IT496: Introduction to Data Mining

Lecture 19

Non-linear Support Vector Machines

(Slides are created from the book Hands-on ML by Aurelien Geron)

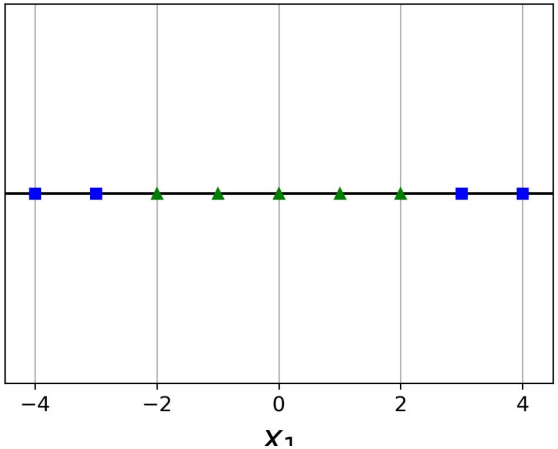
Arpit Rana

3rd October 2023

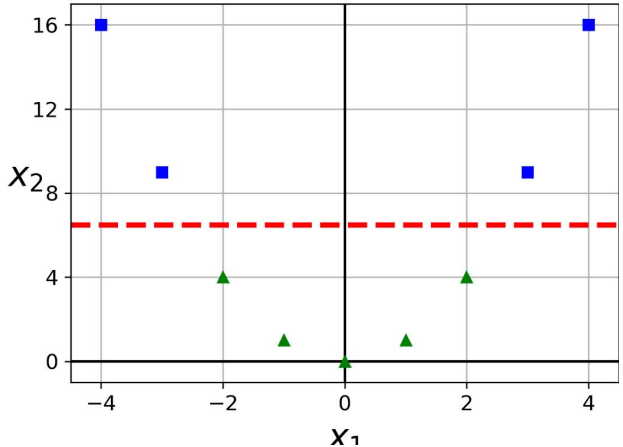
SVM with Polynomial Features

Many datasets are not even close to being linearly separable.

- One approach to handling nonlinear datasets is to add more features, such as polynomial features (as we did in polynomial regression);
- In some cases this can result in a linearly separable dataset.



Not linearly separable with one feature

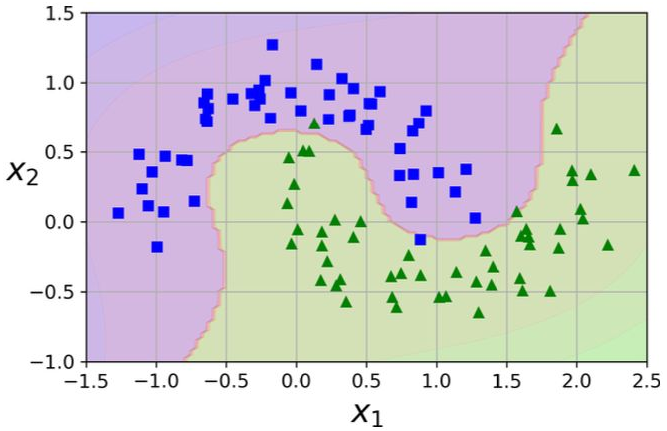


After adding second feature $x_2 = x_1^2$

SVM with Polynomial Features

Adding polynomial features is simple to implement and can work great with all sorts of Machine Learning algorithms (not just SVMs).

For example,




```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge"))
    ])

polynomial_svm_clf.fit(X, y)
```

SVM with Polynomial Features

However,

- at a *low polynomial degree* it cannot deal with very complex datasets, and
- with a *high polynomial degree* it creates a huge number of features, making the model too slow.
    - **`PolynomialFeatures(degree=d)`** transforms a dataset that had **`n`** features into one that has **`(n+d)!/n!d!`** features.

Non-linear SVM

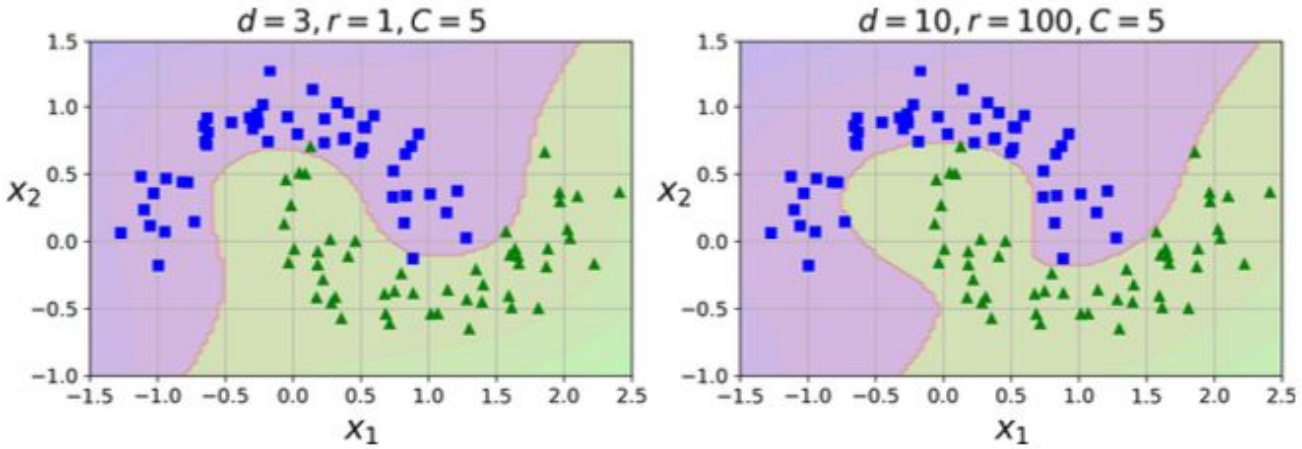
When using SVMs you can apply a mathematical technique called the ***kernel trick*** (it is explained in further slides).

- It makes it possible to get the same result as if you added many polynomial features, even with very high-degree polynomials, without actually having to add them.
- This trick is implemented by the SVC class.

Non-linear SVM

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
```

The hyperparameter `coef0` controls how much the model is influenced by high-degree polynomials versus low-degree polynomials



Understanding the Kernel Trick

Suppose you want to apply a 2nd-degree polynomial transformation to a two-dimensional training set, then train a linear SVM classifier on the transformed training set.

The 2nd-degree polynomial mapping function ϕ that you want to apply.

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

Notice that the transformed vector is three-dimensional instead of two-dimensional.

if we apply this 2nd-degree polynomial mapping and then compute the dot product of the transformed vectors:

$$\phi(\mathbf{a})^T \phi(\mathbf{b}) = \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^T \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^T \mathbf{b}\right)^2$$

The dot product of the transformed vectors is equal to the square of the dot product of the original vectors:

$\phi(\mathbf{a})^T \phi(\mathbf{b}) = (\mathbf{a}^T \mathbf{b})^2$

Understanding the Kernel Trick

So, you don't actually need to transform the training instances at all: just replace the dot product by its square.

$$\langle \phi(a), \phi(b) \rangle = \phi(a)^T \phi(b) = \left(a^T b\right)^2$$

- The result will be strictly the same as if you went through the trouble of actually transforming the training set, and then fitting a linear SVM algorithm.
- This trick makes the whole process much more computationally efficient. This is the essence of the ***kernel trick***.

Kernel Function

A ***kernel*** is a function capable of computing the dot product $\langle \phi(a), \phi(b) \rangle$ based only on the original vectors **a** and **b**, without having to compute (or even to know about) the transformation ϕ.

$$K(a, b) = \langle \phi(a), \phi(b) \rangle = f(a, b)$$

where *f*(.) is a function that follows certain conditions as stated by the Mercer's Theorem.

Some commonly used kernel functions are as follows.

$$\begin{aligned}
\text{Linear:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b} \\
\text{Polynomial:} \quad & K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^T\mathbf{b} + r\right)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T\mathbf{b} + r\right)
\end{aligned}$$

Kernel Function

When to use What

- As a rule of thumb, you should always try the linear kernel first (remember that **`LinearSVC`** is much faster than **`SVC(kernel="linear")`**, especially if the training set is very large or if it has plenty of features.
- If the training set is not too large, you should try the Gaussian RBF kernel as well; it works well in most cases.
- Then, if you have spare time and computing power, you can also experiment with a few other kernels using *cross-validation* and *grid search*.

Learning Non-linear SVM

Using a suitable function, $\phi(.)$, we can transform any data instance $\mathbf{x}$ to $\phi(\mathbf{x})$.

- The linear hyperplane in the transformed space can be expressed as $w^T\phi(\mathbf{x}) + b = 0$.
- To learn the optimal separating hyperplane, we can substitute $\phi(\mathbf{x})$ for $\mathbf{x}$ in the formulation of SVM to obtain the following hard margin optimization (primal) problem:

$$\underset{\mathbf{w}, b}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^T\phi(\mathbf{x}^{(i)}) + b\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m$$

Primal to Dual Problem

$$\underset{\mathbf{w},\,b}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m$$

$$L = \frac{1}{2}w^T w - \sum_i \alpha^{(i)}\left[t^{(i)}\left(w^T x^{(i)} + b\right) - 1\right]$$

w is the linear sum of the samples x.

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha^{(i)} t^{(i)} x^{(i)} = 0 \implies w = \sum_i \alpha^{(i)} t^{(i)} x^{(i)}$$

$$\frac{\partial L}{\partial b} = -\sum_i \alpha^{(i)} t^{(i)} = 0 \implies \sum_i \alpha^{(i)} t^{(i)} = 0$$

Replacing w in the above expression of L

$$L = \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)} t^{(i)} t^{(j)} x^{(i)} \cdot x^{(j)} - \sum_i \alpha^{(i)}$$

Learning Non-linear SVM

It is possible to express a different but closely related problem, called its ***dual problem***.

$$\underset{\alpha}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \underbrace{\mathbf{x}^{(i)T} \mathbf{x}^{(j)}}_{\phi(\mathbf{x}^{(i)}) \,.\, \phi(\mathbf{x}^{(j)})} - \sum_{i=1}^{m} \alpha^{(i)}$$

$$\text{subject to} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

- The solution to the dual problem typically gives a lower bound to the solution of the primal problem,
- however, under some conditions it can even have the same solutions as the primal problem.
    - In the primal, the objective function is convex, and the inequality constraints are continuously differentiable and convex functions, thus, meet those conditions.

Learning Non-linear SVM

$$\underset{\alpha}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \underbrace{\mathbf{x}^{(i)T} \mathbf{x}^{(j)}}_{\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})} - \sum_{i=1}^{m} \alpha^{(i)}$$

$$\text{subject to} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

- Once you find the vector $\hat{\alpha}$ that minimizes this equation (using a QP solver), you can compute $\hat{w}$ and $\hat{b}$ that minimize the primal problem by using the following.

$$\widehat{\mathbf{w}} = \sum_{i=1}^{m} \widehat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$

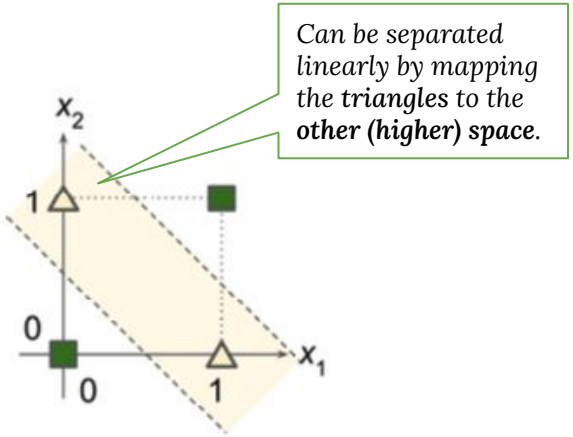
$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \widehat{\mathbf{w}}^T \mathbf{x}^{(i)} \right)$$

- The dual problem is faster to solve than the primal when *the number of training instances is smaller than the number of features.*

Non-linear SVM

- Finally, the decision rule will look like the following.

$$\hat{y} = \begin{cases} +1 & w^T x + b \geq 0, \\ -1 & w^T x + b < 0 \end{cases}$$



$$\phi(\mathbf{x}^{(i)})^{\mathrm{T}}\phi(\mathbf{x}^{(j)}) = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$\sum_i \alpha^{(i)} t^{(i)} x^{(i)} \cdot x^{(j)} + b \geq 0$$

- Dual makes the *kernel trick* possible, while the primal does not.

Next lecture

Decision Trees

5th October 2023