Kernel methods

CSE 250B
Deviations from linear separability

**Noise**

Find a separator that minimizes a convex loss function related to the number of mistakes.

e.g. SVM, logistic regression.
Deviations from linear separability

Find a separator that minimizes a convex loss function related to the number of mistakes.

e.g. SVM, logistic regression.

What to do with this?
Adding new features

Actual boundary is something like $x_1 = x_2^2 + 5$. 
Adding new features

Actual boundary is something like \( x_1 = x_2^2 + 5 \).

- This is quadratic in \( x = (x_1, x_2) \)
- But it is linear in \( \Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1x_2) \)

**Basis expansion**: embed data in higher-dimensional feature space. Then we can use a linear classifier!
Basis expansion for quadratic boundaries

How to deal with a quadratic boundary?

Idea: augment the regular features $x = (x_1, x_2, \ldots, x_d)$ with

$x_1^2, x_2^2, \ldots, x_d^2$
$x_1x_2, x_1x_3, \ldots, x_{d-1}x_d$

Enhanced data vectors of the form:

$\Phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d)$
Quick question

Suppose $x = (x_1, x_2, x_3)$. What is the dimension of $\Phi(x)$?
Suppose $x = (x_1, \ldots, x_d)$. What is the dimension of $\Phi(x)$?
Perceptron revisited

Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y \Phi(x)$
  - $b = b + y$
Perceptron with basis expansion: examples
Perceptron with basis expansion: examples
Perceptron with basis expansion: examples
Perceptron with basis expansion: examples
Perceptron with basis expansion

Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y \Phi(x)$
  - $b = b + y$

Problem: number of features has now increased dramatically. For MNIST, with quadratic boundary: from 784 to 308504.

The kernel trick: implement this without ever writing down a vector in the higher-dimensional space!
Perceptron with basis expansion

Learning in the higher-dimensional feature space:

- \( w = 0 \) and \( b = 0 \)
- while some \( y(w \cdot \Phi(x) + b) \leq 0 \):
  - \( w = w + y \Phi(x) \)
  - \( b = b + y \)

**Problem:** number of features has now increased dramatically. For MNIST, with quadratic boundary: from 784 to 308504.
Perceptron with basis expansion

Learning in the higher-dimensional feature space:

• $w = 0$ and $b = 0$
• while some $y(w \cdot \Phi(x) + b) \leq 0$:
  • $w = w + y \Phi(x)$
  • $b = b + y$

**Problem:** number of features has now increased dramatically. For MNIST, with quadratic boundary: from 784 to 308504.

**The kernel trick:** implement this without ever writing down a vector in the higher-dimensional space!
The kernel trick

- \( w = 0 \) and \( b = 0 \)
- while some \( y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0 \):
  - \( w = w + y^{(i)} \Phi(x^{(i)}) \)
  - \( b = b + y^{(i)} \)
The kernel trick

- \( w = 0 \) and \( b = 0 \)
- While some \( y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0 \):
  - \( w = w + y^{(i)} \Phi(x^{(i)}) \)
  - \( b = b + y^{(i)} \)

1. Represent \( w \) in dual form: \( \alpha = (\alpha_1, \ldots, \alpha_n) \).

\[
w = \sum_{j=1}^{n} \alpha_j y^{(j)} \Phi(x^{(j)})
\]
The kernel trick

- \( w = 0 \) and \( b = 0 \)
- while some \( y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0 \):
  - \( w = w + y^{(i)} \Phi(x^{(i)}) \)
  - \( b = b + y^{(i)} \)

1. Represent \( w \) in **dual** form: \( \alpha = (\alpha_1, \ldots, \alpha_n) \).

\[
w = \sum_{j=1}^{n} \alpha_j y^{(j)} \Phi(x^{(j)})
\]

2. Compute \( w \cdot \Phi(x) \) using the dual representation.

\[
w \cdot \Phi(x) = \sum_{j=1}^{n} \alpha_j y^{(j)} (\Phi(x^{(j)}) \cdot \Phi(x))
\]
**The kernel trick**

- $w = 0$ and $b = 0$
- while some $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$ :
  - $w = w + y^{(i)} \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

1. Represent $w$ in **dual** form: $\alpha = (\alpha_1, \ldots, \alpha_n)$.

   $$w = \sum_{j=1}^{n} \alpha_j y^{(j)} \Phi(x^{(j)})$$

2. Compute $w \cdot \Phi(x)$ using the dual representation.

   $$w \cdot \Phi(x) = \sum_{j=1}^{n} \alpha_j y^{(j)} (\Phi(x^{(j)}) \cdot \Phi(x))$$

3. Compute $\Phi(x) \cdot \Phi(z)$ without ever writing out $\Phi(x)$ or $\Phi(z)$. 
Computing dot products

First, in 2-d.
Suppose $x = (x_1, x_2)$ and $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1x_2)$.

Actually, tweak a little: $\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$

What is $\Phi(x) \cdot \Phi(z)$?
Computing dot products

Suppose \( x = (x_1, x_2, \ldots, x_d) \) and

\[
\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d)
\]

\[
\Phi(x) \cdot \Phi(z) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d) \cdot (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_d, z_1^2, \ldots, z_d^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{d-1}z_d)
\]

\[
= 1 + 2 \sum_i x_i z_i + \sum_i x_i^2 z_i^2 + 2 \sum_{i \neq j} x_i x_j z_i z_j
\]

\[
= (1 + x_1 z_1 + \cdots + x_d z_d)^2 = (1 + x \cdot z)^2
\]

For MNIST:
We are computing dot products in 308504-dimensional space.
But it takes time proportional to 784, the original dimension!
Kernel Perceptron

Learning from data \((x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}\)

**Primal form:**
- \(w = 0\) and \(b = 0\)
- while there is some \(i\) with \(y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0\):
  - \(w = w + y^{(i)} \Phi(x^{(i)})\)
  - \(b = b + y^{(i)}\)

**Dual form:** \(w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})\), where \(\alpha \in \mathbb{R}^n\)
- \(\alpha = 0\) and \(b = 0\)
- while some \(i\) has \(y^{(i)} \left( \sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)}) + b \right) \leq 0\):
  - \(\alpha_i = \alpha_i + 1\)
  - \(b = b + y^{(i)}\)

To classify a new point \(x\): \(\text{sign} \left( \sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) + b \right)\).
Does this work with SVMs?

\begin{align*}
\text{(PRIMAL)} & \quad \min_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.:} & \quad y^{(i)} (w \cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n \\
& \quad \xi \geq 0
\end{align*}

\begin{align*}
\text{(DUAL)} & \quad \max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)} \cdot x^{(j)}) \\
\text{s.t.:} & \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \\
& \quad 0 \leq \alpha_i \leq C
\end{align*}

Solution: \( w = \sum_{i} \alpha_i y^{(i)} x^{(i)} \).
Kernel SVM

1. **Basis expansion.** Mapping $x \mapsto \Phi(x)$.

2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))$$

s.t.: $\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$

$0 \leq \alpha_i \leq C$

This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$. Offset $b$ also follows.

3. **Classification.** Given a new point $x$, classify as

$$\text{sign} \left( \sum_i \alpha_i y^{(i)} (\Phi(x^{(i)}) \cdot \Phi(x)) + b \right).$$
Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

SVM:
Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

SVM:
Polynomial decision boundaries

When decision surface is a polynomial of order $p$:

- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_p - 3^3$.

- Degree-$p$ polynomial in $x \iff$ linear in $\Phi(x)$.

- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.

- Kernel function: $k(x, z) = (1 + x \cdot z)^p$. 
Polynomial decision boundaries

When decision surface is a polynomial of order $p$:

- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1x_2^2x_3^{p-3}$. (How many such terms are there, roughly?)
Polynomial decision boundaries

When decision surface is a polynomial of order $p$:

- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1x_2^2x_3^{p-3}$. (How many such terms are there, roughly?)
- Degree-$p$ polynomial in $x \iff$ linear in $\Phi(x)$. 
Polynomial decision boundaries

When decision surface is a polynomial of order $p$:

- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1x_2^2x_3^{p-3}$. (How many such terms are there, roughly?)
- Degree-$p$ polynomial in $x \iff$ linear in $\Phi(x)$.
- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.
Polynomial decision boundaries

When decision surface is a polynomial of order \( p \):

- Let \( \Phi(x) \) consist of all terms of order \( \leq p \), such as \( x_1 x_2^2 x_3^{p-3} \). (How many such terms are there, roughly?)
- Degree-\( p \) polynomial in \( x \) \( \iff \) linear in \( \Phi(x) \).
- Same trick works: \( \Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p \).
- **Kernel function**: \( k(x, z) = (1 + x \cdot z)^p \).
String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

What kind of embedding $\Phi(x)$ is suitable for variable-length sequences $x$? We will use an infinite-dimensional embedding!
String kernels

Sequence data:
- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

\[ \mathcal{X} = \{A, C, G, T\}^* \]
\[ \mathcal{X} = \{\text{English words}\}^* \]
String kernels

Sequence data:
- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

\[ \mathcal{X} = \{A, C, G, T\}^* \]
\[ \mathcal{X} = \{\text{English words}\}^* \]

What kind of embedding \( \Phi(x) \) is suitable for variable-length sequences \( x \)?
String kernels

Sequence data:
- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

\[ X = \{A, C, G, T\}^* \]
\[ \mathcal{X} = \{\text{English words}\}^* \]

What kind of embedding \( \Phi(x) \) is suitable for variable-length sequences \( x \)?

We will use an infinite-dimensional embedding!
String kernels, cont’d

For each substring $s$, define feature:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$
For each substring $s$, define feature:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of “aardvark” includes features

$$\Phi_{ar}(aardvark) = 2, \Phi_{th}(aardvark) = 0, \ldots$$

Linear classifier based on such features is very expressive.
String kernels, cont’d

For each substring $s$, define feature:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of “aardvark” includes features

$$\Phi_{ar}(aardvark) = 2, \Phi_{th}(aardvark) = 0, \ldots$$

Linear classifier based on such features is very expressive.

To compute $k(x, z) = \Phi(x) \cdot \Phi(z)$:

*for each substring $s$ of $x$: count how often $s$ appears in $z$*
String kernels, cont’d

For each substring $s$, define feature:

$$
\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x
$$

and let $\Phi(x)$ be a vector with one coordinate for each string:

$$
\Phi(x) = (\Phi_s(x) : \text{all strings } s).
$$

Example: the embedding of “aardvark” includes features

$$
\Phi_{ar}(aardvark) = 2, \Phi_{th}(aardvark) = 0, \ldots
$$

Linear classifier based on such features is very expressive.

To compute $k(x, z) = \Phi(x) \cdot \Phi(z)$:

for each substring $s$ of $x$: count how often $s$ appears in $z$

Using dynamic programming, this takes time $O(|x| \cdot |z|)$. 
The kernel function

We never explicitly construct the embedding $\Phi(x)$.  

- What we actually use: **kernel function** $k(x, z) = \Phi(x) \cdot \Phi(z)$.  
- Think of $k(x, z)$ as a **measure of similarity** between $x$ and $z$.  
- Rewrite learning algorithm and final classifier in terms of $k$. 
The kernel function

We never explicitly construct the embedding $\Phi(x)$.

- What we actually use: **kernel function** $k(x, z) = \Phi(x) \cdot \Phi(z)$.
- Think of $k(x, z)$ as a **measure of similarity** between $x$ and $z$.
- Rewrite learning algorithm and final classifier in terms of $k$.

**Kernel Perceptron:**

- $\alpha = 0$ and $b = 0$
- while some $i$ has $y^{(i)} \left( \sum_j \alpha_j y^{(j)} k(x^{(j)}, x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

To classify a new point $x$: $\text{sign} \left( \sum_j \alpha_j y^{(j)} k(x^{(j)}, x) + b \right)$. 
Kernel SVM, revisited

1 Kernel function. Define a similarity function \( k(x, z) \).

2 Learning. Solve the dual problem:

\[
\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)})
\]

s.t.:
\[
\sum_{i=1}^{n} \alpha_i y^{(i)} = 0
\]

\[0 \leq \alpha_i \leq C\]

This yields \( \alpha \). Offset \( b \) also follows.

3 Classification. Given a new point \( x \), classify as

\[
\text{sign} \left( \sum_{i} \alpha_i y^{(i)} k(x^{(i)}, x) + b \right).
\]
Choosing the kernel function

The final classifier is a **similarity-weighted vote**, 

\[ F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x) \]

(plus an offset term, \( b \)).
Choosing the kernel function

The final classifier is a similarity-weighted vote,

\[ F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x) \]

(plus an offset term, \(b\)).

Can we choose \(k\) to be any similarity function?
Choosing the kernel function

The final classifier is a **similarity-weighted vote**, 

\[ F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x) \]

(plus an offset term, \(b\)).

Can we choose \(k\) to be **any** similarity function?

- Not quite: need \(k(x, z) = \Phi(x) \cdot \Phi(z)\) for some embedding \(\Phi\).
Choosing the kernel function

The final classifier is a similarity-weighted vote,

$$F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x)$$

(plus an offset term, $b$).

Can we choose $k$ to be any similarity function?

- Not quite: need $k(x, z) = \Phi(x) \cdot \Phi(z)$ for some embedding $\Phi$.
- **Mercer’s condition**: same as requiring that for any finite set of points $x^{(1)}, \ldots, x^{(m)}$, the $m \times m$ similarity matrix $K$ given by
  $$K_{ij} = k(x^{(i)}, x^{(j)})$$
  is positive semidefinite.
RBF kernel: examples
RBF kernel: examples
RBF kernel: examples
RBF kernel: examples
The scale parameter

Recall prediction function:
\[ F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x). \]

For the RBF kernel, \( k(x, z) = e^{-\|x-z\|^2/s^2} \),

1. How does this function behave as \( s \uparrow \infty \)?
2. How does this function behave as \( s \downarrow 0 \)?
3. As we get more data, should we increase or decrease \( s \)?
Kernels: postscript

1 Customized kernels
   • For different domains (NLP, biology, speech, ...)
   • Over different structures (sequences, sets, graphs, ...)

2 Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.

3 Speeding up learning and prediction
   The $n \times n$ kernel matrix $k(x_i, x_j)$ is a bottleneck for large $n$.
   One idea:
   • Go back to the primal space!
   • Replace the embedding $\Phi$ by a low-dimensional mapping $\tilde{\Phi}$ such that
     $\tilde{\Phi}(x) \cdot \tilde{\Phi}(z) \approx \Phi(x) \cdot \Phi(z)$.
   This can be done, for instance, by writing $\Phi$ in the Fourier basis and then randomly sampling features.
Kernels: postscript

1 Customized kernels
   - For different domains (NLP, biology, speech, ...)
   - Over different structures (sequences, sets, graphs, ...)

2 Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.

3 Speeding up learning and prediction
   The $n \times n$ kernel matrix $k(x(i), x(j))$ is a bottleneck for large $n$.
   One idea:
   - Go back to the primal space!
   - Replace the embedding $\Phi$ by a low-dimensional mapping $\tilde{\Phi}$ such that $\tilde{\Phi}(x) \cdot \tilde{\Phi}(z) \approx \Phi(x) \cdot \Phi(z)$.
   This can be done, for instance, by writing $\Phi$ in the Fourier basis and then randomly sampling features.
Kernels: postscript

1 Customized kernels
   - For different domains (NLP, biology, speech, ...)
   - Over different structures (sequences, sets, graphs, ...)

2 Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.

3 Speeding up learning and prediction
   The $n \times n$ kernel matrix $k(x^{(i)}, x^{(j)})$ is a bottleneck for large $n$.
   One idea:
   - Go back to the primal space!
   - Replace the embedding $\Phi$ by a low-dimensional mapping $\tilde{\Phi}$ such that
     \[
     \tilde{\Phi}(x) \cdot \tilde{\Phi}(z) \approx \Phi(x) \cdot \Phi(z).
     \]
   This can be done, for instance, by writing $\Phi$ in the Fourier basis and then randomly sampling features.