LECTURE 20

• Last class: **Binary Classification Problem**

  \[ \text{test results} \rightarrow \text{diagnosis} \]

  Training data: \((x_i, y_i), i = 1, \ldots, n.\) \quad \text{in } \mathbb{R}^d \in \{-1, 1\}

  Learning: find \(w \in \mathbb{R}^d: \langle w, x_i \rangle \{> 1 \text{ if } y_i = 1 \}

  \Rightarrow \langle w, x_i \rangle y_i > 1 \quad \forall i.\]

  Prediction: diagnosis for a new \(x\):

  \[ f(x) = \text{sign} \langle w, x \rangle \]

• Typical data is not **perfectly separable**: \(\exists\) bad pts, outliers.

  To allow them, we

  • **Penalize**: pay penalty \(1 - \langle w, x_i \rangle y_i\) for each bad pt:

  \[ \Rightarrow \text{loss} \quad l(w) := \sum_{i=1}^{n} (1 - \langle w, x_i \rangle y_i)_+ \quad \text{where} \]

  Minimize \(l(w)\) over \(w \in \mathbb{R}^n\) (convex program)

  • Furthermore, **regularize** (for robustness):

  \[ l(w) = \sum_{i=1}^{n} (1 - \langle w, x_i \rangle y_i)_+ + \lambda \|w\|_2^2. \]

  "**Soft-margi**n SVM"

• And, since data may not be **linearly separated**, **kernelize**...
**KERNEL METHOD** (Lec. 19)

- **Rewrite the algorithm in terms of inner products**: \( W = \sum \alpha_j x_j \Rightarrow \)

\[
\ell(w) = \sum_{i=1}^{n} \left( 1 - \sum_{j=1}^{n} \alpha_j \langle x_i, x_j \rangle y_i \right) + \lambda \sum_{ij=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle,
\]

which we minimize in \( \alpha_j \).

- **Prediction**: \( f(x) = \text{sign} \left( \sum \alpha_j \langle x, x_j \rangle \right) \).

- **Transform data using a nonlinear feature map**: \( \phi: \mathbb{R}^d \rightarrow \mathcal{H} \)

\[
\Rightarrow \langle x, y \rangle \rightarrow \langle \phi(x), \phi(y) \rangle = K(x, y) \quad \text{"kernel" Hilbert space}
\]

- **Choose a nice kernel**, e.g., \( K(x, y) = \exp \left( - \frac{\|x-y\|^2}{2\sigma^2} \right) \) (RBF)

  and replace all \( \langle \cdot, \cdot \rangle \) by \( K(\cdot, \cdot) \).

  "Kernel SVM"

- **Warning**: for practical implementation, see Wikipedia (kernelize the dual program)

**EXAMPLE**:

- Gaussian RBF
- Skiena 2
- Separable
- Bounds 1

---

-2-
WHAT FUNCTIONS $K(x,y)$ ARE ALLOWED?

- Can we express a function $K$ as $K(x,y) = \langle \phi(x), \phi(y) \rangle$?
- No: the matrix $[K(x_i, x_j)]_{i,j=1}^n = [\langle \phi(x_i), \phi(x_j) \rangle]_{i,j=1}^n$ is a Gram matrix $\Rightarrow$ must be PSD.

**DEF** A kernel is a symmetric function $K: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that

$$[K(x_i, x_j)]_{i,j=1}^n \succeq 0 \quad \forall \ x_1, \ldots, x_n \in \mathbb{R}^d.$$

- This is a sufficient condition.

**THM (Mercer's Condition)** A continuous kernel $K$

exists map $\phi: \mathbb{R}^d \rightarrow H$ (Hilbert space) such that

$K(x,y) = \langle \phi(x), \phi(y) \rangle_H \quad \forall x,y \in \mathbb{R}^d$.

This is an infinite-dimensional version of the HW problem:

A PSD matrix is a Gram matrix $(x \mapsto i, y \mapsto j)$.

- Mercer's condition (PSD) is difficult to check.
  However, using it, we can build new kernels from old ones using simple rules (HW), such as addition, exponentiation, etc.

- Examples: (a) RBF $K(x,y) = \exp\left(-\frac{\|x-y\|^2}{\sigma^2}\right)$,
  (b) polynomial $K(x,y) = (1 + \langle x, y \rangle)^d$, ... Kernels arise in:
**NEURAL NETWORKS**

**Def:** A neuron is a composition of linear & nonlinear function

\[ y = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3) \]

\[ \sigma = \text{sign} \text{ or } \tanh \]

**Ex:** SVM is a neuron: \( \text{sign}(w, x) = \text{sign}(w_1 x_1 + \ldots + w_d x_d) \)

We train weights \( w_j \)

**Def:** A neural network is a superposition of neurons

We train weights \( (u_{ij}), (w_j) \)

Connection to kernels:

- The first layer computes a feature map \( \phi: \mathbb{R}^d \rightarrow \mathbb{R}^n \):

\[
\phi(x) = \begin{bmatrix}
\text{sgn}(u_1, x) \\
\vdots \\
\text{sgn}(u_n, x)
\end{bmatrix}
\]

Kernel:

\[
k(u, y) = \langle \phi(x), \phi(y) \rangle = \frac{1}{n} \sum_{i=1}^{n} \text{sgn}(u_i, x) \langle u, y \rangle
\]

Let's initialize all weights \( u_{ij} \sim N(0, \sigma) \) iid

\[ \mathbb{E} \text{sgn}(u, x) \text{sgn}(u, y) = \frac{2}{\pi} \arcsin \langle x, y \rangle \]

Hence: a neural network = kernel SVM.

- But: after initialization, a neural network is training weights \( u_{ij} \)

\[ \Rightarrow \text{trains kernel } k \]

Dynamics of the prediction function is described by the neural tangent kernel (NTK).