实验报告

Kmeans 算法

Kmeans 简介：Kmeans 算法为一般用于在无监督学习（Unsupervised Learning）下的聚类算法，其主要原理是计算每一个点划分至其中心点距离（可以是欧几里得距离、也可以是切比雪夫距离或者是曼哈顿距离）最短的簇，直到满足一定的终止条件为止（如中心点不再发生改变，SSE 或者 SC 系数达到极小值）

二分 Kmeans 简介：改进型的 Kmeans 算法，同样在开始指定 K 值，不同的是，先从 K = 1 开始进行聚类处理，每次挑选 SSE 值最大的一个聚类（SSE 值最大则说明聚类内部还有聚类的可能性更大）进行二分操作，直到 K 值达到所指定的值

优点在于：

- 算法简洁明了，简单易写；
- 适用于无监督学习，在分类未明的时候可以很好进行划分；
- 适用于高维数据，即复数个特征的数据；

缺点在于：

- 最终分类数 K 不明确，经常要手动介入去设置
- 初值设置不当容易陷入局部最优解
- 处理结果受初值影响较大，初值不一样，其生成结果经常也不一样

算法原理（伪代码）

Kmeans 算法

```
KMEANS(DATA， K)

CLUSTER = INIT(K)                                    //初始化中心点

WHILE NOT     Ending(K)                                  //不符合终止条件

    CLASS = CLASSIFY(DATA, CLUSTER)//为每一个数据进行分类

    UPDATE(CLUSTER, CLASS, DATA)                 //更新中心点的位置

RETURN CLUSTER                                       //返回中心点集合
```

二分 Kmeans 算法

```
BIKMEANS(DATA， K)
CLUSTER ← INIT(1)                          //初始化中心点
COUNT = 1                                  //设定 COUNT 从 1 开始
WHILE COUNT NOT EQUAL K                    //不符合终止条件
    CLASS = CLASSIFY(DATA, CLUSTER)        //为每一个数据进行分类
    INDEX = MAXSSE(DATA, CLASS, CLUSTER)
    //返回最大 SSE 的簇的索引
    NEWCOLLECT = KMEANS(DATA[INDEX], 2)
    //将该簇的数据进行二分
    REPLACE(CLUSTER, INDEX, NEWCOLLECT)  //更新中心点
    CLASS = CLASSIFY(DATA, CLUSTER)          //更新分类
    UPDATE(CLUSTER, CLASS, DATA)                 //更新中心点的位
置
RETURN CLUSTER                             //返回中心点集合
```

代码具体实现：详见 Kmeans 中的代码

API：

包 Kmeans

类 Kmeans

构造函数 Kmeans(count, dis)
Par1Int count 最终聚类的数量，即 K 值
Par2Int dis 维度，即每一个数据向量的维度

枚举类 Distance 距离算法
欧几里得距离 Euclidean = 0
曼哈顿距离 Manhattan = 1

实例化方法 fit(data, maxIter, threhold, distance, debug)

对模型进行训练
Par1np.array data 待训练的数据，维度为 N * dis
Par2int maxIter 最大迭代次数，缺省为 10000
Par3int threhold 中心点终止条件，代表 threhold 个中心点停止作为终止条件，-1 则是全部中心点停止，缺省为-1
Par4Kmeans.Distance distance 采用的距离算法，缺省为 Kmeans.Distance.Euclidean
Par5bool 是否开启 debug 模式，缺省为 0

实例化方法 classify(data, dis) 对数据进行分类
Par1np.array data 待分类的数据，维度为 N * dis
Par2Kmeans.Distance distance dis 采用的距离算法，缺省为 Kmeans.Distance.Euclidean

Return np.array 数据的分类结果，维度为 N * 1

静态方法 SSE(collective, data, dataClass) 计算 SSE
Par1np.array collective 中心点集，
Par2np.array data 具体数据
Par3np.array dataClass 每个数据的类型集合

类 BiKmeans
构造函数 Kmeans(count, dis)
Par1Int count 最终聚类的数量，即 K 值
Par2Int dis 维度，即每一个数据向量的维度

枚举类 Distance 距离算法
欧几里得距离 Euclidean = 0
曼哈顿距离 Manhattan = 1

实例化方法 fit(data, maxIter, threhold, distance, debug)
对模型进行训练
Par1np.array data 待训练的数据，维度为 N * dis
Par2int maxIter 最大迭代次数，缺省为 10000
Par3int threhold 中心点终止条件，代表 threhold 个中心点停止作为终止条件，-1 则是全部中心点停止，缺省为-1
Par4Kmeans.Distance distance 采用的距离算法，缺省为 Kmeans.Distance.Euclidean
Par5bool 是否开启 debug 模式，缺省为 0

静态方法 classify(collective, data, dis) 对数据进行分类
Par1np.array collective 中心点的集合，维度为 K * dis

Par2np.array data 待分类的数据，维度为 N * dis

Par3Kmeans.Distance distance dis 采用的距离算法，缺省为 Kmeans.Distance.Euclidean

Return np.array 数据的分类结果，维度为 N * 1

实验 1：

实验材料：iris 数据集

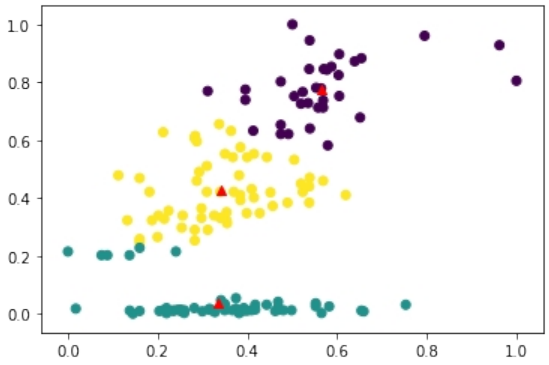
实验对象：Kmeans 包，sklearn.cluster.KMeans 包

对于复杂数据

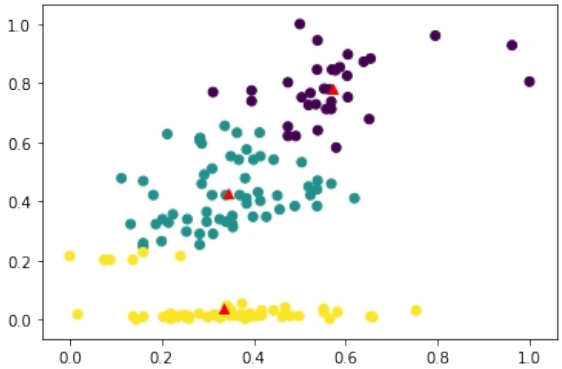
iris['sepal_size'] = iris['sepal_length'] * iris['sepal_width']

iris['petal_size'] = iris['petal_length'] * iris['petal_width']

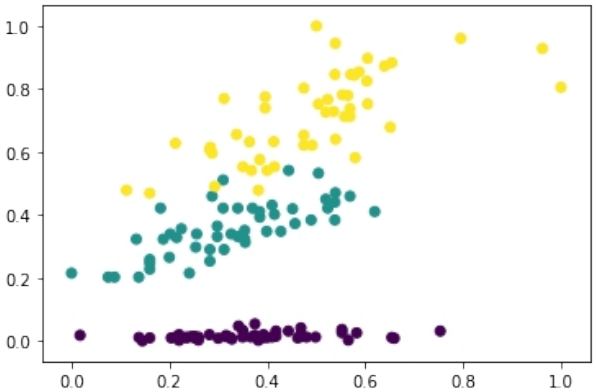
进行聚类处理



采用 SKLearn 的 Kmeans 进行的分类

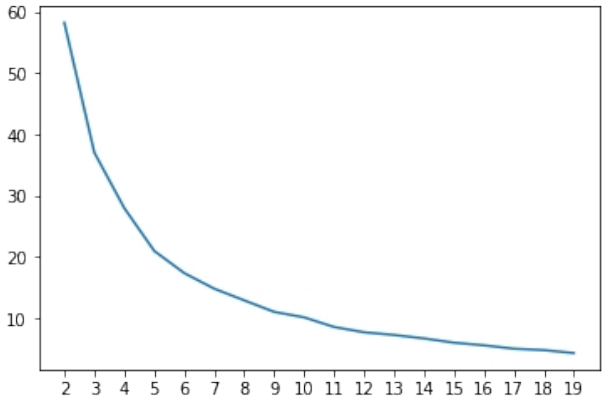


采用土制二分 Kmeans 进行的分类

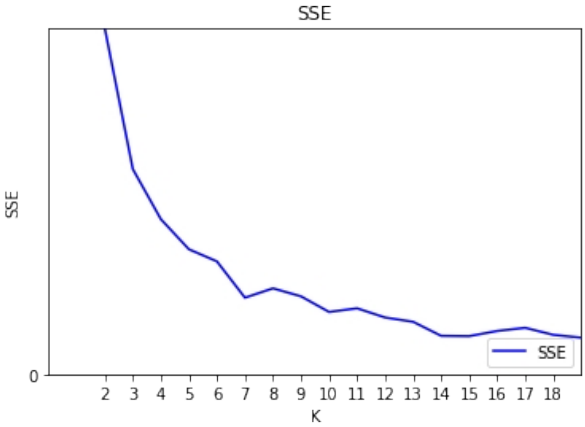


数据集的真正分类

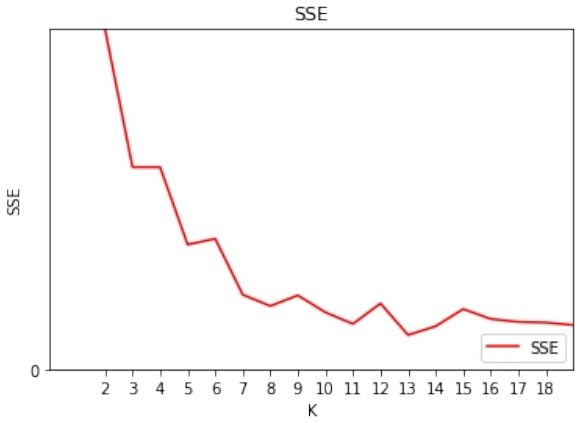
可以看出土制的 Kmeans 算法很好的拟合了 SKLearn 的 Kmeans 算法，几乎一致。
但是我再看一组随着 K 值而变化的 SSE 的图表



SKlearn 的 Kmeans 算法产生的 SSE 图像

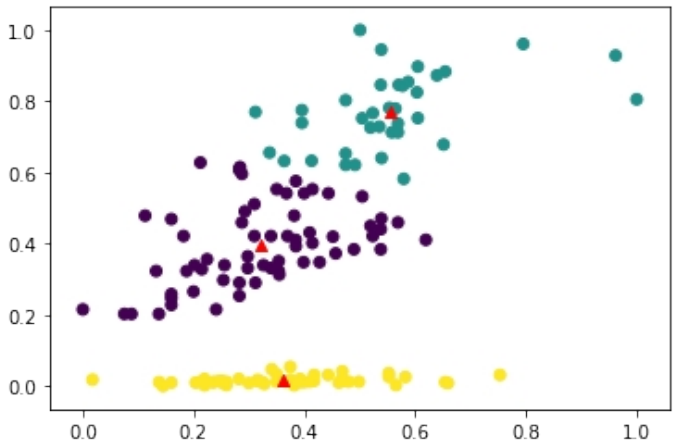


采用土制 Kmeans 算法产生的 SSE 图像



采用土制二分 Kmeans 算法产生的 SSE 图像

很显然，土制 Kmeans 的 SSE 波动厉害，线路曲折。尽管手肘法则得以判断标准的 K 值，但是也说明了在稳定性方面，SKLearn 略胜一筹。



并且看出由土制 Kmeans 法生成的原数据集标签几乎完全拟合的结果也可以发现本算法并不稳定，依赖于初值。

反向传播算法（BP）

BP 算法简介：BP 算法是基于梯度下降（SGD）神经网络的基础上，通过计算代价函数（Cost function）不断调整每个神经元的权重与偏置，使得代价函数最终达到一定的局部最小值，代价函数达到最小值就意味着错误几率最小，即从综合上（正确率、预报率等）最接近真实结果，即适用于预测也适用于分类。

优点在于：

- 可用于分类也可用于预测；
- 通用性好，并且随着隐藏层与输入输出层的调整可以用于高维数据
- 通过调整代价函数，可以实现识别率、精确率优先

缺点在于：

- 运算复杂，机器学习时间与周期长
- 属于黑箱问题，具体工作机理不明，难以人工调整内部算法
- 自由度高，学习速率、正则化参数不当容易陷入局部最优解
- 对数据训练集的要求高，需要人工对其进行预处理与分类

算法基本原理：

利用数学上的梯度性质，梯度表示某一函数在该点处的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大（为该梯度的模）。

也就是说设有代价函数

$E(w_1, w_2, \cdots, w_n, b_1, b_2, \cdots, b_m)$

因为E存在着

那么对E求梯度

有$\nabla E = \frac{\partial E}{\partial w_1} i_1 + \frac{\partial E}{\partial w_2} i_2 + \cdots + \frac{\partial E}{\partial w_n} i_n + \frac{\partial E}{\partial b_1} j_1 + \frac{\partial E}{\partial b_2} j_2 + \cdots + \frac{\partial E}{\partial b_m} j_m$

因此为E的最大下降速度方向，向量$\left(w_1, w_2, \cdots, w_n, b_1, b_2, \cdots, b_m\right) + \eta \Delta E'$（$\eta$为学习速率）

则必定会使得∇E慢慢趋向于0（证明略.），此时

$E(w_1, w_2, \cdots, w_n, b_1, b_2, \cdots, b_m)$得到局部最小值。

其中对函数求偏导数就需要用到数学中的链式法则（Chain Rule）从最高层一直算起直到最低层，故被称为反向传播。

下面我们用$\sigma(z)$表示单个神经元的激活函数

用$w_i^{d,e}$表示是第i层第d个神经元传导至第i + 1层第e个神经元的权重，偏置$b_i^{d,e}$同理

用net_i^d表示第$i-1$层的第d个神经元激活值在第i层第d个神经元的加权求和与第i层某个神经元偏置的和

用o_i^d表示$\sigma(net_i^d)$

用δ_i^d表示$\frac{\partial E}{\partial net_i^d}$即$\frac{\partial E}{\partial o_i^d} \frac{\partial o_i^d}{\partial net_i^d}$

对于代价函数对于权重的偏导有

$$\frac{\partial E}{\partial w_i^{d,e}} = \frac{\partial E}{\partial o_{i+1}^e} \frac{\partial o_{i+1}^e}{\partial net_{i+1}^e} \frac{\partial net_{i+1}^e}{\partial w_i^{d,e}}$$

因此可以得到具体公式（证明略.）

API：

包 network

类 network

构造函数(layers)

Par1List layers 隐藏层的设计（包括输入输出层）

枚举类 costFun：代价函数

MSE 0（Mean Squared Error）均方误差函数

CEE 1（Cross Entropy Error）交叉熵损失函数

实例化方法 feedForward(input) 进行一轮向前传播

Par1np.array input 输入的特征

Return np.array 返回最终的值

实例化方法 backProp(input, ans, eta, cost, lamdan)进行反向传播

Par1np.array input 要训练的特征

Par2np.array ans 准确的结果

Par3float eta 学习速率（Learn rate）

Par4network.network.costFun cost 损失函数

Par5float lamda / n 的值（lamda 为正则化参数，n 为样本总数）

Return np.array 返回最终的值

实验 2：

对 mnist 集进行分类

实验材料：MNIST 数据集

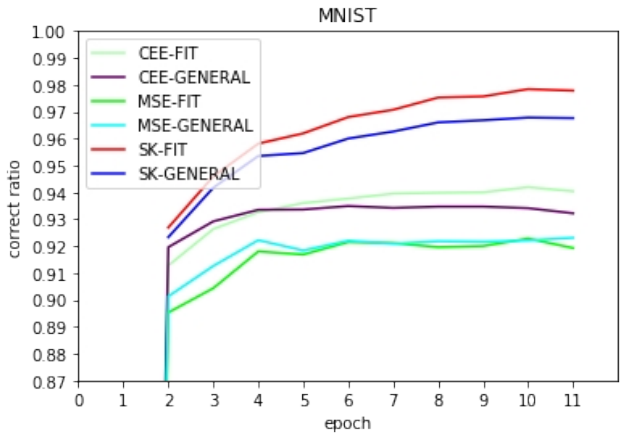
实验对象：

采用 MSE 的本算法（eta = 1.0, lamdn = 0.0, minibatch = 1）

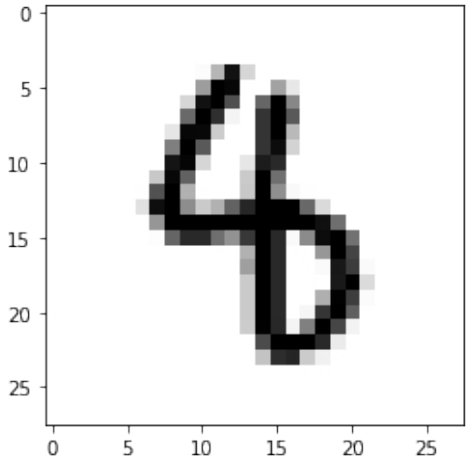
采用 CEE 的本算法（eta = 0.1, lamdan = 0.0，minibatch = 1）

SKLearn 的相同参数模型

实验结果：



由图不难看出，本算法的实现是相当好了，尤其是 CEE 要比 MSE 的学习速度还要高，仅一轮学习已经达到了 0.92 的好成绩，最终收敛于 0.95 左右。但是与 SKLearn 的标准模型相比，简直是被吊打，SKLearn 的拟合度即使是越过了 0.96 的界限仍然在上升。
另外我们再来看几个有趣的随机样本



这个图显然是旋转的 8，但是本模型处于 4 与 8 之间的竞合，最终选择了 4，但是对于我而言第一眼的确是很像异形的 4。因此对模型之改进，仍要加入对旋转图的识别，以及样本存在优化空间。

因此本算法的改进空间：

- 对于旋转的图像进行深度学习
- 增加 miniBatch 的数据处理方式
- 对 8 与 0 之间的识别进行深度学习