Imperial College
London

**BMI - Brain Machine Interfaces – Dr Claudia Clopath & Dr Paul Chadderton**

***Practice Assignment: Support Vector Machines (SVMs) in a binary classification task***

## Before you begin

The main idea of this assignment is to give you some insights and concrete examples of machine learning methods that might help you to begin the BMI challenge. Since MATLAB offers multiple machine learning toolkits and functions, this assignment is not about the implementation of methods presented in the lectures but rather their application.
Here, we are focusing on SVMs, Support Vector Machines,which is a supervised learning method mostly used for binary classification. When the data is linearly separable, SVMs can be easily trained in order to optimize the margin between two classes. When the data is not linearly separable, kernels (function computing the similarities between two data points) can be used to map the data into a feature space where the classes are now linearly separable.
We propose you to build a spam classifier using linear and non-linear SVM classifiers. You will be using three different kind of datasets that are composed of two types of emails: unwanted and wanted emails. According to the dataset, you might want to use different types of kernels.

We have provided several files for you to use. If you are willing to implement your own functions, feel free to do so. Those are either datasets or functions modified based on exercises from the open-source Machine Learning course by Prof. Andrew Ng from Stanford University. If you are interested, the entire course is available here: `https://www.coursera.org/learn/machine-learning`. You can familiarize yourself with each file to get an idea of what it does and how to use it.

Here is a brief listing:

1. `svmTrain.m` - training of an SVM-based classifier.
2. `svmPredict.m` - classification based on the previously trained model.
3. `linearKernel.m` - implementation of a linear kernel function.
4. `gaussianKernel.m` - implementation of a Gaussian radial basis function.
5. `plotData.m` - simple function for visualization of the datasets
6. `visualizeBoundary.m` - function used to visualize the boundary separating two classes.
7. `data1/2/3.mat` - datasets.

## Task: Linear SVM classifier

In this section you will use an SVM to solve rather simple (almost linearly separable) binary classification problem. Here, we restrict our classifier to compute only linear classification boundary by using linear kernel function.

1. Load the dataset `data1.mat`. You should see two variables in your workspace $X$ and $y$. Those are respectively the coordinates of datapoints and the labels associated to each datapoint (spam, non-spam). Visualize the datasets including different markers/colors for each class in this classification problem. You can use `plotData.m` function or do it in any other, feasible, way.

2. Train the linear SVM classifier to that will solve this classification problem. To do this you will need to run the following line:

```
model = svmTrain(X, y, C, @linearKernel);
```

   Where:

   - `X`, `y` - variables from the loaded dataset.
   - `C` - misclassification penalty. (default value: 1.0)
   - `linearKernel` - linear kernel function (already implemented).
   - `model` - data structure containing parameter of the trained linear SVM classifier.

   The misclassification penalty `C` is a regulariser parameter that is adding more or less constrains on the classification. For small `C`, the algorithm will prioritize on classifying correctly large groups of points which can lead to the misclassification of single points. You will have less contrains on your classification. For large `C`, the algorithm will prioritize on classifying correctly all the points without exception, and your classification will be more constrained. You can visualize the classification boundary with different misclassification penalty values to perceive the influence of this regulariser in practice.

3. Now, visualize the classification boundary of you linear SVM. For this you run the following line:

```
visualizeBoundary(X, y, model);
```

   Where:

   - `X`, `y` - variables from the loaded dataset.
   - `model` - linear SVM classifier parameters from the previous paragraph.

   Did the classification boundary separated two datasets and was the classifier able to achieve 100% accuracy? If not, why was that the case? What are possible limitations of the linear SVM classifier? Does visualizing the classification boundary is the best way to evaluate the classifier's performance in this problem? Propose an alternative way to evaluate the classification. Note that training and testing on the same data may yield biased results.

## Task: Non-linear SVM classifier

In this section you will face much more complex, highly non-linear classification problem. However, you will notice that simply changing the kernel function to a Gaussian radial basis function (RBF) can solve what seemed to be very difficult classification problem. The clue to solving this problem is to perform the so-called 'Kernel trick'. Here the kernel function performs remapping of datapoints' original features (here two coordinates) into a different feature space. For example, the RBF kernel allows to measure similarity (in space) between datapoints in the meaning the closer they are in 2D space the bigger the value of the RBF function. For more information about the kernel trick, please have a look at the paper written by Martin Hoffman (2006) attached to this assignment.

1. Load dataset `data2.mat`. The same as before you should see two variables in your workspace $X$ and $y$. Visualize the datasets. You can use `plotData.m` function or complete this task in any other way.

2. Train the linear SVM classifier to that will solve this classification problem. To do this you will need to run the following line:

```
model= svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));
```

   Where:

   - `X`, `y` - variables from the loaded dataset.
   - `C` - misclassification penalty. (default value: 1.0)
   - `sigma` - parameter of RBF kernel function describing its spread (default value: 0.1)
   - `gaussianKernel` - Gaussian RBF kernel function (already implemented)
   - `model` - data structure containing parameter of the trained RBF-based SVM classifier.

3. Now visualize the classification boundary of you non-linear SVM. For this you run the following line:

```
visualizeBoundary(X, y, model);
```

   Where:

   - `X`, `y` - variables from the loaded dataset.
   - `model` - RBF kernel SVM classifier parameters from the previous paragraph.

   Did the classification boundary separated two datasets and was the classifier able to achieve 100% accuracy? If not, why was that the case? What are the possible limitations and downsides of the RBF kernel SVM classifier? Does visualizing the classification boundary is the best way to evaluate the classifier's performance in this problem? Propose an alternative way to evaluate the performance.

## Task: It's your turn!

By this time we have provided you with handful of tools and methods based on the SVM-based classification framework. Now you should use either (or both) of the methods presented above to classify data from the third dataset `data3.mat`. Compare the performance of SVMs based on linear and RBF kernels and choose which one is feasible for this task. Explain your choice and methodology you used, report the best classification accuracy you were able to obtain, list the optimal parameters for your classifier and visualize the classification boundary.