南開大學
Nankai University

数据挖掘实验报告

学　　号:

姓　　名:

年　　级:

学　　院:

专　　业:

完成日期: 2021 年 3 月 27 日

目录

第一章　第一次上机实验（LBP 提取图像的纹理特征）

1.1　实验要求

- 1. 给定若干张图像，利用局部二值模式特征 (LBP) 对这些图像进行特征提取
- 2. 图象是 W * H * 3 的矩阵
- 3. 将最终提取到的特征通过 plot 的形式展示，绘制特征曲线图直观对比不同类图片纹理提取到的特征的不同
- 4. 使用 Python 编程实现

1.2　实验步骤与原理

1.2.1　LBP 特征的基本定义

局部二值模式（Local Binary Pattern, LBP）通过比较像素与其邻域像素的灰度关系，编码局部纹理的微结构。给定中心像素 g_c 及以其为中心、半径为 R 的圆形邻域上 P 个等角度采样点的灰度 $\{g_p\}_{p=0}^{P-1}$，标准 LBP 的定义为

$$\mathrm{LBP}_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)\, 2^p, \qquad s(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0, \end{cases}$$

其中邻域采样点坐标为

$$(x_p, y_p) = \big(x_c + R\cos(2\pi p/P),\ y_c - R\sin(2\pi p/P)\big),$$

本次实验只考虑以 g_c 为中心的九宫格的局部的 LBP 特征。

1.2.2 直方图特征

将整幅图像（或图像块）内的 LBP 代码统计为直方图作为纹理特征：

$$H[k] = \sum_{(x,y)} \mathbf{1}\{\mathrm{LBP}_{P,R}(x,y) = k\}, \qquad k \in \{0, \ldots, 2^P - 1\}.$$

常见做法是对直方图进行 ℓ_1 归一化以消除尺寸影响：

$$\hat{H}[k] = \frac{H[k]}{\sum_j H[j]}.$$

为表征空间布局，可将图像划分为 $M \times N$ 个网格单元，分别计算直方图并按行优先串接，得到最终特征向量。

1.2.3 实现细节（本实验手写 Python 要点）

1. **预处理：**彩色图像先转灰度；可选高斯平滑抑制噪声。
2. 按上述规则计算出图片的 LBP 特征直方图
3. **可视化：**使用 Matplotlib 绘制折线；多类对比时可叠加均值曲线与标准差带。

1.2.4 复杂度与并行优化

- 时间复杂度约为 $O(PWH)$，W, H 为图像宽高；P 通常较小，易于并行/向量化。
- 下面所呈现的代码采用串行方式计算 LBP 特征，但本人也给出了基于 cython 的并行加速版本。

加速计算技巧：

- 使用 cython 的 memoryview 接口直接操作 numpy.ndarray
- 在 cython 层开启 python 的 nogil 模式，绕开全局解释器锁，使用 OpenMP 实现并行计算

并行计算代码及各类计算方法的 benchmark 详见 https://github.com/flyingbucket/machinelearning/tree/main/LBP。

1.3 实验结果与分析

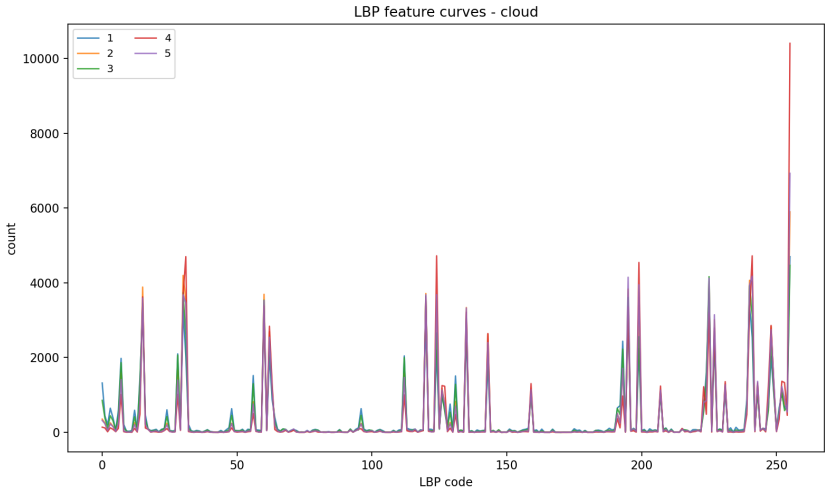

图 1.1: cloud LBP 特征曲线对比图

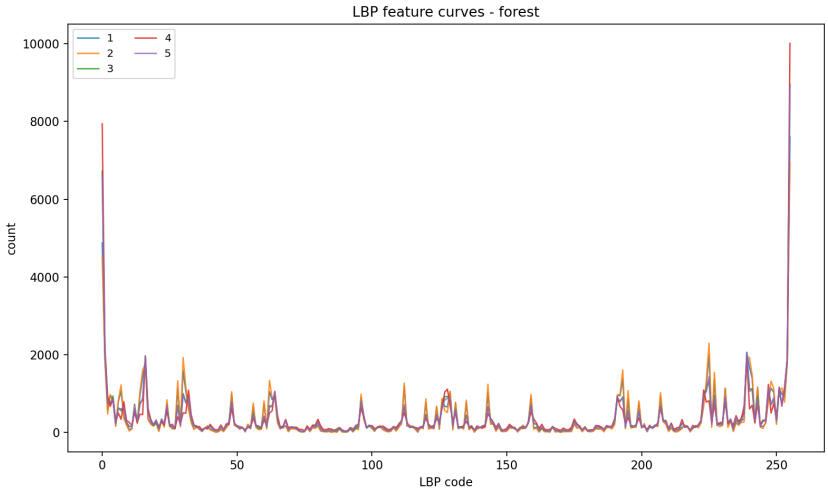

图 1.2: forestLBP 特征曲线对比图

1.4 实验代码

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3  from collections import Counter
4  from PIL import Image
5
```

```
6   class LBP:
7       @staticmethod
8       def _read_img(imPath: str, pad: int = 1, mode: str =
        "reflect") -> np.ndarray:
9           im = Image.open(imPath).convert("L")
10          arr = np.array(im)
11          padded = np.pad(arr, pad_width=((pad, pad), (pad,
         pad)), mode=mode)
12          return padded
13
14      @staticmethod
15      def LBPkernel(im: np.ndarray, x, y) -> int:
16          h, w = im.shape
17          assert x + 2 < h and y + 2 < w, (
18              f"Index out of bound,please check padding. x
        :{x},y:{y},h:{h},w:{w}"
19          )
20          patch = im[x : x + 3, y : y + 3].copy()
21          patch = (patch >= patch[1, 1]).astype(np.uint8)
22          idxs = [0, 1, 2, 5, 8, 7, 6, 3]
23          bits = patch.reshape(-1)[idxs]
24          val = int("".join(map(str, bits)), 2)
25          return val
26
27  def walk_dir(root_dir: str, out_dir: str = "EX1/outputs")
        :
28      root = Path(root_dir)
29      out = Path(out_dir)
30      out.mkdir(parents=True, exist_ok=True)
31      LBPcyExecutor = LBP()
32
33      for class_dir in sorted([p for p in root.iterdir() if
         p.is_dir()]):
34          hist_list = []
35          img_names = []
36          all_codes = set()
37          for img_path in sorted(class_dir.iterdir()):
```

```
38              try:
39                  res_dict = LBPcyExecutor(str(img_path))
        # {code: count}
40                  if not isinstance(res_dict, dict) or len(
        res_dict) == 0:
41                      print(f"[WARN] 空直方图：{img_path}")
42                      continue
43                  hist_list.append(res_dict)
44                  img_names.append(img_path.stem)
45                  all_codes.update(res_dict.keys())
46              except Exception as e:
47                  print(f"[WARN] 处理失败：{img_path} -> {e}
        ")
48
49          codes = sorted(all_codes)  # 所有出现过的 LBP code
50          X = []  # 每张图对齐后的频率向量
51
52          for h in hist_list:
53              vec = np.array([h.get(c, 0) for c in codes],
        dtype=np.float64)
54              X.append(vec)
55
56          plt.figure(figsize=(10, 6))
57          for vec, name in zip(X, img_names):
58              plt.plot(codes, vec, linewidth=1.2, alpha
        =0.85, label=name)
59          plt.xlabel("LBP code")
60          plt.ylabel("count")
61          plt.title(f"LBP feature curves - {class_dir.name}
        ")
62          plt.legend(ncol=2, fontsize=9, loc="best")
63          plt.tight_layout()
64
65          save_path = out / f"{class_dir.name}_lbp_curves.
        png"
66          plt.savefig(save_path, dpi=160)
67          plt.close()
```

```
68            print(f"[OK] Saved: {save_path}")
69
70    if __name__ == "__main__":
71        dir = "./EX1/data"
72        walk_dir(dir)
```

第二章 第二次上机实验

2.1 实验要求

- 1. 根据分类结果（result.csv）绘制 PR 曲线
- 2. 使用 Python 编程实现

2.2 数据分析与处理

数据分析

数据给出了分类器在测试集上的推理结果，包含两列，lable 和 pred

数据处理

将数据按照预测值递减排序

2.3 实验步骤与原理

2.3.1 原理说明

在二分类任务中，分类器的输出通常为一个介于 $[0,1]$ 之间的预测概率或置信度分数。通过设定不同的阈值（Threshold），可以将样本划分为正类或负类，从而得到不同的分类结果。

针对每一个阈值 θ，可计算以下指标：

- **真正例（TP）**：预测为正类且实际为正类的样本数；
- **假正例（FP）**：预测为正类但实际为负类的样本数；
- **假负例（FN）**：预测为负类但实际为正类的样本数；
- **真负例（TN）**：预测为负类且实际为负类的样本数。

由此可计算两个关键性能指标：

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}$$

当阈值从 1 逐渐减小到 0 时，Recall 通常单调递增，而 Precision 可能上升或下降。将各个阈值对应的 (Recall, Precision) 点连接起来，即得到 **Precision–Recall (PR) 曲线**。

PR 曲线反映了模型在不同阈值下的精确率与召回率的权衡关系，常用于评估类别分布不平衡的分类任务。曲线下的面积（AUC-PR）越大，说明模型整体性能越优。

2.3.2 实验步骤

1. **数据读取与排序**：使用 `pandas` 读取 `result.csv` 文件，并按照预测值 `pred` 从大到小排序；
2. **计算累计统计量**：
   - 通过布尔判断 `(label == 1)` 计算真正例的累计和（`tp_cumsum`）；
   - 通过 `(label == 0)` 计算假正例的累计和（`fp_cumsum`）；
3. **计算 Precision 与 Recall**：

$$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}, \qquad \text{Recall}_i = \frac{\text{TP}_i}{\text{TotalPos}}$$

   其中 `TotalPos` 为真实正样本总数。
4. **绘制 PR 曲线**：使用 `matplotlib` 将 Recall 作为横轴，Precision 作为纵轴，绘制曲线图；
5. **性能评估（可选）**：计算 PR 曲线下的面积（AUC–PR），作为模型整体性能指标。

2.3.3 实验意义

通过本实验，掌握了从分类结果计算 Precision–Recall 曲线的完整流程，理解了模型阈值调整对分类性能的影响，并熟悉了使用 Python 对实验结果进行可视化的基本方法。

2.4 实验结论与分析

根据给出的示例数据绘制出的 PR 曲线如图2.1所示：

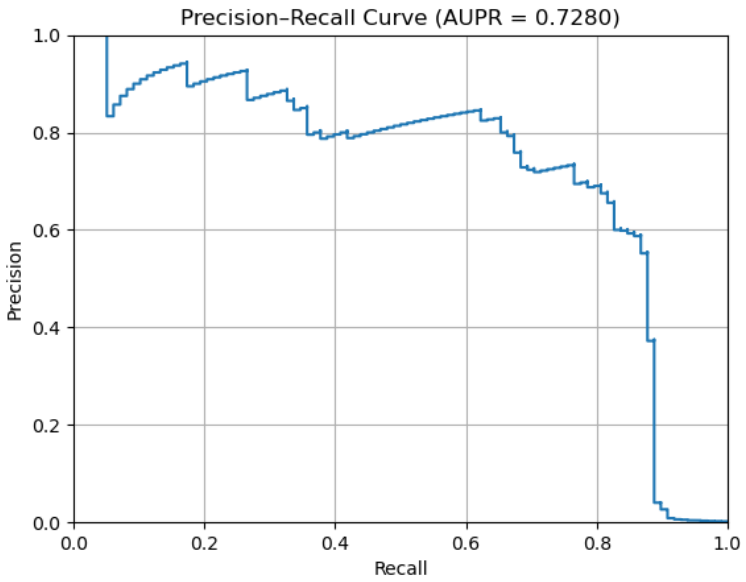

图 2.1: 示例数据的 PR 曲线

2.5 实验代码

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

data = pd.read_csv("./EX2/data/result.csv").sort_values(
    by="pred",ascending=False).reset_index(drop=True)

data["tp_cumsum"] = (data["label"] == 1).cumsum()
data["fp_cumsum"] = (data["label"] == 0).cumsum()

total_pos = (data["label"] == 1).sum()

data["precision"] = data["tp_cumsum"] / (data["tp_cumsum"
    ] + data["fp_cumsum"])
data["recall"] = data["tp_cumsum"] / total_pos

recall = np.r_[0.0, data["recall"].to_numpy()]
precision = np.r_[1.0, data["precision"].to_numpy()]

```

```
18 # 计算 AUPR (recall 单调增时可用梯形法则)
19   aupr = np.trapz(precision, recall)
20
21   plt.figure()
22   plt.step(recall, precision, where="post")
23   plt.xlabel("Recall")
24   plt.ylabel("Precision")
25   plt.title(f"Precision-Recall Curve (AUPR = {aupr:.4f})")
26   plt.xlim(0, 1)
27   plt.ylim(0, 1)
28   plt.grid(True)
29   plt.savefig("./EX2/PR_curve.png")
30   plt.show()
```

第三章 第三次上机实验

3.1 实验要求

1. 使用给定 sklearn 库中内置乳腺癌数据集，学习得到一个 Fisher 分类器 2. 对测试样本进行分类 3. 使用 Python 编程实现

3.2 数据分析与处理

本实验使用的是 `scikit-learn` 库提供的乳腺癌数据集。该数据集包含 569 个样本、30 个连续特征以及一个二分类标签（0 表示恶性，1 表示良性），数据字典中"data"和"target" 分别给出了特征和对应的分类标签。

将数据集按 8:2 的比例随机划分为训练集和测试集，

3.3 实验步骤与原理

本实验旨在基于乳腺癌数据集构建 Fisher 线性判别分类器，并对测试样本进行二分类评估。Fisher 判别的核心思想是寻找一个投影方向，使得不同类别在该方向上的类间距离最大、类内距离最小，从而实现最优线性可分。

3.3.1 实验原理

设原始数据为特征矩阵 $\mathbf{X} \in \mathbb{R}^{n\times d}$，标签向量为 y。Fisher 判别通过构造两类散度矩阵来刻画数据结构：

- **类内散度矩阵（Within-Class Scatter）**

$$\mathbf{S}_w = \sum_k \sum_{x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T,$$

其中 μ_k 为第 k 类的均值。该矩阵衡量同一类别样本的紧凑程度。

- **类间散度矩阵（Between-Class Scatter）**

$$\mathbf{S}_b = \sum_k n_k(\mu_k - \mu)(\mu_k - \mu)^T,$$

其中 μ 为全局均值，n_k 为类别 k 的样本数。该矩阵表示类别均值之间的分离程度。

Fisher 判别寻求一个最优投影方向 w，使得类间散度与类内散度之比最大，即：

$$w = \arg\max_w \frac{w^T\mathbf{S}_b w}{w^T\mathbf{S}_w w}.$$

这一优化问题, 利用拉格朗日乘子法，可转化为广义特征值问题：

$$\mathbf{S}_b w = \lambda \mathbf{S}_w w,$$

从中选取对应最大特征值的特征向量作为最优判别方向。

得到投影方向后，将训练样本投影到该方向，并计算各类别投影均值。类别之间的阈值取为相邻类别投影均值的中点，随后根据投影大小实现分类。

3.3.2 实验步骤

实验步骤如下：

1. **数据加载与标注**：导入乳腺癌数据集，获取特征矩阵与标签向量。
2. **数据划分**：采用随机划分方式，将数据按 8:2 的比例分为训练集与测试集。
3. **计算散度矩阵**：对训练集分别计算类内散度矩阵 $\mathbf{S}_w$ 和类间散度矩阵 $\mathbf{S}_b$。
4. **求解最优判别方向**：通过广义特征分解求得 Fisher 判别方向 w。
5. **生成分类阈值**：将训练样本投影到 w 上，根据各类别投影均值计算分类阈值。
6. **模型预测**：将测试集样本投影到 w 上，依据阈值进行类别判别。
7. **模型评估**：计算预测精度，评价 Fisher 分类器在测试集上的分类性能。

该流程完整实现了 Fisher 线性判别的训练与预测过程，验证了其在二分类任务中的有效性。

3.4 实验结论与分析

在完成 Fisher 线性判别分类器的训练与测试后，对测试集样本进行了分类评估。本实验采用准确率（Accuracy）与混淆矩阵（Confusion Matrix）作为主要性能指标。

3.4.1 分类准确率

在固定 numpy 随机种子为 42 的情况下，得到如下结果。
分类准确率为：

$$\text{Accuracy} = 97.37\%$$

该结果表明 Fisher 判别在乳腺癌二分类任务中具有良好的判别能力，能够在一维投影空间中实现有效的类别分离。

3.4.2 混淆矩阵分析

为了进一步观察分类器在不同类别上的预测情况，绘制其混淆矩阵如下：

$$\begin{pmatrix} 44 & 3 \\ 0 & 67 \end{pmatrix}$$

其中：

- 第一行表示真实标签为恶性（0）的样本，共 47 个，其中正确分类 44 个，误判为良性 3 个；
- 第二行表示真实标签为良性（1）的样本，共 67 个，全部被正确分类。

从矩阵可以看出，分类器对良性样本的识别效果好（无误判），对恶性样本也能保持较高的识别率。整体来看，该 Fisher 分类器在测试集上表现稳定，具有较强的泛化能力。但是存在将恶性样本误判为良性样本的情况，从错误代价的角度考虑，仍有较大的改进空间.

3.5 实验代码

```
1
2 """Fisher's Linear Discriminant Classifier Implementation""
        "
3
4 import scipy
5 import numpy as np
6 from sklearn.datasets import load_breast_cancer
7 from typing import List,Tuple
8
9 def get_Sw(X: np.ndarray, y: np.ndarray) -> np.ndarray:
```

```
10      """Coumpute Within-Class Scatter Matrix"""
11      labels = np.unique(y)
12      n_features = X.shape[1]
13      Sw = np.zeros((n_features, n_features))
14      for label in labels:
15          XOfLabel = X[y == label]
16          meanOfClass = np.mean(XOfLabel, axis=0)
17          diff = XOfLabel - meanOfClass
18          Sw += diff.T @ diff
19      return Sw
20
21
22  def get_Sb(X: np.ndarray, y: np.ndarray) -> np.ndarray:
23      """Compute Between-Class Scatter Matrix"""
24      overallMean = np.mean(X, axis=0)
25      labels = np.unique(y)
26      n_features = X.shape[1]
27      Sb = np.zeros((n_features, n_features))
28      for label in labels:
29          XOfLabel = X[y == label]
30          n_samplesOfLabel = XOfLabel.shape[0]
31          meanOfClass = np.mean(XOfLabel, axis=0)
32          meanDiff = (meanOfClass - overallMean).reshape(
        n_features, 1)
33          Sb += n_samplesOfLabel * (meanDiff @ meanDiff.T)
34      return Sb
35
36
37  def solve_fisher_direction(X: np.ndarray, y: np.ndarray) ->
        np.ndarray:
38      """Solve for Fisher's Linear Discriminant Direction"""
39      Sw = get_Sw(X, y)
40      Sb = get_Sb(X, y)
41
42      # Solve the generalized eigenvalue problem equation Sb*w =
        lambda*Sw*w
43      eigvals, eigvecs = scipy.linalg.eigh(Sb, Sw)
```

```
44      maxEigIndex = np.argmax(eigvals)
45      w = eigvecs[:, maxEigIndex]
46      return w
47
48
49  def split_data(X: np.ndarray, y: np.ndarray, train_ratio:
          float = 0.8):
50      """Split data into training and testing sets"""
51      n_samples = X.shape[0]
52      indices = np.arange(n_samples)
53      np.random.shuffle(indices)
54
55      train_size = int(n_samples * train_ratio)
56      train_indices = indices[:train_size]
57      test_indices = indices[train_size:]
58
59      X_train = X[train_indices]
60      y_train = y[train_indices]
61      X_test = X[test_indices]
62      y_test = y[test_indices]
63
64      return X_train, y_train, X_test, y_test
65
66
67  def train_fisher_classifier(X: np.ndarray, y: np.ndarray):
68      """Train Fisher Classifier"""
69      w = solve_fisher_direction(X, y)
70      projections = X @ w  # compute projection
71      labels = np.unique(y)
72      meanProjections = []
73      for label in labels:
74          meanProjections.append((label,np.mean(projections[y
            == label])))
75      sortedMeans = sorted(meanProjections,key=lambda x: x
          [1])
76      thresholds=[]
77      for i in range(len(sortedMeans)-1):
```

```
78          meanPrev=sortedMeans[i][1]
79          meanPost=sortedMeans[i+1][1]
80          thresholds.append((meanPrev+meanPost)/2)
81
82      return w,thresholds,sortedMeans
83
84
85  def predict_fisher_classifier(X: np.ndarray, w: np.ndarray,
            thresholds: list,sortedMeans:List[Tuple[int,float
        ]]) -> np.ndarray:
86      """Predict using Fisher Classifier"""
87      projections = X @ w
88      y_pred = np.zeros(projections.shape[0])
89      labels = [item[0] for item in sortedMeans]
90      for i, projection in enumerate(projections):
91          for j, threshold in enumerate(thresholds):
92              if projection < threshold:
93                  y_pred[i] = labels[j]
94                  break
95          else:
96              y_pred[i] = labels[-1]
97      return y_pred
98
99  def evaluate_classifier(y_true: np.ndarray, y_pred: np.
        ndarray):
100     """Evaluate Classifier Accuracy"""
101     accuracy = np.mean(y_true == y_pred)
102     return accuracy
103
104
105
106
107 if __name__ == "__main__":
108     np.random.seed(42)
109
110     data = load_breast_cancer()
111     X = data["data"]
```

```
112     y = data["target"]
113
114     X_train, y_train, X_test, y_test = split_data(X, y)
115
116     w,thresholds,sortedMeans = train_fisher_classifier(
            X_train, y_train)
117
118     y_pred = predict_fisher_classifier(X_test, w,
            thresholds,sortedMeans)
119
120     accuracy = evaluate_classifier(y_test, y_pred)
121     print(f"Fisher Classifier Accuracy: {accuracy * 100:.2f
            }%")
```

第四章 第四次上机实验

4.1 实验要求

4.2 数据分析与处理

4.3 实验步骤与原理

4.4 实验结论与分析

4.5 实验代码

第五章 第五次上机实验

5.1 实验要求

5.2 数据分析与处理

5.3 实验步骤与原理

5.4 实验结论与分析

5.5 实验代码