Md Toufique Hasan

Student Number: 151129267

Exercise date: 30.03.2023

Machine Learning Algorithms: exercise 3

Answer: 1

```
% Input data as a cell array
data = {
    'weekday' 'spring' 'none'   'none'   'on time'
    'weekday' 'winter' 'none'   'slight' 'on time'
    'weekday', 'winter', 'none', 'slight', 'on time';
    'weekday', 'winter', 'high', 'heavy', 'late';
    'saturday', 'summer', 'normal', 'none', 'on time';
    'weekday', 'autumn', 'normal', 'none', 'very late';
    'holiday', 'summer', 'high', 'slight', 'on time';
    'sunday', 'summer', 'normal', 'none', 'on time';
    'weekday', 'winter', 'high', 'heavy', 'very late';
    'weekday', 'summer', 'none', 'slight', 'on time';
    'saturday', 'spring', 'high', 'heavy', 'cancelled';
    'weekday', 'summer', 'high', 'slight', 'on time';
    'saturday', 'winter', 'normal', 'none', 'late';
    'weekday', 'summer', 'high', 'none', 'on time';
    'weekday', 'winter', 'normal', 'heavy', 'very late';
    'saturday', 'autumn', 'high', 'slight', 'on time';
    'weekday', 'autumn', 'none', 'heavy', 'on time';
    'holiday', 'spring', 'normal', 'slight', 'on time';
    'weekday', 'spring', 'normal', 'none', 'on time';
    'weekday' 'spring' 'normal' 'slight' 'on time'
};

% Convert categorical data to numbers
categories = unique(data);
[~, numericData] = ismember(data, categories);

% Extract feature matrix (X) and class labels (Y)
X = numericData(:, 1:4);
Y = numericData(:, 5);

% Remove rows with missing values
missingValues = any(X == 0, 2);
X(missingValues, :) = [];
Y(missingValues) = [];

% Train the Naïve Bayes classifier
```

```
NBModel = fitcnb(X, Y, 'Distribution', 'mn', 'Prior', 'empirical', 'ClassNames', unique(Y));

% Test case: (weekday, winter, high, heavy, ?)
testCase = {'weekday', 'winter', 'high', 'heavy'};
[~, numericTestCase] = ismember(testCase, categories);
predictedClass = predict(NBModel, numericTestCase);

% Convert predicted class number back to the corresponding label
predictedLabel = categories{predictedClass};

% Display the result
fprintf('The most probable class for the test case is: %s\n', predictedLabel);
```

```
The most probable class for the test case is: late
```

Answer: 2

```
% Load data from Excel file
data = readmatrix(['D:\TUNI\Courses\Period-4\DATA.ML.210 ' ...
    '[Machine Learning Algorithms]\Exercises 3\data3.xlsx']);
c1 = data(1:100,:); % Class C1 data
c2 = data(101:200,:); % Class C2 data
test_data = data(201:220,:); % Test data

% Estimate 2D Gaussian models for classes C1 and C2
mu_c1 = mean(c1);
sigma_c1 = cov(c1);
mu_c2 = mean(c2);
sigma_c2 = cov(c2);

% Classify test data using the estimated models
labels = zeros(20,1);
for i = 1:20
    % Calculate class-conditional probabilities for each class
    p_c1 = mvnpdf(test_data(i,:), mu_c1, sigma_c1);
    p_c2 = mvnpdf(test_data(i,:), mu_c2, sigma_c2);

    % Classify test data point based on which class has higher probability
    if p_c1 > p_c2
        labels(i) = 1; % Classify as C1
    else
        labels(i) = 2; % Classify as C2
    end
end

% Calculate accuracy, specificity, and sensitivity of the classifier
true_labels = [ones(10,1);2*ones(10,1)]; % True labels for the test data
accuracy = sum(labels == true_labels)/20;
tp = sum(labels(1:10) == 1); % True positives
fp = sum(labels(11:20) == 1); % False positives
```

```
tn = sum(labels(11:20) == 2); % True negatives
fn = sum(labels(1:10) == 2); % False negatives
sensitivity = tp/(tp+fn);
specificity = tn/(tn+fp);

% Display results
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 1.00
```

```
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```

Answer: 3

```
c1 = data(1:100,:); % Class C1 data
c2 = data(101:200,:); % Class C2 data
test_data = data(201:220,:); % Test data

% Calculate means for classes C1 and C2
mu_c1 = mean(c1);
mu_c2 = mean(c2);

% Calculate covariance matrices for classes C1 and C2
sigma_c1 = cov(c1);
sigma_c2 = cov(c2);

% Classify test data using Mahalanobis distance
labels = zeros(20,1);
for i = 1:20
    % Calculate Mahalanobis distance between the test point and class means
    dist_c1 = mahal(test_data(i,:), c1);
    dist_c2 = mahal(test_data(i,:), c2);

    % Classify test data point based on which class has smaller distance
    if dist_c1 < dist_c2
        labels(i) = 1; % Classify as C1
    else
        labels(i) = 2; % Classify as C2
    end
end

% Calculate accuracy, specificity, and sensitivity of the classifier
true_labels = [ones(10,1);2*ones(10,1)]; % True labels for the test data
accuracy = sum(labels == true_labels)/20;
tp = sum(labels(1:10) == 1); % True positives
```

```
fp = sum(labels(11:20) == 1); % False positives
tn = sum(labels(11:20) == 2); % True negatives
fn = sum(labels(1:10) == 2); % False negatives
sensitivity = tp/(tp+fn);
specificity = tn/(tn+fp);

% Display results
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 1.00
```

```
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```

Answer: 4

```
c1 = data(1:100,:); % Class C1 data
c2 = data(101:200,:); % Class C2 data
test_data = data(201:220,:); % Test data

% Define parameters
k_values = [1, 3]; % Values of k to try
dist_metrics = {'euclidean', 'cityblock', 'cosine', 'chebychev'}; % Distance metrics to try
true_labels = [ones(10,1);2*ones(10,1)]; % True labels for the test data

% Loop over distance metrics and k values
for d = 1:length(dist_metrics)
    for k = k_values
        % Classify test data using KNN rule with current distance metric and k value
        labels = knnsearch([c1;c2], test_data, 'K', k, 'Distance', dist_metrics{d});
        labels = ceil(labels/100); % Convert labels to 1 or 2

        % Calculate accuracy, specificity, and sensitivity of the classifier
        accuracy = sum(labels == true_labels)/20;
        tp = sum(labels(1:10) == 1); % True positives
        fp = sum(labels(11:20) == 1); % False positives
        tn = sum(labels(11:20) == 2); % True negatives
        fn = sum(labels(1:10) == 2); % False negatives
        sensitivity = tp/(tp+fn);
        specificity = tn/(tn+fp);

        % Display results
        fprintf('Distance metric: %s, k = %d\n', dist_metrics{d}, k);
        fprintf('Accuracy: %.2f\n', accuracy);
        fprintf('Sensitivity: %.2f\n', sensitivity);
        fprintf('Specificity: %.2f\n\n', specificity);
```

```
    end
end
```

```
Distance metric: euclidean, k = 1
Accuracy: 1.00
Sensitivity: 1.00
Specificity: 1.00
Distance metric: euclidean, k = 3
Accuracy: 1.00
Accuracy: 1.00
Accuracy: 0.90
Sensitivity: 1.00
Specificity: 1.00
Distance metric: cityblock, k = 1
Accuracy: 1.00
Sensitivity: 1.00
Specificity: 1.00
Distance metric: cityblock, k = 3
Accuracy: 1.00
Accuracy: 1.00
Accuracy: 0.90
Sensitivity: 1.00
Specificity: 1.00
Distance metric: cosine, k = 1
Accuracy: 0.75
Sensitivity: 0.70
Specificity: 0.80
Distance metric: cosine, k = 3
Accuracy: 0.75
Accuracy: 0.85
Accuracy: 0.80
Sensitivity: 0.70
Specificity: 0.80
Distance metric: chebychev, k = 1
Accuracy: 1.00
Sensitivity: 1.00
Specificity: 1.00
Distance metric: chebychev, k = 3
Accuracy: 1.00
Accuracy: 1.00
Accuracy: 0.95
Sensitivity: 1.00
Specificity: 1.00
```

Answer: 5

```
c1 = data(1:100,:); % Class C1 data
c2 = data(101:200,:); % Class C2 data
test_data = data(201:220,:); % Test data

% Train a linear discriminant classifier on the training data
X_train = [c1;c2];
y_train = [ones(size(c1,1),1);2*ones(size(c2,1),1)];
classifier = fitcdiscr(X_train, y_train, 'discrimType', 'linear');

% Classify test data using the linear discriminant classifier
labels = predict(classifier, test_data);

% Calculate accuracy, specificity, and sensitivity of the classifier
```

```
true_labels = [ones(10,1);2*ones(10,1)]; % True labels for the test data
accuracy = sum(labels == true_labels)/20;
tp = sum(labels(1:10) == 1); % True positives
fp = sum(labels(11:20) == 1); % False positives
tn = sum(labels(11:20) == 2); % True negatives
fn = sum(labels(1:10) == 2); % False negatives
sensitivity = tp/(tp+fn);
specificity = tn/(tn+fp);

% Display results
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 1.00
```

```
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```

Answer: 6

```
c1 = data(1:100,:); % Class C1 data
c2 = data(101:200,:); % Class C2 data
test_data = data(201:220,:); % Test data

% Train a Naive Bayes classifier on the training data
X_train = [c1;c2];
y_train = [ones(size(c1,1),1);2*ones(size(c2,1),1)];
classifier = fitcnb(X_train, y_train);

% Classify test data using the Naive Bayes classifier
labels = predict(classifier, test_data);

% Calculate accuracy, specificity, and sensitivity of the classifier
true_labels = [ones(10,1);2*ones(10,1)]; % True labels for the test data
accuracy = sum(labels == true_labels)/20;
tp = sum(labels(1:10) == 1); % True positives
fp = sum(labels(11:20) == 1); % False positives
tn = sum(labels(11:20) == 2); % True negatives
fn = sum(labels(1:10) == 2); % False negatives
sensitivity = tp/(tp+fn);
specificity = tn/(tn+fp);

% Display results
fprintf('Accuracy: %.2f\n', accuracy);
```

```
Accuracy: 1.00
```

```
fprintf('Sensitivity: %.2f\n', sensitivity);
```

```
Sensitivity: 1.00
```

```
fprintf('Specificity: %.2f\n', specificity);
```

```
Specificity: 1.00
```