UNIVERSITY OF FLORIDA

Final Project: CAP6610 Spring 2016

Comparison of Supervised Learning Algorithms

Submitted By:

Rahul Maliakkal 80995471

Maithili Gokhale 91353676

Divya Chopra 06416863

Shruthi Gopalswamy 18912377

Prathamesh Singh 67431916

Our Code:

The kSVM code is included in this report. Code for the remainder of the algorithms is archived and uploaded at this link:

https://drive.google.com/folderview?id=0Byk3xpITQa1zekVQaVJDVlJ1aG8&usp=sharing

I] Introduction:

In this project, we have implemented different supervised learning algorithms and then compared their performances. The following algorithms were implemented:

1) libSvm
2) Multiclass KSVM
3) Random Forests
4) Deep Learning (CNN, MLP)
5) Naïve Bayes
6) AdaBoost

This report includes description, implementation details, performance, and results of each mentioned algorithm on given datasets. It also includes the comparison of the accuracies of the aforementioned algorithms.

Data Description:

1) Breast Cancer Wisconsin (Original) Data Set:
   Number of Instances: 699
   Number of Attributes: 9
   Number of Classes: 2

2) Optical Recognition of Handwritten Digits Data Set:
   Number of Instances: 5620
   Number of Attributes: 63
   Number of Classes: 10

3) Forest type mapping Data Set:
   Number of Instances: 326
   Number of Attributes: 26
   Number of Classes: 4

   The number of attributes have been reduced by 1 as it does not include the class labels.

II] Support Vector Machine:

Support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. It is a non-probabilistic binary linear classifier. It plots the examples as coordinates in space, in a way that the members of the separate categories are divided by a clear gap in the most distinct fashion. Later samples are then plotted into that same space and t is predicted which side of the gap they will fall in.

We implemented library SVM using the open source Machine Learning library LIBSVM in scikit.

RESULT:

	DataSet1(Breast Cancer)	DataSet2(Handwriting)	DataSet3 (forests)
10% Training	4-6%	3-4%	16% error
30% Training	4-5%	3-4%	
50% Training	2-3%	3-4%	

Table (1) Linear SVM

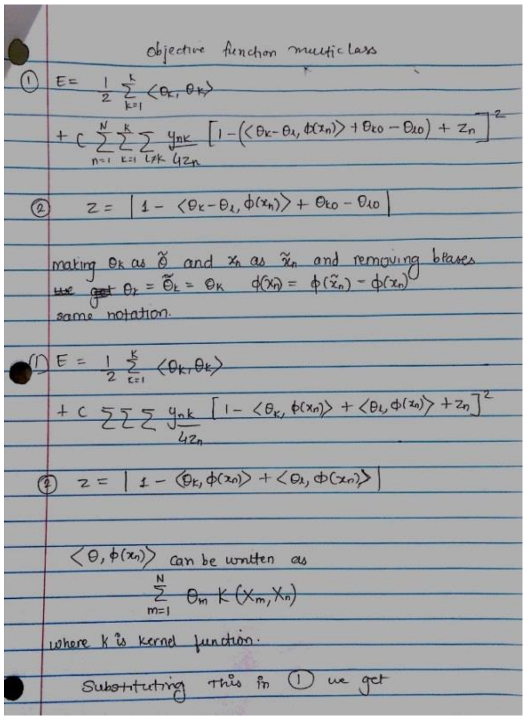
	DataSet1(Breast Cancer)	DataSet2(Handwriting)	DataSet3 (forests)
10% Training	4-5%	2-3%	16% error
30% Training	5-6%	2-3%	
50% Training	4-5%	2-3%	

Table (2) Kernel SVM

<u>III] Multi Class SVM:</u>

This is our version of Multi Class Kernel Support Vector Machine using Gaussian Kernel built from scratch (in octave). For the multiclass kernel Support Vector Machine we use a majorisation approach to optimize the non-smooth hinge loss objective function. To implement multiclass SVM for K classes, we use the One-versus-All method, where we discriminate between each class and every other class. We minimize the following majorised objective function with respect to Θ_k, where Θ_k is the Θ vector for the k^{th} class, to find out update equations for the majorisation variable Z and Θ_k and solve alternatively till convergence to get a minimum Θ. We'll get a Θ vector for every class.

The Math:



objective function multiclass

(1) $E = \frac{1}{2}\sum_{k=1}^{k} \langle \theta_k, \theta_k \rangle$

$+ C \sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{l \neq k} \frac{y_{nk}}{4z_n}\left[1 - \left(\langle \theta_k - \theta_l, \phi(x_n) \rangle + \theta_{k0} - \theta_{l0}\right) + z_n\right]^2$

(2) $z = \left| 1 - \langle \theta_k - \theta_l, \phi(x_n) \rangle + \theta_{k0} - \theta_{l0} \right|$

making θ_k as $\tilde{\theta}$ and x_n as $\tilde{x}_n$ and removing biases ~~the~~ ~~get~~ $\theta_k = \tilde{\theta}_k = \theta_k$ $\phi(x_n) = \phi(\tilde{x}_n) - \phi(x_n)$ same notation.

(1) $E = \frac{1}{2}\sum_{k=1}^{K} \langle \theta_k, \theta_k \rangle$

$+ C \sum\sum\sum \frac{y_{nk}}{4z_n}\left[1 - \langle \theta_k, \phi(x_n) \rangle + \langle \theta_l, \phi(x_n) \rangle + z_n\right]^2$

(2) $z = \left| 1 - \langle \theta_k, \phi(x_n) \rangle + \langle \theta_l, \phi(x_n) \rangle \right|$

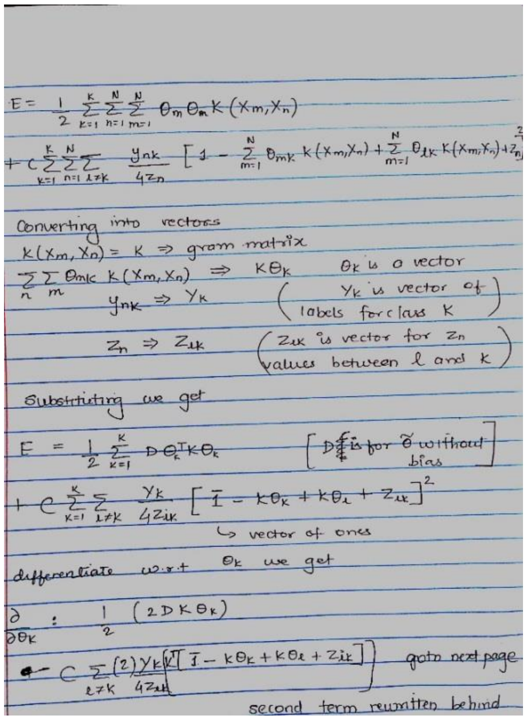
$\langle \theta, \phi(x_n) \rangle$ can be written as

$$\sum_{m=1}^{N} \theta_m k(x_m, x_n)$$

where k is kernel function.

Substituting this in (1) we get

Fig. (1(a)) Math of KSVM



$E = \frac{1}{2}\sum_{k=1}^{K}\sum_{n=1}^{N}\sum_{m=1}^{N} \theta_m \theta_m k(x_m, x_n)$

$+ C \sum_{k=1}^{K}\sum_{n=1}^{N}\sum_{l \neq k} \frac{y_{nk}}{4z_n}\left[1 - \sum_{m=1}^{N} \theta_{mk} k(x_m, x_n) + \sum_{m=1}^{N} \theta_{lk} k(x_m, x_n) + z_n\right]^2$

Converting into vectors

$k(x_m, x_n) = K \Rightarrow$ gram matrix

$\sum_n \sum_m \theta_{mk} k(x_m, x_n) \Rightarrow K\theta_k$ θ_k is a vector

$y_{nk} \Rightarrow Y_k$ (Y_k is vector of labels for class k)

$z_n \Rightarrow Z_{lk}$ (Z_{lk} is vector for z_n values between l and k)

Substituting we get

$E = \frac{1}{2}\sum_{k=1}^{K} D\theta_k^T K \theta_k$ [D is for $\tilde{\theta}$ without bias]

$+ C \sum_{k=1}^{K}\sum_{l \neq k} \frac{Y_k}{4Z_{lk}}\left[\bar{I} - K\theta_k + K\theta_l + Z_{lk}\right]^2$

↳ vector of ones

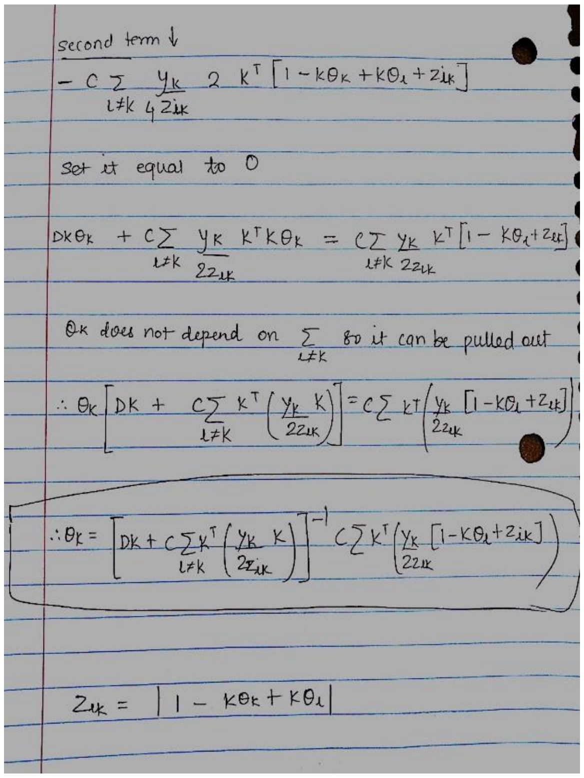
differentiate w.r.t θ_k we get

$\frac{\partial}{\partial \theta_k} : \frac{1}{2}(2DK\theta_k)$

$- C \sum_{l \neq k} \frac{(2)Y_k}{4Z_{lk}}\left[K^T\left[\bar{I} - K\theta_k + K\theta_l + Z_{lk}\right]\right]$ goto next page

second term rewritten behind

Fig. (1(b)) Math of KSVM



second term ↓

$- C \sum_{l \neq k} \frac{y_k}{4Z_{lk}} 2 K^T \left[1 - K\theta_k + K\theta_l + Z_{lk}\right]$

Set it equal to 0

$DK\theta_k + C\sum_{l \neq k} \frac{y_k}{2Z_{lk}} K^T K \theta_k = C\sum_{l \neq k} \frac{y_k}{2Z_{lk}} K^T \left[1 - K\theta_l + Z_{lk}\right]$

θ_k does not depend on $\sum_{l \neq k}$ so it can be pulled out

$\therefore \theta_k \left[DK + C\sum_{l \neq k} K^T \left(\frac{y_k}{2Z_{lk}} K\right)\right] = C\sum K^T \left(\frac{y_k}{2Z_{lk}}\left[1 - K\theta_l + Z_{lk}\right]\right)$

$$\boxed{\therefore \theta_k = \left[DK + C\sum_{l \neq k} K^T \left(\frac{y_k}{2Z_{lk}} K\right)\right]^{-1} C\sum K^T \left(\frac{y_k}{2Z_{lk}}\left[1 - K\theta_l + Z_{lk}\right]\right)}$$

$Z_{lk} = \left| 1 - K\theta_k + K\theta_l \right|$

Fig. (1(c)) Math of KSVM

Thus after differentiating the majorised objective function and setting it equal to zero we get the update equations for Θ_k and Z_{lk} as shown in the Math pictures above. We use the above update equations to alternatively solve for Z and Θ till convergence, and thus obtain an optimum Θ which is our multiclass *k*SVM machine.

The Code:
Octave Implementation of Multiclass Kernel Support Vector Machine.

Function Descriptions:

Defn: No.of classes=K, No.of training set patterns=N, No.of test set patterns=P, Dimension of patterns=D.
Dimensions: Θ_k (Nx1), Θ matrix (NxK), Y matrix (NxK), X matrix (NxD), testY (PxK), testX(PxD), K gram matrix (NxN), Z (KxNxK) explained below.
(see data description for information on data)

[X, Y, testX, testY] = extract1(percent):
This function extracts the Breast Cancer dataset into Training and testing sets X and testX, and training and testing class labels Y and testY. Takes an argument percent that partitions the dataset into training and testing using the given percentage. The patterns taken in the training data are picked at random using randperm.

[X, Y, testX, testY] = extract2(percent):
This function extracts the Handwriting recognition dataset into Training and testing sets in the same way as extract1.

[X, Y, testX, testY] = extract3():
This function extracts the Forests dataset into Training and testing sets that are already given for this data, no need to partition. The class labels are d=1, h=2, o=3, s=4.

K = gram(X):
Constructs the kernel Gram matrix from giving training set patterns X. We use the Gaussian kernel for our *k*SVM implementation. Matrix K is NxN.

T = linear(K,C,Y):
This initializes the Θ values for each class using the kernel Least Squares solution. The equation for Θ in for the multiclass kernel Least Squares is
$\Theta = (K + CI)^{-1}Y$
Where Θ is a matrix and the kth column corresponds to the Θ vector of the kth class (Θ_k). Y is also an NxK matrix where the kth column corresponds to the class labels of the kth class. If an nth pattern belongs to a class A, its value in Y is 1 for class A and 0 for every other class. Thus Y is an NxK matrix of 0s and 1s. C is a regularization parameter.

Z = zee(K,T):
This updates the Z values using a given Θ matrix (alternate between Z and Θ solutions). We have a Z vector between every class k and all other classes l which are not equal to k. An Nx1 vector Z_{kl} where k and l both range from 1 to K. The Z we use in the octave code is a 3D construct of dimensions KxNxK, one can imagine it as a cuboid, where we have K^2 vectors, and each vector is Nx1. If l=k the corresponding Z vector contains all zeros. For example to get the Nx1 Z_{12} vector (Z

vector between class 1 and class 2) we get it by Z(1,:,2). So this function updates the Z construct (or the Z_{lk} vectors) from the gram matrix and the Θ matrix using the update equation.

newt = theta(Y,K,Z,T,C,L):
This function uses the multiclass *k*SVM update function for Θ which we derived earlier to update the Θ_k vectors (or the Θ matrix) to give newt (new theta). It takes the multiclass labels Y matrix, gram matrix K, Z matrix and the current Θ matrix "t" (old theta). It also takes a regularization parameter C and another parameter L which is used to lift the eigenvalues of the matrix inverse term in the Θ update equation, so that they are all greater than 0, and the matrix becomes invertible.

T=SVM(X,Y,C,L):
This function runs the entire *k*SVM algorithm by using all the above helper functions. It takes as input the training set (X,Y) and the parameters (C,L).
<u>Algorithm:</u>

1. It first constructs the gram matrix from X.
2. Then it calls the function linear to initialize the Θ matrix with the linear least squares solution for Θ.
3. Then it calls the function zee to initialize the Z vectors.
4. Then it calls the function theta to get the kSVM solution for the Θ matrix from Z.
5. Calls zee and theta repeatedly to alternate between solutions for Z and Θ matrix, till convergence. (We repeat 10 times because we found that convergence is reached in 6-7 iterations).
6. Returns the final machine, Θ matrix (T).

R = test(T, X, testX, testY):
This function uses the machine on the input testX patterns to get the result matrix R which contains the class labels. Every row of R contains some numbers, we find the max of these numbers and set it equal to 1 and we set the rest of the row to 0. Which means that pattern(row) belongs to the class which corresponds to the column that contains a 1. Now the format of R matrix becomes exactly like the testY matrix, which is a PxK (P is the number of test patterns) matrix of 1s and 0s, where each column represents the class label vector for its corresponding class. A 1 denotes membership to a class and 0 denotes not a member. We then find error by calculating the no.of rows in R and testY that do not match. We divide this number by total number of rows (patterns) in testY and multiply it by 100 to get percentage error as output.

<u>The Code File kSVM.m:</u>

```
1;

###################### LOAD FUNCTIONS ##################################

function [X,Y,TX,TY] = extract3()

c=[0,0,197,27];
X=csvread("training",c);

Y=zeros(198,4);
for i=1:rows(X)
```

```
if(X(i,1)==1)
Y(i,1)=1;
endif
if(X(i,1)==2)
Y(i,2)=1;
endif
if(X(i,1)==3)
Y(i,3)=1;
endif
if(X(i,1)==4)
Y(i,4)=1;
endif
endfor
X(:,1)=[];

c=[0,0,324,27];
TX=csvread("testing",c);

TY=zeros(325,4);
for i=1:rows(TX)
if(TX(i,1)==1)
TY(i,1)=1;
endif
if(TX(i,1)==2)
TY(i,2)=1;
endif
if(TX(i,1)==3)
TY(i,3)=1;
endif
if(TX(i,1)==4)
TY(i,4)=1;
endif
endfor
TX(:,1)=[];
endfunction

function [X,Y,testX,testY] = extract2(percent)

n=(percent/100)*1796;
c=[0,1,1796,64];
X=csvread("data2",c);

Y=zeros(1796,10);
for i=1:rows(X)
if(X(i,64)==0)
Y(i,1)=1;
endif
if(X(i,64)==1)
Y(i,2)=1;
endif
if(X(i,64)==2)
Y(i,3)=1;
endif
if(X(i,64)==3)
Y(i,4)=1;
endif
if(X(i,64)==4)
Y(i,5)=1;
endif
if(X(i,64)==5)
Y(i,6)=1;
endif
if(X(i,64)==6)
Y(i,7)=1;
```

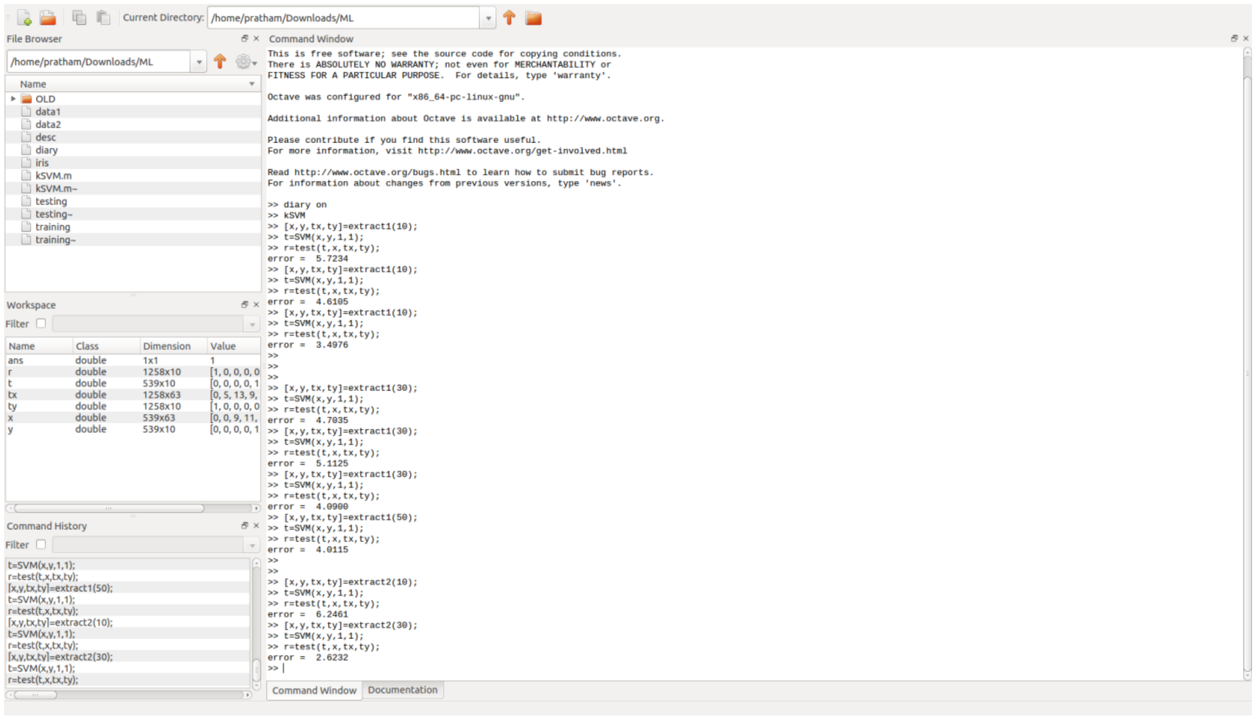
```
endif
if(X(i,64)==7)
Y(i,8)=1;
endif
if(X(i,64)==8)
Y(i,9)=1;
endif
if(X(i,64)==9)
Y(i,10)=1;
endif
endfor

X(:,64)=[];
id=randperm(1796,int32(n));
x1=X(id,:);
X(id,:)=[];
y1=Y(id,:);
Y(id,:)=[];

testX=X;
testY=Y;
X=x1;
Y=y1;
endfunction

function [X,Y,testX,testY] = extract1(percent)

n=(percent/100)*699;
c=[0,1,698,10];
X=csvread("data1",c);

Y=zeros(699,2);
for i=1:rows(X)
if(X(i,10)==2)
Y(i,1)=1;
else
Y(i,2)=1;
endif
endfor

X(:,10)=[];
id=randperm(699,int32(n));
x1=X(id,:);
X(id,:)=[];
y1=Y(id,:);
Y(id,:)=[];

testX=X;
testY=Y;
X=x1;
Y=y1;
endfunction

#################### SUPPORT VECTOR MACHINE ############################

function K=gram(X)
for i=1:rows(X)
for j=1:rows(X)
K(i,j)=exp(-(((norm(X(i,:)-X(j,:)))^2)/2));
endfor
endfor
endfunction

function t=linear(K,C,Y)
```

```
A=inv(K+C*eye(rows(K)));
t=A*Y;
endfunction

function z=zee(K,t)

I=ones(rows(K),1);
O=zeros(rows(K),1);
for i=1:columns(t)
for j=1:columns(t)
if(i==j)
z(i,:,j)=O;
else
z(i,:,j)=abs(I-K*t(:,i)+K*t(:,j));
endif
endfor
endfor
endfunction

function newt=theta(Y,K,Z,t,C,L)

newt=t;
for k=1:columns(t)

tmp=0;
tmp2=0;
D=eye(rows(K));
D(rows(D),rows(D))=0;
sum=zeros(rows(K));
sum2=zeros(rows(K),1);

for l=1:columns(t)
if(k==l)
continue;
else
for i=1:rows(K)
A1(i,:)=(Y(i,k)/(2*Z(k,i,l)))*K(i,:);
endfor
A2=K'*A1;
sum+=A2;
endif
endfor
sum=C*sum;
tmp=K + sum + L*eye(rows(K));

tmp=inv(tmp);

I=ones(rows(K),1);
for l=1:columns(t)
if(k==l)
continue;
else
A3=K*t(:,l);
A4=I+A3+Z(k,:,l)';
for j=1:rows(K)
A5(j,:)=(Y(j,k)/(2*Z(k,j,l)))*A4(j,:);
endfor
A6=K'*A5;
sum2+=A6;
endif
endfor
sum2=C*sum2;
tmp2=sum2;
```

```
newt(:,k)=tmp*tmp2;
endfor
endfunction

function T=SVM(X,Y,C,L)
k=gram(X);
t=linear(k,1,Y);
z=zee(k,t);
nt=theta(Y,k,z,t,C,L);
for i=1:9
z=zee(k,nt);
nt=theta(Y,k,z,nt,C,L);
endfor
T=nt;
endfunction

########################### TEST FUNCTION ################################

function R=test(t,X,tx,ty)
for j=1:rows(tx)
for i=1:rows(X)
u(i,1)=exp(-(((norm(X(i,:)-tx(j,:)))^2)/2));
endfor
f=u'*t;
R(j,:)=f;
endfor
error=0;
for a=1:rows(R)
max=1;
for b=1:columns(R)
if(R(a,b)>R(a,max))
max=b;
endif
endfor
R(a,:)=0;
R(a,max)=1;
if(ty(a,max)!=1)
error+=1;
endif
endfor
error=(error/rows(ty))*100;
error
endfunction
```

The Tests:

Multiple tests were run to fine tune performance using different values of parameters and changing the octave functions. Training data was selected as 10%, 30% and 50% of all data. Except in the case of forests data in which we already had separated training and test sets. After fine tuning the machine here we display the latest results of our optimized machine. We run random tests for 10%, 30% and 50% data for dataset 1(Breast Cancer) and dataset 2(Handwriting) using extract1 and extract2.

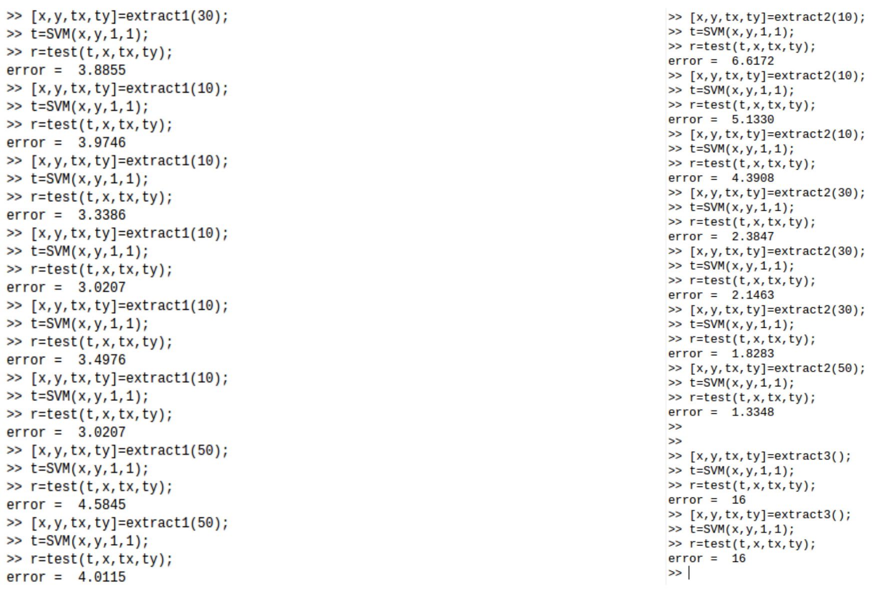


Name	Class	Dimension	Value
ans	double	1x1	1
r	double	1258x10	[1, 0, 0, 0, 0
t	double	539x10	[0, 0, 0, 0, 1
tx	double	1258x63	[0, 5, 13, 9,
ty	double	1258x10	[1, 0, 0, 0, 0
x	double	539x63	[0, 0, 9, 11,
y	double	539x10	[0, 0, 0, 0, 1

```
t=SVM(x,y,1,1);
r=test(t,x,tx,ty);
[x,y,tx,ty]=extract1(50);
t=SVM(x,y,1,1);
r=test(t,x,tx,ty);
[x,y,tx,ty]=extract2(10);
t=SVM(x,y,1,1);
r=test(t,x,tx,ty);
[x,y,tx,ty]=extract2(30);
t=SVM(x,y,1,1);
r=test(t,x,tx,ty);
```

```
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> diary on
>> kSVM
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  5.7234
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.6105
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.4976
>>
>>
>>
>> [x,y,tx,ty]=extract1(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.7035
>> [x,y,tx,ty]=extract1(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  5.1125
>> [x,y,tx,ty]=extract1(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.0900
>> [x,y,tx,ty]=extract1(50);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.0115
>>
>>
>> [x,y,tx,ty]=extract2(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  6.2461
>> [x,y,tx,ty]=extract2(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  2.6232
>> |
```

Fig. (2 (a)) Result of KSVM



```
>> [x,y,tx,ty]=extract1(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.8855
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.9746
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.3386
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.0207
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.4976
>> [x,y,tx,ty]=extract1(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  3.0207
>> [x,y,tx,ty]=extract1(50);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.5845
>> [x,y,tx,ty]=extract1(50);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.0115
```

```
>> [x,y,tx,ty]=extract2(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  6.6172
>> [x,y,tx,ty]=extract2(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  5.1330
>> [x,y,tx,ty]=extract2(10);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  4.3908
>> [x,y,tx,ty]=extract2(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  2.3847
>> [x,y,tx,ty]=extract2(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  2.1463
>> [x,y,tx,ty]=extract2(30);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  1.8283
>> [x,y,tx,ty]=extract2(50);
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  1.3348
>>
>>
>> [x,y,tx,ty]=extract3();
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  16
>> [x,y,tx,ty]=extract3();
>> t=SVM(x,y,1,1);
>> r=test(t,x,tx,ty);
error =  16
>> |
```

Fig. (2 (b)) Results of the final tests run on Octave.

The Outcome:

	DataSet1(Breast Cancer)	DataSet2(Handwriting)	DataSet3 (forests)
10% Training	3-5%	4-6%	16% error
30% Training	4-5%	1.5-2.5%	
50% Training	4-5%	1-2%	

The above table shows the error values for each dataset with different percentages of training data. For DataSet 3 we get a high error of 16% on the given training and test sets. While on the other two datasets we do quite well with our majorised version of multiclass kernel SVM.

Comparison with libSVM

As one can see by comparing our SVM with the libSVM, we find that our SVM performs almost as well as the libSVM version, although the computation is expensive because of the matrix inverse. Our SVM does not scale well, because for large data the computation time is too much. Except for the third dataset which gives 16% error but same error is given by the libSVM version.

IV] Random Forests:

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest). Random forests does not overfit.

To implement Random Forest algorithm we used the RandomForest Classifier library in Scikit. We had to choose an optimum value for n_estimators parameter, which is the no.of trees in the Random Forest. If the no.of trees is too less then we will have less votes, and hence there will be poor generalization. On the other hand if the no.of trees is too large the program takes too long to run. Thus we experimented with different number of trees, and found that increasing the number of trees beyond 100 does not give much difference in accuracy, but increases the run time a lot. Therefore we set n_estimators to 100.

Upon using k-fold cross validation, we were able to obtain better results from the random forest algorithm.

Results:

	DataSet1(Breast Cancer)	DataSet2(Handwriting)	DataSet3 (forests)
10% Training	5-6%	2.5-3.5%	18-19%
30% Training	2.5-3.5%	2-3%	
50% Training	2-3%	2-3%	

Table(3)

V] Deep Learning:

a) CNN:

In CNN, spatially located correlation is exploited by enforcing a local connectivity pattern between neurons of adjacent layers. In CNNs, each filter hi is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map.

Implementation of CNN in Theano

ConvOp is the main workhorse for implementing a convolutional layer in Theano. ConvOp is used by theano.tensor.signal.conv2d, which takes two symbolic inputs:
• 4D tensor corresponding to a mini-batch of input images. The shape of the tensor is as follows: [mini-batch size, number of input feature maps, image height, image width].
• 4D tensor corresponding to the weight matrix W. The shape of the tensor is: [number of feature maps at layer m, number of feature maps at layer m-1, filter height, filter width]
By eliminating non-maximal values, it reduces computation for upper layers.
It provides a form of translation invariance. Imagine cascading a max-pooling layer with a convolutional layer. There are 8 directions in which one can translate the input image by a single pixel. If max-pooling is done over a 2x2 region, 3 out of these 8 possible configurations will produce exactly the same output at the convolutional layer. For max-pooling over a 3x3 window, this jumps to 5/8. Since it provides additional robustness to position, max-pooling is a "smart" way of reducing the dimensionality of intermediate representations. Max-pooling is done in Theano by way of theano.tensor.signal.downsample.max_pool_2d. This function takes as input an N dimensional tensor (where N >= 2) and a downscaling factor and performs max-pooling over the 2 trailing dimensions of the tensor.

Parameters selected for Image File:
As the image is described by 64 input values, i.e. 8*8 we have used only layer and filter of size 3,3. The pooling sizing chosen is 2*2. filtering reduces the image size to (8-3+1,8-3+1)=(6,6) maxpooling reduces this further to (6/2,6/2) = (3,3). This reduces the image to size 3*3.
MLP operates on it operates on 2D matrices of flattened values. We construct a fully-connected sigmoidal layer using MLP and classify the values of the fully-connected sigmoidal layer using Logistic Regression

Results:

Batch Size=100
Epoch=50

Dataset 2: Handwriting Recognition

Percentage	Validation Error Rate	Test Error Rate
10	4.712	4.632
50	4.515	4.23
70	3.8	3.6

b) MLP:

MLP in Theano uses Logistic Regression with hidden layer output as its input. So first we implemented Logistic regression layer. The program takes pickled file as input and this was our first encounter with pickled files. *Pickling* is the common term among Python programmers for serialization. Pickle file is in the following format: training, validation and testing. While implementing MLP, we tried different configurations by varying number of epochs and batch size. Number of epochs is the iteration for which the classifier runs on the training data. We observed that after a particular value of epoch, test error rate starts increasing. This might be because of overfitting of training data

Batch Size=10
Epoch=130

Dataset 1: Breast Cancer

Percentage	Validation Error Rate	Test Error Rate
10	4.210526	1.428571
50	3.88	1.328571
70	2.4	1.265

Dataset 2: Handwriting Recognition

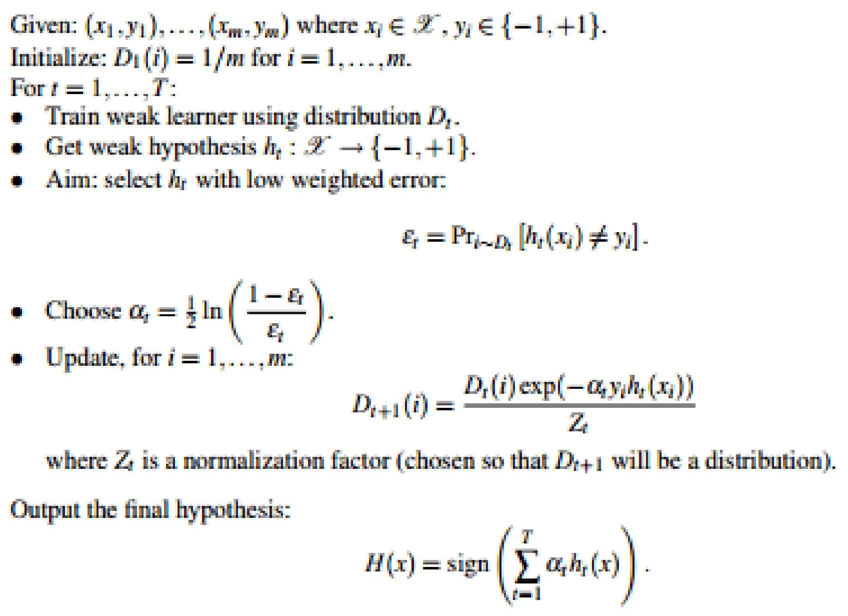
Percentage	Validation Error Rate	Test Error Rate
10	3.98751	4.5
50	3.88	4.324
70	2.19	4.82

Dataset 3: Forest

Percentage	Validation Error Rate	Test Error Rate
10	10	20
50	9.166667	13.846154
70	13.333333	13.571429

VI] AdaBoost:

AdaBoost is a type of "Ensemble Learning" where multiple learners are employed to build a stronger learning algorithm. AdaBoost works by choosing a base algorithm (e.g. decision trees) and iteratively improving it by accounting for the incorrectly classified examples in the training set.
We assign equal weights to all the training examples and choose a base algorithm. At each step of iteration, we apply the base algorithm to the training set and increase the weights of the incorrectly classified examples. We iterate n times, each time applying base learner on the training set with updated weights. The final model is the weighted sum of the n learners.
AdaBoost is extremely successful machine learning method and Schapire and Freund won the Godel Prize in 2003 for their construction of AdaBoost algorithm.



```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:
  • Train weak learner using distribution $D_t$.
  • Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
  • Aim: select $h_t$ with low weighted error:
        $\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
  • Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
  • Update, for $i = 1, \ldots, m$:
        $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
Output the final hypothesis:
        $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

Fig. () Adaboost algorithm

Result:

	DataSet1(Breast Cancer)	DataSet2(Handwriting)	DataSet3 (forests)
10% Training	4-8%	30-31%	
30% Training	3-5%	30-31%	17.53%
50% Training	3-4%	30-31%	

VII] Conclusion:

- Thus, we have presented several supervised learning algorithms for the three given datasets.
- For SVM, our own kernel SVM and the libSVM version both give almost similar error on all 3 datasets. For first 2 we get 2-5% error, and for the third one we get 16%.
- On the other hand we get comparable results using Random Forests. Almost same results, but random forests is a much simpler algorithm and is computationally less expensive than the majorised kernel SVM.
- We get slightly better results in some cases using deep learning, but the results are still comparable.
- We think that Random Forests was the best supervised learning algorithm we implemented. Very easy to implement and computationally less expensive.