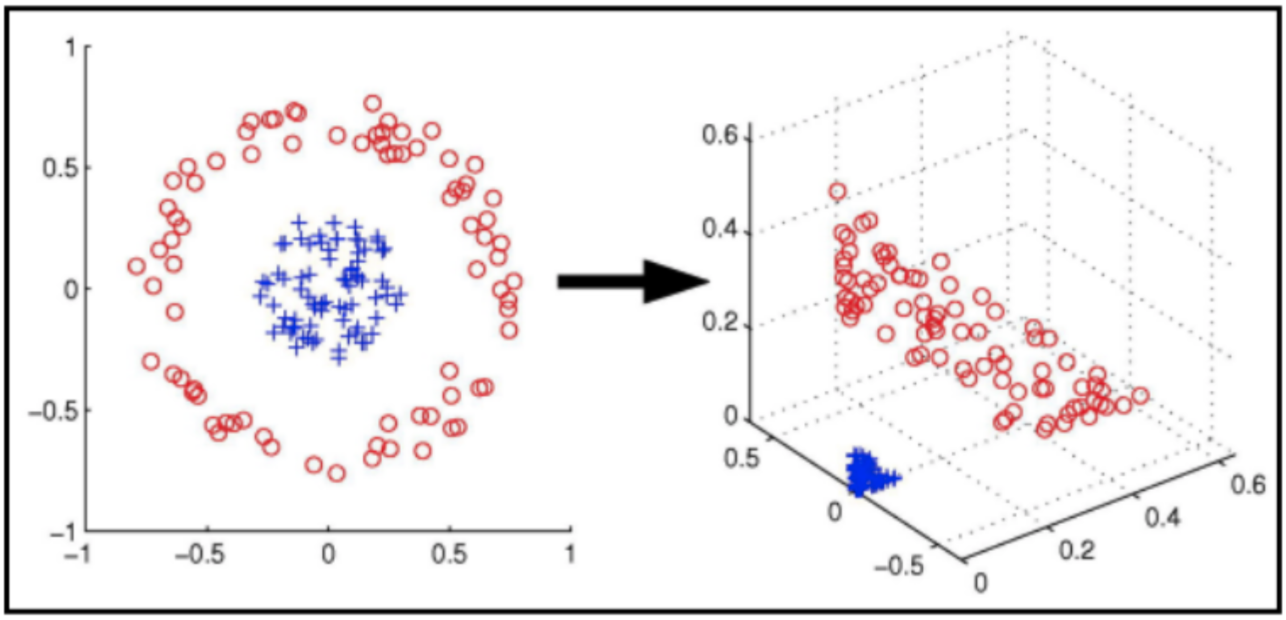
Support Vector Machines Explained

Zach Bedell
Dec 8, 2018 · 7 min read ★



Support vector machines (SVMs) are a popular linear classifier, the current version of which was developed by Vladimir Vapnik and Corinna Cortes. SVMs are supervised learning models, meaning sample data must be labeled, that can be applied to almost any type of data.

They are especially effective at **classification, numeral prediction, and pattern recognition tasks**. SVMs find a line (or hyperplane in dimensions greater than 2) in between different classes of data such that the distance on either side of that line or hyperplane to the next-closest data points is maximized.

In other words, support vector machines calculate a maximum-margin boundary that leads to a homogeneous partition of all data points. This classifies an SVM as a maximum margin classifier.

On the edge of either side of a margin lies sample data labeled as **support vectors**, with at least 1 support vector for each class of data. These support vectors represent the bounds of the margin, and can be used to construct the hyperplane bisecting that margin.

$$\vec{w} \cdot \vec{x} + b = 0 \qquad (1)$$
$$y = mx + b \qquad (2)$$

Equations 2 and 1 represent the formulas for a line or hyperplane respectively. For all sample data x, an SVM should find weights such that the data points will be separated according to a decision rule. To elaborate, lets assume we have a set of negative and positive values in a two-dimensional euclidean space, along with an initial straight line (drawn in green) between the two classes of data points:

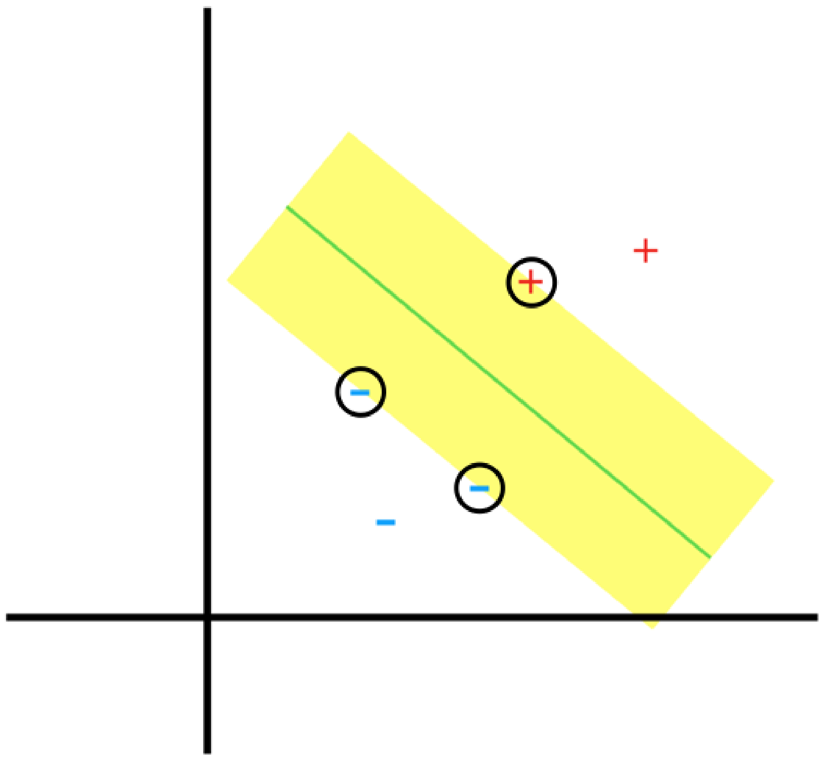


Fig. 1

The yellow space in Fig. 1 shows the margin between the points of opposing classes that are closest to each other. The sample points that are circled are the support vectors.

Now, we imagine a vector *w*, of arbitrary length constrained to be perpendicular to the median of that margin. Then, we take an unknown vector *u*, and we want to determine whether it is on the + or the - side of the margin. To do this, we project *u* onto our perpendicular vector *w*, which gives us the distance to *w*.

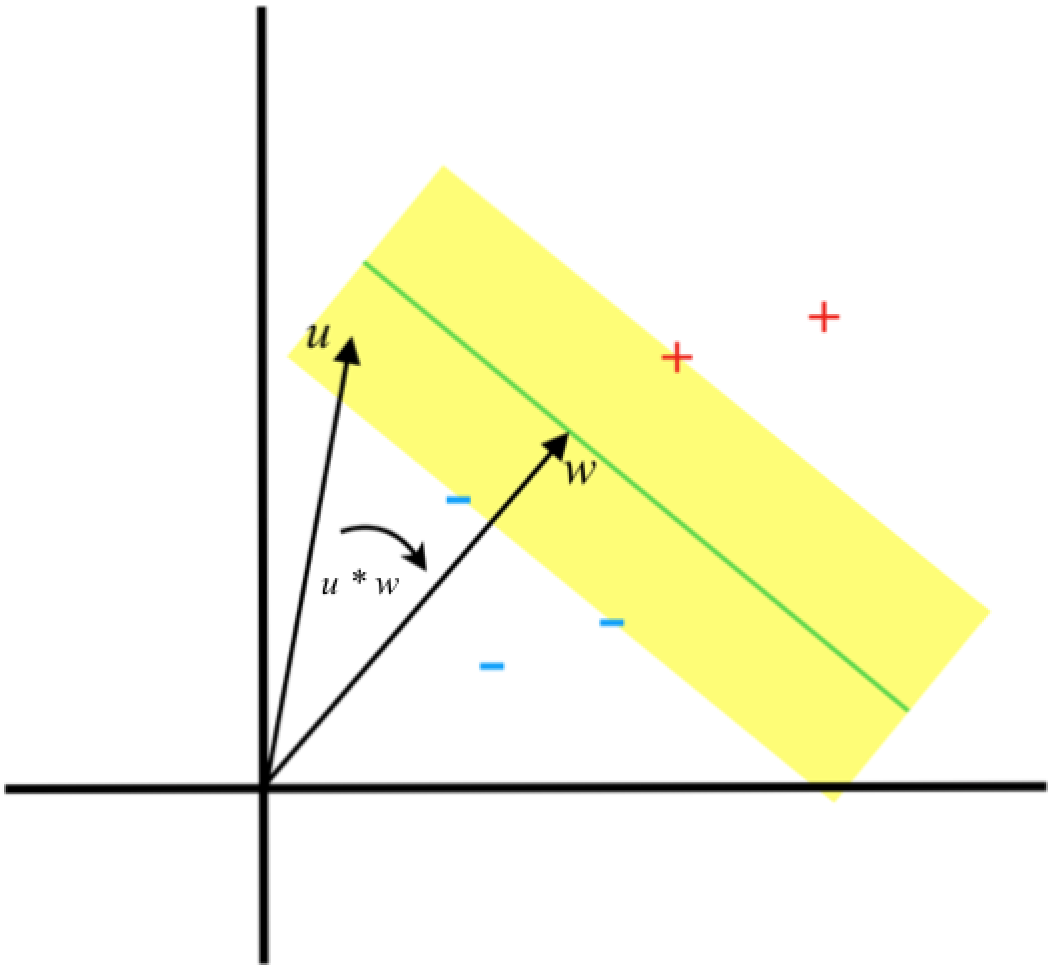


Fig. 2

Since *w* is perpendicular to the median, we know that the further we expand *w*, the closer we are to being on the + side of the margin. This means taking the dot product of our vectors *u* and *w* and checking whether it is greater than or equal to some constant, *c*. From here we have a decision rule, stating that if the equation

$$\vec{w} \cdot \vec{x} + b \geq 0 \qquad (3)$$

is satisfied, the sample must be of the + class. At this point, we still don't know what *b* or *w* is. Now we need to apply additional constraints to allow us to calculate them.

. . .

If we replace *u* in equation 3 with a sample we know to be positive, then with a sample we know to be negative, those equations must be greater than or equal to 1 and less than or equal to -1, respectively.

$$\overrightarrow{w} \cdot \overrightarrow{x_+} + b \geq 1 \qquad (4)$$
$$\overrightarrow{w} \cdot \overrightarrow{x_-} + b \leq 1 \qquad (5)$$

For convenience, we can introduce a new variable *y* such that *y* = 1 for positive samples and *y* = -1 for negative samples. This converts equations 4 and 5 into one equation.

$$y_i(\overrightarrow{x_i} \cdot \overrightarrow{w} + b)(-1) = 0 \qquad (6)$$

Equation 6 should hold for any samples that are classified as support vectors.

. . .

Since our initial goal was to establish a margin that is as wide as possible, we must determine a way to express the distance between the boundaries of the margin.

Let's draw two vectors from the origin to support vectors in our negative and positive classes, labeling them as *x*- and *x*+, respectively. Then we can draw a third vector between *x*- and *x*+ to show the distance vector between the two. Now we have enough vectors to calculate width by taking the dot product of our distance vector and our perpendicular vector *w*, then dividing by the magnitude of *w*.

$$width = (x_+ - x_-) \cdot \frac{w}{\| w \|} \quad (7)$$

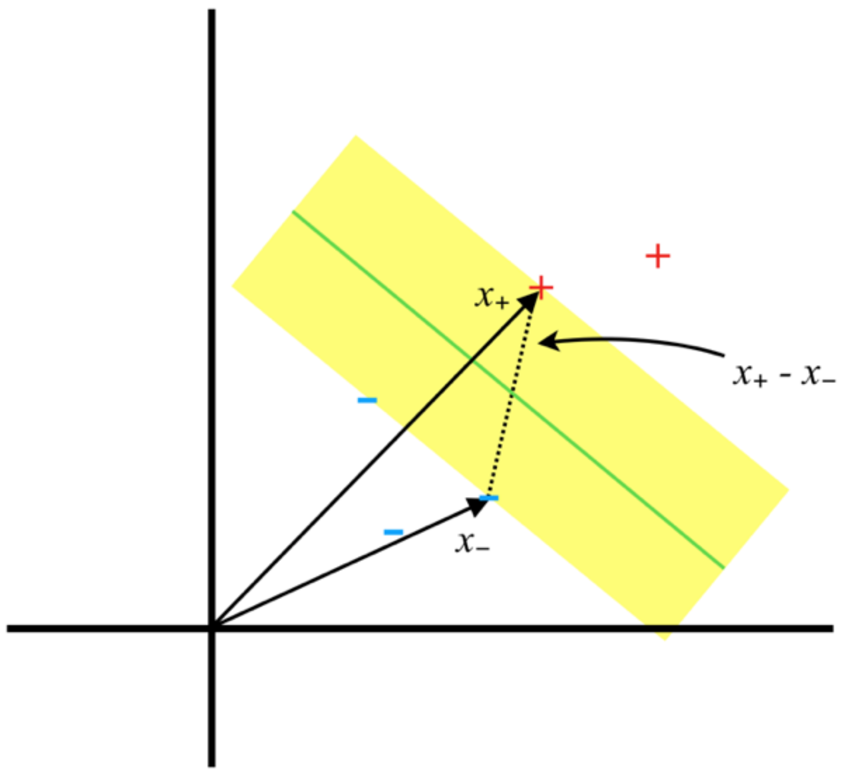


Fig. 3

From equation 6, and knowing that $y \geq 1$ for positive samples and $y \leq -1$, we can do some algebraic reduction and rewrite equation 7 to equation 8.

$$width = \frac{1}{\ } \cdot \frac{w}{\ }^{2} \quad (8)$$

$$2 \quad || w ||$$

To maximize equation 8, a function with constraints, we must use **LaGrange multipliers**. This will provide us a new function to maximize without needing to consider the constraints.

$$L = \frac{1}{2} \cdot \frac{w}{|| w ||}^2 - \sum_i^n [y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} + 1) - 1] \qquad (9)$$

First, we differentiate L with respect to *w* and find that the vector *w* is a linear linear sum of all or some of the samples.

$$\overrightarrow{w} = \sum_i^n \alpha_i \cdot y_i \cdot \overrightarrow{x_i} \qquad (10)$$

Differentiating *L* with respect to *b* gives:

$$b = \sum_i^n \alpha_i \cdot y_i = 0 \qquad (11)$$

Plugging our value for *w* in equation 10 into equation 9, we end up with equation 12.

$$L = \frac{1}{2}(\sum_i^n \alpha_i \cdot y_i \cdot \overrightarrow{x_i})(\sum_j^n \alpha_j \cdot y_j \cdot \overrightarrow{x_j}) - (\sum_i^n \alpha_i \cdot y_i \cdot \overrightarrow{x_i})(\sum_j^n \alpha_j \cdot y_j \cdot \overrightarrow{x_j}) - b(\sum_i^n \alpha_i \cdot y_i) + \sum_i^n \alpha_i$$

(12)

Further reduction gives us

$$L = \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \qquad (13)$$

If the result for equation 14 is ≥ 0, our sample is in the + class.

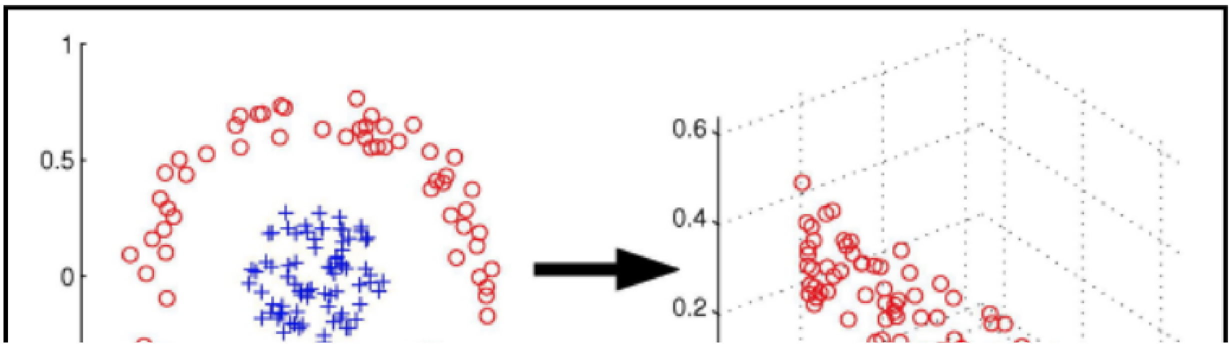
. . .

So far we've examined cases in which our classes of sample points are linearly separable.

In the case that our sample points are not linearly separable, we must do a transformation into a new space. This is accomplished with the use of a **kernel function**.

Kernel functions can provide us with the dot product of two vectors in a new space without us needing to know the transformation into that space. The most simple kernel function is the **linear kernel**, shown as equation 15.

$$k(x, y) = x^T y + c \qquad (15)$$



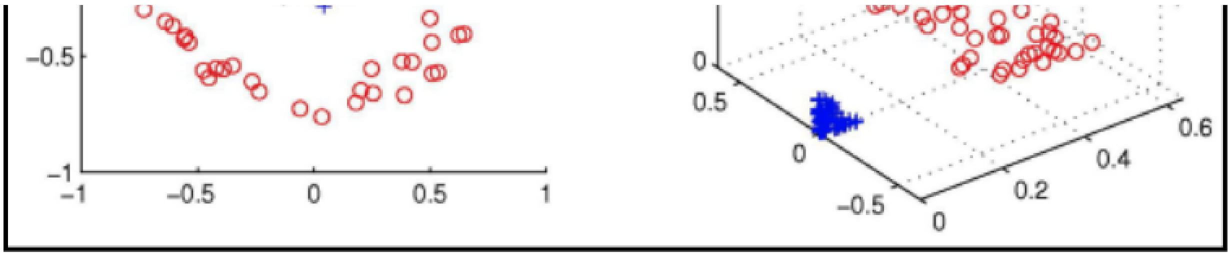


Fig. 4

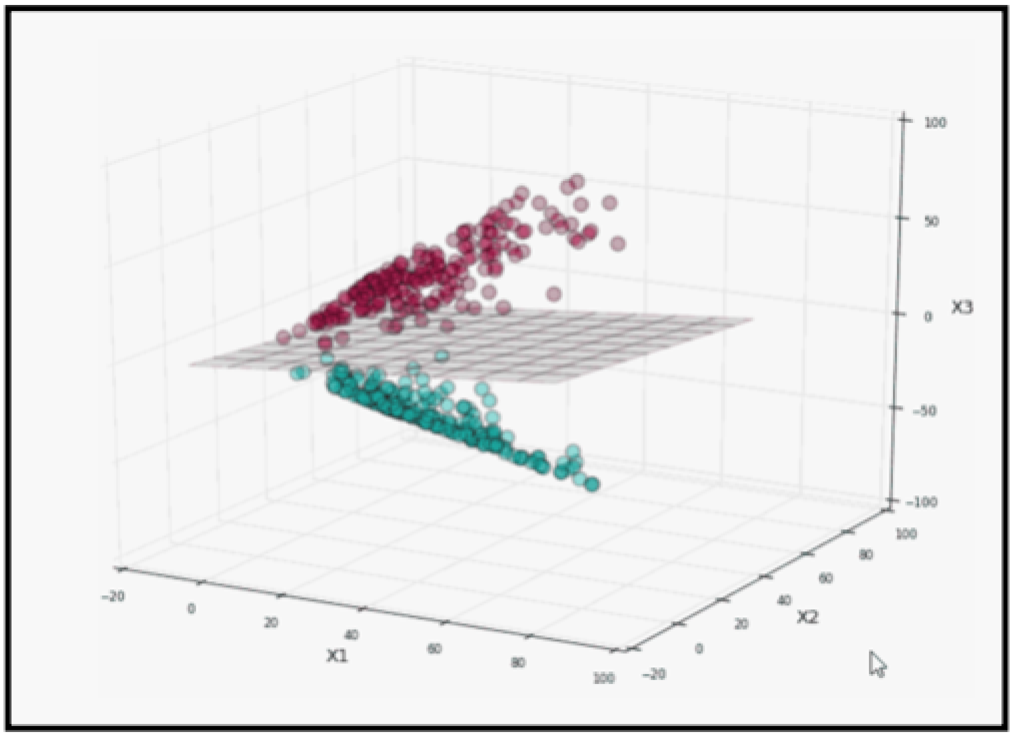
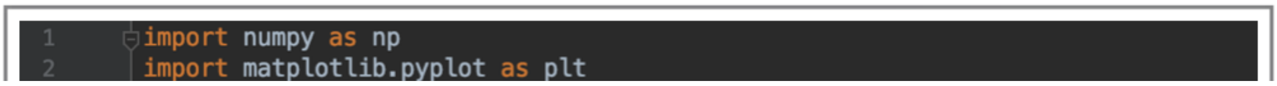


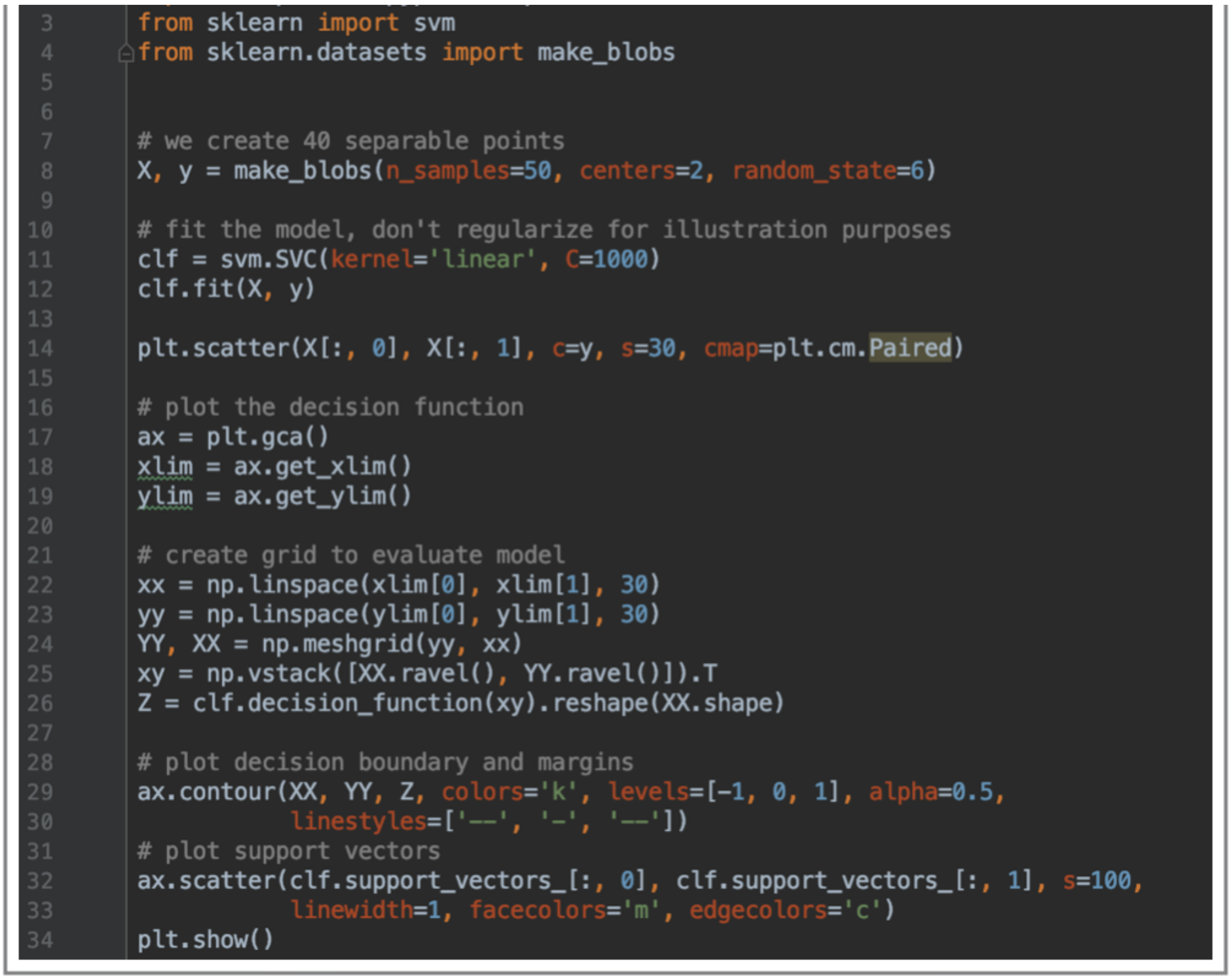
Fig. 5

Figure 4 shows an example of two classes of data that are non-linearly separable in two dimensions, but become separable once they are transformed to a three-dimensional space. Figure 5 shows a hyperplane separating two classes of data in three dimensions.

Current machine learning libraries such as Scikit-Learn or TensorFlow allow us implement SVMs while obscuring much of the complex math. Figure 6 shows a slightly modified example of a SVM implemented in Python with ScikitLearn. Figure 7 displays the result using the Matplotlib library.



```
import numpy as np
import matplotlib.pyplot as plt
```



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


# we create 40 separable points
X, y = make_blobs(n_samples=50, centers=2, random_state=6)

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='m', edgecolors='c')
plt.show()
```

Fig. 6

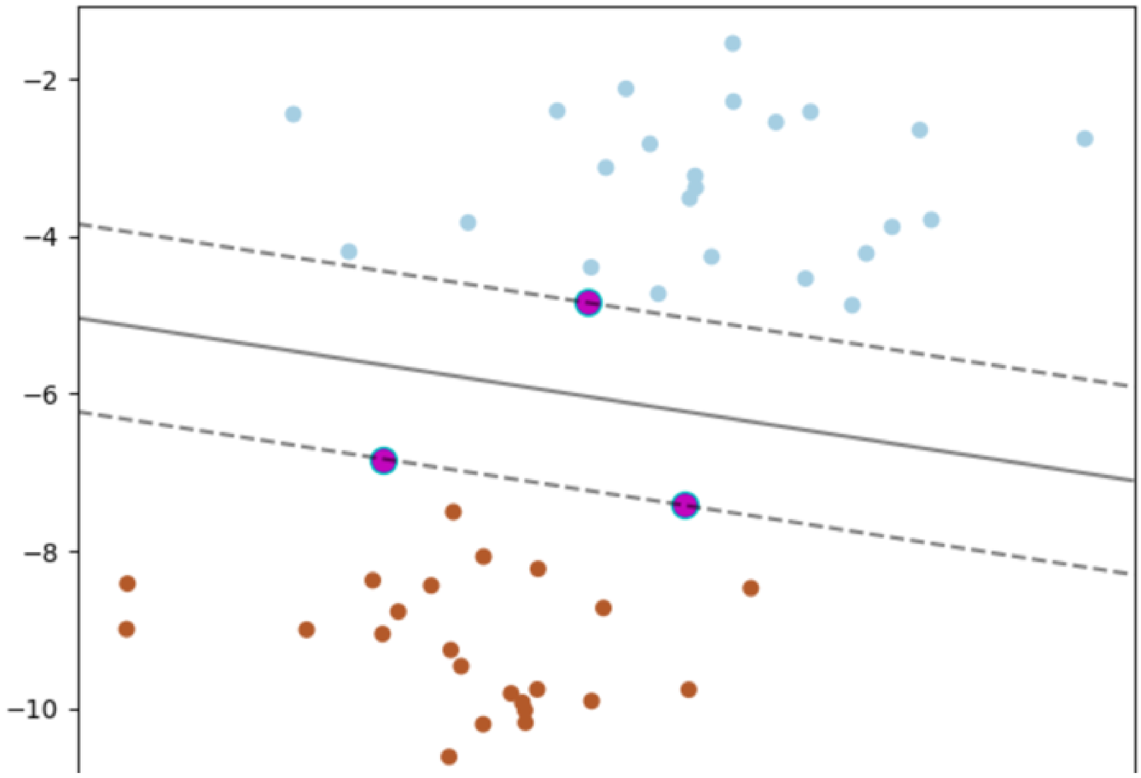


Fig. 7

Much of the code shown in figure 6 is housekeeping to correctly plot our result, but some areas of interest exist in lines 11 and 26. Line 11 invokes Scikit-learn's SVM tool, which takes a kernel type and penalty parameter C of the error term as parameters. Line 26 feeds our sample data to the SVM decision function.

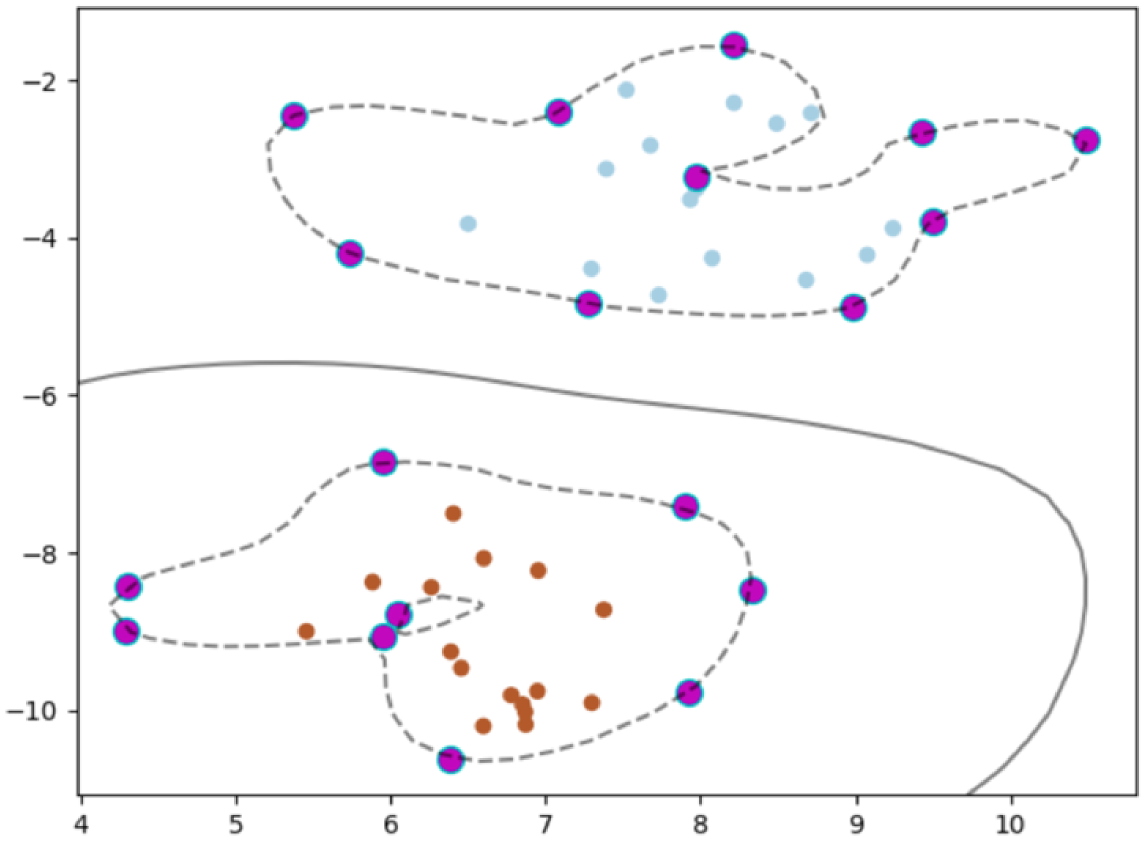


Fig. 8

Figure 8 shows the result of running the code with the radial basis function as the kernel. The radial basis function separates the two classes of data by setting the samples that represent the outer bounds of each class as the support vectors.

Vladimir Vapnik's research demonstrated the groundbreaking ability to classify data in infinite dimensions using SVMs, and they remain one of the most powerful classification tools for a variety of machine learning tasks.

Though his original concepts were conceived in the early 1960's, it took the advent of modern computers to allow his concepts to be thoroughly tested and implemented. In the early 1990's Vapnik was able to prove his prediction that SVMs provided a better classifier than neural networks, particularly for tasks involving images, such as handwriting recognition.

Thanks for reading!

Machine Learning Svm Python Data Science AI