1. We first process the dataset we select, and transfer it to a csv file with 20 features column and lable column. Then we use kmeans to do clustering. And finally we will use d3 script to do visualization. Preprocess the dataset as above:

In [1]:
```
import pandas as pd
import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import *
from pyspark.sql import SQLContext
from pyspark.ml.feature import StopWordsRemover
from pyspark.ml.feature import NGram
from pyspark.ml.feature import Word2Vec
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.feature import HashingTF, IDF

spark = SparkSession.builder.appName("data_process").getOrCreate()
```

In [2]:
```
df0 = spark.read.format('json').load('/home/hz2558/wiki1')
df1 = spark.read.format('json').load('/home/hz2558/wiki2')
pd_final = df1.toPandas().append(df0.toPandas())
#import two datasets, then combine to one pandas frame

sc = spark.sparkContext
sqlContest = SQLContext(sc)
df = sqlContest.createDataFrame(pd_final)
```

In [4]:
```
# token, filter and featurize the dataset
regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="[^A-Za-z]+", toLowercase=True)
tokenized_data = regexTokenizer.transform(df)
#filter the tokened data
stopWordsRemover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
filtered_data = stopWordsRemover.transform(tokenized_data)

hashingTF = HashingTF(inputCol="filtered_words", outputCol="raw_features", numFeatures=20)
featurizedData = hashingTF.transform(filtered_data)

idf= IDF(inputCol="raw_features", outputCol="features")
idfModel = idf.fit(featurizedData)
featurized_data = idfModel.transform(featurizedData)
data = featurized_data.drop('id','text','title','url','words','filtered_words','raw_features')
```

Import the rawdata to csv file, with 20 feaures columns and label column

In [5]:
```
process1 = udf(lambda x: x.toArray().tolist(), ArrayType(DoubleType()))
#process1:a user defined function used to transfer data to list
process2 = data.select(process1("features").alias("features")).collect()
#porcess2: used to select specific column and do process1 to it
process3 = pd.DataFrame(list(map(lambda x:x.features,process2)))
#procees3: used to map all the data.features with process 2
```

In [6]:
```
import numpy as np
pred = data.toPandas()
df = pd.concat([pred, process3], axis=1)
df = df.drop(["features"], axis=1)
id = np.arange(0,1564,1)
#we total have 1564 rows data

label0 = np.zeros(872)
#dataset labeled with 0
label1 = np.zeros(692)+1
#label1 labeled with 1

label = np.append(label0,label1)
df.insert(0,'ID', id)
df.insert(21, 'label', label)
#insert another column "ID"

df.to_csv("rawdata.csv", index =False, sep=',')
#import df with csv format
```

User kmeans model to do clustering, then use the prediction result as the label.

In [8]:
```
import numpy as np
from pyspark.ml.clustering import KMeans
# use kmeans model to pporcess data
kmeans = KMeans(featuresCol='features', predictionCol='label', k=2, maxIter=20, seed=1)
model = kmeans.fit(data)
result = model.transform(data)
#use model to divide the dataset into two clusters

data = result.select("features")
label = result.select("label").toPandas()

process1 = udf(lambda x: x.toArray().tolist(), ArrayType(DoubleType()))
process2 = data.select(a("features").alias("features")).collect()
process3 = pd.DataFrame(list(map(lambda x:x.features,process2)))
#same as aboved

kdata = data.toPandas()
df = pd.concat([kdata, process3], axis=1)
#use the prediction result as the label
df = df.drop(["features"], axis=1)
#drop feature

id = np.arange(0,1564,1)
df.insert(0,'ID', id)
df.insert(21, 'label', label)
#insert ID column

df.to_csv("kmeansdata.csv", index =False, sep=',')
```

The processed dataset format would be as below:

ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	label
0	0	0.17025409	0	0	0	0	0	0.15970388	0	0	0	0	0	0	0	0	0	0.15521621	0	0	0
1	0.97173171	0.51076226	0.93129726	0.33294673	0.09652497	1.1284318	1.28963401	0.15970388	0.32842355	1.61443206	0.94025917	0.72511006	0.87336885	0.60731962	0.81353441	0.51439342	1.3634263	0.15521621	0	0.3450588	0
2	0.48586585	0.17025409	1.39694589	0.83236682	0.86872477	1.01558862	0.96722551	0.79851938	0.65684709	0.40360801	0.94025917	0.31076145	0.5822459	0.36439177	1.13894817	0.41151474	1.06044268	0.31043242	0.35143279	0	0
3	1.78150813	1.3620327	0.62086484	0.16647336	0.09652497	0.22568636	0.64481701	0	0.49263532	0.40360801	0.62683945	0	1.31005327	0.48585569	2.11518946	0.30863605	1.66640992	0.46564863	0	0.6901176	0
4	0.32391057	0	0	0	0.09652497	0	0.16120425	0	0	0.26907201	0	0.10358715	0.14556147	0	0	0.20575737	0	0	0	0	0
5	0.80977642	0.68101635	0.77608105	0	1.25482467	0.90274544	0.80602126	2.07615039	0.49263532	0.53814402	0.62683945	0.31076145	0.5822459	0.60731962	0.97624129	0.20575737	0.45447543	0.31043242	0.82000983	0.3450588	0
6	0.64782114	0.17025409	0.46564863	0.33294673	0.3860999	0.45137272	0.48361276	1.75674264	0.16421177	0.134536	0.31341972	0.2071743	0.14556147	0	0.81353441	0.10287868	0	0.15521621	0.11714426	0	0
7	0.16195528	0	0.15521621	0.33294673	0.3860999	0.11284318	0.16120425	0	0.16421177	0.134536	0	0.10358715	0	0	0.16270688	0.30863605	0.30298362	0.62086484	0.23428852	0.5175882	0
8	0.32391057	1.19177861	0	0	0	0.33852954	1.12842976	0.47911163	0.16421177	0.94175203	0.31341972	0.2071743	0	0	0.65082753	0.10287868	0.45447543	0.15521621	0	0.5175882	0
9	0	0.34050817	0.31043242	0	0.09652497	0	0.16120425	0.31940775	0	0	0.31341972	0.2071743	0.29112295	0.24292785	0	0.20575737	0.60596724	0.15521621	0.23428852	0	0
10	0.16195528	0.68101635	0	0.16647336	0.09652497	0.5642159	0.16120425	0.31940775	0.65684709	0.40360801	0.15670986	0.2071743	0.43668442	0.60731962	0.65082753	0.20575737	0.75745905	0.46564863	0	0.1725294	0
11	1.13368699	0.51076226	0.31043242	0.16647336	0.09652497	0.22568636	0	0.63881551	0.65684709	0.40360801	0.78354931	0.31076145	0.5822459	0.48585569	0.65082753	0.10287868	0.90895087	0.31043242	0.23428852	0.6901176	0
12	1.61955285	0.34050817	1.39694589	0.49942009	0.19304995	0.90274544	0.96722551	0.15970388	0.32842355	1.07628804	1.09696904	0.62152291	1.01893032	0.85024746	0.65082753	0.51439342	0.60596724	1.08651347	0.46857705	0.5175882	0
13	0.80977642	2.38355722	0.77608105	0.66589346	0.57914985	0.67705908	0.96722551	0.31940775	2.62738837	2.01804007	0.47012959	1.03587151	0.72780737	0.60731962	1.95248258	1.13166553	1.06044268	0.77608105	1.40573114	1.89782341	0
14	1.45759756	0.68101635	0.77608105	1.16531355	0.19304995	0.90274544	0.3224085	0.31940775	0.82105886	0.67268002	1.56709862	0.4143486	0.43668442	0.72878354	1.13894817	0.51439342	1.06044268	0.46564863	1.05429836	1.20770581	0
15	0.80977642	1.19177861	0.62086484	0.49942009	0.48262487	0.11284318	0.96722551	0.31940775	0.32842355	0.80721603	0.78354931	0.82869721	0.5822459	0.60731962	0.32541376	0.41151474	0.30298362	0.31043242	0.23428852	0.3450588	0
16	1.13368699	0.34050817	0.15521621	0	0.48262487	0.5642159	0	0	0.65684709	0.134536	0.47012959	0.2071743	0.14556147	0.48585569	0.81353441	0.10287868	0.15149181	0	0.23428852	0.1725294	0
17	0.48586585	0.51076226	1.24172968	0.83236682	0.86872477	0.90274544	0.3224085	0.95822326	0.65684709	0.53814402	0.94025917	0.4143486	1.1644918	0.60731962	0.97624129	0.72015079	0.30298362	0.62086484	0.23428852	0.5175882	0
18	0.16195528	0	0	0.16647336	0.19304995	0.45137272	0.48361276	0	0.49263532	0.26907201	0.31341972	0.2071743	0.43668442	0	0.32541376	0	0.45447543	0.77608105	0	0	0
19	0.64782114	0.51076226	0.31043242	0.16647336	0.3860999	0.78990226	1.28963401	0.15970388	0.49263532	0.26907201	0.94025917	0.2071743	1.89229917	0.48585569	0.81353441	0.41151474	1.81790173	0.46564863	0.58572131	1.03517641	0
20	0.64782114	0.68101635	0.62086484	2.3306271	0.57914985	1.01558862	0.96722551	0.15970388	1.14948241	0.94175203	0.78354931	0.51793576	1.01893032	0.60731962	0.16270688	0.20575737	0.30298362	0.62086484	1.05429836	0.5175882	0

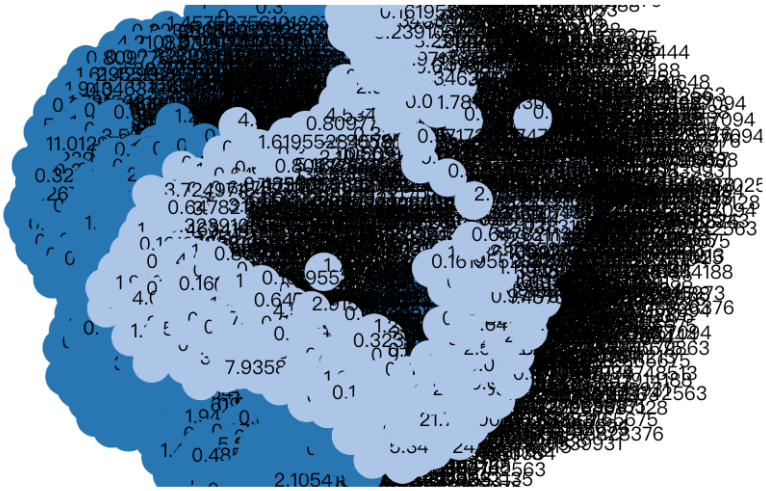
Then we will use the html file we designed to do visualization:

```
<!DOCTYPE html>

<head>
    <meta charset="utf-8">
    <script src="https://d3js.org/d3.v4.min.js"></script>
    <script src='https://rawgit.com/karpathy/tsnejs/master/tsne.js'></script>
</head>

<body>
<script>
    const width = 960,
        height = 500,
        margin = 40,

        colorScale = d3.scaleOrdinal(d3.schemeCategory20),

        centerx = d3.scaleLinear()
            .range([width / 2 - height / 2 + margin, width / 2 + height / 2 - margin]),
        centery = d3.scaleLinear()
            .range([margin, height - margin]);

    d3.csv('kmeansdata.csv', function (src_data) {

        const data = src_data.map((d, i) => Object.values(d));

        const canvas = d3.select("body").append("canvas")
            .attr("width", width)
            .attr("height", height);

        const model = new tsnejs.tSNE({
            dim: 2,
            perplexity: 30,
        });

        var features = data.map(function(value, index) { return value.slice(1, -1); });
        model.initDataRaw(features);

        const forcetsne = d3.forceSimulation(
            data.map(
                d => (d.x = width / 2, d.y = height / 2, d)
            )
        )
        .alphaDecay(0.005)
        .alpha(0.1);
```

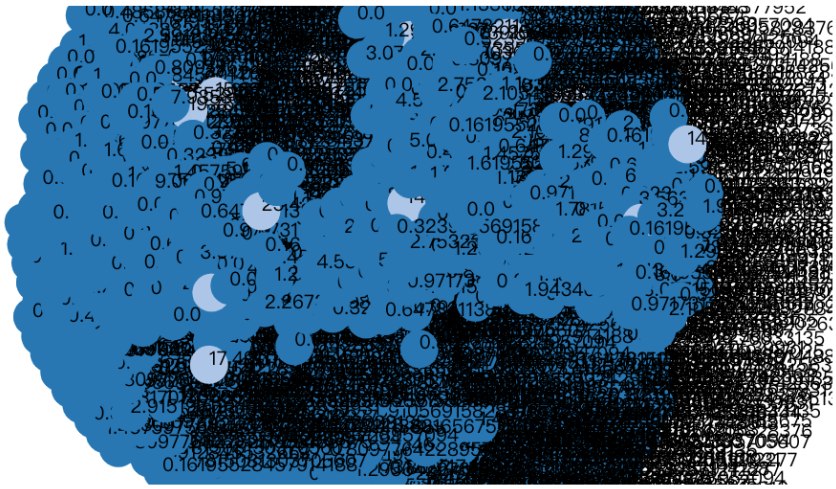
```
            forcetsne.force('tsne', function (alpha) {
                model.step();

                let pos = model.getSolution();

                centerx.domain(d3.extent(pos.map(d => d[0])));
                centery.domain(d3.extent(pos.map(d => d[1])));

                data.forEach((d, i) => {
                    d.x += alpha * (centerx(pos[i][0]) - d.x);
                    d.y += alpha * (centery(pos[i][1]) - d.y);
                });
            })
            .force('collide', d3.forceCollide().radius(d => 10));

            forcetsne.on('tick', function () {

                let nodes = data.map((d, i) => {
                    return {
                        x: d.x,
                        y: d.y,
                        color: colorScale(d[d.length - 1]),
                        text: d[0]
                    };
                });

                draw(canvas, nodes);

            });

        function draw(canvas, nodes) {
            let context = canvas.node().getContext("2d");
            context.clearRect(0, 0, width, height);

            for (var i = 0, n = nodes.length; i < n; ++i) {
                var node = nodes[i];
                context.beginPath();
                context.moveTo(node.x, node.y);
                context.arc(node.x, node.y, 20, 0, 2 * Math.PI);

                context.lineWidth = 0.5;
                context.fillStyle = node.color;
                context.fill();

                context.fillStyle = "black";
                context.font = "20px Open Sans";
                context.fillText(node.text,node.x,node.y);
            }
        }

    });
</script>
</body>
```

Visualization result:

Rawdata:



Kmeans data:

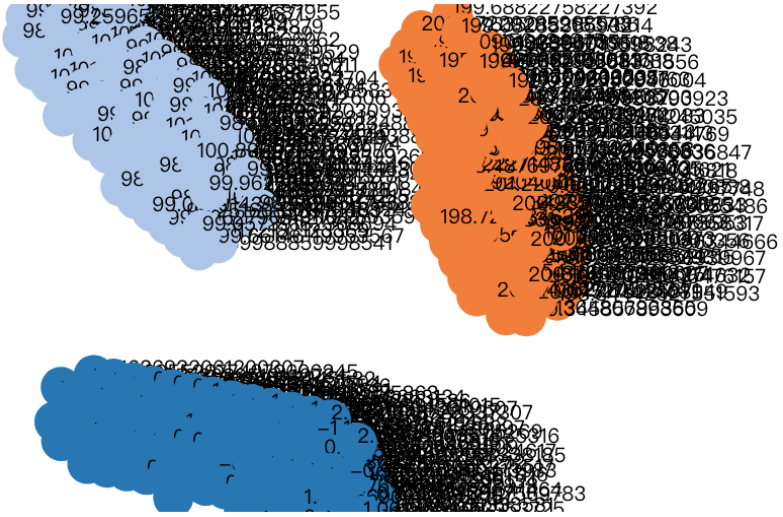


The dataset we choose have some problems, so that the data cannot be clustered well(the prediction of the dataset based on features is not well). In order to prove the correctness of the data visualization, we will 3 random dataset to show the cluster.

In [1]:

```
import numpy as np
import csv
import pandas as pd

#generate three random dataset
data0 = np.random.multivariate_normal([0]*20, np.identity(20), 100)
data1 = np.random.multivariate_normal([1]*20, np.identity(20), 100)
data2 = np.random.multivariate_normal([2]*20, np.identity(20), 100)
data_temp = np.vstack((data0, data1))
data = np.vstack((data_temp, data2))

id = np.arange(0,300,1)

label0 = np.zeros(100)
label1 = label0+1
label2 = label0+2
# each datset labeled with 0, 1, 2
label_temp = np.append(label0,label1)
label = np.append(label_temp, label2)

df = pd.DataFrame(data)
df.insert(0,'id', id)
df.insert(21, 'label', label)

df.to_csv("data.csv",index=False,sep=',')
```

The clustering result is as below:



As can be see above, the dataset is divided into three clusters well.

2. The app.js will be used to communicate with the http in the server. The app will call the twitter API to fetch the twitter data (we just need to know consumer_key, consumer_secret, access_token, access_token_secret). Then the app can listen to the Twitter to get the data with the track item and send the data to the index.html.

```
var twitter = require("twit"),
    credentials = require("./credentials.js"),
    express = require("express"),
    app = express(),
    server = require("http").createServer(app),
    io = require("socket.io").listen(server);

var t = new twitter({
    consumer_key : credentials.consumer_key,
    consumer_secret : credentials.consumer_secret ,
    access_token : credentials.access_token_key,
    access_token_secret : credentials.access_token_secret,
});

app.get("/", function(req, res){
    res.sendFile(__dirname + '/index.html');
});

server.listen(5000);

function constainsAny(str, substrings){
    for (var i = 0 ; i != substrings.length; i++){
        var substring = substrings[i];
        if (str.indexOf(substring) != -1){
            return substring;
        }
    }
    return null;
}

io.sockets.on("connection", function(socket){
    console.log("SOCKET CONNECTED\n");

    var track_item = ["#data","#girl","#love","#dog","#trump",
            "#movie","#music","#art","#game","#ball"];

    var twitterStream = t.stream(
        "statuses/filter",
        { track: track_item }
    );

    twitterStream.on("tweet", function(tweet){
        matchedHashtag = constainsAny(tweet.text, track_item)
        if(tweet && matchedHashtag){
            console.log(tweet.text)
            io.sockets.emit("stream",{ detail: tweet, hashtag: matchedHashtag });
        }
    });

    twitterStream.on("error", function(error){
        throw error;
     });
});
```

Package.json:

```
{
    "name": "twitter-hashtag-monitor",
    "description":"monotor hasgtag and show 20 latest items in realtime.",
    "version": "0.0.1",
    "private": true,
    "dependencies" :{
    "socket.io":"*",
    "express": "*",
    "twitter": "*",
    "twit": "^2.2.5"
    }
}
```

The track items we choose are: data, girl, love dog, trump, movie, music, game, and ball. The app can get the data from twitter as below(node app.js):

```
hz2558@big-data-analysis:~$ node app.js
SOCKET CONNECTED

RT @shadowscapes: "Gathering" #watercolor #painting #art #cat #cats #moonlight #moon #night #nightforest #woods #fa
irytale #fairytale #myth…
RT @kuriharan: Create the phone by your own customize senses via @Seeker

#tech #digital #data

@HaroldSinnott @DigitalVipul @CancerGeek…
RT @sivadigitalart: #Maari2 First Look Poster!
❤️❤️❤️
```

Then we will use html file to deal with the received data:

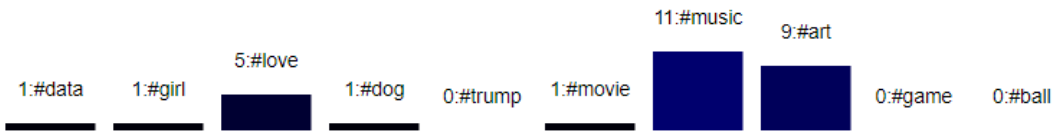
```
<!DOCTYPE html>
    <html>
    <head>
        <meta charset = "utf-8">
        <meta http-equiv="X-UA-Compatible" content= "IE=edge">
        <title>Twitter Hashtag Monitoring with Node JS, Socket.io, Express, ntwitter</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.js"></script>
        <script src="/socket.io/socket.io.js"></script>
        <script src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            var track_item = ["#data","#girl","#love","#dog","#trump",
            "#movie","#music","#art","#game","#ball"];
            var count = {}
            track_item.forEach(function(term){
                count[term] = 0;
            });
            var w = 600;
            var h = 600;
            var barPadding = 10;
            var svg = d3.select("body")
                .append("svg")
                .attr("width",w)
                .attr("height",h);


            var dataset = Object.keys(count).map(function(key){
                return count[key];

            });
               svg.selectAll("rect")
               .data(dataset)
               .enter()
               .append("rect")
               .attr("x",function(d,i){
                    return i * (w /dataset.length);
               })
               .attr("y",function(d){
                    return h - (d*4);
               })
               .attr("width",w / dataset.length-barPadding)
               .attr("height",function(d){
                    return d*4;
               })
               .attr("fill",function(d){
                return "rgb(0, 0, " + (d * 10)+ ")";
               });
```

```
    svg.selectAll("text")
       .data(dataset)
       .enter()
       .append("text")
       .text(function(d){
            return d;
       })
       .attr("text-anchor", "middle")
       .attr("x",function(d,i){
            return i * (w/dataset.length) * (w/dataset.length-barPadding)/2;
       })
       .attr("y", function(d){
            return h;
       })
       .attr("font-family","sans-serif")
       .attr("font-size","11px")
       .attr("fill","white");

    $(document).ready(function(){
        var socket = io.connect("http://104.196.116.201:5000");
        socket.on("stream", function(data){
            console.log(count)

            count[data.hashtag] += 1;

            var dataset = Object.keys(count).map(function(key){
                return count[key];
            });

        svg.selectAll("rect")
       .data(dataset)
       .transition()
       .attr("x", function(d,i){
            return i * (w/dataset.length);
       })
       .attr("y", function(d){
            return h - (d*4);
       })
       .attr("width", w/dataset.length - barPadding)
       .attr("height",function(d){
            return d*4;
       })
       .attr("fill", function(d){
            return "rgb(0, 0, " + (d*10) + ")";
       });

        svg.selectAll("text")
          .data(dataset)
          .transition()
          .text(function(d, i){
              return d + ":" + track_item[i];
          })
          .attr("text-anchor", "middle")
          .attr("x",function(d,i){
              return i * (w/dataset.length) + (w/dataset.length-barPadding)/2;
          })
```

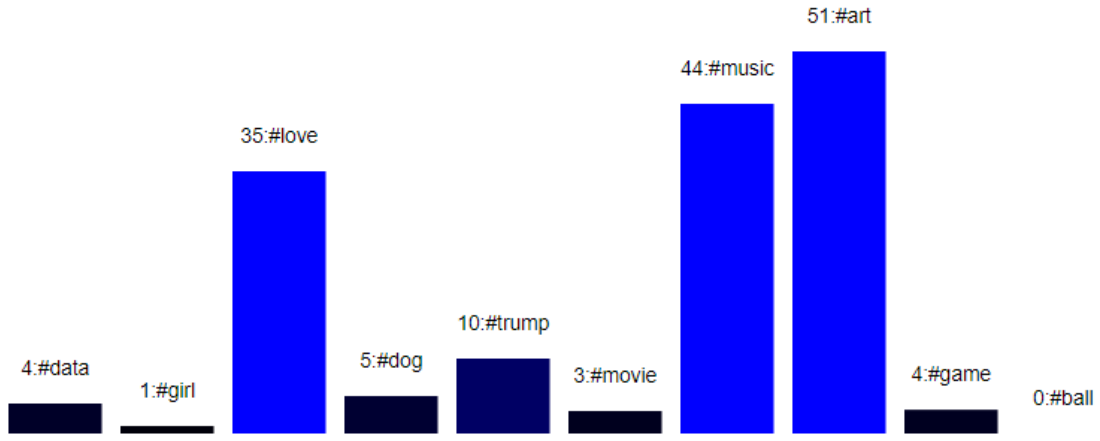
```
                .attr("y",function(d,i){
                    return h - (d*4) - 15;
                })
                .attr("font-family","sans-serif")
                .attr("font-size","11px")
                .attr("fill","black");
                });

            });
        </script>
        <ul id = "tweets"></ul>
</body>
</html>
```

Then we can display real-time stream of Tweets that the hashtags in a query bar.

Initial:



Finally:



As can be seen, the art, music and love are the top3 topics(51, 44, 35 seperately). And the ball topic is not very popular.

3. The html file for Q3

(1)

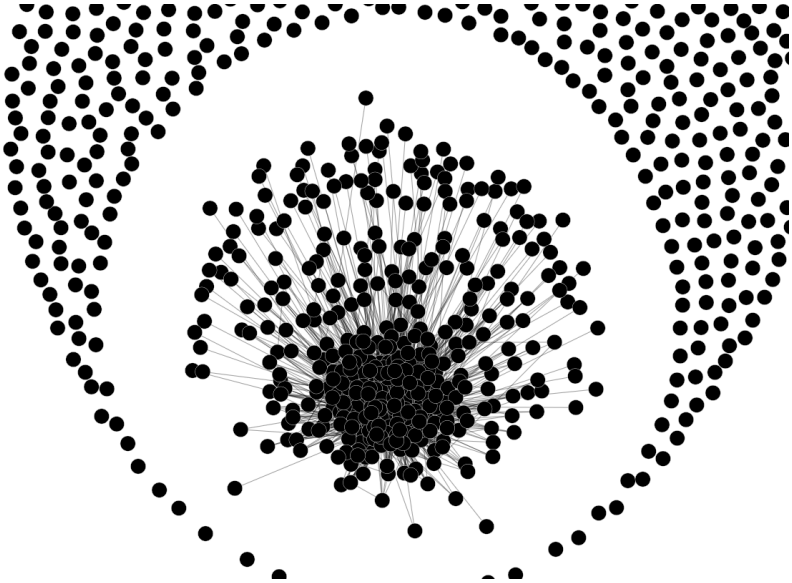
Preprocess Data:

```
V = spark.read.csv(vfile)
E = spark.read.csv(efile)

v = V.select(V._c1,V._c2).selectExpr("_c1 as id","_c2 as features")
e = E.select(E._c1,E._c2,E._c3.cast("float")).selectExpr("_c1 as src","_c2
as dst","_c3 as similarity").filter("similarity > 0.95")
g = GraphFrame(v, e)

PR = g.pageRank(resetProbability = 0.15, tol = 0.01)
pr = PR.vertices.select("id", "PageRank")

sc.setCheckPointDir("./checkpoint")
CC = g.connectedComponents()
cc = CC.select("id", "Component")

import pandas as pd
pr = pr.toPandas()
cc = cc.toPandas()
result = pd.merge(pr, cc, on = "id")
result.to_csv("/home/hz2558/node.csv")
```

Above is to get node.csv

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <script src="https://d3js.org/d3.v4.min.js"></script>
    <script src='https://rawgit.com/karpathy/tsnejs/master/tsne.js'></script>
</head>
<body>
    <script type="text/javascript">
        // var width = 960, height = 500;
        var width = 6000, height = 8000;

        var svg = d3.select("body").append("svg")
            .attr("width", width)
            .attr("height", height);

        var g = svg.append("g").attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

        var radiusScale = d3.scaleLinear()
            .domain([0, 2])
            .range([5, 30])

        d3.csv('node.csv', function (node_data){
            d3.csv('edge.csv', function (edge_data){
                // DO SOME THINGS HERE
                // Initialize force simulation
                const nodes = node_data;
                const links = edge_data.map(function (a) {
                    return {source: Number(a.source), target: Number(a.target)}
                });

                console.log(nodes)
                console.log(links)

                var simulation = d3.forceSimulation(nodes)
                    .force("charge", d3.forceManyBody().strength(-500))
                    .force("link", d3.forceLink(links).distance(20).strength(0.01).iterations(10))
                    .force("x", d3.forceX())
                    .force("y", d3.forceY())
                    .on('tick', tick);

                // Set attributes for line and node
                var lineG = g.append("g")
                        .attr("stroke", "#000")
                        .attr("stroke-width", 1.5)
                    .selectAll("line")
                    .data(links)
                    .enter().append("line");

                var nodeG = g.append("g")
                        .attr("stroke", "#fff")
                        .attr("stroke-width", 1.5)
                    .selectAll("circle")
                    .data(nodes)
                    .enter().append("circle");
```

```
        function tick(){
            lineG.attr("x1", function(d) {return d.source.x;})
                 .attr("y1", function(d) {return d.source.y;})
                 .attr("x2", function(d) {return d.target.x;})
                 .attr("y2", function(d) {return d.target.y;});
            nodeG.attr("cx", function(d) {return d.x;})
                 .attr("cy", function(d) {return d.y;})
                 .attr("r", function(d) {return radiusScale(+2;});

        }

        var colorScale = d3.scaleOrdinal(d3.schemeCategory20);
        nodeG.style("fill", function(d) {
            console.log(d.Component)
            if (d.Component == 0){
                return colorScale(d.Componet)
            }
        });
    })
})
</script>
</body>
</html>
```



(2)

If the PageRank of the node is larger, then the radius of the circle will be bigger. The main component will be specified.

```
                // Larger radius for nodes with larger PageRank
                function tick (){
                    lineG.attr("x1", function(d) {return d.source.x; })
                        .attr("y1", function(d) {return d.source.y; })
                        .attr("x2", function(d) {return d.target.x; })
                        .attr("y2", function(d) {return d.target.y; });
                    nodeG.attr("cx", function(d) {return d.x; })
                        .attr("cy", function(d) {return d.y; })
                        .attr("r", function(d) {return radiusScale(+d.PageRank); });
                }

                // Same color for nodes within same components
                var colorScale = d3.scaleOrdinal(d3.schemeCategory20);
                nodeG.style("fill", function (d){
                    console.log(d.Component)
                    if (d.Component == 0){
                        return "#0080FF"
                    }else{
                        return "grey"
                    }

                });


            });
        });
    </script>
</body>
</html>
```