16 The Kernel Trick

KERNELS

Recall featurizing map $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$. d input features; D features after featurization (Φ).
Degree-p polynomials blow up to $D \in \Theta(d^p)$ features.
[When d and p are not small, this gets computationally intractable really fast. As I said in Lecture 4, if you have 100 features per feature vector and you want to use degree-4 polynomial decision functions, then each featurized feature vector has a length of roughly 4 million.]
Today, magically, we use those features without computing them!

Observation: In many learning algs,
- the weights can be written as a linear combo of training points, &
- we can use inner products of $\Phi(x)$'s only $\Rightarrow$ don't need to compute $\Phi(x)$!

Suppose $w = X^\top a = \sum_{i=1}^{n} a_i X_i$ for some $a \in \mathbb{R}^n$.
Substitute this identity into alg. and optimize n dual weights a (aka dual parameters) instead of D primal weights w.

Kernel Ridge Regression

Center X and y so their means are zero: $X_i \leftarrow X_i - \mu_X$, $y_i \leftarrow y_i - \mu_y$, $X_{i,d+1} = 1$ [don't center the 1's!]
This lets us replace I' with I in normal equations:

$$(X^\top X + \lambda I)w = X^\top y.$$

[To kernelize ridge regression, we need the weights to be a linear combination of the training points. Unfortunately, that only happens if we penalize the bias term $w_{d+1} = \alpha$, as these normal equations do. Fortunately, when we center X and y, the "expected" value of the bias term is zero. The actual bias won't usually be exactly zero, but it will often be close enough that we won't do much harm by penalizing the bias term.]

Suppose a is a solution to

$$(XX^\top + \lambda I)a = y. \qquad \text{[Always has a solution if } \lambda > 0.]$$

Then $X^\top y = X^\top XX^\top a + \lambda X^\top a = (X^\top X + \lambda I)X^\top a$.
Therefore, $w = X^\top a$ is a solution to the normal equations, **and** w is a linear combo of training points!

a is a dual solution; solves the dual form of ridge regression:

$$\boxed{\text{Find } a \text{ that minimizes } \|XX^\top a - y\|^2 + \lambda \|X^\top a\|^2.}$$

[We obtain this dual form by substituting $w = X^\top a$ into the original ridge regression cost function.]

Training: Solve $(XX^\top + \lambda I)a = y$ for a.
Testing: Regression fn is

$$h(z) = w^\top z = a^\top X z = \sum_{i=1}^{n} a_i \, (X_i^\top z) \qquad \Leftarrow \text{ weighted sum of inner products}$$

Let $k(x, z) = x^\top z$ be kernel fn.
[Later, we'll replace x and z with $\Phi(x)$ and $\Phi(z)$, and that's where the magic will happen.]

Let $K = XX^\top$ be $n \times n$ kernel matrix. Note $K_{ij} = k(X_i, X_j)$.

K may be singular. If so, probably no solution if $\lambda = 0$. [Then we must choose a positive λ. But that's okay.] Always singular if $n > d + 1$. [But don't worry about the case $n > d + 1$, because you would only want to use the dual form when $d > n$, i.e., for polynomial features. But K could still be singular when $d > n$.]

Dual/kernel ridge regr. alg:

```
$\forall i, j,$   $K_{ij} \leftarrow k(X_i, X_j)$            $\Leftarrow O(n^2 d)$ time
Solve $(K + \lambda I)\, a = y$ for $a$          $\Leftarrow O(n^3)$ time
for each test pt $z$
    $h(z) \leftarrow \sum_{i=1}^n a_i\, k(X_i, z)$        $\Leftarrow O(nd)$ time/test pt
```

Does not use X_i directly! Only k. [This will become important soon.]

[**Important**: dual ridge regression produces the same predictions as primal ridge regression (with a penalized bias term)! The difference is the running time; the dual algorithm is faster if $d > n$, because the primal algorithm solves a $d \times d$ linear system, whereas the dual algorithm solves an $n \times n$ linear system.]

The Kernel Trick (aka Kernelization)

[Here's the magic part. We can compute a polynomial kernel without actually computing the features.]

The polynomial kernel of degree p is $k(x, z) = (x^\top z + 1)^p$.

Theorem: $(x^\top z + 1)^p = \Phi(x)^\top \Phi(z)$ for some $\Phi(x)$ containing every monomial in x of degree $0 \ldots p$.

Example for $d = 2$, $p = 2$:

$$\begin{aligned}(x^\top z + 1)^2 &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\ &= [x_1^2 \quad x_2^2 \quad \sqrt{2}x_1x_2 \quad \sqrt{2}x_1 \quad \sqrt{2}x_2 \quad 1]\,[z_1^2 \quad z_2^2 \quad \sqrt{2}z_1z_2 \quad \sqrt{2}z_1 \quad \sqrt{2}z_2 \quad 1]^\top \\ &= \Phi(x)^\top \Phi(z) \qquad \text{[This is how we're defining } \Phi.]\end{aligned}$$

[Notice the factors of $\sqrt{2}$. If you try a higher polynomial degree p, you'll see a wider variety of these constants. We have no control of the constants that appear in $\Phi(x)$, but they don't matter much, because the primal weights w will scale themselves to compensate. Even though we don't directly compute the primal weights, they implicitly exist in the form $w = X^\top a$.]

Key win: compute $\Phi(x)^\top \Phi(z)$ in $O(d)$ time instead of $O(D) = O(d^p)$, even though $\Phi(x)$ has length D.

Kernel ridge regr. replaces X_i with $\Phi(X_i)$: let $k(x, z) = \Phi(x)^\top \Phi(z)$,
but doesn't compute $\Phi(x)$ or $\Phi(z)$; it computes $k(x, z) = (x^\top z + 1)^p$.

Running times for 3 ridge algs:

	primal	dual (no kernel trick)	kernel
train	$O(D^3 + D^2 n)$	$O(n^3 + n^2 D)$	$O(n^3 + n^2 d)$
test (per test pt)	$O(D)$	$O(nD)$	$O(nd)$

[I think what we've done here is pretty mind-blowing: we can now do polynomial regression with an exponentially long, high-order polynomial in less time than it would take even to write out the final polynomial. The running time can be asymptotically smaller than D, the number of terms in the polynomial.]

Kernel Logistic Regression

Let $\Phi(X)$ be $n \times D$ matrix with rows $\Phi(X_i)^\top$. [$\Phi(X)$ is the design matrix of the featurized training points.]

Featurized logi. regr. with batch grad. descent:

```
$w \leftarrow 0$                                          [starting point is arbitrary]
repeat until convergence
    $w \leftarrow w + \epsilon\, \Phi(X)^\top (y - s(\Phi(X)\, w))$        apply $s$ component-wise to vector $\Phi(X)\, w$
for each test pt $z$
    $h(z) \leftarrow s(w^\top \Phi(z))$
```

Dualize with $w = \Phi(X)^\top a$.
Then the code "$a \leftarrow a + \epsilon\, (y - s(\Phi(X)\, w))$" has same effect as "$w \leftarrow w + \epsilon\, \Phi(X)^\top (y - s(\Phi(X)\, w))$".
Let $K = \Phi(X)\, \Phi(X)^\top$. [The $n \times n$ kernel matrix; but we don't compute $\Phi(X)$—we use the kernel trick.]
Note that $Ka = \Phi(X)\, \Phi(X)^\top a = \Phi(X)\, w$. [And $\Phi(X)\, w$ appears in the algorithm above.]

Dual/kernel logistic regression:

```
$a \leftarrow 0$                                          [starting point is arbitrary]
$\forall i, j, \quad K_{ij} \leftarrow k(X_i, X_j)$                  $\Leftarrow O(n^2 d)$ time (kernel trick)
repeat until convergence
    $a \leftarrow a + \epsilon\, (y - s(Ka))$                    $\Leftarrow O(n^2)$ time/iteration [apply $s$ component-wise]
for each test pt $z$
    $h(z) \leftarrow s\left(\sum_{i=1}^{n} a_i\, k(X_i, z)\right)$          $\Leftarrow O(nd)$ time/test pt [kernel trick]
```

[For classification, you can skip the logistic function $s(\cdot)$ and just compute the sign of the summation.]

[Kernel logistic regression computes the same answer as the primal algorithm, but the running time changes.]
Important: running times depend on original dimension d, not on length D of $\Phi(\cdot)$! Training for j iterations:
Primal: $O(nDj)$ time Dual (no kernel trick): $O(n^2 D + n^2 j)$ time Kernel: $O(n^2 d + n^2 j)$ time

Alternative training: stochastic gradient descent (SGD). Primal logistic SGD step is

$$w \leftarrow w + \epsilon\, (y_i - s(\Phi(X_i)^\top w))\, \Phi(X_i).$$

Dual logistic SGD maintains a vector $q = Ka \in \mathbb{R}^n$. Note that $q_i = (\Phi(X)\, w)_i = \Phi(X_i)^\top w$.
Let K_{*i} denote column i of K.

```
$a \leftarrow 0$; $q \leftarrow 0$; $\forall i, j, K_{ij} \leftarrow k(X_i, X_j)$      [If you choose a different starting point, set $q \leftarrow Ka$.]
repeat until convergence
    choose random $i \in [1, n]$
    $a_i \leftarrow a_i + \epsilon\, (y_i - s(q_i))$                 $\Leftarrow O(1)$ time
    $q \leftarrow q + \epsilon\, (y_i - s(q_i))\, K_{*i}$              $\Leftarrow$ computes $q = Ka$ in $O(n)$ time, *not* $O(n^2)$ time
```

[SGD updates only one dual weight a_i per iteration; that's a nice benefit of the dual formulation. We cleverly update $q = Ka$ in linear time instead of performing a quadratic-time matrix-vector multiplication.]

Primal: $O(Dj)$ time Dual (no kernel trick): $O(n^2 D + nj)$ time Kernel: $O(n^2 d + nj)$ time

Alternative testing: If # of training points and test points both exceed D/d, classifying with primal weights w may be faster. [This appies to ridge regression as well.]

```
$w = \Phi(X)^\top a$                                      $\Leftarrow O(nD)$ time (once only)
for each test pt $z$
    $h(z) \leftarrow s\,(w^\top \Phi(z))$                          $\Leftarrow O(D)$ time/test pt
```

The Gaussian Kernel

[Mind-blowing as the polynomial kernel is, I think our next trick is even more mind-blowing. Since we can now do fast computations in spaces with exponentially large dimensions, why don't we go all the way and generate feature vectors in an infinite-dimensional space?]

Gaussian kernel, aka radial basis fn kernel: there exists a $\Phi : \mathbb{R}^d \to \mathbb{R}^\infty$ such that

$$k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$ [This kernel takes $O(d)$ time to compute.]

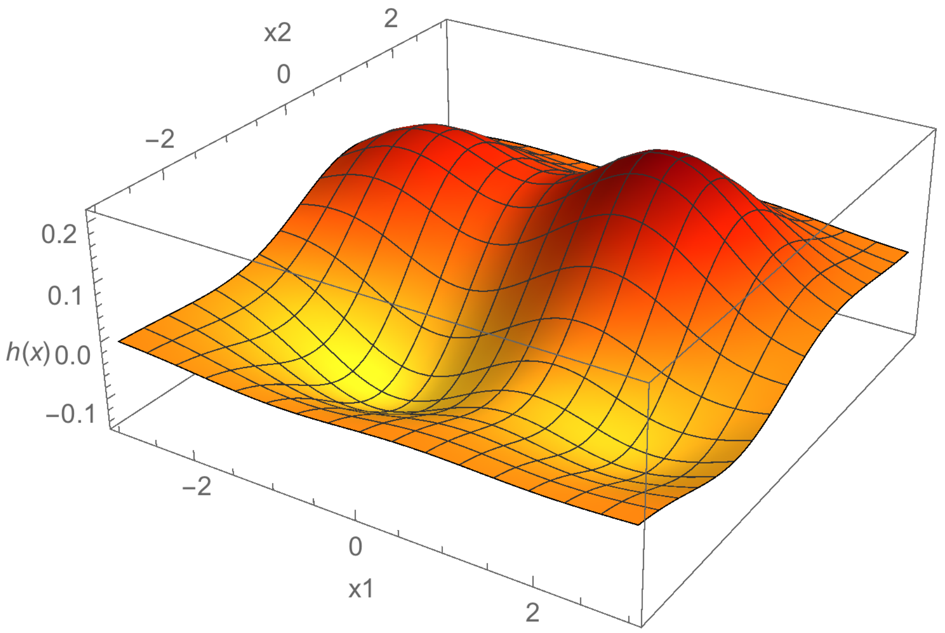
[In case you're curious, here's the feature vector that gives you this kernel, for the case where you have only one input feature per sample point.]

e.g., for $d = 1$,

$$\Phi(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right)\left[1, \frac{x}{\sigma\sqrt{1!}}, \frac{x^2}{\sigma^2\sqrt{2!}}, \frac{x^3}{\sigma^3\sqrt{3!}}, \ldots\right]^\top$$

[This is an infinite vector, and $\Phi(x)\cdot\Phi(z)$ is a series that converges to $k(x,z)$. Nobody actually uses this value of $\Phi(x)$ directly, or even cares about it; they just use the kernel function $k(\cdot,\cdot)$.]
[At this point, it's best *not* to think of points in a high-dimensional space. It's no longer a useful intuition. Instead, think of the kernel k as a measure of how similar or close together two points are to each other.]

Key observation: hypothesis $h(z) = \sum_{j=1}^n a_j\, k(X_j, z)$ is a linear combo of Gaussians centered at training pts.
[The dual weights are the coefficients of the linear combination.]
[The Gaussians are a basis for the hypothesis.]



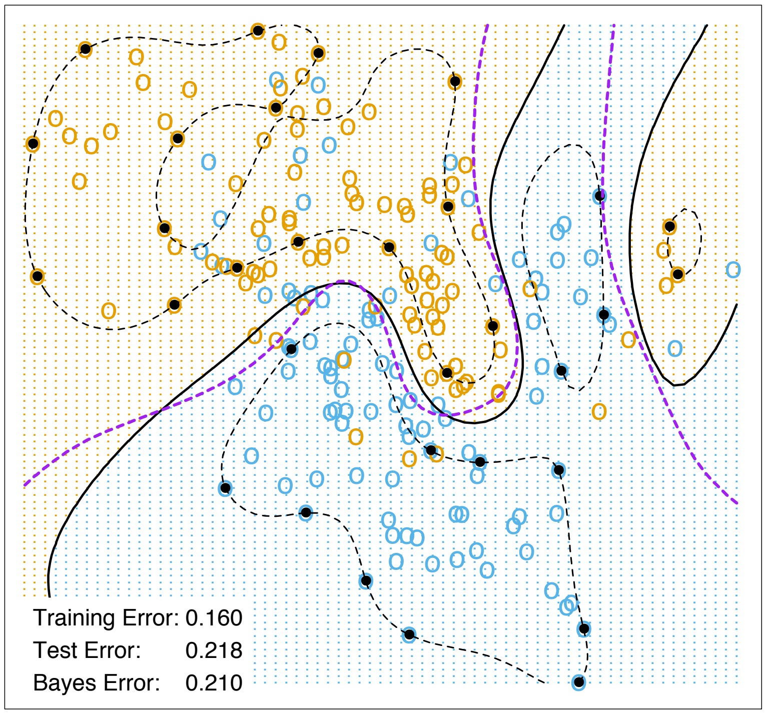
gausskernel.pdf [A hypothesis h that is a linear combination of Gaussians centered at four training points, two with positive weights and two with negative weights. If you use ridge regression with a Gaussian kernel, your "linear" regression will look something like this.]

Very popular in practice! Why?

- Gives very smooth h [In fact, h is infinitely differentiable; it's C^∞-continuous.]
- Behaves somewhat like k-nearest neighbors, but smoother
- Oscillates less than polynomials (depending on σ)
- $k(x, z)$ interpreted as a similarity measure. Maximum when $z = x$; goes to 0 as distance increases.
- Training points "vote" for value at z, but closer points get weightier vote.

[The "standard" kernel $k(x, z) = x \cdot z$ assigns more weight to training point vectors that point in roughly the same direction as z. By contrast, the Gaussian kernel assigns more weight to training points near z.]

Choose σ by validation.
σ trades off bias vs. variance:
larger σ $\rightarrow$ wider Gaussians & smoother h $\rightarrow$ more bias & less variance



gausskernelsvm.pdf (ESL, Figure 12.3) [The decision boundary (solid black) of a soft-margin SVM with a Gaussian kernel. Observe that in this example, it comes reasonably close to the Bayes optimal decision boundary (dashed purple). The dashed black curves are the boundaries of the margin. The small black disks are the support vectors that lie on the margin boundary.]

[By the way, there are many other kernels that, like the Gaussian kernel, are defined directly as kernel functions without worrying about Φ. But not every function can be a kernel function. A function is qualified only if it always generates a positive semidefinite kernel matrix, for every sample. There is an elaborate theory about how to construct valid kernel functions. However, you probably won't need it. The polynomial and Gaussian kernels are the two most popular by far.]

[As a final word, be aware that not every featurization Φ leads to a kernel function that can be computed faster than $\Theta(D)$ time. In fact, the vast majority cannot. Featurizations that can are rare and special.]