Programming Exercise # 4
Support Vector Machine (SVM) and kernels

1 Introduction

The purpose of this assignment is to familiarize you with the use of support vector machines. You will explore the use of different kernels, and use a methodology for selecting the parameters required by the different kernels. There are some freely available libraries with efficient implementatios of SVM's. For this assignment you will be using *libsvm*. The required files are already included in the *zip* folder of the assignment.

1.1 libsvm

libsvm is a library that allows us to use SVM in a simple and effective way. The two main functions in this library are: `svmtrain` and `svmpredict` for training and predicting respectively.
Usage:

```
   model = svmtrain(labels, featuresSet, param)
   [predictions, accuracy, prob_est] = svmpredict(labels, featureSet,
model)
```

The inputs of `smvtrain` are: a vector `labels` of length m, where m is the number of samples in the training set. The i^{th} entry of the vector *labels* is an integer that represents the true class of the i^{th} sample. `featuresSet` is a matrix of size $m \times n$, where n is the number of features. `param` is a string that specifies the parameters of the SVM, such as type of the kernel to be used, the cost parameter, etc. The parameters that we are going to set in this assignment are:

-t kernel_type : set type of kernel function (default 2)

  0 – linear

  1 – polynomial

  2 – radial basis function

-c cost: set the cost parameter C (default 1).

-d degree: set degree when using polynomial kernels (default 3).

-g sigma: set the value of sigma used in the RBF kernel (default 1/num_features).

For example, if we want to train a SVM with a polynomial kernel, with the parameters: degree = 5, C = 8, we would call the function as:

```
model = svmtrain(labels, trainSet, '-t 1 -d 5 -c 8');
```

The output of this function is a structure `model` with all the information needed to make predictions (i.e., weights vector w, bias b, etc.).

After training the model, we can use the function `svmpredict` to classify new samples (i.e., test set). The inputs of this function are: a vector `labels` of length k, where k is the number of samples in the test set. The i^{th} entry of the vector *labels* is an integer that represents the true class of the i^{th} sample. If you are predicting unlabeled instances, then `labels` might be a vector of length m with zeros in all the entries. `testSet` is a matrix of size $k \times n$, where k is the number of test samples and n is the number of features. The outputs of `svmpredict` are: a vector `predictions` of length k that contains the predicted class for every instance. `accuracy` is a vector of length 3 where the first entry is the accuracy (percentage of correctly classified instances) of the predictions (using the input vector `labels` as the ground truth), and a vector of length k `prob_est` that we will not be using in this assignment.

The full documentation of this library can be consulted at: `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

1.2 Installing libsvm

The process for installing libsvm is straightforward. Simply unzip the file `libsvm-3.20.zip`. Then, from inside MATLAB or Octave, go to the folder `libsvm-3.20\matlab` and type `make`. Finally, add the folder `libsvm-3.20\matlab` to the path. You can use the function `addpath()` to perform this step.

2 Programming exercise - Part 1

In this part of the exercise you will explore the use of three different kernels: linear, polynomial, and radial basis function (RBF). You will use a two-dimensional dataset that is not linearly separable, and you will also visualize and compare the decision boundaries created by the support vector machines when trained with different kernels.

The file `PE_4_Part_1.m` will guide you through the first part of the assignment. This file loads a training set (`X_train, Y_train`), and a matrix named `folds`. Similar to the previous assignment, the matrix `folds` holds information regarding the arrangement of samples for performing cross-validation (to create the sets S_{-i}, S_i). The first column of `folds` contains a random permutation of the X indexes (i.e., the index of a row of X which represents one instance/sample in the dataset). The second column specifies the fold to which this sample is assigned. For example, if the first entry of the matrix is $[1, 3]$ it means that the row 1 of X belongs to the 3^{rd} fold.

You can use the function `generateSets.m` to extract from `X` the training and test folds for cross-validation as well as the respective labels. This function takes as input 4 variables: `X`, `Y`, `folds`, and `iterNum`, where `X` is the part of data on which we want to perform cross-validation, `Y` is a vector of the respective labels, `folds` is that same as described in the previous paragraph, and `iterNum` specifies the fold that is assigned as the test fold (and of course, the rest of the folds are assigned as training).

2.1 Complete file: `findBestModel.m`

You now need to complete the file `findBestModel.m`, which defines a function with the same name. This function receives 4 inputs:

kernelType	A string with the values 'linear', 'rbf', or 'poly' that indicates the type of kernel that you want to use.
X_train	Matrix of $m \times n$, where m is the number of samples in the training set, and n is the number of features.
Y_train	Vector of length m, where m is the number of samples. It contains the true class of every instance in the training set.
folds	A matrix of $m \times 2$ with the characteristics described in the previous section.

With this code you will perform 5-fold cross-validation in order to find the best set of parameters for the given kernel. Try the following parameters:

Cost	$[2^{-5}, 2^{-3}, 2^{-1} \dots 2^{15}]$
Sigma	$[2^{-15}, 2^{-13}, 2^{-11} \dots 2^{3}]$
Degree	$[2, 3, 4, 5, 6, 7, 8]$

You should try all possible combinations of these values with the function `svmtrain`, and choose the parameters which return the highest cross-validation accuracy (for the linear kernel you only need to set the Cost parameter, for the polynomial kernel you need the Cost and the Degree, for the RBF kernel you need the Cost and the Sigma). Your function will output the highest cross-validation accuracy, along with the parameters that you selected (e.g., '-t 0 -c 32768'). Note that this cross-validation accuracy is not a good estimate of the expected performance of your learning algorithm; you would need to perform internal cross-validation instead (which is not part of this assignment). Use these selected parameters to train a model based on the entire training set. You will also return this model.

Once you complete the file `findBestModel.m`, you will be able to graph the decision boundaries created by the SVM with different kernels. The code to create this graphs is provided. You should have graphs similar to the ones shown in figures 1, 2 and 3. (Note: this graphs were created using Matlab 2015a running on a Linux machine, different versions of Matlab/Octave running on different operating systems might create slightly different graphs).

3 Programming exercise - Part 2

In this part of the exercise you will use the information stored in the file `PE_4_Part_2.mat`. This file contains two matrices: `trainLabels` and `trainSet`. You will use this matrices to train a SVM using the RBF kernel (you will need to decide what parameters to use).

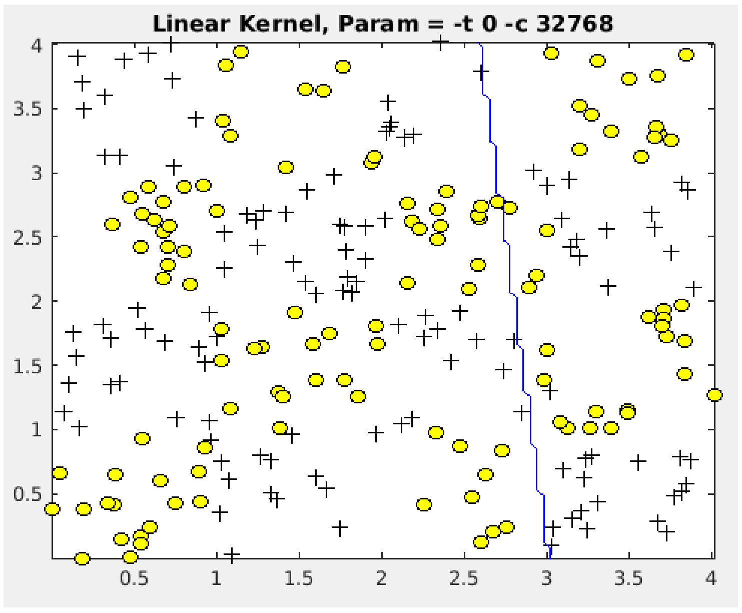

Figure 1: Decision surface created by a SVM using a linear kernel.

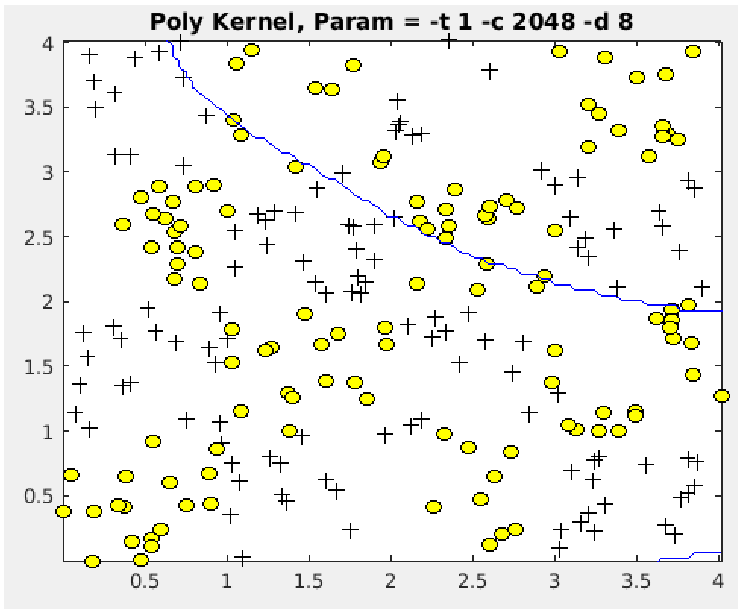

Figure 2: Decision surface created by a SVM using a polynomial kernel.

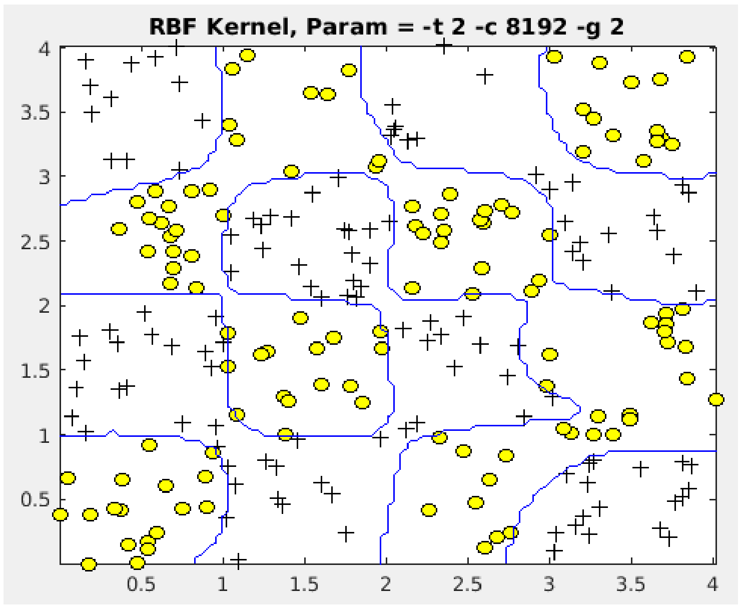

Figure 3: Decision surface created by a SVM using a RBF kernel.

Now complete the file `predictUsingSVM.m`. The function defined inside this file should receive as input an unlabeled dataset, and it should return the class predictions for all its samples. Note that the dataset used in this part of the exercise has 50 features, therefore, its visualization is not possible.

Your function will be evaluated using a hold-out set (which is not included in the assignment files). You will receive full marks if the percentage of instances correctly classified on this hold-out set is above 81%.