# Final iSure Report

Keyu Han[2], Zhiyong Zhang[1],

[1]Lab for Big Data Methodology, University of Notre Dame

[2]School of Management, Jiangsu University

In this summer project, I finished the unsupervised learning methods on my own and was responsible for the data preprocessing progress and participated in the final summary writing. Our team, Xueyang Li and I, tried to use supervised and unsupervised learning methods to finish the task. The model responded the best on the random forest algorithm and scored 89.8% in model accuracy.

To operate with 1,186,913 rows chatting data and ensured the code modular, I used class and user-defined functions to decrease code redundancy. Considering that the code in jupyter was better for visualization but ran more slowly in the large dataset, I used PyCharm and Spyder. The most important part was to transform the chatting data into the text vector with several ways.

In the first step, word cloud and word frequency were generated to show the distribution of the words. Then I used TfidfVectorizor algorithm to transfer chatting data into the text matrix and used KMeans algorithm to cluster. In the second process, considering to the shortcomings of the Bag of Words, I used Word2Vec model to train word vector and KMeans and MiniBatchKMeans to cluster. In the end, to classify different sentences, I used Doc2Vec to create the sentences vector. I also trained a new hierarchical cluster algorithm called HDBSCAN to classify varied chatting types. In the evaluation step, I used caliski_harabaz_score and Adjusted Rand Index.

Through Prof. Zhang's supervision and knowledge sharing from my partner, I learned several skills.

By reading the papers and blogs, I knew the development of the short text in unsupervised learning. The most important step is to translate Chinese context into the text matrix. The basic method of word embedding is to use TF-IDF algorithm in Bag of words model. However, the Bag of Word model has the shortcomings such as sparse matrix, high vector dimension and unmeasurable relationship between words. So I decided to use neural network methods to train the text vector model.

After reading the paper called *Distributed Representations of Sentences and Documents,* I decided to use Word2Vec model, which was put forward in 2013 by Google. It can solve the problems in Bag of Words, and has faster training speed. This model, which uses an unsupervised matrix method, is used to transform normal words into a matrix before training the neural network model. By using lots of unmarked

raw data, word2Vec model can learn the relationship in the context and put out the word vector of every word.

While Word2Vector can only generate the single words, it can't dealing with the whole sentence. So I tried to use Doc2Vec which was introduced in the paper called *Distributed Representations of Sentences and Documents* in 2017. In this model, every sentence can be described as an unique vector. It created an element called Paragraph Vector, which can be treated as a word vector, realizing the memorizing function.

As I practiced the methods above, I also learned other new methods by reading papers.

There are other word embedding methods such as Fasttext, Transformer, Self-Attention, Infersent, Universal Sentence Encoder and so on. In order to dig the more details, researchers are focusing on unsupervised learning, deep learning, meta-learning, semi-supervised learning methods. New classified mothods such as Aspect-Based Sentiment Analysis and Mutilevel Model are developed to subdivide the short text data.

I'm now learning deep learning methods and try to using PyTorch to train a neural network model. In our text mining team, we shared the useful paper and our code and discussed the solutions when we faced problems. I also learn the good points of other members through the weekly meeting. It's a really precious researching experience.

# Contents

# 1 Data

## 1.1 Community User Chat Records

### 1.1.1 unmarked dataset

The raw dataset has 1186913 rows and 1 columns.

```
import pandas as pd

data = pd.read_csv("content_2020_04.csv", header=0, delimiter="\t")
data.shape
```

```
(1186913, 1)
```

```
data.loc[[0, 1, 2]]
```

| | a.to_id\|b.chatroom_name\|_morf.a\|a.talker_name\|a.content\|a.create_time |
|---|---|
| 0 | 10026117318@chatroom\|XMZ官方643群\|xsxvhmwyhadb\|瑜\|... |
| 1 | 10026117318@chatroom\|XMZ官方643群\|4t4b5ppy3fcf\|二郎... |
| 2 | 10026117318@chatroom\|XMZ官方643群\|4t4b5ppy3fcf\|二郎... |

The shape of unlabeled user chatting dataset has 1186913 rows and 6 columns

Every row means a chat from one user. The columns include "idList", "roomName", "useridList", "nameList", "chatList", "timeLists"

```
df[:5]
```

| | idList | roomName | useridList | nameList | chatList | timeList |
|---|---|---|---|---|---|---|
| 0 | 10026117318@chatroom | XMZ官方643群 | xsxvhmwyhadb | 瑜 | eth | 2020-04-01 13:33:11 |
| 1 | 10026117318@chatroom | XMZ官方643群 | 4t4b5ppy3fcf | 二郎神 | "云"修改群名为"惊鸿社区poc2群" | 2020-04-03 16:20:56 |
| 2 | 10026117318@chatroom | XMZ官方643群 | 4t4b5ppy3fcf | 二郎神 | "云"修改群名为"惊鸿社区eclc2群" | 2020-04-07 19:40:35 |
| 3 | 10026117318@chatroom | XMZ官方643群 | 4t4b5ppy3fcf | 二郎神 | "云"修改群名为"币圈群" | 2020-04-14 09:11:19 |
| 4 | 10026117318@chatroom | XMZ官方643群 | vmyzzsgwi9n4 | 云 | + | 2020-04-18 21:16:02 |

Cut the Words in every chat

```
import pandas as pd
import re
df = pd.read_csv("uidGroup.csv").dropna()

r = r'[\@\[\.\!\?\\\/\_\,\$\%\^\*\(\+\)\"\'\|\! \。\, \? \、\：\:\《\》\ “\” \ (\) \"\]  ;#\+]'  # 之前有\s+ 表示去掉空格
df["chatList"] = [re.sub(r, "", i) for i in df["chatList"]]

df.head()
```

...

```
df["chatList"] = [jieba.lcut(i) for i in df["chatList"]]
df.head(6)
```

| | idList | roomName | useridList | nameList | chatList | timeList |
|---|---|---|---|---|---|---|
| 3 | 10347966871@chatroom | odin浏览器官方□□群 | -gnehsneroahiem | 罗心宏2345-966 | [CuRLy, 👹, HaiR, 时] | 2020-04-15 22:29:26 |
| 4 | 10347966871@chatroom | odin浏览器官方□□群 | -gnehsneroahiem | 罗心宏2345-966 | [是] | 2020-04-15 22:29:29 |
| 5 | 10347966871@chatroom | odin浏览器官方□□群 | -gnehsneroahiem | 罗心宏2345-966 | [聊天记录] | 2020-04-28 18:40:13 |
| 13 | 10440271134@chatroom | Fowin量化交易 | 0047739qq | ~佛系少年 | [总体, 还是, 亏, 的, 捂, 脸, 捂, 脸] | 2020-04-28 17:55:28 |
| 14 | 10440271134@chatroom | Fowin量化交易 | 0047739qq | ~佛系少年 | [陈, 大区, , 你, 清仓, 试试, 就, 知道, 了] | 2020-04-28 18:25:00 |
| 15 | 10347966871@chatroom | odin浏览器官方□□群 | 0056izgnep | 鹏 | [这是, 什么, 项目] | 2020-04-15 16:52:05 |

## 1.1.2 marked dataset

The shape of marked data is (10022, 3)

```
data = df.copy().dropna()
print(data.shape)
data.head()
```

(10022, 8)

| | Unnamed: 0 | a.to_id | chatroom_name | talker_id | talker_name | content | create_time | type |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | 人生，靠的不是时间，靠的是珍惜，人生的路到底有多远并不重要，当务之急是珍惜生命的每一秒 | 2020-04-29 14:26:11 | 1 |
| 1 | 1 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | 成功永远属于善于把握机会的人 | 2020-04-29 14:26:11 | 1 |
| 2 | 2 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | 行动起来吧，机会不等人 | 2020-04-29 14:26:12 | 1 |
| 3 | 3 | 10026117318@chatroom | 云朋友币圈群 | wxid_mkhghjwwdn1g22 | 海暖 | 肯定赚钱 | 2020-04-29 16:50:17 | 0 |
| 4 | 4 | 10026117318@chatroom | 云朋友币圈群 | wxid_mkhghjwwdn1g22 | 海暖 | 加仓搞起来就对了 | 2020-04-29 16:50:17 | 1 |

## 1.1.3 Order the marked data

I use groupby() to aggregate the "content" columns from the same user.

Concat() is used to connect talker_id", "cotent" and "type" columns.

```
groupid = data.groupby("talker_id")
newData = groupid["content"].apply(list)
typeList = groupid["type"].apply(list)

newData = pd.DataFrame(data=newData, columns=["content"])
typeListMid = pd.DataFrame(data=typeList, columns=["type"])

croData = pd.concat([newData, typeListMid], axis=1)

croData.to_csv("cutLabelChat2572.csv")
#croData.to_csv("labelChat2572.csv")
croData
```

| talker_id | content | type |
|---|---|---|
| ABBY415921916 | [[全世界, 都, 在, 看, 多], [币, 今天, 很骚], [大饼, 很久没, 这么,... | [1, 1, 1] |
| CRufus | [[既然, 上, 了, BHD, 的, 贼船, 就, 做, 一只, 快乐, 的, 海鸟], ... | [1, 1, 1, 1, 0, 0, 0, 1, 1] |
| Changfu135 | [[, 微笑, ], 下午, 好], [赚钱, 去, 了, [, 呲, 牙, ]], [5... | [0, 0, 0, 1, 0, 1, 0, 0] |
| DFC521LXM | [[@, ZGCOM, 助手, , 电脑, 版本, 连接, 麻烦, 发个], [@, ZG... | [0, 0] |
| DY252937111 | [[b5d3c3e39224215943719795adf3c21e], [美容, 养颜, ... | [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ... |
| Damon1991 | [[666666], [打, call], [搞起]] | [0, 0, 0] |
| Easonkong315 | [[发生, 什么, 了], [@, 寒星, , 你, 做, 空, 了], [不输, 为, 赢]] | [0, 1, 1] |
| Emotiona-2013 | [[12, 点, 左右], [263705027719aaf4ffff3ba79724c43... | [0, 0, 1, 1, 1, 1, 1, 0, 0] |
| F750584322 | [[ETF, 必火, 🔥, 🔥, 🔥, 🔥, 🔥, ETF, 必火, 🔥, 🔥, 🔥, 🔥,... | [1, 1, 1] |
| Fy825924 | [[马上, Baby, 进入, 5, 毛, 别错过, B轮, Ⅱ, ], [母婴, 链, ... | [1, 1, 0, 0, 0, 1] |

This is the table without group the users("talker_id" columns).

| | Unnamed: 0 | a.to_id | chatroom_name | talker_id | talker_name | content | create_time | type |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | [人生, 靠, 的, 不是, 时间, 靠, 的, 是, 珍惜, 人生, 的, 路, 到底, ... | 2020-04-29 14:26:11 | 1 |
| 1 | 1 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | [成功, 永远, 属于, 善于, 把握, 机会, 的, 人] | 2020-04-29 14:26:11 | 1 |
| 2 | 2 | 10026117318@chatroom | 云朋友币圈群 | wxid_gnu6weiev3qt22 | 付海 | [行动, 起来, 吧, 机会, 不, 等, 人] | 2020-04-29 14:26:12 | 1 |
| 3 | 3 | 10026117318@chatroom | 云朋友币圈群 | wxid_mkhghjwwdn1g22 | 海暖 | [肯定, 赚钱] | 2020-04-29 16:50:17 | 0 |
| 4 | 4 | 10026117318@chatroom | 云朋友币圈群 | wxid_mkhghjwwdn1g22 | 海暖 | [加仓, 搞, 起来, 就, 对, 了] | 2020-04-29 16:50:17 | 1 |
| 5 | 5 | 10026117318@chatroom | 云朋友币圈群 | wxid_mkhghjwwdn1g22 | 海暖 | [群里, 项目, 给力] | 2020-04-29 16:50:17 | 1 |

# 1.2 Preprocessing Data

In this step, I use re, jieba, groupby() to preprocessing data.

## 1.2.1 Using Regular Expression & Using jieba

### def dealData()

In this method, re is used to clean the punctuation; jieba is used to cut the long string into some words.

### def stopwords()

As I train Word2Vec model, I found some unnecessary chatting data in the document. In order to improve the accuracy of word2Vec model, I use stopwords here.

```
2
3    def dealData(chat_list):  # 传入  data["chatList"].dropna().copy(),要处理的内容
4
5        r = r'[\@\[\.\!\?\\\/\_\,\$\%\^\*\(\+\)\"\'\|\! \。\， \？ \、 \： \:\《\》 \“\” \（\） \"\]
6        chat_list = [re.sub(r, "", i) for i in chat_list]
7
8        chat_list = [jieba.lcut(i) for i in chat_list]
9
10       return chat_list
11
12   def stopwords(chat_list, stop_filename):
13       stopwords = []
14       with open(stop_filename) as f:
15           for line in f.readlines():
16               stopwords.append(line.strip())
17
18
19       nostop = []
20       for i in chat_list:
21           if i not in stopwords:
22               nostop.append(i)
23
24       return nostop
```

**OUTPUT**

```
和‘币圈’最相似的词语 :
 [('到时候', 0.9998137950897217), ('每天', 0.9998063445091248), ('真是', 0.9997912049293518),
('因为', 0.9997912049293518), ('别人', 0.9997683763504028), ('亏了', 0.9997477531433105),
('出现', 0.9997438192367554), ('只是', 0.9997314214706421), ('市场', 0.9997304081916809),
('有点', 0.9997120499610901)]
和‘圈里’最相似的词语 :
 [('美元', 0.9957764148712158), ('到位', 0.9951027035713196), ('这么久', 0.9950501918792725),
('心思', 0.9949938654899597), ('心存', 0.9948680400848389), ('跳', 0.9948317408561707), ('短
线', 0.9947868585586548), ('高端', 0.9947682619094849), ('万美元', 0.9947114586830139), ('一
秒', 0.9946306943893433)]
和‘比特’最相似的词语 :
 [('翻倍', 0.9993884563446045), ('开', 0.9993346333503723), ('跑', 0.9992882013320923),
('CX', 0.9992290735244751), ('奇点', 0.9992255568504333), ('如果', 0.9992146492004395), ('哪
里', 0.9992099404335022), ('咱们', 0.9992090463638306), ('不行', 0.9991886019706726), ('找',
0.9991669654846191)]
和‘区块链’最相似的词语 :
 [('仙女', 0.9995677471160889), ('提前', 0.9993887543678284), ('买入', 0.9993792772293091),
('版', 0.9993792176246643), ('穷开心', 0.999366044998169), ('十年', 0.9993610978126526), ('电
脑', 0.9993535876274109), ('分批', 0.9993487596511841), ('投', 0.9993364810943604), ('只要',
0.9993287920951843)]
```

Although there are still some irrelevant words here, the accuracy of the similar words in word2Vec has improved, when using function wv.most_similar( ).

## 1.2.2 WordFrequency & WordCloud

In this project, I use **cPicture(cut_chat, extra_chat, topicName)** to create WordCloud and WordFrequency at the same time.

**cut_chat** is the words cut by jieba

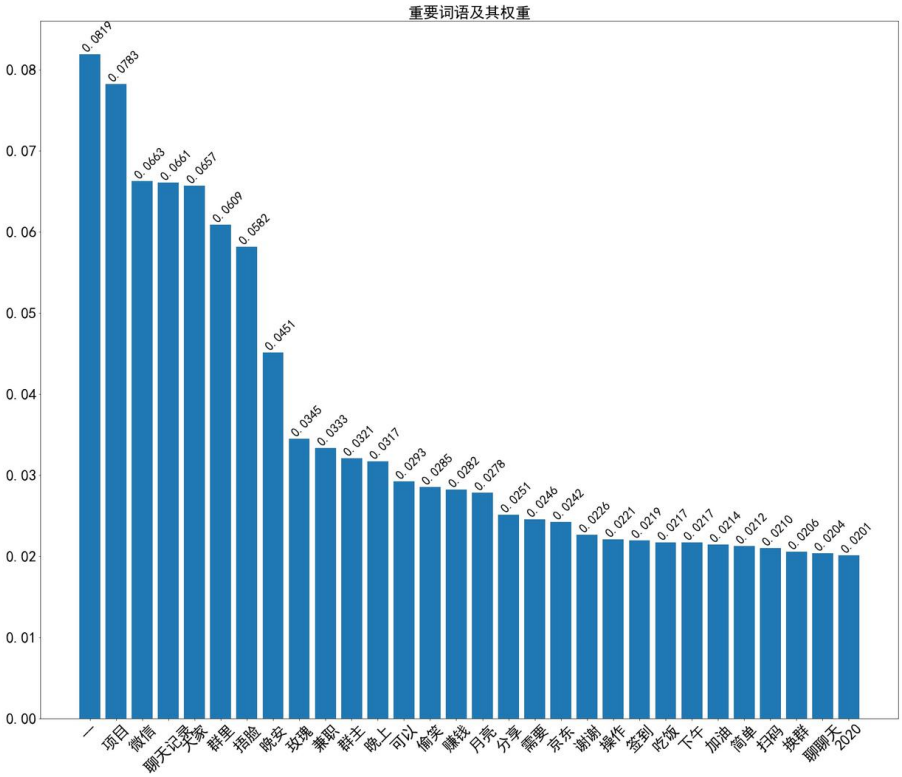
**extra_chat** includes the words and their frequency cut by jieba

**topicName** means the pictures' name

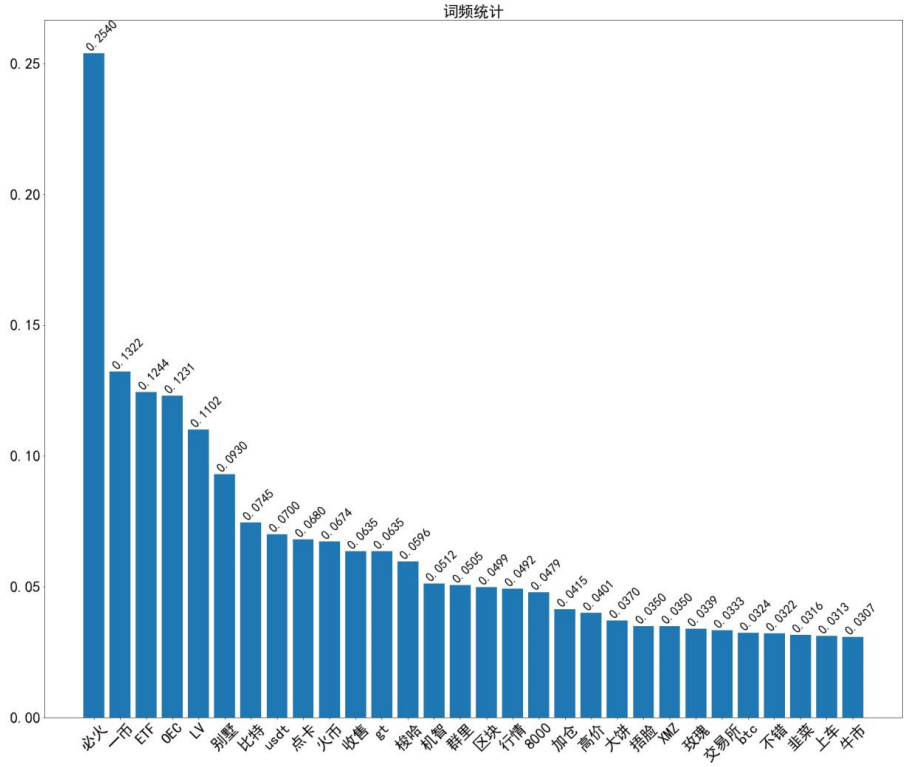
```python
# 生成词频和词云
def cPicture(cut_chat, extra_chat, topicName):

    FreqName = topicName + "词频统计.jpg"
    CloudName = topicName + "词云统计.jpg"

    print("*"*20, CloudName, "*"*20)

    # 词频
    xwords = []
    yweight = []

    for i in range(len(extra_chat)):
        xwords.append(extra_chat[i][0])
        yweight.append(extra_chat[i][1])


    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.figure(figsize=(30,25))
    plt.bar(range(30), yweight[:30])
    plt.xticks(range(30),xwords[:30],rotation=45, fontsize=30)
    plt.yticks(fontsize=30)
    for a,b in zip(range(0,len(xwords)), yweight[:30]):
        plt.text(a+0.3, b , '%0.4f' % b, ha='center', va= 'bottom',fontsize=7, rotation=45, size
    plt.title('词频统计', fontsize=30)
    plt.savefig(FreqName)
```

```python
    # 词云
    color_list=['#CD853F','#DC143C','#00FF7F','#FF6347','#8B008B','#00FFFF',
                '#0000FF','#8B0000','#FF8C00', '#1E90FF','#00FF00','#FFD700',
                '#008080','#008B8B','#8A2BE2','#228B22','#FA8072','#808080']#建立颜色数组
    colormap=colors.ListedColormap(color_list)#调用
    background_image = np.array(Image.open('China.jpg'))
    wc = WordCloud(background_color='white',
                   max_words=100,
                   font_path='C:/Windows/fonts/simsun.ttc',
                   random_state=30,
                   width=1500,
                   height=500,
                   mask = background_image,
                   colormap = colormap)
    wc.generate(cut_chat)
    plt.figure(figsize=(20,15))
    plt.imshow(wc)
    plt.axis('off')
    plt.show()
    wc.to_file(CloudName) #将词云保存为图片
```

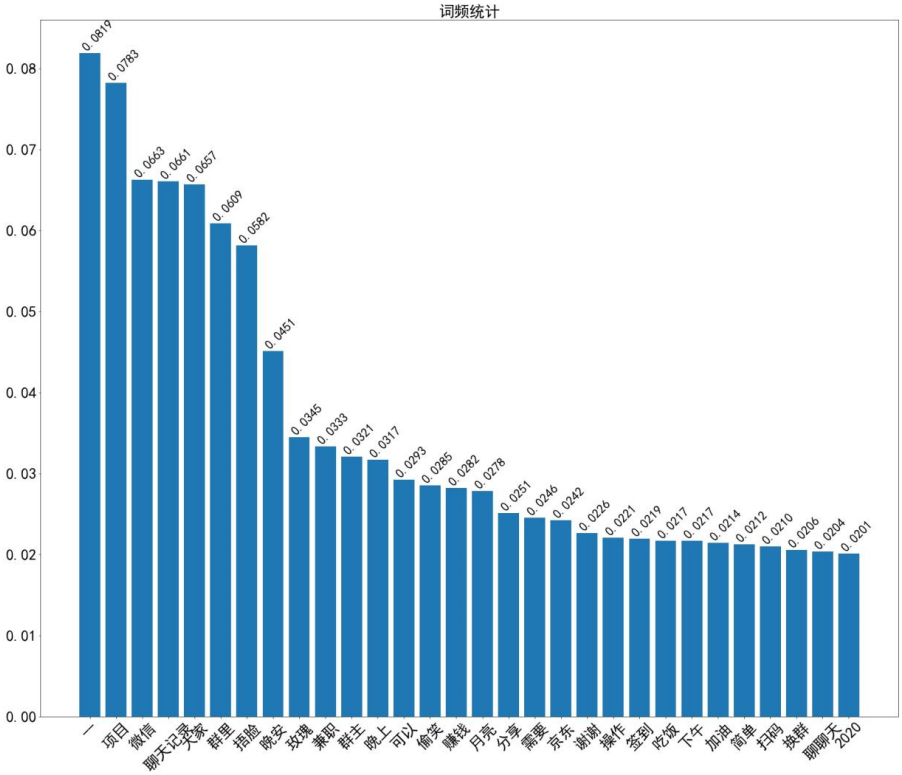
The Unmarked Data's WordCloud and WordFrequency

The chats labeled by 1 in the dataset

The chats labeled by 0 in the dataset

## 1.3 Using groupby to Realise Grouping Aggregation

Groupby is an efficient method to deal with table data, which is similar to SQL in database language. As I tried to aggregate all the chats from the same userid, this method provides me lots of conveniences.

# 2 Bag of Words

## 2.1 TfidfVectorizor & KMeans & Loop Iteration

Bag of Words model allows us not to consider the relationship between context and ignore their order. This model just uses the weight of all the words, and using tf-idf to express words in the sentences.

Before using the Bag of Words, we should cut the words and count the frequency of every word appearing in the articals. Then, we can find out the characters of the context based on these words and use TFIDF value to correct the vectors and nomalize them.

In this project, I use **TFidfVectorizor() t**o train words model. The reson I choose is that this method's result is similar to CounterVectorizor() and TFidfTransformer(), and is more concise.

After the bag of words has been generated, I put them into KMeans model with loop structure. I defined clusters' number from 2 to 8, and used CH value to find the best KMeans model.

**def BOW()  & def KModel(n, X_train):**

```python
# 训练词袋模型
def BOW(data, features):
    tf_model = TfidfVectorizer(lowercase=False, max_features=features)
    tf_array = tf_model.fit_transform(data)
    tf_word = tf_array.toarray()

    return tf_model, tf_array, tf_word


# 进行聚类
def KModel(n, X_train):
    km = KMeans(n_clusters=n, init="k-means++", max_iter=300, n_init=1)
    km.fit(X_train)

    return km
```

```
******************** 输出模型CH值 ********************
当簇的数量为 2 时的模型CH值:  5.055010127869553
当簇的数量为 3 时的模型CH值:  14.45488735338542
当簇的数量为 4 时的模型CH值:  18.770494682154897
当簇的数量为 5 时的模型CH值:  11.487996584710332
当簇的数量为 6 时的模型CH值:  8.081502316417962
当簇的数量为 7 时的模型CH值:  10.226136320872103
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\ADMINI~1\AppData\Local\Temp\jieba.cache
当簇的数量为 8 时的模型CH值:  7.023577292767934
模型标签:
 [0 0 0 ... 0 0 0]
模型准确度:  -1662.6627120651465
模型的ARI值:
 -0.003889576435636392
******************** 对大样本进行预测，通过CH值判断模型准确度 ********************
测试集维度 (10000,)
******************** 输出模型CH值 ********************
当簇的数量为 2 时的模型CH值:  69.96311459291198
当簇的数量为 3 时的模型CH值:  27.214014682445647
当簇的数量为 4 时的模型CH值:  18.74797466142062
当簇的数量为 5 时的模型CH值:  15.628216429381224
当簇的数量为 6 时的模型CH值:  13.875594656015833
当簇的数量为 7 时的模型CH值:  13.890958453367022
当簇的数量为 8 时的模型CH值:  19.40828533590873
当簇的数量为 9 时的模型CH值:  15.263017677836896
```

However, although these methods are simple, in large datasets, TFIDF methods sometimes will occupy all of the CPU. It's depending on the computer hardware we have, sometimes it will throw the "Memory ERROR". It also ignored the relationship between context, which would deeply influence the result of clustering. In order to solve these problems, I found a method called "Word2Vec model" on the Internet, and decided to use this method to train words vector.

# 3 Word Embedding & Clustering

## 3.1 Word2Vec & KMeans & MiniBatchKMeans

The Bag of Word model has the shortcoming such as sparse matrix, high vector dimension and unmeasurable relationship between words.

Word2Vec model was put forward in 2013 by Google. It can solve the problems in Bag of Words, and has faster training speed. It's a model used to transform normal words into a matrix before training the neural network model. It is an unsupervising matrix method. By using lots of unmarked raw data, word2Vec model can learn the relationship in the context and put out the word vector of every word.

Learning the paper published in 2013 called '**Research on Chinese word Clustering with Word2vec**' on CNKI, I am more familiar to use the word2vec and KMeans algorithm in order to find the most similar words of the input.

In my project, I defined 2 important functions in class w2vec: **word2vecModel()** to train the word vector and use **ClusterModel()** to cluster word vector by KMeans and MiniBatchKMeans, looping them for twice respectively. Finally, I compared their CH value to measure the cluster's result.

**def word2vecModel(data):**

```python
def word2vecModel(data):
    model = gensim.models.Word2Vec(data, min_count=2, size=500)

    model.save("model/word2vec.model")  # 保存模型
    # model_1 = gensim.models.Word2Vec.load("data/word2vec_1.model") # 加载使用模型
    word_list = list(model.wv.vocab)  # 查看词语

    print("加载模型： \n", model)
    print("model.wv.vectors 向量维度： ", model.wv.vectors.shape)  # 查看词向量维度
    print("len(model.wv.vocab) 词语长度: ", len(model.wv.vocab))
    print("词语 model.wv.vocab: \n", model.accuracy)
    print("词向量model.wv.vectors: \n", model.wv.vectors)  # 查看词向量
    print("词语 model.wv.vocab: \n", word_list)
    # print(model.wv.most_similar('币圈')) # 查看相似词

    #print("根据下标索引关键词 list(model.wv.vocab.keys())[num] : ", list(model.wv.vocab.keys())
    #print("查看个关键词对应的词向量 model[上面那个] : ", model[list(model.wv.vocab.keys())[3]])

    # 上面两个繁琐的方法可以用下面替代
    key_words = list(model.wv.vocab.keys())
    key_wordsVec = []
    for key in key_words:
        key_wordsVec.append(model[key])  # 将关键词向量放在words中，也就直接简化了model_1[list(mo
```

**def ClusterModel(keys, w2vmodel, Cnumber, kmodel):**

I use this function to generate KMeans model and MiniBatchKMeans

```
    ######################  KMeans 模型 ######################
    Kmodel = KMeans(n_clusters=Cnumber, random_state=123)
    Kmodel.fit(words) # 用关键词向量训练模型

    # 进行预测，这里没有分训练集和测试集
    y_pred = Kmodel.fit_predict(words)
    print("聚类的CH值为：", metrics.calinski_harabaz_score(words, y_pred))

    # 输出标签
    labels = Kmodel.labels_
    print("向量标签：\n", labels)

else:

    print("-"*10, "MiniBatchKMeans", "-"*10)


    ###################### MiniBatchKMeans 模型 ######################
    Kmodel = MiniBatchKMeans(n_clusters=Cnumber, random_state=123)
    Kmodel.fit(words) # 用关键词向量训练模型

    # 进行预测，这里没有分训练集和测试集
    y_pred = Kmodel.fit_predict(words)
    print("聚类的CH值为：", metrics.calinski_harabaz_score(words, y_pred))
```

**def useKmeans(self):**

I use this function to call the **ClusterModel()** and compare the results by CH value

```
# call the functions
keys = list(w2vmodel.wv.vocab.keys())

for i in [1, 2]:
    print("="*20, "第一次训练", "="*20)
    # 进行第一次聚类
    classk1 = w2vec.ClusterModel(keys, w2vmodel, 2, i)

    label_1 = classk1[1]
    label_2 = classk1[0]
    print("type(label_1)", type(label_1))


    print("="*20, "第二次训练", "="*20)
    # 第二次聚类
    classk2 = w2vec.ClusterModel(label_2, w2vmodel, 2, i)
    label_1 += classk2[1]

    label_2 += classk2[0]

#print("标签为1 : 总长度为", len(label_1), "\n", label_1) # 显示第一类簇的关键词
#print("标签为0 : 总长度为", len(label_2), "\n", label_0) # 显示第二类簇的关键词

return label_1, label_2
```

**OUTPUT**

## KMeans Model

```
==================== 第一次训练 ====================
关键词向量的维度： (5881, 500)
查看第三个关键词： 群
---------- KMeans ----------
聚类的CH值为： 8315.757916949955
向量标签:
 [0 0 1 ... 0 0 0]
type(label_1) <class 'list'>
==================== 第二次训练 ====================
关键词向量的维度： (5261, 500)
查看第三个关键词： 小小
---------- KMeans ----------
聚类的CH值为： 11649.575183725101
向量标签:
 [0 1 0 ... 1 0 0]
```

**MiniBatchKMeans**

```
==================== 第一次训练 ====================
关键词向量的维度： (5881, 500)
查看第三个关键词： 群
---------- MiniBatchKMeans ----------
聚类的CH值为： 8315.57352567848
向量标签:
 [0 0 1 ... 0 0 0]
type(label_1) <class 'list'>
==================== 第二次训练 ====================
关键词向量的维度： (5257, 500)
查看第三个关键词： 小小
---------- MiniBatchKMeans ----------
聚类的CH值为： 11621.415348869488
向量标签:
 [0 1 0 ... 1 0 0]
```

```
和‘币圈’最相似的词语 :
 [('每天', 0.9998537302017212), ('有点', 0.999813437461853), ('真是', 0.9997962713241577), ('价格', 0.9997631907463074), ('到时候',
0.9997608661651611), ('别人', 0.9997516870498657), ('因为', 0.999751031398732), ('亏了', 0.9997168183326721), ('只是',
0.9997149705886841), ('出现', 0.9997016191482544)]
和‘圈里’最相似的词语 :
 [('牛市', 0.9963019490242004), ('长期', 0.9961912631988525), ('美元', 0.9961227178573608), ('横盘', 0.9959972500801086), ('心存',
0.9959952235221863), ('到位', 0.9959677457809448), ('Fn', 0.9959142208099365), ('发车', 0.9958900809288025), ('合作',
0.9958726167678833), ('赶快', 0.9958264231681824)]
和‘比特’最相似的词语 :
 [('赶紧', 0.9994589686393738), ('如果', 0.9993711709976196), ('不行', 0.9993613362312317), ('看到', 0.9993579387664795), ('开',
0.9993499517440796), ('翻倍', 0.999306321144104), ('找', 0.9992974996566772), ('多久', 0.9992722272872925), ('捂脸',
0.999271035194397), ('情况', 0.9992524981498718)]
和‘区块链’最相似的词语 :
 [('仙女', 0.9995137453079224), ('真实', 0.9994981288909912), ('明月', 0.9994800686836243), ('8000', 0.9994419813156128), ('分批',
0.9994194507598877), ('版', 0.9994088411331177), ('话费', 0.9993969798088074), ('提前', 0.9993888735771179), ('左右',
0.999381422996521), ('本金', 0.9993621110916138)]
```

The shape of the word vector is (5881, 500). there are 5881 words and every words have 500 characters.

The CH values between two models are similar. However, there are also some problems about this word embedding method. Word2Vec model can only directly analyse words. Because of the complexity to transform word vector into sentences, after browsing some blogs on the Internet, I decided to use doc2Vec instead of word2Vec model.

## 3.2 Doc2Vec & HDBSCAN

Doc2Vec model enables the variable length of sentences. It is an unsupervised method as well.

In this model, every sentence can be described as a unique vector. It also increase an element called Paragraph Vector, which can be treated as a word vector, realizing the memorizing function.

In this project, I defined **function doc2Vector()** to train the sentence vector, and **HDBSCANmodel()** to train cluster model by the sentence vector.

Owing to the HDBSCAN function parameters are all the defaulted, the sentences are clustered into 299 clusters. Among which, the cluster labeled -1 includes the most points. -1 is also the label of noise points with no meanings, the second largest points are labeled 298 including 2011 points.

**def doc2Vector(docts):**

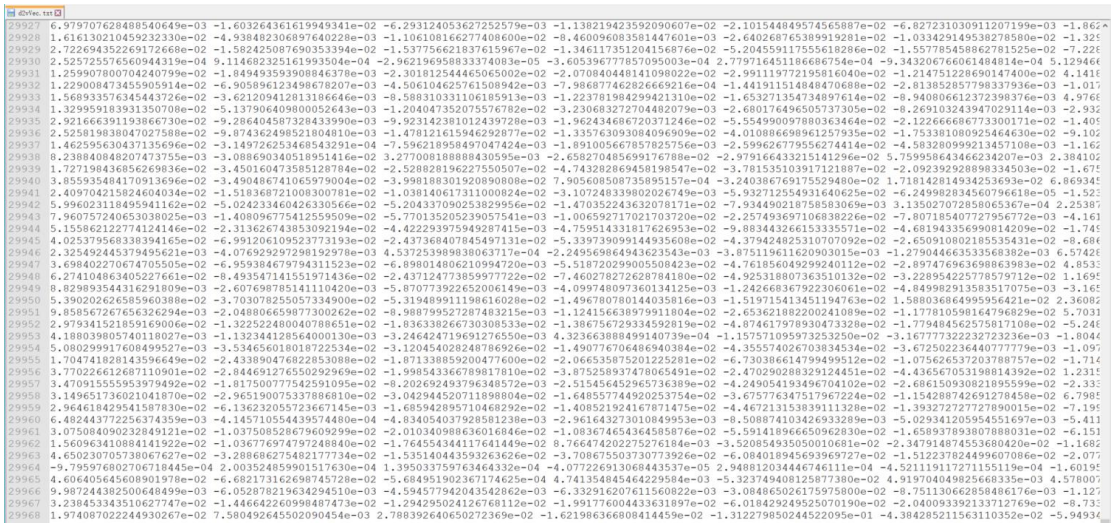
I use this function to train sentence vector. Here are the main code in function.

```
# 训练Doc2vec模型
def doc2Vector(docts):
    # TaggedDocument用于语料预处理,将语料转换为Doc2Vec可以用的格式

    x_train = []
    for i, test in enumerate(docts):

        #print('test原始文本', test)
        l = len(test)
        #print('原始文本长度', l)
        #test[l-1] = test[l-1].strip()
        document = gensim.models.doc2vec.TaggedDocument(test, tags=[i])
        #print("TaggededDocument(test, tags=[i]):", document)
        x_train.append(document)
        #print("x_train[i]:   ", x_train[i])
        #print("===============")
```

```
    vector_size = 500
    window_size = 16
    min_count = 2
    sampling_threshold = 1e-5
    negative_size = 5
    train_epoch = 100
    dm = 0          # 0 = dbow; 1 = dmpv
    worker_count = 1 #number of parallel processes


    model = gensim.models.Doc2Vec(x_train,vector_size=vector_size, window=window_size,
                                  min_count=min_count,
                                  sample=sampling_threshold,workers=worker_count,
                                  hs=0, dm=dm, negative=negative_size, dbow_words=1, dm_concat=1

    model.save("model/doc2vec")
    return model

#model = doc2Vector()
model = gensim.models.Doc2Vec.load("model/doc2vec")
```

**OUTPUT**

The sentence vector has 29968 rows and 500 columns, which is same as the raw comments data.



**def HDBSCANmodel(sect_vect):**

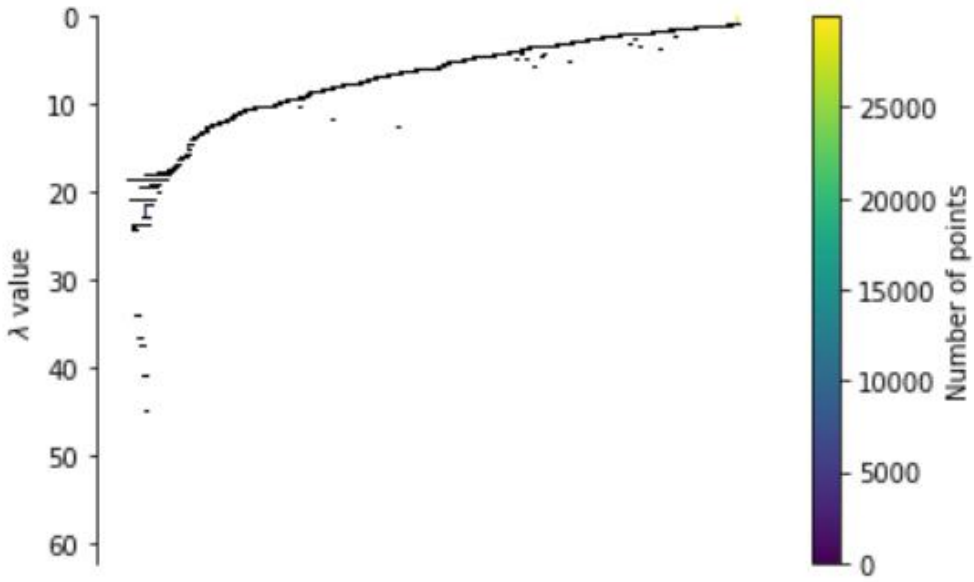
```
# 对文档进行聚类
def HDBSCANmodel(sect_vect):
    import hdbscan

    cluster = hdbscan.HDBSCAN() # 使用默认参数，min_cluster_size=5
    joblib.dump(cluster, 'model/HDBSCANcluster.pkl')
    cluster.fit(sect_vect)


    label = cluster.labels_
    print(type(label))
    print("聚类标签的维度：", label.shape)  # 一共有标签数和词向量维度相同
    print("聚类标签的最大值：", label.max())
    print("聚类标签的最小值：", label.min())

    return cluster, label


#cluster, label = HDBSCANmodel(sect_vect)
cluster = joblib.load('model/HDBSCANcluster.pkl')
label = cluster.labels_
```

```
# 使用Counter计数
from collections import Counter

count = Counter(label)
print(type(count))
print('统计出现次数最多的标签', count.most_common(10)) # 统计出现次数最多的2类标签


# 使用hdbscan内置函数进行可视化
import matplotlib.pyplot as plt
cluster.condensed_tree_.plot()


import seaborn as sns
#cluster.condensed_tree_.plot(select_clusters=True, selection_palette=sns.color_palette('deep',

network = cluster.condensed_tree_.to_networkx()
print("聚类的网络节点个数：\n", network.number_of_nodes())

tree = cluster.condensed_tree_.to_pandas() # 显示树结构的详细信息
print("聚类的树节点信息：\n", tree.head(6))
print("聚类的树节点信息：\n", tree.describe())
```

**OUTPUT**

These are the result and visualization of HDBSCAN.

```
聚类标签的维度:  (29968,)
聚类标签的最大值:  298
聚类标签的最小值:  -1
<class 'collections.Counter'>
统计出现次数最多的标签 [(-1, 24351), (298, 2011), (284, 155), (103, 81), (190, 63), (135, 54),
(293, 49), (64, 42), (52, 39), (107, 36)]
```




聚类的网络节点个数:

```
 30585
```

聚类的树节点信息:

| | parent | child | lambda_val | child_size |
|---|---|---|---|---|
| 0 | 29968 | 20322 | 0.461213 | 1 |
| 1 | 29968 | 4807 | 0.470937 | 1 |
| 2 | 29968 | 14759 | 0.554829 | 1 |
| 3 | 29968 | 21411 | 0.561444 | 1 |
| 4 | 29968 | 521 | 0.563079 | 1 |
| 5 | 29968 | 14752 | 0.563402 | 1 |

聚类的树节点信息:

| | parent | child | lambda_val | child_size |
|---|---|---|---|---|
| count | 30584.00000 | 30584.000000 | 30584.000000 | 30584.000000 |
| mean | 30381.09129 | 15291.520141 | 11.873446 | 187.312353 |
| std | 161.44186 | 8829.018837 | 8.937446 | 2072.494863 |
| min | 29968.00000 | 0.000000 | 0.461213 | 1.000000 |
| 25% | 30261.00000 | 7645.750000 | 6.049682 | 1.000000 |
| 50% | 30409.00000 | 15291.500000 | 9.827303 | 1.000000 |
| 75% | 30521.00000 | 22937.250000 | 15.368832 | 1.000000 |
| max | 30584.00000 | 30584.000000 | 59.321917 | 29923.000000 |

# 4 Evaluation

## 4.1 caliski_harabaz_score

**distance**: This is the distance between labeled data and the center of the clusters

The greater CH value means a better model.

## 4.2 Adjusted Rand Index

This index is used to measure the accuracy of the predicted results and the true y data in cluster. The range of ARI is [-1, 1]. If ARI is smaller than 0, it reveals that the cluster couldn't better express the result of raw data.

The cluster would be more accurate if ARI value is closer to 1.

## 4.3 Visualization

This step is used in the **Doc2Vec and HDBSCAN**, by ploting the **condensed tree.**

# 5 Extension : HDBSCAN

## 5.1 The Definition

Traditional clustering assigns each point in a data set to a cluster (or to noise). This is a hard assignment; there are no mixed memberships. If the sample of dataset is too large, the clustering convergence time is long. Also, when the density of some clustering is uneven or the large difference between clusters, the quality of clustering will be poor, because the parameters of Minpts and Eps would hard to choose.

HDBSCAN* can be thought of as a natural extension of the popular DBSCAN algorithm. It's a robust hierarchical version of DBSCAN. The accelerated HDBSCAN* algorithm provides comparable performance to DBSCAN, while supporting variable density clusters, and eliminating the need for the difficult to tune distance scale parameter. This makes accelerated HDBSCAN* the default choice for density based clustering.

The basic flow of this algorithm includes steps below:

Transform the space according to the density/sparsity.

Build the minimum spanning tree of the distance weighted graph.

Construct a cluster hierarchy of connected components.

Condense the cluster hierarchy based on minimum cluster size.

Extract the stable clusters from the condensed tree.

**cite**

https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html#hdbscan

https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html

**There are some details of the algorithm in paper.**

## 2.1 Statistically Motivated HDBSCAN*

A statistically oriented view of density clustering begins with the assumption that there exists some unknown density function from which the observed data is drawn. From the density function $f$, defined on a metric space $(\mathcal{X}, d)$, one can construct a hierarchical cluster structure, where a cluster is a connected subset of an $f$-level set $\{x \in (\mathcal{X}, d) \mid f(x) \geq \lambda\}$. As $\lambda \geq 0$ varies these $f$-level sets nest in such a way as to construct an infinite tree, which is referred to as the *cluster tree* (see figure 2 for an example). Each cluster is a branch of this tree, extending over the range of $\lambda$ values for which it is distinct. The goal of a clustering algorithm is to suitably approximate the cluster tree, converging to it in the limit of infinite observed data points.

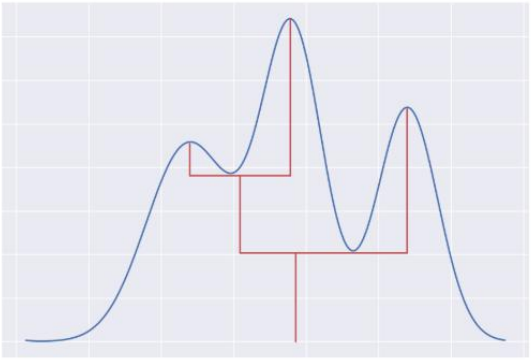


Figure 2: The cluster tree (red) induced by a density function (blue).

A point $X_i$ is called a *core point* with respect to $\varepsilon$ and $k$ if its $\varepsilon$-neighbourhood contains at least $k$ many points, i.e. if $|B(X_i, \varepsilon) \cap X| \geq k$. That is, the open ball of radius $\varepsilon$ contains at at least $k$ many points from X.

Two core points $X_i$ and $X_j$ are *$\varepsilon$-reachable* with respect to $\varepsilon$ and $k$ if $X_i \in B(X_j, \varepsilon)$ and $X_j \in B(X_i, \varepsilon)$. That is, they are both core points with respect to $k$, and are both contained within each others $\varepsilon$-neighbourhood. Two core points $X_i$ and $X_j$ are *density-connected* with respect to $\varepsilon$ and $k$ if they are directly or transitively $\varepsilon$-reachable.

A *cluster* $C$, with respect to $\varepsilon$ and $k$, is a non-empty maximal subset of $X$ such that every pair of points in $C$ is density-connected. This definition of cluster results in the DBSCAN* algorithm.

To extend the algorithm to get HDBSCAN* we need to build a hierarchy of DBSCAN* clusterings for varying $\varepsilon$ values. The key to doing this is to redefine how we measure distance between points in $X$. For a given fixed value $k$, we define a new distance metric derived from the metric $d$, called the *mutual reachability distance*, as follows. For any point $X_i$ we define the *core-distance* of $X_i$, denoted $\kappa(X_i)$ to be the distance to the $k^{\text{th}}$ nearest neighbor of $X_i$; then, given points $X_i$ and $X_j$ we define

$$d_{\text{mreach}}(X_i, X_j) = \begin{cases} \max\{\kappa(X_i), \kappa(X_j), d(X_j, X_j)\} & X_i \neq X_j \\ 0 & X_i = X_j \end{cases}.$$

It is straightforward to show that this is indeed a metric on $X$. We can then apply standard Single Linkage Clustering [51] to the discrete metric space $(X, d_{\text{mreach}})$ to obtain a hierarchical clustering of $X$. The clusters at level $\varepsilon$ of this hierarchical clustering are precisely the clusters obtained by DBSCAN* for the parameter choices $k$ and $\varepsilon$; in this sense we have derived a hierarchical DBSCAN* clustering.

cite: 《Accelerated Hierarchical Density Clustering》

## 5.2 The Advantage

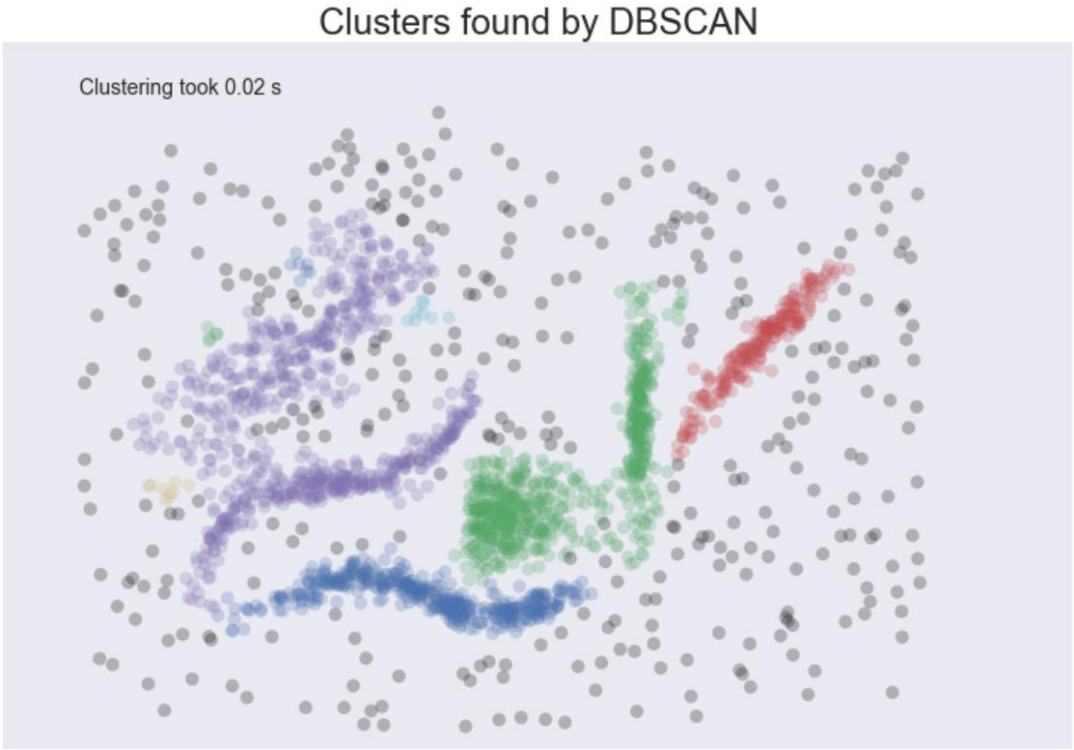
HDBSCAN can find clusters of varying densities (unlike DBSCAN), and can be more robust to parameter selection.

Here are some visualization results on the official website between HDBSCAN, DBSCAN and AgglomerativeClustering.

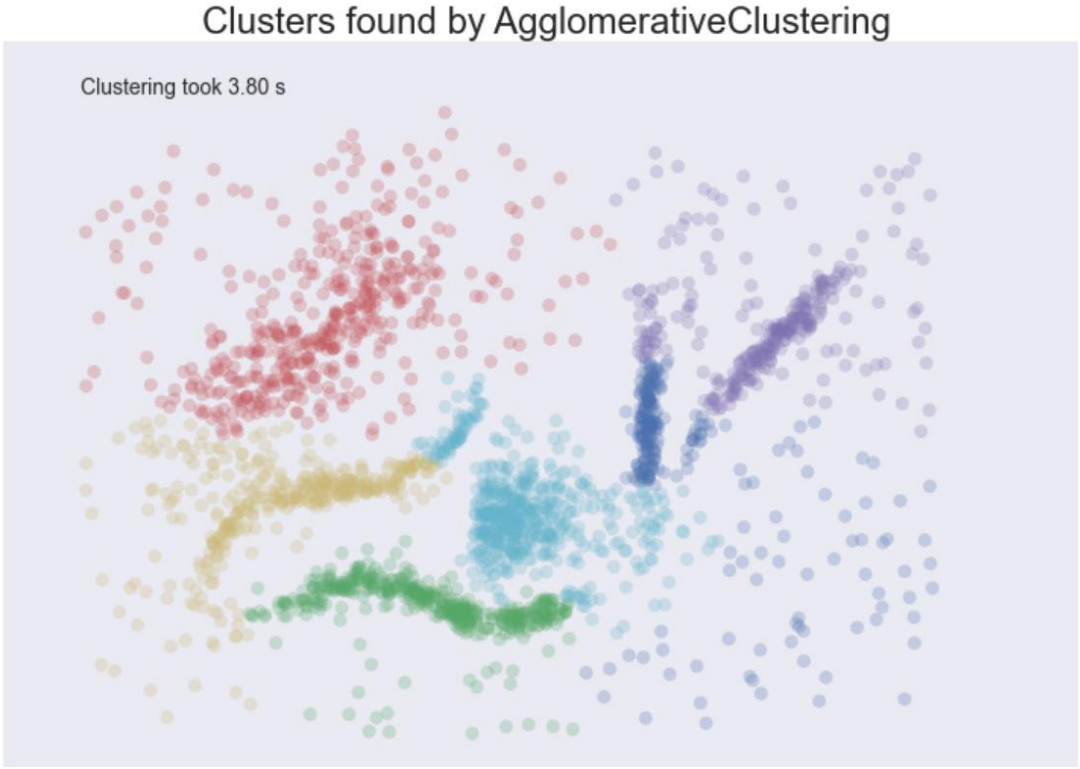
The dataset is its own called 'clusterable_data.npy', so I just put the screenshot here.

```
data = np.load('clusterable_data.npy')
```

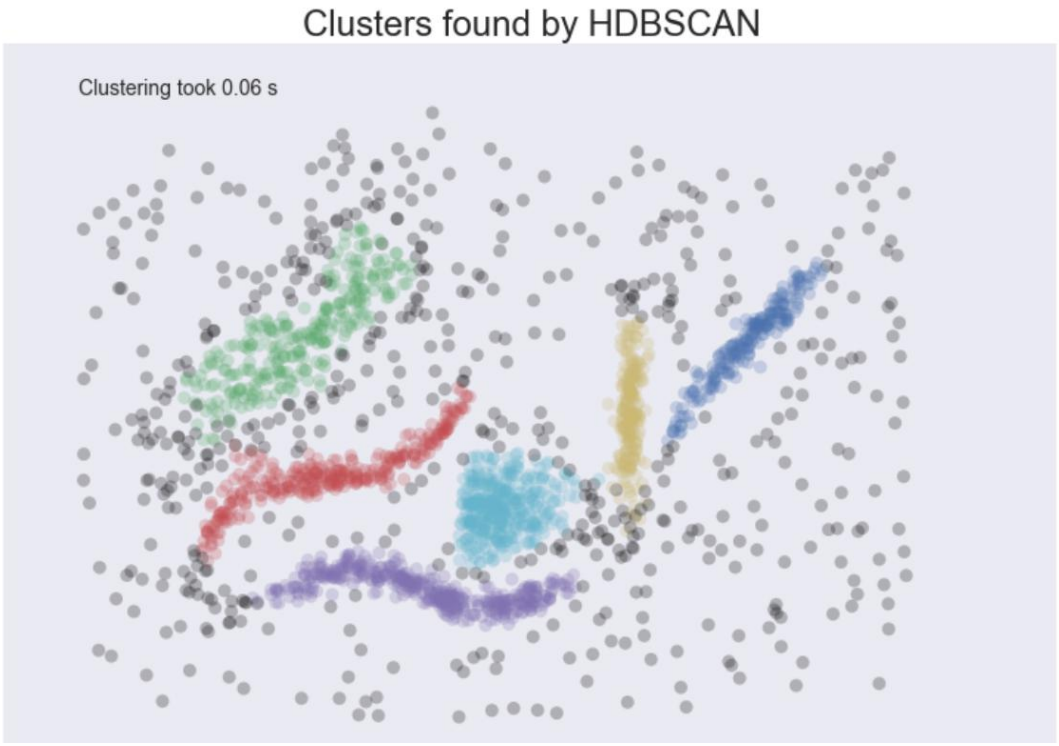
```
plot_clusters(data, cluster.DBSCAN, (), {'eps':0.025})
```




```
plot_clusters(data, cluster.AgglomerativeClustering, (), {'n_clusters':6, 'linkage':'ward'})
```



```
plot_clusters(data, hdbscan.HDBSCAN, (), {'min_cluster_size':15})
```




cite: Comparing Python Clustering Algorithms

## 5.3 How to Use

**classhdbscan.hdbscan_.HDBSCAN(min_cluster_size**=5, **min_samples**=None, **cluster_selection_epsilon**=0.0, **metric**='euclidean', **alpha**=1.0, p=None, **algorithm**='best', **leaf_size**=40, **memory=Memory**(location=None), **approx_min_span_tree**=True, **gen_min_span_tree**=False, **core_dist_n_jobs**=4, **cluster_selection_method**='eom', **allow_single_cluster**=False, **prediction_data**=False, **match_reference_implementation**=False, **kwargs)

Here are some important parameters and methods I use in the project, more details can be seen in the official document.(HDBSCAN api cite)

### 5.3.1 Parameters

**min_cluster_size** :int, optional (default=5)

This parameter is the minimum size of clusters

**min_samples** :int, optional (default=None)

The number of samples in a neighbourhood for a point to be considered a core point.

**algorithm** :string, optional (default='best')

hdbscan has variants specialised for different characteristics of the data. If we default the parameter as 'best', it will automaticly choose the best algorithm amonng 'generic', 'prims_kdtree', 'prims_balltree', 'boruvka_kdtree' and 'boruvka_balltree'.

**metric** :string, or callable, optional (default='euclidean')

used to calculate distance between instances in a feature array

**p** :int, optional (default=None)

p value to use if using the minkowski metric.

cite: HDBSCAN api

### 5.3.2 Properties

**labels_** : ndarray, shape (n_samples, )

Cluster labels for each point in the dataset given to fit(). **Noisy samples** are given the label **-1.**

**probabilities_** : ndarray, shape (n_samples, )

The strength with which each sample is a member of its assigned cluster. Noise points have probability zero; points in clusters have values assigned proportional to the degree that they persist as part of the cluster.

**outlier *scores*_** : ndarray, shape (n_samples, )

Outlier scores for clustered points; the larger the score the more outlier-like the point. It is useful as an outlier detection technique.

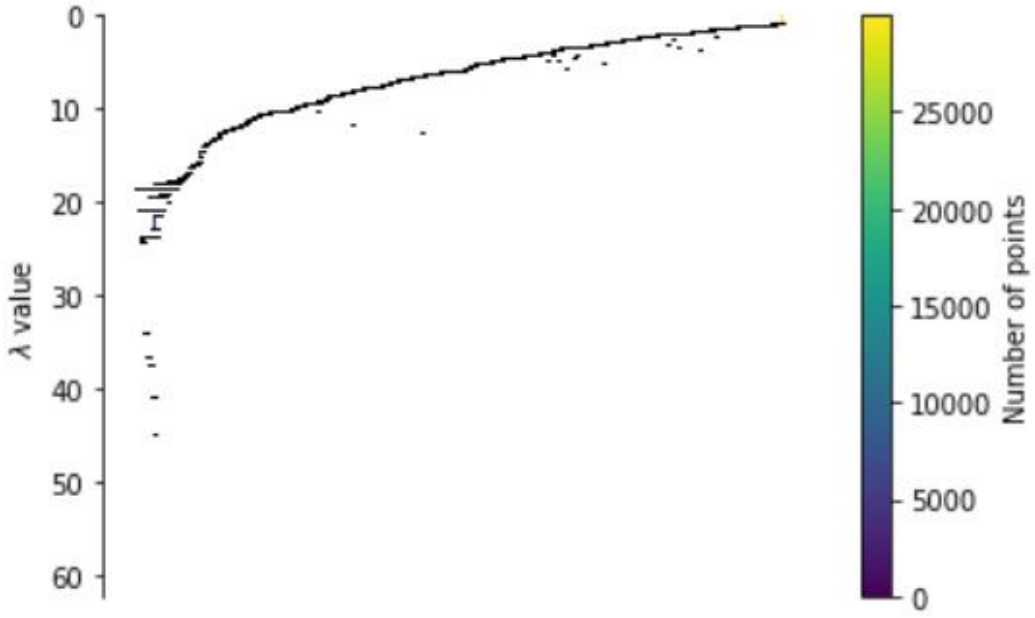
**condensed *tree*_** : CondensedTree object

the condensed tree structure, which provides a simplified or smoothed version of the SingleLinkageTree.

This object has methods for converting to pandas, networkx, and plotting.

The **lambda_val value** is the value (1/distance) at which the child node leaves the cluster.

The number of points in the child node will be described as the **child_size.**




**condensed *tree*_.to_networkx()**

Return a NetworkX DiGraph object representing the condensed tree.

Edge weights in the graph are the lamba values at which child nodes 'leave' the parent cluster.

Nodes have a size attribute attached giving the number of points that are in the cluster (or 1 if it is a singleton point) at the point of cluster creation (fewer points may be in the cluster at larger lambda values).

**CODE**

```
network = cluster.condensed_tree_.to_networkx()
print("聚类的网络节点个数：\n", network.number_of_nodes())
```

**OUTPUT**

```
聚类的网络节点个数:
 30585
```

**condensed *tree*_.to_pandas()**

Return a Pandas-DataFrame representation of the condensed tree.

Each row of the dataframe corresponds to an edge in the tree. The columns of the dataframe are parent, child, lambda_val and child_size.

The **lambda_val value** is the value (1/distance) at which the child node leaves the cluster.

The child_size is the number of points in the child node.

**CODE**

```
tree = cluster.condensed_tree_.to_pandas() # 显示树结构的详细信息
print("聚类的树节点信息：\n", tree.head(6))
print("聚类的树节点信息：\n", tree.describe())
```

**OUTPUT**

```
聚类的树节点信息：
    parent  child  lambda_val  child_size
0   29968  20322    0.461213           1
1   29968   4807    0.470937           1
2   29968  14759    0.554829           1
3   29968  21411    0.561444           1
4   29968    521    0.563079           1
5   29968  14752    0.563402           1
聚类的树节点信息：
             parent         child    lambda_val    child_size
count  30584.00000  30584.000000  30584.000000  30584.000000
mean   30381.09129  15291.520141     11.873446    187.312353
std      161.44186   8829.018837      8.937446   2072.494863
min    29968.00000      0.000000      0.461213      1.000000
25%    30261.00000   7645.750000      6.049682      1.000000
50%    30409.00000  15291.500000      9.827303      1.000000
75%    30521.00000  22937.250000     15.368832      1.000000
max    30584.00000  30584.000000     59.321917  29923.000000
```

## 5.3.3 Methods

fit(X, y=None)

Performs HDBSCAN clustering from features or distance matrix.

fit_predict(X, y=None)

Performs clustering on X and returns cluster labels.

cite:

https://hdbscan.readthedocs.io/en/latest/api.html

https://hdbscan.readthedocs.io/en/latest/api.html?highlight=CondensedTree#hdbscan.plots.CondensedTree.plot

https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

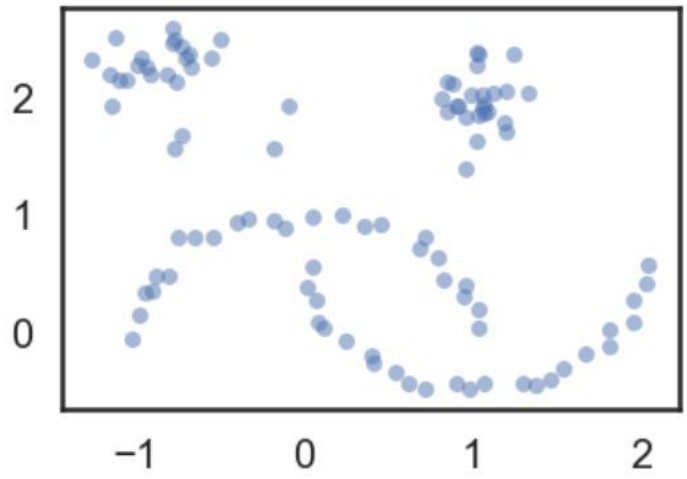
# Practice |

In this example, I use the dataset from make_moons. The array's shape is (50, 2)

Python

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.datasets as data
%matplotlib inline
sns.set_context('poster')
sns.set_style('white')
sns.set_color_codes()
plot_kwds = {'alpha' : 0.5, 's' : 80, 'linewidths':0}


moons, _ = data.make_moons(n_samples=50, noise=0.05)
blobs, _ = data.make_blobs(n_samples=50, centers=[(-0.75,2.25), (1.0, 2.0)], cluster_std=0.25)
test_data = np.vstack([moons, blobs])
plt.scatter(test_data.T[0], test_data.T[1], color='b', **plot_kwds)
```

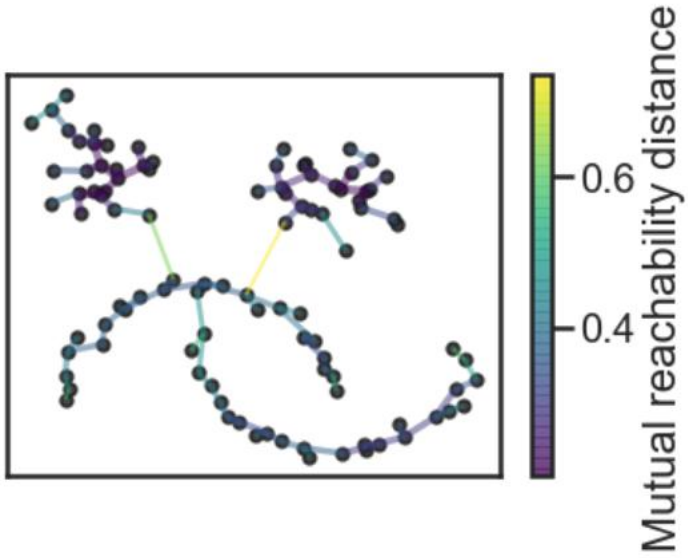
```
<matplotlib.collections.PathCollection at 0x211b80a8438>
```




Python

```python
# test_data is a two-dimensional array

import hdbscan

cluster = hdbscan.HDBSCAN(min_cluster_size=5, gen_min_span_tree=True)
cluster.fit(test_data)


cluster.minimum_spanning_tree_.plot(edge_cmap='viridis')
```

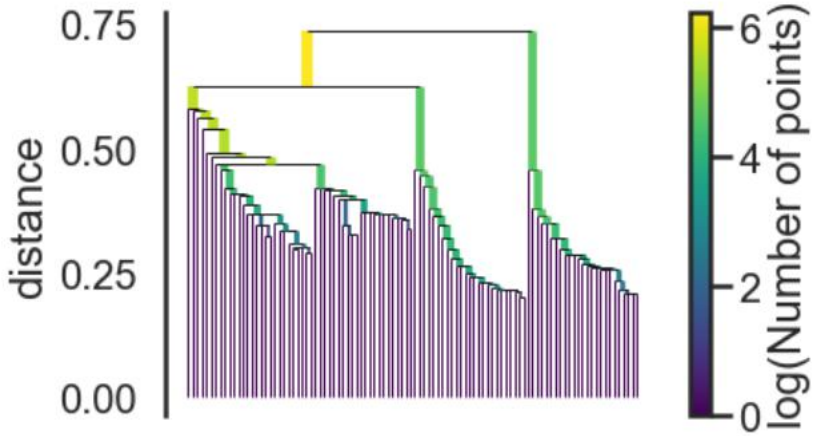
<matplotlib.axes._subplots.AxesSubplot at 0x18e7694c7b8>




Python

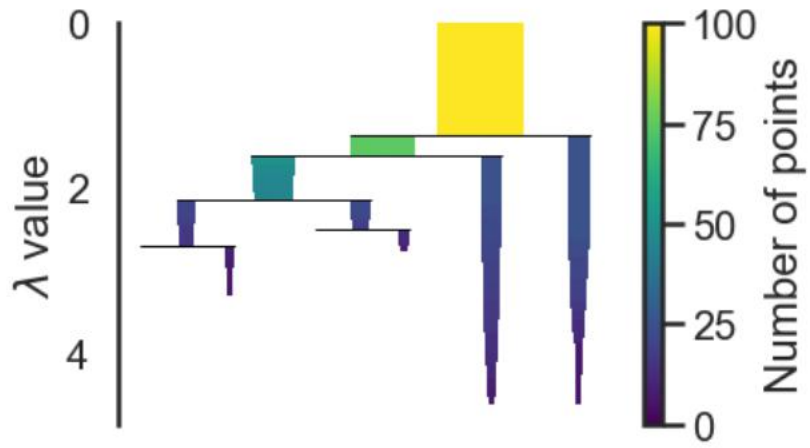
```
cluster.single_linkage_tree_.plot(cmap='viridis', colorbar=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x18e77f0e518>



```python
cluster.condensed_tree_.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18e78233eb8>
```




```python
print(cluster.labels_)
print(cluster.probabilities_)
# from the results. there are 3 types of label(0, 1, 2)
# the probabilities of every points in there cluster
```

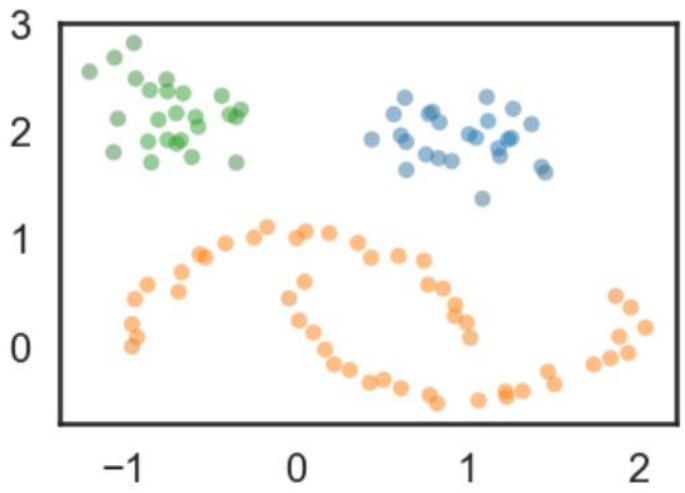
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 2, 2, 0, 0, 2,
       0, 2, 2, 0, 0, 0, 2, 0, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2,
       2,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

```
array([1.        , 1.        , 0.80789394, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.95420034, 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.80517357, 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.86501868,
       1.        , 1.        , 1.        , 0.96356793, 0.8315837 ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.89223355, 0.61917754, 1.        , 0.81596332, 0.83797048,
       0.72772031, 0.77454401, 0.99922062, 1.        , 0.83286601,
       0.48338051, 0.5912331 , 0.73046966, 0.7528101 , 1.        ,
       0.95800899, 0.61297003, 0.57285423, 1.        , 0.82576458,
       0.83797048, 0.67054388, 1.        , 1.        , 1.        ,
       0.93610554, 0.59377358, 0.47389305, 0.77977942, 0.67787218,
       0.86819383, 0.8139032 , 0.81864856, 0.83797048, 0.90126655,
       0.75013164, 1.        , 0.51193752, 0.62664097, 1.        ,
       0.9356931 , 0.98326729, 1.        , 0.66958595, 0.57024862,
       0.74855591, 1.        , 0.47199353, 0.92063353, 0.8075204 ])
```

Python

```python
palette = sns.color_palette()
cluster_colors = [sns.desaturate(palette[col], sat)
                  if (col>=0) else (0.5, 0.5, 0.5) for col, sat in
                  zip(cluster.labels_, cluster.probabilities_)]
plt.scatter(test_data.T[0], test_data.T[1], c=cluster_colors, **plot_kwds)
# the result of clustering
```

```
<matplotlib.collections.PathCollection at 0x18e76ceb860>
```




## Practice ||

To fit the chatting dataset, I also try to use the array shaped (6000, 500) in the make_blobs dataset.

Python

```
from sklearn.datasets import make_blobs
import pandas as pd

blobs, labels = make_blobs(n_samples=6000, n_features=500)
data = pd.DataFrame(data=blobs)
print(data.shape)
print(labels)
data.head(7)
```

```
(6000, 500)
[0 0 0 ... 0 0 1]
```

```
data.head(7)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 490 | 491 | 492 | 493 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4.840971 | 7.130291 | -6.936840 | 2.446009 | 2.049424 | 3.873181 | 6.569192 | 3.485492 | -2.986632 | 4.865946 | ... | -4.644166 | 10.747358 | -6.566318 | -5.263030 |
| 1 | 4.930530 | 8.319283 | -7.115806 | 0.883318 | -0.842033 | 3.210325 | 7.055222 | 3.395649 | -4.754589 | 5.939645 | ... | -5.159861 | 8.298399 | -7.020673 | -4.863626 |
| 2 | 6.485531 | 7.459185 | -8.889313 | 2.303936 | 1.266680 | 5.548280 | 5.190321 | 2.565770 | -2.892329 | 6.301714 | ... | -5.963291 | 7.941843 | -6.219212 | -6.271021 |
| 3 | -5.678500 | 0.280614 | 7.685208 | 5.738521 | 10.131719 | -0.823959 | -3.618885 | 1.118169 | 6.330838 | 5.201983 | ... | 1.312370 | -0.455472 | 6.507566 | -7.471248 |
| 4 | -4.891223 | 1.856525 | 7.304199 | 6.360201 | 11.325607 | -0.619695 | -5.327115 | 1.049755 | 4.331539 | 6.396216 | ... | 0.386110 | -2.293028 | 4.612655 | -6.716202 |
| 5 | 5.004414 | 7.985083 | -8.064721 | 2.336760 | 1.232019 | 4.710146 | 5.620391 | 3.707330 | -1.690951 | 4.295537 | ... | -5.003607 | 10.944366 | -5.615939 | -5.642530 |
| 6 | 2.585946 | 9.626607 | 0.886764 | 7.922988 | -0.251214 | 6.245540 | -7.009837 | -2.783938 | 6.840805 | -5.121439 | ... | 6.244042 | 0.238707 | -2.486246 | -2.072747 |

Python

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Columns: 500 entries, 0 to 499
dtypes: float64(500)
memory usage: 22.9 MB
```

Python

```
data.describe()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 490 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | 6000.000000 | ... | 6000.000000 |
| mean | 0.842007 | 5.860589 | 0.210236 | 4.913558 | 3.717486 | 3.558684 | -1.825809 | 0.260447 | 2.940322 | 1.768782 | ... | 0.841701 |
| std | 4.630380 | 3.994243 | 6.183047 | 2.904665 | 4.403821 | 3.618591 | 5.656989 | 2.287402 | 4.706358 | 5.494250 | ... | 4.923521 |
| min | -8.783172 | -2.820215 | -10.517310 | -2.622289 | -4.045605 | -5.020166 | -9.638952 | -5.678180 | -6.642129 | -9.315617 | ... | -9.038802 |
| 25% | -4.535833 | 1.293918 | -6.977922 | 1.788295 | 0.000716 | -0.446403 | -6.131830 | -1.833485 | -2.742710 | -5.160019 | ... | -4.633336 |
| 50% | 2.137761 | 7.157161 | 0.990636 | 6.008750 | 2.186931 | 4.611244 | -4.932633 | 0.765212 | 4.775884 | 4.900868 | ... | 1.330423 |
| 75% | 4.975571 | 9.169860 | 6.555648 | 7.195765 | 8.916472 | 6.587407 | 5.346642 | 2.083600 | 6.814287 | 5.921950 | ... | 5.793194 |
| max | 8.913562 | 12.988190 | 10.597644 | 10.988681 | 12.617204 | 10.966377 | 9.363488 | 5.992707 | 10.687226 | 9.125977 | ... | 10.529517 |

8 rows × 500 columns

Python

```python
import hdbscan
cluster = hdbscan.HDBSCAN()
cluster.fit(blobs)
cluster.labels_
```

```
HDBSCAN(algorithm='best', allow_single_cluster=False, alpha=1.0,
    approx_min_span_tree=True, cluster_selection_epsilon=0.0,
    cluster_selection_method='eom', core_dist_n_jobs=4,
    gen_min_span_tree=False, leaf_size=40,
    match_reference_implementation=False, memory=Memory(location=None),
    metric='euclidean', min_cluster_size=5, min_samples=None, p=None,
    prediction_data=False)
```

Python

```python
cluster.labels_.max()
cluster.probabilities_ # 概率
```

```
cluster.labels_
```

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

```
cluster.labels_.max()
```

```
2
```

```
cluster.probabilities_ # 概率
```

```
array([0.99896982, 0.96256504, 0.93892017, ..., 0.93873826, 0.97505015,
       0.96077108])
```