第 3 章　机器学习基础实践

机器学习是人工智能领域内的一个重要分支，旨在通过计算的手段，利用经验来改善计算机系统的性能，通常，这里的经验即历史数据。从大量的数据中抽象出一个算法模型，然后将新数据输入到模型中，得到模型对其的判断（例如类型、预测实数值等），也就是说，机器学习是一门主要研究学习算法的学科。

实践八：基于线性回归实现房价预测

回归算法是机器学习领域一个非常经典的学习算法，主要用于对输入自变量产生一个对应的输出因变量值，通常，因变量为实数范围内的数值类型数据，形式上，对于一个点集，用一条曲线去拟合其分布的过程，就叫作回归。而线性回归算法是指自变量之间通过一个线性组合便可得到因变量的预测结果的算法，是回归算法中最为简单的一种，对于一些线性可分的数据集，可以尝试使用线性回归模型进行建模。

线性回归算法的表达形式为 $\boldsymbol{y}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}$，$\boldsymbol{w}$ 即为所学习的参数，$\boldsymbol{x}$、$\boldsymbol{y}$ 分别为自变量与因变量，在机器学习任务中，称之为输入特征与输出结果。回归任务最常用的性能度量方式为均方误差，即计算真实值与预测值之间的差平方的均值，也就是真实值与预测值之间的欧氏距离，最小化该值可以使预测误差尽可能小，并且对均方误差值的优化是一个凸优化过程（二次损失函数，可以求得最小值），可以使用最小二乘法对模型进行求解，使得所有样本到所拟合曲线上的距离之和最小。

本书就简单的线性回归模型进行代码演示，在波士顿房价数据集上进行线性建模，对于模型未见过的数据，使用建模的线性回归模型预测其房价，该建模过程主要分为以下四个步骤：数据加载、模型配置、模型训练、模型评估，本次实验平台为百度 AI Studio，实验环境为 Python 3.7，sklearn。

步骤 1：数据加载

（1）数据集下载：首先，从网络中获取开源波士顿房价数据集。该数据集包含 506 条数据，每条数据包含 13 个输入变量和 1 个输出变量，输入变量包含房屋以及房屋周围的详细信息，例如：城镇犯罪率，一氧化氮浓度，住宅平均房间数，到中心区域的加权距离以及自住房平均房价等。在 AI Studio 项目 Notebook 页面的代码模块输入下列命令，即可获得该数据集：

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data -O housing.data
```

（2）数据预处理：对于下载的数据集，由于该数据集中原始的特征尺度不一，因此首先需要对原始数据进行归一化操作，方可进行后续的模型训练，本实验将每一个特征值进行如下归一化处理：（原始值-该特征均值）/（该特征最大值-该特征最小值），归一化后，将其切分为训练集与测试集两个子集：

```
# 加载相关包
import numpy as np
import os
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import linear_model

# 从文件导入数据
datafile = './housing.data'
housing_data = np.fromfile(datafile, sep = ' ')
feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE','DIS', 'RAD', 'TAX', 'PTRATIO',
                 'B', 'LSTAT', 'MEDV']
feature_num = len(feature_names)
# 将原始数据进行 Reshape,变成[N, 14]这样的形状
housing_data = housing_data.reshape([housing_data.shape[0] // feature_num, feature_num])
print(housing_data[:2])
# 输出数据格式如图 3-1 所示
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00 2.4000e+01]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00 2.1600e+01]]
```

图 3-1　波士顿房价原始数据格式

```
# 定义归一化操作：取最大、最小、均值操作
features_max = housing_data.max(axis = 0)
features_min = housing_data.min(axis = 0)
features_avg = housing_data.sum(axis = 0) / housing_data.shape[0]

# 归一化函数
def feature_norm(input):
  f_size = input.shape
  output_features = np.zeros(f_size, np.float32)
  for batch_id in range(f_size[0]):
    for index in range(13):
      output_features[batch_id][index] = (input[batch_id][index] - features_avg[index]) /
(features_max[index] - features_min[index])
  return output_features

# 调用归一化函数
housing_features = feature_norm(housing_data[:, :13])
```

```
# 拼接特征与标签值
housing_data = np.c_[housing_features, housing_data[:-1]].astype(np.float32)
# 将数据集按照 8:2 的比例分为训练集和测试集
ratio = 0.8
offset = int(housing_data.shape[0] * ratio)
train_data = housing_data[:offset]
test_data = housing_data[offset:]
print(train_data[:2])
# 归一化后的数据如图 3-2 所示
```

```
[[-0.0405441   0.06636363 -0.32356226 -0.06916996 -0.03435197  0.05563625
  -0.03475696  0.02682186 -0.37171334 -0.21419305 -0.33569506  0.10143217
  -0.21172912 24.        ]
 [-0.04030818 -0.11363637 -0.14907546 -0.06916996 -0.17632729  0.02612869
   0.10633469  0.1065807  -0.3282351  -0.31724647 -0.06973761  0.10143217
  -0.09693883 21.6       ]]
```

图 3-2　波士顿房价归一化数据

步骤 2：模型配置

本实验使用 sklearn.linear_model.LinearRegression 类实现线性回归：

```
# 实例化模型函数
def Model():
  model = linear_model.LinearRegression()
  return model
# 拟合函数
def train(model,x,y):
    model.fit(x,y)
```

步骤 3：模型训练

首先将训练集的特征值与回归值分开，然后实例化模型，调用 fit()函数训练模型：

```
# 将训练集的特征与回归值分开
x,y = train_data[:,:13],train_data[:,-1:]
model = Model()                          # 实例化一个模型
train(model,x,y)                         # 在训练数据上拟合模型
```

步骤 4：模型评估

模型训练结束后，根据训练好的模型，在测试数据上进行评估。理想状态下，模型的预测值与真实值相等，即 $y'=y$，即两者应该在直线 $y=x$ 上分布，绘制图像，观察预测值与真实值与 $y=x$ 直线的分布差异，可直观判断线性回归模型的性能：

```
# 定义函数绘制预测值与真实值的分布
def draw_infer_result(ground_truths,infer_results):
  title='Boston'
  plt.title(title, fontsize=24)
```

```
    x = np.arange(1,40)
    y = x
    plt.plot(x, y)
    plt.xlabel('ground truth', fontsize=14)
    plt.ylabel('infer result', fontsize=14)
    plt.scatter(ground_truths, infer_results,color='green',label='training cost')
    plt.grid()
    plt.show()
# 测试数据特征值与回归值切分
x_test,y_test = test_data[:,:13],test_data[:,-1:]
# 预测
predict = model.predict(x_test)
# 绘制对比图
draw_infer_result(y_test,predict)
# 对比图输出如图 3-3 所示
```

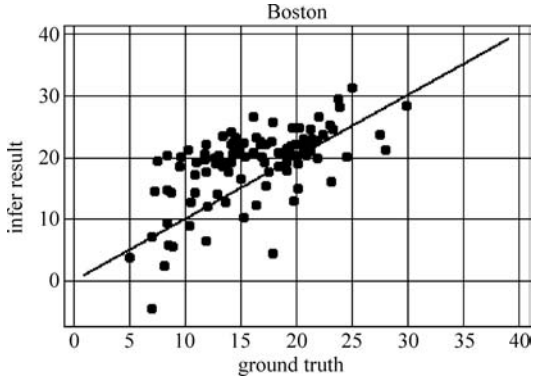

图 3-3　线性回归真实值与预测值分布结果

线性回归算法只能处理线性可分的数据，对于线性不可分数据，需要使用对数线性回归、广义线性回归或者其他回归算法，感兴趣的读者可以自行查阅资料学习。

实践九：基于逻辑回归模型实现手写数字识别

逻辑回归是线性回归的一个变体版本，即建模函数 $\ln\frac{y}{1-y}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-\boldsymbol{b}$，此处，$y$ 为样本 x 作为正样本的可能性，$1-y$ 为其为负样本的可能性，两者的比值 $\frac{y}{1-y}$ 称为几率，反映了 x 作为正样本的相对可能性，因此，逻辑回归又称作对数几率回归。

逻辑回归虽然称作回归，但实际上是一种分类学习算法，无需事先假设数据的分布即可进行建模，避免了先验假设分布偏差带来的影响，并且得到的是近似概率预测，对需要概率结果辅助决策的任务十分友好。逻辑回归使用极大似然估计进行参数学习，即最大化模型的对数似然值，使得每个样本属于真实标签的概率越大越好，该优化目标可以通过牛顿法、梯度下降法等求得最优解。

sklearn 是 Python 的一个机器学习库，它有比较完整的监督学习与非监督学习的算法实现，本节将利用 sklearn 中的逻辑回归算法，实现 MNIST 手写数字识别，本次实验平台为百度 AI Studio，实验环境为 Python 3.7。

步骤 1：数据集加载及预处理

MNIST 数据集来自美国国家标准与技术研究所，训练集由来自 250 个不同人手写的数字构成，其中 50%是高中学生，50%为人口普查局的工作人员，测试集也包含同样比例人群的手写数字图片。由于数据集存储格式为二进制，因此在读取时需要逐字节进行解析。首先将数据集挂在到当前工作空间下，然后解压（在 AI Studio 可编辑 Notebook 界面中，若要执行 Linux 命令，只需在命令前加“!”即可），读取图片数据：

```
!unzip data/data7869/mnist.zip
!gzip -dfq mnist/train-labels-idx1-ubyte.gz
!gzip -dfq mnist/t10k-labels-idx1-ubyte.gz
!gzip -dfq mnist/train-images-idx3-ubyte.gz
!gzip -dfq mnist/t10k-images-idx3-ubyte.gz

# 导入相关包
import struct,os
import numpy as np
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
from sklearn.metrics import accuracy_score,classification_report
import matplotlib.pyplot as plt

# 定义加载 MNIST 数据集的函数
def load_mnist(image_file, label_file, path="mnist"):
  digits=np.arange(10)

  fname_image = os.path.join(path, image_file)
  fname_label = os.path.join(path, label_file)

  flbl = open(fname_label, 'rb')                          # 读取标签文件
  magic_nr, size = struct.unpack(">II", flbl.read(8))
  lbl = pyarray("b", flbl.read())
  flbl.close()

  fimg = open(fname_image, 'rb')                          # 读取图片文件
  magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
  img = pyarray("B", fimg.read())
  fimg.close()

  ind = [ k for k in range(size) if lbl[k] in digits ]
  N = len(ind)

  images = zeros((N, rows*cols), dtype=uint8)
  labels = zeros((N, 1), dtype=int8)
```

```
    for i in range(len(ind)):                                    # 将图片转换为像素矩阵格式
        images[i] = array(img[ ind[i] * rows * cols : (ind[i] + 1) * rows * cols ]).reshape((1, rows
                                * cols))
        labels[i] = lbl[ind[i]]

        return images, labels

# 定义图片展示函数
def show_image(imgdata, imgtarget, show_column, show_row):
    # 注意这里的 show_column * show_row == len(imgdata)
    for index,(im,it) in enumerate(list(zip(imgdata,imgtarget))):
        xx = im.reshape(28,28)
        plt.subplots_adjust(left = 1, bottom = None, right = 3, top = 2, wspace = None, hspace = None)
        plt.subplot(show_row, show_column, index + 1)
        plt.axis('off')
        plt.imshow(xx , cmap = 'gray',interpolation = 'nearest')
        plt.title('label: % i' % it)

# 调用函数,加载训练集数据
train_image, train_label = load_mnist("train - images - idx3 - ubyte", "train - labels - idx1 -
                                ubyte")
# 调用函数,加载测试集数据
test_image, test_label = load_mnist("t10k - images - idx3 - ubyte", "t10k - labels - idx1 -
                                ubyte")
# 显示训练集前 50 数字
show_image(train_image[:50], train_label[:50], 10,5)
# 灰度图展示如图 3-4 所示
```

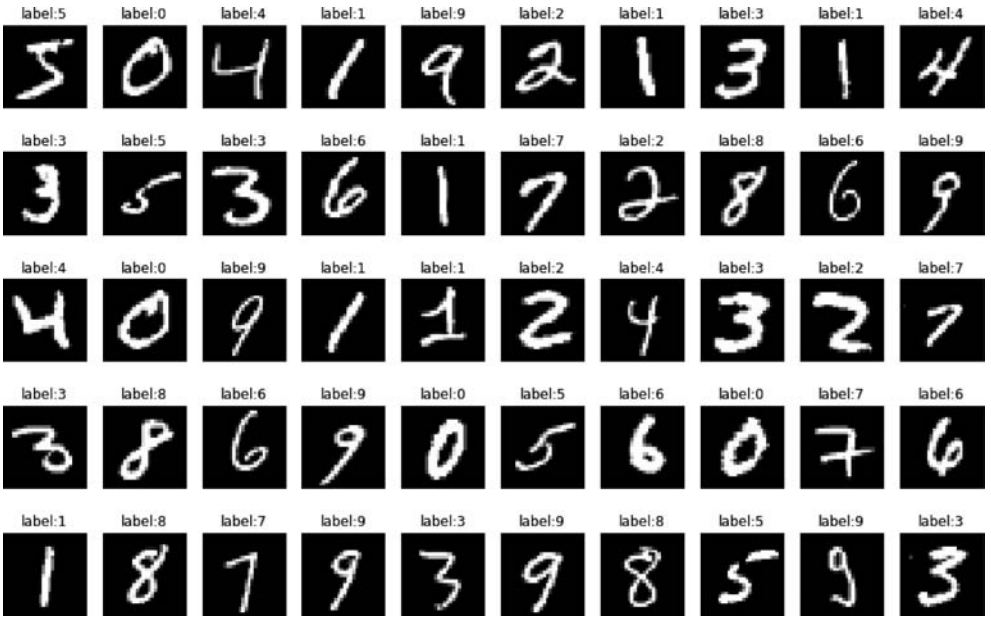

图 3-4　MINST 手写数字

步骤 2：模型定义

此处直接将 sklearn. linear_model 中的 LogisticRegression 导入即可，注意，虽然逻辑回归并没有直接建模输出 y 与输入特征 x 之间的映射关系，但它本质上是线性回归算法的一种变体，且回归参数 w 对于输入特征而言仍是线性的，因此也属于线性模型的范畴。

```
# 导入 LogisticRegression 类
from sklearn.linear_model import LogisticRegression
# 实例化 LogisticRegression 类
lr = LogisticRegression()
```

步骤 3：模型学习

由于图片数据的像素值取值范围为 0～255，过大的计算值可能导致计算结果非常大，或者梯度变化剧烈，因此不利于模型的学习与收敛。为避免上述情况出现，首先需要对训练数据做预处理，也就是尺度缩放，比如对每个像素值都除以其最大像素值 255，将所有像素值压缩到 0～1 的范围内，然后再进行学习。

```
# 数据缩放
train_image = [im/255.0 for im in train_image]
# 训练模型
lr.fit(train_image,train_label)
```

步骤 4：模型验证

模型训练结束后，可在验证集或测试集上测试其性能，对于分类任务，最常见的评价指标包括准确率（Accuracy）、精确率（Precision）、召回率（Recall）、F1 值（F1-Score）等，其中，精确率反映正样本的判断准确率，召回率反映正样本中被实际识别的样本比例，而 F 值则是精确率与召回率的折中，在各类型样本数量不均衡时，该指标可以很好地反映模型的性能。

```
# 数据缩放
test_image = [im/255.0 for im in test_image]
# 测试集结果预测
predict = lr.predict(test_image)
# 打印准确率及各分类评价指标
print("accuracy_score: %.4lf" % accuracy_score(predict,test_label))
print("Classification report for classifier %s:\n%s\n" % (lr, classification_report(test_
label, predict)))
# 各指标输出如图 3-5 所示
```

accuracy_score: 0.9257
Classification report for classifier LogisticRegression

	precision	recall	f1-score	support
0	0.95	0.98	0.96	980
1	0.96	0.98	0.97	1135
2	0.93	0.90	0.91	1032
3	0.90	0.91	0.91	1010
4	0.94	0.93	0.93	982
5	0.91	0.88	0.89	892
6	0.94	0.95	0.94	958
7	0.94	0.92	0.93	1028
8	0.87	0.88	0.88	974
9	0.91	0.92	0.91	1009
accuracy			0.93	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.93	0.93	0.93	10000

图 3-5　逻辑回归手写数字识别结果

实践十：基于朴素贝叶斯实现文本分类

贝叶斯分类算法是以贝叶斯定理为基础的一系列分类算法，包含朴素贝叶斯算法与树增强型朴素贝叶斯算法，朴素贝叶斯算法是最简单但是十分高效的贝叶斯分类算法，因为其假设输入特征之间相互独立，因此得名“朴素”。

在文本分类中，根据贝叶斯定理 $P(c|d)=\frac{P(d|c)\cdot P(c)}{P(d)}$，文档 d 属于类型 c 的概率等于文档 d 对类型 c 的条件概率乘以类型 c 的出现概率，再除以文档 d 的出现概率，取概率最大的类型作为文本的判别类型，可形式化为 $y'=\underset{c\in C}{\operatorname{argmax}}\frac{P(d|c)P(c)}{P(d)}$，其中同一文档计算概率大小时，$P(d)$ 相同，故可省略，因此 $y'=\underset{c\in C}{\operatorname{argmax}}P(d|c)P(c)$，假设文档的特征为 $d=(x_1,x_2,x_3,\cdots,x_n)$，根据朴素贝叶斯的核心思想，各变量之间相互独立，则有 $P(d|c)=P(x_1|c)P(x_2|c)P(x_3|c)\cdots P(x_n|c)$，因此，最终的分类结果变为：$y'=\underset{c\in C}{\operatorname{argmax}}P(x_1|c)P(x_2|c)P(x_3|c)\cdots P(x_n|c)P(c)=\underset{c\in C}{\operatorname{argmax}}P(c)\prod_{x\in d}P(x|d)$。根据上述观察，只需在全局数据集上统计 $P(c)$ 以及 $P(x|c)$，便可轻松获得文本的类型。

本节依旧使用 sklearn 包中封装好的朴素贝叶斯算法，实现文本分类，本次实验平台为百度 AI Studio，实验环境为 Python 3.7。

步骤 1：数据集简介

本实验采用的数据集为网上公开的从中文新闻网站上爬取 56 821 条新闻摘要数据，数据集中包含 10 个类型（各类型数据量统计如表 3-1 所示），本次实验将其中 90% 作为训练集，10% 作为验证集。

表 3-1　新闻数据集样本数统计

国际	4354	汽车	7469
文化	5110	教育	8066
娱乐	6043	科技	6017
体育	4818	证券	3654
财经	7432	房产	3858

步骤 2：文本数据预处理

文本数据由于其自然语言形式，无法直接输入到计算机进行处理，需要对齐进行自然语言到数字的转化。本实验最终将文本表示为 one-hot 形式，即，对于给定词表，若文本中出现了词表中的词，则将与词表大小相同的向量中该词对应的位置置为 1，否则为 0。因此，需要在全局语料上构建一个词表，首先使用 jieba 分词对语料进行分词，为了不使词表过大造成过度复杂的计算，本实验只采样一定数量的高频词作为词表集合，同时，为了避免一些高频无意义的词干扰文本表示，在构建词表时，首先也会将上述高频无意义的停用词去除。

```
#导入必要的包
import random
import jieba                                    # 处理中文
from sklearn import model_selection
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,classification_report
import re,string
```

首先，加载文本，过滤其中的特殊字符：

```
# jieba 分词,将文本转换为词列表
def text_to_words(file_path):
  sentences_arr = []
  lab_arr = []
  with open(file_path,'r',encoding = 'utf8') as f:
    for line in f.readlines():
      lab_arr.append(line.split('_!_')[1]) # 文本所属标签
      sentence = line.split('_!_')[-1].strip()
          # 去除标点符号
      sentence = re.sub("[\s+\.\!\/_,$%^*(+\"\')]+|[+——()?【】“”!,。?、~@#¥%
                        …&*()《》:]+", "",sentence)
      sentence = jieba.lcut(sentence, cut_all = False)
      sentences_arr.append(sentence)
    return sentences_arr, lab_arr
```

加载停用词表，对文本词频进行统计，过滤掉停用词及词频较低的词，构建词表：

```
# 加载停用词表
def load_stopwords(file_path):
  stopwords = [line.strip() for line in open(file_path, encoding = 'UTF-8').readlines()]
    return stopwords
```

```
# 词频统计
def get_dict(sentences_arr,stopswords):
  word_dic = {}
  for sentence in sentences_arr:
    for word in sentence:
      if word != ''and word.isalpha():
        if word not in stopswords:          # 停用词处理
          word_dic[word] = word_dic.get(word,1) + 1
    # 按词频序排列
    word_dic = sorted(word_dic.items(),key = lambda x:x[1],reverse = True)
    return word_dic

# 构建词表,过滤掉频率低于 word_num 的单词
def get_feature_words(word_dic,word_num):
  '''
```

从词典中选取 N 个特征词,形成特征词列表。

```
  return: 特征词列表
  '''
  n = 0
  feature_words = []
  for word in word_dic:
    if n < word_num:
      feature_words.append(word[0])
    n += 1
  return feature_words

# 文本特征表示
def get_text_features(train_data_list, test_data_list, feature_words):
    # 根据特征词,将数据集中的句子转换为特征向量
    def text_features(text, feature_words):
    text_words = set(text)
    features = [1 if word in text_words else 0 for word in feature_words]
    return features # 返回特征
  train_feature_list = [text_features(text, feature_words) for text in train_data_list]
  test_feature_list = [text_features(text, feature_words) for text in test_data_list]
    return train_feature_list, test_feature_list

# 调用上述函数,完成词表构建
sentences_arr, lab_arr = text_to_words('data/data6826/news_classify_data.txt')
#加载停用词
stopwords = load_stopwords('data/data43470/stopwords_cn.txt')
# 生成词典
word_dic = get_dict(sentences_arr,stopwords)
#生成特征词列表,此处使用词维度为 10000
feature_words = get_feature_words(word_dic,10000)
```

切分数据集,并将文本数据转换为固定长度的 id 向量:

```
#数据集划分
```

```
train_data_list, test_data_list, train_class_list, test_class_list = model_selection.train_
test_split(sentences_arr,lab_arr, test_size = 0.1)
#生成特征向量
train_feature_list,test_feature_list = get_text_features(train_data_list,test_data_list,
feature_words)
```

步骤 3：模型定义与训练

上述概率计算中，可能存在某一个单词在某个类型中从来没有出现过，即某个属性的条件概率为 0（$P(x|c)=0$），此时会导致整体概率为零，为了避免这种情况出现，引入拉普拉斯平滑参数，将条件概率为 0 的属性的概率设定为固定值，具体的，对每个类型下所有单词的计数加 1，当训练样本集数量充分大时，并不会对结果产生影响。下面调用接口的参数中，alpha 为 1 时，表示使用拉普拉斯平滑方式，若设置为 0，则不使用平滑；fit_prior 代表是否学习先验概率 $P(Y=c)$，如果设置为 False，则所有的样本类别输出都有相同的类别先验概率；class_prior 为各类型的先验概率，如果没有给出具体的先验概率则自动根据数据来进行计算。

```
# 获取朴素贝叶斯分类器
classifier = MultinomialNB(alpha = 1.0,      # 拉普拉斯平滑
                fit_prior = True,            # 是否要考虑先验概率
                class_prior = None)

#进行训练
classifier.fit(train_feature_list, train_class_list)
```

步骤 4：模型验证

模型训练结束后，可使用验证集测试模型的性能，同上一小节，输出准确率的同时，对各个类型的精确率、召回率以及 F1 值也进行输出。

```
# 在验证集上进行验证
test_accuracy = classifier.score(test_feature_list, test_class_list)
print(test_accuracy)
predict = classifier.predict(test_feature_list)
print(classification_report(test_class_list, predict))
# 输出结果如图 3-6 所示
```

步骤 5：模型预测

使用上述训练好的模型，对任意给定的文本数据，可进行预测，观察模型的泛化性能。

```
# 加载句子,对句子进行预处理：去除标点、分词
def load_sentence(sentence):
    # 去除标点符号
  sentence = re.sub("[\s+\.\!\/_,$ %^*(+\"\')]+|[+——()?【】“ ”！，。？、～@#￥%…&*（）
                    《》：]+", "",sentence)
  sentence = jieba.lcut(sentence, cut_all = False)
```

```
accuracy_score: 0.7700
Classification report for classifier:
              precision    recall  f1-score   support

           0       0.73      0.70      0.72       522
           1       0.74      0.86      0.79       558
           2       0.89      0.82      0.86       504
           3       0.64      0.66      0.65       784
           4       0.82      0.79      0.81       371
           5       0.85      0.85      0.85       733
           6       0.82      0.83      0.83       847
           7       0.71      0.69      0.70       572
           8       0.78      0.66      0.72       433
           9       0.76      0.81      0.78       359

    accuracy                           0.77      5683
   macro avg       0.77      0.77      0.77      5683
weighted avg       0.77      0.77      0.77      5683
```

图 3-6　朴素贝叶斯文本分类结果

```
    return sentence

lab = [ '文化', '娱乐', '体育', '财经','房产', '汽车', '教育', '科技', '国际', '证券']

p_data = '【中国稳健前行】应对风险挑战必须发挥制度优势'
sentence = load_sentence(p_data)
sentence = [sentence]
print('分词结果:', sentence)
#形成特征向量
p_words = get_text_features(sentence,sentence,feature_words)
res = classifier.predict(p_words[0])
print(lab[int(res)])
# 输出结果如图 3-7 所示
```

```
分词结果: [['中国', '稳健', '前行', '应对', '风险', '挑战', '必须', '发挥', '制度', '优势']]
所属类型:  财经
```

图 3-7　文本分类预测结果

实践十一：基于支持向量机实现鸢尾花分类

支持向量机(SVM)是机器学习中经典的分类算法，主要思想为最大化不同类型的样本到分类超平面之间的距离和。当数据完全线性可分时，得到的最大间隔是硬间隔，即两个平行的超平面(间隔带)之间不存在样本点；当数据部分线性可分时，两个超平面之间允许存在一些样本点，此时得到的最大间隔平面是软间隔平面。对于完全线性不可分的数据，一般的支持向量机算法无法满足要求，但是适当使用核技巧，将非线性样本特征映射到高维线性可分空间，然后便可应用支持向量机进行分类，此时的支持向量机称为非线性支持向量机，常用的核技巧包括：线性核函数、多项式核函数、高斯核函数(径向基函数)，其中，高斯核函

数需要进行调参，即核变换的带宽，它控制径向作用范围。

本节仍旧使用 sklearn 中封装好的支持向量机算法，实现鸢尾花分类，并绘制分类超平面，可视化分类效果。本次实验平台为百度 AI Studio，实验环境为 Python 3.7。

步骤 1：数据集加载

在实践九中，本书直接从 sklearn.datasets 中加载集成的数据集，现在采用另一种数据加载方式，从挂载在当前目录下的数据集文件中读取数据，用于训练。

```
# 加载相关包
import numpy as np
from matplotlib import colors
from sklearn import svm
from sklearn import model_selection
import matplotlib.pyplot as plt
import matplotlib as mpl

# 将字符串转换为整型
def iris_type(s):
  it = {b'Iris-setosa':0, b'Iris-versicolor':1, b'Iris-virginica':2}
  return it[s]

# 加载数据
data = np.loadtxt('/home/aistudio/data/data2301/iris.data',
                  dtype=float,                          # 数据类型
          delimiter=',',                                # 数据分割符
          converters={4:iris_type})                     # 将标签用 iris_type 进行转换
# 数据分割，将样本特征与样本标签进行分割
x, y = np.split(data, (4, ), axis=1)
x = x[:, :2]                                            # 取前两个特征进行分类
# 调用 model_selection 函数进行训练集、测试集切分
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, random_state=1,
test_size=0.2)
```

步骤 2：模型配置及训练

sklearn.svm.SVC()函数提供多个可配置参数，其中，C 为错误项的惩罚系数。C 越大，对训练集错误项的惩罚越大。模型在训练集上的准确率越高，越容易过拟合。C 越小，越允许训练样本中有一些误分类错误样本，泛化能力强。对于训练样本带有噪声的情况，一般采用较小的 C，把训练样本集中错误分类的样本作为噪声；Kernel 为采用的核函数，默认为线性核，可选的为 linear/poly/rbf/sigmoid/precomputed，decision_function_shape 为 ovr，一对多分类决策函数。

```
# SVM 分类器构建
def classifier():
  clf = svm.SVC(C=0.8,                                  # 误差项惩罚系数
          kernel='linear',
```

```
            decision_function_shape = 'ovr')                    # 决策函数
    return clf

# 训练模型函数
def train(clf, x_train, y_train):
    clf.fit(x_train, y_train.ravel())                           # 训练集特征向量和 训练集目标值

# SVM 模型定义
clf = classifier()
# 调用函数训练模型
train(clf, x_train, y_train)
```

步骤 3：模型验证

在划分好的测试集上测试模型的准确率，使用两种方法计算模型预测结果的准确率：自定义方法 show_accuracy()以及 sklearn 中机器学习模型封装好的方法 score()，验证两者的一致性，并且输出样本 x 到各个决策超平面的距离，选择正的最大值对应的类型作为分类结果。

```
# 自定义准确率计算方法
def show_accuracy(a, b, tip):
    acc = a.ravel() == b.ravel()
    print('%s Accuracy: %.3f' %(tip, np.mean(acc)))

# 调用两种准确率计算方法，输出对比
def print_accuracy(clf, x_train, y_train, x_test, y_test):
    # 输出封装函数 score()的结果
    print('training prediction: %.3f' %(clf.score(x_train, y_train)))
    print('test data prediction: %.3f' %(clf.score(x_test, y_test)))
    # 输出自定义方法准确率计算结果
    show_accuracy(clf.predict(x_train), y_train, 'traing data')
    show_accuracy(clf.predict(x_test), y_test, 'testing data')
    # 计算决策函数的值，表示 x 到各个分割平面的距离
    print('decision_function:\n', clf.decision_function(x_train)[:2])

# 模型评估：调用 print_accuracy()函数
print_accuracy(clf, x_train, y_train, x_test, y_test)
# 输出结果如图 3-8 所示
```

```
training prediction:0.808
test data prediction:0.767
traing data Accuracy:0.808
testing data Accuracy:0.767
decision_function:
 [[-0.24991711  1.2042151   2.19527349]
 [-0.30144975  1.25525744  2.28694265]]
```

图 3-8　SVM 鸢尾花分类结果

步骤 4：模型可视化展示

若要绘制各个类型对应的空间区域，需要采样大量的样本点，但是本数据集仅包含 150 条数据，绘制的区域不太精细，因此，需要生成大规模的样本数据，根据生成的数据进行分类区域的绘制，过程如下（本实验采用样本的前两维特征进行分类）：首先在各维特征的最大值与最小值区间内进行采样，生成行相同矩阵（矩阵每行向量中各元素值都相同）与列相同矩阵（矩阵每列向量中各元素值都相同），然后将两矩阵拉平为两个长向量，两个长向量每个元素分别作为样本的第一个特征与第二个特征，使用训练好的 SVM 模型对生成的样本点进行预测，将生成的样本点使用不同的颜色散落在坐标空间中，当样本点足够多时，分类边界便会显示地更加精细。其中，生成的辅助绘图的样本点及其预测结果如图 3-9 所示，我们取前两个样点进行展示，最终绘制的可视化展示结果如图 3-10 所示。

```
def draw(clf, x):
    iris_feature = 'sepal length', 'sepal width', 'petal length', 'petal width'
    # 获取第 1、2 维特征的最大值与最小值
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max()
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max()
    # 生成网格采样点
    x1, x2 = np.mgrid[x1_min:x1_max:200j, x2_min:x2_max:200j]
    # 生成样本点
    grid_test = np.stack((x1.flat, x2.flat), axis = 1)
    print('grid_test:\n', grid_test[:2])
    # 计算样本到决策面的距离
    z = clf.decision_function(grid_test)
    print('the distance to decision plane:\n', z[:2])
    grid_hat = clf.predict(grid_test)
    # 预测分类值：得到[0, 0, …, 2, 2]
    print('grid_hat:\n', grid_hat[:2])
    # 使得 grid_hat 和 x1 形状一致
    grid_hat = grid_hat.reshape(x1.shape)
    cm_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
      cm_dark = mpl.colors.ListedColormap(['g', 'b', 'r'])
      # 绘制分类区域：能够直观表现出分类边界
    plt.pcolormesh(x1, x2, grid_hat, cmap = cm_light)
    # 训练集与测试集数据：散点图
    plt.scatter(x[:, 0], x[:, 1], c = np.squeeze(y), edgecolor = 'k', s = 50, cmap = cm_dark )
    plt.scatter(x_test[:, 0], x_test[:, 1], s = 120, facecolor = 'none',zorder = 10)
    plt.xlabel(iris_feature[0], fontsize = 20) # 注意单词的拼写 label
    plt.ylabel(iris_feature[1], fontsize = 20)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.title('Iris data classification via SVM', fontsize = 30)
    plt.grid()
    plt.show()
draw(clf, x)
```

```
grid_test:
 [[4.3       2.       ]
 [4.3       2.0120603]]
the distance to decision plane:
 [[ 1.15418548  2.24935988 -0.26432263]
 [ 1.15805875  2.2485129  -0.26434377]]
grid_hat:
 [1. 1.]
```

图 3-9　SVM 鸢尾花分类-预测结果

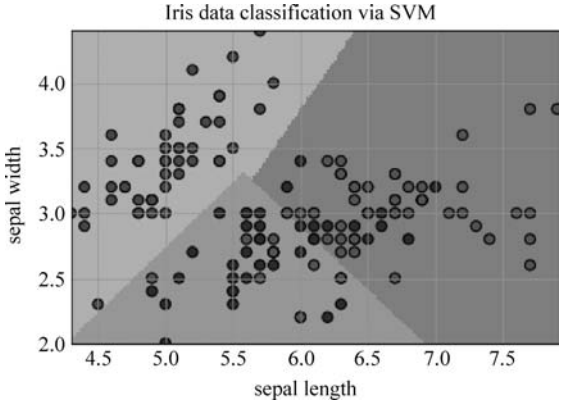

图 3-10　SVM 鸢尾花分类可视化展示

实践十二：基于 K-means 实现鸢尾花聚类

K-means 是一种经典的无监督聚类算法，对于给定的样本集，按照样本之间的距离大小，将样本集划分为 K 个簇，让簇内的点尽量紧密地连在一起，而让簇间的距离尽量大。K-means 的学习过程本质上是不停更新簇心的过程，一旦簇心确定，该算法便完成了学习过程。K 的取值也需要人为定义，K 很大时，模型趋向于在训练集上表现地好，即过拟合，但在测试集上性能可能较差。K 过小时，可能导致簇心不准确，在训练集与测试集上的性能均较差。因此，虽然 K-means 算法较为简单，但是也存在天然的弊端，且对离群点很敏感。

K-means 算法首先随机初始化或随机抽取 K 个样本点作为簇心，然后以这 K 个簇心进行聚类，聚类后重新计算簇心（一般为同一簇内样本的均值），重复上述操作，直至簇心趋于稳定或者达到指定迭代次数时停止迭代。本书使用两种方法实现 K-means 的聚类，前者手动实现，后者通过调用 sklearn 封装好的库快速实现。本次实验平台为百度 AI Studio，实验环境为 Python 3.7。

步骤 1：加载数据集

本书使用鸢尾花数据集进行聚类演示，鸢尾花数据集中包含三种类型，共 150 条数据，每条数据包含 4 项特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度，sklearn.datasets 已经集成了该数据集，因此可直接加载使用：

```
# 加载相应的包
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn import datasets

# 直接从 sklearn 中获取数据集
iris = datasets.load_iris()
X = iris.data[:, :4]                                    # 表示取特征空间中的 4 个维度
```

步骤 2：手动实现 K-means

(1) 首先定义距离测量标准，本书使用欧氏距离衡量两样本之间的距离，定义如下：

```
# 欧氏距离计算
def distEclud(x,y):
    return np.sqrt(np.sum((x-y)* *2))                   # 计算欧氏距离
```

(2) 定义簇心，此处使用随机抽取的 k 个样本点为簇心，进行后续计算：

```
# 为给定数据集构建一个包含 K 个随机簇心 centroids 的集合
def randCent(dataSet,k):
  m,n = dataSet.shape                                   # m=150,n=4
  centroids = np.zeros((k,n))                           # k*4
  for i in range(k):                                    # 执行四次
    index = int(np.random.uniform(0,m))                 # 产生 0 到 150 的随机数
    centroids[i,:] = dataSet[index,:]                   # 把对应行的四个维度赋值到簇心
    return centroids
```

(3) 实现 K-means 算法：首先初始化簇心，然后遍历所有点，找到其对应的簇，更新簇心，重复迭代上述过程，直到簇心不再发生变化。

```
# k 均值聚类算法
def KMeans(dataSet, k):
    m = np.shape(dataSet)[0]                            # 样本数
    # np.mat()创建 150*2 的矩阵
  # 第一列存每个样本属于哪一簇,第二列存每个样本到簇心的误差
  clusterAssment = np.mat(np.zeros((m,2)))
  clusterChange = True

  # 初始化质心 centroids
  centroids = randCent(dataSet,k)
  while clusterChange:
    # 样本所属簇不再更新时停止迭代
    clusterChange = False
    # 遍历所有的样本
    for i in range(m):
      minDist = 100000.0
      minIndex = -1
      # 遍历所有的簇心
```

```
            # 找出最近的簇心
            for j in range(k):
                # 计算该样本到 k 个簇心的欧式距离
                    # 找到距离最近的那个簇心 minIndex
                distance = distEclud(centroids[j,:],dataSet[i,:])
                if distance < minDist:
                    minDist = distance
                    minIndex = j
            # 更新该行样本所属的簇
            if clusterAssment[i,0] != minIndex:
                clusterChange = True
                clusterAssment[i, :] = minIndex, minDist * * 2
        # 更新簇心
        for j in range(k):
            # 获取对应簇类所有的点
            pointsInCluster = dataSet[np.nonzero(clusterAssment[:, 0].A == j)[0]]
            # 求均值,产生新的质心
            centroids[j, :] = np.mean(pointsInCluster, axis = 0)

        return centroids, clusterAssment
```

(4) 可视化展示函数定义,分别取前两个维度的特征与后两个维度的特征绘图,便于观察聚类效果:

```
def draw(data,center,assment):
    length = len(center)
    fig = plt.figure
    data1 = data[np.nonzero(assment[:,0].A == 0)[0]]
    data2 = data[np.nonzero(assment[:,0].A == 1)[0]]
    data3 = data[np.nonzero(assment[:,0].A == 2)[0]]
    # 选取前两个维度绘制原始数据的散点图
    plt.scatter(data1[:,0],data1[:,1],c = "red",marker = 'o',label = 'label0')
    plt.scatter(data2[:,0],data2[:,1],c = "green", marker = ' * ', label = 'label1')
    plt.scatter(data3[:,0],data3[:,1],c = "blue", marker = ' + ', label = 'label2')
    # 绘制簇的质心点
    for i in range(length):
        plt.annotate('center',xy = (center[i,0],center[i,1]),xytext = \
        (center[i,0] + 1,center[i,1] + 1),arrowprops = dict(facecolor = 'yellow'))
        # plt.annotate('center',xy = (center[i,0],center[i,1]),xytext = \
        # (center[i,0] + 1,center[i,1] + 1),arrowprops = dict(facecolor = 'red'))
    plt.show()

    # 选取后两个维度绘制原始数据的散点图
    plt.scatter(data1[:,2],data1[:,3],c = "red",marker = 'o',label = 'label0')
    plt.scatter(data2[:,2],data2[:,3],c = "green", marker = ' * ', label = 'label1')
    plt.scatter(data3[:,2],data3[:,3],c = "blue", marker = ' + ', label = 'label2')
    # 绘制簇的质心点
    for i in range(length):
        plt.annotate('center',xy = (center[i,2],center[i,3]),xytext = \
        (center[i,2] + 1,center[i,3] + 1),arrowprops = dict(facecolor = 'yellow'))
```

```
    plt.show()
```

(5) 执行 K-means 过程,实现鸢尾花数据集的聚类,因为鸢尾花数据集一共包含三种类型,因此此处直接设置 K=3:

```
dataSet = X
k = 3
centroids, clusterAssment = KMeans(dataSet,k)
draw(dataSet, centroids, clusterAssment)
# 可视化结果如下,其中黄色箭头指向簇心,如图 3-11 所示
```

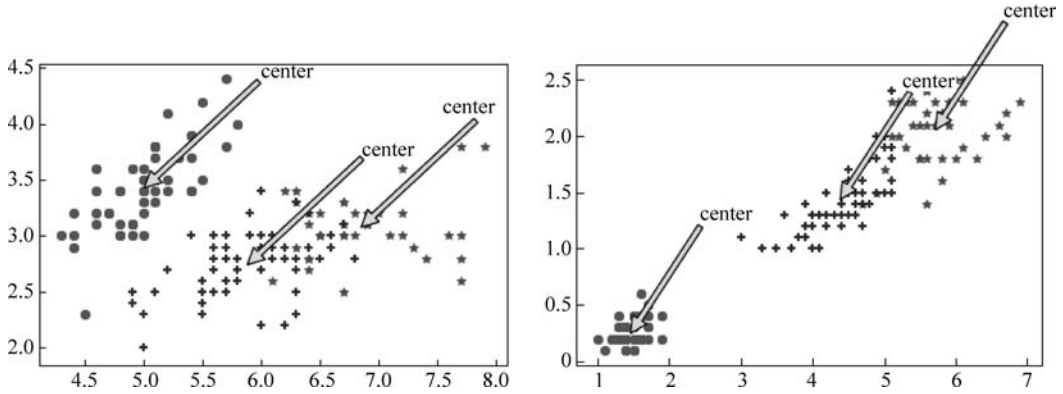

图 3-11　手动实现 K-means 对鸢尾花聚类的结果

步骤 3:调用 sklearn 库实现 K-means

sklearn.cluster.KMeans 包封装了 K-means 的实现,因此,读者可以非常方便地实现 K-means 聚类:

```
def Model(n_clusters):
    estimator = KMeans(n_clusters = n_clusters)          # 构造聚类器
    return estimator

def train(estimator):
    estimator.fit(X)                                     # 聚类

# 初始化实例,并开启训练迭代计算簇心
estimator = Model(3)
train(estimator)

label_pred = estimator.labels_                           # 获取聚类标签
# 绘制 k - means 结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c = "red", marker = 'o', label = 'label0')
plt.scatter(x1[:, 0], x1[:, 1], c = "green", marker = '*', label = 'label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "blue", marker = '+', label = 'label2')
```

```
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc = 2)
plt.show()
# 可视化结果如图 3-12 所示
```

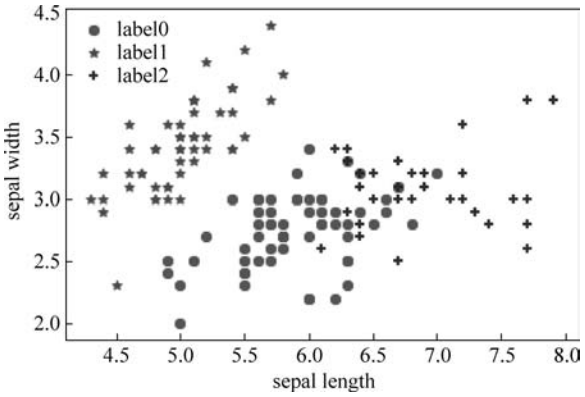

图 3-12　sklearn 库 K-means 进行鸢尾花聚类的结果