**MATLAB**

**MACHINE LEARNING**

**ONRAMP**

**Data science assignment**

Name:Anish.D

SRN:PES1201801334

ROLL NO:29

SECTION:A

DEPARTMENT:ECE

CAMPUS:RR CAMPUS

SEM:5

**Unit 3**

Improving Predictive Models

Task 1:

Create a five-fold cross-validation partition named cvpt of the data in the table groupData. The response is the table variable group.

Task 2:

Create a discriminant analysis model named mdl for the data in the table groupData. Use the cross-validation partition cvpt.

Task 3:

Save the loss to kfLoss.

Task 4:

Create a discriminant analysis model named mdl2 for the data in the table groupData that uses leave-one-out validation. Then calculate and save the loss of the leave-one-out model to kfLoss2.

load data

cvpt = cvpartition(groupData.group,"HoldOut",0.3);

trainData = groupData(training(cvpt),:);

testData = groupData(test(cvpt),:);

holdoutMdl = fitcdiscr(trainData,"group");

holdoutLoss = loss(holdoutMdl,testData)

## Task 1

cvpt=cvpartition(groupData.group,"KFold",5);

## Task 2

mdl=fitcdiscr(groupData,"group","CVPartition",cvpt)

## Task 3

kfLoss=kfoldLoss(mdl);

disp("k fold loss :" + kfLoss);

## Task 4

mdl2=fitcdiscr(groupData,"group","Leaveout","on");

kfLoss2=kfoldLoss(mdl2);

disp("k fold loss :" + kfLoss2)

## Further Practice

mdl3=fitcdiscr(groupData,"group","KFold",10);

kfLoss3=kfoldLoss(mdl3);

disp("k fold loss :" + kfLoss3);



**4.2 Cross Validation**: (7/8) Heart Health

Task 1:

Using the data stored in the table heartData, create the following models in order. Use seven-fold cross-validation and calculate their loss values.

| **Classifier** | **Model Variable Name** | **Loss Variable Name** |
| --- | --- | --- |
| *k*-Nearest Neighbor | mdlKnn | lossKnn |
| Discriminant Analysis | mdlDa | lossDa |

What happens to the loss values if you use leave-one-out validation?

This code loads the data.

rng(0)

heartData = readtable("heartdataNumMulti.txt");

The script holdoutAnalysis calculates the loss for different models using holdout validation.

holdoutAnalysis

## Task 1

Create models and calculate loss

cvpt=cvpartition(heartData.HeartDisease,"KFold",7);

mdlKnn=fitcknn(heartData,"HeartDisease","CVPartition",cvpt);

lossKnn=kfoldLoss(mdlKnn)

disp("knn k fold loss:" + lossKnn);

mdlDa=fitcdiscr(heartData,"HeartDisease","CVPartition",cvpt);

lossDa=kfoldLoss(mdlDa)

disp("da k fld loss :" + lossDa);

Display the results.

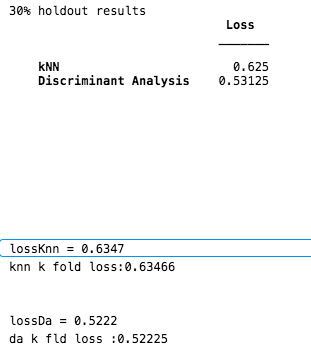
KFoldLoss = [lossKnn;lossDa];

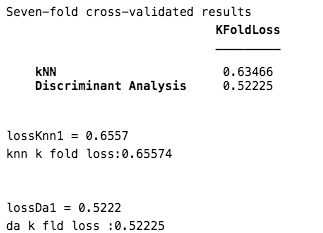
results = table(KFoldLoss);

results.Properties.RowNames = ["kNN" "Discriminant Analysis"];

disp("Seven-fold cross-validated results")

disp(results)





**4.3 Hyperparameter Optimization**: (3/4) Optimize kNN Hyperparameters

Task :

In the script, a kNN model is trained on dataTrain using the default property values.  
  
Train another kNN model mdl, and tune the hyperparameters by setting "OptimizeHyperparamters" to "auto". Use the training data dataTrain with response variable group.  
  
Run the script by clicking **Run** in the MATLAB Toolstrip. This optimization takes approximately 1 to 2 minutes to run.  
  
What did the optimization select for the property values?

mdl.NumNeighbors  
mdl.Distance

To view a solution and see the output, enter the following command:

open optimizeknnSoln.mlx

# Optimize k-NN Parameters

This code loads and partitions data.

load data

groupData

rng(1234)

pt = cvpartition(groupData.group,"Holdout",0.2);

dataTrain = groupData(training(pt),:);

dataTest = groupData(~training(pt),:);

This code fits a k-NN model and calculates the loss.

m = fitcknn(dataTrain,"group");

trainLoss = resubLoss(m)

testLoss = loss(m,dataTest)

## Optimize hyperparameters

mdl=fitcknn(dataTrain,"group","OptimizeHyperparameters","auto");

Calculate the loss.

trainLossOpt = resubLoss(mdl)

testLossOpt = loss(mdl,dataTest)

**4.3 Hyperparameter Optimization**: (4/4) Heart Health

Task 1:

Create a 10-fold cross-validation partition named cvpt of the training data heartTrain.  
  
Then use the **struct** function to create a structure named opt which sets the following optimization options:

* Set "CVPartition" to cvpt.
* Set "MaxObjectiveEvaluations" to 20.

# Heart Disease Analysis

This code loads and partitions the data.

heartData = readtable("heartdataNumMulti.txt")

rng(1234)

pt = cvpartition(heartData.HeartDisease,"Holdout",0.2);

heartTrain = heartData(training(pt),:);

heartTest = heartData(~training(pt),:);

This code fits a k-NN model to the training data and calculates the loss.

m = fitcknn(heartTrain,"HeartDisease");

trainLoss = resubLoss(m)

testLoss = loss(m,heartTest)

## Task 1

cvpt = cvpartition(heartTrain.HeartDisease,"KFold",10);

opt = struct("CVPartition",cvpt,"MaxObjectiveEvaluations",20);

## Optimize hyperparameters

mdl=fitcknn(heartTrain,"HeartDisease","OptimizeHyperparameters","NumNeighbors");

**4.4 Reducing Predictors - Feature Transformation**: (4/6) PCA

Task1:

Perform PCA on the predictive variables of the table dataOrig and save transformed data in scrs. Save the percentage explained as pexp.

Task 2:

Create a Pareto chart of the percent variance explained values.

Task 3;

Create a new variable named dataRed that contains only the first three columns of scrs.

Task 4:

Fit a 10-fold cross-validated Naive Bayes model to the data containing only the first three principal components. Name the result mdl. Then calculate the loss of mdl and save it as mdlLoss.

rng(0)

load data

dataOrig

This code fits a 10-fold cross-validated Naive Bayes model to the original data and calculates the loss.

mdlOrig = fitcnb(dataOrig,"R","KFold",10);

kfoldLoss(mdlOrig)

## Task 1

[~,scrs,~,~,pexp]=pca(dataOrig{:,1:end-1})

## Task 2

pareto(pexp)

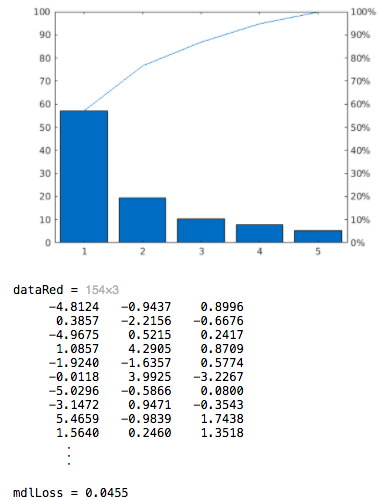
## Task 3

dataRed=scrs(:,1:3)

## Task 4

mdl=fitcnb(dataRed,dataOrig.R,"KFold",10);

mdlLoss=kfoldLoss(mdl)



**4.4 Reducing Predictors - Feature Transformation**: (6/6) Heart Health

Task 1:

Use PCA to transform your data. Save the principal component coefficients as pcs, the transformed data as scrs, and the percent explained as pexp.  
  
Create a Pareto chart of the percent variance explained values.

Task2:

Create a heat map of the absolute values of the first eight columns of the principal component coefficients.

Task 3:

Create a 10-fold cross-validated Naive Bayes model named mdl using the kernel distribution. Use only the first eight features returned from PCA.  
  
Calculate the loss of this model and save it to pcaLoss.

This code loads the data and splits the data into predictors (numData) and response (resp).

rng(0)

heartData = readtable("heartDataNum.txt");

vars = heartData.Properties.VariableNames(1:end-1)

numData = heartData{:,1:end-1};

resp = categorical(heartData.HeartDisease);

This code fits a 10-fold cross-validated Naive Bayes model using the kernel distribution to the original data and calculates the loss.

mdlOrig = fitcnb(numData,resp,"DistributionNames","kernel","KFold",10);

lossOrig = kfoldLoss(mdlOrig)

## Task 1

[pcs,scrs,~,~,pexp] = pca(numData);

pareto(pexp)

## Task 2

heatmap(abs(pcs(:,1:8)),"YDisplayLabels",vars)

## Task 3

mdl = fitcnb(scrs(:,1:8),resp,"DistributionNames","kernel","KFold",10);

pcaLoss = kfoldLoss(mdl)

**4.5 Reducing Predictors - Feature Selection**: (2/12) Built-in Feature Selection

**Task 1:**

Create a classification tree model named mdl without cross-validation. Then calculate the importance of each predictor and store the result in p.

Task 2:

Visualize the values in p using the bar function.

Task 3:

Create a logical vector toKeep whose values are true if the corresponding values of p are greater than 0.005 and false otherwise. Use toKeep to create a table dataPart which contains only the predictors with p greater than 0.005 and the response R.

Task 4:

Create a seven-fold cross-validated classification tree model named mdlPart which uses only the selected predictors. Calculate the loss and save it to partLoss.

rng(0)

load data.mat

data

This code fits a 7-fold cross-validated classification tree model to the original data and calculates the loss.

mdlFull = fitctree(data,"R","KFold",7);

fullLoss = kfoldLoss(mdlFull)

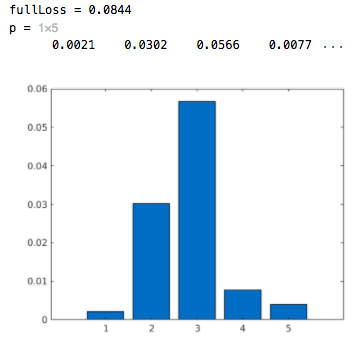
## Task 1

mdl=fitctree(data,"R");

p= predictorImportance(mdl)

## Task 2

bar(p)



## Task 3

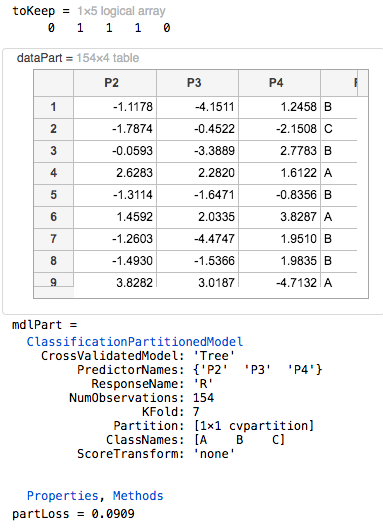
toKeep = p > 0.005

dataPart = data(:,[toKeep true])

## Task 4

mdlPart=fitctree(dataPart,"R","KFold",7)

partLoss=kfoldLoss(mdlPart)



**4.5 Reducing Predictors - Feature Selection**: (5/12) Sequential Feature Selection

Task 1:

Create an anonymous function named ferror that takes four inputs: XTrain, yTrain, XTest, and yTest, and returns the number of inaccurate predictions for yTest. Use a *k*-Nearest Neighbor classification model.

Task 2:

Use feature selection to create a logical vector named toKeep. The predictors are in the first five columns of the data table. The response is the last column, R.  
  
Use toKeep to create a table dataPart which contains only the selected predictors and the response R.

Task 3:

Create a 7-fold cross-validated *k*-nearest neighbor model named mdlPart with only the selected predictors. Calculate the loss and save it to partLoss.

This code loads and displays the data.

rng(0)

load data.mat

data

This code fits a 7-fold cross-validated k-NN model to the original data and calculates the loss.

mdlFull = fitcknn(data,"R","KFold",7);

fullLoss = kfoldLoss(mdlFull)

## Task 1

ferror = @(XTrain,yTrain,XTest,yTest) nnz(yTest ~= predict(fitcknn(XTrain,yTrain),XTest))

## Task 2

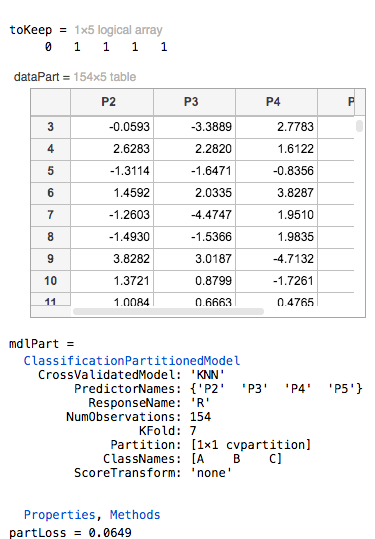
toKeep = sequentialfs(ferror,data{:,1:end-1},data.R)

dataPart=data(:,[toKeep true])

## Task 3

mdlPart=fitcknn(dataPart,"R","KFold",7)

partLoss = kfoldLoss(mdlPart)



**4.5 Reducing Predictors - Feature Selection**: (7/12) Creating Dummy Variables

Task 1:

Create an anonymous function named ferror that takes four inputs: XTrain, yTrain, XTest, and yTest, and returns the number of inaccurate predictions for yTest. Use a *k*-Nearest Neighbor classification model.

Task 2:

Use feature selection to create a logical vector named toKeep. The predictors are in the first five columns of the data table. The response is the last column, R.  
  
Use toKeep to create a table dataPart which contains only the selected predictors and the response R.

Task 3:

Create a 7-fold cross-validated *k*-nearest neighbor model named mdlPart with only the selected predictors. Calculate the loss and save it to partLoss.

This code loads and displays the data.

rng(0)

load data.mat

data

This code fits a 7-fold cross-validated k-NN model to the original data and calculates the loss.

mdlFull = fitcknn(data,"R","KFold",7);

fullLoss = kfoldLoss(mdlFull)

## Task 1

ferror = @(XTrain,yTrain,XTest,yTest) nnz(yTest ~= predict(fitcknn(XTrain,yTrain),XTest))

## Task 2

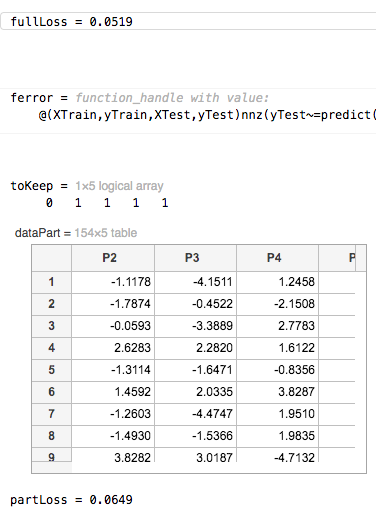
toKeep = sequentialfs(ferror,data{:,1:end-1},data.R)

dataPart = data(:,[toKeep true])

## Task 3

mdlPart = fitcknn(dataPart,"R","KFold",7);

partLoss = kfoldLoss(mdlPart)



**4.5 Reducing Predictors - Feature Selection**: (12/12) Heart Health – Feature Selection with Categorical Data

Task :

The open live script performs feature selection on a Naive Bayes classifier for the heart disease data set. You can look at the local function cattable2mat to see how dummy variables are created.  
  
Click **Run** in the MATLAB Toolstrip to execute the script. The feature selection is computationally intensive, and may take a long time to run. Thus, it may not complete.

Load the data.

heartData = readtable("heartDataAll.txt");

heartData = convertvars(heartData,12:22,"categorical");

Extract the response variable and make a partition for evaluation.

HD = heartData.HeartDisease;

rng(1234)

cvpt = cvpartition(HD,"KFold",10);

### Convert categorical predictors to numeric dummy variables

[X,XNames] = cattable2mat(heartData(:,1:end-1))

### Fit a Naive Bayes model to the full data

dists = [repmat("kernel",1,11),repmat("mvmn",1,10)];

mFull = fitcnb(heartData,"HeartDisease","DistributionNames",dists,"CVPartition",cvpt);

### Perform sequential feature selection

rng(1234)

fmodel = @(X,y) fitcnb(X,y,"DistributionNames","kernel");

ferror = @(Xtrain,ytrain,Xtest,ytest) nnz(predict(fmodel(Xtrain,ytrain),Xtest) ~= ytest);

toKeep = sequentialfs(ferror,X,HD,"cv",cvpt,"options",statset("Display","iter"));

% Which variables are in the final model?

XNames(toKeep)

% Fit a model with just the given variables

mPart = fitcnb(X(:,toKeep),HD,'Distribution','kernel','CVPartition',cvpt);

% Display loss values

lossFull = kfoldLoss(mFull)

lossPart = kfoldLoss(mPart)

Local function: cattable2mat

function [dummymat,dummyvarnames] = cattable2mat(data)

% Makes a matrix from a table, with categorical variables

% replaced by (numeric) dummy variables

vars = string(data.Properties.VariableNames);

idxCat = varfun(@iscategorical,data,"OutputFormat","uniform");

for k = find(idxCat)

% get list of categories

c = categories(data.(vars(k)));

% replace variable with matrix of dummy variables

data = convertvars(data,vars(k),@dummyvar);

% split dummy variable and make new variable names (by appending

% the category value to the categorical variable name)

varnames = vars(k) + "\_" + replace(c," ","\_");

data = splitvars(data,vars(k),"NewVariableNames",varnames);

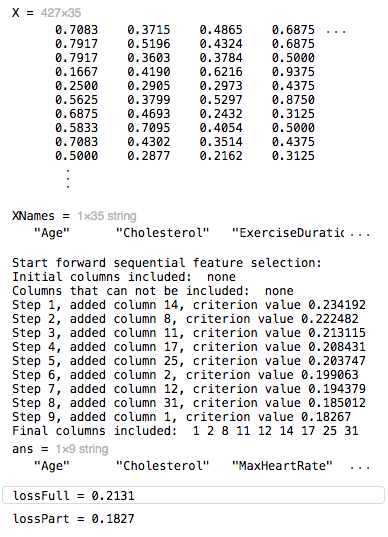
end

% return the numeric values

dummymat = data.Variables;

dummyvarnames = string(data.Properties.VariableNames);

end



**4.6 Ensemble Learning**: (5/7) Fit an Ensemble

Task 1:

Create an ensemble named mdlEns for the data in data with response "R". Use classification tree learners and "Bag" as the ensemble creation method.   
  
Calculate the resubstitution loss and save it to a variable named lossEns.

Task 2:

Create another ensemble of bagged trees named mdlEns2. Use 30 learners and the cross-validation partition cvpt. Calculate the *k*-fold loss and name it lossEns2.

This code loads and displays the data.

rng(1234)

load data

data

This code fits a classification tree and calculates the loss.

cvpt = cvpartition(data.R,"KFold",3);

mdlTree = fitctree(data,"R","CVPartition",cvpt);

lossTree = kfoldLoss(mdlTree)

## Task 1

mdlEns = fitcensemble(data,"R","Method","Bag");

lossEns = resubLoss(mdlEns)

## Task 2

mdlEns2 = fitcensemble(data,"R","Method","Bag",...

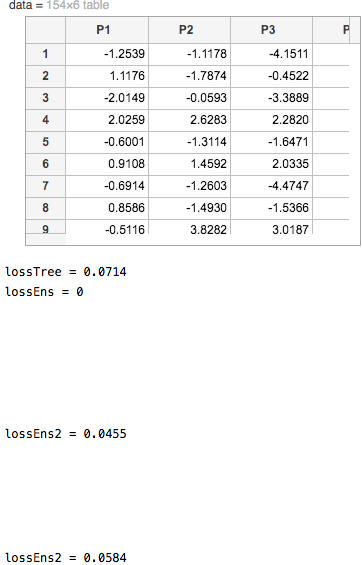
"NumLearningCycles",30,"CVPartition",cvpt);

lossEns2 = kfoldLoss(mdlEns2)

## Further Practice

mdlEns3 = fitcensemble(data,"R","Method","Bag","Learners","discriminant","NumLearningCycles",30,"CVPartition",cvpt);

lossEns2 = kfoldLoss(mdlEns3)



**4.6 Ensemble Learning**: (6/7) Use Learner Templates

Task 1:

Create an ensemble named mdlEns for the data in data with response "R". Use kNN learners and the cross-validation partition cvpt.   
  
Calculate the loss and save it to a variable named lossEns.

Task 2:

Create a template kNN learner knnmodel with the following properties:

| **Property Name** | **Property Value** |
| --- | --- |
| "NumNeighbors" | 5 |
| "DistanceWeight" | "inverse" |

Then create another ensemble named mdlEns2 using the template kNN learners and the cross-validation partition cvpt. Calculate the *k*-fold loss and name it lossEns2.

This code loads and displays the data.

rng(1234)

load data

data

This code partitions the data, fits a k-NN model and calculates the loss.

cvpt = cvpartition(data.R,"KFold",3);

mdlKNN = fitcknn(data,"R","NumNeighbors",5,...

"DistanceWeight","inverse","CVPartition",cvpt);

lossKNN = kfoldLoss(mdlKNN)

## Task 1

mdlEns= fitcensemble(data,"R","Learners","knn","CVPartition",cvpt)

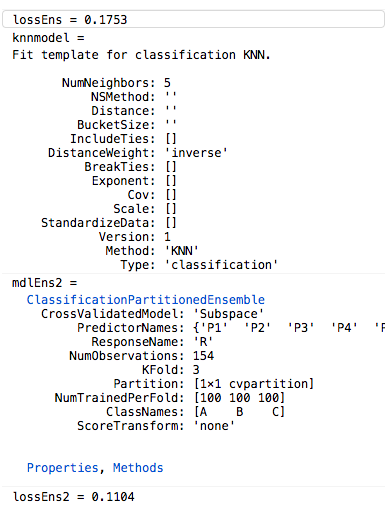
lossEns = kfoldLoss(mdlEns)

## Task 2

knnmodel = templateKNN("NumNeighbors",5,"DistanceWeight","inverse")

mdlEns2= fitcensemble(data,"R","Learners",knnmodel,"CVPartition",cvpt)

lossEns2 = kfoldLoss(mdlEns2)



**4.6 Ensemble Learning**: (7/7) Heart Health

Task 1:

Create an ensemble named mdlEns of 50 seven-fold cross-validated classification trees for the data in heartDataNum with response "HeartDisease". Use the cross-validation partition cvpt. Calculate the loss and name it lossEns.

Task 2:

Modify your tree ensemble mdlEns so that the seven-fold loss is below 0.3.

This code loads and displays the data.

heartData = readtable("heartDataAll.txt");

heartData = convertvars(heartData,12:22,"categorical");

heartDataNum = heartData(:,[1:11 end])

This code partitions the data and fits a tree model.

rng(0)

cvpt = cvpartition(heartDataNum.HeartDisease,"KFold",7);

mdlTree = fitctree(heartDataNum,"HeartDisease","CVPartition",cvpt);

lossTree = kfoldLoss(mdlTree)

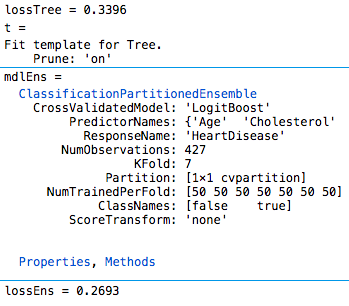
## Tasks 1 & 2

Fit tree ensemble and calculate loss.

t=templateTree("Prune","on")

mdlEns = fitcensemble(heartDataNum,"HeartDisease","Learners",t,"CVPartition",cvpt,"NumLearningCycles",50)

lossEns = kfoldLoss(mdlEns)



**4.7 Project - Improving Predictive Models**: (1/2) Wine Quality

Task 1:

Reduce the number of predictors in redData. You may use feature selection and/or feature transformation.  
  
Create a cross-validated model of any type with the reduced data. Name the model mdl, and calculate the loss mdlLoss.  
  
Try different models to see if you can get the loss value below 0.45, that of the full quadratic discriminant analysis model.

This code loads and displays the wine data and a trained 7-fold cross-validated quadratic DA model of redData.

rng(0)

load wineDataRed

redData

mdlFull

fullLoss = kfoldLoss(mdlFull)

**Reduce Predictors**

Fit model with fewer predictors. Target loss value less than 0.45.

dataset = redData(:,1:end)

cvpt = cvpartition(dataset.QCLabel,"KFold",7);

ferror = @(XTrain,yTrain,XTest,yTest) nnz(yTest ~= predict(fitcknn(XTrain,yTrain),XTest))

toKeep = sequentialfs(ferror,redData{:,1:end-1},redData.QCLabel)

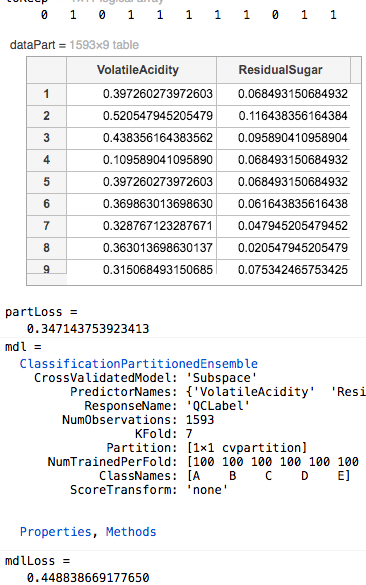
dataPart = redData(:,[toKeep true])

mdlPart = fitcknn(dataPart,"QCLabel","KFold",7);

partLoss = kfoldLoss(mdlPart)

mdl = fitcensemble(dataPart,"QCLabel","CVPartition",cvpt,"Learners","discriminant")

mdlLoss= kfoldLoss(mdl)



**4.7 Project - Improving Predictive Models**: (2/2) Credit Ratings

Task 1:

Create a reduced data set with three or fewer predictor variables. You may use feature selection and/or feature transformation. Then, create a seven-fold cross-validated tree model named mdl and calculate the loss, mdlLoss.

Task 2:

Create an ensemble of 50 seven-fold cross-validated classification trees using the bag method. Create your model with three or fewer predictors selected from the previous task. Name the ensemble mdlEns. Calculate the loss and name it lossEns.

rng(0)

load creditData

creditRatings

mdlFull

fullLoss = kfoldLoss(mdlFull)

## Task 1

Fit model with 3 or fewer predictors.

ferror = @(XTrain,yTrain,XTest,yTest) nnz(yTest ~=predict(fitcknn(XTrain,yTrain),XTest))

toKeep = sequentialfs(ferror,creditRatings{:,1:end-1},creditRatings.Rating)

dataPart = creditRatings(:,[toKeep true])

mdl = fitctree(dataPart,"Rating","KFold",7)

mdlLoss= kfoldLoss(mdl)

## Task 2

Fit ensemble with 3 or fewer predictors.

cvpt = cvpartition(creditRatings.Rating,"KFold",7)

mdlEns = fitcensemble(dataPart,"Rating","Method","Bag","NumLearningCycles",50,"CVPartition",cvpt)

lossEns = kfoldLoss(mdlEns)

**Unit 4**

Regression Methods

**5.2 Linear Models**: (4/11) Fit a Line

Task 1:

Create a regression model named mdl using the training data.

Task 2:

Use mdl to predict the response on the test data. Name the predicted response yPred.

load data

whos data dataTrain dataTest

plot(data.x,data.y,".")

## Task 1

mdl=fitlm(dataTrain);

## Task 2

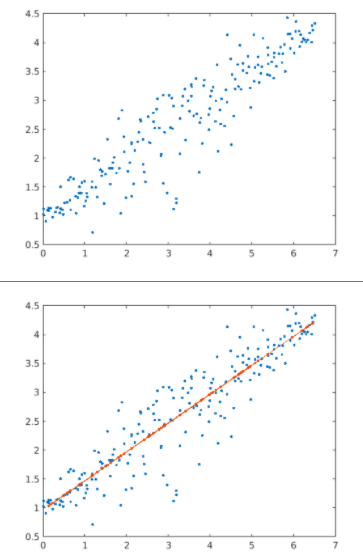
yPred=predict(mdl,dataTest);

## Further Practice

hold on

plot(dataTest.x,yPred,".-")

hold off



**5.2 Linear Models**: (6/11) Fit a Polynomial

Task 1:

Create a linear regression model named mdl which fits a line to the training data. Then, use the model to predict the response on the test data and name it yPred.

Task 2:

Modify your code so that mdl fits a quadratic polynomial to the training data.

Task 3:

Modify your code so that the influence of the outliers on the model mdl is reduced.

load data

whos data dataTrain dataTest

## Tasks 1, 2, 3

mdl=fitlm(dataTrain)

yPred=predict(mdl,dataTest);

mdl=fitlm(dataTrain,"quadratic","RobustOpts","on");

yPred=predict(mdl,dataTest);

Plot the results.

plot(data.x,data.y,".")

hold on

plot(dataTest.x,yPred,".")

hold off

legend("All Data","Predicted Response")

## Further Practice

mdl=fitlm(dataTrain,"quadratic","RobustOpts","cauchy");

yPred=predict(mdl,dataTest);

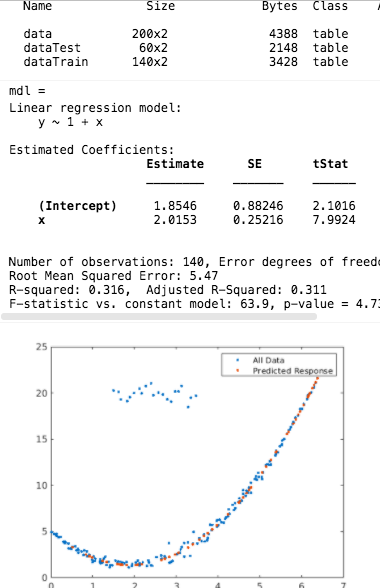
plot(data.x,data.y,".")

hold on

plot(dataTest.x,yPred,".")

hold off

legend("All Data","Predicted Response")



**5.2 Linear Models**: (8/11) Multivariable Linear Regression

TASK 1:

Create a regression model named mdl fitting the training data. Use a linear combination of the predictor variables for the model. Then predict the response on the test data and name it yPred.

Task2 :

Modify your code so that mdl fits a model where the response Y is a function of X1, X1^2, and X3.

load data

whos data dataTrain dataTest

data.Properties.VariableNames

## Tasks 1 & 2

mdl=fitlm(dataTrain)

yPred=predict(mdl,dataTest);

mdl=fitlm(dataTrain,"Y~X1+(X1^2)+X3")

plot(dataTest.Y,yPred,".")

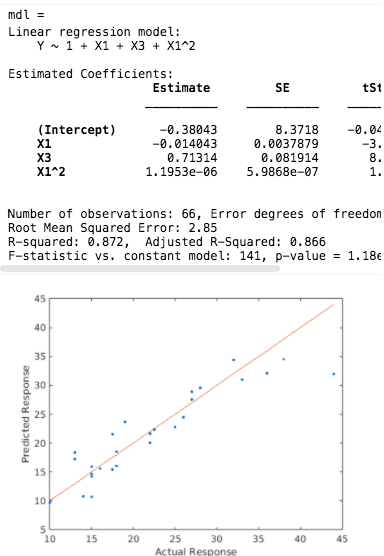
hold on

plot(dataTest.Y,dataTest.Y)

hold off

xlabel("Actual Response")

ylabel("Predicted Response")



**5.2 Linear Models**: (10/11) Multivariable Linear Regression with Numeric Arrays

Task 1:

Create a regression model named mdl fitting the training data. Use a linear combination of the predictor variables for the model.

Task 2:

Create a matrix XTrain13 to represent a regression formula for the training data with terms *x*1, *x*3,

and *x*1∗*x*3.  
Then create a regression model named mdl13 which fits the training data, using the model.

This code loads the data.

load data

whos

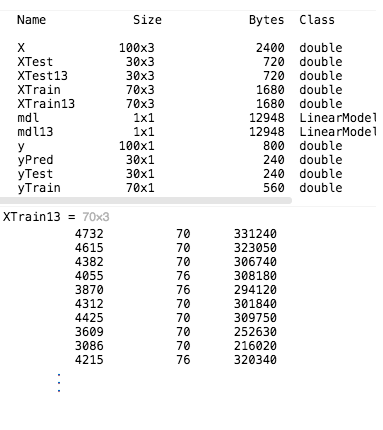
## Task 1

mdl=fitlm(XTrain,yTrain);

## Task 2

XTrain13=[XTrain(:,1) XTrain(:,3) XTrain(:,1).\*XTrain(:,3)]

mdl13=fitlm(XTrain13,yTrain);



**5.2 Linear Models**: (11/11) Predict Fuel Economy

Task 1:

Create a multivariate linear model named mdl with the training data. Then, use the model to predict the values, econPred.

Task 2:

Modify your code so that the model mdl uses "cauchy" for the "RobustOpts" property value.

load carEcon

whos carData carTrain carTest

## Tasks 1 and 2

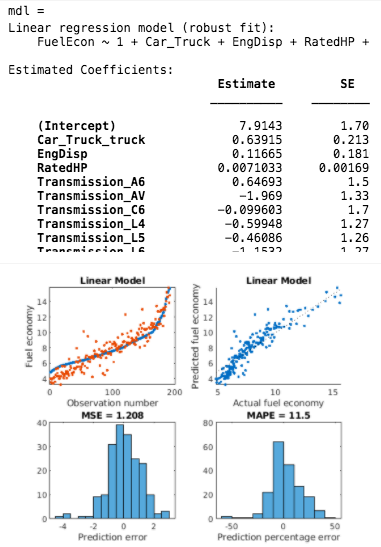
mdl=fitlm(carTrain);

econPred=predict(mdl,carTest);

mdl=fitlm(carTrain,"RobustOpts","cauchy")

econPred=predict(mdl,carTest);

evaluateFit(carTest.FuelEcon,econPred,"Linear Model")



**5.3 Stepwise Fitting**: (4/6) Stepwise Feature Selection

Task 1:

Perform a stepwise linear fit of the training data and name it mdl. Predict the response on the test data and name the predicted response yPred.

Task 2:

Extract the model formula and name it formula.

Task 3:

Modify your stepwise linear fit from Task 1. Provide the model spec value "purequadratic" so that the initial model has pure quadratic terms.

Code:

This code loads the data.

load data

whos data dataTrain dataTest

## Tasks 1 & 3

mdl = stepwiselm(dataTrain,"purequadratic");

yPred = predict(mdl,dataTest);

## Task 2

formula=mdl.Formula

Compare predicted and actual responses.

plot(dataTest.Y,yPred,".")

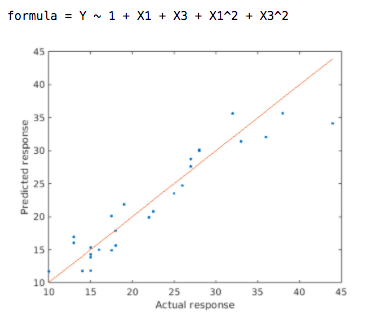
hold on

plot(dataTest.Y,dataTest.Y)

hold off

xlabel("Actual response")

ylabel("Predicted response")



**5.3 Stepwise Fitting**: (6/6) Predict Fuel Economy – Stepwise

Task 1:

Perform a stepwise linear fit of the training data to create mdl. Limit the model to use at most a constant plus linear terms. Predict the response on the test data and name the predicted response econPred.

Task 2:

Extract the root mean squared error (RMSE) of the model and name it RMSE.

Task 3:

Modify your stepwise linear fit from Task 1. Add an optional property so the model is chosen using the "aic" criterion.

Code:

## Tasks 1 & 3

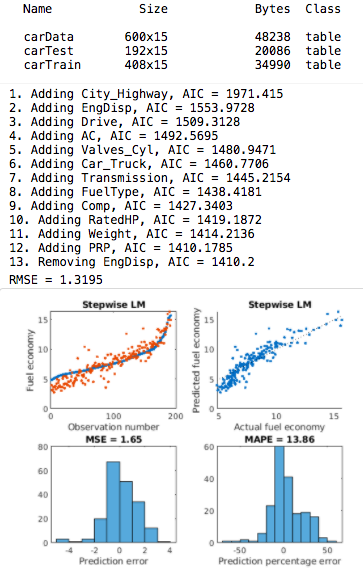
mdl = stepwiselm(carTrain,"Upper","linear","Criterion","aic");

econPred = predict(mdl,carTest);

## Task 2

RMSE = mdl.RMSE

evaluateFit(carTest.FuelEcon,econPred,"Stepwise LM"



**5.4 Regularized Linear Models**: (5/10) Fit a Ridge Regression Model

Task 1:

Create a vector lambda of integers from 0 to 100. Then perform ridge regression on the training data for the values in lambda. Return the coefficients in the scale of the original data, and name the coefficients b.

Task2:

Plot b against lambda to see how the coefficients change as λ increases.  
  
This plot is called the *ridge trace*.

Task 3:

Predict the response for the test data XTest. Name the result yPred.

Task 4:

Calculate mean squared error and name the result mdlMSE. Plot mdlMSE against lambda

Task5:

Find the smallest MSE and the index where it occurs. Name the results minMSE and idx, respectively.

This code loads and plots the data.

load data

whos data XTrain XTest yTrain yTest

scatter3(data.X1,data.X2,data.Y)

## Task 1

lambda = 0:100;

b = ridge(yTrain,XTrain,lambda,0);

## Task 2

plot(lambda,b)

## Task 3

yPred = XTest\*b(2:end,:) + b(1,:)

## Task 4

err = yPred - yTest;

mdlMSE = mean(err.^2);

plot(lambda,mdlMSE)

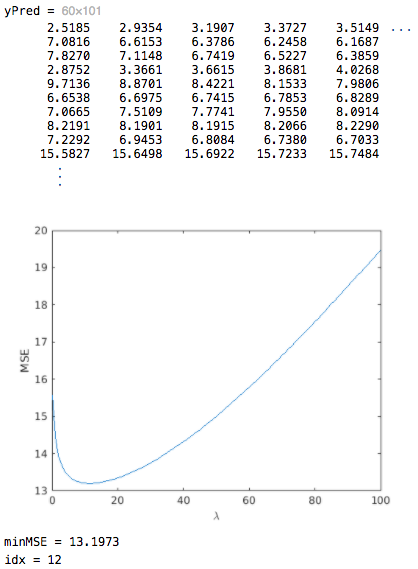
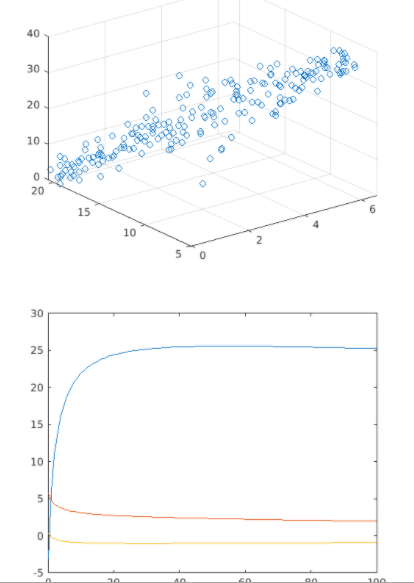
xlabel("\lambda")

ylabel("MSE")

## Task 5

Find smallest MSE

[minMSE,idx] = min(mdlMSE)



**5.4 Regularized Linear Models**: (8/10) Fit Lasso and Elastic Net Models

Task1 :

Perform lasso regression on the training data. For λ values, create a vector lambda of integers from 0 to 100 scaled by the number of observations. Name the fit coefficients b, and save information about the model fit as fitInfo.

Task 2:

Predict the response yPred for the test data XTest.  
  
Calculate the mean squared error mdlMSE, and plot mdlMSE against lambda.  
  
Find the smallest MSE and the index where it occurs. Name the results minMSE and idx, respectively.

Task 3:

Modify the call to the lasso function to use an alpha value of 0.4. This will perform elastic net regression.

load data

whos data XTrain XTest yTrain yTest

scatter3(data.X1,data.X2,data.Y)

## Tasks 1 & 3

lambda = (0:100)/length(yTrain);

[b,fitInfo]= lasso(XTrain,yTrain,"Lambda",lambda,"Alpha",0.4)

Plot coefficients

plot(lambda,[fitInfo.Intercept;b],"LineWidth",2)

legend("intercept","X1","X2")

xlabel("\lambda")

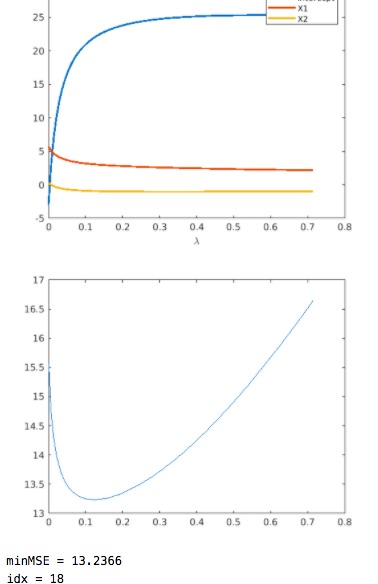
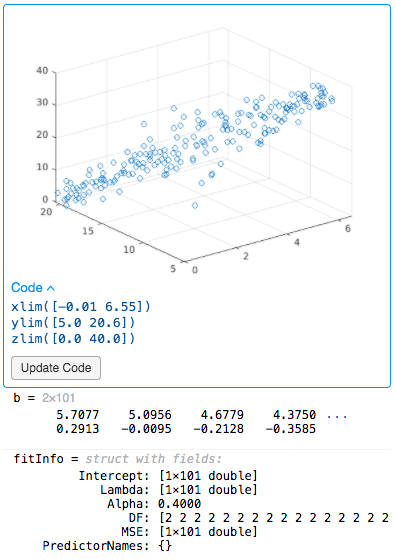
## Task 2

yPred = fitInfo.Intercept + XTest\*b;

mdlMSE=mean((yPred - yTest).^2);

plot(lambda,mdlMSE)

[minMSE,idx]=min(mdlMSE)



**5.4 Regularized Linear Models**: (10/10) Predict Fuel Economy – Regularized Model

Task 1:

Perform ridge regression on the training data. Create a vector lambdaR with values 0:300 to use as λ values. Return the coefficients in the scale of the original data, and name the coefficients bR.

Task 2:

Predict the response econPredR for the test data XTest, then calculate the MSE and name it MSER. Plot MSER against lambdaR.  
Find the smallest MSE and the index where it occurs, and name the results minMSER and idxR, respectively.

Task 3:

Now perform lasso regression on the training data. For λ values, create a vector lambdaL with values 0:300 scaled by the number of observations.  
Name the coefficients bL, and save information about the model fit as fitInfo.

Task 4:

For your lasso model, predict the response econPredL for the test data XTest, then calculate the MSE and name it MSEL. Plot MSEL against lambdaL.  
  
Find the smallest MSE and the index where it occurs, and name the results minMSEL and idxL, respectively.

This code loads the data.

load carEcon

whos XTrain XTest econTrain econTest

## Task 1

Fit ridge model

lambdaR = 0:300

bR = ridge(econTrain,XTrain,lambdaR,0)

## Task 2

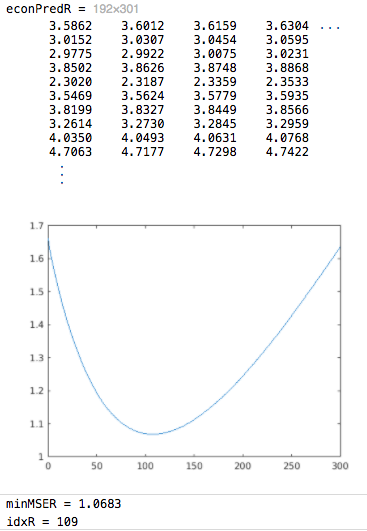
Calculate and plot MSE. Find smallest MSE.

econPredR = XTest\*bR(2:end,:) + bR(1,:)

MSER = mean((econPredR-econTest).^2);

plot(lambdaR,MSER)

[minMSER,idxR]=min(MSER)



## Task 3

Fit lasso model

lambdaL = (0:300)/length(econTrain);

[bL,fitInfo] = lasso(XTrain,econTrain,"Lambda",lambdaL)

## Task 4

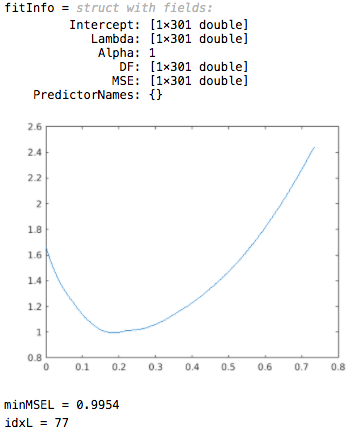
Calculate and plot MSE. Find smallest MSE.

econPredL = fitInfo.Intercept + XTest\*bL;

MSEL = mean((econPredL-econTest).^2);

plot(lambdaL,MSEL)

[minMSEL,idxL]=min(MSEL)



**5.5 SVMs and Trees**: (2/6) Fit Tree and SVM Models

Task 1:

Create a decision tree model named treeMdl using the training data.  
  
Calculate the model loss for the test data and name it treeLoss.  
  
Then, use the model to predict the response for the test data and name it yPred.

Task 2:

Create an SVM model named svmMdl using the training data. Calculate the loss svmLoss and find the predicted response yPred for the test data.

Task 3:

Create another SVM model named svmMdl2 which uses a "polynomial" kernel function. Calculate the loss svmLoss2. Again, find the predicted response yPred for the test data.

This code loads the data.

load data

whos data dataTrain dataTest

## Task 1

treeMdl = fitrtree(dataTrain,"y");

treeLoss=loss(treeMdl,dataTest);

disp(treeLoss)

yPred = predict(treeMdl,dataTest)

## Task 2

svmMdl= fitrsvm(dataTrain,"y");

svmLoss = loss(svmMdl,dataTest);

disp(svmLoss)

yPred = predict(svmMdl,dataTest)

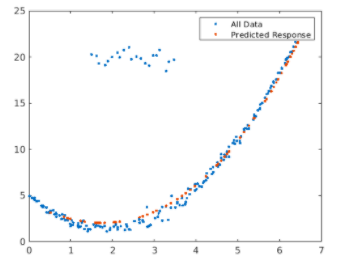
## Task 3

svmMdl2= fitrsvm(dataTrain,"y","KernelFunction","polynomial");

svmLoss2 = loss(svmMdl2,dataTest);

disp(svmLoss2)

yPred = predict(svmMdl2,dataTest)



**5.5 SVMs and Trees**: (4/6) Choose a Regression Model

Task 1:

Create either a tree or SVM regression model fitting the training data, name it mdl. Try to get the loss value below 0.15.

This code loads and plots the data.

load data

whos data dataTrain dataTest

plot(data.x,data.y,".")

## Fit a model

Fit a model

mdl = fitrsvm(dataTrain,"y","KernelFunction","gaussian");

Evaluate at test values.

mdlLoss = loss(mdl,dataTest)

yPred = predict(mdl,dataTest);

Plot the response.

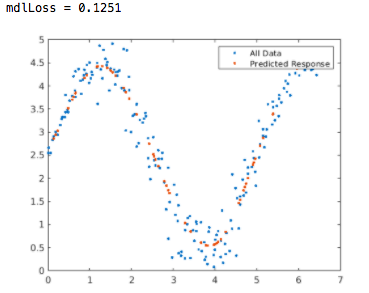
plot(data.x,data.y,".")

hold on

plot(dataTest.x,yPred,".")

hold off

legend("All Data","Predicted Response")



**5.5 SVMs and Trees**: (5/6) Predict Fuel Economy – Tree

Task 1:

Create a decision tree regression model with the training data and name it mdl. Calculate the loss of the model and name it mdlLoss.  
  
Use the model to predict the test values and store the predictions in a variable named econPred.

Task 2:

Modify your code to prune mdl to level 10.

Task 3:

Modify your code again. Remove the pruning step, and instead update mdl to set the "MinLeafSize" property to 5.

This code loads the data.

load carEcon

whos carData carTrain carTest

## Tasks 1,2,3 - Fit the model

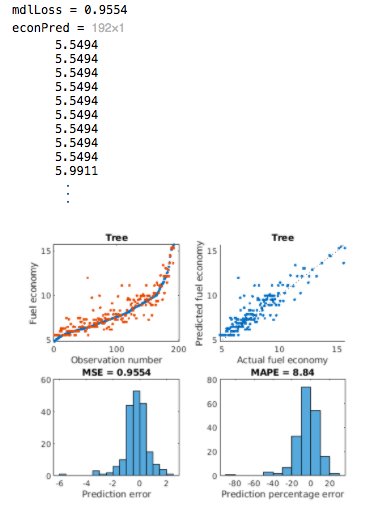
Fit a model and evaluate at test values.

mdl = fitrtree(carTrain,"FuelEcon","MinLeafSize",5)

mdlLoss = loss(mdl,carTest)

econPred = predict(mdl,carTest)

evaluateFit(carTest.FuelEcon,econPred,"Tree")



**5.5 SVMs and Trees**: (6/6) Predict Fuel Economy – SVM

Task 1:

Create an SVM regression model with the training data and name it mdl. Use a polynomial kernel function.  
  
Calculate the loss of the model and name it mdlLoss. Use the model to predict the response for the test values, and store the predictions in a variable named econPred.

Task 2:

Modify your code to set an option in [fitrsvm](http://www.mathworks.com/help/stats/fitrsvm.html" \l "input_argument_namevalue_standardize" \t "_blank) which will standardize the variables in carTrain before fitting the regression model.

This code loads the data.

load carEcon

whos carData carTrain carTest

## Tasks 1 and 2 - Fit the model

Fit a model and evaluate at test values.

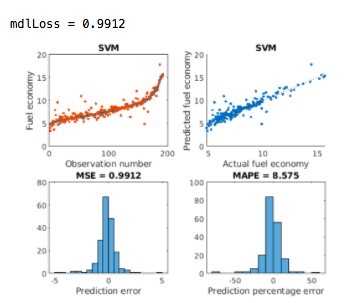
mdl = fitrsvm(carTrain,"FuelEcon","KernelFunction","polynomial","Standardize",true)

mdlLoss = loss(mdl,carTest)

econPred = predict(mdl,carTest)

Plot the results.

evaluateFit(carTest.FuelEcon,econPred,"SVM")



**5.6 Gaussian Process Regression**: (4/6) Fit a GPR Model

Task :

Create a GPR model named mdl using the training data. Then, use the model to predict the response for the test data and name it yPred. Calculate the loss, or mean squared error, of the predictions and name it mdlMSE.

load data

whos data dataTrain dataTest

## Task 1

Fit model and evaluate at test values.

mdl = fitrgp(dataTrain,"y");

yPred = predict(mdl,dataTest)

mdlMSE = loss(mdl,dataTest)

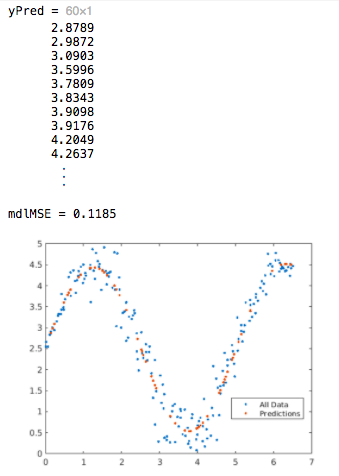
plot(data.x,data.y,".")

hold on

plot(dataTest.x,yPred,".")

hold off

legend("All Data","Predictions","Location","Best")



**5.6 Gaussian Process Regression**: (6/6) Predict Fuel Economy – GPR

Task 1:

Create a Gaussian process regression model with the training data and name it mdl.

This code loads the data.

load carEcon

whos carData carTrain carTest

## Task 1

Fit a model.

mdl= fitrgp(carTrain,"FuelEcon");

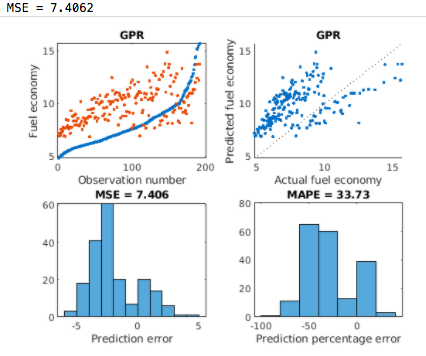
Evaluate at test values.

MSE = loss(mdl,carTest)

econPred = predict(mdl,carTest);

Plot the results.

evaluateFit(carTest.FuelEcon,econPred,"GPR")



**5.7 Project - Regression**

Task :

Use any regression technique with the training set to predict the outcome for the test set. Name the predictions yPred. Find a model where the MSE for the test set is less than 20.  
**Note**  Some models take longer to fit than others. Run your script to try different models. Your script should run in **under 30 seconds**. Submit your script after you have found a sufficient model.

This code loads the data.

load parkinsons

whos data dataTrain dataTest

## Fit regression model

Fit model then predict response for test data.

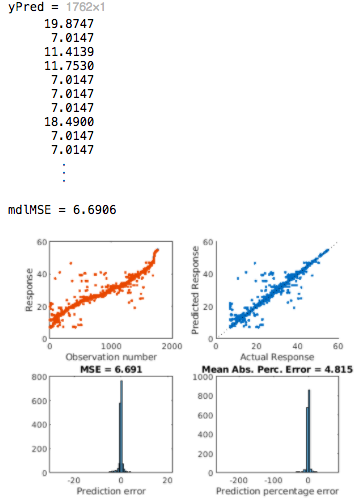
mdl = fitrtree(dataTrain,"total\_UPDRS");

yPred = predict(mdl,dataTest)

mdlMSE = loss(mdl,dataTest)

Compare predicted and actual responses.

evaluateFit(dataTest.total\_UPDRS,yPred,"")



**Unit 5**

Neural Networks

**6.3 Feed-Forward Networks**: (7/9) Use Commands to Create a Feed-Forward Classification Network

TASK 1:

Initialize a classification neural network named net with one hidden layer containing 15 neurons.  
  
Divide the data so that 70% is for training, 10% is for testing, and 20% is for validation.

Task 2:

Train the network net with the data in heartData and the target values in HDC. Save the training record to tr.  
  
Note that the target values are stored as categorical data, and samples for the data should be along the matrix columns rather than rows.

Task 3:

Make predictions on the test data. Store the predicted values in a variable named scoreTest.  
  
Then, create a vector named yPred which contains the row index value with the largest value for each column in the predicted matrix.

Task 4:

Plot the confusion matrix for the test data.

Task 5:

Calculate the percentage of misclassified predicted test values and save the value in a variable named validErr.

This code loads and displays the heart disease data set.

load heartDisease

whos HDC heartData

## Task 1

Initialize neural network

net = patternnet(15);

net.divideParam.trainRatio = 70/100;

net.divideParam.valRatio = 20/100;

net.divideParam.testRatio = 10/100;

## Task 2

Train the network

heartData = heartData'; % transpose samples

HD = dummyvar(HDC)'; % convert categorical to dummy variable

[net,tr] = train(net,heartData,HD);

## Task 3

Predict response

scoreTest = net(heartData(:,tr.testInd));

[~,yPred] = max(scoreTest);

## Task 4

Evaluate classification with confusion matrix

HDtest = HDC(tr.testInd);

yTrue = double(HDtest);

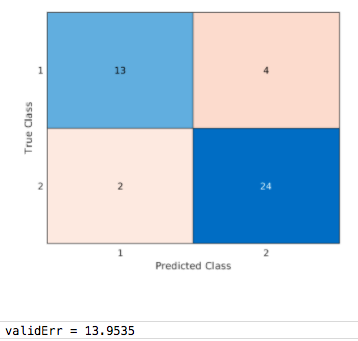
confusionchart(yTrue,yPred');

## Task 5

Determine validation error

HDtest = HDC(tr.testInd);

validErr = 100\*nnz(yPred' ~= double(HDtest))/length(HDtest)



**6.3 Feed-Forward Networks**: (9/9) Use Commands to Create a Feed-Forward Regression Network

Task 1:

Initialize a neural network named net with one hidden layer containing 15 neurons.  
  
Train the network with the data in carData and use the target values in econ. Save the training record in a structure named tr.  
  
Note that the samples for the data should be along the matrix columns rather than rows as input to the train function.

Task 2:

Make predictions named econPred on the test set.  
  
Then, you may use plotregression to show how well the predictions do.

Task 3:

Update net so there are two layers where the first layer has eight neurons and the second layer has 12 neurons.

Task 4:

Update net so the first layer uses the 'logsig' transfer function, and the second layer uses the 'radbas' transfer function.

load fuel

whos econ carData

## Tasks 1, 3, & 4

Initialize and train the neural network

net = fitnet([8 12]);

net.layers{1}.transferFcn = "logsig";

net.layers{2}.transferFcn = "radbas";

carData = carData';

econ = econ';

[net,tr] = train(net,carData,econ);

## Task 2

Predict response and evaluate network performance

econPred = net(carData(:,tr.testInd));

plotregression(econ(tr.testInd),econPred)

