2/6/2024: Kernels Contd.

y	x_1	x_2
+1	2	3
-1	0	1

Simple dataset

transform each row $\longrightarrow$ to add features

y	x_1	x_2	1	x_1^2	x_2^2	x_1x_2
+1	2	3	1	4	9	6
-1	0	1	1	0	1	0
.	.	.	.	.	.	.

Dataset w/ more features

Let $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ be the function that does this transformation.

One drawback of adding more features: Slower runtime

For some ϕ, there are fast ways to compute

$K(x,z) = \phi(x)^T \phi(z)$ for any x, z

without computing $\phi(x)$ or $\phi(z)$ directly

E.g. Quadratic Kernel:

$$K(x,z) = \underbrace{(x^Tz+1)^2}_{} = \phi(x)^T\phi(z) \quad \text{where}$$

fast to compute using just x & z ie. $O(d)$

$$\phi(x) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_3 \\ \vdots \\ \sqrt{2}x_d \\ x_1^2 \\ \vdots \\ x_d^2 \\ \sqrt{2}x_1x_2 \\ \vdots \\ \sqrt{2}x_{d-1}x_d \end{bmatrix} \begin{matrix} \text{Constant term} \\ \\ \text{All linear terms} \\ \\ \\ \\ \text{All } x_j^2 \text{ terms} \\ \\ \\ \text{All } x_ix_j \text{ terms} \\ \\ \end{matrix}$$

has $O(d^2)$ entries

"Kernel trick" working with kernels faster than working with big feature vectors

Polynomial Kernel: For any degree p $(=2,3,4\dots)$

$$K(x,z) = \boxed{(x^T z + 1)^p} = \phi(x)^T \phi(z)$$

$O(d)$ to compute

where $\boxed{\phi(x)}$ is a really big feature vector with all monomials up to degree p

size $O(d^p)$

RBF Kernel:

Fact:

$$\exp\left(\frac{\|x-z\|^2}{2\sigma^2}\right) = \phi(x)^T \phi(z)$$

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

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

Original L.R.

- Map each $x^{(i)}$ to a $O(d^p)$-dim. vector $\phi(x^{(i)})$
- Training: 1 iteration takes $O(nd^p)$
- Testing: $O(d^p)$

(because dot products take $O(d^p)$)

Better dependence on n

Kernelized L.R.

- Use kernel trick
- Training: 1 iteration takes $O(n^2 \cdot d)$
- Testing: compute $\sum_{i=1}^{n} \alpha_i K(x, x^{(i)})$ takes $O(n \cdot d)$

Better dependence on p

Bad if very large dataset

Support Vector Machine (SVM)

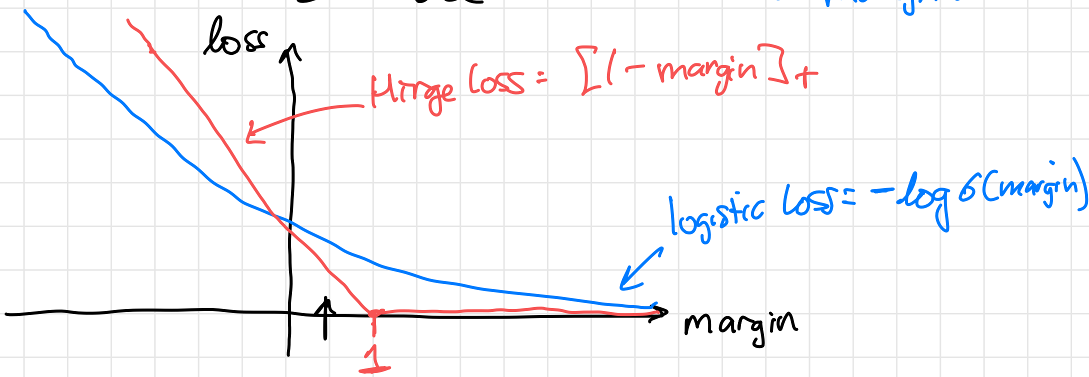
Similar to Logistic Regression
- Binary classification
- Learn linear decision boundary
- Parameter to learn is $w \in \mathbb{R}^d$
- Decision boundary defined by $w^T x = 0$

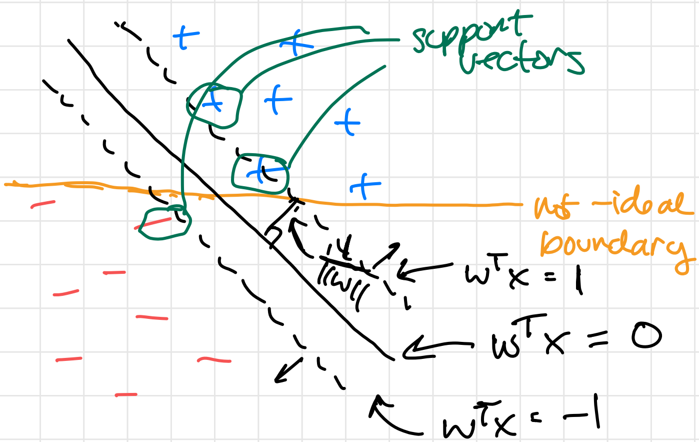
Differences:
- Different loss function
- No probabilistic story

SVM: Minimize the following loss:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left[1 - \underbrace{y^{(i)} w^T x^{(i)}}_{\text{margin}}\right]_+}_{\text{Called "hinge loss" of margin}} + \underbrace{\lambda \|w\|^2}_{\text{L2 regularization}}$$

where $[z]_+ = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{else} \end{cases}$





SVM wants:
- All points to have margin ≥ 1 (ie, far from decision boundary)
- $\|w\|$ to stay small $\implies \frac{1}{\|w\|}$ to be large

Fact: Decision boundary depends only on subset of data called support vectors.

= points with margin ≤ 1

$\Downarrow$

Ideal w is a linear combination only of support vectors

Connection to Kernels:

We can also kernelize SVMs, i.e.

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

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

Test-time: Instead of $O(n \cdot d)$,

cost is $O(\#\text{support vectors} \times d)$

Takeaway: To use kernels, use SVM