

# Support Vector Machines

Ryan Henning

Galvanize

June 2, 2016

# Objectives

## Support Vector Machines (SVMs) Lecture

1. Gain an intuition about the *purpose* and *power* of SVMs.
2. Explore (some) of the mathematics behind SVMs.
3. Supercharge SVMs with kernels and soft margins.
4. Gain an intuition about the Bias-Variance tradeoff while using SVMs.

# Support Vector Machines

## A rough history

**Maximum Margin Classifier:** (morning lecture)

1963: Vapnik, Chervonenkis

**Soft Margins and the “Kernel Trick”:** (afternoon lecture)

1992-1995: Vapnik, Boser, Guyon, Cortes

This is the modern Support Vector Machine (SVM).

# Supervised Learning

**High level:** What is supervised learning?

Learn an unknown function from a set of **labeled** training data.

- ▶ Our training data is limited and finite. A useful algorithm must generalize well to “unseen” data.
- ▶ Example: Children learning colors.
- ▶ Support Vector Machines (SVMs) are a *supervised* learning algorithm.

# Notation

## Goal:

Learn a model of a function  $F : X \rightarrow Y$  from a training set  $D$ .

$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ , where

- ▶  $x^{(j)} \in X$  is often called the “input”.
- ▶  $y^{(j)} \in Y$  is often called the “label” or “target”.

## Notation (cont.)

$$F : X \rightarrow Y$$

- ▶ Often,  $X = \mathbb{R}^n$
- ▶ Often,  $Y$  is a finite set (i.e. a classification task)




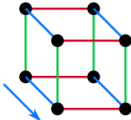
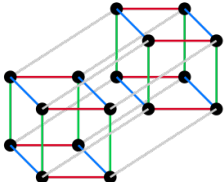

We want our learned model to *generalize* well.

Explain the concept of “generalization error”.

Generalization error is a measure of the model's performance on all possible “unseen” data.

# Dimensions

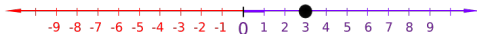
Basic stuff, I know.

					<div>X Y Z W </div>
0	1	2	3	4	#Dim

<sup>1</sup>By NerdBoy1392 (Own work) [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

# 1D

How do you split this space?



Split a line (1D) with a point (0D).

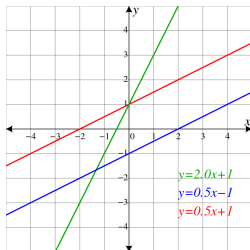
---

<sup>1</sup>By HakunamentaMathsIsFun at en.wikipedia [CC0], from Wikimedia Commons, Public Domain



## 2D

How do you split this space?



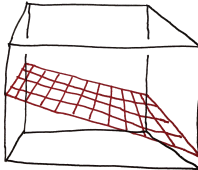
Split a plane (2D) with a line (1D).

---

<sup>1</sup>By ElectroKid (talk • contribs). Original: HiTe. (Modification from the original work.) [CC BY-SA 1.0 (<http://creativecommons.org/licenses/by-sa/1.0>)], via Wikimedia Commons

# 3D

How do you split this space?



Split space (3D) with a plane (2D).

## 4D, 5D, etc...

Hard to visualize... :/

In general, an  $n$ -dimensional space can be separated by an  $(n - 1)$ -dimensional *hyperplane*.

In an  $n$ -dimensional space any hyperplane can be defined by  $w \in \mathbb{R}^n$  and  $b \in \mathbb{R}$ . The hyperplane includes all  $x \in \mathbb{R}^n$  where:

$$w_0x_0 + w_1x_1 + \dots + w_{n-1}x_{n-1} - b = 0$$

usually written:

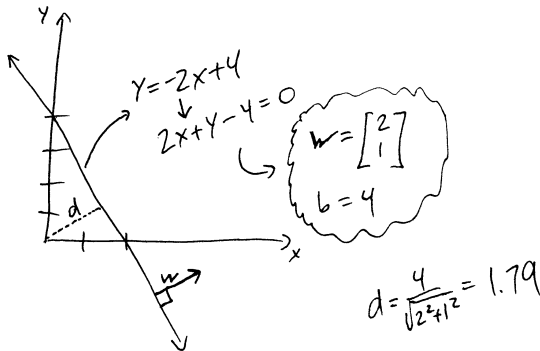
$$w \cdot x - b = 0$$

## How to interpret $w$ and $b$

So,  $w$  and  $b$  define a hyperplane. Is there an interpretation of  $w$  and  $b$  that can help us visualize this hyperplane?

- ▶  $\frac{w}{\|w\|}$  is the hyperplane's normal vector.
- ▶  $\frac{b}{\|w\|}$  is the hyperplane's distance from the origin.

## Example in 2D



# Binary Classification

## A supervised learning problem

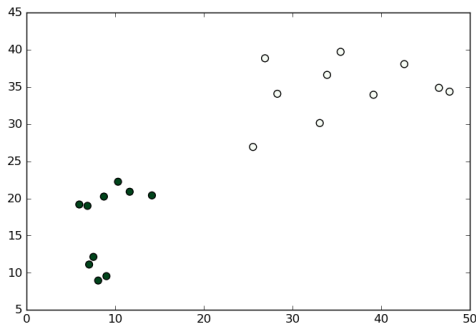
Recall, we're trying to learn  $F : X \rightarrow Y$ .

- ▶ Let,  $X = \mathbb{R}^n$
- ▶ For binary classification,  $Y = \{-1, 1\}$   
We're using -1 instead of 0 for future mathematical convenience.

Big idea: Let's have our model find a hyperplane that splits our  $n$ -dimensional data  $X$  into the set where  $y = -1$  and the set where  $y = 1$ .

# Binary Classification: Example

How many ways can we use a hyperplane to classify this dataset correctly?

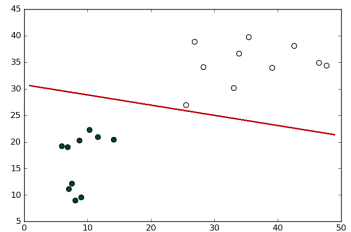
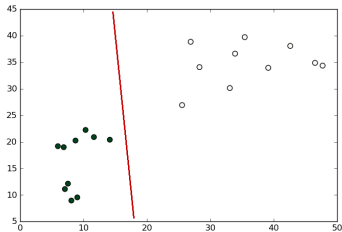


$$X = \mathbb{R}^2$$

$$Y = \{-1, 1\}$$

# Binary Classification: Example

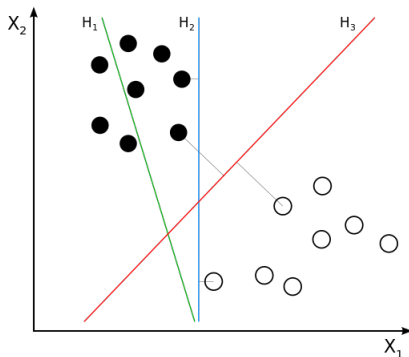
## Two Example Solutions





## Defining Margin

The distance from the hyperplane to the nearest training-data point.



<sup>1</sup>By User:ZackWeinberg, based on PNG version by User:Cyc [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>)], via Wikimedia Commons

# Why Maximize the Margin?

Our goal is to train a model that generalizes to “unseen” data.

Large margin means better generalization.

- ▶ Intuitively, this makes sense (see previous slide)
- ▶ As margin increases, VC-dimension decreases, meaning variance decreases

# Maximum Margin Classifier

## Goal

Goal: Calculate  $w$  and  $b$  of the hyperplane:

$$w \cdot x - b = 0$$

... such that the classes are split correctly and the margin is maximized.

First, some house cleaning: What happens to the hyperplane when we scale  $w$  and  $b$  by some factor  $c$ ?

# Maximum Margin Classifier

## Setup

We need to define a “canonical”  $w$  and  $b$ . This will help later.

Let

$$|w \cdot x^{(i)} - b| = 1$$

where  $x^{(i)}$  is the closest point to the hyperplane.

There will be a unique scaled  $w$  and  $b$  to achieve this.

# Maximum Margin Classifier

## Margin

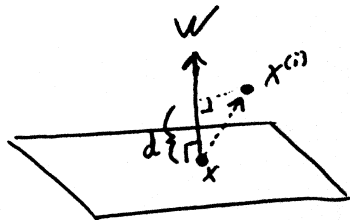
Now, if  $x^{(i)}$  is the closest point to the hyperplane, then the distance from  $x^{(i)}$  to the hyperplane is our margin. What is that distance?

$$\left| \frac{w}{\|w\|} \cdot (x^{(i)} - x) \right| = d$$

$$\frac{|w \cdot x^{(i)} - w \cdot x|}{\|w\|} = d$$

$$\frac{|w \cdot x^{(i)} - b - w \cdot x + b|}{\|w\|} = d$$

$$\frac{1}{\|w\|} = d = \text{margin}$$



# Maximum Margin Classifier

## First Attempt

Maximize  $\frac{1}{\|w\|}$

subject to:

$$|w \cdot x^{(i)} - b| \geq 1,$$

for all  $x^{(i)} \in D$

... but we don't know how to solve this optimization problem.  
Let's reformulate.

# Maximum Margin Classifier

## Reformulated

Minimize  $\frac{1}{2} ||w||^2$

subject to:

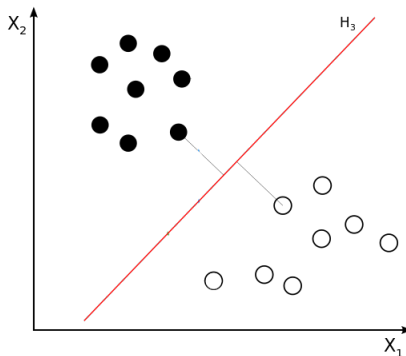
$$y^{(i)}(w \cdot x^{(i)} - b) \geq 1,$$

$$\text{for all } (y^{(i)}, x^{(i)}) \in D$$

... plus more steps... and we eventually get a quadratic programming formulation.

## Support Vectors

The maximum margin hyperplane is defined only by the points that touch the margin. These are called the “support vectors”.





# sklearn's interface

## LogisticRegression vs SVC

### LogisticRegression:

▶ [Link](#)

### SVC:

▶ [Link](#)

(end of morning lecture)

# Soft Margin Motivation

What if:

1. Your data isn't linearly separable?
2. Your data is noisy / has outliers?

Soft Margins address these problems.

# Soft Margin

## The $C$ hyperparameter

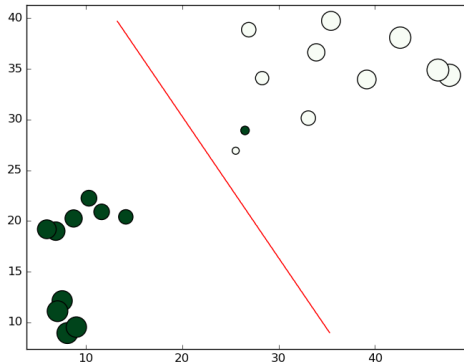
An extension to Maximum Margin Classifiers adds a  $C$  constant that gives the misclassification error penalty.

**Large  $C$ :** Harder margins: value classification accuracy over a large margin

**Small  $C$ :** Softer margins: value a large margin over classification accuracy

# Soft Margins

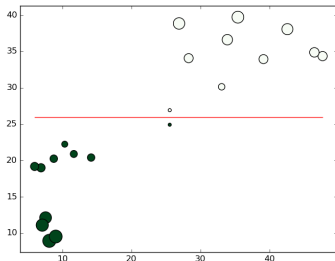
## Inseparable Data



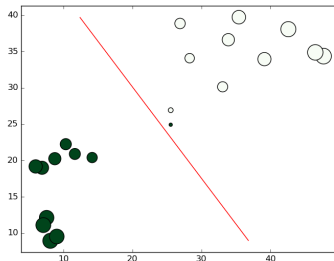
Only possible with a soft margin.

# Soft Margins

## Outliers in Data



Hard Margin



Soft Margin

# Soft Margins

scikit-learn code

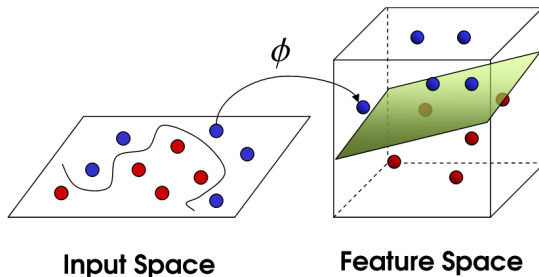
```
from sklearn.svm import SVC
...
svc = SVC(C=1.0, kernel='linear')
svc.fit(x, y)
```

SVC supports the  $C$  parameter as the soft-margin hyperparameter.

# The “Kernel Trick”

## The idea...

Idea: If data is inseparable in its input space, maybe it will be separable in a higher-dimensional space.



---

<sup>1</sup>Unknown source

# The “Kernel Trick”

## Back to the math...

In our optimization problem to maximize the margin, we eventually end up optimizing a vector alpha  $\alpha^{(i)}, i \in [1, m]$  in the following equation:

$$\mathcal{L}(\alpha) = \sum_{i=1}^m \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})$$

$$\mathcal{L}(\alpha) = \sum_{i=1}^m \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} (\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)}))$$



# The “Kernel Trick”

Creating a kernel...

$$\phi(x^{(i)}) \cdot \phi(x^{(j)}) \in \mathbb{R}$$

... this is just a real number.

What if we never applied  $\phi$  and we never took the dot product, but we instead replaced this whole thing with a “kernel function”.

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)}) \in \mathbb{R}$$

# The “Kernel Trick”

Why is this so cool?

- ▶ Saves some computation. We never need to compute  $\phi$ .
- ▶ Opens new possibilities. A kernel can operate in infinite dimensions!

You can use any  $K(x^{(i)}, x^{(j)})$  as long as there **exists** some  $\phi$  such that

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)})$$

... but you don't have to know what  $\phi$  actually **is**!

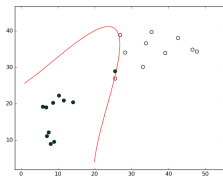
# The Polynomial Kernel

$$K(x^{(i)}, x^{(j)}) = (1 + x^{(i)} \cdot x^{(j)})^d$$

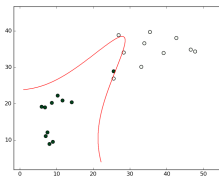
- ▶ equivalent to the dot product in the  $d$ -order  $\phi$  space
- ▶ requires an extra hyper-parameter,  $d$ , for “degree”

# The Polynomial Kernel

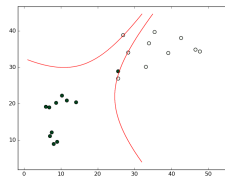
## Example



SVC =  
SVC(C=10000.0,  
kernel='poly',  
degree=3)



SVC =  
SVC(C=10000.0,  
kernel='poly',  
degree=5)



SVC =  
SVC(C=10000.0,  
kernel='poly',  
degree=10)

# The RBF Kernel

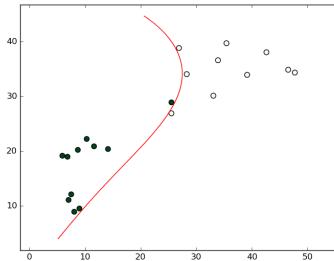
(Radial Basis Function)

$$K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2)$$

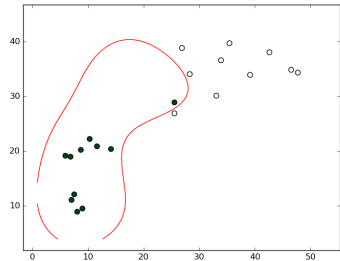
- ▶ equivalent to the dot product in the Hilbert space of infinite dimensions
- ▶ requires an extra hyper-parameter,  $\gamma$ , “gamma”

# The RBF Kernel

## Examples



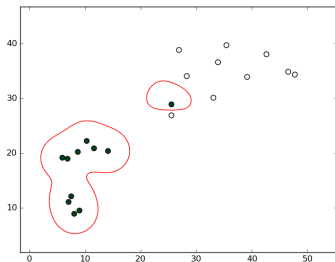
```
svc = SVC(C=10000.0,  
kernel='rbf', gamma=0.001)
```



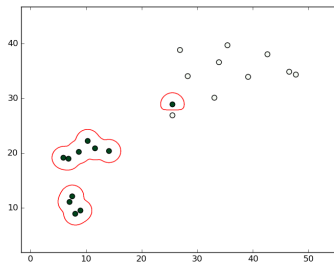
```
svc = SVC(C=10000.0,  
kernel='rbf', gamma=0.01)
```

# The RBF Kernel

## More Examples



```
svc = SVC(C=10000.0,  
kernel='rbf', gamma=0.1)
```



```
svc = SVC(C=10000.0,  
kernel='rbf', gamma=1.0)
```

# Bias-Variance tradeoff

## Explanation

### Bias

A high-“bias” model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

For complex data, high-bias models often *underfit* the data.

### Variance

A high-“variance” model makes fewer assumptions and has more representational power.

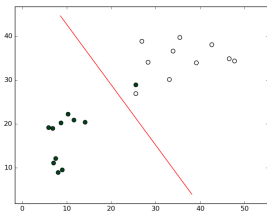
E.g. An RBF SVM looks for dividing hyperplanes in an infinite-dimensional space.

For simple data, high-variance models often *overfit* the data.

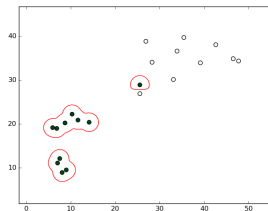


# Bias-Variance tradeoff

## Example



`svc = SVC(C=0.01,  
kernel="linear")`



`svc = SVC(C=10000.0,  
kernel='rbf', gamma=1.0)`

Which is a better fit for this dataset?

# SVMs vs Logistic Regression

(some rules of thumb)

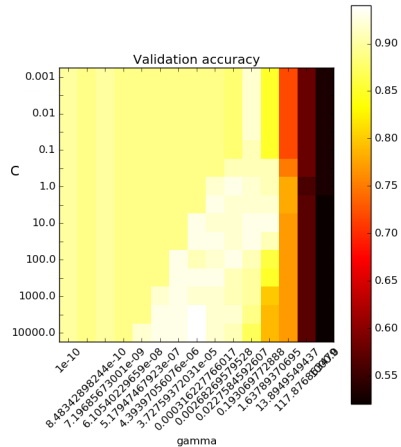
1. Logistic Regression maximizes the *Binomial Log Likelihood* function.
2. SVMs maximize the *margin*.
3. When classes are nearly separable, SVMs tends to do better than Logistic Regression.
4. Otherwise, Logistic Regression (with Ridge) and SVMs are similar.
5. However, if you want to estimate probabilities, Logistic Regression is the better choice.
6. With kernels, SVMs work well. Logistic Regression works fine with kernels but can get computationally too expensive.

# Grid Search

## Hyperparameter Tuning

Let's find  $C$  and  $\gamma$  by searching through values we expect might work well.

Use cross-validation accuracy to determine which values are best.



# Grid Search

## code

```
svc_rbf = SVC(kernel='rbf')

param_space = {'C':      np.logspace(-3, 4, 15),
               'gamma': np.logspace(-10, 3, 15)}

grid_search = GridSearchCV(svc_rbf, param_space,
                           scoring='accuracy', cv=10)

grid_search.fit(x, y)

print grid_search.grid_scores_
print grid_search.best_params_
print grid_search.best_score_
print grid_search.best_estimator_
```