In [1]:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
import re
```

In [2]:

```
documents = [
    "This little kitty came to play when I was eating at a restaurant.",
    "Merley has the best squooshy kitten belly.",
    "Google Translate app is incredible",
    "If you open 100 tab in google you get a smiley face.",
    "Best cat photo I've ever taken.",
    "Climbing ninja cat.",
    "Impressed with google map feedback",
    "Key promoter extension for Google Chrome."
]
```

In [3]:

```
from collections import defaultdict

DTM = defaultdict(lambda:defaultdict(int))
for i,d in enumerate(documents):
    for t in word_tokenize(d.lower()):
        if t not in stopwords.words("english") and not re.search(r"[%s]" % re.escape(punctuati
            DTM[i][t] += 1
```

In [6]:

```
TDM = defaultdict(lambda:defaultdict(int))
for d, termList in DTM.items():
    for t, f in termList.items():
        TDM[t][d] = f
```

In [ ]:

```
'''
TF * IDF
Tf = freq(i,j) / max freq(j) => j번째 문서에서 나온, i번째 빈도를 freq(i,j)
freq(i,j) => TDM[단어][문서:빈도], max freq => DTM[문서] max([단어:빈도])
IDF = log N / df(i) => i번째 단어의 문서빈도 df(i), N => 전체 문서 수
df => len(TDM[단어]), N => len(DTM)
'''
```

In [8]:

```
from math import log2

TWM = defaultdict(lambda:defaultdict(float))
N = len(DTM)
for term, docList in TDM.items():
    df = len(docList)
    for d,f in docList.items():
        maxFreq = max(DTM[d].values())
        TF = f/maxFreq
        IDF = log2(N/df)
        TWM[term][d] = TF*IDF
#TWM[단어][문서:가중치,...]
```

In [9]:

```
TWM
```

...

K-Means

In [ ]:

```
'''
TWM => 단어-문서:가중치, ...
단어 -> concept(차원 축)
문서->BoW(Bag of words)[단어,단어,단어, .. V=TDM.keys()]
문서1 = [0,0,1,0,0...., V] <- vectorize
문서2 = [1,0,1,0,0...., V]
'''
'''
K = ?
centroid = ? <- k개 만큼.. [for _ in range(K)]
_iter = 10
for _ in range(_iter):
    expectation()
    maximization()
'''
```

In [12]:

```
docVectorList = [[0 for _ in range(len(TDM))] for _ in range(N)]
D = list(DTM.keys())
V = list(TDM.keys())
for t, docList in TWM.items():
    for d,w in docList.items():
        docVectorList[D.index(d)][V.index(t)] = w
```

In [71]:

```
V
```

Out[71]:

```
['little',
 'kitty',
 'came',
 'play',
 'eating',
 'restaurant',
 'merley',
 'best',
 'squooshy',
 'kitten',
 'belly',
 'google',
 'translate',
 'app',
 'incredible',
 'open',
 '100',
 'tab',
 'get',
 'smiley',
 'face',
 'cat',
 'photo',
 'ever',
 'taken',
 'climbing',
 'ninja',
 'impressed',
 'map',
 'feedback',
 'key',
 'promoter',
 'extension',
 'chrome']
```

In [16]:

```
print(docVectorList[0])
print(D[0], V[:6])
```

```
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
0 ['little', 'kitty', 'came', 'play', 'eating', 'restaurant']
```

In [11]:

```
len(docVectorList), len(documents), len(docVectorList[0]), len(TDM)
```

Out[11]:

```
(8, 8, 34, 34)
```

In [52]:

```
from math import sqrt

def distance(x1, x2):
    _sum = 0
    for i in range(len(x1)):
        _sum += (x1[i]-x2[i])**2
    return sqrt(_sum)

def angle(x1, x2):
    _innerProduct = 0
    for i in range(len(x1)):
        _innerProduct += x1[i]*x2[i]
    x1VecLength = distance(x1, [0 for _ in range(len(x1))])
    x2VecLength = distance(x2, [0 for _ in range(len(x2))])
    return _innerProduct/(x1VecLength*x2VecLength)
```

In [51]:

```
def expectation(x, c, opt=False):
    candidates = list()

    nearest = distance if not opt else angle
    best = min if not opt else max

    for _ in c:
        candidates.append(nearest(x, _))
    return candidates.index(best(candidates))
```

In [47]:

```
def maximization(X):
    _sum = [0 for _ in range(len(X[0]))]
    D = len(X)
    for x in X:
        for i in range(len(x)):
            _sum[i] += x[i]
    return [_/D for _ in _sum]
```

In [49]:

```
def sse(X, c, opt=False): # sum squard error
    error = 0
    nearest = distance if not opt else angle
    for x in X:
        error += nearest(x,c)
    return error
```

In [54]:

```
from random import randrange

K = 2
centroid = [[randrange(0,4) for _ in range(len(V))] for _ in range(K)]
_iter = 10
sseList = list()

for _ in range(_iter):
    rnk = list(list(0 for _ in range(K)) for _ in range(len(D)))
    for i,x in enumerate(docVectorList):
        idx = expectation(x, centroid)
        rnk[i][idx] = 1

    _sum = 0.0
    for k in range(K):
        _X = [docVectorList[i] for i in range(len(D)) if rnk[i][k]]
        _sum += sse(_X, centroid[k]) #sum squared error
        centroid[k] = maximization(_X)
    sseList.append(_sum)
```

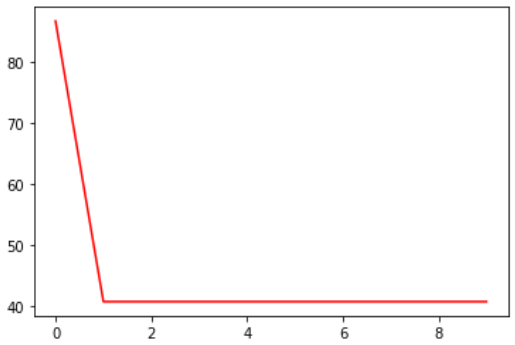
In [57]:

```
import matplotlib.pyplot as plt

plt.plot(range(_iter), sseList, "r-")
plt.show
```

Out[57]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [58]:

```
for k in range(K):
    _X = [i for i in range(len(D)) if rnk[i][k]]
    print("Cluster:",k)
    for x in _X:
        print(documents[x])
```

```
Cluster: 0
This little kitty came to play when I was eating at a restaurant.
If you open 100 tab in google you get a smiley face.
Cluster: 1
Merley has the best squooshy kitten belly.
Google Translate app is incredible
Best cat photo I've ever taken.
Climbing ninja cat.
Impressed with google map feedback
Key promoter extension for Google Chrome.
```

In [59]:

```
[{V[i]:centroid[0][i]} for i in range(len(V))]
```

...

In [61]:

```
for k in range(K):
    print([{V[i]:centroid[k][i]} for i in range(len(V)) if centroid[k][i]])
```

```
[{'little': 1.5}, {'kitty': 1.5}, {'came': 1.5}, {'play': 1.5}, {'eating': 1.5}, {'r
estaurant': 1.5}, {'google': 0.5}, {'open': 1.5}, {'100': 1.5}, {'tab': 1.5}, {'ge
t': 1.5}, {'smiley': 1.5}, {'face': 1.5}]
[{'merley': 0.5}, {'best': 0.6666666666666666}, {'squooshy': 0.5}, {'kitten': 0.5},
{'belly': 0.5}, {'google': 0.5}, {'translate': 0.5}, {'app': 0.5}, {'incredible': 0.
5}, {'cat': 0.6666666666666666}, {'photo': 0.5}, {'ever': 0.5}, {'taken': 0.5}, {'cl
imbing': 0.5}, {'ninja': 0.5}, {'impressed': 0.5}, {'map': 0.5}, {'feedback': 0.5},
{'key': 0.5}, {'promoter': 0.5}, {'extension': 0.5}, {'chrome': 0.5}]
```

In [65]:

```
for k in range(K):
    W = [(V[i],centroid[k][i]) for i in range(len(V)) if centroid[k][i]]
    print(sorted(W, key=lambda x:x[1], reverse=True)[:5])
```

```
[('little', 1.5), ('kitty', 1.5), ('came', 1.5), ('play', 1.5), ('eating', 1.5)]
[('best', 0.6666666666666666), ('cat', 0.6666666666666666), ('merley', 0.5), ('squoo
shy', 0.5), ('kitten', 0.5)]
```

In [69]:

```
result = list()
for k in range(K):
    W = [(V[i],centroid[k][i]) for i in range(len(V)) if centroid[k][i]]
    result.append(sorted(W, key=lambda x:x[1], reverse=True)[:5])
    #print(sorted(W, key=lambda x:x[1], reverse=True)[:5])
result
```

Out[69]:

```
[[('little', 1.5),
  ('kitty', 1.5),
  ('came', 1.5),
  ('play', 1.5),
  ('eating', 1.5)],
 [('best', 0.6666666666666666),
  ('cat', 0.6666666666666666),
  ('merley', 0.5),
  ('squooshy', 0.5),
  ('kitten', 0.5)]]
```

In [70]:

```
from wordcloud import WordCloud

data = {x[0]:x[1] for x in result[0]}
wc = WordCloud(background_color="white")
wc.generate_from_frequencies(data)
wc.to_image()
```

Out[70]:

In [ ]:

```
V = len(distinct words:Vocaburary)
K = 6
from random import sample
len(sample(docVeectorlist, K))
```