南開大學
Nankai University

数据挖掘实验报告

学　　号:　2213117

姓　　名:　蔡 佳 良

年　　级:　2022 级

学　　院:　统计与数据科学学院

专　　业:　统 计 学

完成日期:　2024 年 12 月 20 日

目录

第一章 第一次上机实验（提取图像的纹理特征）

1.1 实验要求

- 给定若干张图像，利用灰度共生矩阵特征或局部二值模式特征对这些图像进行特征提取；
- 图象是 W * H * 3 的矩阵；
- 将最终提取到的特征通过 plot 的形式展示，直观对比不同纹理提取到的特征；
- 使用 Python 编程实现。

1.2 数据分析与处理

将图像转化为灰度矩阵。

1.3 实验步骤与原理

1.3.1 灰度共生矩阵

灰度共生矩阵是从 $N \times N$ 的图像 $f(x, y)$ 的灰度为 i 的像素出发，统计与距离为 $\delta = (\mathrm{d}x^2 + \mathrm{d}y^2)^{1/2}$，灰度为 j 的像素同时出现的概率为, 数学表达式为：

$$P(i, j, \delta, \theta) = \{[(x, y), (x + \mathrm{d}x, y + \mathrm{d}y)] | f(x, y) = i, f(x + \mathrm{d}x, y + \mathrm{d}y) = j\},$$

分别考虑 $\theta = 0°, 45°, 90°, 135°$,
当 $\theta = 0°$ 时，$\mathrm{d}x = 1, \mathrm{d}y = 0$；
当 $\theta = 45°$ 时，$\mathrm{d}x = 1, \mathrm{d}y = -1$；
当 $\theta = 90°$ 时，$\mathrm{d}x = 0, \mathrm{d}y = -1$；
当 $\theta = 135°$ 时，$\mathrm{d}x = -1, \mathrm{d}y = -1$。

1.3.2 局部二值模式

LBP 算子定义为在 3*3 的窗口内，以窗口中心像素为阈值，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样，3*3 邻域内的 8 个点经比较可产生 8 位二进制数（通常转换为十进制数即 LBP 码，共 256 种），即得到该窗口中心像素点的 LBP 值，并用这个值来反映该区域的纹理信息（例如亮点和暗点），数学表达式为：

$$LBP = \sum_{p=0}^{P-1} s(g_p - g_c) \cdot 2^p,$$

其中，

- $s(x)$ 是符号函数：

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases},$$

- P 是邻域像素数（通常为 8）。

1.4 实验结果与分析

通过两种方法下生成的特征图可以看出，两类图有明显不同的纹理特征。

1.5 实验代码

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from PIL import Image, ImageDraw, ImageFont

mpl.rcParams["font.sans-serif"] = ["SimHei"]  # 中文
plt.rcParams["axes.unicode_minus"] = False  # 正负号

# 更改当前工作目录为脚本所在目录
os.chdir(os.path.dirname(os.path.abspath(__file__)))
c_path = "./cloud/"
f_path = "./forest/"
```

```
14
15 os.makedirs("./output/glcm/", exist_ok=True)  # 创建输出文件夹
16 os.makedirs("./output/lbp/", exist_ok=True)
17
18 c_name = [c_path + i for i in os.listdir(c_path)]
19 f_name = [f_path + i for i in os.listdir(f_path)]
20
21
22 def show_grey(root=c_name):
23     for n, i in enumerate(c_name):
24         plt.subplot(2, 5, n + 1)
25         img = Image.open(i)
26         img = img.convert("L")  # 转为灰度图
27         plt.imshow(img, cmap="gray")  # 显示灰度图
28         plt.axis("off")  # 不显示坐标轴
29     plt.show()
30
31
32 def get_mat(root):
33     for n, i in enumerate(root):
34         # plt.subplot(2, 5, n + 1)
35         img = Image.open(i)
36         img = img.convert("L")  # 转为灰度图
37         img = np.array(img)  # 转为ndarray
38         yield i, img
39
40
41 # 计算灰度共生矩阵(gray-level co-occurrence matrix)
42 def get_glcm(img, angle=0, distance=1, gray_levels=256):
43     # 归一化到0-gray_levels灰度级
44     img = (img / (np.max(img) + 1e-5) * (gray_levels - 1)).
          astype(int)
45     h, w = img.shape  # 获取图像的尺寸
46     glcm = np.zeros((gray_levels, gray_levels), dtype=np.
          int64)
47
48     for i in range(h):
```

```
49          for j in range(w):
50              match angle:
51                  case 0:
52                      if j + distance < w:
53                          glcm[img[i, j], img[i, j + distance
        ]] += 1
54                  case 90:
55                      if i + distance < h:
56                          glcm[img[i, j], img[i + distance, j
        ]] += 1
57                  case 45:
58                      if i + distance < h and j + distance <
        w:
59                          glcm[img[i, j], img[i + distance, j
         + distance]] += 1
60                  case 135:
61                      if i + distance < h and j - distance >=
         0:
62                          glcm[img[i, j], img[i + distance, j
         - distance]] += 1
63
64      glcm = glcm / np.sum(glcm)  # 归一化
65      return glcm
66
67
68  # 绘制灰度共生矩阵
69  def get_glcm_fig(root, angles=[0, 45, 90, 135]):
70      for i, img in get_mat(root):
71          glcms = []  # 存储各角度的 GLCM 图像
72          for angle in angles:
73              glcm = get_glcm(img, angle=angle)
74              glcm = (glcm / glcm.max() * 255).astype(np.
        uint8)
75              glcms.append((Image.fromarray(glcm), f"Angle: {
        angle}°"))  # 保存图像和标题
76
77          # 计算单张 GLCM 图像的大小
```

```
78          width, height = glcms[0][0].size
79          title_height = 20  # 为标题预留的高度
80
81          # 2x2 布局，标题增加额外空间
82          canvas = Image.new("L", (2 * width, 2 * (height +
        title_height)), "white")
83
84          draw = ImageDraw.Draw(canvas)
85          font = ImageFont.load_default()
86
87          for idx, (glcm, title) in enumerate(glcms):
88              x_offset = (idx % 2) * width
89              y_offset = (idx // 2) * (height + title_height)
90
91              text_bbox = draw.textbbox((0, 0), title, font=
        font)
92              text_width = text_bbox[2] - text_bbox[0]
93              text_x = x_offset + (width - text_width) // 2
94              draw.text((text_x, y_offset), title, fill="
        black", font=font)
95
96              canvas.paste(glcm, (x_offset, y_offset +
        title_height))
97
98          canvas.save(f"./output/glcm/{i}")
99
100
101 # 计算局部二值模式(local binary pattern)
102 def get_lbp(img):
103     h, w = img.shape  # 获取图像的尺寸
104     lbp = np.zeros((h - 2, w - 2), dtype=np.int64)  # 忽略边
        缘像素
105     for i in range(1, h - 1):
106         for j in range(1, w - 1):
107             center = img[i, j]
108             neighbors = [
109                 img[i, j - 1],
```

```
110                     img[i + 1, j - 1],
111                     img[i + 1, j],
112                     img[i + 1, j + 1],
113                     img[i, j + 1],
114                     img[i - 1, j + 1],
115                     img[i - 1, j],
116                     img[i - 1, j - 1],
117                 ]
118                 code = 0
119                 for idx, neighbor in enumerate(neighbors):
120                     if neighbor > center:
121                         code += 2**idx
122                 lbp[i - 1, j - 1] = code
123     return lbp
124
125
126 # 绘制局部二值模式
127 def get_lbp_fig(root):
128     for i, img in get_mat(root):
129         lbp = get_lbp(img)
130         Image.fromarray(lbp.astype(np.uint8)).save(f"./
        output/lbp/{i}")
131
132
133 def main():
134     roots = [c_name, f_name]
135     for root in roots:
136         get_glcm_fig(root)
137         get_lbp_fig(root)
138
139
140 if __name__ == "__main__":
141     main()
```

第二章 第二次上机实验（垂直平分分类器）

2.1 实验要求

- 由训练数据，训练一个垂直平分分类器；
- 对测试数据进行分类；
- 使用 Python 编程实现。

2.2 数据分析与处理

测试集与训练集均为已分类的二维数据。

2.3 实验步骤与原理

训练集中两类数据的均值分别为记为 m_1 和 m_2，则垂直平分形式的线性分类器为

$$g(\boldsymbol{x}) = (m_1)'(\boldsymbol{x} - (m_1 - m_2)/2),$$

决策面方程为

$$g(\boldsymbol{x}) = 0,$$

其中 $\boldsymbol{x}$ 为向量。

当 $g(\boldsymbol{x}) > 0$ 时，决策为“1”类，否则为“2”类。

2.4 实验结论与分析

图 2.1 展示了训练集与决策面的可视化图，图 2.2 展示了测试集与决策面可视化图。进而可以计算出模型的准确率（accuracy），精确率（precision），召回率（recall）和 F1 得分的值分别为 99.25%，99.00%，99.50%，99.25%（保留两位小数）。说明模型的预测性能表现优异。

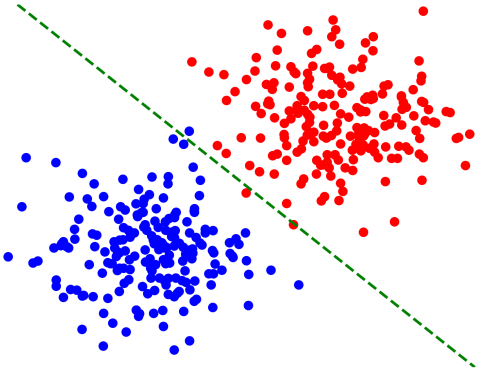

图 2.1: 训练集与决策面

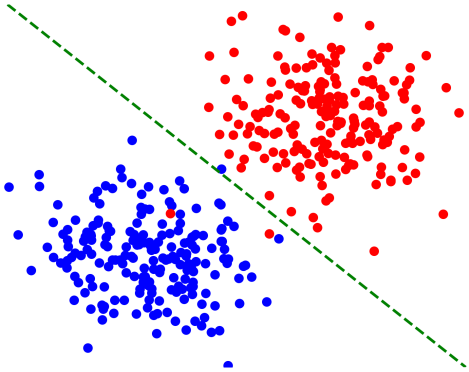

图 2.2: 测试集与决策面

2.5 实验代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import os

mpl.rcParams["font.sans-serif"] = ["SimHei"]  # 中文
plt.rcParams["axes.unicode_minus"] = False  # 负号
os.chdir(os.path.dirname(os.path.abspath(__file__)))  # 更改当前工作目录为脚本所在目录

train_path = "data/train.txt"
test_path = "data/test.txt"


def train(data):
    mean = data.groupby("Label").mean()
    m_1 = mean.loc[1]
    m_2 = mean.loc[2]
    point = (m_1 + m_2) / 2
    slope = m_1 - m_2
    return point, slope


```

```
24 def test(data, point, slope):
25     data["Predict"] = np.where(
26         (data.X - point.X) * slope.X + (data.Y - point.Y) *
           slope.Y > 0, 1, 2
27     )
28     data.to_csv("output/test_predict.csv")
29     return data
30
31
32 def plot(data, point, slope, filename):
33     plt.axline(
34         point,
35         slope=-slope.X / slope.Y,
36         color="green",
37         linewidth=2,
38         linestyle="--",
39         label="Perpendicular Bisector Classifier",
40     )
41     color = ["r", "b"]
42     for _, i in data.iterrows():
43         x, y, c = i.iloc[0], i.iloc[1], color[int(i.iloc
           [2]) - 1]
44         plt.scatter(x, y, color=c)
45     plt.xlim(-8, 8)
46     plt.ylim(-8, 8)
47     plt.axis("off")
48     plt.savefig(filename, dpi=300, bbox_inches="tight")
49     # plt.show()
50     plt.close()
51
52
53 def evaluate(data):
54     TP = 0
55     TN = 0
56     FP = 0
57     FN = 0
58     for _, i in data.iterrows():
```

```
59          if i.iloc[2] == i.iloc[3]:
60              if i.iloc[2] == 1:
61                  TP += 1
62              else:
63                  TN += 1
64          else:
65              if i.iloc[2] == 1:
66                  FP += 1
67              else:
68                  FN += 1
69      accuracy = (TP + TN) / (TP + TN + FP + FN)
70      precision = TP / (TP + FP)
71      recall = TP / (TP + FN)
72      F1 = 2 * precision * recall / (precision + recall)
73      print("accuracy:", accuracy)
74      print("precision:", precision)
75      print("recall:", recall)
76      print("F1 score:", F1)
77
78
79  def main():
80      train_data = pd.read_csv(train_path, index_col=0)
81      test_data = pd.read_csv(test_path, index_col=0)
82      point, slope = train(train_data)
83      # 绘制测试集与决策界
84      plot(train_data, point, slope, "./output/train.png")
85      # 绘制训练集与决策界
86      plot(test_data, point, slope, "./output/test.png")
87      test_data = test(test_data, point, slope)
88      # 评估测试结果
89      evaluate(test_data)
90
91
92  if __name__ == "__main__":
93      main()
```

第三章　第三次上机实验（使用朴素贝叶斯方法进行预测）

3.1　实验要求

- 给定一定时间内的犯罪数据集，使用朴素贝叶斯方法进行犯罪类型预测；
- 使用 Python 编程实现。

3.2　数据分析与处理

原始数据包括 Dates, Category, Descript, DayOfWeek, PdDistrict。其中 Category 是我们要尽心预测的对象，我们对其进行编码，然后我们对 Dates, DayOfWeek, PdDistrict 进行 one-hot 编码，利用 TD-IDF 特征从 Descript 中提取关键词并进行编码。

3.3　实验步骤与原理

给定特征 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ 和类别 $\boldsymbol{v} = (v_1, v_2, \cdots, v_m)$。在分类任务中，我们想要找到使得 $P(v|\boldsymbol{x})$ 最大的 v。利用贝叶斯公式

$$P(v|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|v)P(v)}{P(\boldsymbol{x})},$$

又 $P(x)$ 为定值，则我们比较 $P(\boldsymbol{x}|v)P(v)$ 即可。

在朴素贝叶斯中，我们假设特征 $x_1, x_2, \cdots, x_n$ 之间是相互独立的，即

$$P(\boldsymbol{x}|v) = \prod_{i=1}^{n} P(x_i|v).$$

则使得 $P(v|\boldsymbol{x})$ 最大的

$$v = \arg\max_{v_j} P(v_j) \prod_{i=1}^{n} P(x_i|v_j)$$

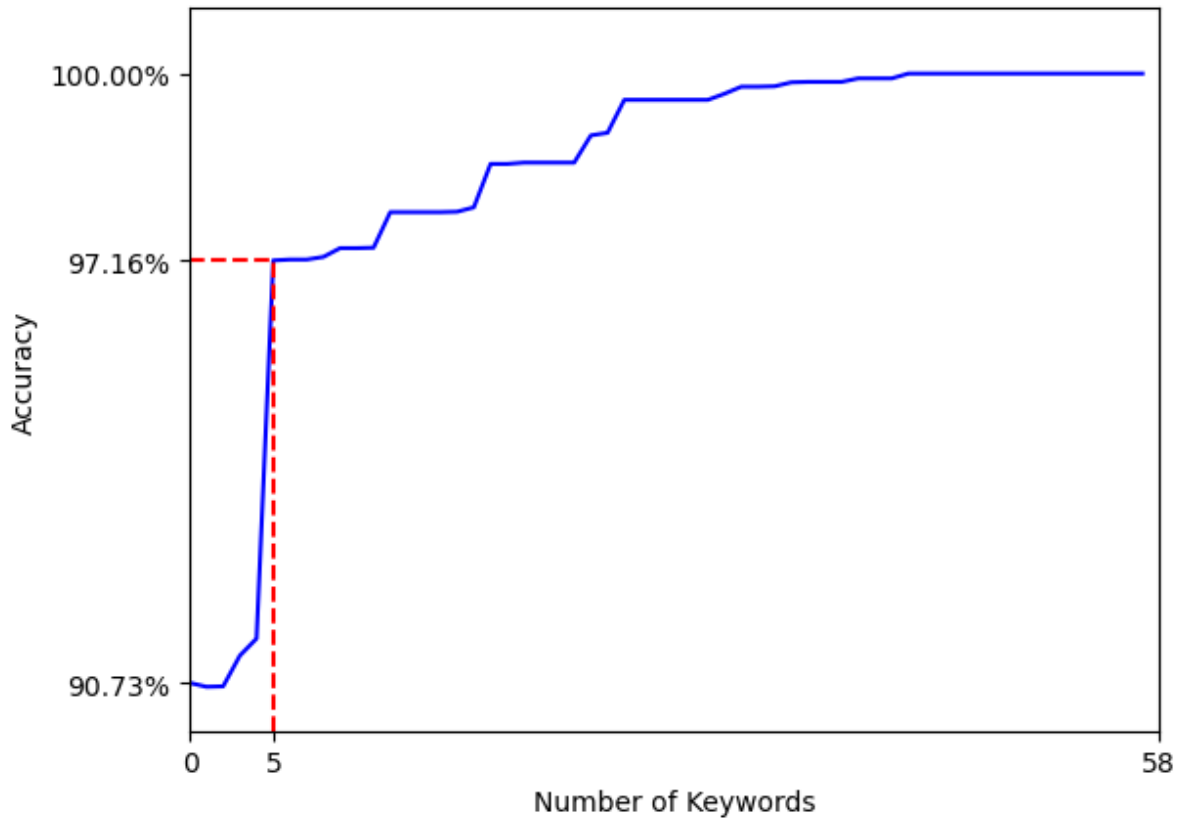

图 3.1: 测试集上的准确率与关键词个数关系图

3.4 实验结论与分析

根据 TF-IDF 值对每一个 Descript 选取一个关键词，选取出现频率最高的 k 个关键词进行模型拟合，在测试集上的准确率如图 3.1 所示。为了防止模型过拟合并保证其泛化能力，选取频率最高的前 5 个关键词。

考虑测试集和训练集 Category 中 DRUG/NARCOTIC 类数据分别占据 90.79% 和 89.81%，为不均衡数据，本实验模型的分类正确率达到了 97.16%，效果较好。

3.5 实验代码

```
1 import os
2 import pandas as pd
3 from sklearn import preprocessing
4 from sklearn.naive_bayes import BernoulliNB
```

```
5  from sklearn.metrics import accuracy_score
6  from sklearn.feature_extraction.text import TfidfVectorizer
7  import matplotlib.pyplot as plt
8
9  os.chdir(os.path.dirname(os.path.abspath(__file__)))  # 更改
       当前工作目录为脚本所在目录
10
11 train_path = "train.csv"
12 test_path = "test.csv"
13
14
15 def keyword_extract(path="train.csv"):
16     df = pd.read_csv(path)
17     vectorizer = TfidfVectorizer(stop_words="english",
           token_pattern=r"(?u)\b\w+\b")
18     tfidf = pd.DataFrame(
19         vectorizer.fit_transform(df["Descript"]).toarray(),
20         columns=vectorizer.get_feature_names_out(),
21     )
22     keywords = tfidf.idxmax(axis=1)
23     keywords = keywords.value_counts(normalize=True).index.
           str.upper().tolist()
24     return keywords
25
26
27 def Pre_Process(path, keywords):
28     df = pd.read_csv(path)
29
30     # 将Category进行整数编码
31     encoder = preprocessing.LabelEncoder()
32     crime_type_encode = encoder.fit_transform(df["Category"
           ])
33
34     # 将时间进行one-hot编码
35     hour = pd.to_datetime(df["Dates"]).dt.hour
36     hour = pd.get_dummies(hour)
37     day = pd.get_dummies(df["DayOfWeek"])
```

```
38
39      # 将所属警区进行one-hot编码
40      police_district = pd.get_dummies(df["PdDistrict"])
41
42      # 利用 TF-IDF 特征进行编码
43      matrix = pd.DataFrame(0, index=df.index, columns=
            keywords)
44      for keyword in keywords:
45          matrix[keyword] = df["Descript"].apply(
46              lambda x: True if keyword in x else False
47          )
48
49      # 将特征合并
50      data = pd.concat([hour, day, police_district, matrix],
            axis=1)
51      data["Crime type"] = crime_type_encode
52
53      # Feature names are only supported if all input features
            have string names
54      data.columns = data.columns.astype(str)
55      return data
56
57
58  keywords = keyword_extract()
59  acc = []
60  for k in range(0, len(keywords) + 1):
61      train = Pre_Process(train_path, keywords[:k])
62      test = Pre_Process(test_path, keywords[:k])
63      # 训练模型
64      model = BernoulliNB()
65      model.fit(train.drop("Crime type", axis=1), train["
            Crime type"])
66
67      # 预测结果
68      pred = model.predict(test.drop("Crime type", axis=1))
69      acc.append(
70          accuracy_score(test["Crime type"], pred),
```

```
71      )
72
73  # plot
74  plt.plot(range(0, len(keywords) + 1), acc, color="b")
75  plt.xlabel("Number of Keywords")
76  plt.ylabel("Accuracy")
77  points = [(0, acc[0]), (5, acc[5]), (len(acc), acc[len(acc)
            - 1])]
78  xticks = [x for x, _ in points]
79  yticks = [y for _, y in points]
80  plt.xticks(xticks, [f"{x}" for x in xticks])
81  plt.yticks(yticks, [f"{y * 100:.2f}%" for y in yticks])
82  plt.plot([points[1][0], points[1][0]], [0.90, points
           [1][1]], color="r", linestyle="--")
83  plt.plot([0.00, points[1][0]], [points[1][1], points
           [1][1]], color="r", linestyle="--")
84  plt.xlim(0, len(keywords) + 1)
85  plt.ylim(0.90, 1.01)
86  # plt.show()
87  plt.savefig("accuracy.png", bbox_inches="tight")
```

第四章 第四次上机实验（使用决策树进行预测）

4.1 实验要求

- 给定一定时间内的犯罪数据集，使用决策树方法进行犯罪类型预测；
- 使用 Python 编程实现。

4.2 数据分析与处理

同 3.2 节。

4.3 实验步骤与原理

本试验采用 CART 算法，其不纯度（Gini 系数）定义如下：

$$\text{Gini} = 1 - \sum_{i=1}^{N} p_i^2,$$

其中 p_i 是每个类 i 的概率，N 是类别总数。信息增益定义为依某个特征进行分类前后的不纯度之差，选择使得信息增益最高的特征。

4.4 实验结论与分析

根据 TF-IDF 值对每一个 Descript 选取一个关键词，选取出现频率最高的 k 个关键词进行模型拟合，在测试集上的准确率如图 4.1 所示。可以看出决策树的训练与大部分频率较高的关键词并无较大帮助，部分频率较低的关键词反而可以很好地减少决策树的深度。我们选取所有关键词进行模型训练，最终决策树如图 4.2，训练集上的准确率达到了 100%。

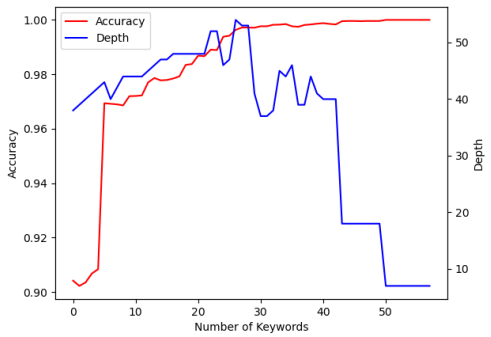

图 4.1: 决策树深度和测试集上的准确率和与关键词个数关系图

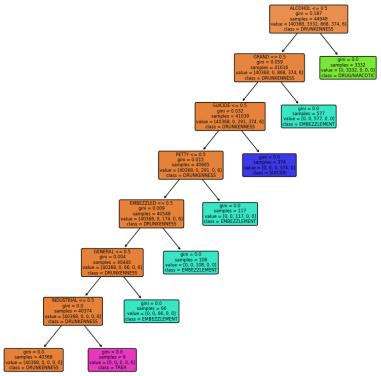

图 4.2: 决策树可视化图 (Depth=7)

4.5 实验代码

```
1  import os
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  from sklearn import preprocessing
5  from sklearn.metrics import accuracy_score
6  from sklearn.feature_extraction.text import TfidfVectorizer
7  from sklearn.tree import DecisionTreeClassifier, plot_tree
8
9
10 os.chdir(os.path.dirname(os.path.abspath(__file__)))  # 更改
       当前工作目录为脚本所在目录
11
12 train_path = "train.csv"
13 test_path = "test.csv"
14
15
16 def keyword_extract(path="train.csv"):
17     df = pd.read_csv(path)
18     vectorizer = TfidfVectorizer(stop_words="english",
          token_pattern=r"(?u)\b\w+\b")
19     tfidf = pd.DataFrame(
20         vectorizer.fit_transform(df["Descript"]).toarray(),
```

```
21          columns=vectorizer.get_feature_names_out(),
22      )
23      keywords = tfidf.idxmax(axis=1)
24      keywords = keywords.value_counts(normalize=True).index.
            str.upper().tolist()
25      return keywords
26
27
28  def Pre_Process(path, keywords):
29      df = pd.read_csv(path)
30
31      # 将Category进行整数编码
32      encoder = preprocessing.LabelEncoder()
33      crime_type_encode = encoder.fit_transform(df["Category"
            ])
34
35      # 将时间进行one-hot编码
36      hour = pd.to_datetime(df["Dates"]).dt.hour
37      hour = pd.get_dummies(hour)
38      day = pd.get_dummies(df["DayOfWeek"])
39
40      # 将所属警区进行one-hot编码
41      police_district = pd.get_dummies(df["PdDistrict"])
42
43      # 利用 TF-IDF 特征进行编码
44      matrix = pd.DataFrame(0, index=df.index, columns=
            keywords)
45      for keyword in keywords:
46          matrix[keyword] = df["Descript"].apply(
47              lambda x: True if keyword in x else False
48          )
49
50      # 将特征合并
51      data = pd.concat([hour, day, police_district, matrix],
            axis=1)
52      data["Crime type"] = crime_type_encode
53
```

```
54      # Feature names are only supported if all input features
            have string names
55      data.columns = data.columns.astype(str)
56      return data
57
58
59  keywords = keyword_extract()
60  acc = []
61  depth = []
62  for k in range(0, len(keywords) + 1):
63      train = Pre_Process(train_path, keywords[:k])
64      test = Pre_Process(test_path, keywords[:k])
65      # 训练模型
66      model = DecisionTreeClassifier()
67      model.fit(train.drop("Crime type", axis=1), train["
            Crime type"])
68      # 预测结果
69      y_pred = model.predict(test.drop("Crime type", axis=1))
70      acc.append(accuracy_score(test["Crime type"], y_pred))
71      depth.append(model.get_depth())
72
73  # plot
74  fig, ax1 = plt.subplots()
75  ax1.set_xlabel("Number of Keywords")
76  ax1.set_ylabel("Accuracy")
77  ax1.plot(range(0, len(keywords) + 1), acc, color="r", label
          ="Accuracy")
78  ax1.tick_params(axis="y")
79  # 创建第二个坐标轴共享x轴
80  ax2 = ax1.twinx()
81  ax2.set_ylabel("Depth")
82  ax2.plot(range(0, len(keywords) + 1), depth, color="b",
          label="Depth")
83  ax2.tick_params(axis="y")
84  # 显示图例
85  lines, labels = ax1.get_legend_handles_labels()
86  lines2, labels2 = ax2.get_legend_handles_labels()
```

```
87 ax1.legend(lines + lines2, labels + labels2, loc="upper
       left")
88 # plt.show()
89 plt.savefig("accuracy+depth.png")
90 # 绘制决策树
91
92 plt.figure(figsize=(10, 10))
93 plot_tree(
94     model,
95     filled=True,
96     rounded=True,
97     class_names=list(pd.read_csv(train_path)["Category"].
         unique()),
98     feature_names=list(train.drop("Crime type", axis=1).
         columns),
99 )
100 plt.savefig("decision_tree.png")
```

第五章 第五次上机实验（信用卡欺诈预测）

5.1 实验要求

- 给定信用卡欺诈数据集，判断是否存在欺诈
- 请使用 BP 神经网络或基于 delta 准则的感知机实现

5.2 数据分析与处理

测试集和训练集均已划分好，包括进行 PCA 处理后的 30 个特征，对其进行归一化处理。

5.3 实验步骤与原理

本实验采用 BP 神经网络，Input layer 是一个 30 维的数据，Hidden layer 设定为 30 维，Output layer 为 2 维标签数据，激活函数为 sigmoid 函数。

梯度下降法则为：

$$\Delta w = -\eta \bigtriangledown E(w)$$

其中 w 为模型参数，η 为学习率，$\bigtriangledown$ 为梯度算子，E 为总偏差。

下面我们利用反向传播，来计算 $\bigtriangledown E(w)$。总偏差定义为

$$E = \frac{1}{2}\sum_{c}\sum_{j}(y_{j,c} - d_{j,c})^2 \tag{5.1}$$

其中 c 是案例 (输入-输出对) 的索引, j 是输出单元的索引, y 是输出单元的实际值而 d 是目标值. 我们的 * 目标 * 变为寻找 $\text{argmin}_w E$. 不失一般性, 我们考虑单元 j 的输入 x_j 与输出 y_j, 总输入 x_j 定义为下层单元输出值的线性函数:

$$x_j = \sum_i y_i w_{ji} \tag{5.2}$$

其中 y_i 为第 i 个单元的输出, w_{ji} 表示连接第 i 个单元到第 j 个单元的参数. 每个单元

都有其输出值, 通过非线性函数 Sigmoid 函数定义

$$y_j = \frac{1}{1 + \mathrm{e}^{-x_j}} \tag{5.3}$$

按照反向传播的定义, 我们对 E 关于 y 求偏导. 固定 5.1 中的 c, 对单元 j 的输出 y_j 求偏导

$$\frac{\partial E}{\partial y_j} = y_j - d_j \tag{5.4}$$

再对 x_j 求偏导

$$\frac{\partial y_j}{x_j} = y_j(1 - y_j) \tag{5.5}$$

对 5.4 与 5.5 应用链式法则

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} = (y_j - d_j) \cdot y_j(1 - y_j) \tag{5.6}$$

再次利用链式法则与 5.2 对 w_{ji} 求偏导

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} \cdot \frac{\partial x_j}{w_{ji}} = \frac{\partial E}{\partial x_j} \cdot y_i \tag{5.7}$$

5.4 实验结论与分析

本实验的准确率为 99.20%，AUC 值为 0.9038（见图 5.1），可以认为模型分类效果优秀。

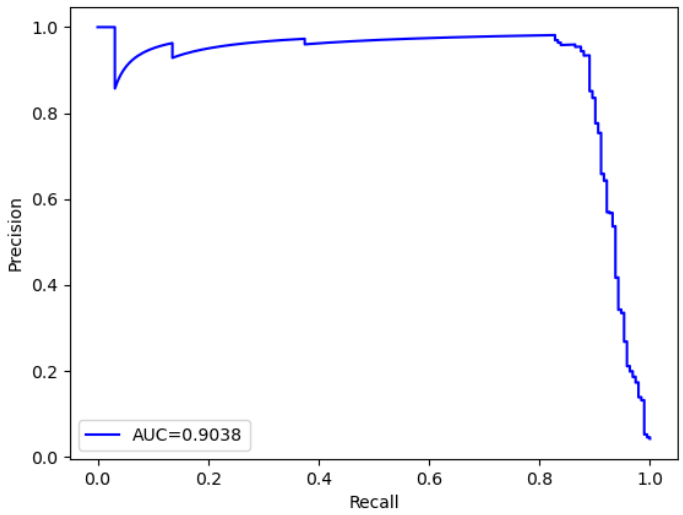

图 5.1: Recall-Precision 曲线图

5.5 实验代码

```
import os
import pandas as pd
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve, auc

os.chdir(os.path.dirname(os.path.abspath(__file__)))  # 更改当前工作目录为脚本所在目录
train_path = "train.csv"
test_path = "test.csv"

train = pd.read_csv(train_path, index_col=0)
test = pd.read_csv(test_path, index_col=0)


def Pre_process(path):
    data = pd.read_csv(path, index_col=0)
    X = data.drop("Class", axis=1).values
    y = data["Class"].values
    X = scaler.fit_transform(X)
    X = torch.from_numpy(X).float()
    y = torch.from_numpy(y).long()
    return X, y


class BpNet(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(BpNet, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.sigmoid = nn.Sigmoid()

```

```
34      def forward(self, x):
35          x = self.sigmoid(self.fc1(x))
36          x = self.fc2(x)
37          return x
38
39
40  scaler = StandardScaler()
41  X_train, y_train = Pre_process(train_path)
42  X_test, y_test = Pre_process(test_path)
43
44  seed = 233
45  epochs = 100
46  torch.manual_seed(seed)
47  model = BpNet(input_dim=X_train.shape[1], hidden_dim=30,
        output_dim=2)
48  criterion = nn.CrossEntropyLoss()
49  optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
50
51  for epoch in range(epochs):
52      model.train()
53      optimizer.zero_grad()
54      output = model(X_train)
55      loss = criterion(output, y_train)
56      loss.backward()
57      optimizer.step()
58      if epoch % 10 == 0:
59          print("Epoch: {}/{}, Loss: {:.4f}".format(epoch +
        1, epochs, loss.item()))
60
61  model.eval()
62  with torch.no_grad():
63      output = model(X_test)
64      pred = output.argmax(dim=1, keepdim=True)
65      correct = pred.eq(y_test.view_as(pred)).sum().item()
66      print("Accuracy: {:.2f}%".format(correct / len(y_test)
        * 100))
67
```

```
68 # AUC
69 y_score = output[:, 1].numpy()
70 precision, recall, _ = precision_recall_curve(y_test.numpy
       (), y_score)
71 area = auc(recall, precision)
72 print("AUC: {:.4f}".format(area))
73
74 # plot auc
75 plt.plot(recall, precision, label="AUC={:.4f}".format(area)
       , color="b")
76 plt.xlabel("Recall")
77 plt.ylabel("Precision")
78 plt.legend(loc="best")
79 # plt.show()
80 plt.savefig("auc.png")
```