Nonlinearity & Preprocessing
• Concatenated (combined) features
• XOR: \( x = (x_1, x_2, x_1x_2) \)
• income: add “degree + major”

- Perceptron
  - Map data into feature space \( x \rightarrow \phi(x) \)
  - Solution in span of \( \phi(x_i) \)
• Separating surfaces are Circles, hyperbolae, parabolae
Kernels as dot products

Problem
- Extracting features can sometimes be very costly.
- Example: second order features in 1000 dimensions. This leads to $5 \cdot 10^5$ numbers. For higher order polynomial features much worse.

Solution
Don’t compute the features, try to compute dot products implicitly. For some features this works . . .

Definition
A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function in its arguments for which the following property holds

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

for some feature map $\Phi$.

If $k(x, x')$ is much cheaper to compute than $\Phi(x)$ . . .
**Quadratic Kernel**

**Quadratic Features in $\mathbb{R}^2$**

$$\Phi(x) := \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

**Dot Product**

$$\langle \Phi(x), \Phi(x') \rangle = \left\langle \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right), \left(x_1'^2, \sqrt{2}x_1'x_2', x_2'^2\right) \right\rangle$$

$$= \langle x, x' \rangle^2 = k(x, x')$$

**Insight**

Trick works for any polynomials of order $d$ via $\langle x, x' \rangle^d$. 
The Perceptron on features

initialize $w, b = 0$
repeat
    Pick $(x_i, y_i)$ from data
    if $y_i(w \cdot \Phi(x_i) + b) \leq 0$ then
        $w' = w + y_i\Phi(x_i)$
        $b' = b + y_i$
    until $y_i(w \cdot \Phi(x_i) + b) > 0$ for all $i$

• Nothing happens if classified correctly
• Weight vector is linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
• Classifier is (implicitly) a linear combination of inner products $f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$
Kernelized Perceptron

initialize $f = 0$

repeat
  Pick $(x_i, y_i)$ from data
  if $y_i f(x_i) \leq 0$ then
    $f(\cdot) \leftarrow f(\cdot) + y_i k(x_i, \cdot) + y_i$
    $\alpha_i \leftarrow \alpha_i + y_i$
  increase its vote by 1
until $y_i f(x_i) > 0$ for all $i$

• instead of updating $w$, now update $\alpha_i$
• Weight vector is linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
• Classifier is linear combination of inner products
  
  $$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$
Kernelized Perceptron

**Primal Form**
- update weights
  \[ w \leftarrow w + y_i \phi(x_i) \]
- classify
  \[ f(k) = w \cdot \phi(x) \]

**Dual Form**
- update linear coefficients
  \[ \alpha_i \leftarrow \alpha_i + y_i \]

Implicitly equivalent to:
\[ w = \sum_{i \in I} \alpha_i \phi(x_i) \]

- Nothing happens if classified correctly
- Weight vector is linear combination
- Classifier is linear combination of inner products
\[ f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x) \]
Kernelized Perceptron

**Primal Form**
- **update weights**
  
  \[ w \leftarrow w + y_i \phi(x_i) \]
- **classify**
  
  \[ f(k) = w \cdot \phi(x) \]

**Dual Form**
- **update linear coefficients**
  
  \[ \alpha_i \leftarrow \alpha_i + y_i \]
- **classify**
  
  \[ w = \sum_{i \in I} \alpha_i \phi(x_i) \]

\[ f(x) = w \cdot \phi(x) = [\sum_{i \in I} \alpha_i \phi(x_i)] \phi(x) \]

**Slow** \( O(d^2) \)

\[ = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle \]

**Fast** \( O(d) \)

\[ = \sum_{i \in I} \alpha_i k(x_i, x) \]
Kernelized Perceptron

initialize $\alpha_i = 0$ for all $i$

repeat

Pick $(x_i, y_i)$ from data

if $y_i f(x_i) \leq 0$ then

$\alpha_i \leftarrow \alpha_i + y_i$

until $y_i f(x_i) > 0$ for all $i$

Dual Form

update linear coefficients

$\alpha_i \leftarrow \alpha_i + y_i$

implicitly

$w = \sum_{i \in I} \alpha_i \phi(x_i)$

classify

slow $O(d^2)$

$f(x) = w \cdot \phi(x) = \left[ \sum_{i \in I} \alpha_i \phi(x_i) \right] \phi(x)$

slow $O(d^2)$

$= \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$

fast $O(d)$

slow $O(d^2)$

$= \sum_{i \in I} \alpha_i k(x_i, x)$

if #features $>>$ #examples, dual is easier; otherwise primal is easier
Kernelized Perceptron

**Primal Perceptron**
- update weights
  \[ w \leftarrow w + y_i \phi(x_i) \]
- classify
  \[ f(k) = w \cdot \phi(x) \]

**Dual Perceptron**
- update linear coefficients
  \[ \alpha_i \leftarrow \alpha_i + y_i \]
  implicitly
  \[ w = \sum_{i \in I} \alpha_i \phi(x_i) \]

Q: when is #features >> #examples?
A: higher-order polynomial kernels or exponential kernels (inf. dim.)
Pros/Cons of Kernel in Dual

- **pros:**
  - no need to compute \( \phi(x) \) (time)
  - no need to store \( \phi(x) \) and \( w \) (memory)

- **cons:**
  - sum over all misclassified training examples for test
  - need to store all misclassified training examples (memory)
  - called “support vector set”
  - SVM will minimize this set!

Dual Perceptron

**update linear coefficients**

\[
\alpha_i \leftarrow \alpha_i + y_i
\]

implicitly

\[
w = \sum_{i \in I} \alpha_i \phi(x_i)
\]

classify

\[
f(x) = w \cdot \phi(x) = \left[ \sum_{i \in I} \alpha_i \phi(x_i) \right] \phi(x)
\]

\[
= \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle
\]

slow \( O(d^2) \)

\[
= \sum_{i \in I} \alpha_i k(x_i, x)
\]

fast \( O(d) \)
## Kernelized Perceptron

### Primal Perceptron

<table>
<thead>
<tr>
<th>update on</th>
<th>new param.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$: -1</td>
<td>$w = (0, -1)$</td>
</tr>
<tr>
<td>$x_2$: +1</td>
<td>$w = (2, 0)$</td>
</tr>
<tr>
<td>$x_3$: +1</td>
<td>$w = (2, -1)$</td>
</tr>
</tbody>
</table>

### Dual Perceptron

<table>
<thead>
<tr>
<th>update on</th>
<th>new param.</th>
<th>$W$ (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$: -1</td>
<td>$\alpha = (-1, 0, 0)$</td>
<td>$-x_1$</td>
</tr>
<tr>
<td>$x_2$: +1</td>
<td>$\alpha = (-1, 1, 0)$</td>
<td>$-x_1 + x_2$</td>
</tr>
<tr>
<td>$x_3$: +1</td>
<td>$\alpha = (-1, 1, 1)$</td>
<td>$-x_1 + x_2 + x_3$</td>
</tr>
</tbody>
</table>

**linear kernel (identity map)**

**final implicit** $w = (2, -1)$

**geometric interpretation of dual classification:**

sum of dot-products with $x_2$ & $x_3$ bigger than dot-product with $x_1$

(agreement w/ positive > w/ negative)
XOR Example

Dual Perceptron

update on | new param. | $W$ (implicit)
--- | --- | ---
$x_1$: +1 | $\alpha = (+1, 0, 0, 0)$ | $\phi(x_1)$
$x_2$: -1 | $\alpha = (+1, -1, 0, 0)$ | $\phi(x_1) - \phi(x_2)$

$k(x, x') = (x \cdot x')^2 \iff \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

$w = (0, 0, 2\sqrt{2})$

classification rule in dual/geom:

$$(x \cdot x_1)^2 > (x \cdot x_2)^2$$

$\Rightarrow \cos^2 \theta_1 > \cos^2 \theta_2$

$\Rightarrow |\cos \theta_1| > |\cos \theta_2|$

in dual/algebra:

$$(x \cdot x_1)^2 > (x \cdot x_2)^2$$

$\Rightarrow (x_1 + x_2)^2 > (x_1 - x_2)^2$

$\Rightarrow x_1x_2 > 0$

also verify in primal
Circle Example??

Dual Perceptron

<table>
<thead>
<tr>
<th>update on</th>
<th>new param.</th>
<th>$W$ (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$: +1</td>
<td>$\alpha = (+1, 0, 0, 0)$</td>
<td>$\phi(x_1)$</td>
</tr>
<tr>
<td>$x_2$: -1</td>
<td>$\alpha = (+1, -1, 0, 0)$</td>
<td>$\phi(x_1) - \phi(x_2)$</td>
</tr>
</tbody>
</table>

$k(x, x') = (x \cdot x')^2 \iff \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
Idea

We want to extend \( k(x, x') = \langle x, x' \rangle^2 \) to

\[
k(x, x') = (\langle x, x' \rangle + c)^d \quad \text{where } c > 0 \text{ and } d \in \mathbb{N}.
\]

Prove that such a kernel corresponds to a dot product.

Proof strategy

Simple and straightforward: compute the explicit sum given by the kernel, i.e.

\[
k(x, x') = (\langle x, x' \rangle + c)^d = \sum_{i=0}^{m} \binom{d}{i} (\langle x, x' \rangle)^i c^{d-i}
\]

Individual terms \((\langle x, x' \rangle)^i\) are dot products for some \(\Phi_i(x)\).

+c is just augmenting space.

Simpler proof: set \(x_0 = \sqrt{c}\)
**Circle Example**

**Dual Perceptron**

<table>
<thead>
<tr>
<th>update on</th>
<th>new param.</th>
<th>$W$ (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$: +1</td>
<td>$\alpha = (+1, 0, 0, 0, 0)$</td>
<td>$\phi(x_1)$</td>
</tr>
<tr>
<td>$x_2$: -1</td>
<td>$\alpha = (+1, -1, 0, 0, 0)$</td>
<td>$\phi(x_1) - \phi(x_2)$</td>
</tr>
<tr>
<td>$x_3$: -1</td>
<td>$\alpha = (+1, -1, -1, 0, 0)$</td>
<td></td>
</tr>
</tbody>
</table>

$$k(x, x') = (x \cdot x')^2 \iff \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$k(x, x') = (x \cdot x' + 1)^2 \iff \phi(x) = ?$$
### Examples of kernels $k(x, x')$

<table>
<thead>
<tr>
<th>Kernel Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$\langle x, x' \rangle$</td>
</tr>
<tr>
<td>Laplacian RBF</td>
<td>$\exp \left( -\lambda |x - x'| \right)$</td>
</tr>
<tr>
<td>Gaussian RBF</td>
<td>$\exp \left( -\lambda |x - x'|^2 \right)$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$(\langle x, x' \rangle + c)^d$, $c \geq 0$, $d \in \mathbb{N}$</td>
</tr>
<tr>
<td>B-Spline</td>
<td>$B_{2n+1}(x - x')$</td>
</tr>
<tr>
<td>Cond. Expectation</td>
<td>$E_c[p(x</td>
</tr>
</tbody>
</table>

- **Linear**
- **Laplacian RBF** distorts distance
- **Gaussian RBF**
- **Polynomial**: you only need to know polynomial and gaussian.
- **B-Spline** distorts angle
- **Cond. Expectation**
Kernel Summary

- For a feature map $\phi$, find a magic function $k$, s.t.:
  - the dot-product $\phi(x) \cdot \phi(x') = k(x, x')$
  - this $k(x, x')$ should be much faster than $\phi(x)$
  - $k(x, x')$ should be computable in $O(n)$ if $x$ in $\mathbb{R}^n$
  - $\phi(x)$ is much slower: $O(n^d)$ for poly $d$, more for Gaussian
- But for any $k$ function, is there a $\phi$ s.t. $\phi(x) \cdot \phi(x') = k(x,x')$?

Examples of kernels $k(x, x')$

- **Linear**
  $$\langle x, x' \rangle$$
- **Laplacian RBF**
  $$\exp \left( -\lambda \| x - x' \| \right)$$
- **Gaussian RBF**
  $$\exp \left( -\lambda \| x - x' \|^2 \right)$$
- ** Polynomial**
  $$\left( \langle x, x' \rangle + c \right)^d, c \geq 0, \ d \in \mathbb{N}$$
Theorem

For any symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and which satisfies

$$\int_{\mathcal{X} \times \mathcal{X}} k(x, x') f(x) f(x') dx dx' \geq 0 \text{ for all } f \in L_2(\mathcal{X})$$

there exist $\phi_i : \mathcal{X} \to \mathbb{R}$ and numbers $\lambda_i \geq 0$ where

$$k(x, x') = \sum_i \lambda_i \phi_i(x) \phi_i(x') \text{ for all } x, x' \in \mathcal{X}.$$

Interpretation

Double integral is the continuous version of a vector-matrix-vector multiplication. For positive semidefinite matrices we have

$$\sum_i \sum_j k(x_i, x_j) \alpha_i \alpha_j \geq 0.$$
Properties

Distance in Feature Space
Distance between points in feature space via
\[ d(x, x')^2 := \|\Phi(x) - \Phi(x')\|^2 \]
\[ = \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \Phi(x') \rangle + \langle \Phi(x'), \Phi(x') \rangle \]
\[ = k(x, x) + k(x', x') - 2k(x, x) \]

Kernel Matrix
To compare observations we compute dot products, so we study the matrix \( K \) given by
\[ K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_i, x_j) \]
where \( x_i \) are the training patterns.

Similarity Measure
The entries \( K_{ij} \) tell us the overlap between \( \Phi(x_i) \) and \( \Phi(x_j) \), so \( k(x_i, x_j) \) is a similarity measure.
for HW2, you don’t need to randomly choose training examples. just go over all training examples in the original order, and call that an epoch (same as HW1).
$\sigma = 1.0 \quad C = \infty$

\[
f(x) = \begin{cases} 
1 & \text{if } f(x) = 1 \\
0 & \text{if } f(x) = 0 \\
-1 & \text{if } f(x) = -1 
\end{cases}
\]

\[
f(x) = \sum_{i=1}^{N} \alpha_i y_i \exp \left( -\frac{||x - x_i||^2}{2\sigma^2} \right) + b
\]

Gaussian RBF kernel (default in sklearn)
\[ \sigma = 1.0 \quad C = 100 \]

Decrease C, gives wider (soft) margin
$f(x) = \sum_{i}^{N} \alpha_i y_i \exp \left( -\frac{||x - x_i||^2}{2\sigma^2} \right) + b$
$\sigma = 1.0 \quad C = \infty$

$$f(x) = \sum_{i}^{N} \alpha_i y_i \exp \left( -\frac{||x - x_i||^2}{2\sigma^2} \right) + b$$
$\sigma = 0.25 \quad C = \infty$

Decrease sigma, moves towards nearest neighbour classifier
\[ \sigma = 0.1 \quad C = \infty \]

\[ f(x) = \sum_{i}^{N} \alpha_i y_i \exp \left( -\frac{||x - x_i||^2}{2\sigma^2} \right) + b \]
Polynomial Kernels

[Increasing the degree like this accomplishes two things.
  – First, the data might become linearly separable when you lift them to a high enough degree, even if the original data are not linearly separable.
  – Second, raising the degree can increase the margin, so you might get a more robust separator.

However, if you raise the degree too high, you will overfit the data.]

this is in contrast with C:
smaller C => wide margin (underfitting)
larger C => narrow margin (overfitting)
Overfitting vs. Overfitting

UNDERFITTING (high bias)

OVERFITTING (high variance)

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1 x_2^2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \ldots)$

$M = 1$

$M = 9$

Training error vs. true error vs. model complexity.

Graphs showing the relationship between training error, true error, and model complexity.
From SVM to Nearest Neighbor

• for each test example \( x \), decide its label by the training example closest to \( x \)
• decision boundary highly non-linear (Voronoi)
• \( k \)-nearest neighbor (\( k \)-NN): smoother boundaries
small $k$: overfitting

K = 1

K = 3

K = 7

large $k$: underfitting

What about $k = N$?
# SVM vs. Nearest Neighbor

<table>
<thead>
<tr>
<th></th>
<th>Maximum Margin</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Need training</td>
<td>No training</td>
</tr>
<tr>
<td>Testing</td>
<td>Fast</td>
<td>slow</td>
</tr>
<tr>
<td>High Dimension</td>
<td>Usually good</td>
<td>Not so good</td>
</tr>
<tr>
<td>Multi-category</td>
<td>Expensive</td>
<td>Simple</td>
</tr>
</tbody>
</table>

- support vectors: few, all