Kernels & Probabilistic Modeling

Mengye Ren

NYU

October 3, 2023

Logistics

- Oct 10 (next week): Legislative Day No Class
- Oct 17: Homework 2 Due
- Oct 24: Midterm, in class, covers everything up until Oct 17

Last Lecture

Two ways to derive the SVM optimization problem:

- Maximize the margin
- Minimize the hinge loss with ℓ_2 regularization

Both leads to the minimum norm solution satisfying certain margin constraints.

- **Hard-margin SVM**: all points must be correctly classified with the margin constraints
- **Soft-margin SVM**: allow for margin constraint violation with some penalty

Subgradient: generalize gradient for non-differentiable convex functions

Dual problem: Lagrange multiplier α_i for each example.

Strong duality: For some convex problems, the primal and dual have the same solution.

Dual Problem: Dependence on x through inner products

- SVM Dual Problem:

$$\begin{aligned} \sup_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_j^T x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \in \left[0, \frac{c}{n}\right] \; i = 1, \ldots, n. \end{aligned}$$

- Note that all dependence on inputs x_i and x_j is through their inner product: $\langle x_j, x_i \rangle = x_j^T x_i$.
- We can replace $x_j^T x_i$ by other products...
- This is a "kernelized" objective function.

Feature Maps

The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$
- But $\mathcal{X} = \mathrm{R}^d$ for the specific methods we've developed:
  - Ridge regression
  - Lasso regression
  - Support Vector Machines
- Our hypothesis space for these was all affine functions on R^d:

$$\mathcal{F} = \left\{ x \mapsto w^T x + b \mid w \in \mathrm{R}^d, b \in \mathrm{R} \right\}.$$

- What if we want to do prediction on inputs not natively in R^d?

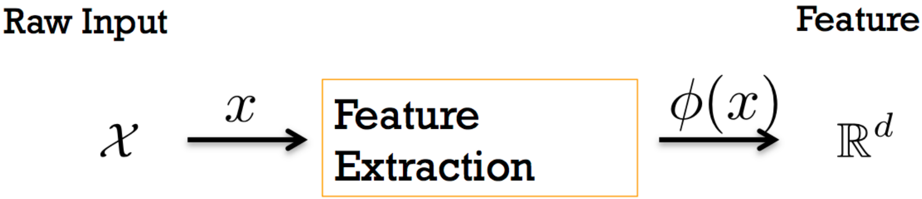
The Input Space $\mathcal{X}$

- Often want to use inputs not natively in R^d:
  - Text documents
  - Image files
  - Sound recordings
  - DNA sequences
- They may be represented in numbers, but...
- The ith entry of each sequence should have the same “meaning”
- All the sequences should have the same length

Feature Extraction

Definition

Mapping an input from $\mathcal{X}$ to a vector in R^d is called **feature extraction** or **featurization**.



Linear Models with Explicit Feature Map

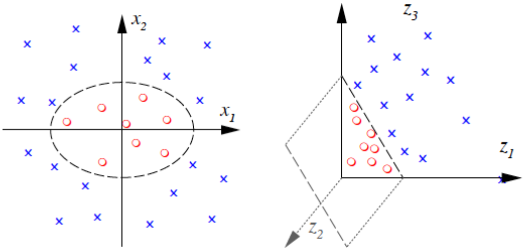
- Input space: $\mathcal{X}$ (no assumptions)
- Introduce **feature map** $\phi : \mathcal{X} \to \mathrm{R}^d$
- The feature map maps into the **feature space** R^d.
- Hypothesis space of affine functions on feature space:

$$\mathcal{F} = \left\{ x \mapsto w^T \phi(x) + b \mid w \in \mathrm{R}^d, b \in \mathrm{R} \right\}.$$

Geometric Example: Two class problem, nonlinear boundary

$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)}x_1x_2, x_2^2)$$



- With identity feature map $\phi(x) = (x_1, x_2)$ and linear models, can't separate regions
- With appropriate featurization $\phi(x) = \left(x_1, x_2, x_1^2 + x_2^2\right)$, becomes linearly separable .
- Video: http://youtu.be/3liCbRZPrZA

Expressivity of Hypothesis Space

- For linear models, to grow the hypothesis spaces, we must add features.
- Sometimes we say a larger hypothesis is more expressive.
    - (can fit more relationships between input and action)
- Many ways to create new features.

Handling Nonlinearity with Linear Methods

Example Task: Predicting Health

- General Philosophy: Extract every feature that might be relevant
- Features for medical diagnosis
  - height
  - weight
  - body temperature
  - blood pressure
  - etc...

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Issues for Linear Predictors

- For linear predictors, it's important **how** features are added
  - The relation between a feature and the label may not be linear
  - There may be complex dependence among features
- Three types of nonlinearities can cause problems:
  - Non-monotonicity
  - Saturation
  - Interactions between features

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathrm{R}$ (positive is good)
- Hypothesis Space $\mathcal{F}$=\{affine functions of temperature\}
- Issue:
  - Health is not an affine function of temperature.
  - Affine function can either say
    - Very high is bad and very low is good, or
    - Very low is bad and very high is good,
    - But here, both extremes are bad.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 1

- Transform the input:

$$\phi(x) = \left[1, \{\text{temperature(x)-37}\}^2\right],$$

  where 37 is “normal” temperature in Celsius.
- Ok, but requires manually-specified domain knowledge
  - Do we really need that?
  - What does $w^T\phi(x)$ look like?

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 2

- Think less, put in more:

$$\phi(x) = \left[1, \text{temperature}(x), \{\text{temperature}(x)\}^2\right].$$

- More expressive than Solution 1.

General Rule

Features should be simple building blocks that can be pieced together.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

- Setting: Find products relevant to user's query
- Input: Product x
- Output: Score the relevance of x to user's query
- Feature Map:

$$\phi(x) = [1, N(x)],$$

  where $N(x) =$ number of people who bought x.
- We expect a monotonic relationship between $N(x)$ and relevance, but also expect diminishing return.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: Solve with nonlinear transform

- Smooth nonlinear transformation:

$$\phi(x) = [1, \log\{1 + N(x)\}]$$

  - $\log(\cdot)$ good for values with large dynamic ranges

- Discretization (a discontinuous transformation):

$$\phi(x) = (\mathbb{1}[0 \leqslant N(x) < 10], \mathbb{1}[10 \leqslant N(x) < 100], \ldots)$$

  - Small buckets allow quite flexible relationship

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: The Issue

- Input: Patient information x
- Action: Health score $y \in \mathrm{R}$ (higher is better)
- Feature Map

$$\phi(x) = [\text{height}(x), \text{weight}(x)]$$

- Issue: It's the weight *relative* to the height that's important.
- Impossible to get with these features and a linear classifier.
- Need some **interaction** between height and weight.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 1

- Google “ideal weight from height”
- J. D. Robinson’s “ideal weight” formula:

$$\text{weight(kg)} = 52 + 1.9\,[\text{height(in)} - 60]$$

- Make score square deviation between height(h) and ideal weight(w)

$$f(x) = (52 + 1.9\,[h(x) - 60] - w(x))^2$$

- WolframAlpha for complicated Mathematics:

$$f(x) = 3.61h(x)^2 - 3.8h(x)w(x) - 235.6h(x) + w(x)^2 + 124w(x) + 3844$$

From Percy Liang’s "Lecture 3" slides from Stanford’s CS221, Autumn 2014.

Interactions: Approach 2

- Just include all second order features:

$$\phi(x) = \left[1, h(x), w(x), h(x)^2, w(x)^2, \underbrace{h(x)w(x)}_{\text{cross term}} \right]$$

- More flexible, no Google, no WolframAlpha.

General Principle

Simpler building blocks replace a single “smart” feature.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Monomial Interaction Terms

Interaction terms are useful building blocks to model non-linearities in features.

- Suppose we start with $x = (1, x_1, \ldots, x_d) \in \mathrm{R}^{d+1} = \mathcal{X}$.
- Consider adding all **monomials** of degree M: $x_1^{p_1} \cdots x_d^{p_d}$, with $p_1 + \cdots + p_d = M$.
  - Monomials with degree 2 in 2D space: x_1^2, x_2^2, x_1x_2
- How many features will we end up with?

$$\begin{array}{rcl}
x_1^3 & \longleftrightarrow & \star\ \star\ \star \mid \mid \\
x_1^2x_2 & \longleftrightarrow & \star\ \star \mid \star \mid \\
x_1^2x_3 & \longleftrightarrow & \star\ \star \mid \mid \star \\
x_1x_2^2 & \longleftrightarrow & \star \mid \star\ \star \mid \\
x_1x_2x_3 & \longleftrightarrow & \star \mid \star \mid \star \\
x_1x_3^2 & \longleftrightarrow & \star \mid \mid \star\ \star \\
x_2^3 & \longleftrightarrow & \mid \star\ \star\ \star \mid \\
x_2^2x_3 & \longleftrightarrow & \mid \star\ \star \mid \star \\
x_2x_3^2 & \longleftrightarrow & \mid \star \mid \star\ \star \\
x_3^3 & \longleftrightarrow & \mid \mid \star\ \star\ \star
\end{array}$$

Big Feature Spaces

This leads to extremely large data matrices

- For $d = 40$ and $M = 8$, we get 314457495 features.

Very large feature spaces have two potential issues:

- Overfitting
- Memory and computational costs

Solutions:

- Overfitting we handle with regularization.
- **Kernel methods** can help with memory and computational costs when we go to high (or infinite) dimensional spaces.

The Kernel Trick

SVM with Explicit Feature Map

- Let $\psi : \mathcal{X} \to \mathbf{R}^d$ be a feature map.
- The SVM objective (with explicit feature map):

$$\min_{w \in \mathbf{R}^d} \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i w^T \psi(x_i)\right).$$

- Computation is costly if d is large (e.g. with high-degree monomials)
- Last time we mentioned an equivalent optimization problem from Lagrangian duality.

SVM Dual Problem

- By Lagrangian duality, it is equivalent to solve the following dual problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \psi(x_j)^T \psi(x_i) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \in \left[0, \frac{c}{n}\right] \quad \forall i. \end{aligned}$$

- If α^* is an optimal value, then

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i) \quad \text{and} \quad \hat{f}(x) = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i)^T \psi(x).$$

- Key observation: $\psi(x)$ only shows up in inner products with another $\psi(x')$ *for both training and inference*.

Compute the Inner Products

Consider 2D data. Let's introduce degree-2 monomials using $\psi : \mathbb{R}^2 \to \mathbb{R}^3$.

$$(x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

The inner product is

$$\begin{aligned}
\psi(x)^T\psi(x') &= x_1^2x_1'^2 + (\sqrt{2}x_1x_2)(\sqrt{2}x_1'x_2') + x_2^2x_2'^2 \\
&= (x_1x_1')^2 + 2(x_1x_1')(x_2x_2') + (x_2x_2')^2 \\
&= (x_1x_1' + x_2x_2')^2 \\
&= (x^Tx')^2
\end{aligned}$$

We can calculate the inner product $\psi(x)^T\psi(x')$ in the original input space without accessing the features $\psi(x)$!

Compute the Inner Products

Now, consider monomials up to degree-2:

$$(x_1, x_2) \mapsto (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

The inner product can be computed by

$$\psi(x)^T\psi(x') = (1 + x^T x')^2 \quad \text{(check)}.$$

More generally, for features maps producing monomials up to degree-p, we have

$$\psi(x)^T\psi(x') = (1 + x^T x')^p.$$

(Note that the coefficients of each monomial in ψ may not be 1)

Kernel trick: we do not need explicit features to calculate inner products.

- Using explicit features: $O(d^p)$
- Using implicit computation: $O(d)$

Kernel Function

The Kernel Function

- **Input space**: $\mathcal{X}$
- **Feature space**: $\mathcal{H}$ (a Hilbert space, e.g. $\mathbf{R}^d$)
- **Feature map**: $\psi : \mathcal{X} \to \mathcal{H}$
- The **kernel function** corresponding to ψ is

  $$k(x, x') = \left\langle \psi(x), \psi(x') \right\rangle,$$

  where $\langle \cdot, \cdot \rangle$ is the inner product associated with $\mathcal{H}$.

Why introduce this new notation $k(x, x')$?

- We can often evaluate $k(x, x')$ without explicitly computing $\psi(x)$ and $\psi(x')$.

When can we use the kernel trick?

Some Methods Can Be "Kernelized"

Definition

A method is **kernelized** if every feature vector $\psi(x)$ only appears inside an inner product with another feature vector $\psi(x')$. This applies to both the optimization problem and the prediction function.

The SVM Dual is a kernelization of the original SVM formulation.

Optimization:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \psi\left(x_j\right)^T \psi(x_i) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \in \left[0, \frac{c}{n}\right] \quad \forall i. \end{aligned}$$

Prediction:

$$\hat{f}(x) = \sum_{i=1}^{n} \alpha_i^* y_i \psi(x_i)^T \psi(x).$$

The Kernel Matrix

Definition

The **kernel matrix** for a kernel k on $x_1, \dots, x_n \in \mathcal{X}$ is

$$K = \left(k(x_i, x_j)\right)_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \dots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \in \mathrm{R}^{n \times n}.$$

- In ML this is also called a **Gram matrix**, but traditionally (in linear algebra), Gram matrices are defined without reference to a kernel or feature map.

The Kernel Matrix

- The kernel matrix summarizes all the information we need about the training inputs $x_1, \ldots, x_n$ to solve a kernelized optimization problem.
- In the kernelized SVM, we can replace $\psi(x_i)^T \psi(x_j)$ with K_{ij}:

$$\begin{array}{ll} \text{maximize}_{\alpha} & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K_{ij} \\ \text{s.t.} & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \in \left[0, \frac{c}{n}\right] \ i = 1, \ldots, n. \end{array}$$

Kernel Methods

Given a kernelized ML algorithm (i.e. all $\psi(x)$'s show up as $\langle\psi(x),\psi(x')\rangle$),

- Can swap out the inner product for a new kernel function.
- New kernel may correspond to a very high-dimensional feature space.
- Once the kernel matrix is computed, the computational cost depends on number of data points n, rather than the dimension of feature space d.
- Useful when $d >> n$.
- Computing the kernel matrix may still depend on d and the essence of the **trick** is getting around this $O(d)$ dependence.

Example Kernels

Kernels as Similarity Scores

- Often useful to think of the $k(x, x')$ as a **similarity score** for x and x'.
- We can design similarity functions without thinking about the explicit feature map, e.g. “string kernels”, “graph kernels”.
- How do we know that our kernel functions actually correspond to inner products in some feature space?

How to Get Kernels?

- Explicitly construct $\psi(x): \mathcal{X} \to \mathrm{R}^d$ (e.g. monomials) and define $k(x,x') = \psi(x)^T\psi(x')$.
- Directly define the kernel function $k(x,x')$ (“similarity score”), and verify it corresponds to $\langle\psi(x),\psi(x')\rangle$ for some ψ.

There are many theorems to help us with the second approach.

Linear Algebra Review: Positive Semidefinite Matrices

Definition

A real, symmetric matrix $M \in \mathbb{R}^{n \times n}$ is **positive semidefinite (psd)** if for any $x \in \mathbb{R}^n$,

$$x^T M x \geqslant 0.$$

Theorem

The following conditions are each necessary and sufficient for a symmetric matrix M to be positive semidefinite:

- *M can be factorized as $M = R^T R$, for some matrix R.*
- *All eigenvalues of M are greater than or equal to 0.*

Positive Definite Kernel

Definition

A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathrm{R}$ is a **positive definite (pd)** kernel on $\mathcal{X}$ if for any finite set $\{x_1, \ldots, x_n\} \in \mathcal{X}$ ($n \in \mathbb{N}$), the kernel matrix on this set

$$K = \left(k(x_i, x_j)\right)_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \ldots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

is a positive semidefinite matrix.

- Symmetric: $k(x, x') = k(x', x)$
- The kernel matrix needs to be positive semidefinite for any finite set of points.
- Equivalent definition: $\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geqslant 0$ given $\alpha_i \in \mathrm{R}\ \forall i$.

Mercer's Theorem

Theorem

A symmetric function $k(x, x')$ can be expressed as an inner product

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

*for some ψ if and only if $k(x, x')$ is **positive definite**.*

- Proving a kernel function is positive definite is typically not easy.
- But we can construct new kernels from valid kernels.

Generating New Kernels from Old

- Suppose $k, k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$ are pd kernels. Then so are the following:

$$\begin{aligned}
k_{\text{new}}(x,x') &= \alpha k(x,x') \quad \text{for } \alpha \geqslant 0 \quad \text{(non-negative scaling)} \\
k_{\text{new}}(x,x') &= k_1(x,x') + k_2(x,x') \quad \text{(sum)} \\
k_{\text{new}}(x,x') &= k_1(x,x')k_2(x,x') \quad \text{(product)} \\
k_{\text{new}}(x,x') &= k(\psi(x),\psi(x')) \text{ for any function } \psi(\cdot) \quad \text{(recursion)} \\
k_{\text{new}}(x,x') &= f(x)f(x') \text{ for any function } f(\cdot) \quad (f \text{ as 1D feature map})
\end{aligned}$$

- Lots more theorems to help you construct new kernels from old.

Based on Mark Schmidt's slides:https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L12.5.pdf

Linear Kernel

- Input space: $\mathcal{X} = \mathbf{R}^d$
- Feature space: $\mathcal{H} = \mathbf{R}^d$, with standard inner product
- Feature map

$$\psi(x) = x$$

- Kernel:

$$k(x, x') = x^T x'$$

Quadratic Kernel in $\mathbf{R}^d$

- Input space $\mathcal{X} = \mathbf{R}^d$
- Feature space: $\mathcal{H} = \mathbf{R}^D$, where $D = d + \binom{d}{2} \approx d^2/2$.
- Feature map:

$$\psi(x) = (x_1, \dots, x_d, x_1^2, \dots, x_d^2, \sqrt{2}x_1x_2, \dots, \sqrt{2}x_ix_j, \dots \sqrt{2}x_{d-1}x_d)^T$$

- Then for $\forall x, x' \in \mathbf{R}^d$

$$\begin{aligned} k(x,x') &= \langle \psi(x), \psi(x') \rangle \\ &= \langle x, x' \rangle + \langle x, x' \rangle^2 \end{aligned}$$

- Computation for inner product with explicit mapping: $O(d^2)$
- Computation for implicit kernel calculation: $O(d)$.

Polynomial Kernel in $\mathbf{R}^d$

- Input space $\mathcal{X} = \mathbf{R}^d$
- Kernel function:
$$k(x, x') = \left(1 + \langle x, x' \rangle\right)^M$$
- Corresponds to a feature map with all monomials up to degree M.
- For any M, computing the kernel has same computational cost
- Cost of explicit inner product computation grows rapidly in M.

Radial Basis Function (RBF) / Gaussian Kernel

Input space $\mathcal{X} = \mathrm{R}^d$

$$k(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right),$$

where σ^2 is known as the bandwidth parameter.

- Probably the most common nonlinear kernel.
- Does it act like a similarity score?
- Have we departed from our "inner product of feature vector" recipe?
    - Yes and no: corresponds to an infinite dimensional feature vector

Remaining Questions

Our current recipe:

- Recognize kernelized problem: $\psi(x)$ only occur in inner products $\psi(x)^T\psi(x')$
- Pick a kernel function ("similarity score")
- Compute the kernel matrix (n by n where n is the dataset size)
- Optimize the model and make predictions by accessing the kernel matrix

Next: When can we apply kernelization?

SVM solution is in the "span of the data"

- We found the SVM dual problem can be written as:

$$\begin{aligned}\sup_{\alpha\in\mathbf{R}^n} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j y_i y_j x_j^T x_i \\ \text{s.t.} \quad & \sum_{i=1}^n \alpha_i y_i = 0 \\ & \alpha_i \in \left[0, \frac{c}{n}\right] \;\; i = 1,\ldots,n.\end{aligned}$$

- Given dual solution α^*, primal solution is $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$.
- Notice: w^* is a linear combination of training inputs $x_1,\ldots,x_n$.
- We refer to this phenomenon by saying "w^* is in the **span of the data**."
  - Or in math, $w^* \in \operatorname{span}(x_1,\ldots,x_n)$.

Ridge regression solution is in the "span of the data"

- The ridge regression solution for regularization parameter $\lambda > 0$ is

$$w^* = \underset{w \in \mathbf{R}^d}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_2^2.$$

- This has a closed form solution:

$$w^* = \left(X^T X + \lambda I \right)^{-1} X^T y,$$

where X is the design matrix, with $x_1, \ldots, x_n$ as rows.

Ridge regression solution is in the "span of the data"

- Rearranging $w^* = \left(X^TX + \lambda I\right)^{-1} X^Ty$, we can show that:

$$\begin{aligned} w^* &= X^T \underbrace{\left(\frac{1}{\lambda}y - \frac{1}{\lambda}Xw^*\right)}_{\alpha^*} \\ &= X^T\alpha^* = \sum_{i=1}^{n} \alpha_i^* x_i. \end{aligned}$$

- So w^* is in the span of the data.
  - i.e. $w^* \in \operatorname{span}\left(x_1, \ldots, x_n\right)$

If solution is in the span of the data, we can reparameterize

- The ridge regression solution for regularization parameter $\lambda > 0$ is

$$w^* = \operatorname*{arg\,min}_{w \in \mathrm{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_2^2.$$

- We now know that $w^* \in \operatorname{span}(x_1, \ldots, x_n) \subset \mathrm{R}^d$.
- So rather than minimizing over all of R^d, we can minimize over $\operatorname{span}(x_1, \ldots, x_n)$.

$$w^* = \operatorname*{arg\,min}_{w \in \operatorname{span}(x_1, \ldots, x_n)} \frac{1}{n} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_2^2.$$

- Let's reparameterize the objective by replacing w as a linear combination of the inputs.

If solution is in the span of the data, we can reparameterize

- Note that for any $w \in \operatorname{span}(x_1, \ldots, x_n)$, we have $w = X^T \alpha$, for some $\alpha \in \mathbf{R}^n$.
- So let's replace w with $X^T \alpha$ in our optimization problem:

$$\begin{aligned}
[\text{original}]\ w^* &= \underset{w \in \mathbf{R}^d}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \left\{ w^T x_i - y_i \right\}^2 + \lambda \|w\|_2^2 \\
[\text{reparameterized}]\ \alpha^* &= \underset{\alpha \in \mathbf{R}^n}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \left\{ \left(X^T \alpha \right)^T x_i - y_i \right\}^2 + \lambda \|X^T \alpha\|_2^2.
\end{aligned}$$

- To get w^* from the reparameterized optimization problem, we just take $w^* = X^T \alpha^*$.
- We changed the dimension of our optimization variable from d to n. Is this useful?

Consider very large feature spaces

- Suppose we have a 300-million dimension feature space [very large]
    - (e.g. using high order monomial interaction terms as features, as described last lecture)
- Suppose we have a training set of 300,000 examples [fairly large]
- In the original formulation, we solve a 300-million dimension optimization problem.
- In the reparameterized formulation, we solve a 300,000-dimension optimization problem.
- This is why we care about when the solution is in the span of the data.
- This reparameterization is interesting when we have more features than data ($d \gg n$).

More General

- For SVM and ridge regression, we found that the solution is in the span of the data.
- The Representer Theorem shows that this "span of the data" result occurs far more generally.

The Representer Theorem (Optional)

- **Generalized objective**:

$$w^* = \operatorname*{arg\,min}_{w \in \mathcal{H}} R\left(\|w\|\right) + L\left(\langle w, x_1\rangle, \ldots, \langle w, x_n\rangle\right)$$

- Representer theorem tells us we can look for w^* in the span of the data:

$$w^* = \operatorname*{arg\,min}_{w \in \operatorname{span}(x_1,\ldots,x_n)} R\left(\|w\|\right) + L\left(\langle w, x_1\rangle, \ldots, \langle w, x_n\rangle\right).$$

- So we can reparameterize as before:

$$\alpha^* = \operatorname*{arg\,min}_{\alpha \in \mathbf{R}^n} R\left(\left\|\sum_{i=1}^n \alpha_i x_i\right\|\right) + L\left(\left\langle \sum_{i=1}^n \alpha_i x_i, x_1 \right\rangle, \ldots, \left\langle \sum_{i=1}^n \alpha_i x_i, x_n \right\rangle\right).$$

- Our reparameterization trick applies much more broadly than SVM and ridge.

Summary

- We formualte the kernelized verions of SVM and ridge regression.
- Many other algorithms can be kernelized.
- Our principled tool for kernelization is reparameterization by the representer theorem.
- Representer theorem says that all norm-regularized linear models can be kernelized.
- Once kernelized, we can apply the kernel trick: doesn't need to represent $\phi(x)$ explicitly.

Overview

Why probabilistic modeling?

- A unified framework that covers many models, e.g., linear regression, logistic regression
- Learning as **statistical inference**
- Principled ways to incorporate your belief on the data generating distribution (inductive biases)

Two ways of generating data

- Two ways to model how the data is generated:
  - **Conditional**: $p(y \mid x)$
  - **Generative**: $p(x, y)$
- How to estimate the parameters of our model? Maximum likelihood estimation.
- Compare and contrast conditional and generative models.

Conditional models

Linear regression

Linear regression is one of the most important methods in machine learning and statistics.

Goal: Predict a real-valued **target** y (also called response) from a vector of **features** x (also called covariates).

Examples:

- Predicting house price given location, condition, build year etc.
- Predicting medical cost of a person given age, sex, region, BMI etc.
- Predicting age of a person based on their photos.

Problem setup

Data Training examples $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$.

Model A *linear* function h (parametrized by θ) to predict y from x:

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x, \tag{1}$$

where $\theta \in \mathbb{R}^d$ are the **parameters** (also called weights).

Note that

- We incorporate the **bias term** (also called the intercept term) into x (i.e. $x_0 = 1$).
- We use superscript to denote the example id and subscript to denote the dimension id.

Parameter estimation

Loss function We estimate θ by minimizing the **squared loss** (the least square method):

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)} - \theta^T x^{(n)}\right)^2. \quad \text{(empirical risk)} \tag{2}$$

Matrix form

- Let $X \in \mathbb{R}^{N \times d}$ be the **design matrix** whose rows are input features.
- Let $\mathrm{y} \in \mathbb{R}^N$ be the vector of all targets.
- We want to solve

$$\hat{\theta} = \arg\min_{\theta} (X\theta - \mathrm{y})^T (X\theta - \mathrm{y}). \tag{3}$$

Solution Closed-form solution: $\hat{\theta} = (X^T X)^{-1} X^T \mathrm{y}$.

Review questions

- Derive the solution for linear regression.
- What if $X^T X$ is not invertible?

Review

We've seen

- Linear regression: response is a linear function of the inputs
- Estimate parameters by minimize the squared loss

But...

- Why squared loss is a reasonable choice for regression problems?
- What assumptions are we making on the data? (inductive bias)

Next,

- Derive linear regression from a probabilistic modeling perspective.

Assumptions in linear regression

- x and y are related through a linear function:

$$y = \theta^T x + \epsilon, \tag{4}$$

where ϵ is the **residual error** capturing all unmodeled effects (e.g., noise).

- The errors are distributed *iid* (independently and identically distributed):

$$\epsilon \sim \mathcal{N}(0, \sigma^2). \tag{5}$$

What's the distribution of $Y \mid X = x$?

$$p(y \mid x; \theta) = \mathcal{N}(\theta^T x, \sigma^2). \tag{6}$$

Imagine putting a Gaussian bump around the output of the linear predictor.

Maximum likelihood estimation (MLE)

Given a probabilistic model and a dataset $\mathcal{D}$, how to estimate the model parameters θ?

The **maximum likelihood principle** says that we should maximize the (conditional) likelihood of the data:

$$
\begin{aligned}
L(\theta) &\overset{\text{def}}{=} p(\mathcal{D};\theta) && (7)\\
&= \prod_{n=1}^{N} p(y^{(n)} \mid x^{(n)};\theta). && \text{(examples are distributed } \textit{iid}\text{)} \quad (8)
\end{aligned}
$$

In practice, we maximize the log **likelihood** $\ell(\theta)$, or equivalently, minimize the negative log likelihood (NLL).

MLE for linear regression

Let's find the MLE solution for our model. Recall that $Y \mid X = x \sim \mathcal{N}(\theta^T x, \sigma^2)$.

$$\begin{aligned}
\ell(\theta) &\overset{\text{def}}{=} \log L(\theta) && (9)\\
&= \log \prod_{n=1}^{N} p(y^{(n)} \mid x^{(n)}; \theta) && (10)\\
&= \sum_{n=1}^{N} \log p(y^{(n)} \mid x^{(n)}; \theta) && (11)\\
&= \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\left(y^{(n)} - \theta^T x^{(n)}\right)^2}{2\sigma^2}\right) && (12)\\
&= {\color{blue} N \log \frac{1}{\sqrt{2\pi}\sigma}} {\color{red} - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y^{(n)} - \theta^T x^{(n)}\right)^2} && (13)
\end{aligned}$$