人工智能Lab2-监督学习问题

廖洲洲 PB17081504

实验目的

本部分实验目的为加强同学们对于SVM，KNN 以及其他经典机器学习算法的掌握，感受数据科学的魅力。

实验内容

数据预处理

在调用各算法前，先对数据进行预处理，实现包括字符串型属性转换为整型，属性和标签的划分，训练集和测试集的划分等。

主要代码

```
def data_preprocessing(data):
    """
    实现数据的预处理，包括字符串型属性转换为整型，属性和标签的划分，训练集和测试集的划分
    :param
        data:从csv文件中读取出的DataFrame类型
    :return:
        train_set:numpy数组类型，训练集的属性
        train_label:numpy数组类型，训练集的标签
        test_set:numpy数组类型，测试集的属性
        test_label:numpy数组类型，测试集的标签
    """
    columns = data.columns
    # 将字符型转换为整型
    for column in columns:
        if str(data[column].dtypes) != "int64":
            data[column] = LabelEncoder().fit_transform(data[column])
    # 将成绩G3转换为合格与不合格
    data["G3"] = data["G3"].apply(lambda v: 1 if v >= 10 else -1)
    # print(data)
    # 分割属性和标签，同时将DataFrame转换为array
    data_attr = np.array(data.loc[:, "school":"G2"])
    data_label = np.array(data["G3"])
    # 分割训练集和测试集
    train_set, test_set, train_label, test_lael = train_test_split(data_attr,
data_label, test_size=0.3, random_state=1)
    return train_set, train_label, test_set, test_lael
```

KNN算法

算法思想

给定测试样本，基于某种距离度量找出训练集中与其最靠近的k个训练样本，然后基于这k个邻居的信息进行预测。通常，在分类任务中可使用“投票法”，即选择这k个样本中出现最多的类别标记作为预测结果；在回归任务中可以使用“平均法”，即将这k个样本的实值输出标记的平均值作为预测结果；还可以基于远近进行加权平均或加权投票，距离越近的样本权重越大。

工作过程

对未知类别属性的数据集中的每个点依次执行以下操作：
1. 计算已知类别数据集中的点与当前点之间的距离；
2. 按照距离递增次序排序；
3. 选取与当前点距离最小的k个点；
4. 确定前k个点所在类别的出现频率；
5. 返回前k个点出现频率最高的类别作为当前点的预测分类。

主要代码

```
def KNN(train_set, train_label, test_set, k_neighbors):
    """
    输入训练集数据和测试集的属性及k值，输出预测标签集
    :param train_set:numpy数组类型，训练集的属性
    :param train_label:numpy数组类型，训练集的标签
    :param test_set:numpy数组类型，测试集的属性
    :param k_neighbors:整型，k个邻居
    :return:predict_label:list类型，预测的标签
    """
    predict_label = []
    for test_instance in test_set:
        # 对测试集的每个样本进行预测
        distance = [euc_dis(test_instance, train_instance) for train_instance in
train_set]
        # 对欧式距离进行排序，得到数组值从小到大的索引值
        neighbors = np.argsort(distance)
        k_near_neighbors = neighbors[:k_neighbors]
        # 统计k个邻居最多的标签值
        predict_label.append(stats.mode(train_label[k_near_neighbors])[0][0])
    return predict_label
```

实验结果说明与分析

选取不同的特征和k值进行的实验结果，“all”表示选取所有特征，“noG1G2”表示不含G1G2，“onlyG1G1”表示只选取特征G1G2

```
KNN
student-mat.csv
训练集大小 276
测试集大小 119
           k=1   k=2   k=3   k=4   k=5   k=6   k=7   k=8   k=9  k=10  k=11  \
all      0.897 0.861 0.928 0.911 0.928 0.926 0.929 0.932 0.942 0.934 0.947
noG1G2   0.707 0.648 0.767 0.658 0.802 0.757 0.815 0.800 0.828 0.814 0.830
onlyG1G2 0.922 0.945 0.929 0.945 0.939 0.945 0.939 0.933 0.933 0.933 0.933

          k=12  k=13  k=14  k=15  k=16  k=17  k=18  k=19  k=20
all      0.964 0.942 0.935 0.925 0.941 0.930 0.929 0.930 0.936
noG1G2   0.829 0.829 0.851 0.839 0.847 0.841 0.853 0.845 0.850
onlyG1G2 0.925 0.932 0.932 0.932 0.932 0.932 0.932 0.932 0.932
student-por.csv
训练集大小 454
测试集大小 195
           k=1   k=2   k=3   k=4   k=5   k=6   k=7   k=8   k=9  k=10  k=11  \
all      0.950 0.936 0.942 0.952 0.950 0.955 0.953 0.956 0.945 0.945 0.942
noG1G2   0.914 0.886 0.921 0.917 0.921 0.921 0.919 0.924 0.917 0.917 0.917
onlyG1G2 0.945 0.930 0.947 0.947 0.947 0.947 0.947 0.949 0.950 0.950 0.950

          k=12  k=13  k=14  k=15  k=16  k=17  k=18  k=19  k=20
all      0.948 0.945 0.945 0.942 0.945 0.942 0.948 0.945 0.945
noG1G2   0.917 0.917 0.917 0.917 0.917 0.917 0.917 0.917 0.917
onlyG1G2 0.950 0.950 0.950 0.950 0.953 0.953 0.950 0.950 0.950
```

- **对KNN算法存在K值的选择问题，k值从1增大到20，当k值很小，一般增大k时，性能会有所提升，这是因为有周围更多样本借鉴了，分类效果会变好。当k值更大时，一般性能会有所下降或者趋于稳定**
- **对数据集选择不同的属性进行训练，发现整体上只使用G1G2的性能大于使用所有属性的性能，使用所有属性的性能大于不使用G1G2的性能，由此可以发现目标属性与属性G1、G2有很强的相关性**
- **KNN算法比较简洁明了，同时模型训练时间快，并且预测的效果也好，可以看出性能基本上大于0.9**

SVM算法

算法思想

针对线性可分情况进行分析，对于线性不可分的情况，通过使用非线性映射算法将低维输入空间线性不可分的样本转化为高维特征空间使其线性可分，从而使得高维特征空间采用线性算法对样本的非线性特征进行线性分析成为可能。它基于结构风险最小化理论之上在特征空间中构建最优超平面，使得学习器得到全局最优化。

SVM是一种二分类模型，处理的数据可分为三类：

1. 线性可分，通过硬间隔最大化，学习线性分类器
2. 近似线性可分，通过软间隔最大化，学习线性分类器
3. 线性不可分，通过核函数以及软间隔最大化，学习非线性分类器

对于求解（近似）线性可分问题：

- 由最大间隔法，得到凸二次规划问题，这类问题是有最优解的
- 我们得到以上凸优化问题的对偶问题，一是因为对偶问题更容易求解，二是引入核函数，推广到非线性问题。
- 求解对偶问题得到原始问题的解，进而确定分离超平面和分类决策函数。由于对偶问题里目标函数和分类决策函数只涉及实例与实例之间的内积，即**<xi,xj>**。我们引入核函数的概念。

拓展到求解线性不可分问题：

- 对于线性不可分的数据集的任意两个实例：**xi,xj**。当我们取某个特定映射**f**之后，**f(xi)**与**f(xj)**在高维空间中线性可分，运用上述的求解（近似）线性可分问题的方法，我们看到目标函数和分类决策

函数只涉及内积**<xi,xj>**。由于高维空间中的内积计算非常复杂，我们可以引入核函数**K(xi,xj)=<f(xi),f(xj)>**，因此内积问题变成了求函数值问题。

核函数

通过核函数将某个特征空间映射到另一个特征空间，可以将原问题的内积运算替换成核函数。

最终求解目标

求解优化目标函数

- $max_\alpha[\sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y_i y_j K(X_i^T X_j)]$
- 约束条件：$\alpha_i \geq 0 \quad \sum_{i=1}^m \alpha_i y_i = 0$

引入松弛变量后，优化目标保持不变，约束条件变为

- 约束条件：$C \geq \alpha_i \geq 0 \quad \sum_{i=1}^m \alpha_i y_i = 0$

对于b

- $b = y_i - \sum_{j=1}^m \alpha_j y_j K(X_i, X_j)\ for\ any\ i\ that\ \alpha_i \neq 0$
- $b = y_i - \sum_{j=1}^m \alpha_j y_j K(X_i, X_j)\ for\ any\ i\ with\ maximal\ \alpha_i$
- $b = avg_{i:\alpha_i \neq 0}(y_i - \sum_{j=1}^m \alpha_j y_j K(X_i, X_j))$

对于w

- $w = \sum_{i=1}^m \alpha_i y_i X_i = \sum_{i \in SV} \alpha_i y_i X_i$

预测

$y^* = sign(\sum_{i \in SV} \alpha_i y_i K(X_i, X^`) + b)$

SMO算法求解α最优值

SMO表示序列最小化，将大优化问题分解为多个小优化问题来求解的。SMO算法的目标是求出一系列α和b,一旦求出了这些α, 就很容易计算出权重向量w，并得到分隔超平面。

工作原理：**每次循环中选择两个α进行优化处理。一旦找到一对合适的α,那么就增大其中一个同时减小另一个。这里所谓的"合适" 就是指两个α必须要符合一定的条件，条件之一就是这两个α必须要在间隔边界之外，而其第二个条件则是这两个α还没有进行过区间化处理或者不在边界上。Platt SMO算法是通过一个外循环来选择第一个α 值的，并且其选择过程会在两种方式之间进行交替：一种方式是在所有数据集上进行单遍扫描，另一种方式则是在非边界α中实现单遍扫描。而所谓非边界α指的就是那些不等于边界0或C的α值。在选择第一个α值后，算法会通过一个内循环来选择第二个α值。在优化过程中，会通过最大化步长的方式来获得第二个α值。**

SMO工作过程

```
创建一个alpha向量并将其初始化为0向量
当迭代次数小于最大迭代次数时（外循环）
        对数据集中的每个数据向量（内循环）
                如果该数据向量可以被优化：
                        选择另外一个数据向量
                        同时优化这两个向量
                        如果两个向量都不能被优化，退出内循环
        如果所有向量都没被优化，增加迭代数目，继续下一次循环
```

部分主要代码

实现了支持软间隔与线性核、多项式核和径向基函数核的SVM。

```
def SVM(train_set, train_label, test_set, c=200, epsilon=0.001, maxIter=60000,
kernel=("linear", 0)):
    """
    SVM主函数，支持向量机
    :param train_set: 训练集特征
    :param train_label: 训练集标签
    :param test_set: 测试机特征
    :param c: 软间隔的参数,默认值为200
    :param epsilon:容错率，停止训练的误差精度
    :param maxIter:最大迭代次数
    :param kernel:核函数
    :return:predictLabel：预测测试集的标签
    """
    b,alphas = smoP(train_set,train_label,c,epsilon,maxIter,kernel)
    w=caclW(alphas,train_set,train_label)
    test_set = mat(test_set)
    train_set = mat(train_set)
    train_label = mat(train_label).T
    svIndex = nonzero(alphas.A > 0)[0] #找出alphas中大于0的元素位置
    sv_set = train_set[svIndex]  #获得支持向量的矩阵
    sv_label = train_label[svIndex] #获得支持向量的标签
    m,n=shape(test_set)
    predictLabel = []
    for i in range(m):
        kernelValue = kernelTrans(sv_set,test_set[i,:],kernel)
        predictLabel.append(sign(kernelValue.T *
multiply(sv_label,alphas[svIndex])+b))
    return predictLabel

def smoP(train_set, train_label, c, epsilon, maxIter, kernel=("linear", 0)):
    """
    Platter SMO算法，求解α和b值
    :param train_set: 训练集特征
    :param train_label: 训练集标签
    :param test_set: 测试机特征
    :param c: 软间隔的参数
    :param epsilon:精度
    :param maxIter:最大迭代次数
    :param kernel:核函数
    :return:SVMClassifier.b:求得的b值
    :return:SVMClassifier.alphas：求得的α矩阵
    """
    SVMClassifier = SVMdata(mat(train_set), mat(train_label).T, c, epsilon,
kernel)
    iter = 0  # 计算循环的次数
    entireSet = True  # α被初始化为零向量，所以先遍历整个样本集
    alphaPairsChanged = 0
    while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):
        # 循环在整个样本集或非边界点集上切换，达到最大循环次数时退出
        alphaPairsChanged = 0
        if entireSet:  # 循环遍历整个样本集
            for i in range(SVMClassifier.m):  # 外层循环
                alphaPairsChanged += innerL(SVMClassifier, i)
            iter += 1
        else:  # 循环遍历非边界点集
            nonBoundIs = nonzero((SVMClassifier.alphas.A > 0) *
(SVMClassifier.alphas.A < c))[0]
            for i in nonBoundIs:  # 外层循环
```

```
                alphaPairsChanged += innerL(SVMClassifier, i)
            iter += 1
        if entireSet:
            entireSet = False
        elif (alphaPairsChanged == 0):  # 非边界点全部满足KKT条件，循环遍历整个样本集
            entireSet = True
    return SVMClassifier.b, SVMClassifier.alphas
```

实验结果说明与分析

- **student-mat.csv**

```
SVM
student-mat.csv
训练集大小 276
测试集大小 119
未指明的参数默认c(惩罚参数)=200,epsilon(容错率)=0.01,maxInter(最大迭代次数)=60
kernel:linear
          c=200  c=500
all       0.610  0.916
noG1G2    0.413  0.304
onlyG1G2  0.816  0.816
kernel:poly (a*Xi*Xj+b)^p
          a=1,b=1,p=2  a=1,b=1,p=3  a=1,b=1,p=4  a=1,b=1,p=5
all             0.816        0.623        0.713        0.779
noG1G2          0.816        0.600        0.725        0.610
onlyG1G2        0.816        0.308        0.565        0.682
kernel:Gaussian
          gamma=2  gamma=4  gamma=8
all         0.816    0.920    0.936
noG1G2      0.816    0.788    0.817
onlyG1G2    0.880    0.886    0.923
```

对各种特征组合进行算法调用，每种特征组合使用多种参数调用，发现无G1G2特征性能较差，修改参数单独调用如下图

```
SVM
student-mat.csv
训练集大小 276
测试集大小 119
未指明的参数默认c(惩罚参数)=400,epsilon(容错率)=0.001,maxInter(最大迭代次数)=10
kernel:linear
         c=200  c=500
noG1G2  0.785  0.798
kernel:poly (a*Xi*Xj+b)^p
         a=1,b=1,p=2  a=1,b=1,p=3  a=1,b=1,p=4  a=1,b=1,p=5
noG1G2         0.816        0.600        0.727        0.610
kernel:Gaussian
         gamma=2  gamma=4  gamma=8
noG1G2     0.816    0.788    0.798
```

```
SVM
student-mat.csv
训练集大小 276
测试集大小 119
未指明的参数默认c(惩罚参数)=400,epsilon(容错率)=0.001,maxInter(最大迭代次数)=600
kernel:linear
          c=200  c=500
all       0.610  0.887
onlyG1G2  0.850  0.920
kernel:poly (a*Xi*Xj+b)^p
          a=1,b=1,p=2  a=1,b=1,p=3  a=1,b=1,p=4  a=1,b=1,p=5
all             0.828        0.534        0.713        0.814
onlyG1G2        0.816        0.281        0.565        0.597
kernel:Gaussian
          gamma=2  gamma=4  gamma=8
all         0.816    0.920    0.915
onlyG1G2    0.872    0.930    0.908
```

- **student-por.csv**

```
SVM
student-por.csv
训练集大小 454
测试集大小 195
未指明的参数默认c(惩罚参数)=200,epsilon(容错率)=0.01,maxInter(最大迭代次数)=100
kernel:linear
          c=200  c=500
all       0.940  0.940
noG1G2    0.850  0.850
onlyG1G2  0.946  0.944
kernel:poly (a*Xi*Xj+b)^p
          a=1,b=1,p=2  a=1,b=1,p=3  a=1,b=1,p=4  a=1,b=1,p=5
all             0.917        0.396        0.771        0.659
noG1G2          0.000        0.674        0.656        0.718
onlyG1G2        0.917        0.390        0.605        0.566
kernel:Gaussian
          gamma=2  gamma=4  gamma=8
all         0.917    0.964    0.947
noG1G2      0.917    0.922    0.904
onlyG1G2    0.801    0.934    0.890
```

```
SVM
student-por.csv
训练集大小 454
测试集大小 195
未指明的参数默认c(惩罚参数)=200,epsilon(容错率)=0.01,maxInter(最大迭代次数)=60
kernel:linear
          c=200  c=500
all       0.909  0.939
noG1G2    0.912  0.912
onlyG1G2  0.931  0.931
```

- **对比两个文件，发现在student-por.csv上使用相关算法的性能普遍高于student-mat.csv，这可能与前者数据集更大有关**
- **对同一数据集调用不同算法时，一般只使用G1G2特征和使用全部特征性能相当，但不使用G1G2特征的数据性能差些，由此说明G3特征与G1G2特征有很强的相关性。**
- **关于核函数，发现对于预测G3，多项式核函数性能较差，并且随着最高项次数的增加性能越来越差，因此本数据不适合使用多项式核函数。对于线性核函数，通过调参，使用student-por.csv数据的性能达到较好水平，但student-mat.csv的性能一般。对于整个实验，可以发现核函数Gaussian (径向基函数)有着非常好的性能，因此使用核函数Gaussian是一个好的选择。**

Logistic 回归算法

算法思想

根据现有数据对分类边界线建立回归公式，以此进行分类。为了实现Logistic回归分类器，我们可以在每个特征上都乘以一个回归系数，然后把所有的结果值相加，将这个总和代入Sigmoid函数中，进而得到一个范围在0~1之间的数值。任何大于0.5的数据被分入1类，小于0.5即被归入0类。所以，Logistic回归也可以被看成是一种概率估计。为此，我们通过梯度上升算法计算最佳回归系数。梯度上升基于的思想是：要找到某函数的最大值，最好的方法是沿着该函数的梯度方向探寻。

Sigmoid函数:$\frac{1}{1+e^{-z}}$

梯度上升的迭代公式：$w = w + \alpha \nabla_w f(w)$

工作过程

```
每个回归系数初始化为1
重复R次:
    计算整个数据集的梯度
    使用alpha × gradient更新回归系数的向量
    返回回归系数
```

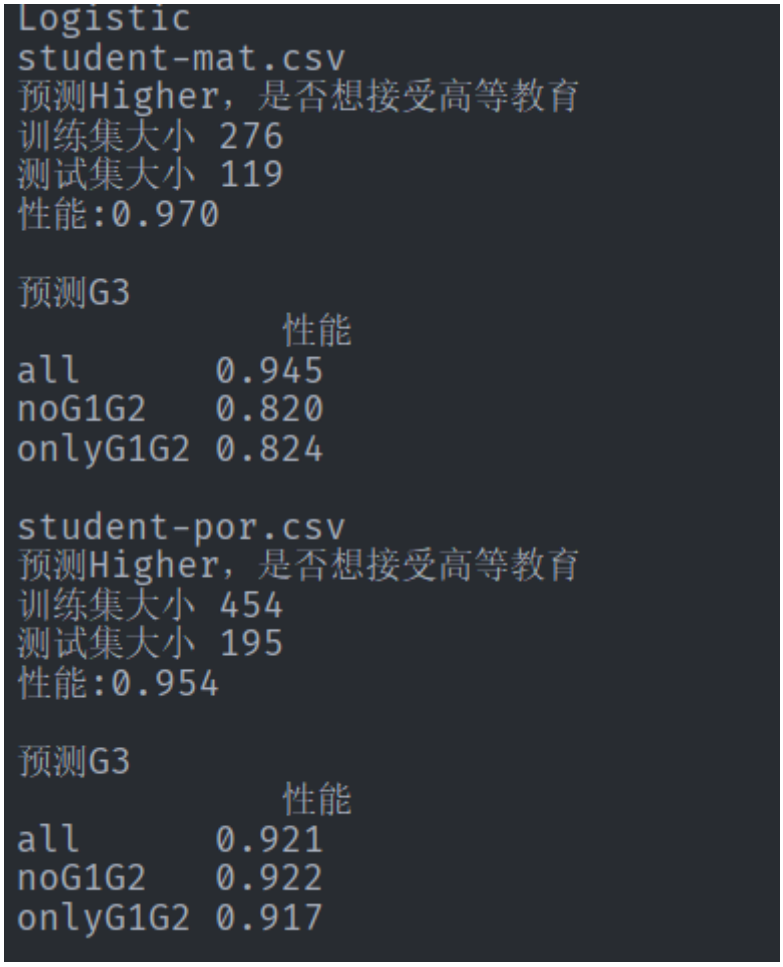
主要代码

```
def Logisitic(train_set, train_label, test_set):
    """
    Logisitic算法主函数，根据训练集训练出的模型预测测试集的标签
    :param train_set:训练集特征(1,0)
    :param train_label:训练集标签
    :param test_set:测试集特征(1,0)
    :return:
    """
    weights = gradAscent(train_set, train_label)
    test_set = np.mat(test_set)
    m, n = test_set.shape
    predictLabel = []
    for i in range(m):
        predictLabel.append(classifier(test_set[i, :], weights))
    return predictLabel

def gradAscent(train_set, train_label):
    """
    梯度上升算法求回归系数
    :param train_set: 训练集特征
    :param train_label:训练集标签
    :return weights:返回的回归系数矩阵
    """
    train_set = np.mat(train_set)
    train_label = np.mat(train_label).transpose()
    m, n = train_set.shape
    alpha = 0.0001    # 移动步长，控制更新的弧度
    maxIter = 500    # 最大迭代次数
    weights = np.ones((n, 1))
    for k in range(maxIter):
        h = sigmoid(train_set * weights)
        error = train_label - h
        weights = weights + alpha * train_set.T * error
    return weights
```

实验结果说明与分析

预测G3，同时选择与家庭背景有关的特征（address,famsize,Pstatus,Medu,Fedu,Mjob,Fjob)预测"higher"(是否想接受高等教育)



- **发现性能均大于0.9，说明是否想接受高等教育与家庭背景有较大关系，预测效果不错**
- **对于预测G3，在student-por.csv数据集上性能均超过了0.9，在student-mat.csv数据集上效果也不错，但不如student-por.csv**

总结

- **实现了KNN、SVM、Logistic算法，对一些经典的机器学习算法有了更加深入的了解**
- **对比三种算法，KNN算法简单，精度高，对异常值不敏感，且无数据输入假定，但计算复杂度、空间复杂度高；支持向量机一般错误率较低，但对参数调节和核函数的选择敏感，可能需要多次调参才能得出好的结果；Logistic回归算法计算代价不高，易于理解和实现，但容易欠拟合，分类精度可能不高**