11-Dimensionality-Reduction-SVD-II

October 17, 2017

1 Dimensionality Reduction - SVD II

In the last lecture we learned about the SVD as a tool for constructing low-rank matrices.

Today we'll look at it as a way to transform our data objects.

As a reminder, here is what the SVD looks like:

$$\text{objects}\left\{\begin{array}{c} \\ \\ \\ \\ \\ \end{array}\right. \overbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{a_1} & \mathbf{a_2} & \dots & \mathbf{a_n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}}^{\text{features}} = \overbrace{\begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \\ \sigma_1\mathbf{u}_1 & \sigma_k\mathbf{u}_k \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}}^{k} \times \begin{bmatrix} \dots & \dots & \mathbf{v}_1 & \dots & \dots \\ \dots & \dots & \mathbf{v}_k & \dots & \dots \end{bmatrix}$$

$$A = U\Sigma V^T$$

Notice that U contains a row for each object.

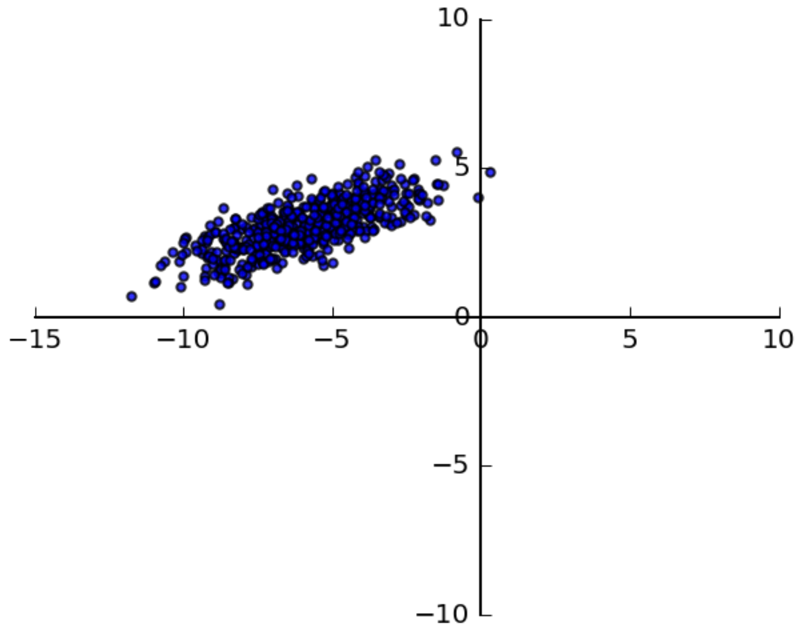
In a sense we have transformed objects from an n dimensional space to a k dimensional space, where k is (probably much) smaller than n.

This suggests an idea: is there an **optimal** transformation of the data into k dimensions?

What would that mean?

One criterion: a transformation that captures the maximum **variance** in the data.

```
In [3]: n_samples = 500
        C = np.array([[0.1, 0.6], [2., .6]])
        X = np.random.randn(n_samples, 2) @ C + np.array([-6, 3])
        ax = ut.plotSetup(-10,10,-10,10,(6,6))
        ut.centerAxes(ax)
        plt.axis('equal')
        _ = plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
```



What would happen if we used SVD, and kept only rank-1 approximation to the data?

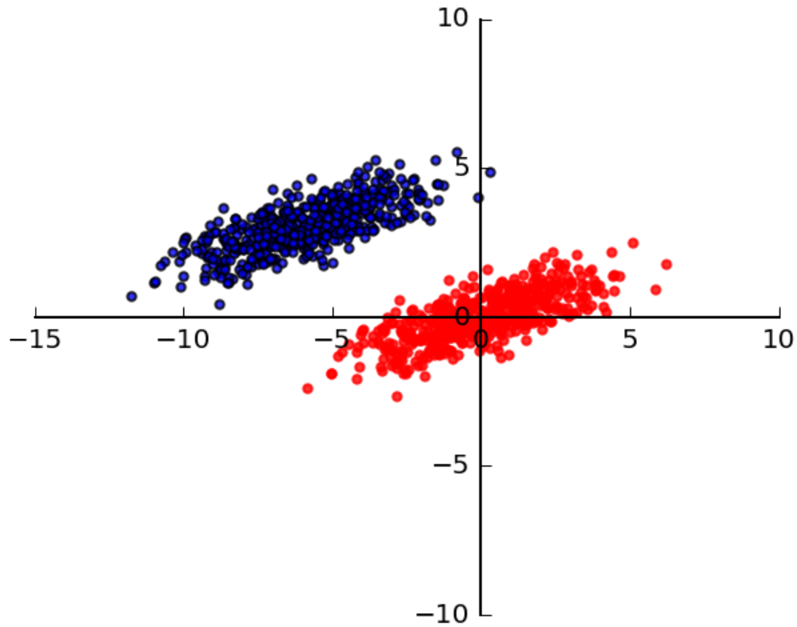
This would be the 1-D **subspace** that approximates the data best.

However the variance in the data is defined with respect to the data mean, so we need to mean-center the data first, before using SVD.

That is, SVD in this case finds the best 1-D subspace, not the best line though the data (which might not pass through the origin).

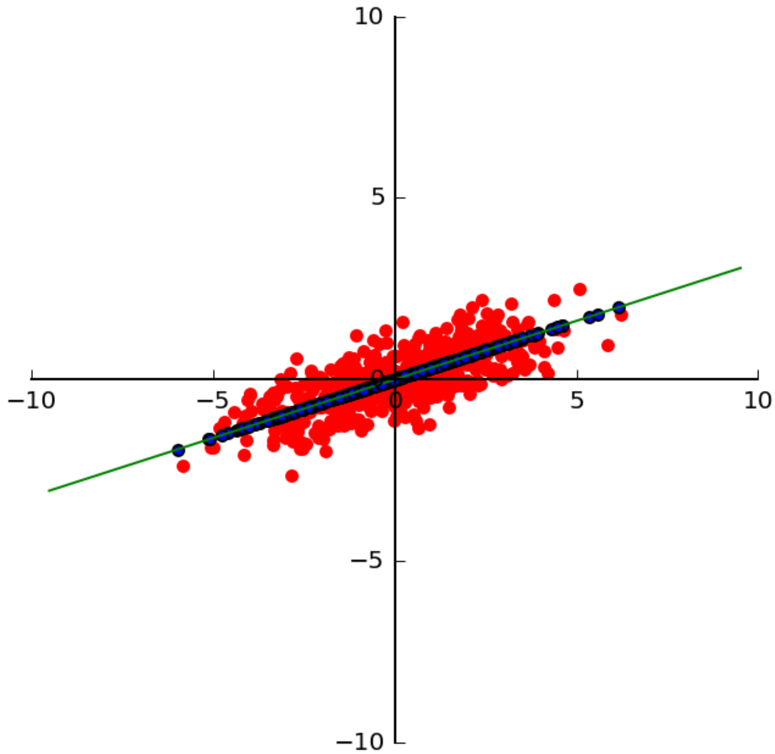
So to capture the best line through the data, we first move the data points to the origin:

```
In [4]: Xc = X - np.mean(X,axis=0)
        ax = ut.plotSetup(-10,10,-10,10,(6,6))
        ut.centerAxes(ax)
        plt.axis('equal')
        plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.8)
        _ = plt.scatter(Xc[:, 0], Xc[:, 1], s=10, alpha=0.8, color='r')
```



Now let's construct the best 1-D approximation of the mean-centered data:

```
In [5]: u, s, vt = np.linalg.svd(Xc,full_matrices=False)
        scopy = s.copy()
        scopy[1] = 0.
        reducedX = u @ np.diag(scopy) @ vt
        ax = ut.plotSetup(-10,10,-10,10,(8,8))
        ut.centerAxes(ax)
        plt.axis('equal')
        plt.scatter(Xc[:,0],Xc[:,1], color='r')
        plt.scatter(reducedX[:,0], reducedX[:,1])
        endpoints = np.array([[-10],[10]]) @ vt[[0],:]
        _ = plt.plot(endpoints[:,0], endpoints[:,1], 'g-')
```



This method is called **Principal Component Analysis.**

In summary, PCA consists of:

1. Mean center the data, and
2. Reduce the dimension of the mean-centered data via SVD.

This is equivalent to projecting the data onto the subspace that captures the maximum variance in the data.

It winds up constructing the **best low dimensional approximation of the data.**

What are "principal components"?

These are nothing more than the columns of U (or the rows of V^T). Because they capture the direction of maximum variation, they are called "principal" components.

1.1 Uses of PCA/SVD

There are many uses of PCA (and SVD).

We'll cover three of the main uses:

1. Visualization
2. Denoising
3. Anomaly Detection

As already mentioned, SVD is also useful for data compression -- we won't discuss it in detail, but it is the principle behind audio and video compression (MP3s, HDTV, etc).

1.2 Visualization and Denoising -- Extended Example.

We will study both visualization and denoising in the context of text processing.

As we have seen, a common way to work with documents is using the bag-of-words model (perhaps considering n-grams), which results in a term-document matrix.

Entries in the matrix are generally TF-IDF scores.

Often, terms are correlated -- they appear together in combinations that suggest a certain "concept".

That is, term-document matrices often show low effective rank -- many columns can be approximated as combinations of other columns.

When PCA is used for dimensionality reduction of documents, it tends to to extract these "concept" vectors.

The application of PCA to term-document matrices is called **Latent Semantic Analysis (LSA).**

Among other benefits, LSA can improve the performance of clustering of documents.

This happens because the important concepts are captured in the most significant principal components.

1.3 Data: 20 Newsgroups

```
In [6]: from sklearn.datasets import fetch_20newsgroups

        categories = ['comp.os.ms-windows.misc', 'sci.space','rec.sport.baseball']
        news_data = fetch_20newsgroups(subset='train', categories=categories)
```

```
In [7]: print(news_data.target_names)
        print(news_data.target)

        # import nltk
        # nltk.download()
```

```
['comp.os.ms-windows.misc', 'rec.sport.baseball', 'sci.space']
[2 0 0 ..., 2 1 2]
```

1.3.1 Basic Clustering

To get started, let's compute tf-idf scores.

Notice that we will let the tokenizer compute *n*-grams for $n=$1 and 2.

An *n*-gram is a set of *n* consecutive terms.

We'll compute a document-term matrix `dtm`.

```
In [8]: from sklearn.feature_extraction.text import TfidfVectorizer

        vectorizer = TfidfVectorizer(stop_words='english', min_df=4,max_df=0.8)
        dtm = vectorizer.fit_transform(news_data.data)

In [9]: print(type(dtm), dtm.shape)
        terms = vectorizer.get_feature_names()
        print(terms)

<class 'scipy.sparse.csr.csr_matrix'> (1781, 9409)
['00', '000', '0005', '0062', '0096b0f0', '00bjgood', '00mbstultz', '01', '0114', '01wb', '02',
```

The usual solution taken is to simply 'chop off' the part of the word that indicates a variation from the base word.

(For those of you who studied Latin or Greek, this will sound familiar -- we are removing the 'inflection.')

The process is called 'stemming.'

A very good stemmer is the "Snowball" stemmer.

You can read more at http://www.nltk.org and http://www.nltk.org/howto/stem.html.

Installation Note: From a cell you need to call `nltk.download()` and select the appropriate packages from the interface that appears. In particular you need to download: `stopwords` from *corpora* and `punkt` and `snowball_data` from *models*.

Let's stem the data using the Snowball stemmer:

```
In [12]: from nltk.stem.snowball import SnowballStemmer
         from nltk.tokenize import word_tokenize, sent_tokenize


         stemmed_data = [" ".join(SnowballStemmer("english", ignore_stopwords=True).stem(word)
                   for sent in sent_tokenize(message)
                  for word in word_tokenize(sent))
                  for message in news_data.data]

         # stemmed_data = news_data.data
```

```
In [13]: dtm = vectorizer.fit_transform(stemmed_data)
         terms = vectorizer.get_feature_names()
         print(terms)
```

```
['00', '000', '0005', '0062', '0096b0f0', '00bjgood', '00mbstultz', '01', '0114', '01wb', '02',
```

1.4 Demonstrating PCA

OK. Now, let's apply PCA.

Our data matrix is in sparse form.

First, we mean center the data. Note that `vectors` is a sparse matrix, but once it is mean centered it is not sparse any longer.

```
In [16]: dtm_dense = dtm.todense()
         centered_dtm = dtm_dense - np.mean(dtm_dense, axis=0)
         np.sum(centered_dtm,axis=0)[:,:10]
```

```
Out[16]: matrix([[ -2.67060679e-15,  -2.02008549e-15,   1.43851944e-15,
                    3.62058473e-15,   5.39368897e-15,  -1.66370823e-15,
                   -2.71657696e-15,   1.05475524e-14,   6.37825296e-15,
                    4.50692001e-15]])
```

```
In [17]: u, s, vt = np.linalg.svd(centered_dtm)
```

Note that if you have sparse data, you may want to use `scipy.sparse.linalg.svds()` and for large data it may be advantageous to use `sklearn.decomposition.TruncatedSVD()`.

$$\text{objects}\left\{ \overbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{a}_1 & \mathbf{a}_2 & \dots & \mathbf{a}_n \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}}^{\text{features}} \right. = \overbrace{\begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \\ \sigma_1\mathbf{u}_1 & \sigma_k\mathbf{u}_k \\ \vdots & \vdots \\ \vdots & \vdots \end{bmatrix}}^{k} \times \begin{bmatrix} \dots & \dots & \mathbf{v}_1 & \dots & \dots \\ \dots & \dots & \mathbf{v}_k & \dots & \dots \end{bmatrix}$$

$$A = U\Sigma V^T$$

The principal components (rows of V^T) encode the extracted concepts.

Each LSA **concept** is a linear combination of words.

```
In [18]: pd.DataFrame(vt,columns=vectorizer.get_feature_names())
```

```
Out[18]:           00           000      0005      0062  0096b0f0  00bjgood  \
         0   0.007885  1.233932e-02  0.000591  0.005538  0.001029  0.002070
         1  -0.005886  9.505666e-03  0.002096 -0.010693 -0.001650 -0.003491
         2  -0.012564 -1.201289e-02 -0.002483  0.001447  0.000431  0.000050
         3   0.013521  1.779776e-02  0.003623  0.000994  0.003338 -0.000420
         4  -0.002048 -7.629171e-03 -0.005712  0.001722  0.000314  0.000811
         5  -0.002853 -1.776525e-02 -0.001143  0.021926  0.003219  0.000665
         6   0.005453  9.739383e-03 -0.002027 -0.044426 -0.003053 -0.000241
         7  -0.017610 -1.251235e-03  0.002503  0.041705  0.001032  0.000374
         8  -0.005070  9.930212e-03 -0.001484 -0.011102  0.002228 -0.000190
         9  -0.012347  4.293474e-03 -0.002194  0.057858 -0.008261 -0.004892
         10  0.019830  9.634970e-03  0.005410 -0.002062 -0.001075 -0.000955
```

11	-0.011969	-1.185035e-02	0.004851	-0.000715	-0.000383	0.001777
12	0.045026	3.488553e-03	-0.001457	0.009678	0.000071	0.007143
13	0.001603	1.836974e-02	0.010058	-0.006219	-0.000117	-0.001881
14	0.020302	-4.063356e-03	0.001184	-0.001073	0.002072	-0.000296
15	-0.014153	-2.934435e-02	0.001914	-0.004362	0.002385	0.000029
16	0.018074	2.121548e-02	0.007850	-0.003002	-0.001862	0.001760
17	-0.037716	-7.859865e-03	-0.005198	0.009552	-0.001860	-0.001758
18	0.062188	1.533966e-02	0.003208	-0.001076	0.002648	0.004524
19	0.009863	1.362330e-02	0.003877	0.010903	-0.007652	-0.006105
20	-0.002392	-2.741838e-02	0.000370	-0.005332	0.002976	0.001881
21	0.060240	-1.897149e-02	-0.011127	-0.010881	0.002168	0.001426
22	-0.015986	7.926941e-03	-0.007186	-0.000617	0.005857	0.001825
23	-0.043422	2.190333e-02	0.017745	-0.009493	0.004282	0.005267
24	-0.044437	-1.455195e-02	-0.015202	-0.017293	-0.002759	0.002604
25	0.016403	-1.632085e-02	-0.016043	-0.004275	-0.001701	0.002056
26	0.030019	3.307837e-02	0.018532	-0.005862	-0.001601	0.001594
27	-0.046451	3.670277e-02	0.016128	-0.000483	-0.010971	-0.010615
28	0.031531	2.890389e-02	0.006755	-0.011681	-0.002021	-0.007612
29	-0.055213	-1.060479e-02	-0.001368	0.005921	0.010555	-0.004898
...	...	...	...	...	...	...
8029	-0.000113	3.003353e-04	0.014047	0.000139	0.009076	0.000529
8030	-0.001263	-1.664282e-04	-0.022273	-0.006657	0.014768	-0.011241
8031	-0.000151	1.103191e-04	-0.000693	0.000121	-0.001185	-0.000329
8032	0.000167	-1.743944e-03	-0.000377	-0.000341	-0.001013	0.000046
8033	0.000113	-9.916591e-05	-0.000681	-0.000037	-0.001026	-0.000126
8034	-0.000093	-1.528359e-03	-0.000387	-0.000277	-0.001387	-0.000137
8035	-0.000352	-2.506609e-04	-0.000087	0.000046	-0.001037	0.000016
8036	-0.000077	-1.057066e-03	-0.000246	-0.000182	-0.000785	-0.000109
8037	-0.000485	1.775672e-04	0.000056	0.000112	-0.000386	-0.000037
8038	-0.000007	-2.174756e-04	-0.000089	-0.000059	-0.000612	0.000007
8039	-0.000191	-8.281751e-07	-0.000059	0.000047	-0.000593	0.000040
8040	-0.000002	-2.530187e-04	0.000016	-0.000137	-0.000494	0.000027
8041	-0.000188	6.040552e-04	-0.000690	0.000154	-0.001186	-0.000521
8042	0.000240	-9.226712e-04	-0.004174	-0.008978	-0.003440	0.003963
8043	-0.003968	3.138345e-03	-0.006069	0.003930	0.016228	0.011624
8044	0.005090	1.363833e-03	-0.002889	-0.003098	0.006454	-0.013265
8045	-0.000108	5.985203e-04	0.000099	-0.000027	-0.000879	-0.000237
8046	0.000277	1.064382e-04	0.000088	-0.000608	-0.001392	0.000085
8047	0.000011	-8.327484e-04	-0.000187	-0.000299	-0.000927	-0.000073
8048	-0.000058	-1.872409e-04	-0.000013	-0.000029	-0.000905	-0.000055
8049	0.000083	-3.361604e-04	-0.000068	-0.000115	-0.000562	-0.000070
8050	0.000053	-2.962910e-04	-0.000040	-0.000095	-0.000589	-0.000049
8051	0.000052	-3.152899e-04	-0.000160	0.000045	-0.000709	-0.000224
8052	-0.000025	-1.370518e-04	0.000170	-0.000174	-0.000320	-0.000048
8053	0.000244	-2.871237e-04	-0.000498	-0.000414	-0.000935	0.000047
8054	-0.000009	-3.399028e-04	0.000187	-0.000418	-0.003914	-0.000498
8055	0.000112	1.512468e-04	-0.000088	-0.000108	-0.000204	-0.000041
8056	-0.000038	8.101495e-04	-0.000193	0.000014	-0.000711	-0.000212

```
8057  0.000078  4.524350e-04 -0.000172 -0.000135 -0.000815 -0.000042
8058 -0.000370 -1.166714e-03 -0.001545 -0.000563 -0.004363  0.005670

      00mbstultz        01          0114      01wb    ...          zri  \
0       0.002007  0.005579  1.241019e-03  0.000815    ...    -0.000028
1      -0.002693  0.002124 -3.395350e-03  0.002451    ...    -0.000015
2       0.000349 -0.006843  6.915764e-04 -0.001129    ...    -0.000095
3       0.000616  0.011659  2.224969e-03  0.001975    ...     0.000205
4       0.000187 -0.002988 -7.502258e-05  0.001228    ...    -0.000246
5       0.000816 -0.002939  2.555581e-03 -0.001191    ...     0.000376
6      -0.002329  0.002230 -2.425873e-03  0.001412    ...     0.000020
7       0.002267 -0.003667 -1.126606e-03 -0.000390    ...    -0.000021
8      -0.000772  0.001158  4.409476e-03  0.003383    ...     0.000161
9       0.002658 -0.007998 -7.094297e-03  0.002262    ...    -0.000257
10     -0.000567  0.003255  7.808977e-04 -0.000850    ...     0.000312
11     -0.001836 -0.011424 -2.380736e-03 -0.003327    ...    -0.000001
12     -0.005261  0.011956 -1.247550e-02  0.001854    ...    -0.000015
13     -0.001118  0.001763  3.730263e-03  0.000568    ...     0.000488
14      0.000540  0.004839  2.087347e-03  0.000371    ...     0.000088
15     -0.000823 -0.011319  3.466898e-03 -0.002251    ...     0.000126
16      0.000963 -0.001614 -8.171490e-03 -0.000172    ...     0.000104
17     -0.001514  0.004472  3.257196e-04  0.000162    ...     0.000022
18     -0.001288  0.022608  2.311536e-03  0.000697    ...     0.000490
19     -0.004511  0.009124  1.716752e-04  0.001386    ...     0.000213
20     -0.000803  0.006057 -7.898089e-04  0.000441    ...     0.000003
21      0.003095  0.009050  2.282049e-03 -0.001160    ...    -0.000072
22     -0.001975  0.002426 -2.411440e-03 -0.001009    ...     0.000140
23      0.007370 -0.002113  2.032921e-03  0.000218    ...    -0.000013
24      0.009233 -0.009791  5.814262e-04 -0.001681    ...     0.000123
25      0.001999 -0.004222 -8.426358e-03  0.000203    ...    -0.000111
26      0.004161  0.021882 -4.951961e-03  0.001026    ...     0.000589
27     -0.004011 -0.012474  5.773966e-04  0.000817    ...    -0.000409
28      0.007432  0.018244 -3.468594e-03  0.000772    ...     0.000473
29     -0.005397 -0.015494 -1.108879e-05 -0.001484    ...     0.000075
...          ...       ...           ...       ...    ...          ...
8029    0.003807 -0.000258  2.790690e-04  0.000497    ...     0.000046
8030   -0.003749  0.002201 -2.599797e-03 -0.000089    ...     0.000088
8031   -0.000279  0.000625 -1.818615e-04  0.000102    ...     0.000108
8032    0.000208  0.000784  1.284705e-04 -0.000658    ...    -0.000088
8033   -0.000147 -0.000257 -1.117703e-04 -0.000137    ...     0.000070
8034    0.000113  0.001595 -8.162644e-05 -0.000451    ...    -0.000098
8035   -0.000038  0.000727 -7.984581e-05 -0.000125    ...    -0.000060
8036    0.000067  0.001386 -3.191401e-05 -0.000352    ...    -0.000106
8037    0.000051  0.000613  3.944347e-05  0.000004    ...     0.000043
8038   -0.000058  0.000145 -8.161358e-05 -0.000135    ...    -0.000082
8039   -0.000068  0.000250 -8.937095e-05 -0.000026    ...    -0.000013
8040    0.000019  0.000036 -6.851833e-05 -0.000126    ...    -0.000002
8041    0.000471  0.001336 -5.141053e-04 -0.000161    ...     0.000170
```

8042	-0.001113	0.004839	-1.286544e-03	-0.001980	...	-0.000049
8043	0.000343	0.006907	-2.804303e-03	0.000090	...	-0.000046
8044	-0.003271	-0.014726	-4.516786e-03	-0.007658	...	0.000002
8045	-0.000136	-0.000162	-1.905786e-04	0.000389	...	0.000223
8046	0.000304	-0.001979	-2.535499e-05	-0.000334	...	0.000019
8047	0.000140	0.000615	-4.780270e-05	-0.000364	...	-0.000017
8048	-0.000078	0.000321	-6.949618e-05	-0.000046	...	-0.000175
8049	0.000019	0.000367	4.890606e-07	-0.000130	...	0.999358
8050	-0.000011	0.000349	-1.479869e-05	-0.000100	...	-0.000499
8051	-0.000039	-0.000437	1.288591e-04	0.000130	...	0.000113
8052	0.000081	0.000079	-5.370634e-05	-0.000032	...	0.000056
8053	0.000193	-0.000468	-2.023443e-05	-0.000511	...	0.000146
8054	-0.000210	0.000787	-2.981389e-04	0.000069	...	-0.001385
8055	0.000008	-0.000595	6.641749e-07	-0.000010	...	0.000111
8056	-0.000128	-0.000300	-1.178447e-04	0.000283	...	0.000249
8057	-0.000079	-0.000779	-1.358339e-04	0.000043	...	0.000195
8058	0.001090	0.001241	1.325372e-03	-0.000522	...	-0.000024

	zrlk	zs	zt	zu	zv	zw \
0	-2.473049e-05	-0.000194	-0.000025	-0.000129	-0.000207	-0.000087
1	-1.319113e-05	-0.000048	-0.000013	-0.000042	-0.000100	-0.000025
2	-8.662512e-05	-0.000270	-0.000087	-0.000253	-0.000578	-0.000134
3	1.864593e-04	0.000452	0.000172	0.000465	0.001143	0.000221
4	-2.242582e-04	-0.000611	-0.000208	-0.000581	-0.001387	-0.000283
5	3.442932e-04	0.000675	0.000256	0.000682	0.001908	0.000282
6	2.120784e-05	0.000356	0.000084	0.000270	0.000174	0.000124
7	-2.035344e-05	0.000019	-0.000020	-0.000028	-0.000118	-0.000001
8	1.455818e-04	0.000410	0.000124	0.000396	0.000966	0.000216
9	-2.306122e-04	-0.000378	-0.000144	-0.000398	-0.001358	-0.000202
10	2.835633e-04	0.000649	0.000275	0.000737	0.001835	0.000365
11	-3.595590e-06	-0.000013	-0.000025	-0.000043	-0.000032	-0.000008
12	-1.290929e-05	-0.000188	-0.000056	-0.000163	-0.000196	-0.000101
13	4.386390e-04	0.001237	0.000355	0.001103	0.002834	0.000614
14	8.129619e-05	0.000304	0.000108	0.000395	0.000619	0.000186
15	1.122185e-04	0.000116	0.000034	0.000141	0.000593	0.000059
16	9.542367e-05	0.000282	0.000077	0.000248	0.000616	0.000134
17	2.191298e-05	0.000313	0.000061	0.000234	0.000241	0.000139
18	4.437226e-04	0.000995	0.000369	0.001028	0.002813	0.000527
19	1.945776e-04	0.000703	0.000212	0.000634	0.001420	0.000360
20	1.616008e-06	0.000412	0.000089	0.000307	0.000247	0.000204
21	-6.429702e-05	-0.000389	-0.000097	-0.000347	-0.000578	-0.000208
22	1.280889e-04	0.000152	0.000073	0.000234	0.000733	0.000091
23	-1.541939e-05	-0.000219	-0.000096	-0.000268	-0.000231	-0.000128
24	1.159293e-04	0.000491	0.000141	0.000491	0.000883	0.000257
25	-9.693051e-05	-0.000079	-0.000014	-0.000078	-0.000502	-0.000040
26	5.326279e-04	0.000994	0.000387	0.001055	0.003259	0.000527
27	-3.754482e-04	-0.000942	-0.000319	-0.000899	-0.002405	-0.000468
28	4.315400e-04	0.001282	0.000401	0.001195	0.002891	0.000648

29	6.906250e-05	-0.000032	0.000021	-0.000001	0.000323	-0.000021
...	...	...	...	...	...	...
8029	2.700047e-05	0.000254	0.000166	-0.000138	0.000785	0.000147
8030	1.415319e-04	-0.000220	0.000442	0.000330	0.000959	0.000073
8031	8.597462e-05	-0.001859	0.000723	-0.000422	0.000533	-0.000405
8032	-1.191955e-04	-0.000923	-0.000523	-0.003411	0.000496	-0.000314
8033	6.433813e-05	-0.001566	0.000866	-0.001769	0.000732	-0.000740
8034	-1.529328e-04	-0.001050	-0.001156	-0.001834	-0.001010	-0.000015
8035	-9.552242e-05	0.000164	0.000046	0.000204	-0.000194	0.000190
8036	-1.918850e-04	-0.000303	-0.000608	-0.001053	-0.001417	0.000165
8037	3.737764e-05	0.000318	0.000118	0.000279	0.000656	0.000074
8038	-5.208127e-05	0.000151	0.000176	0.000565	0.000719	-0.000006
8039	6.038126e-06	0.000126	0.000127	0.000364	0.000678	0.000089
8040	5.941179e-06	0.000077	-0.000544	-0.000128	-0.000173	-0.000128
8041	1.702214e-04	-0.001423	0.000140	-0.000542	0.000673	-0.000548
8042	-3.526404e-05	0.000063	-0.000125	-0.000233	0.000040	-0.000033
8043	-1.459617e-05	-0.000272	-0.000301	-0.000143	0.000619	-0.000364
8044	4.045671e-07	0.000358	0.000132	-0.000137	0.000112	0.000345
8045	1.890384e-04	-0.000968	-0.001249	0.000662	-0.001326	-0.000316
8046	8.716143e-05	-0.001056	-0.001462	-0.003489	-0.001072	-0.001788
8047	-4.055907e-05	-0.000516	-0.000990	-0.001571	-0.000427	-0.000303
8048	-3.414221e-04	0.000045	0.000113	0.000370	-0.004201	0.000105
8049	-4.997819e-04	0.000086	0.000053	0.000148	-0.001389	0.000120
8050	9.995655e-01	0.000074	0.000030	0.000106	-0.001957	0.000109
8051	1.096385e-04	0.997616	-0.000060	-0.000708	0.000004	-0.000666
8052	3.552299e-05	-0.000052	0.998833	-0.000080	-0.001132	-0.000147
8053	9.473840e-05	-0.000715	-0.000099	0.995374	0.000625	-0.001079
8054	-1.950269e-03	-0.000298	-0.001183	0.000910	0.976013	-0.000068
8055	1.005950e-04	-0.000638	-0.000150	-0.001069	-0.000146	0.999355
8056	2.115051e-04	-0.001017	-0.000187	-0.000844	0.000150	-0.000498
8057	1.787668e-04	-0.000671	-0.000274	-0.001188	-0.000174	-0.000748
8058	2.847989e-07	-0.000478	-0.001082	-0.001944	0.000433	-0.000762

	zx	zy	zz
0	-0.000151	-1.133502e-04	0.000531
1	-0.000063	-3.977839e-05	-0.001043
2	-0.000294	-2.047762e-04	-0.000015
3	0.000510	3.530937e-04	0.000210
4	-0.000620	-4.477918e-04	-0.001012
5	0.000634	4.767288e-04	-0.000272
6	0.000240	2.170860e-04	0.001156
7	-0.000052	-1.694585e-05	-0.000748
8	0.000492	3.154438e-04	0.000612
9	-0.000440	-2.912447e-04	0.000477
10	0.000857	5.629118e-04	0.000907
11	-0.000068	-2.639282e-05	-0.000421
12	-0.000235	-1.553090e-04	-0.006931
13	0.001284	8.825428e-04	0.003020

14	0.000466	2.998201e-04	0.000319
15	0.000151	8.914734e-05	0.002374
16	0.000304	1.973389e-04	-0.002199
17	0.000277	1.893288e-04	0.000200
18	0.001213	8.033687e-04	0.001759
19	0.000783	5.337477e-04	0.003545
20	0.000389	2.853246e-04	0.000105
21	-0.000461	-3.101139e-04	-0.001745
22	0.000264	1.564825e-04	-0.002240
23	-0.000335	-2.088942e-04	-0.002465
24	0.000560	3.804607e-04	-0.000128
25	-0.000095	-5.980248e-05	0.001837
26	0.001286	8.278958e-04	0.003346
27	-0.001039	-6.946305e-04	-0.004749
28	0.001422	9.604788e-04	-0.001658
29	0.000011	-1.625493e-05	-0.000826
...	...	...	...
8029	-0.000181	-6.231988e-07	0.000081
8030	-0.000656	-9.366309e-05	0.002432
8031	-0.002032	-4.739790e-04	0.001452
8032	0.000376	-2.270770e-05	-0.001814
8033	-0.000526	-6.982079e-04	-0.000229
8034	0.000207	5.519988e-04	-0.001413
8035	0.000191	1.266837e-04	-0.000675
8036	-0.000156	4.568107e-04	-0.000202
8037	0.000064	1.077877e-04	-0.000386
8038	0.000189	-1.203629e-04	-0.000541
8039	0.000074	-4.039464e-05	-0.000313
8040	0.000264	-2.692784e-04	-0.000998
8041	-0.001243	-6.853256e-04	0.000132
8042	0.000737	1.065745e-04	0.000516
8043	0.000389	-2.500299e-04	-0.000930
8044	0.000438	4.460723e-04	-0.000325
8045	-0.002341	-8.709601e-04	0.000077
8046	0.000674	-2.167075e-03	-0.005545
8047	-0.000025	-1.855029e-04	-0.001716
8048	0.000172	1.396324e-04	0.000026
8049	0.000267	2.110059e-04	-0.000059
8050	0.000225	1.931220e-04	-0.000017
8051	-0.001046	-7.066692e-04	-0.000549
8052	-0.000169	-2.730460e-04	-0.001055
8053	-0.000733	-1.188785e-03	-0.002132
8054	0.000087	-1.057563e-04	0.000519
8055	-0.000476	-7.560081e-04	-0.000854
8056	0.996905	-1.101992e-03	0.000801
8057	-0.001116	9.987337e-01	-0.000780
8058	0.000995	-6.815379e-04	0.978480

```
[8059 rows x 8059 columns]
```

The rows of U correpond to documents, which are linear combinations of **concepts**.

1.5 Denoising

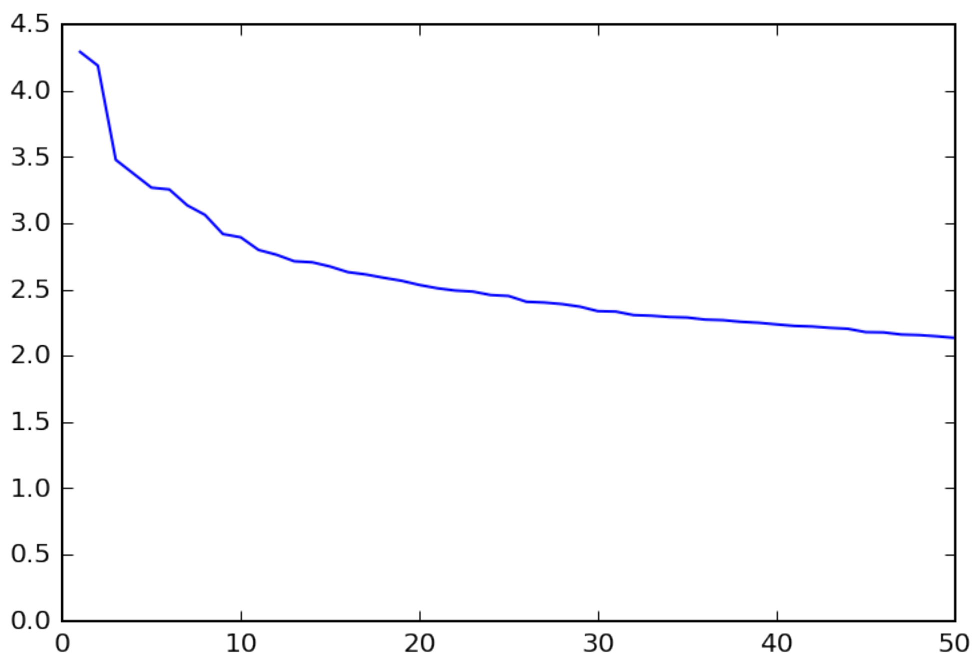
In order to improve our clustering accuracy, we will **exclude** the less significant concepts from the documents' feature vectors.

That is, we will choose the leftmost k columns of U and the topmost k rows of V^T.

The reduced set of columns of U are our new document encodings, and it is those that we will cluster.

```
In [19]: plt.xlim([0,50])
         plt.plot(range(1,len(s)+1),s)
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x117442978>]
```



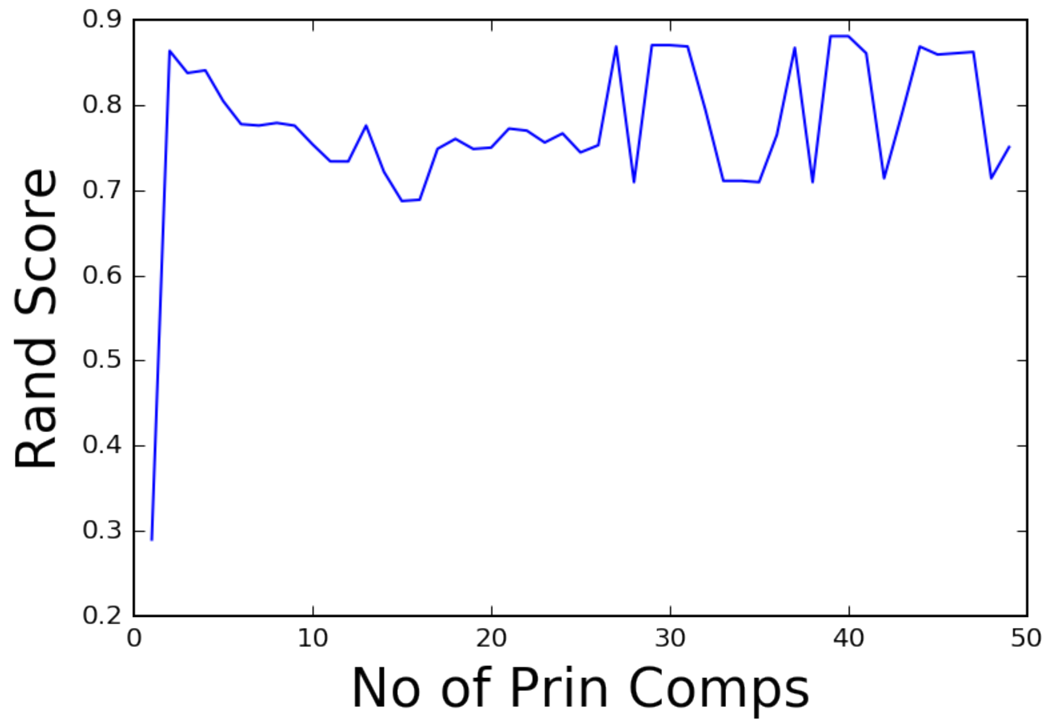
It looks like 2 is a reasonable number of principal components.

```
In [20]: ri = []
         ss = []
         max = len(u)
         for k in range(1,50):
             vectorsk = u[:,:k] @ np.diag(s[:k])
             kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=100, n_init=10, random_sta
             kmeans.fit_predict(vectorsk)
             labelsk = kmeans.labels_
```

```
            ri.append(metrics.adjusted_rand_score(labelsk,news_data.target))
            ss.append(metrics.silhouette_score(vectorsk,kmeans.labels_,metric='euclidean'))
```

In [21]:
```
plt.plot(range(1,50),ri)
plt.ylabel('Rand Score',size=20)
plt.xlabel('No of Prin Comps',size=20)
```

Out[21]: `<matplotlib.text.Text at 0x117457470>`

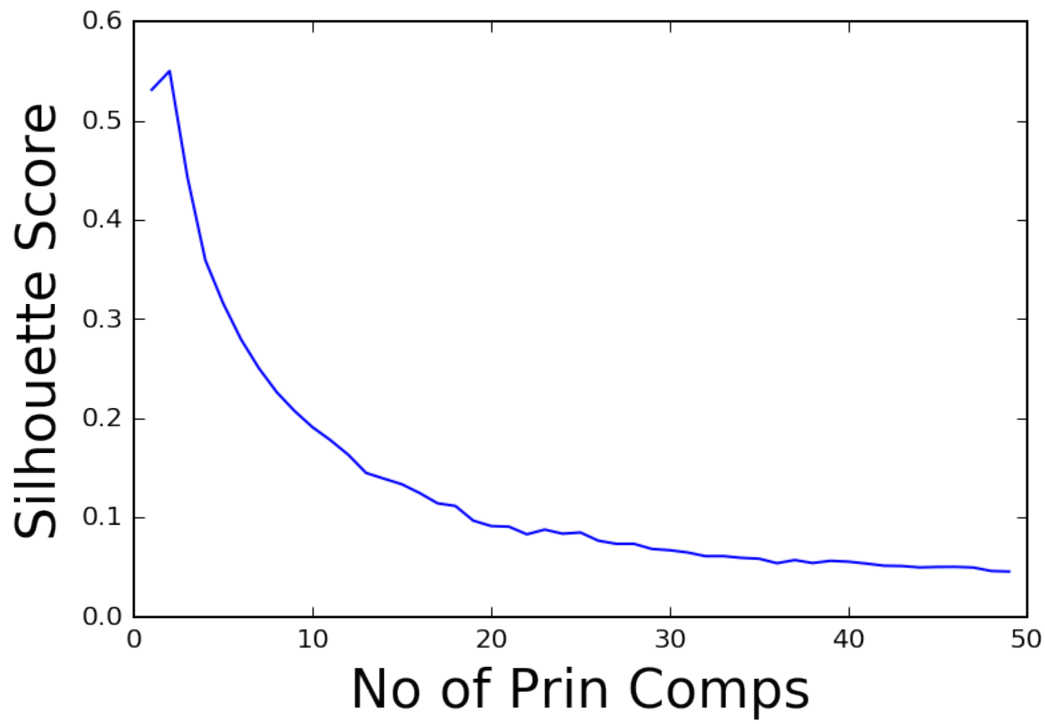


In [22]:
```
news_data.target_names
```

Out[22]: `['comp.os.ms-windows.misc', 'rec.sport.baseball', 'sci.space']`

In [23]:
```
plt.plot(range(1,50),ss)
plt.ylabel('Silhouette Score',size=20)
plt.xlabel('No of Prin Comps',size=20)
```

Out[23]: `<matplotlib.text.Text at 0x1174579e8>`



Note that we can get good accuracy with just **two** principal components.

1.6 Visualization

That's a good thing, because it means that we can **visualize** the data well with the help of PCA.

Recall that the challenge of visualization is that the data live in a high dimensional space.

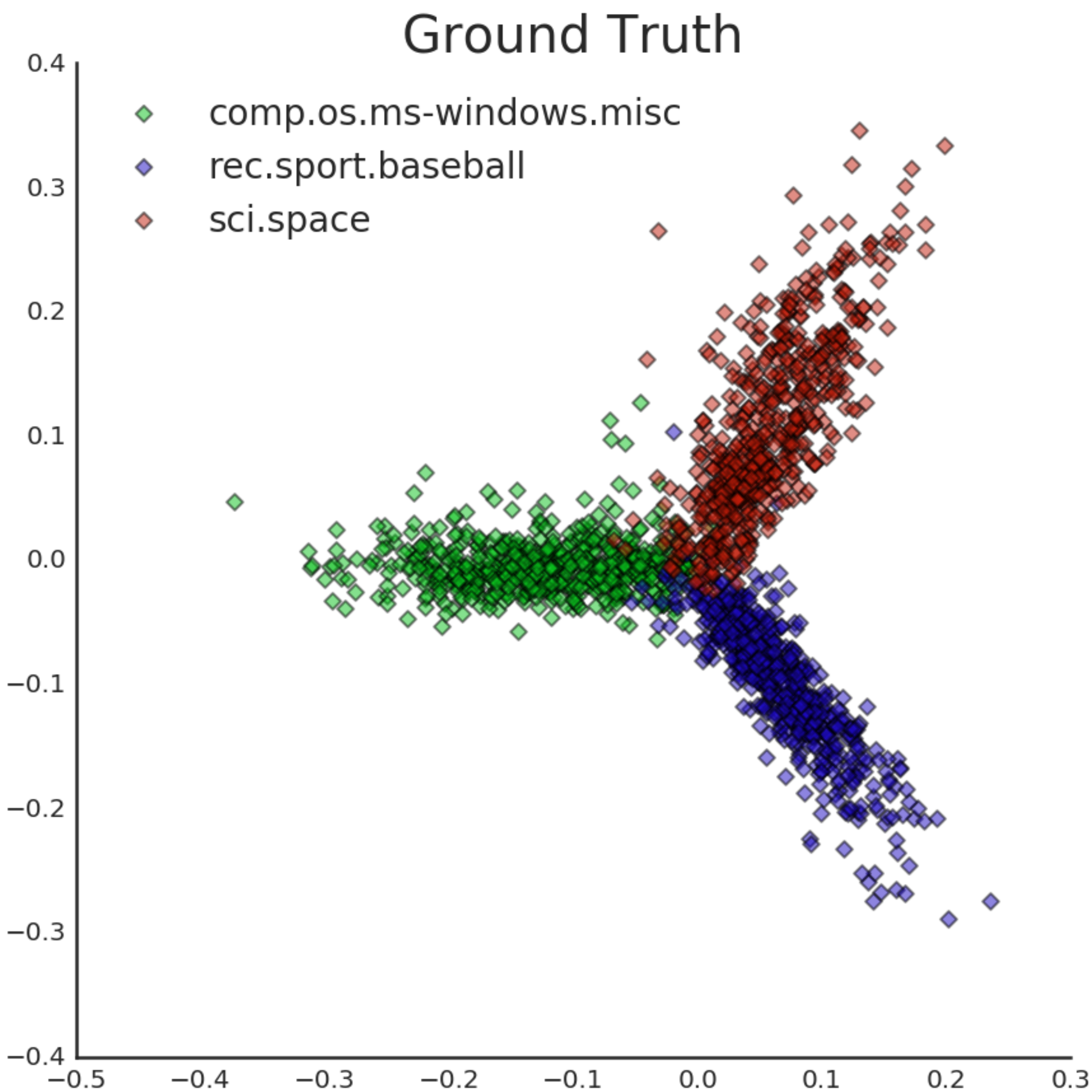
We can only look at 2 (or maybe 3) dimensions at a time, so it's not clear **which** dimensions to look at.

The idea behind using PCA for visualization is that since low-numbered principal components capture most of the **variance** in the data, these are the "directions" from which it is most useful to inspect the data.

We saw that the first two principal components were particularly large -- let's start by using them for visualization.

```
In [24]: import seaborn as sns
         Xk = u @ np.diag(s)
         with sns.axes_style("white"):
             fig, ax = plt.subplots(1,1,figsize=(7,7))
             cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
             for i, label in enumerate(set(news_data.target)):
                 point_indices = np.where(news_data.target == label)[0]
                 point_indices = point_indices.tolist()
                 plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
         label=news_data.target_names[i])
```

```
        plt.legend(prop={'size':14}, loc=2)
    sns.despine()
_ = plt.title('Ground Truth', size=20)
```



Points in this plot have been labelled with their "true" (aka "ground truth") cluster labels.

Notice how clearly the clusters separate and how coherently they present themselves. This is obvious an excellent visualization that is provided by PCA.

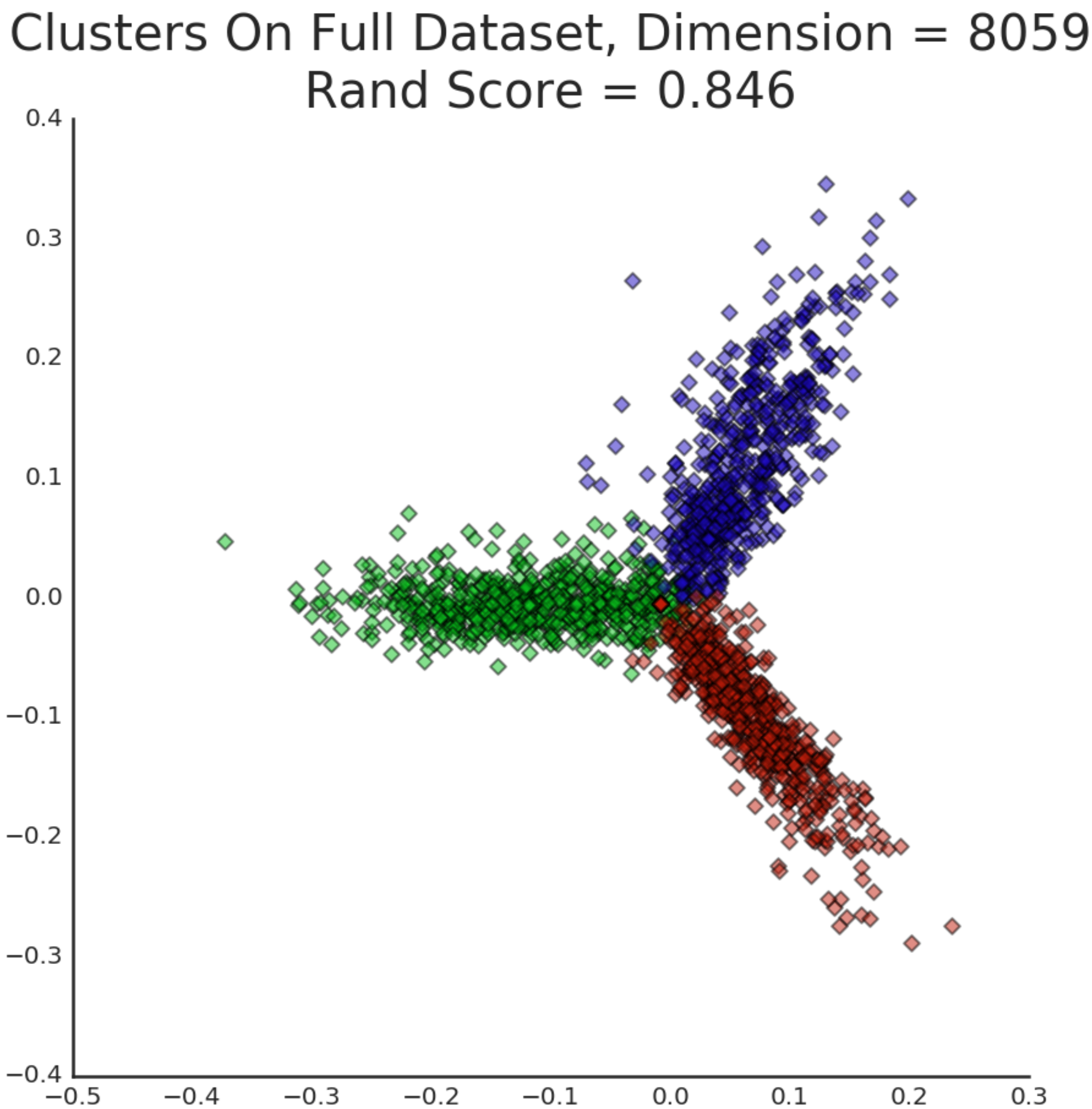
Since this visualization is so clear, we can use it to examine the results of our various clustering methods and get some insight into how they differ.

```
In [25]: k = 3
         kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=10,random_state=0)
         kmeans.fit_predict(dtm)
         centroids = kmeans.cluster_centers_
```

```
labels = kmeans.labels_
error = kmeans.inertia_

with sns.axes_style("white"):
    fig, ax = plt.subplots(1,1,figsize=(7,7))
    cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
    for i in range(k):
        point_indices = np.where(labels == i)[0]
        point_indices = point_indices.tolist()
        plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
label=news_data.target_names[i])
    sns.despine()
plt.title('Clusters On Full Dataset, Dimension = {}\nRand Score = {:0.3f}'.format(dtm.s
                                                                                  metrics.ad
          size=20)
```

Out[25]: <matplotlib.text.Text at 0x11756ef60>

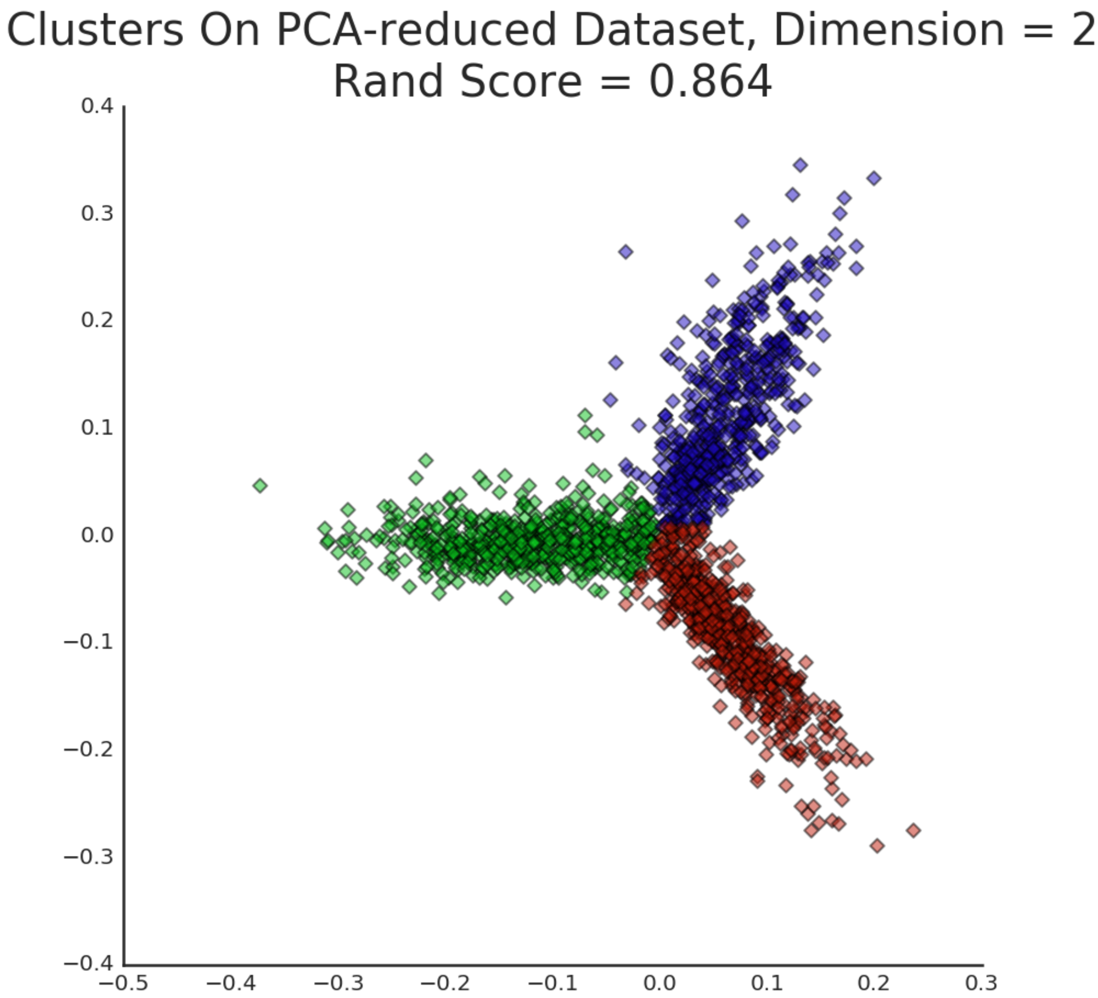



```
In [26]: k = 3
         kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=10,random_state=0)
         kmeans.fit_predict(Xk[:,:2])
         centroids = kmeans.cluster_centers_
         Xklabels = kmeans.labels_
         error = kmeans.inertia_

         with sns.axes_style("white"):
             fig, ax = plt.subplots(1,1,figsize=(7,7))
             cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
             for i, label in enumerate(set(news_data.target)):
                 point_indices = np.where(Xklabels == label)[0]
                 point_indices = point_indices.tolist()
```

```
        plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
    sns.despine()
plt.title('Clusters On PCA-reduced Dataset, Dimension = 2\nRand Score = {:0.3f}'.format
                                                                                metric
          size=20)
```

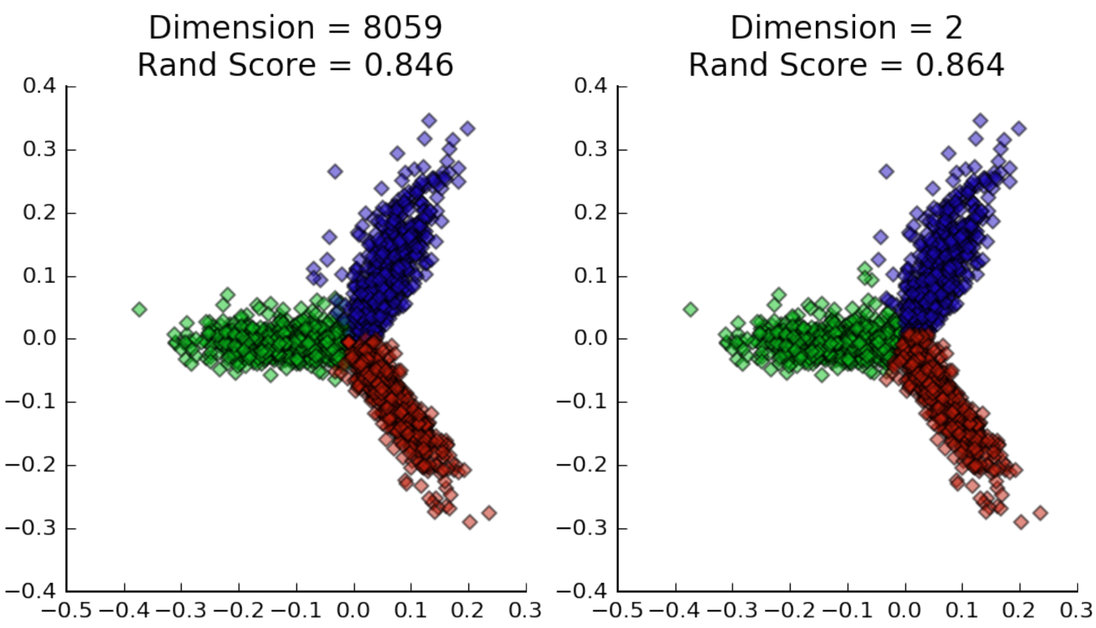
Out[26]: <matplotlib.text.Text at 0x1173c8630>



In [27]:
```
plt.figure(figsize=(8,4))
plt.subplot(121)
cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
for i in range(k):
        point_indices = np.where(labels == i)[0]
        point_indices = point_indices.tolist()
        plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
sns.despine()
```

```
    plt.title('Dimension = {}\nRand Score = {:0.3f}'.format(dtm.shape[1],
                                                                    metrics.ad
              size=14)
    plt.subplot(122)
    cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
    for i in range(k):
            point_indices = np.where(Xklabels == i)[0]
            point_indices = point_indices.tolist()
            plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
    sns.despine()
    plt.title('Dimension = 2\nRand Score = {:0.3f}'.format(
                                                                        metric
              size=14)
```

Out[27]: <matplotlib.text.Text at 0x117227d68>



What happens if we misjudge the number of clusters? Let's form 6 clusters.

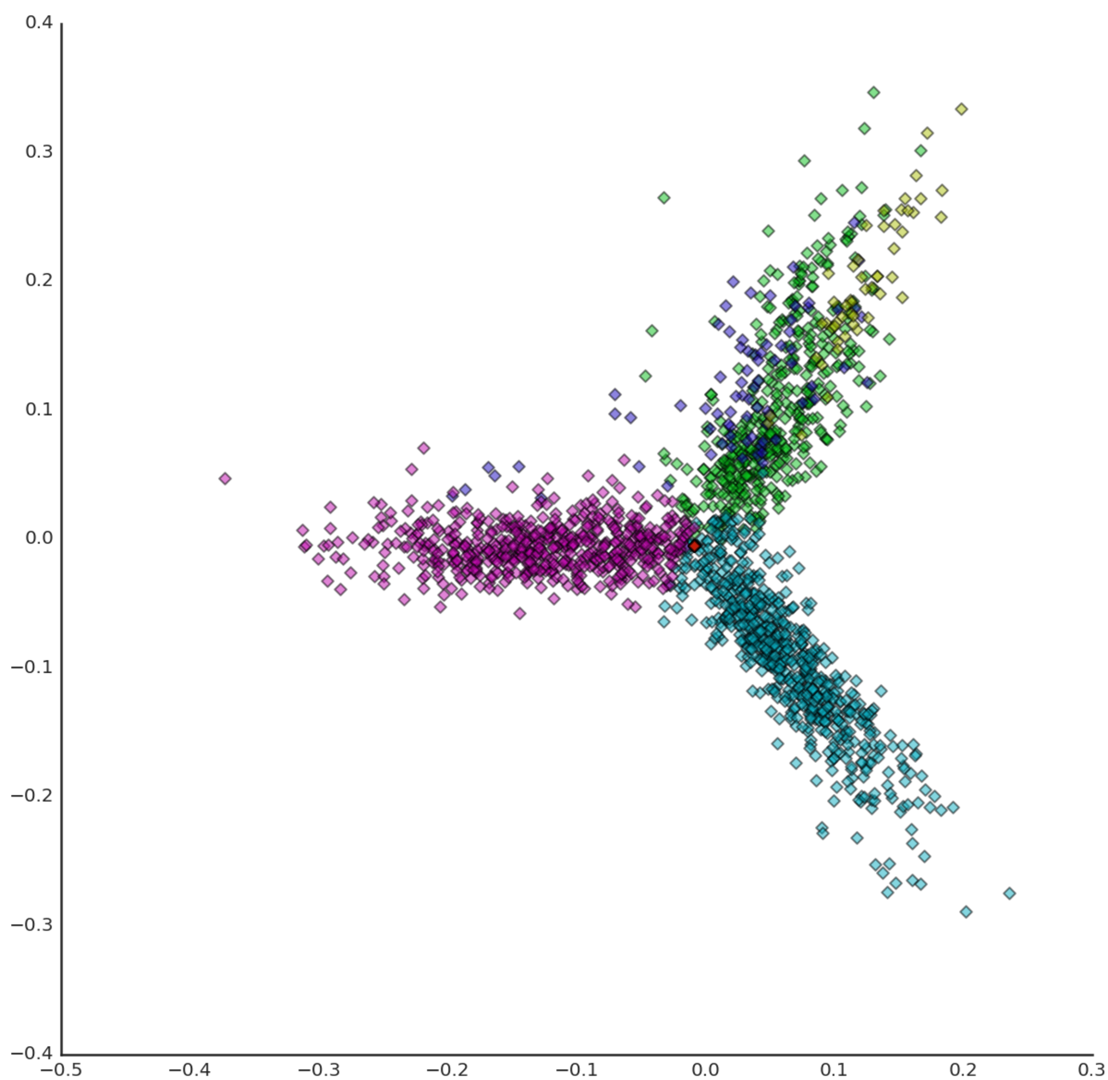
```
In [28]: k = 6
         kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=10,random_state=0)
         kmeans.fit_predict(Xk[:,:6])
         centroids = kmeans.cluster_centers_
         labels = kmeans.labels_
         error = kmeans.inertia_

         with sns.axes_style("white"):
             fig, ax = plt.subplots(1,1,figsize=(10,10))
             cmap = sns.hls_palette(n_colors=k, h=0.35, l=0.4, s=0.9)
```

```
    for i in range(k):
        point_indices = np.where(labels == i)[0]
        point_indices = point_indices.tolist()
        plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
    sns.despine()

print(metrics.adjusted_rand_score(labels,news_data.target))
```

```
0.747192242217
```



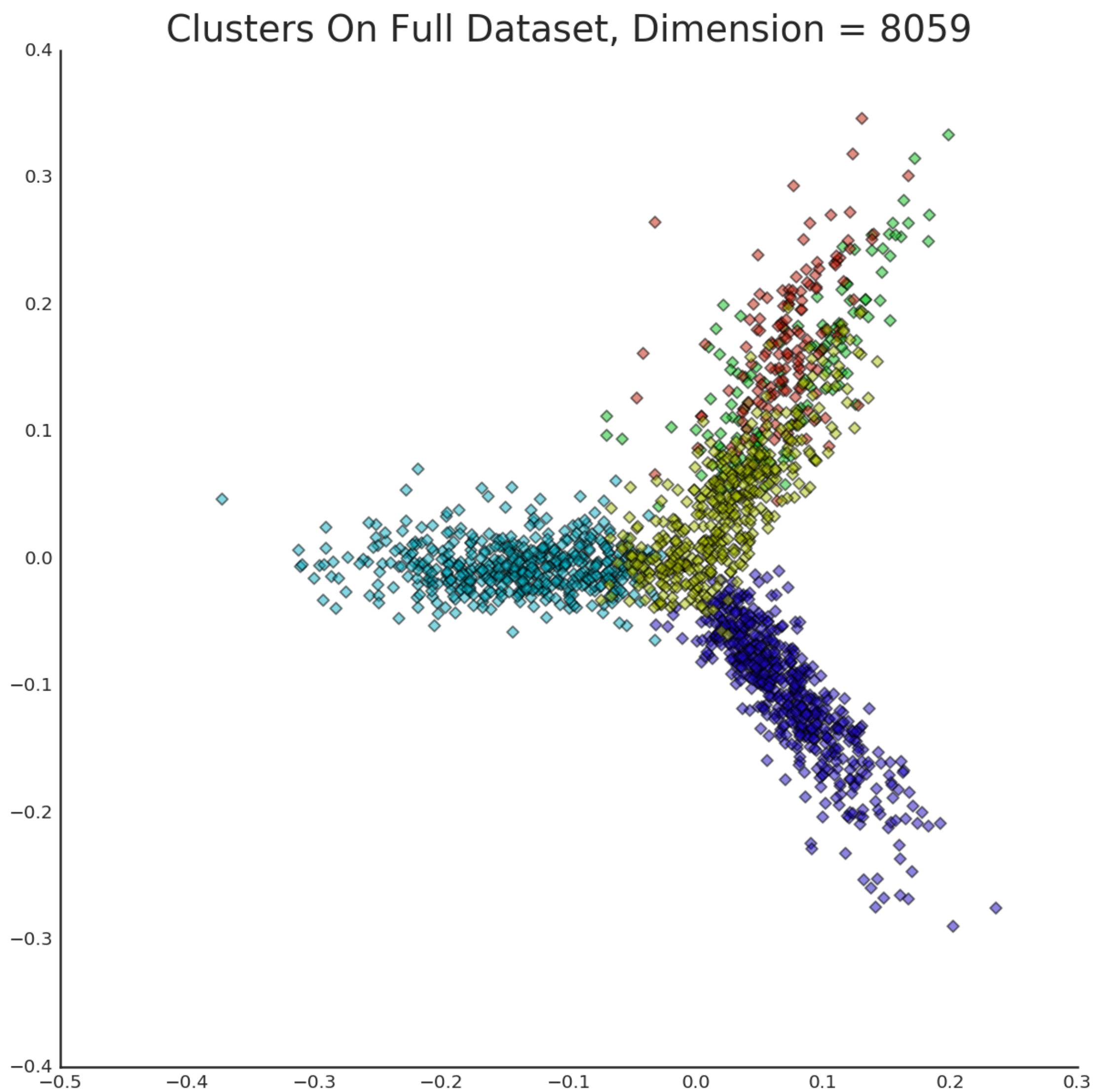

```
In [29]: k = 6
         kmeans = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=10,random_state=0)
         kmeans.fit_predict(dtm)
```

```python
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
error = kmeans.inertia_

with sns.axes_style("white"):
    fig, ax = plt.subplots(1,1,figsize=(10,10))
    cmap = sns.hls_palette(n_colors=k, h=0.35, l=0.4, s=0.9)
    for i in range(k):
        point_indices = np.where(labels == i)[0]
        point_indices = point_indices.tolist()
        plt.scatter(Xk[point_indices,0], Xk[point_indices,1], s=20, alpha=0.5, c=cmap[i
    sns.despine()
plt.title('Clusters On Full Dataset, Dimension = {}'.format(dtm.shape[1]),size=20)

print(metrics.adjusted_rand_score(labels,news_data.target))
```

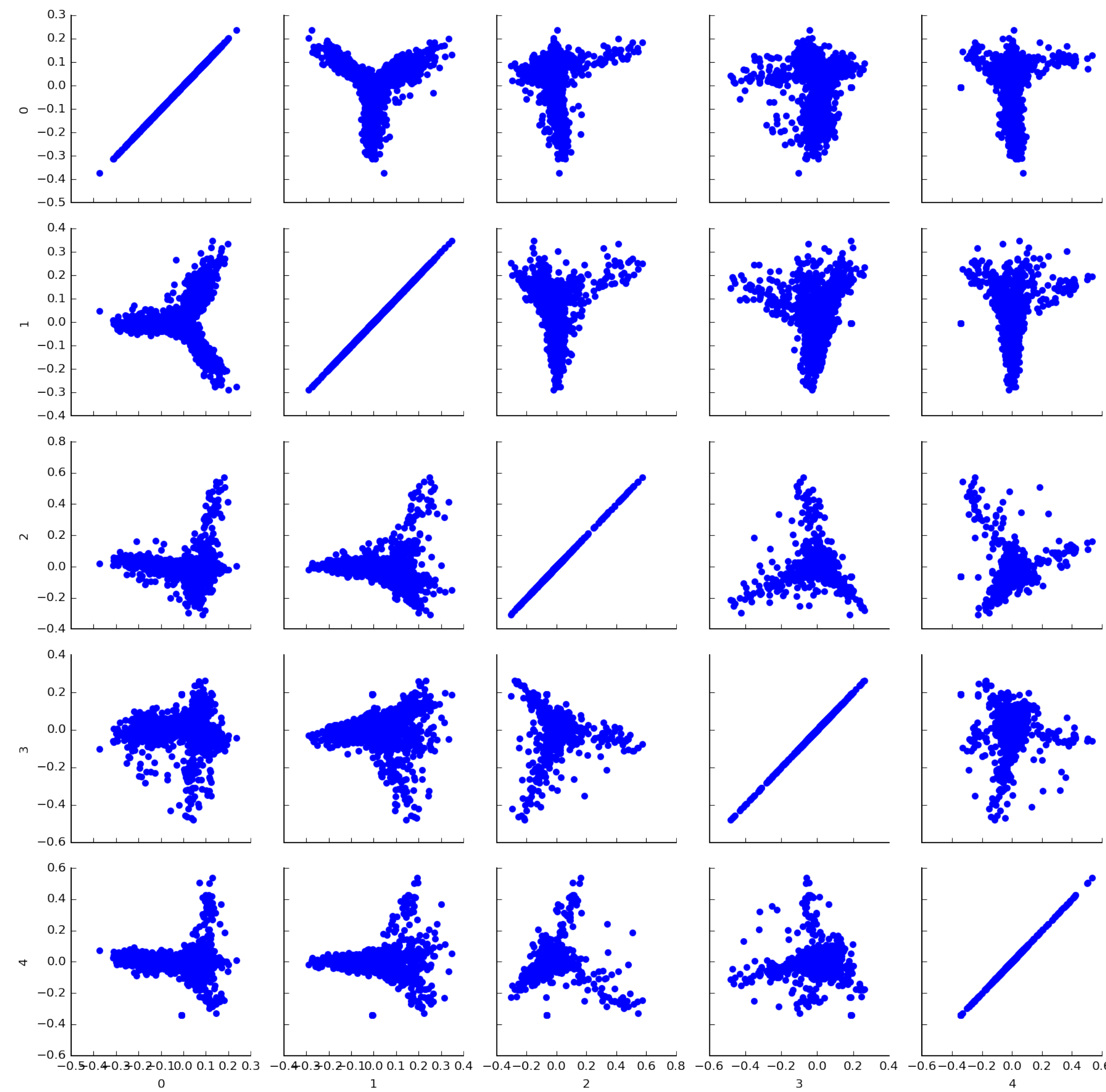
```
0.659952440171
```



What about the other principal components? Are they useful for visualization?
A common approach is to look at all pairs of (low-numbered) principal components.

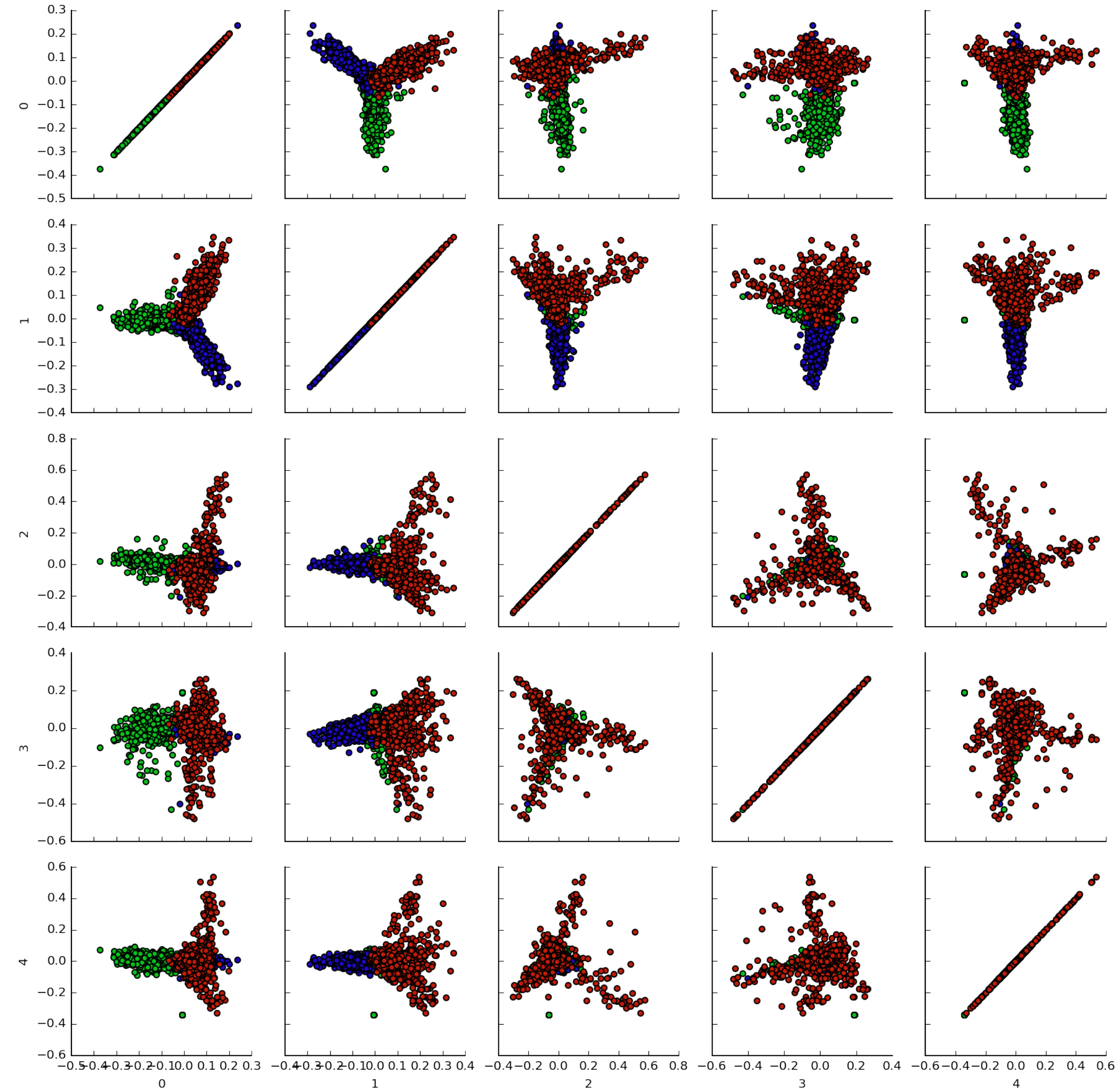
```
In [30]: import seaborn as sns
         k = 5
         Xk = u[:,:k] @ np.diag(s[:k])
         X_df = pd.DataFrame(Xk)
         g = sns.PairGrid(X_df)
         g.map(plt.scatter)
```

```
Out[30]: <seaborn.axisgrid.PairGrid at 0x11743bd30>
```



```
In [31]: k = 5
         Xk = u[:,:k] @ np.diag(s[:k])
         X_df = pd.DataFrame(Xk)
         g = sns.PairGrid(X_df)
         def pltColor(x,y,label,color):
             cmap = sns.hls_palette(n_colors=3, h=0.35, l=0.4, s=0.9)
             for i in range(3):
                 point_indices = np.where(news_data.target == i)[0]
                 point_indices = point_indices.tolist()
                 plt.scatter(x[point_indices], y[point_indices], c=cmap[i])
             sns.despine()
         g.map(pltColor)

Out[31]: <seaborn.axisgrid.PairGrid at 0x119561a90>
```



1.7 Looking at the Topics

```
In [32]: for i in range(6):
             top = np.argsort(vt[i])
             topterms = [terms[top[0,f]] for f in range(12)]
             print (i, topterms)
```

```
0 ['window', 'file', 'driver', 'dos', 'use', 'card', 'font', 'ms', 'problem', 'program', 'mous',
1 ['game', 'team', 'player', 'pitch', 'run', 'basebal', 'win', 'hit', 'edu', 'score', 'year', 'p
2 ['access', 'nasa', 'digex', 'pat', 'gov', 'jpl', 'baalk', 'com', '___', 'kelvin', '__', 'prb']
3 ['access', 'digex', 'pat', 'com', 'prb', 'net', 'express', 'onlin', 'communic', 'usa', 'dseg',
4 ['ax', 'henri', 'toronto', 'zoo', 'com', 'spencer', 'zoolog', 'access', 'jpl', 'baalk', 'orbit
5 ['window', 'run', 'nasa', 'year', 'file', 'team', 'game', 'win', 'dos', 'hit', 'henri', 'orbit
```