```
%{
This function builds a multiclass svm model and evaluates it using
Posterior Probabilities

Arguments
- `labelledData`    -> the data labelled through one-hot encoding
                        (these labels are irrelevant for SVMs and will
                        be removed by the code below)
- `svmTargets`      -> the Nx1 unique string class labels for SVM
 targets
- `kernelFunction`  -> the kernel function to use for training the SVM
 model
- `boxConstraint`   -> the C parameter to use for training the SVM
 model.

Returns
- `accuracy`        -> classification accuracy obtained when the
 created
SVM model is tested with the unseen test set
%}

function [accuracy]=svmPosterior(labelledData, svmTargets,
 kernelFunction, boxConstraint)

    % ######### Set aside some of the data for testing ##########

    % extract only the inputs from the labelled data. The SVM targets
 are
    % separately given
    svmInputs = labelledData(:,1:end-5);

    % We want to shuffle both inputs and outputs while preserving the
    % correlation
    p = randperm(length(svmInputs));
    random_final_inputs = svmInputs(p, :);
    random_final_targets = svmTargets(p, :);

    % Standardise and normalise the input data.
    % normalize() normalises the data such that the center is 0 and
 the
    % standard deviation is 1. Function normalises each column by
 default.
    % 'range' makes all the values be between 0 and 1.
    random_final_inputs = normalize(random_final_inputs, 'range');

    % set some percentage of it aside for testing
    test_percent = 30;
    test_element_count =
 uint32((test_percent/100)*length(random_final_inputs));


    % Define which features to include in the input set.
```

```
    train_inputs = random_final_inputs(1:end-test_element_count,:);
  % Take all the rows, and all the 10 features as inputs. Could also
 use: inputs = dataSet(:,1:end-2).
    test_inputs= random_final_inputs(end-test_element_count+1:end,:);

    % Define the target set
    train_targets = random_final_targets(1:end-test_element_count, :);
    test_targets = random_final_targets(end-test_element_count
+1:end,:);


    % ######### Model Training ##################################

    % Create a SVM Model template to fit into fitcecoc()
    if kernelFunction == "polynomial"
        % if it's a polynomial kernel function then set the order to 2
 (i.e. quadratic)
        t =
 templateSVM('Standardize',true,'KernelFunction',kernelFunction, 'BoxConstraint',
 boxConstraint, 'PolynomialOrder', 2);
    else
        t =
 templateSVM('Standardize',true,'KernelFunction',kernelFunction, 'BoxConstraint',
 boxConstraint);
    end

    %{
    Train the ECOC classifier using the SVM template.
    - 'FitPosterior':   -> To transform classification scores to class
 posterior
                           probabilities (which are returned by
 predict or resubPredict)
    - 'ClassNames'      -> To specify the class order.
    - 'Verbose'         -> To display diagnostic messages during
 training
    %}
    fprintf("Training SVM binary learners...\n")
    SVMModel =
 fitcecoc(train_inputs,train_targets,'Learners',t,'FitPosterior',true,...
        'ClassNames',{'LGW','RA','RD','SiS','StS'},...
        'Verbose',1);

    % Predict the training-sample labels and class posterior
 probabilities.
    [label,~,~,Posterior] = resubPredict(SVMModel,'Verbose',1);

    % obtain a random sample from the data
    idx = randsample(size(train_inputs,1),10,1);

    % generate a table showing the predicted label and the
    % posterior probabilites of the sample data.
    % The 5 columns in the 'Posterior' columns correlate to:
 {'LGW','RA','RD','SiS','StS'}
    table(train_targets(idx),label(idx),Posterior(idx,:),...
```

```
        'VariableNames',{'TrueLabel','PredLabel','Posterior'})


    % ######### Model Evaluation ##################################

    fprintf("\nEvaluating the SVM model...\n\n")
    %{
    Predict the posterior probabilities for each instance in the test
 data.
    %}
    [~,~,~,TestSamplePosteriorRegion] = predict(SVMModel,test_inputs);

    % Convert the Nx5 TestSamplePosteriorRegion matrix into an Nx1
 array
    % with the index number of the class with the highest posterior
 prob.
    [~,I] = max(TestSamplePosteriorRegion, [],2);

    % Translate each class number into a string representing the class
    sets_for_labels = [{'LGW'} {'RA'} {'RD'} {'SiS'} {'StS'}];
    for ii=1 : length(I)
        targets_from_posterior_test_prediction(ii,1) =
 sets_for_labels(I(ii,1));
    end

    % Plot Confusion Matrix. This diplays the confusion matrix by
 default
    plotTitle = sprintf('%i Feature Confusion Matrix for SVM based on
 Posterior Prediction',size(svmInputs,2));
    cm =
 confusionchart(test_targets,targets_from_posterior_test_prediction,...
        'Title', plotTitle,...
        'RowSummary', 'absolute',...
        'ColumnSummary', 'absolute');

    % Calculate the classification accuracy from the confusion matrix
    % Need to first obtain the number of correct classifications, this
 will
    % be equal to the sum of the values in the diagonal of the CM
    confusionMatrixResults = cm.NormalizedValues;
    correct_predictions = 0;
    for ii=1 : length(confusionMatrixResults)
        correct_predictions = correct_predictions +
 confusionMatrixResults(ii,ii);
    end
    accuracy = (correct_predictions/length(test_targets))*100;

    fprintf("\n-------------\nSVM model accuracy using %i features: %f
\n", size(svmInputs,2), accuracy)
    fprintf("\nModel binary loss: %s\n-------------\n",
 SVMModel.BinaryLoss)
    % Binary Loss is quadratic since posterior probabilities are
    % being found by all the binary learners
end
```

Published with MATLAB® R2020a