实验二 KNN 分类任务

1 问题描述

通过 KNN 算法对 Iris 鸢尾花数据集中的测试集进行分类，分别使用欧式距离以及马氏距离作为度量实现，同时实现基于马氏距离的度量学习算法 NCA

2 实现步骤与流程

KNN 算法

KNN 算法选取训练集中与未见演本的度量最小的 k 个样本中类别众数作为预测值

NCA 算法

样本的散布矩阵

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

马式距离

$$d_M(\boldsymbol{x}_i,\ \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x_i} - \boldsymbol{x}_j)^{\mathrm{T}}\mathbf{S}^{-1}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

可学习的马式距离（伪马式距离）

$$d_M(\boldsymbol{x}_i,\ \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x_i} - \boldsymbol{x}_j)^{\mathrm{T}}\mathbf{M}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

其中，$\mathbf{M}$ 是半正定矩阵，重写以上距离数学形式

$$d_M(\boldsymbol{x}_i,\ \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$$

其中变换矩阵 $\mathbf{A} \in \mathbb{R}^{k\times d}$ ，并且 $k \geq rank(\mathbf{M})$。设样本 $\boldsymbol{x}_i$ 的近邻分布为

$$p_{ij} = \exp\{-d_M^2(\boldsymbol{x}_i,\ \boldsymbol{x}_j)\} \Big/ Z_i$$

$$Z_i = \sum_{k\neq i}\exp\{-d_M^2(\boldsymbol{x}_i,\ \boldsymbol{x}_k)\}$$

代表样本 $\boldsymbol{x}_j$ 与样本 $\boldsymbol{x}_i$ 属于相同类别的概率。针对变换矩阵 $\mathbf{A}$ 最大化同类样本相似程度

$$\max_{\mathbf{A}} f(\mathbf{A}) = \sum_{i=1}^{n} \sum_{j \in \Omega_i} p_{ij}$$

其中 Ω_i 表示和样本 $\boldsymbol{x}_i$ 属于同一类别的其它样本的集合。令

$$d_{ij} = d_M^2(\boldsymbol{x}_i,\ \boldsymbol{x}_j)$$

准则函数对变换矩阵 $\mathbf{A}$ 求偏导数得

$$\frac{\partial p_{ij}}{\partial d_{ik}} = \begin{cases} -p_{ij}(1-p_{ij}) & k = j \\ \\ p_{ij}p_{ik} & k \neq j \end{cases}$$

$$\frac{\partial d_{ik}}{\partial \mathbf{A}} = 2\mathbf{A}(\boldsymbol{x}_i - \boldsymbol{x}_k)(\boldsymbol{x}_i - \boldsymbol{x}_k)^{\mathrm{T}}$$

$$\frac{\partial p_{ij}}{\partial \mathbf{A}} = 2p_{ij}\mathbf{A}\left[\sum_{k \neq i} p_{ik}(\boldsymbol{x}_i - \boldsymbol{x}_k)(\boldsymbol{x}_i - \boldsymbol{x}_k)^{\mathrm{T}} - (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}}\right]$$

$$\frac{\partial f}{\partial \mathbf{A}} = \sum_{i=1}^{n} \sum_{j \in \Omega_i} \frac{\partial p_{ij}}{\partial \mathbf{A}}$$

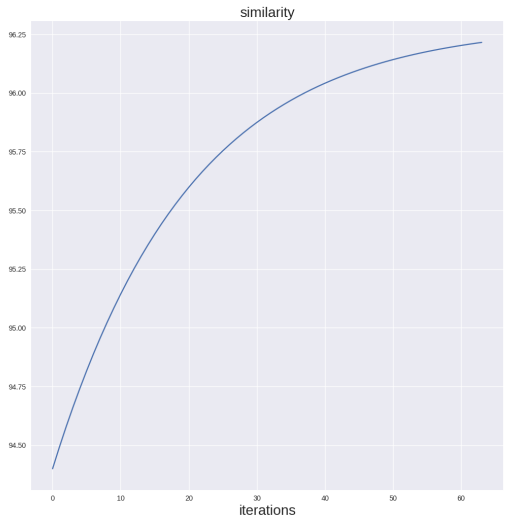
利用梯度下降法即可学习到一个较优的变换矩阵，使得变换后的同类样本距离，异类样本远离。但注意到该优化问题非凸，意味着最终收敛得到的解可能是局部最优解，并且与迭代初始值密切相关，实际实验中需要多次尝试才能得到满足要求的解。

3 实验结果与分析

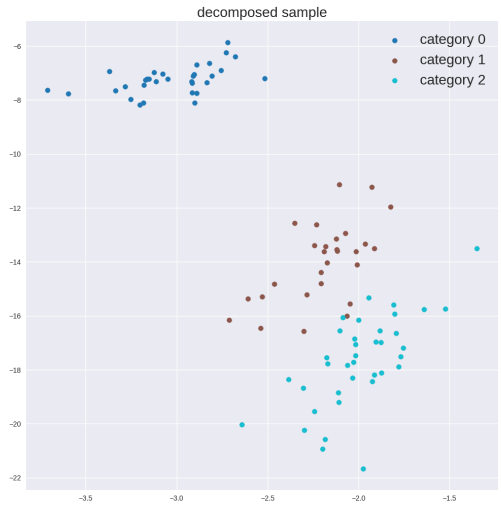
由于实验要求的 MindSpore 框架相关的资料较为匮乏，因此本次实验采取开源机器学习算法库 sklearn 与实验中的算法进行对比。sklearn 是一个开源的基于 python 语言的机器学习工具包。它通过 numpy, scipy 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

NCA 算法

以下是 NCA 算法的度量学习历史以及最终收敛的模型的变换矩阵对样本进行降维后的分布图



(a) 度量学习历史



(b) 降维样本分布

图 1: NCA 算法

观察可得，随着模型的迭代，同类样本的相似度逐渐提高，这点也可以从降维样本上看出

KNN 算法

以下是实验 KNN 算法（欧式距离、马氏距离）与 sklearn 的 KNN 算法的在验证集上的准确率对比

```
(python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验2$
euclidean accuracy: 1.0
100%
mahalanobis accuracy: 1.0
```

(a) KNN（实验）

```
(python) wzh@wzh:~/workspace/course/pattern recognition/实验/实验2$
euclidean accuracy: 1.0
```

(b) KNN（sklearn）

图 2: KNN 算法准确率

经过实验，采用 5 近邻在验证集上可以达到最高的准确率 100%，但是由于样本数量较少，因此结果只能在一定程度上作为参考

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

- KNN（neighbors KNeighborsClassifier）

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。

5 代码附录

mine.py knn.py

实验2 > src > knn.py > ...

```python
import numpy as np
from tqdm import tqdm

class KNN(object):

    def __init__(self, solver:str):
        self.solver = solver

    def fit(
            self,
            data,
            label,
            **kwargs
    ):
        self.data  = data
        self.label = label
        self.c = np.max(label) + 1
        if self.solver == 'euclidean':
            return
        index_split = [
            np.where(label == i)[0]
            for i in range(self.c)
        ]
        n = data.shape[0]
        input_dim = data.shape[-1]
        output_dim = kwargs['transform_dim']
        self.A = 0.75 * np.random.randn(
            input_dim,
            output_dim
        )
        iterations = kwargs['iterations']
        lr = kwargs['lr']
        decay = kwargs['decay']
        history = []
        for iteration in tqdm(range(iterations)):
            tmp = np.matmul(data, self.A)
            tmp = tmp[:, np.newaxis, :] - \
                  tmp[np.newaxis, :, :]
            d = np.sum(tmp ** 2, axis=-1)
            z = np.sum(np.exp(-d), axis=-1, keepdims=True)
            z = z - 1
            p = np.exp(-d) / (z + 0.001)
            grad = np.zeros_like(self.A)
            similarity = 0.
            for i in range(n):
                index_i = index_split[label[i]]
                tmp = np.zeros((input_dim, input_dim))
                for k in range(n):
                    if k == i: continue
                    tmp += p[i][k] * np.outer(
                        data[i] - data[k],
                        data[i] - data[k]
                    )
                for j in index_i:
                    similarity += p[i][j]
```

scikit-learn.py

实验2 > src > scikit-learn.py > ...

```python
import numpy as np
import pandas as pd
from sklearn.neighbors import\
KNeighborsClassifier as KNN
from data import IrisData

args = dict()
args['dataroot'] = r'./data/'
args['predict'] = r'./prediction/'

class SciIrisData(IrisData):

    def __init__(self, dataroot):
        super().__init__(dataroot)

    def euclidean(self, k):
        model = KNN(k)
        data = self.train[:, :-1]
        label = self.train[:, -1].astype(np.int32)
        model.fit(data, label)
        data = self.val[:, :-1]
        label = self.val[:, -1].astype(np.int32)
        y_pred = model.predict(data)
        acc = np.sum(y_pred == label)
        acc /= data.shape[0]
        return model, acc

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

    iris = SciIrisData(args['dataroot'])
    k = 5

    model, acc = iris.euclidean(k)
    print('euclidean accuracy: {}'.format(acc))
    y_pred = model.predict(iris.test)
    pd.DataFrame(y_pred).to_csv(
        args['predict'] + 'task3_test_prediction.csv',
        index=False, header=['label']
    )
```

图 3: 部分代码截图