模式识别大作业报告

高童　计 43　2014011357

一、问题描述

本次大作业是一个手写数字识别问题，是一个经典的分类问题。输入图像是 32×32 的黑白手写数字图像，目标是将其分类到 $0 \sim 9$ 的十类中。

根据题目提示，实验要点包括：

- 构造受限玻尔兹曼机 RBM，以图像为输入，对输出单元用传统分类器进行分类；
- 图像格式不一，因此需要进行预处理；
- 由于原始图像有 $32 \times 32 = 1024$ 个像素，将全部像素作为输入单元会导致输入单元数量过多，RBM 训练过慢，因此需要通过预处理降低输入单元数量。

二、代码设计

1. 整体设计

整体代码用 Python 实现。

代码预处理部分的框架如下。首先是读入图片，分别经过的函数分别包括：

1. `Image.open()`。用 PIL 库打开图片。
2. `np.array()`。将打开的图片转换为 `numpy` 格式多维数组。
3. `singleTunnel()`。如果图片有超过 1 个通道，将所有通道上的值平均，返回一个灰度图像。
4. `compressImg()`。降低输入单元的函数。

示例代码如下：

```
# load second directory
trainDigitPath = trainPath + 'hjk_picture'
p = Path(trainDigitPath)
for f in p.iterdir():
    if f.suffix in ['.png', '.jpg']:
        print(f.name)
        x = np.array(Image.open(f))
        y = compressImg(singleTunnel(x))
        I.append(x)
        X.append(y)
        if f.name[1] == '_':
            Y.append(int(f.name[0]))
        else:
            Y.append(int(f.name[2]))
```

具体分类和测试的部分框架如下[1]。首先读入训练数据和标签，并将训练数据归一化到 0—1 区间。

```
X_train = digits['data']
Y_train = digits['target']
X_train = (X_train - np.min(X_train, 0)) / (np.max(X_train, 0) + 0.0001)
```

随后构造一个 Pipeline，设置相关参数，并在其上进行训练：

```
rbm = BernoulliRBM(random_state=0, verbose=True)
logistic = linear_model.LogisticRegression()
classifier = Pipeline(steps=[('rbm', rbm), ('logistic', logistic)])

# Hyper-parameters.
rbm.learning_rate = 0.04
rbm.n_iter = 20
rbm.n_components = 10000
```

[1]参考 Scikit-learn 网站的教程，标题为 Restricted Boltzmann Machine features for digit classification，网址http://scikit-learn.org/stable/auto_examples/neural_networks/plot_rbm_logistic_classification.html。

```
logistic.C = 5000.0

print("Training")
classifier.fit(X_train, Y_train)
```

最后，利用 scikit-learn 自带的函数生成测试报告：

```
print()
print("Logistic regression using RBM features:\n%s\n" % (
    metrics.classification_report(
        Y_test,
        classifier.predict(X_test))))
```

2. 降低输入单元

为了降低 RBM 的输入单元数量，采用的策略如下：将图像在横、竖两个方向上分别隔一个像素取一个像素，这样图像大小就变味原来的四分之一，总共只剩 $16 \times 16 = 256$ 个像素。在后来的测试中，可以发现这样不会降低分类的准确率。

相关函数代码如下：

```
def compressImg(x):
    [r, c] = [32, 32]

    comp = np.empty([math.floor(r/2), math.floor(c/2)])
    for i in range(0, comp.shape[0]):
        for j in range(0, comp.shape[1]):
            comp[i, j] = x[2*i, 2*j];

    return comp.flatten()
```

三、测试结果

在给定的训练集和测试集上，分类效果比较好。此处着重讨论降维程度在不同 RBM 规模下对分类效果的影响。

在限定参数：学习率 =0.04，循环次数 =20 的情况下，分别改变输入单元压缩的程度和隐藏单元的数量，观察分类正确率和所耗时间，结果列于表 1中。

表 1: 各个参数下的正确率和程序运行时间

输入参数	正确率	所耗时间
1024 维，内单元 3000	97%	149.8 秒
256 维，内单元 3000	98%	47.8 秒
100 维，内单元 3000	78%	26.3 秒
256 维，内单元 10000	100%	165.2 秒
100 维，内单元 10000	83%	92.6 秒

可以看出，在图像大小变为原来 1/4 时，模型训练时间明显减少，但正确率没有下降。在运行时间相同的情况下，1/4 大小输入时，内单元数可以增加到约 10000，对模型有更好的拟合能力。但当图像大小变为原来的约 1/9 时，由于原图信息丢失，准确率下降较多。

综上可知，每次图像都缩大小为原来 1/4 可以作为类似数据集上的标准预处理程序。在不同的训练集上，都可以做类似的测试，寻找缩小原图的最好比例。

四、实验结论和收获

RBM 是理论上可以达到全局最优的一种模型。但由于 RBM 采用类似模拟退火的随机策略，收敛比较慢。为加速 RBM 收敛，采用合适方法降低输入单元数是必要的。

本次实验中，我亲身体会了降低输入单元数对分类器性能的巨大改善，加深了对 RBM 的理解，也从改善分类器中获得了成就感。感谢陶老师和助教这一学期的指导。