CAS in Teaching Basics of Statistical Learning

Aleksandr Mylläri, Tatiana Mylläri

Dept. of Computers & Technology,
School of Arts and Sciences
St.George's University
St.George's, Grenada, West Indies
amyllari@sgu.edu

Abstract. *Visual demonstrations provide a convenient way to illustrate the work of algorithms and help students to understand them. Modern Computer Algebra Systems (such as Mathematica or Maple) provide not only opportunity for analytic and numerical calculations, but also convenient tools for solving optimization problems, image construction and animations. It makes CAS a good resource to use in the classroom since the code is usually compact and clear, allowing students to make changes and experiment easily. We model the work of Rosenblatt's perceptron and simple Support Vector Machines (SVMs) using Wolfram Mathematica. Constructed models are used to demonstrate basic concepts and ideas in the introductory courses on SVMs and Statistical Learning.*

1 Introduction

Modern systems of Computer Algebra, such as *Mathematica* or *Maple*, provide good facilities for simulations and visualization. Visual demonstrations provide a convenient way to show the work of algorithms and help in understanding them.

We use Support Vector Machines (SVMs) to make an introduction to Statistical Learning Theory. As an example, we consider the problem of binary classification. SVMs are attractive from the educational point of view since it is easy to introduce them stepwise from simple perceptron (in the linearly separable case) to more advanced classifiers. We start with the perceptron and generalize it to a maximum margin classifier; then we introduce a kernel trick that allows generalization to nonlinearly separable cases; and finally we consider soft margin classifiers (accepting misclassifications during the training stage). Constructed models are used in introductory courses on SVMs and Statistical Learning; we use visualization as an explantory tool to illustrate the work of algorithms using *Mathematica*.

Our introductory course on Machine Learning and Support Vector Machines (SVMs) is based on the classic books by N. Cristianini and J. Shave-Taylor [1] and B. Schölkopf and A.J. Smola [2]. Here, we illustrate the work of the algorithms; detailed descriptions and explanations can be found in [1] and [2].

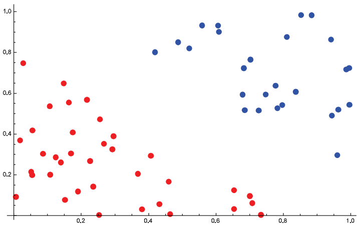

Figure 1: An example of linearly separable data (to use with the perceptron algorithm).

SVMs among other Machine Learning algorithms have been included in *Mathematica* since version 10[3]; there were implementations of SVMs done for earlier versions of *Mathematica* (see, e.g. [4], [5]). Though classifiers implemented in *Mathematica* are very efficient, we choose our own simple classifiers and experiments because we feel that our primary goal - students' understanding of how the algorithms work - is best served by using practical simulations.

The binary classification problem is one of the basic problems of machine learning. We consider supervised learning - the case when the learning machine is given a training set of examples with associated labels (usually $+1$ and -1). We use X to denote the input space and Y to denote the output domain. We have $X \in R^2$; for binary classification $Y = \{-1, 1\}$.

2 Training data

A training set is a collection of training examples (also called training data). It is usually denoted by

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)) \subseteq (X \times Y)^l,$$

where l is the number of examples. We refer to the $\mathbf{x}_i$ as examples and the y_i as their labels.

Mathematica as well as other Computer Algebra systems easily allows one to generate data with desired properties, such as a linearly separable case with clear margin (Fig. 1), non-linearly separable data (Fig. 2), noisy data with some cases mislabelled, etc. Same data generation algorithms can be used to generate test data sets with specified distributions to check the classifier obtained.

3 Classification

Binary classification is frequently performed by using a real-valued function $f : X \subseteq R^n \to R$ in the following way: the input $\mathbf{x} = (x_1, \ldots, x_n)'$ is assigned to the positive class if $f(\mathbf{x}) \geq 0$, and to the negative class otherwise. The simplest case is when

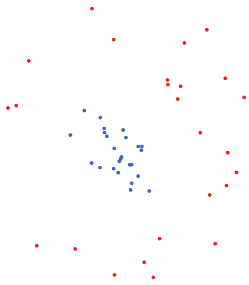

Figure 2: An example of non-linearly separable data.

$f(\mathbf{x})$ is a linear function of $\mathbf{x} \in X$, so that it can be written as

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i=1}^{n} w_i x_i + b$$

where $(\mathbf{w}, b) \in R^n \times R$ are the parameters that control the function and the decision rule is given by $\operatorname{sgn}(f(x))$, where we use the convention that $\operatorname{sgn}(0) = 1$. The learning methodology implies that these parameters must be learned from the data.

A geometric interpretation is that the input space X is split into two parts by the hyperplane defined by the equation $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ (see Figure 3). In the simple case we use for visualization, we deal with points on a plane, and a hyperplane is a straight line which divides the plane into two halves corresponding to the inputs of the two distinct classes. In Figure 3, the positive region is above and the negative region below the line.

As was said in the Introduction, we use a basic sequence of actions: we start with linearly separable data and a perceptron; continue with a maximum margin classifier; generalize it to the non-linear (separable) case; and then generalize to the non-separable case (noisy data). We have no intention to give detailed descriptions of algorithms and methods here. The formulae and descriptions below are only provided to demonstrate that SVMs can be easily programmed in *Mathematica*.

3.1 Perceptron

The first iterative algorithm for learning linear classifications is the perceptron proposed by Frank Rosenblatt in 1956. It is an 'on-line' and 'mistake-driven' procedure, which starts with an initial weight vector $\mathbf{w}_0$ (usually $\mathbf{w}_0 = \mathbf{0}$ - the all zero vector) and adapts it each time a training point is misclassified by the current weights. The perceptron algorithm updates the weight vector $\mathbf{w}$ and bias b directly. This procedure is guaranteed to converge provided there exists a hyperplane that correctly classifies the training data (i.e. data are linearly separable). If no such hyperplane exists the data are said to be nonseparable. Example of the resulting classifier for the data from Figure 1 is shown on Figure 3.

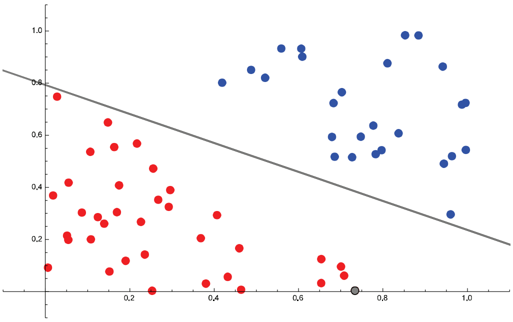

Figure 3: Hyperplane (line in this case) found by the perceptron algorithm.

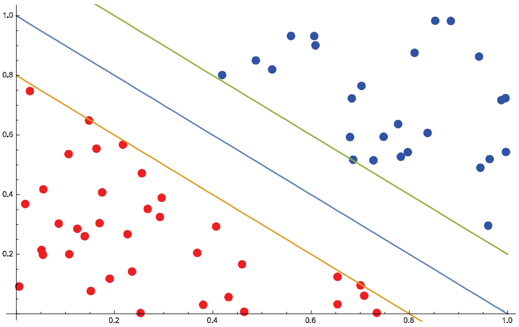

Figure 4: Margin of the data from Fig. 1.

3.2 Maximum margin classifier

The number of iterations performed by the perceptron algorithm depends on a quantity called the margin. This quantity plays a central role in SVM theory so we give a more formal definition [1]:

Definition *The (functional) margin of an example* $(\mathbf{x}_i, y_i)$ *with respect to a hyperplane* $(\mathbf{w}, b)$ *is the quantity*

$$\gamma_i = y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b).$$

Note that $\gamma_i > 0$ implies correct classification of $(\mathbf{x}_i, y_i)$. If we replace functional margin by geometric margin we obtain the equivalent quantity for the normalized linear function $\left(\frac{1}{\|\mathbf{w}\|}\mathbf{w}, \frac{1}{\|\mathbf{w}\|}b\right)$, which measures the Euclidean distances of the points from the decision boundary in the input space. The margin of a training set S is the maximum geometric margin over all hyperplanes (straight lines in our case). A hyperplane realizing this maximum is called a maximal margin hyperplane. The size of its margin will be positive for a linearly separable training set. Figure 4 shows the margin of a training set, while Fig. 5 compares perceptron with maximum margin.

Having our (linear) classifier in the form $f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$ we are free to scale the parameters $\mathbf{w}$ and b. By requiring that this scaling be such that the point(s)

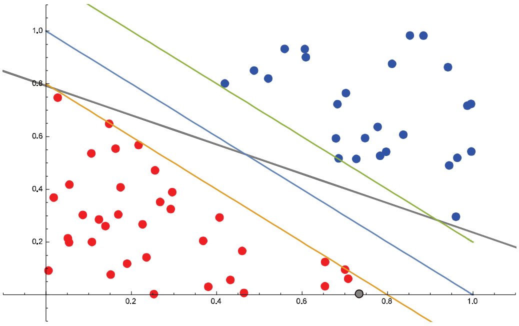

Figure 5: Perceptron (Fig. 3) vs. maximum margin.

closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain its canonical form $(\mathbf{w}, b)$ by solving the following optimization problem [2]:

$$\text{minimize }_{w \in R^n, b \in R} \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2,$$
$$\text{subject to } y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) \geq 1 \text{ for all } i = 1, \ldots, m.$$

In practice, it is convenient to solve the dual optimization problem that is derived by introducing a Lagrangian:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b\right) - 1\right) \tag{1}$$

with Lagrange multipliers $\alpha_i \geq 0$. The Lagrangian L must be maximized with respect to α_i, and minimized with respect to $\mathbf{w}$ and b. We have

$$\sum_{i=1}^{m} \alpha_i y_i = 0,$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i.$$

The patterns x_i for which $\alpha_i > 0.$ are called Support Vectors.

The dual optimization problem takes the form

$$\text{maximize }_{\alpha \in R^m} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle,$$
$$\text{subject to } \alpha_i \geq 0, i = 1, \ldots, m,$$
$$\text{and } \sum_{i=1}^{m} \alpha_i y_i = 0.$$

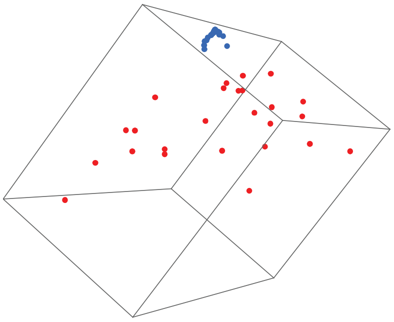

Figure 6: Data from Fig. 2 after nonlinear transformation.

This optimization problem can be easily solved in *Mathematica* by using **Maximize**. The classifier takes the form

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right)$$

3.3 Non-linear case

Consider the case shown in Figure 2. Data are not linearly separable, but if we map our data nonlinearly into a higher-dimensional space, data could became linearly separable. In the case of Figure 2, mapping $(x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ will do the trick (see Figure 6).

Let us note that in the formulation of the dual optimization problem and final classifier, training data is present only in the form of dot products. So, if we make a nonlinear mapping Φ, in the target nonlinear space we will be interested only in the dot product $\langle \Phi(x), \Phi(x_i) \rangle$. Let us introduce a (positive definite, see [2] for details) kernel function $k(x, x_i)$:

$$\langle \Phi(x), \Phi(x_i) \rangle = k(x, x_i)$$

The optimization problem to solve will take the form

$$\begin{aligned} \text{maximize }_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x, x_i), \\ \text{subject to } \alpha_i \geq 0, i = 1, \ldots, m, \\ \text{and } \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned}$$

and the final classifier

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b\right)$$

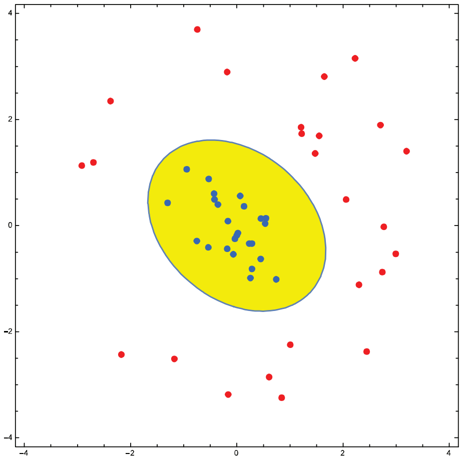

Figure 7: Classifier with polynomial kernel.

Again, this optimization problem can be easily solved in *Mathematica* by using **Maximize**.

Examples of such classifiers are given in Figure 7 (for data shown in Figure 2, using a polynomial kernel function) and Figure 8 (using an exponential kernel function).

In practice, a separating hyperplane could be non-existent - e.g. data could be noisy, with outliers, or some training examples could be mislabelled. So, it would be good to have an algorithm that can deal with such cases. One approach is to introduce so-called slack variables, $\xi_i \geq 0$, where $i = 1, \ldots, m$, and use relaxed separation constraints

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i, i = 1, \ldots, m.$$

Obviously, by making ξ_i large enough, the constraints on $(\mathbf{x}_i, y_i)$ can always be met. To avoid this trivial solution with large values of ξ_i, a term to penalize large values of ξ_i can be added to the optimization problem. In the simplest case of the so-called C-SV classifier (for some $C > 0$), the optimization problem takes the form:

$$\text{minimize }_{w \in H, \xi \in R*m} \tau(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \xi_i,$$

subject to

$$\xi_i \geq 0, i = 1, \ldots, m,$$

and

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i, i = 1, \ldots, m.$$

The solution, as in the linearly separable case, will have the form

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

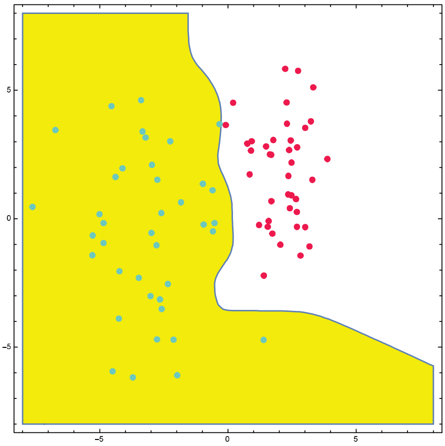

Figure 8: Classifier with exponential kernel.

Coefficients α_i can be found by solving the corresponding dual (quadratic) optimization problem:

$$\begin{aligned}
&\text{maximize }_{\alpha \in R^m} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j),\\
&\text{subject to } 0 \le \alpha_i \le \frac{C}{m} \text{ for all } i = 1, \ldots, m,\\
&\text{and } \sum_{i=1}^{m} \alpha_i y_i = 0.
\end{aligned}$$

Similar to the separable case, this optimization problem can be easily solved in *Mathematica* by using **Maximize**. The threshold b can be obtained by averaging

$$b = y_j - \sum_{i=1}^{m} y_i \alpha_i k(x_j, x_i)$$

over all points with $\alpha_i > 0$; in other words, all support vectors. An example of the work of a soft-margin classifier is given in Figure 9.

4 Conclusions

Mathematica provides good facilities for modeling and visualization of SVMs. Built-in methods to solve optimization problems, compactness of the *Mathematica* code, and good graphics make experimentation easy. Thus, students can generate training and test data easily. As an option, advanced students can experiment with making their own optimization problem solvers. If the data set is of reasonable size, multiple

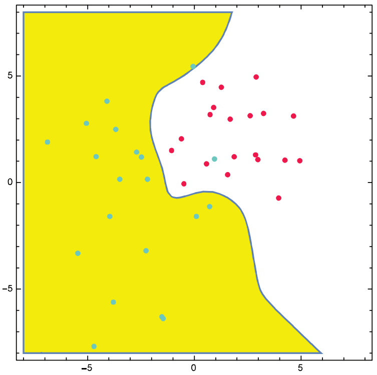

Figure 9: Example of training soft-margin classifier.

simulations can be done in moderate time with different parameters, etc. Students can experiment with different kernel functions, observe overfitting/underfitting, etc. A rough estimate of the quality of learning could be also easily done by looking at the proportion of non-zero coefficients α_i – if all (or too large a fraction) of training examples are found to be support vectors, overlearning takes place. All of the abovementioned cases of easily constructed learning examples using *Mathematica* show it to be a useful resourse to use in the classroom for introductory courses on SVMs and Statistical Learning Theory.

References

[1] *Shawe-Taylor J., Cristianini N.* Support Vector Machines and other kernel-based learning method. Cambridge University Press (2000)

[2] *Schölkopf B., Smola A.J.* Learning with Kernels Support Vector Machines, Regularization, Optimization and Beyond. Massachusetts Institute of Technology (2001)

[3] *h*ttps://www.wolfram.com/mathematica/new-in-10/highly-automated-machine-learning/

[4] *Paláncz B, Völgyesi L.* Support Vector Classifier via Mathematica. Periodica Polytechnica Civ. Eng, Vol. 48, Nr. 1-2. pp. 15-37. (2004)

[5] *Nilsson R., Björkegren J., Tegnér J.* A Flexible Implementation for Support Vector Machines. The Mathematica Journal, Vol. 10, pp.114-127. (2006)