Analytics for Business (Spring 2025)

Individual Assignment

Niklas Pulliainen

AI was not utilized in this work.

Dataset Overview

This assignment analyzes a dataset of weather station measurements in Helsinki from January 2010 to December 2012. The provided dataset contains measurements (both daily mean and daily variance) on the following:

- T: Air temperature (°C)
- Po: Atmospheric pressure at station level (mmHg)
- P: Atmospheric pressure reduced to mean sea level (mmHg)
- Ff: Mean wind speed (m/s)
- Tn: Minimum air temperature over the past day (°C)
- Tx: Maximum air temperature over the past day (°C)
- W: Horizontal visibility (km)
- Td: Dewpoint temperature (°C)
- U: Relative humidity (%)

The dataset consists of 19 attributes: one datetime variable, 16 continuous variables and two target variables, of which one is binary and the other is continuous.

Data Import & Initialization

Import weather_dataset.xlsx as a table:

```
data_table = readtable("weather_dataset.xlsx");
```

Set random seed to default for reproducibility:

```
rng('default');
```

Task 1

The target variable OBSERVED is binary:

- 0 means that precipitation exceeded 0.3 millimeters (not-dry).
- 1 means that precipitation was below that (dry).

1) Maximum value of the variable Td_mu for the groups in OBSERVED:

```
% Identify groups in OBSERVED:
[OBS_groups,OBS_vals] = findgroups(data_table.OBSERVED);
% Calculate the maximum value for both groups:
max_Td_mu_byGroup = splitapply(@max,data_table.Td_mu,OBS_groups)
```

```
max_Td_mu_byGroup = 2×1
   20.3000
   21.4625
```

The maximum value of Td_mu for not-dry (0) is 20.3 and 21.4625 for dry (1).

2) Mean value of P_var for the same groups as above:

```
mean_P_var_byGroup = splitapply(@mean,data_table.P_var,OBS_groups)
```

```
mean_P_var_byGroup = 2×1
    5.7050
    2.8374
```

The mean value of P_var for not-dry (0) is 5.705 and 2.8374 for dry (1).

3) Convert the datetime variable into separate numeric columns:

```
% Create three columns for year, month and day:
[data_table.YY,data_table.MM,data_table.DD] = ymd(data_table.datetime);
% Remove the original datetime column:
data_table.datetime = [];
```

Three new columns are created (YY, MM, DD) and the original datetime column is removed.

4) Separate target and predictor variables:

```
% Target variable OBSERVED (binary):
Y1 = data_table.OBSERVED;
% Target variable U_mu (continuous):
Y2 = data_table.U_mu;
% Predictor matrix:
X = data_table;
X(:,17:18) = [];
% Convert the table to array format:
X = table2array(X);
```

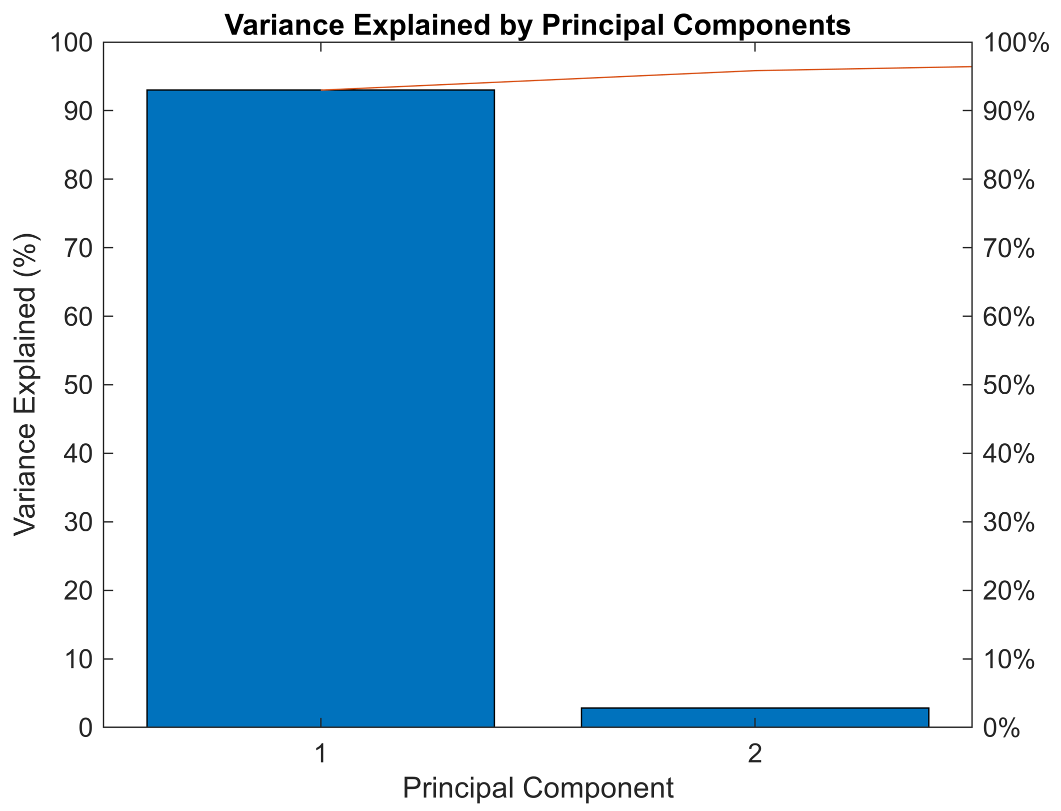
The target variables Y1 (binary) and Y2 (continuous) are extracted from the data. The predictor matrix X is created and the last two columns are removed since they are the target variables. Finally, the predictor matrix X is converted into array format.

Task 2

1) Principal Component Analysis (PCA):

```
% Reduce dimensionality with PCA:
[coeff,score,latent,~,explained] = pca(X);
```

```
% Plot the cumulative explained variance:
figure;
pareto(explained)
xlabel('Principal Component')
ylabel('Variance Explained (%)')
title('Variance Explained by Principal Components')
```



```
% Number of components:
enough_explained = cumsum(explained)/sum(explained) >= 94/100;
number_of_components = find(enough_explained,1)
```

```
number_of_components = 
2
```

```
% PCA-transformed dataset:
PCA_data = score(:,1:2);
pc1 = PCA_data(:,1);
pc2 = PCA_data(:,2);
```

The first component explains nearly 93% of the variance, whereas the second component explains 2.84% of the variance. Two components are thus needed in order to explain at least 94% of the total variance. A new PCA dataset is created and the two components are also extracted separately to make further analysis more convenient.

2) K-Means Clustering:

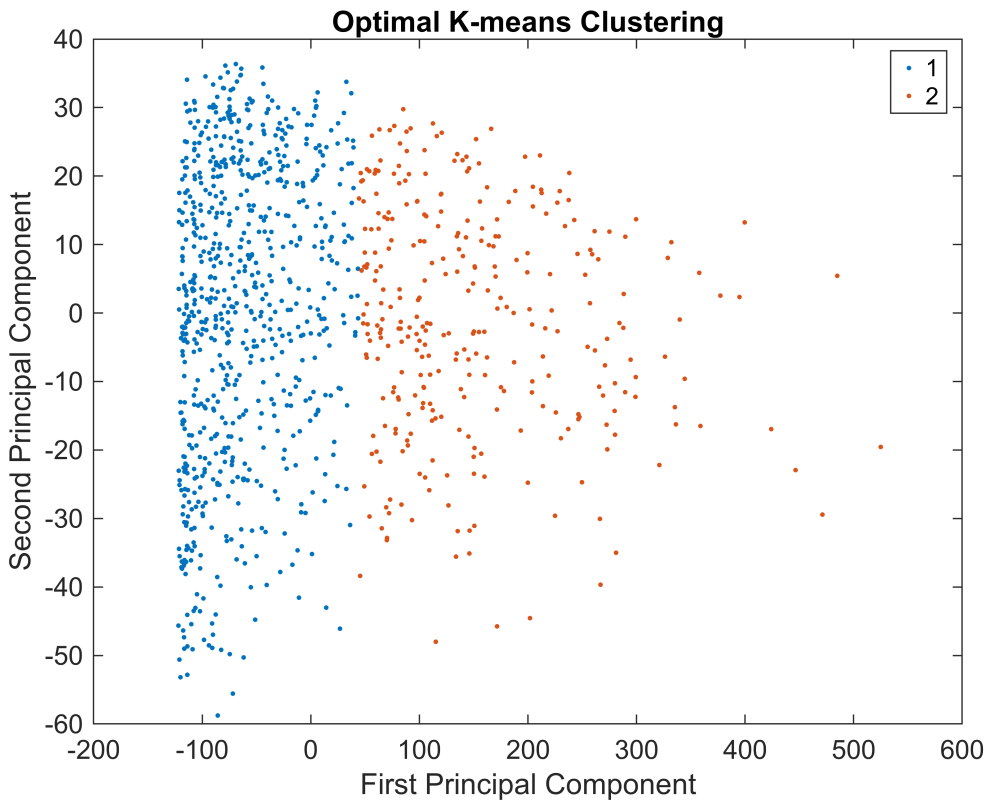
```
% K-means clustering with Squared Euclidian distance
% with 2, 3 and 5 clusters:
kList = [2, 3, 5];
E = evalclusters(score,'kmeans','silhouette','KList',kList);
% Silhouette scores and optimal number of clusters:
silhouette_scores = E.CriterionValues
```

```
silhouette_scores = 1×3
    0.7776    0.7294    0.5016
```

```
optimal_clusters = E.OptimalK
```

```
optimal_clusters =
2
```

```
% Scatter plot of the optimal number of clusters:
figure;
gscatter(pc1,pc2, E.OptimalY)
xlabel('First Principal Component')
ylabel('Second Principal Component')
title('Optimal K-means Clustering')
```



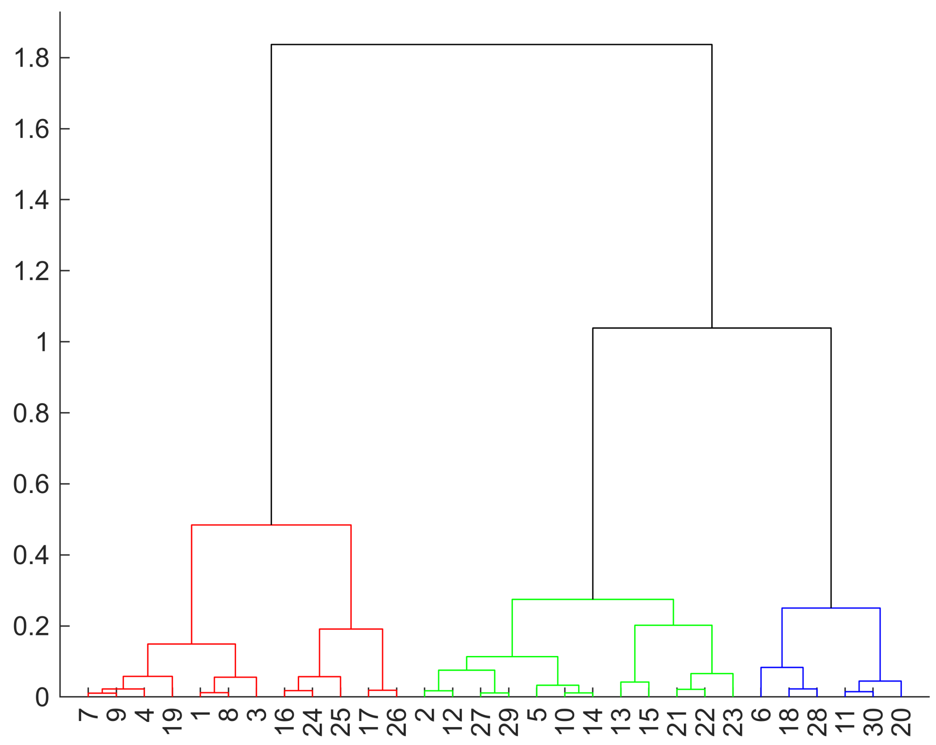
The function evalclusters() provides both the silhouette scores and the optimal number of clusters. The optimal number of clusters is 2 which provides the best silhouette score (0.7776). 3 clusters follows closely behind, whereas 5 clusters provides a distinctively worse score.

3) Hierarchical Clustering:

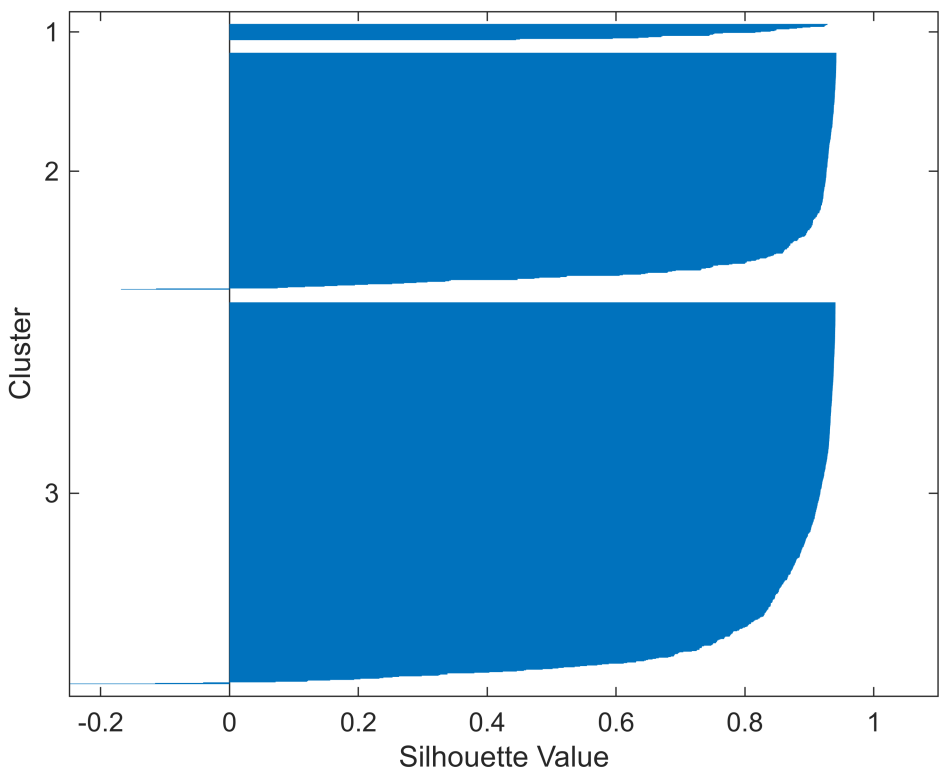
```
% Hierarchical clustering with cosine distance
% and average linkage:
cos_dist = pdist(PCA_data, "cosine");
cos_tree = linkage(cos_dist,'average');
% Define and visualize the clusters (cut-off = 0.6):
cos_tree_clusters = cluster(cos_tree,'criterion','distance',...
    'cutoff',0.6);
optimal_clusters2 = length(unique(cos_tree_clusters))
```

```
optimal_clusters2 =
3
```

```
dendrogram(cos_tree,'ColorThreshold',0.6)
```



```
% Silhouette score:
[silh_cos,h] = silhouette(PCA_data,cos_tree_clusters,"cosine");
```



```
cos_tree_silhouette = mean(silh_cos)
```

```
cos_tree_silhouette =
0.8627
```

The number of clusters with cosine distance, average linkage and a linkage distance of 0.6 is 3 as calculated and shown by the dendogram. The corresponding silhouette score is 0.8627, which is better than in K-means clustering and also quite high, indicating well-separated clusters.

4) Gaussian Mixture Model (GMM):

```
% Gaussian Mixture Model (GMM) with 3 clusters and
% 1000 iterations:
gmm_clusters = 3;
gmm_options = statset('MaxIter',1000);
gmm_pc_model = fitgmdist(PCA_data,gmm_clusters,'Options',...
    gmm_options)
```

```
gmm_pc_model =

Gaussian mixture distribution with 3 components in 2 dimensions
Component 1:
Mixing proportion: 0.376679
Mean:  -91.1310   -6.2220

Component 2:
Mixing proportion: 0.448306
Mean:   87.6509   -3.5917
```

```
Component 3:
Mixing proportion: 0.175015
Mean:  -28.3817    22.5916
```

```
% Probability of the 123rd observation belonging to
% each of the three clusters:
obs123 = PCA_data(123,:);
P = posterior(gmm_pc_model,obs123)
```

```
P = 1×3
    0.7598    0.2199    0.0203
```

The probability of the 123rd observation from the PCA-transformed dataset belonging to cluster 1 is 75,98%, 21,99% for cluster 2 and 2,03% for cluster 3.

Task 3

1) Train & Test Split:

```
% Split the original data into 60% training and
% 40% testing:
cv1 = cvpartition(length(Y2),'holdout',0.4);
X_train = X(training(cv1),:);
X_test = X(test(cv1),:);
Y2_train = Y2(training(cv1),:);
Y2_test = Y2(test(cv1),:);
```

Split the predictor matrix X into training (60%) and test (40%) data, as well as the continuous target variable Y2 (U_mu).

2) Linear Regression:

```
% Linear regression using the entire training set:
linear_model = fitlm(X_train,Y2_train,'linear');
```

A simple linear regression model using all the data.

3) Validation Set Creation:

```
% Divide the data further:
cv2 = cvpartition(length(Y2_train),'holdout',0.2);
% Model training:
X_train2 = X_train(training(cv2),:);
Y2_train2 = Y2_train(training(cv2),:);
% Model validation:
X_val = X_train(test(cv2),:);
Y2_val = Y2_train(test(cv2),:);
```

Split 20% of the data into a validation set to compare model performance with hold-out cross-validation.

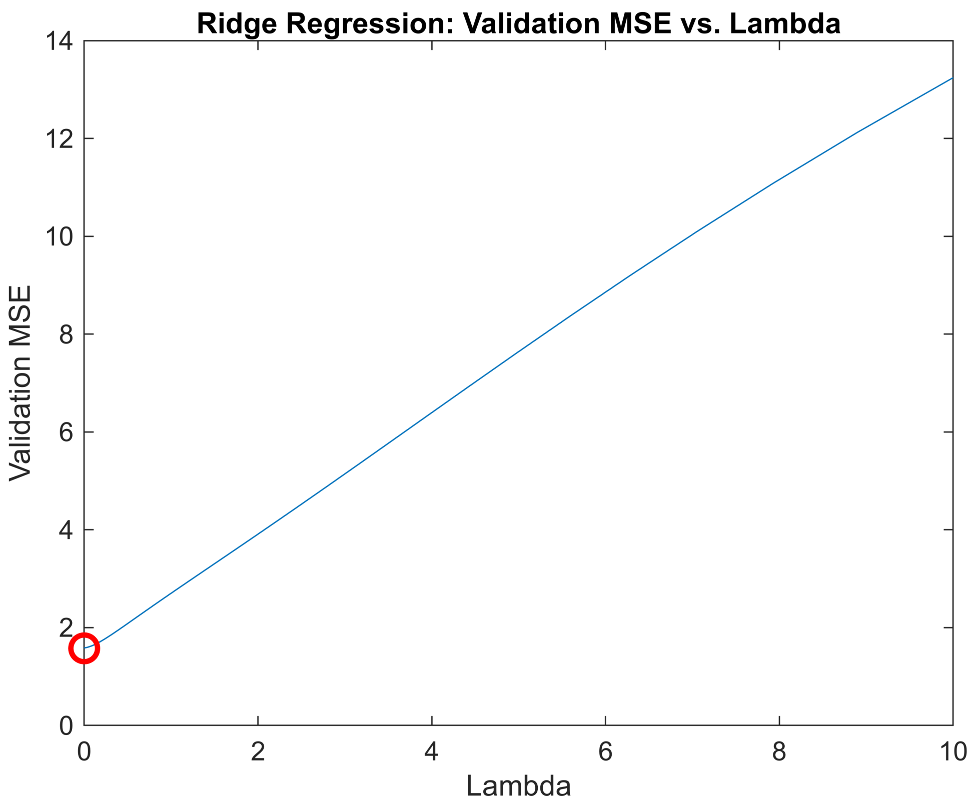
4) Ridge Regression:

```
% Generate 100 lambda values (from 10^(-4) to 10^1):
```

```
lambda_values_ridge = logspace(-4,1,100);
immse_ridge = zeros(length(lambda_values_ridge),1);
% Train 100 ridge regressions:
for i=1:length(lambda_values_ridge)
    % Train ridge
    b=ridge(Y2_train2,X_train2,lambda_values_ridge(i),0);
    % Predict on validation set
    ypred_ridge=b(1)+X_val*b(2:end);
    % Compute MSE
    immse_ridge(i) = immse(Y2_val,ypred_ridge);
end
% Optimal lambda
[~,best_ridge_idx]=min(immse_ridge);
best_ridge_lambda=lambda_values_ridge(best_ridge_idx)
```

```
best_ridge_lambda = 
1.0000e-04
```

```
% Plot lambda (x) and validation MSE (y):
figure;
plot(lambda_values_ridge,immse_ridge)
xlabel('Lambda');
ylabel('Validation MSE');
title('Ridge Regression: Validation MSE vs. Lambda');
% Highlight optimal lambda
hold on;
plot(best_ridge_lambda, immse_ridge(best_ridge_idx), 'ro',...
'MarkerSize', 10,'LineWidth', 2);
hold off;
```



```
% Retrain the model on full training
% set using optimal lambda:
b_final_ridge=ridge(Y2_train,X_train,best_ridge_lambda,0);
ypred_final_ridge=b_final_ridge(1)+X_test*b_final_ridge(2:end);
MSE_final_ridge=immse(Y2_test,ypred_final_ridge);
```

Train 100 ridge regression models in a loop using logarithmically spaced lambda values between 10^(-4) and 10^1. The optimal lambda for the ridge regression is 0.0001 based on the lowest validation MSE. The model is retrained on the full training set using this lambda value and its MSE is calculated using the test set, which is later used in step 7 to evaluate the different models based on RMSE.

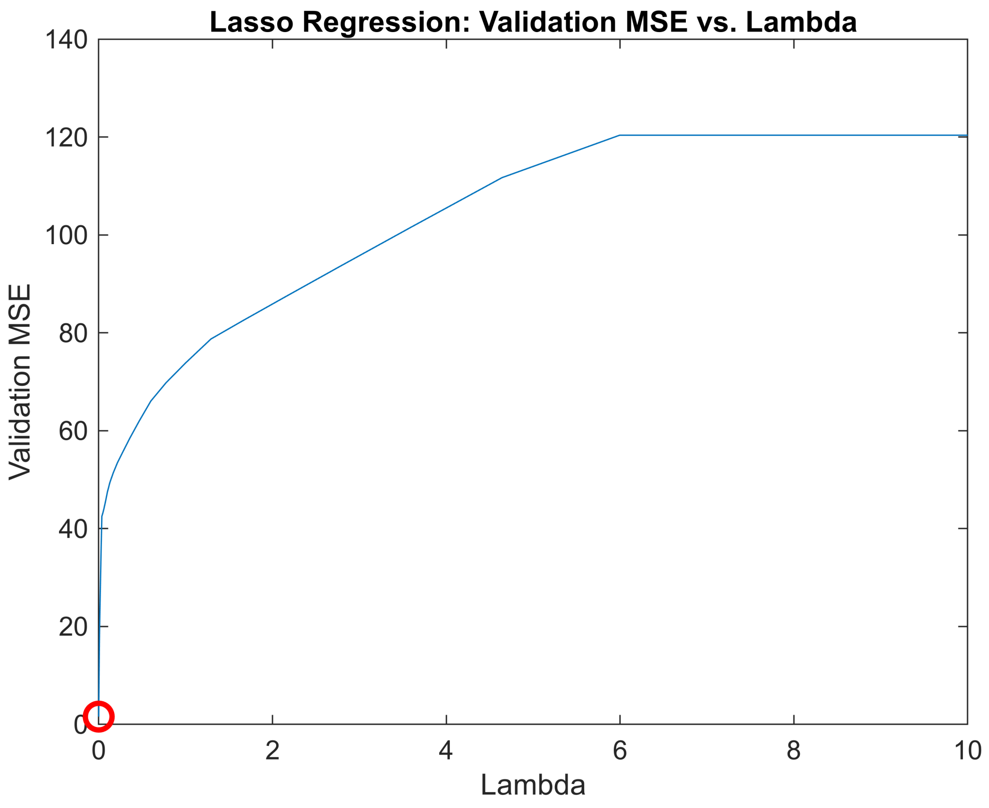
5) Lasso Regression:

```
% Generate 100 lambda values (from 10^(-10) to 10^1):
lambda_values_lasso = logspace(-10,1,100);
immse_lasso = zeros(length(lambda_values_lasso),1);
% Train 100 lasso regressions:
for i=1:length(lambda_values_lasso)
    % Train ridge
    [B,fitinfo]=lasso(X_train2,Y2_train2,'Lambda',lambda_values_ridge(i));
    % Predict on validation set
    ypred_lasso=fitinfo.Intercept+X_val*B;
    % Compute MSE
    immse_lasso(i) = immse(Y2_val,ypred_lasso);
end
% Optimal lambda
```

```
[~,best_lasso_idx]=min(immse_lasso);
best_lasso_lambda=lambda_values_lasso(best_lasso_idx)
```

```
best_lasso_lambda =
1.0000e-10
```

```
% Plot lambda (x) and validation MSE (y):
figure;
plot(lambda_values_lasso,immse_lasso)
xlabel('Lambda');
ylabel('Validation MSE');
title('Lasso Regression: Validation MSE vs. Lambda');
% Highlight optimal lambda
hold on;
plot(best_lasso_lambda, immse_lasso(best_lasso_idx), 'ro',...
'MarkerSize', 10,'LineWidth', 2);
hold off;
```



```
% Retrain the model on full training set using optimal lambda:
[B_final_lasso, fitinfo_final_lasso] = lasso(X_train,Y2_train,...
'Lambda',best_lasso_lambda);
ypred_final_lasso = fitinfo_final_lasso.Intercept + X_test * B_final_lasso;
MSE_final_lasso=immse(Y2_test,ypred_final_lasso);
```

The same process is repeated for lasso regression, with 100 lambda values ranging from 10^(-10) to 10^1. The optimal lambda is 10^(-10) based on the lowest validation MSE, and the model is retrained on the full training set using this lambda value, and MSE is calculated using the test set to determine RMSE later on in step 7.

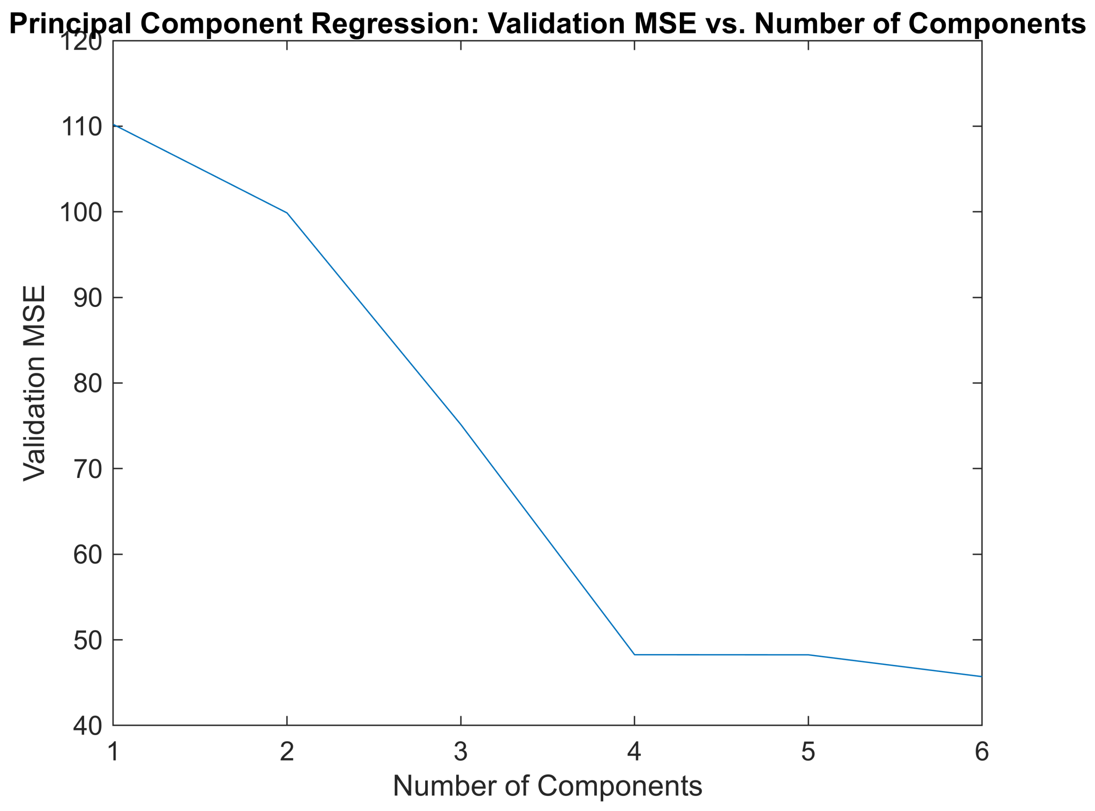
6) Principal Component Regression (PCR):

```
% Train six linear regressions PCR
[coeff2,score2,~,~,explained2,mu2] = pca(X_train2);
PCR_6_components = score2(:,1:6);
immse_pcr = zeros(6,1);

for i=1:6
    % Train PCR
    Xtrain_pca = PCR_6_components(:,1:i);
    PCR_model = fitlm(Xtrain_pca,Y2_train2);
    % Predict on validation set
    val_score = (X_val-mu2)*coeff2(:,1:i);
    ypred_PCR = predict(PCR_model,val_score);
    % Compute MSE
    immse_pcr(i) = immse(Y2_val,ypred_PCR);
end
% Optimal number of components
[~,best_n_components] = min(immse_pcr)
```

```
best_n_components =
6
```

```
% Plot the number of components (x) and validation MSE (y)
figure;
plot(1:6,immse_pcr)
xlabel('Number of Components');
ylabel('Validation MSE');
title('Principal Component Regression: Validation MSE vs. Number of Components');
```



```
% Perform PCA on full training set and retrain a linear regression model
[coeff_final,score_final,~,~,explained_final,mu_final]=pca(X_train);
score_train_final = score_final(:,1:best_n_components);
PCR_model_final = fitlm(score_train_final,Y2_train)
```

```
PCR_model_final = 
Linear regression model:
    y ~ 1 + x1 + x2 + x3 + x4 + x5 + x6

Estimated Coefficients:
                   Estimate        SE         tStat       pValue  
                   _________    _________    ________    __________

    (Intercept)        80.75      0.29251      276.06             0
    x1              0.027448    0.0026071      10.528    5.6017e-24
    x2              -0.20774     0.014414     -14.413     6.955e-41
    x3              -0.42454     0.022818     -18.605    7.4752e-62
    x4              -0.36877      0.02423     -15.219    9.5714e-45
    x5             -0.027034     0.032094    -0.84236       0.39991
    x6               0.13972     0.032713       4.271    2.2483e-05


Number of observations: 634, Error degrees of freedom: 627
Root Mean Squared Error: 7.37
R-squared: 0.593,  Adjusted R-Squared: 0.59
F-statistic vs. constant model: 153, p-value = 4.96e-119
```

```
% Transform
score_test_final = (X_test - mu_final)*coeff_final(:,1:best_n_components);
ypred_final_pcr = predict(PCR_model_final,score_test_final);
MSE_final_pcr = immse(Y2_test,ypred_final_pcr);
```

The optimal number of components is 6 based on the lowest validation MSE. PCA is performed on the full training set and a linear regression is retrained and its MSE is calculated using the test set to be used later in step 7.

7) Model Evaluation & Interpretation:

```
% Model performance based on RMSE
RMSE_linear = linear_model.RMSE
```

```
RMSE_linear =
1.4095
```

```
RMSE_ridge = sqrt(MSE_final_ridge)
```

```
RMSE_ridge =
1.2691
```

```
RMSE_lasso = sqrt(MSE_final_lasso)
```

```
RMSE_lasso =
1.2674
```

```
RMSE_pcr = sqrt(MSE_final_pcr)
```

```
RMSE_pcr =
7.2061
```

```
% Best model:
best_RMSE = min([RMSE_linear,RMSE_ridge,RMSE_lasso,RMSE_pcr])
```

```
best_RMSE =
1.2674
```

```
% Coefficient comparison:
reg_names = {'Linear','Ridge','Lasso'}
```

```
reg_names = 1×3 cell
'Linear'     'Ridge'      'Lasso'
```

```
reg_table = table(...
    [linear_model.Coefficients(2:end,"Estimate")],...
    [b_final_ridge(2:end)],...
    [B_final_lasso],...
    'VariableNames',reg_names)
```

reg_table = 19×3 table

	Linear		Ridge	Lasso
	Row	Estimate		
1	x1	-4.1707	-4.1699	-4.1598
2	x2	2.9781	2.8349	0.0649
3	x3	-3.0159	-2.8729	-0.1049
4	x4	-0.2559	-0.2560	-0.2566

	Linear		Ridge	Lasso
	Row	Estimate		
5	x5	0.1082	0.1079	0.1056
6	x6	-0.2883	-0.2887	-0.2942
7	x7	-0.0722	-0.0722	-0.0720
8	x8	4.4615	4.4615	4.4629
9	x9	0.0200	0.0200	0.0203
10	x10	0.4440	0.4347	0.1914
11	x11	-0.4335	-0.4243	-0.1821
12	x12	-0.0875	-0.0875	-0.0881
13	x13	0.0526	0.0526	0.0521
14	x14	-0.0218	-0.0218	-0.0225
15	x15	0.0011	0.0011	0.0011
16	x16	0.0565	0.0565	0.0566
17	x17	-0.0698	-0.0700	-0.0736
18	x18	0.0026	0.0026	0.0026
19	x19	-3.1740e-04	-3.3707e-04	-7.0483e-04

Root mean squared error is calculated for each model using their respective MSEs calculated earlier.

a) RMSE for each model:

- Linear: 1.4095
- Ridge: 1.2691
- Lasso: 1.2674
- PCR: 7.2061

The best performing model on test data based on the lowest RMSE is lasso regression.

b) Analysis of regularized models:

- The optimal lambdas were 10^(-4) and 10^(-10) for the ridge and lasso models respectively. These were also the minimum values that were pre-determined.
- Using reg_table, it can be seen that linear and ridge regressions have coefficients of similar magnitude, whereas lasso provided lower results on x2, x3, x10, x11 and x19, but it did not drop them to zero.
- Based on RMSE, regularization improved the model performance when compared to the baseline linear model. Although RMSE is the lowest for lasso, it is still quite similar to the ridge model. Greater difference can be seen from the coefficients, where lasso regression has lowered the 5 beforementioned coefficients significantly in contrast to the other models.
- By introducing bias, the model performed better. This process helps in avoiding over-fitting, so the data was too complex for the linear regression.

c) Evaluation of PCR model:

- The PCR model performed the worst with RMSE of 7.2061.
- There was no benefit in applying dimensionality reduction as it made the model perform even worse. Important information on the data was most likely lost during the dimensionality reduction process.

Conclusion:

The choice of the most suitable model is between ridge and lasso, as their RMSEs are the lowest and nearly the same. Lasso did not perform any feature selection with this dataset, which is its advantage over ridge regression. However, since the lasso model had the lowest RMSE, although marginally so, it can be considered as the most suitable model for this dataset.

Task 4

1) Dataset Partition:

```
% Split target variable Y1:
Y1_train = Y1(training(cv1),:);
Y1_test = Y1(test(cv1),:);
```

Split the binary target variable Y1 into training and testing.

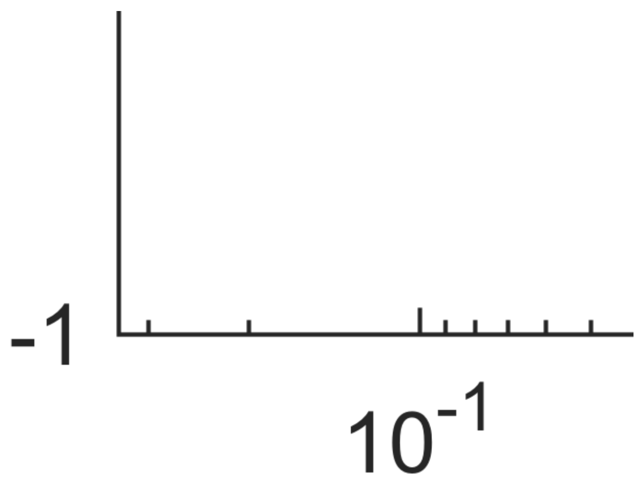
2) K-Fold Cross-Validation:

```
% Apply K-fold cross-validation on the training data (K=4):
cv_kfold = cvpartition(length(Y1_train),'KFold',4);
```

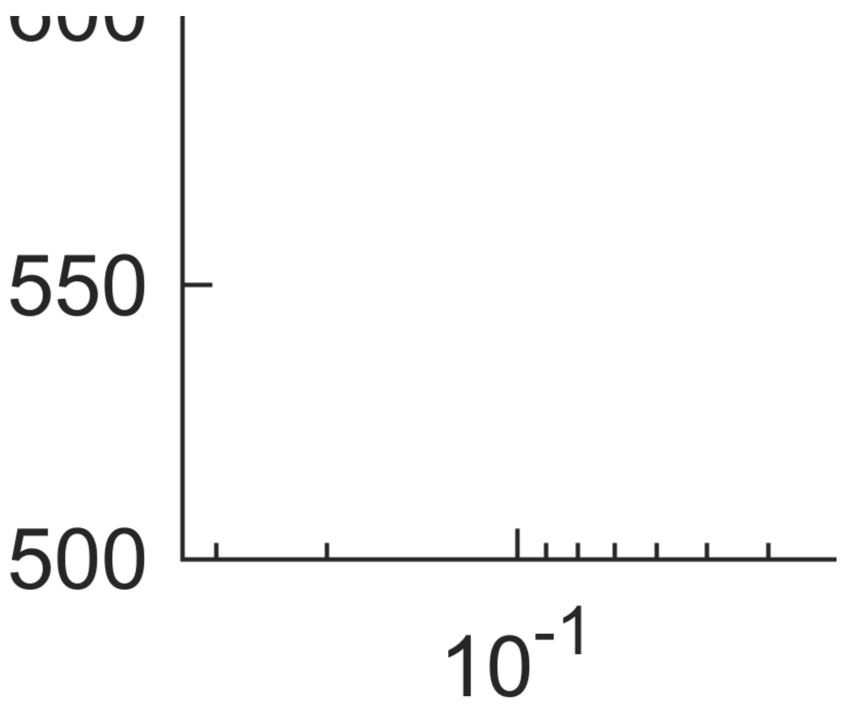
4-fold partition on the training data to be used in the following steps.

3) Lasso Logistic Regression:

```
% Train and cross-validate a model with 50 lambda values:
[B4,fitinfo4] =
lassoglm(X_train,Y1_train,'binomial','CV',cv_kfold,'MaxIter',1e5,'NumLambda',50);
% Plot the number of selected predictors as a function of lambda:
lassoPlot(B4,fitinfo4,'PlotType','Lambda','XScale','log');
```



```
% Plot the cross-validated deviance as a function of lambda:
lassoPlot(B4,fitinfo4,'plottype','CV');
legend('show')
```



```
% Number of selected features using LambdaMinDeviance:
num_coeffs_lambdaMinDev = fitinfo4.DF(fitinfo4.IndexMinDeviance)
```

```
num_coeffs_lambdaMinDev =
18
```

```
% Number of selected features using Lambda1SE:
num_coeffs_lambda1SE = fitinfo4.DF(fitinfo4.Index1SE)
```

```
num_coeffs_lambda1SE =
10
```

```
% Retrain the Lasso Model using Lambda1SE:
[B4_final,fitinfo4_final]=lassoglm(X_train,Y1_train,"binomial","Lambda",fitinfo4.Lam
bda1SE);
% Full coefficient vector:
coef_final_1SE = [fitinfo4_final.Intercept;B4_final];
ypred_lasso_1SE = glmval(coef_final_1SE,X_test,'logit');
```

The number of features selected using LambdaMinDeviance is 18 and 10 with Lambda1SE. LambdaMinDeviance, as the name suggests, is the model with the lowest deviance (error) and thus the best fit. Lambda1SE is a model which is within one standard deviation of the minimum deviance, but this model is more simpler (10 predictors instead of 18) with a higher lambda value and thus feature selection. The model is retrained using Lambda1SE.

4) Support Vector Machines (SVM):

```
% Model 1 with RBF kernel, KernelScale 'auto' and standardized data:
SVM1 = fitcsvm(X_train,Y1_train,'Standardize',true,'KernelFunction',...
    'rbf','KernelScale','auto','CVPartition',cv_kfold);
% Model 2 with polynomial kernel and standardized data:
SVM2 = fitcsvm(X_train,Y1_train,'Standardize',true,'KernelFunction',...
    'polynomial','CVPartition',cv_kfold);
% Accuracy of the third fold for the SVM with RBF kernel:
accuracy3_SVM1 = 1 - kfoldLoss(SVM1,"Mode","individual");
disp(accuracy3_SVM1(3))
```

```
    0.7547
```

```
% Best-performing SVM model based on cross-validation accuracy:
accuracy_SVM1 = mean(accuracy3_SVM1)
```

```
accuracy_SVM1 =
0.7319
```

```
accuracy_SVM2 = 1 - kfoldLoss(SVM2)
```

```
accuracy_SVM2 =
0.6909
```

```
% Retrain the SVM model on the full training set using RBF kernel:
final_SVM = fitcsvm(X_train,Y1_train,'Standardize',true,'KernelFunction',...
    'rbf','KernelScale','auto');
ypred_SVM = predict(final_SVM,X_test);
```

The accuracy on the third fold for the model with RBF kernel is 0.7547. This model is also overall the best, as its accuracy is 0.7319, whereas the model with polynomial kernel has an accuracy of 0.6909. The model is retrained on the full training set using the model with RBF kernel.

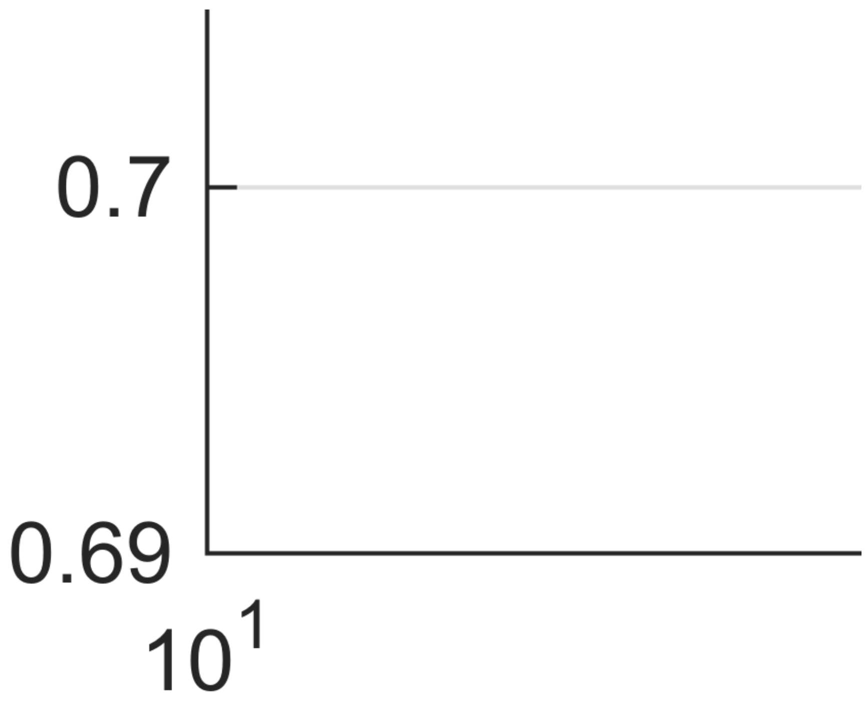
5) Classification Tree:

```
% Generate 10 Min Leaf Size values from 10^1 to 10^2:
leafs = logspace(1,2,10);
% Train a classification tree model for each value of Min Leaf Size:
accuracy_ctree = zeros(10,1);
for n=1:10
    ctree_model = fitctree(X_train,Y1_train,'MinLeafSize',leafs(n),...
        'CVPartition',cv_kfold);
    % Cross-validated accuracy of each model:
    accuracy_ctree(n) = 1 - kfoldLoss(ctree_model);
end
% Optimal Min Leaf Size:
[~,best_ctree_idx] = max(accuracy_ctree);
optimal_leafs = leafs(best_ctree_idx)
```

```
optimal_leafs =
21.5443
```

```
% Plot Min Leaf Size (x) vs. cross-validated accuracy (y):
figure;
```

```
semilogx(leafs,accuracy_ctree, '-o');
xlabel('Min Leaf Size');
ylabel('Cross-Validates Accuracy');
title('Optimizing Leaf Size');
grid on;
```



```
% Retrain the classification tree on the full training set using optimal
% Min Leaf Size:
final_ctree = fitctree(X_train,Y1_train,'MinLeafSize',optimal_leafs);
ypred_ctree = predict(final_ctree,X_test);
```

A classification tree model is trained in a loop using 10 Min Leaf Size values between 10^1 and 10^2. The optimal Min Leaf Size is 21.5443, which is used to retrain the model on the full training set.

6) Model Comparison Using AUC:

```
% Lasso Logistic Regression:
[xx1SE,yy1SE,thr1SE,auctest_lasso_1SE,optROCpt] =
perfcurve(Y1_test,ypred_lasso_1SE,'1')
```

```
xx1SE = 423×1
     0
     0
     0
     0
     0
     0
     0
     0
```

```
      0
      0
      ⋮
yy1SE = 423×1
          0
    0.0042
    0.0084
    0.0126
    0.0167
    0.0209
    0.0251
    0.0293
    0.0335
    0.0377
      ⋮
thr1SE = 423×1
    0.9825
    0.9825
    0.9725
    0.9696
    0.9619
    0.9594
    0.9570
    0.9561
    0.9558
    0.9550
      ⋮
auctest_lasso_1SE = 
0.8849
optROCpt = 1×2
    0.2842    0.8870
```

```
% SVM:
[~,~,~,auctest_SVM] = perfcurve(Y1_test,ypred_SVM,'1')
```

```
auctest_SVM = 
0.7734
```

```
% Classification tree:
[~,~,~,auctest_ctree] = perfcurve(Y1_test,ypred_ctree,'1')
```

```
auctest_ctree = 
0.7434
```

Model AUCs:

- Lasso logistic (lambda1SE): 0.8849
- SVM (RBF): 0.7734
- Classification tree (~21.54): 0.7434

The lasso logistic regression has the highest AUC and is thus the best performing one according to this evaluation.

7) Model Comparison Using Confusion Matrix:

```
% Lasso Logistic Regression:
```

```
thr_opt1SE = thr1SE((xx1SE==optROCpt(1))&(yy1SE==optROCpt(2)));
ypred_lasso_1SE_binary = double(ypred_lasso_1SE >= thr_opt1SE);
[confMat_lasso_1SE,~] = confusionmat(Y1_test,ypred_lasso_1SE_binary);
TP_lasso_1SE = confMat_lasso_1SE(2,2);
TN_lasso_1SE = confMat_lasso_1SE(1,1);
FP_lasso_1SE = confMat_lasso_1SE(1,2);
FN_lasso_1SE = confMat_lasso_1SE(2,1);

accuracy_lasso_1SE = (TP_lasso_1SE + TN_lasso_1SE) /
(TP_lasso_1SE+TN_lasso_1SE+FP_lasso_1SE+FN_lasso_1SE);
precision_lasso_1SE = TP_lasso_1SE / (TP_lasso_1SE + FP_lasso_1SE);
recall_lasso_1SE = TP_lasso_1SE / (TP_lasso_1SE + FN_lasso_1SE);
f1_lasso_1SE = 2 * (precision_lasso_1SE * recall_lasso_1SE) / (precision_lasso_1SE
+ recall_lasso_1SE);

% SVM:
[confMat_SVM,~] = confusionmat(Y1_test,ypred_SVM);
TP_SVM = confMat_SVM(2,2);
TN_SVM = confMat_SVM(1,1);
FP_SVM = confMat_SVM(1,2);
FN_SVM = confMat_SVM(2,1);

accuracy_SVM = (TP_SVM + TN_SVM) / (TP_SVM+TN_SVM+FP_SVM+FN_SVM);
precision_SVM = TP_SVM / (TP_SVM + FP_SVM);
recall_SVM = TP_SVM / (TP_SVM + FN_SVM);
f1_SVM = 2 * (precision_SVM * recall_SVM) / (precision_SVM + recall_SVM);

% Classification tree:
[confMat_ctree,~] = confusionmat(Y1_test,ypred_ctree);
TP_ctree = confMat_ctree(2,2);
TN_ctree = confMat_ctree(1,1);
FP_ctree = confMat_ctree(1,2);
FN_ctree = confMat_ctree(2,1);

accuracy_ctree = (TP_ctree + TN_ctree) / (TP_ctree+TN_ctree+FP_ctree+FN_ctree);
precision_ctree = TP_ctree / (TP_ctree + FP_ctree);
recall_ctree = TP_ctree / (TP_ctree + FN_ctree);
f1_ctree = 2 * (precision_ctree * recall_ctree) / (precision_ctree + recall_ctree);

% Summary table of the three models:
model_names = {'Lasso 1SE','SVM (RBF)','Optimal Tree'};
metrics_table = table(...
    [accuracy_lasso_1SE;accuracy_SVM;accuracy_ctree],...
    [precision_lasso_1SE;precision_SVM;precision_ctree],...
    [recall_lasso_1SE;recall_SVM;recall_ctree],...
    [f1_lasso_1SE;f1_SVM;f1_ctree],...
    'VariableNames',{'Accuracy','Precision','Recall','F1Score'},...
    'RowNames',model_names)
```

```
metrics_table = 3×4 table
```

	Accuracy	Precision	Recall	F1Score
1 Lasso 1SE	0.8128	0.8030	0.8870	0.8429
2 SVM (RBF)	0.7796	0.7967	0.8201	0.8082
3 Optimal Tree	0.7536	0.7626	0.8201	0.7903

According to the summary table (metrics_table), the lasso logistic regression (lambda1SE) retains its position as the best performing model on the test data. It has the highest value in each of the four metrics, meaning that:

- The overall correctness of the model is 81,28% (accuracy).
- 80,3% of predicted positives are actually positives (precision).
- 88,7% true positive rate (recall)
- 84,29% harmonic mean of precision and recall.