12-Anomaly-Detection-SVD-III

October 19, 2017

1 Anomaly Detection (and SVD-III)

Today we'll discuss an important topic related to unsupervised learning: **anomaly detection.**

Anomalies are objects that are different from most other objects.

Anomalies are also called "outliers".

Furthermore, we usually expect that anomalies are different in a **qualitative** sense as well.

> An outlier is an observation that differs so much from other observations as to arouse suspicion that it was generated by a different mechanism

-- Douglas Hawkins

Why might we be interested in anomalies?

- **Fraud Detection** - stolen credit cards
- **Intrusion Detection** - attacks on computer systems
- **Public Health** - occurrence of disease in a particular area
- **Medicine** - a set of symptoms may indicate a disease

Anomaly detection presents a number of challenges.

It is an **unsupervised** method -- so validation is hard * It is hard to know that your set of anomalies is correct * It is hard to know how many anomalies there are in the data

The main assumption made in anomaly detection:

There are many more "normal" observations than "abnormal" (anomalies) in the data.

Methodologically, anomaly detection proceeds as follows:

1. Build a profile of "normal" data objects
    - These can be patterns, summary statistics, or more complicated models
2. Use the "normal" profile to detect anomalies
    - These are observations whose characteristics differ significantly from the normal profile.

1.1 Approaches To Anomaly Detection

The idea that "normal behavior is what is most frequently observed" is the basis for most anomaly detection methods.

It suggests a number of approaches.

Model-Based Methods.

Here, we assume that a model for the data will describe most of the data. Data points that are not well described by the model are potentially anomalies.

1) Use the data to estimate the parameters of a probability distribution. For example, one might estimate a normal distribution from the data.

- Then an object that is very **unlikely** under the model may be an anomaly.

2) Model the data as a set of clusters (cluster the data).

- Then an object that does not strongly belong to any cluster may be an anomaly.

3) Model the data using a regression.

- Then an object that is far from its predicted value may be an anomaly.

Other Methods.

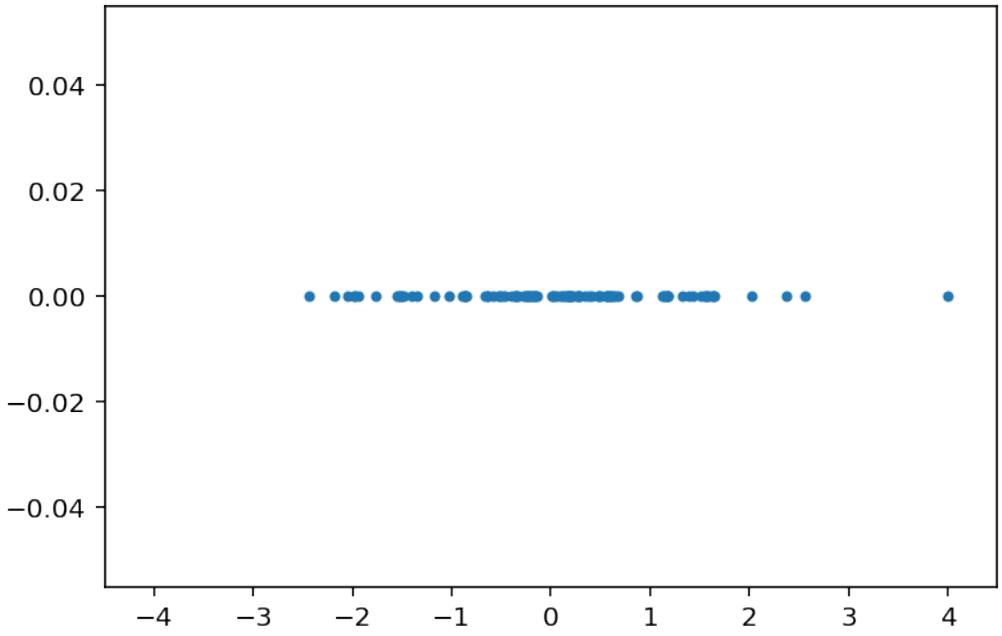
If you cannot build a model of the data, you can still:

1. Define an anomaly as one that is distant from all (or most) other objects.
2. Define an anomaly as one that is in an unusually-low-density region.

1.2 Model-Based Detection: 1-D Gaussian

To start, we will examine a very simple case: anomaly detection in 1-D.

In [33]:
```
n_samples = 100

# construct random samples from a normal distribution
data = np.random.randn(n_samples, 1)

# add an outlier
data = np.concatenate([data, np.array([[4]])])

# plot
plt.plot(data, np.zeros(n_samples+1), '.')
_ = plt.xlim([-4.5, 4.5])
```

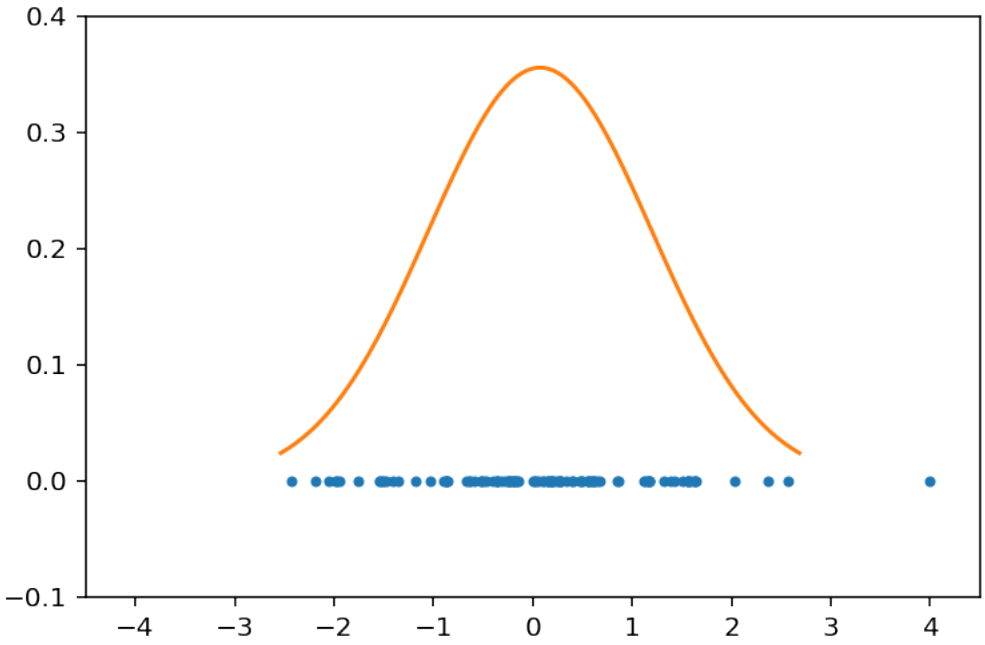


In [34]:
```
# construct MLE estimates of mean and standard deviation
m_est = np.mean(data)
std_est = np.std(data)
```

In [35]:
```
# create a normal model for the data using estimated parameters
import scipy.stats
norm = scipy.stats.distributions.norm(m_est,std_est)
```

In [36]:
```
plt.plot(data, np.zeros(n_samples+1), '.')
plt.xlim([-4.5, 4.5])
plt.ylim([-0.1,0.4])
x = np.linspace(norm.ppf(0.01),
                norm.ppf(0.99), 100)
plt.plot(x, norm.pdf(x), '-')
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x113595080>]
```



Let's calculate the probability desnity at each data point under this model.

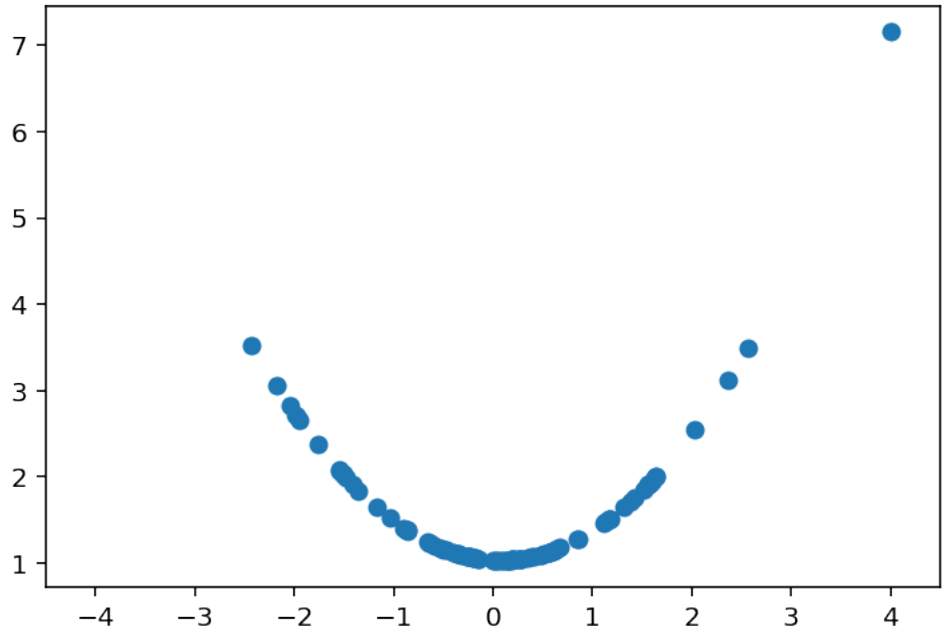
```
In [37]: prob_data = [norm.pdf(x) for x in data]
         prob_data

Out[37]: [array([ 0.34832788]),
          array([ 0.02927776]),
          array([ 0.33002965]),
          array([ 0.33383069]),
          array([ 0.13096349]),
          array([ 0.24919316]),
          array([ 0.2504266]),
          array([ 0.31749407]),
          array([ 0.32361927]),
          array([ 0.15601613]),
          array([ 0.31206965]),
          array([ 0.15848651]),
          array([ 0.14828404]),
          array([ 0.22831897]),
          array([ 0.14685824]),
          array([ 0.34206688]),
          array([ 0.3239734]),
          array([ 0.13198598]),
          array([ 0.0704705]),
          array([ 0.19140898]),
          array([ 0.33019906]),
          array([ 0.30878862]),
```

```
array([ 0.34270767]),
array([ 0.31314548]),
array([ 0.30752154]),
array([ 0.17186493]),
array([ 0.35537951]),
array([ 0.07786713]),
array([ 0.29121463]),
array([ 0.30824244]),
array([ 0.34200884]),
array([ 0.33046903]),
array([ 0.21942757]),
array([ 0.2901119]),
array([ 0.19152925]),
array([ 0.35357615]),
array([ 0.03024428]),
array([ 0.31686288]),
array([ 0.3448875]),
array([ 0.35487699]),
array([ 0.04711625]),
array([ 0.31836348]),
array([ 0.33303006]),
array([ 0.35541862]),
array([ 0.12552462]),
array([ 0.12517527]),
array([ 0.32042719]),
array([ 0.21978762]),
array([ 0.14205773]),
array([ 0.24467862]),
array([ 0.21884151]),
array([ 0.33901898]),
array([ 0.06693899]),
array([ 0.29852549]),
array([ 0.279742]),
array([ 0.13525063]),
array([ 0.34972975]),
array([ 0.13425488]),
array([ 0.33997384]),
array([ 0.35366944]),
array([ 0.34713598]),
array([ 0.35381189]),
array([ 0.34057821]),
array([ 0.34908498]),
array([ 0.28665413]),
array([ 0.35081862]),
array([ 0.35507674]),
array([ 0.13621085]),
array([ 0.34181514]),
array([ 0.25213594]),
```

```
 array([ 0.05936141]),
 array([ 0.33270255]),
 array([ 0.14825339]),
 array([ 0.04387729]),
 array([ 0.1476018]),
 array([ 0.24930937]),
 array([ 0.21977488]),
 array([ 0.23039161]),
 array([ 0.06562876]),
 array([ 0.27809624]),
 array([ 0.31497866]),
 array([ 0.35449228]),
 array([ 0.34918015]),
 array([ 0.35555059]),
 array([ 0.13569797]),
 array([ 0.32492799]),
 array([ 0.34509585]),
 array([ 0.35494984]),
 array([ 0.3499265]),
 array([ 0.09294136]),
 array([ 0.17922125]),
 array([ 0.35235899]),
 array([ 0.32267609]),
 array([ 0.35346933]),
 array([ 0.34608849]),
 array([ 0.33041909]),
 array([ 0.35552327]),
 array([ 0.35474312]),
 array([ 0.22361327]),
 array([ 0.34533434]),
 array([ 0.0007812])]
```

To make this easier to analyze, let's work with the (negative) log probability of the data:

```
In [38]: import math
         log_prob_data = [-math.log(x) for x in prob_data]
         plt.plot(data, log_prob_data, 'o')
         _ = plt.xlim([-4.5,4.5])
```



This plot makes clear that there is one point that is **very** unlikely under our model. We would be justified in identifying this as an anomaly.

In particular, the probability of seeing a point this extreme or worse under this model is 0.0002337.

```
In [43]: 1 - norm.cdf(data[-1])
```

```
Out[43]: array([ 0.0002337])
```

We would not expect to see a sample this extreme unless we sampled the distribution $1/0.0002337 = 4278$ times.

However we have only 101 data points.

An important point.

Notice that we estimated the mean and standard deviation of our model using **all** the data -- including the point that we later decided was an anomaly.

Of course the correct parameter estimation should **not** have included the anomalous data point, if it truly "was generated by a different mechanism."

On one hand, our estimation approach gives us an approximation to the true distribution.

This approximation may be justified under the assumption that "most of the data points are not anomalies".

And since we don't know the anomalies in advance (of course) we cannot simply remove them before we estimate the parameters.

There are more sophisticated approaches to try to address this problem -- we won't discuss them, but you should be aware that it is possible to do better than what we did here.

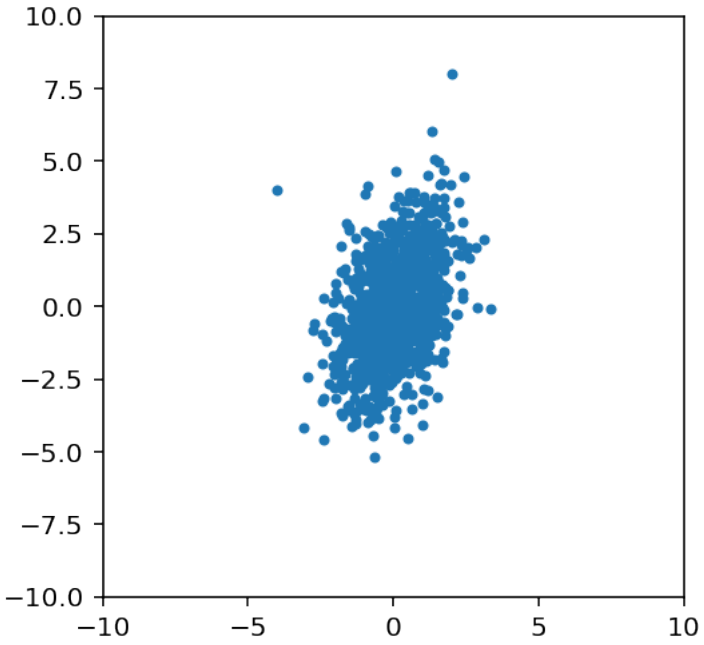
1.3 Extending to Multiple Dimensions

For data with multiple features we would like to take the same approach.

We would like to identify points as outliers if they have low probability under a **multivariate** Gaussian model.

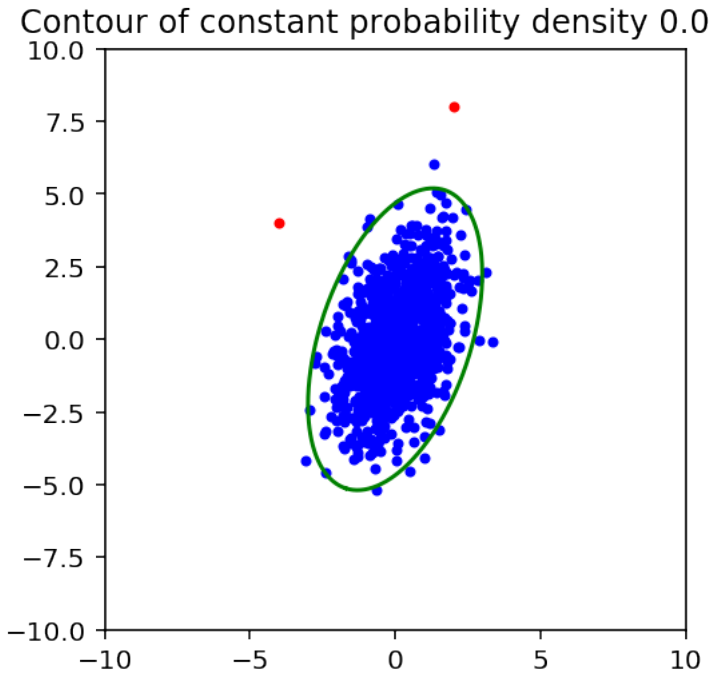
```
In [9]: n_samples = 1000
        cov = np.array([[1., 0.75],[0.75, 3]])
        mean = np.array([0.,0])
        apt1 = np.array([2,8])
        apt2 = np.array([-4,4])
        data = np.random.multivariate_normal(mean, cov, n_samples)
        data = np.concatenate([data,np.array([apt1,apt2])])
```

```
In [10]: plt.plot(data[:,0],data[:,1],'o',markersize=3)
         plt.axis('square')
         plt.xlim([-10,10])
         _ = plt.ylim([-10,10])
```




```
In [11]: theta = np.linspace(0,2*math.pi,500)
         coords = np.array([np.sin(theta), np.cos(theta)])
         cov = np.array([[1., 0.75],[0.75, 3]])
         lam, evec = np.linalg.eig(cov)
         c = evec @ np.diag(np.sqrt(lam))
         coords = 3 * c @ coords
         plt.plot(data[:-2,0],data[:-2,1],'o',markersize=3,color='b')
```

```
plt.plot(data[-2:,0],data[-2:,1],'o',markersize=3,color='r')
plt.plot(coords[0], coords[1], '-', color='g')
plt.axis('square')
plt.xlim([-10,10])
plt.ylim([-10,10])
_ = plt.title(r'Contour of constant probability density {:0.4f}'.format(norm.pdf(3)))
```



Consider the following candidates for outliers: (2,8) and (-4,4).
These are marked in red.

```
In [12]: print('{} is {:0.3f} from the cluster center.'.format(apt1,np.linalg.norm(apt1)))
         print('{} is {:0.3f} from the cluster center.'.format(apt2,np.linalg.norm(apt2)))
```

```
[2 8] is 8.246 from the cluster center.
[-4  4] is 5.657 from the cluster center.
```

Which one is more of an outlier?
We have to take into account the **probability** of the point under the (presumed) multivariate Gaussian distribution.
Using standard (MLE) methods (on all the data), we estimate the Gaussian distribution to have mean

$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and covariance

$$\Sigma = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 3 \end{bmatrix}$$

A convenient way to express the probability is via its negative log:

$$-\log(P[\mathbf{x}]) - C = (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)$$

(C is a constant that does not depend on $\mathbf{x}$.)

The expression on the right is called the **Mahalanobis distance.**

It is essentially the distance to the center of the Gaussian, scaled to take into account the **shape** of the Gaussian.

```
In [13]: p = np.array([apt1]).T
         mp = p.T @ np.linalg.inv(cov) @ p
         print('{} has Mahalanobis distance {:0.3f}.'.format(apt1,mp[0,0]))
         p = np.array([apt2]).T
         mp = p.T @ np.linalg.inv(cov) @ p
         print('{} has Mahalanobis distance {:0.3f}.'.format(apt2,mp[0,0]))
```

```
[2 8] has Mahalanobis distance 21.333.
[-4  4] has Mahalanobis distance 36.103.
```

Hence we can conclude that (-4, 4) is more of an outlier than (2, 8):

- Although it is closer to the cluster center in Euclidean distance,
- It is further from the cluster center in Mahalanobis distance.

1.4 High Dimensional Data

When our data objects have hundreds (or millions) of features, we can no longer build a distributional model.

For a dataset with 1,000 features, we need to build a 1,000 $\times$ 1,000 covariance matrix -- which has a million elements.

We have a problem of high dimensionality -- and a natural approach is to consider dimensionality reduction.

SVD to the rescue again!

As we've seen, dimensionality reduction will work if the dataset shows low effective rank.

Let's consider how we might do anomaly detection when data has low effective rank.

Since * the principle of anomaly detection is that most objects are normal, and
* the data has low effective rank,

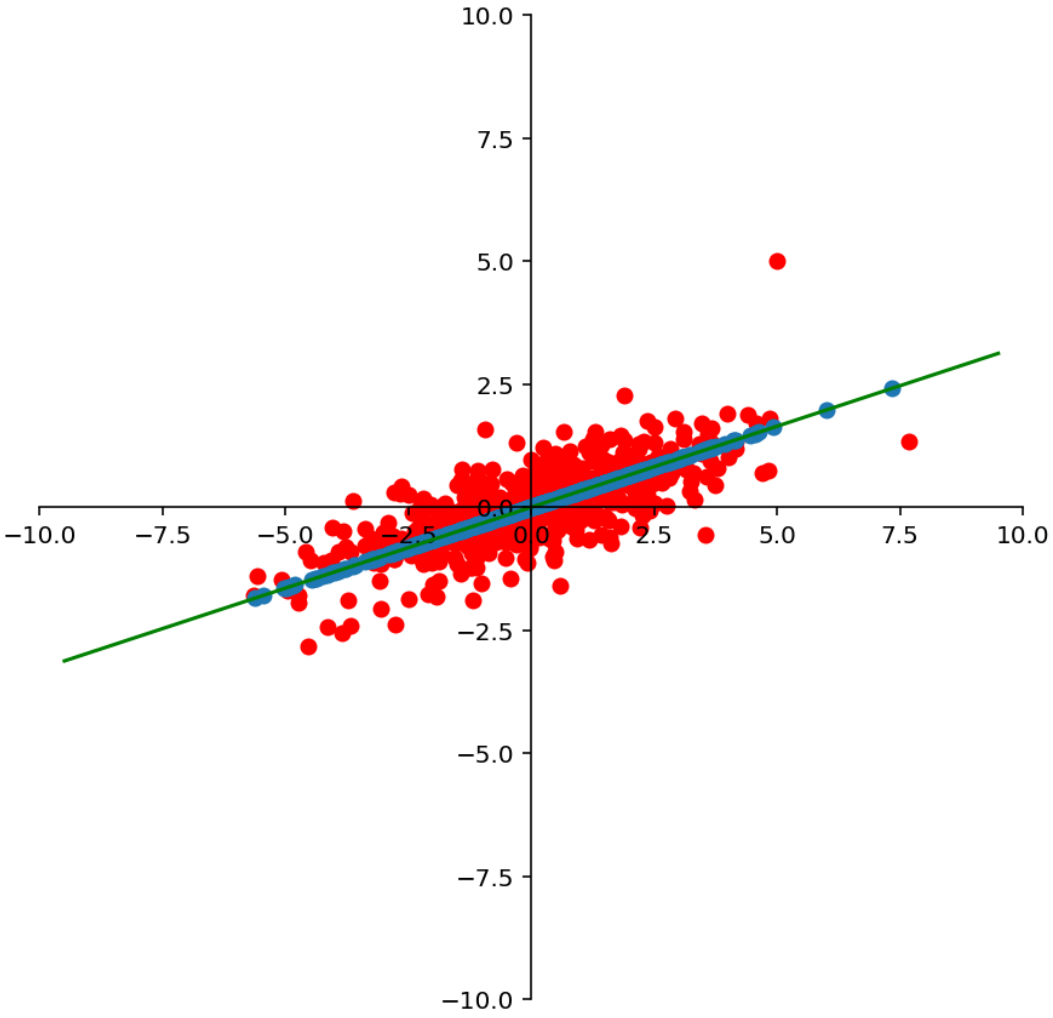
one way to look for anomalies is to find objects that are **not** well described by the low rank model.

Let's start with our usual (toy) model in 2-D.

(Keeping in mind that this is to build intuition for the real problem, which is in high dimension.)

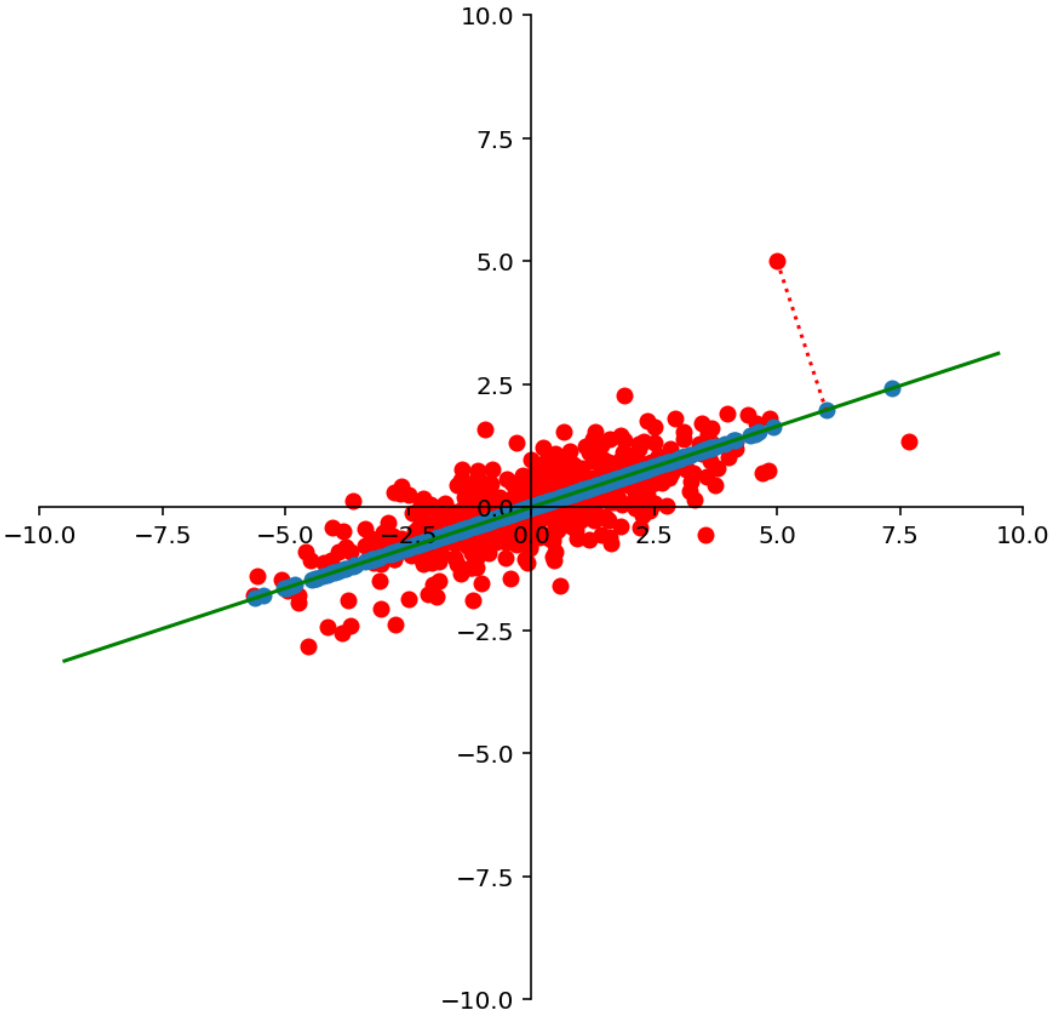
In [14]:
```
n_samples = 500
# Create correlated multivariate Gaussian samples
C = np.array([[0.1, 0.6], [2., .6]])
np.random.seed(1)
X = np.random.randn(n_samples, 2) @ C + np.array([-6, 3])
# Mean center
Xc = X - np.mean(X,axis=0)
# Create an anomalous data point
apt = np.array([5,5])
Xc = np.concatenate([Xc, np.array([apt])],axis=0)
# SVD of all data
u, s, vt = np.linalg.svd(Xc,full_matrices=False)
orthog_dir = np.array([-vt[0,1], vt[0,0]])
# project points onto subspace
scopy = s.copy()
scopy[1] = 0.
reducedX = u @ np.diag(scopy) @ vt
apt_proj = reducedX[-1,:]
```

In [15]:
```
# plot
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-')
```



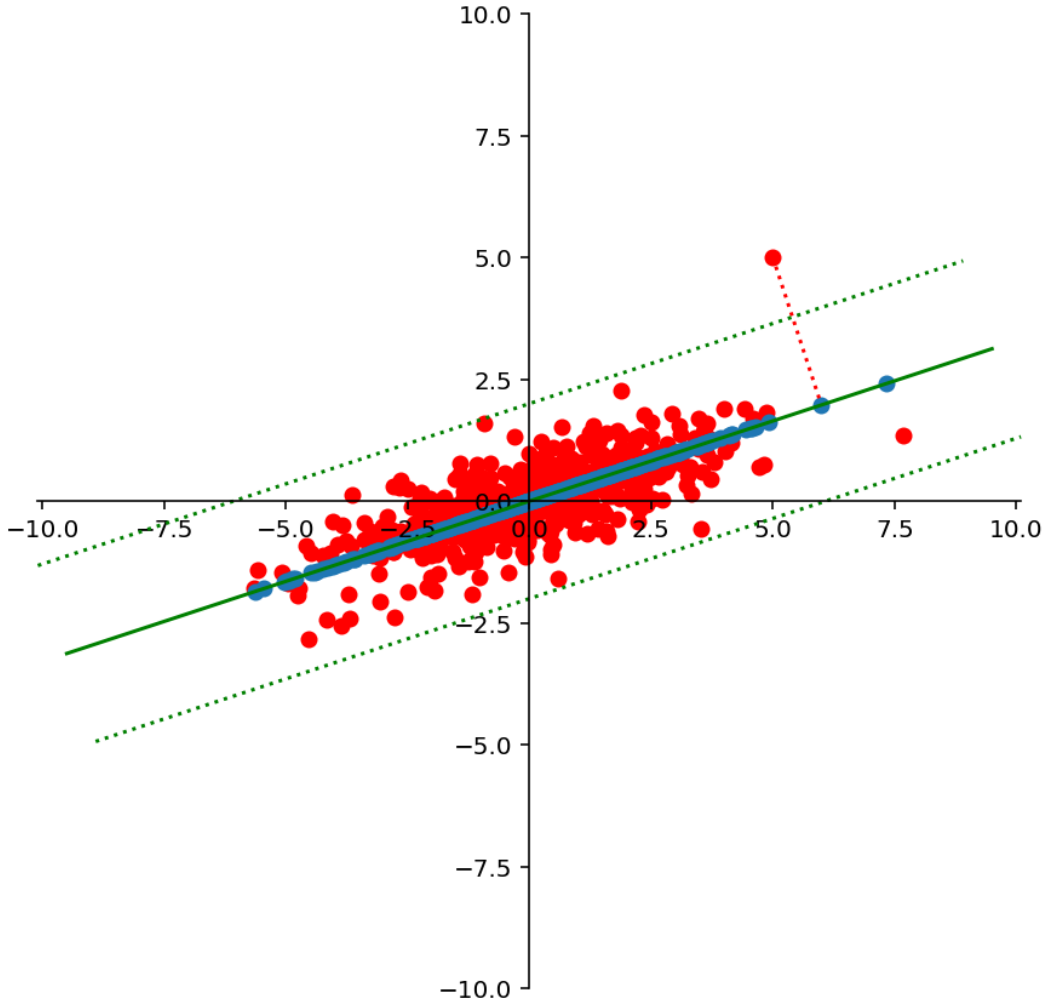
Is there a point here that is not well described by the low-rank (rank-1) model? How would we quantify this fact?

```
In [16]: # plot
         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])
         plt.plot([apt[0],apt_proj[0]],[apt[1],apt_proj[1]],'r:')
         endpoints = np.array([[-10],[10]]) @ vt[[0],:]
         _ = plt.plot(endpoints[:,0], endpoints[:,1], 'g-')
```



What is the distance of the anomalous point from the subspace?
It is simply the **length of the difference** between the point and its projection in the subspace.

```
In [17]: # plot
         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])
         plt.plot([apt[0],apt_proj[0]],[apt[1],apt_proj[1]],'r:')
         endpoints = np.array([[-10],[10]]) @ vt[[0],:]
         alpha = 1.9
         plt.plot(alpha*orthog_dir[0]+endpoints[:,0], alpha*orthog_dir[1]+endpoints[:,1], 'g:')
         plt.plot(endpoints[:,0]-alpha*orthog_dir[0], endpoints[:,1]-alpha*orthog_dir[1], 'g:')
         _ = plt.plot(endpoints[:,0], endpoints[:,1], 'g-')
```



So to do anomaly detection via the **subspace** method, we set a threshold on the distance of each point from the subspace.

1.5 Anomaly Detection via the Low-Rank Approximation

In practice, this is a simple process.

Given a data matrix A:

1. Compute the Singular value decomposition of A,

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

2. Compute a low-rank approximation to A,

$$N = U'\Sigma'(V')^T.$$

3. Compute the residuals not explained by N:

$$O = A - N.$$

4. Identify the rows of O with largest ℓ_2 norm: these rows correspond to anomalies.

In this recipe, rows of O are the difference vectors between each point and its projection in the subspace.

So the ℓ_2 norm gives us the distance of each point from the subspace.

There are two unspecified steps in the process:

1. Selecting the columns of U to be used in forming N
2. Deciding how many of the largest rows of O are anomalies.

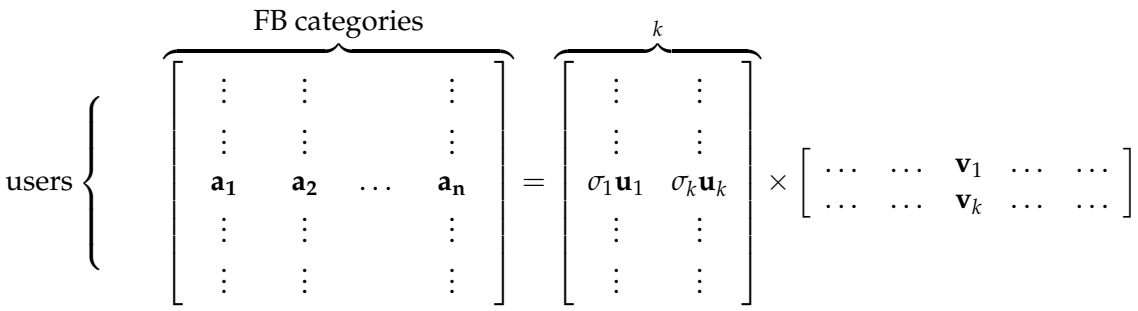
For 1, the general idea is to choose a k at the knee of the singular value plot.

For 2, there are statistical methods that generally work reasonably well. However, one can always just rank the points by their residual norm, which is what we'll do.

1.5.1 Example 1: Facebook Spatial Likes

This data consists of the number of 'Likes' during a six month period, for each of 9000 users across the 210 content categories that Facebook assigns to pages.

Rows are users, Columns are categories. A is 9000×210.



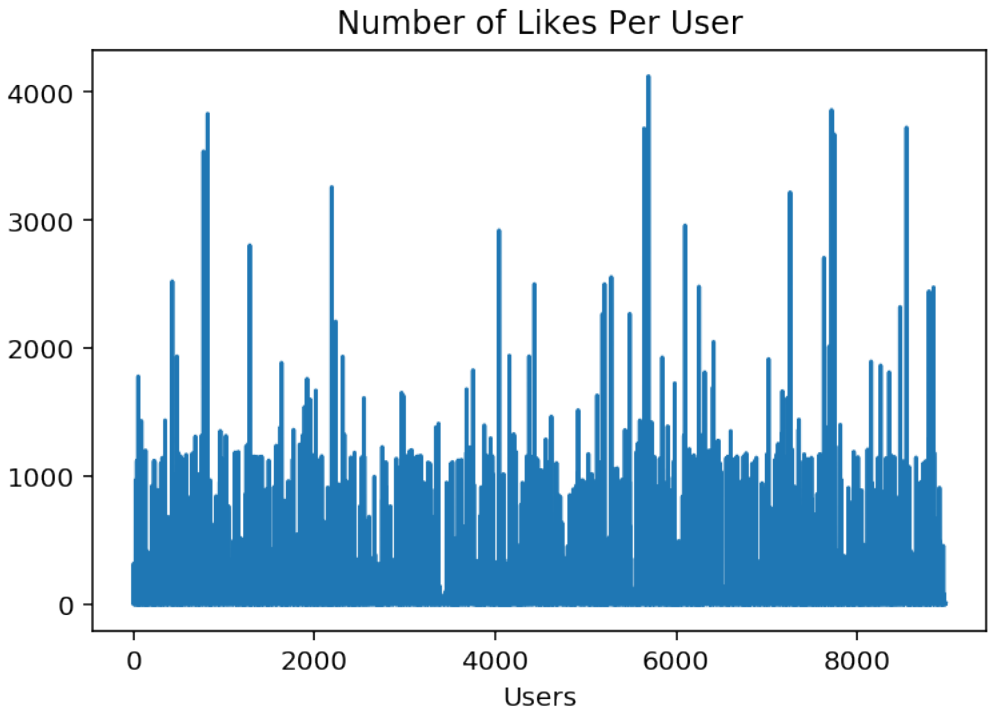
$$A = U\Sigma V^T$$

```
In [18]: data = np.loadtxt('data/social/spatial_data.txt')
         data[:10]
```

```
Out[18]: array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 1.,  0.,  0., ...,  0.,  2.,  8.],
                ...,
                [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                [ 0.,  0.,  0., ...,  0.,  0.,  1.],
                [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

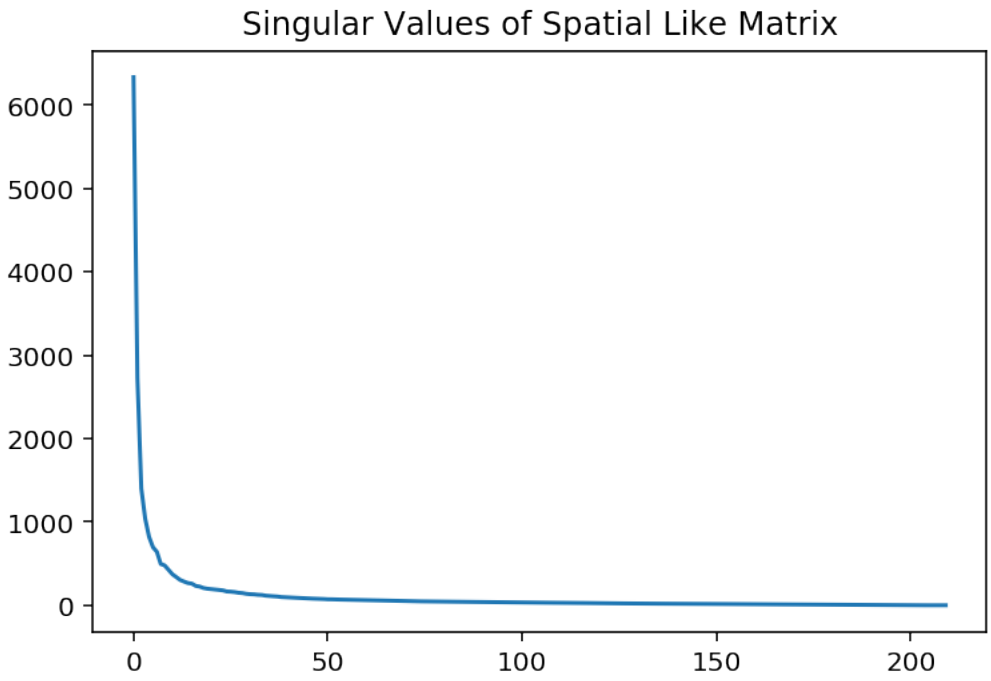
First we'll look at the total number of likes for each user (the row sums).

```
In [19]: FBSpatial = data[:,1:]
         FBSnorm = np.linalg.norm(FBSpatial,axis=1,ord=1)
         plt.plot(FBSnorm)
         plt.title('Number of Likes Per User')
         _ = plt.xlabel('Users')
```



Now let's check whether the low rank approximation holds.

```
In [20]: u,s,vt = np.linalg.svd(FBSpatial,full_matrices=False)
         plt.plot(s)
         _ = plt.title('Singular Values of Spatial Like Matrix')
```

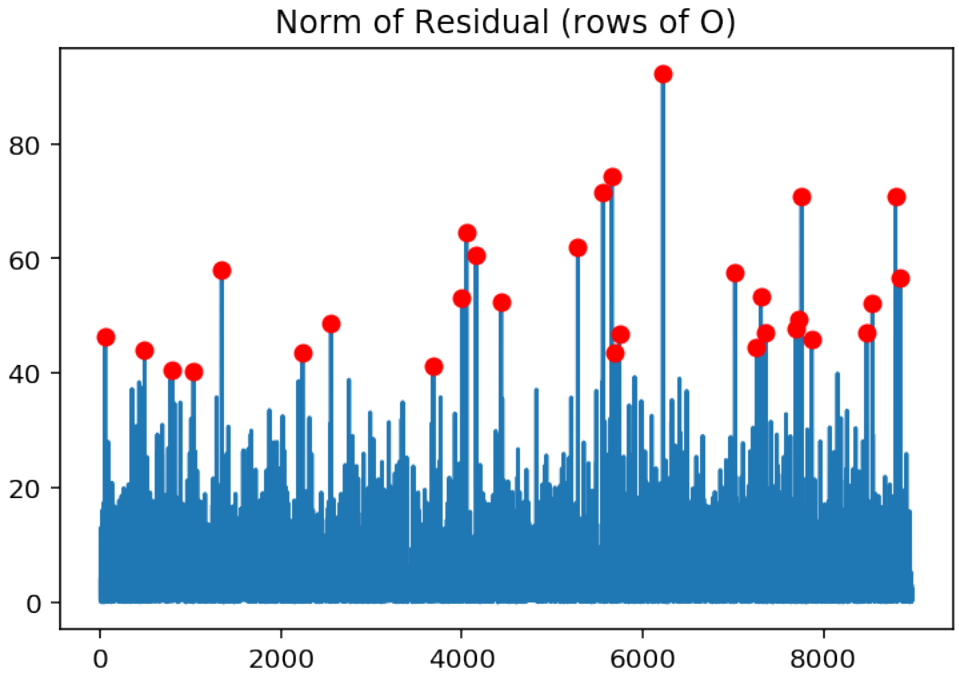


We'll approximate this data as having effective rank 25.
Now let's

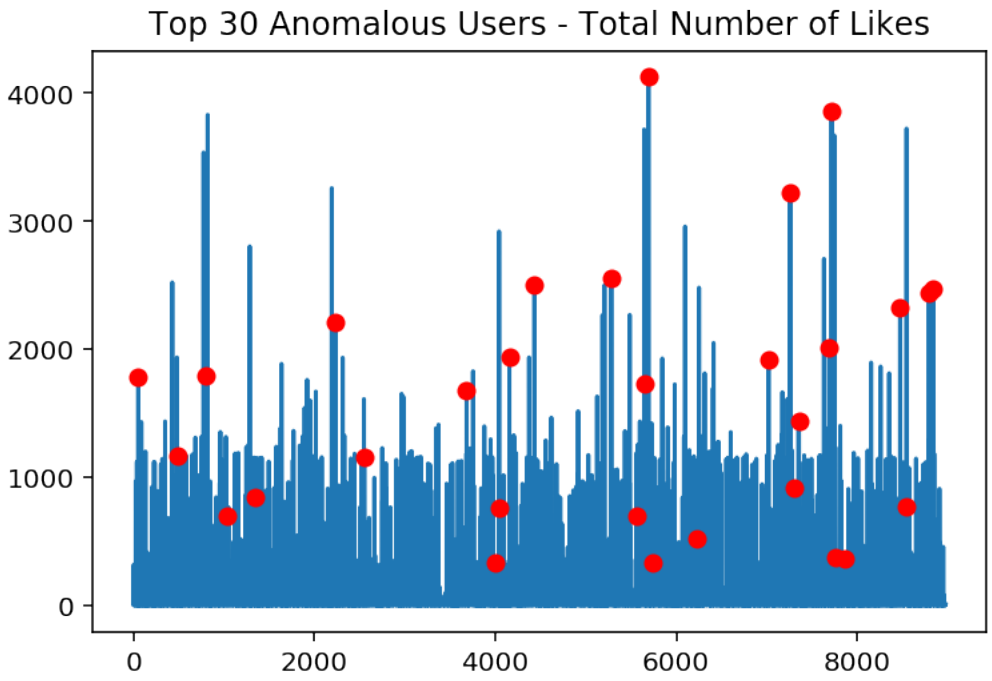
1. Separate the portion of the data lying in the normal space from the anomalous space,
2. Identify the top 30 anomalous users (having the largest residual component), and
3. Plot their total number of likes against the set of all users.

```
In [21]: # zero out all singular values other than the first 25
         scopy = s.copy()
         scopy[25:] = 0.

         # compute the low-rank approximation to the data
         N = u @ np.diag(scopy) @ vt

         # compute the residuals (unexplained variation)
         O = FBSpatial - N

         # compute the l2 norm of the residuals
         Onorm = np.linalg.norm(O,axis=1)

         # find the 30 largest residuals
         anomSet = np.argsort(Onorm)[-30:]

         # plot them
         plt.plot(Onorm)
```

```
plt.plot(anomSet,Onorm[anomSet],'ro')
_ = plt.title('Norm of Residual (rows of O)')
```



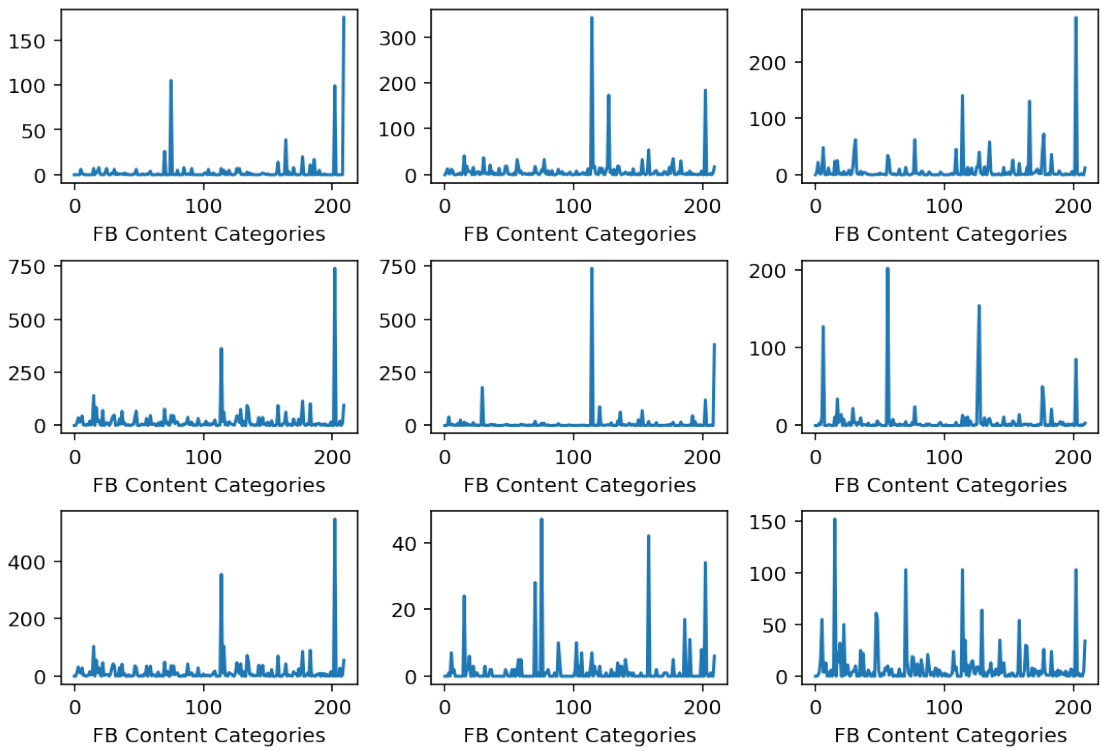
In [22]:
```
# large = np.nonzero(Onorm>100))
# get top 30 anomalies
anomSet = np.argsort(Onorm)[-30:]
plt.plot(FBSnorm)
plt.plot(anomSet,FBSnorm[anomSet],'ro')
_ = plt.title('Top 30 Anomalous Users - Total Number of Likes')
```



Notice that the anomalous users are not necessarily those that have made the most `likes`.

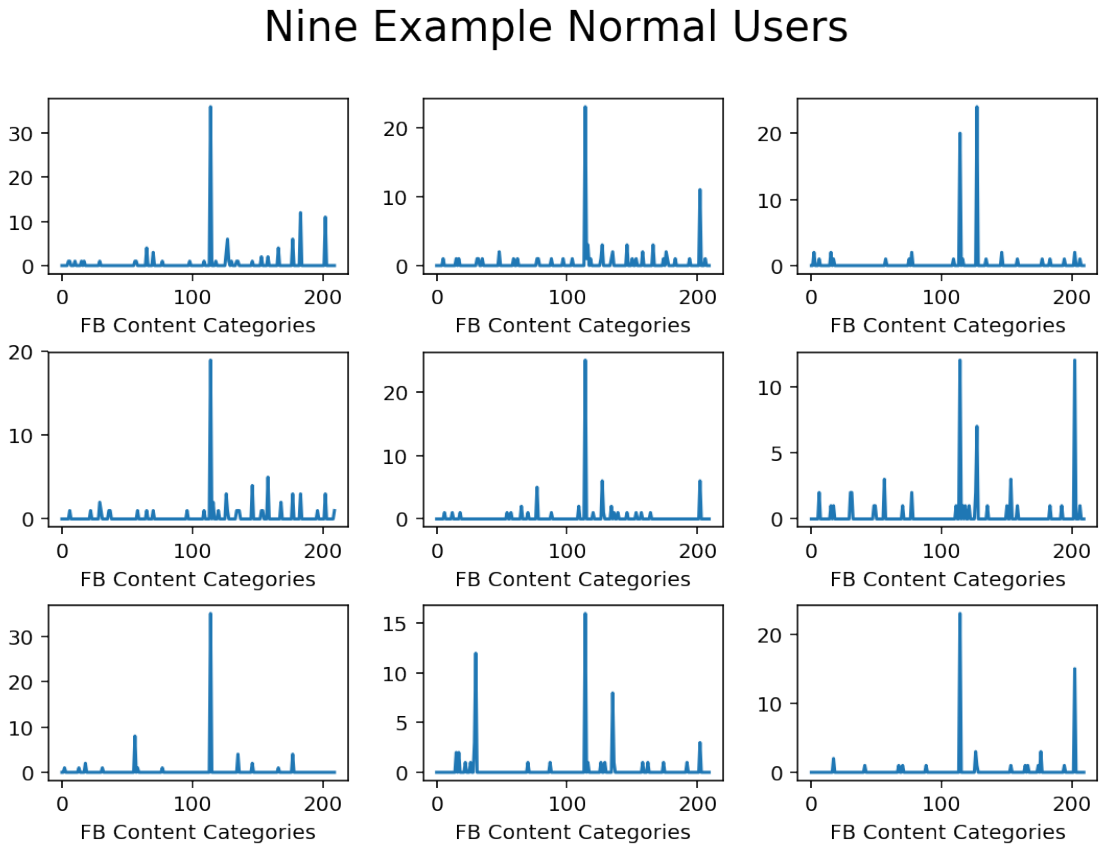
Next we'll pick out nine anomalous users and look at their pattern of likes across the 210 categories.

```
In [23]: plt.figure(figsize=(9,6))
         for i in range(1,10):
             ax = plt.subplot(3,3,i)
             plt.plot(FBSpatial[anomSet[i-1],:])
             plt.xlabel('FB Content Categories')
         plt.subplots_adjust(wspace=0.25,hspace=0.45)
         _ = plt.suptitle('Nine Example Anomalous Users',size=20)
```



And let's do the same for nine normal users.

```
In [24]: # choose non-anomalous users
         set = np.argsort(Onorm)[0:7000]
         # that have high overall volume
         max = np.argsort(FBSnorm[set])[::-1]
         plt.figure(figsize=(9,6))
         for i in range(1,10):
             ax = plt.subplot(3,3,i)
             plt.plot(FBSpatial[set[max[i-1]],:])
             plt.xlabel('FB Content Categories')
         plt.subplots_adjust(wspace=0.25,hspace=0.45)
         _ = plt.suptitle('Nine Example Normal Users',size=20)
```



1.5.2 Example 2: Facebook Temporal LIkes

This data consists of the number of 'Likes' for each of 9000 users, over 6 months, on a daily basis
Rows are users, Columns are days.

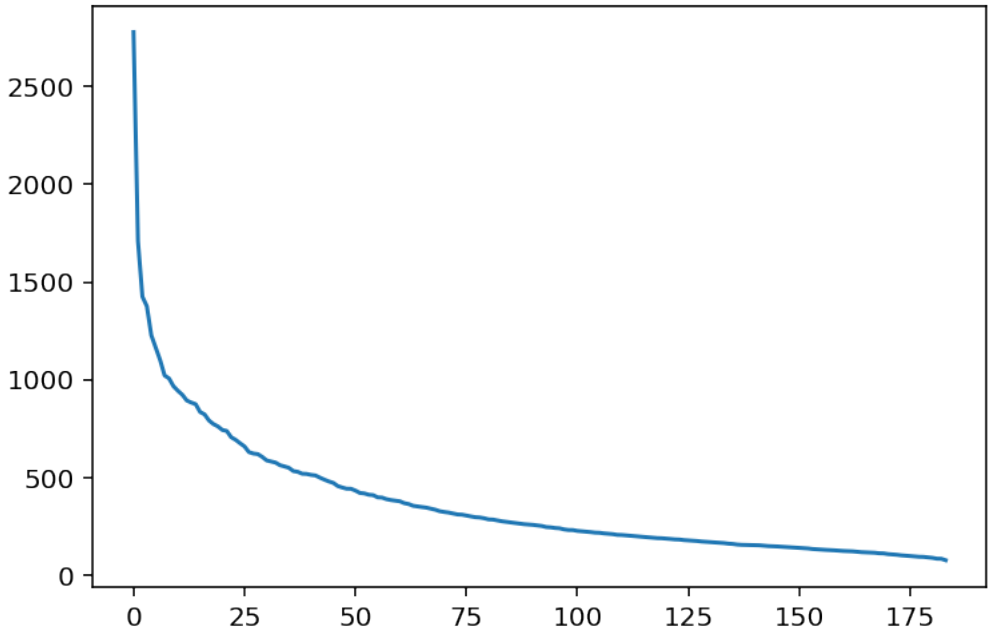
$$\text{users} \left\{ \overbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{a_1} & \mathbf{a_2} & \dots & \mathbf{a_n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}}^{\text{days}} = \overbrace{\begin{bmatrix} \vdots & \vdots \\ \vdots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

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

First we'll look at the singular values.

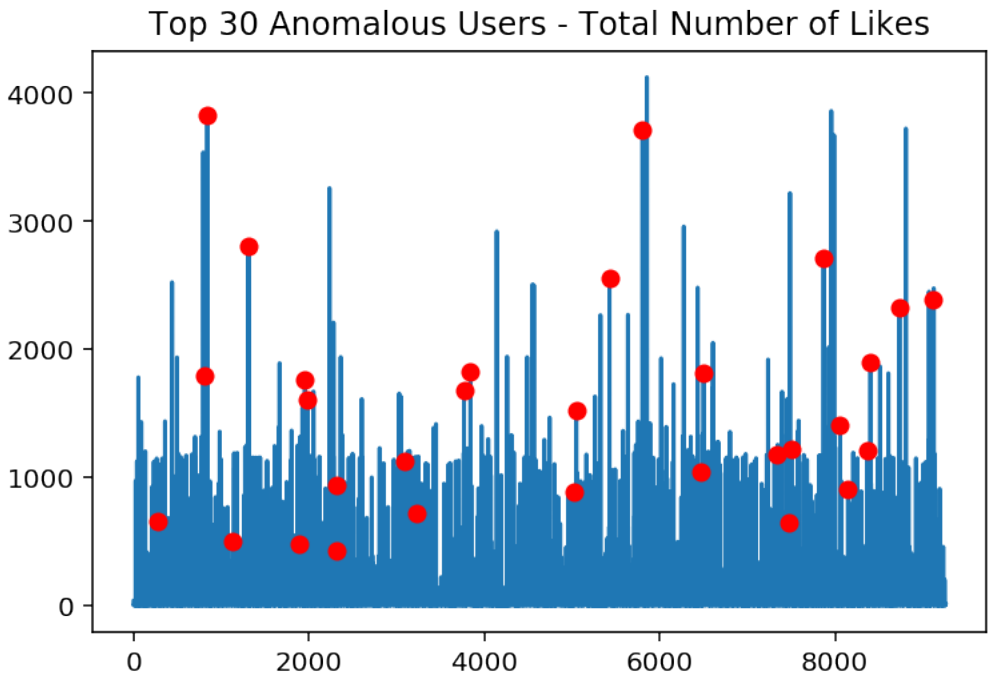
```
In [25]: data = np.loadtxt('data/social/temporal_data.txt')
         FBTemporal = data[:,1:]
```

```
FBTnorm = np.linalg.norm(FBTemporal,axis=1,ord=1)
u,s,vt = np.linalg.svd(FBTemporal,full_matrices=False)
_ = plt.plot(s)
```



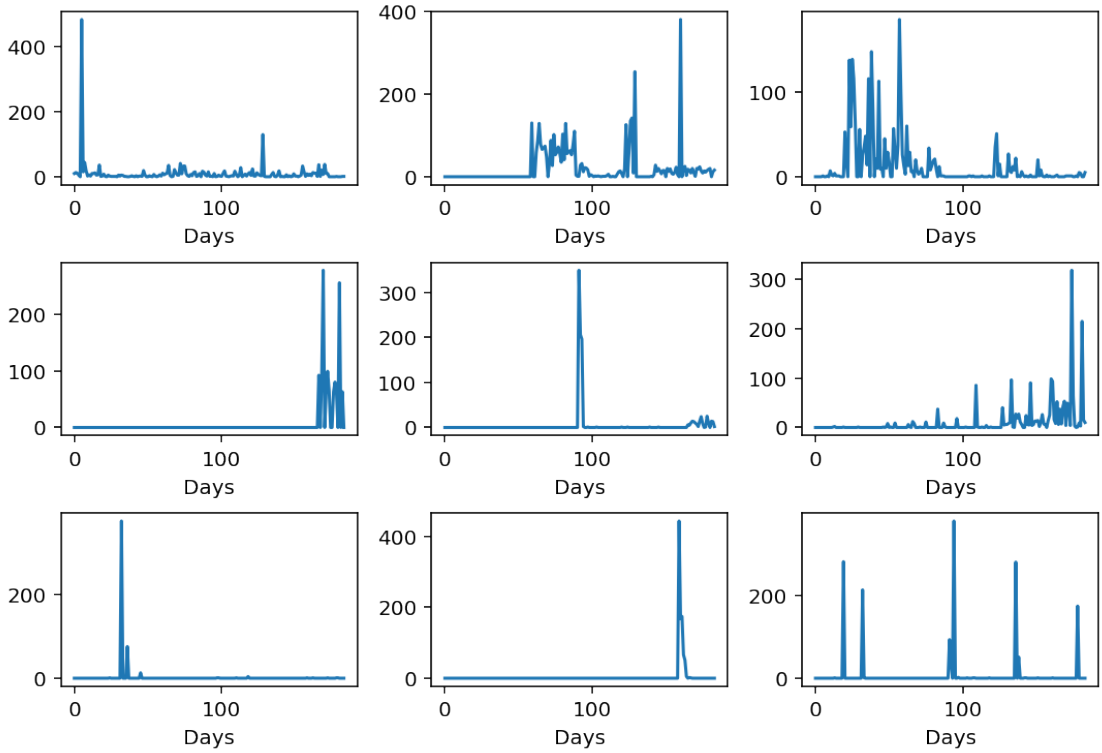
We'll again assume an effective rank of 25.
Next, plot the anomalous users as before.

In [26]:
```
# choose the top 25 columns of U for the normal space
unorm = u[:,0:24]
P = unorm.dot(unorm.T)
N = P.dot(FBTemporal)
O = FBTemporal - N
Onorm = np.linalg.norm(O,axis=1)
# get top 30 anomalies
anomSet = np.argsort(Onorm)[-30:]
plt.plot(FBTnorm)
plt.plot(anomSet,FBTnorm[anomSet],'ro')
_ = plt.title('Top 30 Anomalous Users - Total Number of Likes')
```

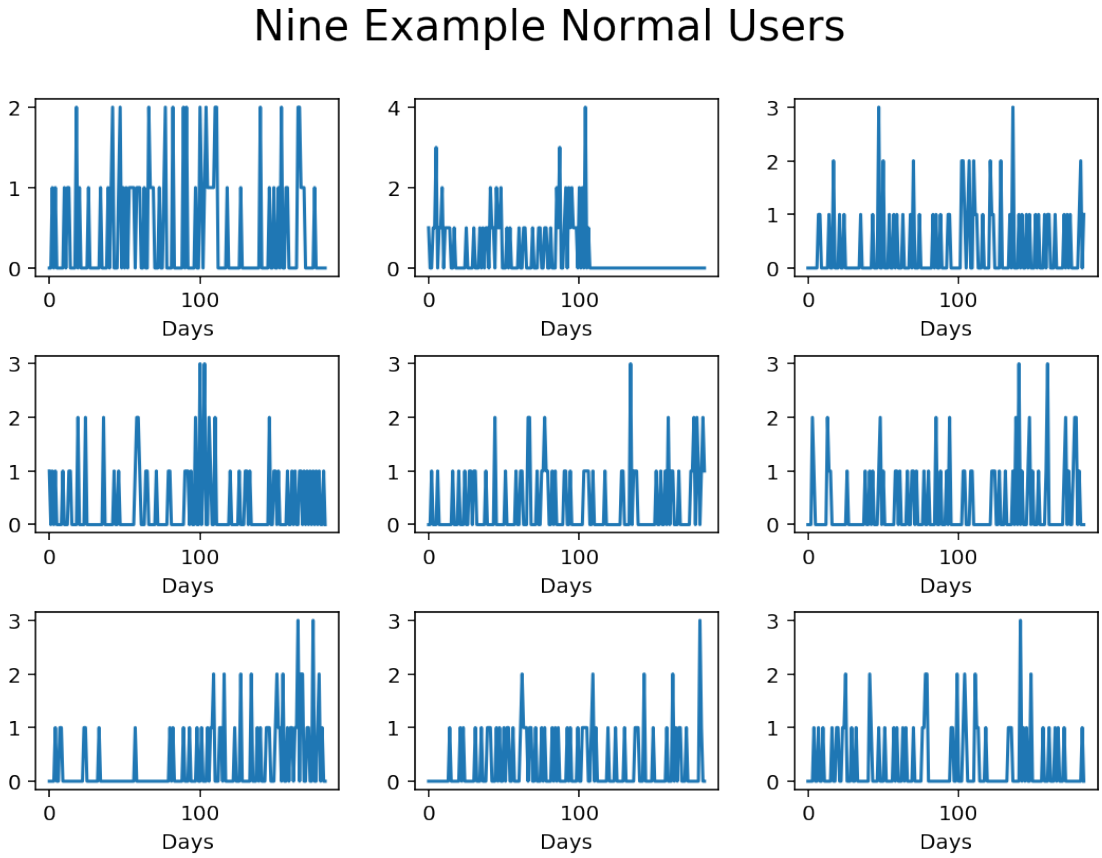


Now let's look at sample anomalous and normal users.

```
In [27]: plt.figure(figsize=(9,6))
         for i in range(1,10):
             ax = plt.subplot(3,3,i)
             plt.plot(FBTemporal[anomSet[i-1],:])
             plt.xlabel('Days')
         plt.subplots_adjust(wspace=0.25,hspace=0.45)
         _ = plt.suptitle('Nine Example Anomalous Users',size=20)
```




```
In [28]: # choose non-anomalous users
         set = np.argsort(Onorm)[0:7000]
         # that have high overall volume
         max = np.argsort(FBTnorm[set])[::-1]
         plt.figure(figsize=(9,6))
         for i in range(1,10):
             ax = plt.subplot(3,3,i)
             plt.plot(FBTemporal[set[max[i-1]],:])
             plt.xlabel('Days')
         plt.subplots_adjust(wspace=0.25,hspace=0.45)
         _ = plt.suptitle('Nine Example Normal Users',size=20)
```



Interestingly, what makes a user anomalous seems to have reversed from the case of the spatial data.