Logitic回归

大纲

1. **Logistic 回归概述**

- **从回归到分类**：解释如何将线性回归模型扩展到分类任务，引出 logistic 回归。
- **Logistic 函数**：介绍 sigmoid 函数，解释其特性以及如何将线性回归模型的输出转换为概率。
- **分类规则**：使用 logistic 函数计算样本属于正类的概率，并根据阈值进行分类。

2. **Logistic 回归的工作原理**

- **模型函数**：定义 logistic 回归模型函数，包含权重向量 w 和截距 b。
- **损失函数**：引入交叉熵损失函数，解释其意义并给出计算公式。
- **优化算法**：介绍梯度下降法，解释如何通过迭代更新参数 w 和 b 以最小化损失函数。
- **与线性回归的比较**：总结 logistic 回归与线性回归在函数集、损失函数和优化算法方面的区别。

3. **Scikit-learn 中的 Logistic 回归类**

- **LogisticRegression 类**：介绍 Scikit-learn 中的 LogisticRegression 类，包括其主要参数、方法和属性。
- **参数调整**：解释参数 C 的作用，并说明如何通过调整 C 来控制模型的复杂度。
- **案例分析**：以 iris 数据集和乳腺癌数据集为例，演示如何使用 LogisticRegression 类进行分类，并分析不同参数对模型性能的影响。

- **数据预处理**：介绍数据规范化预处理，解释其意义并说明如何使用 StandardScaler 进行数据标准化。
- **模型评估**：演示如何使用准确率得分、混淆矩阵和分类性能报告评估模型性能。

4. **Softmax 回归**

- **Softmax 函数**：介绍 Softmax 函数，解释其如何将多个线性函数的输出转换为概率分布。
- **Softmax 回归的工作原理**：说明 Softmax 回归是 Logistic 回归的推广，并解释其在多分类任务中的应用。
- **案例分析**：以手写数字数据集为例，演示如何使用 Softmax 回归进行多类分类。

5. **总结**

- **模型、损失函数和优化算法**：回顾基于 Logistic/Softmax 回归的机器学习算法构建的三步曲。

Logistic函数

Logistic 函数（sigmoid 函数） 是 logistic 回归模型的核心，它将线性回归模型的输出转换为概率值，使得模型可以应用于分类任务。

特性：

- **S 型曲线**： Logistic 函数的图像是一条 S 型曲线，其值域在 0 到 1 之间。
- **单调递增**： 随着 z 值的增加， Logistic 函数的值也随之增加。
- **连续可微**： Logistic 函数在定义域内处处可导，这使得可以使用梯度下降法进行优化。
- **中心对称**： Logistic 函数关于 y=0.5 对称。

将线性回归模型的输出转换为概率：

线性回归模型的输出 z = w^T x + b 是一个实数值，无法直接表示样本属于某一类别的概率。而 Logistic 函数可以将 z 值转换为概率值，其公式为：

p = σ(z) = 1 / (1 + e^(-z))

其中，σ 是 Logistic 函数，z 是线性回归模型的输出，p 是样本属于正类 C1 的概率。

分类规则：

根据样本属于正类的概率 p，可以制定以下分类规则：

- 如果 p > 0.5，则预测样本属于正类 C1。
- 如果 p ⩽ 0.5，则预测样本属于负类 C2。

Logistic 函数的优势：

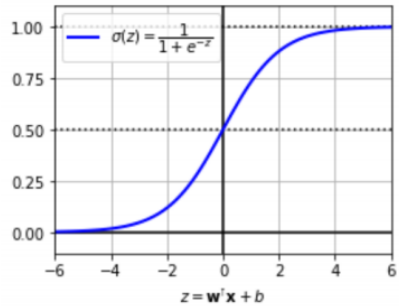
- **平滑转换**： Logistic 函数能够平滑地将线性回归模型的输出转换为概率值，避免了直接使用线性函数进行分类时可能出现的极端情况（例如，线性函数输出为负值或大于 1）。

- **概率解释**： Logistic 函数的输出值具有概率的解释，这使得模型可以提供更丰富的信息，例如样本属于不同类别的置信度。
- **易于优化**： Logistic 函数的连续可微性使得可以使用梯度下降法等优化算法来寻找最优模型参数。

Logistic回归原理

模型函数

如何利用logistic 函数进行类别预测?



若 w 和 b 已确定，样本 x 属于C_1 的概率为

$$P(C_1|\boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T\boldsymbol{x}+b\right) = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

如果 $\boldsymbol{w}^T\boldsymbol{x}+b \geq 0 \Rightarrow P(C_1|\boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T\boldsymbol{x}+b\right) \geq 0.5$,

则预测$\boldsymbol{x}$的类别是1（正类），即 $\hat{y}=1$；

如果 $\boldsymbol{w}^T\boldsymbol{x}+b < 0 \Rightarrow P(C_1|\boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T\boldsymbol{x}+b\right) < 0.5$,

则预测$\boldsymbol{x}$的类别为0（负类），即 $\hat{y}=0$。

损失函数

在机器学习中，交叉熵（Cross Entropy）是一种衡量两个概率分布差异的度量方式，常用于分类问题作为损失函数。对于单个样本 $x^{(i)}$，其真实标签为 $y^{(i)}$，预测的概率分布为 $P_{w,b}(C_1|x^{(i)})$ 和 $P_{w,b}(C_0|x^{(i)})$。其中 C_1 表示正类，C_0 表示负类，w 是权重参数，b 是偏置项。

当真实标签 $y^{(i)} = 1$ 时，我们希望模型正确地预测到正类的概率接近于1，因此损失函数定义为：

$$L(y^{(i)}, P_{w,b}(C_1|x^{(i)})) = -\log(P_{w,b}(C_1|x^{(i)}))$$

这里取对数是因为它能够平滑梯度下降的过程，使得学习过程更加稳定。

同理，当真实标签 $y^{(i)} = 0$ 时，我们希望模型预测到负类的概率接近于1，所以损失函数定义为：

$$L(y^{(i)}, P_{w,b}(C_0|x^{(i)})) = -\log(1 - P_{w,b}(C_1|x^{(i)}))$$

这里的 $1 - P_{w,b}(C_1|x^{(i)})$ 实际上就是 $P_{w,b}(C_0|x^{(i)})$，因为所有可能的类别概率之和应该等于1。

综上所述，这两个公式分别计算了当真实标签为1或0时，模型的预测与实际之间的差距。通过最小化这些损失值，可以优化模型参数 w 和 b，从而提高模型的预测准确性。

逻辑回归的损失函数：

整个训练集的损失函数是所有训练样本的**平均损失**。 m是样本数目。

设一样本$\boldsymbol{x}^{(i)}$，其真实标签$y^{(i)}$；模型输出值 $\hat{p}^{(i)} = P_{\boldsymbol{w},b}\left(C_1 \middle| \boldsymbol{x}^{(i)}\right)$，$i = 1,2,\cdots m$

$$L(\boldsymbol{w}, b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)\log(1 - \hat{p}^{(i)})\right]$$

交叉熵

这个函数没有闭式解，
但该函数是凸函数，可以用梯度下降等迭代优化算法，
找到全局最小值。

定义　交叉熵（cross-entropy）　　信息熵：$H(X) = -\sum P(x)\log_2 P(x)$

对于同一个随机变量x有两个单独的概率分布$P(x)$和$Q(x)$，分布P和Q的交叉熵为，

$$H(P,Q) = -E_{x\sim P}\log Q(x) = -\sum_x P(x)\log Q(x)$$

x的真实 (伯努利) 分布P：	交叉熵	x的假设(伯努利)分布Q：
$P(x=1) = y^{(i)}$ $P(x=0) = 1-y^{(i)}$	$H(P,Q) = -\sum_x P(x)\log(Q(x))$	$Q(x=1) = f(x^{(i)})$ $Q(x=0) = 1-f(x^{(i)})$

优化算法

机器学习三步曲：

Step1 **函数集**：　$P_{w,b}(C_1|x) = \sigma(w^T x + b)$

寻找最优函数（对应最优参数w和b），使得模型对真实类别是正类的样本（C_1）输出的$P_{w,b}(C_1|x^{(i)})$高，而对真实类别是负类的样本（C_2）输出的$P_{w,b}(C_1|x^{(i)})$低的估算。

Step2 **损失函数**：目标真实值y与输出值$\hat{y}$的**交叉熵**

Step3 **优化算法**：梯度下降法，寻找最优参数w和b。

梯度计算

$$L(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{p}^{(i)}) + (1-y^{(i)})\log(1-\hat{p}^{(i)})\right]$$

令$w \leftarrow \begin{bmatrix} b \\ w \end{bmatrix}$是损失函数的参数，　$\hat{p}^{(i)} = P_w(C_1|x^{(i)}) = \sigma(w^T x^{(i)})$

$$\frac{\partial L(w)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(w^T x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

预测值与真实类别的误差

梯度下降法参数更新：

$$w_j \leftarrow w_j - \frac{\eta}{m}\sum_{i=1}^{m}\left(\sigma(w^T x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

Logistic回归代码案例

sklearn.linear_model.LogisticRegression类：用于**二分类或多分类**问题

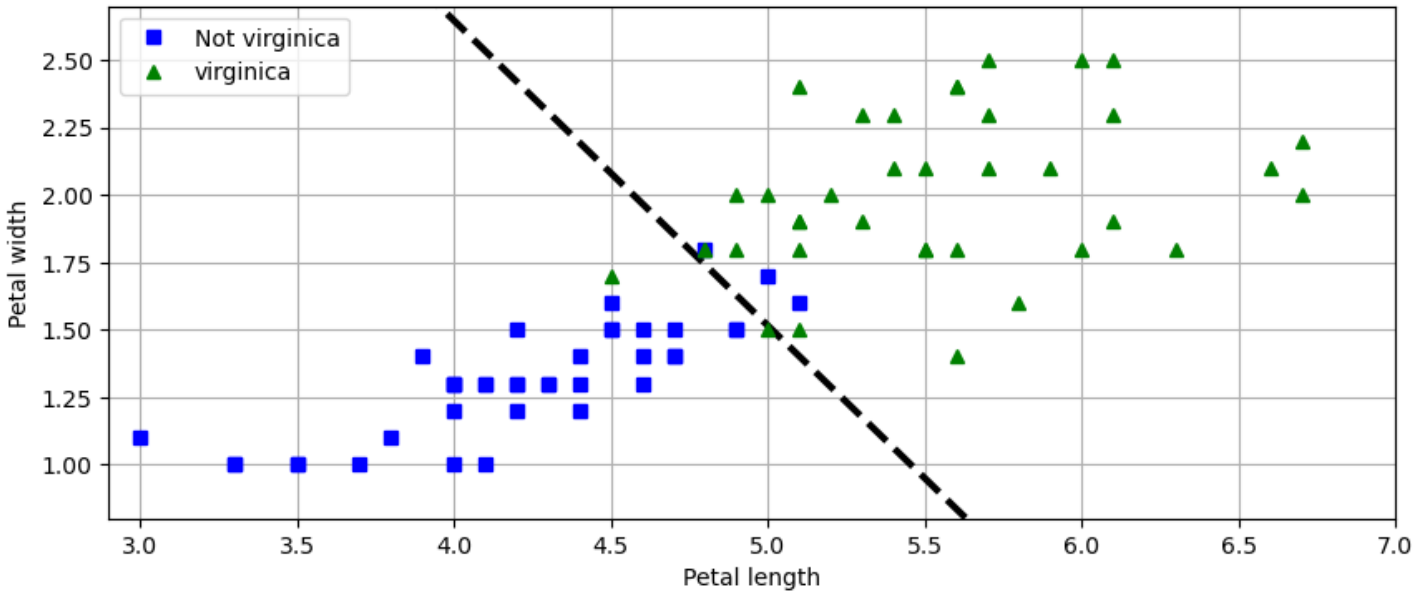
`LogisticRegression(penalty='l2', C=1.0, max_iter=100, …)`

主要参数	• ***penalty***：指定惩罚中使用的范数，如*l1,l2,Elastic-Net*。默认值为*l2*。 • ***C***：正则化强度的平衡参数，是正则化强度的倒数。Float，默认1.0。C越小，正则强度越大。 • ***max_iter***：最大迭代次数，默认100。 • ***multi_class***：*{'auto', 'ovr', 'multinomial'}*, *默认'auto'*。多分类用multinomial。**若选auto，则自动根据数据类别调整，对二类用ovr，多类用multinomial（此时切换成softmax回归）**。
方法	• **fit(X,y)** 训练模型，X为特征矩阵；y为目标向量。 • **score(X,y)** 计算预测的平均精度。 • **predict(newX)** 预测新数据的类别。返回一维数组(样本数)。 • **predict_proba(newX)** 预测新数据属于各类别概率。返回二维数组,形状(样本数, 类别数)。
属性	• **coef_** 决策函数中的特征系数。**二维数组**。 • **intercept_** 截距，float型数。

案例1：花二分类

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

数据处理
iris = load_iris()
X = iris.data[:,2:] # “petal length”和“petal width”
y = (iris.target_names[iris.target] == 'virginica').astype(int)

训练Logistic回归分类器，令C=2
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=42)
log_reg = LogisticRegression(C=2, random_state=42)
log_reg.fit(X_train, y_train)

绘制决策边界
x0_left_right = np.array([2.9, 7]) #取"petal length:2.9和7两个值"作为x轴最小和最大值
x1_top_down = -((log_reg.coef_[0, 0] * x0_left_right + log_reg.intercept_[0]) / log_reg.coef_[0, 1]) # "计算petal width对应值"
plt.figure(figsize=(10, 4))
plt.plot(x0_left_right, x1_top_down, "k--", linewidth=3)
plt.plot(X_train[y_train==0, 0],X_train[y_train==0,1],"bs",label="Not virginica")
plt.plot(X_train[y_train==1, 0],X_train[y_train==1,1],"g^",label="virginica")
plt.xlabel("Petal length")
```

```
plt.ylabel("Petal width")
plt.legend(loc="best")
plt.axis([2.9, 7, 0.8, 2.7])
plt.grid()
plt.show()
```



案例2：乳腺癌预测_v1

✏️ 任务：① 用sklearn的LogisticRegression，采用默认参数的“L2”正则化，强 度系数C=1，迭代次数　max_iter=10000，构建一个logistic回归分类 器，应用到乳腺癌数据集上。训练模型，然后输出模型在训练集和测试集上的准确率得分。

② 尝试调整参数C=100、0.01，观察训练得分和测试得分。

③ 比较正则化参数取三个不同的值(1.0,100,0.01)时模型学得的系数，绘 制对应的散点图。

读取数据

```
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer() #返回bunch对象
cancer.data.shape # cancer.data是数据矩阵
```

正则化为1时候：

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X=cancer.data
y=cancer.target
X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,
random_state=42)
logreg=LogisticRegression(max_iter=10000).fit(X_train,y_train)
print("训练得分：{:.3f}".format(logreg.score(X_train,y_train)))
print("测试得分：{:.3f}".format(logreg.score(X_test,y_test)))
```

正则化为100时候：

```
logreg100=LogisticRegression(C=100,max_iter=10000).fit(X_train,y_train)
print("训练得分：{:.3f}".format(logreg100.score(X_train,y_train)))
print("测试得分：{:.3f}".format(logreg100.score(X_test,y_test)))
```

正则化为0.01时候：

```
logreg001=LogisticRegression(C=0.01,max_iter=10000).fit(X_train,y_train)
print("训练得分：{:.3f}".format(logreg001.score(X_train,y_train)))
print("测试得分：{:.3f}".format(logreg001.score(X_test,y_test)))
```

可视化不同正则化参数时候情况：

```
import matplotlib.pyplot as plt
plt.plot(*logreg.coef_,'o',label='C=1') #取出二维数组logreg.coef_的元素,一维数组
plt.plot(*logreg100.coef_,'^',label="C=100")
plt.plot(*logreg001.coef_,'v',label="C=0.01")
plt.xticks(range(cancer.data.shape[1]),cancer.feature_names,rotation=90)
plt.hlines(0,0,cancer.data.shape[1]) # 横轴0~30，纵轴都是0
plt.ylim(-5,5)
plt.xlabel("coefficient index")
plt.ylabel("coefficient magnitude")
plt.legend()
```

案例3：乳腺癌预测_v2

任务：

① 观察原数据X，计算各特征的均值（最大与最小）、方差（最大与最小）、取值区间（各区间上下界差）。

② 数据规范化预处理，调用sklearn.preprocessing中的StandardScaler ，使得处理后的数据集均值为零，方差为1。

③ 用规范化后的数据，令C分别等于0.1,1,10,100，找到最优C对应的模型。

④ 用最优C重新训练一个新模型。预测测试数据所属类别和概率，用DataFrame表示。

⑤ 输出混淆矩阵和分类性能报告。

先对原数据进行观察：

```
import numpy as np
from sklearn.datasets import load_breast_cancer

import pandas as pd

cancer=load_breast_cancer() # bunch对象
x=cancer.data # 形状(569, 30)
y=cancer.target # 形状(569,)


features_mean = [X.mean(axis=0)] # 各列的均值
features_var = [X.var(axis=0)]
features_scale =[X.max(axis=0)-X.min(axis=0)]

df = pd.DataFrame({'特征均值':[np.min(features_mean),np.max(features_mean)],
                   '特征方差':[np.min(features_var),np.max(features_var)],
                   '特征区间范围':
[np.min(features_scale),np.max(features_scale)]}, index=['最小','最大'])
print(df)
```

输出：

```
特征均值         特征方差   特征区间范围
最小   0.000000   0.000000     0.0
最大  12.089037  42.721065    16.0
```

数据规范化处理与模型训练：

```python
# 数据规范化预处理
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np


# 进行训练集和测试集的划分
X_train, X_test, y_train, y_test = train_test_split(X, y,stratify=y,
random_state=42)
scaler = StandardScaler()
Xtrain_new = scaler.fit_transform(X_train)
Xtest_new = scaler.transform(X_test)

# 用规范化后的训练集数据训练模型
from sklearn.linear_model import LogisticRegression

models = []
for i in [0.1, 1, 10, 100]:
    logreg = LogisticRegression(C=i, max_iter=1000).fit(Xtrain_new, y_train)
    print("C={}, 训练得分: {:.3f}".format(i, logreg.score(Xtrain_new, y_train)))
    print("C={}, 测试得分: {:.3f}\n".format(i, logreg.score(Xtest_new, y_test)))
    models.append((i, logreg))
```

用C=1训练模型，并且输出预测类别和概率（前10个）

```python
# 用规范化后的训练集数据训练模型
from sklearn.linear_model import LogisticRegression

models = []
# 训练模型
logreg = LogisticRegression(C=1, max_iter=1000).fit(Xtrain_new, y_train)

# 预测类别和概率
predictions = logreg.predict(Xtest_new)  # 预测类别
probabilities = logreg.predict_proba(Xtest_new)  # 预测概率

# 输出预测结果
for i in range(1,11):
    print("样本 {}: 预测类别 = {}, 概率 = {}".format(i, predictions[i],
probabilities[i]))
```

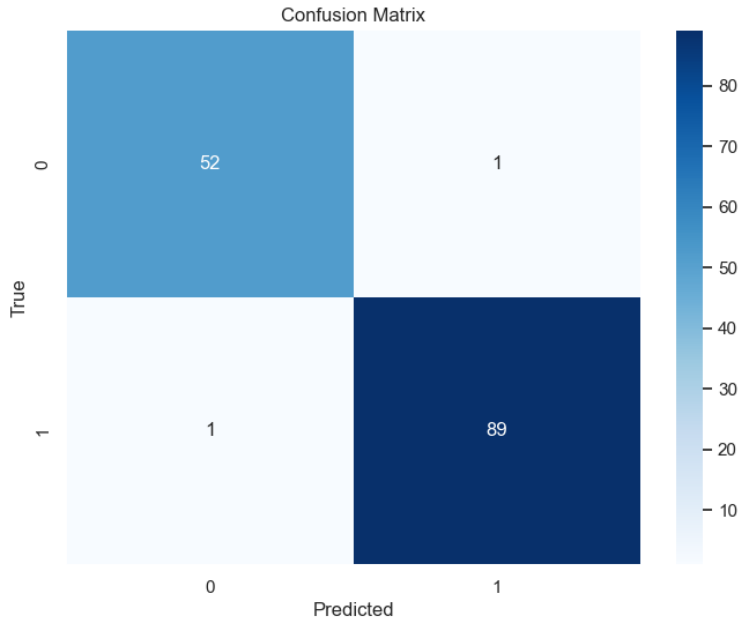
输出：

```
样本 1: 预测类别 = 0, 概率 = [9.9965176e-01 3.4823951e-04]
样本 2: 预测类别 = 1, 概率 = [0.44076125 0.55923875]
样本 3: 预测类别 = 1, 概率 = [0.06011142 0.93988858]
样本 4: 预测类别 = 0, 概率 = [0.8244832 0.1755168]
样本 5: 预测类别 = 0, 概率 = [9.99999040e-01 9.59859873e-07]
样本 6: 预测类别 = 0, 概率 = [0.99726694 0.00273306]
样本 7: 预测类别 = 0, 概率 = [9.99242420e-01 7.57580232e-04]
样本 8: 预测类别 = 0, 概率 = [9.99320208e-01 6.79792123e-04]
样本 9: 预测类别 = 1, 概率 = [9.13822710e-05 9.99908618e-01]
样本 10: 预测类别 = 1, 概率 = [0.3021715 0.6978285]
```

打印混淆矩阵和分类报告：

```python
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# 生成混淆矩阵
cm = confusion_matrix(y_test, predictions)

# 可视化混淆矩阵
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# 生成分类报告
report = classification_report(y_test, predictions)
print(report)
```

输出：




```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        53
           1       0.99      0.99      0.99        90

    accuracy                           0.99       143
   macro avg       0.99      0.99      0.99       143
weighted avg       0.99      0.99      0.99       143
```

Softmax多分类回归

softmax函数

Softmax函数

定义： Softmax函数是一种在多分类问题中常用的激活函数，它将一个实数向量转换成一个概率分布。具体来说，对于给定的向量 $z \in \mathbb{R}^K$，Softmax函数 $S(z)$ 会输出一个大小为 K 的向量，其中每个元素的范围在 $(0, 1)$ 之间，并且所有元素的和为1。

公式：

$$S(z)_i = \frac{e^{z_i}}{\sum_{j-1}^{K} e^{z_j}}$$

其中，$S(z)_i$ 是向量 $S(z)$ 的第 i 个元素，z_i 是向量 z 的第 i 个元素。

原理： Softmax函数通过指数函数将输入的实数值转换成正数，并通过除以所有指数和的方式将它们转换成概率值。指数函数确保了较大的输入值将具有较大的输出概率，而较小的输入值将具有较小的输出概率。

sofxmax回归原理

Softmax回归

定义： Softmax回归是逻辑回归在多分类问题上的推广。它通常用于多分类模型的输出层，将原始的得分（例如，来自神经网络的输出）转换为概率分布。

原理： 在Softmax回归中，模型首先计算每个类别的得分（通常是线性组合，即 $z = Wx + b$），然后应用Softmax函数将这些得分转换为概率。模型的参数 W 和 b 通过最小化损失函数（通常是交叉熵损失）来训练。

交叉熵损失： 交叉熵损失函数 L 用于衡量预测概率分布 $\hat{y}$ 和真实标签 y 的差异：

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

其中，y 是一个one-hot编码的向量，表示真实的类别标签，而 $\hat{y}$ 是模型预测的概率分布。

应用场景

- **多分类问题**：Softmax回归是处理多分类问题的一种常用方法，例如，手写数字识别（0到9共10个类别）、图像分类（如CIFAR-10、ImageNet等）。
- **神经网络的输出层**：在深度学习中，Softmax回归常作为最后一层，将神经网络的输出转换为概率分布，用于分类任务。
- **概率输出**：在需要模型输出概率而不是直接类别标签的应用中，Softmax回归非常有用，例如，在推荐系统中预测用户对某个项目的偏好概率。

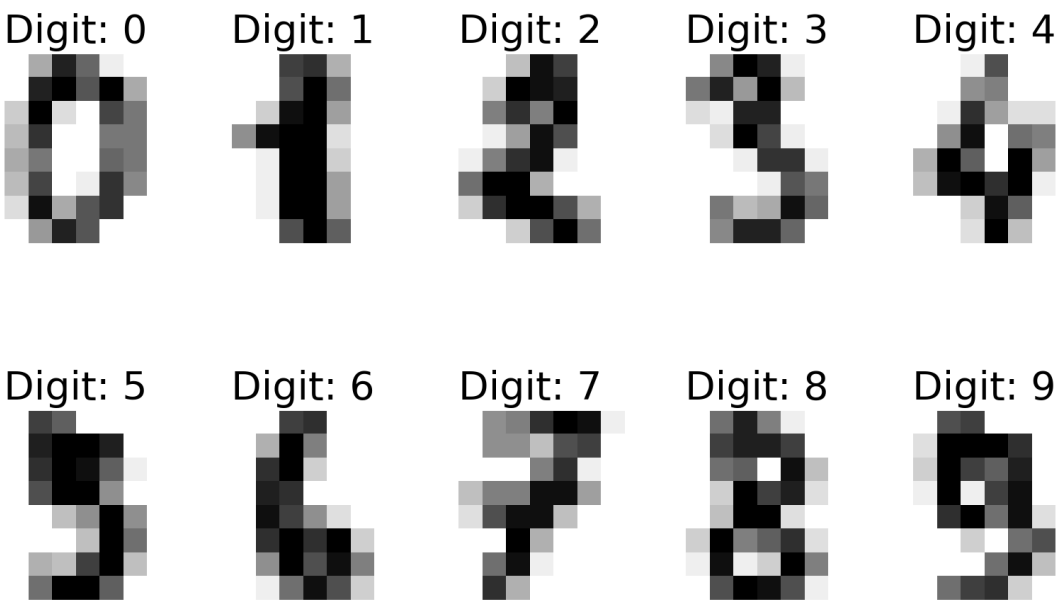
总结来说，Softmax函数和Softmax回归是多分类问题中非常重要的工具，它们帮助模型输出一个可解释的概率分布，从而对数据进行有效的分类。

Softmax手写字符识别

1.加载和可视化数据集

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

# 加载数据集
digits = datasets.load_digits()

# 准备图像和标签
images_and_labels = list(zip(digits.images, digits.target))

# 创建一个2x5的子图布局
plt.figure(figsize=(10, 6), dpi=200)

```

```
# 遍历前10个图像
for index, (image, label) in enumerate(images_and_labels[:10]):
    # 在2x5布局中添加子图
    plt.subplot(2, 5, index + 1)
    plt.axis('off')  # 关闭坐标轴
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')  # 显示图像
    plt.title('Digit: %i' % label, fontsize=20)  # 设置标题

# 显示所有子图
plt.show()
```



方法1：深度学习softmax回归（用keras）

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# 构建模型
model = Sequential([
    Dense(64, input_shape=(64,), activation='relu'),  # 输入层和第一个隐藏层
    Dense(10, activation='softmax')  # 输出层
])

# 编译模型
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

训练与评估模型：

```
# 训练模型
history = model.fit(X_train, y_train_one_hot, epochs=100, batch_size=32, validation_split=0.1)

# 评估模型
loss, accuracy = model.evaluate(X_test, y_test_one_hot)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

输出：

```
Epoch 1/100
41/41 [==============================] - 1s 6ms/step - loss: 1.9362 - accuracy: 0.3596 - val_loss: 1.4417 - val_accuracy: 0.6389
Epoch 2/100
41/41 [==============================] - 0s 2ms/step - loss: 1.0055 - accuracy: 0.7773 - val_loss: 0.8793 - val_accuracy: 0.8056
Epoch 3/100
41/41 [==============================] - 0s 2ms/step - loss: 0.6245 - accuracy: 0.8770 - val_loss: 0.6083 - val_accuracy: 0.8750
Epoch 4/100
41/41 [==============================] - 0s 3ms/step - loss: 0.4305 - accuracy: 0.9258 - val_loss: 0.4573 - val_accuracy: 0.9167
Epoch 5/100
41/41 [==============================] - 0s 3ms/step - loss: 0.3176 - accuracy: 0.9435 - val_loss: 0.3695 - val_accuracy: 0.9236
Epoch 6/100
41/41 [==============================] - 0s 3ms/step - loss: 0.2474 - accuracy: 0.9590 - val_loss: 0.3229 - val_accuracy: 0.9236
Epoch 7/100
41/41 [==============================] - 0s 2ms/step - loss: 0.1984 - accuracy: 0.9683 - val_loss: 0.2837 - val_accuracy: 0.9375
Epoch 8/100
41/41 [==============================] - 0s 3ms/step - loss: 0.1647 - accuracy: 0.9783 - val_loss: 0.2587 - val_accuracy: 0.9375
Epoch 9/100
41/41 [==============================] - 0s 3ms/step - loss: 0.1386 - accuracy: 0.9799 - val_loss: 0.2436 - val_accuracy: 0.9375
Epoch 10/100
41/41 [==============================] - 0s 3ms/step - loss: 0.1188 - accuracy: 0.9861 - val_loss: 0.2252 - val_accuracy: 0.9514
Epoch 11/100
```

```
41/41 [==============================] - 0s 3ms/step - loss: 0.1032 -
accuracy: 0.9869 - val_loss: 0.2175 - val_accuracy: 0.9444
Epoch 12/100
41/41 [==============================] - 0s 3ms/step - loss: 0.0905 -
accuracy: 0.9907 - val_loss: 0.2081 - val_accuracy: 0.9514
Epoch 13/100
...
Epoch 100/100
41/41 [==============================] - 0s 3ms/step - loss: 9.5014e-04 -
accuracy: 1.0000 - val_loss: 0.1351 - val_accuracy: 0.9722
12/12 [==============================] - 0s 2ms/step - loss: 0.0955 -
accuracy: 0.9778
Test Accuracy: 97.78%
```

方法2：简化为logistic回归

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
Xtrain,Xtest,Ytrain,Ytest=train_test_split(digits.data,digits.target,
test_size=0.2,random_state=42);
# 默认multi_class='auto'自适应类别数
clf = LogisticRegression(C=0.01, random_state=42,max_iter=10_000)
clf.fit(Xtrain, Ytrain)
```

输出准确度和性能报告：

```
print("在测试集上的准确度为：{:.3f}".format(clf.score(Xtest,Ytest)))
print("在训练集上的准确度为：{:.3f}".format(clf.score(Xtrain,Ytrain)))
from sklearn.metrics import classification_report
Y_pred = clf.predict(Xtest)
print(classification_report(Ytest,Y_pred))
```

输出：

```
在测试集上的准确度为：0.975
在训练集上的准确度为：0.991
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.98        28
           2       0.97      1.00      0.99        33
```

```
              3       1.00      0.97      0.99        34
              4       1.00      0.98      0.99        46
              5       0.94      0.94      0.94        47
              6       0.97      0.97      0.97        35
              7       1.00      0.97      0.99        34
              8       0.97      0.97      0.97        30
              9       0.95      0.97      0.96        40

       accuracy                           0.97       360
      macro avg       0.98      0.98      0.98       360
   weighted avg       0.98      0.97      0.98       360
```