机器学习大作业报告

吕恒磊

2018K8009926007

1. 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。

1. Naïve Bayes + MNIST
2. 数据处理
3. 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。
4. 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.

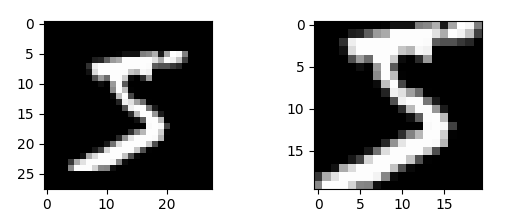


图 1 原图像（左）和剪切后图像（右）

1. （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，特征为情况下标签为的概率为

可以看出预测标签需要计算类别频率和后验概率

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)

——————————————————————————————————

1. 实验结果

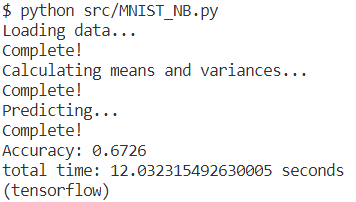


图 2 NB+MNIST运行结果截图

测试准确率: 0.6726

1. 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。

1. Naïve Bayes + SST-2
2. 数据处理
3. 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。
4. 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:,:]

——————————————————————————————————

1. 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]

——————————————————————————————————

1. 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)

————————————————————————————————————

1. 实验结果

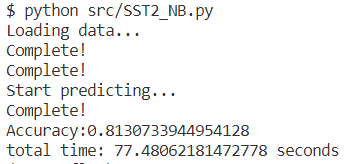


图 3 NB+SST-2运行结果截图

测试准确率: 0.8131

1. 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。

1. SVM + MNIST
2. 数据处理

和NB + MNIST处理方式一样

1. 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：

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。

1. 计算和，构造分类决策函数。

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。

1. 实验结果

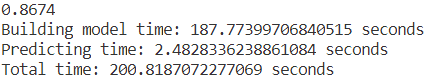


图 4 SVM+MNIST运行结果截图

测试准确率: 0.8674

1. 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。

1. SVM + SST-2
2. 数据处理
3. 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。
4. 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:,:]

——————————————————————————————————

1. 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]

——————————————————————————————————

1. 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。

1. 实验结果

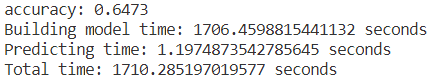


图 5 SVM + SST-2运行结果截图

测试准确率: 0.6473

1. 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。

1. Adaboost + MNIST
2. 数据处理
3. 解析并剪裁图像数据，和NB+MNIST中处理方法一样。
4. 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。
5. 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：

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()

———————————————————————————————————————

1. 实验结果

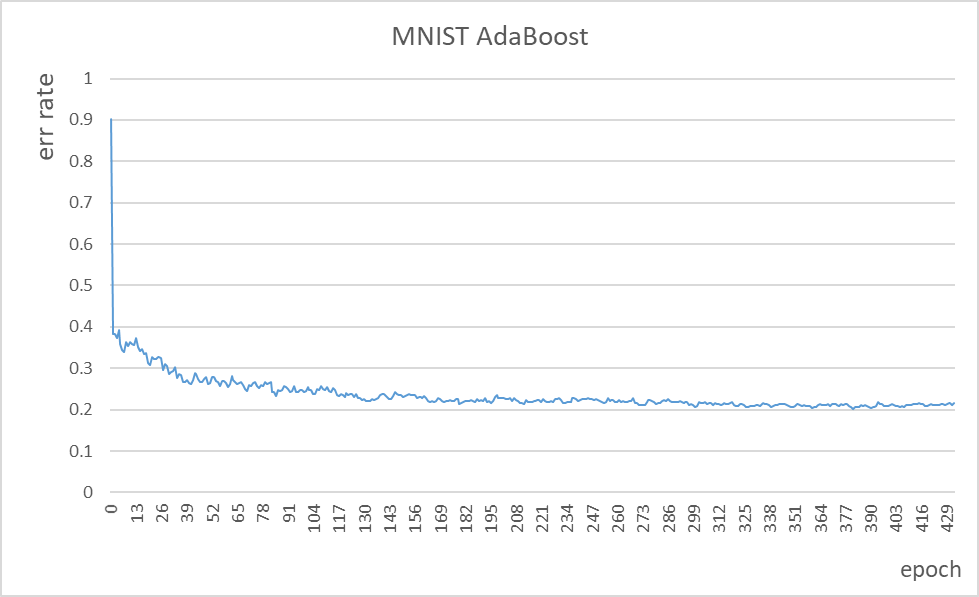


图 6 AdaBoost+MNIST运行结果截图

最终测试准确率: 0.79左右

结果保存在results/MNIST\_Adaboost\_test\_error\_rate.txt中

1. 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。

1. Adaboost + SST-2
2. 数据处理

和NB+SST-2中处理方法一样

1. 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()

———————————————————————————————————————

1. 实验结果

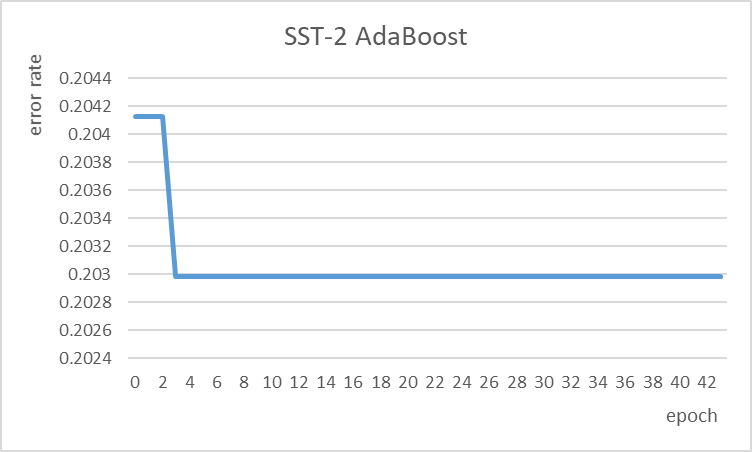


图 6 AdaBoost+MNIST运行结果截图

最终测试准确率: 0.807左右

1. 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。

1. 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