一、 实验题目，

1. 用 kNN 实现分类和回归的预测

二、 实验内容

1. 算法原理

kNN 是指，找到测试数据的 k 个最近邻，然后从中得到测试数据的预测。

分类一般找的是 k 个最近邻的众数，而回归则可以用加权值计算。

我们可以从词汇中得到 one_hot、TF 以及 TF_IDF 矩阵，这三种矩阵都可以用来代表文本，实现距离计算。

距离的计算方式有我们常用的欧式距离，还有街区距离以及余弦距离等等。

预测时候，k 的取值也有不用的取值。

2. 伪代码

分类：

1. 计算距离：
   欧氏距离(testX 是测试样本,dataSet 是训练集)
   ① testX 扩展成和 dataSet 一样行数的矩阵
   ② testX 矩阵减去 dataSet
   ③ 矩阵中每个数做平方
   ④ 每一行相加，然后开方
2. 找到邻居：
   ① 距离按照升序排序，得到索引
3. 实现分类：
   ① 前 k 个邻居分类
   ② 按照每一类含有样本个数降序排列
   ③ 有最多样本的一类就是预测类

回归：

1. 计算距离：
   欧氏距离(testX 是测试样本,dataSet 是训练集)
   ① testX 扩展成和 dataSet 一样行数的矩阵
   ② testX 矩阵减去 dataSet
   ③ 矩阵中每个数做平方
   ④ 每一行相加，然后开方
2. 找到邻居：
   ① 距离按照升序排序，得到索引
3. 实现回归：
   ① 前 k 个邻居的概率分别乘以它们距离的倒数，相加
   ② 处理得到的新概率，使得六种情感总和为 1

3. 关键代码截图（带注释）

Classification

```python
# 进行kNN计算
def kNN(testX, dataSet, labels, k):

    # 计算测试样本与训练样本集的距离
    dataMat = (tile(testX, (dataSet.shape[0], 1)) - dataSet) ** 2
    distances = dataMat.sum(axis=1)

    # 距离升序得到索引
    sortedIndex = argsort(distances)
    labelCount = {}

    # 计算k个最近邻
    for i in range(k):
        Label = labels[sortedIndex[i]]
        labelCount[Label] = labelCount.get(Label , 0) + 1

    # 输出k个最近邻的众数
    sortedLabelCount = sorted(labelCount.iteritems(),
                              key=operator.itemgetter(1), reverse=True)

    return sortedLabelCount[0][0]
```

Regression

计算相关系数

```python
# 计算两个list相乘
def XmulY(X,Y):
    newList=[]
    for i in range(len(X)):
        newList.append(X[i] * Y[i])
    return newList

# 计算两个序列的相关系数
def correlation(X,Y):
    cov = mean(XmulY(X,Y))-mean(X) * mean(Y)
    varX = var(X)
    varY = var(Y)

    return cov / sqrt(double(varX * varY))
```

kNN 计算每个情感概率

```python
# 进行kNN计算
def kNN(testX, dataSet, labels, k):

    # 计算测试样本与训练样本集的距离
    dataMat = (tile(testX, (dataSet.shape[0], 1)) - dataSet) ** 2
    distances = dataMat.sum(axis=1)

    # 距离升序得到索引
    sortedIndex = argsort(distances)

    here = 0

    # 计算k个最近邻共同作用得到的值
    Label = labels[sortedIndex[0]] / distances[0]

    for i in range(1,k):
        Label += labels[sortedIndex[i]] / distances[i]

    # 所有情感值总和为1
    return Label / Label.sum(axis=0)
```

4. 创新点&优化（如果有）

1. 用 one_hot、TF 和 TF_IDF 矩阵
2. 用多种距离找 k 个邻居

曼哈顿距离、欧氏距离与切比雪夫距离

```python
# 曼哈顿距离
if disType == 1:
    distances = abs((tile(testX, (dataSet.shape[0], 1)) - dataSet)).sum(axis=1)
# 欧式距离
if disType == 2:
    dataMat = (tile(testX, (dataSet.shape[0], 1)) - dataSet) ** 2
    distances = dataMat.sum(axis=1) ** 0.5
#切比雪夫距离
if disType == 3:
    distances = abs((tile(testX, (dataSet.shape[0], 1)) - dataSet)).max(axis=1)
```

余弦距离

```python
def cosDist(testX, dataSet, labels, k,disType):
    distances = []
    index = 0
    for i in dataSet:
        # 值比较大意味着距离近，为了升序，这里加上负号
        distances.append(-spatial.distance.cosine(testX,i))

    # 距离升序得到索引
    sortedIndex = argsort(distances)
    labelCount = {}

    # 计算k个最近邻
    for i in range(k):
        Label = labels[sortedIndex[i]]
        labelCount[Label] = labelCount.get(Label , 0) + 1

    # 输出k个最近邻的众数
    sortedLabelCount = sorted(labelCount.iteritems(),
                              key=operator.itemgetter(1), reverse=True)

    return sortedLabelCount[0][0]
```

加权（y=exp(-x)）：

```python
rangeDis = distances[sortedIndex[dataSet.shape[0] - 1]] - distances[sortedIndex[0]]

distances = (distances - distances[sortedIndex[0]])/ rangeDis

# 计算k个最近邻
for i in range(k):
    Label = labels[sortedIndex[i]]
    labelCount[Label] = exp(-distances[i]) + labelCount.get(Label , 0)
```

加权：（y=1/x）

```python
for i in range(k):
    Label = labels[sortedIndex[i]]
    labelCount[Label] = 1.0/double(distances[i]) + labelCount.get(Label , 0)
```

还有加入常量的加权：

```python
Label = labels[sortedIndex[0]] / (1.5 + distances[0] ** 0.75)
    for i in range(1,k):
        Label += labels[sortedIndex[i]]  /(1.5 + distances[i] ** 0.75)
```

归一化：

```python
rangeDis = distances[sortedIndex[dataSet.shape[0] - 1]] - distances[sortedIndex[0]]

distances = (distances - distances[sortedIndex[0]])/ rangeDis
```

三、 实验结果及分析

1. 实验结果展示示例

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

1NN：

```
In [11]: test("one_hot")
0.155

In [12]: test("TF")
0.263

In [13]: test("TF_IDF")
0.209
```

1NN-regression：

```
In [2]: test("one_hot")
0.0639512856657
0.068076676096
0.213788268814
0.0234161708452
0.0847480762534
0.157751211095
Out[2]: 0.10195528146145723

In [3]: test("TF")
0.00160040679766
0.114232831957
0.129617809741
0.0275376101734
0.173386972099
0.118044250422
Out[3]: 0.094069980198229167

In [4]: test("TF_IDF")
0.0196354774199
-0.00652123239872
0.104374230374
0.0325362327549
0.185637121561
0.111887506369
Out[4]: 0.074591556013313812
```

(原始算法结果)

|-----------如有优化，请重复 1，2，分析优化后的算法结果-----------------------|

优化后：

1.结果

kNN 分类：

曼哈顿距离	欧氏距离	切比雪夫距离	余弦距离

k = 1: 0.155000	k = 1: 0.155000	k = 1: 0.202000	k = 1: 0.183000
k = 2: 0.111000	k = 2: 0.111000	k = 2: 0.202000	k = 2: 0.156000
k = 3: 0.178000	k = 3: 0.178000	k = 3: 0.202000	k = 3: 0.132000
k = 4: 0.258000	k = 4: 0.258000	k = 4: 0.202000	k = 4: 0.138000
k = 5: 0.280000	k = 5: 0.280000	k = 5: 0.202000	k = 5: 0.160000
k = 6: 0.312000	k = 6: 0.312000	k = 6: 0.160000	k = 6: 0.174000
k = 7: 0.295000	k = 7: 0.295000	k = 7: 0.160000	k = 7: 0.174000
k = 8: 0.329000	k = 8: 0.329000	k = 8: 0.160000	k = 8: 0.167000
k = 9: 0.341000	k = 9: 0.341000	k = 9: 0.160000	k = 9: 0.164000
k = 10: 0.332000	k = 10: 0.332000	k = 10: 0.160000	k = 10: 0.159000
k = 11: 0.292000	k = 11: 0.292000	k = 11: 0.160000	k = 11: 0.161000
k = 12: 0.318000	k = 12: 0.318000	k = 12: 0.160000	k = 12: 0.154000
k = 13: 0.304000	k = 13: 0.304000	k = 13: 0.160000	k = 13: 0.154000
k = 14: 0.309000	k = 14: 0.309000	k = 14: 0.202000	k = 14: 0.149000
k = 15: 0.303000	k = 15: 0.303000	k = 15: 0.202000	k = 15: 0.156000
k = 16: 0.311000	k = 16: 0.311000	k = 16: 0.202000	k = 16: 0.158000
k = 17: 0.303000	k = 17: 0.303000	k = 17: 0.202000	k = 17: 0.162000
k = 18: 0.310000	k = 18: 0.310000	k = 18: 0.202000	k = 18: 0.183000
k = 19: 0.303000	k = 19: 0.303000	k = 19: 0.202000	k = 19: 0.188000
k = 20: 0.313000	k = 20: 0.313000	k = 20: 0.202000	k = 20: 0.196000
k = 21: 0.336000	k = 21: 0.336000	k = 21: 0.202000	k = 21: 0.193000
k = 22: 0.354000	k = 22: 0.354000	k = 22: 0.202000	k = 22: 0.188000
k = 23: 0.354000	k = 23: 0.354000	k = 23: 0.202000	k = 23: 0.185000
k = 24: 0.366000	k = 24: 0.366000	k = 24: 0.202000	k = 24: 0.185000
k = 25: 0.372000	k = 25: 0.372000	k = 25: 0.202000	k = 25: 0.188000
k = 26: 0.364000	k = 26: 0.364000	k = 26: 0.202000	k = 26: 0.187000
k = 27: 0.360000	k = 27: 0.360000	k = 27: 0.202000	k = 27: 0.187000
k = 28: 0.352000	k = 28: 0.352000	k = 28: 0.202000	k = 28: 0.189000
k = 29: 0.364000	k = 29: 0.364000	k = 29: 0.202000	k = 29: 0.196000

加权之后

(one_hot + 欧氏距离 + exp(-x))

```
k = 22: 0.373000
k = 23: 0.372000
k = 24: 0.374000
```

回归：

用了 TF 矩阵的预测

0.11248201549	0.149056555436	0.144490506763
0.116186876948	0.135873219271	0.137531002635
0.229937318978	0.329857714803	0.269100678053
0.152495709661	0.179793528724	0.120315062382
0.167868179523	0.20776979146	0.15138726733
0.0701675712256	0.213225603962	0.16560806813
k = 1: 0.141523	k = 6: 0.202596	k = 11: 0.164739
0.122495198031	0.157990638313	0.151185661437
0.144499585848	0.140383452689	0.140754765334
0.284360762735	0.322485180359	0.288210749826
0.164320633263	0.17396849491	0.122555053002
0.195785124905	0.200890016679	0.144997940824
0.11052948039	0.203486878552	0.144190777915
k = 2: 0.170332	k = 7: 0.199867	k = 12: 0.165316
0.157337017892	0.155450473519	0.143039170806
0.1553600629	0.127969014498	0.163618050927
0.323088106233	0.293190151784	0.288988296997
0.213106879288	0.15116529465	0.117491662537
0.229948362593	0.192452817645	0.142045945265
0.156084493843	0.182338998447	0.154776808119
k = 3: 0.205821	k = 8: 0.183761	k = 13: 0.168327
0.134813589527	0.172142828883	0.125784433497
0.138023485008	0.134040606725	0.141084624679
0.324576383593	0.27116040455	0.285949025168
0.189004604492	0.138110654463	0.13286969468
0.194272762708	0.1608748627	0.170596357509
0.177144174465	0.167550656215	0.140444232837
k = 4: 0.192972	k = 9: 0.173980	k = 14: 0.166121
0.130703265934	0.157459547572	0.107655273985
0.136110407599	0.137105642624	0.140162386632
0.337147862248	0.266416768037	0.294448689116
0.186219462709	0.122429824869	0.145131684541
0.206247925163	0.162210779687	0.166962685161
0.191070018499	0.161042776911	0.150801887412
k = 5: 0.197916	k = 10: 0.167778	k = 15: 0.167527

```
0.10549278268
0.156542393988
0.289388816332
0.15223425152
0.157598026937
0.158665738623
k = 16: 0.169987
0.111244924674
0.137111358305
0.293031610813
0.166998956538
0.146523070997
0.155010889532
k = 17: 0.168320
0.109318400264
0.130039754848
0.280963824008
0.151932331733
0.138872513059
0.147413660907
k = 18: 0.159757
0.0998975261735
0.135956870855
0.278351398994
0.16174467029
0.146456663722
0.15229215588
k = 19: 0.162450
0.107524568687
0.159455782828
0.263993913337
0.15608609385
0.142504757646
0.159557360998
k = 20: 0.164854
```

```
0.113473533336
0.163565626871
0.234996600331
0.146967744063
0.151568651723
0.152753922846
k = 21: 0.160554
0.108644546899
0.168124695254
0.217631405417
0.142294756613
0.153176907534
0.152205154515
k = 22: 0.157013
0.10542227171
0.1817388135
0.215108881951
0.134723420082
0.153710122378
0.149278110338
k = 23: 0.156664
0.106520030103
0.167720910667
0.221004065573
0.126734969749
0.144881223539
0.161682276144
k = 24: 0.154757
```

```
0.11484344405
0.167526216186
0.223794409314
0.137644569523
0.134439495942
0.169017334052
k = 25: 0.157878
0.13632034205
0.177691612132
0.225108978189
0.131237969489
0.123189427015
0.146385768598
k = 26: 0.156656
0.136016205346
0.174157170513
0.221906206553
0.1202722713
0.127096806033
0.146055395504
k = 27: 0.154251
0.13856778562
0.158025161541
0.217902091481
0.107052859115
0.125319404706
0.157269552961
k = 28: 0.150689
0.139035238746
0.168987666654
0.213688859412
0.0937257386358
0.104000354902
0.152731008939
k = 29: 0.145361
```

加权方式为：

```
Label = labels[sortedIndex[0]] /(1.0+distances[0])
for i in range(1,k):
    Label += labels[sortedIndex[i]]  /(1.0+distances[i])
```

One_hot,k=15,欧式

	anger	disgust	fear	joy	sad	surprise
r	0.038686	-0.02958	0.054606	0.072676	0.046585	0.10737
average	0.048391					
evaluation	极弱相关 加油哦					

K=15，曼哈顿距离

	anger	disgust	fear	joy	sad	surprise
r	0.039329	-0.02916	0.054329	0.07324	0.046566	0.107861
average	0.048694					
evaluation	极弱相关 加油哦					

加权方式（k=20）：

```python
Label = labels[sortedIndex[0]] * (1.0- 1.0/(2.0 + distances[0]) )
for i in range(1,k):
    Label += labels[sortedIndex[i]]  *(1.0- 1.0 / (2.0 + distances[i]) )
```

	anger	disgust	fear	joy	sad	surprise
r	0.058982	-0.03075	0.070186	0.086042	0.04643	0.121802
average	0.058782					
evaluation	极弱相关 加油哦					

加权方式（k=20）：

```python
Label = labels[sortedIndex[0]] / (1.5 + distances[0] ** 0.75)
for i in range(1,k):
    Label += labels[sortedIndex[i]]  /(1.5 + distances[i] ** 0.75)
```

	anger	disgust	fear	joy	sad	surprise
r	0.067045	-0.03382	0.071875	0.085842	0.040288	0.124336
average	0.059261					
evaluation	极弱相关 加油哦					

加权方式（k=20）

```python
Label = labels[sortedIndex[0]] / (1.5 + distances[0] ** 5)
for i in range(1,k):
    Label += labels[sortedIndex[i]]  /(1.5 + distances[i] ** 5)
```

	anger	disgust	fear	joy	sad	surprise
r	0.065345	-0.03825	0.080718	0.090522	0.040613	0.124852
average	0.060633					
evaluation	极弱相关 加油哦					

由上可见，在 validation 里面跑的还可以的方法，在 test 里面并不好。我的 test 是由 validation 改过来的，所以应该使用于同样分布的 test，不知道是不是放进表格方法不对。修改了很久，却还是没有成功。