9/19/2023: Kernels Continued

$K(x, x')$ measures similarity between x & x'

$\underbrace{x}, \underbrace{x'}$: inputs to model

Similar pairs (x, x') should have large $K(x, x')$

Idea: make prediction by computing $\sum_{i=1}^{n} \alpha_i K(x^{(i)}, x^{test})$

- α_i: learned parameter
- $x^{(i)}$: ith training example
- x^{test}: test example

If >0 $\to$ predict 1

If <0 $\to$ predict -1

logistic regression already does this (in a way) if you define

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

Original L.R

Training: $w^{(0)} \leftarrow \underline{0}$

$$w^{(t)} \leftarrow w^{(t-1)} + \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \underbrace{\sigma(-y^{(i)} w^{(t-1)T} x^{(i)}) \cdot y^{(i)}}_{\text{Scalar}} \cdot x^{(i)}$$

Test time: given x^{test}

Compute $w^{(final)T} x^{test}$

Kernelized L.R

Define $w = \sum_{i=1}^{n} \alpha_i x^{(i)}$

Training: $\alpha^{(0)} \leftarrow \underline{0}$

$$\alpha_i^{(t)} \leftarrow \alpha_i^{(t-1)} + \eta \cdot \frac{1}{n} \cdot \sigma(-y^{(i)} \boxed{w^{(t-1)T} x^{(i)}}) \cdot y^{(i)}$$

Do this for $i = 1, \dots, n$

$$= \sum_{j=1}^{n} \alpha_j^{(t-1)} \boxed{x^{(j)T} x^{(i)}} = K(x^{(j)}, x^{(i)})$$

Test time:

Compute $\sum_{i=1}^{n} \alpha_i^{(final)} x^{(i)T} x^{test}$

Why do this?

Run kernelized algorithm replacing dot products with $\underbrace{\text{other } K(x, x')}_{\text{hyperparameter you choose}}$

Kernels & Features

y	x_1	x_2
+1	2	3
-1	0	1
$\vdots$		

transform each row $\longrightarrow$

y	$\sqrt{2}\,x_1$	$\sqrt{2}\,x_2$	1	x_1^2	x_2^2	$\sqrt{2}\,x_1x_2$
+1	$2\sqrt{2}$	$3\sqrt{2}$	1	4	9	$6\sqrt{2}$
-1	0	$1\sqrt{2}$	1	0	1	0
$\vdots$			$\vdots$			

think of this as function $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^6$

Drawback: using ϕ + logistic regression is $\approx 3x$ slower

For some ϕ, you can quickly compute

$$K(x, x') = \phi(x)^T \phi(x')^T$$

without actually computing $\phi(x)$ or $\phi(x')$

"Kernel trick"

E.g. Quadratic Kernel:

$$K(x, x') = \underbrace{(x^T x' + 1)^2}_{\text{only requires operations on original vector space } \mathbb{R}^d} = \phi(x)^T \phi(x')$$

where

$$\phi(x) = \underbrace{\begin{bmatrix} 1 \\ \sqrt{2}\,x_1 \\ \vdots \\ \sqrt{2}\,x_d \\ x_1^2 \\ \vdots \\ x_d^2 \\ \sqrt{2}\,x_1x_2 \\ \vdots \\ \sqrt{2}\,x_{d-1}x_d \end{bmatrix}}_{\text{vector is size } O(d^2)}
\begin{array}{l} \} \text{ constant term} \\ \} \text{ All linear terms } \times \sqrt{2} \\ \} \text{ All } x_i^2 \text{ terms} \\ \} \text{ All } x_ix_j \text{ terms } \times \sqrt{2} \end{array}$$

More Generally: for degree p,

$$K(x, x') = (x^T x' + 1)^p = \phi(x)^T \phi(x')$$

for some ϕ that includes all monomials of degree $\leq p$

What about RBF?

Fact:

$$\exp\left(\frac{\|x - x'\|^2}{2\sigma^2}\right) = \phi(x)^T \phi(x')$$

for some $\phi(x)$ that is infinite-dimensional

Runtime Comparison: Let's use polynomial kernel of degree p

Original L.R.	Kernelized L.R.
• map each $x^{(i)}$ to $O(d^p)$-size feature vector	• Use kernel trick
• Training: 1 iteration takes $O(n d^p)$	• Training: 1 iteration takes $O(n^2 d)$
• Testing: takes $O(d^p)$	• Testing: compute $\sum_{i=1}^{n} \alpha_i K(x^{(i)}, x^{test})$ takes $O(nd)$
Better dependence on n	Better dependence on d & p
	Bad if dataset is very large

Support vector machines (SVM)

Similarities to Logistic Regression

- Binary classification
- Learn linear decision boundary
- Parameter is $w \in \mathbb{R}^d$
- Decision boundary defined by $w^T x = 0$

Difference: SVM has no probabilistic interpretation

SVM minimizes the following loss:

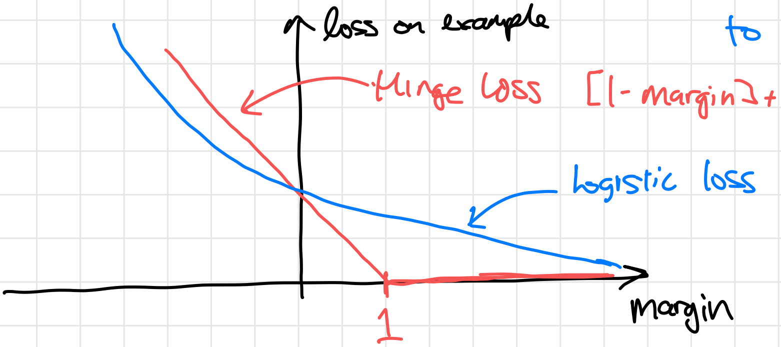
$$L(w) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\Big[1 - \underbrace{y^{(i)} w^T x^{(i)}}_{\text{margin}}\Big]_+}_{} + \underbrace{\lambda \|w\|^2}_{L_2 \text{ regularization}}$$

Applies "hinge loss" function $f(z) = [1 - z]_+$ to margin

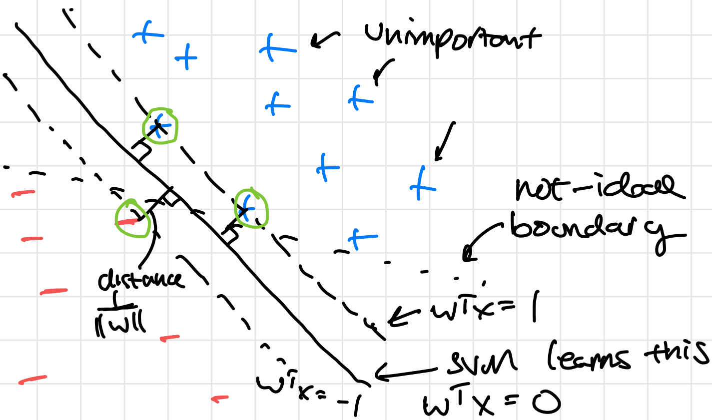
where $[z]_+ = z$ if $z > 0$

$\qquad\qquad\;\; 0$ if $z \le 0$

AKA $[z]_+ = \max(z, 0)$



What does hinge loss do?



SVM wants:

- 0 hinge loss
- Small $\|w\|$

$\iff$ large $\frac{1}{\|w\|}$

$\iff$ large distance between examples & decision boundary

Certain examples are <u>support vectors</u> (green)
in particular, ones where margin ≤ 1

Ideal w only depends on support vectors
i.e. it is a linear combination of support vectors only

<u>Connection to Kernels:</u>

We can kernelize SVM's,
ie write $w = \sum_{i=1}^{n} \underbrace{\alpha_i}_{} x^{(i)}$

$\alpha_i = 0$ if $x^{(i)}$ is not support vector

$\Rightarrow$ Test time: only evaluate #support vector kernel calls
instead of n

<u>Takeaway</u>: In practice, to use kernels, use SVM
don't use kernelized logistic regression