

CSCI567 Machine Learning (Fall 2025)

Haipeng Luo

University of Southern California

Sep 19, 2025

1 / 46

Outline

- 1 Review of Last Lecture
- 2 Multiclass Classification
- 3 Kernel methods

3 / 46

Administration

Will discuss HW1 solutions in today discussion session.

HW2 will be released next week.

2 / 46

Review of Last Lecture

Outline

- 1 Review of Last Lecture
- 2 Multiclass Classification
- 3 Kernel methods

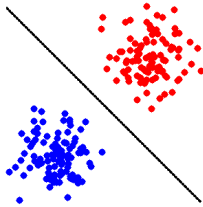
4 / 46

Linear classifiers

Linear models for **binary** classification:

Step 1. Model is the set of **separating hyperplanes**

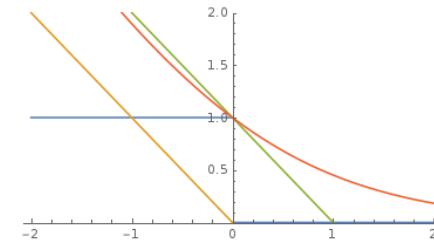
$$\mathcal{F} = \{f(x) = \text{sgn}(w^T x) \mid w \in \mathbb{R}^D\}$$



5 / 46

Linear classifiers

Step 2. Pick the **surrogate loss**



- **perceptron loss** $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- **hinge loss** $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$ (used in SVM and many others)
- **logistic loss** $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression)

6 / 46

Linear classifiers

Step 3. Find empirical risk minimizer (ERM):

$$w^* = \underset{w \in \mathbb{R}^D}{\text{argmin}} F(w) = \underset{w \in \mathbb{R}^D}{\text{argmin}} \frac{1}{N} \sum_{n=1}^N \ell(y_n w^T x_n)$$

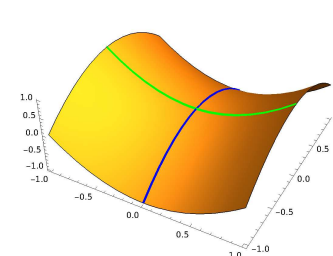
using

- **GD:** $w \leftarrow w - \eta \nabla F(w)$
 - **SGD:** $w \leftarrow w - \eta \tilde{\nabla} F(w)$
 - **Newton:** $w \leftarrow w - (\nabla^2 F(w))^{-1} \nabla F(w)$
- $$(\mathbb{E}[\tilde{\nabla} F(w)] = \nabla F(w))$$

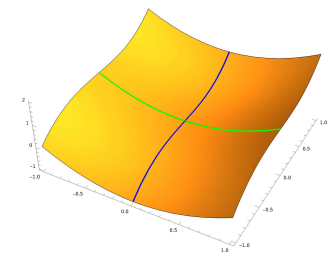
7 / 46

Convergence guarantees of GD/SGD

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points)



“good” saddle points



“bad” saddle points

8 / 46

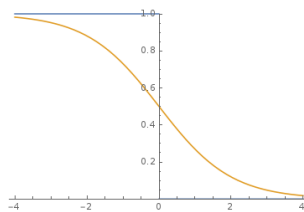
Perceptron and logistic regression

Initialize $\mathbf{w} = \mathbf{0}$ or randomly.

Repeat:

- pick a data point $\mathbf{x}_n$ uniformly at random (**common trick for SGD**)
- update parameter:

$$\mathbf{w} \leftarrow \mathbf{w} + \begin{cases} \mathbb{I}[y_n \mathbf{w}^T \mathbf{x}_n \leq 0] y_n \mathbf{x}_n & \text{(Perceptron)} \\ \eta \sigma(-y_n \mathbf{w}^T \mathbf{x}_n) y_n \mathbf{x}_n & \text{(logistic regression)} \end{cases}$$



9 / 46

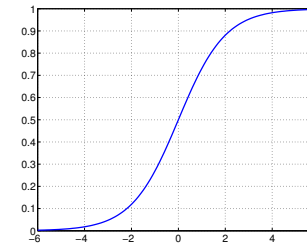
A Probabilistic view of logistic regression

Minimizing logistic loss = MLE for the sigmoid model

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^N \ell_{\text{logistic}}(y_n \mathbf{w}^T \mathbf{x}_n) = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{n=1}^N \mathbb{P}(y_n | \mathbf{x}_n; \mathbf{w})$$

where

$$\mathbb{P}(y | \mathbf{x}; \mathbf{w}) = \sigma(y \mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-y \mathbf{w}^T \mathbf{x}}}$$



10 / 46

Outline

- 1 Review of Last Lecture
- 2 **Multiclass Classification**
 - Multinomial logistic regression
 - Reduction to binary classification
- 3 Kernel methods

11 / 46

Classification

Recall the setup:

- input (feature vector): $\mathbf{x} \in \mathbb{R}^D$
- output (label): $y \in [C] = \{1, 2, \dots, C\}$
- goal: learn a mapping $f: \mathbb{R}^D \rightarrow [C]$

Examples:

- recognizing digits ($C = 10$) or letters ($C = 26$ or 52)
- predicting weather: sunny, cloudy, rainy, etc
- predicting image category: ImageNet dataset ($C \approx 20K$)

Nearest Neighbor Classifier naturally works for arbitrary C .

12 / 46

Linear models: from binary to multiclass

Step 1: *What should a linear model look like for multiclass tasks?*

Note: a linear model for binary tasks (switching from $\{-1, +1\}$ to $\{1, 2\}$)

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} \geq 0 \\ 2 & \text{if } \mathbf{w}^T \mathbf{x} < 0 \end{cases}$$

can be written as

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}_1^T \mathbf{x} \geq \mathbf{w}_2^T \mathbf{x} \\ 2 & \text{if } \mathbf{w}_2^T \mathbf{x} > \mathbf{w}_1^T \mathbf{x} \end{cases}$$

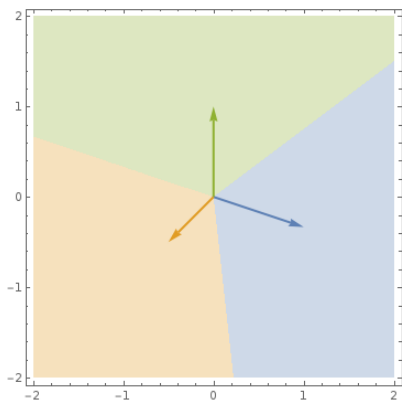
$$= \operatorname{argmax}_{k \in \{1, 2\}} \mathbf{w}_k^T \mathbf{x}$$

for any $\mathbf{w}_1, \mathbf{w}_2$ s.t. $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2$

Think of $\mathbf{w}_k^T \mathbf{x}$ as a **score for class k** .

13 / 46

Linear models: from binary to multiclass

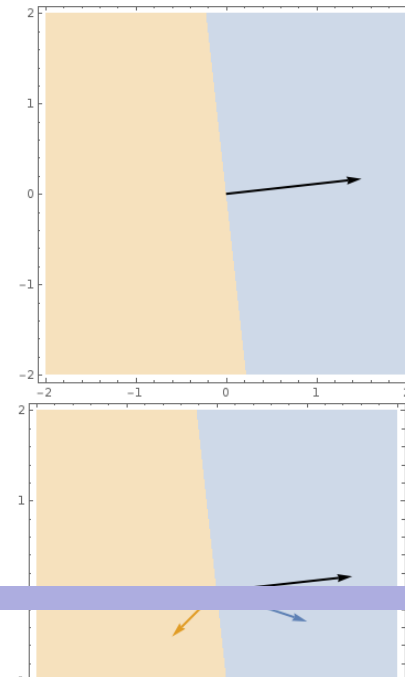


$$\begin{aligned} \mathbf{w}_1 &= (1, -\frac{1}{3}) \\ \mathbf{w}_2 &= (-\frac{1}{2}, -\frac{1}{2}) \\ \mathbf{w}_3 &= (0, 1) \end{aligned}$$

- **Blue class:**
 $\{\mathbf{x} : 1 = \operatorname{argmax}_k \mathbf{w}_k^T \mathbf{x}\}$
- **Orange class:**
 $\{\mathbf{x} : 2 = \operatorname{argmax}_k \mathbf{w}_k^T \mathbf{x}\}$
- **Green class:**
 $\{\mathbf{x} : 3 = \operatorname{argmax}_k \mathbf{w}_k^T \mathbf{x}\}$

15 / 46

Linear models: from binary to multiclass



$$\begin{aligned} \mathbf{w} &= (\frac{3}{2}, \frac{1}{6}) = \mathbf{w}_1 - \mathbf{w}_2 \\ \mathbf{w}_1 &= (1, -\frac{1}{3}) \\ \mathbf{w}_2 &= (-\frac{1}{2}, -\frac{1}{2}) \end{aligned}$$

- **Blue class:**

$$\{\mathbf{x} : 1 = \operatorname{argmax}_k \mathbf{w}_k^T \mathbf{x}\}$$

14 / 46

Linear models for multiclass classification

$$\begin{aligned} \mathcal{F} &= \left\{ f(\mathbf{x}) = \operatorname{argmax}_{k \in [C]} \mathbf{w}_k^T \mathbf{x} \mid \mathbf{w}_1, \dots, \mathbf{w}_C \in \mathbb{R}^D \right\} \\ &= \left\{ f(\mathbf{x}) = \operatorname{argmax}_{k \in [C]} (\mathbf{W} \mathbf{x})_k \mid \mathbf{W} \in \mathbb{R}^{C \times D} \right\} \end{aligned}$$

Step 2: *How do we generalize perceptron/hinge/logistic loss?*

This lecture: focus on the more popular **logistic loss**

16 / 46

Multinomial logistic regression: a probabilistic view

Observe: for binary logistic regression, with $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2$:

$$\mathbb{P}(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} = \frac{e^{\mathbf{w}_1^T \mathbf{x}}}{e^{\mathbf{w}_1^T \mathbf{x}} + e^{\mathbf{w}_2^T \mathbf{x}}} \propto e^{\mathbf{w}_1^T \mathbf{x}}$$

Naturally, for multiclass:

$$\text{softmax}(\mathbf{W}\mathbf{x})_k = \mathbb{P}(y = k \mid \mathbf{x}; \mathbf{W}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{k' \in [C]} e^{\mathbf{w}_{k'}^T \mathbf{x}}} \propto e^{\mathbf{w}_k^T \mathbf{x}}$$

Important operator: **softmax function** (or really, “softargmax”)

$$\text{For a vector } s \in \mathbb{R}^C, \text{ softmax}(s) = \left(\frac{e^{s_1}}{\sum_{k \in [C]} e^{s_k}}, \dots, \frac{e^{s_C}}{\sum_{k \in [C]} e^{s_k}} \right)$$

17 / 46

Applying MLE again

Maximize probability of seeing labels $y_1, \dots, y_N$ given $\mathbf{x}_1, \dots, \mathbf{x}_N$

$$P(\mathbf{W}) = \prod_{n=1}^N \mathbb{P}(y_n \mid \mathbf{x}_n; \mathbf{W}) = \prod_{n=1}^N \frac{e^{\mathbf{w}_{y_n}^T \mathbf{x}_n}}{\sum_{k \in [C]} e^{\mathbf{w}_k^T \mathbf{x}_n}}$$

By taking **negative log**, this is equivalent to minimizing

$$F(\mathbf{W}) = \sum_{n=1}^N \ln \left(\frac{\sum_{k \in [C]} e^{\mathbf{w}_k^T \mathbf{x}_n}}{e^{\mathbf{w}_{y_n}^T \mathbf{x}_n}} \right) = \sum_{n=1}^N \ln \left(1 + \sum_{k \neq y_n} e^{(\mathbf{w}_k - \mathbf{w}_{y_n})^T \mathbf{x}_n} \right)$$

This is the **multiclass logistic loss**, a.k.a. **cross-entropy loss**.

When $C = 2$, this is the same as binary logistic loss.

18 / 46

Step 3: Optimization

Apply **SGD**: what is the gradient of

$$F_n(\mathbf{W}) = \ln \left(1 + \sum_{k' \neq y_n} e^{(\mathbf{w}_{k'} - \mathbf{w}_{y_n})^T \mathbf{x}_n} \right) ?$$

It's a $C \times D$ matrix. Let's focus on the k -th row:

If $k \neq y_n$:

$$\nabla_{\mathbf{w}_k^T} F_n(\mathbf{W}) = \frac{e^{(\mathbf{w}_k - \mathbf{w}_{y_n})^T \mathbf{x}_n}}{1 + \sum_{k' \neq y_n} e^{(\mathbf{w}_{k'} - \mathbf{w}_{y_n})^T \mathbf{x}_n}} \mathbf{x}_n^T = \mathbb{P}(k \mid \mathbf{x}_n; \mathbf{W}) \mathbf{x}_n^T$$

else:

$$\nabla_{\mathbf{w}_k^T} F_n(\mathbf{W}) = \frac{-\left(\sum_{k' \neq y_n} e^{(\mathbf{w}_{k'} - \mathbf{w}_{y_n})^T \mathbf{x}_n}\right)}{1 + \sum_{k' \neq y_n} e^{(\mathbf{w}_{k'} - \mathbf{w}_{y_n})^T \mathbf{x}_n}} \mathbf{x}_n^T = (\mathbb{P}(y_n \mid \mathbf{x}_n; \mathbf{W}) - 1) \mathbf{x}_n^T$$

19 / 46

SGD for multinomial logistic regression

Initialize $\mathbf{W} = \mathbf{0}$ (or randomly). Repeat:

- 1 pick $n \in [N]$ uniformly at random
- 2 update the parameters

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \begin{pmatrix} \mathbb{P}(y = 1 \mid \mathbf{x}_n; \mathbf{W}) \\ \vdots \\ \mathbb{P}(y = y_n \mid \mathbf{x}_n; \mathbf{W}) - 1 \\ \vdots \\ \mathbb{P}(y = C \mid \mathbf{x}_n; \mathbf{W}) \end{pmatrix} \mathbf{x}_n^T$$

Think about why the algorithm makes sense intuitively.

20 / 46

A note on prediction

Having learned $\mathbf{W}$, we can either

- make a *deterministic* prediction $\operatorname{argmax}_{k \in [C]} \mathbf{w}_k^T \mathbf{x}$
- make a *randomized* prediction drawn from $\operatorname{softmax}(\mathbf{W}\mathbf{x})$

21 / 46

Reduce multiclass to binary

Is there an *even more general and simpler approach* to derive multiclass classification algorithms?

Given a binary classification algorithm (*any one*, not just linear methods), can we turn it to a multiclass algorithm, *in a black-box manner*?

Yes, there are in fact many ways to do it.

- **one-versus-all** (one-versus-rest, one-against-all, etc.)
- **one-versus-one** (all-versus-all, etc.)
- **tree-based reduction**

23 / 46

Generalization of cross-entropy loss

Given a general model class:

$$\mathcal{F} = \left\{ f(\mathbf{x}) = \operatorname{argmax}_{k \in [C]} s(\mathbf{x})_k \right\}$$

where $s : \mathbb{R}^D \rightarrow \mathbb{R}^C$ is a “**scoring**” function.

The *cross-entropy loss* of f for a training sample $(\mathbf{x}, y)$ is

$$-\ln(\operatorname{softmax}(s(\mathbf{x}))_y) = -\ln\left(\frac{e^{s(\mathbf{x})_y}}{\sum_{k \in [C]} e^{s(\mathbf{x})_k}}\right) = \ln\left(1 + \sum_{k \neq y} e^{s(\mathbf{x})_k - s(\mathbf{x})_y}\right)$$

22 / 46

One-versus-all (OvA)

(picture credit: [link](#))

Idea: train C binary classifiers to learn “**is class k or not?**” for each k .

Training: for each class $k \in [C]$,

- relabel examples with class k as $+1$, and all others as -1
- train a binary classifier h_k using this new dataset

				
x_1 	x_1 —	x_1 +	x_1 —	x_1 —
x_2 	x_2 —	x_2 —	x_2 +	x_2 —
x_3 	x_3 —	x_3 —	x_3 —	x_3 +
x_4 	x_4 —	x_4 +	x_4 —	x_4 —
x_5 	x_5 +	x_5 —	x_5 —	x_5 —
	$\Downarrow$ h_1	$\Downarrow$ h_2	$\Downarrow$ h_3	$\Downarrow$ h_4

24 / 46

One-versus-all (OvA)

Prediction: for a new example x

- ask each h_k : **does this belong to class k ?** (i.e. $h_k(x)$)
- randomly pick among all k 's s.t. $h_k(x) = +1$.

Issue: will (probably) make a mistake *as long as one of h_k errs*.

25 / 46

One-versus-one (OvO)

(picture credit: link)

Idea: train $\binom{C}{2}$ binary classifiers to learn “**is class k or k' ?**”.

Training: for each pair (k, k') ,

- relabel examples with class k as $+1$ and examples with class k' as -1
- discard all other examples
- train a binary classifier $h_{(k,k')}$ using this new dataset

		■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■
		■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■	■ vs. ■
x_1	■	x_1 —		x_1 —		x_1 —	
x_2	■		x_2 —	x_2 +		x_2 +	
x_3	■			x_3 —	x_3 +	x_3 —	
x_4	■	x_4 —			x_4 —		x_4 —
x_5	■	x_5 +	x_5 +			x_5 +	
		↓	↓	↓	↓	↓	↓
		$h_{(1,2)}$	$h_{(1,3)}$	$h_{(3,4)}$	$h_{(4,2)}$	$h_{(1,4)}$	$h_{(3,2)}$

26 / 46

One-versus-one (OvO)

Prediction: for a new example x

- ask each classifier $h_{(k,k')}$ to **vote for either class k or k'**
- predict the class with the most votes (break tie in some way)

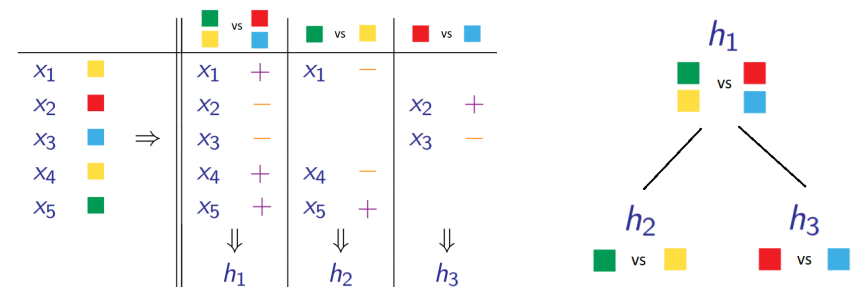
More robust than one-versus-all, but *slower* in prediction.

27 / 46

Tree based method

Idea: train $\approx C$ binary classifiers to learn “**belongs to which half?**”.

Training: see pictures

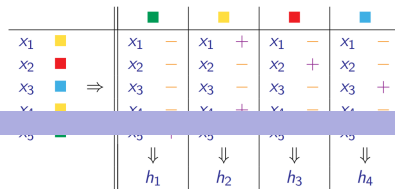


Prediction is also natural, *but is very fast!* (think ImageNet where $C \approx 20K$)

28 / 46

Comparisons

Reduction	training time	prediction time	remark
OvA	CN	C	not robust
OvO	$(C - 1)N$	$\mathcal{O}(C^2)$	can achieve very small training error
Tree	$\mathcal{O}((\log_2 C)N)$	$\mathcal{O}(\log_2 C)$	good for "extreme classification"



29 / 46

training points are created

Motivation

Recall: when linear models are not good enough, we can use a **nonlinear feature map** $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ to transform all x to $\phi(x)$.

Issue: what if M is huge, or even infinity?

Solution: kernel methods

Outline

- 1 Review of Last Lecture
- 2 Multiclass Classification
- 3 Kernel methods
 - Motivation
 - Example: Perceptron
 - Kernel Trick
 - Kernelized Perceptron

30 / 46

Case study: Perceptron for binary classification

Perceptron

Initialize $\mathbf{w} = \mathbf{0}$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$, update $\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$

Observation: $\mathbf{w}$ is a **linear combination** of training data

$$\mathbf{w} = \sum_{m=1}^N \alpha_m \mathbf{x}_m$$

where $\alpha_m = y_m \times$ **number of times x_m has been misclassified**

Dual form of Perceptron

Perceptron (primal form)

Initialize $\mathbf{w} = \mathbf{0}$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\mathbf{w}^T \mathbf{x}_n) \neq y_n$, update $\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$

How to update $\alpha_1, \dots, \alpha_N$ so that $\sum_{m=1}^N \alpha_m \mathbf{x}_m \equiv \mathbf{w}$ holds always?

Perceptron (dual form)

Initialize $\alpha_m = 0$ for all $m \in [N]$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\sum_{m=1}^N \alpha_m \mathbf{x}_m^T \mathbf{x}_n) \neq y_n$, update $\alpha_n \leftarrow \alpha_n + y_n$

33 / 46

Applying a feature map

Perceptron (primal form with ϕ)

issue: time/space linear in M

Initialize $\mathbf{w} = \mathbf{0} \in \mathbb{R}^M$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\mathbf{w}^T \phi(\mathbf{x}_n)) \neq y_n$, update $\mathbf{w} \leftarrow \mathbf{w} + y_n \phi(\mathbf{x}_n)$

Perceptron (dual form with ϕ)

Initialize $\alpha_m = 0$ for all $m \in [N]$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\sum_{m=1}^N \alpha_m \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)) \neq y_n$, update $\alpha_n \leftarrow \alpha_n + y_n$

If we can compute $\phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$ without explicitly evaluating $\phi(\mathbf{x}_m)$ and $\phi(\mathbf{x}_n)$, then time/space is independent of M!

34 / 46

Example

Consider the following polynomial basis $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$:

$$\phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

What is the inner product between $\phi(\mathbf{x})$ and $\phi(\mathbf{x}')$?

$$\begin{aligned} \phi(\mathbf{x})^T \phi(\mathbf{x}') &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2 x_2'^2 \\ &= (x_1x_1' + x_2x_2')^2 = (\mathbf{x}^T \mathbf{x}')^2 \end{aligned}$$

Therefore, *the inner product in the new space is simply a function of the inner product in the original space.*

35 / 46

Another example

$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^{2D}$ is parameterized by θ :

$$\phi_\theta(\mathbf{x}) = \begin{pmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \vdots \\ \cos(\theta x_D) \\ \sin(\theta x_D) \end{pmatrix}$$

What is the inner product between $\phi_\theta(\mathbf{x})$ and $\phi_\theta(\mathbf{x}')$?

$$\begin{aligned} \phi_\theta(\mathbf{x})^T \phi_\theta(\mathbf{x}') &= \sum_{d=1}^D \cos(\theta x_d) \cos(\theta x'_d) + \sin(\theta x_d) \sin(\theta x'_d) \\ &= \sum_{d=1}^D \cos(\theta(x_d - x'_d)) \quad (\text{trigonometric identity}) \end{aligned}$$

Once again, *the inner product in the new space is a simple function of the features in the original space.*

36 / 46

More complicated example

Based on ϕ_θ , define $\phi_L : \mathbb{R}^D \rightarrow \mathbb{R}^{2D(L+1)}$ for some integer L :

$$\phi_L(\mathbf{x}) = \begin{pmatrix} \phi_0(\mathbf{x}) \\ \phi_{\frac{2\pi}{L}}(\mathbf{x}) \\ \phi_{2\frac{2\pi}{L}}(\mathbf{x}) \\ \vdots \\ \phi_{L\frac{2\pi}{L}}(\mathbf{x}) \end{pmatrix}$$

What is the inner product between $\phi_L(\mathbf{x})$ and $\phi_L(\mathbf{x}')$?

$$\begin{aligned} \phi_L(\mathbf{x})^T \phi_L(\mathbf{x}') &= \sum_{\ell=0}^L \phi_{\frac{2\pi\ell}{L}}(\mathbf{x})^T \phi_{\frac{2\pi\ell}{L}}(\mathbf{x}') \\ &= \sum_{\ell=0}^L \sum_{d=1}^D \cos\left(\frac{2\pi\ell}{L}(x_d - x'_d)\right) \end{aligned}$$

37 / 46

Infinite dimensional mapping

When $L \rightarrow \infty$, even if we cannot compute $\phi(\mathbf{x})$, a vector of *infinite dimension*, we can still compute inner product:

$$\begin{aligned} \phi_\infty(\mathbf{x})^T \phi_\infty(\mathbf{x}') &= \int_0^{2\pi} \sum_{d=1}^D \cos(\theta(x_d - x'_d)) d\theta \\ &= \sum_{d=1}^D \frac{\sin(2\pi(x_d - x'_d))}{x_d - x'_d} \end{aligned}$$

Again, a simple function of the original features.

Note that using this mapping in linear classification, we are *learning a weight $\mathbf{w}$ with infinite dimension!*

38 / 46

Kernel functions

Definition: a function $k : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is called a *kernel function* if there exists a function $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ so that for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$,

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

Can be seen as a kind of similarity measure.

Examples we have seen

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}')^2 \\ k(\mathbf{x}, \mathbf{x}') &= \sum_{d=1}^D \frac{\sin(2\pi(x_d - x'_d))}{x_d - x'_d} \end{aligned}$$

39 / 46

Composing kernels

Creating more kernel functions using the following rules:

If $k_1(\cdot, \cdot)$ and $k_2(\cdot, \cdot)$ are kernels, the followings are kernels too

- **conical combination:** $\alpha k_1(\cdot, \cdot) + \beta k_2(\cdot, \cdot)$ if $\alpha, \beta \geq 0$
- **product:** $k_1(\cdot, \cdot) k_2(\cdot, \cdot)$
- **exponential:** $e^{k(\cdot, \cdot)}$
- ...

Verify using the definition of kernel!

40 / 46

Common kernel functions

Two most commonly used kernel functions in practice:

Polynomial kernel

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

for $c \geq 0$ and d is a positive integer.

Gaussian kernel or Radial Basis Function (RBF) kernel

$$k(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}}$$

for some $\sigma > 0$.

Think about *what the corresponding ϕ is* for each kernel.

Back to Perceptron

Perceptron (dual form with ϕ)

Initialize $\alpha_m = 0$ for all $m \in [N]$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\sum_{m=1}^N \alpha_m \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)) \neq y_n$, update $\alpha_n \leftarrow \alpha_n + y_n$

Instead of choosing $\phi: \mathbb{R}^D \rightarrow \mathbb{R}^M$ explicitly, we choose a kernel function k .

Kernelized Perceptron

Initialize $\alpha_m = 0$ for all $m \in [N]$

Repeat:

- Pick a data point index n uniformly at random
- If $\text{sgn}(\sum_{m=1}^N \alpha_m k(\mathbf{x}_m, \mathbf{x}_n)) \neq y_n$, update $\alpha_n \leftarrow \alpha_n + y_n$

Completely *M-independent*, becomes a *non-parametric* method

Gram/kernel matrix

When N is small, can precompute all inner products as a **Gram matrix**

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} = \Phi \Phi^T$$

Recall: $\Phi = \begin{pmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{pmatrix} \in \mathbb{R}^{N \times M}$

Gram matrix vs covariance matrix

	dimensions	entry (i, j)	property
$\Phi \Phi^T$	$N \times N$	$\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$	both are symmetric and positive semidefinite
$\Phi^T \Phi$	$M \times M$	$\sum_{n=1}^N \phi(\mathbf{x}_n)_i \phi(\mathbf{x}_n)_j$	

Mercer Theorem

$k : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is a kernel function if and only if the Gram matrix $\mathbf{K}$ for *any* N and *any* $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ is positive semidefinite.

- useful for showing that a function is not a kernel

Example:

$$k(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2^2$$

is *not a kernel*, why?

If it is a kernel, the kernel matrix for two data points $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\mathbf{K} = \begin{pmatrix} 0 & \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \\ \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 & 0 \end{pmatrix}$$

must be positive semidefinite, but it is not (contradiction).

Kernelizing ML algorithms

Many other ML algorithms can be **kernelized**:

- nearest neighbor classifier
- linear regression
- logistic regression
- SVM
- ...

Key idea: rewrite the algorithm so that its dependence on the transformed dataset Φ is only through the Gram matrix $\mathbf{K} = \Phi\Phi^T$.