Genetic algorithm for Quantum Support Vector Machines

Lorenzo Tasca

October 2024

Contents

1 Introduction

2 Theoretical Background

2.1 Classical machine learning

baa

2.1.1 Support vector machine

The Support Vector Machine (SVM) is a binary classification alghoritm, whose goal is to build the maximum margin separator between the two classes, that is the separator that maximizes the distance of the closest point from each class. The standard SVM alghoritm is a linear alghoritm, so in particular it will try and build the separating margin as a hyperplane in a d-dimensional space (so a $(d-1)$-dimensional plane), where d is the number of features. The points that touch the margin, or that are on the wrong side of it, are called support vectors. The distance between the decision boundary and the support vectors is called margin. The alghoritm will find the biggest possible margin.

Let's start assuming that the classes are linearly separable. We are provided with a dataset with N d-dimensional istances $\{\mathbf{x}_i\}_{i=0,\cdots,N-1}$. The two classes will be labelled with

$$y \in \{-1, 1\}.$$

The margin will be the the set of points

$$\{\mathbf{x} \in \mathbb{R}^d : w_0 + \mathbf{w}^T \cdot \mathbf{x} = 0\}, \tag{1}$$

for appropriate parameters $w_0 \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^d$, which define the hyperplane and must be found by the alghoritm. Now we have to find

$$\max_{w_0, \mathbf{w}}(m),$$

with the constraint

$$\frac{1}{||\mathbf{w}||} y_i(w_0 + \mathbf{w}^T \cdot \mathbf{x_i}) \geq m, \, \forall i = 0, \cdots, N-1, \tag{2}$$

that can be rewritten as

$$y_i(w_0 + \mathbf{w}^T \cdot \mathbf{x_i}) \geq m||\mathbf{w}||,\ \forall i = 0, \cdots, N-1. \tag{3}$$

The constraint prevents data points from falling into the margin. Rescaling $\mathbf{w}$ up to a multiplicative factor does not change the hyperplane it defines, so for convenience we can choose its norm such that

$$||\mathbf{w}|| = \frac{1}{m}. \tag{4}$$

Therefore the problem becomes minimizing

$$\frac{1}{2}||\mathbf{w}||,$$

with the constraint

$$y_i(w_0 + \mathbf{w}^T \cdot \mathbf{x_i}) \geq 1,\ \forall i = 0, \cdots, N-1. \tag{5}$$

In the theory of convex optimization one can solve for the Lagrangian dual of this problem. We can introduce the dual variables α_i such that

$$\mathbf{w} = \sum_{i=0}^{N-1} \alpha_i y_i \mathbf{x}_i. \tag{6}$$

One would obtain that the dual problem consists in maximizing, with respect to to the weight vector $\alpha \in \mathbb{R}^N$, the expression

$$f(\alpha_0, \cdots, \alpha_{N-1}) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i, \mathbf{x}_j), \tag{7}$$

with the constraint

$$\alpha_i \geq 0,\ \forall i = 0, \cdots, N-1, \tag{8}$$

$$\sum_{i=0}^{N-1} \alpha_i y_i = 0. \tag{9}$$

In eq. (7) we indicated as $(\mathbf{x}_i, \mathbf{x}_j)$ the standard dot product of $\mathbb{R}^d$, explicitly

$$(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \cdot \mathbf{x}_j = \sum_{k=0}^{d-1} (\mathbf{x}_i)_k (\mathbf{x}_j)_k.$$

Eq. (7) defines a quadratic programming optimization problem, therefore the global maximum of f can be efficiently found in the context of convex analysis. The parameter w_0 can be found by imposing that, for a support vector

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) = 1,$$

that is

$$w_0 = \mathbf{w}^T\mathbf{x}_i - y_i. \tag{10}$$

Once the optimal α_i have been found, given a new istance $\tilde{\mathbf{x}}$, according to eq. (6) and eq. (1), we can predict its class calculating

$$\text{sign}\left[\sum_{i=0}^{N-1} \alpha_i y_i(\tilde{\mathbf{x}}, \mathbf{x}_i) + w_0\right]. \tag{11}$$

What we just described is the so called hard margin SVM, because we did not allow points to fall inside the margin. One could relax this assumption, modifing the constraint in eq. (5) into

$$y_i(w_0 + \mathbf{w}^T \cdot \mathbf{x_i}) \geq 1 - \xi_i,\ \forall i = 0, \cdots, N-1, \tag{12}$$

where we introduced the slack variables ξ_i. We limit the softness of the margin by setting a positive constant C such that

$$\begin{aligned} \xi &\geq 0, \\ \sum_{i=0}^{N-1} \xi_i &\leq C. \end{aligned} \tag{13}$$

This is called soft margin SVM.

scikit-learn provides a straightforward implementation of the SVM algorithm, which we can use to observe the algorithm in action through an example. We use a mock dataset with 2 features, so we can easily print the data, the decision boundary and the margin.

```
from sklearn.svm import SVC
from sklearn.datasets import make_blobs

```

```
X,y = make_blobs(n_samples=100) #create mock dataset
svm = SVC(kernel='linear', C=1) #create svm
svm.fit(X, y) #fit the svm
```

The result of the fit is shown in Figure (1). We can observe how the alghoritm built the largest possible margin.

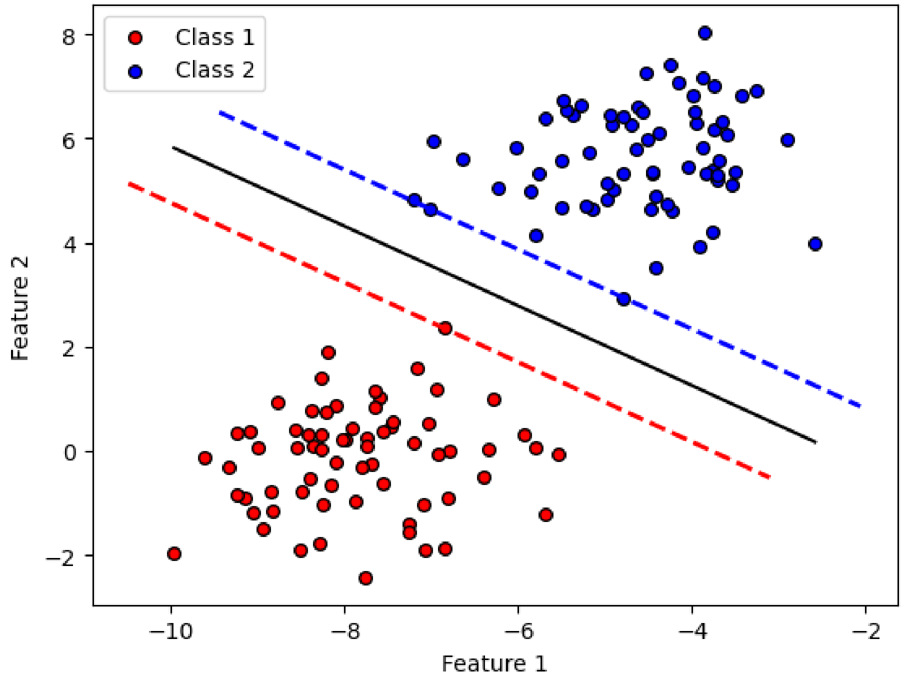

Figure 1: SVM decision boundary and margin border, fitted on a 2 feature mock dataset of 200 istances.

We now need to address the issue of dealing with a highly non-linearly separable dataset. Let's consider as an example another mock dataset, shown in Figure (2). It is clear that in this case we cannot use the SVM alghoritm in its basic form, not even with a soft margin. We must introduce the idea of kernelization. Let's introduce a function, called feature map, which projects the data in a higher dimensional space. That means a function

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D, \tag{14}$$

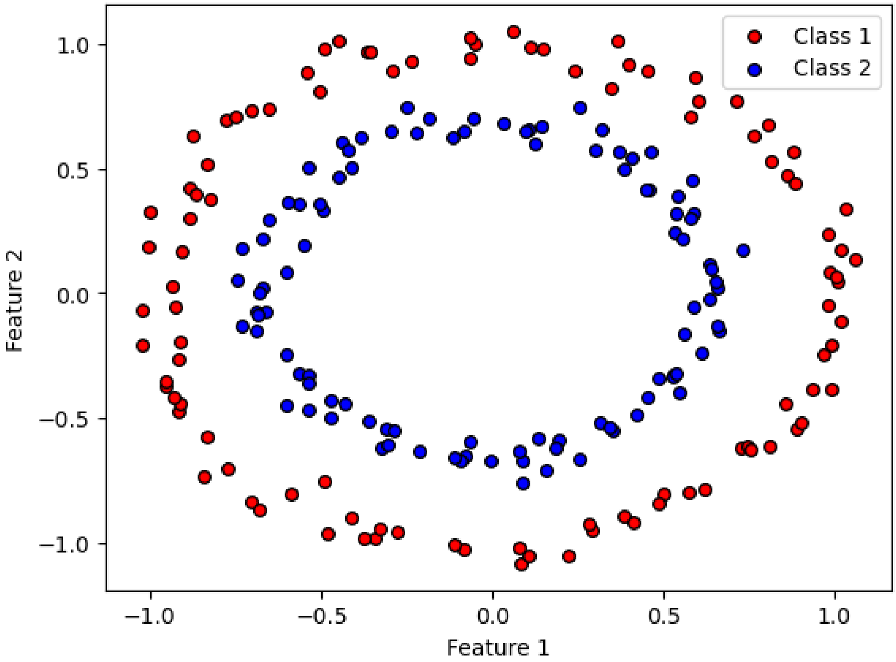

Figure 2: Higly non-linear 2 feature mock dataset with 200 istances.

with $D > d$. The codomain of the feature map is called feature space. If we choose a suitable feature map we can hope to obtain a linearly separable dataset in the feature space. The choice of the feature map is completely arbitrary, as long as it is a bijective function. Therefore, in principle, each time we are given a dataset we must choose an appropriate feature map for this strategy to work. For our example let's consider the feature map

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3,$$

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto \begin{pmatrix} x_0^2 \\ x_1^2 \\ \sqrt{2}x_0x_1 \end{pmatrix}. \tag{15}$$

The data of Figure (2) after the application of the feature map ϕ are represented in Figure (3). The dataset is now linearly separable in the feature space, so the strategy worked. We can now apply the SVM alghoritm in this space Consider the two central equations of the algorithm: equation (7), which provides the

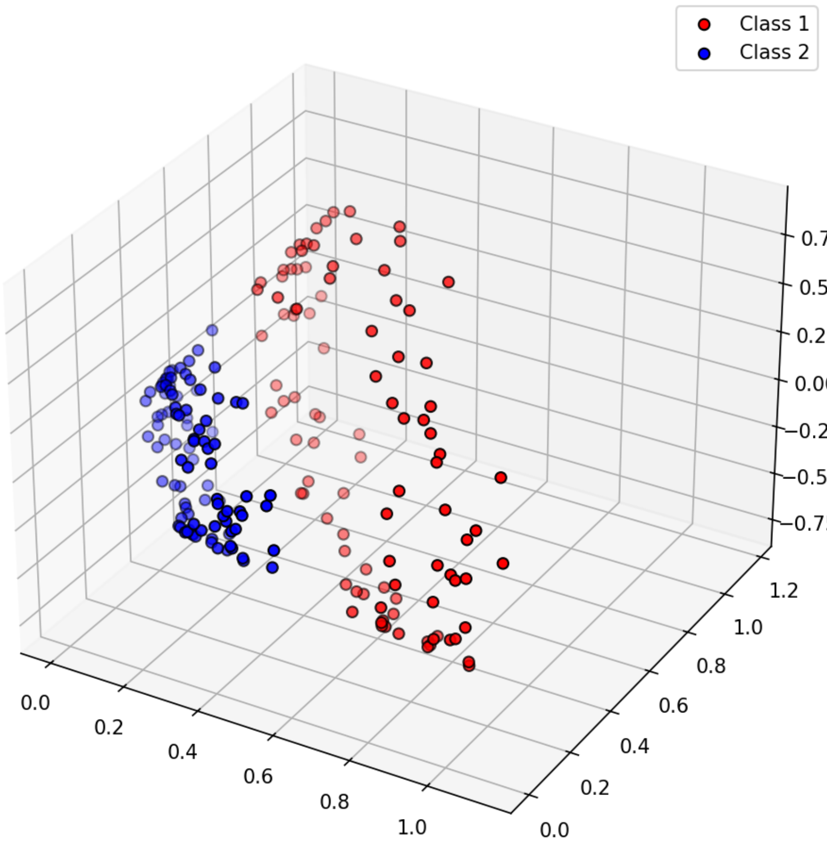

Figure 3: Higly non-linear mock dataset in the feature space after the application of the feature map ϕ. We observe now that the dataset is linearly separable.

expression to maximize in order to find the margin, and equation (11), which gives the rule for predicting the class of a new instance. This two equations are now modified into

$$f(\alpha_0, \cdots, \alpha_{N-1}) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \alpha_i \alpha_j y_i y_j (\phi(\mathbf{x}_i,) \phi(\mathbf{x}_j)), \tag{16}$$

$$\text{sign} \left[\sum_{i=0}^{N-1} \alpha_i y_i (\phi(\tilde{\mathbf{x}}), \phi(\mathbf{x}_i)) + w_0 \right]. \tag{17}$$

A crucial observation is that in these two expressions only the scalar product of the feature map values appears. Therefore we can conclude that the specific form of the feature map is not important, but rather the scalar product it produces. We can define the kernel K as

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R},$$

$$\mathbf{x}, \mathbf{y} \mapsto (\phi(\mathbf{x}), \phi(\mathbf{y})), \tag{18}$$

where $(\phi(\mathbf{x}), \phi(\mathbf{y})) = \phi(\mathbf{x})^T \cdot \phi(\mathbf{y})$ is the standard scalar product of $\mathbb{R}^D$. Eq. (16) and eq. (17) now become

$$f(\alpha_0, \cdots, \alpha_{N-1}) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{19}$$

$$\text{sign} \left[\sum_{i=0}^{N-1} \alpha_i y_i K(\tilde{\mathbf{x}}, \mathbf{x}_i) + w_0 \right]. \tag{20}$$

We see explicitly that the only quantity that matters is the kernel K. In our specific example the value of the kernel is

$$K(\mathbf{x}, \mathbf{y}) = \begin{pmatrix} x_0^2 & x_1^2 & \sqrt{2} x_0 x_1 \end{pmatrix} \cdot \begin{pmatrix} y_0^2 \\ y_1^2 \\ \sqrt{2} y_0 y_1 \end{pmatrix} = (\mathbf{x}^T \cdot \mathbf{y})^2. \tag{21}$$

Therefore once we are given a dataset it is sufficient for us to choose an appropriate kernel, and forget about the feature map. Once the kernel has been chosen the SVM can be trained using eq. (19), and we can use it to predict a new class using eq. (20). There are some properties that the kernel must satisfy:

- The kernel must be symmetric, that is

$$\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^d,\ K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x}).$$

- The kernel must be positive definite, that is

$$\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^d,\ K(\mathbf{x}, \mathbf{y}) \geq 0.$$

Common choices of kernels are

- Linear kernel:

  $$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \cdot \mathbf{y}.$$

  This goes back to the standard SVM we used in Figure (1). It is suitable only for linearly separable (or close to, using soft margin) datasets.

- Polynomial kernel:

  $$K(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^T \cdot \mathbf{y} + c)^{\delta}.$$

  For $c = 0$, $\gamma = 1$ and $\delta = 2$ we obtain the kernel of eq. (21).

- Gaussian kernel:

  $$K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma ||\mathbf{x} - \mathbf{y}||).$$

  This is also known as Radial Basis Function (RBF) kernel.

- Sigmoid kernel:

  $$K(\mathbf{x}, \mathbf{y}) = \tanh(\mathbf{x}^T \cdot \mathbf{y} + c).$$

scikit learn offers an easy way to easily implement all these common kernels. For example the kernel in eq. (21) can be implemented as

```
svm = SVC(kernel='poly', degree=2, gamma=1, coef0=0)
```

One can also create a custom kernel, passing as an argument a callable function to be used to calculate the kernel. Fitting this SVC function to the non-linear dataset of Figure (2) yields the result shown in Figure (4).

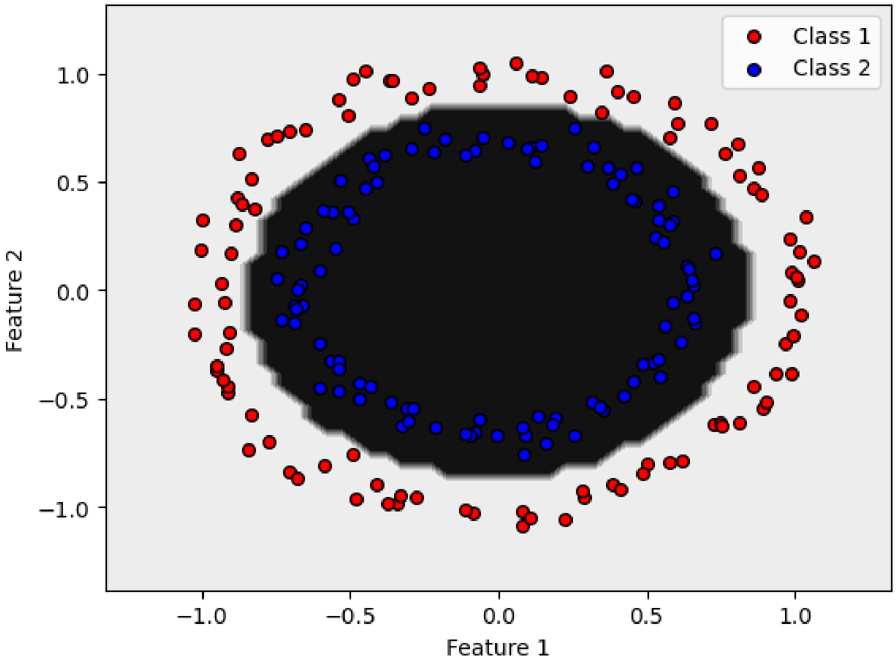

Figure 4: Higly non-linear mock dataset decision boundary, fitted with a SVM using kernel of eq. (21). The white and black areas are the two predicted classes. We observe how, using the kernel, we obtained a non-linear decision boundary.

2.2 Quantum machine learning

Introduction bla bla

2.2.1 Quantum Support Vector Machine

The SVM alghoritm faces some important limitations when the feature space becomes large, as estimating kernel functions becomes computationally intensive. Quantum computing could enhance the algorithm's performace by providing access to exponentially large Hilbert feature spaces. The idea is to construct a feature map which maps classical data into a quantum state which lives in an exponentially large Hilbert feature space. Therefore in this context a feature map is a function

$$\phi : \mathbb{R}^d \to \mathcal{H}, \tag{22}$$

$$\mathbf{x} \mapsto \phi(\mathbf{x}) \equiv |\phi(\mathbf{x})\rangle.$$

In the framework of quantum computing $\mathcal{H}$ is a n-qubit Hilbert space, that is a space of the form

$$\mathcal{H} = \bigotimes_{i=0}^{n} \mathcal{H}_{qubit}, \tag{23}$$

where $\mathcal{H}_{qubit}$ is the Hilbert space of a single qubit. The dimension of $\mathcal{H}$ is 2^n. The feature map will implemented by the means of a parametrised quantum circuit. That means that it exists a unitary operator that depends on d classical parameters $U(\mathbf{x})$ such that

$$|\phi(\mathbf{x})\rangle = U(\mathbf{x})|0\rangle^{\otimes n}. \tag{24}$$

This circuit is called quantum encoding circuit, because it encodes classical data into a quantum state. The classical data is passed to the circuit as a parameter. Once we have the feature map the kernel is constructed as

$$K(\mathbf{x}, \mathbf{y}) = \left|\langle\phi(\mathbf{x})|\phi(\mathbf{y})\rangle\right|^2. \tag{25}$$

Here $\langle\ ,\ \rangle$ denotes the standard internal scalar product between vectors in $\mathcal{H}$. This definition clearly yields a kernel that satisfies the two kernel properties. How do we calculate the kernel in practice?