gets very slow very fast

The kernel Trick

recall polynomial features: w/ d input features degree-p polynomials blow up to $O(d^p)$ features

e.g. $p=4$, features blown up to d^4

Today, we use those features w/o computing them

key observations: In many algorithms,

- weights are linear combo of the sample points
- can use inner products of lifted vectors $\Phi(\vec{x})$ only – don't need each $\Phi(\vec{x})$ explicitly computed

$$\vec{w} = X^T\vec{a} = \sum_{i=1}^{n} a_i x_i \quad , \vec{a} \in \mathbb{R}^n$$

each comp. of $\vec{a}$ is a dual weight

If this is true for an algorithm, plug in $\vec{w} = X^T\vec{a}$ into the algo.

↳ get rid of $\vec{w}$, optimize the n dual weights $\vec{a}$ (dual parameters) instead of the $d+1$ (or d^p if polynomial features) primal weights of $\vec{w}$

kernel ridge Regression

Problem: Impossible to write weights as lin. combo. of data if we have a bias term

↳ λ needs to penalize bias too

↳ to mitigate this effect, center X & $\vec{y}$ → mean of 0 (col. of 1's appended)

↳ $x_i \leftarrow x_i - \mu_x$, $y_i \leftarrow y_i - \mu_y$, $x_{i,d+1} = 1$

- This lets us replace I' w/ I in normal equations:

$$(X^TX + \lambda I)\vec{w} = X^T\vec{y}$$ primal

↳ I' is w/ bias term

Suppose $\vec{a}$ is a solution to: $(XX^T + \lambda I)\vec{a} = \vec{y}$ dual

$$\Rightarrow X^T\vec{y} = X^TXX^T\vec{a} + \lambda X^T\vec{a} = (X^TX + \lambda I)X^T\vec{a}$$

(lin. combo of sample pts.)

(dual solution) $\Rightarrow \vec{a}$ sol as long as $\vec{w} = X^T\vec{a}$

Primal: $\|Xw-y\|^2 + \lambda\|w\|^2$

↓

dual form of ridge regression: $\min_{\vec{a}} \|XX^T\alpha - \vec{y}\|^2 + \lambda\|X^T a\|^2$

training part of algo: Solve $(XX^T + \lambda I)\vec{a} = \vec{y}$ for $\vec{a}$ ← big if using llisted features

Testing: Regression function is: $h(\vec{z}) = \vec{w}^T\vec{z} = \vec{a}^T(X)\vec{z} = \sum_{i=1}^{n} a_i(x_i^T\vec{z})$ (inner prod.)

Weighted sum of inner products

define $k(\vec{x}, \vec{z})$ as the kernel fn.

For now, $k(\vec{x}, \vec{z}) = \vec{x}^T\vec{z}$. Later, $\vec{x} = \Phi(\vec{x})$, $\vec{z} = \Phi(\vec{z})$

let $K = XX^T$ be the $n \times n$ kernel matrix, $K_{ij} = k(x_i, x_j)$ always PSD

Singular K if $n > d+1$ (Sometimes still singular even if this isn't the case)
↳ tall X
↳ almost always true w/ large datasets
↳ choose $\lambda > 0$ to solve system

d^2 if no kernel trick

dual ridge regression algorithm:

training
1. Compute the kernel matrix one entry at a time: $K_{ij} \leftarrow k(\mathbf{X}_i, \mathbf{X}_j)$
2. Solve $(\mathbf{K}+\lambda\mathbf{I})\vec{a} = \vec{y}$ for $\vec{a}$ $O(n^3)$ time

testing
3. for each test point $\vec{z}$:
4. $h(\vec{z}) = \sum_{i=1}^{n} a_i k(\mathbf{X}_i, \vec{z})$ $O(nd)$ time

$O(n^2 d)$ time

* again, same sol. as primal ridge regr. algo (if bias penalized)

Notice: Sample pts. only used as inner products

Dual: Solve $n \times n$ linear system $O(n^3 + n^2 d)$ time ← preferred when $d > n$

Primal: solve $d \times d$ linear system $O(d^3 + d^2 n)$ time

advantage of dual: when using polynomial features, d in primal algo increases but d in dual won't

The kernel Trick / kernelization: We can compute a polynomial kernel that involves many monomial terms w/o ever computing those terms

Polynomial kernel of degree p: $k(\vec{x}, \vec{z}) = (\vec{x}^T \vec{z} + 1)^p$

Theorem: $(\vec{x}^T \vec{z} + 1)^p = \Phi(\vec{x})^T \Phi(\vec{z})$, $\Phi(\vec{x})$ is lifting fn that contains every monomial term w.r.t. $\vec{x}$ of degree $0 \ldots p$

e.g. $d=2$, $p=2$

$$\Rightarrow (\vec{x}^T \vec{z} + 1) = x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_2 + 2x_1 z_1 + 2x_2 z_2 + 1$$

$O(d)$ $O(1)$

$$= \begin{bmatrix} x_1^2 & z_1^2 & \sqrt{2}\, x_1 x_2 & \sqrt{2}\, x_1 & \sqrt{2}\, x_2 & 1 \end{bmatrix} \begin{bmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}\, z_1 z_2 \\ \sqrt{2}\, z_1 \\ \sqrt{2}\, z_2 \\ 1 \end{bmatrix}$$

Can't control the constants, but that's okay

$$= \Phi(\vec{x})^T \Phi(\vec{z})$$

↳ When optimized, primal weights will automatically adjust themselves to compensate for the constants

Key Win: Compute $\Phi(\vec{x})^T \Phi(\vec{z})$ in $O(d)$ time instead of $O(d^p)$ time even though $\Phi(x)$ & $\Phi(z)$ have length $O(d^p)$

Kernel ridge regr. w/ polynomial features: replace x_i w/ $\Phi(\vec{x}_i)$:

let $k(\vec{x}, \vec{z}) = \Phi(\vec{x})^T \Phi(\vec{z}) = (\vec{x}^T \vec{z} + 1)^p$

NEVER COMPUTED

Kernel Perceptrons:

featurized perceptron algo:

train: $w \leftarrow y_1 \Phi(\vec{x}_1)$; while some $y_i \Phi(\vec{x}_i) \cdot \vec{w} < 0$: $w \leftarrow w + \varepsilon y_i \Phi(\vec{x}_i)$ — Primal

test: for each test pt. $\vec{z}$: $h(\vec{z}) = \vec{w}^T \Phi(\vec{z})$

Let $\Phi(X)$ be the $n \times D$ matrix w/ rows $\Phi(\vec{x}_i)^T$, D = length of $\Phi(\vec{x}_i)$

$\Rightarrow K = \Phi(X)\Phi(X)^T$

→ Dualize w/ $\vec{w} = \Phi(X)^T \vec{a}$

↳ Then the code "$a_i \leftarrow a_i + \varepsilon y_i$" $\equiv \vec{w} \leftarrow \vec{w} + \varepsilon y_i \Phi(\vec{x}_i)$ (dual / primal)

also, $\vec{w} = I(X)^T \vec{a}$

$\rightarrow I(X_i)^T \vec{w} = (I(X)\vec{w})_i = (I(X)I(X)^T \vec{a})_i = (K\vec{a})_i$

⇒ Dual Perceptron Algo:

```
1  a ← [y_1  0  ···  0]^T
2  Compute the kernel matrix K_ij ← k(X_i, X_j)   O(n^2 d) time (kernel trick)
3  while some y_i (Ka)_i < 0:
4      a_i ← a_i + ε y_i   O(1)
5
6  for each test point z
7      h(z) ← Σ_{j=1}^n a_j k(X_j, z)   O(nd) time (kernel trick)
```

Kernel Logistic Regression

Sigmoid

Stochastic gradient descent Step:

$$a_i \leftarrow a_i + \varepsilon(y_i - S((Ka)_i))$$

Batch gradient descent: $a \leftarrow a + \varepsilon(y - S(Ka))$

applied component-wise to vector

for each test pt. $\vec{z}$

$$h(z) = S\left(\sum_{j=1}^{n} a_j k(x_j, \vec{z})\right)$$

$S(\cdot)$ for posterior probabilities

Gaussian kernel: use feature vectors in an infinite-dimensional space

gaussian kernel based on observation that $\exists\ \Phi(\vec{x})$ such that

$$k(\vec{x}, \vec{z}) = \exp\left(-\frac{\|\vec{x}-\vec{z}\|^2}{2\sigma^2}\right) = \Phi(\vec{x})^T \Phi(\vec{z})$$

isotropic, centered at $\vec{z}/\vec{x}$

e.g. $d = 1$

$$\Phi(\vec{x}) = \exp\left(\frac{-x^2}{2\sigma^2}\right)\left[1 \quad \frac{x}{\sigma\sqrt{1!}} \quad \frac{x^2}{\sigma^2\sqrt{2!}} \quad \cdots\right]^T$$

$$\Phi(x)^T \Phi(\vec{z}) = k(\vec{x}, \vec{z})$$

more helpful to think in terms of fn $k(\cdot)$ rather than $\Phi(\cdot)$

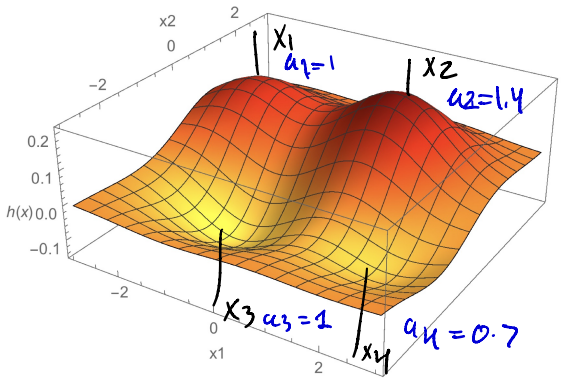
key observation: hypothesis $h(z) = \sum_{j=1}^{n} a_j k(x_j, z)$ is a lin. combo of Gaussians centered at sample pts.

dual weights are coefficients

basis for hypothesis

4 sample pts.

- height dictated by coeff. a_i;



Gaussian kernel popular in practice, but why?

- gives very smooth regression fn. $h(\cdot)$; infinitely differentiable, C^∞ continuous
- $h(\cdot)$ behaves somewhat like k-nearest-neighbors, but smoother
- oscillates less than polynomials (depending on σ)
- $k(\vec{x}, \vec{z})$ can be interpreted as a similarity measure
  - ↳ max when $\vec{z} = \vec{x}$
  - ↳ goes to 0 as distance increases
- Sample pts. "vote" for value at $\vec{z}$, but closer pts. get more weight of vote

σ is an h-param chosen through validation, trades off B vs. V

- larger σ → wider Gaussians → smoother $h(\cdot)$ → more bias (fn. can't adapt to data)
  → less variance