Week 14 笔记

1. 分类算法

最近邻分类算法（Nearest Neighbor，简称 NN）
用欧氏距离：

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

直接比较与不同类的 each point 的距离，有 k 个最小距离 point 的那一个类就是新样本点的类（监督学习）
能够针对类内间距较小的数据集获得很好的效果，但是它很难应用到各类数据有重合的情况

```
def compute_distance(point_1, point_2):
    """ 计算两个点之间的距离"""
    if point_1.shape != point_2.shape:
        raise ValueError("shape of image must be equal to reference")
    point_1 = point_1.astype(np.float32)
    point_2 = point_2.astype(np.float32)
    distance = np.mean(np.square(point_1-point_2))
    return distance
```

朴素贝叶斯：

$$\mathrm{p}(\text{类别}|\text{特征}) = \frac{p(\text{特征}|\text{类别})\mathrm{p}(\text{类别})}{\mathrm{p}(\text{特征})}$$

用联合概率把分类 A 和分类 B 的 subject to 特征的条件概率分别求出，然后其中概率最大的就是我们 naive beyas 会 pick 的分类标签

支持向量

1. 线性可分：在二维空间上，两类点被一条直线完全分开叫做线性可分.
2. 扩展到更高维度，从二维扩展到多维空间中时，将两类点分隔开的超平面
3. 样本中距离超平面最近的一些点，这些点叫做支持向量。
4. 在二维空间上，分割点的直线为：

$$w^T x + b = 0$$

距离被定义为：

$$\frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$

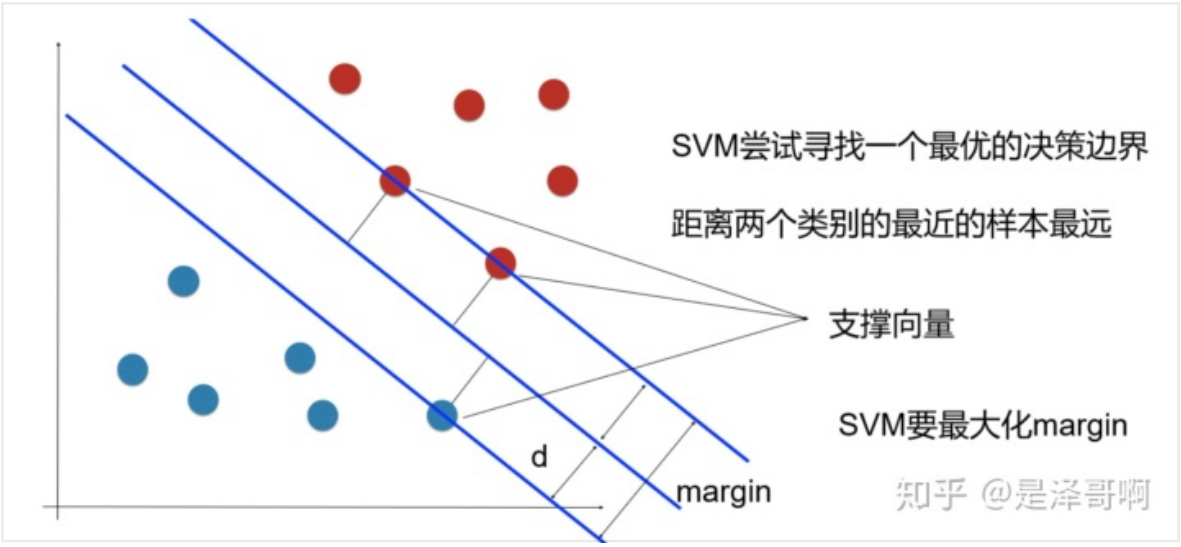
在更高维度，距离可以写作：

$$\frac{|w^T x + b|}{||w||}$$

其中 w 为：

$$\text{其中 } ||w|| = \sqrt{w_1^2 + \ldots w_n^2} \text{ 。}$$

如图所示，根据支持向量的定义我们知道，支持向量到超平面的距离为 d，其他点到超平面的距离大于 d：



所以直线的分类器功能可以写作：

$$\begin{cases} \dfrac{w^T x + b}{||w||d} \geq 1 \quad y = 1 \\ \dfrac{w^T x + b}{||w||d} \leq -1 \quad y = -1 \end{cases}$$

然后可以简化方程为：

$$\begin{cases} w^T x + b \geq 1 \quad y = 1 \\ w^T x + b \leq -1 \quad y = -1 \end{cases}$$

将两个方程合并，我们可以简写为：

$$y(w^T x + b) \geq 1$$

所以写成优化形式，目标最大化支持向量的大小，就有：

每个支持向量到超平面的距离可以写为：

$$d = \frac{|w^T x + b|}{||w||}$$

由上述 $y(w^T x + b) > 1 > 0$ 可以得到 $y(w^T x + b) = |w^T x + b|$ ，所以我们得到：

$$d = \frac{y(w^T x + b)}{||w||}$$

最大化这个距离：

$$\max 2 * \frac{y(w^T x + b)}{||w||}$$

对这个式子取倒数，就有：

$$\min \frac{1}{2}||w||^2 \ s.t. \quad y_i \ (w^T x_i + b) \ \geq 1$$

后续是大量的数学证明（之后系统学习的时候记得补拉格朗日乘数法）……

优点

- 有严格的数学理论支持，可解释性强，不依靠统计方法，从而简化了通常的分类和回归问题；
- 能找出对任务至关重要的关键样本（即：支持向量）；
- 采用核技巧之后，可以处理非线性分类/回归任务；
- 最终决策函数只由少数的支持向量所确定，计算的复杂性取决于支持向量的数目，而不是样本空间的维数，这在某种意义上避免了“维数灾难”。

6.2 缺点

- 训练时间长。当采用 SMO 算法时，由于每次都需要挑选一对参数，因此时间复杂度较大
- ，其中 N 为训练样本的数量；
- 当采用核技巧时，如果需要存储核矩阵，则空间复杂度较大
- 模型预测时，预测时间与支持向量的个数成正比。当支持向量的数量较大时，预测计算复杂度较高。

因此支持向量机目前只适合小批量样本的任务，无法适应百万甚至上亿样本的任务。

（ref：【机器学习】支持向量机 SVM（非常详细） - 阿泽的文章 - 知乎

https://zhuanlan.zhihu.com/p/77750026）

决策树：
监督学习，决策树是一种树形结构，其中每个内部节点表示一个属性上的判断，每个分支代表一个判断结果的输出，最后每个叶节点代表一种分类结果。

1. 节点的分裂：一般当一个节点所代表的属性无法给出判断时，则选择将这一节点分成 2 个
子节点（如不是二叉树的情况会分成 n 个子节点）

2. 阈值的确定：选择适当的阈值使得分类错误率最小 （Training Error）。
ID3: 由增熵（Entropy）原理来决定那个做父节点，那个节点需要分裂。对于一组数据，熵越小说明分类结果越好。熵定义如下：

$$Entropy = - sum [p(x\_i) * log2(P(x\_i)]$$

其中 p(x_i) 为 x_i 出现的概率。假如是 2 分类问题，当 A 类和 B 类各占 50% 的时候，即我们的分类效果最差的时候，有 max 熵

$$Entropy = - (0.5*log\_2(0.5)+0.5*log\_2(0.5))= 1$$

当只有 A 类，或只有 B 类的时候，即确定性，有 min 熵

$$Entropy= - (1*log\_2(1) +0) =0$$

直观来说就是选选择信息增益最大的特征进行分裂。

2. 聚类算法（Clustering，非监督学习）
K-means 是我们最常用的基于欧式距离的聚类算法，其认为两个目标的距离越近，相似度越大。

选择初始化的 k 个样本作为初始聚类中心 a=a1, a2, a3...
For each xi, 分配到 k 个聚类中心中，分配规则为分配到距离其最近的聚类中心所对应的类中
在完成分配每个 xi 后，针对每个类别 ai，重新计算类的聚类中心

伪代码：
获取数据 n 个 m 维的数据
随机生成 K 个 m 维的点

```
while(t)
    for(int i=0;i < n;i++)
        for(int j=0;j < k;j++)
            计算点 i 到类 j 的距离
    for(int i=0;i < k;i++)
        1. 找出所有属于自己这一类的所有数据点
        2. 把自己的坐标修改为这些数据点的中心点坐标
end
```

2.1 优点
容易理解，聚类效果不错，虽然是局部最优， 但往往局部最优就够了；
处理大数据集的时候，该算法可以保证较好的伸缩性；
当簇近似高斯分布的时候，效果非常不错；
算法复杂度低。

2.2 缺点
K 值需要人为设定，不同 K 值得到的结果不一样；
对初始的簇中心敏感，不同选取方式会得到不同结果；
对异常值敏感；
样本只能归为一类，不适合多分类任务；
不适合太离散的分类、样本类别不平衡的分类、非凸形状的分类。
（cite：【机器学习】K-means（非常详细） - 阿泽的文章 - 知乎
https://zhuanlan.zhihu.com/p/78798251）

层次聚类：
因为 k-mean 需要预先决定簇数 k，可能和现实情况不准确，所以提出不需要 k 的层次聚类；
Step:
有两种类型的层次聚类：凝聚层次聚类和分类层次聚类
凝聚层次聚类是从所有数据点都是单独的数据点开始，然后通过相似性不断组合，直到最后只有一个簇为止
分裂层次聚类正好反过来，它是从单个集群开始逐步分裂，直到无法分裂，即每个点都是一个簇

如何判定数据点之间的相似性：
计算这些簇的质心之间的距离（欧几里得距离）。距离最小的点称为相似点，我们可以合并它们，也可以将其称为**基于距离的算法**。
Example：

Student_ID	Marks
1	10
2	7
3	28
4	20
5	35

首先创建邻近矩阵（邻近矩阵是针对于簇的，而非针对于数据点）：

ID	1	2	3	4	5
1	0	3	18	10	25
2	3	0	21	13	28
3	18	21	0	8	7
4	10	13	8	0	15
5	25	28	7	15	0

Note：对角元素为 0，自身距离为 0；点与点之间的距离为欧几里得距离（可以发现矩阵完全对称，因为两点之间的距离为固定值）

凝聚层次距离：

1. 将单个数据点作为簇，即有 5 个簇；
2. 查找矩阵中距离最小的值所对应的点，合并两点成为新的簇，新簇的 value 可以用最大值/最小值/平均值来代替，基于新的簇 value table 更新邻近矩阵；

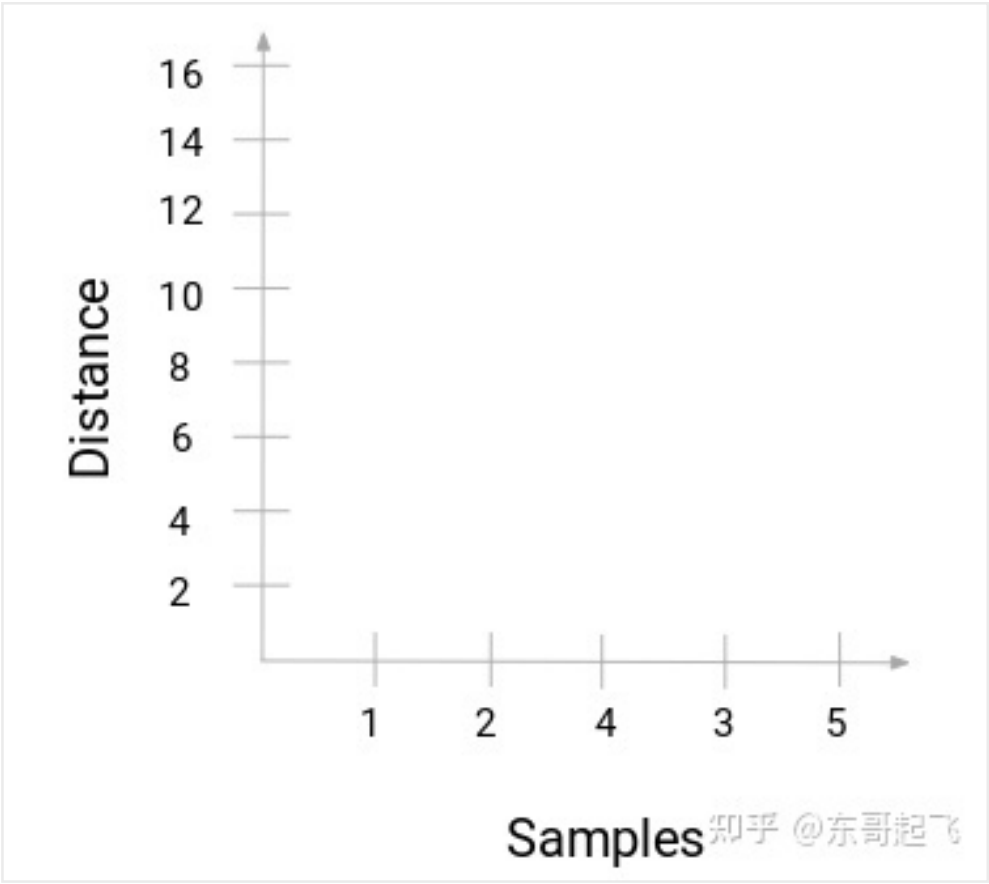
Student_ID	Marks
(1,2)	10
3	28
4	20
5	35

ID	(1,2)	3	4	5
(1,2)	0	18	10	25
3	18	0	8	7
4	10	8	0	15
5	25	7	15	0

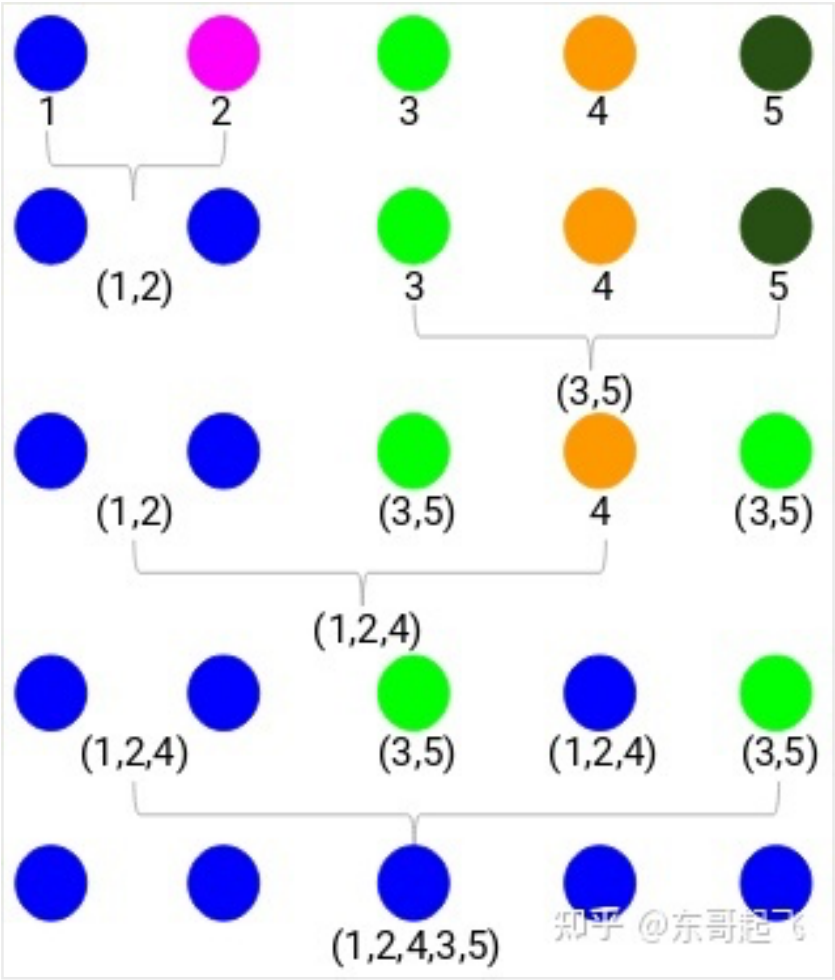
3. 重复 until 所有 element 都在一个簇中

如何选择聚类数：建 树

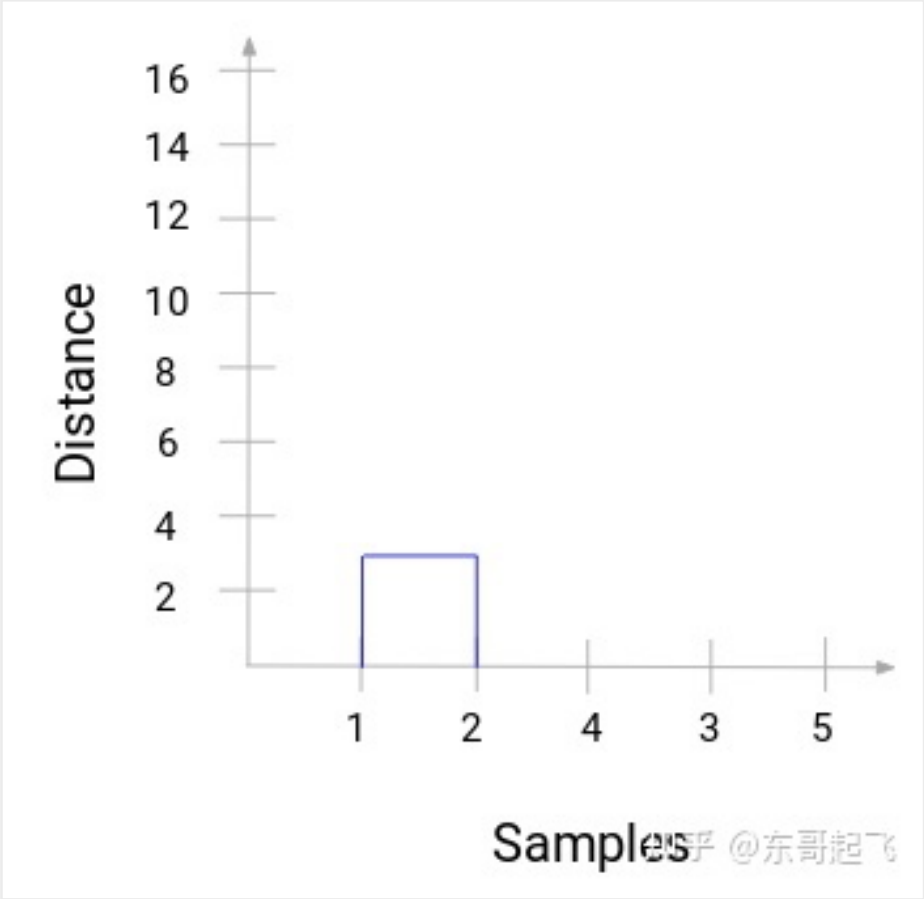
首先我们构建一个空的表格，包含所有的样本，横坐标是样本编号，纵坐标是样本点之间的距离：



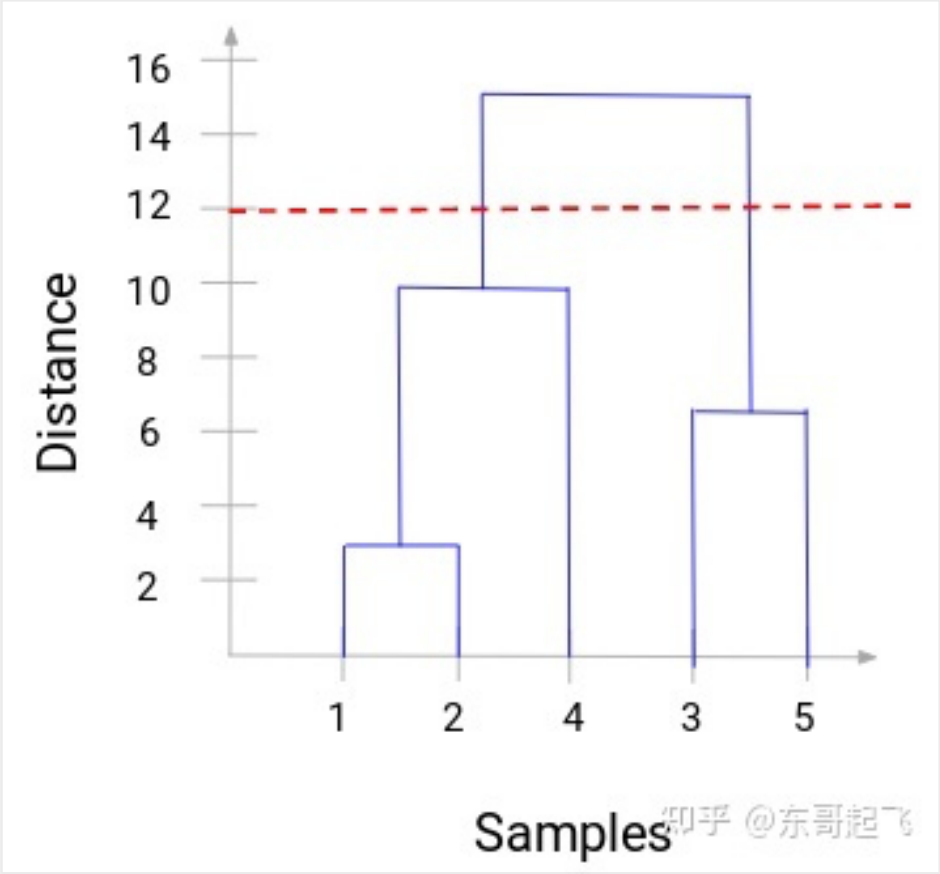
在上图中重新绘制我们聚类过程如下：



第一步，将点1和点2合并，其中他们的距离为3，所以高为3



第二步，合并工程中的始终用真实值来表示（因为更小的已经被 eliminated 了，所以不用担心后续合并过程中新的边短于已经合并了的边）
然后我们设置一个 distance 阈值来判定我们需要的簇：
阈值距离，绘制一条水平线。比如我们将阈值设置为 12，并绘制一条水平线，



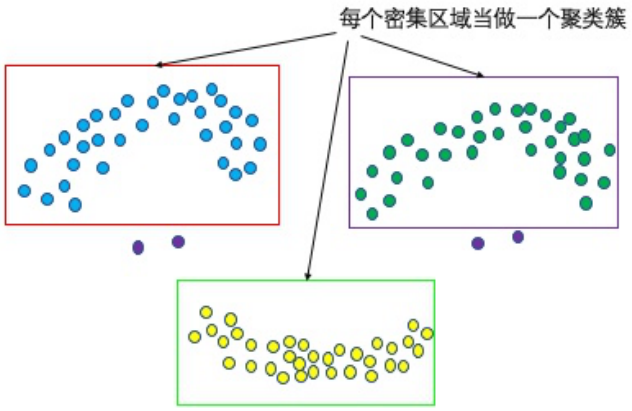
从交点中可以看到，聚类的数量就是与阈值水平线与垂直线相交的数量（红线与 2 条垂直线相交，我们将有 2 个簇）。与横坐标相对应的，一个簇将有一个样本集合为 (1,2,4)，另一个集群将有一个样本集合 (3,5)。
（ref：https://zhuanlan.zhihu.com/p/435987610）

DBSCAN：一种基于密度，对噪声鲁棒的空间聚类算法
Density-Based Spatial Clustering of Applications with Noise
DBSCAN算法可以找到样本点的全部密集区域，并把这些密集区域当做一个一个的聚类簇。

优点：
基于密度，对远离密度核心的噪声点鲁棒
无需知道聚类簇的数量
可以发现任意形状的聚类簇

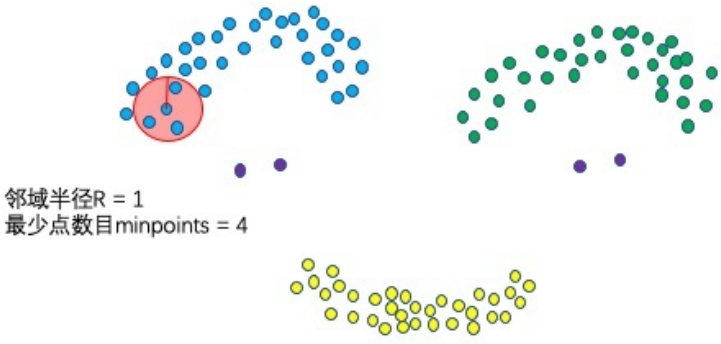
缺点
需要**为算法指定eps和MinPts参数**，这对分析人员是一个很大的挑战；
DBSCAN聚类算法对**参数eps（即半径r）和MinPts的设置是非常敏感**的，如果指定不当，该算法将造成聚类质量的下降。

1个核心思想：基于密度。**直观效果上看，DBSCAN算法可以找到样本点的全部密集区域，并把这些密集区域当做一个一个的聚类簇。**



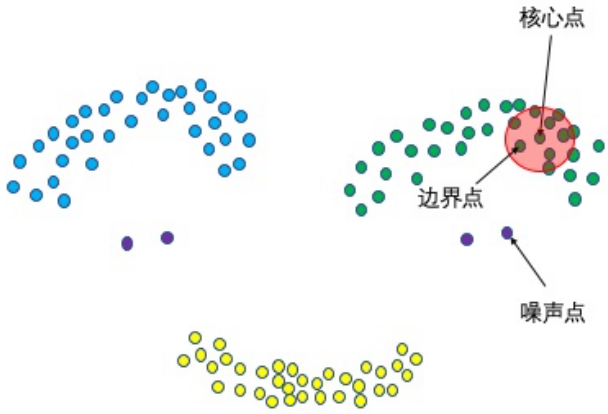
DBSCAN的1个核心思想

2个算法参数：邻域半径R和最少点数目MinPoints。这两个算法参数实际可以刻画什么叫密集：当邻域半径R内的点的个数大于最少点数目MinPoints时，就是密集。



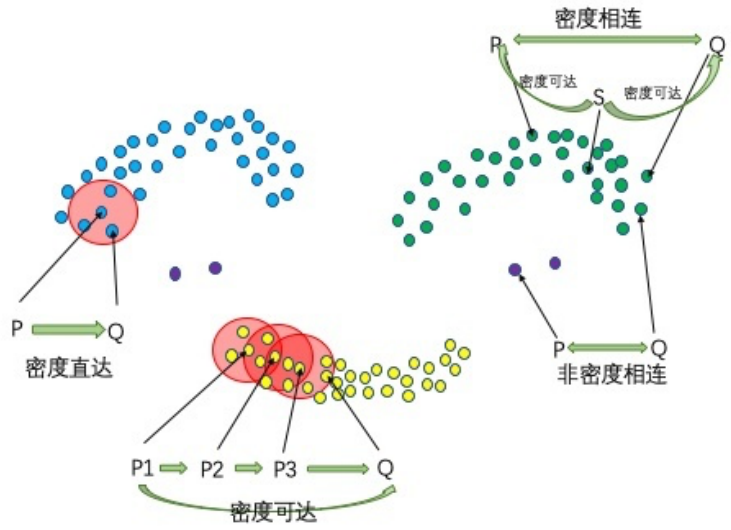
DBSCAN中2个算法参数

3种点的类别：核心点，边界点和噪声点。邻域半径R内样本点的数量大于等于minpoints的点叫做核心点。不属于核心点但在某个核心点的邻域内的点叫做边界点。既不是核心点也不是边界点的是噪声点。



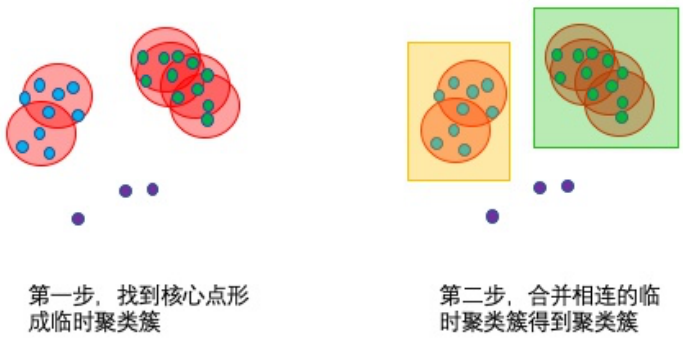
DBSCAN中3种点的类别

4种点的关系：密度直达，密度可达，密度相连，非密度相连。
如果P为核心点，Q在P的R邻域内，那么称P到Q密度直达。任何核心点到其自身密度直达，密度直达不具有对称性，如果P到Q密度可达，那么Q到P不一定密度可达。
如果存在核心点P2，P3，……，Pn，且P1到P2密度直达，P2到P3密度直达，……，P(n-1)到Pn密度直达，Pn到Q密度直达，则P1到Q密度可达。密度可达也不具有对称性。
如果存在核心点S，使得S到P和Q都密度可达，则P和Q密度相连。密度相连具有对称性，如果P和Q密度相连，那么Q和P也一定密度相连。密度相连的两个点属于同一个聚类簇。
如果两个点不属于密度相连关系，则两个点非密度相连。非密度相连的两个点属于不同的聚类簇，或者其中存在噪声点。



DBSCAN中4种点的关系

1，寻找核心点形成临时聚类簇。
扫描全部样本点，如果某个样本点R半径范围内点数目>=MinPoints，则将其纳入核心点列表，并将其密度直达的点形成对应的临时聚类簇。
2，合并临时聚类簇得到聚类簇。
对于每一个临时聚类簇，检查其中的点是否为核心点，如果是，将该点对应的临时聚类簇和当前临时聚类簇合并，得到新的临时聚类簇。
重复此操作，直到当前临时聚类簇中的每一个点要么不在核心点列表，要么其密度直达的点都已经在该临时聚类簇，该临时聚类簇升级成为聚类簇。
继续对剩余的临时聚类簇进行相同的合并操作，直到全部临时聚类簇被处理。



第一步，找到核心点形成临时聚类簇

第二步，合并相连的临时聚类簇得到聚类簇

DBSCAN的算法实现步骤

3. 深度学习：

卷积神经网络

核心是用卷积层提取**特征（feature）**，用池化层降低复杂度，然后用FP和BP实现优化

卷积层（convolution filter）：用feature 去卷积数据，得到**feature map （特征图）**：
feature map是每一个feature从原始图像中提取出来的“特征”。其中的值，越接近为**1**表示对应位置和feature的**匹配越完整**，越是接近-1，表示对应位置和feature的反面匹配越完整，而值接近**0**的表示对应位置没有任何匹配或者说**没有什么关联**。

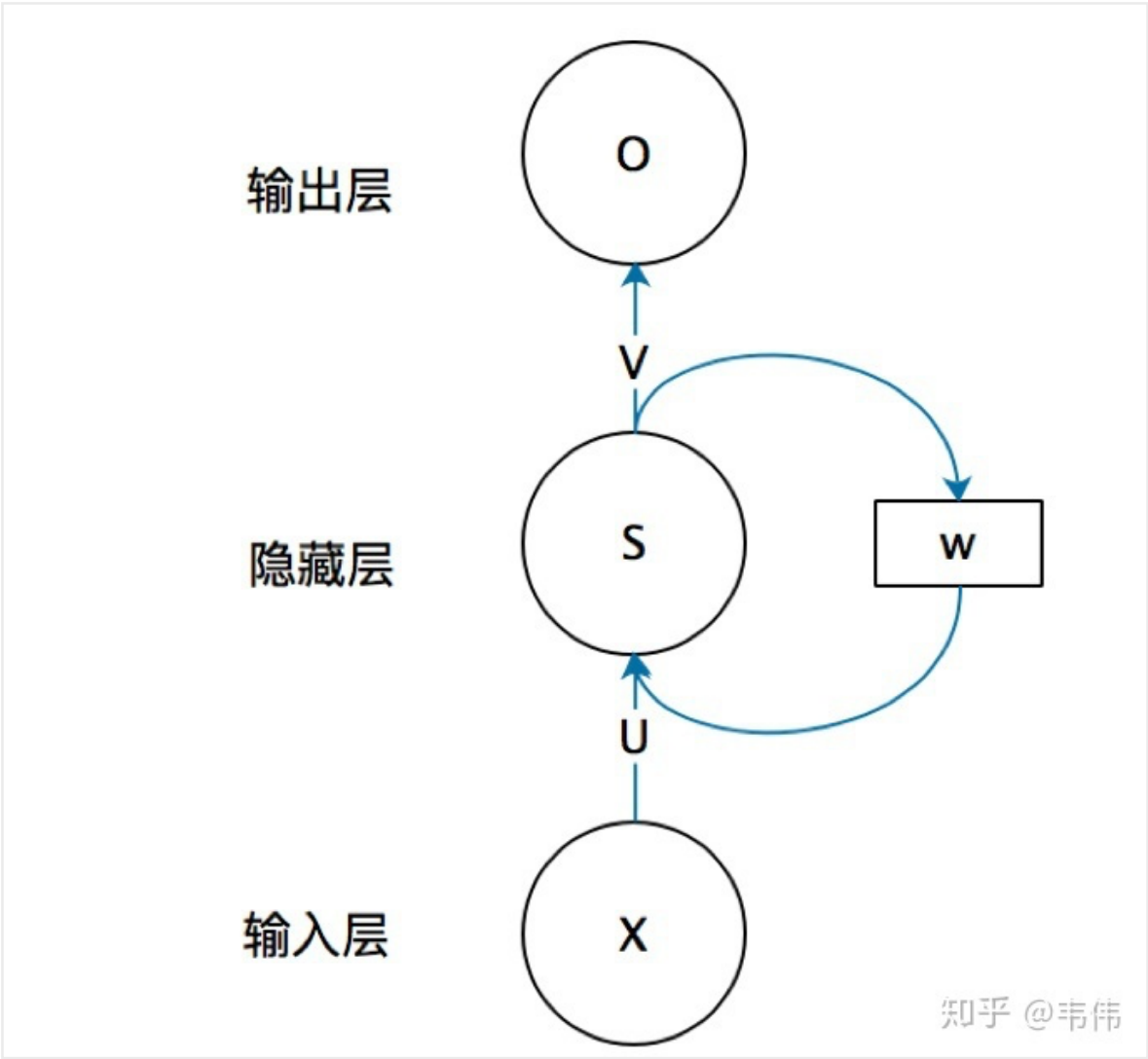
Activation：就是类似sigmoid的函数，将线性模型进行一个映射（线性+线性还是线性，需要加入一个非线性的activation来做非线性特征）

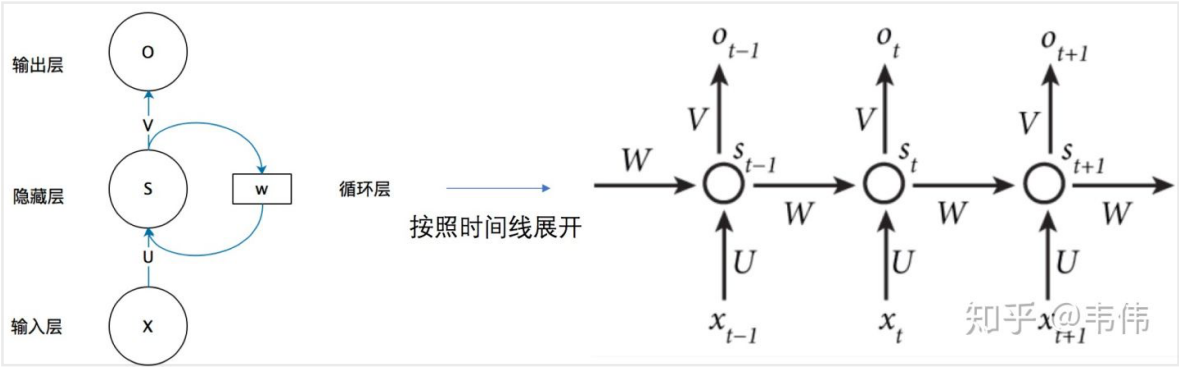
Pooling池化：
池化分为两种，Max Pooling 最大池化、Average Pooling平均池化。顾名思义，最大池化就是取最大值，平均池化就是取平均值。
因为最大池化保留了每一个小块内的最大值，所以它相当于保留了这一块最佳匹配结果（因为值越接近1表示匹配越好）。这也就意味着它不会具体关注窗口内到底是哪一个地方匹配了，而只关注是不是有某个地方匹配上了。这也就能够看出，CNN能够发现图像中是否具有某种特征，而不用在意到底在哪里具有这种特征。这也就能够帮助解决之前提到的计算机逐一像素匹配的死板做法。

全联接层：
全连接层，**因为空间结构特性被忽略了，所以全连接层不适合用于在方位上找Pattern的任务，比如segmentation （直接展开成vector）**

卷积层采用的是“**局部连接**”的思想，卷积层的操作是用一个3X3的图与原图进行连接操作，原图中只有一个3X3的窗口能够与它连接起来。
卷积窗口（conv filter）通过窗口滑动起来的方法后续进行连接。这个方法的思想就是**“参数共享”，**参数指的就是filter，用滑动窗口的方式，将这个filter值共享给原图中的每一块区域连接进行卷积运算。

循环神经网络（Rerrent Neural Network, RNN）
RNN对具有序列特性的数据非常有效，它能挖掘数据中的时序信息以及语义信息，利用了RNN的这种能力，使深度学习模型在解决语音识别、语言模型、机器翻译以及时序分析等NLP领域的问题时有所突破。RNN的特性，可以处理序列数据，同时对序列敏感。
序列特性：符合时间顺序，逻辑顺序，或者其他顺序就叫序列特性
全连接神经网络没有结合上下文去训练模型，而是单独的在训练apple这个单词的label，所以需要循环神经网络。

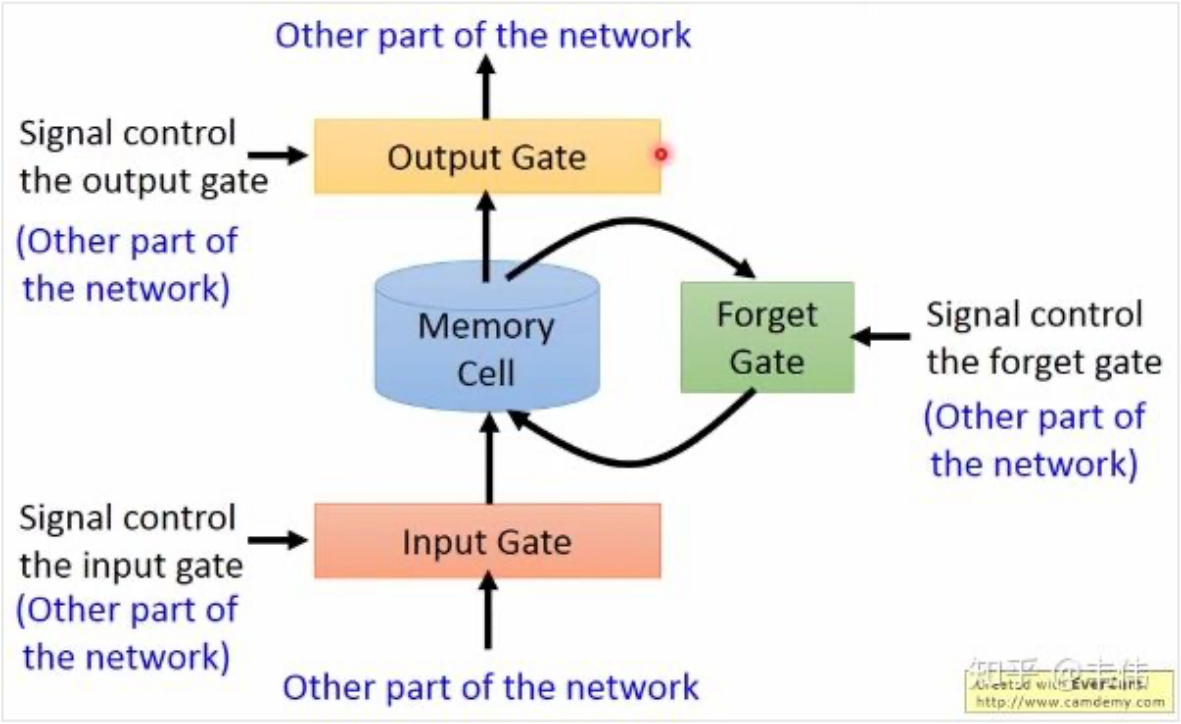




For example:
I love you
x_t-1代表的就是I这个单词的向量， x代表的是love这个单词的向量，x_t+1代表的是you这个单词的向量， W不变是每个时间点之间的权重矩阵，我们注意到，RNN之所以可以解决序列问题，是因为它可以记住每一时刻的信息，每一时刻的隐藏层不仅由该时刻的输入层决定，还由上一时刻的隐藏层决定

LSTM（Long short-term memory）：长短期记忆
RNN缺陷：一个句子很长，到句子末尾时，它将记不住这个句子的开头的内容详细内容
LSTM通过它的“门控装置”有效的缓解了这个问题， 所以LSTM而非普通RNN

LSTM区别于classic RNN，它会通过门控装置选择性的存储信息



Input Gate：中文是输入门，在每一时刻从输入层输入的信息会首先经过输入门，输入门的开关会决定这一时刻是否会有信息输入到Memory Cell。

Output Gate：中文是输出门，每一时刻是否有信息从Memory Cell输出取决于这一道门。

Forget Gate：中文是遗忘门，每一时刻Memory Cell里的值都会经历一个是否被遗忘的过程，就是由该门控制的，如果打开，那么将会把Memory Cell里的值清除（用空值替代），也就是遗忘掉。

MNIST 手写数字识别实验报告：

机器学习算法：
Logistic回归（一种经典的分类算法）
通过OneVsAll 来将10个label拆分成10个输出（10个01变量来表示10个不同的label）
然后我们需要自行定义和计算我们的Loss function和梯度，然后用求解器来基于梯度和Loss Function完成optimization。
其中我们定义：

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

作为我们的Loss Function，其中第一项是我们的Loss Function，第二项是regularized项，theta越多（即维度越高）则其在loss function里的值会越大，可以确保我们的模型不会过拟合。第一项是sigmoid函数的COST function，当远离我们的预测目标值时候其会趋于无限大；

Cost function的经典定义为：

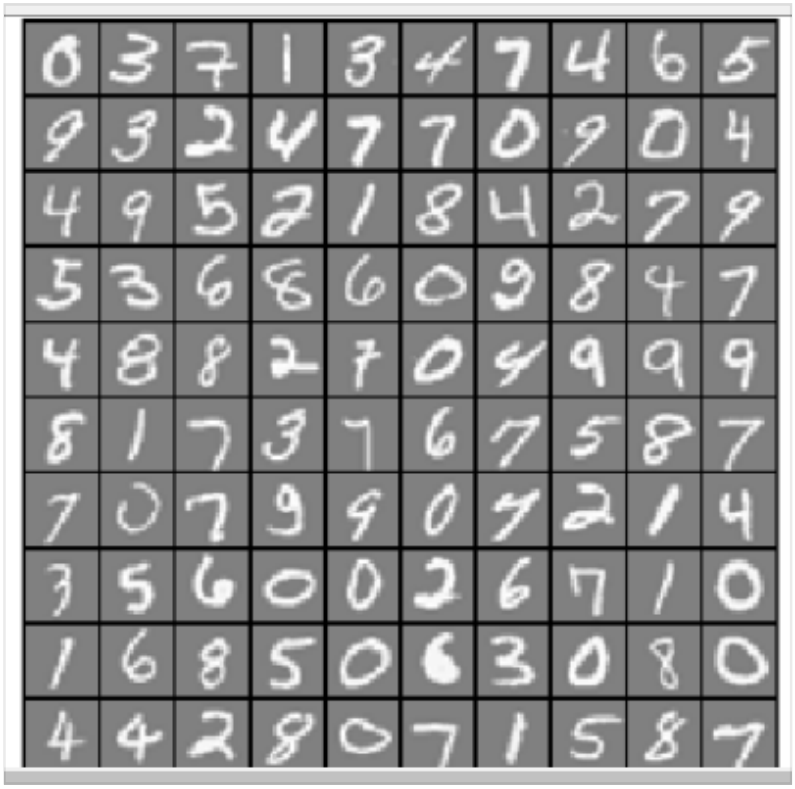
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

梯度为：

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \qquad \text{for } j = 0,$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \qquad \text{for } j \geq 1$$

上式的特别之处在于regularized，对于第0项，他不应该被纳入到regularized项中（因为是常数，始终存在）



准确率94.9%

```
pred = predictOneVsAll(all_theta, X);
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

```
Training Set Accuracy: 94.900000
```

卷积神经网络：

Layer	Remark	Activation Function
Input	28×28 nodes	-
Convolution	20 convolution filters (9×9)	ReLU
Pooling	1 mean pooling (2×2)	-
Hidden	100 nodes	ReLU
Output	10 nodes	Softmax

只有3个epoch的时候，准确率93.5%，欠拟合，考虑增加训练次数；

```
Accuracy is 0.935000
```

有50个epoch的时候，准确率上升到98%，所以神经网络需要的训练量远大于相同任务的ML解决方案，cost也会大为上升；

```
epoch =

    50

Accuracy is 0.979500
```

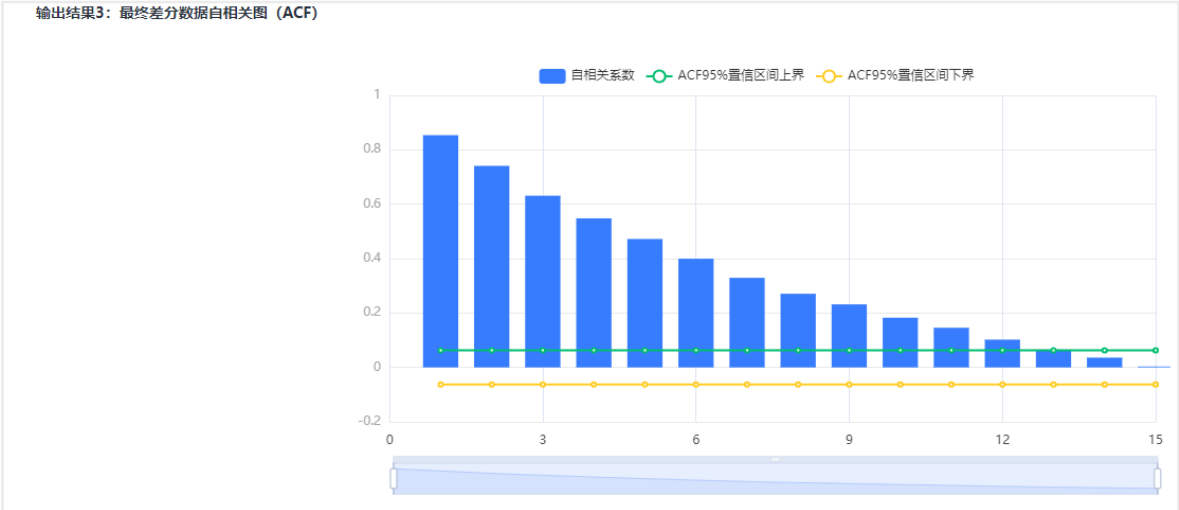
PM 2.5 （BP神经网络不太行，没有试别的了，应该LSTM会比较好）

参数名　参数值
训练用时　　0.097s
数据切分　　0.7
数据洗牌　　是
交叉验证　　否
激活函数　　identity
求解器　lbfgs
学习率　0.1
L2正则项　　1
迭代次数　　1000
隐藏第1层神经元数量　　100

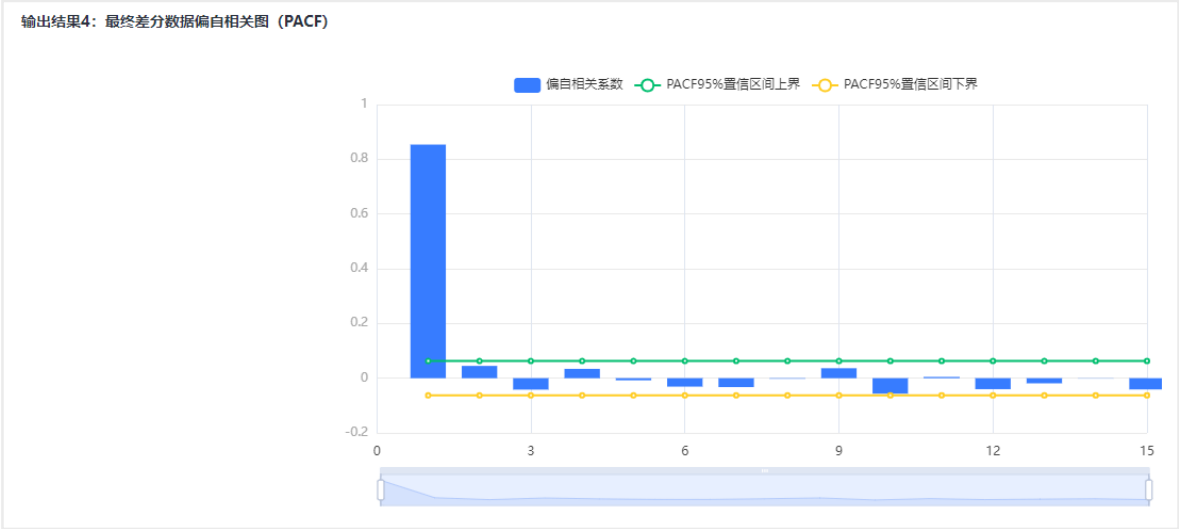
MSE RMSE　　MAE　　MAPE　　R^2
训练集　1681.646 41.008　　30.938　　49.195　　0.373

测试集　1552.759 39.405　29.495　53.101　0.436

时间序列分析是MA模型（ 偏自相关（PACF）图在p阶进行截尾，自相关（ACF）图拖尾，ARMA模型可简化为AR（p）模型；）：

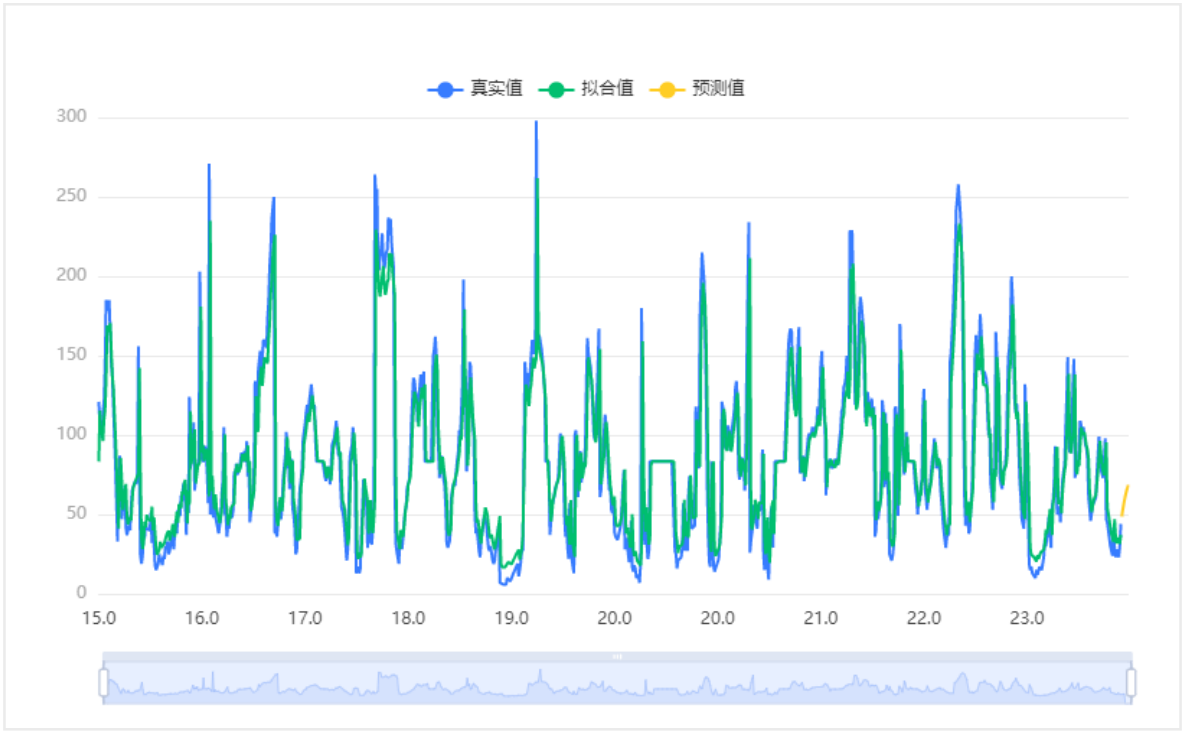


最终差分数据自相关图（ACF）



基于AIC信息准则自动寻找最优参数，模型结果为ARIMA模型（2,0,0）检验表，基于字段：pm2.5，从Q统计量结果分析可以得到：Q6在水平上不呈现显著性，不能拒绝模型的残差为白噪声序列的假设，同时模型的拟合优度R2为0.73，模型表现较为良好，模型基本满足要求。
模型参数表
系数　　标准差　t　　p>|t|　　0.025　　0.975
常数　　83.3 6.136　　13.575　0　　71.273　95.326
ar.L1.pm2.5　0.815　　0.032　　25.58　　0　　0.752　　0.877
ar.L2.pm2.5　0.046　　0.032　　1.428　　0.153　　-0.017　0.108
注：***、**、*分别代表1%、5%、10%的显著性水平

T检验不合适，可能需要想点别的办法


真实值
拟合值
预测值
300
250
200
150
100
50
0
15.0
16.0
17.0
18.0
19.0
20.0
20.0
21.0
22.0
23.0