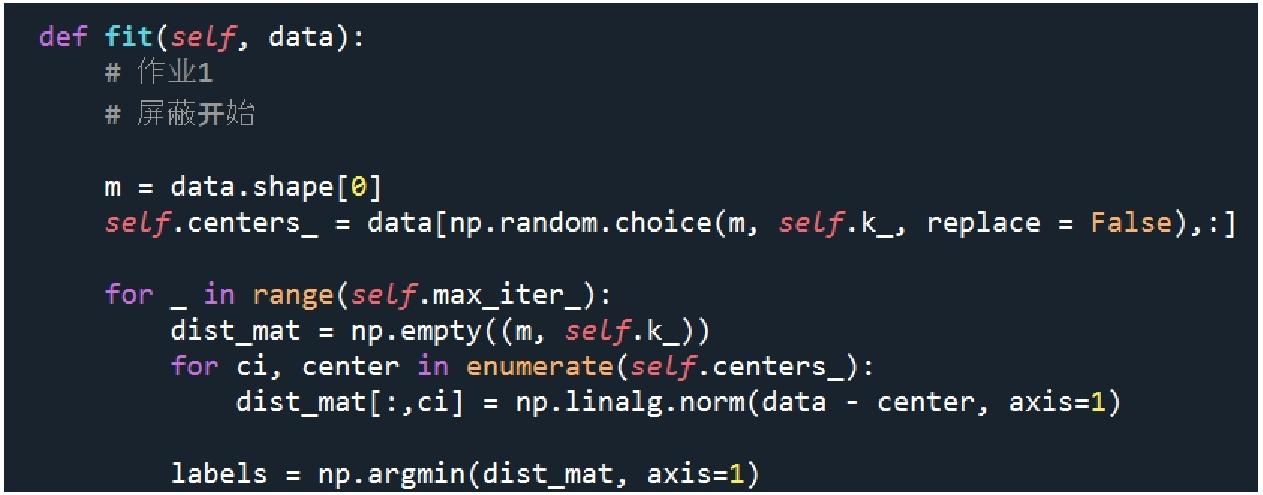
深蓝学院
shenlanxueyuan.com

三维点云处理第三次作业

主讲人 Lorenzo

K-Means

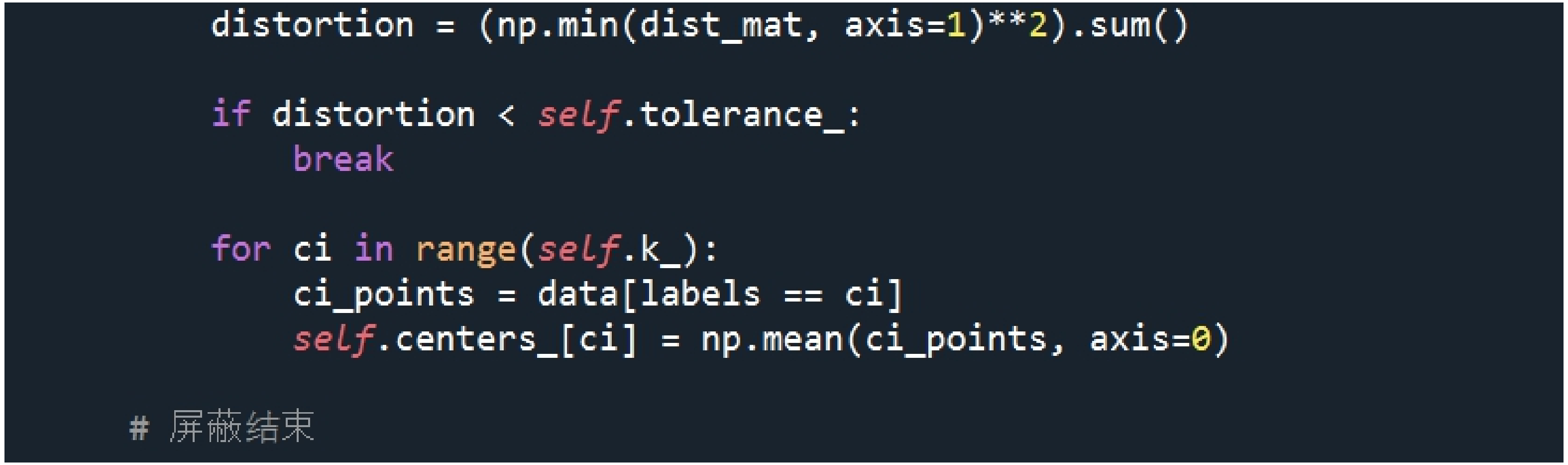
- fit函数E-step
  - 循环计算所有数据与每个聚类中心的距离，存储在dist_mat
  - 每笔数据的标签：与该笔数据最近的聚类中心的id



```python
def fit(self, data):
    # 作业1
    # 屏蔽开始

    m = data.shape[0]
    self.centers_ = data[np.random.choice(m, self.k_, replace = False),:]

    for _ in range(self.max_iter_):
        dist_mat = np.empty((m, self.k_))
        for ci, center in enumerate(self.centers_):
            dist_mat[:,ci] = np.linalg.norm(data - center, axis=1)

        labels = np.argmin(dist_mat, axis=1)
```

K-Means

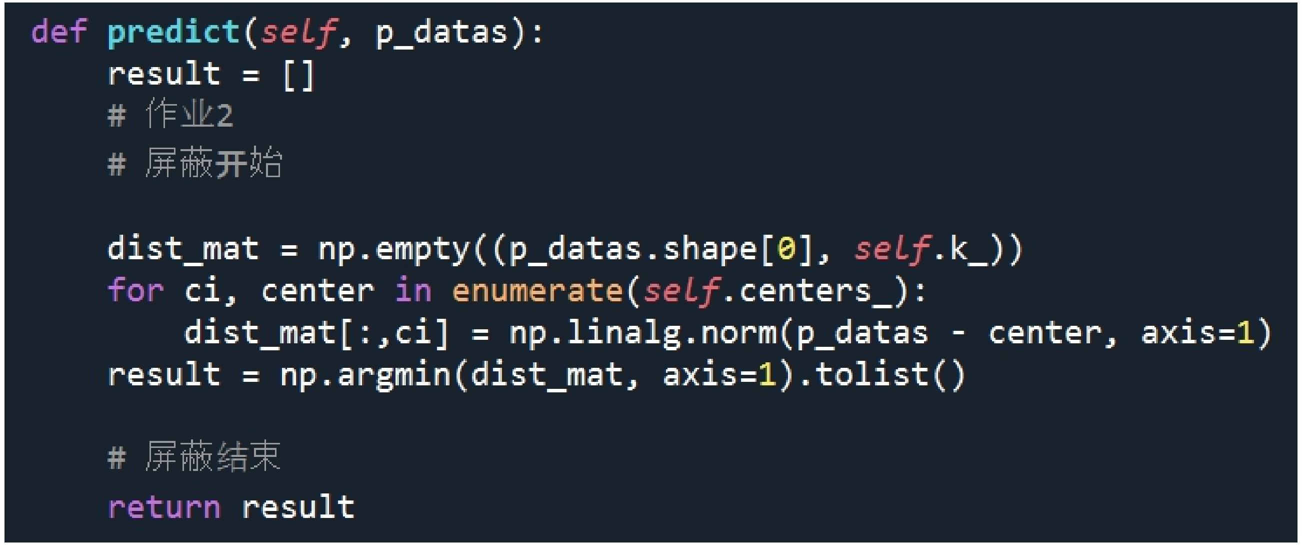
- fit函数M-step
  - distortion：所有数据点到所属聚类中心距离的平方和
  - 更新聚类中心：属于该聚类的所有点的坐标平均值



```
            distortion = (np.min(dist_mat, axis=1)**2).sum()

            if distortion < self.tolerance_:
                break

            for ci in range(self.k_):
                ci_points = data[labels == ci]
                self.centers_[ci] = np.mean(ci_points, axis=0)

        # 屏蔽结束
```

K-Means

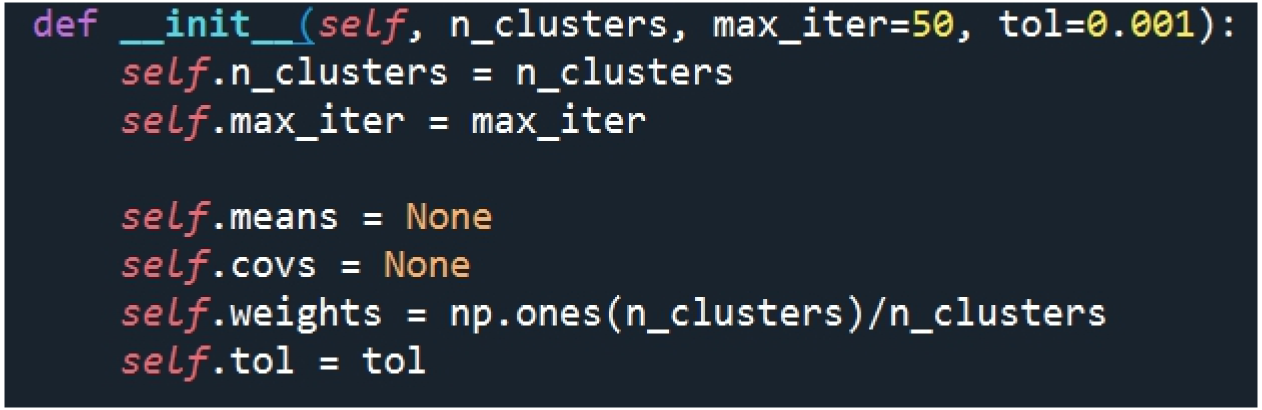
- predict函数
  - 循环计算所有数据与每个聚类中心的距离，存储在dist_mat
  - 每笔数据的标签：与该笔数据最近的聚类中心的id



```python
def predict(self, p_datas):
    result = []
    # 作业2
    # 屏蔽开始

    dist_mat = np.empty((p_datas.shape[0], self.k_))
    for ci, center in enumerate(self.centers_):
        dist_mat[:,ci] = np.linalg.norm(p_datas - center, axis=1)
    result = np.argmin(dist_mat, axis=1).tolist()

    # 屏蔽结束
    return result
```

GMM

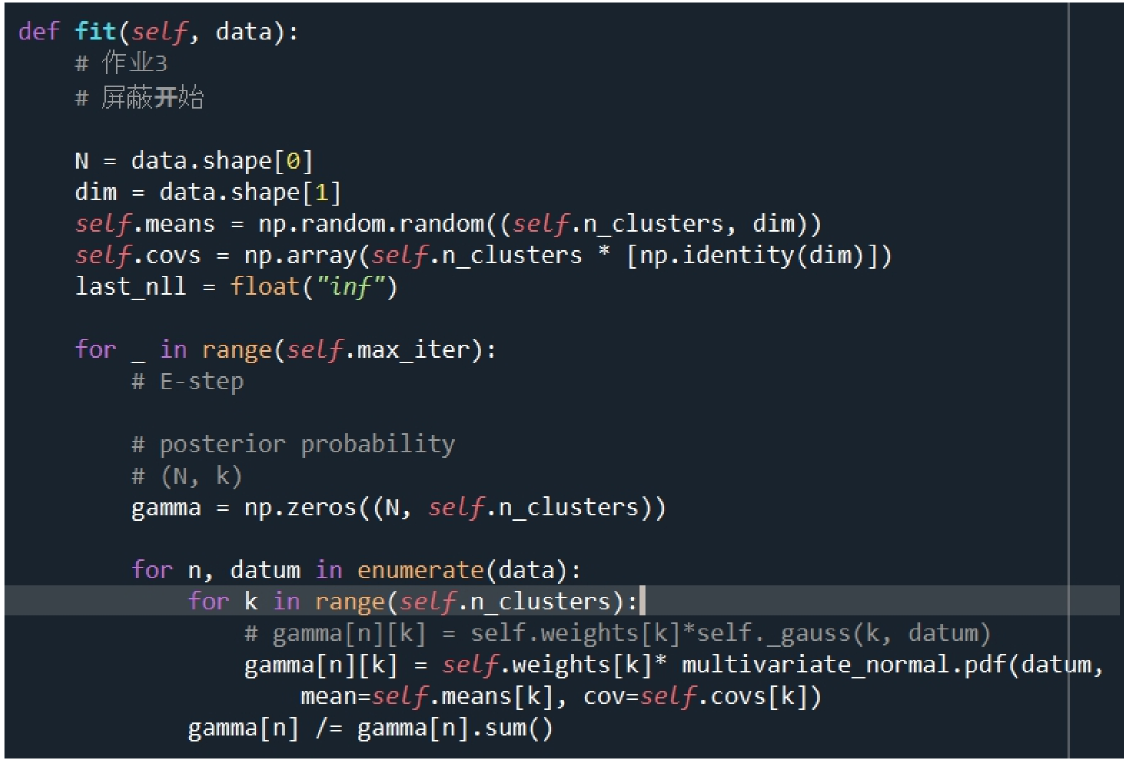
- 初始化



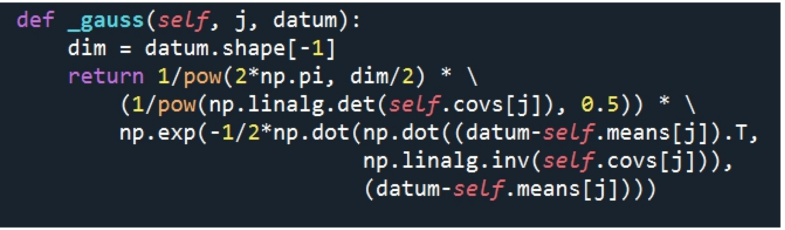
```python
def __init__(self, n_clusters, max_iter=50, tol=0.001):
    self.n_clusters = n_clusters
    self.max_iter = max_iter

    self.means = None
    self.covs = None
    self.weights = np.ones(n_clusters)/n_clusters
    self.tol = tol
```

GMM

- fit函数E-step
  - 计算数据点属于k个高斯模型的概率，即gamma



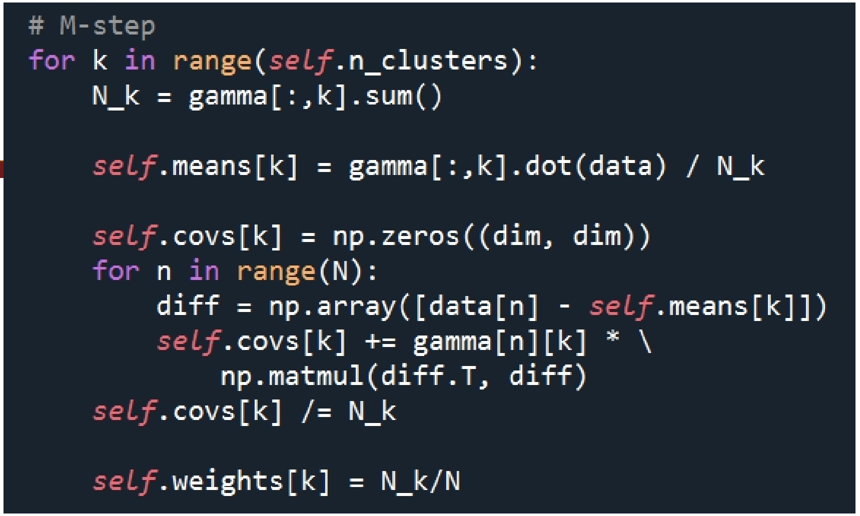
```
def fit(self, data):
    # 作业3
    # 屏蔽开始

    N = data.shape[0]
    dim = data.shape[1]
    self.means = np.random.random((self.n_clusters, dim))
    self.covs = np.array(self.n_clusters * [np.identity(dim)])
    last_nll = float("inf")

    for _ in range(self.max_iter):
        # E-step

        # posterior probability
        # (N, k)
        gamma = np.zeros((N, self.n_clusters))

        for n, datum in enumerate(data):
            for k in range(self.n_clusters):
                # gamma[n][k] = self.weights[k]*self._gauss(k, datum)
                gamma[n][k] = self.weights[k]* multivariate_normal.pdf(datum,
                    mean=self.means[k], cov=self.covs[k])
            gamma[n] /= gamma[n].sum()
```



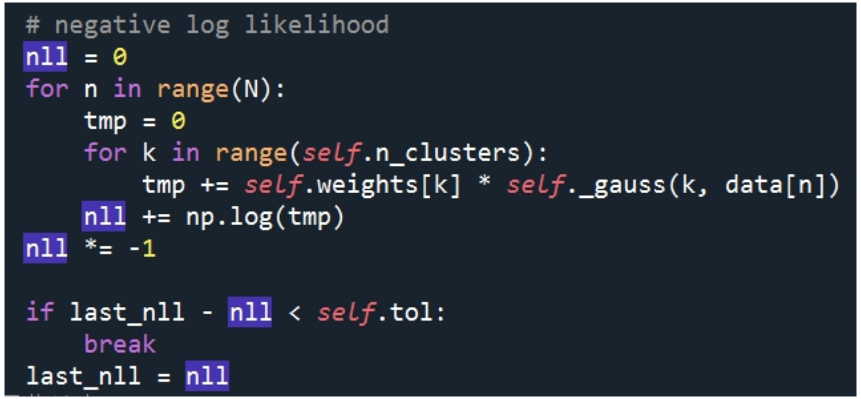
```
def _gauss(self, j, datum):
    dim = datum.shape[-1]
    return 1/pow(2*np.pi, dim/2) * \
        (1/pow(np.linalg.det(self.covs[j]), 0.5)) * \
        np.exp(-1/2*np.dot(np.dot((datum-self.means[j]).T,
                                  np.linalg.inv(self.covs[j])),
                                  (datum-self.means[j])))
```

GMM

- fit函数M-step
  - 更新k个高斯模型的mean, covariance matrix及weight
- 计算negative log likelihood, 判断是否该跳出循环



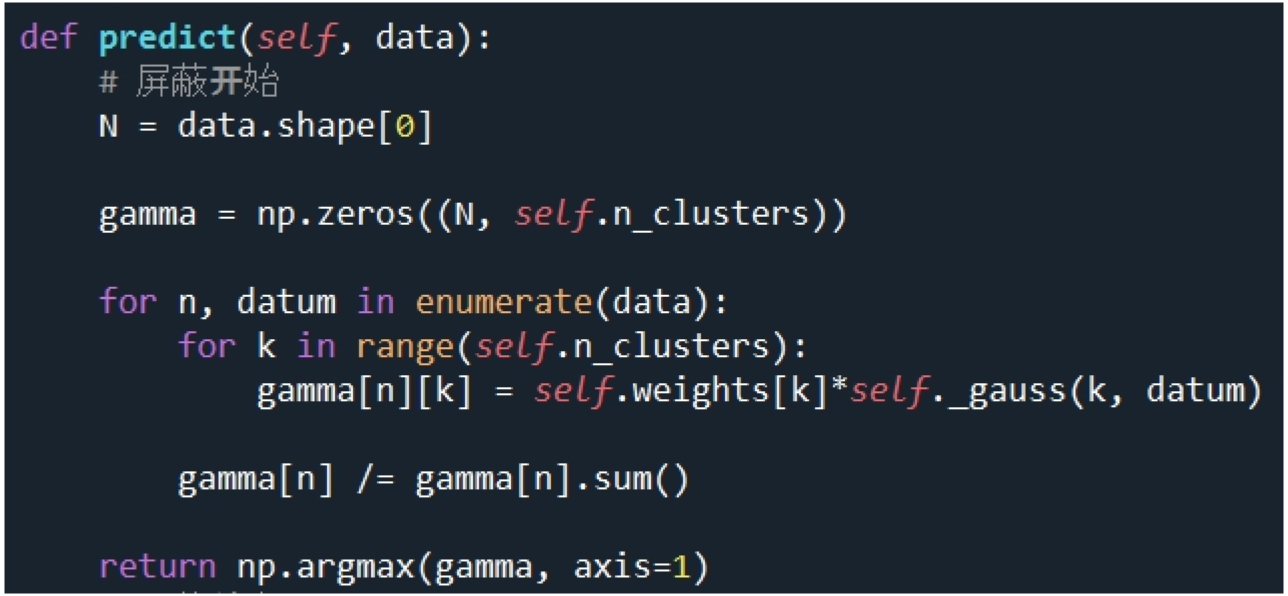
```
# M-step
for k in range(self.n_clusters):
    N_k = gamma[:,k].sum()

    self.means[k] = gamma[:,k].dot(data) / N_k

    self.covs[k] = np.zeros((dim, dim))
    for n in range(N):
        diff = np.array([data[n] - self.means[k]])
        self.covs[k] += gamma[n][k] * \
            np.matmul(diff.T, diff)
    self.covs[k] /= N_k

    self.weights[k] = N_k/N
```



```
# negative log likelihood
nll = 0
for n in range(N):
    tmp = 0
    for k in range(self.n_clusters):
        tmp += self.weights[k] * self._gauss(k, data[n])
    nll += np.log(tmp)
nll *= -1

if last_nll - nll < self.tol:
    break
last_nll = nll
```

GMM

●predict函数

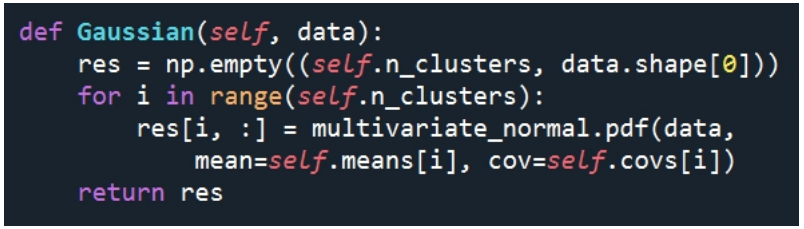


```
def predict(self, data):
    # 屏蔽开始
    N = data.shape[0]

    gamma = np.zeros((N, self.n_clusters))

    for n, datum in enumerate(data):
        for k in range(self.n_clusters):
            gamma[n][k] = self.weights[k]*self._gauss(k, datum)

        gamma[n] /= gamma[n].sum()

    return np.argmax(gamma, axis=1)
```

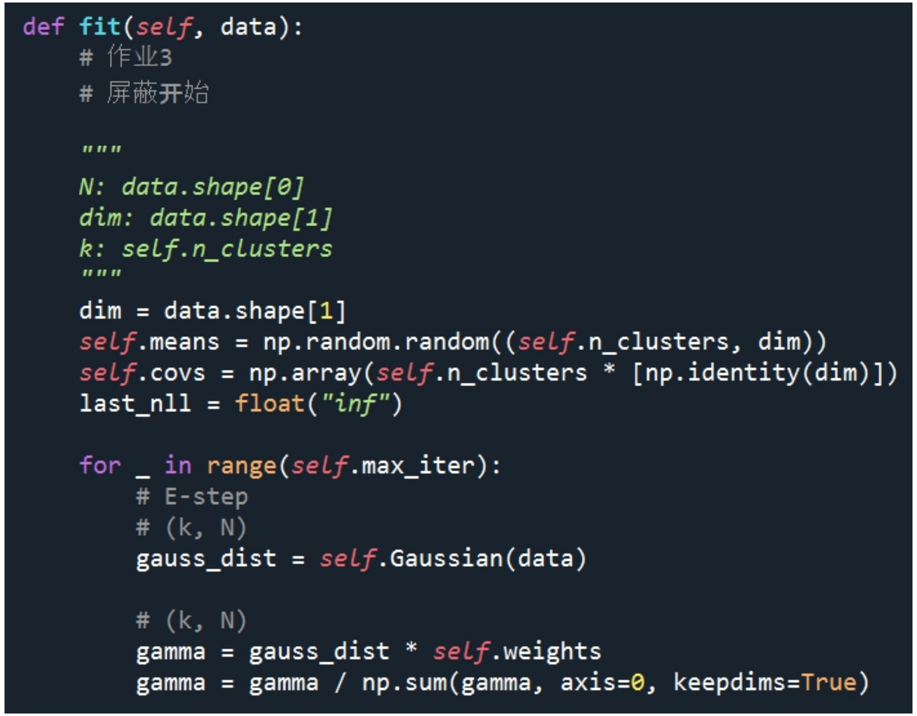
GMM-向量化

深蓝学院 shenlanxueyuan.com

- fit函数E-step
  - Gaussian函数：计算“在k个高斯模型里出现这N笔数据的概率”
  - 计算gamma



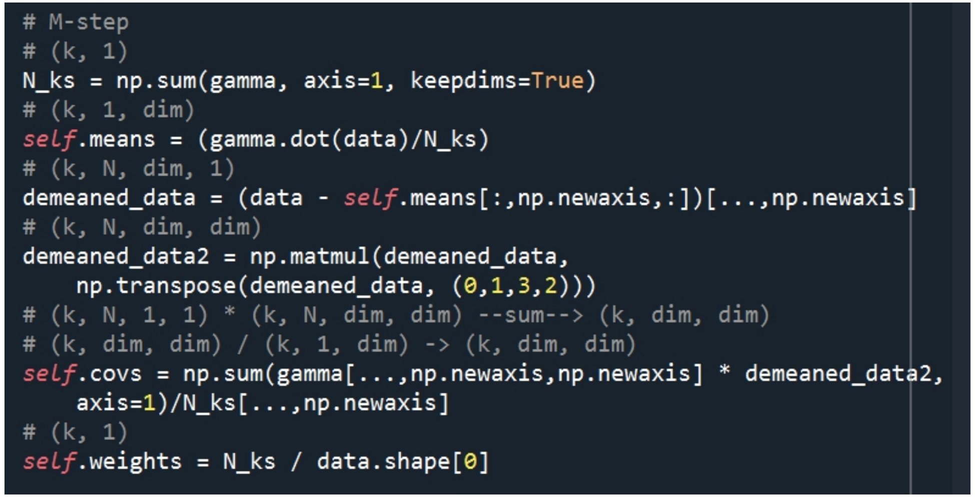
```python
def Gaussian(self, data):
    res = np.empty((self.n_clusters, data.shape[0]))
    for i in range(self.n_clusters):
        res[i, :] = multivariate_normal.pdf(data,
            mean=self.means[i], cov=self.covs[i])
    return res
```



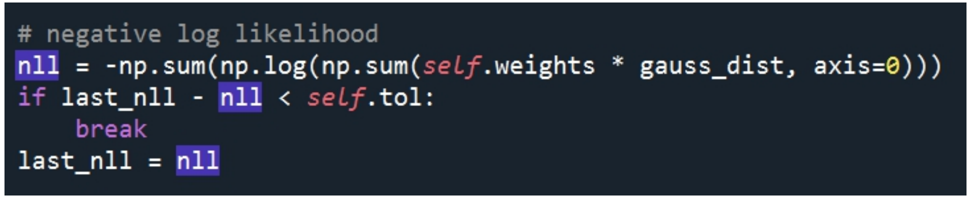
```python
def fit(self, data):
    # 作业3
    # 屏蔽开始

    """
    N: data.shape[0]
    dim: data.shape[1]
    k: self.n_clusters
    """
    dim = data.shape[1]
    self.means = np.random.random((self.n_clusters, dim))
    self.covs = np.array(self.n_clusters * [np.identity(dim)])
    last_nll = float("inf")

    for _ in range(self.max_iter):
        # E-step
        # (k, N)
        gauss_dist = self.Gaussian(data)

        # (k, N)
        gamma = gauss_dist * self.weights
        gamma = gamma / np.sum(gamma, axis=0, keepdims=True)
```

GMM-向量化

- fit函数M-step
  - 更新n_clusters个高斯模型的mean, covariance matrix及weight
- 计算negative log likelihood



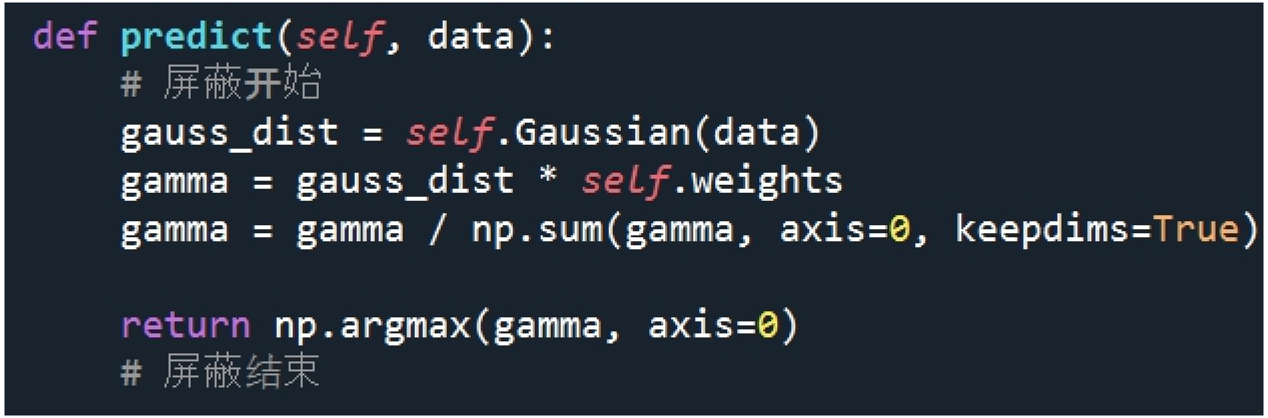
```
# M-step
# (k, 1)
N_ks = np.sum(gamma, axis=1, keepdims=True)
# (k, 1, dim)
self.means = (gamma.dot(data)/N_ks)
# (k, N, dim, 1)
demeaned_data = (data - self.means[:,np.newaxis,:])[...,np.newaxis]
# (k, N, dim, dim)
demeaned_data2 = np.matmul(demeaned_data,
    np.transpose(demeaned_data, (0,1,3,2)))
# (k, N, 1, 1) * (k, N, dim, dim) --sum--> (k, dim, dim)
# (k, dim, dim) / (k, 1, dim) -> (k, dim, dim)
self.covs = np.sum(gamma[...,np.newaxis,np.newaxis] * demeaned_data2,
    axis=1)/N_ks[...,np.newaxis]
# (k, 1)
self.weights = N_ks / data.shape[0]
```



```
# negative log likelihood
nll = -np.sum(np.log(np.sum(self.weights * gauss_dist, axis=0)))
if last_nll - nll < self.tol:
    break
last_nll = nll
```

GMM-向量化

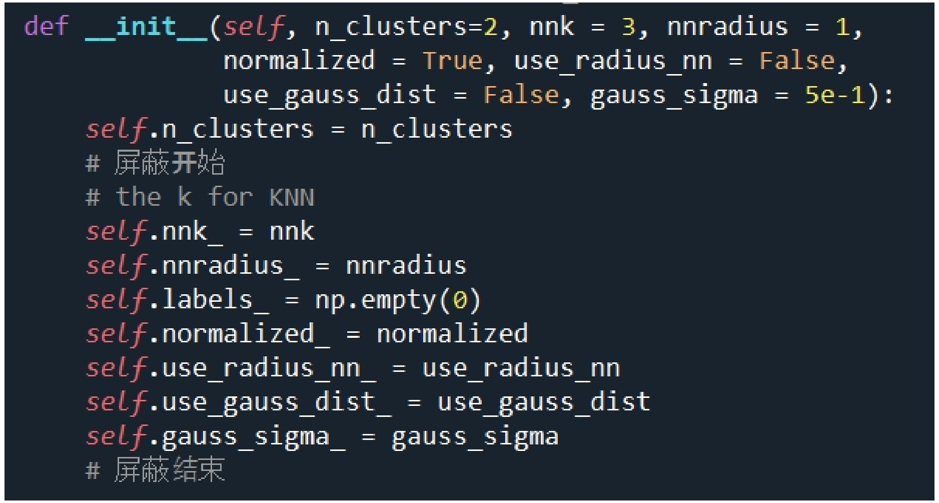
- predict函数
  - 计算gamma
  - 把数据分配到可能性最大的类别里



```python
def predict(self, data):
    # 屏蔽开始
    gauss_dist = self.Gaussian(data)
    gamma = gauss_dist * self.weights
    gamma = gamma / np.sum(gamma, axis=0, keepdims=True)

    return np.argmax(gamma, axis=0)
    # 屏蔽结束
```

Spectral Clustering

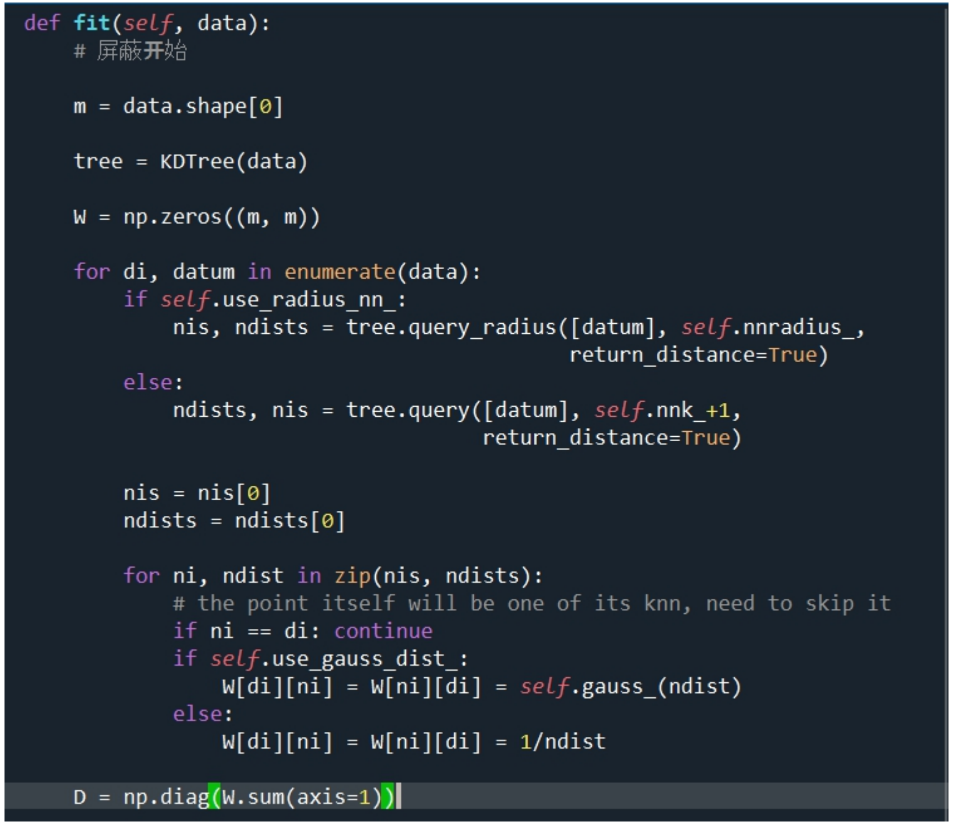
- 初始化
  - normalized_：Laplacian是否做normalization
  - use_radius_nn_：使用radius nn或knn
  - nnk_，nnradius_：用于寻找最近邻
  - use_gauss_dist：使用距离的倒数或高斯函数计算权重
  - gauss_sigma：高斯函数的标准差



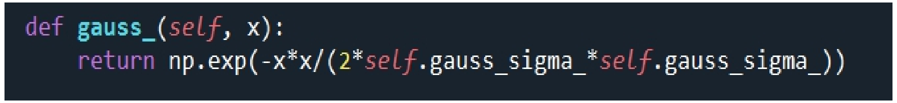
```
def __init__(self, n_clusters=2, nnk = 3, nnradius = 1,
             normalized = True, use_radius_nn = False,
             use_gauss_dist = False, gauss_sigma = 5e-1):
    self.n_clusters = n_clusters
    # 屏蔽开始
    # the k for KNN
    self.nnk_ = nnk
    self.nnradius_ = nnradius
    self.labels_ = np.empty(0)
    self.normalized_ = normalized
    self.use_radius_nn_ = use_radius_nn
    self.use_gauss_dist_ = use_gauss_dist
    self.gauss_sigma_ = gauss_sigma
    # 屏蔽结束
```

Spectral Clustering

- fit函数
  - 近邻查找方式：radius nn或knn
  - 建构adjacency matrix W
  - 将权重设为
    - 距离的倒数 或是
    - 将距离代入高斯函数得到的值
  - 计算degree matrix D



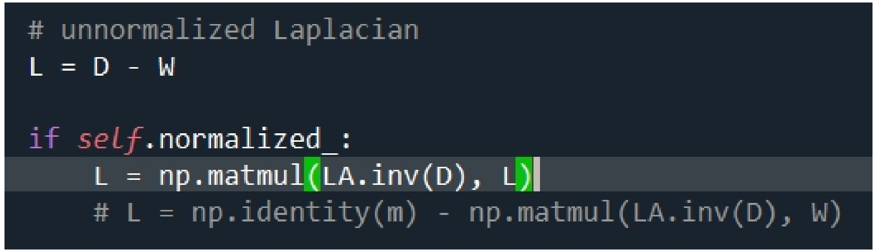
```python
def fit(self, data):
    # 屏蔽开始

    m = data.shape[0]

    tree = KDTree(data)

    W = np.zeros((m, m))

    for di, datum in enumerate(data):
        if self.use_radius_nn_:
            nis, ndists = tree.query_radius([datum], self.nnradius_,
                                            return_distance=True)
        else:
            ndists, nis = tree.query([datum], self.nnk_+1,
                                     return_distance=True)

        nis = nis[0]
        ndists = ndists[0]

        for ni, ndist in zip(nis, ndists):
            # the point itself will be one of its knn, need to skip it
            if ni == di: continue
            if self.use_gauss_dist_:
                W[di][ni] = W[ni][di] = self.gauss_(ndist)
            else:
                W[di][ni] = W[ni][di] = 1/ndist

    D = np.diag(W.sum(axis=1))
```



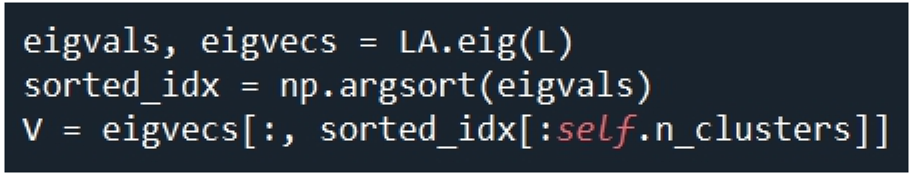
```python
def gauss_(self, x):
    return np.exp(-x*x/(2*self.gauss_sigma_*self.gauss_sigma_))
```

Spectral Clustering

- fit函数
  - 由D及W计算L
  - L_rw有两种计算方式 $L_{rw} = D^{-1}L = I - D^{-1}W$
  - 选取最小的k个特征值对应的特征向量建构V矩阵
  - 注意numpy.linalg.eig返回的特征值及特征向量是未排序的
  - (import numpy.linalg as LA)



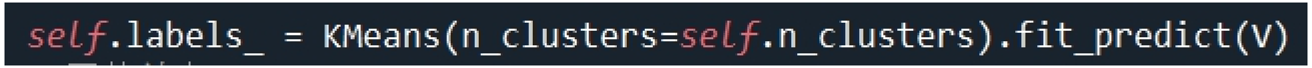
```python
# unnormalized Laplacian
L = D - W

if self.normalized_:
    L = np.matmul(LA.inv(D), L)
    # L = np.identity(m) - np.matmul(LA.inv(D), W)
```



```python
eigvals, eigvecs = LA.eig(L)
sorted_idx = np.argsort(eigvals)
V = eigvecs[:, sorted_idx[:self.n_clusters]]
```

Spectral Clustering

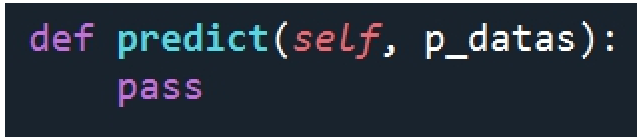
- fit函数
  - 使用KMeans对V矩阵的各row进行分类
  - 将结果储存于self.labels_



```python
self.labels_ = KMeans(n_clusters=self.n_clusters).fit_predict(V)
```

Spectral Clustering

- predict函数
  - 谱聚类算法较难对新数据进行分类
  - 参考sklearn的API，同样没有predict函数：https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html



```python
def predict(self, p_datas):
    pass
```

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听!
Thanks for Listening