{cases}1, & F_i>F_{i'},i\notin A\\ 0, & \text{其他}\end{cases}$$

式(5.2.14)表示，对于 $i'\in A$，找到一个 $i\notin A$，使得当 $F_i>F_{i'}$ 成立时，有 $f_{i',A}(i)=1$。

下面给出黑箱 $\boldsymbol{O}_{i',A}$ 的实现方法，量子线路图如图 5.7 所示。

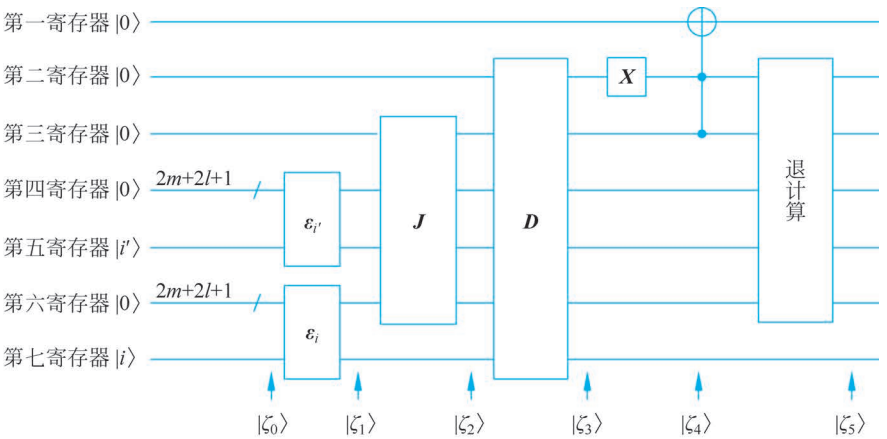


图 5.7　$\boldsymbol{O}_{i',A}$ 的量子线路图

初始输入为第四寄存器至第七寄存器的 $|\zeta_0\rangle=|0\rangle|i'\rangle|0\rangle|i\rangle$，利用 $\boldsymbol{\varepsilon}_i$ 和 $\boldsymbol{\varepsilon}_{i'}$ 可以将其演化为

$$|\zeta_1\rangle=|F_{i'}\rangle\,|i'\rangle\,|F_i\rangle\,|i\rangle \tag{5.2.15}$$

使用比较门($\boldsymbol{J}$)比较第四寄存器和第六寄存器，并将结果存储在第三寄存器中，式(5.2.15)演化为

$$|\zeta_2\rangle=|g(i)\rangle\,|F_{i'}\rangle\,|i'\rangle\,|F_i\rangle\,|i\rangle \tag{5.2.16}$$

式中

$$g(i)=\begin{cases}1, & F_i>F_{i'}\\ 0, & F_i\leqslant F_{i'}\end{cases}$$

增加一个辅助量子比特用来判断下标 i 是否属于集合 A，这里使用不等门($\boldsymbol{D}$)来实现。若 $\boldsymbol{D}^{(i_l)}$ 用于标记下标 i_l 是否属于 A，则 $\boldsymbol{D}=\boldsymbol{D}^{(i_0)}\cdots\boldsymbol{D}^{(i_{K-1})}$。将 $\boldsymbol{D}$ 作用于式(5.2.16)中的第七寄存器并将结果存储于第二寄存器时，可得

$$|\zeta_3\rangle=|\chi_A(i)\rangle\,|g(i)\rangle\,|F_{i'}\rangle\,|i'\rangle\,|F_i\rangle\,|i\rangle \tag{5.2.17}$$

式中

$$\chi_A(i)=\begin{cases}1, & i\in A\\ 0, & i\notin A\end{cases}$$

也就是说，$\boldsymbol{D}^{(i_0)}\cdots\boldsymbol{D}^{(i_{K-1})}$ 标记了已经属于 A 的 K 个下标，以避免前 K 个保真度有重复。

最终要构建一个黑箱使得 $g(i)=1$ 且 $\chi_A(i)=0$，因此使用一个 $\boldsymbol{X}$ 门作用于第二寄存器，再使用 Toffoli 门作用于第一寄存器至第三寄存器，可得

$$|\zeta_4\rangle=|f_{i',A}(i)\rangle|\chi_A(i)\rangle|g(i)\rangle|F_{i'}\rangle|i'\rangle|F_i\rangle|i\rangle \tag{5.2.18}$$

此时，如果 $f_{i',A}(i)=1$，则满足条件；否则，不满足条件。

对第二寄存器至第六寄存器执行退计算可得

$$|\zeta_5\rangle=|f_{i',A}(i)\rangle|i\rangle \tag{5.2.19}$$

令 $\boldsymbol{O}_{i',A}$ 表示上述所有操作的集合，则

$$\boldsymbol{O}_{i',A}|0\rangle|i\rangle=|f_{i',A}(i)\rangle|i\rangle \tag{5.2.20}$$

5.2.5 实现

假设 $|x_1\rangle=\frac{\sqrt{3}}{2}|0\rangle+\frac{1}{2}|1\rangle$ 和 $|x_2\rangle=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$ 分别是属于类别 1 和类别 2 的数据，本实验旨在计算一个待分类数据 $|x\rangle=\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$ 与 $|x_1\rangle$ 和 $|x_2\rangle$ 的距离，并比较出待分类数据与哪个类别的数据更接近，也就是与哪个类别的保真度更大。

考虑到量子线路的复杂性，本实验分成两部分：第一部分将保真度信息存储在振幅中，并通过测量得到该振幅；第二部分将上述振幅存储到基态中，并比较大小，进而得到待分类数据距离哪个类别更近。

下面先介绍第一部分实验，量子线路图如图 5.8 所示。

第一步：使用交换测试将保真度信息存储到振幅中。

用两个不同的 $\boldsymbol{W}$ 门分别在 q_7 和 q_{15} 上制备训练数据 $|x_1\rangle=\frac{\sqrt{3}}{2}|0\rangle+\frac{1}{2}|1\rangle$ 和 $|x_2\rangle=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle$，用 $\boldsymbol{U}$ 门分别在 q_6 和 q_{14} 制备一个待分类数据 $|x\rangle=\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle$，并执行一次交换测试得到 $|x\rangle$ 与 $|x_1\rangle$ 的保真度 $F_1=2\sin^2(\pi\theta_1)-1=\cos(2\pi\theta_1)$，以及 $|x\rangle$ 与 $|x_2\rangle$ 的保真度 $F_2=2\sin^2(\pi\theta_2)-1=\cos(2\pi\theta_2)$。

第二步：添加辅助量子比特 $q_1q_2q_3q_4$ 和 $q_9q_{10}q_{11}q_{12}$，执行量子振幅估计算法，将和保真度相关的 $2^4\theta_1$ 和 $2^4\theta_2$ 存储在基态 $q_1q_2q_3q_4$ 和 $q_9q_{10}q_{11}q_{12}$ 上。

第三步：添加辅助量子比特 q_0 和 q_8，使用受控 $\boldsymbol{R}_y\left(\frac{8\pi}{8}\right)$、$\boldsymbol{R}_y\left(\frac{4\pi}{8}\right)$、$\boldsymbol{R}_y\left(\frac{2\pi}{8}\right)$、$\boldsymbol{R}_y\left(\frac{\pi}{8}\right)$ 将保真度 F_1 和 F_2 存储到量子比特 q_0 和 q_8 上。

最后分别对 q_0 和 q_8 进行测量，得到 $|0\rangle$ 的概率，再开根号即为保真度。测量结果如图 5.9 所示。由图 5.9(a)可以看出，对 q_0 测量得到 0 的概率为 0.854，因此保真度 $F_1=\sqrt{0.854}=0.9236$。由图 5.9(b)可以看出，对 q_8 测量得到 0 的概率为 0，因此保真度 $F_2=\sqrt{0}=0$。

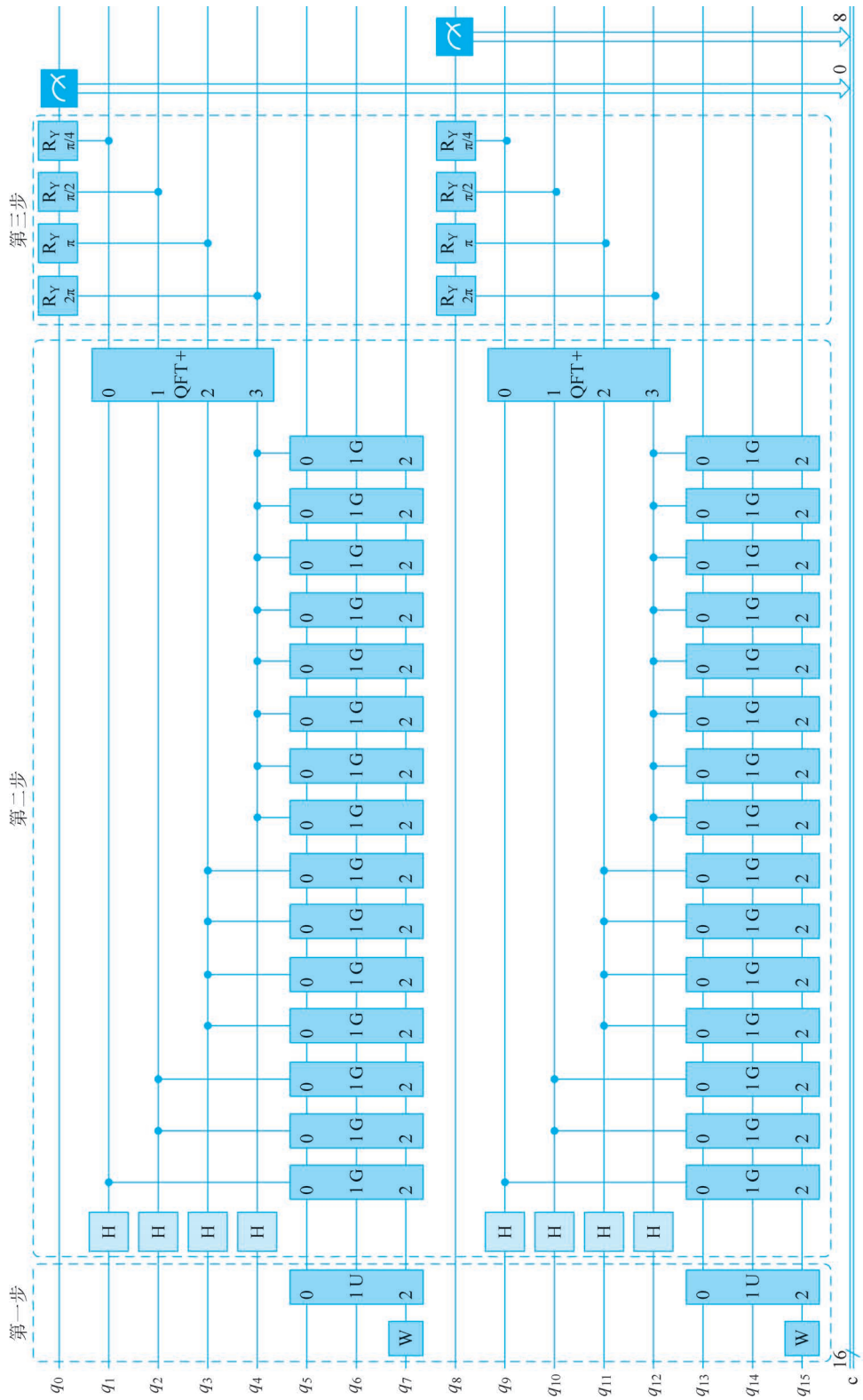
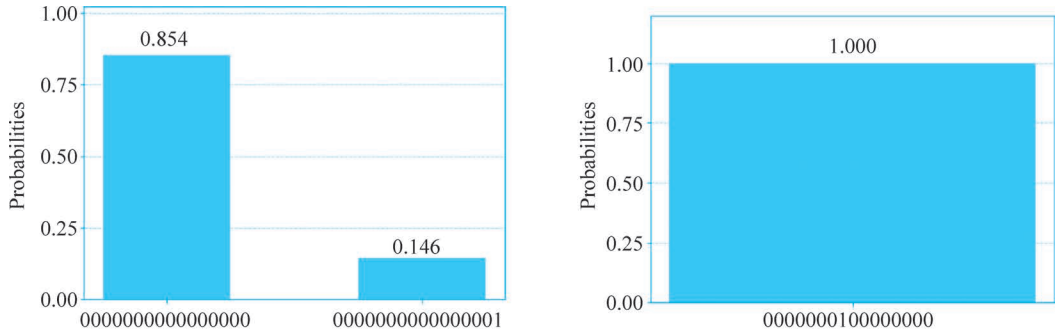


图 5.8　计算保真度的量子线路图



(a) 量子比特q_0的测量结果　　(b) 量子比特q_8的测量结果

图 5.9　量子比特 q_0 和 q_8 的测量结果

第二部分实验完成了分类，其量子线路图如图 5.10 所示。

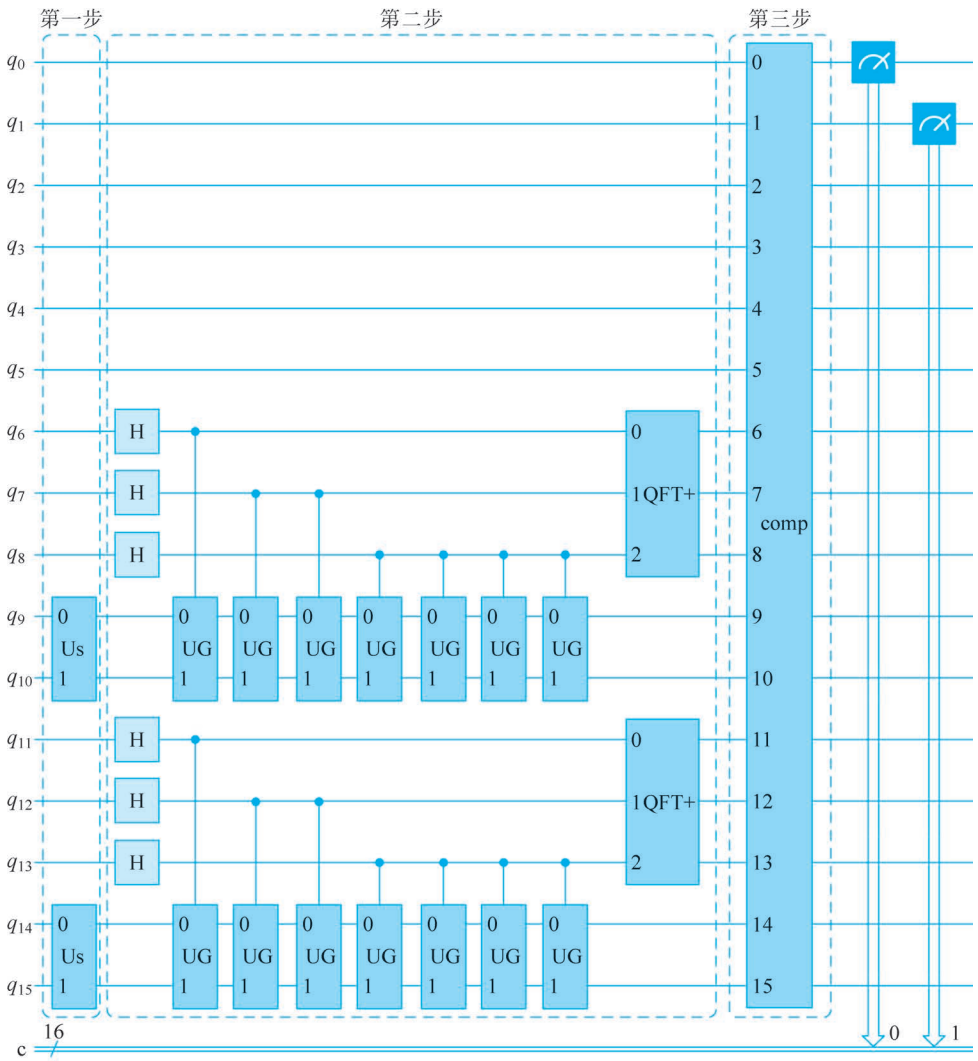


图 5.10　分类量子线路图

在第一部分的实验中，将保真度信息存储到 q_0 和 q_8 的振幅中，为了比较保真度的大小，利用量子振幅估计将存储在振幅中的保真度 $F=2\sin^2(\pi\theta)-1=\cos(2\pi\theta)$ 转移到辅助量子比特 $q_6q_7q_8$ 和 $q_{11}q_{12}q_{13}$ 中，再利用量子比较器比较两个保真度的大小。

第一步：在 q_{10} 和 q_{15} 上制备与第一部分中 q_0 和 q_8 相同的振幅。

第二步：执行量子振幅估计将振幅存储到 $q_6q_7q_8$ 和 $q_{11}q_{12}q_{13}$ 上。

第三步：使用比较门比较 $q_6q_7q_8$ 和 $q_{11}q_{12}q_{13}$ 的大小关系。两组的三个量子比特代表的十进制数分别为 y_1 和 y_2，则保真度分别为 $F_1=\frac{1}{2}\left(1-\cos\frac{\pi y_1}{2^{n-1}}\right)$ 和 $F_2=\frac{1}{2}\left(1-\cos\frac{\pi y_2}{2^{n-1}}\right)$。如果 $0\leqslant\frac{\pi y}{2^{n-1}}\leqslant\pi$，其中 $y\in\{y_1,y_2\}$，则可以根据 y_1 和 y_2 判断 F_1 和 F_2 的大小，即当 $y_1>y_2$ 时，有 $F_1>F_2$。由图 5.11 可以看出，$q_0q_1=|10\rangle$，即 $y_1>y_2$，因此 $F_1>F_2$。也就是将 $|x\rangle$ 划分到样本 $|x_1\rangle$ 所在的类中。

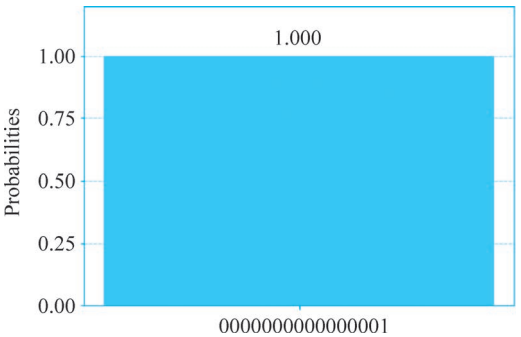


图 5.11　分类测量结果

量子 K 近邻算法的两部分的代码如下所示：

第一部分代码：

```
%matplotlib inline
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import BasicAer
from qiskit import IBMQ
from qiskit import Aer
from math import pi
from qiskit.tools.visualization import plot_histogram

circuit = QuantumCircuit(16,16)

#定义 W 门：制备训练集
def W_gate(n):
    circuit = QuantumCircuit(1)
    circuit.ry(n,0)
    circuit = circuit.to_gate()
    circuit.name = "W"
```

```
    return circuit

#定义 U 门: 先制备测试数据,再对训练数据和测试数据做 swap - test 得到两者的保真度
def U_gate():
    circuit = QuantumCircuit(3)
    circuit.h(2)
    circuit.h(0)
    circuit.cswap(0,1,2)
    circuit.h(0)
    circuit = circuit.to_gate()
    circuit.name = "U"
    return circuit

#定义 G 门中的子门
def xcz():
    circuit = QuantumCircuit(2)
    circuit.x(0)
    circuit.x(1)
    circuit.cz(1,0)
    circuit.x(1)
    circuit.x(0)
    circuit = circuit.to_gate()
    c_U = circuit.control()
    return c_U

#定义 G 门: 量子相位估计
def G_gate(n):
    circuit = QuantumCircuit(3)
    circuit.z(0)
    circuit.append(U_gate().inverse(),[i for i in range(3)])
    circuit.append(W_gate(n).inverse(),[i + 2 for i in range(1)])
    circuit.x(2)
    circuit.append(xcz(),[2] + [i for i in range(2)])
    circuit.x(2)
    circuit.append(W_gate(n),[i + 2 for i in range(1)])
    circuit.append(U_gate(),[i for i in range(3)])
    circuit = circuit.to_gate()
    circuit.name = "G"
    c_U = circuit.control()
    return c_U

#定义量子傅里叶逆变换
def qft_dagger(n):
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n - qubit - 1)
    for j in range(n):
        for m in range(j):
            qc.cp( - pi/float(2 ** (j - m)), m, j)
        qc.h(j)
    qc.name = "QFT + "
```

```
    return qc

#第一步
circuit.append(W_gate(pi/3),[i+7 for i in range(1)])
circuit.append(U_gate(),[i+5 for i in range(3)])
circuit.barrier(0,1,2,3,4,5,6,7)

#第二步
for i in range(4):
    circuit.h(i+1)
for i in range(4):
    for j in range(2**i):
        circuit.append(G_gate(pi/3),[i+1]+[m+5 for m in range(3)])
circuit.append(qft_dagger(4),[i+1 for i in range(4)])

#第三步:计算保真度到辅助量子比特
circuit.cry(8*pi/4,4,0)
circuit.cry(4*pi/4,3,0)
circuit.cry(2*pi/4,2,0)
circuit.cry(pi/4,1,0)

#测量
circuit.measure(0,0)

#在线路的下半部分重复上述操作
circuit.append(W_gate(-pi/2),[i+15 for i in range(1)])
circuit.append(U_gate(),[i+13 for i in range(3)])
circuit.barrier(8,9,10,11,12,13,14,15)
for i in range(4):
    circuit.h(i+9)
for i in range(4):
    for j in range(2**i):
        circuit.append(G_gate(-pi/2),[i+9]+[m+13 for m in range(3)])
circuit.append(qft_dagger(4),[i+9 for i in range(4)])
circuit.cry(8*pi/4,12,8)
circuit.cry(4*pi/4,11,8)
circuit.cry(2*pi/4,10,8)
circuit.cry(pi/4,9,8)
circuit.measure(8,8)

#绘制线路图
circuit.draw(output='mpl', plot_barriers=False, fold=-1)
backend = Aer.get_backend('qasm_simulator')
job_sim = execute(circuit, backend, shots=8192)
sim_result = job_sim.result()

#绘制结果图
measurement_result = sim_result.get_counts(circuit)
print(measurement_result)
plot_histogram(measurement_result)
```

第二部分代码：

```
%matplotlib inline
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import Aer
from math import pi
from qiskit.tools.visualization import plot_histogram
from comp import comp

circuit = QuantumCircuit(16,16)

#定义 Us 门
def Us():
    circuit = QuantumCircuit(2)
    circuit.h(0)
    circuit.cry(7 * pi/4,0,1)
    circuit.x(0)
    circuit.cry( - 7 * pi/4,0,1)
    circuit.x(0)
    circuit.h(0)
    circuit = circuit.to_gate()
    circuit.name = "Us"
    return circuit

#定义 Us1 门
def Us1():
    circuit = QuantumCircuit(2)
    circuit.h(0)
    circuit.cry(pi,0,1)
    circuit.x(0)
    circuit.cry( - pi,0,1)
    circuit.x(0)
    circuit.h(0)
    circuit = circuit.to_gate()
    circuit.name = "Us1"
    return circuit

#定义 UG 门
def UG():
    circuit = QuantumCircuit(2)
    circuit.z(0)
    circuit.append(Us().inverse(),[i for i in range(2)])
    circuit.x(1)
    circuit.x(0)
    circuit.cz(0,1)
    circuit.x(0)
    circuit.x(1)
    circuit.append(Us(),[i for i in range(2)])
    circuit = circuit.to_gate()
    circuit.name = "UG"
```

```
    c_U = circuit.control()
    return c_U

#定义 UG1 门
def UG1():
    circuit = QuantumCircuit(2)
    circuit.z(0)
    circuit.append(Us1().inverse(),[i for i in range(2)])
    circuit.x(1)
    circuit.x(0)
    circuit.cz(0,1)
    circuit.x(0)
    circuit.x(1)
    circuit.append(Us1(),[i for i in range(2)])
    circuit = circuit.to_gate()
    circuit.name = "UG1"
    c_U = circuit.control()
    return c_U

#定义量子傅里叶逆变换
def qft_dagger(n):
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT+"
    return qc

#第一步
circuit.append(Us(),[i+9 for i in range(2)])
circuit.barrier(6,7,8,9,10)
circuit.append(Us1(),[i+14 for i in range(2)])
circuit.barrier(11,12,13,14,15)

#第二步
for i in range(3):
    circuit.h(i+6)
for i in range(3):
    for j in range(2**i):
        circuit.append(UG(),[i+6]+[m+9 for m in range(2)])
circuit.append(qft_dagger(3),[i+6 for i in range(3)])
#在线路的下半部分重复上述操作
for i in range(3):
    circuit.h(i+11)
for i in range(3):
    for j in range(2**i):
        circuit.append(UG1(),[i+11]+[m+14 for m in range(2)])
circuit.append(qft_dagger(3),[i+11 for i in range(3)])
```

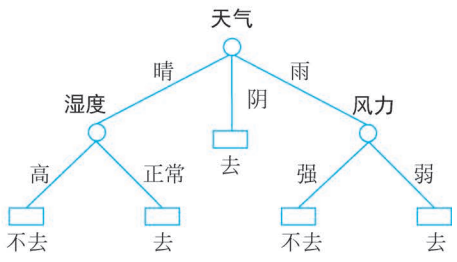
```
101.
102. #第三步
103. circuit.append(comp(),[i for i in range(16)])
104.
105. #测量
106. circuit.measure(0,0)
107. circuit.measure(1,1)
108.
109. #绘制线路图
110. circuit.draw(output = 'mpl', plot_barriers = False, fold = -1)
111. backend = Aer.get_backend('qasm_simulator')
112. job_sim = execute(circuit, backend, shots = 8192)
113. sim_result = job_sim.result()
114.
115. #绘制结果图
116. measurement_result = sim_result.get_counts(circuit)
117. plot_histogram(measurement_result)
```

5.3 量子决策树

决策树(Decision Tree,DT)是一种具有树形结构的机器学习方法。在学习过程中,由样本集形成一棵可分层判决的树。例如,通过天气情况判断是否去打网球,有天气、湿度和风力三个特征,其中天气包括晴、阴和雨,湿度包括高和正常,风力包括强和弱。图 5.12(a)是在各种天气特征下是否打网球的样本。决策树就是要根据已经给出的带标签的样本训练一棵树,当给出一个新的样本时,通过这棵树做出决策。图 5.12(b)是根据打网球样本给出的决策树示例。本节首先介绍决策树的基本原理,然后介绍决策树的量子版本——量子决策树。

序号	天气	湿度	风力	决定
1	晴	高	强	不去
2	晴	高	弱	不去
3	晴	正常	强	去
4	阴	高	弱	去
5	阴	正常	强	去
6	雨	高	强	不去
7	雨	正常	弱	去

(a) 样本集



(b) 决策树

图 5.12 样本集及其对应的决策树示例

5.3.1 决策树基本原理

假设 N 个训练样本组成的数据集为

$$X=\{(\boldsymbol{x}_i,y_i):\boldsymbol{x}_i\in\mathbf{R}^M,y_i\in\{0,1,\cdots,K-1\}\}_{i=0,1,\cdots,N-1} \tag{5.3.1}$$

式中：$\boldsymbol{x}_i$ 为单个样本的 M 维特征向量，$\boldsymbol{x}_i=(x_{0i}\quad x_{1i}\quad\cdots\quad x_{(M-1)i})^{\mathrm{T}}(i=0,1,\cdots,N-1)$；$y_i$ 表示第 i 个样本的类别，共有 $K\geqslant 2$ 个取值，表示数据集共有 K 类。

一棵决策树由若干节点和分支组成，最顶端的节点称为根节点，所有节点都是从根节点开始。节点对应数据中的某一个特征，该特征有几个取值就从这个节点向下引出几个分支，也就是根据这个特征将数据分为若干类别。每个叶节点代表一种分类结果，除根节点和叶节点之外的节点称为内部节点。

由于样本包含多个特征，将哪一个特征作为根节点以及选择哪个特征作为下一个节点是决策树算法的核心。目前常用的算法包括 ID3 算法、ID4.5 算法以及 CART 算法。下面给出最简单的 ID3 算法，为此首先给出熵和条件熵的定义。

对于式(5.3.1)中的数据集来说，假设一个样本属于类别 $k(k=0,1,\cdots,K-1)$ 的概率为 p_k，则数据集的熵定义为

$$H(X)=-\sum_{k=0}^{K-1}p_k\log p_k \tag{5.3.2}$$

式中：p_k 通常取值为 $\frac{c_k}{N}$，c_k 为第 k 类样本的数目。

则计算熵的公式为

$$H(X)=-\sum_{k=0}^{K-1}\frac{c_k}{N}\log\frac{c_k}{N} \tag{5.3.3}$$

熵表示对样本进行分类时的平均不确定性。

进一步，下面给出条件熵的定义。假设一个特征表示为 A，该特征将数据集 X 划分为 I 个子集，分别为 $X_0,X_1,\cdots,X_{I-1}$，各子集的样本数量分别为 $N_i(i=0,1,\cdots,I-1)$，c_{ik} 为子集 X_i 中标注为第 k 类的样本数量。则 $\frac{c_{ik}}{N_i}$ 表示子集 X_i 中一个样本属于类别 k 的概率，$\frac{N_i}{N}$ 表示 X 中一个样本属于子集 X_i 的概率，特征 A 的条件熵定义为

$$H(X\mid A)=-\sum_{i=0}^{I-1}\sum_{k=0}^{K-1}\frac{N_i}{N}\frac{c_{ik}}{N_i}\log\frac{c_{ik}}{N_i}=-\sum_{i=0}^{I-1}\sum_{k=0}^{K-1}\frac{c_{ik}}{N}\log\frac{c_{ik}}{N_i} \tag{5.3.4}$$

式中：$\frac{c_{ik}}{N_i}$ 为在知道一个样本属于子集 X_i 的条件下属于类别 k 的条件概率；$\frac{c_{ik}}{N}$ 为一个样本既属于子集 X_i 又属于类别 k 的联合概率。

特征 A 的条件熵表示根据特征 A，在已经知道一个样本属于某个子集 X_i 的条件下，对数据进行分类时的平均不确定性。

特征 A 的熵增益定义如下：

$$G(X,A)=H(X)-H(X\mid A) \tag{5.3.5}$$

表示在知道特征 A 的前后，样本分类的不确定的差值，也就是特征 A 带来的信息量。

ID3 决策树算法选择熵增益最大的特征作为根节点，也就是一次分类能带来最大信息量的特征。具体算法：首先计算所有特征的熵增益，熵增益最大的特征作为根节点；

然后根据该特征的取值向下引出分支；再计算除根节点之外内部节点其他特征的熵增益，再选择熵增益最大的特征。依次进行下去，当子集 X_i 的熵为 $H(X_i)=0$ 时，该节点为叶节点，输出其对应的标签类型。$H(X_i)=0$ 意味着 X_i 中只有一个类。

5.3.2 量子决策树算法

与经典决策树算法类似，量子决策树算法也要找到能判别哪些节点可以作为根节点和内部节点的判别准则。下面首先给出该算法中样本的表示方法，然后介绍节点划分标准，最后给出量子决策树。

1. 样本表示方法

与之前的算法有所不同，该算法将每个样本中的特征都以叠加态的形式存储，但是并没有将所有样本以叠加态的形式存储在量子态中，数据集的量子表示为

$$X=\{(|x_0\rangle,|y_0\rangle),(|x_1\rangle,|y_1\rangle),\cdots,(|x_{N-1}\rangle,|y_{N-1}\rangle)\} \tag{5.3.6}$$

由于数据集 X 共分成 K 类，则用 $Y=\{|Y_0\rangle,|Y_1\rangle,\cdots,|Y_{K-1}\rangle\}$ 表示类别集，其中 $Y_k=k(k=0,1,\cdots,K-1)$ 表示类别标签，即 $Y=\{|0\rangle,|1\rangle,\cdots,|K-1\rangle\}$。

2. 节点划分标准

在经典算法中选择根节点和内部节点是决策树的重要内容，在量子算法中也是如此。和经典熵定义不同，量子算法使用量子熵。下面介绍量子熵的定义。

对于节点 t 来说，假设属于节点 t 的样本分属 N_t 个类别，则分类集记作 $Y^{(t)}=\{|Y_0^{(t)}\rangle,|Y_1^{(t)}\rangle,\cdots,|Y_{N_t-1}^{(t)}\rangle\}$。则量子熵定义为

$$S(\rho^{(t)})=-\mathrm{tr}(\rho^{(t)}\log\rho^{(t)}) \tag{5.3.7}$$

式中：$\rho^{(t)}$ 为 $Y^{(t)}$ 的密度算子，$\rho^{(t)}=\sum_{i=0}^{N_t-1}p_i^{(t)}\,|Y_i^{(t)}\rangle\langle Y_i^{(t)}|$，$p_i^{(t)}$ 表示 $Y^{(t)}$ 中属于类别 $Y_i^{(t)}$ 的样本的比例。

假设根据特征 A，$Y^{(t)}$ 可分成 J 个子节点，即

$$Y^{(t)}=\{Y^{(t_0)},Y^{(t_1)},\cdots,Y^{(t_{J-1})}\} \tag{5.3.8}$$

式中：$Y^{(t_n)}=\{|Y_0^{(t_n)}\rangle,|Y_1^{(t_n)}\rangle,\cdots,|Y_{N_{t_n}-1}^{(t_n)}\rangle\}(t_n=t_0,t_1,\cdots,t_{J-1})$，$N_{t_n}$ 表示属于子节点 t_n 的样本共有 N_{t_n} 类。

则量子期望熵可定义为

$$S_e(\rho^{(t)})=\sum_{n=0}^{J-1}p_{t_n}^{(t)}S(\rho^{(t_n)}) \tag{5.3.9}$$

式中：$\rho^{(t_n)}$ 为 $Y^{(t_n)}$ 的密度算子；$p_{t_n}^{(t)}$ 为 $Y^{(t)}$ 中属于 $Y^{(t_n)}$ 的样本的比例。

量子熵表示将 $Y^{(t)}$ 存储在量子态中所需要的最少量子比特数量，量子熵越小，需要的量子比特数量越少。对于节点 t 来说，根据不同的特征划分时会有不同的量子期望熵，该量子期望熵相当于经典算法中的条件熵，期望熵越小越好。也就是说，选择量子期望熵最小的特征作为下一个子节点。

3. 量子决策树

量子决策树首先选择期望熵最小的特征作为根节点，然后计算根节点中每个特征的量子期望熵，选择量子期望熵最小的特征作为下一层的根节点。之后，对每个内部节点重复以上选择新特征的过程，一直持续到满足以下两个停止标准中的任何一个为止：

（1）每个特征都已经包含在树中；

（2）与当前节点相关的特征都有相同的目标，即它们的量子熵为零。

最终完成量子决策树的构建。

5.4 本章小结

本章讨论了三种量子分类方法，分别为量子支持向量机、量子 K 近邻和量子决策树。

量子支持向量机是一种二分类机器学习算法。本章主要介绍了线性最小二乘支持向量机的量子形式，并简单介绍了核函数。基于此算法，目前已经产生了大量的量子支持向量机算法，包括 QSVM 多分类算法和基于标准支持向量机的量子算法等，感兴趣的读者可参见文献[8-12]。

K 近邻算法不需要训练就能预测分类。而量子 K 近邻算法利用量子叠加、并行特性能够更快地完成任务。此算法主要使用了两种量子技术：交换测试用于计算样本间的保真度；量子最大值搜索算法用于找到最近邻的 K 个值。

相较于前两种算法，量子决策树算法是一种计算复杂度不高、对中间值的缺失不敏感的分类方法。

参考文献