Python编程及人工智能应用

第三章 线性回归及Python编程

https://bolei-zhang.github.io/course/python-ai.html

内容提要

- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 自定义求解单变量线性回归
  - 基于最小二乘法
  - 基于梯度下降法
- 多变量线性回归问题

人工智能 vs 机器学习

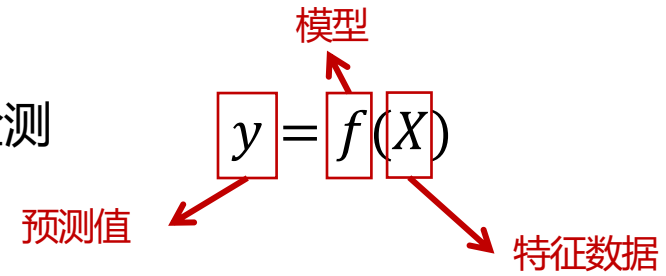
- 符号主义（Symbolism）：
  - 一种基于逻辑推理的智能模拟方法
  - 又称逻辑主义、心理学派或计算机学派。
  - 启发式算法→专家系统→知识工程，知识图谱

- 连接主义（Connectionism）：
  - 又称仿生学派或生理学派
  - 是一种基于神经网络及网络间的连接机制与学习算法的智能模拟方法，比如深度学习
  - 从历史经验中去学习

- 行为主义（Actionism）：
  - 强化学习：“感知——行动”的行为智能模拟方法

机器学习

- 机器学习
  - 让计算机模仿人类，从过去经验中学习一个“模型”，通过学到的模型再对新情况给出一个预测
  - “经验”通常是以“数据”的形式存在
- 机器学习“预测”场景
  - 根据房屋位置、面积、装修等预测房价
  - 根据图像预测图像的分类，对图像进行分割、检测
  - 机器翻译、自动问答、文本理解
- 监督学习
  - 分类（classification）：预测的值是“离散”的
  - 回归（regression）：预测的值是“连续”的



问题定义

- 数据集
  - 训练数据集(X_{train}, y_{train}) 特征 feature 标签 label

样本 instance

x_1	x_2	x_3	x_4	x_5	…	y
1.2	100	200	3	12		5.2
3.5	200	35	12	4.6		7.8
4.5	7.2	100	5	13.5		9.2
…	…	…	…	…		…

  - 测试数据集X_{test}

x_1	x_2	x_3	x_4	x_5	…	y
1.8	200	20	4	13		?
13.5	300	15	11	4.8		?
14.5	52	2	3	12.5		?

- 目标
  - 从训练数据集中学习模型f（如何形式化f）
  - 使得在测试数据集中$f(X_{test})$和真实的y_{test}尽量接近（如何定义接近）

线性回归问题与模型

- 基本形式
  - 给定有d个属性的样例$x = (x_1; x_2; x_3; \ldots; x_d)$，其中$x_i$是$x$在第$i$个属性上的取值，线性回归试图学习得到一个预测函数$f(x)$，该函数的值是各属性值的线性加权和，公式如下

$$f(x) = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + b$$

- 向量形式为：$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$
  - w和b是可学习和调整的参数，可自动从数据中学习获得
  - 预测某套商品房的总价

$$f(\mathbf{x}) = 3 \times \text{面积大小} + 0.5 \times \text{楼层指数} + 0.2 \times \text{卧室数量指数}$$

- 单变量线性回归与多变量线性回归

单变量线性回归问题

- 当样本仅1个属性时（即只有x_1），只要求解两个参数（w_1和b），是单变量线性回归模型

$$f(x) = w_1 x_1 + b$$

- 案例描述：设某小区通过某房产中介处已售出5套房，房屋总价与房屋面积之间有如下的数据关系。现有该小区的一位业主想要通过该房产中介出售房屋，在业主报出房屋面积后，根据训练数据，中介能否能估算出该房屋的合适挂售价格？

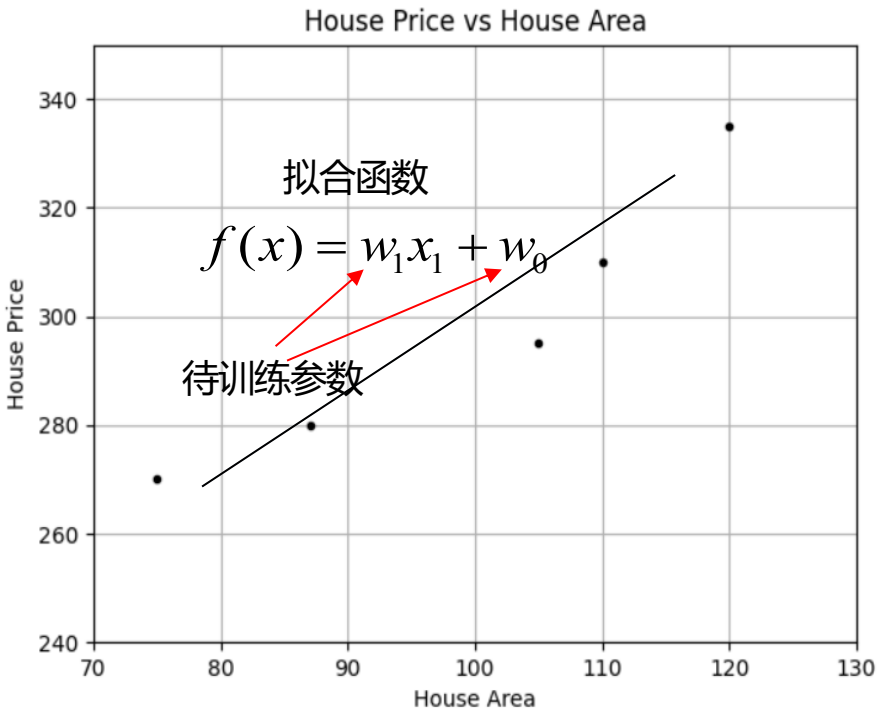
训练样本	房屋面积（平方米）	房屋总价（万元）
1	75	270
2	87	280
3	105	295
4	110	310
5	120	335

案例分析

•把房屋面积看成自变量x，房屋总价看成因变量y，先通过绘图看出二者之间的关系。

1. 房屋总价随着房屋面积的变化，大致呈现线性变化趋势；
2. 如果根据现有的训练样本数据能够拟合出一条直线，使之与各训练样本数据点都比较接近，那么根据该直线，就可以计算出任意房屋面积的房屋总价了。

```
    # 设置x轴和y轴的值域分别为70~130和240~350
    plt.axis([70, 130, 240, 350])
    plt.grid(True) #显示网格
    return plt
plt = initPlot()
xTrain = np.array([75, 87, 105,110,120])
yTrain = np.array([270,280,295,310,335])
plt.plot(xTrain, yTrain, 'k.') #k表示绘制颜色为黑色，点表示绘制散点图
plt.show()
```



模型拟合

- 使得在测试数据集中$f(X_{test})$和真实的y_{test}尽量接近
- 悲观的是，我们并不知道y_{test}

- 退而求其次，希望在训练数据集中让$f(X_{train})$和真实的y_{train}尽量接近
  - 在测试数据集中有泛化性（generalization）吗？
  - 如何定义接近？

- 对于回归问题，可以定义不同损失函数（loss function）
  - 均方误差（mean squared error）

$$mse = \sum_{i=1}^{N} (f(X_i) - y_i)^2$$

LinearRegression类

- Scikit-learn提供了sklearn.linear_model.LinearRegression线性回归类，可以解决大部分常见的线性回归操作

- 构造方法

  model = LinearRegression (fit_intercept = True, normalize = False, copy_X = True, n_jobs = 1)

  - fit_intercept：是否计算模型的截距，默认值是True，为False时则进行数据中心化处理；
  - normalize：是否归一化，默认值是False；
  - copy_X：默认True，否则X会被改写；
  - n_jobs：表示使用CPU的个数，默认1，当-1时，代表使用全部的CPU

LinearRegression类

- LinearRegression类的属性和方法
  - coef_：训练后的输入端模型系数，如果label有两个，即y值有两列，是一个2D的数组；
  - intercept_：截距，即公式 中的w_0值；
  - fit(x, y)：拟合函数，通过训练数据x和训练数据的标签y来拟合模型；
  - predict(x)：预测函数，通过拟合好的模型，对数据x预测y值；
  - score：评价分数值，用于评价模型好坏；

思考

- 通过以下两个特征是否可以用LinearRegression实现房价预测
  - 房子长度
  - 房子宽度

Python编程及人工智能应用

第三章 线性回归及Python编程

https://bolei-zhang.github.io/course/python-ai.html

内容提要

- 线性回归问题简介
- 单变量线性回归问题
- 基于Scikit-learn库求解单变量线性回归
- 多变量线性回归问题
- 自定义求解单变量线性回归
  - 基于最小二乘法
  - 基于梯度下降法

求解步骤

- 第一步：准备训练数据
  - X_train = np.array([[75], [87], [105], [110], [120]])
  - y_train = np.array([270, 280, 295, 310, 335])
- 第二步：创建模型对象
  - model = LinearRegression ( )
- 第三步：执行拟合
  - **model.fit(X_train, y_train)**
- 第四步：准备测试数据
  - X_test = np.array([[82], [104]])
  - model.predict (X_test)

问题1：如何优化模型？

问题2：模型结果是否准确？

最小二乘法求解

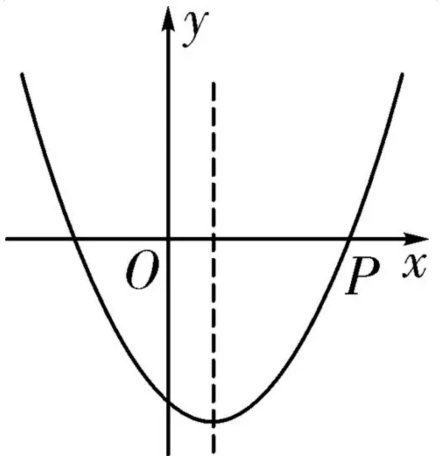
- **训练目标**是最小化训练数据残差平方之和
  - 为什么选择这个目标
  - 误差符合高斯分布
  - 其他损失函数：huber loss, quantile loss
- 采用数据的残差平方和和作为优化目标，求解最优的参数（w和b），使得残差平方和最小化，这种方法被称为最小二乘法（Ordinary least squares method）

单变量线性回归求解

- 优化目标

$$\min \sum_{i=1}^{n} \left((w_1 x_{i1} + \mathrm{w}_0) - y_i \right)^2$$

- 导数为0的位置取到最小值



导数法求解最小二乘法

$$\mathrm{L} = \sum_{i=1}^{n} \left((w_1 x_{i1} + \mathrm{w}_0) - y_i \right)^2$$

- 分别对w0和w1求一阶偏导，并使之为0

- 求得 $\begin{cases} \frac{\partial L}{\partial w_0} = 2\sum_{i=1}^{n} \left((w_1 x_{i1} + \mathrm{w}_0) - y_i \right) = 0 \\ \frac{\partial L}{\partial w_1} = 2\sum_{i=1}^{n} x_{i1} \left((w_1 x_{i1} + \mathrm{w}_0) - y_i \right) = 0 \end{cases}$ ，

- 最终公式： $w_1 = \frac{\sum_{i=1}^{n} y_i (x_i - \bar{x})}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_{i1} \right)^2}$ $w_0 = \frac{1}{n} \sum_{i=1}^{n} (y_i - w_1 x_{i1})$

线性回归问题具有解析解（closed-form solution）

代码

```
#代码3.4 使用导数求解最小二乘法
import numpy as np
X_train = np.array ( [ 75, 87, 105, 110, 120 ] ) # 训练数据(面积)
y_train = np.array( [ 270,280,295,310,335 ] )          # 训练数据(总价)

w1 = np.cov (X_train , y_train, ddof = 1) [ 1, 0 ] / np.var ( X_train, ddof = 1 )
w0 = np.mean ( y_train ) - w1 * np.mean ( X_train )
```

w1= 1.350592165198907

w0= 163.75113877922863

多变量线性回归

- 使用矩阵法对最小二乘法的优化目标进行求解 $y_i = \mathbf{w}^{\mathbf{T}}\mathbf{x_i} + b$

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} & 1 \\ x_{21} & x_{22} & \dots & x_{2d} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nd} & 1 \end{bmatrix}$$

- 目标损失函数:

$$\mathrm{L} = (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

- 对**w**求导[1]：

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{X}^{\mathbf{T}}(\mathbf{X}\mathbf{w} - \boldsymbol{y})$$

- 当XX^{T}为可逆时求得结果

$$w^* = \left(\mathbf{X}^{\mathbf{T}}\mathbf{X}\right)^{-\mathbf{1}}\mathbf{X}^{\mathbf{T}}\mathbf{y}$$

[1] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

代码

#代码3.5 求参数w的矩阵方法代码：linreg_matrix.py

```
import numpy as np
def linreg_matrix ( x, y ) :
    X_X_T = np.matmul ( x, x.T )
    X_X_T_1 = np.linalg.inv ( X_X_T )
    X_X_T_1_X = np.matmul ( X_X_T_1, x )
    X_X_T_1_X_Y_T = np.matmul ( X_X_T_1_X, y.T )
    return X_X_T_1_X_Y_T
```

```
x= [[  1   1   1   1   1]
 [ 75  87 105 110 120]]
y= [[270 280 295 310 335]]
w= [[163.75113878]
 [  1.35059217]]
```

```
X_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] )
y_train = np.array ( [ 270, 280, 295, 310, 335 ] ) [ :, np.newaxis ]

def make_ext ( x ) : #对x进行扩展，加入一个全1的行
    ones = np.ones ( 1 ) [ :, np.newaxis ] #生成全1的行向量
    new_x = np.insert ( x, 0, ones, axis = 0 )
    return new_x

x = make_ext ( X_train.T )
y = y_train.T
print ( "x=", x )
print ( "y=", y )
w = linreg_matrix ( x, y )
print ( "w=", w )
```

开始训练模型

```
# 以矩阵形式表达(对于单变量，矩阵就是列向量形式
X_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] ) )
y_train = np.array ( [ 270, 280, 295, 310, 335 ] )

# 模型训练
model = LinearRegression ( ) # 创建模型对象
model.fit ( X_train, y_train )      # 根据训练数据拟合出直线(以得到假设函数)

#查看模型参数
print ( "截距b或w0=", model.intercept_ )    # 截距
print ( "斜率w1=", model.coef_ )          # 斜率

# 模型预测
X_test = np.array ( [ [85], [90], [93], [109] ] )
y_pred= model.predict ( X_test )
print ( "预测新房源数据的总价：", y_pred )
```

```
def initPlot ( ) :
    plt.figure ( )
    plt.title ( 'House Price vs House Area' )
    plt.xlabel ( 'House Area' )
    plt.ylabel ( 'House Price'  )
    # 设置x轴和y轴的值域分别为70~130和240~350
    plt.axis ( [ 70, 130, 240, 350 ] )
    plt.grid ( True )
    return plt

plt = initPlot ( )
plt.plot ( X_train, y_train, c='k') #格式字符串'k.'，表示绘制黑色的散点

#画出蓝色的拟合线
plt.plot ( [ [ 70 ], [ 130 ] ], model.predict ( [ [ 70 ], [ 130 ] ] ), 'b-' )
plt.show ( )
```

代码3.2运行结果

•输出

截距b或w0= 163.75113877922868

斜率w1= [1.35059217]

新房源数据的面积： [[85], [90], [93], [109]]

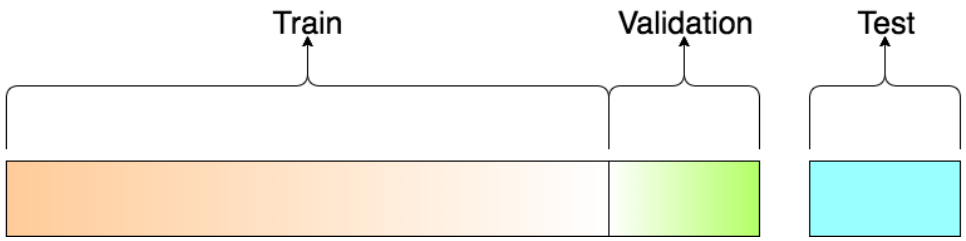
预测新房源数据的总价：

[278.55, 285.30, 289.35, 310.96]

问题2：预测的模型是否准确？

模型结果评价

- 数据集
  - 1. 在测试数据集X_test中，并不知道测试数据集中的数据所对应的真实的结果，因此无法用测试数据集来评估模型
  - 2. 在训练数据集（X_train, y_train）中，虽然可以进行比较，但该结果是否在测试数据集中也有效？
  - 3. 验证数据集：
    - 从训练数据集中切分出一部分数据
    - 该部分数据并不参与训练



- 用什么指标对拟合模型进行评价？
  - 计算回归（regression）误差的指标主要包括
    - MSE（均方误差）
    - r2（r-squared，决定系数， coefficient of determination ）
    - mape（平均百分比误差） $MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$
  - 具体采用哪一种指标更合理是和实际问题高度相关的

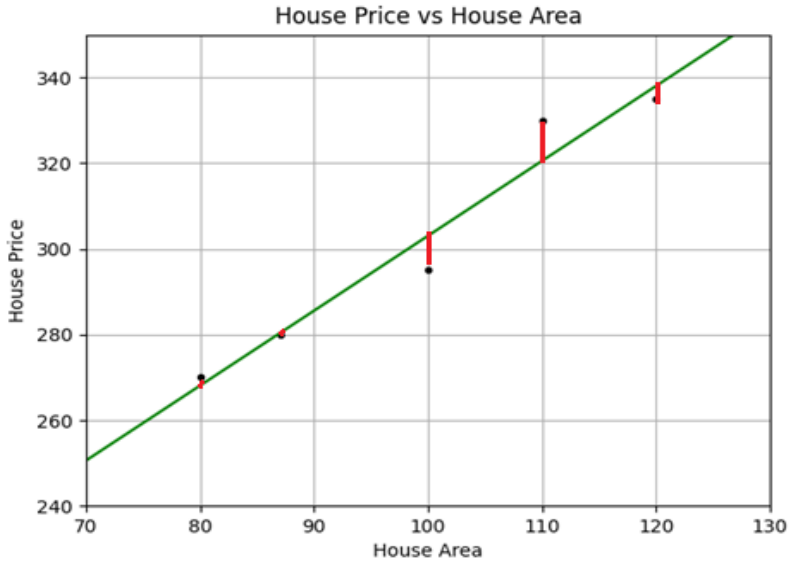
重新回顾这个问题

- 第一步：准备训练数据和验证数据
  - X_train = np.array([[75], [87], [105], [110], [120]])
  - y_train = np.array ([270, 280, 295, 310, 335]
  - X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2)
- 第二步：创建模型对象
  - Model = LinearRegression ( )
- 第三步：执行拟合
  - model.fit (X_train, y_train)
- 第四步：模型验证
  - y_pred = model.predict(X_val)
  - r2 = r2_score(y_val, y_pred)
  - 如果结果不好，则回到第一步，考虑优化模型
- 第四步：准备测试数据并预测
  - X_test=np.array([[82], [104]])
  - mode.predict (np.array ([[70]]))

残差

- 实际y值与通过模型判别函数计算的y值之间的差异
- 残差平方和是训练数据或者测试数据中所有样本残差的平方和
- 残差值越小则对应数据的拟合度越好，当残差为0时，预测 y 值与实际 y 值完全一致。

$$SS_{res} = \sum_{i=1}^{m} (f(x_i) - y_i)^2$$



R方

- R方（R-Square），又称决定系数（coefficient of determination）
  - 表达因变量与自变量之间的总体关系
  - 反映了因变量y（标签）的波动，有多少百分比能被自变量x（特征）的波动所描述
  - 与残差平方和在方差中所占的比率有关

$$SS_{res} = \sum_{i=1}^{m} (f(x_i) - y_i)^2 \qquad SS_{total} = \sum_{i=1}^{m} (\bar{y} - y_i)^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{total}}$$

  - r2 为0、1分别代表什么？

残差与R方的计算

- LinearRegression提供的自动计算方法：
  - 训练数据残差平方和：model._residues，通过训练数据训练模型后自动获得；
  - 自动计算R方的函数：
    - metrics.r2_score(X_test, y_test)
    - model.score(X_test, y_test)

代码

```
#代码3.3 加入评价的基于Scikit-learn实现房价预测线性回归代码
X_train = np.array ( [ [ 75 ], [ 87 ], [ 105 ], [ 110 ], [ 120 ] ] ) # 训练数据(面积)
Y_train = np.array ( [ 270,280,295,310,335 ] )          # 验证数据(总价)
X_val = np.array ( [ 85, 90, 93, 109 ] ) [ :, np.newaxis ] #验证数据(面积)
y_val = np.array ( [ 280, 282, 284, 305 ] )             # 测试数据(总价)
model = LinearRegression ( )
model.fit ( X_train, y_train )

y_pred = model.predict ( X_val )  # 针对验证数据进行预测
ss_res = sum ( ( y_pred – y_val) ** 2) # 计算验证数据集残差
ss_total = sum ( ( np.mean ( y_val ) – y_val ) ** 2 )     # 手动计算验证数据集y值偏差平方和
r2 = 1 – ss_res / ss_total          # 手动计算验证数据R方
print (r2 )
print ( model.score ( X_val, y_val ) )                  # 自动计算的验证数据集的R方
def initPlot ( ) :
    plt.figure ( )
    plt.title ( 'House Price vs House Area' )
    plt.xlabel ( 'House Area' )
    plt.ylabel ( 'House Price' )
    plt.axis ( [ 70, 130, 250, 350 ] ) # 设置x轴和y轴的值域分别为70~130和240~350
    plt.grid ( True )
    return plt
plt = initPlot ( )
plt.plot ( X_train, y_train, 'r.' )         # 训练点数据(红色，小点)
plt.plot ( X_val, y_val, 'bo' )             #验证数据(蓝色，大点)
plt.plot ( X_val, y_pred, 'g-' ) # 拟合的函数直线(绿色)
plt.show ( )
```

227.29

227.29

0.80

0.80

思考

- 对于样本、特征非常多的数据集，例如100万个样本，1000维特征，用最小二乘法求解是否合适？
- 如果数据集的特征多余样本，例如1000个样本，10000维特征，是否可以用最小二乘法？