國立雲林科技大學

資訊管理系所

資料探勘專案作業報告

學生/學號：

廖帝登 D10823001

張　靖 M10823001

劉博茜 M10823015

盧廼安 M10823023

指導教授：許中川

中華民國 108 年 12 月

摘要

本研究針對網路提供之線上資料集，進行分群的訓練和績效的預測，隨著大量資料的快速累積以及演算法與雲端運算的發展，大數據分析已經成為學術界與各行業關注與學習的焦點。從巨量資料中篩選有用的資訊，其運用的方法為資料探勘。本研究將使用 Python 撰寫 K-means、階層式分群和 DBSCAN 之程式進行模型訓練，分別使用 K-means、階層式分群、DBSCAN，將資料分成 20 群，並比較分群所花費時間，自選評估指標比較分群結果品質，包括使用 Purity 指標。K-means 所花費的時間約莫 60 秒，績效純度為 0.263；階層式分群所花費的時間約莫 108 秒，績效純度為 0.204；DBSCAN 所花費的時間約莫 40 秒，績效純度為 0.09。可以得知 K 均值可以用於稀疏的高維資料，如文檔資料。DBSCAN 通常在這類資料上的性能很差，因為對於高維資料，傳統的歐幾里得密度定義不能很好處理它們。

關鍵字：資料探勘、Python、K-means、階層式分群、DBSCAN

目錄

第1章 緒論

1.1 研究背景與動機

隨著資訊科技發展下，資料的處理與分析變得更加重要，資料探勘則是現今挖掘資訊最重要的一項技術。分群(Clustering)隸屬於非監督式學習，其主要根據資料本身的特性和特徵，來進行資料分析的一種方法。實務上當我們對資料還沒有深入了解時，便可以先使用分群的方法，觀察潛藏在資料中的特性，再擬定後續分析的手法。當然在使用分群方法時，不同的分群數目和一些參數的設定，往往會對最後的結果有著巨大的影響。

本研究將 20 Newsgroups 作為本次訓練和測試的資料集，20 Newsgroups 的資料集裡邊約有 2,000 Newsgroup documents，並且平均分散在 20 different newsgroups。本研究將使用 Python 撰寫 K-means、階層式分群和 DBSCAN 之程式進行模型訓練，分別使用 K-means、階層式分群、DBSCAN，將資料分成 20 群，並比較分群所花費時間，自選評估指標比較分群結果品質，包括使用 Purity 指標。

1.2 研究目的

本研究將使用 Python 撰寫 K-means、階層式分群和 DBSCAN 之程式進行模型訓練，分別使用 K-means、階層式分群、DBSCAN，將資料分成 20 群，並比較分群所花費時間，自選評估指標比較分群結果品質，包括使用 Purity 指標。

第2章 方法

2.1 程式架構- K-means

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import cluster, metrics
import collections
import os
import time

path = "D:/Python/DN_report3/mini_newsgroups"  # 文件目錄
files = os.listdir(path)  # 得到文件夾下的所有文件名稱
X = []
Y = []

for file in files:  # 尋訪文件夾
    newpath = path + "/" + file
    files2 = os.listdir(newpath)
    for sub in files2:
        if not os.path.isdir(sub):  # 判斷是否是文件夾，不是文件夾打開
            f = open(newpath + "/" + sub,encoding="cp1252")  # 打開文件
            iter_f = iter(f)  # 創建迭代
            str = ""
            for line in iter_f:  # 尋訪文件，一行行尋訪，讀取文本
                str = str + line
            Y.append(file)
            X.append(str)  # 每個文件的文本存到 list 中

# 詞頻向量化
vectorizer = CountVectorizer(min_df=1)
X_counts = vectorizer.fit_transform(X)

# 進行 TF-IDF 預處理
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X_counts)
```

```
# 計算純度函式
def purity(result, label):
    total_num = len(label)
    cluster_counter = collections.Counter(result)
    original_counter = collections.Counter(label)

    t = []
    for k in cluster_counter:
        p_k = []
        for j in original_counter:
            count = 0
            for i in range(len(result)):
                if result[i] == k and label[i] == j:  # 求交集
                    count += 1
            p_k.append(count)
        temp_t = max(p_k)
        t.append(temp_t)

    return sum(t) / total_num

# K-means
start_time = time.time()
kmeans = cluster.KMeans( n_clusters= 20 ,random_state=0).fit(tfidf)
cluster_label=kmeans.labels_
end_time = time.time()
cost_time = end_time - start_time
print("The K-means cost time is : ",format(cost_time,'5.3f')," seconds.")

# 績效評估
silhouette_avg = metrics.silhouette_score(tfidf, cluster_label)
print("Silhouette Coefficient : ",silhouette_avg)
print("Purity : ",purity(cluster_label,Y))

print("All Finish!")
```

2.2 程式架構-階層式分群

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import os
```

```
from sklearn import cluster
from sklearn import metrics
import collections
from scipy.cluster.hierarchy import linkage,dendrogram
import matplotlib.pyplot as plt
import time

path = "D:/Python/DN_report3/mini_newsgroups"  # 文件目錄
files = os.listdir(path)  # 得到文件夾下的所有文件名稱
X = []
Y = []

for file in files:  # 尋訪文件夾
    newpath = path + "/" + file
    files2 = os.listdir(newpath)
    for sub in files2:
        if not os.path.isdir(sub):  # 判斷是否是文件夾，不是文件夾打開
            f = open(newpath + "/" + sub,encoding="cp1252")  # 打開文件
            iter_f = iter(f)  # 創建迭代
            str = ""
            for line in iter_f:  # 尋訪文件，一行行尋訪，讀取文本
                str = str + line
            Y.append(file)
            X.append(str)  # 每個文件的文本存到 list 中

# 詞頻向量化
vectorizer = CountVectorizer(min_df=1)
X_counts = vectorizer.fit_transform(X)

# 進行 TF-IDF 預處理
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X_counts)

# 計算純度函式
def purity(result, label):
    total_num = len(label)
    cluster_counter = collections.Counter(result)
    original_counter = collections.Counter(label)
```

```
    t = []
    for k in cluster_counter:
        p_k = []
        for j in original_counter:
            count = 0
            for i in range(len(result)):
                if result[i] == k and label[i] == j:  # 求交集
                    count += 1
            p_k.append(count)
        temp_t = max(p_k)
        t.append(temp_t)

    return sum(t) / total_num

# Hierarchical
start_time = time.time()
hclust = cluster.AgglomerativeClustering(linkage = 'ward', affinity = 'euclidean',
n_clusters = 20)
hclust.fit(tfidf.toarray())
cluster_label = hclust.labels_
end_time = time.time()
cost_time = end_time - start_time
print("The Hierarchical cost time is : ",format(cost_time,'5.3f')," seconds.")
# 績效評估
silhouette_avg = metrics.silhouette_score(tfidf.toarray(), cluster_label)
print("Silhouette Coefficient : ",silhouette_avg)
print("Purity : ",purity(cluster_label,Y))

# 繪圖
Z = linkage(tfidf.toarray(), method ='ward',metric='euclidean')
p = dendrogram(Z,truncate_mode='lastp',labels=tfidf.toarray())
plt.show()

print("All Finish!")
```

2.3 程式架構-DBSCAN

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics import homogeneity_score as h_s,silhouette_score as s_s
```

```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from sklearn import cluster, metrics
import collections
import os
import time

path = "D:/Python/DN_report3/mini_newsgroups"  # 文件目錄
files = os.listdir(path)  # 得到文件夾下的所有文件名稱
X = []
Y = []

for file in files:  # 尋訪文件夾
    newpath = path + "/" + file
    files2 = os.listdir(newpath)
    for sub in files2:
        if not os.path.isdir(sub):  # 判斷是否是文件夾，不是文件夾打開
            f = open(newpath + "/" + sub,encoding="cp1252")  # 打開文件
            iter_f = iter(f)  # 創建迭代
            str = ""
            for line in iter_f:  # 尋訪文件，一行行尋訪，讀取文本
                str = str + line
            Y.append(file)
            X.append(str)  # 每個文件的文本存到 list 中

# 詞頻向量化
vectorizer = CountVectorizer(min_df=1)
X_counts = vectorizer.fit_transform(X)

# 進行 TF-IDF 預處理
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X_counts)

# 計算純度
def purity(result, label):
    total_num = len(label)
    cluster_counter = collections.Counter(result)
    original_counter = collections.Counter(label)
```

```
        t = []
        for k in cluster_counter:
            p_k = []
            for j in original_counter:
                count = 0
                for i in range(len(result)):
                    if result[i] == k and label[i] == j:    # 求交集
                        count += 1
                p_k.append(count)
            temp_t = max(p_k)
            t.append(temp_t)

        return sum(t) / total_num

# DBSCAN
print("DBSCAN")
start_time = time.time()
for s in range(1,5):
        h = []
        print('min_samples',s)

for e in range(1,101):
        e=e/100
        dbscan = DBSCAN(eps=e, min_samples= s)
        dbscan.fit(tfidf)
        cluster_label = dbscan.labels_
        h.append(h_s(Y,dbscan.labels_))
end_time = time.time()
cost_time = end_time - start_time
print("The DBSCAN cost time is : ",format(cost_time,'5.3f')," seconds.")
silhouette_avg = metrics.silhouette_score(tfidf, cluster_label)
print("Silhouette Coefficient : ",silhouette_avg)
print("Purity : ",purity(cluster_label,Y))
plt.plot(h)
plt.show()

print("All Finish!")
```

第3章 實驗

3.1 資料集

表格 1 20 Newsgroups 資料集

alt.atheism	100
comp.graphics	100
comp.os.ms-windows.misc	100
comp.sys.ibm.pc.hardware	100
comp.sys.mac.hardware	100
comp.windows.x	100
misc.forsale	100
rec.autos	100
rec.motorcycles	100
rec.sport.baseball	100
rec.sport.hockey	100
sci.crypt	100
sci.electronics	100
sci.med	100
sci.space	100
soc.religion.christian	100
talk.politics.guns	100
talk.politics.mideast	100
talk.politics.misc	100
talk.religion.misc	100
資料總筆數 ：2000	

3.2 實驗設計

利用 python 進行實驗，透過原始數據(training data)進行 K-means、階層式分群和 DBSCAN 的訓練，藉此建立該數據的分類模型，透過測試數據(testing data)進行結果的預測分群，透過純度的比較來了解，三者演算法績效的好壞。

3.3 實驗結果

利用 python 進行 K-means、階層式分群和 DBSCAN 的程式撰寫，績效結果如下圖。

```
C:\Users\asus\AppData\Local\Programs\Python\Python37\python.exe D:/Python/DN_report3/K-means.py
K-means
The K-means cost time is :  59.944  seconds.
Silhouette Coefficient :  0.0018164529887103348
Purity :  0.263
All Finish!

Process finished with exit code 0
```

Figure 1 K-means

```
C:\Users\asus\AppData\Local\Programs\Python\Python37\python.exe D:/Python/DN_report3/Hierarchical.py
Hierarchical
The Hierarchical cost time is :  107.195  seconds.
Silhouette Coefficient :  -0.005577889593443225
Purity :  0.204
All Finish!

Process finished with exit code 0
```

Figure 2 階層式分群

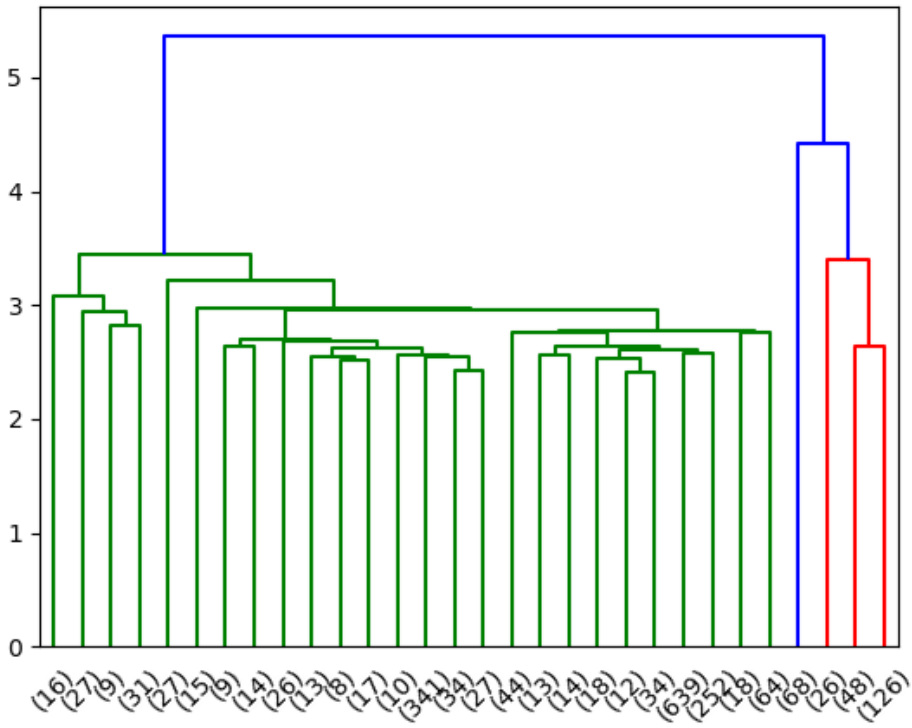

Figure 3 階層式分群視覺化

```
C:\Users\asus\AppData\Local\Programs\Python\Python37\python.exe D:/Python/DN_report3/DBSCAN.py
DBSCAN
min_samples 1
min_samples 2
min_samples 3
min_samples 4
The DBSCAN cost time is :  40.020  seconds.
Silhouette Coefficient :  -0.032306634470815394
Purity :  0.09
All Finish!
```

Figure 4 DBSCAN

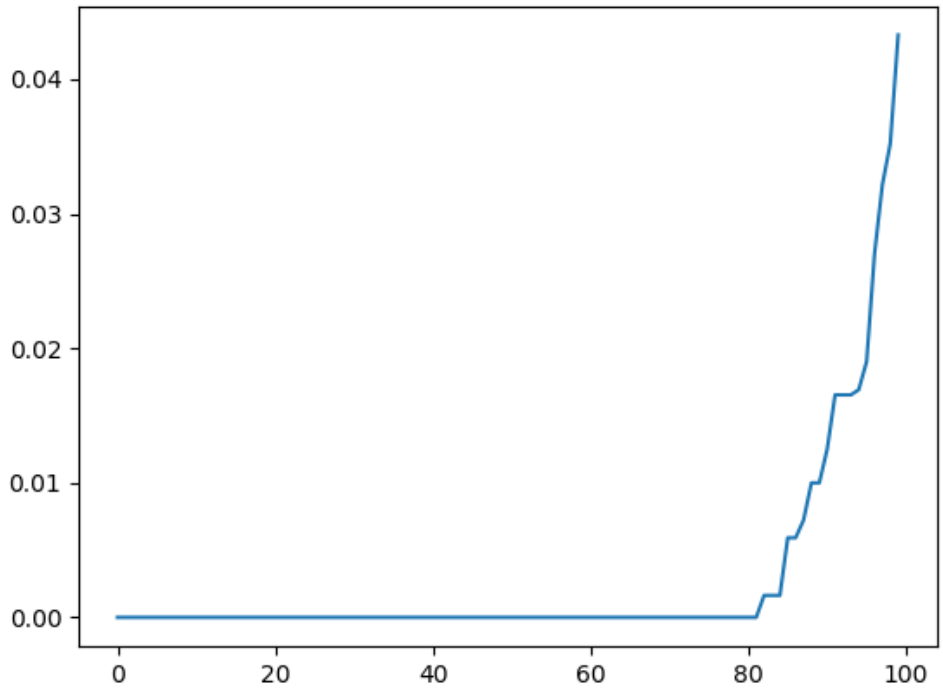

Figure 5 DBSCAN 視覺化

第4章 結論

本研究的目的是利用 python 建構 K-means、階層式分群和 DBSCAN 在 20 Newsgroups 資料集的分群，在 K-means、階層式分群和 DBSCAN 訓練之下， K-means 所花費的時間約莫 60 秒，績效純度為 0.263；階層式分群所花費的時間約莫 108 秒，績效純度為 0.204；DBSCAN 所花費的時間約莫 40 秒，績效純度為 0.09。可以得知 K 均值可以用於稀疏的高維資料，如文檔資料。DBSCAN 通常在這類資料上的性能很差，因為對於高維資料，傳統的歐幾里得密度定義不能很好處理它們。