Kernels

Transforming Data Features

- So far we are using features as they are, $X = (x_1, x_2, \ldots, x_n)$

- For example, a training sample $x = (3, -2, 4)$, we are feeding values 3, -2 and 4 to the model.

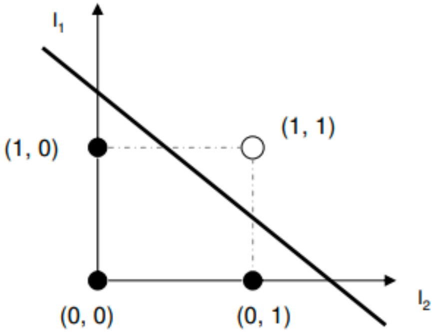
- Can we transform and create new features? $x_1 x_2$?

VIRGINIA TECH

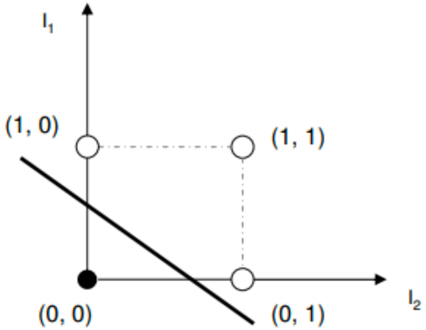
Why Transforming Features

- To obtain another perspective of data (XOR)

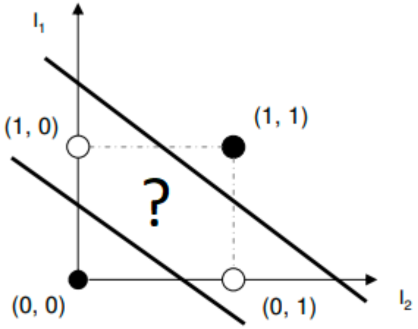
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



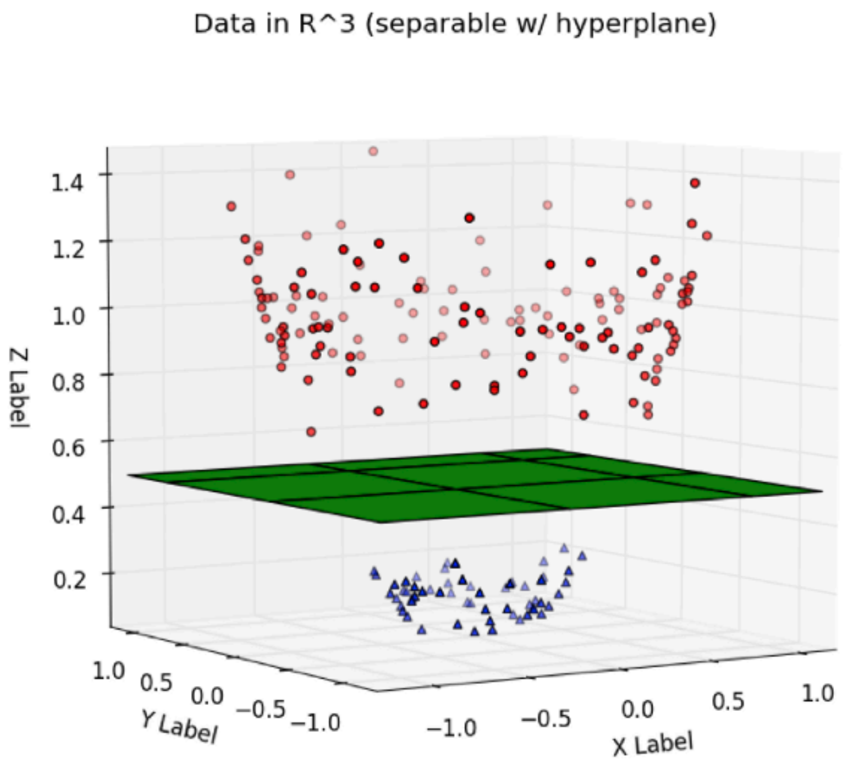
Add $x_3 = (x_1 - x_2)^2$

X1	X2	X3	Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Let $y = x_3$

- Also mapping data to higher dimension

Mapping to higher dimensional space



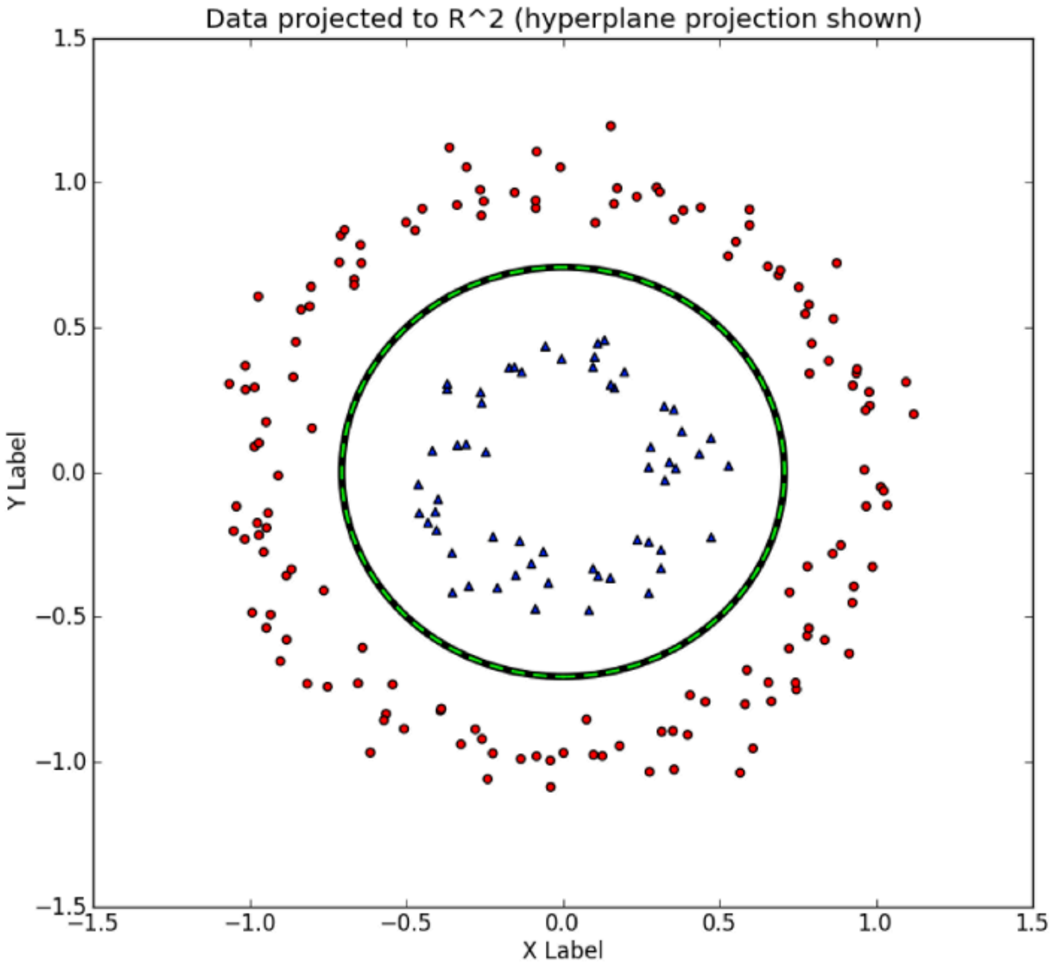
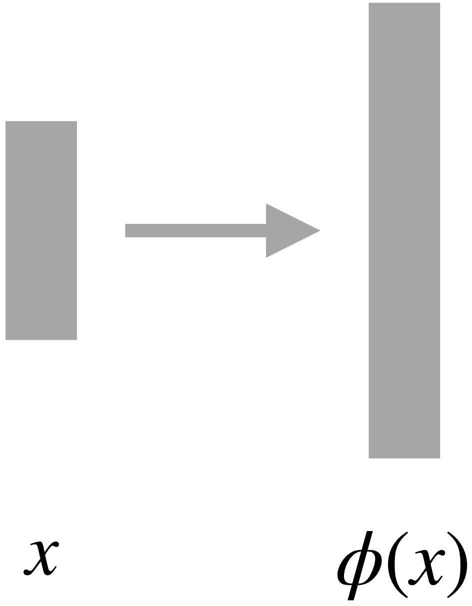


Figure credit: [link]

Create New Features

We can transform and create features.

Challenge: Feature space grows rapidly.



$$\phi(x) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ x_1 x_2 \\ x_2 x_3 \\ \vdots \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_n \end{bmatrix}$$

Higher Order Polynomials

- Assume n dimensions, and d degree of polynomial
- Number of terms $\binom{n+d}{d} = \dfrac{(n+d)!}{n!d!}$
- Rapid growth
  - m=100, d=6
  - ~1.6 billion terms
- *Proof

VIRGINIA TECH

Feature Mapping

- Pros: turn non-linearly separable classification into linear one

- Cons: feature explosion
  - Computationally expensive
  - Require more training examples to avoid overfitting

Kernel Methods

- Goal: capture non-linear patterns

- Mapping data to higher dimensions without explicitly computing the mapping.

    - How?

Kernel Trick

- Rewrite learning algortihms so they only depend on the **dot product between two samples**

- Replace dot product $\phi(\mathbf{x})\phi(\mathbf{z})$ by kernel function $k(\mathbf{x}, \mathbf{z})$,
- $k(\cdot)$ computed the dot product implicitly.

Kernel Function Example

- Consider two samples $\mathbf{x} = \{x_1, x_2\}$ and $\mathbf{z} = \{z_1, z_2\}$
- Assume we have a kernel
  - $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$
  - $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})\phi(\mathbf{z})$, what is the form of $\phi(\mathbf{x})$

Kernel Function Example

- Consider two samples $\mathbf{x} = \{x_1, x_2\}$ and $\mathbf{z} = \{z_1, z_2\}$
- Assume we have a kernel

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x} \cdot \mathbf{z})^2 \\
&= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2 \, . \, x_2^2) \cdot (z_1^2, \sqrt{2} z_1 z_2, z_2^2)
\end{aligned}
$$

$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})\phi(\mathbf{z})$, what is the form of $\phi(\mathbf{x})$

- $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2 \, . \, x_2^2)$

Compute k instead of ϕ

Kernelize Learning Algorithms

- Using $\phi(\mathbf{x})$ is straightfoward, how to use $k(\mathbf{x}, \mathbf{z})$?

- Algorithm with $\phi(\mathbf{x})$
  - Assume $\mathbf{x} \in \mathbb{R}^n$, and $\phi(\mathbf{x}) \in \mathbb{R}^m$, $n < m$
  - Learn weight parameters $\mathbf{w} \in \mathbb{R}^m$
  - Predict y with $\mathbf{w} \cdot \phi(\mathbf{x})$

Recall a Normal Perceptron

- Let $y \in \{-1,1\}\ \forall y$
- Initialize weights $\mathbf{w}, b$

We take b out here

- Repeat until satisfied
  - For each training sample $(\mathbf{x}^l, y^l)$
    - If $y^l(\mathbf{w} \cdot \mathbf{x}^l + b) < 0$, update $\begin{cases} \mathbf{w} \leftarrow \mathbf{w} + y^l \mathbf{x}^l \\ b \leftarrow b + y^l \end{cases}$

Kernelize the Perceptron

- Naïve approach, replace $\mathbf{x}$ with $\phi(\mathbf{x})$
- Let $y \in \{-1,1\}\ \forall y$
- Initialize weights $\mathbf{w}, b$

- Repeat until satisfied
  - For each training sample $(\mathbf{x}^l, y^l)$
    - If $y^l(\mathbf{w} \cdot \phi(\mathbf{x}^l) + b) < 0$, update $\begin{cases} \mathbf{w} & \leftarrow \mathbf{w} + y^l \phi(\mathbf{x}^l) \\ b & \leftarrow b + y^l \end{cases}$

Rewrite $\mathbf{w}$

- Naïve approach, replace $\mathbf{x}$ with $\phi(\mathbf{x})$
- Let $y \in \{-1,1\}\,\forall y$
- Initialize weights $\mathbf{w}, b$

- Repeat until satisfied
  - For each training sample $(\mathbf{x}^l, y^l)$, assume L samples in total
    - If $y^l(\mathbf{w}\cdot\phi(\mathbf{x}^l)+b) < 0$, update $\begin{cases} \mathbf{w} \leftarrow \mathbf{w} + y^l\phi(\mathbf{x}^l) \\ b \leftarrow b + y^l \end{cases}$
- Rewrite $\mathbf{w}$
  - $\mathbf{w} = \sum_{j=1}^{L} \alpha_j y^j \phi(\mathbf{x}^j)$ where α_j is the number of misclassifications for j(th) sample
- To make prediction on a new sample x^{new}
  - $\mathbf{w}\cdot\phi(\mathbf{x}^{new}) + b = \sum_{j=1}^{L} \alpha_j y^j \phi(\mathbf{x}^j)\cdot\phi(\mathbf{x}^{new}) + b = \sum_{j=1}^{L} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^{new}) + b$

Rewrite Condition

- Naïve approach, replace $\mathbf{x}$ with $\phi(\mathbf{x})$
- Let $y \in \{-1,1\}\ \forall y$
- Initialize weights $\mathbf{w}, b$

- Repeat until satisfied
  - For each training sample $(\mathbf{x}^l, y^l)$, assume L samples in total
    - If $y^l(\mathbf{w} \cdot \phi(\mathbf{x}^l) + b) < 0$, update $\begin{cases} \mathbf{w} \leftarrow \mathbf{w} + y^l \phi(\mathbf{x}^l) \\ b \leftarrow b + y^l \end{cases}$

- $\mathbf{w} = \sum_{j=1}^{L} \alpha_j y^j \phi(\mathbf{x}^j)$ where α_j is the number of misclassifications for j(th) sample
- If $y^l(\sum_{j=1}^{L} \alpha_j y^j \phi(\mathbf{x}^j) \cdot \phi(\mathbf{x}^l) + b) = y^l(\sum_{j=1}^{L} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^l) + b) < 0$, update $\begin{cases} \alpha^l \leftarrow \alpha^l + 1 \\ b \leftarrow b + y^l \end{cases}$

Kernelized Perceptron Algorithm

- Let $y \in \{-1,1\}\ \forall y$
- Initialize $\alpha_1 = \alpha_2 = \ldots = \alpha_L = 0,\ b = 0$, assume there are L samples
- Repeat until satisfied
  - For each training sample $(\mathbf{x}^l, y^l)$
    - If $y^l(\sum_{j=1}^{L} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^l) + b) < 0$, update $\begin{cases} \alpha^l \leftarrow \alpha^l + 1 \\ b \leftarrow b + y^l \end{cases}$
- No more $\phi(\cdot)$ 👍

Primal and Dual Forms

Primal form

- Let $y \in \{-1,1\} \forall y$
- Initialize weights $\mathbf{w}, b$
- For each training sample $(\mathbf{x}^l, y^l)$
  - If $y^l(\mathbf{w} \cdot \phi(\mathbf{x}^l) + b) < 0$,
  - $\begin{cases} \mathbf{w} & \leftarrow \mathbf{w} + y^l \phi(\mathbf{x}^l) \\ b & \leftarrow b + y^l \end{cases}$

Dual form

- Let $y \in \{-1,1\} \forall y$
- Initialize $\alpha_1 = \alpha_2 = \ldots = \alpha_L = 0,\ b = 0,$ assume there are L samples
- For each training sample $(\mathbf{x}^l, y^l)$
  - If $y^l(\sum_{j=1}^{L} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^l) + b) < 0,$
  - $\begin{cases} \alpha^l \leftarrow \alpha^l + 1 \\ b \leftarrow b + y^l \end{cases}$

Popular Kernels

- Polynomial Kernel of degree exactly d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomial Kernel of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian Kernel

$$K(\mathbf{u}, \mathbf{v}) = e^{-\frac{\|\mathbf{u} - \mathbf{v}\|^2}{2\sigma^2}}$$

Among many others

Design a Kernel

- Not any function can be kernel
- For some kernel definitions, there is no corresponding $\phi(\cdot)$
- Extend to Kernel upon strings, trees, or graphs
- Explore more: https://doi.org/10.7551/mitpress/4170.001.0001

VIRGINIA TECH

More than Kernel Perceptron

- Other algorithms can be Kernelized
  - Logistic Regression

- Do Kernel methods address
  - Expensive computation, how?
  - Overfitting