Cognitive Algorithms - Assignment 5 (30 points)

Cognitive Algorithms
Summer term 2018
Technische Universität Berlin
Fachgebiet Maschinelles Lernen

Due on July 4, 2018 10am via ISIS

After completing all tasks, run the whole notebook so that the content of each cell is properly displayed. Make sure that the code was ran and the entire output (e.g. figures) is printed. Print the notebook as a PDF file and again make sure that all lines are readable - use line breaks in the Python Code '\' if necessary. Points will be deducted, if code or content is not readable!

Upload the PDF file that contains a copy of your notebook on ISIS.

Group: Group08
Members:

- Chen, Yang
- Liu, Huiran
- Smejkal, Karel
- Tian, Qihang
- Arat, Emrecan

Part 1: Multiple Choice Questions (5 points)

A) Which statements about unsupervised learning are true?

- [ ] Its goal is to learn a mapping from input data to output data
- [x] Its goal is to find structure in the data
- [ ] It needs labels for training
- [x] It does not need labels for training

B) Which of the following methods solve a supervised learning problem?

- [x] Linear Discriminant Analysis
- [x] Nearest Centroid Classifier
- [ ] Non-negative Matrix Factorization
- [ ] K-Means Clustering
- [x] Perceptron
- [x] Ordinary Least Squares
- [x] (Kernel) Ridge Regression
- [ ] Principal Component Analysis

C) Which statement about the Principal Components Analysis is **not** true?

- [ ] PCA finds the direction that maximizes the variance of the projected data
- [ ] The PCs are uncorrelated
- [x] The first k PCs are the eigenvectors corresponding to the smallest k eigenvalues

D) Cross-Validation can be used to ...

- [ ] ... estimate the generalization error
- [x] ... find optimal parameter values

E) Nested Cross-Validation can be used to ...

- [x] ... estimate the generalization error
- [x] ... find optimal parameter values

2.1

Part 2: Programming (25 points)

Task 1: Principal Component Analysis (16 points)

In this assignment, you will detect trends in text data and implement Principal Component Analysis (PCA). The text data consists of preprocessed news feeds gathered from http://beta.wunderfacts.com/ (http://beta.wunderfacts.com/) in October 2011, and you will be able to detect a trend related to Steve Jobs death on 5th October 2011.

The data consists of 26800 Bag-of-Words (BOW) features of news published every hour, i.e. the news are represented in a vector which contains the occurence of each word. Here we have many more dimensions (26800) than data points (645). This is why we will implement Linear Kernel PCA instead of standard PCA.

Download the data set `newsdata.npz`, if not done yet.

In [1]:

```
import numpy as np
import pylab as pl
import scipy as sp
%matplotlib inline
```

In [6]:

```
def pca(X,ncomp=10):
    ''' Principal Component Analysis
    INPUT:  X       - DxN array of N data points with D features
            ncomp   - number of principal components to estimate
    OUTPUT: W       - D x ncomp array of directions of maximal variance,
                     sorted by their eigenvalues
            H       - ncomp x N array of projected data '''
    ncomp = min(np.hstack((X.shape, ncomp)))
    #center the data
    X=X-np.mean(X,axis=1).reshape(len(X),1)

    # compute linear kernel
    K=np.dot(X.T,X)

    # compute eigenvectors and sort them according to their eigenvalues
    d,u=np.linalg.eigh(K)
    idx=np.argsort(-d)
    d=d[idx]
    u=u[:,idx]

    # compute W and H
    W=np.dot(X,u[:,:ncomp])
    H=np.dot(W.T,X)

    return W,H

def get_data(fname='newsdata_BOW.npz'):
    foo = np.load(fname,encoding = 'latin1')
    dates = foo['dates']
    BOW = np.array(foo['BOW_features'].tolist().todense())
    words = foo['words']
    return BOW,words,dates

def nmf(X,ncomp=10,its=100):
    '''Non-negative matrix factorization as in Lee and Seung http://dx.doi.org/1
0.1038/44565
    INPUT:  X       -  DxN array of N data points with D features
            ncomp   - number of factors to estimate
            its     - number of iterations
    OUTPUT: W       - D x ncomp array
            H       - ncomp x N array '''
    ncomp = min(np.hstack((X.shape, 10)))
    X = X + 1e-19
    # initialize randomly
    W = sp.random.rand(X.shape[0],ncomp)
    H = sp.random.rand(X.shape[1],ncomp).T
    # update for its iterations
    for it in sp.arange(its):
        H = H * (W.T.dot(X)/(W.T.dot(W.dot(H))))
        W = W * (X.dot(H.T)/(W.dot(H.dot(H.T))))
    return W,H

def plot_trends(ntopics=8,method=nmf,topwhat=10):
    #load data
    BOW,words,dates = get_data()
    topics,trends = method(BOW,ntopics)
    for itopic in range(ntopics):
        pl.figure(figsize=(8,6))
        pl.plot(trends[itopic,:].T)
```

```
        ranks = (-abs(topics[:,itopic])).argsort()
        thislabel = words[ranks[:topwhat]]
        pl.legend([thislabel])
        days = sp.arange(0,BOW.shape[-1],24*7)
        pl.xticks(days,dates[days],rotation=20)

def test_assignment6():
    ##Example 1
    X = sp.array([[0, 1], [0, 1]])
    W, H = pca(X, ncomp = 1)
    assert(sp.all(W / W[0] == [[1], [1]]))
    print ('2 datapoint test passed')

    ##Example 2
    #generate 2D data
    N =100
    cov = sp.array([[10, 4], [4, 5]])
    X = sp.random.multivariate_normal([0, -20], cov, N).T
    #do pca
    W, H = pca(X)
    #plot result
    pl.figure()
    pc0 = 10*W[:,0] / np.linalg.norm(W[:,0])
    pc1 = 10*W[:,1] / np.linalg.norm(W[:,1])
    pl.plot([-pc0[0], pc0[0]], [-pc0[1]-20, pc0[1]-20], '-k', label='1st PC')
#     pl.hold(True)
    pl.plot([-pc1[0], pc1[0]], [-pc1[1]-20, pc1[1]-20], '-.r', label='2nd PC')
    pl.plot(X[0,:], X[1,:], '+', color='k')
    pl.axis('equal')
    pl.legend(loc=1)
```

A) (7 points) Implement Linear Kernel Principal Component Analysis by completing the function stub `pca`. Given data $X \in \mathbb{R}^{D \times N}$, PCA finds a decomposition of the data in k orthogonal principal components that maximize the variance in the data,

$$X = W \cdot H$$

with $W \in \mathbb{R}^{D \times k}$ and $H \in \mathbb{R}^{k \times N}$. The Pseudocode is given below. The function `test_assignment6` helps you to debug your code. It plots for a 2D data set the two principal components.

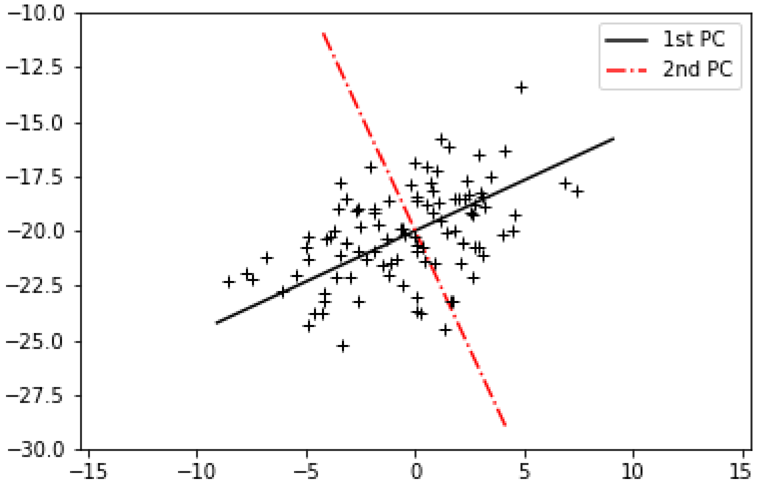
`PCA( X, k ):`

1. `# Require: data` $x_1, \ldots, x_N \in \mathbb{R}^d, N \ll d$ `, number of principal components` k
2. `# Center Data`
3. $X = X - 1/N \sum_i x_i$
4. `# Compute Linear Kernel`
5. $K = X^\top X$
6. `# Compute eigenvectors corresponding to the` k `largest eigenvalues`
7. $\alpha = \text{eig}(K)$
8. $W = X\alpha$
9. `# Project data onto` W
10. $H = W^\top X$
11. `return W, H`

In [3]:

```
test_assignment6()
```

2 datapoint test passed



B) (3 points) What happens when you forget to center the data in `pca`? Show the resulting plot for the 2D toydata example and explain the result.

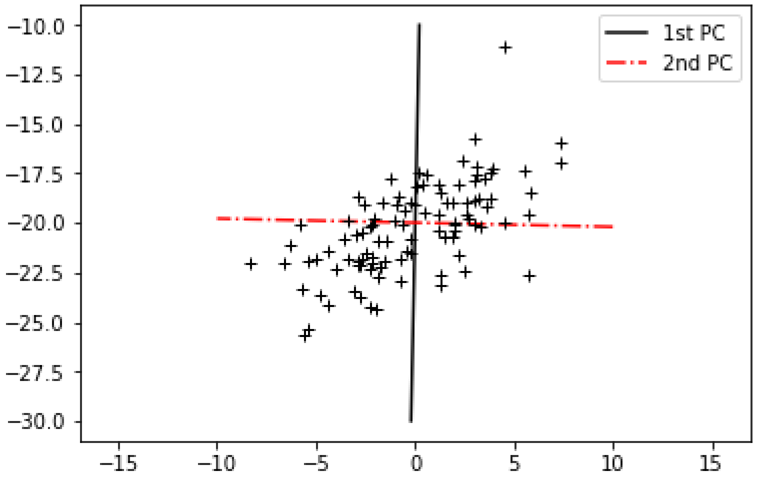
[Your answer for B) here]

The first PC isn't lie at the largest variance, and the second one is just orthogonal to the first one. Since the definition of covariance is $\operatorname{cov}(X, Y) = \operatorname{E}\left[(X - \operatorname{E}[X])(Y - \operatorname{E}[Y])\right]$, but in PCA, we apply it by $cov = XX^T$ with $\operatorname{E}[X] = 0$

In [5]:

```
#without centering the data in pca
test_assignment6()
```

2 datapoint test passed



C) (3 points) Suppose we only have two data points, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}$. What would be the principal directions $W = [\mathbf{w}_1, \mathbf{w}_2]$? What will be the variance of the projected data onto each of the principal components $\mathbb{V}(\mathbf{w}_1^T X)$, $\mathbb{V}(\mathbf{w}_2^T X)$? What is H?

Hint: You can obtain the result simply by visualizing the two data points and remembering PCA's objective. Or you can calculate the result using `standard` PCA. With Linear Kernel PCA, you will not be able to compute $\mathbf{w_2}$, because the corresponding eigenvalue is 0.

In [52]:

```
X=np.array([[0,0],[1,2]])
cov=np.dot(X,X.T)
d,W=np.linalg.eig(cov)
d=d[::-1]
W=W[::-1]
print(d)
print('W is\n',W)
print('variance:')
print(np.var(np.dot(w[:,0],X)))
print(np.var(np.dot(w[:,1],X)))
print('H is\n',np.dot(w.T,X))
```

```
[5. 0.]
W is
 [[0. 1.]
 [1. 0.]]
variance:
0.25
0.0
H is
 [[1. 2.]
 [0. 0.]]
```

[Your answer for C) here]

$\mathbf{w}_1 = [0, 1]$

$\mathbf{w}_2 = [1, 0]$

$\mathbb{V}(\mathbf{w}_1^T X) = 0.25$

$\mathbb{V}(\mathbf{w}_2^T X) = 0$

$H = [[1, 2], [0, 0]]$

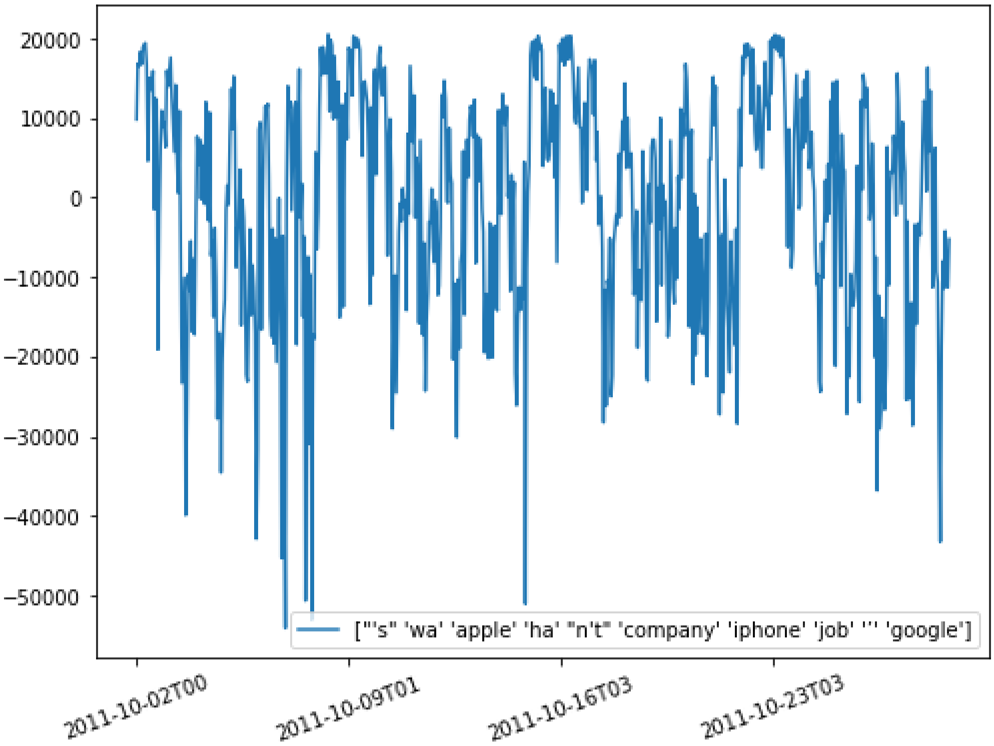
D) (3 points) Detect trends in the text data by calling the provided function `plot_trends` once for PCA and once for Non-Negative Matrix Factorization (NMF) (the code for NMF is provided as well). Which differences do you notice between the algorithms? Which method would you prefer for this task? Hand in the plot of the most prominent trend related to Steve Jobs death for each algorithm.

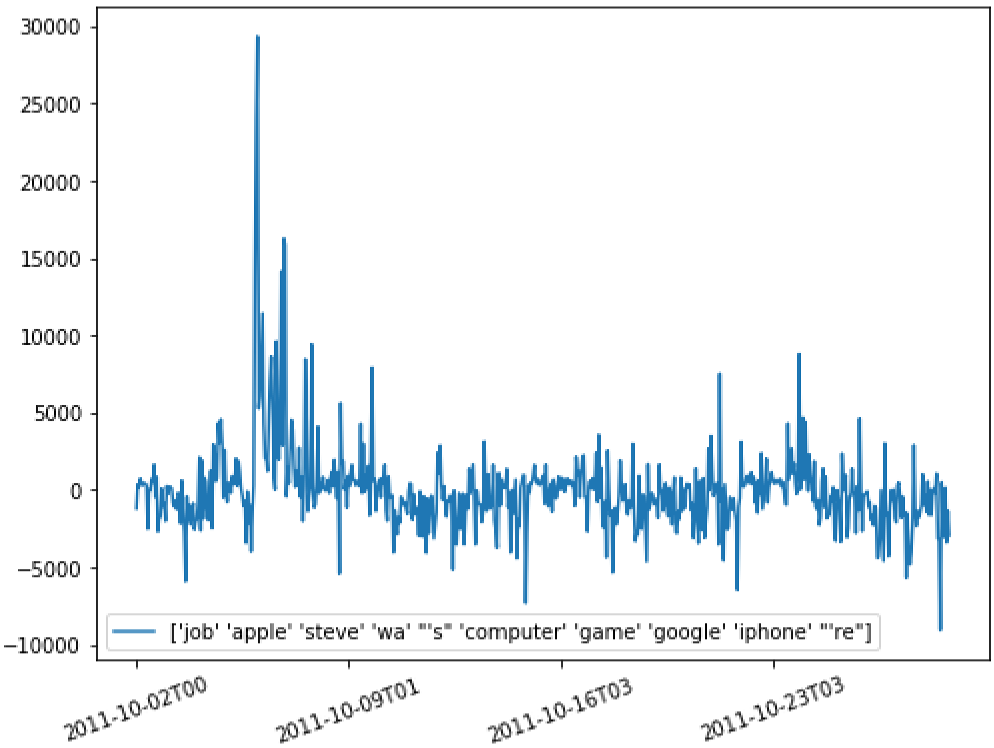
answer for D)

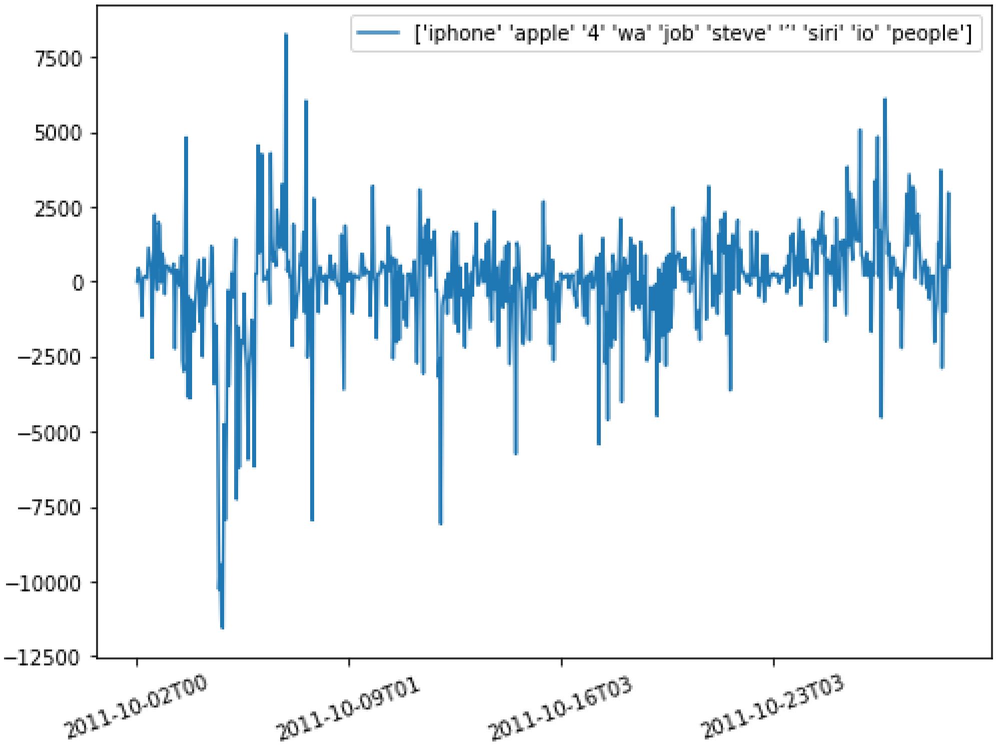
There are some negative values in PCA directions while NMF have the non negative. Since negative values is hard to interpret for the task, thus I prefer NMF for this task.

In [7]:

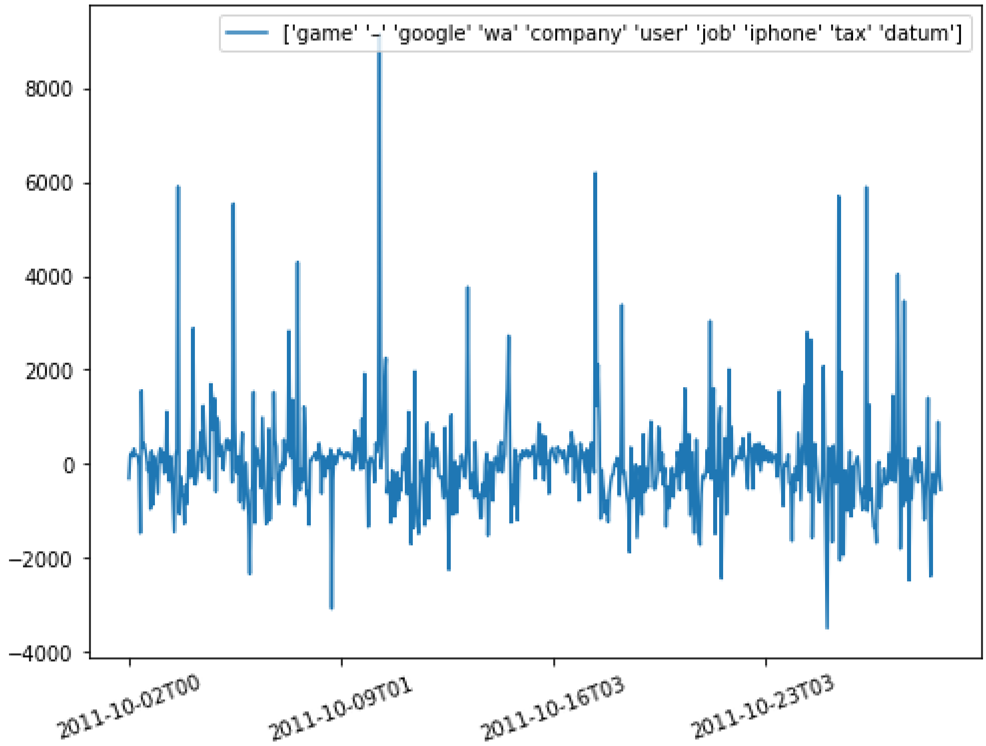
```
plot_trends(method=pca)
```

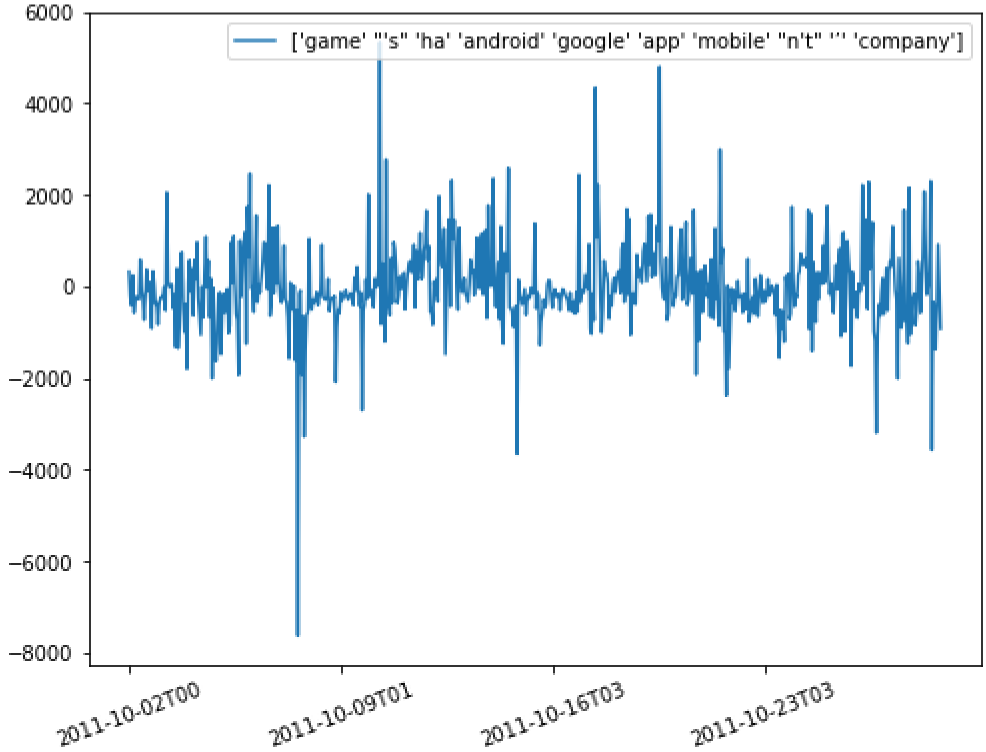

20000
10000
0
−10000
−20000
−30000
−40000
−50000
["'s" 'wa' 'apple' 'ha' "n't" 'company' 'iphone' 'job' "' 'google']
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

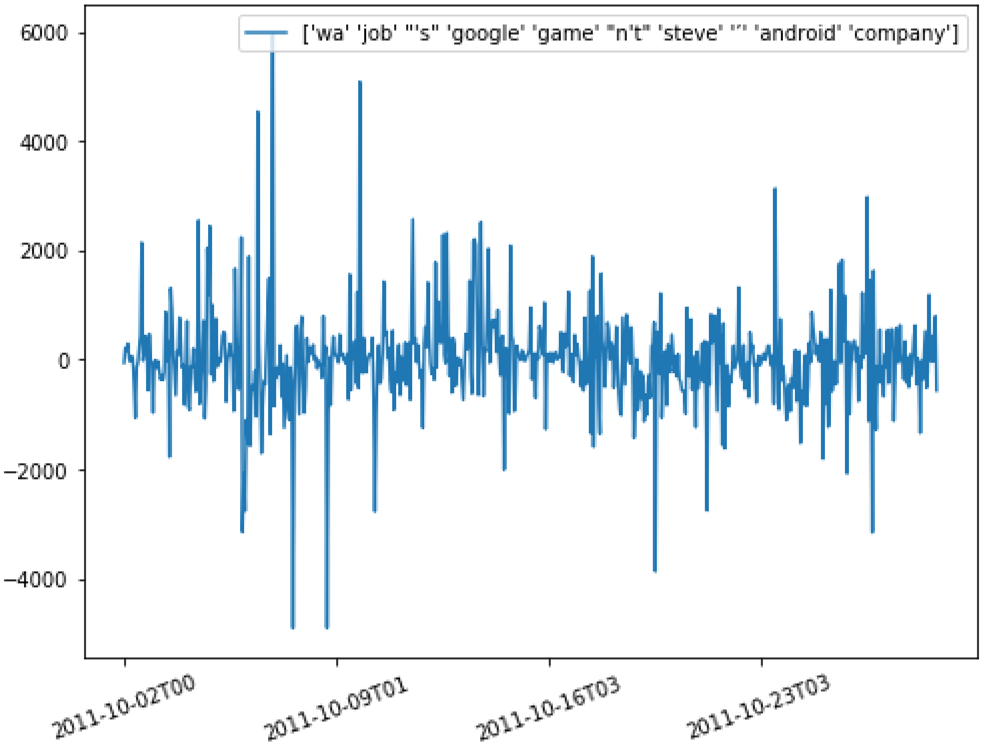

30000
25000
20000
15000
10000
5000
0
−5000
−10000
['job' 'apple' 'steve' 'wa' "'s" 'computer' 'game' 'google' 'iphone' "'re"]
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

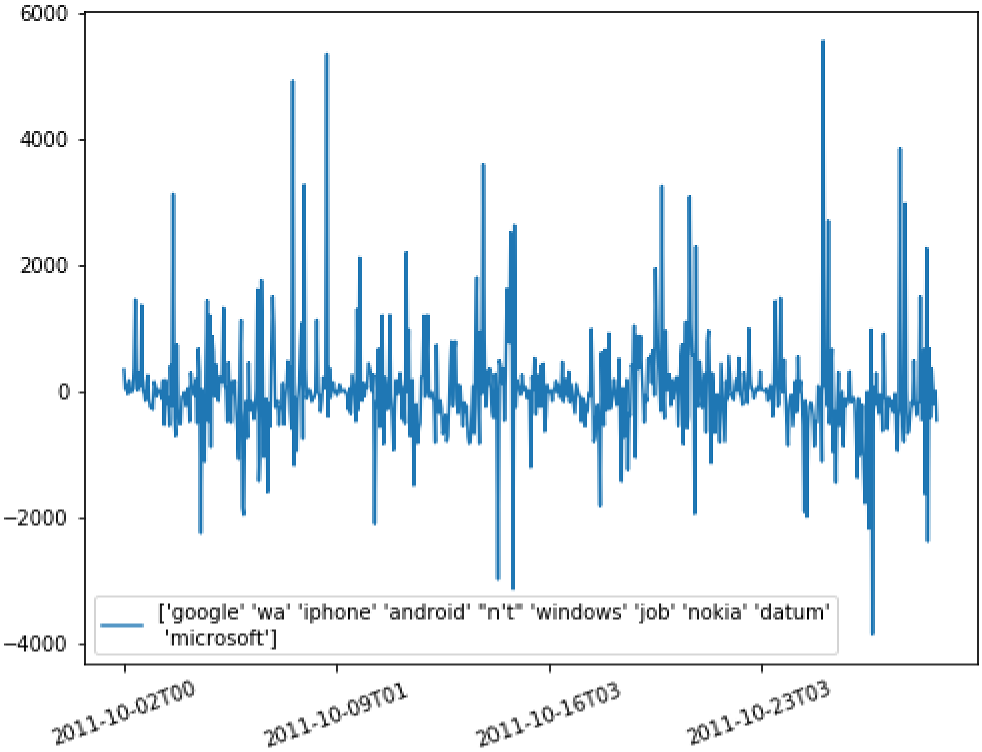

['iphone' 'apple' '4' 'wa' 'job' 'steve' '"' 'siri' 'io' 'people']
7500
5000
2500
0
−2500
−5000
−7500
−10000
−12500
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


['"' "'s" 'piece' 'job' 'sec' "n't" 'steve' 'music' 'company' 'orchestra']
5000
0
−5000
−10000
−15000
−20000
−25000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


['game' '-' 'google' 'wa' 'company' 'user' 'job' 'iphone' 'tax' 'datum']
8000
6000
4000
2000
0
−2000
−4000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

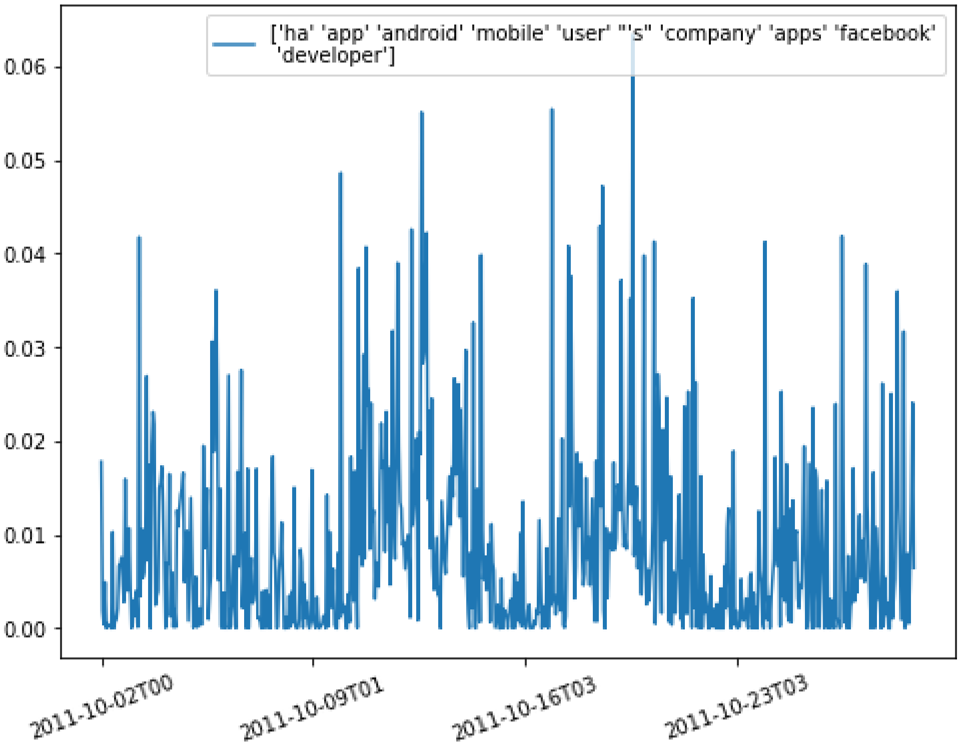

['game' "'s" 'ha' 'android' 'google' 'app' 'mobile' "n't" '' 'company']
6000
4000
2000
0
−2000
−4000
−6000
−8000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

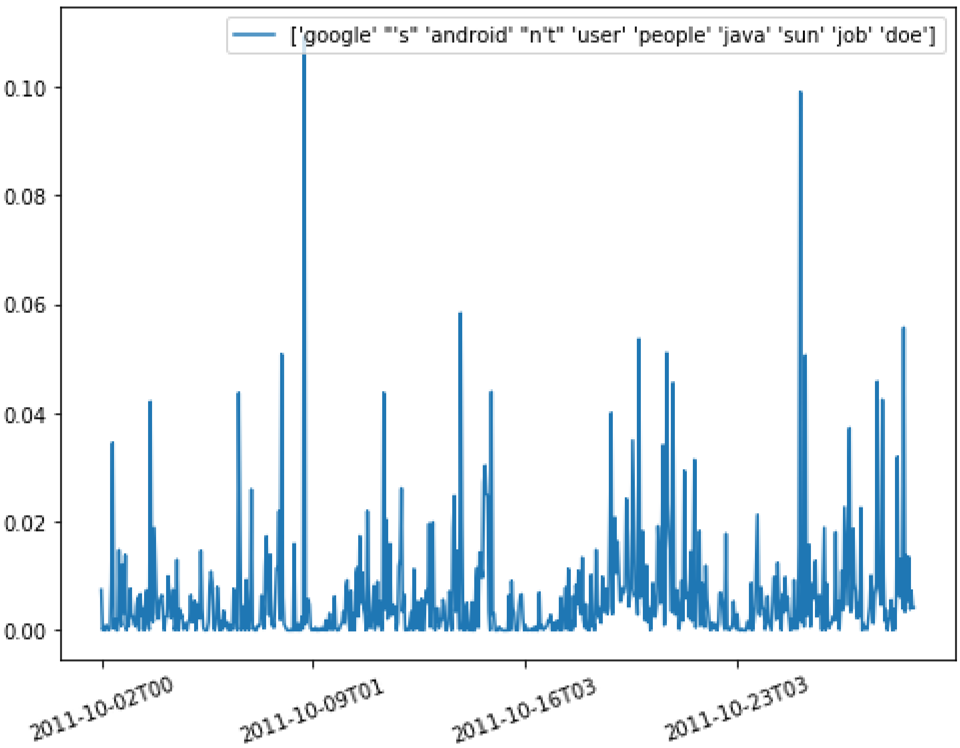

['wa' 'job' "'s" 'google' 'game' "n't" 'steve' '`' 'android' 'company']
6000
4000
2000
0
−2000
−4000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

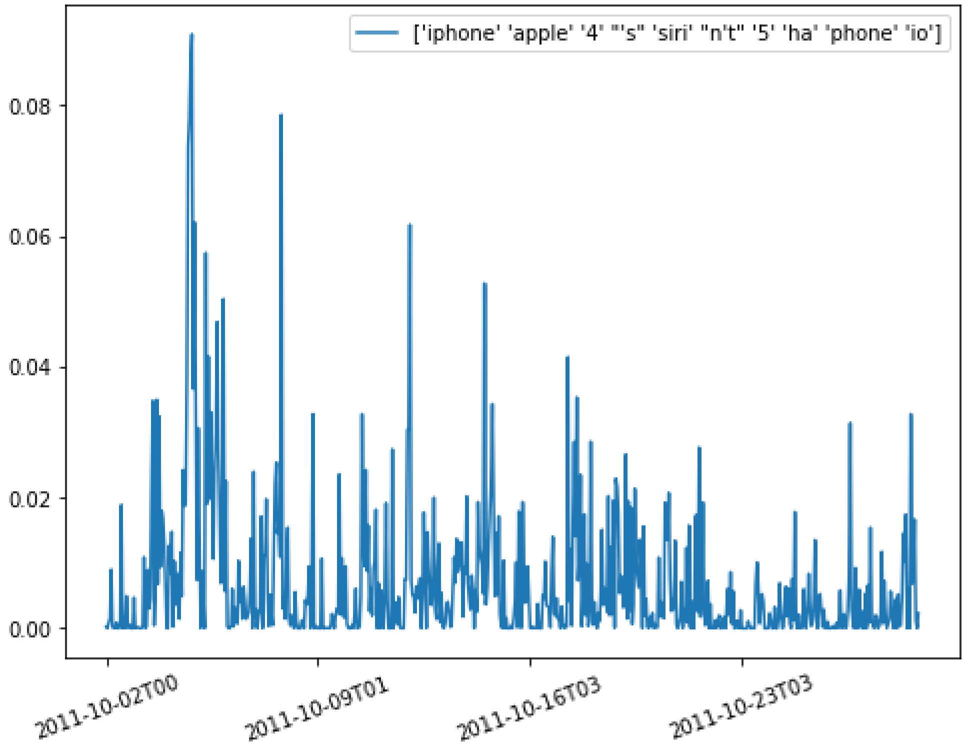

['google' 'wa' 'iphone' 'android' "n't" 'windows' 'job' 'nokia' 'datum'
'microsoft']
6000
4000
2000
0
−2000
−4000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

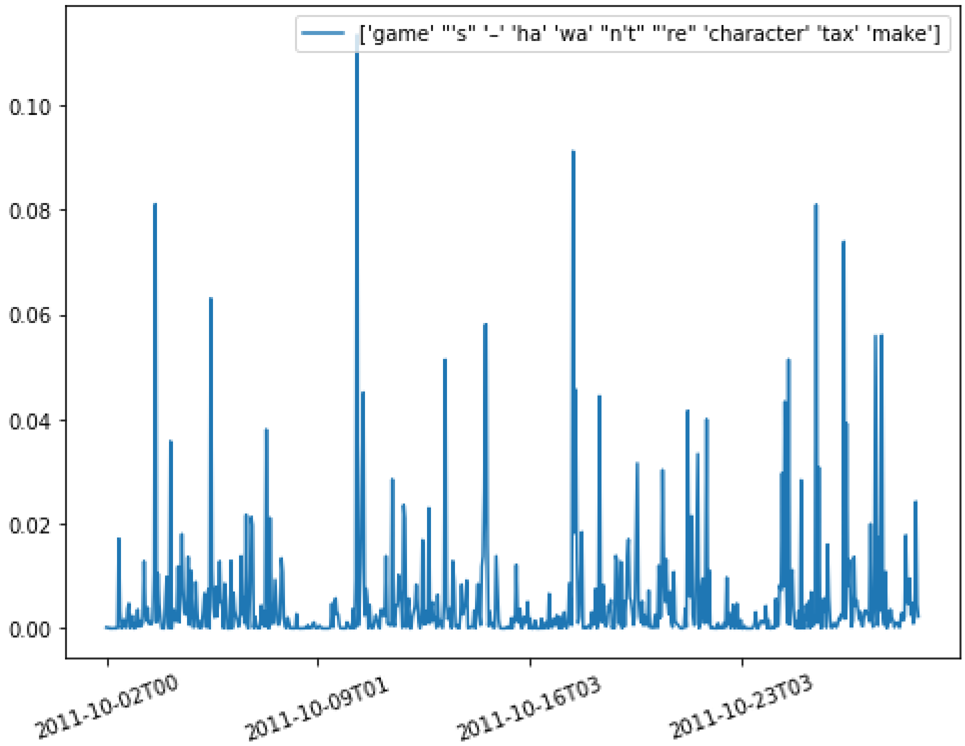
In [8]:

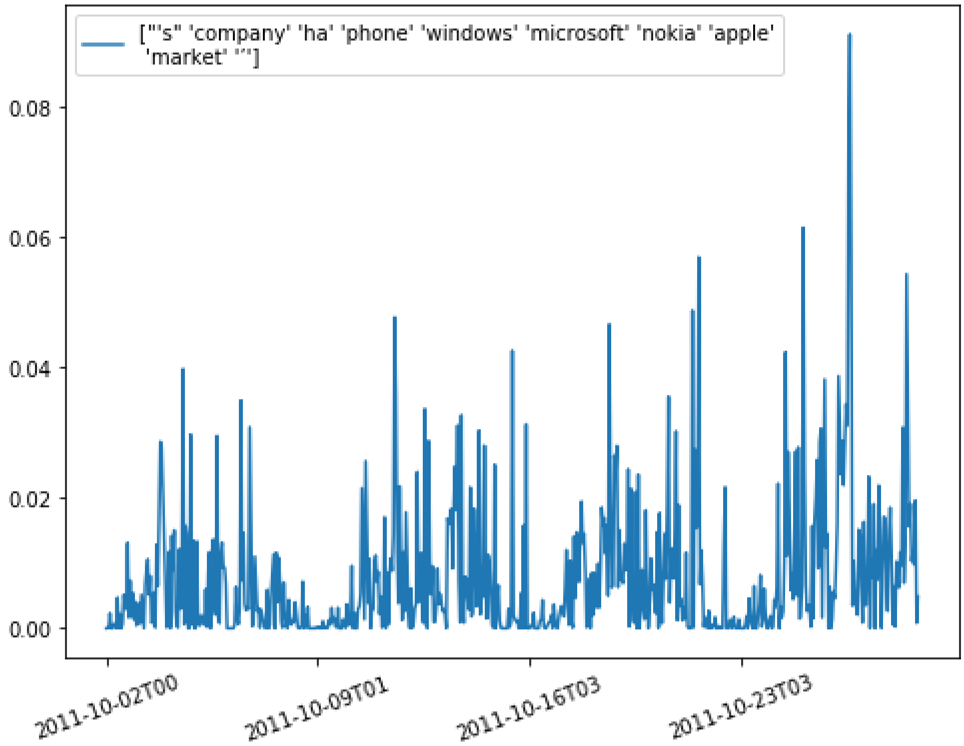
```
plot_trends(method=nmf)
```

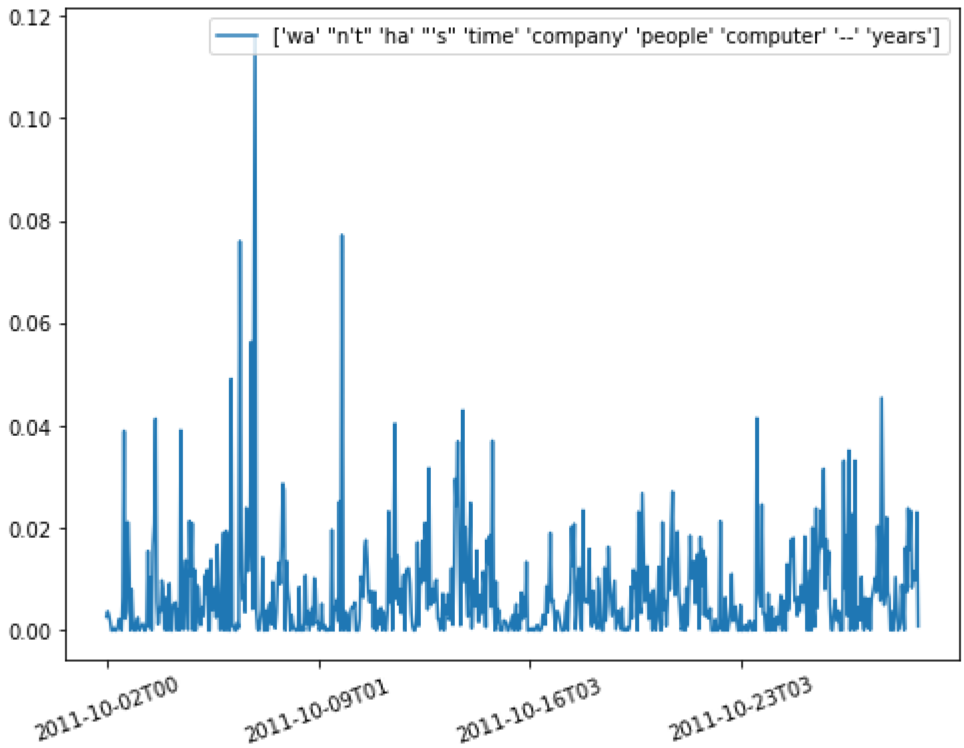

['ha' 'app' 'android' 'mobile' 'user' "'s" 'company' 'apps' 'facebook' 'developer']
0.06
0.05
0.04
0.03
0.02
0.01
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

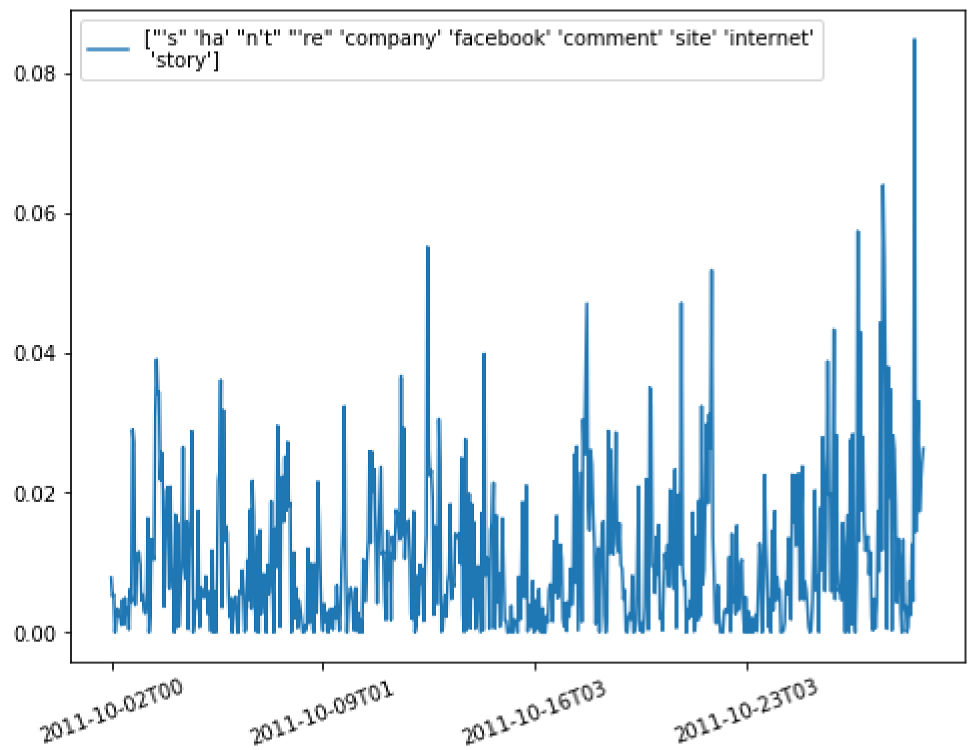

['google' "'s" 'android' "n't" 'user' 'people' 'java' 'sun' 'job' 'doe']
0.10
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

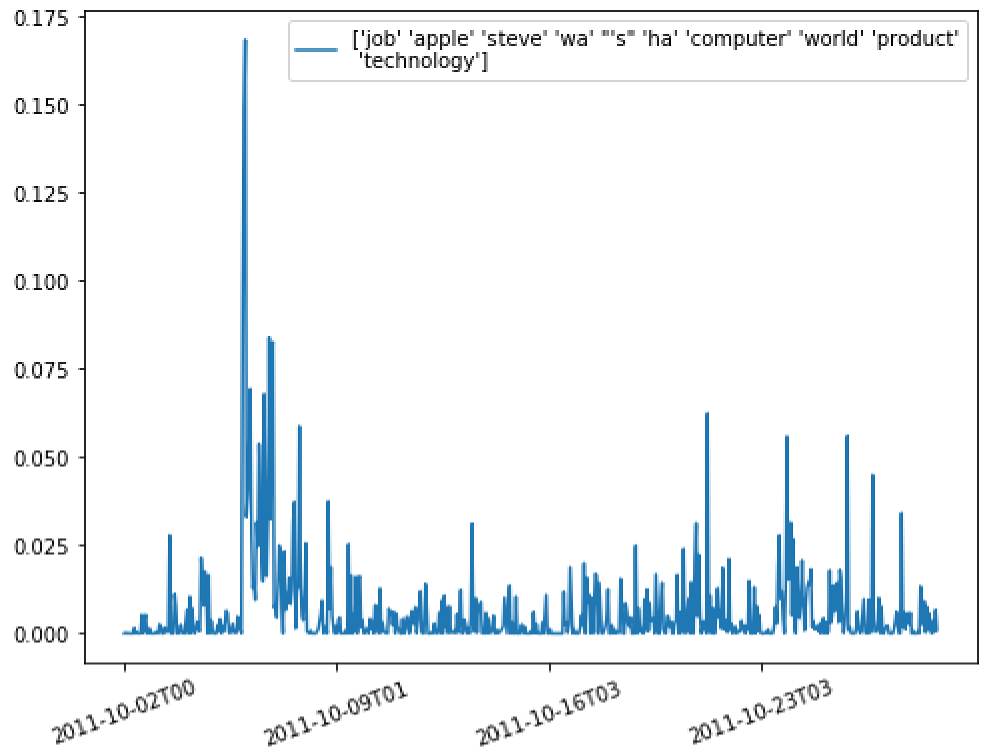

['iphone' 'apple' '4' "'s" 'siri' "n't" '5' 'ha' 'phone' 'io']
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


['game' "'s" '–' 'ha' 'wa' "n't" "'re" 'character' 'tax' 'make']
0.10
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


["'s" 'company' 'ha' 'phone' 'windows' 'microsoft' 'nokia' 'apple' 'market' '"']
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


['wa' "n't" 'ha' "'s" 'time' 'company' 'people' 'computer' '--' 'years']
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


["'s" 'ha' "n't" "'re" 'company' 'facebook' 'comment' 'site' 'internet' 'story']
0.08
0.06
0.04
0.02
0.00
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03


['job' 'apple' 'steve' 'wa' "'s" 'ha' 'computer' 'world' 'product' 'technology']
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
2011-10-02T00
2011-10-09T01
2011-10-16T03
2011-10-23T03

Task 2: K-Means Clustering (9 points)

In this exercise we want to implement the K-Means Clustering algorithm. It finds cluster centers $\mu_1 \dots \mu_K$ such that the distance of the data points to their respective cluster center are minimized. This is done by re-iterating two steps:

1. Assign each data point x_n to their closest cluster μ_k (for all $n = 1 \dots N$)
2. Update each cluster center μ_k to the mean of the members in that cluster k (for all $k = 1 \dots K$)

Complete the function `kmeans` (see Task 2.A to 2.D for more detail). `test_kmeans` helps you to debug your code. It generates a simple 2D toy dataset. Your `kmeans` implementation should correctly identify the three clusters and should converge after only a few iterations (less than 10).

A) (2 points) Initialize the centroids. To do so calculate the mean of the whole data set and add some standard normal distributed noise to it, i.e. for all $k = 1 \dots K$

$$\mu_k = \bar{x} + \epsilon_k$$

where $\bar{x}, \epsilon_k \in \mathbb{R}^D$ and $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x_n}$ and $\epsilon_k \sim \mathcal{N}(\mathbf{0}, I)$

B) (4 points) For step 1 of the algorithm, we need the distance between each data point x_n and each centroid μ_k. Complete the function `distmat` that calculates a matrix $Dist \in \mathbb{R}^{N,K}$ such that

$$Dist_{n,k} = ||x_n - \mu_k||_2^2$$

We can calculate the matrix $Dist$ without the use of for-loops by using following formula:

$$Dist = A - 2B + C$$

where $A_{n,k} = x_n^T x_n$, $B_{n,k} = x_n^T \mu_k$ and $C_{n,k} = \mu_k^T \mu_k$

C) (1 point) Assign each data point to its closest centroid. To do so, construct a matrix $Closest \in \mathbb{R}^{N,K}$ such that

$$Closest_{n,k} = \begin{cases} 1 & \texttt{if } \mu_k \texttt{ is the closest centroid to } x_k \\ 0 & \text{otherwise} \end{cases}$$

i.e. each row of $Closest$ holds only one non-zero element.

D) (2 points) Update each cluster center to the mean of the members in that cluster, i.e. for all $k = 1 \dots K$

$$\mu_k = \frac{1}{|\mathcal{X}_k|} \sum_{x \in \mathcal{X}_k} x$$

$$\mathcal{X}_k = \{x_n \in X \mid \text{the closest centroid to } x_n \text{ is } \mu_k\}$$

In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
%matplotlib inline
```

In [10]:

```
def kmeans(X, K, max_iter=50, eta=0.05):
    """ k-Means Clustering
    INPUT:  X            - DxN array of N data points with D features
            K            - number of clusters
            max_iter     - maximum number of iterations
            eta          - small threshold for convergence criterion
    OUTPUT: centroids    - DxK array of K centroids with D features
            closest      - NxK array that indicates for each of the N data point
s
                           in X the closest centroid after convergence.
                           Each row in closest only holds one non-zero entry.
                           closest[n,k] == 1 <=>
                           centroids[:,k] is closest to data point X[:,n]
    """
    D,N = np.shape(X)
    dist = np.zeros([N,K])
    closest = np.zeros([N,K])
    # initialize the centroids (close to the mean of X)
    meanX=np.mean(X,axis=1).reshape(D,1)
    noise=np.random.normal(0,1,K)
    #D x K
    centroids=meanX+noise


    cur_iter = 0
    while cur_iter < max_iter:
        plot_cluster(X, centroids, closest)
        cur_iter += 1
        old_centroids = centroids.copy()
        # calculate the distance between each data point and each centroid
        # N x K
        dist = distmat(X,centroids)
        # get for each data point in X it's closest centroid
        # N x 1
        idx=np.argmin(dist,axis=1)
        closest = np.zeros([N,K])
        closest[range(N),idx]=1
        # update the estimation of the centroids
        # ... your code here ...
        for i in range(K):
            centroids[:,i]=np.mean(X[:,i==idx],axis=1)      18.1

        if np.linalg.norm(old_centroids - centroids) < eta:
            print ('Converged after ' ,str(cur_iter) ,' iterations.')
            break;
    return centroids, closest

def distmat(X, Y):
    """ Distance Matrix
    INPUT:      X           - DxN array of N data points with D features
                Y           - DxM array of M data points with D features
    OUTPUT:     distmat     - NxM array s.t. D[n,m] = || x_n - y_m ||^2
    Hint: np.tile might be helpful
    """
    D_x,N = np.shape(X)
    D_y,M = np.shape(Y)
    assert D_x == D_y
    # calculate the distance matrix
    # ... your code here ...
```

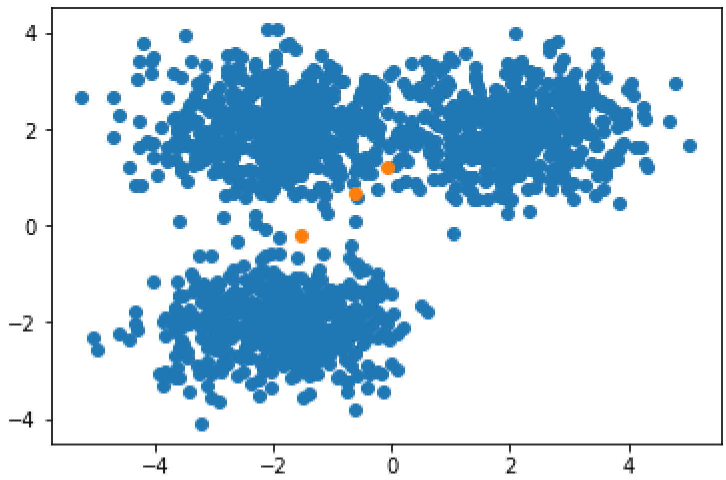
```
    return cdist(X.T,Y.T,metric='euclidean')


def test_kmeans():
    #generate 2D data
    N =500
    cov = np.array([[1, 0], [0, 0.5]])
    # generate for each of the three clusters N data points
    x1 = np.random.multivariate_normal([-2, 2], cov, N)
    x2 = np.random.multivariate_normal([2, 2], cov, N)
    x3 = np.random.multivariate_normal([-2, -2], cov, N)
    X = np.vstack((x1, x2, x3)).transpose()

    # run kmeans and plot the result
    centroids, closest = kmeans(X, 3)
    plot_cluster(X, centroids, closest)


def plot_cluster(X, centroids, closest):
    K = np.shape(centroids)[1]
    plt.figure()
    plt.scatter(X[0], X[1])
    if (closest != np.zeros(np.shape(closest))).any():
        for k in range(K):
            # get for each centroid the assigned data points
            Xk = X[:, closest[:,k]==1]
            # plot each cluster in a different color
            plt.scatter(Xk[0], Xk[1])
    # plot each centroid (should be center of cloud)
    plt.scatter(centroids[0], centroids[1])
```

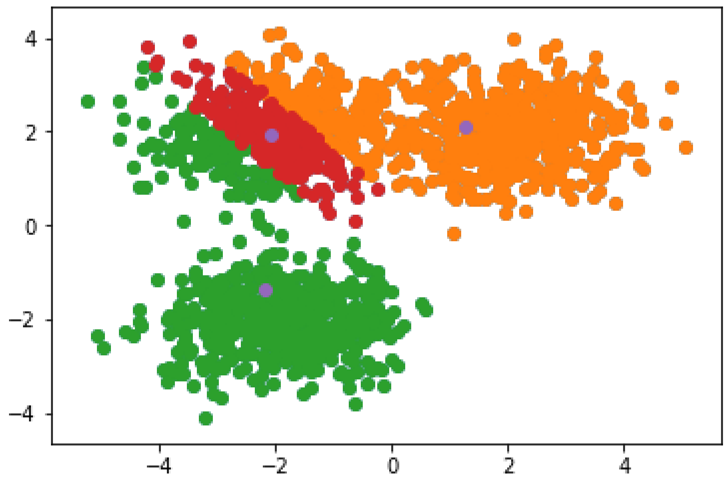
19.1

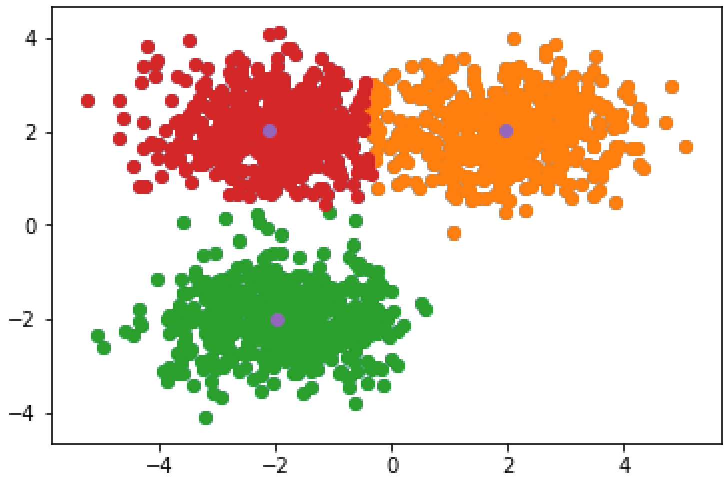
In [12]:

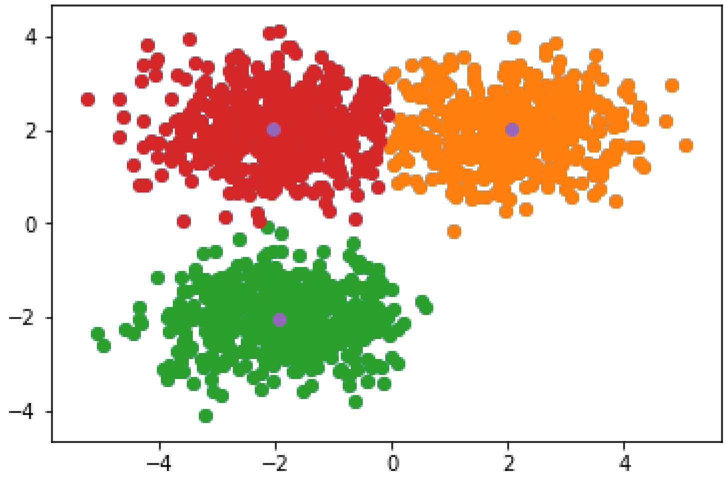
```
test_kmeans()
```

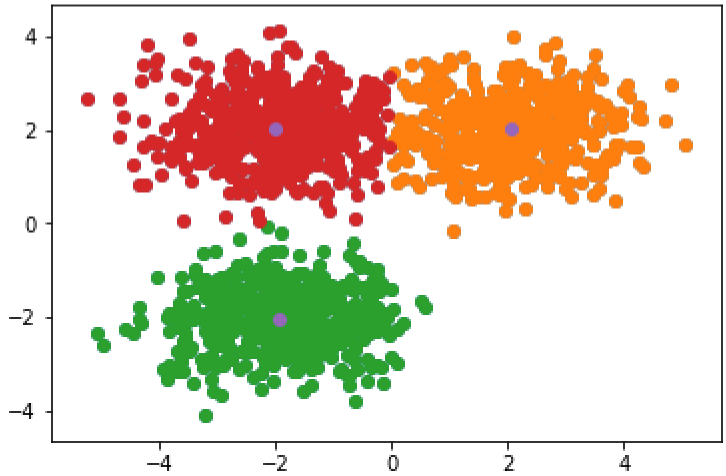
```
Converged after  4  iterations.
```








4
2
0
−2
−4
−4
−2
0
2
4


4
2
0
−2
−4
−4
−2
0
2
4

Index of comments

2.1 -1 only first one is right

18.1 -1 for loop not necessary

19.1 -1 you were supposed to use the given formular