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

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

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

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

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

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

$$s_i(v)^2 = \sum_{x\ in\ class\ i} \left(v * x_i - m_i(v)\right)^2 = v^T \sum\nolimits_{x\ in\ class\ i} (x - m_i)(x - m_i)^T\ v$$

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

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

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

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

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

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

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

$$where\ m\ is\ center\ of\ training\ sample\ and\ x\ is\ signle\ test\ sample$$

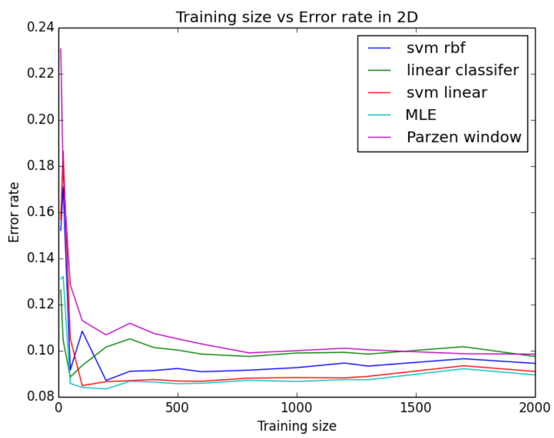
Section 3 Support vector machine

Given labeled trained samples $X_1, X_2, X_3, \ldots . X_n$, 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.

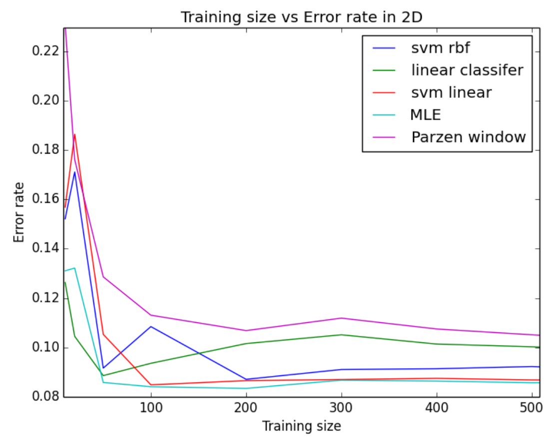
Section 4 Size of training samples

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:

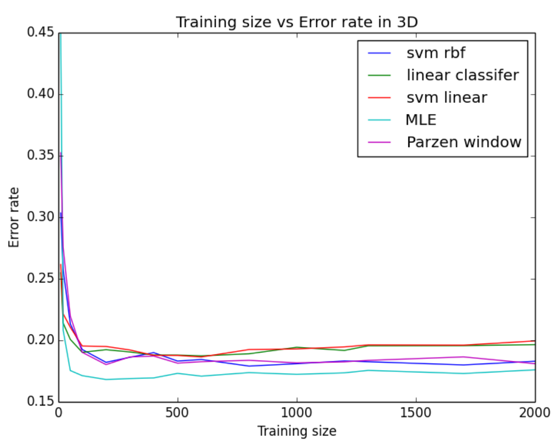
- Mean distance between two class is 3
- $P(\omega 1) = p(\omega 2) = 0.5$
- $\Sigma_1 \neq \Sigma_2$, variance of data on each dimension(diagonal) is less than 5
- The size of entire data is 4000
- The data is synthetically generated with Gaussian distribution
- The size of Parzen window is h = 3



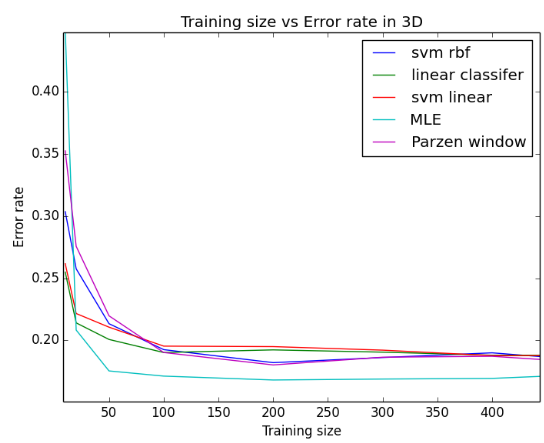
2D size range from 0 to 2000



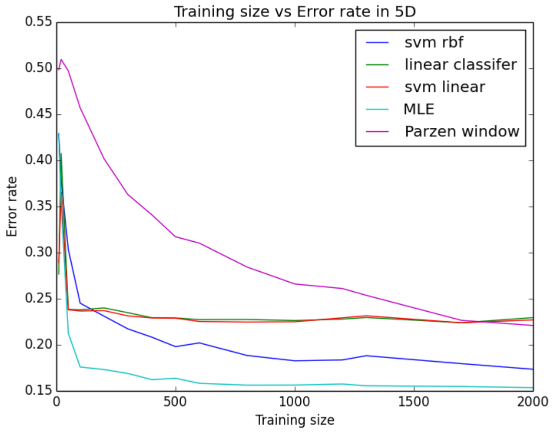
2D size range from 0 to 500



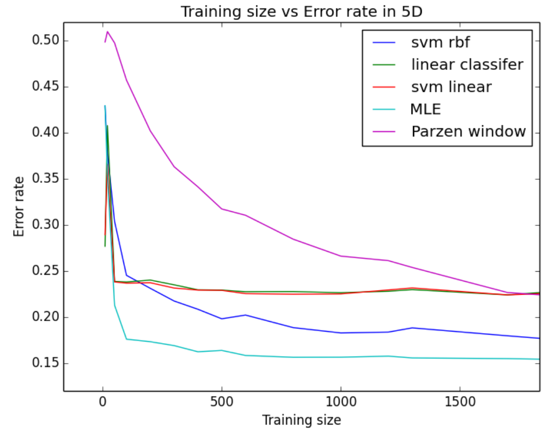
3D size range from 0 to 2000



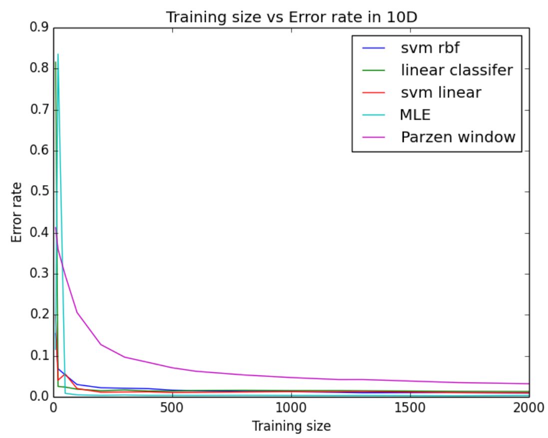
3D size range from 0 to 500



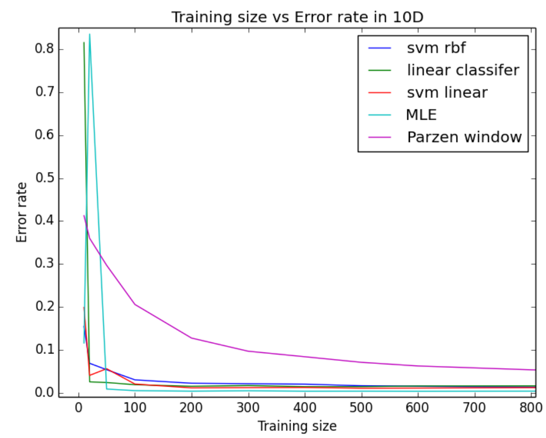
5D size range from 0 to 2000



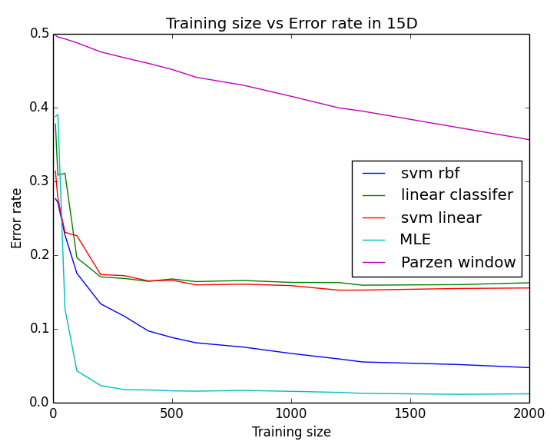
5D size range from 0 to 500



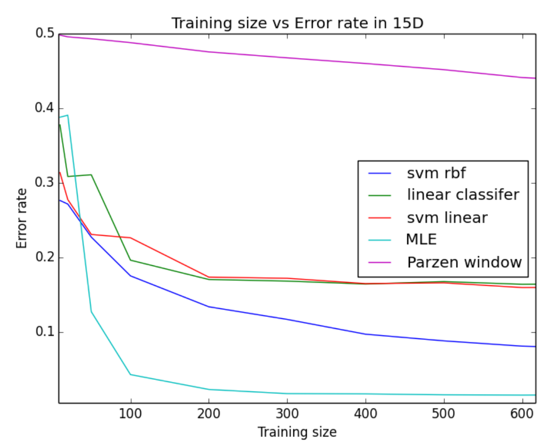
10D size range from 0 to 2000



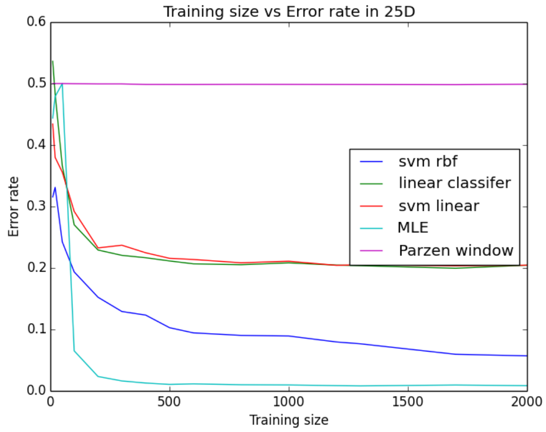
10D size range from 0 to 500

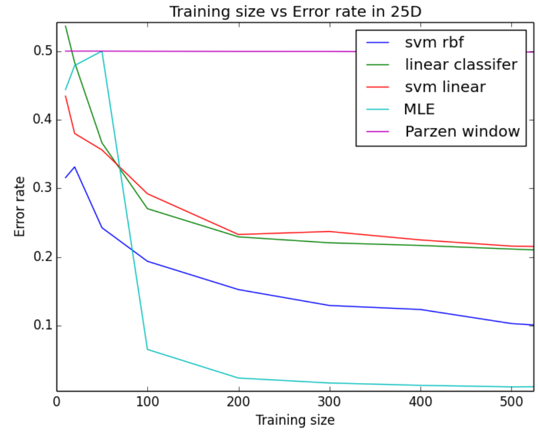


15D size range from 0 to 2000



15D size range from 0 to 500





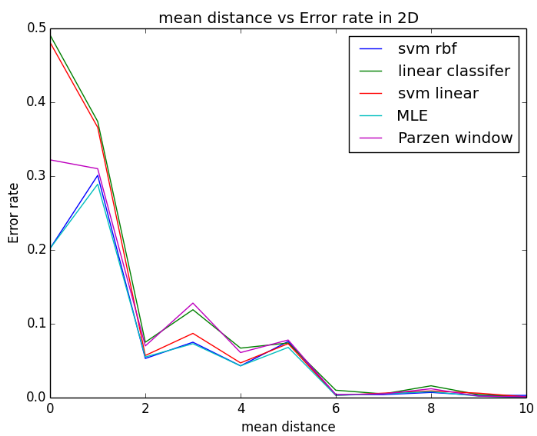
25D size range from 0 to 2000   25D size range from 0 to 500

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.

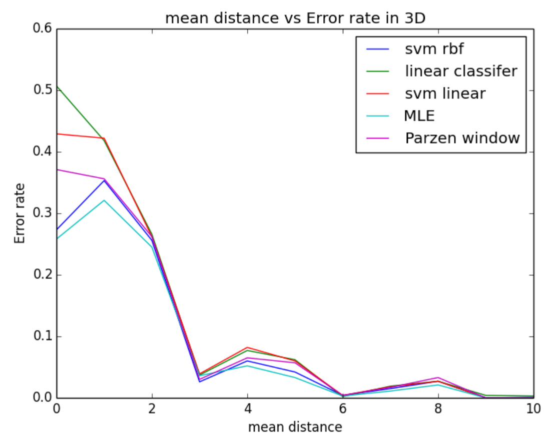
Section 5 Mean distance

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:

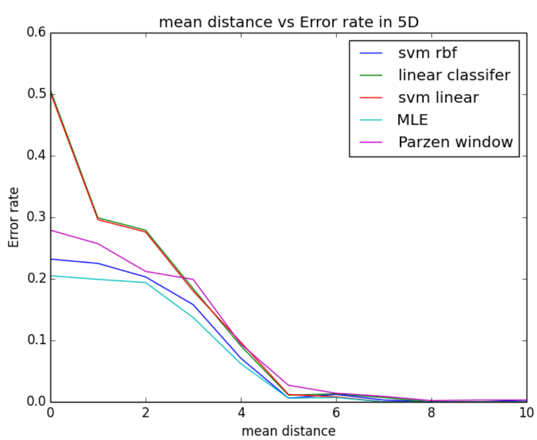
- The size of entire data is 2000, and the size of training data is 1000
- $P(\omega 1) = p(\omega 2) = 0.5$
- $\Sigma_1 \neq \Sigma_2$, variance of data on each dimension(diagonal) is less than 5
- The data is synthetically generated with Gaussian distribution
- The size of Parzen window is h = 3



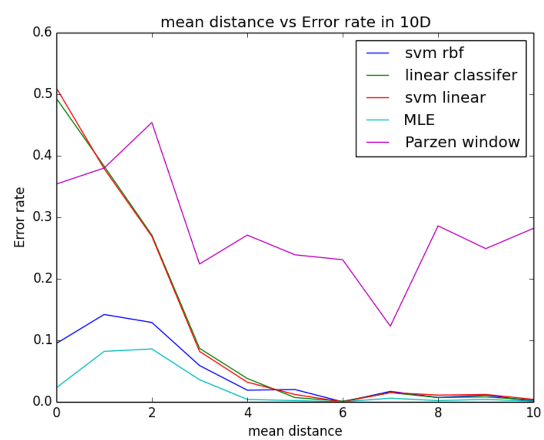
2D



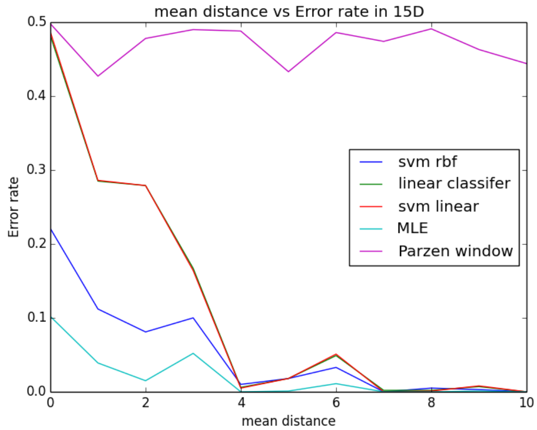
3D

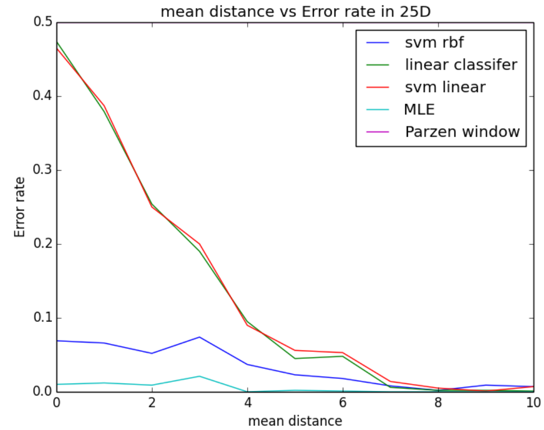


5D



10D





15D 25D

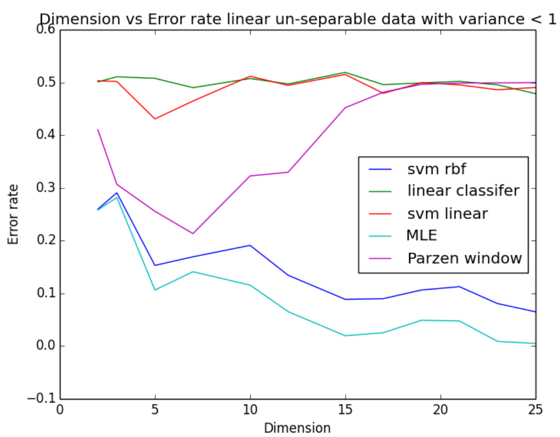
According to the graph, we can find that the error rate goes down with the mean distance between two class increasing no matter what dimension is. It probably because the larger the mean distance between two class, the more obvious for the classifiers to distinct each class of data. The SVM with Gaussian 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

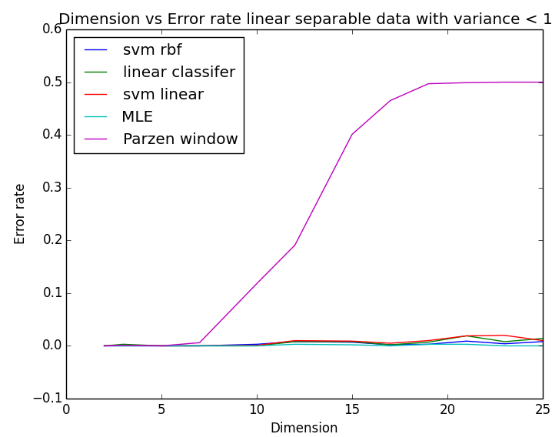
This section tests the performance of different classifiers with linear separable data and non linear separable data under different variance of data (diagonal value of covariance). The experiment controls several parameters:

- 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.

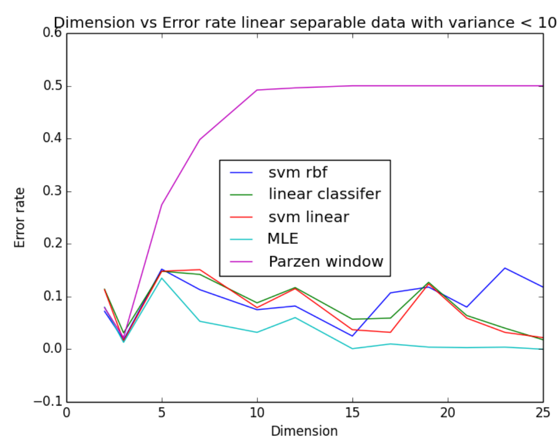


non linear separable data, variance < 1

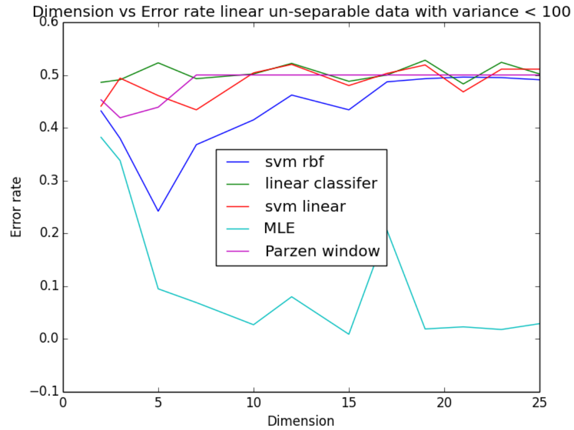


separable data, variance < 1

non linear separable data, variance < 10

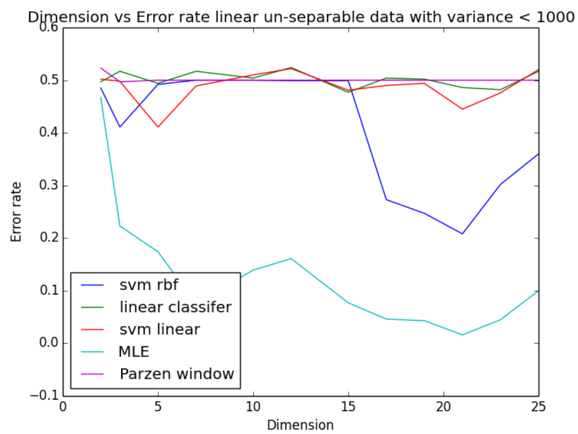


separable data, variance < 10



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

```
import numpy as np
import math


def separate_data(train_data,class_label):
    train_1 = []
    train_2 = []

    for data,label in zip(train_data,class_label):
            if label == 0:
                train_1.append(np.array(data).T)
            else:
                train_2.append(np.array(data).T)
        train_1 = np.matrix(train_1)
        train_2 = np.matrix(train_2)
        return train_1,train_2


    def compute_var(data,mean):
        val = 0
        for d in data:
            #print d,mean
            a = d-mean

            #print a.T*a
            val += np.dot(a.T,a)
        return val

    def train(train_data, class_label):
        train_1,train_2 = separate_data(train_data,class_label)
        mean_1 = np.mean(train_1,axis=0)
        mean_2 = np.mean(train_2,axis=0)
        mean = np.mean(train_data,axis=0)


```

```
        s1_w = compute_var(train_1,mean_1)
        #print s1_w
        s2_w = compute_var(train_2,mean_2)
        #print s2_w
        sw = s1_w + s2_w
        w = sw.I*np.matrix(mean_1-mean_2).I
        #print np.linalg.norm(w,0)
        return w,mean

    def test(test_data,w,mean):
        res = []
        #print w
        #print mean
        for d in test_data:

            v = (np.matrix(d-mean)*w)[0,0]
            #print v
            if v > 0:
                res.append(0)
            else:
                res.append(1)
        return res
```

svm.py

```
from sklearn import svm

def train(train_data,train_label,k):
    X = train_data
    y = train_label
    clf = svm.SVC(kernel=k)
    clf.fit(X, y)
    return clf

    def test(test_data,clf):
        res = clf.predict(test_data)
        return res
```

mle.py

```
#mle py
#function of mle
```

```

import numpy as np

def estimate_mu(data):
    data =  np.array(data)
    return np.mean(data,axis=0)

def estimate_covariance(data):
    data =  np.array(data)
    return np.cov(data.T)
```

parzen.py

```
#parzen.py
#function of parzen window
import math
import numpy as np

def window_function(sample,test_data,h):
    #print sample
    #print "x",test_data

    for x,x_i in zip(sample,test_data):

        if abs(x-x_i) > float(h)/2:
            return 0
    return 1

def estimate_single_px(samples,test_data,h):
    inRegion = 0
    outRegion = 0

    for sample in samples:
        if window_function(sample,test_data,h) == 1:
            inRegion  += 1
        else:
            outRegion += 1

```

```
        return float(inRegion)/((inRegion+outRegion)*(h**len(test_data)))


    def get_px(samples,test_datas,h):
        px =[]

        for x in test_datas:
            x =np.array(x)

            val = estimate_single_px(samples,x,h)
            px.append(val)
        return px
```

testing_parzen.py

```
#parzen.py
#function of parzen window
import math
import numpy as np

def window_function(sample,test_data,h):
    #print sample
    #print "x",test_data

        for x,x_i in zip(sample,test_data):

            if abs(x-x_i) > float(h)/2:
                return 0
        return 1

    def estimate_single_px(samples,test_data,h):
        inRegion = 0
        outRegion = 0

        for sample in samples:
            if window_function(sample,test_data,h) == 1:
                inRegion  += 1
            else:
```

```
                outRegion += 1

        return float(inRegion)/((inRegion+outRegion)*(h**len(test_data)))


def get_px(samples,test_datas,h):
    px =[]

    for x in test_datas:
        x =np.array(x)

        val = estimate_single_px(samples,x,h)
        px.append(val)
    return px
```

testing_mle.py

```
import mle
import numpy as np
import random
import parzen
from numpy import matrix
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib.mlab import bivariate_normal
import math
from numpy.linalg import linalg
import time


def compute_gx(mu_1,mu_2,cov_1,cov_2,w1,w2,x):

    mu_1 =matrix(mu_1)
    mu_2 = matrix(mu_2)
    cov_1 = matrix(cov_1)
    cov_2 = matrix(cov_2)


```

```
        const = 0.5 * math.log(linalg.det(cov_2)/linalg.det(cov_1)) + math.log(float(w1)/w2)

        #print "const",const
        #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 )
        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 )
        #print gx
        v = gx.tolist()
        return v[0][0]+const



    def testing(train_samples_1,test_samples_1,train_samples_2,test_samples_2):

        now = time.time()


        estimate_mu_1 = mle.estimate_mu(train_samples_1)
        estimate_cov_1 = mle.estimate_covariance(train_samples_1)

        estimate_mu_2 = mle.estimate_mu(train_samples_2)
        estimate_cov_2 = mle.estimate_covariance(train_samples_2)

        w1 = len(train_samples_1)/float(len(train_samples_1)+len(train_samples_2))
        w2 = len(train_samples_2)/float(len(train_samples_1)+len(train_samples_2))

        error = 0
        for d in test_samples_1:
            val = compute_gx(estimate_mu_1 ,estimate_mu_2,estimate_cov_1,estimate_cov_2,w1,w2,d)
```

```
        if val > 0:
            pass
        else:
            error += 1

    for d in test_samples_2:
        val = compute_gx(estimate_mu_1 ,estimate_mu_2,estimate_cov_1,estimate_cov_2,w1,w2,d)
        if val > 0:
            error += 1
        else:
            pass

    res =  float(error)/float(len(test_samples_1)+len(test_samples_2))

    return res
    #errors.append(res)
    #times.append(time.time()-now)

    #print errors
    #print "error rate:",sum(errors)/len(errors)
    #print "average_time:",sum(times)/len(times)
    #return sum(errors)/len(errors)
    #return sum(times)/len(times)
```

test.py

```
import svm
import linear
import numpy as np
import random
import matplotlib.pyplot as plt
import testing_mle
import testing_parzen

def generate_gaussian_data(mean,cov,size,label):
    x = np.random.multivariate_normal(mean, cov,size)
```

```
    y = np.repeat(label, size)
    return x,y

def error_rate(predict_label, true_label):
    error = 0
    accurate = 0
    for l1,l2 in zip(predict_label, true_label):
        if l1 == l2:
            accurate += 1
        else:
            error += 1
    return float(error)/float(accurate+error)
    #return error

def separate_data(prior,datas,labels):
    train_data = []
    train_label = []
    test_data = []
    test_label = []

    for data,label in zip(datas,labels):
        if random.random() < prior:
            train_data.append(data)
            train_label.append(label)
        else:
            test_data.append(data)
            test_label.append(label)
    return train_data,train_label,test_data,test_label

def create_covariance_matrix(dimension):
    cov = np.zeros((dimension, dimension))
    for i in range(dimension):
        for j in range(i,dimension,1):
            if i == j:
                cov[i][j] = random.random()*5
            else:
                val = 2*random.random()-1
                cov[i][j] = val
```

```
                    cov[j][i] = val
        return cov

    def set_up_parameter(dimension,distance):
        mean_1 = np.zeros(dimension)
        mean_2 = np.zeros(dimension)
        for d in range(dimension):
            val = random.random()
            mean_1[d] = val
            mean_2[d] = val

        index = int(random.random()*dimension)
        mean_2[index] += distance

        cov_1 = create_covariance_matrix(dimension)
        cov_2 = create_covariance_matrix(dimension)
        return mean_1,cov_1,mean_2,cov_2

    def generate_unseparateable_data(size,dimension):
        mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,distance)



        x_1,y_1 = generate_gaussian_data(mean_1,cov_1,2000,0)
        x_2,y_2 = generate_gaussian_data(mean_1,cov_2,2000,1)

        #x_1 = 20*np.random.random((size,dimension))-10
        #x_2 = 20*np.random.random((size,dimension))-10
        #y_1 = np.repeat(0, size)
        #y_2 = np.repeat(1, size)
        return x_1,y_1,x_2,y_2

    '''''
    testing
```

```
    '''

    dimension = 50
    distance = 3
    sizes = [10,20,50,100,200,300,400,500,600,800,1000,1200,1300,1700,2000]
    dimensions = [25]
    #dimensions = [2,3]
    prior_1 = 0.5
    prior_2 = 1 - prior_1


    '''''
    mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,distance)
    x_1,y_1 = generate_gaussian_data(mean_1,cov_1,10000,0)
    x_2,y_2 = generate_gaussian_data(mean_2,cov_2,10000,1)
    '''
    '''''
    #testing training size vs error rate
    for dimension in dimensions:

        print dimension,"dimension"
        mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,distance)
        x_1,y_1 = generate_gaussian_data(mean_1,cov_1,2000,0)
        x_2,y_2 = generate_gaussian_data(mean_2,cov_2,2000,1)
        svm_rbf_errors = []
        svm_linear_errors = []
        linear_errors = []
        svm_poly_errors = []
        mle_errors=[]
        parzen_errors =[]
        for size in sizes:

```

```
        print "size:",size

        #separate data
        s1 = size * prior_1
        s2 = size * prior_2

        train_data_1 = x_1[0:s1]
        train_label_1 = y_1[0:s1]
        train_data_2 = x_2[0:s2]
        train_label_2 = y_2[0:s2]

        test_data_1 = x_1[s1:]
        test_label_1 = y_1[s1:]
        test_data_2 = x_2[s2:]
        test_label_2 = y_2[s2:]


        for a in train_data_1:
            plt.scatter(a[0],a[1],color='blue')
        for a in train_data_2:
            plt.scatter(a[0],a[1],color='red')
        plt.show()


        #generate traing data
        x = np.concatenate((train_data_1,train_data_2), axis=0)
        y = np.concatenate((train_label_1,train_label_2), axis=0)


        #traning a model
        svm_rbf = svm.train(x,y,"rbf")
        svm_linear = svm.train(x,y,"linear")
        w,mean = linear.train(x,y)

        #generate test data
        test_data  = np.concatenate((test_data_1,test_data_2), axis=0)
```

```
            test_label = np.concatenate((test_label_1,test_label_2), axis=0)

            #prediction
            svm_rbf_label = svm.test(test_data,svm_rbf)
            linear_label = linear.test(test_data,w,mean)
            svm_linear_label = svm.test(test_data,svm_linear)

            #get result
            svm_rbf_error = error_rate(test_label,svm_rbf_label)
            linear_error = error_rate(test_label,linear_label)
            svm_linear_error = error_rate(test_label,svm_linear_label)
            mle_error = testing_mle.testing(train_data_1,test_data_1,train_data_2,test_data_2)
            parzen_error = testing_parzen.testing(train_data_1,test_data_1,train_data_2,test_data_2,3)
            print "svm error(rbf):",svm_rbf_error
            print "svm error(linear):",svm_linear_error
            print "linaer_classifer error:",linear_error
            print "mle error:",mle_error
            print "parzen error:",parzen_error
            #store results
            svm_rbf_errors.append(svm_rbf_error)
            linear_errors.append(linear_error)
            svm_linear_errors.append(svm_linear_error)
            mle_errors.append(mle_error)
            parzen_errors.append(parzen_error)
            print ""

        plt.title("Training size vs Error rate in "+str(dimension)+"D")
```

```
        plt.xlabel("Training size")
        plt.ylabel("Error rate")
        plt.plot(sizes,svm_rbf_errors,label="svm rbf")
        plt.plot(sizes,linear_errors,label="linear clasifer")
        plt.plot(sizes,svm_linear_errors,label="svm linear")
        plt.plot(sizes,mle_errors,label="MLE")
        plt.plot(sizes,parzen_errors,label="Parzen window")

        plt.legend(loc="best")
        plt.show()
    '''


    #testing mean distance vs error rate
    distances = [0,1,2,3,4,5,6,7,8,9,10]

    for dimension in dimensions:
        print dimension,"dimension"

        svm_rbf_errors = []
        svm_linear_errors = []
        linear_errors = []
        svm_poly_errors = []
        mle_errors=[]
        parzen_errors =[]
        size = 1000
        for distance in distances:
            print distance,"distance"
            mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,distance)
            x_1,y_1 = generate_gaussian_data(mean_1,cov_1,1000,0)
            x_2,y_2 = generate_gaussian_data(mean_2,cov_2,1000,1)

            #separate data
```

```
            s1 = size * prior_1
            s2 = size * prior_2

            train_data_1 = x_1[0:s1]
            train_label_1 = y_1[0:s1]
            train_data_2 = x_2[0:s2]
            train_label_2 = y_2[0:s2]

            test_data_1 = x_1[s1:]
            test_label_1 = y_1[s1:]
            test_data_2 = x_2[s2:]
            test_label_2 = y_2[s2:]

            #generate traing data
            x = np.concatenate((train_data_1,train_data_2), axis=0)
            y = np.concatenate((train_label_1,train_label_2), axis=0)


            #traning a model
            svm_rbf = svm.train(x,y,"rbf")
            svm_linear = svm.train(x,y,"linear")
            w,mean = linear.train(x,y)

            #generate test data
            test_data  = np.concatenate((test_data_1,test_data_2), axis=0)
            test_label = np.concatenate((test_label_1,test_label_2), axis=0)

            #prediction
            svm_rbf_label = svm.test(test_data,svm_rbf)

            linear_label = linear.test(test_data,w,mean)
            svm_linear_label = svm.test(test_data,svm_linear)

```

```
            #get result
            svm_rbf_error = error_rate(test_label,svm_rbf_label)
            linear_error = error_rate(test_label,linear_label)
            svm_linear_error = error_rate(test_label,svm_linear_label)
            mle_error = testing_mle.testing(train_data_1,test_data_1,train_data_2,test_data_2)
            parzen_error = testing_parzen.testing(train_data_1,test_data_1,train_data_2,test_data_2,3)

            print "svm error(rbf):",svm_rbf_error
            print "svm error(linear):",svm_linear_error
            print "linaer_classifer error:",linear_error
            print "mle error:",mle_error
            print "parzen error:",parzen_error
            #store results
            svm_rbf_errors.append(svm_rbf_error)
            linear_errors.append(linear_error)
            svm_linear_errors.append(svm_linear_error)
            mle_errors.append(mle_error)
            parzen_errors.append(parzen_error)
            print ""

        plt.title("mean distance vs Error rate in "+str(dimension)+"D")
        plt.xlabel("mean distance")
        plt.ylabel("Error rate")
        plt.ylim(-0.1,0.6)
        plt.plot(distances,svm_rbf_errors,label="svm rbf")
        plt.plot(distances,linear_errors,label="linear classifer")
        plt.plot(distances,svm_linear_errors,label="svm linear")
```

```
        plt.plot(distances,mle_errors,label="MLE")
        plt.plot(distances,parzen_errors,label="Parzen window")
        plt.legend(loc="best")
        plt.show()



    size = 1000
    svm_rbf_errors = []
    svm_linear_errors = []
    linear_errors = []
    svm_poly_errors = []
    mle_errors=[]
    parzen_errors =[]

    '''''
    mean_1,cov_1,mean_2,cov_2 = set_up_parameter(25,0)

    for dimension in dimensions:
        print dimension,"dimension"


        #size = 1000
        mean_1,cov_1,mean_2,cov_2 = set_up_parameter(dimension,6)
        sigma_1 = cov_1[0:dimension,0:dimension]
        sigma_2 = cov_2[0:dimension,0:dimension]

        x_1,y_1 = generate_gaussian_data(mean_1,sigma_1,1000,0)
        x_2,y_2 = generate_gaussian_data(mean_2,sigma_2,1000,1)


        #separate data
        s1 = size * prior_1
        s2 = size * prior_2

```

```
        train_data_1 = x_1[0:s1]
        train_label_1 = y_1[0:s1]
        train_data_2 = x_2[0:s2]
        train_label_2 = y_2[0:s2]

        test_data_1 = x_1[s1:]
        test_label_1 = y_1[s1:]
        test_data_2 = x_2[s2:]
        test_label_2 = y_2[s2:]

        #generate traing data
        x = np.concatenate((train_data_1,train_data_2), axis=0)
        y = np.concatenate((train_label_1,train_label_2), axis=0)


        #traning a model
        svm_rbf = svm.train(x,y,"rbf")
        svm_linear = svm.train(x,y,"linear")
        w,mean = linear.train(x,y)

        #generate test data
        test_data  = np.concatenate((test_data_1,test_data_2), axis=0)
        test_label = np.concatenate((test_label_1,test_label_2), axis=0)

        #prediction
        svm_rbf_label = svm.test(test_data,svm_rbf)
        linear_label = linear.test(test_data,w,mean)
        svm_linear_label = svm.test(test_data,svm_linear)

        #get result
        svm_rbf_error = error_rate(test_label,svm_rbf_label)
        print "svm error(rbf):",svm_rbf_error

```

```
        linear_error = error_rate(test_label,linear_label)
        print "linaer_classifer error:",linear_error


        svm_linear_error = error_rate(test_label,svm_linear_label)
        print "svm error(linear):",svm_linear_error

        mle_error = testing_mle.testing(train_data_1,test_data_1,train_data_2,test_data_2)
        print "mle error:",mle_error

        parzen_error = testing_parzen.testing(train_data_1,test_data_1,train_data_2,test_data_2,3)
        print "parzen error:",parzen_error

        #store results
        svm_rbf_errors.append(svm_rbf_error)
        linear_errors.append(linear_error)
        svm_linear_errors.append(svm_linear_error)
        mle_errors.append(mle_error)
        parzen_errors.append(parzen_error)
        print ""

    plt.title("Dimension vs Error rate linear separable data with variance < 10000")
    plt.xlabel("Dimension")
    plt.ylabel("Error rate")
    plt.ylim(-0.1,0.6)
    plt.plot(dimensions,svm_rbf_errors,label="svm rbf")
    plt.plot(dimensions,linear_errors,label="linear classifer")
    plt.plot(dimensions,svm_linear_errors,label="svm linear")
    plt.plot(dimensions,mle_errors,label="MLE")
    plt.plot(dimensions,parzen_errors,label="Parzen window")
```

```
    plt.legend(loc="best")
    plt.show()
    '''

    '''''
    x_1,y_1,x_2,y_2 = generate_unseparateable_data(size,dimension)
    for a in x_1:
        plt.scatter(a[0],a[1],color='blue')
    for a in x_2:
        plt.scatter(a[0],a[1],color='red')
    plt.show()
    '''
```