

# *LINEAR CLASSIFIER AND SUPPORT VECTOR MACHINE*

CS 662 Project 3

*Xiaobo Zhang*

## Section 1 Introduction

The goal of this project is experimenting different classifiers: such as linear classifier, support vector machine (SVM) with different kernels, maximum likelihood estimation (MLE) and Parzen window estimation, and find out when is good, when is not good, and when is one over another for classification. Section 2 and Section 3 will introduce the concept of the linear classifier and support vector machine. Section 4 will experiment these classifiers with different size of training sample. Section 5 will experiment these classifiers with the different mean distance between two class. Section 6 will experiment these classifiers with linear separable data and non linear separable data distribution under different variance of data. Section 7 will make the conclusion for the entire project.

## Section 2 Linear classifier

The idea of linear classifier is to find the best projection  $\pi: R^n \rightarrow R^{\bar{n}}$  where  $\bar{n} < n$ , such that  $\pi(x_1), \pi(x_2), \pi(x_3) \dots \pi(x_4)$  are easy to separate. Assume there is 2 class, the function for finding best projection on vector  $v$  is:

$$\text{find } v, \text{ such that } J(v) = \frac{|m_1(v) - m_2(v)|^2}{s_1(v)^2 + s_2(v)^2} \text{ is maximum}$$

$$\text{where } m_i(v) = \frac{1}{N_i} \sum_{i=1}^{N_i} v * x_i \text{ and}$$

$$s_i(v)^2 = \sum_{i=1}^{N_i} (v * x_i - v * m_i)^2 \text{ and } N_i \text{ is \# of class } i$$

Then replace formula  $J(v)$  by  $J(v) = \frac{v^T S_B v}{v^T S_W v}$ , because:

$$|m_1(v) - m_2(v)|^2 = v^T (m_1 - m_2)(m_1 - m_2)^T v$$

$$S_i(v)^2 = \sum_{x \text{ in class } i} (v * x_i - m_i(v))^2 = v^T \sum_{x \text{ in class } i} (x - m_i)(x - m_i)^T v$$

Let  $S_B = (m_1 - m_2)(m_1 - m_2)^T$ ,  $S_i w = \sum_{x \text{ in class } i} (x - m_i)(x - m_i)^T$  and  $S_w = S_1 w + S_2 w$  for 2 class.

Now, the problem changes to eigenvalue problem: find  $v_0$  such that  $S_b v_0 = \lambda S_w v_0$ .

$$S_b v_0 = (m_1 - m_2)(m_1 - m_2)^T * v_0 = (m_1 - m_2) * c = \lambda S_w v_0$$

$$S^{-1} w (m_1 - m_2) * c = \lambda * v_0$$

$$v_0 = v_{max} = S^{-1} w (m_1 - m_2)$$

Using  $v_{max}$  to classify new data, the discriminate function is:

$$v_{max} * (x - m) \begin{cases} > 0 \text{ for class 1} \\ < 0 \text{ for class 2} \end{cases}$$

where  $m$  is center of training sample and  $x$  is single test sample

### Section 3 Support vector machine

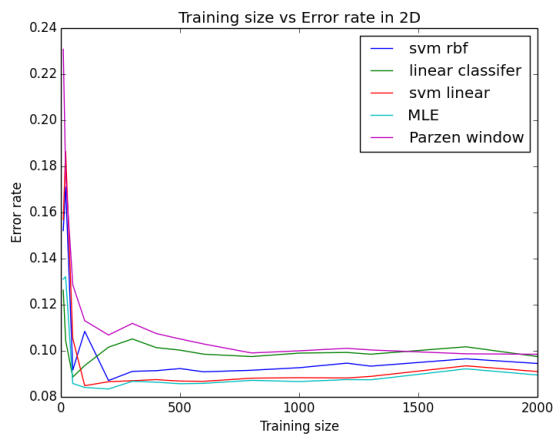
Given labeled trained samples  $X_1, X_2, X_3, \dots, X_n$ , let  $Z_i = (1, X_i)$  if  $X_i$  is class 1 and  $Z_i = (-1, X_i)$  if  $X_i$  is class 2. The support vector can be found by let  $v \in R^{n+1}$  with  $|v| = 1$ , such that for all training samples,  $v * Z_i \geq b$ , with  $b > 0$  and as large as possible. The claim is also equivalent to find  $v \in R^{n+1}$  with minimum  $|v|$  such that  $v * Z_i = 1$  for all support vector  $Z_i$ , and  $v * Z_i > 1$  for all other training samples.

### Section 4 Size of training samples

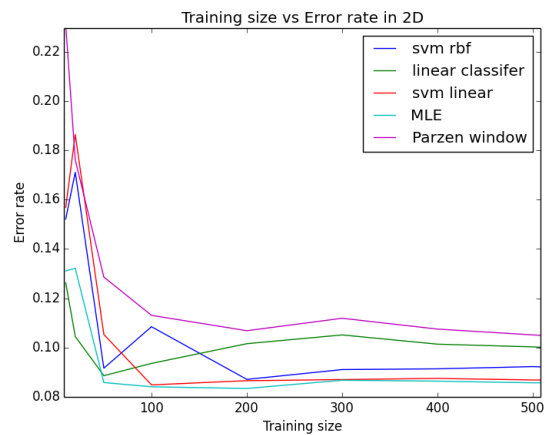
In this section, the experiment tests the performance of each classifier with the different size of training sample. The test case will include the data with the

different dimension in 2D, 3D, 5D, 10D,15D and 25D, and the different size of training sample from 10 to 2000. The experiment controls several parameters:

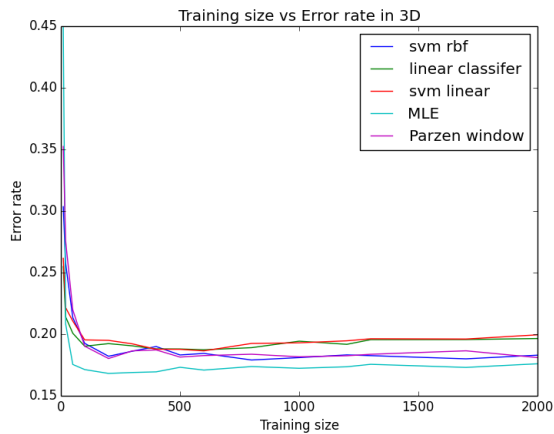
- Mean distance between two class is 3
- $P(\omega_1) = p(\omega_2) = 0.5$
- $\Sigma_1 \neq \Sigma_2$ , variance of data on each dimension(diagonal) is less than 5
- The size of entire data is 4000
- The data is synthetically generated with Gaussian distribution
- The size of Parzen window is  $h = 3$



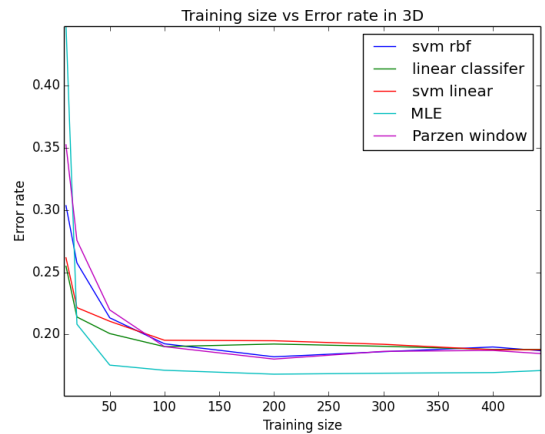
2D size range from 0 to 2000



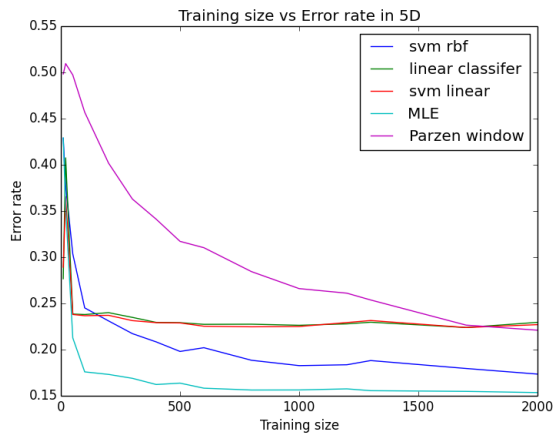
2D size range from 0 to 500



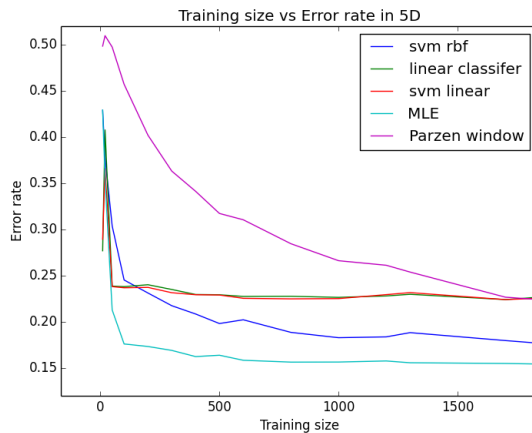
3D size range from 0 to 2000



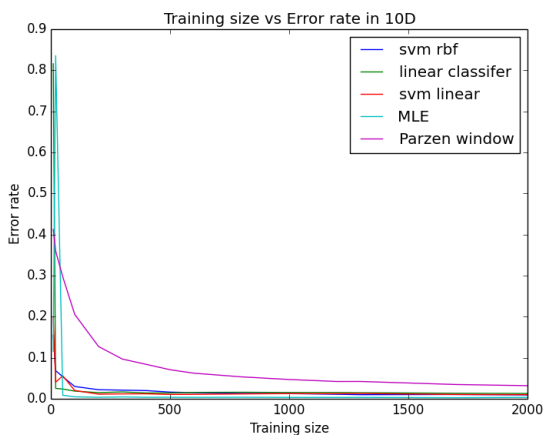
3D size range from 0 to 500



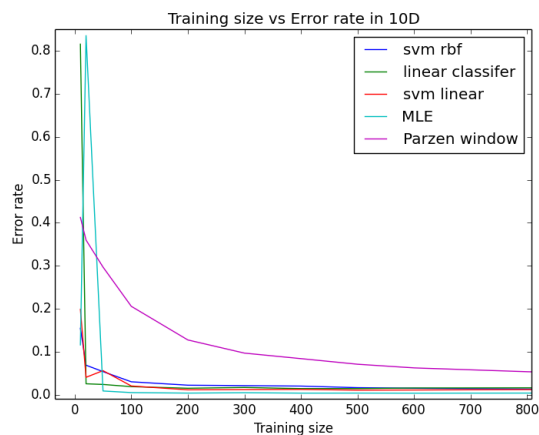
5D size range from 0 to 2000



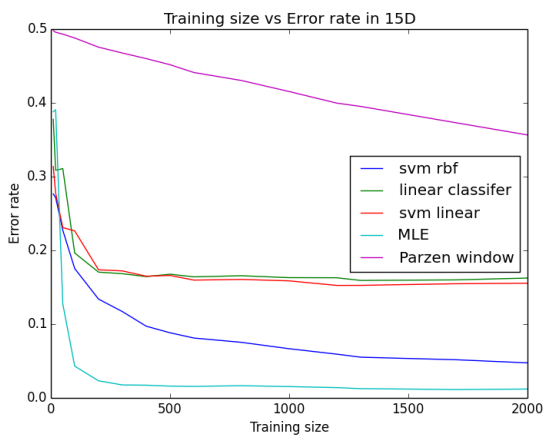
5D size range from 0 to 500



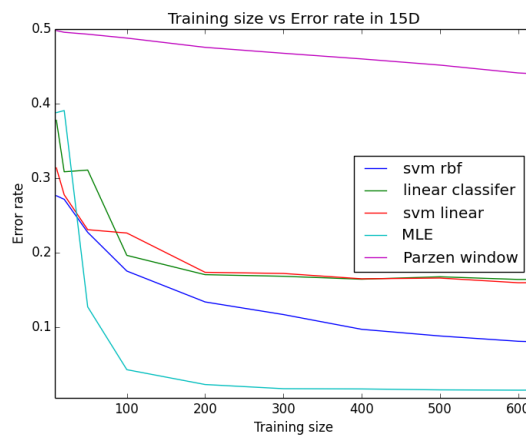
10D size range from 0 to 2000



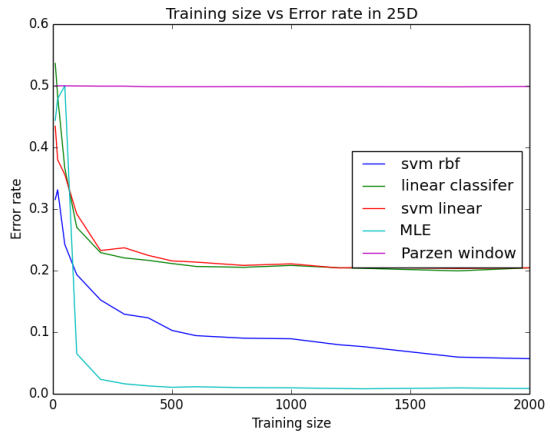
10D size range from 0 to 500



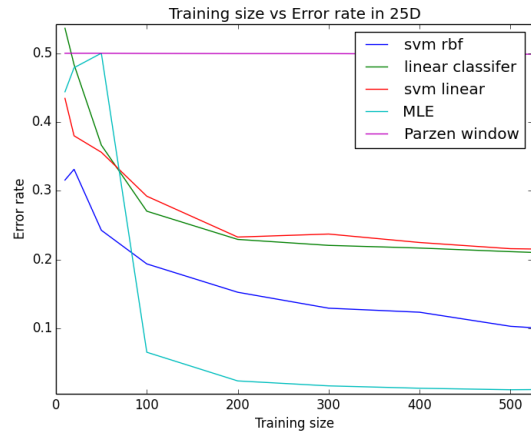
15D size range from 0 to 2000



15D size range from 0 to 500



25D size range from 0 to 2000



25D size range from 0 to 500

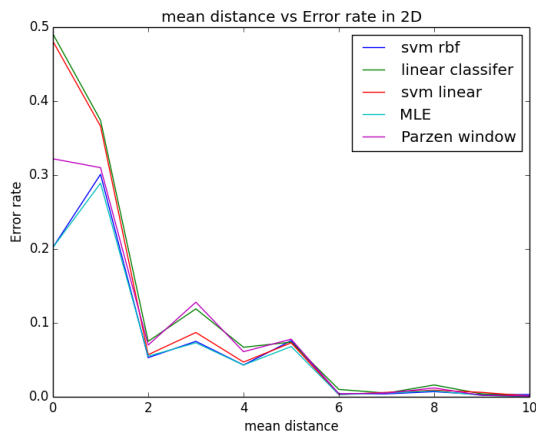
According to the graph above, it tells us that all classifiers perform not good when the size of the training sample is really small (less than 100), but the error rate will decrease rapidly and finally stop at some point with the size of training sample increasing. It probably because all the classifier need large number of training samples to make the trained model as complete as possible. The error rate of linear classifier and the SVM with linear kernel is almost same, and the SVM with Gaussian kernel performs better than linear classifier and the SVM with linear kernel when the dimension of data is high. It probably because the distribution of data is the Gaussian distribution which fits the Gaussian kernel. The MLE method performs better than any other classifiers in the entire of this experiment. When the dimension increase, the error rate of Parzen window goes up, it probably because the size of the window is too small to hold enough data and to make a good prediction.

## Section 5 Mean distance

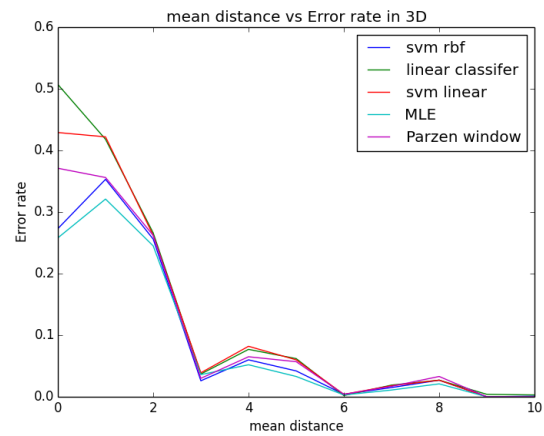
In this section, the experiment tests the performance of each classifier under different mean distance between two class. The test case will include the data has

different dimension in 2D, 3D, 5D, 10D, 15D and 25D, and different distance from 0 to 10. The experiment controls several parameters:

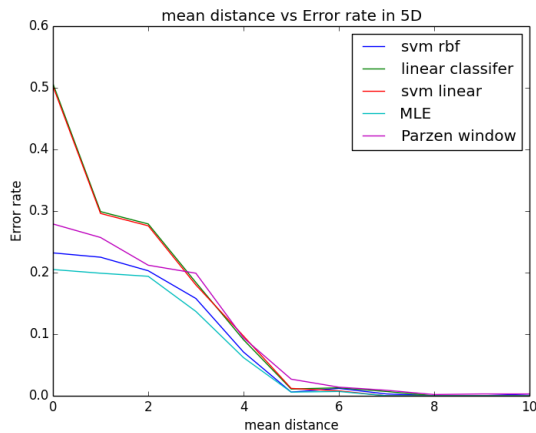
- The size of entire data is 2000, and the size of training data is 1000
- $P(\omega_1) = p(\omega_2) = 0.5$
- $\Sigma_1 \neq \Sigma_2$ , variance of data on each dimension(diagonal) is less than 5
- The data is synthetically generated with Gaussian distribution
- The size of Parzen window is  $h = 3$



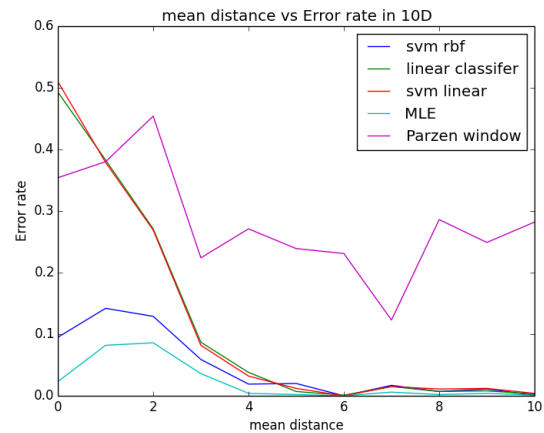
2D



3D



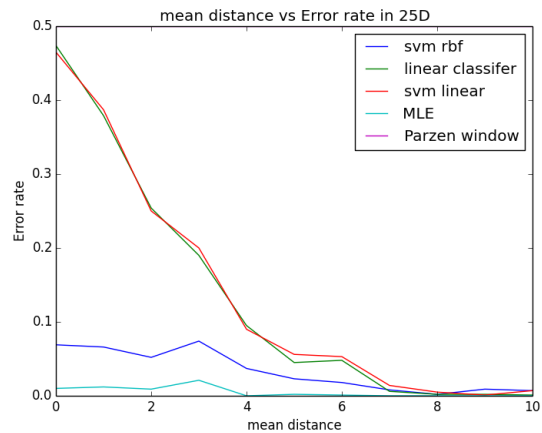
5D



10D



15D



25D

According to the graph, we can find that the error rate goes down with the mean distance between two class increasing no matter what dimension is. It probably because the larger the mean distance between two class, the more obvious for the classifiers to distinct each class of data. The SVM with Gaussian kernel has lower error rate than linear classifier and the SVM with linear kernel. The reason is when the mean distance between two class, the linear classifier is not able to find a clear boundary to distinct two class, but the Gaussian kernel can convert the data to the higher dimension to make the classifier work. MLE performs best among all classifiers in every dimension. The Parzen window estimation doesn't perform well in high dimension, the potential reason is the window for estimation too small to hold enough points in the high dimension. But if the window size increase, the prediction will be better than before.

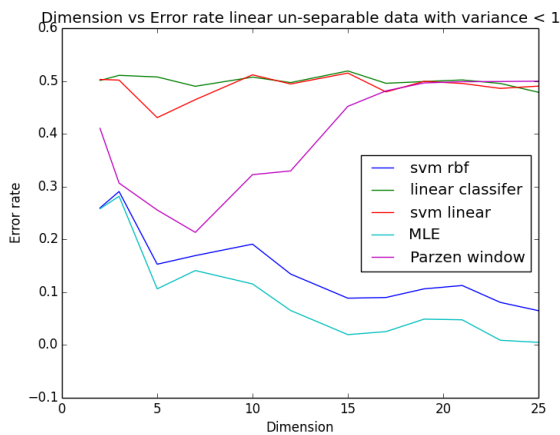
## Section 6 Linear separable data and non linear separable data

This section tests the performance of different classifiers with linear separable data and non linear separable data under different variance of data (diagonal value of covariance). The experiment controls several parameters:

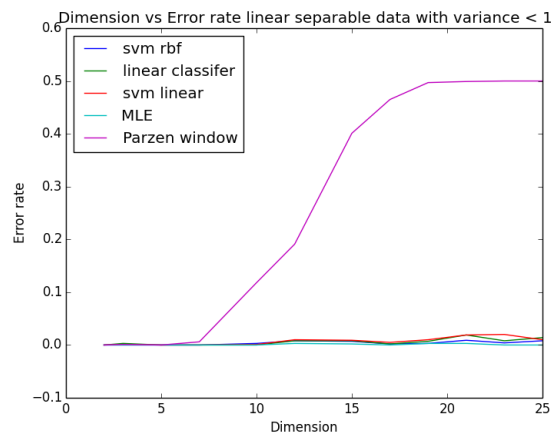


- The size of entire data is 2000, and the size of training data is 1000
- For linear separable data, the mean distance between two class is 6
- For non linear separable data, the mean distance between two class is 0
- $P(\omega_1) = p(\omega_2) = 0.5$
- $\Sigma_1 \neq \Sigma_2$
- The data is synthetically generated with Gaussian distribution
- The size of Parzen window is  $h = 3$

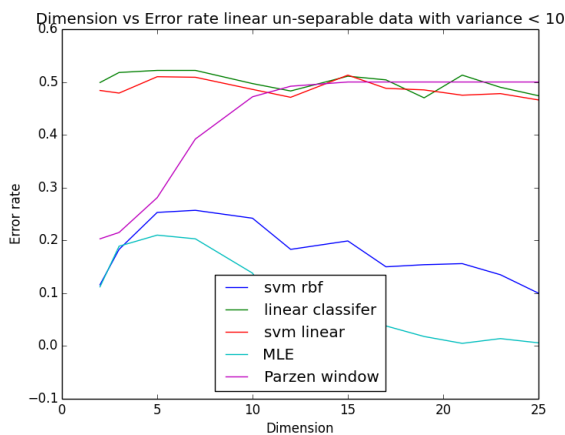
The experiment will test linear separable and non linear separable data with different variance: less than 1, less than 10, less than 100, and less than 1000.



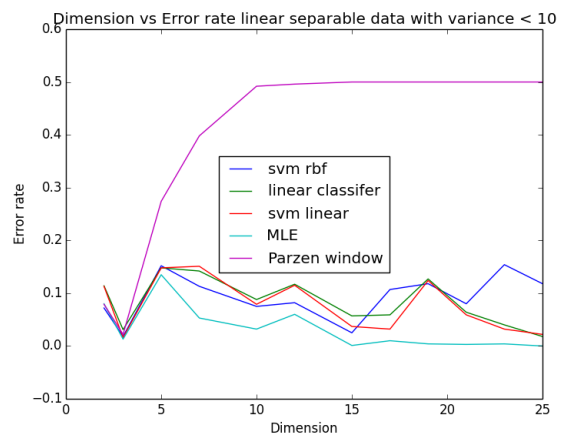
non linear separable data, variance < 1



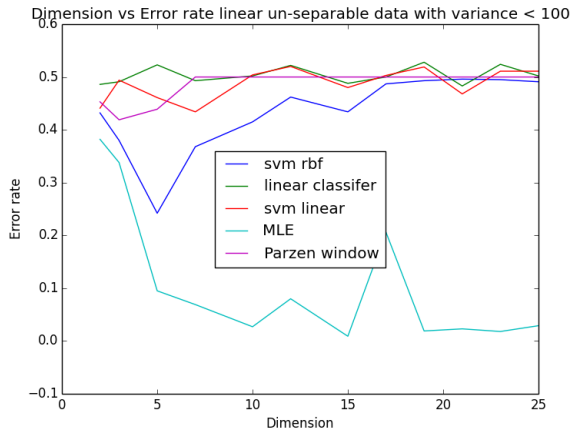
separable data, variance < 1



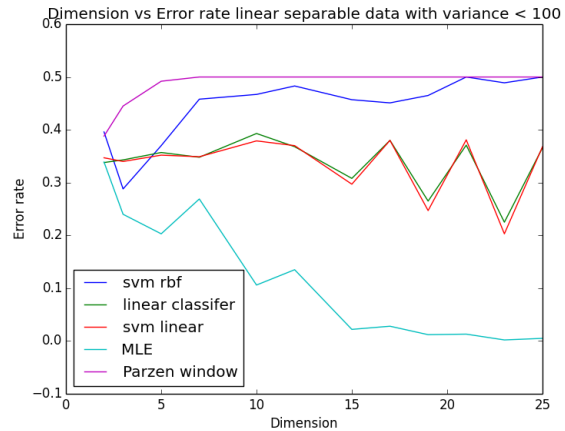
non linear separable data, variance < 10



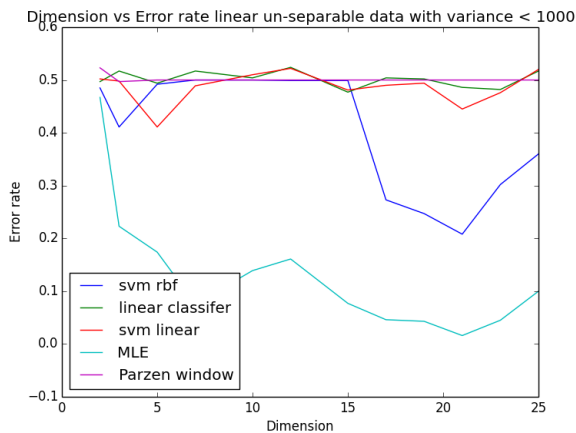
separable data, variance < 10



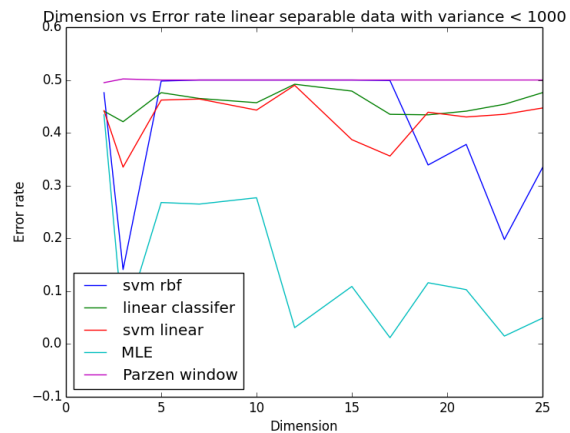
non linear separable data, variance < 100



separable data, variance < 100



non linear separable data, variance < 1000



separable data, variance < 1000

For the SVM with linear and linear classifier, the performance of those two classifiers always stays around 0.5 if the data is not linearly separable. Also, if the data is linearly separable and the variance is very large, the error rate still stays around 0.5. Those two models work well only if the data is linear separable and variance is very small, otherwise the classifier just random guesses the class label. For the SVM with Gaussian kernel, it works well with the small variance of data in any dimension for both linear separable and non linear separable data. But given high variance data, the error rate of classifier is very high in any dimension for both linearly separable and non linear separable data.

For MLE, the error rate of the model goes down with the dimension increase for both linearly separable and non linear separable data. No matter how complex the data is, MLE always has lower error rate than any other models.

For Parzen window estimation, the error rate is small in low dimension and low variance, but the error rate goes up to 0.5 when both the dimension and variance of data increase. The size for window function will influence the error rate in high dimension.

## **Section 7 Conclusion**

After the entire experiment, we can conclude that if the complexity of data is not extreme, the larger size of training samples and the larger mean distance between two class will make good prediction. But under some extreme case, the different classifier model has different performance. MLE performs best among all classifiers in most of the cases, but it needs number of training samples to create a good classifier. Linear classifier and the SVM with linear kernel performs well if the data is linear separable, and not good if the data is not linear separable.

Parzen window estimation performs not good for high dimension data, but good for low dimension data. The SVM with Gaussian kernel is good for small variance data, but not good for large variance data. Generally speaking, the classifier model performs better with linear separable data than non linear separable data, and better with small variance data than large variance data.

linear.py

```
1. import numpy as np
2. import math
3.
4.
5. def separate_data(train_data, class_label):
6.     train_1 = []
7.     train_2 = []
8.
9.     for data, label in zip(train_data, class_label):
10.         if label == 0:
11.             train_1.append(np.array(data).T)
12.         else:
13.             train_2.append(np.array(data).T)
14.     train_1 = np.matrix(train_1)
15.     train_2 = np.matrix(train_2)
16.     return train_1, train_2
17.
18.
19. def compute_var(data, mean):
20.     val = 0
21.     for d in data:
22.         #print d, mean
23.         a = d - mean
24.
25.         #print a.T * a
26.         val += np.dot(a.T, a)
27.     return val
28.
29. def train(train_data, class_label):
30.     train_1, train_2 = separate_data(train_data, class_label)
31.     mean_1 = np.mean(train_1, axis=0)
32.     mean_2 = np.mean(train_2, axis=0)
33.     mean = np.mean(train_data, axis=0)
34.
35.
```

```

36.         s1_w = compute_var(train_1,mean_1)
37.         #print s1_w
38.         s2_w = compute_var(train_2,mean_2)
39.         #print s2_w
40.         sw = s1_w + s2_w
41.         w = sw.I*np.matrix(mean_1-mean_2).I
42.         #print np.linalg.norm(w,0)
43.         return w,mean
44.
45.     def test(test_data,w,mean):
46.         res = []
47.         #print w
48.         #print mean
49.         for d in test_data:
50.
51.             v = (np.matrix(d-mean)*w)[0,0]
52.             #print v
53.             if v > 0:
54.                 res.append(0)
55.             else:
56.                 res.append(1)
57.         return res

```

svm.py

```

1. from sklearn import svm
2.
3. def train(train_data,train_label,k):
4.     X = train_data
5.     y = train_label
6.     clf = svm.SVC(kernel=k)
7.     clf.fit(X, y)
8.     return clf
9.
10.    def test(test_data,clf):
11.        res = clf.predict(test_data)
12.        return res

```

mle.py

```

1. #mle py
2. #function of mle

```

```

3.
4. import numpy as np
5.
6. def estimate_mu(data):
7.     data = np.array(data)
8.     return np.mean(data,axis=0)
9.
10.    def estimate_covariance(data):
11.        data = np.array(data)
12.        return np.cov(data.T)

```

parzen.py

```

1. #parzen.py
2. #function of parzen window
3. import math
4. import numpy as np
5.
6. def window_function(sample,test_data,h):
7.     #print sample
8.     #print "x",test_data
9.
10.    for x,x_i in zip(sample,test_data):
11.
12.        if abs(x-x_i) > float(h)/2:
13.            return 0
14.    return 1
15.
16.    def estimate_single_px(samples,test_data,h):
17.        inRegion = 0
18.        outRegion = 0
19.
20.        for sample in samples:
21.            if window_function(sample,test_data,h) == 1
22.                :
23.                    inRegion += 1
24.            else:
25.                outRegion += 1

```



```

24.             outRegion += 1
25.
26.         return float(inRegion)/((inRegion+outRegion)*(h
            **len(test_data)))
27.
28.
29.     def get_px(samples,test_datas,h):
30.         px =[]
31.
32.         for x in test_datas:
33.             x =np.array(x)
34.
35.             val = estimate_single_px(samples,x,h)
36.             px.append(val)
37.         return px

```

testing\_mle.py

```

1. import mle
2. import numpy as np
3. import random
4. import parzen
5. from numpy import matrix
6. from mpl_toolkits.mplot3d import Axes3D
7. import matplotlib.pyplot as plt
8. from matplotlib.mlab import bivariate_normal
9. import math
10.    from numpy.linalg import linalg
11.    import time
12.
13.
14.    def compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,x):
15.
16.        mu_1 =matrix(mu_1)
17.        mu_2 = matrix(mu_2)
18.        cov_1 = matrix(cov_1)
19.        cov_2 = matrix(cov_2)
20.
21.

```



```

22.         const = 0.5 * math.log(linalg.det(cov_2)/linalg
    .det(cov_1)) + math.log(float(w1)/w2)
23.
24.         #print "const",const
25.         #print "xx",(0.5* (x-mu_2) * cov_2.I * (x-
    mu_2).T ) - (0.5* (x-mu_1) * cov_1.I * (x-mu_1).T )
26.         gx = (0.5 * (x-mu_2) * cov_2.I * (x-
    mu_2).T ) - (0.5* (x-mu_1) * cov_1.I * (x-mu_1).T )
27.         #print gx
28.         v = gx.tolist()
29.         return v[0][0]+const
30.
31.
32.
33.     def testing(train_samples_1,test_samples_1,train_sa
    mples_2,test_samples_2):
34.
35.         now = time.time()
36.
37.
38.         estimate_mu_1 = mle.estimate_mu(train_samples_1
    )
39.         estimate_cov_1 = mle.estimate_covariance(train_
    samples_1)
40.
41.         estimate_mu_2 = mle.estimate_mu(train_samples_2
    )
42.         estimate_cov_2 = mle.estimate_covariance(train_
    samples_2)
43.
44.         w1 = len(train_samples_1)/float(len(train_sampl
    es_1)+len(train_samples_2))
45.         w2 = len(train_samples_2)/float(len(train_sampl
    es_1)+len(train_samples_2))
46.
47.         error = 0
48.         for d in test_samples_1:
49.             val = compute_gx(estimate_mu_1 ,estimate_mu
    _2,estimate_cov_1,estimate_cov_2,w1,w2,d)

```

```

50.         if val > 0:
51.             pass
52.         else:
53.             error += 1
54.
55.         for d in test_samples_2:
56.             val = compute_gx(estimate_mu_1 ,estimate_mu
57.             _2,estimate_cov_1,estimate_cov_2,w1,w2,d)
58.             if val > 0:
59.                 error += 1
60.             else:
61.                 pass
62.         res = float(error)/float(len(test_samples_1)+l
63.         en(test_samples_2))
64.         return res
65.         #errors.append(res)
66.         #times.append(time.time()-now)
67.
68.         #print errors
69.         #print "error rate:",sum(errors)/len(errors)
70.         #print "average_time:",sum(times)/len(times)
71.         #return sum(errors)/len(errors)
72.         #return sum(times)/len(times)

```

test.py

```

1. import svm
2. import linear
3. import numpy as np
4. import random
5. import matplotlib.pyplot as plt
6. import testing_mle
7. import testing_parzen
8.
9. def generate_gaussian_data(mean,cov,size,label):
10.     x = np.random.multivariate_normal(mean, cov,siz
11.     e)

```

```

11.         y = np.repeat(label, size)
12.         return x,y
13.
14.     def error_rate(predict_label, true_label):
15.         error = 0
16.         accurate = 0
17.         for l1,l2 in zip(predict_label, true_label):
18.             if l1 == l2:
19.                 accurate += 1
20.             else:
21.                 error += 1
22.         return float(error)/float(accurate+error)
23.         #return error
24.
25.     def separate_data(prior,datas,labels):
26.         train_data = []
27.         train_label = []
28.         test_data = []
29.         test_label = []
30.
31.         for data,label in zip(datas,labels):
32.             if random.random() < prior:
33.                 train_data.append(data)
34.                 train_label.append(label)
35.             else:
36.                 test_data.append(data)
37.                 test_label.append(label)
38.         return train_data,train_label,test_data,test_label
39.
40.     def create_covariance_matrix(dimension):
41.         cov = np.zeros((dimension, dimension))
42.         for i in range(dimension):
43.             for j in range(i,dimension,1):
44.                 if i == j:
45.                     cov[i][j] = random.random()*5
46.                 else:
47.                     val = 2*random.random()-1
48.                     cov[i][j] = val

```

```

49.             cov[j][i] = val
50.         return cov
51.
52.     def set_up_parameter(dimension,distance):
53.         mean_1 = np.zeros(dimension)
54.         mean_2 = np.zeros(dimension)
55.         for d in range(dimension):
56.             val = random.random()
57.             mean_1[d] = val
58.             mean_2[d] = val
59.
60.         index = int(random.random()*dimension)
61.         mean_2[index] += distance
62.
63.         cov_1 = create_covariance_matrix(dimension)
64.         cov_2 = create_covariance_matrix(dimension)
65.         return mean_1,cov_1,mean_2,cov_2
66.
67.     def generate_unseparateable_data(size,dimension):
68.         mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,distance)
69.
70.
71.
72.         x_1,y_1 = generate_gaussian_data(mean_1,cov_1,2
73.             000,0)
74.         x_2,y_2 = generate_gaussian_data(mean_1,cov_2,2
75.             000,1)
76.
77.         #x_1 = 20*np.random.random((size,dimension))-
78.             10
79.         #x_2 = 20*np.random.random((size,dimension))-
80.             10
81.         #y_1 = np.repeat(0, size)
82.         #y_2 = np.repeat(1, size)
83.         return x_1,y_1,x_2,y_2
84.
85.     .....
86.     testing

```

```

83.     '''
84.
85.     dimension = 50
86.     distance = 3
87.     sizes = [10,20,50,100,200,300,400,500,600,800,1000,
1200,1300,1700,2000]
88.     dimensions = [25]
89.     #dimensions = [2,3]
90.     prior_1 = 0.5
91.     prior_2 = 1 - prior_1
92.
93.
94.     .....
95.     mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimens
ion,distance)
96.     x_1,y_1 = generate_gaussian_data(mean_1,cov_1,10000
,0)
97.     x_2,y_2 = generate_gaussian_data(mean_2,cov_2,10000
,1)
98.     '''
99.     .....
100.    #testing training size vs error rate
101.    for dimension in dimensions:
102.
103.        print dimension,"dimension"
104.        mean_1,cov_1,mean_2,cov_2 = set_up_parameter(di
mension,distance)
105.        x_1,y_1 = generate_gaussian_data(mean_1,cov_1,2
000,0)
106.        x_2,y_2 = generate_gaussian_data(mean_2,cov_2,2
000,1)
107.        svm_rbf_errors = []
108.        svm_linear_errors = []
109.        linear_errors = []
110.        svm_poly_errors = []
111.        mle_errors=[]
112.        parzen_errors=[]
113.        for size in sizes:
114.

```

```

115.         print "size:",size
116.
117.         #separate data
118.         s1 = size * prior_1
119.         s2 = size * prior_2
120.
121.         train_data_1 = x_1[0:s1]
122.         train_label_1 = y_1[0:s1]
123.         train_data_2 = x_2[0:s2]
124.         train_label_2 = y_2[0:s2]
125.
126.         test_data_1 = x_1[s1:]
127.         test_label_1 = y_1[s1:]
128.         test_data_2 = x_2[s2:]
129.         test_label_2 = y_2[s2:]
130.
131.
132.         for a in train_data_1:
133.             plt.scatter(a[0],a[1],color='blue')
134.         for a in train_data_2:
135.             plt.scatter(a[0],a[1],color='red')
136.         plt.show()
137.
138.
139.         #generate traing data
140.         x = np.concatenate((train_data_1,train_data
141.             _2), axis=0)
142.         y = np.concatenate((train_label_1,train_lab
143.             el_2), axis=0)
144.
145.         #traning a model
146.         svm_rbf = svm.train(x,y,"rbf")
147.         svm_linear = svm.train(x,y,"linear")
148.         w,mean = linear.train(x,y)
149.
150.         #generate test data
151.         test_data = np.concatenate((test_data_1,te
152.             st data 2), axis=0)

```

```

151.         test_label = np.concatenate((test_label_1, test_label_2), axis=0)
152.
153.         #prediction
154.         svm_rbf_label = svm.test(test_data,svm_rbf)
155.         linear_label = linear.test(test_data,w,mean
    )
156.         svm_linear_label = svm.test(test_data,svm_linear)
157.
158.         #get result
159.         svm_rbf_error = error_rate(test_label,svm_rbf_label)
160.         linear_error = error_rate(test_label,linear_label)
161.         svm_linear_error = error_rate(test_label,svm_linear_label)
162.         mle_error = testing_mle.testing(train_data_1,test_data_1,train_data_2,test_data_2)
163.         parzen_error = testing_parzen.testing(train_data_1,test_data_1,train_data_2,test_data_2,3)
164.         print "svm error(rbf):",svm_rbf_error
165.         print "svm error(linear):",svm_linear_error
166.         print "linear_classifier error:",linear_error
167.         print "mle error:",mle_error
168.         print "parzen error:",parzen_error
169.         #store results
170.         svm_rbf_errors.append(svm_rbf_error)
171.         linear_errors.append(linear_error)
172.         svm_linear_errors.append(svm_linear_error)
173.         mle_errors.append(mle_error)
174.         parzen_errors.append(parzen_error)
175.         print ""
176.
177.         plt.title("Training size vs Error rate in "+str
    (dimension)+"D")

```

```

178.     plt.xlabel("Training size")
179.     plt.ylabel("Error rate")
180.     plt.plot(sizes,svm_rbf_errors,label="svm rbf")
181.     plt.plot(sizes,linear_errors,label="linear clas
sifer")
182.     plt.plot(sizes,svm_linear_errors,label="svm lin
ear")
183.     plt.plot(sizes,mle_errors,label="MLE")
184.     plt.plot(sizes,parzen_errors,label="Parzen wind
ow")
185.
186.     plt.legend(loc="best")
187.     plt.show()
188.     '''
189.
190.
191.     #testing mean distance vs error rate
192.     distances = [0,1,2,3,4,5,6,7,8,9,10]
193.
194.     for dimension in dimensions:
195.         print dimension,"dimension"
196.
197.         svm_rbf_errors = []
198.         svm_linear_errors = []
199.         linear_errors = []
200.         svm_poly_errors = []
201.         mle_errors=[]
202.         parzen_errors =[]
203.         size = 1000
204.         for distance in distances:
205.             print distance,"distance"
206.             mean_1,cov_1,mean_2,cov_2 = set_up_paramete
r(dimension,distance)
207.             x_1,y_1 = generate_gaussian_data(mean_1,cov
_1,1000,0)
208.             x_2,y_2 = generate_gaussian_data(mean_2,cov
_2,1000,1)
209.
210.             #separate data

```



```

211.         s1 = size * prior_1
212.         s2 = size * prior_2
213.
214.         train_data_1 = x_1[0:s1]
215.         train_label_1 = y_1[0:s1]
216.         train_data_2 = x_2[0:s2]
217.         train_label_2 = y_2[0:s2]
218.
219.         test_data_1 = x_1[s1:]
220.         test_label_1 = y_1[s1:]
221.         test_data_2 = x_2[s2:]
222.         test_label_2 = y_2[s2:]
223.
224.         #generate traing data
225.         x = np.concatenate((train_data_1,train_data
226.         _2), axis=0)
227.         y = np.concatenate((train_label_1,train_lab
228.         el_2), axis=0)
229.
230.         #traning a model
231.         svm_rbf = svm.train(x,y,"rbf")
232.         svm_linear = svm.train(x,y,"linear")
233.         w,mean = linear.train(x,y)
234.
235.         #generate test data
236.         test_data = np.concatenate((test_data_1,te
237.         st_data_2), axis=0)
238.         test_label = np.concatenate((test_label_1,t
239.         est_label_2), axis=0)
240.
241.         #prediction
242.         svm_rbf_label = svm.test(test_data,svm_rbf)
243.
244.         linear_label = linear.test(test_data,w,mean
245.         )
246.         svm_linear_label = svm.test(test_data,svm_l
247.         inear)
248.

```

```

243.         #get result
244.         svm_rbf_error = error_rate(test_label,svm_r
bf_label)
245.         linear_error = error_rate(test_label,linear
_label)
246.         svm_linear_error = error_rate(test_label,sv
m_linear_label)
247.         mle_error = testing_mle.testing(train_data_
1,test_data_1,train_data_2,test_data_2)
248.         parzen_error = testing_parzen.testing(train
_data_1,test_data_1,train_data_2,test_data_2,3)
249.
250.         print "svm error(rbf):",svm_rbf_error
251.         print "svm error(linear):",svm_linear_error
252.         print "linaer_classifer error:",linear_erro
r
253.         print "mle error:",mle_error
254.         print "parzen error:",parzen_error
255.         #store results
256.         svm_rbf_errors.append(svm_rbf_error)
257.         linear_errors.append(linear_error)
258.         svm_linear_errors.append(svm_linear_error)
259.         mle_errors.append(mle_error)
260.         parzen_errors.append(parzen_error)
261.         print ""
262.
263.         plt.title("mean distance vs Error rate in "+str
(dimension)+"D")
264.         plt.xlabel("mean distance")
265.         plt.ylabel("Error rate")
266.         plt.ylim(-0.1,0.6)
267.         plt.plot(distances,svm_rbf_errors,label="svm rb
f")
268.         plt.plot(distances,linear_errors,label="linear
classifer")
269.         plt.plot(distances,svm_linear_errors,label="svm
linear")

```

```

270.     plt.plot(distances,mle_errors,label="MLE")
271.     plt.plot(distances,parzen_errors,label="Parzen
    window")
272.     plt.legend(loc="best")
273.     plt.show()
274.
275.
276.
277.     size = 1000
278.     svm_rbf_errors = []
279.     svm_linear_errors = []
280.     linear_errors = []
281.     svm_poly_errors = []
282.     mle_errors=[]
283.     parzen_errors =[]
284.
285.     .....
286.     mean_1,cov_1,mean_2,cov_2 = set_up_parameter(25,0)
287.
288.     for dimension in dimensions:
289.         print dimension,"dimension"
290.
291.
292.         #size = 1000
293.         mean_1,cov_1,mean_2,cov_2 = set_up_parameter(di
    mension,6)
294.         sigma_1 = cov_1[0:dimension,0:dimension]
295.         sigma_2 = cov_2[0:dimension,0:dimension]
296.
297.         x_1,y_1 = generate_gaussian_data(mean_1,sigma_1
    ,1000,0)
298.         x_2,y_2 = generate_gaussian_data(mean_2,sigma_2
    ,1000,1)
299.
300.
301.         #separate data
302.         s1 = size * prior_1
303.         s2 = size * prior_2
304.

```

```

305.     train_data_1 = x_1[0:s1]
306.     train_label_1 = y_1[0:s1]
307.     train_data_2 = x_2[0:s2]
308.     train_label_2 = y_2[0:s2]
309.
310.     test_data_1 = x_1[s1:]
311.     test_label_1 = y_1[s1:]
312.     test_data_2 = x_2[s2:]
313.     test_label_2 = y_2[s2:]
314.
315.     #generate traing data
316.     x = np.concatenate((train_data_1,train_data_2),
        axis=0)
317.     y = np.concatenate((train_label_1,train_label_2
        ), axis=0)
318.
319.
320.     #traning a model
321.     svm_rbf = svm.train(x,y,"rbf")
322.     svm_linear = svm.train(x,y,"linear")
323.     w,mean = linear.train(x,y)
324.
325.     #generate test data
326.     test_data = np.concatenate((test_data_1,test_d
        ata_2), axis=0)
327.     test_label = np.concatenate((test_label_1,test_
        label_2), axis=0)
328.
329.     #prediction
330.     svm_rbf_label = svm.test(test_data,svm_rbf)
331.     linear_label = linear.test(test_data,w,mean)
332.     svm_linear_label = svm.test(test_data,svm_linea
        r)
333.
334.     #get result
335.     svm_rbf_error = error_rate(test_label,svm_rbf_l
        abel)
336.     print "svm error(rbf):",svm_rbf_error
337.

```

```

338.         linear_error = error_rate(test_label,linear_label)
339.         print "linear classifier error:",linear_error
340.
341.
342.         svm_linear_error = error_rate(test_label,svm_linear_label)
343.         print "svm error(linear):",svm_linear_error
344.
345.         mle_error = testing_mle.testing(train_data_1,test_data_1,train_data_2,test_data_2)
346.         print "mle error:",mle_error
347.
348.         parzen_error = testing_parzen.testing(train_data_1,test_data_1,train_data_2,test_data_2,3)
349.         print "parzen error:",parzen_error
350.
351.         #store results
352.         svm_rbf_errors.append(svm_rbf_error)
353.         linear_errors.append(linear_error)
354.         svm_linear_errors.append(svm_linear_error)
355.         mle_errors.append(mle_error)
356.         parzen_errors.append(parzen_error)
357.         print ""
358.
359.         plt.title("Dimension vs Error rate linear separable data with variance < 10000")
360.         plt.xlabel("Dimension")
361.         plt.ylabel("Error rate")
362.         plt.ylim(-0.1,0.6)
363.         plt.plot(dimensions,svm_rbf_errors,label="svm rbf")
364.         plt.plot(dimensions,linear_errors,label="linear classifier")
365.         plt.plot(dimensions,svm_linear_errors,label="svm linear")
366.         plt.plot(dimensions,mle_errors,label="MLE")
367.         plt.plot(dimensions,parzen_errors,label="Parzen window")

```

```
368. plt.legend(loc="best")
369. plt.show()
370. '''
371.
372.     . . . .
373.     x_1,y_1,x_2,y_2 = generate_unseparateable_data(size
        ,dimension)
374.     for a in x_1:
375.         plt.scatter(a[0],a[1],color='blue')
376.     for a in x_2:
377.         plt.scatter(a[0],a[1],color='red')
378. plt.show()
379.     '''
```