Advanced Machine Learning
Practical 3: Classification (SVM, RVM & AdaBoost)

Professor: Aude Billard
Assistants: Guillaume de Chambrier,
Nadia Figueroa and Denys Lamotte

Spring Semester 2016

1 Introduction

During this week's practical we will focus on understanding and comparing the performance of the different classification methods seen in class, namely SVM and its variants (C-SVM and ν-SVM, RVM) and an instance of a Boosting method, specifically AdaBoost.

In non-linear classification methods such as SVM/RVM, we seek to find the optimal hyper-parameters (C: penalty, ν: bounds, σ: kernel width for RBF) 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. A standard way of finding these optimal hyper-parameters is by doing a grid search, i.e. systematically evaluating each possible combination of 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.

On the other hand, Boosting chooses weak classifiers (WC) iteratively among a large set of randomly created WC and combines them to create a strong classifier, by increasing the weights of datapoints 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. We will also evaluate which method is more reliable when handling noisy data, data with outliers and data with unbalanced classes.

2 ML_toolbox

ML_toolbox contains a set of matlab methods and examples for learning about machine learning methods. You can download ML_toolbobx from here: [link] and the matlab scripts for this tutorial from here [link]. The matlab scripts will make use of the toolbox.

Before proceeding make sure that all the sub-directories of the ML_toolbox including the files in **TP3** have been added to your matlab search path. This can be done as follows in the matlab command window:

>> addpath(genpath('path_to_ML_toolbox'))
>> addpath(genpath('path_to_TP3'))

To test that all is working properly you can try out some examples of the toolbox; look in the **examples** sub-directory.

3 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, ν-SVM) and get an underlying intuition of the effects of its hyper-parameters and how to choose them.

3.1 C-SVM

Given a data set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)$ of M samples in which the class label $y_i \in \{-1, +1\}$ is either positive or negative, SVM seeks to optimize the following optimization problem for such binary classification task.

$$\begin{aligned} &\min_{\mathbf{w},\xi} \left(\frac{1}{2}||\mathbf{w}||^2 + \frac{C}{M}\sum_{i=1}^{M}\xi_i \right) \\ &\text{s.t.} \quad y^i(\langle \mathbf{w}^T, \mathbf{x}^i \rangle + b) \geq 1 - \xi_i \\ &\xi_i \geq 0 \quad \forall i = 1, \ldots M \end{aligned} \tag{1}$$

where $\mathbf{w}$ is the separating hyper-plane, ξ_i are the slack variables, b is the bias and C is the misclassification penalty factor used to find a trade-off between maximizing the margin and minimizing classification errors. This yields a decision function $y(\mathbf{x}) \to \{-1, +1\}$ of the following form:

$$y(\mathbf{x}) = \text{sign}\left(\langle \mathbf{w}^T, \mathbf{x}^i \rangle + b\right) = \text{sign}\left(\sum_{i=1}^{M} \alpha_i y^i \langle \mathbf{w}^T, \mathbf{x}^i \rangle + b \right) \tag{2}$$

where α_i are the Lagrangian multipliers which define the support vectors ($\alpha > 0$ are support vectors). As seen in class, instead of transforming the data to a high-dimensional feature space, we use the inner product of the feature space, this is the so-called *kernel trick*, the decision function then becomes:

$$y(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{M} \alpha_m y^i \, k(\mathbf{x}, \mathbf{x}^i) + b \right) \tag{3}$$

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 (Eq. 1) in feature space, this was seen in class and 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 σ is the width or scale of the Gaussian kernel centered at $\mathbf{x}^i$

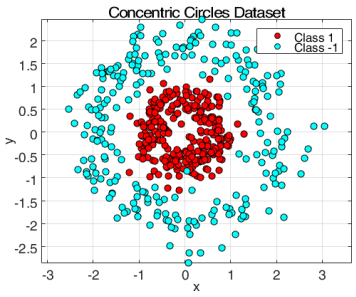
Hyper-parameters: Depending on the kernel, one has several open hyper-parameters to choose. For example, when using the RBF kernel, we would need to find an optimal value for both C and σ. Intuitively, C is a parameter that trades-off the misclassification of training examples against the *simplicity* of the decision function. The lower the C the smoother the decision boundary (with risk of some mis-classifications), the higher the C the more support vector are selected far from the margin yielding a more fitted decision boundary to the data-points. Intuitively, σ is the radius of influence of the selected support vectors, if σ 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 σ is very big, the support vector will influence points further away from them. Thus, the smaller σ the more likely to get more support vectors (for some values of C). See these effects in Figure 1.

You can play around with these parameters for this dataset and generate similar plots by following the instructions in the matlab script **TP3_SVM_RVM.m**.

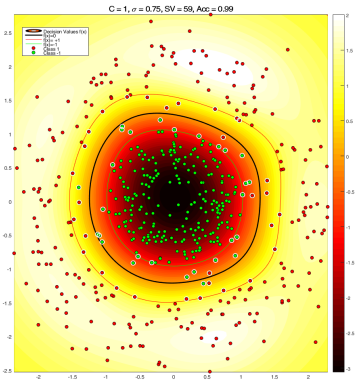
Q: How to find the optimal parameter choices? Manually seeking the optimal combination of these values is quite a tedious task. For this reason, we use grid search on the open parameters. We must choose admissible ranges for these, in the case of RBF it would be for C and σ. Additionally, in order to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

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 $C\_\text{range} = [1 : 100]$ and $\sigma\_\text{range} = [0.25 : 2]$, in the matlab script you can specify the steps to partition the range within these limits.
Q: How did we choose these ranges? Any ideas?

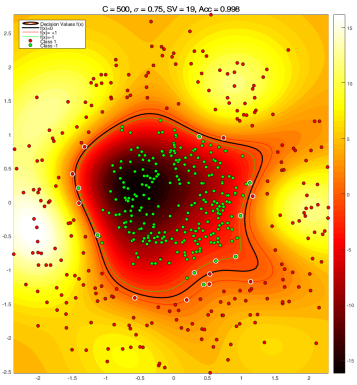
By running this part of the script you should get the plot in Figure 2. This shows a heatmap for the mean Train (left) and Test (right) accuracy (i.e. % of misclassified points).
One way of choosing the optimal hyper-parameter combination is to blindly choose the iteration which yields the maximum Test Accuracy. For this specific run we achieved a



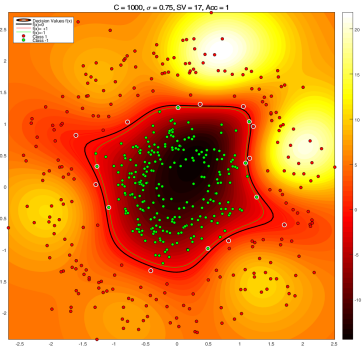
(a) Concentric Circle Classification Problem (500 datapoints)



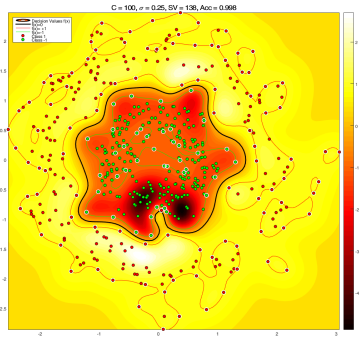
(b)



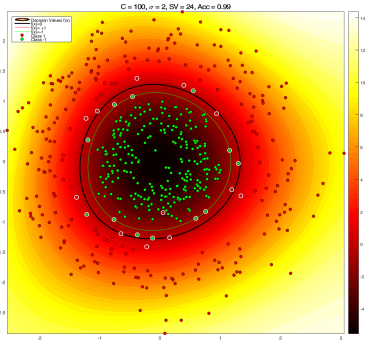
(c)



(d) C=1000, σ=0.75, SV=17



(e) C=100, σ=0.25, SV=138



(f) C=100, σ=2, SV=24

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

Max. Test Accuracy of 98.7% with $C = 1, \sigma = 1.72$, by plotting the decision boundary we get the plot in Figure 3.

As can be seen, the classifier does recover the circular shape of the real boundary from the dataset. However, if we take a look at the support vectors, there seems to be many and quite close to each other.

Q: Could there be a way to recover a similar decision boundary with less support vectors? Let's take a look back at the heatmap for Test Mean accuracy (Figure 2). The max accuracy was chosen from a very tiny white area close to the x-axis of the heatmap. We can see that there is a bigger region with really high accuracy on the bottom-right corner of the heatmap where the C value is not so close to 0. Intuitively, if σ is larger, we should have less support vectors, so, let's choose the mid-point of this area which gives us $C = 10, \sigma = 1.9$, this yields the decision boundary seen in Figure 4, and from our initial guess, we indeed recover the same decision boundary with less than half the amount of support vectors (**SV=38**) than before.

Q: 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.

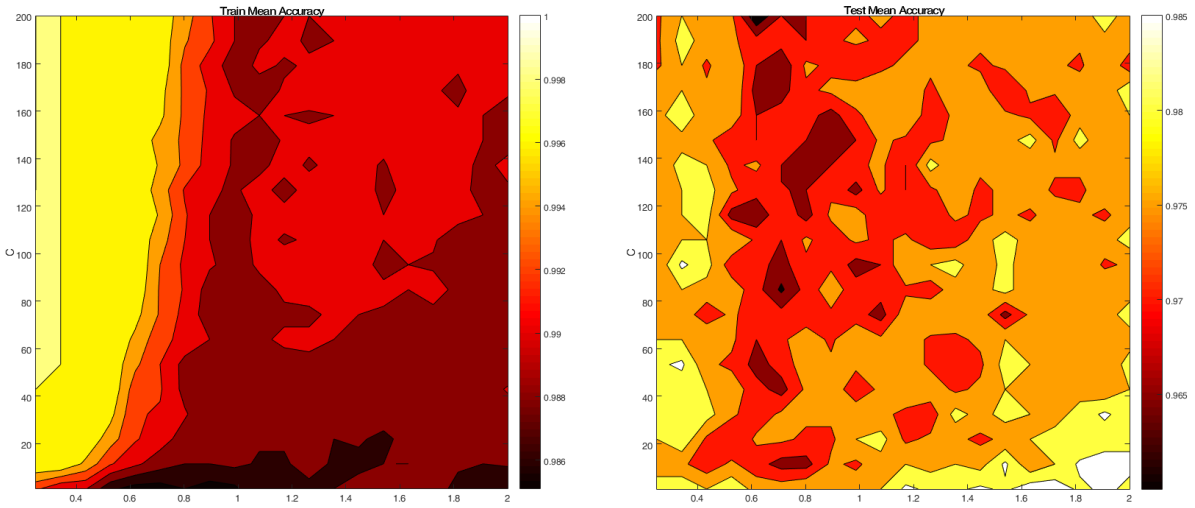


Figure 2: Heatmaps from doing Grid search on C and σ with 10-fold Cross Validation. (left) Mean Train Accuracy and (right) Mean Test Accuracy

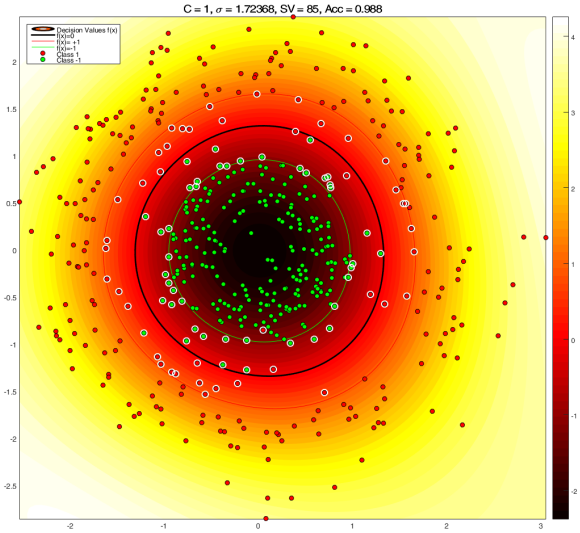


Figure 3: Max Test Accuracy, C=1, σ=1.72, **SV=85**

3.2 ν-SVM

In ν-SVM, rather than setting a fixed value as a penalty (C) for mis-classifications, we use a new variable ρ which controls for the lower bound on $||\mathbf{w}||$ and set the hyper-parameter $\nu \in (0, 1)$ which modulates the effect of ρ on the new objective function:

$$\begin{aligned} &\min_{\mathbf{w},\xi,\rho} \left(\frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{M}\sum_{i=1}^{M} \xi_i \right) \\ &\text{s.t.} \quad y^i(\langle \mathbf{w}^T, \mathbf{x}^i \rangle + b) \geq \rho - \xi_i \\ &\xi_i \geq 0, \rho \geq 0, \quad \forall i = 1, \dots M \end{aligned} \tag{4}$$

ν represents an *upper bound* on the fraction of margin error (i.e. number of data-points misclassified in the margin) and a *lower bound* for the number of support vectors, which is proportional to the ratio $\nu \leq \frac{\#SV}{M}$. Adding these new variables only affects the objective

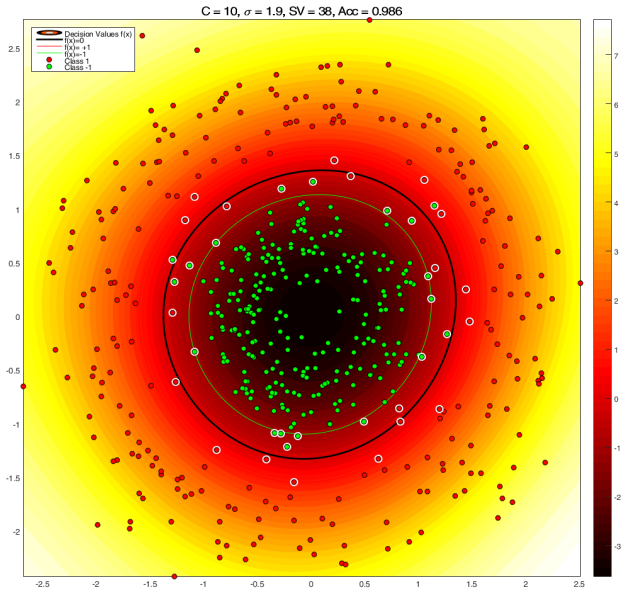


Figure 4: Selection from Heatmap, C=10, σ=1.9, **SV=38**

function (Equation 4), the decision function stays the same as for C-SVM (Eq. 3). In Figure 5, we see the decision boundary and #SVs recovered by setting $\nu = 0.05$ and $\sigma = 1$.

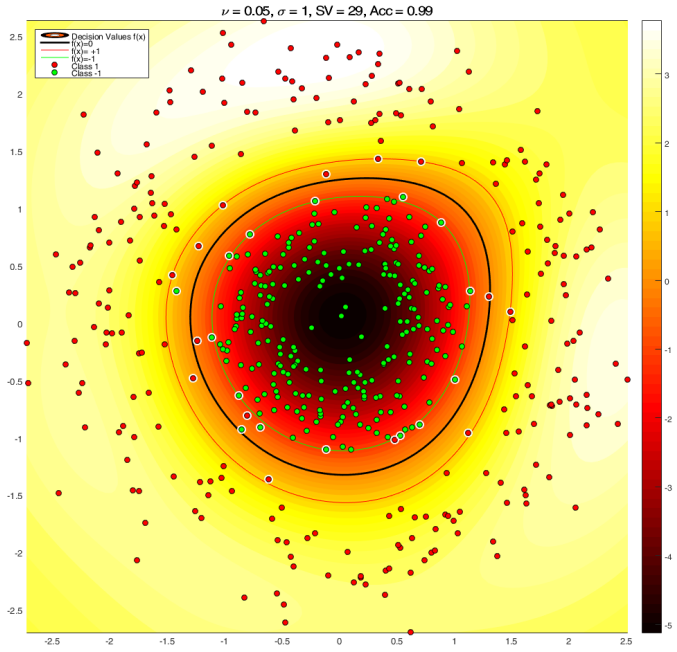


Figure 5: Decision Boundary Plot of ν-SVM on Circle dataset, ν=0.05, σ=1, **SV=29**

3.3 Relevance Vector Machines (RVM)

The Relevance Vector Machine (RVM) applies the Bayesian 'Automatic Relevance Determination' (ARD) methodology to linear kernel models, which have a very similar formulation to the SVM, hence, it is considered as *sparse SVM*. When predictive models are linear in parameters, ARD can be used to infer a flexible, non-linear, predictive model which is both accurate and uses a very small number of relevant basis functions (in our case support vectors) from a large initial set. The predictive linear model for RVM has the following form:

$$y(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, k(\mathbf{x}, \mathbf{x}^i) = \alpha^T \Psi(\mathbf{x}) \tag{5}$$

where $\Psi(\mathbf{x}) = [\Psi_0(\mathbf{x}), \ldots, \Psi_M(\mathbf{x}),]^T$ is a linear combination of basis functions $\Psi(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}^i)$, $\alpha = [1, 0, 1, 0, , \ldots 0, 0, 0, 1]$ is a sparse vector of weights and $y(\mathbf{x})$ is the binary classification decision function $y(\mathbf{x}) \rightarrow \{1, 0\}$. The problem in RVM is now to find a sparse solution for α, where $\alpha_i = 1$ when the datapoint is a relevant 'support' vector and 0 otherwise. Finding such a sparse solution is not trivial. ARD is a method tailor-made to discover the relevance of parameters for a predictive model by imposing a prior on the parameter whose relevance needs to be determined. When attempting to calculate α we take a Bayesian approach and assume that all output labels y^i are i.i.d samples from a true model $y(\mathbf{x^i})$ (Equation 5) subject to some Gaussian noise with zero-mean:

$$\begin{aligned} y^i(\mathbf{x}) &= y(\mathbf{x}^i) + \epsilon \\ &= \alpha^T \Psi(\mathbf{x}^i) + \epsilon \end{aligned} \tag{6}$$

where $\epsilon \propto \mathcal{N}(0, \sigma_\epsilon^2)$. We can then estimate the probability of a label y^i given x with a Gaussian distribution with mean on $y(\mathbf{x^i})$ and variance σ_ϵ^2, this is equivalent to:

$$\begin{aligned} p(y^i|\mathbf{x}) &= \mathcal{N}(y^i|y(\mathbf{x^i}), \sigma_\epsilon^2) \\ &= (2\pi\sigma_\epsilon^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2\sigma_\epsilon^2}(y^i - y(\mathbf{x}^i)^2\right\} \\ &= (2\pi\sigma_\epsilon^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2\sigma_\epsilon^2}(y^i - \alpha^T\Psi(\mathbf{x}^i))^2\right\} \end{aligned} \tag{7}$$

Thanks to the independence assumption we can then estimate the likelihood of the complete dataset with the following form:

$$p(\mathbf{y}|\mathbf{x}, \sigma_\epsilon^2) = (2\pi\sigma_\epsilon^2)^{-\frac{M}{2}} \exp\left\{-\frac{1}{2\sigma_\epsilon^2}||\mathbf{y} - \alpha^T\Psi(\mathbf{x})||^2\right\} \tag{8}$$

We could then approximate α and σ_ϵ^2 through MLE (Maximum Likelihood Estimation), however, this would not yield a sparse vector α, instead it would give us an over-fitting model, setting each point as a relevant basis function. To impose sparsity on the solution, we define an explicit prior probability distribution over α, namely a zero-mean Gaussian:

$$p(\alpha|\mathbf{a}) = \prod_{i=0}^{M} \mathcal{N}(\alpha^i|0, a_i^{-1}) \tag{9}$$

where $\mathbf{a} = [a^0, \ldots, a^M]$ is a vector of hyper-parameters a^i, each corresponding to an independent weight α^i indicating it's strength (or *relevance*). For the binary classification problem, rather than predicting a discriminative membership of a class $y \rightarrow \{1, 0\}$, in RVM we predict the posterior probability $P(y|\mathbf{x})$ of one class $y \in \{1, 0\}$ given input $\mathbf{x}$ by generalizing the linear model from Equation 5 with the logistic sigmoid function $\sigma(y)$ of the form:

$$\sigma(y) = \frac{1}{1 + \exp\{-y\}} \tag{10}$$

The Bernoulli distribution is used for $P(y|\mathbf{x})$ and the likelihood then becomes:

$$P(\mathbf{y}|\alpha) = \prod_{i=1}^{M} \sigma\{y(\mathbf{x}^i)\}^{y^i} [1 - \sigma\{y(\mathbf{x}^i)\}]^{1-y^i} \tag{11}$$

Now, following Bayesian inference, to learn the unknown hyper-parameters α, a we start with the decomposed posterior[1] over unknowns given the data:

$$p(\alpha, \mathbf{a}, |\mathbf{y}) = p(\alpha|\mathbf{y}, \mathbf{a}) p(\mathbf{a}|\mathbf{y}) \tag{12}$$

where unfortunately the posterior distribution over the weights $p(\alpha|\mathbf{y}, \mathbf{a})$ and the hyper-parameter posterior $p(\mathbf{a}|\mathbf{y})$ are not analytically solvable. It can however be approximated by using Laplace's method[2]. Estimating for α then becomes a problem of maximizing the weight posterior distribution $p(\alpha|\mathbf{y}, \mathbf{a})$, and since $p(\alpha|\mathbf{y}, \mathbf{a}) \propto P(\mathbf{y}|\alpha) p(\alpha|\mathbf{a})$ (which are Equations 11 and 9 corresponding to the likelihood and prior of the weights α) this is equivalent to maximizing the following equation over α:

$$\log\{P(\mathbf{y}|\alpha) p(\alpha|\mathbf{a})\} = \sum_{i=1}^{M} [y^i \log \sigma\{y(\mathbf{x}^i)\} + (1 - y^i) \log(1 - \sigma\{y(\mathbf{x}^i)\})] - \frac{1}{2} \alpha^T \mathbf{A} \alpha \tag{13}$$

where $\mathbf{A} = diag(a^0, \ldots, a^M)$. This is optimized in an iterative procedure, a is updated in each step following some derivations from Tipping. Once we have learned our hyper-parameters, Equation 11 is then used to estimate probabilities for each class, in the binary case when $p(y|\mathbf{x}) = 0.5$ this corresponds to the decision boundary, values ≥ 0.5 correspond to the positive class and consequently < 0.5 to the negative class.

As in SVM, the basis functions for the linear model $\Psi_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}^i)$ can be any type of kernel, in this tutorial we will consider only RBF for the RVM. In Figure 6, we see the decision boundary and #RVs recovered by setting $\sigma = 1.22$ found through 10-fold Cross-Validation.

[1]Refer to the Tipping's paper on RVM to understand how this and subsequent terms were derived. Tipping, M. E. (2001). Sparse Bayesian learning and the relevance vector machine. Journal of Machine Learning Research 1, 211–244.

[2]Refer to Tipping paper for Laplace's Approximation.

3.4 Objectives

- Follow the same evaluation as the one done for C-SVM with ν-SVM and RVM on the different datasets (Figure 7) available in the matlab script **TP3_SVM_RVM.m**.
- Try out different kernels and evaluate their performance or feasibility depending on the dataset.
- Do grid search for C-SVM, ν-SVM and RVM
- Find the admissible range of parameters for each dataset and for each method.

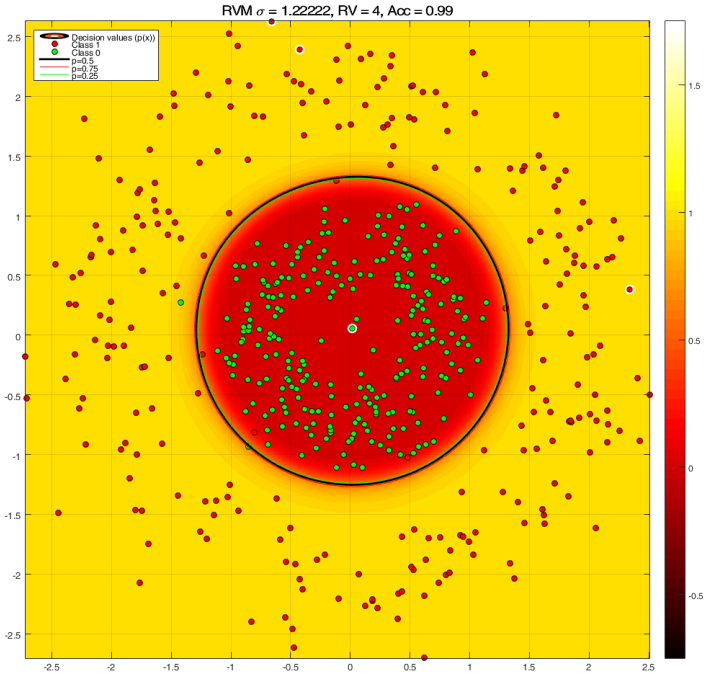
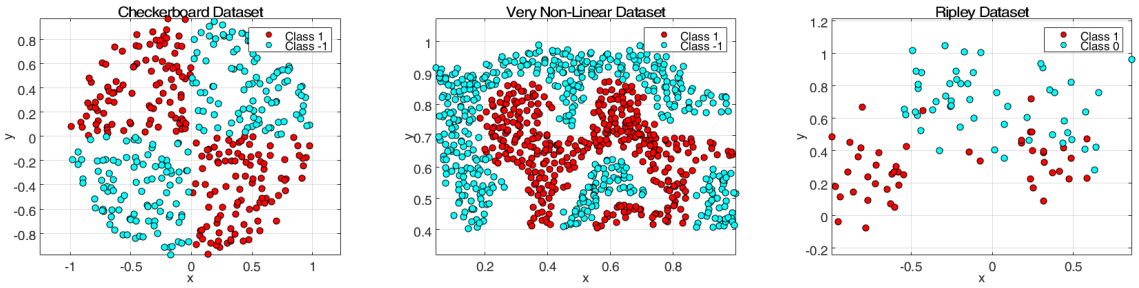


Figure 6: Decision Boundary Plot of RVM on Circle dataset, σ=1.22, **RV=4**



(a) Checkerboard Dataset (b) Very non-linear dataset (c) Ripley Dataset

Figure 7: Datasets to evaluate C-SVM, ν-SVM and RVM .

4 AdaBoost

AdaBoost iteratively builds a strong classifier from a set of weak classifiers. The final strong classifier is a weighted linear combination of simple, also known as weak classifiers $\phi(\mathbf{x}) \rightarrow \{-1, +1\}$, into a strong classifier, $C(\mathbf{x}) \rightarrow \{-1, +1\}$.

The original formulation of AdaBoost considers binary classification problems. There are extensions such to be able to handle multi-class problems, but in this tutorial we will be focusing on the binary case. Given a data set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ of m samples in which the class label $y_i \in \{-1, +1\}$ is either positive or negative we seek to learn a strong classifier $C(\mathbf{x})$, Equation 14,

$$C(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \, \phi(\mathbf{x}_m)\right), \quad \alpha_m \in R \tag{14}$$

In theory you can use any classifier for $\phi(\mathbf{x}_m)$, but usually it should be a very simple and cheap to compute. In this tutorial we will be consider **decision stumps**, Equation 15, as the weak classifier:

$$\phi(\mathbf{x}; \theta) = \begin{cases} +1 & \text{if } \mathbf{x}_{\theta_1} > \theta_2 \\ -1 & \text{otherwise} \end{cases} \tag{15}$$

A decision stump in function which separates the classes along a dimension. It has two parameters $\theta = \{\theta_1, \theta_2\}$. The first parameter indicates in which dimension the decision boundary will be placed and the second parameter decides where along this dimension does the split occur.

To understand the mechanism for Boosting methods we will study how a strong classifier is learned for the circle dataset used in the previous section. The classes are non-linearly separable, in the previous tutorial we used kPCA to find a projection for which the data was linearly separable. This was one approach, now we will proceed to learn a set of linear classifiers which when combined will result in a non-linear classifier.

In Figure 8(a)-8(b) the result of AdaBoost when only one weak classifier is used; we clearly see the decision stump. The figure on the right illustrate the non-signed output of Equation 14 and the figure on the left is the result after taking the sign. Points which are misclassified have a cross on them, but in addition you will see that the width of the data points have changed (*Left column*). The **width of a data point** is proportional to the weight given by the AdaBoost algorithm.

In the second iteration, the number of weak classifiers goes from one to two. The second decision stump is trained on the **new weighted dataset**, for which there classification error from points which high weights are consider more important than data points with low weights. After the 50 iterations (thus 50 weak classifiers), a nearly perfect classification is achieved, see Figure 8(e)-8(f).

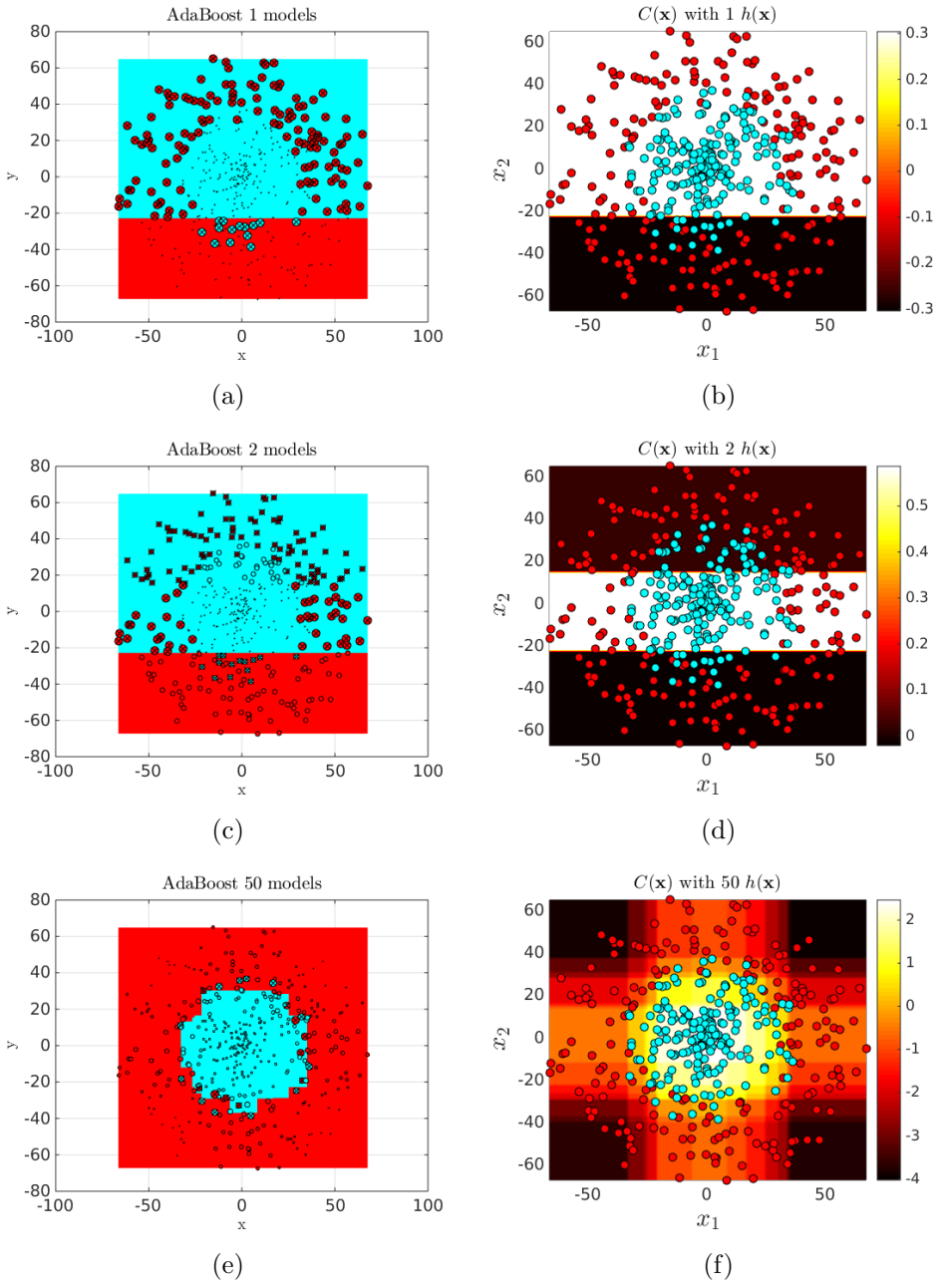


Figure 8: AdaBoost applied to the circle data set. Width of circles on the *Left column* are proportional to the classification error and the crosses depict the points which have been misclassified. In the *Right column*, the value is of the strong classifier function $C(\mathbf{x})$ before the sign operator is applied.

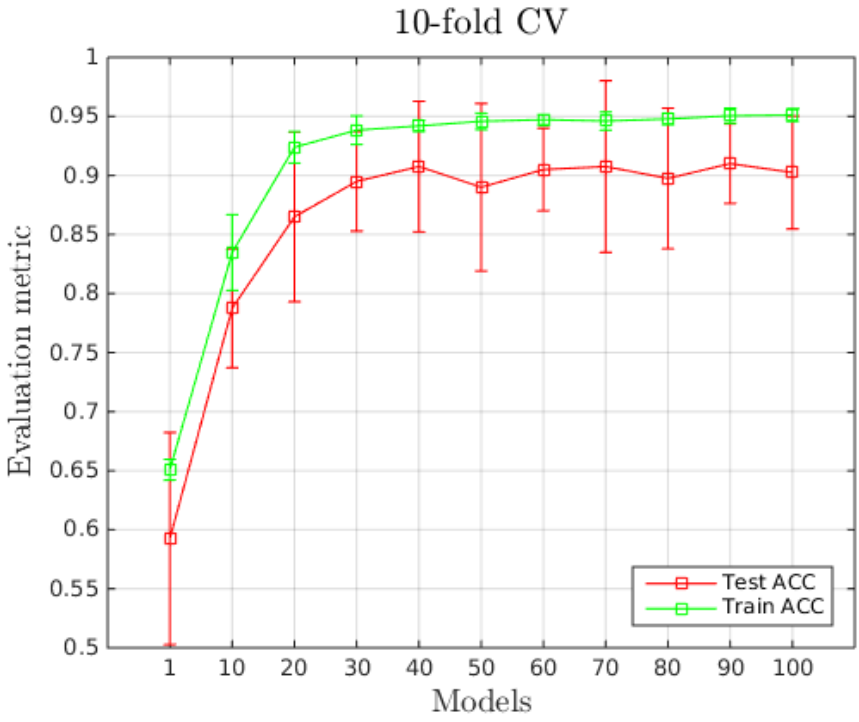


Figure 9: AdaBoost, 10-fold CV over an increasing number of weak classifiers; for the circle data set. The **accuracy** for both the train (green) and test (red) data sets are shown.

10-fold Cross Validation Run AdaBoost with 10-fold CV over the range of parameters specified in the matlab script file **TP3_Boosting.m**. You should get the following plot, Figure 9. You can see that the test error after 40 decision stumps has settled at around a 90%.

4.1 Objectives

- Follow the same evaluation for the different datasets available used in the SVm evaluation (Figure 7) in the matlab script **TP3_Boosting.m**.

This is all good, however the original AdaBoost algorithm can be **sensitive to noise**, which will study next.

4.2 Comparison SVM VS AdaBoost

Sensitivity to noise in comparison to C-SVM AdaBoost is known to be very sensitive to noise. Indeed it has been proved that the objective of AdaBoost is to minimize the exponential loss ($\sum_i e^{-y_i \phi(x_i)}$) of the combined classifier on the training data. As penalties for misclassification grow exponentially with the magnitude of the predictive function output, outlier points could have a very strong influence on the final learned model and so make it very sensitive to noisy data/outliers.

In order to see this you can run these examples in **TP3_SVM_Boosting.m**. In this script, you will see that there is the possibility to add noise in the previously used datasets in different ways :

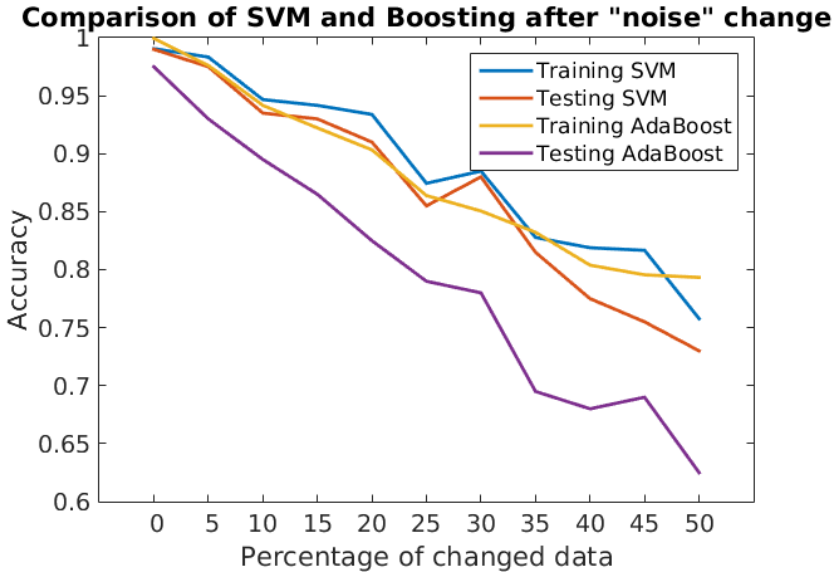


Figure 10: C-SVM and AdaBoost, 10-fold CV over an increasing percentage of noisy data for the circle data set. The **accuracy** for both the train and test data sets are shown.

- Adding some outliers in the dataset (points which are in the region of some class but with a wrong label). This can be seen as class noise.
- Adding noisy data in the dataset (points in each class whose attributes are chosen randomly). This is attribute noise.
- Unbalancing the dataset (one of the class contains more samples than the other(s)).

If we perform a 10-fold cross validation on these modified datasets, using C-SVM and AdaBoost for comparison, we should be able to see that the accuracy of Adaboost decreases more than the one of SVM if we add some noise in the data. See Figure 10.

4.3 Objectives

- Evalute the accuracy for the different datasets (Figure 7) available in the matlab script **TP3_SVM_Boosting.m** (Figure 7).
- Do the 10-fold cross validation for both C-SVM and AdaBoost choosing the optimal hyperparameters found before on the original dataset and compare the performance.
- Try adding different types of noise to the dataset as mentionned before.
- Compare again the accuracy of C-SVM and AdaBoost on these modified datasets. You can can also compare the evolution of it by increasing the percentage of noisy/outlier/unbalanced data.