2022 - 2023 学年第 1 学期

机器学习课程实验报告

姓 名：	张建才
学 号：	209074458
专 业：	智能 202

安 徽 工 业 大 学

计算机科学与技术学院

目 录

实验 1 3
实验 2 4
实验 3 6
实验 4 7
实验 5 8

实验 1

实验内容：

实验步骤：

实验体会：

LSM：最小二乘法需要求解最优参数w^*，

$$J(w)=\frac{1}{2m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right)-y^{(i)}\right)^2$$

其中：$h(x)=w^TX=w_0x_0+w_1x_1+w_2x_2+\ldots+w_nx_n$

最小二乘法将向量表达形式转为矩阵表达形式可以解出最优参数的值：

$$w^*=\left(X^TX\right)^{-1}X^Ty$$

```
def LSM(X, y):
    w = np.linalg.inv(X.T@X)@X.T@y #np.linalg.inv(): 矩阵求逆，X.T@X等价于X.T.dot(X)
    return w
```

```
final_w2=LSM(np.array(X), np.array(y))
final_w2
```

```
array([[13254.93845442]])
```

$J(w)=\frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}w_j^2$，此时称作Ridge Regression:

```
# 划分训练集和测试集
cols = data.shape[1]
X = data.iloc[:,:cols-1] # X是所有行，去掉最后一列
y = data.iloc[:,cols-1:] # y是所有行的最后一列
X_train,X_test,y_train,y_test = train_test_split(data.iloc[:,:cols-1],data.iloc[:,cols-
1:],random_state=0)
```

```
#创建岭回归实例
clf=Ridge(alpha=1.0,fit_intercept = True)
#调用fit函数使用训练集训练岭回归器
clf.fit(X_train,y_train)
#利用测试集计算回归曲线的拟合优度，clf.score返回值为0.7375
#拟合优度，用于评价拟合好坏，最大为1，无最小值，当对所有输入都输出同一个值时，拟合优度为0。
clf.score(X_test,y_test)
```

```
0.977[illegible]0519315693
```

深刻理解了数据处理时预处理的重要性，数据类型的转换关系。
同时，对岭回归的参数也有了深刻认识。
对数据训练过程也有进一步的认识。

实验 2

实验内容：

实验步骤：

实验体会：

感知机基本原理：

假定训练数据集是基本可分，通过定义损失函数，并使之最小，就达到了二分类的目的。

假设输入空间是$\mathcal{X} \subseteq \mathbf{R}^n$输出空间是$\mathcal{Y} = \{+1, -1\}$输入 $x \in \mathcal{X}$表示实例的特征向量。输出$y \in \mathcal{Y}$表示实例的类别。由输入空间到输出空间感知机是如下函数：

$$f(x) = sign(w \cdot x + b)$$

$w \in \mathbf{R}^n$叫做权值， $b \in \mathbf{R}$叫做偏置。$sign$是符号函数

$$sign(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

参数调整之前：

```
X_train,X_test,y_train,y_test=train_test_split(\
    df.loc[:,df.columns !='Outcome'],\
    df['Outcome'],stratify=df['Outcome'],random_state=66)
```

Python

```
clf = Perceptron(fit_intercept=True,max_iter=1000,shuffle=False,n_jobs=-1,)
clf.fit(X_train, y_train)
print(clf.coef_)
print(clf.intercept_)
```

0.8s Python

```
[[1094.     99.    -383.     32.     104.    -129.6    -5.108 -134.   ]]
[-266.]
```

```
acc = clf.score(X_test,y_test)    # 使用测试集进行验证
print(acc)

```

0.7s Python

```
0.65625
```

参数调整之后：

```python
X_train,X_test,y_train,y_test=train_test_split(\
    df.loc[:,df.columns !='Outcome'],\
    df['Outcome'],stratify=df['Outcome'])
```

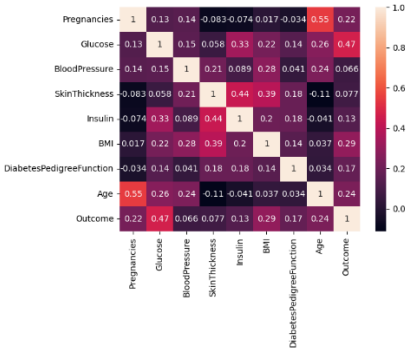
```python
clf = Perceptron(fit_intercept=True,tol = None,max_iter=100000,shuffle=True,n_jobs=-1,)
clf.fit(X_train, y_train)
print(clf.coef_)
print(clf.intercept_)
```

```
[[ 2.1640000e+03  4.6300000e+02 -1.4800000e+02  4.5000000e+01
  -1.3000000e+01  1.5595000e+03  1.4403799e+04  1.1900000e+02]]
[-124890.]
```

```python
acc = clf.score(X_test,y_test)    # 使用测试集进行验证
print(acc)
```

```
0.7864583333333334
```

额外：



从上述热力图可以看出一些明显的特征，如糖尿病的标签 Outcome 和葡萄糖测试值 Glucose 正相关系数比较大，说明葡萄糖测试值高的话，有可能患有糖尿病。同理，年龄 Age 和怀孕次数 Pregnancies 之间的相关性也比较强。

实验 3

实验内容：

实验步骤：

实验体会：

假设数据服从这个分布，然后使用极大似然估计做参数的估计。

假设函数，损失函数，

$$w^T x + b = ln\frac{P(Y=1|x)}{1-P(Y=1|x)}$$

$$P(Y=1|x) = \frac{1}{1+e^{-(w^T x+b)}}$$

分类概率 P(Y=1)，给定数据集 D：

$$D = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N), x_i \subseteq R^n, y_i \in 0, 1, i = 1, 2, \cdots, N$$

使用$w^T x + b$拟合条件概率$p(Y=1|x)$

当$w^T x + b$的值越接近正无穷，$p(Y=1|x)$概率值也就越接近 1。

因此逻辑回归的思路是，先拟合决策边界(不局限于线性，还可以是多项式)，再建立这个边界与分类的概率联系，从而得到了二分类情况下的概率。

实验内容：

```
# 数据归一化后，使用sklearn库中的LogisticRegression算法进行分类
# 首先设置模型的相关参数，模型在使用的时候，模型中参数的选取是十分重要的

# 例，此处设置算法的最大迭代次数max_iter为100，其他参数可自行设置
lr = LogisticRegression(max_iter=100)

# 模型训练
lr.fit(train_image,train_label.ravel())

# 模型验证
predict = lr.predict(test_image)

# 输出相关评价指标值
print("accuracy score: %.4lf"% accuracy_score(predict,test_label))
print("classfication report for %s:\n%s\n"%(lr, classification_report(test_label, predict)))

# 代码运行中，请等待.....
18s
accuracy score: 0.8565
```

使用时不需要创建模型比较方便。

因为循环次数少，却可以达到一个比较高的精度，可能这个数据集之间的关系比较明确。

实验 4

实验内容：

实验步骤：

实验体会：

支持向量机是一类按监督学习方式对数据进行二元分类的广义线性分类器其决策边界是对学习样本求解的最大边距超平面。

为了把两组数据分开，在空心点的类别找到一个或多个点离实心点最近，在实心点中找到一个或多个点与空心点最近，分类实心和空心点取决于这些边界上的点而与离边界较远的点，即在分割两类别点的时候，只需要考虑支持向量，通过支持向量确定分割直线。假设此直线有宽度，左边贴合一个边界点，右边贴和另一组，实现把点分开并且宽度最大。

$$\min_{\mathbf{w},b,\xi\geq 0} \quad \frac{1}{2}\mathbf{w}^{\top}\mathbf{w} + C\sum_{i}\xi_i$$

$$\text{s.t.} \quad y_i(\mathbf{w}^{\top}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n$$

$$\xi_i \geq 0$$

$$h(\mathbf{x}) = sign(\mathbf{w}^{\top}\mathbf{x} + b)$$

内容：

```
modelSVM = SVC(kernel='linear', C=100,cache_size=2048)  # SVC 建模: 使用 SVC类, 线性核函数

modelSVM.fit(train_image,train_label)  # 用样本集 X,y 训练 SVM 模型

C:\Users\zjc20\AppData\Roaming\Python\Python311\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A column-vector y was pass
  y = column_or_1d(y, warn=True)
C:\Users\zjc20\AppData\Roaming\Python\Python311\site-packages\sklearn\svm\_base.py:301: ConvergenceWarning: Solver terminated early (max_iter=100
  warnings.warn(
SVC(C=100, cache_size=1024, kernel='linear', max_iter=10000)

modelSVM = SVC(kernel='linear', C=100,cache_size=2048)  # SVC 建模: 使用 SVC类, 线性核函数
modelSVM.fit(train_image,train_label)  # 用样本集 X,y 训练 SVM 模型

predict = modelSVM.predict(test_image)

print("accuracy score: %.4lf"% accuracy_score(predict,test_label))
print("classfication report for %s:\n%s\n"%(modelSVM, classification_report(test_label, predict)))
accuracy score: 0.6939
```

sklearnSVM 的运算使用 CPU，一次计算不方便。

尝试使用 thundersvm 库进行计算。

实验 5

实验内容：

实验步骤：

实验体会：

决策树思想：

输入： 训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
　　属性集 $A=\{a_1,a_2,\ldots,a_d\}$.
过程： 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A=\varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: 　**if** D_v 为空 **then**
12: 　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: 　**else**
14: 　　以 TreeGenerate(D_v, $A\setminus\{a_*\}$)为分支结点
15: 　**end if**
16: **end for**
输出： 以 node 为根结点的一棵决策树

[1]

信息熵是度量样本集合纯度最常用的一种指标。假定当前样本集合 D 中第类样本所占的比例来为 $p_k\ (k=1,2,...,|y|)$，则有：

$$Ent(D)=-\sum_{k=1}^{|y|}p_k\log_2 p_k$$

实验内容：

```
BC = load_breast_cancer()
x = BC['data']
y = BC['target']

print(x)
print(y)
```

[1] https://blog.csdn.net/weixin_43182102/article/details/122106265

```python
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier() # 这里调整模型参数
dt_model.fit(x_train, y_train)
print('训练出来的决策树模型为: \n', dt_model)
```

```
训练出来的决策树模型为:
 DecisionTreeClassifier()
```

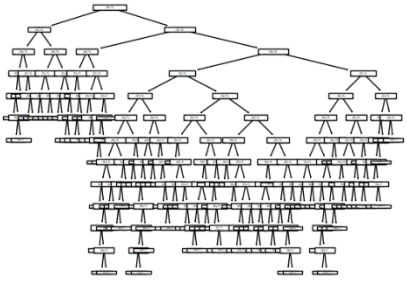
```python
print('在整个测试集上的准确率为: ', dt_model.score(x_test, y_test))
```

```
在整个测试集上的准确率为:  0.9064327485380117
```

```python
DG = load_digits()
X = DG['data']
y = DG['target']
```

```python
print('在整个测试集上的准确率为: ', dt_model.score(x_test, y_test))
```

```
在整个测试集上的准确率为:  0.8203703703703704
```



决策树是基于树结构来进行决策的，一棵决策树包含一个根结点、若干个内部节点和若干个叶结点，最终目的是将样本越分越纯。