function [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)

% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)

% Returns a trained classifier and its accuracy.

% Input:

% trainingData: A table containing the predictor and response

% columns

%

% Output:

% trainedClassifier: A struct containing the trained classifier. The

% struct contains various fields with information about the trained

% classifier.

%

% trainedClassifier.predictFcn: A function to make predictions on new

% data.

%

% validationAccuracy: A double containing the accuracy in percent. In

% the History list displays this overall accuracy score for

% each model.

%

% Use the code to train the model with new data. To retrain your

% classifier, call the function from the command line with your original

% data or new data as the input argument trainingData.

%

% For example, to retrain a classifier trained with the original data set

% T, enter:

% [trainedClassifier, validationAccuracy] = trainClassifier(T)

%

% To make predictions with the returned 'trainedClassifier' on new data T2,

% use

% yfit = trainedClassifier.predictFcn(T2)

%

% T2 must be a table containing at least the same predictor columns as used

% during training. For details, enter:

% trainedClassifier.HowToPredict

% Extract predictors and response

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'SVMacc', 'Arctanacc', 'Arccosacc', 'SVMang', 'Arctanang', 'Arccosang', 'SVMorien', 'Arctanorien', 'Arccosorien', 'SVMmag', 'Arctanmag', 'Arccosmag'};

predictors = inputTable(:, predictorNames);

response = inputTable.Activities;

isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false];

% Train a classifier (Weighted KNN)

% This code specifies all the classifier options and trains the classifier.

classificationKNN = fitcknn(...

predictors, ...

response, ...

'Distance', 'Euclidean', ...

'Exponent', [], ...

'NumNeighbors', 10, ...

'DistanceWeight', 'SquaredInverse', ...

'Standardize', true, ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

% Train a classifier (Gaussian SVM)

% This code specifies all the classifier options and trains the classifier.

template = templateSVM(...

'KernelFunction', 'gaussian', ...

'PolynomialOrder', [], ...

'KernelScale', 0.87, ...

'BoxConstraint', 1, ...

'Standardize', true);

classificationSVM = fitcecoc(...

predictors, ...

response, ...

'Learners', template, ...

'Coding', 'onevsone', ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

% Train a classifier (Gaussian Naïve Bayes)

% This code specifies all the classifier options and trains the classifier.

% Expand the Distribution Names per predictor

% Numerical predictors are assigned either Gaussian or Kernel distribution and categorical predictors are assigned mvmn distribution

% Gaussian is replaced with Normal when passing to the fitcnb function

distributionNames = repmat({'Normal'}, 1, length(isCategoricalPredictor));

distributionNames(isCategoricalPredictor) = {'mvmn'};

if any(strcmp(distributionNames,'Kernel'))

classificationNaiveBayes = fitcnb(...

predictors, ...

response, ...

'Kernel', 'Normal', ...

'Support', 'Unbounded', ...

'DistributionNames', distributionNames, ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

else

classificationNaiveBayes = fitcnb(...

predictors, ...

response, ...

'DistributionNames', distributionNames, ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

end

% Train a classifier (Bagged Tree)

% This code specifies all the classifier options and trains the classifier.

template = templateTree(...

'MaxNumSplits', 4379);

classificationEnsemble = fitcensemble(...

predictors, ...

response, ...

'Method', 'Bag', ...

'NumLearningCycles', 30, ...

'Learners', template, ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

% Train a classifier (Subspace KNN)

% This code specifies all the classifier options and trains the classifier.

subspaceDimension = max(1, min(6, width(predictors) - 1));

classificationEnsemble = fitcensemble(...

predictors, ...

response, ...

'Method', 'Subspace', ...

'NumLearningCycles', 30, ...

'Learners', 'knn', ...

'NPredToSample', subspaceDimension, ...

'ClassNames', categorical({'Climbing Down stairs'; 'Climbing up stairs'; 'Lying on a Flat surface'; 'Sitting'; 'Standing'; 'Walking'}));

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

knnPredictFcn = @(x) predict(classificationKNN, x);

svmPredictFcn = @(x) predict(classificationSVM, x);

naiveBayesPredictFcn = @(x) predict(classificationNaiveBayes, x);

baggedtreeensemblePredictFcn = @(x) predict(classificationEnsemble, x);

subspaceknnensemblePredictFcn = @(x) predict(classificationEnsemble, x);

trainedClassifier.predictFcn = @(x)

knnPredictFcn(predictorExtractionFcn(x)); svmPredictFcn(predictorExtractionFcn(x));

naiveBayesPredictFcn(predictorExtractionFcn(x));

baggedtreeensemblePredictFcn(predictorExtractionFcn(x));

subspaceknnensemblePredictFcn(predictorExtractionFcn(x));

% Perform cross-validation

partitionedModel = crossval(trainedClassifier.ClassificationKNN, 'KFold', 5);

partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 5);

partitionedModel = crossval(trainedClassifier.ClassificationNaiveBayes, 'KFold', 5);

partitionedModel = crossval(trainedClassifier.ClassificationBaggedTreeEnsemble, 'KFold', 5);

partitionedModel = crossval(trainedClassifier.ClassificationSubspaceKNNEnsemble, 'KFold', 5);

% Compute validation predictions

[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy

validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

-----------------------classifiers---------------------------------------

%% KNN

Mdl = fitcknn(X,Y)

%% Support vector machine

Mdl = fitcsvm(X,Y)

%% Naïve Bayesian

Mdl = fitcnb(X,Y)

%%Subspace KNN

Mdl = fitcknn(X,Y,'NumNeighbors',30,'Standardize',1; 'Method','Subspace','Learners',t);

%% Bagged Trees

Mdl = TreeBagger(30,X,Y,'OOBPrediction','On',...

'Method','classification')

---------------------Performance------------------------------------------

%Predict the labels of the training data.

predictedY = resubPredict(Mdl);

%Create a confusion matrix chart from the true labels Y and the predicted labels predictedY.

% Plot confusion matrix

cm = confusionchart(Y,predictedY);