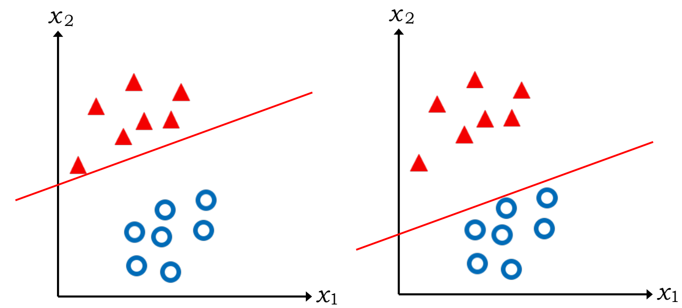
Tufts

Class #10: Kernel Functions and Support Vector Machines (SVMs)

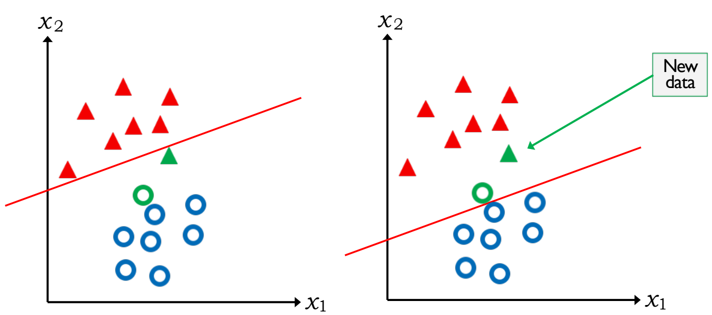
Machine Learning (COMP 135): M. Allen, 07 Oct. 19

Data Separation



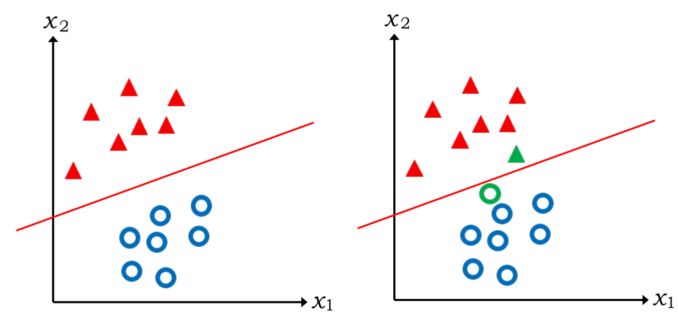
- Linear classification with a perceptron or logistic function look for a dividing line in the data (or a plane, or other linearly defined structure)
  - Often ***multiple*** lines are possible
  - Essentially, the algorithms are ***indifferent***: they don't care which line we pick
  - In the example seen here, either classification line separates data perfectly well

"Fragile" Separation



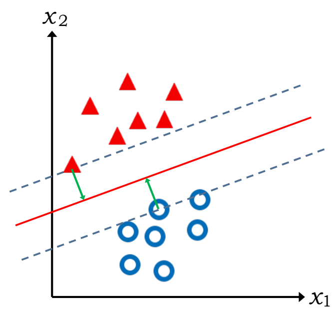
- As more data comes in, these classifiers may start to fail
  - A separator that is too close to one cluster or the other now makes mistakes
  - May happen even if new data follows same distribution seen in the training set

"Robust" Separation



- What we want is a large margin separator: a separation that has the largest distance possible from each part of our data-set
- This will often give much better performance when used on new data

Large Margin Separation



This is sometimes called the "widest road" approach

A support vector machine (SVM) is a technique that finds this road.

The points that define the edges of the road are the support vectors.

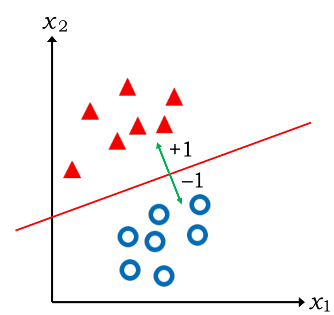
- A new learning problem: ***find*** the separator with the largest margin
- This will be measured from the data points, on opposite sides, that are ***closest together***

Linear Classifiers and SVMs

Linear	
Weight equation	$\mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
Threshold function	$h_{\mathbf{w}} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$

SVM	
Weight equation	$\mathbf{w} \cdot \mathbf{x} + b = (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) + b$
Threshold function	$h_{\mathbf{w}} = \begin{cases} +1 & \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \mathbf{w} \cdot \mathbf{x} < 0 \end{cases}$

Large Margin Separation



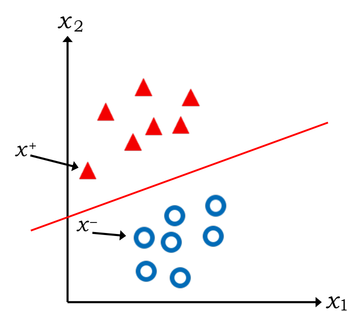
$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

A key difference: the SVM is going to do this without learning and remembering weight vector $\mathbf{w}$.

Instead, it will use features of the ***data-items themselves***.

- Like a linear classifier, the SVM separates at the line where its learned vector of weights is zero

Mathematics of SVMs



$$\mathbf{w} \cdot \mathbf{x}^+ + b = +1$$
$$\mathbf{w} \cdot \mathbf{x}^- + b = -1$$

$$\mathbf{w} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = 2$$
$$\frac{\mathbf{w}}{||\mathbf{w}||} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = \frac{2}{||\mathbf{w}||}$$

$$||\mathbf{w}|| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$$

- It turns out that the weight-vector $\mathbf{w}$ for the largest margin separator has some important properties relative to the closest data-points on each side (x^+ and x^-)

Mathematics of SVMs

- Through the magic of mathematics (Lagrangian multipliers, to be specific), we can derive a quadratic programming problem

1. We start with our data-set:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\} \qquad [\forall i,\ y_i \in \{+1, -1\}]$$

2. We then solve the constrained optimization problem:

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i\, \alpha_j\, y_i\, y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\forall i,\ \alpha_i \geq 0$$

$$\sum_i \alpha_i\, y_i = 0$$

The goal: based on ***known*** values $(\mathbf{x}_i, y_i)$ ***find*** the values we ***don't know*** (α_i) that:

1. Will maximize value $W(\alpha_i)$
2. Satisfy the two numerical constraints

Mathematics of SVMs

- The details of how all this is done are a bit complicated, but a constrained optimization problem like this can be algorithmically solved to get all of the α_i values needed:

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i\, \alpha_j\, y_i\, y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\forall i,\ \alpha_i \geq 0$$

$$\sum_i \alpha_i\, y_i = 0$$

- Once done, we can find the weight-vector and bias term if we want:

$$\mathbf{w} = \sum_i \alpha_i\, y_i\, \mathbf{x}_i \qquad b = -\frac{1}{2}\Big(\max_{i \,|\, y_i = -1} \mathbf{w} \cdot \mathbf{x}_i + \min_{j \,|\, y_j = +1} \mathbf{w} \cdot \mathbf{x}_j \Big)$$

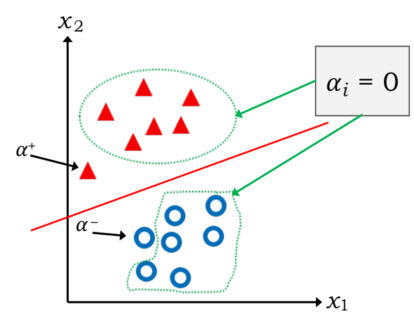
The Dual Formulation

- It turns out that we don't need to use the weights at all
- Instead, we can simply use the α_i values ***directly***:

$$\mathbf{w} \cdot \mathbf{x}_i + b = \sum_j \alpha_j\, y_j\, (\mathbf{x}_i \cdot \mathbf{x}_j) - b$$

- Now, if we had to sum over every data-point like we do on the right-hand side of this equation, this would look very bad for a large data-set
- It turns out that these α_i values have a special property, however, that makes it feasible to use them as part of our classification function…

Sparseness of SVMs



This means that when we do the classification calculation:

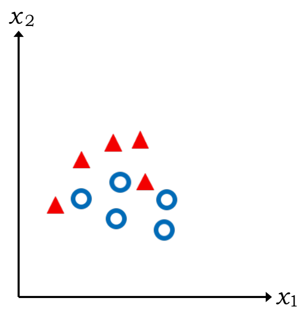
$$\sum_j \alpha_j\, y_j\, (\mathbf{x}_i \cdot \mathbf{x}_j) - b$$

We only have to sum over points $\mathbf{x}_j$ that are in the set of support vectors, ***ignoring*** all others.

Thus, an SVM need only remember and use the values for the few support vectors, not those for all the rest of the data.

- The α_i values are 0 ***everywhere except*** at the support vectors (the points closest to the separator)

Another Nice Trick

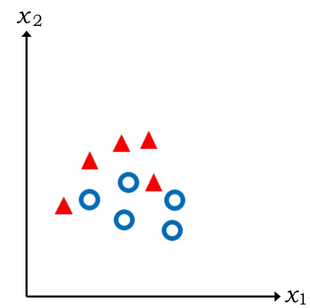


$$\sum_j \alpha_j \, y_j \, (\mathbf{x}_i \cdot \mathbf{x}_j) - b$$

Using a kernel "trick", we can find a function that ***transforms*** the data into another form, where it is actually possible to separate it in a linear manner.

- The calculation uses dot-products of data-points with each other (instead of with weights)
- This will allow us to deal with data that is not linearly separable

Transforming Non-Separable Data



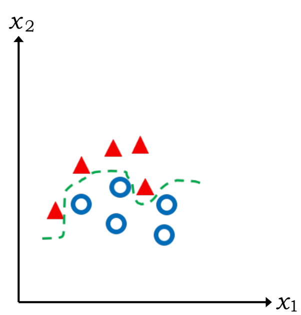
A transformation function:

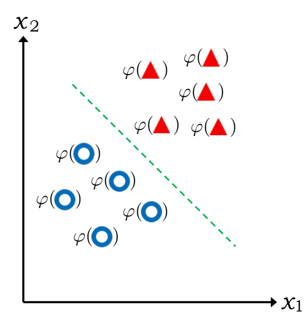
$\varphi(\mathbf{x})$ $\quad \varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$

maps data-vectors to new vectors, of either the same dimensionality ($m = n$) or a different one ($m \neq n$)

- If data that is not linearly separable, we can transform it
  - We ***change*** features used to represent our data
  - Really, we ***don't care*** what the data feature are, so long as we can get classification to work

Transforming Non-Separable Data



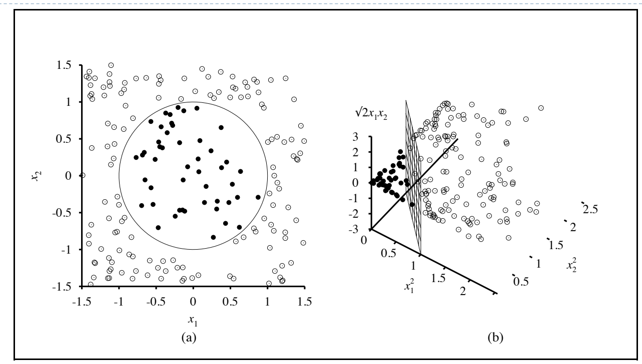


$\varphi(\mathbf{x})$ $\quad \varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$

$$\sum_j \alpha_j \, y_j \, (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)) - b$$

The "Kernel Trick"

Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)



$$\varphi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

Simplifying the Transformation Function

- We can derive a simpler (2-dimensional) equation, ***equivalent to*** the cross-product needed when doing SVM computations in the transformed (3-dimensional) space:

$$
\begin{aligned}
\varphi(\mathbf{x}) \cdot \varphi(\mathbf{z}) &= (x_1^2,\, x_2^2,\, \sqrt{2}x_1x_2) \cdot (z_1^2,\, z_2^2,\, \sqrt{2}z_1z_2) \longleftarrow \text{Needed} \\
&= x_1^2z_1^2 + x_2^2z_2^2 + \sqrt{2}x_1x_2\sqrt{2}z_1z_2 \\
&= \boxed{x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2} \longleftarrow \text{10 multiplications, 2 additions} \\
&= \boxed{(x_1z_1 + x_2z_2)^2} \longleftarrow \text{3 multiplications, 1 addition} \\
&= (\mathbf{x} \cdot \mathbf{z})^2 \longleftarrow \text{Used instead}
\end{aligned}
$$

The Kernel Function

$$k(\mathbf{x},\, \mathbf{z}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$$

- This final function (right side) is what the SVM will actually use to compute dot-products in its equations
- This is called the kernel function
- To make SVMs really useful we look for a kernel that:

1. Separates the data usefully
2. Is relatively efficient to calculate

This Week

- **Today**: Kernels and SVMs
- **Readings**: Linked from class website schedule page.
- **Homework 03**: due Wednesday, 16 October, 9:00 AM
- **Office Hours**: 237 Halligan, Tuesday, 11:00 AM – 1:00 PM
  - TA hours can be found on class website as well