Distances and Neighbors in High Dimensions

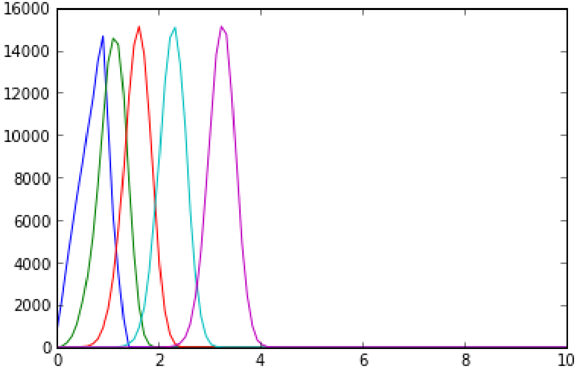
Thomas Breuel

UniKL

Lengths and Distances in High Dimensions

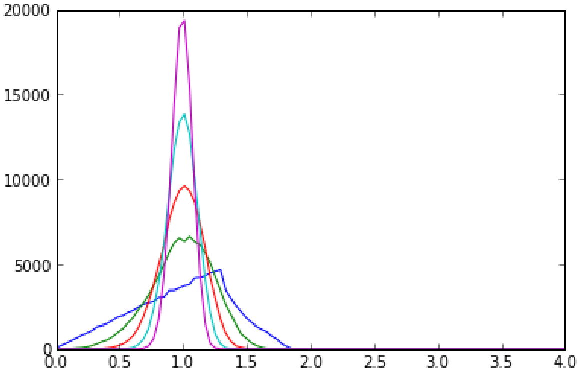
Distribution of Vector Lengths

```
for d in [2,4,8,16,32]:
    vs = rand(100000,d)
    ds = sum(vs**2,axis=1)**.5
    plot(linspace(0,10,100),histogram(ds,100,range=(0,10))[0])
```



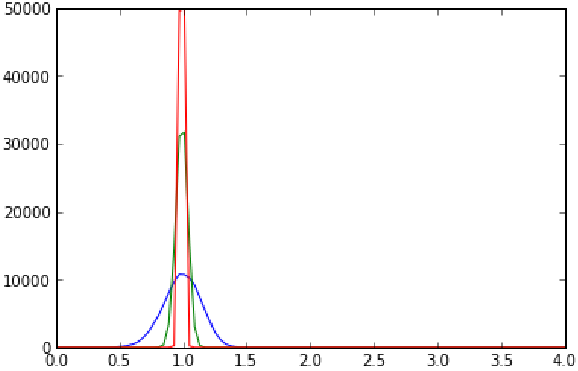
Distribution of Relative Vector Lengths

```
for d in [2,4,8,16,32]:
    vs = rand(100000,d)
    ds = sum(vs**2,axis=1)**.5
    ds /= mean(ds)
    plot(linspace(0,4,100),histogram(ds,100,range=(0,4))[0])
```



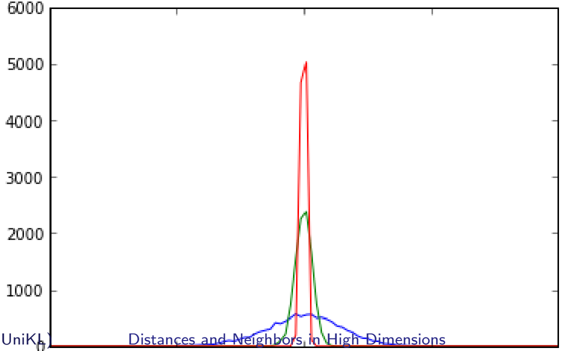
Distribution of Relative Vector Lengths (large dims)

```
for d in [10,100,1000]:
    vs = rand(100000,d)
    ds = sum(vs**2,axis=1)**.5
    ds /= mean(ds)
    plot(linspace(0,4,100),histogram(ds,100,range=(0,4))[0])
```



Pairwise Distances

```
from scipy.spatial.distance import cdist
for d in [10,100,1000]:
    vs = rand(10000,d)
    dists = cdist(vs,vs)
    for i in range(len(dists)): dists[i,i] = inf
    md = amin(dists,axis=1)
    md /= mean(md)
    plot(linspace(0,2,100),histogram(md,100,range=(0,2))[0])
```



ϵ-Approximate Nearest Neighbor

Nearest Neighbor Classification and Approximate Nearest Neighbor Algorithms

- asymptotic error bounds for k -nearest neighbor assume random sampling
- approximate nearest neighbor algorithms do not return random samples
- a priori, this means that asymptotic error bounds do not apply

Instrinsic Dimension

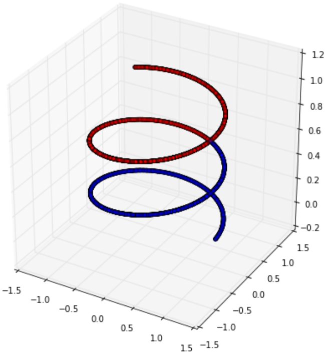
Intrinsic Dimension

- just because a measurement is represented in d dimensions doesn't mean that d measurements are needed to describe it
- fewer variables may be sufficient

```
l = rand(10000)
c = (l<0.5)*1
v = c_[cos(15*l),sin(15*l),l]
```

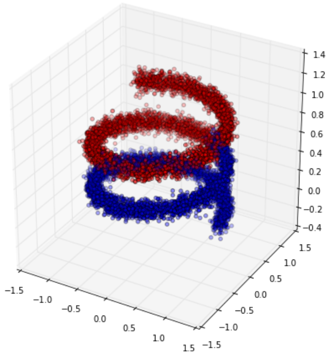
```python
from mpl_toolkits.mplot3d import Axes3D
gcf().add_subplot(111, projection='3d')
gca().scatter(v[:,0], v[:,1],v[:,2],c=array(["r","b"])[c],marker='o',alpha=0.5)
```

```
<mpl_toolkits.mplot3d.art3d.Patch3DCollection at 0x3b16ed0>
```

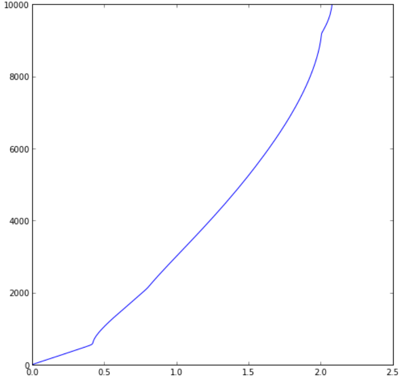


```
v2 = v+0.05*randn(*v.shape)
gcf().add_subplot(111, projection='3d')
gca().scatter(v2[:,0],v2[:,1],v2[:,2],c=array(["r","b"])[c],marker=
    'o',alpha=0.5)
```

<mpl_toolkits.mplot3d.art3d.Patch3DCollection at 0x4064f90>

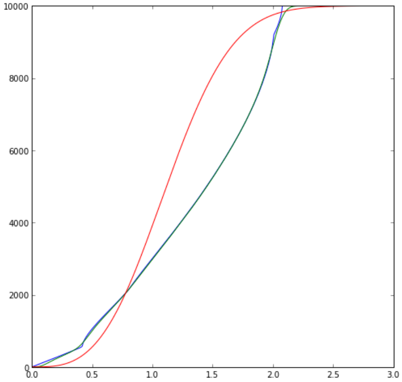


```
vdists = mean(asort(cdist(v,v),axis=1),axis=0)
n = len(vdists)
plot(vdists,arange(n))
```



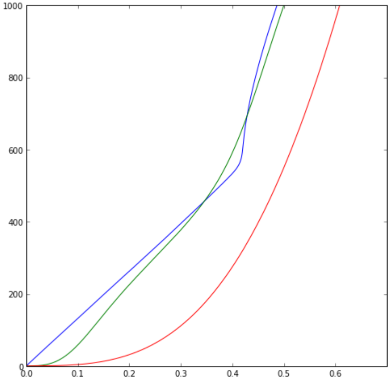
```
v2dists = mean(asort(cdist(v2,v2),axis=1),axis=0)
v3 = randn(*v.shape)/2
v3dists = mean(asort(cdist(v3,v3),axis=1),axis=0)
```

```
plot(vdists,arange(n))
plot(v2dists,arange(n))
plot(v3dists,arange(n))
```



At small scales, we see a linear growth in the number of neighbors for both the noise-free dataset and the noisy intrinsically 1D dataset.

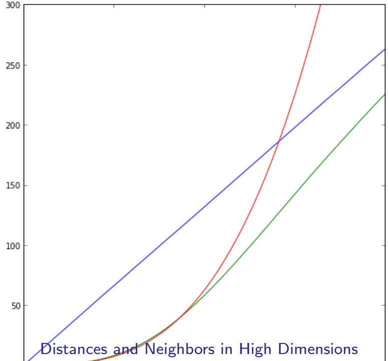
```
xlim((0,0.7)); ylim((0,1000))
plot(vdists,arange(n))
plot(v2dists,arange(n))
plot(v3dists,arange(n))
```



```
v3z = randn(*v.shape)*0.2
v3zdists = mean(asort(cdist(v3z,v3z),axis=1),axis=0)
```

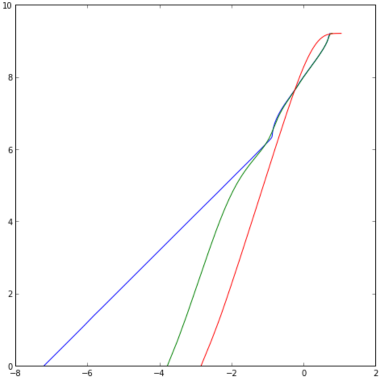
Zooming into the origin, we see that *for small distances*, the error-free intrinsic 1D dataset has a linear growth of the number of neighbors, whereas the dataset disturbed with Gaussian noise grows cubically, just like the intrinsically 3D dataset.

```
xlim((0,0.2)); ylim((0,300))
plot(vdists,arange(n))
plot(v2dists,arange(n))
plot(v3zdists,arange(n))
```



On a log-log plot, we can read off the intrinsic dimensionality of the data at different scales.

```
plot(log(vdists[1:]),log(arange(1,n)))
plot(log(v2dists[1:]),log(arange(1,n)))
plot(log(v3dists[1:]),log(arange(1,n)))
```



Covering dimension

There is a closely related measure of the dimension of a dataset, namely the covering dimension.

We ask: how many ϵ -balls around randomly picked samples does it take to cover the dataset, and how does this number grow with ϵ ?

Determining intrinsic dimensionality

- determined the growth the average distances of the k -th nearest neighbor
- determine, for each, ϵ the number of samples within range ϵ of each sample
- determine the number of ϵ balls needed to cover the data (minimum or random centers from the dataset)

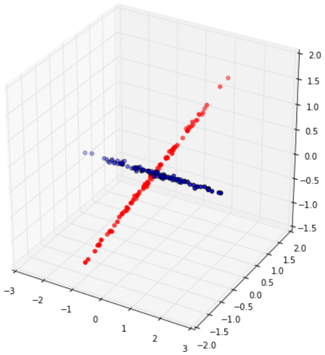
Linear Dimensionality Reduction

```
vs = c_[randn(100,1),0.02*randn(100,1),0.02*randn(100,1)]
```

```
M = randrot(3)
vsr = dot(vs,M.T)
```

```
gcf().add_subplot(111, projection='3d')
gca().scatter(vs[:,0],vs[:,1],vs[:,2])
gca().scatter(vsr[:,0],vsr[:,1],vsr[:,2],color='r')
```

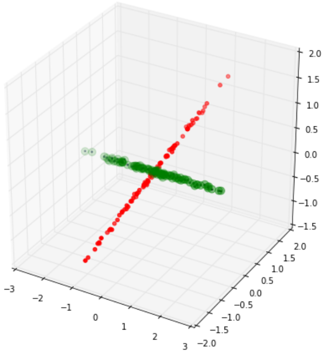
<mpl_toolkits.mplot3d.art3d.Patch3DCollection at 0xda34b50>



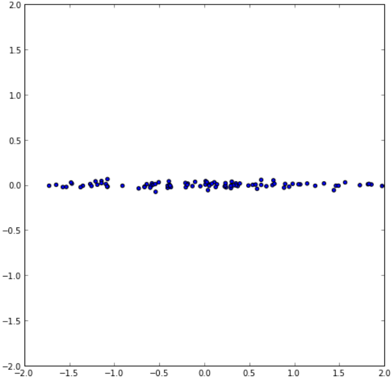
```
e,Md = eig(dot(vsr.T,vsr))
```

```
vsm = dot(vsr,Md)
gcf().add_subplot(111, projection='3d')
gca().scatter(vs[:,0],vs[:,1],vs[:,2],s=3)
gca().scatter(vsr[:,0],vsr[:,1],vsr[:,2],color='r')
gca().scatter(vsm[:,0],vsm[:,1],vsm[:,2],color='g',alpha=0.5,s=80)
```

<mpl_toolkits.mplot3d.art3d.Patch3DCollection at 0xe9b6e90>



```
from sklearn.decomposition import PCA
pca = PCA(2)
vp = pca.fit_transform(vsr)
xlim((-2,2)); ylim((-2,2))
scatter(vp[:,0],vp[:,1])
```



Nonlinear Dimensionality Reduction

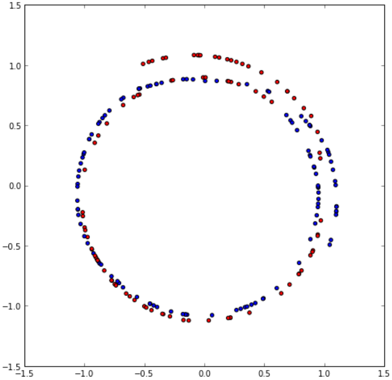
Metric Multidimensional Scaling

Given a set of vectors v_i , find another set of corresponding vectors u_i such that the following error is minimized:

$$E = \sum_{ij} ||d(v_i, v_j) - d(u_i, u_j)||^2$$

Multidimensional Scaling

```
from sklearn import manifold
mds = manifold.MDS(2)
vl = mds.fit_transform(v[::50])
scatter(vl[:,0],vl[:,1],c=array(["r","b"])[c[::50]])
```

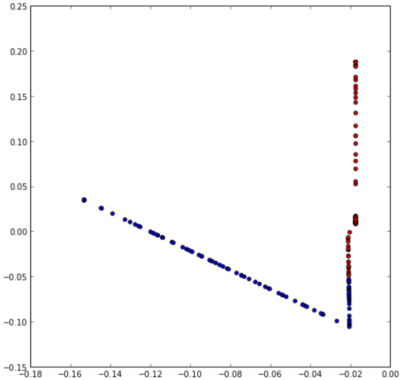


Locally Linear Embedding

- find nearest neighbors for each point
- represent each point as a linear combination of its neighbors
- find a low-dimensional embedding such that each point is still given by approximately the same linear combination of its neighbors

Locally Linear Embedding

```
mds = manifold.LocallyLinearEmbedding()
vl = mds.fit_transform(v[::50])
scatter(vl[:,0],vl[:,1],c=array(["r","b"])[c[::50]])
```

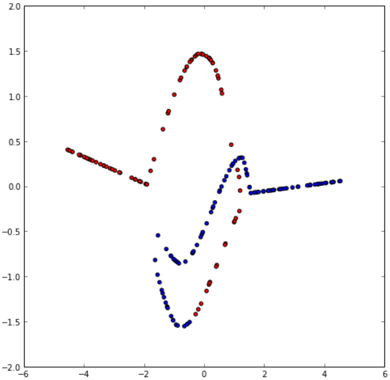


Isomap

- compute k -nearest neighbor graph
- take pair-wise distances between nearby points
- use graph algorithm to find distances to faraway points
- use classic MDS to compute a low dimensional representation

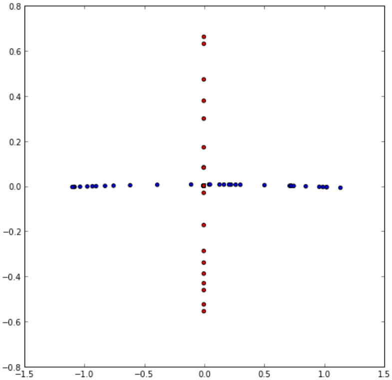
Isomap

```
mds = manifold.Isomap()
vl = mds.fit_transform(v[::50])
scatter(vl[:,0],vl[:,1],c=array(["r","b"])[c[::50]])
```



Isomap

```
mds = manifold.Isomap(n_neighbors=2)
vl = mds.fit_transform(v[::50])
scatter(vl[:,0],vl[:,1],c=array(["r","b"])[c[::50]])
```



Nearest Neighbor Methods and Low-Dimensional Structures

Nearest neighbor methods generally depend only on “the” intrinsic dimension of data, not the dimension of the space that the data is embedded in.

“The” intrinsic dimension depends on scale:

- at a small scale, there is usually high dimensional noise
- at an intermediate scale, data often has low intrinsic dimension
- at a large scale, the intrinsic dimension gets high again