机器学习第一次实验报告——KNN 分类器

March 19, 2018

1 问题描述

本次实验的目标为用 KNN 实现一个手写数字识别的分类器。利用提供的训练集，分别计算在 k=1, k=3,k=5, k=10 的情况下分类器在测试集上的正确率。并且在训练集上划分交叉验证集，利用交叉验证集选择 K 值，绘制出横轴为 K 值，纵轴为交叉验证集上错误率的曲线。

2 解决方案

2.1 KNN 原理介绍

KNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。

2.2 KNN 具体实现

定义函数 kNN_classifier($inX, dataset, labels, k$) 实现 kNN 分类器，其中 inX 为待分类的向量，$dataset$ 为训练集，$labels$ 为训练集样本的标签，k 为 kNN 分类算法中的 k 值。

```
def kNN_classifier(inX, dataset, labels, k):
    sample_num=dataset.shape[0]
    diff= np.tile(inX,(sample_num,1))-dataset
    sqr_mat= diff**2
    sqr_dis= sqr_mat.sum(axis=1)
    #dis is a vector which stores the distance between the inX and each of the training set
    dis=sqr_dis**0.5
    #argsort assendingly sorts the distance and  return the corresponding array of indice
    sorted_index=dis.argsort()
    results=np.array(np.zeros(10))
    for i in range(k):
        #class_i is a vector which stands for the label of the i-th nearest vector'''
        class_i= labels[sorted_index[i]]
        results+=class_i
    #the biggest element indicates the corresponding index appears most times
    #print results
    res= results.argmax()
    return res
```

$sample\_num$ 为训练集样本数量，将行向量 inX 复制 $sample\_num$ 次后与训练集做差，结果矩阵按元素取平方，调用函数 sqr_mat.sum($axis = 1$) 按行求和后按元素开方得到列向量 dis，dis 的每个元素为测试样本与每个训练样本的距离，调用函数 argsort() 将 dis 由小到大排序并返回对应的索引值向量 $sorted\_index$。对于 $sorted\_index$ 中的前 k 个索引值，在标签向量 $labels$ 中找到对应的标签向量 $class\_i$，将 k 个标签向量累加得到结果向量 $results$，$results$ 中数值最大的元素对应的下标 res 就应该是 inX 对应的类别。

2.3 k 值确定

kNN 算法中，k 值的选择影响结果的精确程度，因此分别测试 k 在 1，3，5，10 的情况下，kNN 算法在测试集上的准确率。

2.3.1 数据转换

由于原始数据为 CSV 格式，因此定义函数 createData($data, datasets, labels$) 将原始数据转换为 Python 可以处理的数据

```
def createData(data,datasets,labels):
    for line in data:
        line=line[0]
```

```
        datas= line.split(' ')
        datas_num=len(datas)
        float_datas=[]
        label_datas=[]
        for i in range(datas_num-11):
            float_datas.append(float(datas[i]))
        datasets.append(float_datas)
        for i in range(datas_num-11,datas_num-1):
            label_datas.append(int(datas[i]))
        labels.append(label_datas)
```

参数中 *data* 为原始数据，*datasets* 为转换后的数据集，*labels* 为 *datasets* 对应的标签。由于原始数据为字符串格式，需将函数转换为浮点格式。

2.3.2 计算错误率

转换数据后，在不同的 k 值时计算测试集在训练集上的错误率。

```
for k in [1,3,5,10]:
    right_num=0;
    for i in range(test_datasets.shape[0]):
        #print kNN_classifier(test_datasets[i],train_datasets,train_labels,1),test_labels[i].argmax()
        if kNN_classifier(test_datasets[i],train_datasets,train_labels,k)==test_labels[i].argmax():
            right_num+=1
    #print right_num
    #print test_datasets.shape[0]
    print "The classification error ratio for k= %d is %.4f"%(k, round(1-float(right_num)/float(test_da
```

对于每个 k 值，*right_num* 记录测试集上准确结果的数量，对于测试集的每个数据 $test\_datasets[i]$ 调用 kNN_classifier$(test_datasets[i], train_datasets, train_labels, k)$，若返回的类别与实际类别相等，*right_num* 递增 1，最后除以测试集样本总数即为该 k 值下的正确率。

对应的输出结果为

```
The classification error ratio for k= 1 is 0.1444
The classification error ratio for k= 3 is 0.1611
The classification error ratio for k= 5 is 0.1674
The classification error ratio for k= 10 is 0.1611
```

2.3.3 交叉验证集

在测试集划分出 20% 的数据作为交叉验证集，计算在不同的 k 值下在交叉验证集下的正确率

```
#the number of element in the cross validation set
all_num= train_datasets.shape[0]
cv_num=int(all_num*0.2)
#print type(cv_num)
get_index=range(0,all_num);
#generate the index of element chosen for the cross validation set
np.random.shuffle(get_index);
#print get_index
train_index= get_index[0:cv_num]
val_index=get_index[cv_num:all_num]
#print train_index
#print val_index
#extract the training data for the cross validation
cv_train=train_datasets[train_index,:]
cv_train_labels=train_labels[train_index,:]
#extract the validation data for the cross validation
cv_val=train_datasets[val_index,:]
cv_val_labels=train_labels[val_index,:]
```

变量 *cv_num* 为交叉验证集的大小，*get_index* 为训练集样本数大小的数组，利用 np.random.shuffle(*get_index*) 随机扰乱数组并抽取前 *cv_num* 个元素作为交叉验证集的数组下标，得到训练集数据 *cv_train*，训练集样本标签 *cv_train_labels*，交叉验证集数据 *cv_val*，交叉验证集样本标签 *cv_val_labels*。

然后对于 0 到 10 的 k 值，即可调用 $kNN\_classifier(cv\_val[i], cv\_train, cv\_train\_labels, k)$ 计算不同 k 值时在交叉验证集上的正确率，具体步骤与在测试集上计算正确率类似。

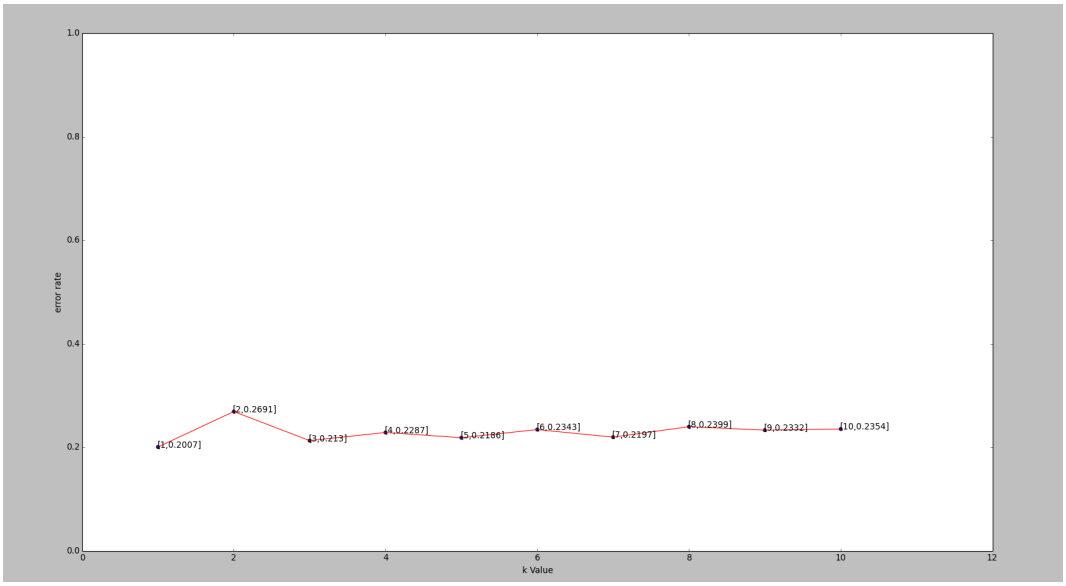
```
for k in range(1,11):
    right_num= 0
    temp_res=0
    for i in range(len(cv_val)):
        if kNN_classifier(cv_val[i],cv_train,cv_train_labels,k)==cv_val_labels[i].argmax():
            right_num+=1
    temp_res = round(1-float(right_num)/float(len(cv_val)),4)
    cv_res.append(temp_res)
    print "The classification error ratio for k= %d is %.4f"%(k,temp_res )
```

对应输出为

```
The classification error ratio for k= 1 is 0.2040
The classification error ratio for k= 2 is 0.2948
The classification error ratio for k= 3 is 0.2422
The classification error ratio for k= 4 is 0.2646
The classification error ratio for k= 5 is 0.2612
The classification error ratio for k= 6 is 0.2713
The classification error ratio for k= 7 is 0.2657
The classification error ratio for k= 8 is 0.2747
The classification error ratio for k= 9 is 0.2758
The classification error ratio for k= 10 is 0.2881
```

最后调用 *matplotlib.pyplot* 的函数将 k 与错误率的对应关系表示出来

```
plt.figure('cv_figure')
plt.ylim(0,1)
plt.xlabel('k Value')
plt.ylabel('error rate')
for i in range(1,11):
   ann_str= '['+str(i)+','+str(cv_res[i-1])+']'
   plt.annotate(ann_str,xy=(i,cv_res[i-1]),xytext=(i,cv_res[i-1]))
plt.plot(range(1,11),cv_res,'r')
plt.scatter(range(1,11),cv_res)
plt.savefig('result.png')
plt.show()
```



3 实验分析

本次实验用 kNN 算法实现了一个分类器，分别用测试集计算准确率并在训练集上划分出交叉验证集测试准确率，试验结果发现，不同的 k 值对最终准确率的影响不是很大，基本稳定在 80% 左右。kNN 算法的优点是实现简单，无需估计参数，无需训练。但缺点也很明显，主要是进行分类时计算量大，要扫描全部训练样本计算距离，内存开销大，不适用于样本数太多的数据。同时由于最后确定类别只是简单的选择出现次数最多的类别，没有考虑近邻的距离的远近，实际距离更近的近邻更应该决定最终的分类，所以可以采用加权投票法来进一步提升准确率。