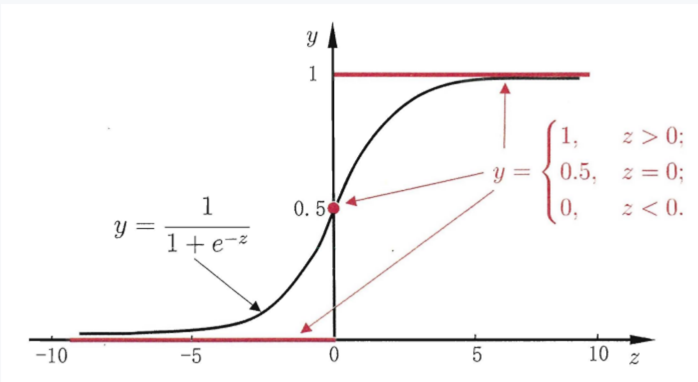
对数几率回归 logit regression

1. 简要介绍

逻辑回归，又称对数几率回归，英文为 logistic regression，或 logit regression。对数几率回归是一种**分类**学习方法，该方法直接对分类的可能性进行建模，无需事先假设数据分布，因此可以避免假设分布不准确带来的问题。

> 1. 对于**二分类**任务，即输出标记为 $y \in \{0,1\}$，因为**线性回归模型** $z = w^T x + b \in R$， 所以需要引进一个**替代函数** $g^{-}(\cdot)$，得到一个**广义的线性回归模型** $y = g^{-}(z) = g^{-}(w^T x + b) \in (0,1)$，即 $g(y)$ 与 $z = w^T x + b$ 之间是线性关系。
> 2. 该模型满足，当 $z \to \infty$ 时，$y \to 1$，当 $z \to -\infty$ 时，$y \to 0$，即引入了一种**概率关系**，当 p 越大，则正例的可能性更大，反例的可能性更小。
> 3. 通常来说，**对数几率函数** (logistic function) 作为**任意阶可导的凸函数**，也是一个常用的替代函数，
>
> $$y = \frac{1}{1+e^{-z}} \tag{1}$$
>
> 
>

> 1. 基于对数几率函数可以导出**广义线性回归模型**，该模型又称为**对数几率回归模型**。
>
> $$ln\frac{y}{1-y} = w^T x + b \tag{2}$$
>
> 2. 若将 y 视为样本 x 是正例的可能性，1-y 视为样本 x 是反例的可能性，则二者比值反映了 x 是正例的相对可能性，称之为**几率** (odds)，因此上述导出的广义线性回归模型又可以称为**对数几率** (log odds 或 logit)。
>
> $$odds = \frac{y}{1-y} \tag{3}$$
>
> 3. 基于以上观点，为了求解对数几率回归模型，即**需要采用一定的方法估计参数** w, b，无妨记 $\beta = (w; b), \hat{x} = (x; 1)$，则有 $z = w^T x + b = \beta^T x$。

4. 如将 y 视为类后验概率估计 $p(y=1|x)$, $1-y$ 视为 $p(y=0|x)$，则式 (2) 可以表示为，

$$ln\frac{p(y=1|x)}{p(y=0|x)} = w^Tx+b \tag{4}$$

进一步计算得到正例和反例的**类后验概率估计**为，

$$p(y=1|x) = \frac{e^{w^Tx+b}}{1+e^{w^Tx+b}} = p_1(\hat{x};\beta) = p(y=1|\hat{x};\beta) \tag{5-1}$$

$$p(y=0|x) = \frac{1}{1+e^{w^Tx+b}} = p_0(\hat{x};\beta) = p(y=0|\hat{x};\beta) \tag{5-2}$$

$$p_1(\hat{x};\beta)+p_0(\hat{x};\beta)=1$$

1. 对数几率回归模型的求解在于参数 w, b 或 $\beta=(w;b)$ 的求解，因此基于已有样本数据，可以采用**极大似然法** (maximum likelihood method) 来进行参数估计，即**令每个样本属于其真实标记的概率越大越好**。
2. 给定数据集 $\{(x_i,y_i)\}_{i=1}^m$，可以得到对数几率回归模型的**对数似然** (log likelihood)，优化方向是使其最大化，

$$max: log\ likelihood = l(w,b) = \sum_{i=1}^{m} ln\ p(y_i|x_i;w,b) = \sum_{i=1}^{m} ln\ p(y_i;\hat{x}_i,\beta) \tag{6}$$

3. 对数似然中的**似然项** (likelihood) 可以改写如下，

$$p(y_i|x_i;w,b) = p(y_i;\hat{x}_i,\beta) = y_ip_1(\hat{x}_i,\beta)+(1-y_i)p_0(\hat{x}_i,\beta) \tag{7}$$

故而**对数似然** 基于式 (5), (6), (7)可以改写如下，

$$\begin{aligned} max: l(\beta) &= \sum_{i=1}^{n} ln[y_ip_1(\hat{x}_i,\beta)+(1-y_i)p_0(\hat{x}_i,\beta)] \\ &= \sum_{i=1}^{n}[y_i\beta^T\hat{x} - ln(1+e^{\beta^T\hat{x}})] \end{aligned} \tag{8}$$

式 (8) 的**最大化**式转换为**最小化**式如下，该式是关于 β 的高阶可导的连续凸函数，

$$min: l(\beta) = \sum_{i=1}^{n}[-y_i\beta^T\hat{x} + ln(1+e^{\beta^T\hat{x}})] \tag{9}$$

根据式 (9) 及**数值优化算法** (梯度下降法 GDM，牛顿法 NM等) 可以求对数几率回归模型的最优解 β^*，

$$\beta^* = \mathop{arg\ min}_{\beta}\ l(\beta) \tag{10}$$

1. **牛顿法**求解对数几率回归模型步骤如下

Algorithm 9.5 *Newton's method.*

given a starting point $x \in \textbf{dom}\, f$, tolerance $\epsilon > 0$.

repeat

1. *Compute the Newton step and decrement.*
   $\Delta x_{\mathrm{nt}} := -\nabla^2 f(x)^{-1} \nabla f(x); \quad \lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$.
2. *Stopping criterion.* **quit** if $\lambda^2/2 \le \epsilon$.
3. *Line search.* Choose step size t by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$.

其中**迭代解的更新公式**为

$$\beta^{t+1} = \beta^t - t(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T})^{-1}\frac{\partial l(\beta)}{\partial\beta} \tag{11}$$

关于 β 的**一阶、二阶导数**为

$$\frac{\partial l(\beta)}{\partial\beta} = -\sum_{i=1}^{n} \hat{x}_i(y_i - p_1) \tag{12-1}$$

$$\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} = \sum_{i=1}^{n} \hat{x}_i \hat{x}_i{}^T p_1 p_0 \tag{12-2}$$

2. 牛顿法中的学习率 t 可以使用 **backtracking line search** 方法求解

Algorithm 9.2 *Backtracking line search.*

given a descent direction Δx for f at $x \in \textbf{dom}\, f$, $\alpha \in (0, 0.5)$, $\beta \in (0, 1)$.

$t := 1$.

while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x, \quad t := \beta t$.

当海森矩阵非正定时，牛顿法失效，因此也可以采用**梯度下降法**求解参数 β，其步骤如下，

Algorithm 9.3 *Gradient descent method.*

given a starting point $x \in \textbf{dom}\, f$.

repeat

1. $\Delta x := -\nabla f(x)$.
2. *Line search.* Choose step size t via exact or backtracking line search.
3. *Update.* $x := x + t\Delta x$.

until stopping criterion is satisfied.

The stopping criterion is usually of the form $\|\nabla f(x)\|_2 \le \eta$, where η is small and positive. In most implementations, this condition is checked after step 1, rather than after the update.

2. 模型训练步骤

dataset: $X=\{x_1,x_2,x_3,...,,x_n\},x_i=[x_{i1},x_{i2},x_{i3},...,x_{im}]^T$ (n samples × m features)

denote dataset as: $X=\left\{\hat{x_1},\hat{x_2},\hat{x_3},...,,\hat{x_n}\right\},\hat{x_i}=[x_{i1},x_{i2},x_{i3},...,x_{im},1]^T;\beta=[w;b]$

牛顿法

given a starting point β, tolorance $\varepsilon=10^{-6}$

repeat 1、2、3、4

1. 计算牛顿步长 $\Delta\beta_{nt}$ 和牛顿减量 λ^2

$$
\begin{aligned}
newton\ stepsize: \Delta\beta_{nt} &:= -(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T})^{-1}\frac{\partial l(\beta)}{\partial\beta} \\
newton\ decrement: \lambda^2 &:= (\frac{\partial l(\beta)}{\partial\beta})^T(\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T})^{-1}\frac{\partial l(\beta)}{\partial\beta} \\
\frac{\partial l(\beta)}{\partial\beta} &= -\sum_{i=1}^{n}\hat{x}_i(y_i-p_1) \\
\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} &= \sum_{i=1}^{n}\hat{x}_i\hat{x}_i^{\ T}p_1p_0
\end{aligned}
$$

2. 停止准则 (当牛顿减量特别小的时候，表示函数此时十分平滑了)

$$if\ \lambda^2/2\le\varepsilon, then\ quit$$

3. 回溯直线搜索，计算学习率 t

   given $\hat{\alpha}\in(0,0.5),\hat{\beta}\in(0,1)$，其中 β 和 $\hat{\beta}$ 代表不同含义

$$
\begin{gathered}
while\ f(\beta^k+t^k\Delta\beta_{nt}{}^k)>f(\beta^k)-\hat{\alpha}t^k\lambda^2 \\
t^k:=\hat{\beta}t^k
\end{gathered}
$$

4. 参数更新

$$\beta^{k+1}:=\beta^k+t^k\Delta\beta_{nt}{}^k$$

梯度下降法

given a starting point β，$\eta=10^-5$

repeat 1、2、3、4

1. 计算梯度下降步长

$$\Delta x:=-\frac{\partial l(\beta)}{\partial\beta}=\sum_{i=1}^{n}\hat{x}_i(y_i-p_1)$$

2. 停止准则判断：exit if

$$||\frac{\partial l(\beta)}{\partial \beta}||_2 \leq \eta$$

3. 回溯直线搜索，计算学习率 t

   given $\hat{\alpha} \in (0, 0.5), \hat{\beta} \in (0, 1)$，其中 β 和 $\hat{\beta}$ 代表不同含义

$$while\ f(\beta^k + t^k \Delta {\beta_{nt}}^k) > f(\beta^k) - \hat{\alpha} t^k \lambda^2$$
$$t^k := \hat{\beta} t^k$$

4. 参数更新

$$\beta^{k+1} := \beta^k + t^k \Delta {\beta_{nt}}^k$$

3. 数据集介绍

DATASET: Loan-Approval-Prediction-Dataset, **4296 × 13 features** (9 integer, 2 string, 1 id, 1 other)

The loan approval dataset is a collection of financial records and associated information used to **determine the eligibility of individuals or organizations for obtaining loans** from a lending institution. It includes various factors such as **cibil score, income, employment status, loan term, loan amount, assets value, and loan status**.

4. 模型训练代码

LogisticRegression_newton.py

此 python 文件定义了一个 LogisticRegression_newton 的类，通过实例化类对象和调用相关方法可以实现**对数几率回归模型**的训练和测试。

1. 实例化对数几率回归模型 `model = LogisticRegression_newton()`
   - 可以指定模型的相关参数：learning_rate 学习率；max_iterations 最大迭代次数
2. 模型训练(实例化模型后) `model.fit(X_train, y_train)`
   - 参数解释：X_train 训练集样本数据集；y_train 训练集样本标签数据集
   - 该方法使用牛顿法求解对数几率回归模型的最优参数 β
3. 模型测试(模型训练后) `model.accuracy(X_test, y_test)`
   - 参数解释：X_test 训练集样本数据集；y_test 训练集样本标签数据集
   - 该方法用于对模型进行测试，返回模型预测的精确度(预测准确样本数量/测试样本数量)

```python
import numpy as np


class LogisticRegression_newton():
    """ 对数几率回归模型 Y=w^T*X+b=beta^T*X_hat """

    def __init__(self, learning_rate=0.5, max_iterations=10):
        """
        :param learning_rate: float, 学习率
        :param max_iterations: int, 最大迭代次数
        """
        self.beta = None
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations

    def initialize_weights(self, m_features):
        """
        :param m_features: 整型，特征维数
        :return: 该方法用于初始化及调整权重
        """
        # 方法调用后，self.beta=[[w_1],[w_2],[w_3],...,[w_m],[0]] (列向量, (m+1,1))
        limit = np.sqrt(1 / m_features)
        w = np.random.uniform(-limit, limit, (m_features, 1))
        b = np.asarray([[0]])
        self.beta = np.append(w, b, axis=0)  # beta=[w;b]，待求参数

    def fit(self, X, y):
        """
        :param X: 2 dimensions matrix 样本数据集，矩阵的每一行是一个样本，即一个行向量是一个样本
        :param y: column vector 样本标签
        :return: 该方法用于求解模型参数 beta
        """
        n_samples, m_features = X.shape
        self.initialize_weights(m_features)

        X_plus_one = np.ones((n_samples, 1))
        X = np.append(X, X_plus_one, axis=1)  # X = X_hat=[X;X_plus], shape of X is (n, m+1)
        y = np.reshape(y, (n_samples, 1))  # shape of y is (n,1)

        """ 模型训练(核心): 基于 X_hat、y、beta 进行训练
            repeat
            1. 计算牛顿步长和牛顿减量
            2. 停止判断
            3. 回溯直线搜索，计算学习率 t
            4. 更新参数 β
        """
```

```
        for i in range(self.max_iterations):
            tolerance = 10 ** -6  # 设定停止阈值
            # step1. 计算牛顿步长和牛顿减量
            der_first = np.zeros((1, m_features + 1))  # 梯度，或者说是损失函数关于参数 beta 的一阶导数
            der_second = np.zeros((m_features + 1, m_features + 1))  # 海森矩阵，或者说是损失函数关于参数 beta 的
二阶导数
            for j in range(n_samples):
                x = X[j, :].reshape(1, -1)  # 取矩阵的第 j 行，并转换为行向量
                eta = np.dot(x, self.beta)
                p1 = np.exp(eta) / (1 + np.exp(eta)) if eta <= 0 else 1.0 / (1 + np.exp(-eta))
                p0 = 1 - p1
                der_first -= x * (y[j] - p1)
                der_second += np.dot(x.T, x) * p1 * p0  # der_second 是矩阵 (m+1, m+1)
            der_first = der_first.reshape(-1, 1)  # der_first 是列向量 (m+1, 1)
            step_size_newton = -np.dot(np.linalg.pinv(der_second), der_first)  # 牛顿步长，是一个列向量，(m+1, 1)
            decrement_newton = np.dot(der_first.T, -step_size_newton)  # 牛顿减量，是一个数字
            # step2. 停止判断
            if decrement_newton / 2 <= tolerance:
                break
            # step3. 回溯直线搜索 backtracking line search
            alpha_search = 0.3
            beta_search = 0.5
            self.learning_rate = 1.0
            while True:
                """ 优化函数为 min: l(self.beta)=SUM[-y_i*self.beta^T*x+ln(1+e^{self.beta^Tx})] """
                fun_plus = 0  # 分别计算两个优化函数值
                fun = 0
                beta_plus = self.beta + self.learning_rate * step_size_newton  # 列向量
                for j in range(n_samples):
                    x = X[j, :].reshape(-1, 1)  # 取出 x 为列向量
                    fun_plus += -y[j] * np.dot(beta_plus.T, x) + np.log(1 + self.sigmod(x.T, beta_plus))
                    fun = -y[j] * np.dot(self.beta.T, x) + np.log(1 + self.sigmod(x.T, self.beta))
                if fun_plus <= fun - alpha_search * self.learning_rate * decrement_newton:
                    break
                self.learning_rate = beta_search * self.learning_rate
            # step4. 参数更新
            self.beta += self.learning_rate * step_size_newton

    def predict(self, X_test):
        """
        :param X_test: 测试数据集，矩阵的每一行是一个样本，即一个行向量是一个样本
        :return: 列向量，表示测试数据集的预测值 (0 还是 1)
        """
        n_test_samples = X_test.shape[0]
        y_predict = np.zeros((n_test_samples, 1))  # 预测值列向量
```

```
        for i in range(n_test_samples):
            x = X_test.iloc[i, :].values  # 测试样本行向量
            x = np.asarray(x).reshape(1, -1)
            x = np.append(x, np.ones((1, 1)), axis=1)
            y_predict[i] = 0 if self.sigmod(x, self.beta) <= 0.5 else 1

        return y_predict

    def accuracy(self, X_test, y_test):
        """
        :param X_test: 测试数据集，矩阵的每一行是一个样本，即一个行向量是一个样本
        :param y_test: 预测值列向量
        :return: 返回模型预测准确度
        """
        n_test_samples = X_test.shape[0]
        y_predict = self.predict(X_test)
        hit_samples = 0  # 表示模型预测准确的样本量

        for i in range(n_test_samples):
            if y_test[i] == y_predict[i]:
                hit_samples += 1
        accuracy = 1.0 * hit_samples / n_test_samples
        return accuracy

    @staticmethod
    def sigmod(x, beta):
        """
        :param beta: (m+1, 1) 列向量，beta=[w;b]=[[w_1],[w_2],[w_3],...,[w_m],[0]]
        :param x: (1, m+1) 一个样本数据，行向量，x=[[x_1,x_2,x_3,...,x_m,1]]
        :return: p in (0,1) 对数几率函数，返回对结果的可能性，大于 0.5 是正例，小于 0.5 是反例
        """
        eta = np.dot(x, beta)  # a real number
        if eta >= 0:  # 引入 if-else 结构避免计算概率值时，exp() 的值过大移出 (避免上溢)
            possibility = 1.0 / (1.0 + np.exp(-eta))
        else:
            possibility = np.exp(eta) / (1.0 + np.exp(eta))

        return possibility
```

5. 模型测试代码

loan_approval_LR.py

> 此 python 文件调用了 Util 文件中的若干方法，对对数几率回归模型 LogisticRegression_newton 进行：
>
> 1. **测试集上的模型性能测试**
> 2. 绘制**学习曲线**，观察训练集规模对模型拟合程度的影响

```
from LoanApprovalPredict.model.LogisticRegression.LogisticRegression_newton import LogisticRegression_newton
from Utils import *

# 导入数据，并且获得预处理过的训练集和测试集合
train, test = loan_approval_data_processed(directory='../data/loan_approval_dataset.csv', minmax=True, scale=0.75)

# 划分训练集和测试集的样本数据、标签
X_train, y_train = dataset_split(train)
X_test, y_test = dataset_split(test)

# 模型训练
lr_model = LogisticRegression_newton()
lr_model.fit(X_train, y_train)

# 模型预测精度测试
accuracy = lr_model.accuracy(X_test, y_test)
print("模型的准确率为:%2.2f" % (accuracy * 100), "%")  # 模型的准确率为:91.92 %

# 绘制学习曲线 learning curve
# 横轴：训练集的样本数量，纵轴：模型在训练集和测试集上的准确度
processed_data = loan_approval_data_processed(directory='../data/loan_approval_dataset.csv', minmax=True,
split=False)
draw_learning_curve(processed_data, LogisticRegression_newton())
```

Utils.py

```
"""
    工具文件，内含一些工具函数
"""


import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import time
from sklearn import preprocessing
from scipy.interpolate import make_interp_spline
from LoanApprovalPredict.model.LogisticRegression.LogisticRegression_newton import *
from LoanApprovalPredict.model.DecisionTree.DecisionTree import *

""" loan_approval_data: 4296 rows × 13 columns, 且数据集干净，无需缺失值处理等
    loan_id                   int64  贷款批准样本的 id (无用特征)
    no_of_dependents          int64  feature1: Number of Dependents of the Applicant(家属个数), int
    education                 object  feature2: 受教育情况(Graduate/Not Graduate), str; Mapping(Graduate:1, Not
Graduate:0)
    self_employed             object  feature3: 是否为自雇人士(Yes/No), str; Mapping(Yes:1, No:0)
    income_annum              int64  feature4: 年收入, int
    loan_amount               int64  feature5: 贷款数额, int
    loan_term                 int64  feature6: 贷款期限, int
    cibil_score               int64  feature7: 信用分数, int
    residential_assets_value  int64  feature8: 住宅资产价值, int
    commercial_assets_value   int64  feature9: 商业资产价值, int
    luxury_assets_value       int64  feature10: 奢侈品资产价值, int
    bank_asset_value          int64  feature11: 银行资产价值, int
    loan_status               object  y: 是否借贷(Approval/Rejected), str; Mapping(Approval:1, Rejected:0)
    综上所述，一个样本总共有 11 个属性 or 特征，1 个值
"""

# 导入、预处理、[划分]数据
def loan_approval_data_processed(directory='../data/loan_approval_dataset.csv', minmax=True, split=True,
scale=0.75):
    """
    :param directory: loan_approval_predict 数据集位置
    :param minmax: 是否进行归一化
    :param split: 是否将数据集划分为训练集和测试集
    :param scale: 训练集比例
    :return: train, test
    """
    # 1. 导入数据
    loan_approval_data = pd.read_csv(directory)

    # 2. 数据预处理(由于提供的数据比较干净：无缺失值，因此只需要进行①删除指定列②将字符串映射为
值，这两项工作即可)
    loan_approval_data.drop("loan_id", axis=1, inplace=True)  # 删除 loan_id 这一列的数据
    loan_approval_data.columns = loan_approval_data.columns.str.strip()  # 清除列名的前后空格
    # loan_approval_data.columns = [name.strip() for name in loan_approval_data.columns]  # 清除列名的前后空格
    # 2.1 将属性 education 中的 Graduated 映射为 1，Not Graduated 映射为 0
    loan_approval_data.loc[:, "education"] = loan_approval_data.loc[:,
```

```
                                         "education"].str.strip()  # 清除列 "education" 每一项的前后空格
    loan_approval_data.loc[:, "education"] = loan_approval_data.loc[:, "education"].map(
        {'Graduate': 1, 'Not Graduate': 0}).astype(int)
    loan_approval_data["education"] = pd.to_numeric(loan_approval_data["education"],
                                            errors='coerce')  # 更改该列数据的属性为 int 类型
    # 2.2 将属性 self_employed 中的 Yes 映射为 1， No 映射为 0
    loan_approval_data.loc[:, "self_employed"] = loan_approval_data.loc[:,
                                         "self_employed"].str.strip()  # 清除列 "self_employed" 每一项的前后空格
    loan_approval_data.loc[:, "self_employed"] = loan_approval_data.loc[:, "self_employed"].map(
        {'Yes': 1, 'No': 0}).astype(int)
    loan_approval_data["self_employed"] = pd.to_numeric(loan_approval_data["self_employed"],
                                            errors='coerce')  # 更改该列数据的属性为
    # 2.3 将属性 loan_status 中的 Approval 映射为 1， Rejected 映射为 0
    loan_approval_data.loc[:, "loan_status"] = loan_approval_data.loc[:,
                                         "loan_status"].str.strip()  # 清除列 "loan_status" 每一项的前后空格
    loan_approval_data.loc[:, "loan_status"] = loan_approval_data.loc[:, "loan_status"].map(
        {'Approved': 1, 'Rejected': 0}).astype(int)
    loan_approval_data["loan_status"] = pd.to_numeric(loan_approval_data["loan_status"], errors='coerce')  # 更改该
列数据的属性为
    # 2.4 对数据进行归一化操作
    processed_data = loan_approval_data
    if minmax:
        minmax_scaler = preprocessing.MinMaxScaler()
        minmax_data = minmax_scaler.fit_transform(loan_approval_data)
        processed_data = pd.DataFrame(minmax_data, columns=loan_approval_data.columns)

    # 3. 划分训练集和验证集
    if split:
        train = processed_data.sample(frac=scale, random_state=int(time.time()))
        test = processed_data.drop(train.index)
        train.reset_index(drop=True, inplace=True)
        test.reset_index(drop=True, inplace=True)

        return train, test
    else:
        return processed_data


# 将数据集划分为样本数据、标签
def dataset_split(data):
    X = data.iloc[:, 0:-1]
    y = data.iloc[:, -1]
    return X, y


def draw_learning_curve(data, model, start=0.001, end=0.3, step=0.001):
```

```
plt.xlabel("Number of training samples")  # 横轴
plt.ylabel("Accuracy")  # 纵轴

# 计算出对应不同训练集规模的时，模型在训练集和测试集合上的比重
train_size_list = list(np.arange(start, end, step))  # 训练集占全部数据集的比重
number_of_training_samples_list = [int(size * data.shape[0]) for size in
                                   train_size_list]  # 训练集的样本数量(横坐标的值)
train_accuracy_list = []  # 训练集随样本数的准确度取值(纵坐标)
test_accuracy_list = []  # 测试集随样本数的准确度取值(纵坐标)
# 获取两个图像的纵坐标取值
for frac in train_size_list:
    # a = time.time()
    # 按照预定的比例划分数据集和测试集合
    train = data.sample(frac=frac, random_state=int(time.time()))
    test = data.drop(train.index)

    if isinstance(model, LogisticRegression_newton):
        plt.title("learning curve(Logistic Regression)")  # 标题
        # 重置索引
        train.reset_index(drop=True, inplace=True)
        test.reset_index(drop=True, inplace=True)
        # 划分训练集和测试集的样本数据、标签
        X_train, y_train = dataset_split(train)
        X_test, y_test = dataset_split(test)
        # 模型训练
        model.fit(X_train, y_train)
        # 添加纵坐标值
        train_accuracy = model.accuracy(X_train, y_train)
        train_accuracy_list.append(train_accuracy)
        test_accuracy = model.accuracy(X_test, y_test)
        test_accuracy_list.append(test_accuracy)
    else:
        plt.title("learning curve(Decision Tree)")  # 标题
        # 重置索引
        train.reset_index(drop=True, inplace=True)
        test.reset_index(drop=True, inplace=True)
        # 模型训练
        model.fit(train)
        # 添加纵坐标值
        train_accuracy = model.accuracy(train)
        train_accuracy_list.append(train_accuracy)
        test_accuracy = model.accuracy(test)
        test_accuracy_list.append(test_accuracy)

    # b = time.time()
    # print("\nfrac=%.2f" % frac)
```

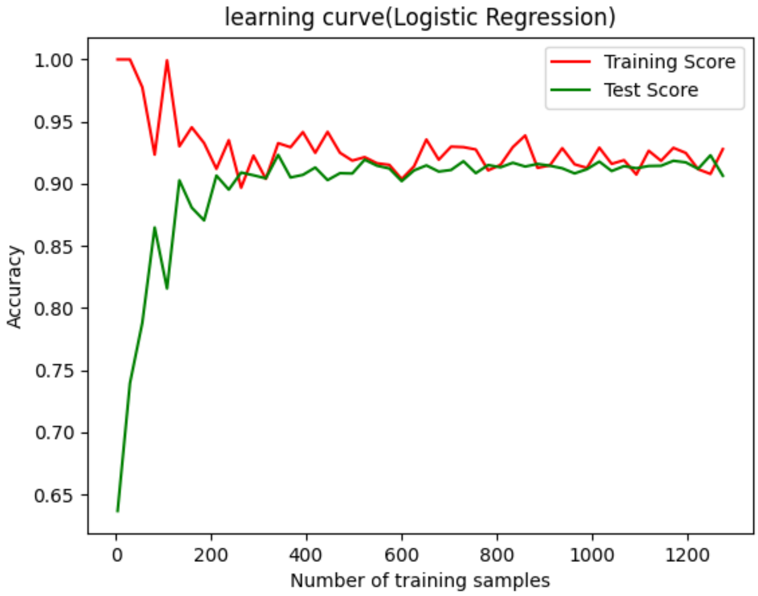
```python
        # print("time=%.2f" % (b - a))
        # print("train_acc%.2f" % train_accuracy)
        # print("test_acc%.2f" % test_accuracy)

    # 绘图
    # 对 x、y_train 插值
    number_of_training_samples_list_smooth = np.linspace(min(number_of_training_samples_list),
                                                         max(number_of_training_samples_list), 50)
    train_accuracy_list_smooth = make_interp_spline(number_of_training_samples_list, train_accuracy_list)(
        number_of_training_samples_list_smooth)
    plt.plot(number_of_training_samples_list_smooth, train_accuracy_list_smooth, color='r', label='Training Score')
    # 对 y_test 插值
    test_accuracy_list_smooth = make_interp_spline(number_of_training_samples_list, test_accuracy_list)(
        number_of_training_samples_list_smooth)
    plt.plot(number_of_training_samples_list_smooth, test_accuracy_list_smooth, color='g', label='Test Score')
    plt.legend(loc='best')
    plt.show()


# 返回模型训练时间
def time_consumption(data, model):
    start = time.time()
    if isinstance(model, DecisionTree):
        model.fit(data)
    elif isinstance(model, LogisticRegression_newton):
        X, y = data
        model.fit(X, y)
    else:
        raise Exception
    end = time.time()
    return end - start
```

5. 结果分析

1. 给定训练集比例 0.75，测试集比例 0.25，模型的准确率为 91.19 %
2. 模型的**学习曲线**绘制如下，发现
    - 当训练集规模 < 200 时，模型出现**欠拟合**状态，即模型在训练集上表现良好，在测试集上表现不佳
    - 当训练集规模 > 200 时，模型表现**趋于稳定**，预测准确率在 92% 左右


learning curve(Logistic Regression)
Training Score
Test Score
Accuracy
Number of training samples
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
200
400
600
800
1000
1200