监督学习部分

PB16060130 顾健鑫

1. 数据预处理
   本次实验中 KNN 与决策树均未作数据预处理，直接使用原始数据进行训练；SVM 由于为二分类器，因此将训练集和测试集进行 18 种不同的标签处理，即是该类的数据标签变为 1，不是的则变为-1
2. 算法伪代码
   所有模型的结构统一为训练与预测两种方法，其余相关计算则封装在类内作为私有方法
   a. **KNN**
      训练：
      即记录使用的训练集和对应的标签。
      预测：
      计算每个测试点与每个训练点之间的距离，在此使用 Euclid 距离。计算方法为采用 Numpy 的广播来实现向量化以消除二重循环以大幅度提升计算的速度
      涉及代码：
      ```
      dists = np.sqrt(np.sum(X_test ** 2, axis=1, keepdims=True) + np.sum(self.X_train ** 2, axis=1) - 2 * X_test.dot(self.X_train.T))
      ```
      Dists 为(N_test, N_train)大小的矩阵，每行为每个测试点到每个训练点的距离，以此为依据选取 K 个距离最近的点，并使用其中出现次数最多的标签作为该测试点的预测标签。

   b. 决策树
      训练：
      即建树，
      计算信息熵以及信息增益，选择信息增益最大的作为新的节点，当以选出的节点作为依据对数据进行划分后，若新的自己中数据均为同一个标签，则以此标签作为一个叶子结点，否则在子集上继续计算剩余为划分类别的信息增益。

      预测：
      即在树上搜索，若为非叶子结点，则以当前节点的属性为依据，选择测试点对应属性的分支，直至查找到叶子结点，并以此作为最终的预测结果。

   c. SVM
      由于本次实验使用的是 cvxopt.solvers 中的 qp 计算框架，因此按照框架的定义，需给出 P、q、G、H、A、b6 个矩阵
      问题的标准形式为

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

在带有松弛边界的 SVM 问题中，问题最终化为

$$\begin{array}{ll} = & \max_\alpha \alpha^\top \mathbf{e} - \frac{1}{2}\alpha^\top (\mathbf{y}\mathbf{y}^\top \circ K)\alpha \\ \text{subject to} & 0 \le \alpha_i \le C, \forall i \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{array}$$

因此将目标函数的符号取反就得到了符合框架要求的标准形式

因此 SVM 的伪代码描述如下：

训练：

通过指定的方法计算出 Kernel 矩阵(Num_train, Num_train)，每个元素表示 K(xi, xj)

P：Y_train 的外积 * Kernel 矩阵（此处的乘号表示点乘，即对应位置元素相乘）

q：长度为 Num_train 的元素全为-1 的向量

G：分为两部分，前半部分为标准部分，表示大于等于 0 的条件，在此可转换为小于等于 0，因此为对角线全为-1 的对角阵；后半部分为松弛部分，表示小于等于 C 的条件，因此为对角线全为 1 的对角阵，将前后两部分垂直方向上堆叠起即可

H：第一部分为长为 Num_train 的 0 向量，第二部分为长为 Num_train 的元素都为 C 的向量，将两个向量垂直堆叠即可

A：表示等号条件，因此为形状为(1, Num_train)的向量，每个对应位置上的取值为 yi

b：取 0 即可

具体涉及代码

```
P = cvxopt.matrix(np.outer(y, y) * self.K)
q = cvxopt.matrix(-1 * np.ones(N))

# For the standard problem(without soft margins)
# The constraints are α_i >= 0 which equals to -α_i <= 0
G_std = cvxopt.matrix(-np.eye(N))
h_std = cvxopt.matrix(np.zeros(N))

# For the SVM with soft margins, the additional constraints are
# α_i <= C
if self.C > 0:
    G_soft = cvxopt.matrix(np.eye(N))
    h_soft = cvxopt.matrix(np.ones(N) * self.C)

else:
    G_soft = None
    h_soft = None

# Combine the two types of constraints above together using
np.vstack()
G = cvxopt.matrix(np.vstack((G_std, G_soft)))
h = cvxopt.matrix(np.vstack((h_std, h_soft)))
```

```
A = cvxopt.matrix(y.astype(float), (1, N))
b = cvxopt.matrix(0.0)
```

至此对应矩阵构造完成调用计算包求解，并返回结果，即为 α 向量

预测：

使用指定的方法计算出 Kernel 矩阵，矩阵大小为(Num_test, Num_train)
使用 1e-5 作为阈值，即小于此值的 αi 视为 0，选出支持向量与其对应的标签。依据下列公式，可以求得 b 的取值

$$b = y_i - \sum_{j=1}^{n} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \quad \text{for any i that } \alpha_i \neq 0$$

再根据下列公式进行测试集上的预测

$$y^* = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x'}) + b\right)$$

具体涉及的代码为：

```
y_pred = np.sum((self.support_multipliers *
self.support_labels).reshape(-1, 1) * K, axis=0) + self.bias
y_pred[y_pred > 0] = 1
y_pred[y_pred < 0] = -1
```

至此完成预测

3. 实验结果
   a. KNN

   可供选择的 K 值有 1, 5, 8, 10, 15, 20, 50, 100；通过交叉验证，计算出每个 K 值对应的三个指标的均值

	1	5	8	10	15	20	50	100
Accuracy	0.1609	0.1779	0.1531	0.1372	0.1143	0.0997	0.0556	0.0336
Macro F1	0.1516	0.1645	0.1461	0.1338	0.1115	0.0976	0.0535	0.0294
Micro F1	0.2531	0.2749	0.2411	0.2179	0.1823	0.1592	0.0882	0.0522

   综合三条属性考虑，取 K = 5
   此时使用测试集进行验证，得到结果为
   Got 4179 / 5611 correct => Accuracy: 0.744787
   Macro F1: 0.6881324621638021
   Micro F1: 0.7447865254859889

   b. 决策树

   由于使用完整数据集进行作图会导致时间过长以及图片冗杂，因此在此展示的为使用 4100 个数据点进行建树得到的结果进行作图，得到的图片由于尺寸关系不在报告中进行展示，详见 part1 的 src 文件夹中 Tree.png 文件
   使用全部数据集进行建树后进行预测，得到的结果如下所示

Accuracy	Macro_F1	Micro_F1
0.373017	0.323856	0.373018

   c. SVM

   使用全部的 20000 个训练点进行 K 矩阵求解由于向量化可以接受，但使用凸优化计算包进行计算得到的时间开销过大，因此放弃使用 5 fold 交

叉验证的方式进行参数选择，同时由于为 2 分类，因此对于每个类进行预测的 SVM 参数也可能存在较大变化，故在此展示的结果为选择后的一组参数使用 RBF 核函数在 5000 大小的均匀采样训练集（保证正例标签与反例标签的比值）上进行训练的预测结果

	Support Vector Count	Accuracy	Macro_F1	Micro_F1
0	1908	0.999109	0.499776	0.999108
1	97	0.997148	0.499285	0.997148
2	58	0.991267	0.497807	0.991266
3	79	0.997148	0.499286	0.997148
4	169	0.992871	0.498211	0.992871
5	290	0.983247	0.495776	0.983247
6	718	0.978970	0.494686	0.978969
7	1902	0.975584	0.493820	0.975583
8	2615	0.948850	0.486877	0.948849
9	3420	0.939048	0.484282	0.939047
10	2366	0.929246	0.481663	0.929246
11	3684	0.898236	0.473195	0.898235
12	731	0.871859	0.465771	0.871859
13	3355	0.850472	0.459597	0.850471
14	2199	0.837640	0.455824	0.837639
15	1441	0.922830	0.479933	0.922829
16	334	0.986099	0.496500	0.986098
17	2110	0.900374	0.473787	0.900373

单从实验结果看，SVM 的预测正确率相对较高，但由于本次实验使用的数据集分布极为不均匀，如标签为 0 的数据在测试集中占比为 5/5611，因此很容易导致 SVM 的预测结果单一化，如对于标签 0 而言，训练集中的反例标签占有绝对的比重，因此经过均匀采样后，可以供模型学习的正例信息屈指可数，这也导致模型学到的大量信息为反例信息，而这样的信息对于 SVM 模型而言并不是很好的学习条件，通过调研，在文章 Understanding Black-box Predictions via Influence Functions 中提到，对于一个二分类问题，相对于更复杂的模型，如 CNN。SVM 学习到的信息更多为 X 是 A 类的表述，因此在分布较均匀的数据集上表现尚可；而对于含有大量 Y 不是 A 的训练集，其学习能力明显低于更复杂的模型，如 CNN，即 SVM 难以通过大量 Y 不是 A 的训练数据生成相对非常有效的预测 X 是 A 的模型。在本次实验中，这样的缺陷很明显的体现了出来，对于某些标签的预测，由于训练集上数据的极度不均衡，使得模型的预测结果呈现单一化，因此出现了，准确度极高的现象。若能应用更大的数据集，如全部训练集并使用交叉验证进行细调参，结果可能会有所改善。

4. 实验总结

通过本次实验，我对实验要求的 3 个线性分类模型有了更深一步的了解，并对其具体实现有了更加直观的掌握。同时也深刻的体会到了 Numpy 中广播与向量化带来的计算性能上的提升，如计算数据集中各点之间的距离。虽然由于计算

规模的问题，支持向量机部分为能实现交叉验证调参，但通过对 KNN 的交叉验证调参也对 5 folds 验证法有了直观的理解，并通过调研一些文献了解到了线性模型在信息学习以及理解上与 state-of-the-art 模型上存在的巨大差距，收获颇多。