机器学习大作业报告

吕恒磊
2018K8009926007

一．作业要求：

使用 Naïve Bayes, SVM, AdaBoost 三种学习方法分别对 MNIST 和 SST-2 数据集进行学习预测。

二．Naïve Bayes + MNIST

1. 数据处理

1) 首先解析图像数据，获得 `train_X, test_X, train_y, test_y` 四个数组，分别代表训练集数据（元素为像素灰度）、测试集数据、训练集标签、测试集标签。

2) 调用 `cv2.boundingRect` 方法剪裁图像的空白边框，留下中间有效信息，再 `resize` 成(20,20)的矩阵，然后 `flatten` 成长度为 400 的数组。此时 `train_X, test_X` 特征数量为 400.

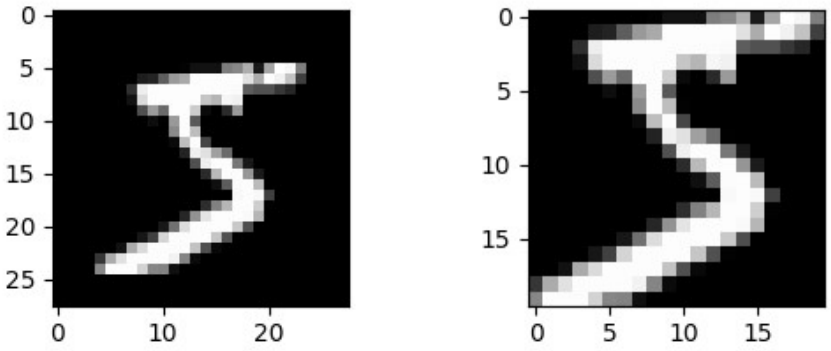

图 1 原图像（左）和剪切后图像（右）

3) （可选）将训练集和测试集数据二值化，灰度值大于 127 的置为 1，小于 127 的置为 0. 这一步是否执行根据选择概率模型是 Gaussian 还是 Bernouli 确定。这里选择的是 Gaussian 分布。

2. 模型设计和实现

根据 Naïve Bayes 原理，特征为x情况下标签为y_k的概率为

$$P(y_k|x) = \frac{P(y_k)\prod_{i=1}^{n} P(X_i|y_k)}{\sum_k P(y_k)\prod_{i=1}^{n} P(X_i|y_k)} \propto P(y_k)\prod_{i=1}^{n} P(X_i|y_k)$$

可以看出预测标签需要计算类别频率$P(y_k)$和后验概率$P(X_i|y_k)$

由于 Naïve Bayes 方法可以计算给定特征下各标签概率，于是可以直接实现一个多分类器。后验概率的计算需要选择一个概率模型。这里我选择了 Gaussian 分布，于是需要计算出各标签下各特征值的均值和方差，根据高斯分布计算后验概率。预测时计算每个测试数据 0~9 标签的概率，取最大值的索引为预测标签值。以下是计算均值和方差的代码。

```
for i in range(10):
    label[i] = train_y == i
means = label @ Train_X / N_train
```

```
variances = label @ np.multiply(Train_X, Train_X) / N_train - \
                    np.multiply(means, means)
```

3. 实验结果

```
$ python src/MNIST_NB.py
Loading data...
Complete!
Calculating means and variances...
Complete!
Predicting...
Complete!
Accuracy: 0.6726
total time: 12.032315492630005 seconds
(tensorflow)
```

图 2 NB+MNIST 运行结果截图

测试准确率: 0.6726

4. 结果分析:

Naïve Bayes 方法原理简单，实现较为容易，计算量也较小，运行时间短，准确率较低。

三．Naïve Bayes + SST-2

1. 数据处理

1) 首先解析文本数据，调用 readlines 方法获得文本每一行字符串的列表，调用 rstrip 函数去除末端无效字符，得到每句话组成的列表和各句话标签。

2) 调用 sklearn.feature_extraction.text.CountVectorizer 里的 fit_transform 方法将文本列表矩阵化。如下代码所示，train_X, test_X 分别为训练集和测试集的共现矩阵。

```
#load dictionary, including train data and test data
list_all = list1+list2
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(list_all)
#load train data / test data
train_X = X.toarray()[0:N_train,:]
test_X = X.toarray()[N_train:,:]
```

3) 将特征（词典数量）进行降维：剔除那些在正例和反例文本中出现频率接近的词汇。频率接近指在某类文本出现频率介于另一类文本频率的 0.5~2 倍之间。如下代码所示。

```
delete = ((doc_freq2 / 2 < doc_freq1).astype("int8") + \
          (doc_freq1 < doc_freq2 * 2).astype("int8")) == 2
reserve_id = np.where(delete == 0)[0]
train_X = train_X[:,reserve_id]
test_X = test_X[:,reserve_id]
```

2. 模型设计和实现

总体模型和 MNIST 数据集上的 NB 类似，不同的是由于 SST 数据集的共现矩阵很稀疏且元素数值小，概率模型选择的是多项式而不是高斯模型，直接计算出正负标签下各特征值(单词)出现的频率作为后验概率。预测时计算每个测试数据正负标签的概率，取大者为预测标签值。以下是计算频率的代码。

```
for i in range(2):
    prob[i] = ((train_y == i) @ train_X + 1) / (train_y == i).sum()
log_prob = np.log(prob)
```

3. 实验结果

```
$ python src/SST2_NB.py
Loading data...
Complete!
Complete!
Start predicting...
Complete!
Accuracy:0.8130733944954128
total time: 77.48062181472778 seconds
```

图 3 NB+SST-2 运行结果截图

测试准确率: 0.8131

4. 结果分析:

Naïve Bayes 方法原理简单，实现较为容易，计算量也较小，运行时间短。

四．SVM + MNIST

1. 数据处理

和 NB + MNIST 处理方式一样

2. 模型设计和实现

SVM 算法详细推导就不描述了，其实现流程如下:

1) 选择惩罚参数 C > 0，求解凸二次规划问题:

$$\underbrace{min}_{\alpha}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\boldsymbol{x_i}\cdot\boldsymbol{x_j})-\sum_{i=1}^{N}\alpha_i$$

$$s.t.\ \sum_{i=1}^{N}\alpha_i\, y_i = 0, 0 \le \alpha_i \le C, i = 1,2,\dots,N$$

求解如上凸二次规划用 smo 算法相对快速且有效。推导过程过于复杂，不在此展示，详细见代码。kernel 函数选择线性 kernel。

2) 计算$\boldsymbol{w}$和b，构造分类决策函数$f(x) = sign(\boldsymbol{w}\cdot\boldsymbol{x^T}+b)$。

这样就实现了一个二分类器。MNIST 数据集需要实现 10 分类，由于数据量庞大，采用 one vs one 方法，针对两两标签分别构造二分类器，共构造 45 个分类器，最终分类结果由这 45 个分类器投票选出。

3. 实验结果

```
0.8674
Building model time: 187.77399706840515 seconds
Predicting time: 2.4828336238861084 seconds
Total time: 200.8187072277069 seconds
```

图 4 SVM+MNIST 运行结果截图

测试准确率: 0.8674

4. 结果分析:

SVM 方法原理较为复杂，实现较难，计算量偏高，运行时间较长，准确率较高。

五．SVM + SST-2

1. 数据处理

1) 首先解析文本数据，调用 readlines 方法获得文本每一行字符串的列表，调用 rstrip 函数去除末端无效字符，得到每句话组成的列表和各句话标签。

2) 调用 sklearn.feature_extraction.text.CountVectorizer 里的 fit_transform 方法将文本列表矩阵化。如下代码所示，train_X, test_X 分别为训练集和测试集的共现矩阵。

```
#load dictionary, including train data and test data
list_all = list1+list2
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(list_all)
#load train data / test data
train_X = X.toarray()[0:N_train,:]
test_X = X.toarray()[N_train:,:]
```

3) 将特征（词典数量）进行降维：剔除那些在正例和反例文本中出现频率接近的词汇。频率接近指在某类文本出现频率介于另一类文本频率的 0.1~10 倍之间，相比 NB 算法，剔除范围增加了，这是为了减少训练开销。如下代码所示。

```
delete = ((doc_freq2 / 10 < doc_freq1).astype("int8") + \
          (doc_freq1 < doc_freq2 * 10).astype("int8")) == 2
reserve_id = np.where(delete == 0)[0]
train_X = train_X[:,reserve_id]
test_X = test_X[:,reserve_id]
```

2. 模型设计和实现

SVM 二分类器的实现和 SVM+NB 里的二分类器一样。由于 SST 特征维数过多，降维后也有 7000 多词汇，只能减少训练集数量来加快训练时间，我只选取了训练数据前 400 句话作为训练集，在 10000 条测试数据上进行测试。

3. 实验结果

```
accuracy: 0.6473
Building model time: 1706.4598815441132 seconds
Predicting time: 1.1974873542785645 seconds
Total time: 1710.285197019577 seconds
```

图 4 SVM + SST-2 运行结果截图

测试准确率: 0.6473

4. 结果分析:

SVM 算法原理复杂，计算开销较大，应用于文本分类时，应该进行适当预处理，将文本特征降维至可接受范围。由于时间紧张，没有仔细学习特征提取与降维的方法，只能采取减少训练集数量的下策来减少开销，训练结果较为一般。

六．Adaboost + MNIST

1. 数据处理
   1) 解析并剪裁图像数据，和 NB+MNIST 中处理方法一样。
   2) 将训练集和测试集数据二值化，灰度值大于 127 的置为 1，小于 127 的置为 0. 做这一步是因为 adaboost 需要进行多轮弱分类器预测，二值化训练集后采用 bernouli 概率分布能减少计算开销。
2. 模型设计和实现

由于 Naïve Bayes 方法计算开销小，选择其作为弱分类器。不同于简单 NB 方法时选择 Gaussian 分布作为概率分布，此处将像素特征二值化，选择 Bernouli 分布。

值得注意的是，MNIST 数据集需要多分类，而 NB 弱分类器本身可以作为多分类器，于是可以采用 Multi-class AdaBoost 算法。其与标准二分类 AdaBoost 算法区别在于，计算分类器权重 alpha 方法不同:

$$\alpha^{(m)} = log\frac{1 - err^{(m)}}{err^{(m)}} + \log(K-1)$$

其中 K 是类型数。具体实现如下代码所示。

```
for i in range(iter_num):
    print("iter "+str(i))
    weighted_X = np.multiply(np.mat(d).T,Train_X)
    prob[i] = label @ weighted_X + 1e-100
    prob[i] = np.log(prob[i])
    for j in range(10):
        train_predict[j] = np.array(np.log(freq[j]) + \
                 np.multiply(prob[i,j],Train_X==1).sum(axis=1))
    for j in range(N_train):
        train_predict_y[j] = np.where(train_predict[:,j] == \
                 np.max(train_predict[:,j]))[0][0]
    error_rate = 1 - d[np.where(train_predict_y == train_y)[0]].sum()
    alpha[i] = np.log((1-error_rate)/error_rate) + np.log(9)
    d_ = np.multiply(d, np.exp(-alpha[i] * \
         ((train_y == train_predict_y).astype(int) - \
         (train_y != train_predict_y).astype(int))))
    d = d_ / d_.sum()
```

3. 实验结果

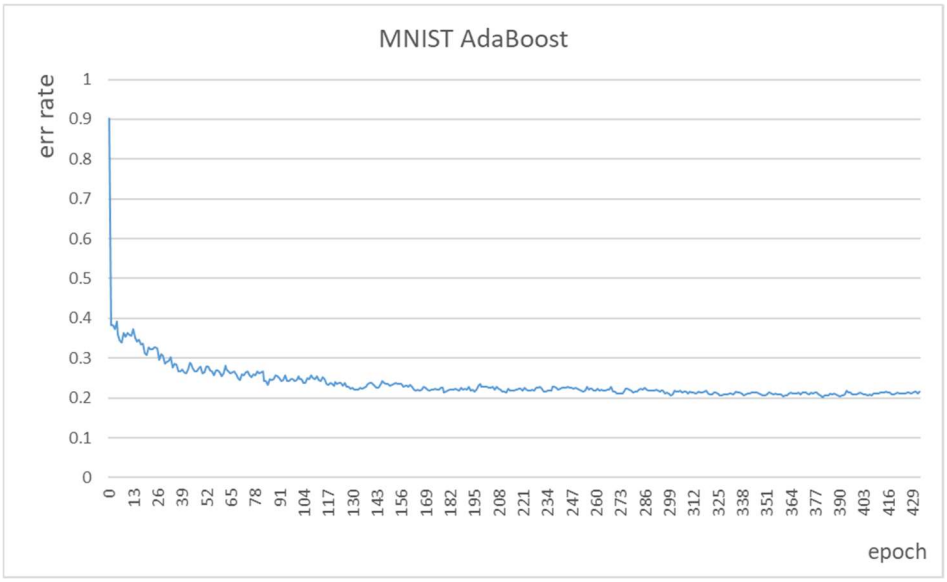

图 6 AdaBoost+MNIST 运行结果截图

最终测试准确率: 0.79 左右

结果保存在 results/MNIST_Adaboost_test_error_rate.txt 中

4. 结果分析:

随着轮数增加，AdaBoost 算法准确率趋于定值，这与基础弱分类器的准确性有关。由于将图像数据二值化处理，NB 弱分类器的准确度只有 0.61 左右，所以 AdaBoost 是能显著提高弱分类器的准确度的。如果用 SVM 作为弱分类器，结果应该更好，但是训练开销也会对应增大许多。

七．Adaboost + SST-2

1. 数据处理

和 NB+SST-2 中处理方法一样

2. 模型设计和实现

同样选择 Naïve Bayes 作为弱分类器。不同于 MNIST 数据集，SST-2 数据集只需要二分类，于是采用标准的二分类 AdaBoost 算法。

具体实现如下代码所示。

```
for i in range(iter_num):
    print("iter "+str(i))
    weighted_X = np.multiply(np.mat(d).T,train_X) * N_train
    tj =(np.vstack((train_y,train_y)) == np.mat([0,1]).T)
    prob[i] =(tj @ weighted_X + 1) / tj.sum(axis=1)
    prob[i] = np.log(prob[i])
    for j in range(2):
        train_predict[j] = np.multiply(train_X, prob[i,j]).sum( \
                           axis = 1).transpose()
    for j in range(N_train):
        train_predict_y[j] = np.where(train_predict[:,j] == \
             np.max(train_predict[:,j]))[0][0]
```

```
error_rate = 1 - d[np.where(train_predict_y == train_y)[0]].sum()
alpha[i] = 0.5 * np.log((1-error_rate)/error_rate)
d_ = np.multiply(d, np.exp(-alpha[i] * \
     ((train_y == train_predict_y).astype(int) - \
     (train_y != train_predict_y).astype(int))))
d = d_ / d_.sum()
```

---

3. 实验结果

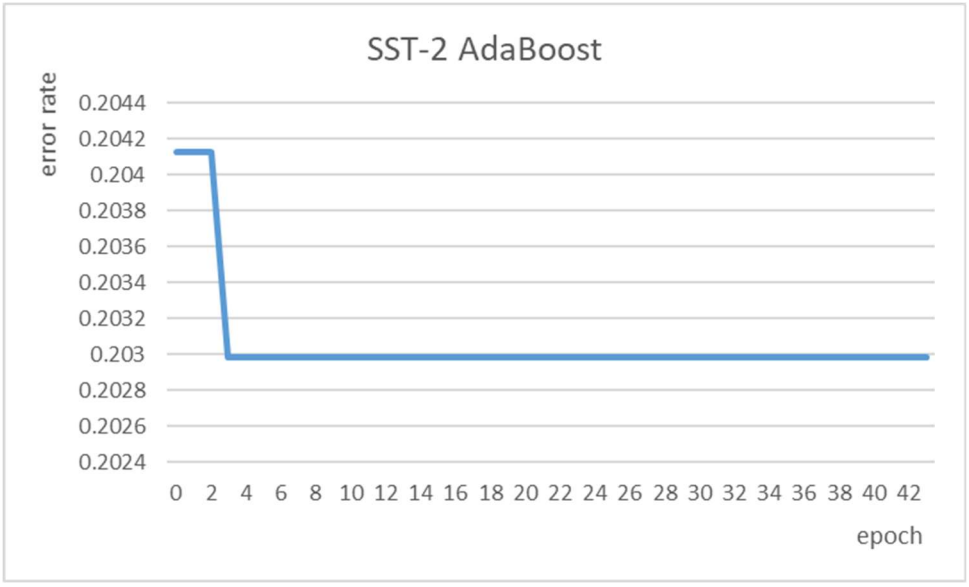

图 6 AdaBoost+MNIST 运行结果截图

最终测试准确率: 0.807 左右

4. 结果分析:

SST 数据集上 AdaBoost 运行得不是很好。只在第二轮有微弱的效果，往后训练集 err 趋近于 0.5，alpha 值趋近于 0，意味着后面层数的弱分类器趋于无效化。这有可能是本身弱分类器的局限性造成的。

八．引用文献

[1] https://zhuanlan.zhihu.com/p/29212107 机器学习算法实践-SVM 中的 SMO 算法

[2] Ji Zhu.et.al, Multi-class AdaBoost.
https://web.stanford.edu/~hastie/Papers/samme.pdf

[3] 周志华. 机器学习. 清华大学出版社