



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

1 第一次上机实验 (LBP 提取图像的纹理特征)	3
1.1 实验要求	3
1.2 实验步骤与原理	3
1.2.1 LBP 特征的基本定义	3
1.2.2 直方图特征	4
1.2.3 实现细节 (本实验手写 Python 要点)	4
1.2.4 复杂度与并行优化	4
1.3 实验结果与分析	5
1.4 实验代码	5
2 第二次上机实验	9
2.1 实验要求	9
2.2 数据分析与处理	9
2.3 实验步骤与原理	9
2.3.1 原理说明	9
2.3.2 实验步骤	10
2.3.3 实验意义	10
2.4 实验结论与分析	10
2.5 实验代码	11
3 第三次上机实验	13
3.1 实验要求	13
3.2 数据分析与处理	13
3.3 实验步骤与原理	13
3.4 实验结论与分析	13
3.5 实验代码	13
4 第四次上机实验	14
4.1 实验要求	14
4.2 数据分析与处理	14

4.3 实验步骤与原理	14
4.4 实验结论与分析	14
4.5 实验代码	14
5 第五次上机实验	15
5.1 实验要求	15
5.2 数据分析与处理	15
5.3 实验步骤与原理	15
5.4 实验结论与分析	15
5.5 实验代码	15

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. 图像是 $W * H * 3$ 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

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

其中邻域采样点坐标为

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

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：

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

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

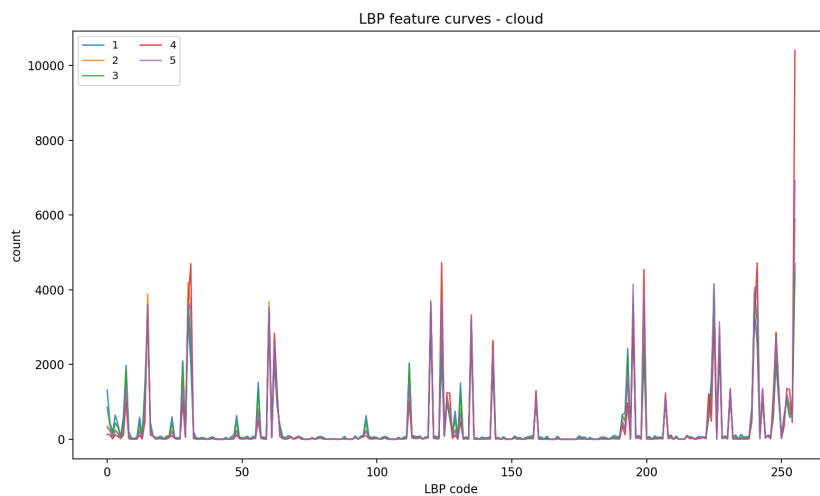


图 1.1: cloud LBP 特征曲线对比图

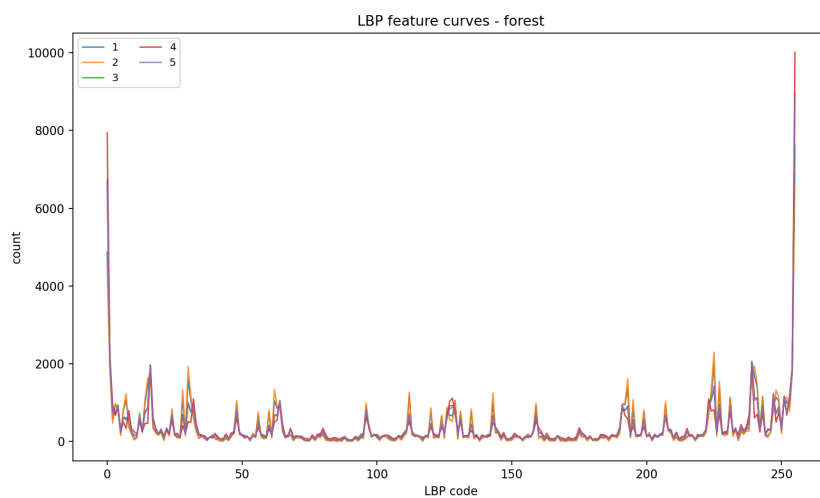


图 1.2: forestLBP 特征曲线对比图

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))
40             # {code: count}
41             if not isinstance(res_dict, dict) or len(
42                 res_dict) == 0:
43                 print(f"[WARN] 空直方图: {img_path}")
44                 continue
45                 hist_list.append(res_dict)
46                 img_names.append(img_path.stem)
47                 all_codes.update(res_dict.keys())
48             except Exception as e:
49                 print(f"[WARN] 处理失败: {img_path} -> {e}
50 ")
51
52     codes = sorted(all_codes) # 所有出现过的 LBP code
53     X = [] # 每张图对齐后的频率向量
54
55     for h in hist_list:
56         vec = np.array([h.get(c, 0) for c in codes],
57             dtype=np.float64)
58         X.append(vec)
59
60     plt.figure(figsize=(10, 6))
61     for vec, name in zip(X, img_names):
62         plt.plot(codes, vec, linewidth=1.2, alpha
63             =0.85, label=name)
64         plt.xlabel("LBP code")
65         plt.ylabel("count")
66         plt.title(f"LBP feature curves - {class_dir.name}
67 ")
68     plt.legend(ncol=2, fontsize=9, loc="best")
69     plt.tight_layout()
70
71     save_path = out / f"{class_dir.name}_lbp_curves.
72 png"
73     plt.savefig(save_path, dpi=160)
74     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：

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

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**:

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

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:

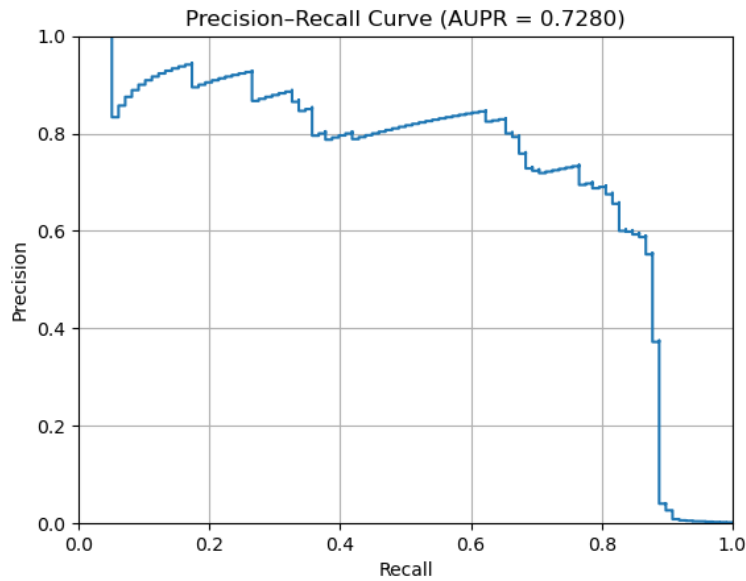


图 2.1: 示例数据的 PR 曲线

2.5 实验代码

```
1  import pandas as pd
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  data = pd.read_csv("./EX2/data/result.csv").sort_values(
        by="pred", ascending=False).reset_index(drop=True)
6
7  data["tp_cumsum"] = (data["label"] == 1).cumsum()
8  data["fp_cumsum"] = (data["label"] == 0).cumsum()
9
10 total_pos = (data["label"] == 1).sum()
11
12 data["precision"] = data["tp_cumsum"] / (data["tp_cumsum"]
        + data["fp_cumsum"])
13 data["recall"] = data["tp_cumsum"] / total_pos
14
15 recall = np.r_[0.0, data["recall"].to_numpy()]
16 precision = np.r_[1.0, data["precision"].to_numpy()]
17
```

```
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

3.2 数据分析与处理

3.3 实验步骤与原理

3.4 实验结论与分析

3.5 实验代码

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