In [1]:
```
import re
import nltk
import pandas as pd
import numpy as np
from os import listdir
from os.path import isfile, join
from nltk.tokenize import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
```

In [2]:
```
tokenizer = RegexpTokenizer(r'\w+')
stopwords = set(nltk.corpus.stopwords.words('english'))
stemmer = SnowballStemmer("english")
```

In [3]:
```
# tokenizer.tokenize('Eighty-seven miles to go, yet.  Onward!')
def tokenize_stop_stem(text):
    tokens = tokenizer.tokenize(text)
    # filter out any tokens not containing letters (e.g., numeric tokens, ra
    filtered_tokens = set()
    for token in tokens:
        token = token.lower()
        if token not in stopwords:
            if not re.search('[0-9]', token):
                try:
                    token = stemmer.stem(token)
                    filtered_tokens.add(token)
                except UnicodeDecodeError:
                    print 'illeagal token ignored:',token
                    pass
    return filtered_tokens
```

In [4]:
```
files = [f for f in listdir("/Users/sanket/Desktop/nlp_emailrecs/sample_data
```

In [5]:
```
all_emails = []
for file in files:
    f = open(join("/Users/sanket/Desktop/nlp_emailrecs/sample_data", file))
    text = f.read()
    f.close()
    all_emails.append(text)
```

In [ ]:
```
alltokens = set()
for file in files[:1000]:
    f = open(join("/Users/sanket/Desktop/nlp_emailrecs/sample_data", file))
    text = f.read()
    f.close()
    alltokens = alltokens.union(tokenize_stop_stem(text))
```

In [ ]:
```
print len(alltokens)
```

Generate TF-IDF matrix on the emails we have

In [6]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [7]:
```
#define vectorizer parameters
tfidf_vectorizer = TfidfVectorizer(max_features=2000000, stop_words='englisl
```

In [8]:
```
%time tfidf_matrix = tfidf_vectorizer.fit_transform(all_emails)
```

```
CPU times: user 4.06 s, sys: 26.5 ms, total: 4.09 s
Wall time: 4.09 s
```

In [9]:
```
print(tfidf_matrix.shape)
```

```
(1000, 5890)
```

In [10]:
```
terms = tfidf_vectorizer.get_feature_names()
```

Generate pairwise cosine similartiy

Distance = 1 - similarity

In [11]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

In [12]:
```
%time dist = 1 - cosine_similarity(tfidf_matrix)
```

```
CPU times: user 63.2 ms, sys: 17.5 ms, total: 80.7 ms
Wall time: 79.8 ms
```

In [13]:
```
dist.shape
```

Out[13]:
```
(1000, 1000)
```

In [14]:
```
dist.mean()
```

Out[14]:
```
0.92334229309306248
```

Kmeans clustering of TF-IDF vectors

In [15]:
```
from sklearn.cluster import KMeans
```

In [16]:
```
def cluster_gridsearch(num_clusters):
    km = KMeans(n_clusters=num_clusters,n_jobs=-1)
    %time km.fit(tfidf_matrix)
    print km.inertia_
    return km.inertia_
```

In [33]:
```
num_clusters = 5
km = KMeans(n_clusters=num_clusters,n_jobs=-1)
```

In [34]:
```
%time km.fit(tfidf_matrix)
```

```
CPU times: user 76.4 ms, sys: 46.4 ms, total: 123 ms
Wall time: 1.43 s
```

Out[34]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=5, n_init=10, n_jobs=-1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [52]:
```
label_df = pd.DataFrame(km.labels_)
label_df[0].value_counts()
```

Out[52]:
```
2    315
0    296
1    179
3    159
4     51
Name: 0, dtype: int64
```

In [59]:
```
km.inertia_
```

Out[59]:
```
836.2358911719807
```

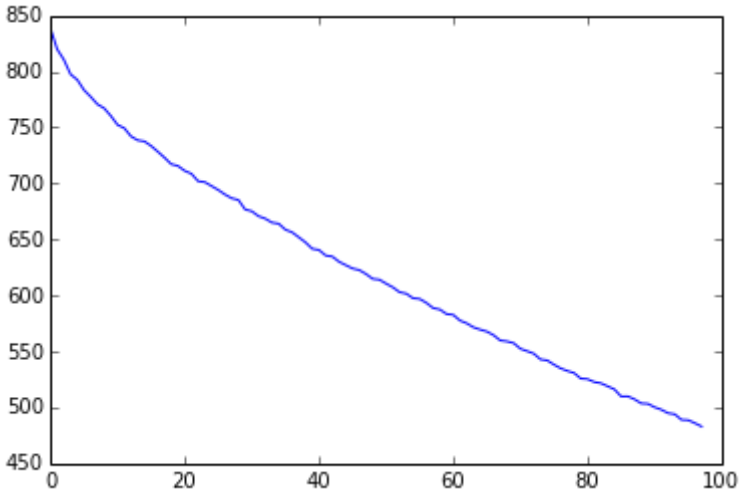
In [17]:
```
error_list = []
```

In [24]:
```
for i in range(200,400,5):
    print i,
    error_list.append(cluster_gridsearch(i))
```

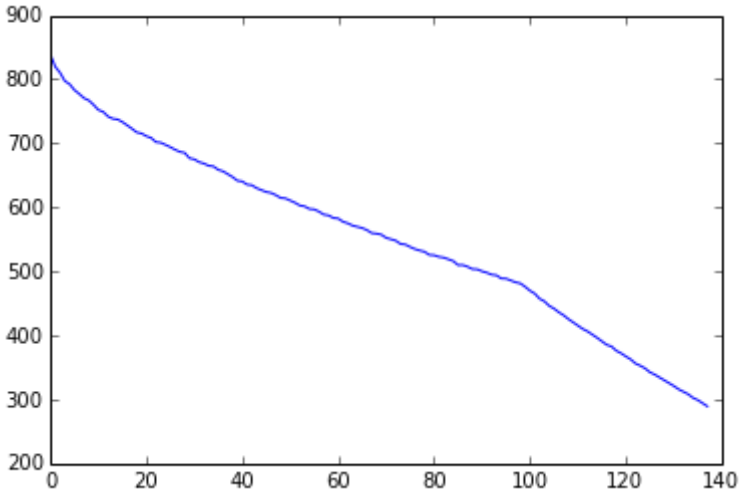
...

In [22]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [23]:
```
plt.plot(error_list)
plt.show()
```



In [25]:
```
plt.plot(error_list)
plt.show()
```



In [ ]: