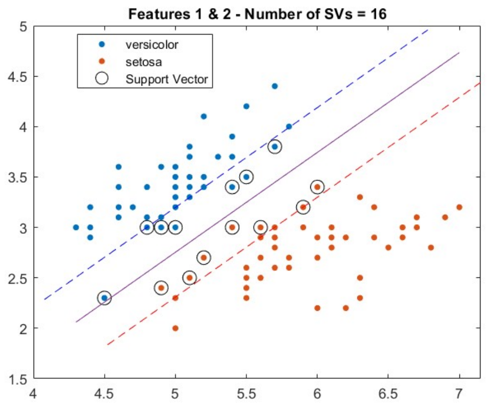
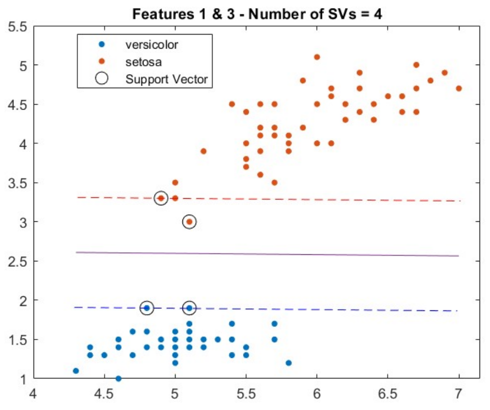
EE 583 Pattern Recognition HW3

Eda Özkaynar 2375582

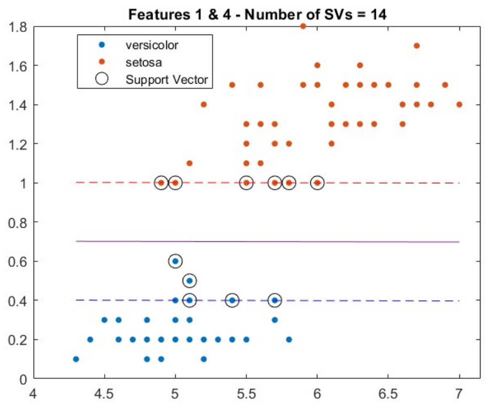
Question 1 Train SVM Classifier



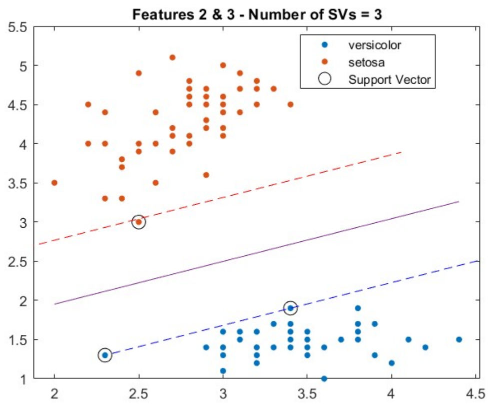
(a) Features 1 & 2 (Sepal Length vs. Sepal Width)



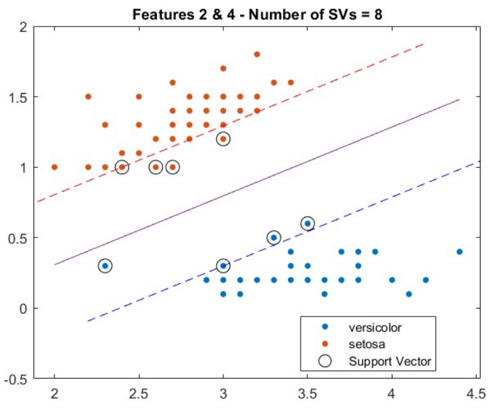
(b) Features 1 & 3 (Sepal Length vs. Petal Length)



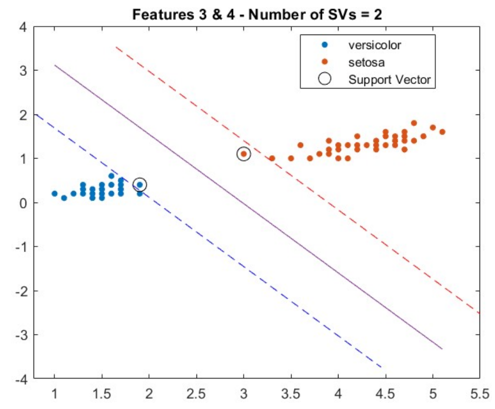
(c) Features 1 & 4 (Sepal Length vs. Petal Width)



(d) Features 2 & 3 (Sepal Width vs. Petal Length)



(e) Features 2 & 4 (Sepal Width vs. Petal Width)



(f) Features 3 & 4 (Petal Length vs. Petal Width)

Fig. 1: Training SVM Classifier with Different Feature Pairs

Figure1 shows the decision boundaries and support vectors of Support Vector Machine classifiers for different feature combinations. It can be observed that as the number of support vectors decreases, the separability of classes increases. For example, in Figure 1f, the classifier has the fewest support vectors and the best class separability. Despite this, in Figure 1a, the classifier has the largest number of support vectors and shows that these two feature alone do not have perfect discrimination to distinguish versicolor from setosa.

Question 2 Cross-validate SVM

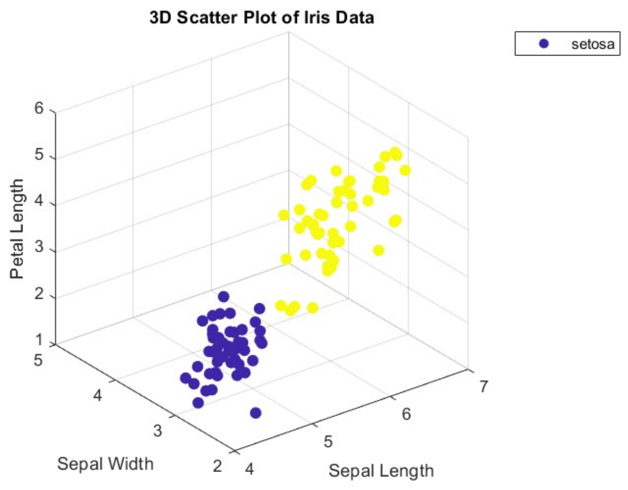


Fig. 2: 3D Scatter Plot of Iris Data for 3 features

A. Results

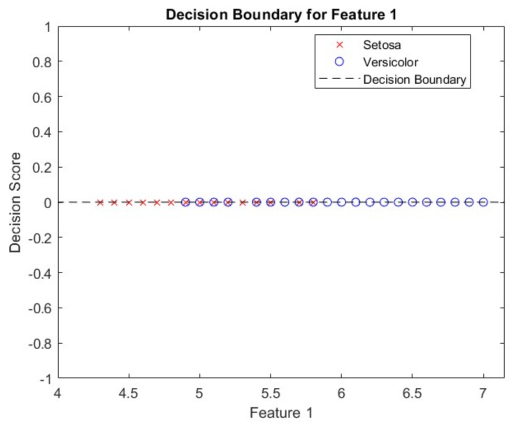
kfold loss = 0 , Leave-one-out loss = 0

The losses are both zero because using four features increases the dimension, making the data linearly separable. This can be observed in Figure 2, where 3D data is linearly separable.

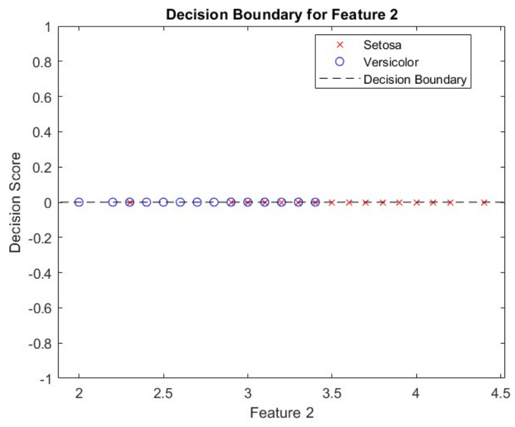
In k-fold cross-validation, we start by dividing our dataset into k equally sized subsets. We then apply the train-test method k times, ensuring that each time one of the k subsets serves as the test set while the remaining k-1 subsets are combined to form the training set. Finally, we estimate the model's performance by averaging the scores obtained from each of the k trials.

In leave-one-out (LOO) cross-validation, we train our machine learning model n times, where n is equal to the size of our dataset. In each iteration, only one sample is used as the test set, while the rest of the samples are used to train the model. It is important to note that LOO is a specific case of k-fold cross-validation where k is equal to n.

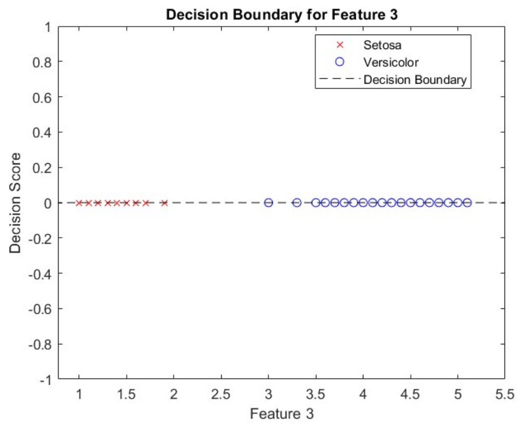
When the dataset is small, leave-one-out (LOO) cross-validation is more suitable because it utilizes more training samples in each iteration. This approach allows our model to learn better representations. On the other hand, for larger datasets, k-fold cross-validation is preferred. LOO involves training one model for each sample in the dataset, which can be time-consuming when there are many samples. Therefore, k-fold cross-validation is a more efficient choice for large datasets.



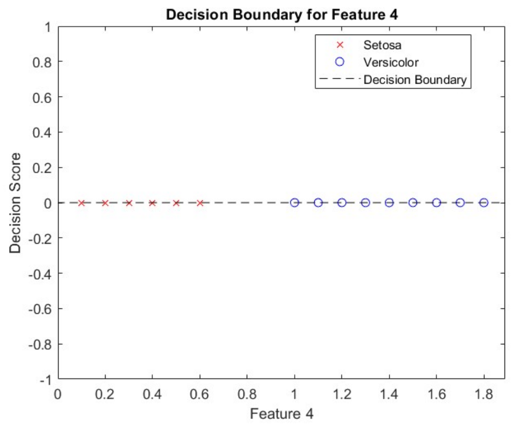
(a) Features 1 (Sepal Length)



(b) Features 2 (Sepal Width)



(c) Features 3 (Petal Length)



(d) Features 4 (Petal Width)

Fig. 3: Training SVM Classifier with Different Feature Pairs

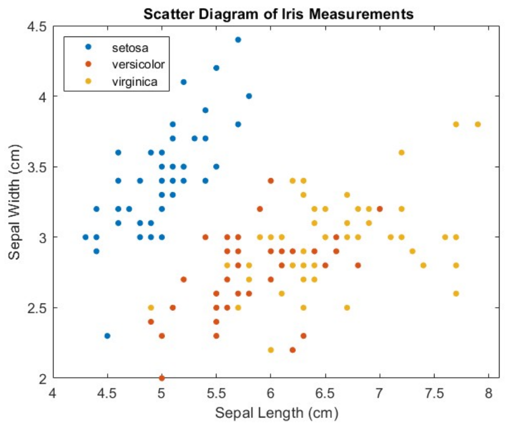
Feature	**10-Fold Error**	**LOOCV Error**
Feature 1	0.11	0.11
Feature 2	0.23	0.17
Feature 3	0	0
Feature 4	0	0

TABLE I: Cross-Validation Error for Each Feature

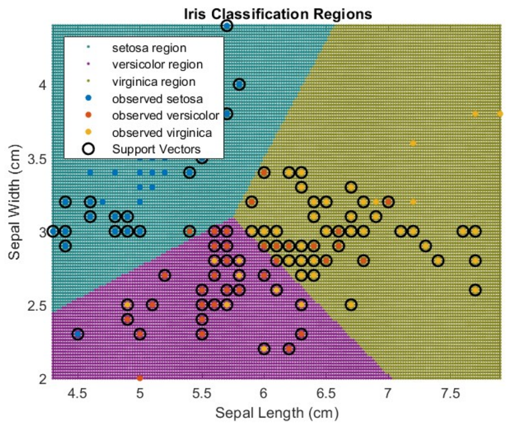
Both 10-fold cross-validation and Leave-One-Out Cross-Validation (LOOCV) result in zero errors for Feature 3 and Feature 4. This outcome implies that the classes are linearly separable when using either Petal Length or Petal Width individually. In contrast, Feature 1 and Feature 2 yield higher error rates, with 10-fold cross-validation errors of 0.11 and 0.23, respectively. This result indicates that Sepal Length and Sepal Width are not as effective at distinguishing the classes as the petal-based features.

In Feature 2, LOOCV has a lower error rate than 10-fold cross-validation. The one reason of this could be using LOOCV, the model sees nearly all data points for training in each iteration, allowing it to capture more information and generalize better for that one left-out data point. Moreover, LOOCV tends to have lower bias but higher variance than 10-fold cross-validation. However, in cases where the dataset is small, the almost-complete training set of LOOCV may help capture underlying patterns better, resulting in a lower observed error.

Question 3 Multi-class SVM



(a)



(b)

Fig. 4: Plots of scatter diagram and classification regions of dataset

In Figure 4a, we can observe that the setosa class is linearly separable from the others. The multi-class SVM is trained using a one-to-one approach, i.e. each binary classifier is trained to discriminate between two specific classes. In Figure 4b, scores were calculated for each grid point for each SVM model. Each score was assigned to the class with the highest score. The decision regions for setosa, versicolor and virginica are drawn in different colors. Since Sepal Length and Sepal Width do not completely separate the classes, the SVM classifier relies on more support vectors to define the decision boundaries. For example, the number of support vectors is high for versicolor and virginica. For versicolor and virginica, there are more misclassifications due to the overlapping regions.

Question 4 Non-parametric Density Estimation

Bayesian optimization is a method for optimizing an objective function that is costly to evaluate, especially when there is no closed-form expression or when gradients cannot be calculated. This technique is particularly preferred for hyperparameter tuning in machine learning, where evaluating the performance of a model (objective function) can be resource-intensive.

In this question, the objective function is designed to minimize the cross-validated misclassification rate. First, using the specified hyperparameters such as BoxConstraint and KernelScale, the SVM model was trained. Then, model performance was evaluated using 10-fold cross-validation and misclassification rate was calculated as the average error across all cross-validation folds. After the final iteration, Bayesian optimization returns the hyperparameters that achieved the lowest misclassification rate found during the search. In this example, SVM hyperparameters such as Box Constrain and Kernel Scale are optimized. The Box Constrain balances the trade-off between maximizing the margin and minimizing classification errors. Higher values cause the SVM model to prioritize reducing training errors, potentially resulting in overfitting. Lower values allow more margin violations, which may enhance generalization. C appears in the objective function as a weight for the penalty on misclassified points. The objective function of a soft-margin SVM is:

$$\text{Minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

subject to:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for all } i$$

$$\xi_i \geq 0 \quad \text{for all } i$$

where:
- $\mathbf{w}$ is the weight vector that defines the decision boundary.
- ξ_i (slack variables) represent the amount by which a data point violates the margin.
- y_i is the label of the i-th sample.
- $\mathbf{x}_i$ is the i-th feature vector.
- b is the bias term.

The other hyperparamater is Kernel Scale, it is defines the scale of the Gaussian (RBF) kernel. Smaller values make the decision boundary more flexible, while larger values make the decision boundary smoother. A smaller gamma value leads to a wider kernel, resulting in a simpler model with a smoother decision boundary. In contrast, a larger gamma value creates a narrower kernel, giving each data point more local influence. This behavior resembles that of the nearest neighbor algorithm. The effect of parameters can be observed in Figure 5

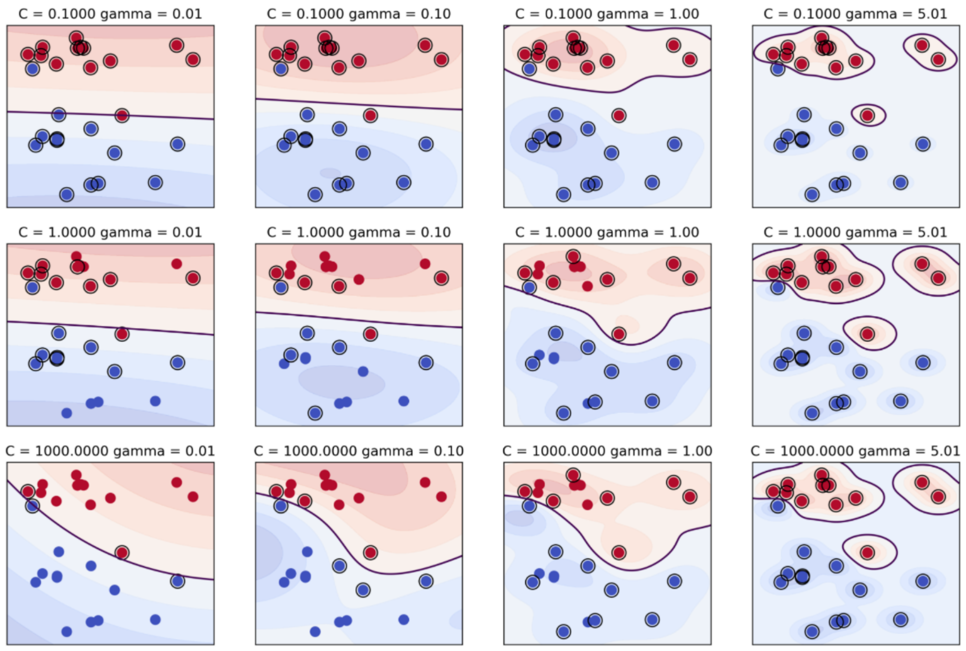


Fig. 5: SVM decision boundaries with varying C and γ values

```
clear; close all; clc;

%% Question 1%%

% Feature pairs (1,2)

% Load dataset X and ys
load fisheriris
% Choose setosa and versicolor
inds = ~strcmp(species,'virginica');
X = meas(inds,1:2);
y = species(inds);

% Feature Pairs (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)
feature_pairs = [1, 2; 1, 3; 1, 4; 2, 3; 2, 4; 3, 4];

% For each feature pairs
for i = 1:size(feature_pairs, 1)
    % Choose one pair
    X = meas(inds, feature_pairs(i, :));

    % Train SVM model
    SVMModel = fitcsvm(X, y);

    % Get SVs
    sv      = SVMModel.SupportVectors;
    beta    = SVMModel.Beta; % Linear predictor coefficients
    b       = SVMModel.Bias; % Bias term

    % Plot
    figure
    gscatter(X(:, 1), X(:, 2), y)
    hold on
    plot(sv(:, 1), sv(:, 2), 'ko', 'MarkerSize', 10)
    X1 = linspace(min(X(:,1)),max(X(:,1)),100);
    X2 = -(beta(1)/beta(2)*X1)-b/beta(2);
    plot(X1,X2,'-')
    m = 1/sqrt(beta(1)^2 + beta(2)^2);  % Margin half-width
    X1margin_low = X1+beta(1)*m^2;
    X2margin_low = X2+beta(2)*m^2;
    X1margin_high = X1-beta(1)*m^2;
    X2margin_high = X2-beta(2)*m^2;
    plot(X1margin_high,X2margin_high,'b--')
    plot(X1margin_low,X2margin_low,'r--')
    legend('versicolor', 'setosa', 'Support Vector')
    title("Features " + int2str(feature_pairs(i, 1)) + " & " + int2str↙
(feature_pairs(i, 2)) + ...
          " - Number of SVs = " + int2str(length(sv)))
    hold off
end
clear; close all; clc;
%% Question 2 - Cross-Validate SVM %%
```

```
% Load dataset
load fisheriris
% Choose setosa and versicolor
inds = ~strcmp(species,'virginica');
% Use all 4 features
X = meas(inds,:);
y = species(inds);

Y= categorical(species(inds)); % Convert species to categorical for coloring


SVMModel = fitcsvm(X,y,'Standardize',true,'KernelFunction','RBF',...
    'KernelScale','auto');

% 10-Fold Cross-Validation
CVSVMModel1 = crossval(SVMModel,'Kfold',10,'Leaveout','off');
classLoss = kfoldLoss(CVSVMModel1);

% Leave-One-Out Cross-Validation (LOOCV)
CVSVMModel2         = crossval(SVMModel,'Leaveout','on');
leave_one_out_cv    = kfoldLoss(CVSVMModel2);

% Initialize arrays to store cross-validation results
results = [];

% Loop through each feature individually
for i = 1:4
    % Select a single feature
    X_single = X(:, i);

    % Train SVM model with a single feature
    SVMModel = fitcsvm(X_single, y, 'Standardize', true, 'KernelFunction', 'RBF',↙
'KernelScale', 'auto');

    % 10-Fold Cross-Validation
    CVSVMModel1 = crossval(SVMModel, 'Kfold', 10);
    classLoss_10fold = kfoldLoss(CVSVMModel1);

    % Leave-One-Out Cross-Validation (LOOCV)
    CVSVMModel2 = crossval(SVMModel, 'Leaveout', 'on');
    classLoss_LOOCV = kfoldLoss(CVSVMModel2);

    % Store the results
    results = [results; {['Feature ', num2str(i)], classLoss_10fold,↙
classLoss_LOOCV}];
end

% Display results
resultsTable = cell2table(results, 'VariableNames', {'Feature', '10FoldError',↙
'LOOCVError'});
disp(resultsTable);
y_categorical = categorical(y);
```

```
% Loop through each feature individually
for i = 1:4
    % Select a single feature
    X_single = X(:, i);

    % Train SVM model with a single feature
    SVMModel = fitcsvm(X_single, y, 'Standardize', true, 'KernelFunction', 'RBF',↙
'KernelScale', 'auto');

    % Create a range of values for plotting decision boundaries
    x_min = min(X_single) - 1;
    x_max = max(X_single) + 1;
    x_range = linspace(x_min, x_max, 100)';

    % Predict scores over the range to find the decision boundary
    [~, scores] = predict(SVMModel, x_range);

    % Plot the data and the decision boundary
    figure
    gscatter(X_single, zeros(size(X_single)), y_categorical, 'rb', 'xo');
    hold on
    % plot(x_range, scores(:, 2), 'k-', 'LineWidth', 2); % Decision boundary
    plot(x_range, zeros(size(x_range)), 'k--'); % Decision line at score 0
    title(['Decision Boundary for Feature ', num2str(i)])
    xlabel(['Feature ', num2str(i)])
    ylabel('Decision Score')
    legend('Setosa', 'Versicolor', 'Decision Boundary', 'Location', 'Best')
    hold off
end
clear; clc; close all;

%% Question 3 Multiclass SVM %%

% Load data
load fisheriris
% Choose features 1 and 2
X = meas(:,1:2);
Y = species;

SVMModels   = cell(3,1);
classes     = unique(Y);
rng(1); % For reproducibility

for j = 1:numel(classes)
    indx            = strcmp(Y,classes(j)); % Create binary classes for each↙
classifier
    SVMModels{j}    = fitcsvm(X,indx,'ClassNames',[false true],'Standardize',↙
false,...
        'KernelFunction','linear','BoxConstraint',1);
end

% Examine scatter plot of the data
figure
```

```
gscatter(X(:,1),X(:,2),Y);
hold on
h = gca;
lims = [h.XLim h.YLim]; % Extract the x and y axis limits
title('{\bf Scatter Diagram of Iris Measurements}');
xlabel('Sepal Length (cm)');
ylabel('Sepal Width (cm)');
legend('Location','Northwest');
hold off

d = 0.02;
[x1Grid,x2Grid] = meshgrid(min(X(:,1)):d:max(X(:,1)),...
    min(X(:,2)):d:max(X(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
N = size(xGrid,1);
Scores = zeros(N,numel(classes));

for j = 1:numel(classes)
    [~,score] = predict(SVMModels{j},xGrid);
    Scores(:,j) = score(:,2); % Second column contains positive-class scores
end

[~,maxScore] = max(Scores,[],2);

figure
h(1:3) = gscatter(xGrid(:,1),xGrid(:,2),maxScore,...
    [0.1 0.5 0.5; 0.5 0.1 0.5; 0.5 0.5 0.1]);
hold on
h(4:6) = gscatter(X(:,1),X(:,2),Y);

for j = 1:numel(classes)
    sv = SVMModels{j}.SupportVectors;
    h(6+j) = plot(sv(:,1), sv(:,2), 'ko', 'MarkerSize', 8, 'LineWidth', 1.5,↙
'DisplayName', 'Support Vectors');
end
title('{\bf Iris Classification Regions}');
xlabel('Sepal Length (cm)');
ylabel('Sepal Width (cm)');
legend(h,{'setosa region','versicolor region','virginica region',...
    'observed setosa','observed versicolor','observed virginica','Support↙
Vectors'},...
    'Location','Northwest');
axis tight
hold off
clear; clc; close all;

%% Question 4 %% Optimize an SVM classifier

% Generate data
rng('default') % For reproducibility
grnpop = mvnrnd([1,0],eye(2),10);
redpop = mvnrnd([0,1],eye(2),10);
```

```
% View the base points
figure;
plot(grnpop(:,1),grnpop(:,2),'go')
hold on
plot(redpop(:,1),redpop(:,2),'ro')
hold off

% Generate 100 data points
redpts = zeros(100,2);
grnpts = redpts;
for i = 1:100
    grnpts(i,:) = mvnrnd(grnpop(randi(10),:),eye(2)*0.02);
    redpts(i,:) = mvnrnd(redpop(randi(10),:),eye(2)*0.02);
end

% View the data points
figure
plot(grnpts(:,1),grnpts(:,2),'go')
hold on
plot(redpts(:,1),redpts(:,2),'ro')
hold off

% Prepare data for classification

cdata = [grnpts;redpts];
grp = ones(200,1);
grp(101:200) = -1;

% Prepare cross validation
c = cvpartition(200,'KFold',10);

% Optimize fit
opts = struct('CVPartition',c,'AcquisitionFunctionName', ...
    'expected-improvement-plus');
Mdl = fitcsvm(cdata,grp,'KernelFunction','rbf', ...
    'OptimizeHyperparameters','auto','HyperparameterOptimizationOptions',opts);
```