# 3 Non-linear Classification Techniques

In classification problems, we typically have a dataset $D = \{(\mathbf{x}^1, y_1), \ldots, (\mathbf{x}^M, y_M)\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ is the $i$-th $N$-dimensional data point (or feature vector) from $M$ samples and $y_i \in \{-1, 1\}$ is the categorical outcome (or class label) corresponding to each data point. The goal is then to learn a mapping function $y = f(\mathbf{x}) : \mathbb{R}^N \to \mathbb{Z}$, such that, given a new sample (or query point) $\mathbf{x}' \in \mathbb{R}^N$ we can predict its label; i.e. $y' = f(\mathbf{x}') \in \{-1, 1\}$ for the binary classification case. If a dataset is linearly separable, then $f(\mathbf{x})$ can be a simple linear function, this, however, is rarely the case for real-world datasets. Hence, one must apply non-linear classification techniques in order to find the non-linear decision boundaries in the dataset.

In this practical, we will cover two types of techniques: (i) *kernel methods* and (ii) *Boosting*. For the former, we will study Support Vector Machines **(SVM)** and Relevance Vector Machines **(RVM)**. These methods encode the decision boundary as a hyper-plane in feature space $\phi(\mathbf{x}) : \mathbb{R}^F$. They follow the same logic as previously seen kernel methods:

1. Lift data to high-dimensional feature space $\phi(\mathbf{x}) : \mathbb{R}^F$, where it might be separable.

2. Apply linear operations in that space.

To learn $f(\mathbf{x})$ with SVM/RVM, we must find the optimal hyper-parameters[1] which will optimize the objective function (and consequently the parameters of the class decision function). Choosing the best hyper-parameters for your dataset is not a trivial task, we will analyze their effect on the classifier and how to choose an admissible range of parameters. For the *Boosting* technique, we will focus on the well-known Adaboost algorithm. This method, constructs a *strong* classifier as a linear combination of *weak* classifiers. The *weak* classifiers (WC) are chosen iteratively among a large set of randomly created WC and combined by updating weights of data-points not well classified by the previous combination of WC. We will compare the performance of SVM vs Adaboost (with decision stumps as the WC) on multiple datasets.

A standard way of finding optimal hyper-parameters is by doing a **grid search** with **cross-validation** over a range of parameters, i.e. systematically evaluating each possible combination of hyper-parameters within a given range. We will do this for different datasets and discuss the difference in performance, model complexity and sensitivity to hyper-parameters.

[1] C: penalty for $C$-SVM, $\nu$: bounds for $\nu$-SVM and kernel type hyper-parameters for all methods.

## Classification Metrics

Many metrics have been proposed to evaluate the performance of a classifier. Most of which, come from a combination of values from the **confusion matrix** (or error matrix)[2]. In this matrix, the rows represents the real classes of the data and the columns the estimated classes. The diagonal represents the well classified examples while the rest indicates confusions. In the case of a binary classifier (i.e. $y \in \{-1, 1\}$), the following quantities have to be computed:

- **True positives (TP)**: number of test samples with a positive estimated label for which the actual label is also positive (good classification)
- **True negatives (TN)**: number of test samples with a negative estimated label for which the actual label is also negative (good classification)
- **False positives (FP)**: number of test samples with a positive estimated label for which the actual label is negative (classification errors)
- **False negatives (FN)**: number of test samples with a negative estimated label for which the actual label is positive (classification errors)

Table 1: Confusion matrix

| | | **Estimated labels** | |
|---|---|---|---|
| | | **Positive** | **Negative** |
| **Real labels** | **Positive** | True positives (TP) | False negatives (FN) |
| | **Negative** | False positives (FP) | True negatives (TN) |

In this practical, we will use the following metrics to evaluate our classification algorithms:

1. Accuracy: The classification accuracy represents the percentage of correctly classified data points, as follows:

$$ACC = \frac{TP + TN}{P + N} \tag{1}$$

2. $F$-measure: The $\mathcal{F}$-measure is a well-known classification metric which represents the harmonic mean between Precision ($P = \frac{TP}{TP+FP}$) and Recall ($R = \frac{TP}{TP+FN}$)

$$\mathcal{F} = \frac{2PR}{P + R} \tag{2}$$

It conveys the balance between *exactness* (i.e. precision) and *completeness* (i.e. recall) of the learned classifier.

[2]See this wikipedia entry for more metrics: `https://en.wikipedia.org/wiki/Confusion_matrix`

### Grid Search with Cross-Validation

To tune the optimal hyper-parameters in methods such as SVM and RVM, we can do an exhaustive search over a grid of the parameter space. This is typically done by learning the decision function for each combination of hyper-parameters and computing a metric of performance. However, testing the learned decision function on the data used to train it is a big mistake, due to over-fitting. This is where **Cross-Validation (CV)** comes in:

*"Cross-validation refers to the practice of confirming an experimental finding by repeating the experiment using an independent assay technique" (Wikipedia).*

In machine learning, CV consists in *holding out* part of the data to *validate* the performance of the model on un-seen samples. The two main CV approaches are $k$-fold and Leave One Out (LOO). In this practical we will use the former. In $k$-fold CV the training set is split into $k$ smaller sets and the following procedure is repeated $k$ times:

1. Train a model using $k-1$ folds as training data.
2. Validate the model by computing a classification metric on the remaining data.

For grid search, the previous steps are repeated $k$ times for each combination of hyper-parameters in the parameter grid. The overall classification performance can then be analyzed through the statistics (e.g. mean, std. deviation, etc) of the metrics computed from the $k$ validation sets.

# 4 Support Vector Machines (SVM)

SVM is one of the most powerful non-linear classification methods to date, as it is able to find a separating hyper-plane for non-seperable data on a high-dimensional feature space using the kernel trick. Yet, in order for it to perform as expected, we need to find the 'best' hyper-parameters given the dataset at hand. In this section we will compare its variants (C-SVM, $\nu$-SVM) and get an underlying intuition of the effects of its hyper-parameters and how to choose them.

## 4.1 C-SVM

Given a data set $D = \{(\mathbf{x}^1, y_1), \ldots, (\mathbf{x}^M, y_M)\}$ of $M$ samples, for $\mathbf{x}^i \in \mathbb{R}^N$, in which the class label $y_i \in \{-1, +1\}$ is either positive or negative, C-SVM seeks to minimize the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{w}\in\mathcal{H},\xi\in\mathbb{R}^M} \left( \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{M}\sum_{i=1}^{M}\xi_i \right) \\ &\text{s.t.} \quad y_i \left( \langle \mathbf{w}, \Phi(\mathbf{x}^i) \rangle + b \right) \geq 1 - \xi_i, \quad \forall i = 1, \ldots M \end{aligned} \tag{3}$$

where $\xi_i \geq 0$ are the slack variables, $b \in \mathbb{R}$ is the bias, $C \in \mathbb{R}$ is a penalty factor used to trade-off between maximizing the margin and minimizing classification errors; $\Phi : \mathbb{R}^N \to \mathbb{R}^F$ is mapping function of the input data into some higher-dimensional Hilbert space $\mathcal{H}$ and $\mathbf{w} \in \mathcal{H}$ is orthogonal to the separating hyper-plane in that space.

As seen in class, $\mathbf{w} \in \mathcal{H}$ can be expressed as a linear combination of the training (feature) vectors:

$$\mathbf{w} = \sum_{i=1}^{M} \alpha_i y_i \Phi(\mathbf{x}^i) \tag{4}$$

where $\alpha_i > 0$ for support vectors. The constraint in (3) can then be represented as: $y_i \left(k(\mathbf{x}, \mathbf{x}_i) + b\right) \geq 1 - \xi_i$, where $k(.)$ is a positive definite kernel, representing the dot product $k(\mathbf{x}, \mathbf{x}_i) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle$ of the data-points in $\mathcal{H}$. The SVM decision function $y(\mathbf{x}) \to \{-1, +1\}$ then takes the following form:

$$y(\mathbf{x}) = \text{sgn}\left( \sum_{i=1}^{M} \alpha_i y_i \, k(\mathbf{x}, \mathbf{x}^i) + b \right), \tag{5}$$

whose parameters are estimated by maximizing the *Lagrangian dual* of (3) wrt. $\alpha$,

$$\begin{aligned} &\underset{\alpha_i \geq 0}{\text{maximize}} \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j k(\mathbf{x}^i, \mathbf{x}^j) \\ &\text{s.t.} \quad 0 \geq \alpha_i \geq C \quad \forall i, \quad \sum_{i=1}^{M} \alpha_i y_i = 0. \end{aligned} \tag{6}$$

where $k(\mathbf{x}, \mathbf{x}^i)$ is the kernel function, which will be the Radial Basis function for this tutorial. The parameters for this decision function are learned by solving the Lagrangian dual of the optimization problem (3) in feature space. As this was seen in class, it will not be covered here. There are extensions such to be able to handle multi-class problems, but in this tutorial we will be focusing on the binary case.

**Kernels:** SVM has the flexibility to handle many types of kernels, we have three options implemented in **ML_toolbox**:

- Homogeneous Polynomial: $k(\mathbf{x}, \mathbf{x}^i) = (\langle \mathbf{x}, \mathbf{x} \rangle)^p$,
  where $p > 0$ and corresponds to the polynomial degree.
- Inhomogeneous Polynomial: $k(\mathbf{x}, \mathbf{x}^i) = (\langle \mathbf{x}, \mathbf{x} + d \rangle)^p$,
  where $p > 0$ and corresponds to the polynomial degree and $d \geq 0$, generally $d = 1$.
- Radial Basis Function (Gaussian): $k(\mathbf{x}, \mathbf{x}^i) = \exp\left\{ -\frac{1}{2\sigma^2} ||\mathbf{x} - \mathbf{x}^i||^2 \right\}$,
  where $\sigma$ is the width or scale of the Gaussian kernel centered at $\mathbf{x}^i$

**Hyper-parameters:** Depending on the kernel, one has several open hyper-parameters to choose. For example, when using the RBF kernel, we would need to find an optimal value for both $C$ and $\sigma$. Intuitively, $C$ is a parameter that trades-off the misclassification of training examples against the *simplicity* of the decision function. The lower the $C$, the smoother the decision boundary (with risk of some misclassifications). Conversely, the higher the $C$, the more support vectors are selected far from the margin yielding a more fitted decision boundary to the data-points. Intuitively, $\sigma$ is the radius of influence of the selected support vectors, if $\sigma$ is very low, it will be able to encapsulate only those points in the feature space which are very close, on the other hand if $\sigma$ is very big, the support vector will influence points further away from them.

`Illustrative example:` To see the effects of the hyper-parameters on $C$-SVM with RBF Kernel, open the accompanying MATLAB script: **TP3_svm.m** and run the first code sub-block `1a` to load the circles dataset shown in Figure 1(a). In code sub-block `2(a)` you can set the desired hyper-parameter values as follows:

```
%% 2a) EXAMPLE SVM OPTIONS FOR C-SVM with RBF Kernel%
clear options
options.svm_type    = 0;    % 0: C-SVM, 1: nu-SVM
options.kernel_type = 0;    % 0: RBF, 1: Polynomial (0 is the default)
options.C           = 1;    % Cost (C \in [1,1000]) in C-SVM
options.sigma       = 1;    % radial basis function: ...
```

The next code sub-block `2(b)` gives an example of setting the hyper-parameters for a polynomial kernel. After choosing these parameters you can run the train-svm code presented in the following sub-block:

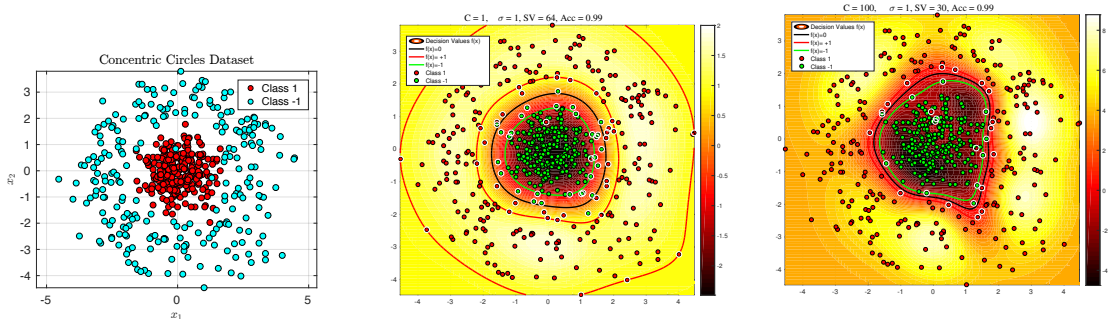
```
%% Train SVM Classifier
[predicted_labels, model] = svm_classifier(X, labels, options, []);

% Plot SVM decision boundary
ml_plot_svm_boundary( X, labels, model, options, 'draw');
```

To visualize a 3D surface mesh for the decision function change `'draw'` to `'surf'`. This will generate the plot in Figure 1(b). Try this yourself and generate the results in Figure 1(c)1(d)1(e)1(f). You will see that for many combinations of $C$ and $\sigma$ a satisfactory classification is achieved (i.e. $ACC = [0.99, 1]$).

**So, how do we find the optimal parameter choices?** Manually seeking the optimal combination of these values is quite a tedious task. For this reason, we use grid search on the open parameters. We must choose admissible ranges for these, in the case of RBF it would be for $C$ and $\sigma$. Additionally, in order to get statistically relevant results one has to cross-validate with different test/train ratio. A standard way of doing this is applying $k$-fold Cross Validation for each of the parameter combinations and keeping some statistics such as mean and std. deviation of the accuracy for the $k$ runs of that specific combination of parameters, a typical number of fold is $k = 10$;

**Grid Search with 10-fold Cross Validation** For the circle dataset we can run C-SVM (RBF kernel) with 10-fold CV over the following range of parameters $C\_\text{range} = [1 : 500]$ and $\sigma\_\text{range} = [0.25 : 3]$, in the same MATLAB script you can specify the steps to partition the range within these limits in code sub-block `2(c)`. This should generate the plots in Figure 2.

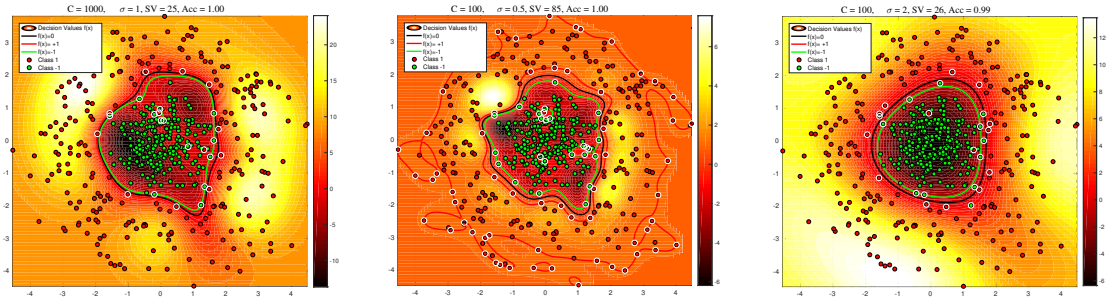
```
%% 2c)  Set options for SVM Grid Search and Execute
clear options
options.svm_type    = 0;          % SVM Type (0:C-SVM, 1:nu-SVM)
options.limits_C    = [1, 500];   % Limits of penalty C
options.limits_w    = [0.25, 3]; % Limits of kernel width \sigma
options.steps       = 10;         % Step of parameter grid
options.K           = 10;         % K-fold CV parameter
options.log_grid    = 1;          % log-sampled grid, set 0 for linear
...
```




(a) Concentric Circle Classification Problem (500 datapoints)

(b) C=1, $\sigma$=1, SV=64

(c) C=100, $\sigma$=1, SV=30,



(d) C=1000, $\sigma$=1, SV=25

(e) C=100, $\sigma$=0.5, SV=85

(f) C=100, $\sigma$=2, SV=26

Figure 1: Decision Boundaries with different hyper-parameter values for the circle dataset. *Support Vector: data-points with white edges.*

The heatmaps in Figure 2 represent the mean $ACC$ and $F$-measure on both training and testing sets. To choose the optimal parameters, we normally ignore the metric on the training test and focus on the performance of the hyper-parameters in the testing set. However, when a classifier is showing very poor performance on the testing set, it is useful to analyze the performance on the training set. If the performance on the training set is poor, perhaps the parameter ranges are off, or you are not using the appropriate kernel or your dataset is simply not separable with this method. Notice the option `options.log_grid = 1;` by setting it to 1, this will generate a logged-grid over the parameter ranges you selected. Using a logged-grid is common in machine learning and might be able to explore better the parameter range. By setting `options.log_grid = 0;` this will generate a linear grid within the range of parameters. Try both to see if there's a difference in the optimal parameter regions.

**Given the grid search results, how do we choose the optimal parameters?** One way of choosing the optimal hyper-parameter combination is to blindly select the iteration which yields the maximum Test $ACC$ or $F$-measure. This is already implemented for you in code sub-block `2(c)`, the optimal values (according to the mean test $ACC$) are stored in `C_opt` and `sigma_opt`. For this specific run we achieved a Max. Test $ACC$ of 97% with $C = 31.58, \sigma = 3$. We can run code sub-block `2(d)` to learn an $C$-SVM with these optimal parameters and plot the decision boundary, as in Figure 3(a).

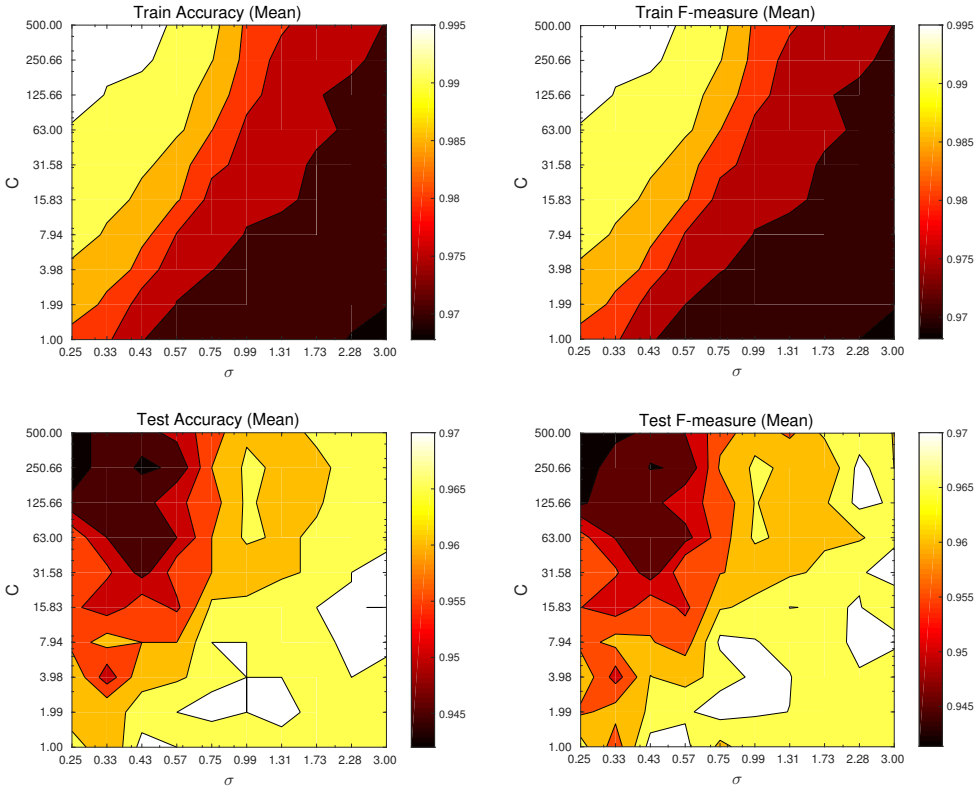
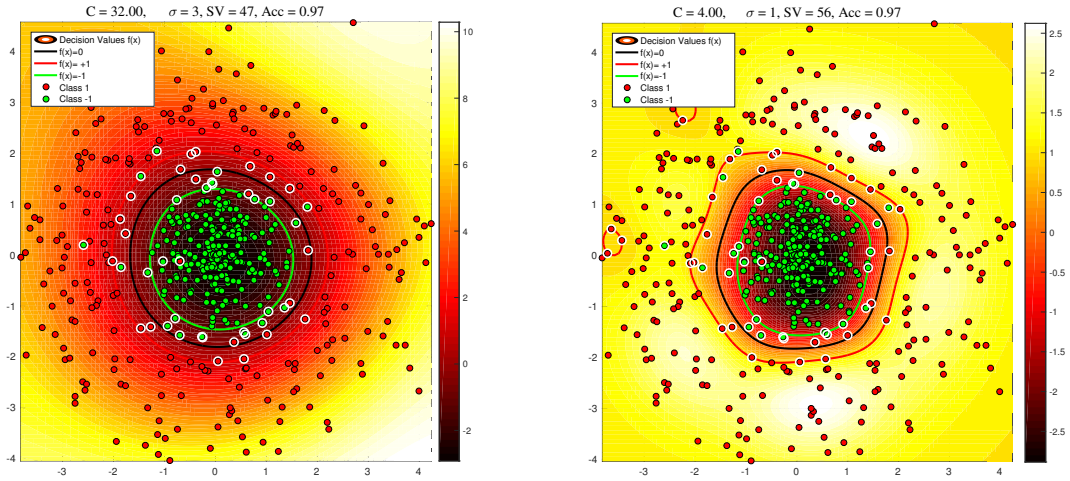



Figure 2: Heatmaps from Grid search on $C$ and $\sigma$ with 10-fold Cross Validation. (top row) Mean Train Accuracy and $\mathcal{F}-measure$ and (right) Mean Test Accuracy and $\mathcal{F}-measure$

As can be seen, the classifier does recover the circular shape of the real boundary from the dataset. However, this is not the only optimal value. If we take a look back at the heatmap for Test Mean $ACC$ (Figure 2). The max $ACC$ was chosen from the white area on the right-bottom region of the heatmap. We can see that there is another big region with really high accuracy at the bottom-center of the heatmap where both $C$ and $\sigma$ values are smaller. Intuitively, if $\sigma$ is smaller, we should have more support vectors, so, let's choose the mid-point of this area which gives us $C = 4, \sigma = 1$, this yields the decision boundary seen in Figure 3(b), it yields the same accuracy, but as we can see, it is a bit more over-fitted to the overlapping points in the boundary. Luckily, in this run we obtained the 'best' model from taking the maximum of the grid search results. Sometimes, this is not the case, and one should analyze the heatmaps.

`NOTE`: It's fine if you can't reproduce exactly the same heatmaps. In fact, due to the random selection of points for the $k$-folds one will tend to get slightly different results.

**Why should we seek for the least number of support vectors?** The number of support vectors needed to recover the decision boundary for our classifier is in fact the measure of model complexity for SVMs.

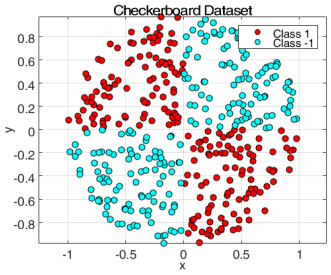



(a) C=32, $\sigma$=3, SV=47,ACC=0.97 (b) C=4, $\sigma$=1, SV=56,ACC=0.97

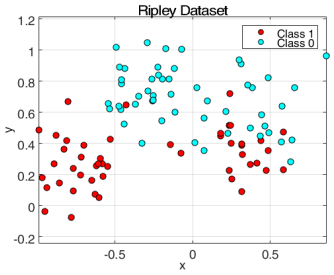
Figure 3: $C$-SVM decision boundary with optimal values from Grid Search with CV.

**TASK 1: Find optimal $C$-SVM parameters for different datasets (linearly separable, checkerboard, Ripley)**

- Follow the same evaluation for C-SVM on the different datasets (Figure 4) available in the MATLAB script from the tutorial
- Try out different kernels and evaluate their performance or feasibility depending on the dataset.
- Find the admissible range of parameters for each dataset and for each method.
- Do grid search with CV on the selected parameter ranges.




(a) Checkerboard Dataset
Samples $M = 400$




(c) Ripley Dataset
Samples $M = 100$

Figure 4: Datasets for non-linear classification.