实验一 数据降维与分类任务

1 问题描述

分别利用 PCA 和 LDA 降维技术对葡萄酒数据进行降维处理，在降维后的数据集上训练和测试 logistic 回归分类器，并比较降维技术前后分类器准确率的变化。

2 实现步骤与流程

PCA 降维

求解样本的散布矩阵

$$\mathbf{S}(\boldsymbol{x}) = \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu}_x)(\boldsymbol{x}_i - \boldsymbol{\mu}_x)^{\mathrm{T}}$$

求解散布矩阵最大的 β 个特征值对应的特征向量作为基向量进行投影

$$\mathbf{S}(\boldsymbol{x})\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i \qquad i = 1,\ 2,\ \cdots,\ k$$

投影后的样本特征向量

$$\boldsymbol{y} = \mathbf{W}^{\mathrm{T}}\boldsymbol{x} \Rightarrow \mathbb{R}^{\beta} = \mathbb{R}^{\beta \times \alpha}\mathbb{R}^{\alpha}$$

其中变换矩阵 $\mathbf{W}$ 的列向量即为散布矩阵的特征向量 $\boldsymbol{w}_i$

LDA 降维

求解类内散布矩阵

$$\mathbf{S}_w(\boldsymbol{x}) = \sum_{i=1}^{c} \mathbf{S}_i = \sum_{i=1}^{c} \sum_{\boldsymbol{x} \in \mathcal{X}_i} (\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathrm{T}}$$

求解类间散布矩阵

$$\mathbf{S}_b(\boldsymbol{x}) = \sum_{i=1}^{c} n_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^{\mathrm{T}}$$

如果类内散布矩阵可逆，投影的 β 个基向量满足

$$\mathbf{S}_w^{-1}\mathbf{S}_b\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i \qquad i = 1,\ 2,\ \cdots,\ k$$

并且对应分别对应最大的 β 个特征值 λ_i。如果类内散布矩阵不可逆，可以将其替换为

$$\mathbf{S}_w \leftarrow \mathbf{S}_w + \epsilon \mathbf{I}_\beta, \quad \epsilon > 0$$

投影后的样本特征向量

$$\boldsymbol{y} = \mathbf{W}^{\mathrm{T}} \boldsymbol{x} \Rightarrow \mathbb{R}^\beta = \mathbb{R}^{\beta \times \alpha} \mathbb{R}^\alpha$$

其中变换矩阵 $\mathbf{W}$ 的列向量即为矩阵 $\mathbf{S}_w^{-1}\mathbf{S}_b$ 的特征向量 $\boldsymbol{w}_i$

logistic 回归

logistic 回归模型

$$\hat{y} = \sigma(z) = \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b)$$

其中 $\sigma(\cdot)$ 代表 sigmoid 函数，对于 logistic 回归可以使用两种损失函数

- 交叉熵损失

$$\ell_{cross-entropy} = -\sum_{i=1}^{n} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

- 负对数似然

$$\ell_{log-likelihood} = \sum_{i=1}^{m} \left[-y_i \hat{y}_i + \log(1 + e^{\hat{y}_i}) \right]$$

本次实验中采取前者进行实现。

3 实验结果与分析

由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取开源机器学习算法库 sklearn 与实验中的算法进行对比。sklearn 是一个开源的基于 python 语言的机器学习工具包。它通过 numpy, scipy 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

PCA 降维

经过实验算法以及 sklearn 算法的 PCA 降维后的样本如下图所示，可以看出，降维后的样本具有较高的散布程度。并且样本经过实验的 PCA 算法降维后与 sklearn 的 PCA 算法相比拥有相同的性态，但二者在某一特征维度上呈镜像对称。

(a) PCA（实验）　　(b) PCA（sklearn）

图 1: PCA 降维效果图

LDA 降维

由于 LDA 限制降维维度不大于类别数减 1，为了达到可视化的效果，本次实验仍然将样本降维到二维经过实验算法降维后的样本如下图所示，可以看出，降维后的样本具有相对较好的可分性。但无法展示 sklearn 的 LDA 降维效果图

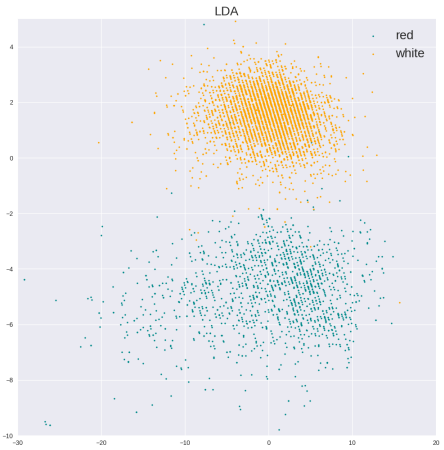


图 2: LDA 降维效果图

3.1 logistic 回归

实验中实现的 logistic 回归在原始数据集、PCA 降维数据集以及 LDA 降维数据集上的训练过程如后图所示，分别展示了模型训练过程中损失函数以及分类准确率的变化。

从图像中可以看出，原始数据集的模型准确率居中，PCA 降维后的数据集的模型准确率较差，LDA 降维后的数据集的模型准确率最佳。

相应地，sklearn 提供的 logistic 回归模型在各个数据集上的准确率如下图所示，与实验算法相同的是，LDA 的准确率最高，原始数据次之，PCA 最低。

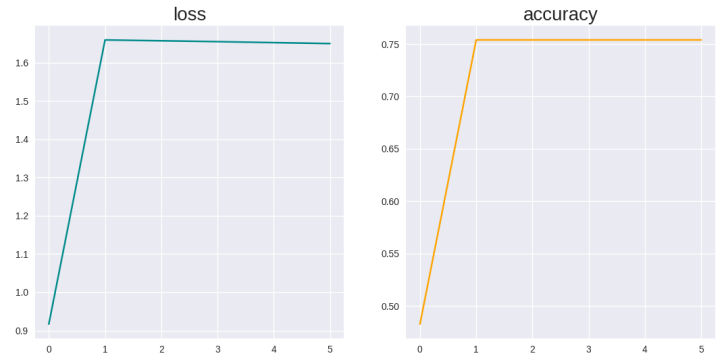
4 MindSpore 学习使用心得体会

由于本次实验并未采用 MindSpore，此部分用 sklearn 的学习使用心得体会来代替。

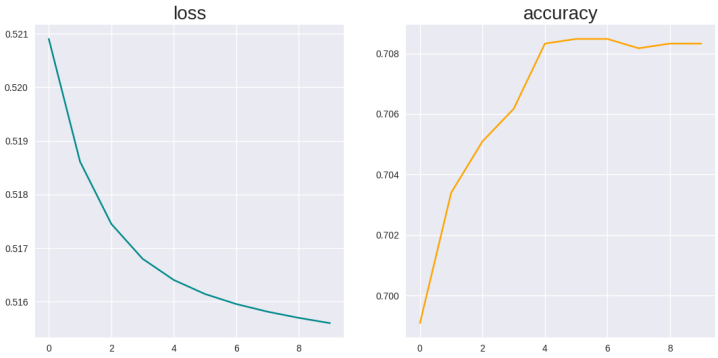
本次实验中调用了 sklearn 中的以下算法接口

- PCA（decomposition PCA）
- LDA（discriminant_analysis LinearDiscriminantAnalysis）
- logistic 回归（linear_model LogisticRegression）

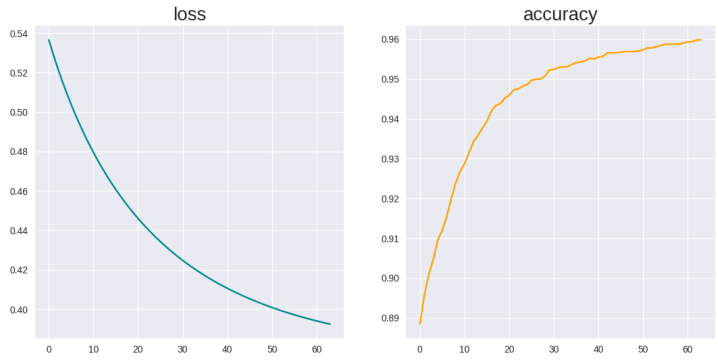
作为实验算法的对照算法，sklearn 提供的算法接口方便调用，并且执行效率较高，能够高效地实现实验的需求。



(a) 训练历史（原始）



(b) 训练历史（PCA）



(c) 训练历史（LDA）

图 3: logistic 回归（实验）

```
(python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验1$
origin data accuracy: 0.9795290133907958
PCA data accuracy: 0.9307372633523164
LDA data accuracy: 0.9947668154532862
```

图 4: logistic 回归（sklearn）

5 代码附录

mine.py M

实验1 > src > mine.py > ...

```
            axs[0].plot(history['loss'], color='darkcyan')
            axs[1].plot(history['acc'],  color='orange')
            plt.show()

    if __name__ == '__main__':
        import warnings
        warnings.filterwarnings('ignore')

        plt.style.use('seaborn')

        # *************** origin ******************** #

        wine_data = MyWineData(args['dataroot'])
        model, history = wine_data.classify(
            lr=0.01,
            decay=0.97,
            iterations=32,
            patience=5
        )
        wine_data.render_history(history)

        # ****************** PCA ********************* #

        wine_data = MyWineData(args['dataroot'])
        wine_data.decompose(PCA, 2)
        wine_data.render_data(
            'PCA',
            (-150, 200),
            (-100, 50)
        )
        model, history = wine_data.classify(
            lr=0.001,
            decay=0.97,
            iterations=64,
            patience=5
        )
        wine_data.render_history(history)

        # ****************** LDA ********************* #

        wine_data = MyWineData(args['dataroot'])
        wine_data.decompose(LDA, 2)
        wine_data.render_data(
            'LDA',
            (-30, 20),
            (-10, 5)
        )
        model, history = wine_data.classify(
            lr=0.01,
            decay=0.97,
            iterations=64,
            patience=5
        )
        wine_data.render_history(history)
```

logistic.py M

实验1 > src > logistic.py > ...

```
class LogisticRegression(object):

    def __init__(self, input_dim, scale=0.1) -> None:
        self.w = scale * np.random.randn(input_dim)
        self.b = scale * np.random.randn()

    def predict(self, data, logit=True):
        y_pred = np.matmul(data, self.w)
        y_pred = y_pred + self.b
        if logit:
            y_pred = sigmoid(y_pred)
        else:
            y_pred = y_pred > 0
        return y_pred

    def fit(
            self,
            data,
            label,
            lr,
            decay,
            iterations,
            patience,
            epsilon=0.001
    ):
        history = dict()
        history['loss'] = []
        history['acc']  = []
        n = data.shape[0]
        best_acc = 0.
        tolerance = 0
        for i in range(iterations):
            y_pred = self.predict(data)
            positive = label * (1 / (y_pred + epsilon))
            negative = (1 - label) * (1 / (y_pred - 1 - epsilon))
            grad = -(positive + negative)
            grad = grad * y_pred * (1 - y_pred)
            grad_w = np.matmul(data.T, grad) / n
            grad_b = np.sum(grad) / n
            self.w -= lr * grad_w
            self.b -= lr * grad_b
            positive = label * np.log(y_pred + epsilon)
            negative = (1 - label) * np.log(1 - y_pred + epsilon)
            loss = -np.sum(positive + negative) / n
            y_pred = y_pred > 0.5
            acc = np.sum(y_pred == label) / n
            if acc > best_acc:
                tolerance = 0
                best_acc = acc
            else:
                tolerance += 1
            if tolerance >= patience:
                break
            lr *= decay
```

图 5: 部分代码截图