# 机器学习DAY07

## 中文分词（jieba）

https://github.com/fxsjy/jieba

## 样本类别均衡化

**上采样与下采样处理样本类别均衡化**

下采样：把样本数据量大的那一类样本减少到与数据量小的那一类样本数量相近。

上采样：把样本数据量小的那一类样本增加到与数据量大的那一类样本数量相近。

通过类别权重的均衡化，使所占比例较小的样本权重较高，而所占比例较大的样本权重较低，以此平均化不同类别样本对分类模型的贡献，提高模型性能。

样本类别均衡化相关API：

```
model = svm.SVC(kernel='linear', class_weight='balanced')
model.fit(train_x, train_y)
class_weight={0:0.9, 1:0.1}
```

案例：修改线性核函数的支持向量机案例，基于样本类别均衡化读取imbalance.txt训练模型。

```
... ...
... ...
data = np.loadtxt('../data/imbalance.txt', delimiter=',', dtype='f8')
x = data[:, :-1]
y = data[:, -1]
train_x, test_x, train_y, test_y = \
    ms.train_test_split(x, y, test_size=0.25, random_state=5)
# 基于线性核函数的支持向量机分类器
model = svm.SVC(kernel='linear', class_weight='balanced')
model.fit(train_x, train_y)
... ...
... ...

LR  SVM  NB  Tree
```

## 置信概率

根据样本与分类边界的距离远近，对其预测类别的可信程度进行量化，离边界越近的样本，置信概率越低，反之，离边界越远的样本，置信概率高。

获取每个样本的置信概率相关API：

```
# 在获取模型时，给出超参数probability=True
model = svm.SVC(kernel='rbf', C=600, gamma=0.01, probability=True)
预测结果 = model.predict(输入样本矩阵)
# 调用model.predict_proba(样本矩阵)可以获取每个样本的置信概率矩阵
置信概率矩阵 = model.predict_proba(输入样本矩阵)
```

置信概率矩阵格式如下：

| | 类别1 | 类别2 |
|---|---|---|
| 样本1 | 0.8 | 0.2 |
| 样本2 | 0.9 | 0.1 |
| 样本3 | 0.5 | 0.5 |

# 聚类模型

分类（class）与聚类（cluster）不同，分类是有监督学习模型，聚类属于无监督学习模型。聚类讲究使用一些算法把样本划分为n个群落。一般情况下，这种算法都需要计算欧氏距离。

欧氏距离即欧几里得距离。

$$P(x_1) - Q(x_2) : |x_1 - x_2| = \sqrt{(x_1 - x_2)^2}$$

$$P(x_1, y_1) - Q(x_2, y_2) : \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$P(x_1, y_1, z_1) - Q(x_2, y_2, z_2) : \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

用两个样本对应特征值之差的平方和之平方根，即欧氏距离，来表示这两个样本的相似性。

## K均值算法

第一步：随机选择k个样本作为k个聚类的中心，计算每个样本到各个聚类中心的欧氏距离，将该样本分配到与之距离最近的聚类中心所在的类别中。

第二步：根据第一步所得到的聚类划分，分别计算每个聚类的几何中心，将几何中心作为新的聚类中心，重复第一步，直到计算所得几何中心与聚类中心重合或接近重合为止。

**注意：**

1. 聚类数k必须事先已知。借助某些评估指标，优选最好的聚类数。
2. 聚类中心的初始选择会影响到最终聚类划分的结果。初始中心尽量选择距离较远的样本。

K均值算法相关API：

```
import sklearn.cluster as sc
# n_clusters: 聚类数
model = sc.KMeans(n_clusters=4)
# 不断调整聚类中心，知道最终聚类中心稳定则聚类完成
model.fit(x)
# 获取训练结果的聚类中心
labels = model.labels_
centers = model.cluster_centers_
pred_y = model.predict(x)
```

案例：加载multiple3.txt，基于K均值算法完成样本的聚类。

```
import numpy as np
import sklearn.cluster as sc
import matplotlib.pyplot as mp
x = np.loadtxt('../data/multiple3.txt', delimiter=',')
# K均值聚类器
model = sc.KMeans(n_clusters=4)
model.fit(x)
centers = model.cluster_centers_
n = 500
l, r = x[:, 0].min() - 1, x[:, 0].max() + 1
b, t = x[:, 1].min() - 1, x[:, 1].max() + 1
grid_x = np.meshgrid(np.linspace(l, r, n),
                     np.linspace(b, t, n))
flat_x = np.column_stack((grid_x[0].ravel(), grid_x[1].ravel()))
flat_y = model.predict(flat_x)
grid_y = flat_y.reshape(grid_x[0].shape)
pred_y = model.predict(x)
mp.figure('K-Means Cluster', facecolor='lightgray')
mp.title('K-Means Cluster', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x[0], grid_x[1], grid_y, cmap='gray')
mp.scatter(x[:, 0], x[:, 1], c=pred_y, cmap='brg', s=80)
mp.scatter(centers[:, 0], centers[:, 1], marker='+', c='gold', s=1000,
linewidth=1)
mp.show()
```

## 均值漂移算法

首先假定样本空间中的每个聚类均服从某种已知的概率分布规则，然后用不同的概率密度函数拟合样本中的统计直方图，不断移动密度函数的中心(均值)的位置，直到获得最佳拟合效果为止。这些概率密度函数的峰值点就是聚类的中心，再根据每个样本距离各个中心的距离，选择最近聚类中心所属的类别作为该样本的类别。

均值漂移算法的特点：

1. 聚类数不必事先已知，算法会自动识别出统计直方图的中心数量。
2. 聚类中心不依据于最初假定，聚类划分的结果相对稳定。
3. 样本空间应该服从某种概率分布规则，否则算法的准确性会大打折扣。

均值漂移算法相关API：

```
# 量化带宽，决定每次调整概率密度函数的步进量
# n_samples：样本数量
# quantile：量化宽度（直方图一条的宽度）
bw = sc.estimate_bandwidth(x, n_samples=len(x), quantile=0.1)
# 均值漂移聚类器
model = sc.MeanShift(bandwidth=bw, bin_seeding=True)
model.fit(x)
```

案例：加载multiple3.txt，使用均值漂移算法对样本完成聚类划分。

```
import numpy as np
```

```
import sklearn.cluster as sc
import matplotlib.pyplot as mp

x = np.loadtxt('../data/multiple3.txt', delimiter=',')
# 量化带宽，决定每次调整概率密度函数的步进量
bw = sc.estimate_bandwidth(x, n_samples=len(x), quantile=0.2)
# 均值漂移聚类器
model = sc.MeanShift(bandwidth=bw, bin_seeding=True)
model.fit(x)
centers = model.cluster_centers_
n = 500
l,  r = x[:, 0].min() - 1, x[:, 0].max() + 1
b,  t = x[:, 1].min() - 1, x[:, 1].max() + 1
grid_x = np.meshgrid(np.linspace(l, r, n),
                     np.linspace(b, t, n))
flat_x = np.column_stack((grid_x[0].ravel(), grid_x[1].ravel()))
flat_y = model.predict(flat_x)
grid_y = flat_y.reshape(grid_x[0].shape)
pred_y = model.predict(x)
mp.figure('Mean Shift Cluster', facecolor='lightgray')
mp.title('Mean Shift Cluster', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
mp.pcolormesh(grid_x[0], grid_x[1], grid_y, cmap='gray')
mp.scatter(x[:, 0], x[:, 1], c=pred_y, cmap='brg', s=80)
mp.scatter(centers[:, 0], centers[:, 1], marker='+', c='gold', s=1000,
linewidth=1)
mp.show()
```

## 轮廓系数

好的聚类：内密外疏，同一个聚类内部的样本要足够密集，不同聚类之间样本要足够疏远。

轮廓系数计算规则：针对样本空间中的一个特定样本，计算它与所在聚类其它样本的平均距离a，以及该样本与距离最近的另一个聚类中所有样本的平均距离b，该样本的轮廓系数为(b-a)/max(a, b)，将整个样本空间中所有样本的轮廓系数取算数平均值，作为聚类划分的性能指标s。

轮廓系数的区间为：[-1, 1]。-1代表分类效果差，1代表分类效果好。0代表聚类重叠，没有很好的划分聚类。

轮廓系数相关API：

```
import sklearn.metrics as sm
# v：平均轮廓系数
# metric：距离算法：使用欧几里得距离(euclidean)
v = sm.silhouette_score(输入集, 输出集, sample_size=样本数, metric=距离算法)
```

案例：输出KMeans算法聚类划分后的轮廓系数。

```
# 打印平均轮廓系数
print(sm.silhouette_score( x, pred_y, sample_size=len(x),
metric='euclidean'))
```

## DBSCAN算法

从样本空间中任意选择一个样本，以事先给定的半径做圆，凡被该圆圈中的样本都视为与该样本处于相同的聚类，以这些被圈中的样本为圆心继续做圆，重复以上过程，不断扩大被圈中样本的规模，直到再也没有新的样本加入为止，至此即得到一个聚类。于剩余样本中，重复以上过程，直到耗尽样本空间中的所有样本为止。

DBSCAN算法的特点：

1. 事先给定的半径会影响最后的聚类效果，可以借助轮廓系数选择较优的方案。
2. 根据聚类的形成过程，把样本细分为以下三类：

   外周样本：被其它样本聚集到某个聚类中，但无法再引入新样本的样本。

   孤立样本：聚类中的样本数低于所设定的下限，则不称其为聚类，反之称其为孤立样本。

   核心样本：除了外周样本和孤立样本以外的样本。

DBSCAN聚类算法相关API：

```
# DBSCAN聚类器
# eps：半径
# min_samples：聚类样本数的下限，若低于该数值，则称为孤立样本
model = sc.DBSCAN(eps=epsilon, min_samples=5)
model.fit(x)
```

案例：修改凝聚层次聚类案例，基于DBSCAN聚类算法进行聚类划分，选择最优半径。

```
import numpy as np
import sklearn.cluster as sc
import sklearn.metrics as sm
import matplotlib.pyplot as mp

x = np.loadtxt('../data/perf.txt', delimiter=',')
epsilons, scores, models = np.linspace(0.3, 1.2, 10), [], []
for epsilon in epsilons:
    # DBSCAN聚类器
    model = sc.DBSCAN(eps=epsilon, min_samples=5)
    model.fit(x)
    score = sm.silhouette_score(
        x, model.labels_, sample_size=len(x), metric='euclidean')
    scores.append(score)
    models.append(model)
scores = np.array(scores)
best_index = scores.argmax()
best_epsilon = epsilons[best_index]
print(best_epsilon)
best_score = scores[best_index]
print(best_score)
best_model = models[best_index]
```

案例：获取核心样本、外周样本、孤立样本。并且使用不同的点型绘图。

```
best_model = models[best_index]
pred_y = best_model.fit_predict(x)
core_mask = np.zeros(len(x), dtype=bool)
core_mask[best_model.core_sample_indices_] = True
offset_mask = best_model.labels_ == -1
periphery_mask = ~(core_mask | offset_mask)
mp.figure('DBSCAN Cluster', facecolor='lightgray')
```

```
mp.title('DBSCAN Cluster', fontsize=20)
mp.xlabel('x', fontsize=14)
mp.ylabel('y', fontsize=14)
mp.tick_params(labelsize=10)
labels = best_model.labels_
mp.scatter(x[core_mask][:, 0], x[core_mask][:, 1], c=labels[core_mask],
           cmap='brg', s=80, label='Core')
mp.scatter(x[periphery_mask][:, 0], x[periphery_mask][:, 1], alpha=0.5,
           c=labels[periphery_mask], cmap='brg', marker='s', s=80,
label='Periphery')
mp.scatter(x[offset_mask][:, 0], x[offset_mask][:, 1],
           c=labels[offset_mask], cmap='brg', marker='x', s=80,
label='Offset')
mp.legend()
mp.show()
```

# 推荐引擎

推荐引擎意在把最需要的推荐给用户。

在不同的机器学习场景中通常需要分析相似样本。而统计相似样本的方式可以基于欧氏距离分数，也可基于皮氏距离分数。

**欧氏距离分数**

$$\text{欧氏距离分数} = \frac{1}{1 + \text{欧氏距离}}$$

计算所得欧氏距离分数区间处于：[0, 1]，越趋于0样本间的欧氏距离越远，样本越不相似；越趋于1，样本间的欧氏距离越近，越相似。

构建样本之间的欧氏距离得分矩阵：

$$\begin{bmatrix} & a & b & c & d & .. \\ a & 1 & 0.2 & 0.3 & 0.4 & .. \\ b & 0.2 & 1 & x & x & .. \\ c & 0.3 & x & 1 & x & .. \\ d & 0.4 & x & x & 1 & .. \\ .. & .. & .. & .. & .. & .. \end{bmatrix}$$

案例：解析ratings.json，根据每个用户对已观看电影的打分计算样本间的欧氏距离，输出欧氏距离得分矩阵。

```
import json
import numpy as np

with open('../data/ratings.json', 'r') as f:
    ratings = json.loads(f.read())
users, scmat = list(ratings.keys()), []
for user1 in users:
    scrow = []
    for user2 in users:
        movies = set()
        for movie in ratings[user1]:
            if movie in ratings[user2]:
                movies.add(movie)
        if len(movies) == 0:
```

```
            score = 0
        else:
            x, y = [], []
            for movie in movies:
                x.append(ratings[user1][movie])
                y.append(ratings[user2][movie])
            x = np.array(x)
            y = np.array(y)
            score = 1 / (1 + np.sqrt(((x - y) ** 2).sum()))
        scrow.append(score)
    scmat.append(scrow)
users = np.array(users)
scmat = np.array(scmat)
for scrow in scmat:
    print('  '.join('{:.2f}'.format(score) for score in scrow))
```

**皮尔逊相关系数**

```
A = [1,2,3,1,2]
B = [3,4,5,3,4]
m = np.corrcoef(A, B)

```

皮尔逊相关系数 = 协方差 / 标准差之积

相关系数处于[-1, 1]区间。越靠近-1代表两组样本反相关，越靠近1代表两组样本正相关。

案例：使用皮尔逊相关系数计算两用户对一组电影评分的相关性。

```
score = np.corrcoef(x, y)[0, 1]
```

**按照相似度从高到低排列每个用户的相似用户**

```
# scmat矩阵中每一行为 每一个用户对所有用户的皮尔逊相关系数
for i, user in enumerate(users):
    # 拿到所有相似用户与相似用户所对应的皮尔逊相关系数
    sorted_indices = scmat[i].argsort()[::-1]
    sorted_indices = sorted_indices[sorted_indices != i]
    similar_users = users[sorted_indices]
    similar_scores = scmat[i, sorted_indices]
    print(user, similar_users, similar_scores, sep='\n')
```

**生成推荐清单**

1. 找到所有皮尔逊系数正相关的用户
2. 遍历当前用户的每个相似用户，拿到相似用户看过但是当前用户没有看过的电影作为推荐电影
3. 多个相似用户有可能推荐同一部电影，则取每个相似用户对该电影的评分得均值作为推荐度。
4. 可以把相似用户的皮尔逊系数作为权重，皮尔逊系数越大，推荐度越高。

```
    # 找到所有皮尔逊系数正相关的用户
    positive_mask = similar_scores > 0
    similar_users = similar_users[positive_mask]
    # 相似用户对应的皮尔逊相关系数
    similar_scores = similar_scores[positive_mask]
    #存储对于当前用户所推荐的电影以及电影的推荐度(推荐电影的平均分)
    recomm_movies = {}
```

```
    #遍历当前用户的每个相似用户
    for i, similar_user in enumerate(similar_users):
        #拿到相似用户看过但是当前用户没有看过的电影
        for movie, score in ratings[similar_user].items():
            if (movie not in ratings[user].keys()):
                if movie not in recomm_movies:
                    recomm_movies[movie] = []
                else:
                    recomm_movies[movie].append(score)

    print(user)
    movie_list = sorted(recomm_movies.items(), key=lambda
x:np.average(x[1]), reverse=True)
    print(movie_list)
```

##