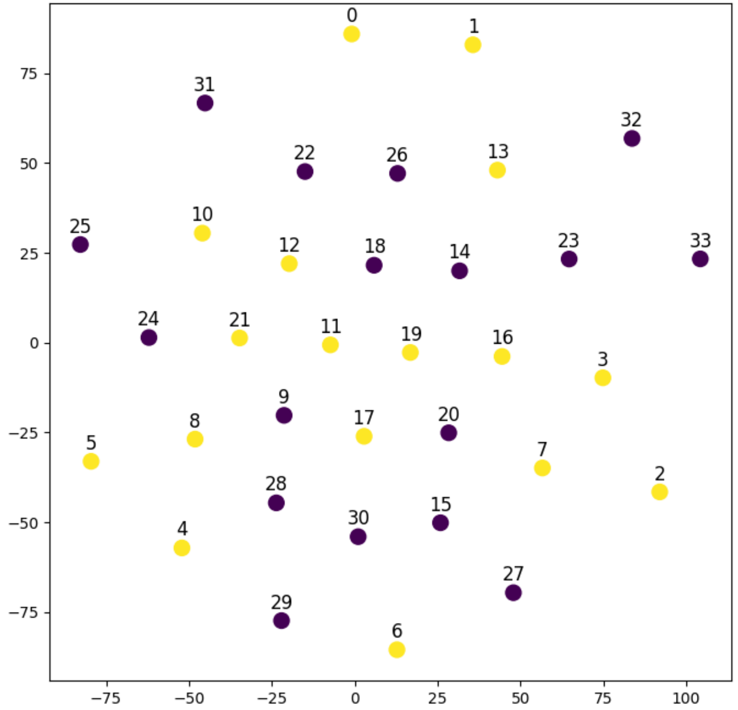
HW12-作业讲评

王瑞环

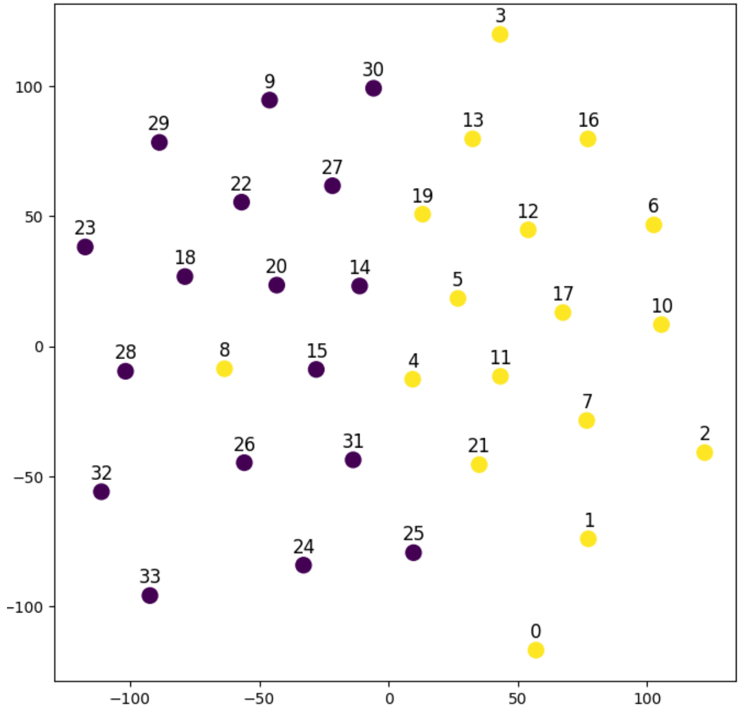
1.2.1 PCA嵌入

```python
def PCA_embed(G, dim):
    pca = PCA(n_components=dim)
    adjacency_matrix = nx.to_numpy_array(G)
    embeddings = pca.fit_transform(adjacency_matrix)
    return {str(i): embeddings[i] for i in G}
```

networkx >= 2.7

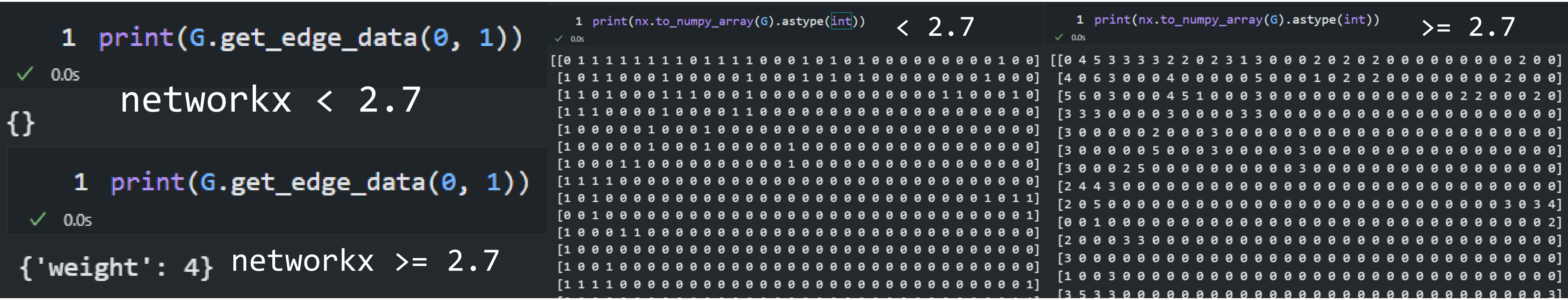


networkx < 2.7



1.2.1 PCA嵌入

- nx.to_numpy_array
  - 得到的邻接矩阵中若边(i,j)带有权重w，则A[i,j] = w
  - 若无权重，则A[i,j]=1
- 在大于等于2.7版本的networkx中，给KarateClub数据集加上了边权



1.2.2 DeepWalk

```
def deep_walk_once(G, start_node, walk_length):
    walk = [str(start_node)]
    current_node = start_node

    for _ in range(walk_length - 1):

        neighbors = list(G.neighbors(current_node))
        current_node = random.choice(neighbors)

        walk.append(str(current_node))
    return walk
```

1.2.3 Node2Vec

- np.random.choice (a, p=...)
  - a: 用于采样的array
  - p: 概率分布
- 这段代码在大图上效率非常低，可以用集合操作邻居、np向量化、并行等进行优化

```python
def node2vec_once(G, start_node, walk_length, p, q):
    walk = [str(start_node)]
    current_node = start_node
    previous_node = None

    for _ in range(walk_length - 1):
        neighbors = list(G.neighbors(current_node))
        if previous_node is None:
            # 初次游走，等概率随机选取邻居节点
            probabilities = [1 / len(neighbors) for nbr in neighbors]
        else:
            probabilities = []
            for neighbor in neighbors:
                weight = 1 / p if neighbor == previous_node \
                    else 1 if G.has_edge(neighbor, previous_node) \
                        else 1 / q
                probabilities.append(weight)
            probabilities = np.array(probabilities) / np.sum(probabilities)
        previous_node = current_node
        current_node = np.random.choice(neighbors, p=probabilities)

        walk.append(str(current_node))

    return walk
```

2. PageRank

- nx.adjacency_matrix
  - 获取邻接矩阵，但得到的是稀疏矩阵
  - 使用todense()转化为array

```
1  nx.adjacency_matrix(G)
✓ 0.0s

<34x34 sparse array of type '<class 'numpy.intc'>'
	with 156 stored elements in Compressed Sparse Row format>
```

```
1  nx.adjacency_matrix(G).todense()
✓ 0.0s

array([[0, 4, 5, ..., 2, 0, 0],
       [4, 0, 6, ..., 0, 0, 0],
       [5, 6, 0, ..., 0, 2, 0],
       ...,
       [2, 0, 0, ..., 0, 4, 4],
       [0, 0, 2, ..., 4, 0, 5],
       [0, 0, 0, ..., 4, 5, 0]], dtype=int32)
```

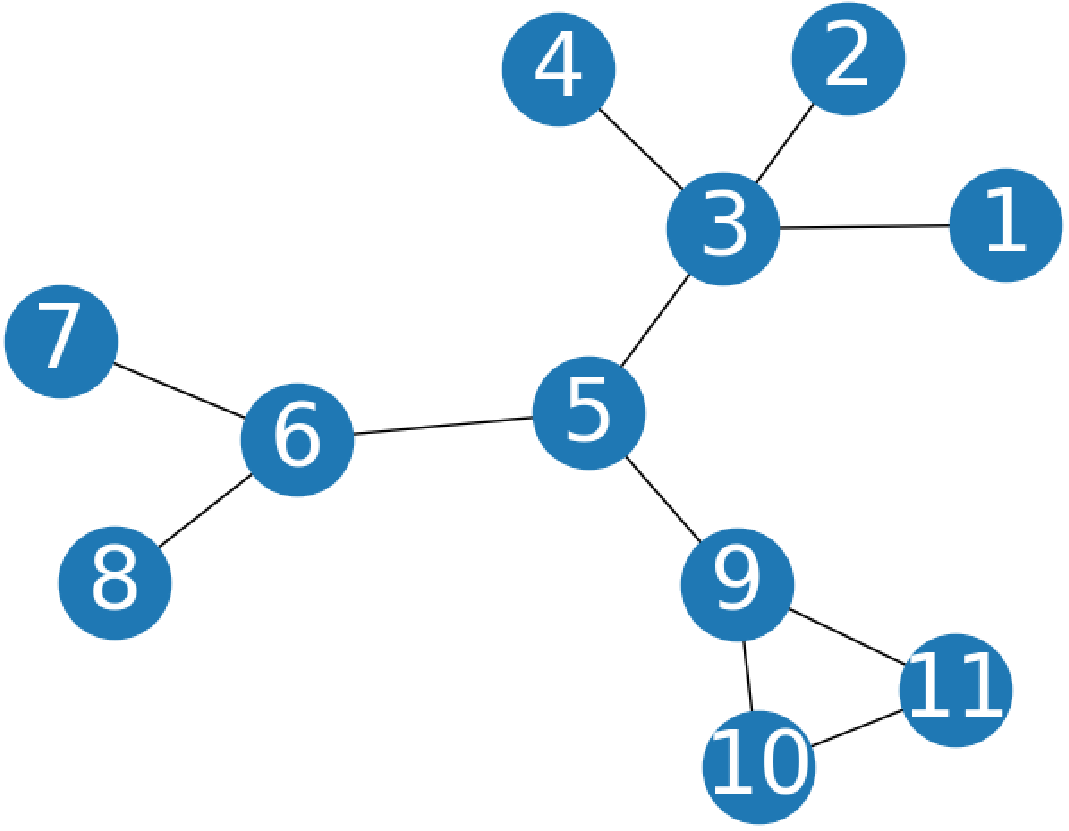
```python
def page_rank(G, d=0.85, tol=1e-2, max_iter=100):
    nodes = G.nodes()

    adj_matrix = nx.adjacency_matrix(G, nodelist=nodes)
    out_degree = adj_matrix.sum(axis=0)
    A = adj_matrix / (out_degree + 1e-10)

    N = G.number_of_nodes()
    pr = np.ones((N, 1)) / N

    yield nodes, pr, "init"
    yield nodes, pr, "init"

    for it in range(max_iter):
        old_pr = pr[:]

        alpha = np.ones((N, 1))
        pr = (1 - d) / N * alpha + d * A @ pr

        yield nodes, pr, it
        err = np.abs(pr - old_pr).sum()
        if err < tol:
            return pr
```

2. PageRank

- 如果使用for循环给A矩阵赋值，注意图的节点编号是从1开始的

```
A = np.zeros((len(nodes), len(nodes)))
for i in range(len(nodes)):
    for j in range(len(nodes)):
        if G.has_edge(i + 1, j + 1):
            A[i, j] = 1 / G.degree(j + 1)
```

```
A = np.zeros((len(nodes), len(nodes)))
for i in range(1, len(nodes) + 1):
  for j in range(1, len(nodes) + 1):
    if G.has_edge(i, j):
      A[i - 1, j - 1] = 1 / G.degree(j)
```



3.1 谱聚类

```python
def getNormLaplacian(G):

    adjacency_matrix = nx.to_numpy_array(G)

    out_degree = adjacency_matrix.sum(axis=0)
    D = np.diag(out_degree)
    Dn = np.linalg.inv(D ** 0.5)

    L_norm = np.eye(G.number_of_nodes()) - (Dn @ adjacency_matrix @ Dn)
    return L_norm
```

3.2 标签传播

```
for node in G:
    count = {}
    for nbr in G.neighbors(node):
        label = G.nodes[nbr]['labels']
        count[label] = count.setdefault(label,0) + 1

    count_items = sorted(count.items(),key=lambda x:-x[-1])
    best_labels = [k for k,v in count_items if v == count_items[0][1]]
    label = random.sample(best_labels,1)[0]

    if G.nodes[node]['labels'] != label:
        is_stopped = False
    G.nodes[node]['labels'] = label
```

3.2 标签传播

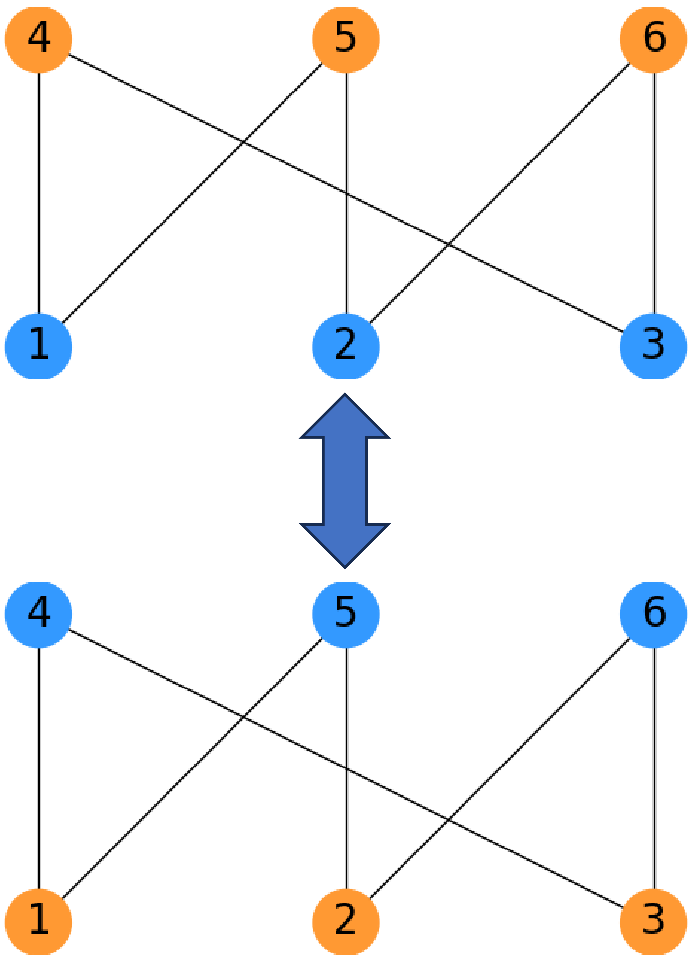
- np.bincount(arr)
  - arr是整数矩阵，返回0~arr.max()的整数数目

```
1 np.bincount([1, 2, 2, 3, 5])
✓ 0.0s

array([0, 1, 2, 1, 0, 1], dtype=int64)
```

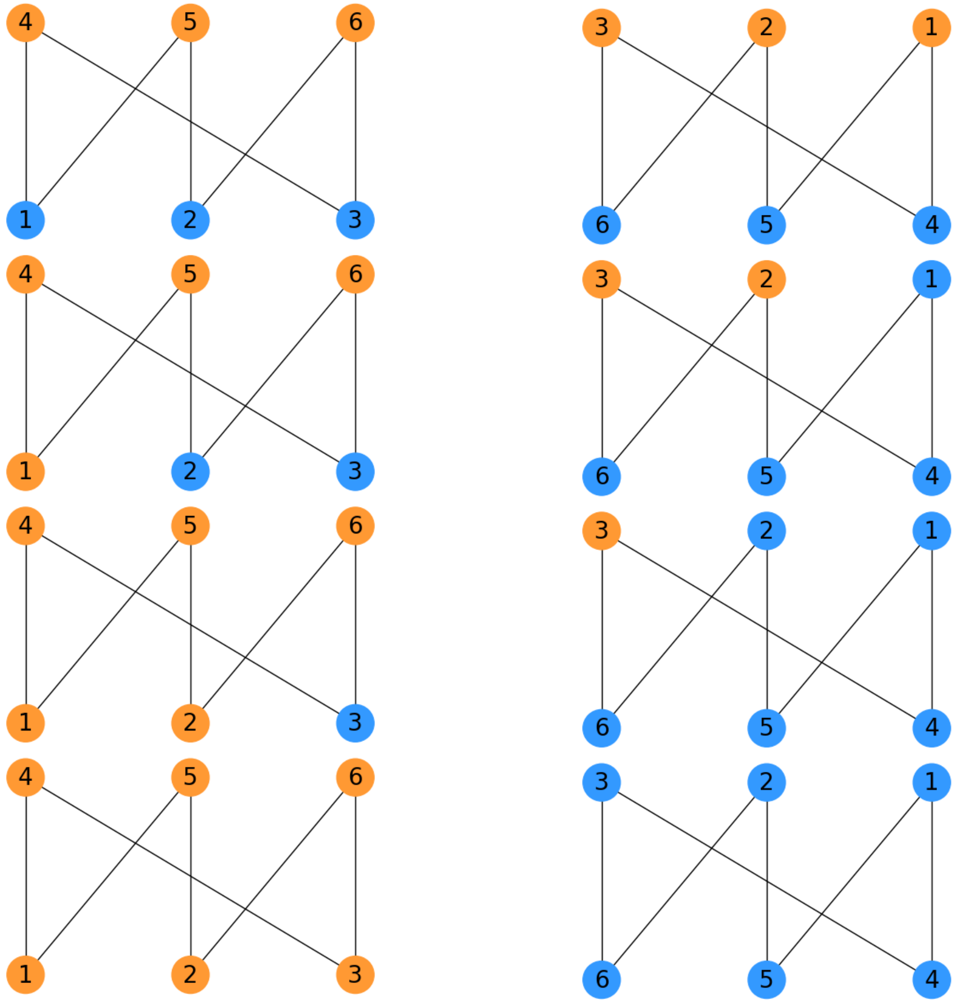
```
# 周宇亮
for node in G:
    lbc = np.bincount(list(map(lambda x:G.nodes[x]["labels"], G.neighbors(node))))
    new = random.choice(np.arange(lbc.shape[0])[lbc == lbc.max()])
    if G.nodes[node]["labels"] != new:
        is_stopped = False
        G.nodes[node]["labels"] = new
```

3.2 标签传播 – 同步和异步

同步：陷入震荡



异步：能收敛，但结果不稳定，与更新节点的顺序有关



4.1 数据预处理

```
ratings_counts = (ratings > 0).loc[:, "1":].sum(axis=1)
ratings_counts_geq100 = ratings_counts[ratings_counts >= 100].index

users_new = users.loc[ratings_counts_geq100]
users_new['counts'] = ratings_counts.loc[ratings_counts_geq100]
ratings_new = ratings.loc[ratings_counts_geq100]
```

```
ratings
```

	user_id	1	2	3	4	5	6	7	8	9	...	3942	3943	3945	3946	3947	3948	3949	3950	3951	3952
0	1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

4.2 特征嵌入

```python
def feature_embedding(X_train, X_test, **kwargs):
    pipeline = make_pipeline(         # macro-f1 0.37
        PCA(n_components=100, random_state=42))
    pipeline = make_pipeline(         # macro-f1 0.40
        PCA(n_components=100, random_state=42, svd_solver='full'))
    pipeline = make_pipeline(         # macro-f1 0.41
        Normalizer(), PCA(n_components=100, random_state=42))
    pipeline = make_pipeline(         # macro-f1 0.33
        TruncatedSVD(n_components=100, random_state=42))
    pipeline = make_pipeline(         # macro-f1 0.34
        StandardScaler(), TruncatedSVD(n_components=100, random_state=42))
    pipeline = make_pipeline(         # macro-f1 0.39
        Normalizer(), TruncatedSVD(n_components=100, random_state=42))

    pipeline.fit(X_train)
    X_train_embedded = pipeline.transform(X_train)
    X_test_embedded = pipeline.transform(X_test)
    return X_train_embedded, X_test_embedded
```

4.2 特征嵌入

- Normalizer和StandardScaler的区别
  - Normalizer.fit不起任何作用，因为Normalizer没有要训练的参数，Normalizer.transform(X)会返回 X/‖X‖
  - StandardScaler.fit(X1)会提取出X1的均值和标准差mean_1, std_1，StandardScaler.transform(X2)会返回 (X2-mean_1) / std_1
- PCA和TruncatedSVD的区别
  - PCA会对数据做中心化，中心化后的数据PCA和TruncatedSVD是一致的
  - 中心化会导致稀疏矩阵变稠密，因此PCA无法接受稀疏数据，但TruncatedSVD可以

```
data = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
print(PCA(n_components=1).fit_transform(data).flatten())
print(TruncatedSVD(n_components=1).fit_transform(data).flatten())
```
✓ 0.0s
```
[ 5.19615242 -0.          -5.19615242]
[ 3.6196      8.77091014 13.92222029]
```

```
data1 = data - np.mean(data, axis=0, keepdims=True)
print(PCA(n_components=1).fit_transform(data1).flatten())
print(TruncatedSVD(n_components=1).fit_transform(data1).flatten())
```
✓ 0.0s
```
[ 5.19615242 -0.          -5.19615242]
[ 5.19615242  0.          -5.19615242]
```

4.3 有序分类

- 其实就是变相地增加模型参数
- 如果分类器换成kNN则不会有明显效果提升

```
def predict(self, X):
    cls_preds = [classifier.predict_proba(X) for classifier in self.all_classifiers]
    predicted = []
    for i in range(len(self.unique_class)):
        if i == 0:
            predicted.append(1 - cls_preds[i][:, 1])
        elif i < len(cls_preds):
            predicted.append(cls_preds[i-1][:, 1] - cls_preds[i][:, 1])
        else:
            predicted.append(cls_preds[i-1][:, 1])
    probs = np.vstack(predicted).T
    y_idx = np.argmax(probs, axis=1)
    y = self.unique_class[y_idx]
    return y
```