

در [مطلب قبل](#) با ساختار کلی کتابخانه Accord.NET آشنا شدیم. در این قسمت پس از فراگیری نحوه‌ی فراخوانی کتابخانه، به اجرای اولین برنامه‌ی کاربردی به کمک آکورد دات نت می‌پردازیم.

برای استفاده از Accord.NET می‌توان به یکی از دو صورت زیر اقدام کرد :
دریافت آخرین نسخه‌ی متن باز و یا d11های پروژه‌ی Accord.NET [از طریق گیت هاب](#)

نصب [از طریق NuGet](#) (با توجه به این که در چارچوب Accord.NET کتابخانه‌های متنوعی وجود دارند و در هر پروژه نیاز به نصب همگی آنها نیست، فضای نام‌های مختلف در بسته‌های مختلف نیوگت قرار گرفته‌اند و برای نصب هر کدام می‌توانیم یکی از فرمان‌های زیر را استفاده کنیم)

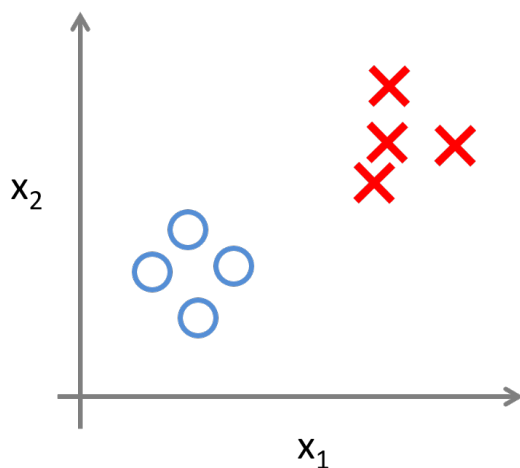
```
PM> Install-Package Accord.MachineLearning
```

```
PM> Install-Package Accord.Imaging
```

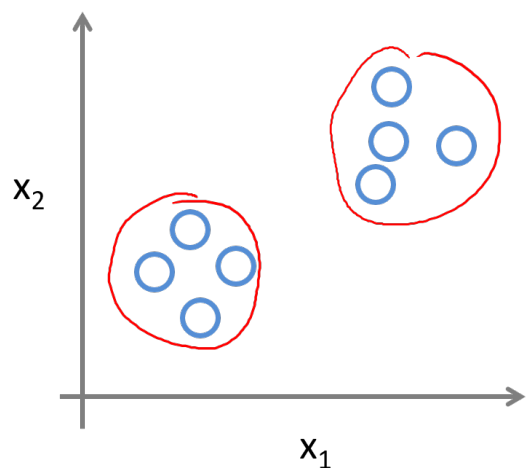
```
PM> Install-Package Accord.Neuro
```

در اولین برنامه‌ی کاربردی خود می‌خواهیم الگوریتم ماشین بردار پشتیبان یا support vector machine را که یکی از روش‌های یادگیری **بانظارت** است برای **طبقه‌بندی** مورد استفاده قرار دهیم.
نکته : روش‌های یادگیری به دو دسته کلی با نظارت (*Supervised learning*) و بدون نظارت (*Unsupervised learning*) تقسیم بندی می‌شوند. در روش با نظارت، داده‌ها دارای برچسب یا label هستند و عملاً نوع کلاس‌ها مشخص هستند و اصطلاحاً برای طبقه بندی (*Classification*) استفاده می‌شوند. در روش بدون نظارت، داده‌هایمان بدون برچسب هستند و فقط تعداد کلاس‌ها و نیز یک معیار تفکیک پذیری مشخص است و برای خوشه بندی (*Clustering*) استفاده می‌شوند.

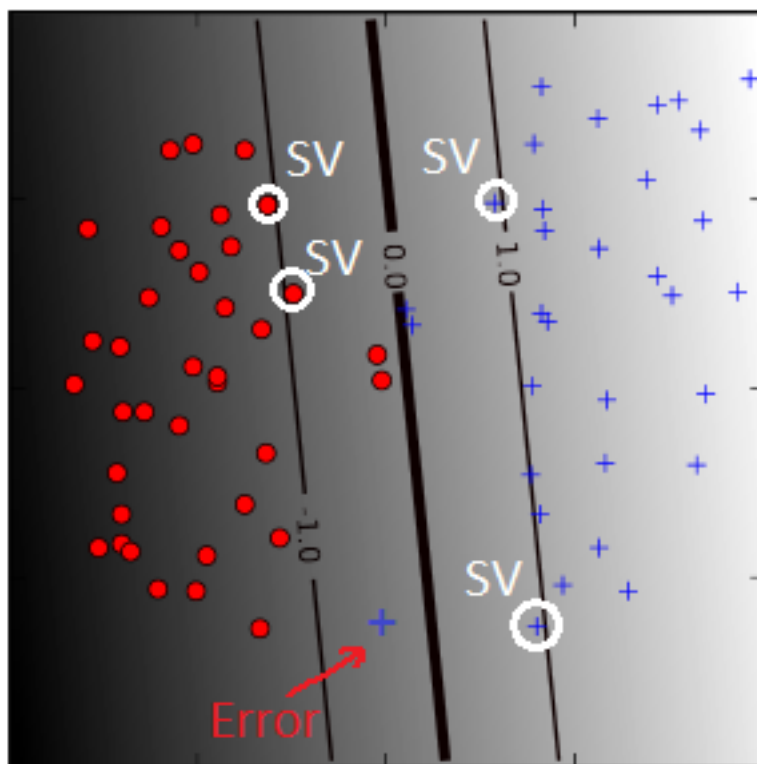
Supervised Learning



Unsupervised Learning

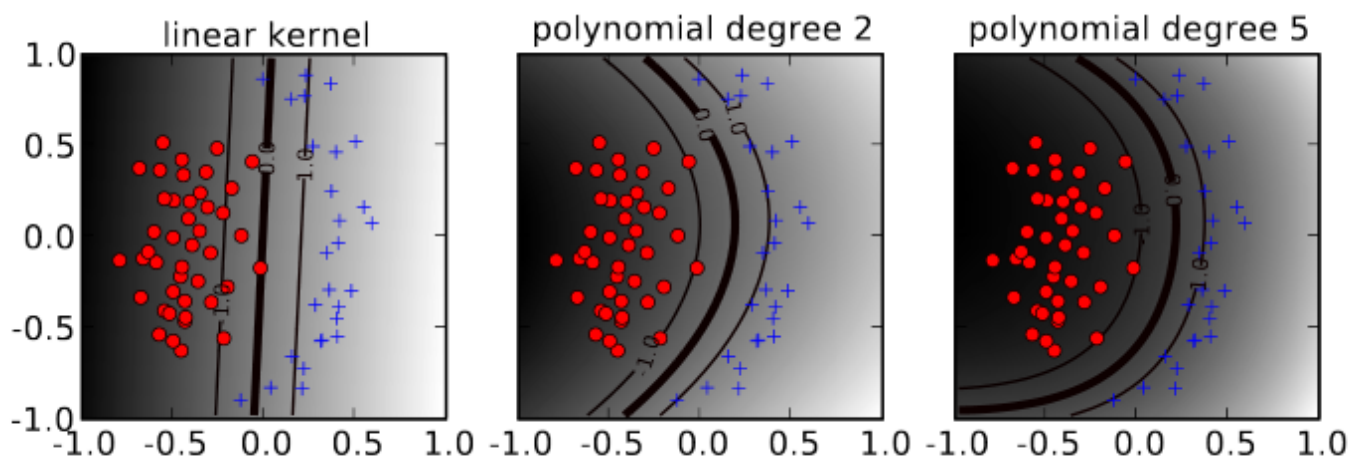


عملکرد SVM یا ماشین بردار پشتیبان به صورت خلاصه به این صورت است که با در نظر گرفتن یک خط یا ابرصفحه جدا کننده فرضی، ماشین یا دسته بندی را ایجاد می کند که از نقاط ابتدایی کلاس های مختلف که بردار پشتیبان یا SV نام دارند، بیشترین فاصله را دارند و در نهایت داده ها را به دو کلاس مجزا تقسیم می کند.



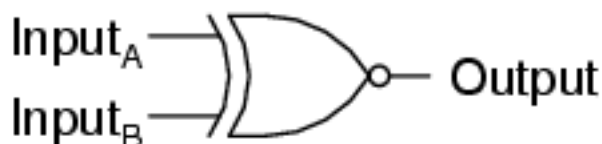
در تصویر بالا مقداری خطا مشاهده می شود که با توجه با خطی بودن جداساز مجبور به پذیرش این خطا هستیم.

در نسخه های جدیدتر این الگوریتم یک Kernel (از نوع خطی Linear، چند جمله ای Polynomial، گوسین Gaussian و یا ...) برای آن در نظر گرفته شد که عملاً نگاشتی را بین خط (نه صرفاً فقط خطی) را با آن ابرصفحه جداکننده برقرار کند. در نتیجه دسته بندی با خطای کمتری را خواهیم داشت. (اطلاعات بیشتر در [+](#) و همچنین مطالب دکتر سعید شیری درباره SVM در [+](#))



یک مثال مفهومی : هدف اصلی در این مثال شبیه سازی تابع XNOR به Kernel SVM می باشد.

Exclusive-NOR gate



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

برای شروع کار از فضای نام MachineLearning استفاده می کنیم و بسته ی نیوگت مربوطه را فرخوانی می کنیم. پس از اجرا، مشاهده می کنیم که فضای نام های Accord.Math و Accord.Statistics نیز به پروژه اضافه می شود.

```
PM> Install-Package Accord.MachineLearning
Attempting to resolve dependency 'Accord (≥ 3.0.2)'.
Attempting to resolve dependency 'Accord.Math (≥ 3.0.2)'.
Attempting to resolve dependency 'Accord.Statistics (≥ 3.0.2)'.
Installing 'Accord 3.0.2'.
```

در ابتدا مقادیر ورودی و برجسب ها را تعریف می کنیم

```
// ورودی
double[][] inputs =
{
    new double[] { 0, 0 }, // 0 xnor 0: 1 (label +1)
    new double[] { 0, 1 }, // 0 xnor 1: 0 (label -1)
    new double[] { 1, 0 }, // 1 xnor 0: 0 (label -1)
    new double[] { 1, 1 }  // 1 xnor 1: 1 (label +1)
};

// خروجی دسته بند ماشین بردار پشتیبان باید -1 یا +1 باشد
int[] labels =
```

```
{
    // 1, 0, 0, 1
    1, -1, -1, 1
};
```

پس از انتخاب نوع کرنل یا هسته، دسته‌بندمان را تعریف می‌کنیم :

```
// ساخت کرنل
IKernel kernel = createKernel();

// ساخت دسته بند به کمک کرنل انتخابی و تنظیم تعداد ویژگی‌ها ورودی‌ها به مقدار 2
KernelSupportVectorMachine machine = new KernelSupportVectorMachine(kernel, 2);
```

تابع ساخت کرنل :

```
private static IKernel createKernel()
{
    //var numPolyConstant = 1;
    //return new Linear(numPolyConstant);

    //var numDegree = 2;
    //var numPolyConstant = 1;
    //return new Polynomial(numDegree, numPolyConstant);

    //var numLaplacianSigma = 1000;
    //return new Laplacian(numLaplacianSigma);

    //var numSigAlpha = 7;
    //var numSigB = 6;
    //return new Sigmoid(numSigAlpha, numSigB);

    var numSigma = 0.1;
    return new Gaussian(numSigma);
}
```

و سپس بایستی این Classifier را به یک الگوریتم یادگیری معرفی کنیم. الگوریتم بهینه سازی حداقلی ترتیبی (Sequential Minimal Optimization) یکی از روش‌های یادگیری است که برای حل مسائل بزرگ درجه دوم بکار می‌رود و معمولاً برای آموزش دسته بندی SVM از همین آموزنده استفاده می‌شود :

```
// معرفی دسته بندمان به الگوریتم یادگیری SMO
SequentialMinimalOptimization teacher_smo = new SequentialMinimalOptimization(machine_svm,
inputs, labels);

// اجرای الگوریتم یادگیری
double error = teacher_smo.Run();
Console.WriteLine(string.Format("error rate : {0}", error));
```

در نهایت می‌توانیم به عنوان نمونه برای آزمایش یکی از مقادیر ورودی را مورد بررسی قرار دهیم و خروجی کلاس را مشاهده کنیم.

```
// بررسی یکی از ورودی‌ها
var sample = inputs[0];
int decision = System.Math.Sign(machine_svm.Compute(sample));
Console.WriteLine(string.Format("result for sample '{0} xor {0}' is : {0}", decision));
```

```
file:///E:/My Project/Smarts/Acco...
error rate : 0
result for sample '0 xor 0' is : 1
```

از این ساختار می‌توانیم برای طبقه بندی‌های با دو کلاس استفاده کنیم؛ مانند تشخیص جنسیت (مرد و زن) از طریق تصویر، تشخیص جنسیت (مرد و زن) از طریق صدا، تشخیص داشتن یا نداشتن یک بیماری خاص و برای ایجاد هر کدام از این برنامه‌ها نیاز به یک مجموعه داده، استخراج ویژگی از آن و سپس نسبت دادن آن به الگوریتم داریم. در جلسات آینده با مفاهیم استخراج ویژگی و SVM چند کلاس آشنا خواهیم شد.

[دریافت کد](#)