radius=a:

$$V_1 = \frac{S \times a^d}{d}$$

radius=$a - \epsilon$:

$$V_2 = \frac{S \times (a - \epsilon)^d}{d} (0 < \epsilon < a)$$

$$\frac{V_2}{V_1} = (\frac{a - \epsilon}{a})^d = (1 - \frac{\epsilon}{a})^d$$

The fraction of the volume:

$$V_f = 1 - \frac{V_2}{V_1} = 1 - (1 - \frac{\epsilon}{a})^d$$

$$\lim_{d \to +\infty} V_f = \lim_{d \to +\infty} 1 - (1 - \frac{\epsilon}{a})^d = 1 - \lim_{d \to +\infty} (1 - \frac{\epsilon}{a})^d$$

$$\because 0 < \epsilon < a$$

$$\therefore 0 < \frac{\epsilon}{a} < 1$$

$$\therefore 0 < 1 - \frac{\epsilon}{a} < 1$$

$$\therefore \lim_{d \to +\infty} (1 - \frac{\epsilon}{a}) = 0$$

$$\therefore 1 - \lim_{d \to +\infty} (1 - \frac{\epsilon}{a}) = 1$$

Thus, for any fixed ϵ, no matter how small, this fraction tends to 1 as $d \to +\infty$.

In [73]:

```
import csv
import numpy as np
import scipy
from copy import deepcopy
import pandas as pd
import random
import math
from sklearn import preprocessing,metrics
from sklearn.decomposition import NMF
```

Import data

In [29]:

```
data=csv.reader(open('/Users/wendy/Documents/2017 Fall/CS 534/HW5/Colleges.txt'))
data_list=[]
for row in data:
    data_list.append(row)
features=data_list[0][0].split('\t')
data_list.pop(0)
print features
```

```
['college name', 'apps received', 'apps accepted', 'new stud enrolled'
, '% new stud from top 10%', '% new stud from top 25%', 'num FT underg
rad', 'num PT undergrad', 'in-state tuition', 'out-of-state tuition',
'room', 'board', 'add fees', 'est book costs', 'est personal costs', '
% fac with PHD', 'stud:fac ratio', 'graduation rate']
```

In [30]:

```
######not important, ignore this cell######
whole_data=[]
university_name=[]
for i in range(len(data_list)):
    whole_data.append(data_list[i][0].split('\t'))
    for j in range(1,len(whole_data[i])):
        if whole_data[i][j]!="":
            whole_data[i][j]=float(whole_data[i][j])
    university_name.append(whole_data[i][0])
    whole_data[i][0]=i
```

use mean to replace missing data

In [86]:

```
dic={}
for i in range(1,len(whole_data[1])):
    s=0
    counts=0
    for j in range(len(whole_data)):
        if whole_data[j][i]!='':
            s+=whole_data[j][i]
            counts+=1
            dic[i]=s/counts
print dic

for i in range(1,len(whole_data[1])):
    for j in range(len(whole_data)):
        if whole_data[j][i]=='':
            whole_data[j][i]=dic[i]

whole_data=np.array(whole_data)
whole_data_scaled = preprocessing.normalize(whole_data[:,1:18])
```

```
{1: 2752.0975232198148, 2: 1870.6831913245549, 3: 778.88049344641468,
4: 25.671977507029148, 5: 52.349999999999895, 6: 3692.6651270207858, 7
: 1081.5267716535427, 8: 7897.2743710691793, 9: 9276.9056162246452, 10
: 2514.6819571865472, 11: 2060.9838308457724, 12: 392.01264591439633,
13: 549.97288676236019, 14: 1389.291703835858, 15: 68.645669291338578,
16: 14.858769230769228, 17: 60.405315614617876}
```

Find the major components,total variance and square_err for PCA

In [89]:

```
# from sklearn.decomposition import PCA

def pca_variance(data):
    for i in range(1,len(data[1])):
        pca = PCA(n_components=i)
        p=pca.fit_transform(data)
        s=sum(pca.explained_variance_ratio_)
        ori_pca=pca.inverse_transform(p)
        square_error=metrics.mean_squared_error(data,ori_pca)
        if s>0.95:
            break
    print "the major components are :",pca.explained_variance_ratio_
    print "total variance is:",s
    #print "singular values are:",sing
    print "square error for pca is", square_error
```

Compare normalized data and unnormalized data

In [84]:

```
pca_variance(whole_data_scaled)
pca_variance(whole_data[:,1:18])
```

```
the major components are : [ 0.7431399   0.09230213  0.05211807  0.034
28852  0.02876712]
total variance is: 0.950615726993
square error for pca is 0.000546298296426
the major components are : [ 0.56627776  0.3470008   0.03455217  0.016
87796]
total variance is: 0.964708692189
square error for pca is 183722.737975
```

We should normalize the data, otherwise the square err would be extremely high.

Compare NMF and PCA

In [90]:

```
from sklearn.decomposition import NMF

def nmf_vari(data):
    model = NMF(n_components=3, init='random', random_state=0)
    nmf=model.fit_transform(data)
    ori_nmf=model.inverse_transform(nmf)
    square_error=metrics.mean_squared_error(data,ori_nmf)
    print model.components_
    return square_error
err=nmf_vari(whole_data_scaled)
print "the square_value for NMF is :" ,err
```

```
[[  1.67714157e+00   1.11567737e+00   4.47547919e-01   3.35132177e-03
    4.59292704e-03   2.26217718e+00   2.93642713e-01   2.72600887e-02
    3.17739562e-01   0.00000000e+00   0.00000000e+00   5.53743607e-02
    6.15110221e-03   0.00000000e+00   4.66871602e-03   5.40358491e-04
    2.23646912e-03]
 [  9.79698105e-02   5.84164627e-02   9.39180613e-03   3.19142949e-03
    6.39028130e-03   0.00000000e+00   0.00000000e+00   1.51689196e+00
    1.40225401e+00   3.07530145e-01   2.57060537e-01   3.55000002e-02
    6.41198766e-02   1.38461604e-01   7.48537535e-03   1.45626858e-03
    8.25034186e-03]
 [  0.00000000e+00   3.27229681e-02   5.28341883e-02   1.85422994e-03
    5.57425552e-03   2.46000858e-01   2.49168906e-01   0.00000000e+00
    5.44942479e-01   3.51137881e-01   2.87817891e-01   5.66606064e-02
    8.70405189e-02   2.72599371e-01   9.50272260e-03   2.89953958e-03
    6.47218688e-03]]
the square_value for NMF is : 0.00170683946637
```

The square err of NMF is 0.00170683946637, higher than the square err of PCA. The contribute of each feature to the 3 components is shown on the matrix.

PCA has a better performance than NMF

In [ ]:

```

```

Simulate data

In [97]:

```
import random
import numpy as np
rangeX = (-50, 50)
rangeY = (-2500, 2500)
#X=[]
#Y=[]
positive=0
negative=0
sample=[]
target=[]
for i in range(1000):
    x = random.randrange(*rangeX)
    #X.append(x)
    y = random.randrange(*rangeY)
    #Y.append(y)
    if y>=x**2:
        sample.append((x,y))
        target.append(1)
        positive+=1
    else:
        sample.append((x,y))
        target.append(0)
        negative+=1
#sample_array=np.array(sample)
#print sample,target,positive,negative
```

Split train and test

In [76]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(sample, target, test_size=0.3, r
```

Use cross validation to find the optimal parameter