本次实验以AAAI 2014会议论文数据为基础，要求实现或调用无监督聚类算法，了解聚类方法。

任务介绍

每年国际上召开的大大小小学术会议不计其数，发表了非常多的论文。在计算机领域的一些大型学术会议上，一次就可以发表涉及各个方向的几百篇论文。按论文的主题、内容进行聚类，有助于人们高效地查找和获得所需要的论文。本案例数据来源于AAAI 2014上发表的约400篇文章，由UCI公开提供，提供包括标题、作者、关键词、摘要在内的信息，希望大家能根据这些信息，合理地构造特征向量来表示这些论文，并设计实现或调用聚类算法对论文进行聚类。最后也可以对聚类结果进行观察，看每一类都是什么样的论文，是否有一些主题。

基本要求：

1. 将文本转化为向量，实现或调用无监督聚类算法，对论文聚类，例如10类（可使用已有工具包例如sklearn）；
2. 观察每一类中的论文，调整算法使结果较为合理；
3. 无监督聚类没有标签，效果较难评价，因此没有硬性指标，跑通即可，主要让大家了解和感受聚类算法，比较简单。

扩展要求：

1. 对文本向量进行降维，并将聚类结果可视化成散点图。

注：group和topic也不能完全算是标签，因为

1. 有些文章作者投稿时可能会选择某个group/topic但实际和另外group/topic也相关甚至更相关；
2. 一篇文章可能有多个group和topic，作为标签会出现有的文章同属多个类别，这里暂不考虑这样的聚类；
3. group和topic的取值很多，但聚类常常希望指定聚合成出例如5/10/20类；
4. 感兴趣但同学可以思考利用group和topic信息来量化评价无监督聚类结果，不作要求。

提示：

1. 高维向量的降维旨在去除一些高相关性的特征维度，保留最有用的信息，用更低维的向量表示高维数据，常用的方法有PCA和t-SNE等；
2. 降维与聚类是两件不同的事情，聚类实际上在降维前的高维向量和降维后的低维向量上都可以进行，结果也可能截然不同；
3. 高维向量做聚类，降维可视化后若有同一类的点不在一起，是正常的。在高维空间中它们可能是在一起的，降维后损失了一些信息。

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from sklearn.decomposition import PCA
from matplotlib.colors import cnames
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import collections
```

数据集加载：去掉含有空值的数据。

In [2]:
```
def load_data(path='.\data\[UCI] AAAI-14 Accepted Papers - Papers.csv'):
    df_data = pd.read_csv(path)
    df_data = df_data.dropna()
    return df_data

df = load_data()
df.describe()
```

Out[2]:

	title	authors	groups	keywords	topics	abstract
count	392	392	392	392	392	392
unique	392	387	125	392	348	392
top	Efficient Object Detection via Adaptive Online...	Fan Xie, Martin Mueller and Robert Holte	Novel Machine Learning Algorithms (NMLA)	Scene layout\nScene understanding\nAcoustic ec...	GTEP: Social Choice / Voting	In this paper, we focus on the Sparse Learning...
freq	1	2	49	1	7	1

特征提取：

- 因为聚类算法对噪点较敏感，考虑到group/topic的不客观性，将它们作为评价指标而不是特征是一种更好的选择。
- 使用词袋模型对每个特征进行抽象。

In [22]:
```
# 去除不客观的特征
df_fea = df.drop(['groups', 'topics'],axis=1)

cv = CountVectorizer(min_df=0.0, max_df=1.0)
X_fea = [[i] for i in range(df.shape[0])]
for col in df_fea.columns:
    tmp = cv.fit_transform(df_fea[col])
    # 每个feature是【样本数×词袋大小】的
    feature = tmp.toarray()
    X_fea = np.concatenate((X_fea, feature), axis=1)

X_fea.shape
```

Out[22]: (392, 9899)

聚类的评估方法有以下两种：

- SC系数：针对单个样本而言。结合了聚类的凝聚度（Cohesion）和分离度（Separation），用于评估聚类的效果。
- CH系数：针对整体结果而言。类别内部数据的协方差越小越好，类别之间的协方差越大越好。

这里通过计算得分方法，参照评价指标（'keywords', 'groups', 'topics'），返回SC系数和CH系数。

In [4]:
```
# 取出最能代表文章主题的三个特征作为后面的评价指标
df_cls = df[['keywords', 'groups', 'topics']]
cv = CountVectorizer(min_df=0.2, max_df=1.0)
X_cls = [[i] for i in range(df.shape[0])]
for col in df_cls.columns:
```

```
    tmp = cv.fit_transform(df_cls[col])
    feature = tmp.toarray()
    X_cls = np.concatenate((X_cls, feature), axis=1)

def cal_score(result_list):
    global X_cls
    sh = silhouette_score(X_cls, result_list)
    ch = calinski_harabasz_score(X_cls, result_list)
    return sh,ch
```

降维：

- PCA：通过矩阵运算的方法提取最主要的特征，节省效率。
- t-SNE：t-Distributed Stochastic Neighbor Embedding是一种降维技术，用于在二维或三维的低维空间中表示高维数据集，从而使其可视化。与其他降维算法(如PCA)相比，t-SNE创建了一个缩小的特征空间，相似的样本由附近的点建模，不相似的样本由高概率的远点建模。便于可视化。

这里使用PCA提取主成分进行降维，对降维后的数据进行聚类，遍历获得表现最好的n_components和k值。

In [5]:
```
rec = collections.defaultdict(list)
for _cmp in [2, 5, 10, 50, 100]:
    X_pca = PCA(n_components=_cmp).fit_transform(X_fea)
    print(X_pca.shape)
    for _k in range(5, 16):
        model = KMeans(n_clusters=_k)
        res = model.fit_predict(X_pca)
        sh,ch = cal_score(res)
        print('k:',_k, 'sh:', sh, 'ch:', ch, 'score:', sh*ch)
        rec[_cmp].append(sh*ch)
```

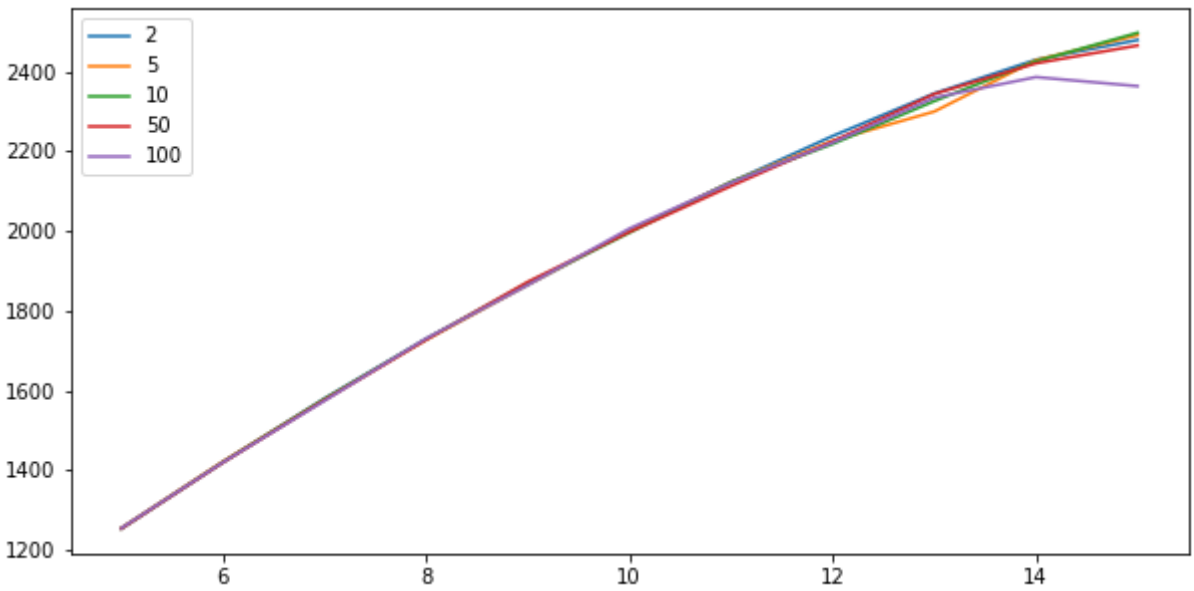
```
(392, 2)
k: 5 sh: 0.5478712999412111 ch: 2287.742371629651 score: 1253.3883870753261
k: 6 sh: 0.5368258953851707 ch: 2645.1587919239505 score: 1419.989736910531
k: 7 sh: 0.5278357569962739 ch: 2992.9492542389025 score: 1579.7856352626247
k: 8 sh: 0.5195710602057096 ch: 3327.143637313055 score: 1728.687547095425
k: 9 sh: 0.512250312677174 ch: 3639.886321930314 score: 1864.5329065181722
k: 10 sh: 0.5050676973059094 ch: 3958.8891396499357 score: 1999.507021652366
k: 11 sh: 0.49914366118535947 ch: 4251.255294684053 score: 2121.9871324222427
k: 12 sh: 0.49191215499742835 ch: 4549.344645421493 score: 2237.877928355298
k: 13 sh: 0.4869146646177637 ch: 4815.040000952501 score: 2344.513587184904
k: 14 sh: 0.47938969900166184 ch: 5067.132308077214 score: 2429.1310319707313
k: 15 sh: 0.47298444084305397 ch: 5240.433617547639 score: 2478.643564370913
(392, 5)
k: 5 sh: 0.5481846758073398 ch: 2288.1708261055273 score: 1254.3401825004712
k: 6 sh: 0.5373779899032458 ch: 2645.2109489855197 score: 1421.478142635896
k: 7 sh: 0.5274896154065446 ch: 2990.8006851038585 score: 1577.6163031430644
k: 8 sh: 0.5191529982729817 ch: 3326.0863625553184 score: 1726.7477076354692
k: 9 sh: 0.5121177788335297 ch: 3641.333825570189 score: 1864.7917907424048
k: 10 sh: 0.5049417173262873 ch: 3960.367021740197 score: 1999.754525199889
k: 11 sh: 0.49830892928990966 ch: 4258.768931731116 score: 2122.182586464065
k: 12 sh: 0.4918339078534414 ch: 4526.020867789286 score: 2226.0505304310286
k: 13 sh: 0.4848991477566246 ch: 4741.7164387376615 score: 2299.2542600474694
k: 14 sh: 0.4798838709333443 ch: 5059.789375228313 score: 2428.1113114919704
k: 15 sh: 0.47356069256027 ch: 5258.834507193574 score: 2490.377311286435
(392, 10)
k: 5 sh: 0.5482075161487522 ch: 2286.3172638426536 score: 1253.3763083391925
k: 6 sh: 0.5368621752748082 ch: 2644.597424712171 score: 1419.7843261571318
k: 7 sh: 0.527429238512545 ch: 2991.505425078837 score: 1577.8074283554781
k: 8 sh: 0.5195976004110082 ch: 3327.0219797672016 score: 1728.7126372017199
k: 9 sh: 0.5128339652192804 ch: 3645.2661357338125 score: 1869.4162866679346
k: 10 sh: 0.5047604987149283 ch: 3953.1318227452866 score: 1995.3847903347646
```

```
k: 11 sh: 0.49883612715943215 ch: 4255.018529537222 score: 2122.55696426597
k: 12 sh: 0.4921130496154944 ch: 4506.893962660573 score: 2217.9013322585547
k: 13 sh: 0.4849625827271093 ch: 4795.176181494262 score: 2325.481025608975
k: 14 sh: 0.47934131512736866 ch: 5059.434351978761 score: 2425.1959160780852
k: 15 sh: 0.47330387502339355 ch: 5274.818483296506 score: 2496.592028189256
(392, 50)
k: 5 sh: 0.5478196034050973 ch: 2287.2795411697407 score: 1253.0165711202003
k: 6 sh: 0.5368168098651356 ch: 2642.796669198256 score: 1418.697677081214
k: 7 sh: 0.5272158588547925 ch: 2988.3262072811854 score: 1575.4929679100348
k: 8 sh: 0.5192163290833933 ch: 3326.2388015295405 score: 1727.0375001849138
k: 9 sh: 0.5133483065225476 ch: 3646.5769539993557 score: 1871.9641039397193
k: 10 sh: 0.5049165906891163 ch: 3955.3563926187967 score: 1997.1250647214847
k: 11 sh: 0.49899449704984583 ch: 4233.351639662873 score: 2112.4191722687156
k: 12 sh: 0.49196031659766676 ch: 4521.353303183082 score: 2224.3264024838554
k: 13 sh: 0.4865187559404428 ch: 4816.508081561428 score: 2343.321519818355
k: 14 sh: 0.4793597565217076 ch: 5048.697002529433 score: 2420.142165884384
k: 15 sh: 0.47222883587130043 ch: 5219.4972013211855 score: 2464.797087213414
(392, 100)
k: 5 sh: 0.5482632996873429 ch: 2287.71507830427 score: 1254.270217575587
k: 6 sh: 0.5369792471846009 ch: 2643.4329083513876 score: 1419.4686131095282
k: 7 sh: 0.5270146911630668 ch: 2986.5464908230215 score: 1573.9538765052355
k: 8 sh: 0.5196621944851698 ch: 3329.2575310432753 score: 1730.0892745882268
k: 9 sh: 0.5119152543943571 ch: 3641.603470915736 score: 1864.1923672172031
k: 10 sh: 0.5056190636501388 ch: 3966.6633424908346 score: 2005.6206050455455
k: 11 sh: 0.49879141934572147 ch: 4251.706027697585 score: 2120.714484196038
k: 12 sh: 0.4912375950276207 ch: 4521.713845262594 score: 2221.2358347498916
k: 13 sh: 0.4861802405181139 ch: 4800.608950586835 score: 2333.961214229718
k: 14 sh: 0.4799061341424035 ch: 4971.123370368506 score: 2385.672599018505
k: 15 sh: 0.47233481484840517 ch: 5002.688825309167 score: 2362.9441000465913
```

将以上结果可视化。

In [6]:
```
fig = plt.figure(figsize=(10, 5))
x = [_ for _ in range(5,16)]
for _k in rec:
    plt.plot(x,rec[_k],label=_k)
plt.legend()
```

Out[6]: <matplotlib.legend.Legend at 0x1876caf9d90>



发现主成分的个数对聚类效果影响不大，提取特征为5维，聚类为15类时，效果较好。

In [77]:
```
X_pca = PCA(n_components=5).fit_transform(X_fea)
model = KMeans(n_clusters=15)
res = model.fit_predict(X_pca)
```

下面通过映射至3维，直观展示聚类效果。分别使用PCA和TSNE两种方法。

In [78]:
```
X_pca = PCA(n_components=3).fit_transform(X_pca)
X_pca = np.hstack((X_pca, [[r] for r in res]))
X_tsne = TSNE(n_components=3).fit_transform(X_pca)
X_tsne = np.hstack((X_tsne, [[r] for r in res]))
```

In [79]:
```
fig = plt.figure(figsize=(14, 6))
ax = fig.add_subplot(121, projection='3d')
ax.scatter(X_tsne[:, 0], X_tsne[:, 1], X_tsne[:, 2], c=X_tsne[:, 3])
ax = fig.add_subplot(122, projection='3d')
ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:,2],c =X_pca[:,3])
plt.show()
```