Support Vector Machine

This script takes the following steps: 1) Load training and test data 2) Initialises results tables for experiments 3) Performs bayesian hyperoptimisation and then performs a grid search around estimated optimum parameters for each kernel type 4) Saves results of experiments in table 'SVM_Results' 5) Obtains optimum hyperparameters for model selection

1. Load Training and Test data

Load training and test data used in the modelling.

```
Training = load('Training.mat');
Testing = load('Testing.mat');
```

2. Initialise Results Tables

Initialise tables for storing results of experiments

```
optimisation_params = {'Kernel', 'BoxConstraint', 'KernelScale', 'MinObjective'};
svm_results = {'Kernel', 'BoxConstraint', 'KernelScale', 'Polynomial_order', 'Avg_Accuracy', 'Avg_Time'}
```

3. Grid Search and 10-fold Cross Validation

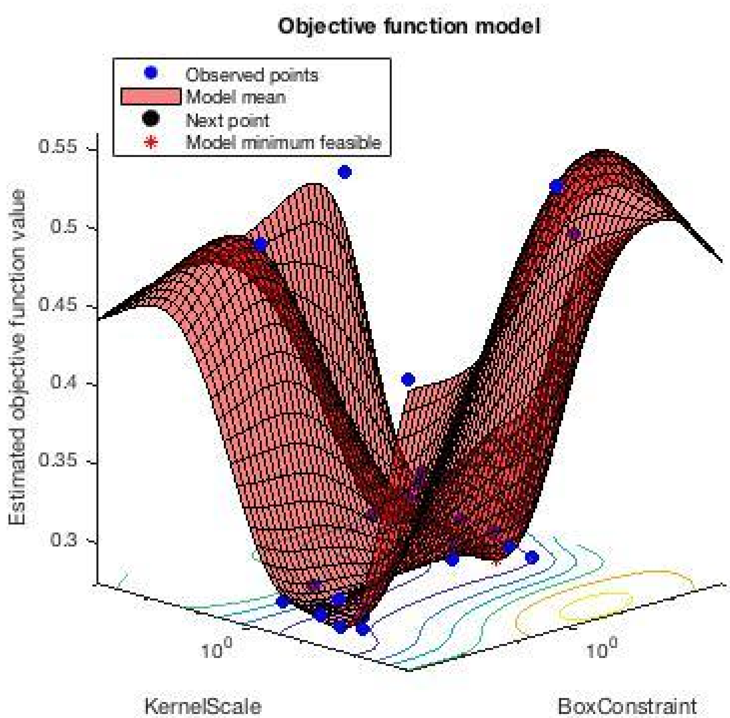
```
%This section carries out bayesian optimisation on each kernel type
%recursively. Each kernel will have a set of 'estimated' best
%hyperparameters, from which a grid search will be conducted around

%define kernels for experiments, order for polynomial
kernel = ["linear", "gaussian", "polynomial"];
%define order(only for 'polynomial' kernel)
poly_order = [2,3,4];

%for each kernel in kernel list
for i = kernel

    %if kernel is linear or gaussian
    if i ~= "polynomial"

        %train svm for kernel i using bayesian hyperoptimisation
        SVMModel_hyp = fitcsvm(Training(:, 1:42), Training(:, 43), 'KernelFunction', i,...
        'OptimizeHyperparameters','auto',...
        'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
        'expected-improvement-plus','ShowPlots',true)); % Bayes' Optimization

        %save the optimum hyperparameters to scale and constraint variables
        optimal_boxc = SVMModel_hyp.ModelParameters.BoxConstraint;
        optimal_scale = SVMModel_hyp.ModelParameters.KernelScale;
        min_objective = SVMModel_hyp.HyperparameterOptimizationResults.MinObjective

        %append the optimum hyperparameters values to the table 'optimisation_params'
        optimisation_params = [optimisation_params;[i, optimal_boxc, optimal_scale, min_objective]];
```

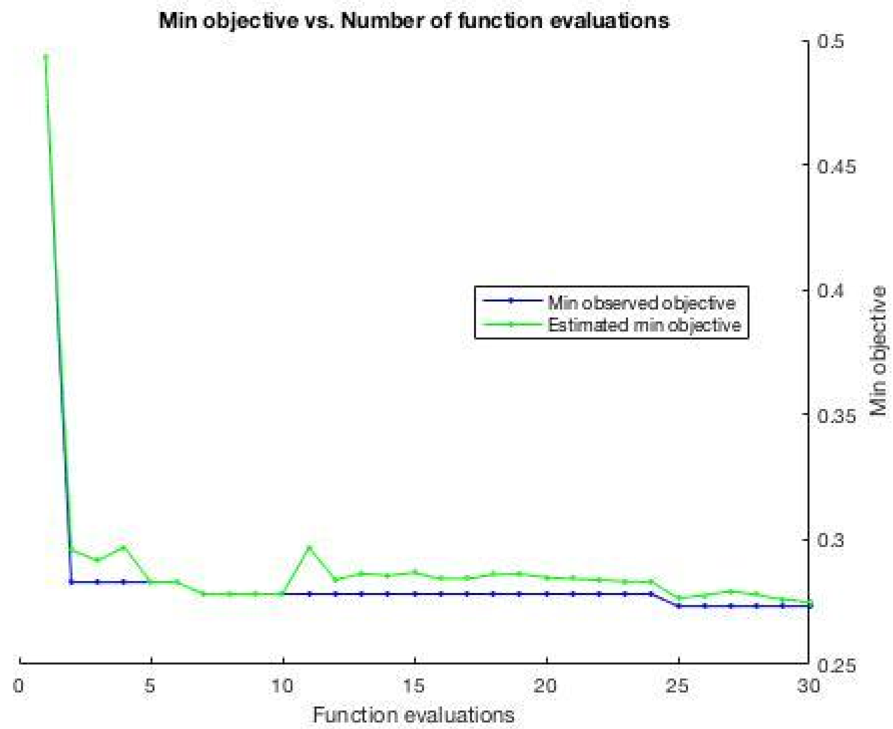
```
    %define grid search parameters based on hyperparameter results (25%
    %less and 25% more than the estimates from automatic)
    p_boxconstraint = [(optimal_boxc * 0.75), (optimal_boxc), (optimal_boxc * 1.25)];
    p_scale = [(optimal_scale * 0.75), (optimal_scale), (optimal_scale * 1.25)];

    %for each 'boxconstraint' optimised hyperparameter estimate
    for j = p_boxconstraint

        %for each 'scale' optimised hyperparameter estimate
        for k = p_scale

            list_acc = []; %initialise accuracy list
            list_time = []; %initialise time list

            %interface prompts
            fprintf('Kernel is %s. ',i);
            fprintf('Boxconstraint is %f. ',j);
            fprintf('Scale is set to %f.\n',k);

            %10-fold cross validation
            n=size(Training,1); %size of dataset
            c = cvpartition(n,'kfold',10); % 10-folds

            %for each fold
            for m = 1:10
                train_idx=training(c,m); %training indexes of fold-m
                train_data= Training(train_idx,:); %get data using indexes of fold-m
                val_idx=test(c,m); %validation indexes of fold-m
                val_data=Training(val_idx,:); %get data using indexes of fold-m
                x = train_data(:,1:42); %fold-m training predictors
                t = train_data(:,43); %fold-m training targets
                y = val_data(:,1:42); %fold-m validation predictors
                target_labels = val_data(:,43); %fold-m validation targets

                %train model on each fold using GS values
                tic %start timer
                SVMModel = fitcsvm(x,t,'Standardize',true,'KernelFunction',i,...
                    'KernelScale',k, 'BoxConstraint', j);
                time=toc; %end timer save time

                %get model accuracy
                test_accuracy = sum((predict(SVMModel, y) == target_labels))/length(target_labels)*1
                fprintf('k-fold %d: ', m)
                disp(test_accuracy)

                %append time and accuracy of fold to lists
                list_acc = [list_acc, test_accuracy];
                list_time = [list_time, time];

            end

            %calculate average of the folds
            average_acc = mean(list_acc)
            average_time = mean(list_time)

            %save metadata and average results of each model to 'svm_results'
            svm_results = vertcat(svm_results, {i, j, k, '-' ,average_acc, average_time});
        end
```

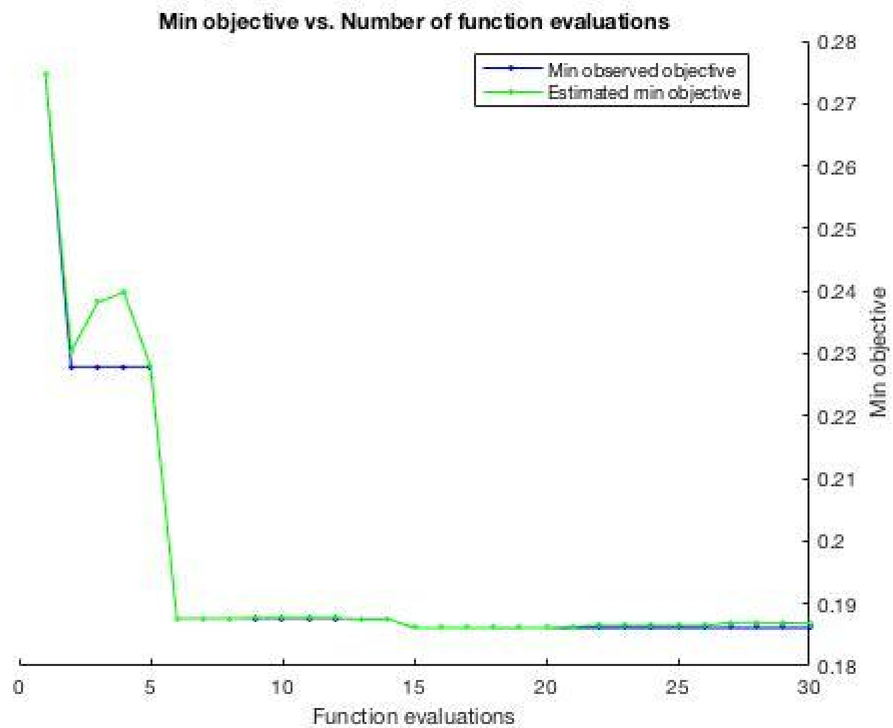
```
        end
    else %for 'polynomial' kernel

        %train svm for kernel i using automatic hyperoptimisation
        SVMModel_hyp = fitcsvm(Training(:, 1:42), Training(:, 43), 'KernelFunction', i,...
        'OptimizeHyperparameters','auto',...
        'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
        'expected-improvement-plus','ShowPlots',true)); % Bayes' Optimization

        %save the optimum hyperparameters to variables
        optimal_boxc = SVMModel_hyp.ModelParameters.BoxConstraint;
        optimal_scale = SVMModel_hyp.ModelParameters.KernelScale;
        min_objective = SVMModel_hyp.HyperparameterOptimizationResults.MinObjective

        %append the optimum hyperparameters values to the table 'optimisation_params'
        optimisation_params = [optimisation_params;[i, optimal_boxc, optimal_scale, min_objective]];

        %define grid search parameters based on hyperparameter results (25%
        %less and 25% more than the estimates from automatic)
        p_boxconstraint = [(optimal_boxc * 0.75), (optimal_boxc), (optimal_boxc * 1.25)];
        p_scale = [(optimal_scale * 0.75), optimal_scale, (optimal_scale * 1.25)];

        %for each 'boxconstraint' hyperparameter in GS
        for j = p_boxconstraint

            %for each 'scale' hyperparameter in GS
            for k = p_scale

                %for each polynomial order
                for o = poly_order

                    list_acc = []; %initialise accuracy list
                    list_time = []; %initialise time list

                    %interface prompts
                    fprintf('Kernel is %s. ',i);
                    fprintf('Boxconstraint is %f. ',j);
                    fprintf('Scale is set to %f.\n',k);
                    fprintf('Order is set to %f.\n',o);

                    %10-fold cross validation
                    n=size(Training,1); %size of dataset
                    c = cvpartition(n,'kfold',10); % 10-folds

                    %for each fold
                    for m = 1:10
                        train_idx=training(c,m); %training indexes of fold-m
                        train_data= Training(train_idx,:); %get data using indexes of fold-m
                        val_idx=test(c,m); %validation indexes of fold-m
                        val_data=Training(val_idx,:); %get data using indexes of fold-m
                        x = train_data(:,1:42); %fold-m training predictors
                        t = train_data(:,43); %fold-m training targets
                        y = val_data(:,1:42); %fold-m validation predictors
                        target_labels = val_data(:,43); %fold-m validation targets

                        %train model on each fold using GS values
                        tic %start timer
                        SVMModel = fitcsvm(x,t,'Standardize',true,'KernelFunction',i,...
```

```
                    'KernelScale',k, 'BoxConstraint', j, 'PolynomialOrder', o);
                time=toc; %end timer save time

                %get model accuracy
                test_accuracy = sum((predict(SVMModel, y) == target_labels))/length(target_label
                fprintf('k-fold %d: ', m)
                disp(test_accuracy)

                %append time and accuracy of fold to lists
                list_acc = [list_acc, test_accuracy];
                list_time = [list_time, time];

            end

            %calculate average of the folds
            average_acc = mean(list_acc)
            average_time = mean(list_time)

            %save metadata and average results of each model to 'svm_results'
            svm_results = vertcat(svm_results, {i, j, k, o ,average_acc, average_time});
        end
    end
    end
  end
end
```

The code takes a while to run, so some example outputs of the bayesian optimisation can be seen in the images below.







4. SVM Results Table

The results are saved to a .mat file so experiments only have to be run once.

```
%compile recorded results into table
SVM_Results = cell2table(svm_results(2:end, :));
SVM_Results.Properties.VariableNames = svm_results(1,:);

% save results to .mat file
save('smv_results.mat','svm_results');
```

The results of the SVM bayesian optimisation and grid search can be seen below.

Rank	Kernel	Box Constraint	Kernel Scale	Polynomial Order	10-fold Accuracy	10-fold Time
1	gaussian	1181.576	0.318	-	89.017	6.304
2	gaussian	708.946	0.530	-	88.834	4.841
3	gaussian	945.261	0.318	-	88.748	6.188
4	gaussian	945.261	0.424	-	88.711	5.407
5	gaussian	1181.576	0.424	-	88.699	5.112
6	gaussian	708.946	0.318	-	88.662	6.368
7	gaussian	1181.576	0.530	-	88.625	4.884
8	gaussian	708.946	0.424	-	88.406	5.197
19	polynomial	0.004	0.261	2	75.284	315.196
20	polynomial	0.006	0.261	2	75.210	306.299
21	polynomial	0.007	0.261	2	75.174	298.073
22	linear	57.138	1.013	-	72.718	305.103
23	linear	71.422	1.013	-	72.645	303.200
24	linear	42.853	1.013	-	71.753	307.035
25	linear	71.422	1.350	-	71.484	307.230
26	linear	57.138	1.350	-	71.289	304.812
27	linear	42.853	1.350	-	70.837	265.461
37	polynomial	0.006	0.348	4	51.423	227.972
38	polynomial	0.007	0.435	4	51.411	222.670
39	polynomial	0.006	0.435	4	50.764	210.770
40	polynomial	0.004	0.348	4	50.629	228.438
41	polynomial	0.007	0.348	4	50.446	232.939
42	polynomial	0.004	0.435	4	50.239	235.738
43	polynomial	0.007	0.261	4	49.896	232.991
44	polynomial	0.004	0.261	4	49.884	238.581
45	polynomial	0.006	0.261	4	49.566	243.933

5. Model Selection

```
%A new model is fitted according to the optimum hyperparameters in the
%previous section. This will be tested on the test data and final accuracy
%will be calculated to assess model generalisability

%find row with best accuracy, and the index
[best_acc, best_acc_idx] = max(SVM_Results.Avg_Accuracy);

%get row using the index
best_parameters = SVM_Results{best_acc_idx,:}
best_kernel = best_parameters(1);
best_scale = str2num(best_parameters(3));
best_boxc = str2num(best_parameters(2));

%use best parameters to fit optimal model
Best_SVMModel = fitcsvm(Training(:, 1:42),Training(:, 43),'Standardize',true,'KernelFunction',best_kerne
```

```
    'KernelScale',best_scale, 'BoxConstraint', best_boxc)

%fit make predictions on test data
predictions = predict(Best_SVMModel, Testing(:, 1:42));

%compare predictions to test
test_accuracy = sum(predictions == Testing(:, 43))/length(Testing(:, 43))*100
```