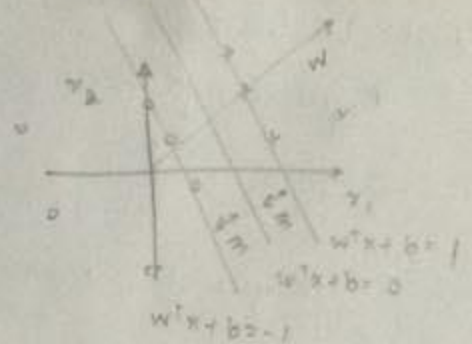
MACHINE LEARNING CS60050

ASSIGNMENT II.

$\{x_1, x_2 \ldots x_n\}$ → data set

$y = +, -1$ → class labels.



$y = 1 \quad w^Tx_i + b \geq 1$

$y = -1 \quad w^Tx_i + b \leq -1$

$y_i(w^Tx_i + b) \geq 1 \ \forall \ (x_i, y_i)$

max. margin $m = 2/\|w\|$

decision boundary $y_i(w^Tx_i + b) \geq 1 \ \forall i$

$\min \ \frac{1}{2}\|w\|^2$

subject to $y_i(w^Tx_i + b) \geq 1$

constrained optimization problem. Using Lagrange multipliers

$L(x, \mu) = f(x) - \sum_i \mu_i g_i(x)$

↳ min $f(x)$ sub to $g_i(x) \leq 0 \ \forall i$

$L = \frac{1}{2}w^Tw + \sum_{i=1}^{n} d_i(1 - y_i(w^Tx_i + b))$

$d_i \geq 0$

$\|w\|^2 = w^Tw$

$L = \frac{1}{2}w^Tw + \sum_{i=1}^{n} d_i(1 - y_i(w^Tx_i + b)) = \frac{1}{2}\sum_{k=1}^{m} w^kw^k + \sum_{i=1}^{n} d_i\left(1 - y_i\left(\sum_{k=1}^{m} w^kw_i^k + b\right)\right)$

Taking gradient w.r.t w, b

$\frac{\partial L}{\partial w} \quad w + \sum_{i=1} d_i(-y_ix_i) = 0 \Rightarrow w = \sum_{i=1}^{n} d_iy_ix_i$

$\frac{\partial L}{\partial b} = 0 \quad \sum_{i=1}^{n} d_iy_i = 0$

substituting $w = \sum_{i=1}^{n} d_iy_ix_i$

$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_id_jy_iy_jx_i^Tx_j + \sum_{i=1}^{n} d_i$

since $\sum_{i=1}^{n} d_iy_i = 0$

This is a function of d_i only.

dual problem :

Objective function needs to be maximized

$$\max \; W(d) = \sum_{i=1}^{n} d_i - \frac{1}{2} \sum_{i=j=1}^{n} d_i d_j y_i y_j x_i^T x_j$$

subject to $d_i \geq 0$ $\quad \sum_{i=1}^{n} d_i y_i = 0$

X1	X2	Y	~~$x_1 x_2$~~
0	0	0	
0	1	1	
1	0	1	
1	1	0	

Lifting the data to a 6 dimensional space using Kernels.

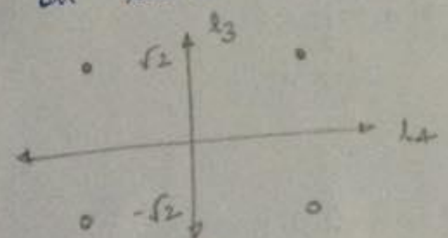
Taking $0 \rightarrow -1$ $\quad 1 \rightarrow 1$

	u x_1	v x_2	Y	$l_3 = \sqrt{2}\, x_1 x_2$	$l_1 = x_1^2$	$l_2 = x_2^2$	$l_4 = \sqrt{2}\, x_1$	$l_5 = \sqrt{2}\, x_2$	$l_6 = 1$
class I	-1	-1	+1	$\sqrt{2}$	1	1	$\sqrt{2}$	$\sqrt{2}$	1
	-1	-1	1	$\sqrt{2}$	1	1	$-\sqrt{2}$	$-\sqrt{2}$	1
class II	1	-1	-1	$-\sqrt{2}$	1	1	$\sqrt{2}$	$-\sqrt{2}$	1
	-1	1	-1	$-\sqrt{2}$	1	1	$-\sqrt{2}$	$\sqrt{2}$	1

$$K(u,v) = (u \cdot v + 1)^2 = (u_1 v_1 + u_2 v_2 + 1)^2$$ ... in 2 dim. plane.

$$= (u_1 v_1)^2 + (u_2 v_2)^2 + 2u_1 u_2 v_1 v_2 + 2u_1 v_1 + 2u_2 v_2 + 1$$

projecting on the $\sqrt{2} x_1 x_2 = l_3$ axis.



optimal w

$= [0, 0, 1, 0, 0, 0]$

Suppose a pair (X, Y) take the values in $\mathbb{R}^d \times \{1,2 \dots K\}$ where Y is the class label of X. The conditional distribution of X, given that the label Y takes value r is given by

$$X \mid Y = r \sim P_r \qquad r = 1, 2 \dots K$$

↳ distributed as.

$P_r \rightarrow$ probability distribution.

$C : \mathbb{R}^d \rightarrow \{1, 2 \dots, K\}$

C classifies x to the class $C(x)$

Prob. of misclassification, or risk $R(C) = P\{C(X) \neq Y\}$

Baye's classifier is

$$C^{Bayes}(x) = \underset{r \in \{1,2,\dots K\}}{\arg\max}\ P(Y = r \mid X = x) \quad \dots\dots \text{①}$$

now : error function $E(f) = \int I(y = f(x))\, p(x,y)\, dx\, dy \quad \dots \text{②}$

$f(x) \rightarrow$ classifier

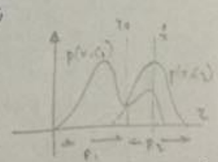
$p(x,y) \rightarrow$ intrinsic data distribution

Bayes decision rule assigns

$$p(x, C_i) = p(C_i \mid x)\, p(x)$$

assign x to the class C_i for which $p(C_i \mid x)$ is the largest. b))

For binary classification ;



$$p(\text{error}) = \int_{-\infty}^{\infty} p(\text{error}, x)\, dx$$

$$= \int_{R_1} p(x, C_2)\, dx + \int_{R_2} p(x, C_1)\, dx.$$

minimizing the number of misclassifications at the decision boundary x_0 gives the condition

$$p(x, C_i) = p(C_i \mid x)\, p(x)$$

↳ i.e assign x to C_i for which $p(C_i \mid x)$ is the largest.

$$p(\text{error}) = \int_{R_1} p(C_2|x)p(x)\,dx + \int_{R_2} p(C_1|x)p(x)\,dx$$

is Bayes error → probability of misclassification

⇒ ② is minimized when condition ① is satisfied
⇒ Baye's solution gives the minimum error possible.

3. FISHER'S LINEAR DISCRIMINANT :

- predictor $y = w^T x$.
- if $y > w_0$ predict C_1 else C_2

Taking $m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n$ $\quad m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n$.

max. separation between projected means

$$m_2 - m_1 = w^T(\bar{m}_2 - \bar{m}_1)$$

constraining $\|w\|^2 = 1$

Lagrangian $L(w, \lambda) = \bar{w}^T(\bar{m}_2 - \bar{m}_1) + \lambda(\|\bar{w}\|^2 - 1)$

solution $\bar{w} \propto \bar{m}_2 - \bar{m}_1$

gives the solution $w \propto (S_W)^{-1}(m_2 - m_1)$

within class variance : $S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T$

to the objective function $J(w) = \frac{w^T S_B w}{w^T S_W w}$

RELATION TO LEAST SQUARES :

The ~~rela~~ least squares approach to the determination of a linear discriminant was based on the goal of making the model predictions as close as possible to a set of target values. By contrast Fisher criteria requires maximum class separation in the output space.

sum of squares error function :

$$E = \frac{1}{2} \sum_{n=1}^{N} (w^T x_n + w_0 - t_n)^2$$

setting the derivative of E w.r.t w_0 and w to be zero

$$\sum_{n=1}^{N} (w^T x_n + w_0 - t_n) = 0$$

$$\sum_{n=1}^{N} (w^T x_n + w_0 - t_n)\, x_n = 0$$

$$w_0 = -w^T \bar{m}.$$

$$\sum_{n=1}^{N} t_n = N_1 \frac{N}{N_1} - N_2 \frac{N}{N_2} = 0$$

$$m = \frac{1}{N} \sum_{n=1}^{N} x_n = \frac{1}{N} (N_1 m_1 + N_2 m_2)$$

$$\therefore \left(S_W + \frac{N_1 N_2}{N} S_B \right) w = N (\bar{m}_1 - \bar{m}_2)$$

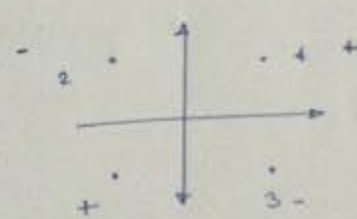
since $S_B w$ is always in the direction of $(m_2 - m_1)$ we can write

$$w \propto S_W^{-1} (\bar{m}_2 - \bar{m}_1)$$

which is the same as the Fisher's solution.

5.

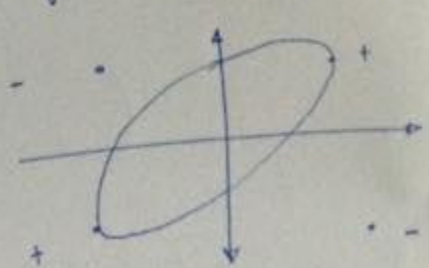
-1	-1	$x(1)$
-1	1	$x(2)$
$+1$	-1	$x(3)$
$+1$	$+1$	$x(4)$



not linearly dependent   linear kernel cannot classify the points

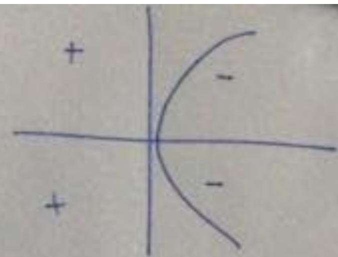
mapping
Linear kernel cannot shatter

Polynomial of degree 2



This can shatter
Because it can correctly classify all possible mappings

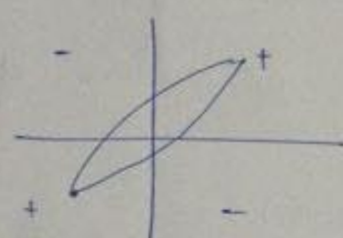
ellipse $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ polynomial of degree 2



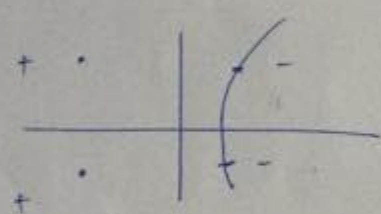
$x = ay^2 \rightarrow$ polynomial in 2^{nd} degree

All other 2^{nd} degree polynomials can be written as a combination of these 2.

c) Gaussian Kernel.



gaussian kernel with very low sigma value.



gaussian kernel with high sigma can correctly classify this.

4. Naive Bayes

Value of k	Error %
3	4.5321
5	4.7103
10	5.089.

```
import csv
import random
import math

def load_Csv(filename):
	lines = csv.reader(open(filename, "rb"))
	dataset = list(lines)
	for j in range(len(dataset)):
		dataset[j] = [float(x) for x in dataset[j]]
	return dataset

def split_dataset(dataset, split_Ratio):
	lol = lambda lst, sz: [lst[i:i+sz] for i in range(0, len(lst), sz)]
	return lol(dataset,split_Ratio)

def separate_By_Class(dataset):
	separate = {}
	for i in range(len(dataset)):
		vector = dataset[i]
		if (vector[-1] not in separated):
			separate[vector[-1]] = []
		separate[vector[-1]].append(vector)
	return separate

def mean(num):
	return sum(num)/float(len(num))

def stdev(num):
	avg = mean(num)
	var = sum([pow(x-avg,2) for x in num])/float(len(num)-1)
	return math.sqrt(var)

def summarize(dataset):
	summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
	del summaries[-1]
	return summaries

def summarizeByClass(dataset):
	separated = separate_By_Class(dataset)
	summaries = {}
	for classValue, instances in separated.iteritems():
		summaries[classValue] = summarize(instances)
	return summaries

def calculateProbability(x, mean, stdev):
	expo = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
	return (1 / (math.sqrt(2*math.pi) * stdev)) * expo

def calculateClassProbabilities(summaries, inputVector):
	probabilities = {}
	for classValue, classSummaries in summaries.iteritems():
		probabilities[classValue] = 1
		for i in range(len(classSummaries)):
			mean, stdev = classSummaries[i]
			x = inputVector[i]
			probabilities[classValue] *= calculateProbability(x, mean, stdev)
```

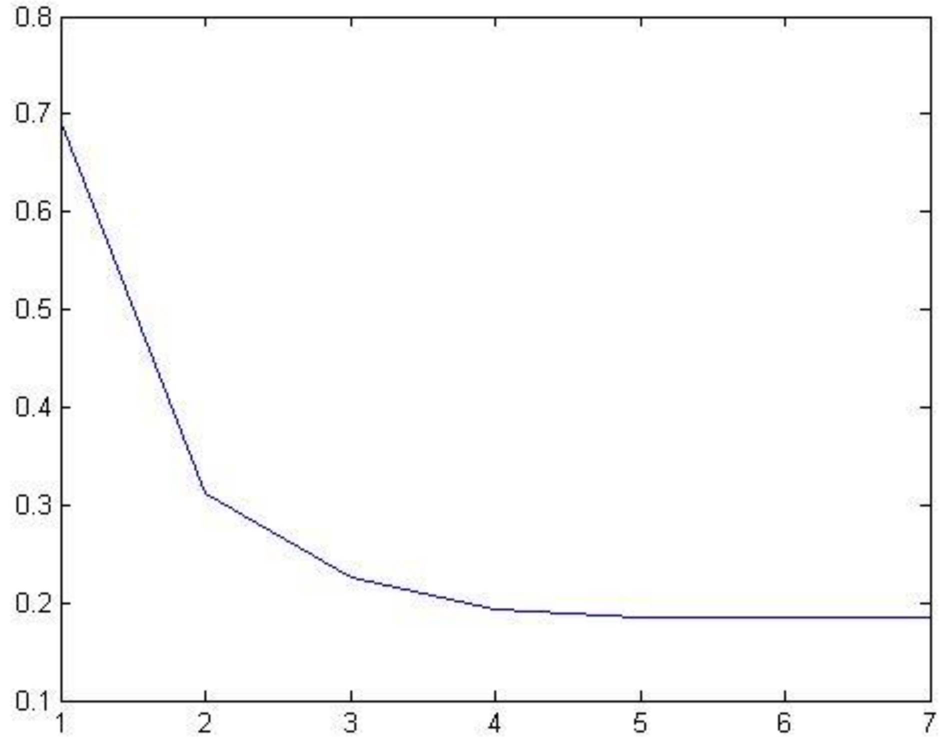
```
	return probabilities

def predict(summaries, inputVector):
	probabilities = calculateClassProbabilities(summaries, inputVector)
	bestLabel, bestProb = None, -1
	for classValue, probability in probabilities.iteritems():
		if bestLabel is None or probability > bestProb:
			bestProb = probability
			bestLabel = classValue
	return bestLabel

def getPredictions(summaries, testset):
	predictions = []
	for i in range(len(testset)):
		result = predict(summaries, testset[i])
		predictions.append(result)
	return predictions

def getError(testset, predictions):
	w = 0
	for i in range(len(testset)):
		if testset[i][-1] != predictions[i]:
			wrong += 1
	return (w/float(len(testset))) * 100.0

def main():
	filename = 'data.csv'
	dataset = load_Csv(filename)
	Error = 0
	for i in range(0,3):
		splitRatio = len(dataset)/3
		sets= split_dataset(dataset, splitRatio)
		#model
		train = []
		for x in range(0,3):
			if(x!=i):
				train += sets[x]
		summaries = summarizeByClass(train)
		test = sets[i]

		predictions = getPredictions(summaries, test)
		Error += getError(test, predictions)
	print('Error for k =3: {0}%').format(Error/3)

	Error =0
	for i in range(0,5):
		splitRatio = len(dataset)/5
		sets= split_dataset(dataset, splitRatio)
		# prepare model
		train = []
		for x in range(0,3):
			if(x!=i):
				train += sets[x]
		summaries = summarizeByClass(train)
		test = sets[i]
```

```
        predictions = getPredictions(summaries, test)
        Error += getError(test, predictions)
    print('Error for k =5: {0}%').format(Error/5)

    Error = 0
    for i in range(0,10):
        splitRatio = len(dataset)/10
        sets= split_dataset(dataset, splitRatio)
        # prepare model
        train = []
        for x in range(0,3):
            if(x!=i):
                train += sets[x]
        summaries = summarizeByClass(trainingSet)
        test = sets[i]
        predictions = getPredictions(summaries, test)
        Error += getError(test, predictions)
    print('Error for K = 10: {0}%').format(Error/10)


main()
```

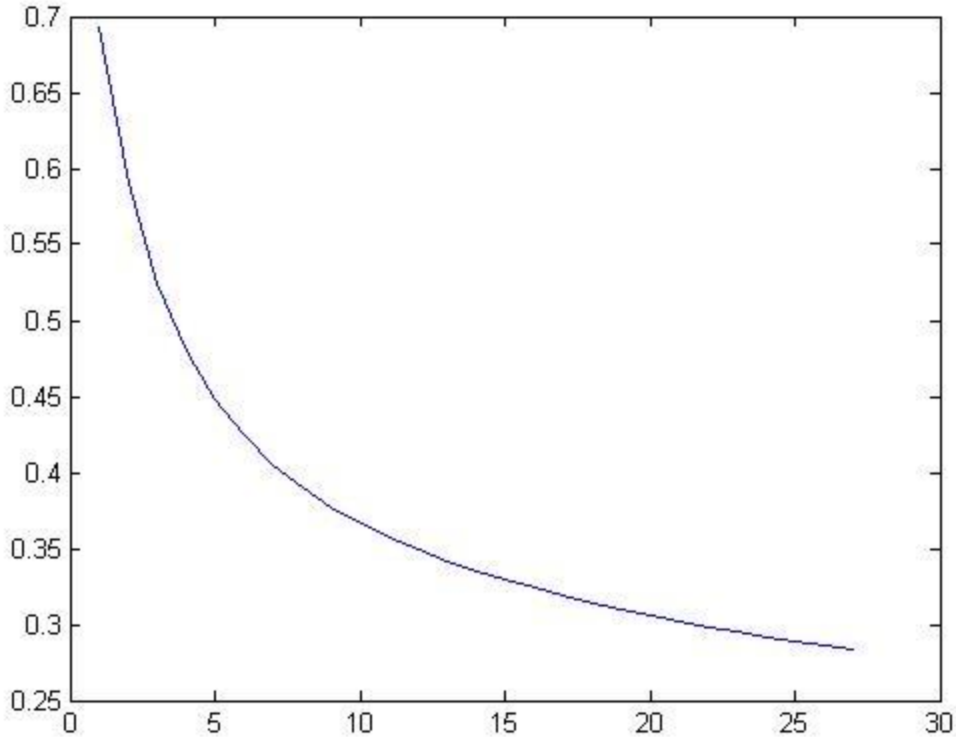
6.

a. converged at 7 iterations
   Iterations reached: 7
   IRLSacc =0.9474

b. converged at 27 iterations
   Iterations reached: 27
   SGDacc =   0.9079



Plot of objective function vs no of iterations for IRLS



Plot of objective function vs no of iterations for Stochastic gradient Descent

CODE:

```
%main.m
%applying logistic regression function on the breast cancer data.

%load the data and divide into the feature set and the labels
breastcancer = dlmread('breast-cancer.txt')
X=breastcancer(:,2:end);

%normalizing the data
m=mean(X);
s=std(X);
for i=1:size(X,1)
    for j=1:size(X,2)
        X1(i,j)=(X(i,j)-m(j))./s(j);
    end
end


%Change the target as value =0 if label='2' and value =1 if label='4'
t=breastcancer(:,1);
for i=1:size(X,1)
    if t(i,1)==2
        t(i,1)=0;
    else
```

```
        t(i,1)=1;
    end
end

%run both IRLS method and stochastic gradient descent method
w_irls=logistic_training(X1,t,100,0.05,'IRLS');
w_sgd=logistic_training(X1,t,100,0.05,'stochastic');

%finding the accuracy
y_irls=zeros(size(X1,1),1);
for i=1:size(X1,1)
    y_irls(i,1)=1./(1+exp(-1*X1(i,:)*w_irls));
    if y_irls(i,1)>0.5
        y_irls(i,1)=1;
    else
        y_irls(i,1)=0;
    end
end
IRLSacc= 1 -(sum(abs(t-y_irls))/size(X,1))

y_sgd=zeros(size(X1,1),1);
for i=1:size(X1,1)
    y_sgd(i,1)=1./(1+exp(-1*X1(i,:)*w_sgd));
    if y_sgd(i,1)>0.5
        y_sgd(i,1)=1;
    else
        y_sgd(i,1)=0;
    end
end
SGDacc= 1 -(sum(abs(t-y_sgd))/size(X,1))




%function implementing the logistic regression function
%function to train the data using logistic regression - Iterative
%re-weighted least squares method and stochastic gradient descent method
function [w_new]=logistic_training(X,t,maxiterations,epsilon,method)
    [rows,columns]=size(X);
    W_old=zeros(columns,1);
    if strcmp(method,'IRLS')==1



        %IRLS method
        for k=1:maxiterations
            [obJ,R,Y]=matrixR(w_old,X,t);
            objectivefunIRLS(k)=obJ;
            z=(transpose(X)*R*X*w_old) - (transpose(X)*(Y-t));
            w_new=(transpose(X)*R*X)\z;
            if(sum(abs(w_new-w_old))<epsilon)
                fprintf('converged at %d iteration\n',k);
                break;
            else
                w_old=w_new;
```

```
            end
        end
        fprintf('Iterations reached: %d\n',k);
        figure, plot(objectivefunIRLS);
    elseif strcmp(method,'stochastic')==1

        %Stochastic gradient descent method
        alpha=0.2;
        for k = 1:maxiterations
            [J, gradient] = lrCostFunction(wold,X,t);
            objectivefunSGD(k)=J;
            w_new = w_old - alpha * gradient;
            if(sum(abs(w_new-w_old))<epsilon)
                fprintf('converged at %d iteration\n',k);
                break;
            else
                w_old=w_new;
            end
        end
        fprintf('Iterations reached: %d\n',k);
        figure, plot(objectivefunSGD);
    end

end



%function calculating the matrices R and y for IRLS method
function [costJ,R,Y]=matrixR(w_old,X,t)
    [n,m]=size(X);
    R=zeros(n,n);
    Y=zeros(n,1);
    y=sigmoid(X*w_old);
    costJ = (-1/n) * sum( t .* log(y) + (1-t) .* log(1-y) );
    for i=1:n
        Y(i)=sigmoid(X(i,:)*w_old);
        R(i,i)=Y(i).*(1-Y(i));
    end
end

%function calculating the gradient and the cost function in Stochastic
%gradient method
function [costJ, grad] = lrCostFunction(w_old, X, t)
    n = size(t,1);
    y = sigmoid(X*w_old);
    grad=zeros(size(w_old));
    costJ = (-1/n) * sum( t .* log(y) + (1-t) .* log(1-y) );
    for i=1:n
        grad=grad+(y(i)-t(i))*transpose(X(i,:));
    end
    grad=(1/n)*grad;
end

%function to obtain the sigmoid of an input
function [value]=sigmoid(input)
    value=1./(1+exp(-1*input));
end
```