监督学习 实验报告

PB17050965 朱子琦

实验环境

- python 3.8
- numpy 1.18.4
- pandas 1.0.5
- scipy 1.4.1
- sklearn 0.0

实验内容

数据预处理与结果评测

数据预处理

本次实验中使用的数据集中存在数值与字符串混合的情况，不便于训练。在读入数据后，使用sklearn库的LabelEncoder对字符串进行预处理,使得数据全部变为数值。

得到数值后再对数据G3列进行处理，大于等于10分记为通过，否则为不通过，并按照通过状态进行{1，-1}的二值化。

由于实验中要求分别在 **考虑G1 G2** 和 **不考虑G1 G2** 的情况下进行训练，因此在预处理阶段引入参数drop_g1g2，用于表明是否要去除 G1 G2 数据，函数返回经过sklearn库 train_test_split函数打乱后的数据集

```
def pre_process(data, drop_g1g2):
    print("=======pre-process of data=======")
    le = LabelEncoder()
    for col in data[["school", "sex", "address", "famsize", "Pstatus", "Mjob", "Fjob", "reason",
                     "paid", "activities", "nursery", "higher", "internet", "romantic"]].columns
        data[col] = le.fit_transform(data[col].values)
    for row in data.index:
        if data['G3'][row] >= 10:
            data['G3'][row] = 1
        else:
            data['G3'][row] = -1

    label = data.pop('G3')
    # 根据需求选择是否去除G1，G2
    if drop_g1g2 == 1:
        data = data.drop(columns=['G1', 'G2'])
    return train_test_split(data, label, test_size=0.3)
```

结果评测

本次实验评价指标为F1 Score 对应公式如下：

$$F1score = \frac{2 \times P \times R}{P + R}$$

需要统计实验结果predictlabel 与 真实标签 testlabel之间的 TP TN　FP　FN 信息，并据此计算 准确率P，召回率R和F1 score。代码实现如下：

```
def evaluate_result(testlabel, predictlabel):
    TP, FP, TN, FN = 0, 0, 0, 0
    for index in predictlabel:
        predict_res = predictlabel[index]
        real_res = testlabel[index]
        if real_res == 1 and predict_res == 1:
            TP += 1
        elif real_res == 1 and predict_res == -1:
            FN += 1
        elif real_res == -1 and predict_res == 1:
            FP += 1
        elif real_res == -1 and predict_res == -1:
            TN += 1
    #print("TP:" + str(TP)+"\nTN:" + str(TN) +
    #        "\nFP:" + str(FP)+"\nFN:" + str(FN))
    try:
        P = TP / (TP + FP)
    except:
        P = 0
    try:
        R = TP / (TP + FN)
    except:
        R = 0
    try:
        F1 = 2 * P * R / (P + R)
    except:
        F1 = 0
    return P, R, F1
```

标签预测

```
def sign(x):
        """
        sign [用于判别testset中的label]

        Args:
            x ([np.array]): [输入测试例]

        Returns:
            [num]: [返回计算得到的标签{1, -1}]
        """
        res = b
        for i in range(n):
            res += alpha[i] * K(x, S[i], kernel) * y[i]
        if res >= 0:
            return 1
        else:
            return - 1

    result = {}
    for i in testset.index:
        x = testset.loc[i].values
        result[i] = sign(x)
    return result
```

KNN

1. 算法思路

KNN 算法的思路是在训练集中寻找与测试数据最相近的k个数据，根据这些邻近数据确定待预测数据的类别。

在实验过程中，调用 **knn(trainset, trainlabel, testset, k_list)** 函数,输入训练集和测试集以及要使用的近邻数 **k** 。当输入测试数据时，将会计算测试数据与trainset中每条数据的距离，并选取距离最小的k条，统计出现次数最多的标签作为测试数据的标签。

2. 核心代码

距离度量

本实验中使用欧氏距离判断两条数据间距，计算方式如下：

```
def get_distance(data1, data2, n):
    # Euclidean distance
    res = 0
    for i in range(n):
        d = data2[i] - data1[i]
        res += d * d
    return res ** 0.5
```

选取最近邻的k条数据

实验过程中，针对每条测试数据，计算出其与各条训练数据间的距离，完成后排序取出前k条作为结果

```
for testrow in testset.index:
    d_list = []
    for trainrow in trainset.index:
        t1 = trainset.loc[trainrow].values
        t2 = testset.loc[testrow].values
        d = get_distance(t1, t2, n)
        d_list.append([d, trainrow])
    d_list.sort()
```

3. 结果分析

经过测试后，选取 k = 21时效果较好。该数值需要因数据集变化进行相应调整。
进行10次测试后取均值如下：

student-mat.csv	F1 score	P	R
含G1 G2	0.9091	0.8731	0.9496
不含G1 G2	0.7994	0.6865	0.9601

student-por.csv	F1 score	P	R
含G1 G2	0.9400	0.8924	0.9933
不含G1 G2	0.9222	0.8559	1.0000

SVM

算法思路

支持向量机中需要解二次优化问题，构造一个超平面将数据集分割成不同的部分，最后判断待预测的数据处于超平面的哪一侧即可得出预测标签.

所要求的二次规划问题是：

$$min_\alpha \ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i i\alpha_j y_i y_j K(x_i,x_j)-\sum_{i=1}^{N}\alpha_i$$
$$s.t.\sum_{i=1}^{N}\alpha_i y_i=0$$
$$0\le\alpha_i\le C,\ \ i=1,2,..,N$$

在 **svm(trainset, trainlabel, testset, C=1, kernel='line')** 函数中使用 **scipy.optimize.minimize** 求解优化问题最小值，其中C为软间隔参数，kernel为核函数类型

核心代码

核函数kernel

在SVM实现过程中使用核函数，以保证其可以使用不同的核方法

```
def K(x, z, type):
    """
    K [核函数]

    Args:
        x ([np.array]): [一维向量x]
        z ([np.array]): [一维向量z]
        type (str, optional): [选取的核函数类型].

    Returns:
        [num]: [两个向量在核函数下的乘积]
    """

    res = 0
    if type == 'line':
        res = x.T @ z
    elif type == 'polynomial':
        temp = x.T @ z
        res = (temp+1)**2
    elif type == 'gauss':
        res =  np.exp(-np.sqrt(np.linalg.norm(x-z) ** 2 / (2 * 0.5 ** 2)))
    return res
```

计算二次项系数矩阵

二次优化问题中，二次项系数构成一个半正定方阵P，通过训练集中数据两两求标签和核函数的乘积得到

```
print("======calculating matrix P=======")
for i in range(n):
    xi = trainset.iloc[i].values
    yi = trainlabel.iloc[i]
    S[i] = xi
    y[i] = yi

for i in range(n):
    for j in range(n):
        P[i][j] = K(S[i], S[j], kernel) * y[i] * y[j]
```

求解二次优化问题

使用 spicy.optimize.minimize 来求解二次优化问题

```
def fun(x):
        n = P.shape[0]
        res = 0
        for i in range(n):
            for j in range(n):
                res += P[i][j] * x[i] * x[j]
            res += - x[i]
        return res

def eqcons(x):
    n = P.shape[0]
    res = 0
    for i in range(n):
        res += y[i] * x[i]
    return res

bounds = Bounds([0] * n, [C] * n)
constraint = {'type': 'eq', 'fun': eqcons}
x0 = np.zeros(n)

res = minimize(fun, x0, method='SLSQP', constraints=[
                constraint], tol=0.0001, bounds=bounds)
print("QPproblem status:" + str(res.success))
alpha = res.x
```

分类决策函数

```
def sign(x):
    """
    sign [用于判别testset中的label]

    Args:
        x ([np.array]): [输入测试例]

    Returns:
        [num]: [返回计算得到的标签{1，-1}]
    """
    res = b
    for i in range(n):
        res += alpha[i] * K(x, S[i], kernel) * y[i]
    if res >= 0:
        return 1
    else:
        return - 1
```

结果分析

分别使用了线性核，多项式核和高斯核进行求解，得到的结果如下：

线性核

student-mat.csv	F1 score	P	R
含G1 G2	0.9299	0.8902	0.9733
不含G1 G2	0.8249	0.8074	0.9913

student-por.csv	F1 score	P	R
含G1 G2	0.9335	0.9439	0.9346
不含G1 G2	0.8861	0.88017	0.8977

多项式核

student-mat.csv	F1 score	P	R
含G1 G2	0.7703	0.8091	0.8263
不含G1 G2	0.6953	0.7315	0.6887

student-por.csv	F1 score	P	R

student-por.csv	F1 score	P	R
含G1 G2	0.9120	0.8385	1.0
不含G1 G2	0.7921	0.7882	0.9853

高斯核

student-mat.csv	F1 score	P	R
含G1 G2	0.9099	0.8214	1.0000
不含G1 G2	0.9233	0.8588	0.9998

student-por.csv	F1 score	P	R
含G1 G2	0.9176	0.8478	1.0000
不含G1 G2	0.9092	0.8223	1.000

多次测量中，发现有些情况下P，R，和 F1 score都非常低，而有时候很高，说明SVM在针对本实验中的问题时稳定性还比较差。也会出现R=0和R=1的情况，说明会出现过拟合的问题。

而对比多个核，发现相比之下，高斯核的性能最为稳定，且效果较好，因此选取高斯核

DT

算法思路

决策树是根据不同属性的信息增益，来分层考察各个属性进行决策。在每一层，比较当前剩余属性的信息增益，选取最大的一个作为当前层考察的属性作为特征，并根据特征的取值进行分类。选取取值最多特征的对应的标签作为当前节点标记，其余数据按照特征取值的不同分为不同的类，继续向下构造节点，直到所有数据集不可再分或信息增益小于阈值，停止分类。实际上是使用了极大似然法构造概率模型。这里使用的是朴素的ID3算法。

核心代码

生成决策树

在构造决策树的过程中，先判断是否符合叶节点条件，然后计算各特征的信息增益，选取最大的，再检查其是否满足大于阈值的条件，否则同样视为叶节点。
当不属于叶节点时，确定当前节点标记并对数据集进行分组，继续向下构造子节点。

```
def create_tree(data, label, A, threhold):
    n, m = data.shape
    c = C(label)
    if len(c) == 1 or len(A) == 0:
        node = TreeNode("leaf")
        node.mark = list(c.keys())[0]
    else:
        g = []
        for i in A:
            g.append((Gda(data,label,i,n), i))
        g.sort()
        maxg, bestfeature = g[-1][0], g[-1][1]
        if maxg < threhold:
            node = TreeNode("leaf")
            node.mark = list(c.keys())[0]
        else:
            Ddict = {}
            di = Di(data,bestfeature)
            for i in di:
                Ddict[i] = []
            for j in range(n):
                Ddict[data[j, bestfeature]].append(j)
            mostkind = list(di.keys())[0]
            node = TreeNode("node")
            node.feature = bestfeature
            node.value = mostkind
            node.mark = list(dict(Counter(label[Ddict[mostkind]])).keys())[0]
            node.nextnode = {}
            di.pop(mostkind)
            for i in di:
                node.nextnode[i] = create_tree(data[Ddict[i],:], label[Ddict[i]],A, threhold)
    return node
```

决策

决策过程如下，在整个树上寻找合适的节点，直到到达叶子节点或符合当前节点的分类模式，返回预测的标签

```
def search_tree(DT, test):
    if DT.type == "leaf":
        return DT.mark
    else:
        fe = DT.feature
        if test[fe] == DT.value or test[fe] not in DT.nextnode:
            return DT.mark
        else:
            return search_tree(DT.nextnode[test[fe]], test)
```

结果分析

使用决策树对两组数据进行10次预测，取测量平均值得：

student-mat.csv	F1 score	P	R
含G1 G2	0.9049	0.9204	0.8918
不含G1 G2	0.7062	0.7051	0.7208

student-por.csv	F1 score	P	R
含G1 G2	0.9383	0.9423	0.9349
不含G1 G2	0.9198	0.85658	0.9941

可以看出，G1G2对预测效果有提升作用，这一点在数据集student-mat.csv上尤为突出。这可能是因为对这组数据，G1G2对最终成绩的影响要更大