What is a Support Vector Machine?

A Support Vector machine (SVM) is a powerful and versatile machine learning model,**capable of performing linear and Nonlinear Classification,Regression and even outlier detection.**

SVMs are particularly **well suited for classification of complex small or medium-sized datasets.**

What made a SVM powerful and versatile?

credit:https://scikit-learn.org/stable/modules/svm.html#

The **advantages** of support vector machines are:

Effective in **high dimensional spaces**.

Still effective in cases **where number of dimensions is greater than the number of samples.**

Uses a **subset of training points** in the **decision function (called support vectors), so it is also memory efficient.**

Versatile: **different Kernel functions** can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Drawbacks?

The **disadvantages** of support vector machines include:

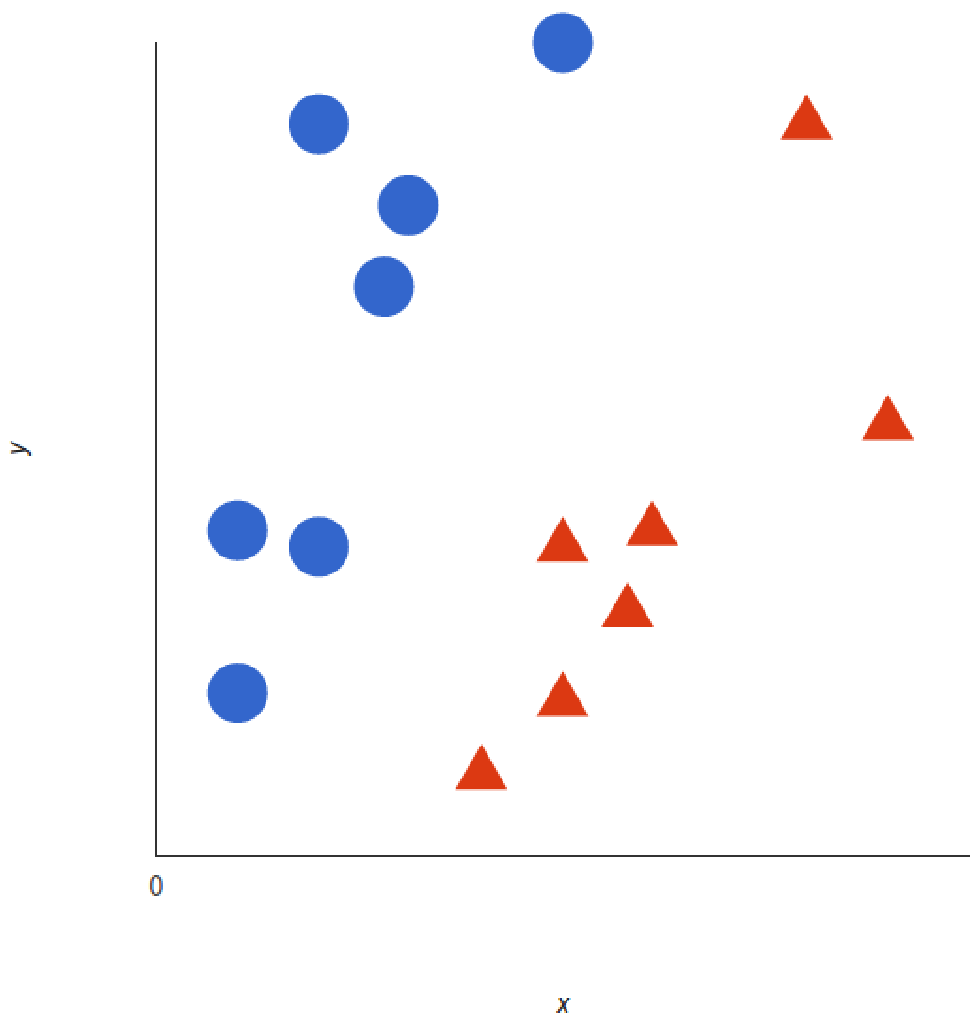
If the **number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.**

SVMs **do not directly provide probability estimates**, these are **calculated** using an **expensive five-fold cross-validation**.

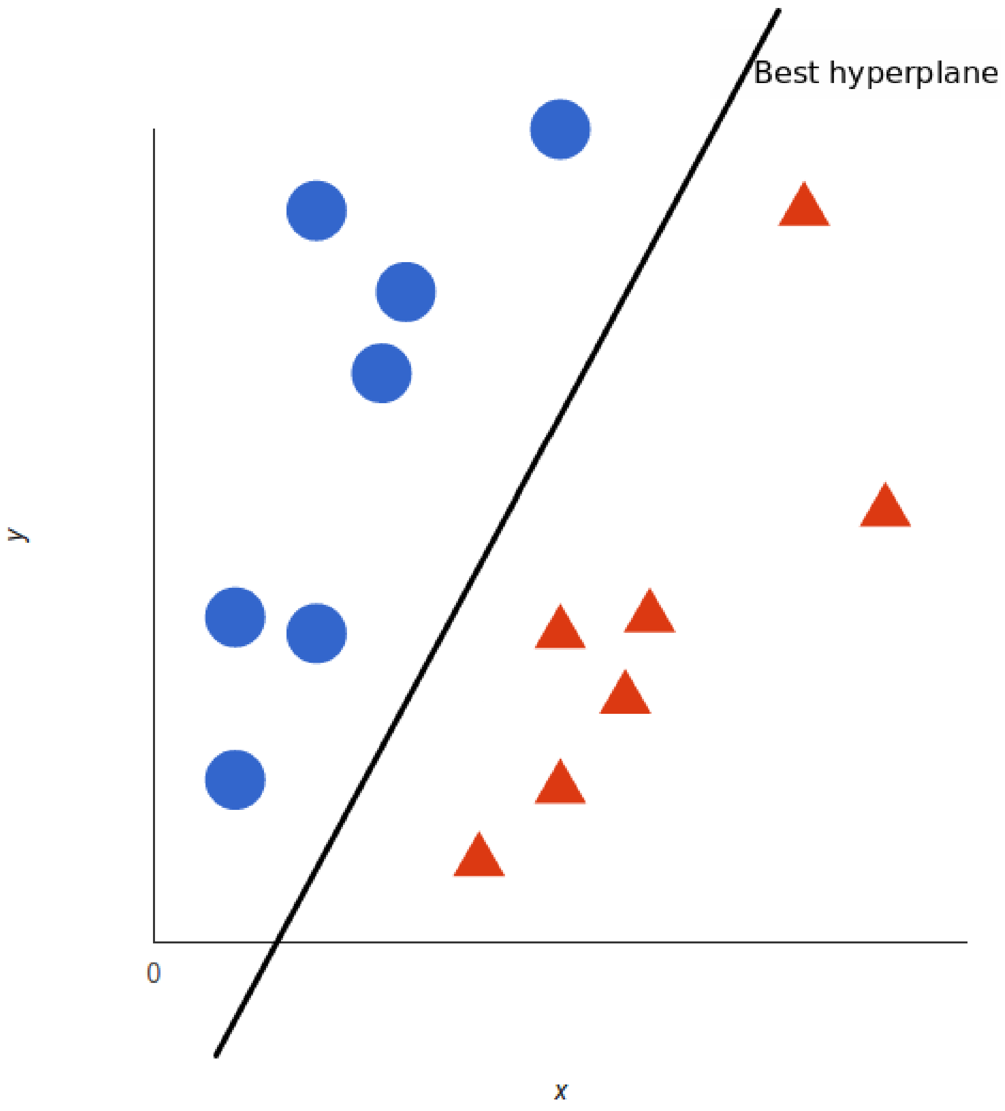
How Does SVM Work?

credits:https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

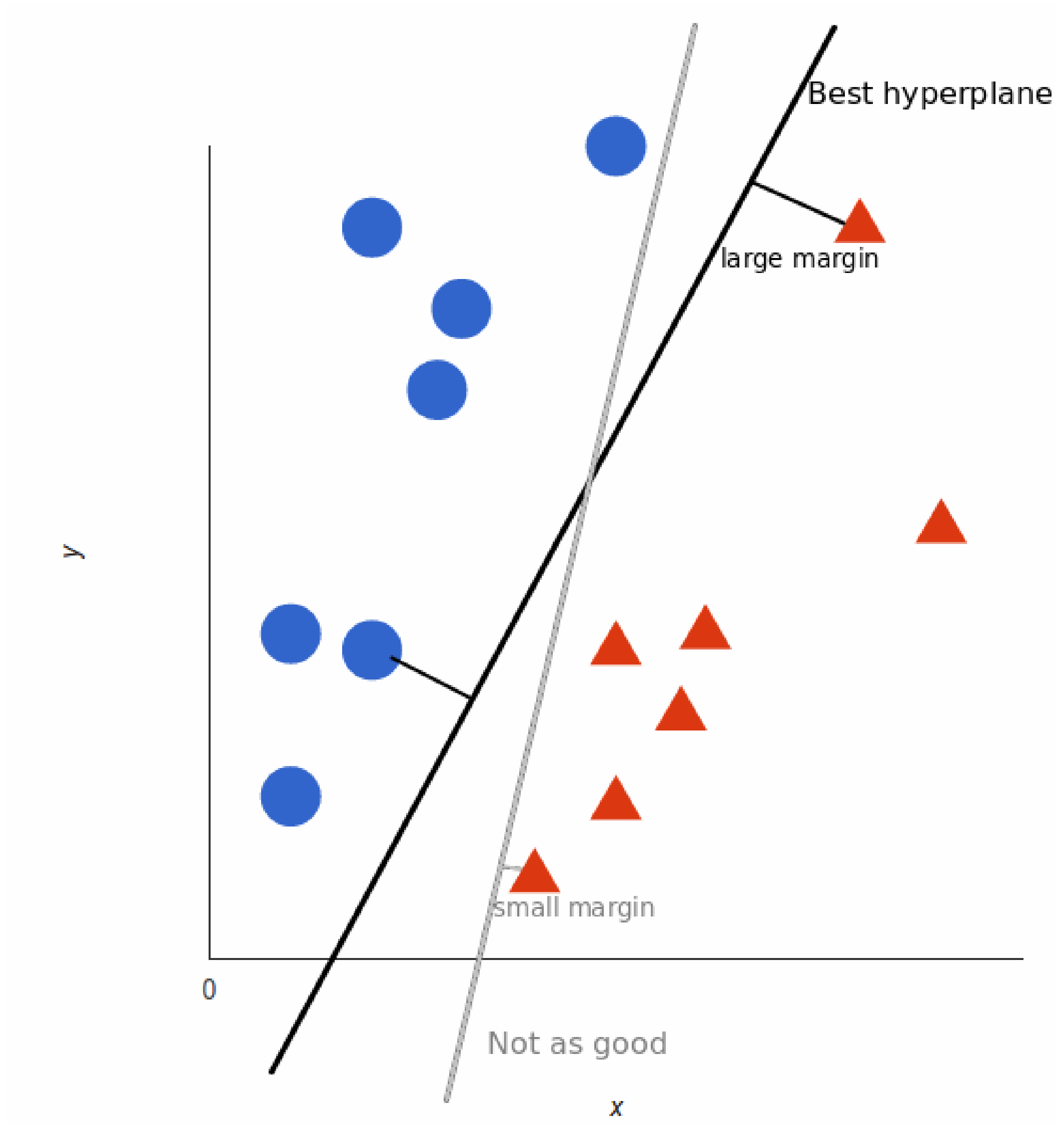
The basics of Support Vector Machines and how it works are best understood with a simple example. Let’s imagine we have two tags: red and blue, and our data has two features: x and y. We want a classifier that, given a pair of (x,y) coordinates, outputs if it’s either red or blue. We plot our already labeled training data on a plane:



A support vector machine takes these data points and outputs the hyperplane (which in two dimensions it’s simply a line) that best separates the tags. This line is the decision boundary: anything that falls to one side of it we will classify as blue, and anything that falls to the other as red.



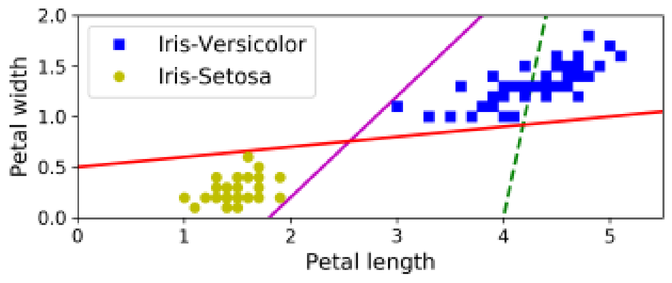
But, what exactly is the best hyperplane? For SVM, it’s the one that maximizes the margins from both tags. In other words: the hyperplane (remember it's a line in this case) whose distance to the nearest element of each tag is the largest.



Double-click (or enter) to edit

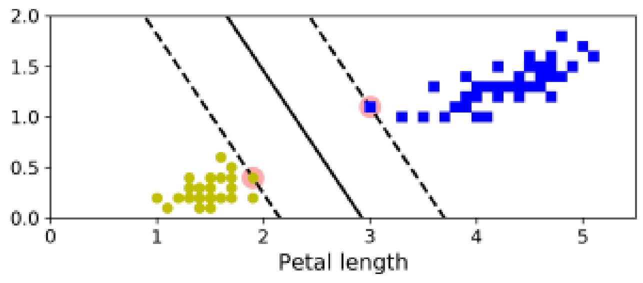
Linear SVM Classification

The fundamental idea behind SVM is,the classes can be clearly separated easily with a straight line [They are linearly separable].



The above figure shows the decision boundary of three possible linear classifiers.

The model whose decision boundary is represented by the dashed line is so bad that it does not even separate the classes properly.



The solid line in the plot on the right represents the decision boundary of an SVM classifier;this line not only separates the two classes but also stays as far away from the closest training instances as possible.

We can think of an SVM classifier as fitting the widest possible street between classes.This is called **large margin classifier.**

Notice that adding more training instances "off the street" will not affect the decision boundary at all;it is fully determined(or "Supported") bythe instances located on the edge of the street.

SVMs are **sensitive** to the **feature scales**

Soft Margin Classification

If we **strictly impose that all instances be off the street** and on the right side, **this is called hard margin classification**.

There are two main **issues** with **hard margin classification**.

First, it **only works** if the **data** is **linearly separable**, and second it is **quite sensitive** to **outliers**.

To avoid these issues it is preferable to use a more flexible model.

The **objective** is to **find a good balance between keeping the street as large as possible and limiting the margin violations** (i.e., instances that end up in the middle of the street or even on the wrong side). **This is called soft margin classification.**

When creating an SVM model using Scikit-learn,we can specify a number of Hyperparmeters.

C is one of those hyperparameters

On the one side, using a **low C value** the **margin is quite larg**e, but **many instances end up on the street**.

On the other side, using a **high C value** the **classifier makes fewer margin violations but ends up with a smaller margin**.

The LinearSVC class **regularizes** the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the **StandardScaler**.

Nonlinear SVM Classifcation

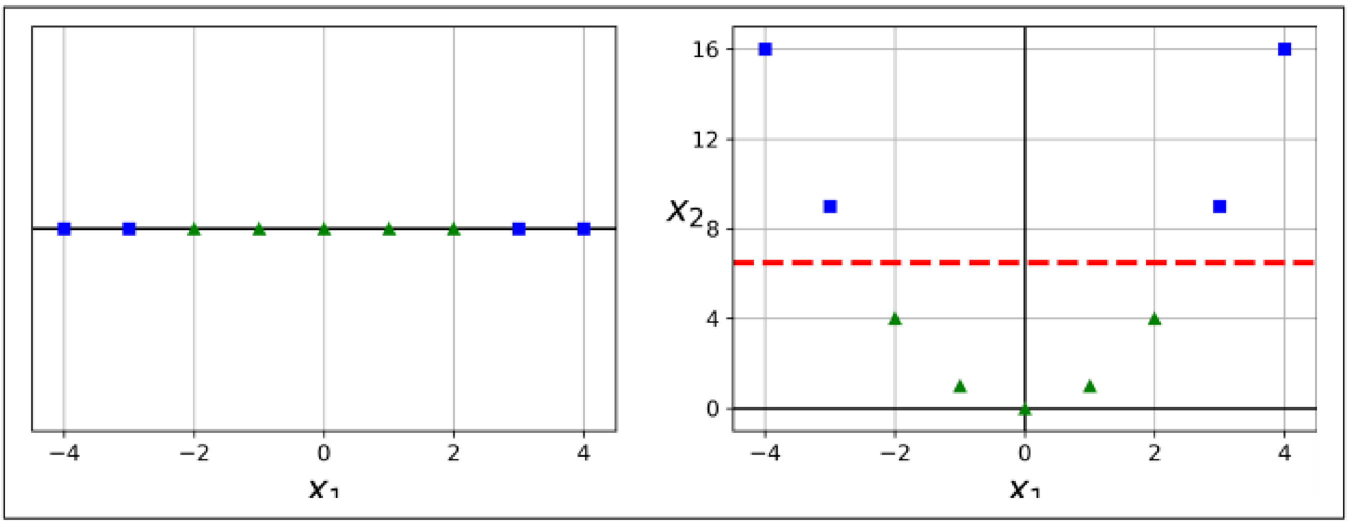
Double-click (or enter) to edit

Although linear SVM classifiers are efficient and work surprisingly well in many cases, 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** in some cases this can result in a linearly separable dataset.

Self explanatory Image creating a Linearly separable data

X2=(X1)**2



Polynomial Kernel

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

Here comes a beautiful concept named as "**Kernal Trick**"

kernel trick

It makes it possible to get the same result as if you added many polynomial features, even with very highdegree polynomials, **without actually having to add them.** So there is no combinatorial explosion of the number of features since you don’t actually add any features. This trick is implemented by the SVC class.

A common approach to find the right hyperparameter values is to use **grid search**. It is often faster to first do a very coarse grid search, then a finer grid search around the best values found.

Having a good sense of what each hyperparameter actually does can also help you search in the right part of the hyperparameter space.

Adding Similarity Features

Another technique to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark.

Gaussian RBF Kernel

Just like the polynomial features method, the similarity features method can be useful with any Machine Learning algorithm, but it may be computationally expensive to compute all the additional features, especially on large training sets. However, once again the kernel trick does its SVM magic: it makes it possible to obtain a similar result as if you had added many similarity features, without actually having to add them.

Computational Complexity

```
1. LinearSVC class-----O(m × n)
2.  SGDClassifier -----O(m × n)
3.  SVC ---------------O(m² × n) to O(m³ × n)
```

SVM Regression

The SVM algorithm is quite versatile: not only does it support linear and nonlinear classification, but it also supports linear and nonlinear regression. The trick is to reverse the objective: instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations

The width of the street is controlled by a **hyperparameter ϵ.**

Hyperparameters to tune in SVM

C parameter adds a penalty for each misclassified data point. If c is small, the penalty for misclassified points is low so a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications.

gamma decreases, the regions separating different classes get more generalized. Very large gamma values result in too specific class regions (overfitting).

The **hyperparameter coef0** controls how much the model is influenced by highdegree polynomials versus low-degree polynomials.

Objective of the model

Hard margin linear SVM classifier objective

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

Soft margin linear SVM classifier objective

$$\begin{aligned} &\underset{\mathbf{w}, b, \zeta}{\text{minimize}} && \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m} \zeta^{(i)} \\ &\text{subject to} && t^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m \end{aligned}$$

To get the soft margin objective, we need to introduce a slack variable ζ (i) ≥ 0 for each instance:4ζ(i) measures how much the ith instance is allowed to violate the margin.

We now have two conflicting objectives: making the slack variables as small as possible to reduce the margin violations, and making 1/2wT*w as small as possible to increase the margin. This is where the C hyperparameter comes in: it allows us to define the trade-off between these two objectives.

Hyperparamter C will takecare the trade-off between the two objectives.