Experiment-8

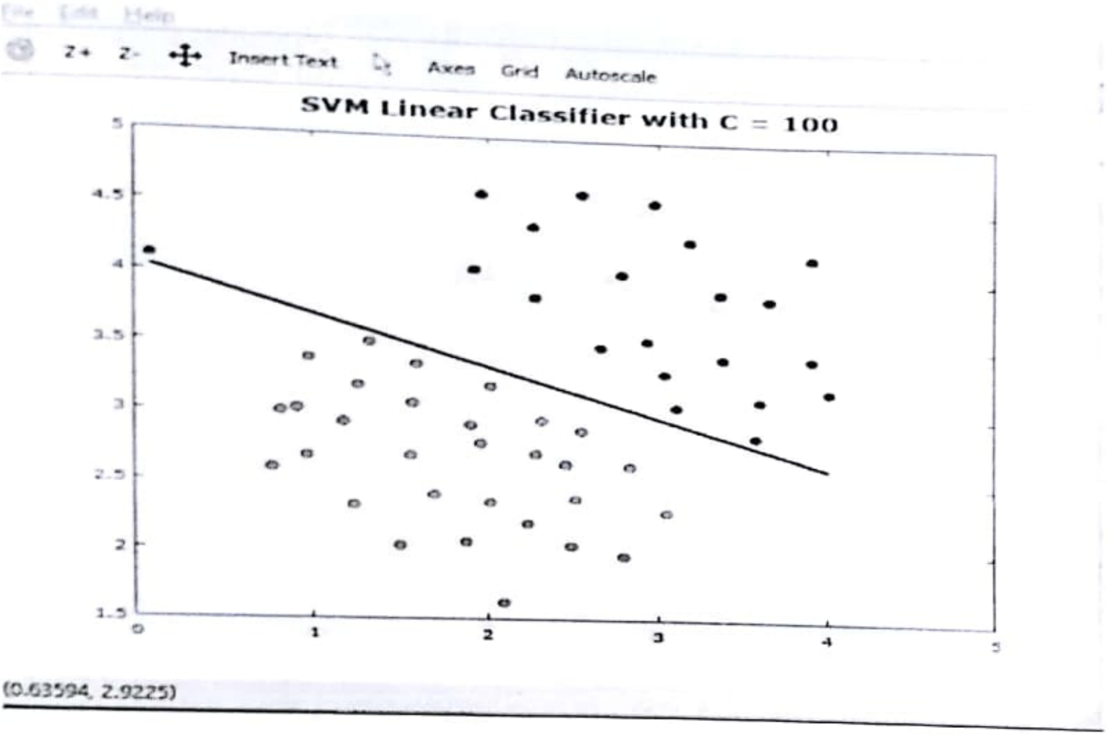
Aim: Classify dataset using SVM's.

Code:

```
% SVM Linear classification
% A 2-feature example

clear all; close all;

% Load training features and labels
[y, x] = libsvmread('twofeature.txt');

% Set the cost
C = 100;

% Train the model and get the primal variables w, b from the
model
% Libsvm options
% -s 0 : classification
% -t 0 : linear kernel
% -c somenumber : set the cost
model = svmtrain(y, x, sprintf('-s 0 -t 0 -c %g', C));
w = model.SVs' * model.sv_coef;
b = -model.rho;
if (model.Label(1) == -1)
    w = -w; b = -b;
end

% Plot the data points
figure
pos = find(y == 1);
neg = find(y == -1);
plot(x(pos,1), x(pos,2), 'ko', 'MarkerFaceColor', 'b'); hold on;
plot(x(neg,1), x(neg,2), 'ko', 'MarkerFaceColor', 'g')

% Plot the decision boundary
plot_x = linspace(min(x(:,1)), max(x(:,1)), 30);
plot_y = (-1/w(2))*(w(1)*plot_x + b);
plot(plot_x, plot_y, 'k-', 'LineWidth', 2)
```

```
title(sprintf('SVM Linear Classifier with C = %g', C),
'FontSize', 14)
```

Output:



Experiment-9

Aim: Classify dataset using any two kernel methods in SVM's.

Code:

```
%% Train a Support Vector Machine Classifier
%%
% Load Fisher's iris data set. Remove the sepal lengths and
widths, and all
% observed setosa irises.

% Copyright 2015 The MathWorks, Inc.

load fisheriris
inds = ~strcmp(species,'setosa');
X = meas(inds,3:4);
y = species(inds);
%%
% Train an SVM classifier using the processed data set.
SVMModel = fitcsvm(X,y,'KernelFunction','polynomial');
%%
% The Command Window shows that |SVMModel| is a trained
|ClassificationSVM|
% classifier and a property list.  Display the
% properties of |SVMModel|, for example, to determine the class
order, by using
% dot notation.
classOrder = SVMModel.ClassNames
%%
% The first class (|'versicolor'|) is the negative class, and
the second
% (|'virginica'|) is the positive class.  You can change the
class order
% during training by using the |'ClassNames'| name-value pair
argument.
%%
% Plot a scatter diagram of the data and circle the support
vectors.
sv = SVMModel.SupportVectors;
figure
gscatter(X(:,1),X(:,2),y)
```

```
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('versicolor','virginica','Support Vector');
hold off;
%%
% The support vectors are observations that occur on or beyond
their
% estimated class boundaries.
%%
% You can adjust the boundaries (and therefore the number of
support
% vectors) by setting a box constraint during training using the
% |'BoxConstraint'| name-value pair argument.
```

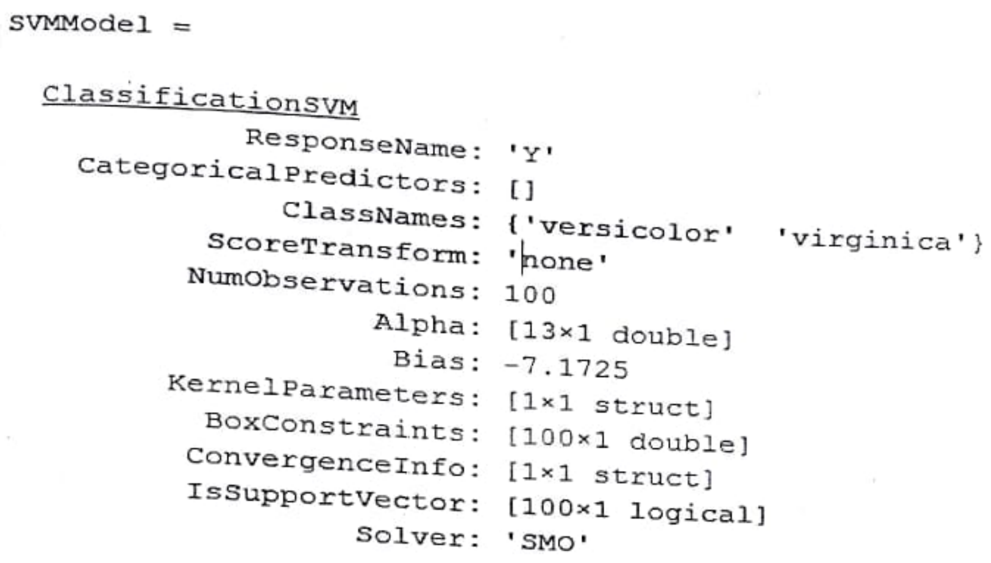
```
SVMModel
SVMModel =

  ClassificationSVM
           ResponseName: 'Y'
  CategoricalPredictors: []
             ClassNames: {'versicolor'  'virginica'}
         ScoreTransform: 'none'
        NumObservations: 100
                  Alpha: [13×1 double]
                   Bias: -7.1725
       KernelParameters: [1×1 struct]
         BoxConstraints: [100×1 double]
        ConvergenceInfo: [1×1 struct]
        IsSupportVector: [100×1 logical]
                 Solver: 'SMO'
```

```
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('versicolor','virginica','Support Vector');
hold off;
%%
% The support vectors are observations that occur on or beyond
their
% estimated class boundaries.
%%
% You can adjust the boundaries (and therefore the number of
support
% vectors) by setting a box constraint during training using the
% |'BoxConstraint'| name-value pair argument.
```

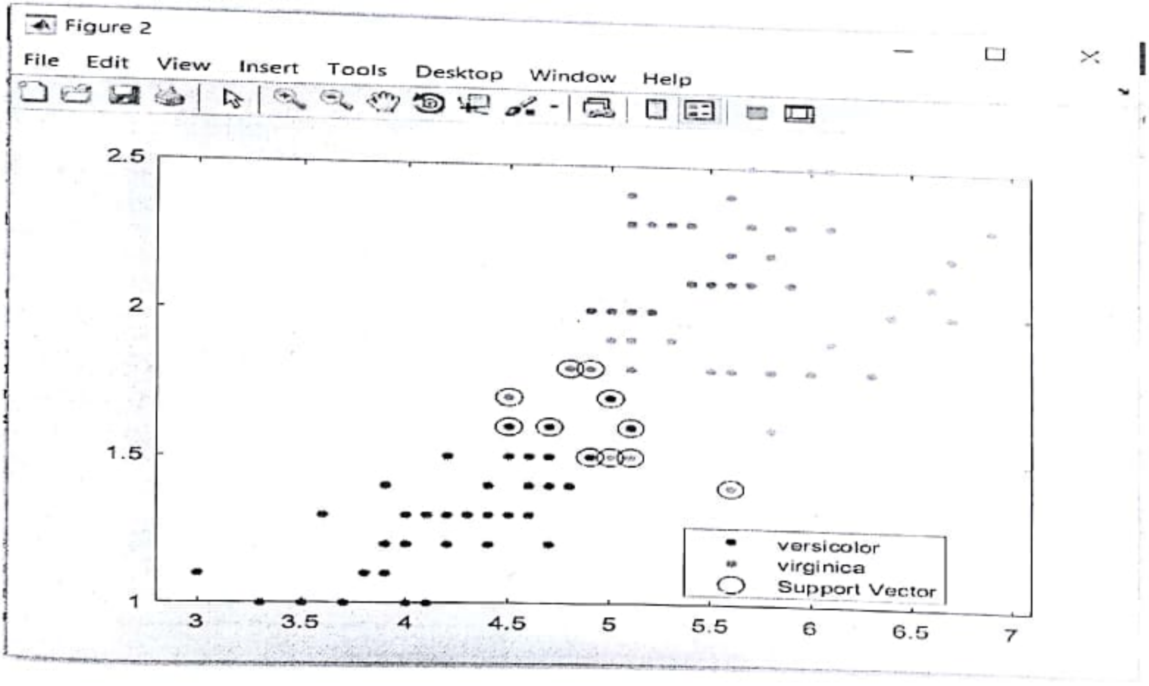
SVMModel

```
SVMModel =

  ClassificationSVM
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: {'versicolor'  'virginica'}
           ScoreTransform: 'none'
          NumObservations: 100
                    Alpha: [13×1 double]
                     Bias: -7.1725
         KernelParameters: [1×1 struct]
           BoxConstraints: [100×1 double]
          ConvergenceInfo: [1×1 struct]
          IsSupportVector: [100×1 logical]
                   Solver: 'SMO'
```

Output:



```
SVMModel =

  ClassificationSVM
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: {'versicolor'  'virginica'}
           ScoreTransform: 'none'
          NumObservations: 100
                    Alpha: [13×1 double]
                     Bias: -7.1725
         KernelParameters: [1×1 struct]
           BoxConstraints: [100×1 double]
          ConvergenceInfo: [1×1 struct]
          IsSupportVector: [100×1 logical]
                   Solver: 'SMO'
```



Code:

```
%% Train a Support Vector Machine Classifier
%%
% Load Fisher's iris data set. Remove the sepal lengths and
widths, and all
% observed setosa irises.

% Copyright 2015 The MathWorks, Inc.

load fisheriris
inds = ~strcmp(species,'setosa');
X = meas(inds,3:4);
y = species(inds);
%%
% Train an SVM classifier using the processed data set.
SVMModel = fitcsvm(X,y,'KernelFunction','rbf');
%%
% The Command Window shows that |SVMModel| is a trained
|ClassificationSVM|
% classifier and a property list.  Display the
% properties of |SVMModel|, for example, to determine the class
order, by using
% dot notation.
classOrder = SVMModel.ClassNames
%%
% The first class (|'versicolor'|) is the negative class, and
the second
% (|'virginica'|) is the positive class.  You can change the
class order
% during training by using the |'ClassNames'| name-value pair
argument.
%%
% Plot a scatter diagram of the data and circle the support
vectors.
sv = SVMModel.SupportVectors;
figure
gscatter(X(:,1),X(:,2),y)
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('versicolor','virginica','Support Vector');
hold off;
```

```
%%
% The support vectors are observations that occur on or beyond
their
% estimated class boundaries.
%%
% You can adjust the boundaries (and therefore the number of
support
% vectors) by setting a box constraint during training using the
% |'BoxConstraint'| name-value pair argument.
```

About SVMModel:

```
SVMModel =

  ClassificationSVM
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: {'versicolor'  'virginica'}
           ScoreTransform: 'none'
          NumObservations: 100
                    Alpha: [23×1 double]
                     Bias: 0.1507
         KernelParameters: [1×1 struct]
           BoxConstraints: [100×1 double]
          ConvergenceInfo: [1×1 struct]
          IsSupportVector: [100×1 logical]
                   Solver: 'SMO'
```