机器学习经典算法学习笔记

第一章 线性回归

1.1 原理

线性回归为**有监督算法（需要标签）**。

假设 θ_1 是 x_1 的参数，θ_2 是 x_2 的参数

拟合的平面： $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

整合：

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

误差：

真实值和预测值之间 存在的差异（ε）

对于每个样本：

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)} \tag{1}$$

误差 $\varepsilon^{(i)}$ 是独立同分布，是服从均值为 0 方差为 θ^2 的高斯分布 （机器学习的基础）

由于误差服从高斯分布：

$$p(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}) \tag{2}$$

将（1）式带入（2）式：

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

求得 (x, θ) 与 y 越接近越好！

似然函数：

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

对数似然：

$$log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

展开化简：

$$\sum_{i=1}^{m} log\frac{1}{\sqrt{2\pi}\sigma}exp(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2})$$

$$= m \cdot log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2}\sum_{i=1}^{m}(y^{(i)}-\theta^T x^{(i)})^2$$

目标：是让似然函数越大越好

最小二乘法：

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)}-\theta^T x^{(i)})^2$$

目标：让$J(\theta)$越小越好！（目标函数，损失函数 Loss Function）

目标函数求解

目标函数：

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 = \frac{1}{2}(X\theta-y)^T(X\theta-y)$$

求偏导：

$$\bigtriangledown_\theta J(\theta) = \bigtriangledown_\theta(\frac{1}{2}(X\theta-y)^T(X\theta-y)$$

$$= \bigtriangledown_\theta(\frac{1}{2}(\theta^T X^T - y^T)(X\theta-y))$$

$$= \bigtriangledown_\theta(\frac{1}{2}(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y))$$

$$= \frac{1}{2}(2X^T X\theta - X^T y - (y^T X)^T) = X^T X\theta - X^T y$$

偏导等于0：

$$\theta = (X^T X)^{-1} X^T y$$

梯度下降

寻找目标函数的“终点”（什么样的参数能使目标函数达到极值点）

目标函数：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))^2$$

批量梯度下降：

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i \theta_j' = \theta_j + \frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$$

(容易得到最优解，但是由于每次考虑所有样本，速度很慢)

随机梯度下降(SGD)：

$$\theta_j' = \theta_j + (y^i - h_\theta(x^i))x_j^i$$

（每次找一个样本，迭代速度快，但不一定每次都朝着收敛方向）

小批量梯度下降法（mini batch）：

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(每次更新选择一小部分数据来算，实用！)

学习率（步长）learning rate：对结果产生巨大的影响，一般小一些

1.2 代码实现

1.2.1 模块编写

归一化模块

normalize.py

计算特征（输入）的均值feature_mean，标准差feature_deviation。

归一化操作：

$$features\_normalized = \frac{x - \mu}{\sigma}$$

返回归一化后的特征，均值，标准差

训练前要做的准备

prepare_for_traning.py

- 计算样本总数
- 预处理（调用normalize模块）
- 返回处理后的数据，均值，标准差

定义线性回归类

linear_regression.py

1.2.2 单变量线性回归实现

源码

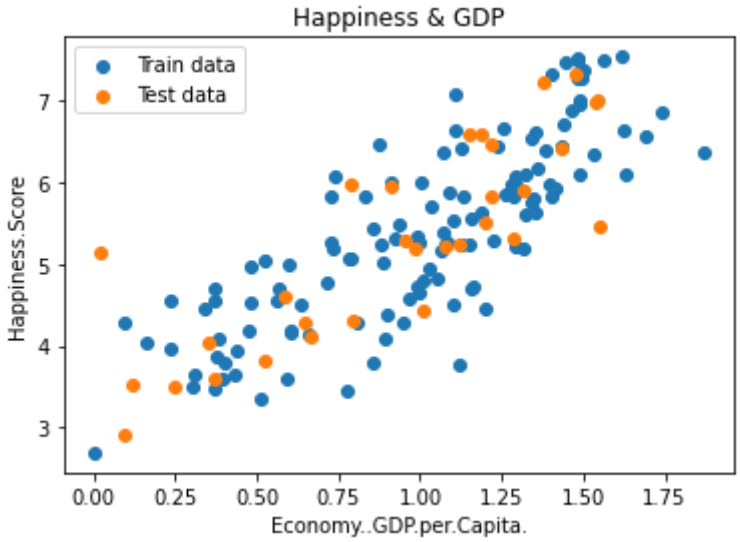
使用的数据为world_happiness_report2017.csv

目的：训练出GDP与幸福指数之间的线性回归方程

导入数据：

```
data = pd.read_csv('data/world_happiness_report2017.csv')
train_data = data.sample(frac=0.8)
test_data = data.drop(train_data.index)
input_param_name = 'Economy..GDP.per.Capita.'
output_param_name = 'Happiness.Score'
x_train = train_data[[input_param_name]].values
y_train = train_data[[output_param_name]].values
x_test = test_data[[input_param_name]].values
y_test = test_data[[output_param_name]].values
```

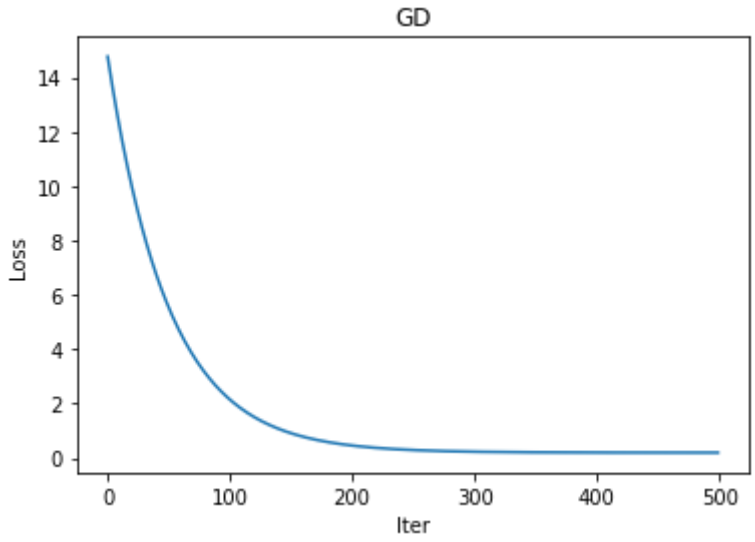
导入后显示：



训练模型：

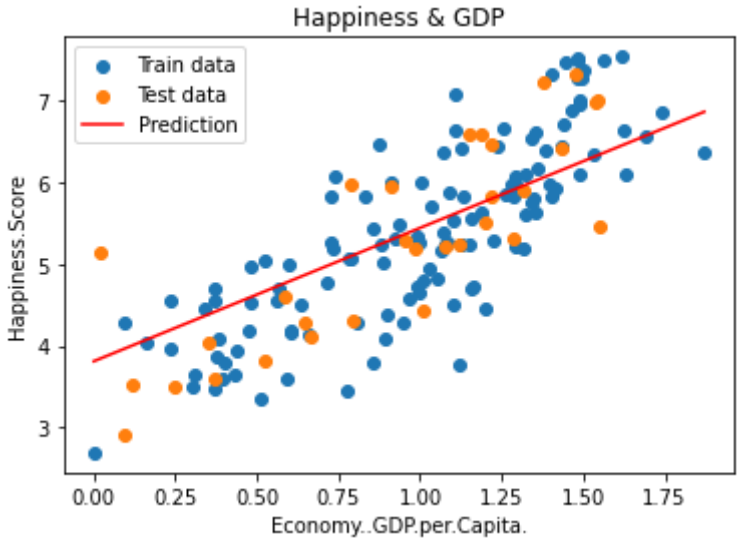
```
num_iterations = 500
lr = 0.01
linear_regression = LinearRegression(x_train, y_train)
(theta, loss_history) = linear_regression.train(lr, num_iterations)
```

梯度下降：



测试及回归结果：

```
preditions_num = 100
x_predictions = np.linspace(x_train.min(), x_train.max(),
preditions_num).reshape(preditions_num, 1)
y_predictions = linear_regression.predict(x_predictions)
```



1.2.3 多特征建立线性回归模型

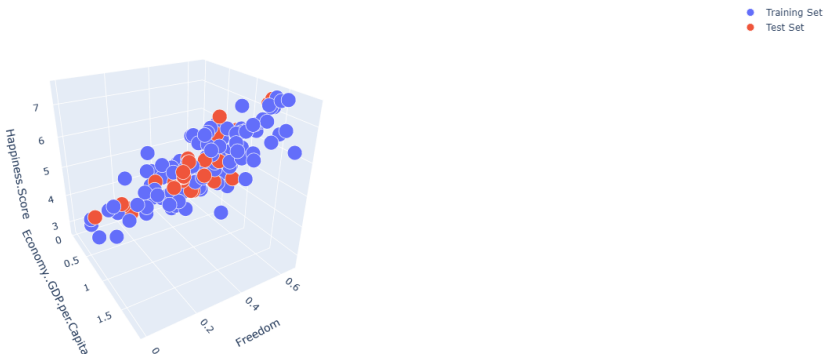
源码

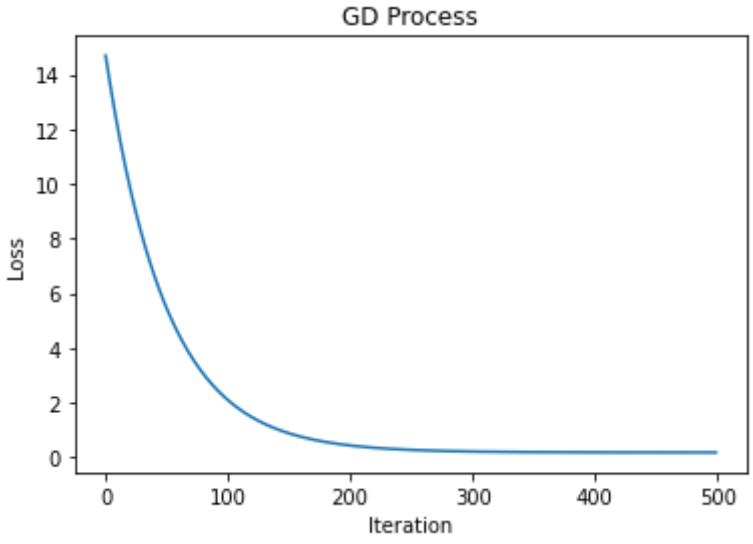
使用的数据为world_happiness_report2017.csv

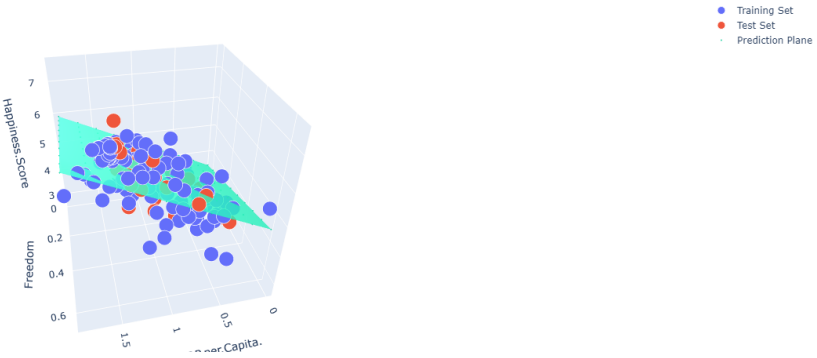
目的：训练出GDP，Freedom与幸福指数的线性回归模型

核心代码：

```
linear_regression = LinearRegression(x_train, y_train, polynomial_degree, sinusoid_degree)
(theta, loss_history) = linear_regression.train(lr, num_iterations)
```





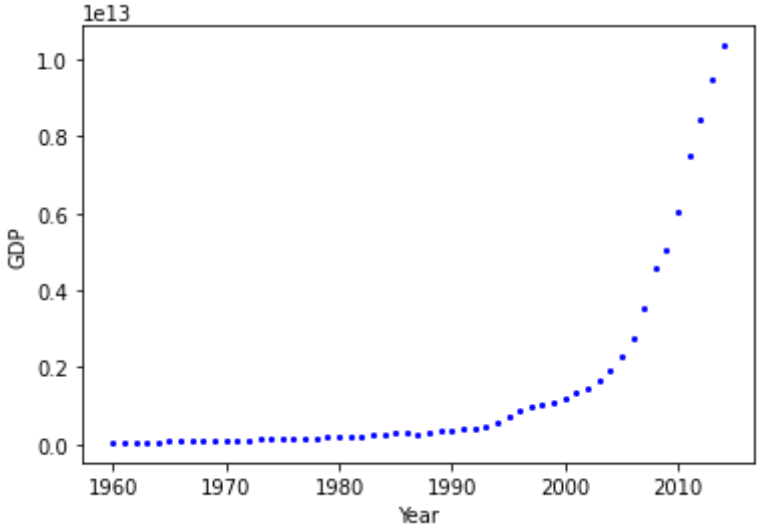


1.2.4 非线性回归模型

源码

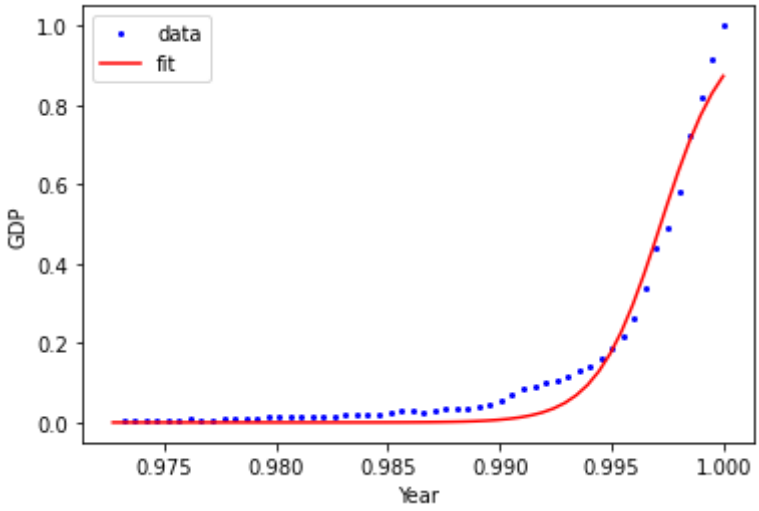
使用的数据为china_gdp.csv

导入数据：



使用 $sigmoid$ 函数进行拟合：

$$Y = \frac{1}{1 + e^{\beta_1 (X - \beta_2)}}$$



第二章 模型评估方法

notebook

2.1 交叉验证

交叉验证（循环估计），是一种统计学上将数据样本切分成较小子集的实用方法。将数据集切分为训练集和测试集(test data)，其中在训练集内再切分为训练集(train date)和验证集(validation data)。交叉验证的目的，是用未训练过的新数据测试模型的性能，以便减少诸如过拟合和选择偏差等问题。

2.1.1 LOOCV

LOOCV(Leave-one-out-cross-validation)方法是只用一个数据作为验证集，其他数据都作为训练集，并将此步骤重复N次（N为数据集的数据数量）。

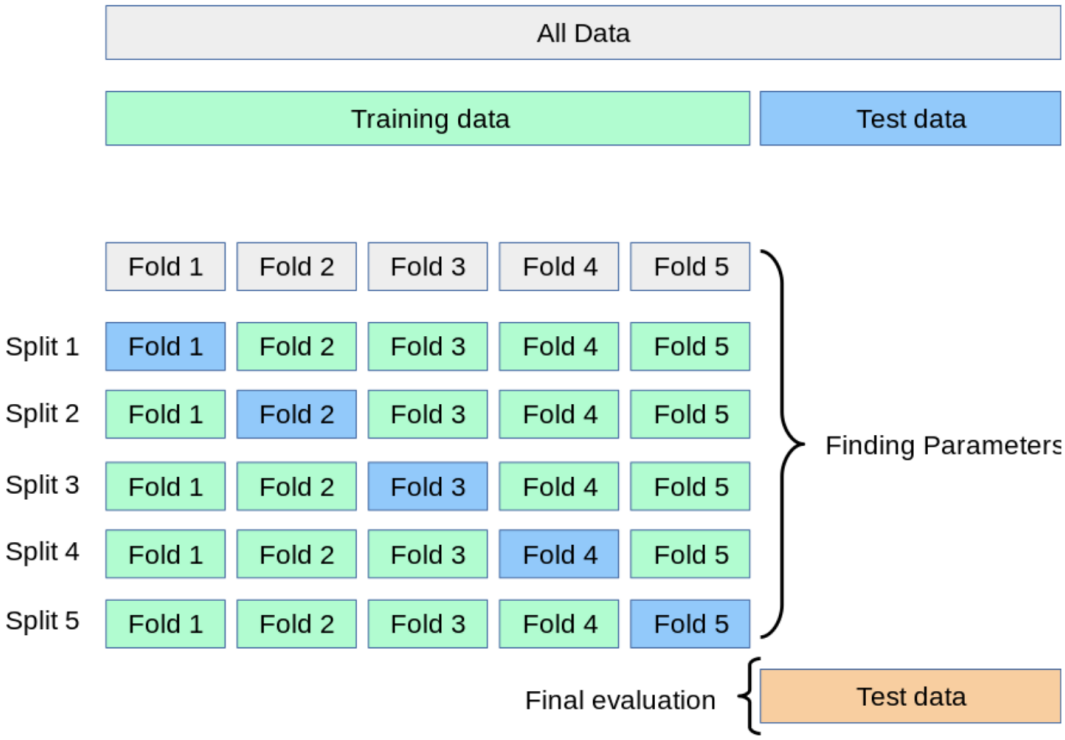
假设现在有n个数据组成的数据集，那么LOOCV方法就是每次取出一个数据作为验证集的唯一元素，而其他n-1个数据都作为训练集用于训练模型和调参。结果就是我们最终训练了n个模型，每次都能得到一个MSE。

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

2.1.2 K-fold Cross Validation

K折交叉验证，每次的验证集不再只包含一个数据，具体数目将根据k的选取决定。比如K=5：

- 将所有的训练集分成5份
- 不重复地每次取其中一份做为验证集，用其他四份做训练集训练模型，之后计算该模型在验证集上的MSE_i
- 将5次的 MSE_i 取平均



$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

不难理解，LOOCV是一种特殊的K-fold Cross Validation(K=N)。

2.1.3 Bias-Variance Trade-Off for K-Fold Cross-Validation

对于K的选取。K越大，每次投入的训练集的数据就越多，模型的偏移越小；但是K越大，又意味着每次选取的训练集之间的相关性越大，而这种相关性会导致结果有更大的方差。一般来说，根据经验，选择K=5或者10。

2.1.4 Cross-Validation on Classification Problems

对于分类问题，可以用下式衡量：

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i$$

其中 Err_i 表示第 i 个模型在第 i 组验证集上的分类错误的个数。

2.2 Confusion Matrix 混淆矩阵

二分类问题	相关（Relevant），正类	无关（NonRelevant），负类
被检索到（Retrieved）	true positive(TP 正类判定为正)	false positive(FP 负类判定为正类)

二分类问题	相关（Relevant），正类	无关（NonRelevant），负类
未被检索到（Not Retrieved）	false negative(FN 正类判定为负类)	true negative(TN 负类判定为负类)

2.2.1 准确率

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{all\ data}$$

在数据集不平衡时，准确率将不能很好的表示模型的性能。可能会存在准确率很高，而少数类样本全分错的情况，此时应该选择其他模型评价指标。

2.2.2 Precision & Recall

$$precision = \frac{TP}{TP + FP}$$

表示在预测为 positive 的样本中真实类别为 true 的样本所占比例。

$$recall = \frac{TP}{TP + FN}$$

表示在真实类别为 true 的样本中模型预测为 positive 的样本所占比例。

2.2.3 F_1 score

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

F_1 值就是 precision 和 recall 的调和平均值。如果召回率和查准率都很高，分类器将获得高 F_1 分数。

2.2.4 ROC curves

recelver operating characteristic (ROC) 曲线是二元分类中的常用评估方法

ROC曲线的纵坐标 TPR 在数值上等于 positive class 的 recall；横坐标 FPR 在数值上等于 （1- negative class 的 recall）：

$$TPR = \frac{TP}{TP + FN} = recall_{positive}$$

$$FPR = \frac{FP}{FP + TN} = \frac{FP + TN - TN}{FP + TN}$$

$$= 1 - \frac{TN}{FP + TN} = 1 - recall_{negative}$$

第三章 逻辑回归

- 目的： 经典的二分类算法

- 机器学习算法选择：先逻辑回归再用复杂的，能用简单还是用简单的
- 逻辑回归的决策边界：可以是非线性的

3.1 Sigmoid 函数

$$g(z) = \frac{1}{1+e^{-z}}$$

- 自变量取值为任意实数，值域[0, 1]
- 将任意的输入映射到了 [0,1] 区间。在线性回归中可以得到一个预测值，再将该值映射到 sigmoid 函数中，这样就完成了由值到概率的转换，也就是分类任务。
- 预测函数：

  $$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

  其中

  $$\theta_0 + \theta_1 x_1 +, \cdots, +\theta_n x_n = \sum_{i=1}^{n} \theta_i x_i = \theta^T x$$

- 分类任务：

  $$P(y=1|x;\theta) = h_\theta(x)$$

  $$P(y=0|x;\theta) = 1 - h_\theta(x)$$

  $\Rightarrow$ 整合：

  $$P(y|x;\theta) = (h_\theta(x))^y (1-h_\theta(x))^{1-y} \qquad (*)$$

- 上式中对于二分类任务（0，1），整合后 y 取 0 只保留了后一项，取 1 只保留了前一项。

3.2 原理

- (*) 式的似然函数：

  $$L(\theta) = \prod_{i=1}^{m} P(y_i|x_i;\theta) = \prod_{i=1}^{m} (h_\theta(x_i))^{y_i} (1-h_\theta(x_i))^{1-y_i}$$

- 对数似然：

  $$l(\theta) = logL(\theta) = \sum_{i=1}^{m} (y_i log h_\theta(x_i) + (1-y_i) log(1-h_\theta(x_i)))$$

  目标：似然函数越大越好，应为梯度上升求最大值。

引入 $J(\theta) = -\frac{1}{m} l(\theta)$ **转换为梯度下降任务**

求导过程：

$$\frac{\delta}{\delta_{\theta_j}}J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}(y_i\frac{1}{h_\theta(x_i)}\frac{\delta}{\delta_{\theta_j}}h_\theta(x_i)-(1-y_i)\frac{1}{1-h_\theta(x_i)}\frac{\delta}{\delta_{\theta_j}}h_\theta(x_i))$$

$$=-\frac{1}{m}\sum_{i=1}^{m}(y_i\frac{1}{g(\theta^Tx_i)}-(1-y_i)\frac{1}{1-g(\theta^Tx_i)})\frac{\delta}{\delta_{\theta_j}}g(\theta^Tx_i)$$

$$=-\frac{1}{m}\sum_{i=1}^{m}(y_i\frac{1}{g(\theta^Tx_i)}-(1-y_i)\frac{1}{1-g(\theta^Tx_i)})g(\theta^Tx_i)(1-g(\theta^Tx_i))\frac{\delta}{\delta_{\theta_j}}\theta^Tx_i$$

$$=-\frac{1}{m}\sum_{i=1}^{m}(y_i(1-g(\theta^Tx_i))-(1-y_i)g(\theta^Tx_i))x_i^j$$

$$=-\frac{1}{m}(y_i-g(\theta^Tx_i))x_i^j$$

- 参数更新：

$$\theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)x_i^j$$

- 多分类的 softmax:

$$h_\theta(x^{(i)})=\begin{bmatrix}p(y^{(i)}=1|x^{(i)};\theta)\\p(y^{(i)}=2|x^{(i)};\theta)\\\vdots\\p(y^{(i)}=k|x^{(i)};\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{k}e^{\theta_j^Tx^{(i)}}}\begin{bmatrix}e^{\theta_1^Tx^{(i)}}\\e^{\theta_2^Tx(i)}\\\vdots\\e^{\theta_k^Tx^{(i)}}\end{bmatrix}$$

第四章 聚类算法

- **聚类概念**：
  - 无监督问题
  - 聚类：相似的东西分到一组
  - 难点：如何评估，如何调参

4.1 K-MEANS

- **K-MEANS** 算法基本概念：
  - 要得到簇的个数，需要指定**K**值
  - 质心：均值，即向量各维度取平均即可
  - 距离的度量：常用**欧氏距离**和**余弦相似度**（先标准化）
  - 优化目标:

$$min\sum_{i=1}^{K}\sum_{x\in C_i}dist(c_i,x)^2$$

- **优势**：
  - 简单，快速， 适合常规数据集

- **劣势**：
  - K值难确定
  - 复杂度与样本呈线性关系
  - 很难发现任意形状的簇

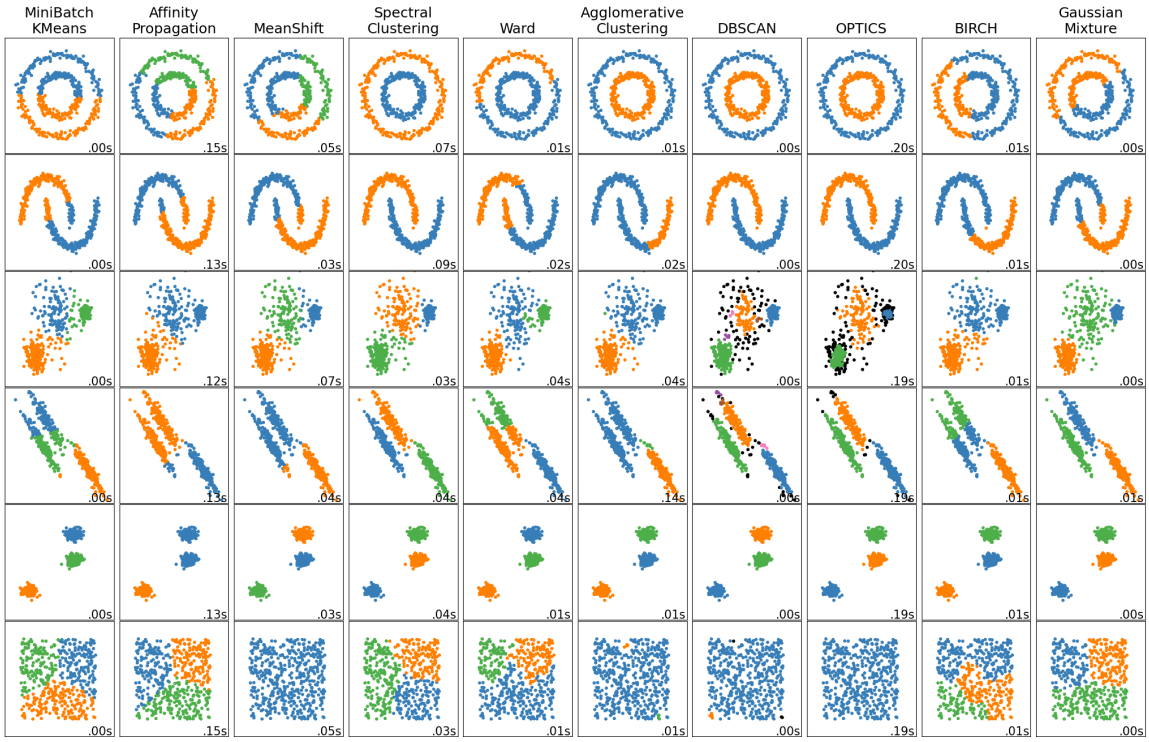
4.2 DBSCAN

- 基本概念：（Density-Based Spatial Clustering of Applications with Noise）
  - 核心对象：若某个点的密度达到算法设定的阈值则其为核心点。（即 r 邻域内点的数量不小于 minPts）
  - ε -邻域的距离阈值：设定的半径 r
  - 直接密度可达：若某点 p 在点 q 的 r 邻域内，且 q 是核心点则 p-q 直接密度可达
  - 密度可达：若有一个点的序列 $q_0, q_1, \cdots, q_k$,对任意 $q_i - q_{i-1}$ 是直接密度可达的，则称从 q_0 到 q_k 密度可达，这实际上是直接密度可达的“传播”
- 工作流程：
  - 参数D：输入数据集
  - 参数ϵ ： 指定半径
  - MinPts：密度阈值

> 标记所有对象为 unvisited
>
> Do
>
> 随机选择一个 unvisited 对象 p
>
> 标记 p 为 visited
>
> if p 的 ϵ -领域至少有 MinPts 个对象
>
> > 创建一个新簇 C，并把 p 添加到 C
> >
> > 令 N 为 p 的 ϵ -领域中的对象集合
> >
> > For N 中每个点 p'
> >
> > > If p' 是 unvisited
> > >
> > > > 标记 p' 为 visited
> > > >
> > > > If p' 的 ϵ -领域至少有 MinPts 个对象，把这些对象添加到 N
> > > >
> > > > If p'还不是任何簇的成员，把p' 添加到C
> > >
> > > End for
> > >
> > > 输出C
> >
> > Else 标记 p 为噪声
> >
> > Until 没有标记为 unvisited 的对象

- 参数选择：
  - 半径 ϵ ，可以根据 K 距离来设定：找突变点
  - K 距离：给定数据集 $P = \{p(i); i = 0, 1, \cdots, n\}$ ，计算点 $P(i)$ 到集合 D 的子集 S 中所有点之间的距离，距离按照从小到大的顺序排序，$d(k)$ 就被称为 k-距离。
  - MinPts：k-距离中 k 的值，一般取得小一些，多次尝试

- 优势：
    - 不需要指定簇个数
    - 可以发现任意形状的簇
    - 擅长找到离群点（检测任务）
    - 两个参数就够了
- 劣势：
    - 高维度数据需要做降维处理
    - 参数难以选择（对结果影响非常大）
    - 需要数据削减，提高效率



第五章 决策树

- 从根节点开始，一步步走到叶子节点（决策）
- 有监督学习
- 所有的数据最终都会落到叶子节点，既可以做分类也可以做回归。
- 树的组成：
    - 根节点：第一个选择点
    - 非叶子节点与分支：中间过程
    - 叶子节点：最终的决策结果
- 决策树的训练与测试
    - 训练阶段：从给定的训练集构造出一个树（从根节点开始选择特征，如何进行**特征切分**
    - 测试阶段：根据构造出来的树模型从上到下去走一遍
- 如何选择节点（特征切分）：通过一种衡量标准，来计算通过不同特征进行分支选择后的分类情况，找到最好的那个，作为节点。
- 衡量标准：熵
    - 表示随机变量不确定性的度量

- $H(x) = -\sum p(i) \cdot logp(i), i = 1, 2, \cdots, n$
- 不确定性越大，得到的熵值越大
- 信息增益：表示特征X使得类Y的不确定性减少的程度。（分类后的专一性，希望分类后的结果是同类在一起）
- GINI系数：$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2$

- 决策树剪枝策略
  - 决策树过拟合风险很大，理论上可以完全分得开数据
  - 预剪枝：边建立决策树边进行剪枝（更实用），限制深度，叶子节点个数，叶子节点样本数，信息增益量等
  - 后剪枝：当建立完决策树后来进行剪枝，通过一定的衡量标准$C_\alpha(T) = C(T) + \alpha \cdot |T_{leaf}|$（叶子节点越多，损失越大）

第六章 集成算法

- Ensemble Leaning
  - 目的：让学习效果更好
  - **Bagging**: 训练多个分类器取平均 $f(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$
  - **Boosting**: 从弱学习器开始加强，通过加权来进行训练

    $F_m(x) = F_{m-1}(x) + argmin_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i)) + h(x_i)$ (加入一颗树，要比原来强)
  - **Stacking**: 聚合多个分类或回归模型（可以分阶段来做）
- Bagging 模型
  - 全称：booststrap aggregation(并行训练一堆分类器)
  - 最典型代表：随机森林模型
  - 随机：**数据采样随机， 特征选择随机**
  - 森林：很多个决策树并行放在一起
  - 之所以进行随机，是要保证泛化能力，如果树都一样，就没有意义了
  - 随机森林优势
    - 能够处理很高维度（feature 很多）的数据，并且不用做特征选择
    - 在训练完成后，能够给出哪些 feature 比较重要（特征重要性）
    - 容易做成并行化方法，速度快
    - 可以进行可视化展示，可解释性强，便于分析
    - 理论上越多的树效果会越好，但实际上基本超过一定数量就差不多上下浮动了
- Boosting 提升算法模型
  - 典型代表：AdaBoost, Xgboost
  - Adaboost 会根据前一次的分类效果调整数据权重
  - 解释：如果某一个数据在这次分错了，那在下一次就会给他更大的权重
  - 最终结果：每个分类器根据自身的准确性来确定各自的权重，再合体
- Stacking 堆叠模型
  - 堆叠：暴力，一堆分类器
  - 可以堆叠各种分类器（KNN，SVM，RF等）
  - 分阶段：第一阶段得出各自结果，第二阶段再用前一阶段结果训练
  - 为了刷好的结果不择手段