机器学习应用实践（实验二）—逻辑回归

一、实验目的

1、掌握逻辑回归算法的原理。

2、掌握 Scikit-Learn 中逻辑回归算法的使用，以及特征可视化和绘制（线性和非线性）决策边界的方法。

3、掌握 OvR、OvO、MvM 等多分类策略，及 softmax 回归的原理及编程实现。

4、掌握 Scikit-Learn 中多种分类评估指标的函数使用方法。

5、熟悉手动调整模型超参数提高泛化能力的方法。

6、熟悉 Scikit-Learn 自带分类数据集（iris 数据集）的使用。

7、具备使用 python 实现二分类和多分类逻辑回归算法的编程能力。

二、实验准备

1、回顾逻辑回归算法的原理。

2、学习 Scikit-Learn 中逻辑回归算法涉及到的相关知识，认真学习“三、相关知识介绍”。

三、相关知识介绍

1、Scikit-Learn 逻辑回归相关 API 介绍

Scikit-Learn 中的 linear_model 模块中提供了逻辑回归类：LogisticRegression。

sklearn.linear_model.LogisticRegression(solver=' lbfgs ', penalty='l2', C=1.0)，默认将类别数量少的当做正例，参数表如表 1。

表 1 LogisticRegression 超参数表

参数	**解释**
penalty	正则化项
dual	是否求解对偶问题
tol	迭代停止条件：两轮迭代损失值差值小于 tol 时，停止迭代
C	经验风险和结构风险在损失函数中的权重
fit_intercept	线性方程中是否包含截距项
intercept_scaling	相当于此前讨论的特征最后一列全为 1 的列，当使用 liblinear 求解参数时用于捕获截距
class_weight*	各类样本权重
random_state	随机数种子
solver*	损失函数求解方法
max_iter	求解参数时最大迭代次数，迭代过程满足 max_iter 或 tol 其一即停止迭代
multi_class*	多分类问题时求解方法
verbose	是否输出任务进程
warm_start	是否使用上次训练结果作为本次运行初始参数
l1_ratio	当采用弹性网正则化时，$l1$ 正则项权重，就是损失函数中的ρ

●solver：优化求解方式。可选择的求解器包括：‘newton-cg’, ‘newton-cholesky’,‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’。

1）liblinear：使用了开源的 liblinear 库实现，内部使用了坐标轴下降法来迭代优化损失函数，相当于 SGDClassifier(loss="log")。

2）lbfgs：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数（sklearn 版本在 0.22 之后默认为这个求解器）。

3）newton-cg：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

4）sag：即随机平均梯度下降，是随机梯度下降的加速版，每次更新梯度时，利用了两个梯度的值，一个是前一次迭代的梯度值，另一个是新的梯度值。

不同的求解器支持不同的正则项和多分类策略，具体见下表（sklearn 版本 1.2）：

	Solvers					
Penalties	**‘lbfgs’**	**‘liblinear’**	**‘newton-cg’**	**‘newton-cholesky’**	**‘sag’**	**‘saga’**
Multinomial + L2 penalty	yes	no	yes	no	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	no	yes
OVR + L1 penalty	no	yes	no	no	no	yes
Elastic-Net	no	no	no	no	no	yes
No penalty (‘none’)	yes	no	yes	yes	yes	yes
Behaviors						
Penalize the intercept (bad)	no	yes	no	no	no	no
Faster for large datasets	no	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	yes	no	no

●penalty：正则化的种类。包括 ‘l1’, ‘l2’, ‘elasticnet’, ‘none’，默认为’l2’正则项。

●C：正则化强度的倒数，必须是正浮点数。与支持向量机一样，较小的值对应更强的正则化。默认为 1.0。

●class_weight：各类样本在进行损失函数计算时的数值权重（只能用于训练）。例如假设一个二分类问题，0、1 两类的样本比例是 2.5:1，此时可以输入一个字典类型对象用于说明两类样本在进行损失值计算时的权重，输入:{0:1, 1:2.5}，代表 1 类样本的每一条数据在进行损失函数值计算时都会在原始数值上乘以 2.5。而当我们将该参数选为 balanced 时，则会自动将这个比例调整为真实样本比例的反比，以达到平衡的效果。默认为‘None’。

●multi_class：多分类选项。默认为‘auto’，能根据数据情况自动选择‘ovr’或‘multinomial’，它们在二分类情况下效果相同，多分类时分类效果‘multinomial’>‘ovr’，分类速度‘ovr’<‘multinomial’。所以二分类会自动选择‘ovr’，多分类会自动选择‘multinomial’。

‘ovr’多分类会在后面进行介绍，‘multinomial’多分类不是‘MvM’，而是交叉熵损失(cross-entropy loss) +softmax。详见官网址：https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression。

交叉熵损失(cross-entropy loss)的相关介绍，详见 sklearn.metrics.log_loss，官网网址：

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html?highlight=cross%20entropy%20loss 和 https://scikit-learn.org/stable/modules/model_evaluation.html#log-loss。

示例程序：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

2、Multiclass classification（多类分类问题）

Multiclass classification（多类分类），就是一个分类任务需要对多于两个类的数据进行分类。比如，对一系列的橘子，苹果或者梨的图片进行分类。多类分类假设每一个样本有且仅有一个标签，一个水果可以被归类为苹果，也可以是梨，但不能同时被归类为两类。

多分类问题解决的基本思想是“先拆分，再集成”。先将数据集进行拆分，根据拆分数据集训练多个二分类模型，然后预测阶段将多个分类器的预测结果进行集成，得到最终的预测结果。依照这种思路设计的多分类策略主要有 3 种：“一对其余”（简称 OvR）、“一对一”（简称 OvO）、“多对多”（简称 MvM）。

Scikit-Learn 中的分类器（如 LogisticRegression 和 LinearSVC 等）都可以通过设置‘multi_class’参数实现多类分类。除此之外，Scikit-Learn 还提供了三种多分类元估计器（meta-estimator）：“one-vs-the-rest / one-vs-all”、“one-vs-one”和“error correcting output codes”。他们必须与基估计器(base estimator，如 LogisticRegression 和 LinearSVC 等)一起使用，能够帮助提高准确性和运行性能。

（1）One-vs-the-Rest，简称 OvR，也叫 One-vs-All，1 对其余分类器。

OvR 的思想是依次将每个类别作为正例，其余类别统一作为反例来构造二分类器，这样可以构造出 N 个二分类器。将新的测试样本放入 N 个二分类器中，选择正例概率值最大的类别作为预测类别。

假设数据集为 $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$，$y_i \in \{C1,C2,C3,C4\}$。对于现有 4 个类构造分类器如表 2 所示：

表 2 OvR 多分类器构造

正例（+）	反例（-）	分类器
C1	C2，C3，C4	f1
C2	C1，C3，C4	f2
C3	C1，C2，C4	f3

C4	C1，C2，C3	f4

在对某个预测样本进行预测时，如表 3 所示，预测结果中 f2 和 f3 均预测为正例，选择正例概率大的作为最终预测结果，为 C2。

表 3 某一预测样本的 OvR 多分类器预测结果

分类器	预测结果（概率）	最终预测结果
f1	-	C2
f2	+（0.8）	
f3	+（0.7）	
f4	-	

LogisticRegression 方法中设置参数 multi_class =‘ovr’，就是这种多分类策略。

Scikit-Learn 中 OvR 元估计器为：sklearn.multiclass.OneVsRestClassifier。

sklearn.multiclass.OneVsRestClassifier(estimator, *, n_jobs=None)

●estimator：要使用的分类模型，可以是 LogisticRegression 和 LinearSVC 等。

●n_jobs：用于计算的工作数量：N 个 OvR 分类器同时进行计算。n_jobs 等于-1 时，使用所有处理器工作。

示例程序：

```
>>> import numpy as np
>>> from sklearn.multiclass import OneVsRestClassifier
>>> from sklearn.svm import SVC
>>> X = np.array([
...     [10, 10],
...     [8, 10],
...     [-5, 5.5],
...     [-5.4, 5.5],
...     [-20, -20],
...     [-15, -20]
... ])
>>> y = np.array([0, 0, 1, 1, 2, 2])
>>> clf = OneVsRestClassifier(SVC()).fit(X, y)
>>> clf.predict([[-19, -20], [9, 9], [-5, 5]])
array([2, 0, 1])
```

（2）One-vs-One，简称 OvO，1 对 1 分类器。

OvO 的思想是每个分类器只挑选所有类别中的两类做二分类器，这样对于 N 分类问题，需要训练C_N^2=N*(N-1)/2 个分类器。在预测阶段，将所有分类器的结果放在一起，收到最多投票的类别即为预测类别。当存在两个类具有同样的票数的时候，1 对 1 分类器会选择总分类置信度最高的类。

假设数据集为 D={$(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)$}，$y_i \in$ {C1,C2,C3,C4}。对于现有 4 个类构造分类器如表 4 所示：

表 4 OvO 多分类器构造

正例	反例	分类器

C1	C2	f1
C1	C3	f2
C1	C4	f3
C2	C3	f4
C2	C4	f5
C3	C4	f6

总共构造 6 个二分类器，之后对于新的测试样本，只需将样本输入构造的每个二分类器中，最后采用投票方式取预测得到票数最多的那一类为预测类别，如表 5 所示。C2 类得票最多，则将其预测为 C2 类。

表 5 某一测试样本的 OvO 多分类预测结果

分类器	预测类别	最终预测类别
f1	C2	C2
f2	C1	
f3	C4	
f4	C2	
f5	C2	
f6	C3	

Scikit-Learn 中 OvR 元估计器为：sklearn.multiclass.OneVsOneClassifier(estimator, n_jobs=None)

示例：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.multiclass import OneVsOneClassifier
>>> from sklearn.svm import LinearSVC
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, shuffle=True, random_state=0)
>>> clf = OneVsOneClassifier(
...     LinearSVC(random_state=0)).fit(X_train, y_train)
>>> clf.predict(X_test[:10])
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1])
```

（3）MvM，多对多分类器

MvM 相比于 OvO 和 OvR 更加复杂，常用一种名为“纠错输入码”（Error Correcting Output Codes，简称 ECOC）的技术来实现 MvM 过程。ECOC 的思想是对训练集做 M 次划分，可以任选其中一类作为正例、其余作为反例，也可以任选其中两类作为正例、其余作为反例，以此类推。于是可以训练得到 M 个二分类器{f1,f2,…,f_M}。针对原数据中的一个类别 Ci，按照 M 次划分，可以得到 M 个编码值[+1,-1,-1,+1,…]，称之为编码串，于是对于 N 个类别可以得到一个编码矩阵，其形状为(N,M)。进行预测时，每个预测样本经过 M 个分类器，可以得到 M 个预测结果，形成一个预测编码串，将该编码串与真实类别的编码串进行比较，通过判断跟真实类别的编码串的接近程度获得预测结果。

假设原数据只有 4 个类别，这里选择对数据做 4 次划分，于是得到 4 个二分类器如图 1 所示，则得到编码矩阵如表 6 所示，可以看到构造的分类器的类别编码是和类别一一对应的，一种编码对应一种类别，不同类别编码不同。

图 1 MvM 分类器构建

表 6 ECOC 编码多分类器构造

样本	类别	f1	f2	f3	f4
1	C1	+1	-1	+1	-1
2	C1	+1	-1	+1	-1
3	C2	-1	-1	+1	-1
4	C2	-1	-1	+1	-1
5	C3	-1	-1	-1	+1
6	C3	-1	-1	-1	+1
7	C4	-1	+1	-1	+1
8	C4	-1	+1	-1	+1
新样本		+1	-1	+1	+1

对于新的测试样本，将样本输入训练好的 4 个二分类器，将预测标签作为新样本的编码，会得到一个编码串，如[+1,-1,+1,+1]。要判定新样本属于哪个类别，就需要判断新样本编码与哪个已知类别编码更接近。可以通过计算新样本编码与已知类别编码之间的距离（如欧氏距离、街道距离、闵可夫斯基距离等），与其距离最小的那个类别就作为新样本的最终预测类别。计算得到的新样本编码与各类别的欧氏距离如表 7 所示。显然，与类别 C1 编码距离最小，因此将测试样本预测为类别 C1。

表 7 ECOC 编码多分类器预测

类别	距离	最终预测类别
C1	2.00	C1
C2	2.83	
C3	2.83	

C4	3.46	

示例程序：

```
>>> from sklearn.multiclass import OutputCodeClassifier
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=100, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = OutputCodeClassifier(
...     estimator=RandomForestClassifier(random_state=0),
...     random_state=0).fit(X, y)
>>> clf.predict([[0, 0, 0, 0]])
array([1])
```

3、决策边界基本概念与绘制

决策边界是对不同类型数据进行划分的分类边界，决策边界的可视化能够直观帮助初学者更好地理解模型判别性能如何，以及模型调整对模型性能的影响。图 2、3 中不同颜色代表不同类别的分类空间，不同颜色间的分界线就是决策边界。显然，决策树的决策边界更曲折，相比逻辑回归的线性决策边界，对于非线性数据具有更强的判别能力。

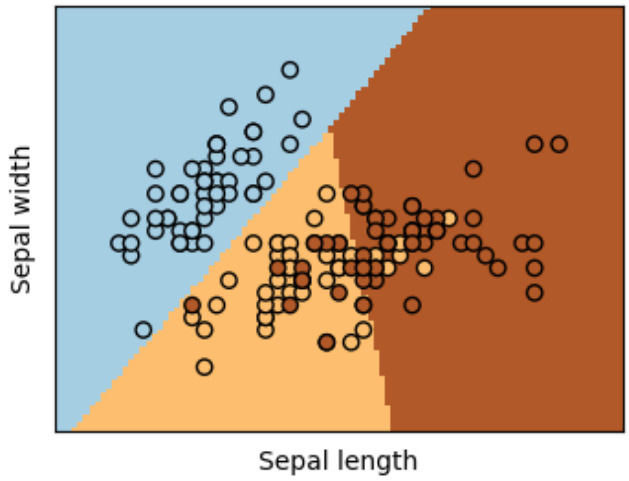


图 2 iris 数据逻辑回归模型决策边界

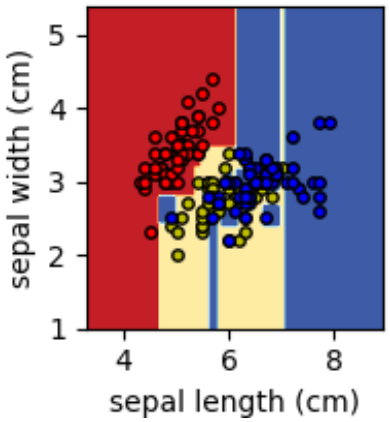


图 3 iris 数据决策树模型决策边界

此外，还能借助决策边界观察模型过拟合倾向，一般来说，模型决策边界越不规则，越是出现不同颜色区域彼此交错的情况，则说明模型越是存在过拟合倾向。

但是，受到平面图形像呈现维度的约束，一般只能观察二维或三维样本空间的决策边界或决策面，对于更高维度样本空间尽管在建模时也存在决策边界，但无法可视化展示。无论如何，在模型学习的初级阶段，从决策边界角度理解模型性能是至关重要的。

二分类决策边界绘制，可根据决策边界表达式直接绘制，多分类决策边界绘制相比二分类更复杂一些，绘制决策边界的思路如下：

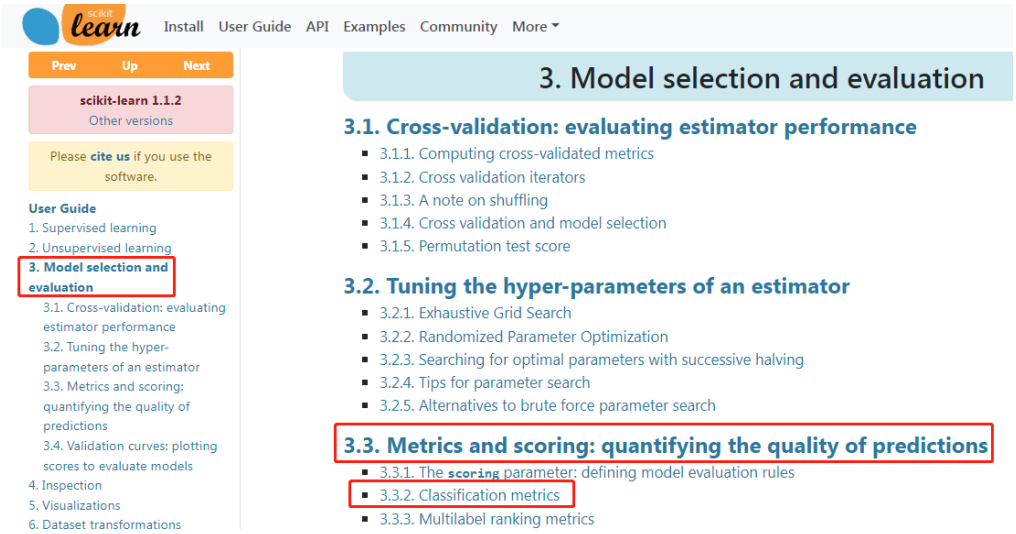
（1）利用训练好的模型对样本空间所有的坐标点进行预测（如 numpy 中 meshgrid 可生成绘图网格数据）；

（2）对模型判别的不同类别的点进行不同着色（如 matplotlib 中 contourf 可对等高线之间进行颜色填充）；

（3）在决策边界图像上添加参与模型训练的样本点。

4、分类问题的常用指标

分类模型作为使用场景最为广泛的机器学习模型，相关模型评估指标也伴随着使用场景的拓展而不断丰富。sklearn 提供了可用于二分类、多分类、多标签等情况的多种模型评估函数，详见官网 https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics。可如下图从 User Guide 中查找。



除了准确率之外，还有查准率(precision)、召回率/查全率(Recall)、F1 指标(F1-Score)、受试者特征曲线(ROC-AUC)、混淆矩阵(Confusion matrix)、分类报告(classification_report)等。

（1）混淆矩阵（Confusion matrix）

混淆矩阵（Confusion matrix）可以更加精细地可视化展示分类模型的结果。对角元素表示预测正确（即预测标签与真实标签相同）的点的数量，而非对角元素是分类错误的点数量。混淆矩阵的对角线值越高，表明预测准确率越高。

详见官网 https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

注意：在 sklearn 中将行与真实标签对应，列与预测标签对应。

示例程序：

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

ConfusionMatrixDisplay 可用于直观地表示混淆矩阵，如图 4 所示，可直观地看出真实标签为‘versicolor’类的样本中有 10 个样本分类正确，有 6 个样本被错分为‘virginica’类。下图中的程序代码详见官网：https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html。

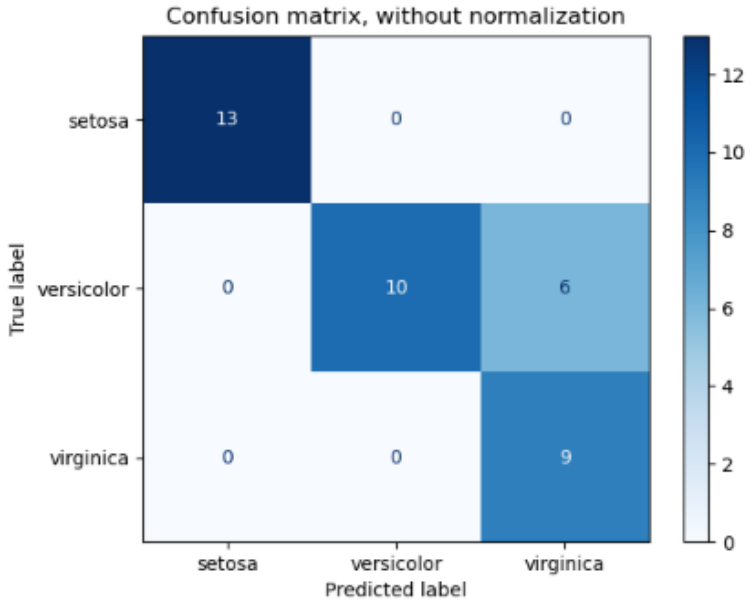


图 4 官网混淆矩阵使用示例

（2）分类报告 classification_report

classification_report 函数能够返回一个显示主要分类指标（包括 precision、recall、F1-Score 等）的文本报告。

●support：表示每个类别真实标签的个数。

●macro avg：表示所有类别对应指标的平均值，示例中 precision 列：(0.67+0+1.00)/3=0.56。

●weighted avg：表示带权重平均，即类别样本占总样本的比重与对应指标的乘积的累加和，例如示例中：(2*0.67+1*0+2*1)/5=0.67。

示例程序：

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 0]
>>> y_pred = [0, 0, 2, 1, 0]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

     class 0       0.67      1.00      0.80         2
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.50      0.67         2

    accuracy                           0.60         5
   macro avg       0.56      0.50      0.49         5
weighted avg       0.67      0.60      0.59         5
```

5、逻辑回归编程实现

表 8 对比了线性回归与逻辑回归算法的模型和参数优化方法的异同，便于同学们在“线性回归”模型的基础上，独立编写逻辑回归算法。

表 8 线性回归与逻辑回归模型和参数优化方法对比

线性回归与逻辑回归模型和参数优化方法对比（梯度下降法）

类别	线性回归 （Linear Regression）	逻辑回归（二分类） （Logistic Regression）
模型	$f_w(x)=\sum_{j=0}^{M} w_j x_j=\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}$ M 为样本的特征/属性的数量+1	$h(x)=\frac{1}{1+e^{-\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}}}$
代价函数	均方误差： $J(\widehat{\boldsymbol{w}})=\frac{1}{2N}\sum_{i=1}^{N}(y_i-\widehat{\boldsymbol{w}}\widehat{\boldsymbol{x}}_i)^2$ N 为训练集的样本量	交叉熵损失： $J(\widehat{\boldsymbol{w}})=-\sum_{x\in X}(yln(h(\boldsymbol{x}))+(1-y)\,ln(1-h(\boldsymbol{x}))$ 即： $J(\widehat{\boldsymbol{w}})=\sum_{x\in X}(-y\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}+ln(1+exp(\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}))$
梯度	$\frac{\partial J(\widehat{\boldsymbol{w}})}{\partial w_j}=\frac{1}{N}\sum_{i=0}^{N}(f_w(x^i)-y^i)x_j^i$	$\frac{\partial J(\widehat{\boldsymbol{w}})}{\partial w_j}=\frac{1}{N}\sum_{i=0}^{N}(h_w(x^i)-y^i)x_j^i$
更新规则	$w_j:=w_j-\alpha\frac{\partial J(\widehat{\boldsymbol{w}})}{\partial w_j}$	
评价指标	均方误差、R^2 等	分类准确度（预测正确数/测试集总数）等

6、交叉熵损失与 softmax 回归

前面已经介绍了 OvR、OvO、MvM 三种多类分类策略，而 Scikit-Learn 中 LogisticRegression 中提供的‘multinomial’多分类方法不是上述中的任何一种，而是交叉熵损失(CrossEntropy Loss)+softmax。“交叉熵损失+softmax”经常被用于（多分类）逻辑回归和（分类问题）神经网络中。

softmax 函数，又称归一化指数函数，可将 K 个连续值转换成 K 个和为 1 的概率值。

softmax 回归其实是逻辑回归的一种变形，逻辑回归模型输出的是两种类别的概率，softmax 回归输出 K 种类别的概率。对于某一样本 i，其 K 分类 softmax 回归的模型计算公式如下：

$$\text{softmax}(\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}_i)=\frac{1}{\sum_{k=1}^{K}e^{\left(\widehat{\boldsymbol{w}}_k{}^T\widehat{\boldsymbol{x}}_i\right)}}\begin{bmatrix}e^{\left(\widehat{\boldsymbol{w}}_1{}^T\widehat{\boldsymbol{x}}_i\right)}\\e^{\left(\widehat{\boldsymbol{w}}_2{}^T\widehat{\boldsymbol{x}}_i\right)}\\\vdots\\e^{\left(\widehat{\boldsymbol{w}}_k{}^T\widehat{\boldsymbol{x}}_i\right)}\\\vdots\\e^{\left(\widehat{\boldsymbol{w}}_K{}^T\widehat{\boldsymbol{x}}_i\right)}\end{bmatrix}^T=\frac{e^{(\widehat{\boldsymbol{w}}^T\widehat{\boldsymbol{x}}_i)}}{\sum_{k=1}^{K}e^{\left(\widehat{\boldsymbol{w}}_k{}^T\widehat{\boldsymbol{x}}_i\right)}}=h(\widehat{\boldsymbol{x}}_i)=\begin{bmatrix}p_{i,1}\\p_{i,2}\\\vdots\\p_{i,k}\\\vdots\\p_{i,K}\end{bmatrix}^T$$

其中，$h(\hat{x}_i)$为样本 i 属于 K 个类别的概率，每个样本属于 K 个类别的概率和为 1。$\hat{x}_i$为样本 i 的特征向量，样本 i 的真实标签为y_i（$y_i\in 1,\dots,K$）。参数$\widehat{\boldsymbol{w}}$是一个矩阵，矩阵的每一行是一个类别所对应分类器的参数，总共有 K 行。$\widehat{\boldsymbol{w}}_k$为模型参数矩阵$\widehat{\boldsymbol{w}}$的第 k 行，即类别 k 对应的分类器参数。最终模型选取概率最高的类别作为最终判定结果。

多分类问题的交叉熵损失为：

$$J(\widehat{\boldsymbol{w}})=-\sum_{i=1}^{N}\sum_{k=1}^{K}y_{i,k}logp_{i,k}$$

其中，$y_{i,k}$为样本 i 属于类别 k 的标签（对真实标签y_i做独热编码处理），$p_{i,k}$为样本 i 预测为类别 k 的概率。

四、实验内容

1、题目一：采用 scikit-learn 中的 LogisticRegression 逻辑回归模型对 iris 数据集进行二分类。

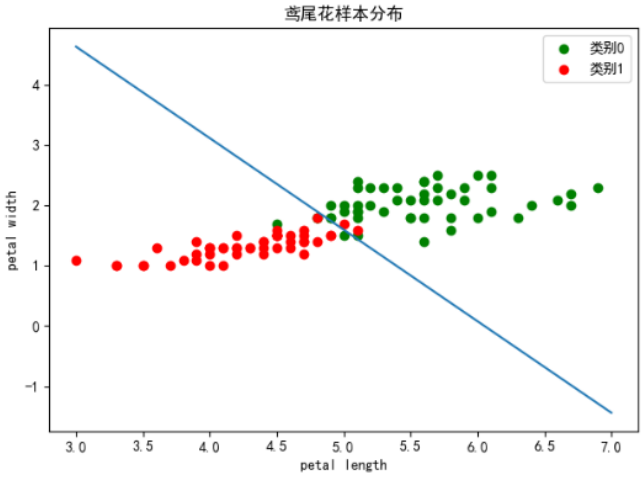
具体内容：

（1）特征可视化：任选两个特征和两种类别进行散点图可视化，观察是否线性可分。

（2）模型建立：使用选取的特征和两种类别建立二分类模型。

（3）输出：决策函数的参数、预测值、分类准确率等。

（4）决策边界可视化：将二分类问题的边界可视化。



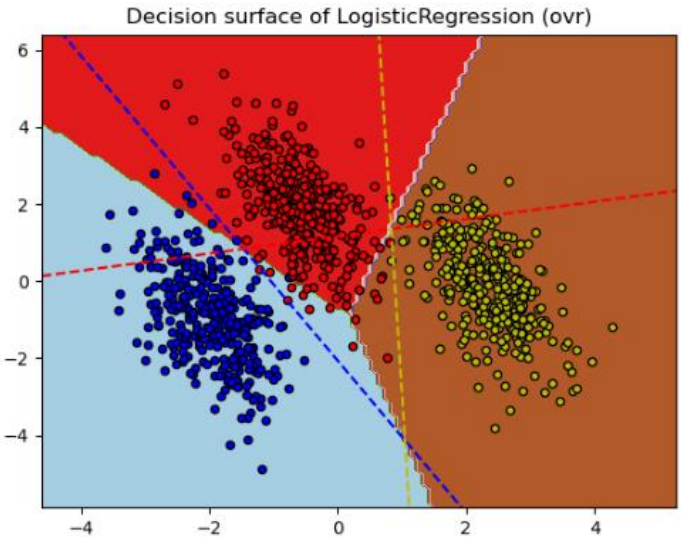
2、题目二：采用 scikit-learn 中的 LogisticRegression 逻辑回归模型对 iris 数据集进行多分类。

具体内容：

（1）模型建立：任选两个特征和全部类别进行散点图可视化，并建立多分类模型。

（2）输出：决策函数的参数、预测值、分类准确率等。

（3）决策边界可视化：将多分类问题的边界可视化。



提示：更为普遍地可使用 numpy 中的 meshgrid 生成绘图网格数据，使用 matplotlib 中的 contourf 将等高线之间颜色进行填充。也可使用 skearn 官网提供的 sklearn.inspection.DecisionBoundaryDisplay

绘制决策边界，具体实现可参考官网

https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html#sphx-glr-auto-examples-linear-model-plot-logistic-multinomial-py

【讨论一】（选做）不同多分类策略的效果如何？有何差异？

（1）尝试对比 LogisticRegression 中的 multi_class =’ovr’或’multinomial’两种多分类的差异。

（2）尝试使用 Multiclass classification 中提供的 3 种多分类策略，并对比效果。

提示：进行对比时，要保证数据集划分一致且分析的特征一致。

可从训练集、测试集准确率，和边界可视化角度进行对比。

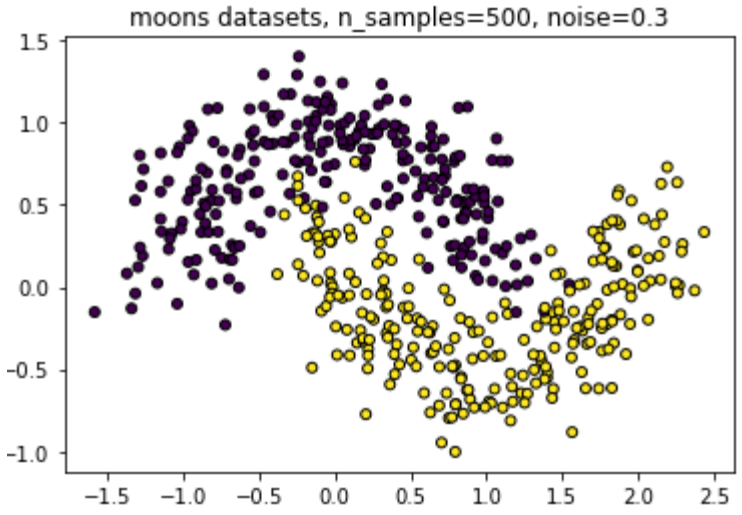
3、题目三：采用 scikit-learn 中的 LogisticRegression 逻辑回归模型对非线性数据集进行分类。

具体内容：

（1）数据集：使用 sklearn 自带数据生成器 make_moons 产生两类数据样本，示例程序如下，参数可自行修改。

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

plt.title("moons datasets, n_samples=500, noise=0.3")
X, y = make_moons(n_samples=500, noise=0.2, random_state=520)
plt.scatter(X[:, 0], X[:, 1], marker="o", c=y, s=25, edgecolor="k")
plt.show()
```



（2）特征衍生（数据增强）：使用 sklearn 自带 sklearn.preprocessing.PolynomialFeatures 生成指定阶次的多项式特征，从而得到所有多项式组合成的新特征矩阵，degree 参数任选。

PolynomialFeatures 使用详见官网：

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures

示例程序：

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> import numpy as np
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(degree=2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

PolynomialFeatures 也可使用 Pipeline 功能，简化程序编写。示例程序如下：

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.pipeline import Pipeline
>>> import numpy as np
>>> model = Pipeline([('poly', PolynomialFeatures(degree=3)),
...                   ('linear', LinearRegression(fit_intercept=False))])
>>> # fit to an order-3 polynomial data
>>> x = np.arange(5)
>>> y = 3 - 2 * x + x ** 2 - x ** 3
>>> model = model.fit(x[:, np.newaxis], y)
>>> model.named_steps['linear'].coef_
array([ 3., -2.,  1., -1.])
```

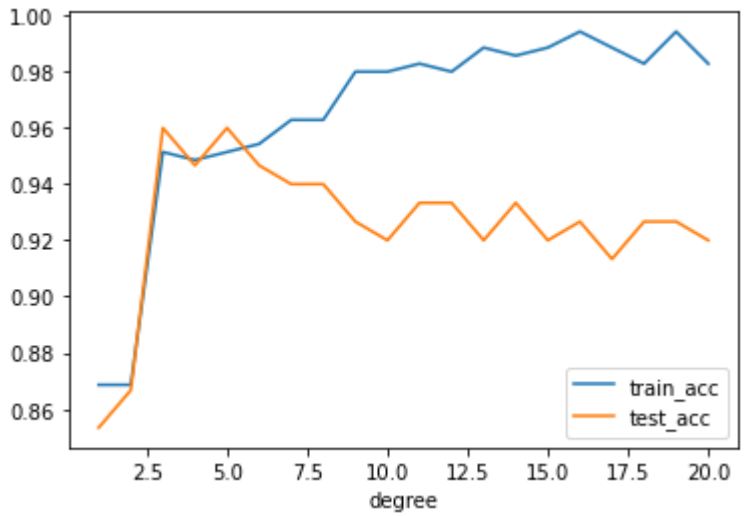
（3）模型建立：在新特征基础上建立逻辑回归二分类模型。

（4）决策边界可视化：绘制决策边界，观察非线性边界的变化。

【讨论二】在不加正则项的情况下，改变特征衍生的特征数量（即 degree 参数），观察决策边界的变化情况，以及训练集和测试集分数，体会模型从欠拟合 ->拟合 ->过拟合的过程。

提示：可使用 for 循环对不同 degree 进行遍历，观察模型的建模结果。

可通过绘制训练集和测试集分数曲线帮助观察（如示例图）。



【讨论三】（选做）在讨论二的基础上选择一种模型过拟合的 degree，在模型中分别加入'l1'和'l2'正则项，观察决策边界的变化情况，以及训练集和测试集分数，体会两种正则项对模型的作用。

【讨论四】可尝试手动调整 degree、正则项系数 C 和正则项种类，寻找使模型泛化性能最好的一组参数。

提示：手动调参采用"单一变量"原则。可先设定正则项种类（如'l1'）和正则项系数 C（如默认），再人为设定特征最高阶次 degree 的范围进行 degree 寻优，在选定的 degree 和'l1'正则化后，设定正则项系数 C 的范围进行寻优。

4、题目四：使用 numpy 编写逻辑回归算法，对 iris 数据进行二分类。

具体内容：

（1）任选两个特征和两个类别进行二分类。

（2）输出：决策函数的参数、预测值、分类准确率等。

（3）可视化：选取两个特征进行散点图可视化，并可视化决策边界。

5、题目五（选做）：使用 numpy 编写逻辑回归算法，对 iris 数据进行多分类。

具体内容：输出决策函数的参数、预测值、分类准确率等。

提示：

（1）可采用 OVR、OVO、ECOC 策略。

（2）可采用 CrossEntropy Loss + softmax 策略。

a）需将三个类别（如 0,1,2）进行 one-hot 编码。

b）每个线性分类器对应一组模型参数，3 个线性分类器对应 3 组模型参数。

c）可通过 softmax 回归计算多种类别的概率（K 种类别概率和为 1）。

d）通过最小化 CrossEntropy Loss 的梯度下降算法进行分类器参数寻优。