使用 KNN 在 01 二值的 MNIST 数据集上的实验报告

作者：户建坤

2017210904 计算机

1. 使用马氏距离

1.1 计算协方差矩阵的伪逆

```
trainX_T = trainX.T
D = np.cov(trainX_T)
print(trainX_T[0][0:10])
invD = np.linalg.pinv(D)
```

1.2 在 kd 是否需要跑另一个分支的时候，同样使用相应的马氏距离来判断

```
if rrt is not None and (len(h) < k or self.dist(x, vir_data)  < (-h[0][0])):
    self.find_dfs(x, h, rrt, k)
```

1.3 在本实验的代码中，需要在初始化 kd 树的时候传入参数字符串以及对应的矩阵

```
clf2 = KDTree_like_sklearn(dist_kind='Mahalanobis', k=ans_max[2], mat=invD)
```

如果普通距离，则为 simple：

```
clf1 = KDTree_like_sklearn(dist_kind='simple', k=ans_max[1])
```

2. 使用 WKNN

2.1 权重公式

在统计最近的 k 个点的时候，每个点的权重和它的距离相关，公式为：

$$w^i = (h^k - h^i)/(h^k - h^1)$$

2.2 在代码的实现中

```
    dist_list = []
    for i in range(h_len):
        t = heapq.heappop(h)
        dist_list.append((-t[0], self.data[t[1]][-1]))
    for i in range(h_len):
        # cnt[dist_list[i][1]] += (dist_list[0][0] - dist_list[i][0])/(dist_list[0][0] - dist_list[
        # print((dist_list[0][0] - dist_list[i][0]) / (dist_list[0][0] - dist_list[-1][0]))
        if np.fabs(dist_list[0][0] - dist_list[-1][0]) < 1e-5:
            # print('use this similar equal weight')
            cnt[dist_list[i][1]] += 1
        else:
            cnt[dist_list[i][1]] += (np.sqrt(dist_list[0][0]) - np.sqrt(dist_list[i][0])) / (
            np.sqrt(dist_list[0][0]) - np.sqrt(dist_list[-1][0]))
return cnt.argmax()
```

需要注意的是，如果所有的 k 个点的距离恰好一样，那么是特殊的情况，分母会为 0，此时只需要所有的权重都是 1 就可以。并且在我们的 1024 个维度都是 0 或者 1 的情况下，很容易出现距离都相等的可能性。

2.3 本实验的代码中，需要在初始化 kd 树的时候传入字符串

```
clf3 = KDTree_like_sklearn(dist_kind='simple', k=ans_max[3], way='wknn')
```

不是 wknn 的时候，不传 way 的值即可。

3. 使用 HEAP 和 KD 树的 KNN 实现

实现了 heap 的 kd 树，优化策略为简单的半径和轴长的比较。跑不动，引入 pca。

3.1 KD 树优化

使用 kd 树来进行优化，每次分成两堆，在选择维度的时候，简单的使用+1%维度的方法，在查找的时候，使用简单的比较坐标轴距离和目前 k 个中最大距离的策略。K 个最小距离用 heap 来维护。

3.2 PCA 预处理数据

使用 kd 树优化，如果使用原始数据，1024 维，并且每维度都是 0 或者 1，那么速度会比线性的慢的多，因为每个维度都是 0 或者 1 使得 kd 树的优化策略几乎没有用，而使用了 heap 和递归也会使得当 kd 树退化到线性的时候没有直接线性快。因此为了体现 kd 树的价值，我对 1024 维度的引入 pca 降维度到 32

4.不同 KNN 的比较，以及选取超参数 K 的策略

4.1 使用 pca 的数据

一共 5 个 knn：

①Sklearn 中的 knn，作为一个比较好的 knn 的对比；

②使用欧式距离，不用 wknn 的正常 knn，基于 kdTree 实现；

③使用马氏距离，不用 wknn 的正常 knn，基于 kdTree 实现；

④使用欧式距离，使用 wknn 的 knn，基于 kdTree 实现；

⑤使用马氏距离，使用 wknn 的 knn，基于 kdTree 实现；

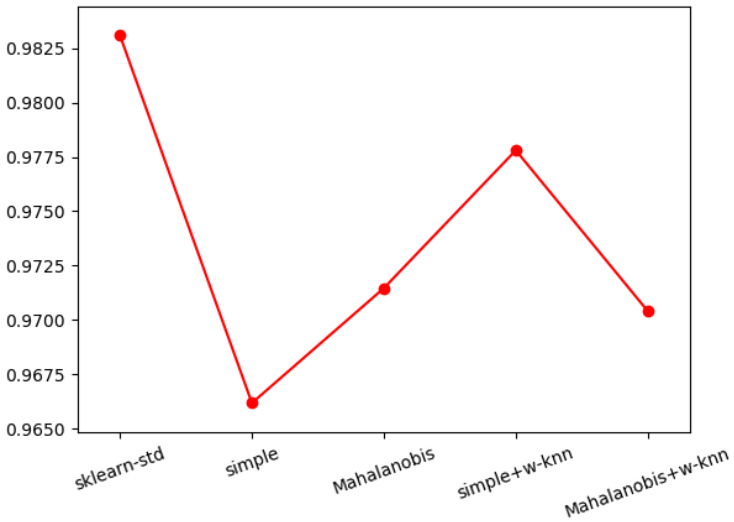
4.1.1 knn 的 k 的选取

采取把 train 集分成 80%的 train 和 20%validation 集，找到最好的 k，然后再用这个 k 训练 100%的 train，最终 5 个 knn 分别的最好的 k 值为[3 3 1 9 5]

4.1.2 在 test 集合上的测试：

正确率的定义为：预测标签和实际标签一样的/总个数

正确率分别为：



图一 使用 pca 降到 32 维，使用[3 3 1 9 5]分别为 k 值

0.9830866807610994

0.9661733615221987

0.9714587737843552

0.9778012684989429

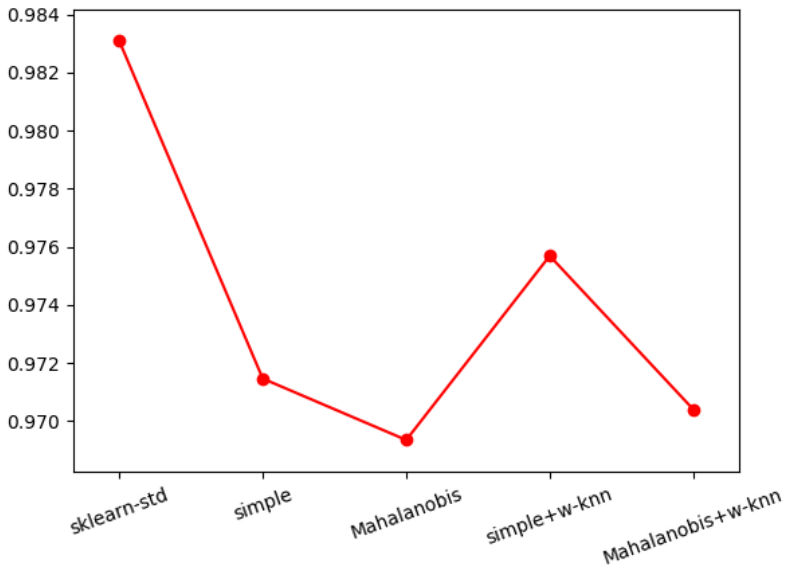
0.9704016913319239

4.1.3 结果分析

（1）Sklearn 的效果最好，在没有传入权重的参数下，sklearn 的方法应该和 simple 的是一样的，但实际上 sklearn 的更好，应该是 sklearn 做的有些优化，对于经过 pca 变换之后的这样的数据处理的更好

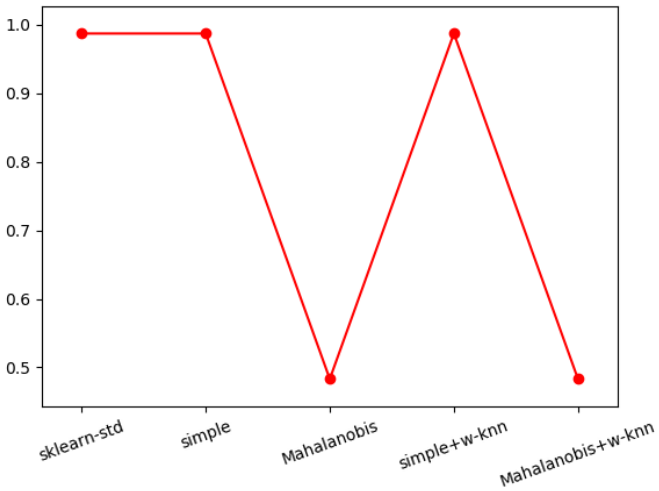
（2）在使用马氏距离之后，wknn 的效果变差，而不正常的 knn 的有提高，说明马氏距离可以更好的反应距离之间的大小关系，但是因为做了伸缩变换，使得距离之间的比值关系并没有变换前的比例关系更加的能够反映出“属于某一类的概率”，因此使用了 wknn 反倒变差了。

（3）实现的最好的情况是欧氏距离+wknn，并且结合选取了 k 均为 5 的下面的结果，发现的确马氏距离的使用不稳定，并且依赖于 knn 的结果选取策略，而对于欧氏距离，wknn 的策略能够稳定的提高，因此 wknn 和二阶矩之间的叠加是不相互矛盾的。



图二 使用 pca 降到 32 维，使用[5 5 5 5 5]分别为 k 值

4.2 使用原始维度（1024）的结果分析



图三 1024 维，使用[3 3 1 9 1]分别为 k 值

0.9873150105708245

0.9873150105708245

0.48308668076109934

0.9873150105708245

0.48308668076109934

结果非常特殊：

（1）sklearn 的 knn 和实现的欧式距离，不用 wknn 的 knn 结果一样，因此对于这样特殊的数据，sklearn 没有做任何决策上的优化。

（2）使用马氏距离会变差，也和维度上取 0 和 1 二值可能有关

（3）使用 wknn 没有用，因为在实现中也发现，这种情况下的 k 个最小维度距离很多都是相等的，因此权重会被特判为等价的，所以 wknn 没有用。

（4）使用原始数据虽然最高的正确率会高，但是在时间上不理想，以及因为数据的二值特殊性，使得实验探讨的马氏距离和 wknn 的效果不理想，因此不作为主要实验结果分析。