Machine Learning II

Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada
Sept. 17, 2024

Course Announcements

Assignment 1 is released and due October 7

No class on Tues, October 15 due to holiday as per SFU guidelines

Midterm is set for Tues, October 29 in class.

Course Overview

Week 1 : Getting to know you
Week 2 : Introduction to Artificial Intelligence
Week 3 : Machine Learning I: Basic Supervised Models (Classification)
Week 4 : Machine Learning II: Supervised Regression, Classification and Gradient Descent
Week 5 : Machine Learning III: Neural Networks and Unsupervised Learning
Week 6 : Search
Week 7 : Markov Decision Processes
Week 8 : Midterm
Week 9 : Reinforcement Learning
Week 10 : Games
Week 11 : Hidden Markov Models and Bayesian Networks
Week 12 : Constraint Satisfaction Problems
Week 13 : Ethics and Explainability

Reflex-based models	State-based models	Variable-based models	Logic-based models
	Search Markov decision processes Games	Constraint satisfaction problems Markov networks Bayesian networks	

Today's Plan

Supervised Learning

19.6 - Linear Regression
19.4.2 - Loss functions
19.6.2 - Gradient Descent
19.6.4 - Linear Classification
19.6.2 - Stochastic Gradient Descent

Recall

Supervised Learning

Last time, we learned about 2 non-parametric classification methods:
K-Nearest Neighbors and Decision Trees.

Today we'll learn about parametric regression and classification.

Supervised Learning

Linear Regression

Regression

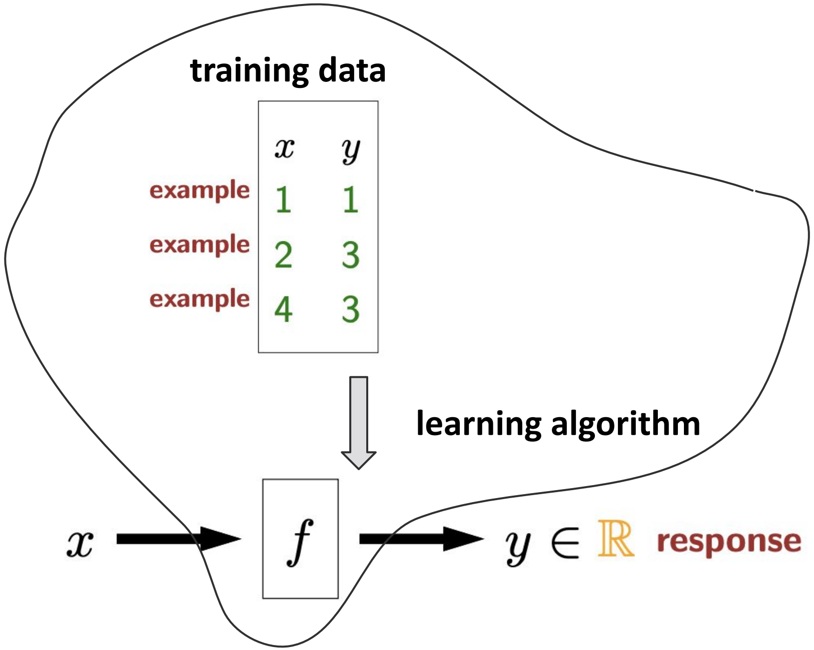
$$x \longrightarrow \boxed{f} \longrightarrow y \in \mathbb{R}$$ **response**

Housing: information about house $\longrightarrow$ price

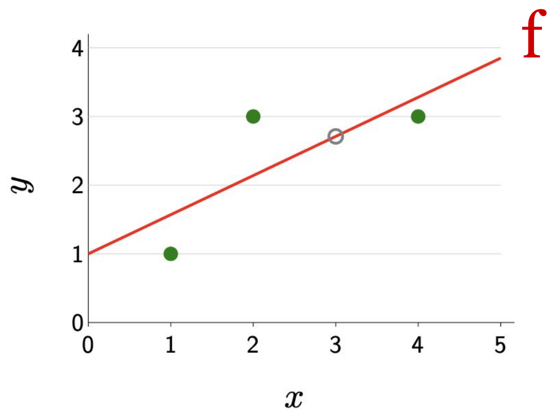
Arrival times: destination, weather, time $\longrightarrow$ time of arrival

Robotic vision: image, xy-coord $\longrightarrow$ distance

Linear regression framework



Training $\longrightarrow$ finding our predictor f



3 **input** $\longrightarrow$ $\boxed{f}$ **predictor** $\longrightarrow$ 2.71 **output**

Which predictors **f** is the learning algorithm **allowed** to produce?

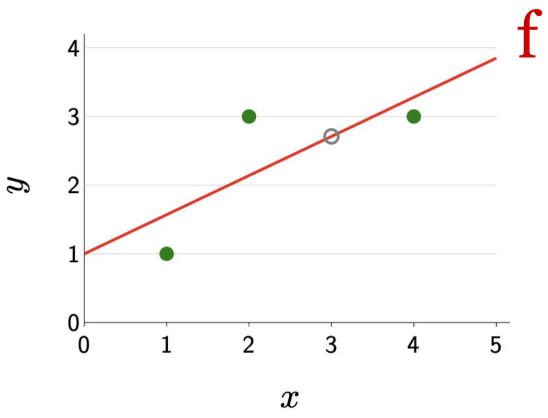
Decide the **hypothesis class** ⇒ *here, linear function*

How **good** is a predictor? Decide the **loss function** ⇒ *here, squared loss*

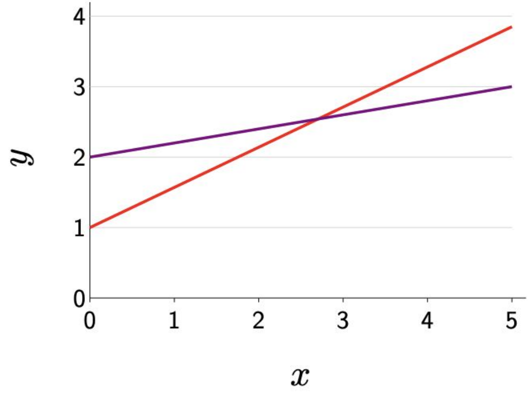
How to **find** the best predictor? Decide the **optimization algorithm** ⇒ *here, gradient descent*

Hypothesis class for Linear Regression: Linear Functions

Training $\longrightarrow$ finding our predictor f



$$f(x) = 1 + 0.57x$$



$$f(x) = 2 + 0.2x$$

$$f(x) = w_1 + w_2 x$$

Here, we opt for a **linear** function, as opposed to a nonlinear function such as a polynomial

Linear Regression

weight vector

feature extractor

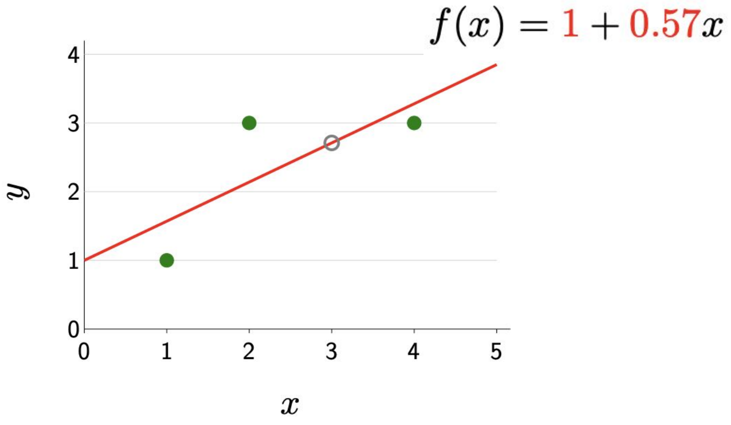
prediction

feature vector

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$

$$[w_1, w_2] \cdot [1, x]$$

These are the parameters we want to tweak for the problem.



We might hypothesize that $\mathbf{w}$ = [1,0.57] produces a good model that fits our training data. If so, we could use this model to predict the outcome of x=3.

$$f_{\mathbf{w}}(3) = [1, 0.57] \cdot [1, 3]$$
$$= 1 + 0.57 \cdot 3$$
$$= \mathbf{2.71}$$

But how do we know if $\mathbf{w}$ is any good?

Linear Regression: How to evaluate a weight vector?

For each sample in our training set, find the difference between the **predicted output** $f_{\mathbf{w}}(x)$ and the **actual** output y. This difference is called the **residual**, $f_{\mathbf{w}}(x)-y$

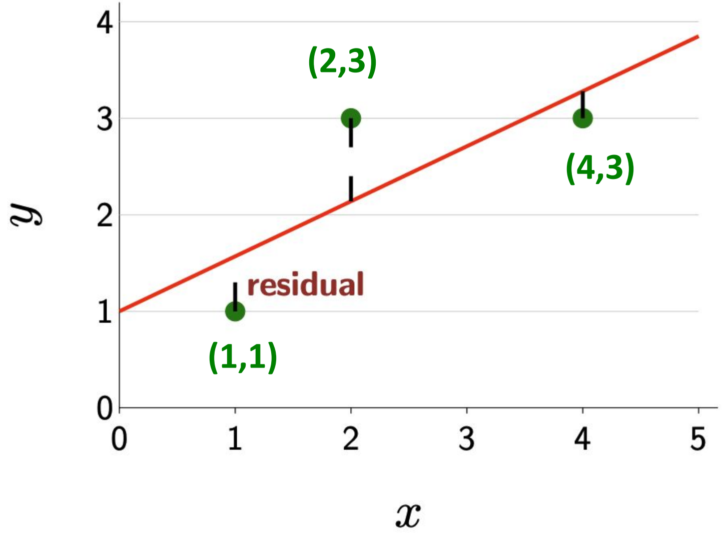
Let's square the residual to ensure the difference is positive, to later sum them. We will call the squared value $(f_{\mathbf{w}}(x)-y)^2$ the **squared error loss** also called **L2** loss.

Remember Gauss' method of least squares? It **minimized** the **sum** of the **squares** of the **residuals**.

training data $\mathcal{D}_{\text{train}}$

x	y
1	1
2	3
4	3

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$
$$\mathbf{w} = [1, 0.57]$$
$$\phi(x) = [1, x]$$

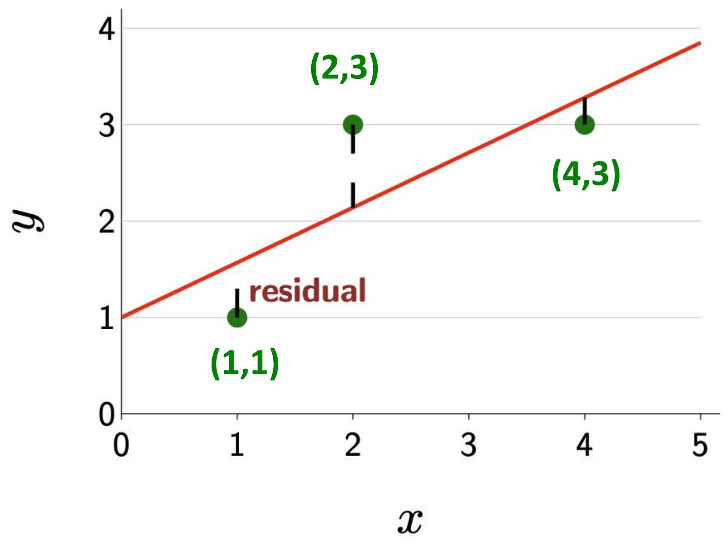


$$\text{Loss}(x,y,\mathbf{w}) = (f_{\mathbf{w}}(x)-y)^2$$
$$\text{Loss}(2,3,\mathbf{w}) = (f_{\mathbf{w}}(x)-3)^2$$
$$= (\mathbf{w}\cdot\phi(x)-3)^2$$
$$= ([1,0.57]\cdot[1,2]-3)^2$$
$$= (1\cdot 1+0.57\cdot 2-3)^2 = 0.74$$

Do this for all the samples in your training set and find the average

SFU

Linear Regression: How to evaluate a weight vector?



$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|}\sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

$f_{\mathbf{w}}(x)=\mathbf{w}\cdot\phi(x)$

$\mathbf{w}=[1,0.57]$

$\phi(x)=[1,x]$

$\text{Loss}(1,1,\mathbf{w})=(\mathbf{w}\cdot\phi(1)-1)^2=0.32$

$\text{Loss}(2,3,\mathbf{w})=(\mathbf{w}\cdot\phi(2)-3)^2=0.74$

$\text{Loss}(4,3,\mathbf{w})=(\mathbf{w}\cdot\phi(4)-3)^2=0.08$

$\text{Loss}(x,y,\mathbf{w})=(f_{\mathbf{w}}(x)-y)^2$

$\text{Loss}(2,3,\mathbf{w})=(f_{\mathbf{w}}(x)-3)^2$ x is 2 here

$=(\mathbf{w}\cdot\phi(x)-3)^2$

$=([1,0.57]\cdot[1,2]-3)^2$

$=(1\cdot 1+0.57\cdot 2-3)^2=0.74$

$\text{TrainLoss}([1,0.57])$

$=(.32+.74+.08)/3 = 0.38$

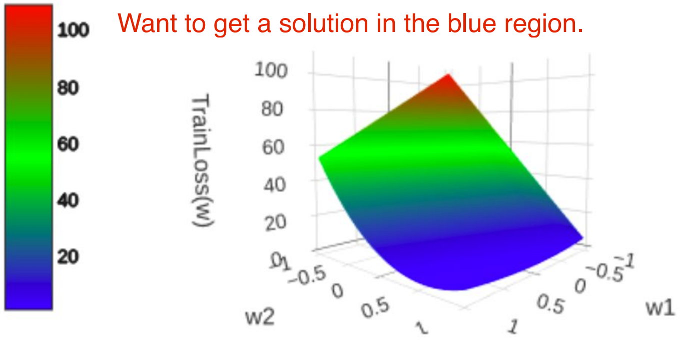
You could use a grid search to find an optimal w with lowest TrainLoss

Finding the optimal predictor using a Loss

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} (f_{\mathbf{w}}(x) - y)^2$$

We can minimize this value using gradients.

Want to get a solution in the blue region.



$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

The best predictor is the one with the lowest training loss. We need to solve this optimization problem.

What is grid search?

You *could* perform a grid search over w1 and w2 to find a w with a minimum training loss, but this will get unwieldy for higher dimensions.

Gradient Descent: Faster than Grid Search

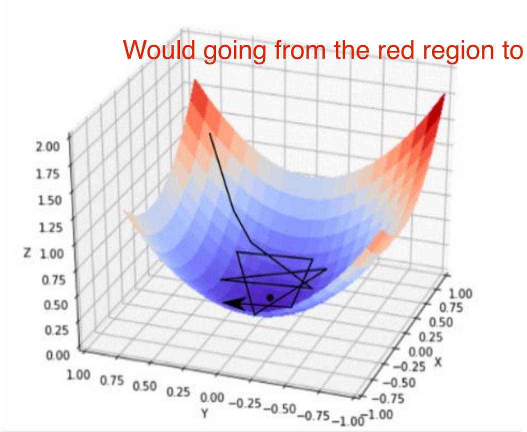
Goal: $\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Potentially faster? In which cases is it slower

Definition: gradient

The gradient $\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$ is the direction that increases the training loss the most.

Would going from the red region to the blue region not be the negative gradient?



Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$: epochs

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

Computing the gradient

Note that in the textbook, you may see **(y-w·ϕ(x))**

Objective function:

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} (\mathbf{w}\cdot\phi(x) - y)^2$$

Review gradient descent.

Remember, we're taking the gradient w.r.t. w, so everything else is treated as a constant

Gradient (use chain rule):

$$\nabla_{\mathbf{w}}\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} 2(\underbrace{\mathbf{w}\cdot\phi(x) - y}_{\text{prediction}-\text{target}})\phi(x)$$

Gradient Descent example

training data $\mathcal{D}_{\text{train}}$

x	y
1	1
2	3
4	3

Residual multiplied by the feature vector!

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} 2\underbrace{(\mathbf{w}\cdot\phi(x) - y)}_{\text{prediction}-\text{target}}\phi(x)$$

Gradient update: $\mathbf{w} \leftarrow \mathbf{w} - 0.1\nabla_{\mathbf{w}}\text{TrainLoss}(\mathbf{w})$

Step size

t	$\mathbf{w}$	$\nabla_{\mathbf{w}}$ TrainLoss(w)
1	[0,0]	$\frac{2([0,0]\cdot[1,1]-1)[1,1] + 2([0,0]\cdot[1,2]-3)[2,3] + 2([0,0]\cdot[1,4]-3)[4,3]}{3}$
		$= [-4.67, -12.67]$ // then update $\mathbf{w}$ with step size (learning rate)
2	[0.47,1.27]	

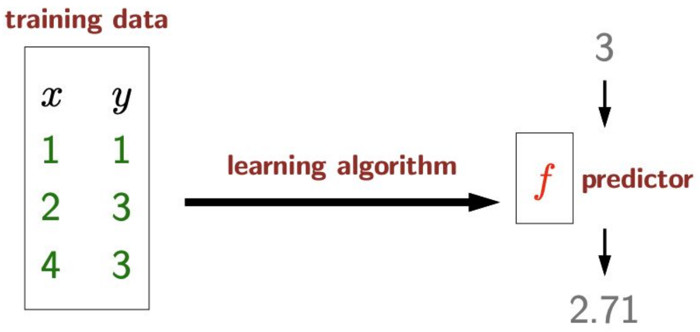
SFU

Gradient Descent example

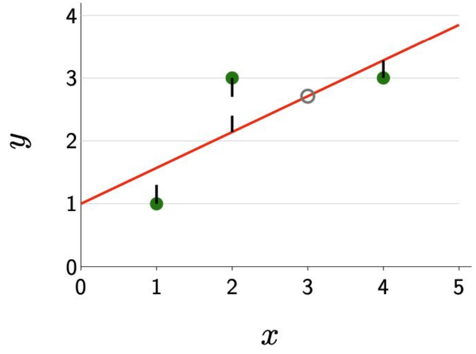
t	$\mathbf{w}$	$\nabla_w \text{TrainLoss}(w)$
1	[0,0]	$\frac{2([0,0]\cdot[1,1]-1)[1,1] + 2([0,0]\cdot[1,2]-3)[2,3] + 2([0,0]\cdot[1,4]-3)[4,3]}{3} = [-4.67,-12.67]$
2	[0.47,1.27]	$\frac{2([.47,1.27]\cdot[1,1]-1)[1,1] + 2([.47,1.27]\cdot[1,2]-3)[2,3] + 2([.47,1.27]\cdot[1,4]-3)[4,3]}{3} = [2.18, 7.24]$
3	[0.25,0.54]	
…		
200	[1, 0.57]	$\frac{2([1,.57]\cdot[1,1]-1)[1,1] + 2([1,.57]\cdot[1,2]-3)[2,3] + 2([1,.57]\cdot[1,4]-3)[4,3]}{3} = [0,0]$

Until gradient is zero and there is no more change, i.e. algorithm has converged

Summary: Linear Regression



We can design these 3 elements, here are the typical values for linear regression



Hypothesis class: Linear functions $\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)\}, \phi(x) = [1, x]$

Loss function: Squared loss $\text{Loss}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$

Optimization algorithm: Gradient descent $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \text{TrainLoss}(\mathbf{w})$

Alternatives: Non-linear functions, absolute-value loss, stochastic gradient descent…

Supervised Learning

Linear Classification

Binary Classification

classifier

$$x \longrightarrow \boxed{f} \longrightarrow y \in \{+1, -1\}$$ label

Spam detection: online comment $\longrightarrow$ toxic or not toxic
Earthquake: measurements of event $\longrightarrow$ earthquake or explosion
Melanoma: mole size and color $\longrightarrow$ cancerous or non-cancerous

Extension: **multiclass** classification $y \in \{1, \ldots, K\}$

Linear classification framework

training data

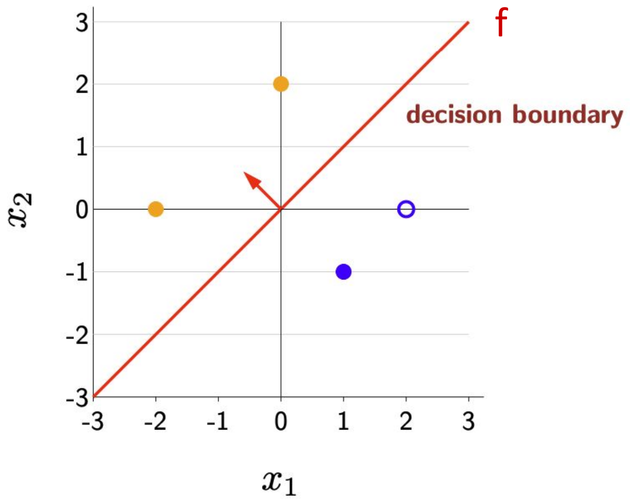
	x_1	x_2	y
example	0	2	1
example	-2	0	1
example	1	-1	-1

⇓ **learning algorithm**

classifier

$x \longrightarrow \boxed{f} \longrightarrow y \in \{+1, -1\}$ **label**

Training ⟶ finding our decision boundary f



$[2, 0]$ **input** $\longrightarrow \boxed{f}$ **predictor** $\longrightarrow$ -1 **label**

Which predictors **f** is the learning algorithm **allowed** to produce?

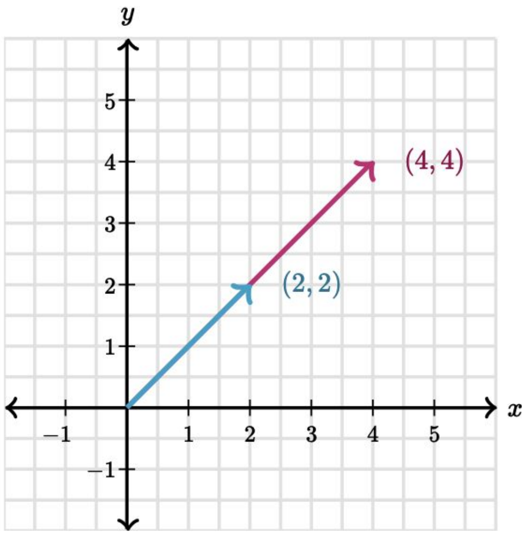
Decide the **hypothesis class** ⇒ *here, linear function with sign(prediction)*

How **good** is a predictor? Decide the **loss function** ⇒ *here, zero-one loss (or hinge, or logistic)*

How to **find** the best predictor? Decide the **optimization algorithm** ⇒ *here, gradient descent*

Recall: Dot Product

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \, \|\vec{b}\| \cos(\theta)$$

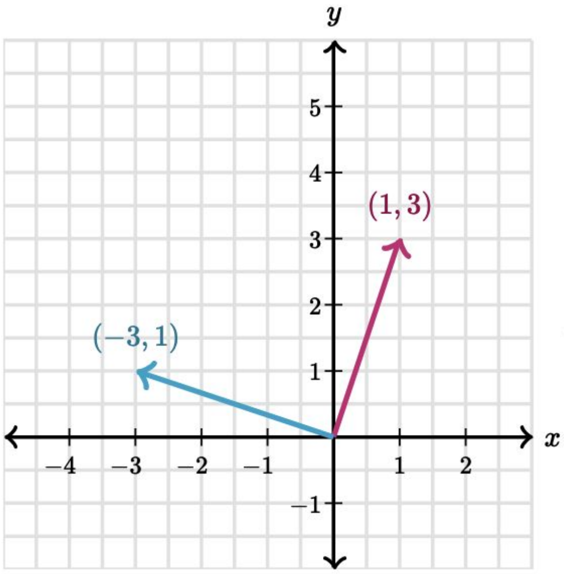


this will help us for determining what class our input falls into

$$\theta = 0$$

$$\cos(0)=1$$

When vectors are pointing in the same direction, their dot product will be 1



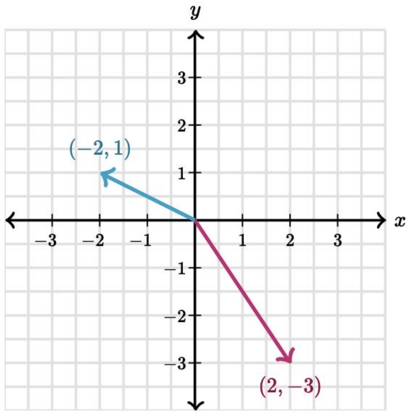
$$\theta = \frac{\pi}{2}$$

$$\cos\left(\frac{\pi}{2}\right) = 0.$$

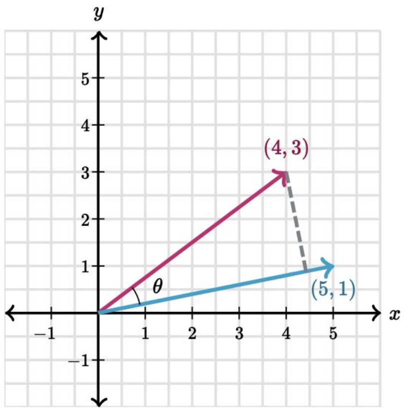
When vectors are perpendicular, their dot product will be 0.

Recall: Dot Product

$$\vec{a} \cdot \vec{b} = ||\vec{a}||\,||\vec{b}||\cos(\theta)$$



$$\frac{\pi}{2} < \theta < \frac{3\pi}{2}$$



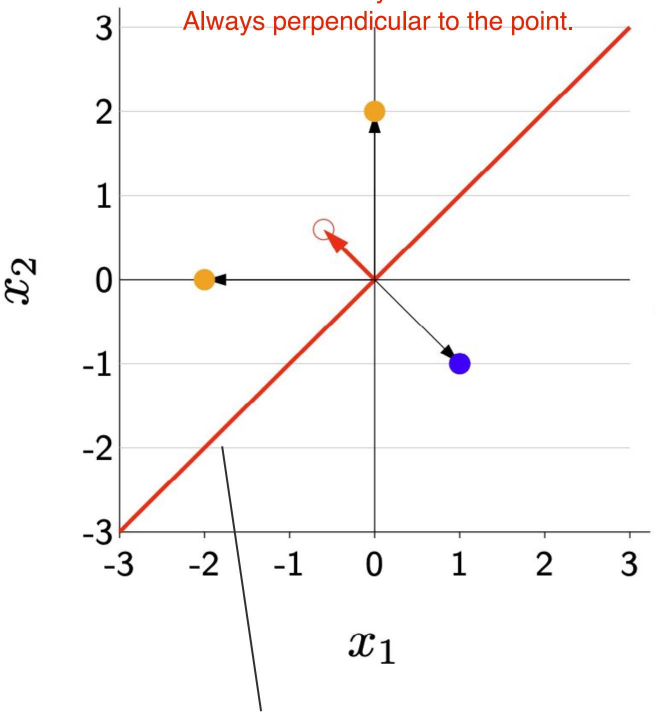
$$\theta < \frac{\pi}{2}$$

When vectors create an **obtuse** angle, their dot product will be **negative**

When vectors create an **acute** angle, their dot product will be **positive**

An example linear classifier

Decision boundary to the red arrow.
Always perpendicular to the point.



Decision boundary :

$\mathbf{x}$ such that $\mathbf{w} \cdot \phi(\mathbf{x}) = 0$

x_1	x_2	$f(x)$
0	2	1
-2	0	1
1	-1	-1

$$\text{sign}(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \end{cases}$$

Reminder:

$$\vec{a} = (a_1, a_2, a_3)$$

$$\vec{b} = (b_1, b_2, b_3)$$

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

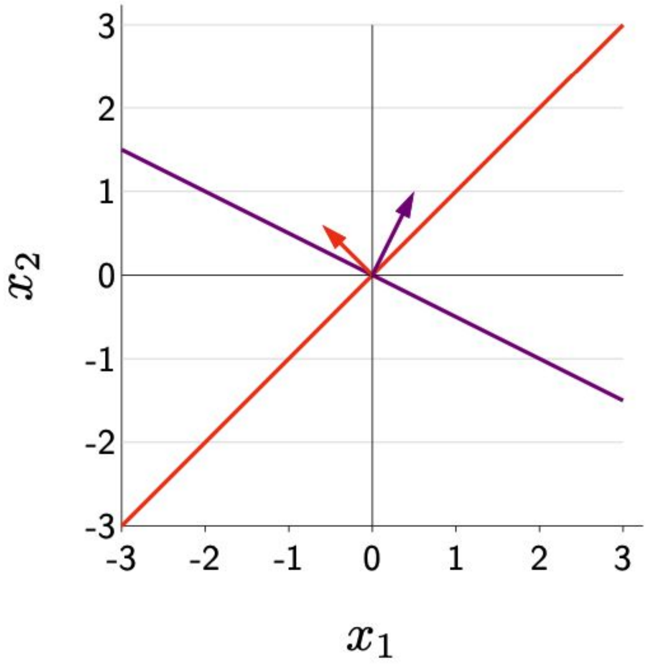
$$f(\mathrm{x}) = \text{sign}(\mathbf{w} \cdot \phi(\mathrm{x}))$$

$$f([0,2]) = \text{sign}([-0.6, 0.6] \cdot [0,2])$$
$$= \text{sign}(-0.6 * 0 + 0.6 * 2) = \text{sign}(1.2) = 1$$

$$f([0,2]) = \text{sign}([-0.6, 0.6] \cdot [-2,0])$$
$$= \text{sign}(-0.6 * -2 + 0.6 * 0) = \text{sign}(1.2) = 1$$

$$f([0,2]) = \text{sign}([-0.6, 0.6] \cdot [1,-1])$$
$$= \text{sign}(-0.6 * 1 + 0.6 * -1) = \text{sign}(-1.2) = -1$$

Binary classification



$\phi(x) = [x_1, x_2]$

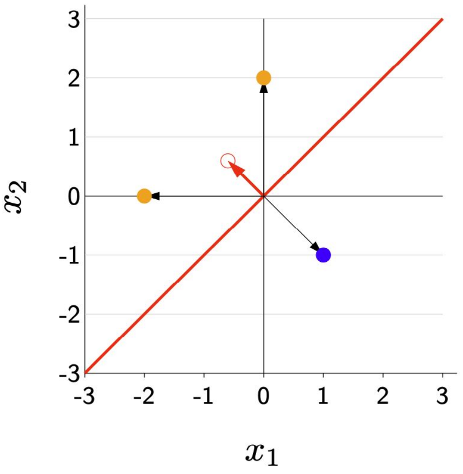
$f(x) = \text{sign}([-0.6, 0.6] \cdot \phi(x))$

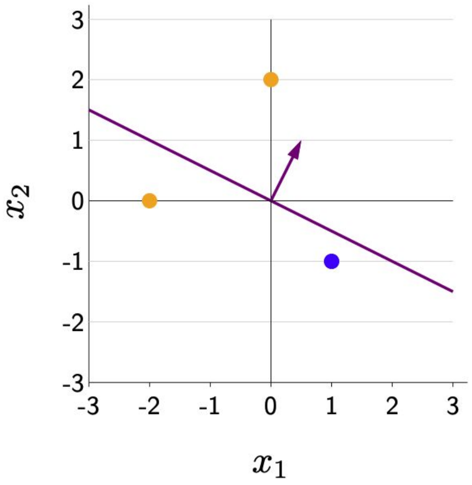
$f(x) = \text{sign}([0.5, 1] \cdot \phi(x))$

General binary classifier: $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

Hypothesis class: $\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^2\}$

Zero-one Loss





training data $\mathcal{D}_{\text{train}}$

x_1	x_2	y
0	2	1
-2	0	1
1	-1	-1

y is the target value.

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$
$$\mathbf{w} = [0.5, 1]$$
$$\phi(x) = [x_1, x_2]$$

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$

- Remember, we want to minimize the loss (better if close to 0).
- 1 if it's incorrect, and 0 if correct.
- Total loss is the average over all training samples.

$$\text{Loss}([0,2], 1, [0.5,1]) = \mathbf{1}[\text{sign}([0.5,1] \cdot [0,2]) \neq 1] = 0$$

Incorrectly classified

$$\text{Loss}([-2,0], 1, [0.5,1]) = \mathbf{1}[\text{sign}([0.5,1] \cdot [-2,0]) \neq 1] = 1$$

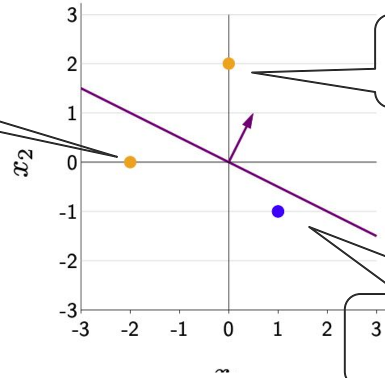
$$\text{Loss}([1,-1], -1, [0.5,1]) = \mathbf{1}[\text{sign}([0.5,1] \cdot [1,-1]) \neq -1] = 0$$

$$\text{TrainLoss}([0.5, 1]) = 1/3 = 0.33$$

Score and margin

Negative, medium **margin**

Positive, large **margin**



Positive, small **margin**

Predicted label: $f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

Target label: y

Definition: margin

The margin on an example (x, y) is $(\mathbf{w} \cdot \phi(x))y$, how **correct** we are.

Text
Text

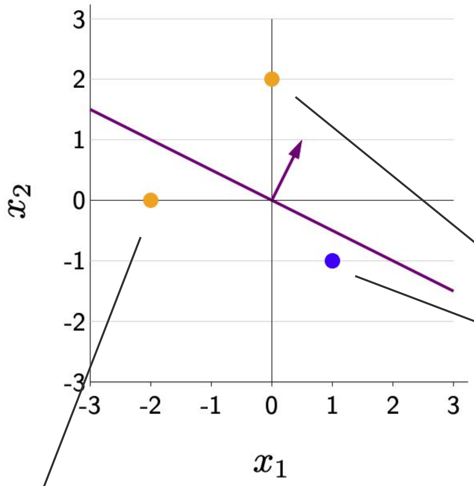
Text

Margin example

training data $\mathcal{D}_{\text{train}}$

x_1	x_2	y
0	2	1
-2	0	1
1	-1	-1

Review margin



Margin($\mathbf{w} \cdot \phi(x)$, y)

Margin([0.5, 1]·[x1,x2],y)

Correct by a large (positive) margin

Margin([0.5, 1]·[0,2],1) = [0.5,1][0,2](1)= **2**

Margin([0.5, 1]·[1,-1],-1) = [0.5,1][1,-1](-1)= **0.5**

Margin([0.5, 1]·[-2,0],1) = [0.5,1][-2,0](1)= **-1**

Negative margin when **incorrectly** classified

Zero-one loss with margin

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$

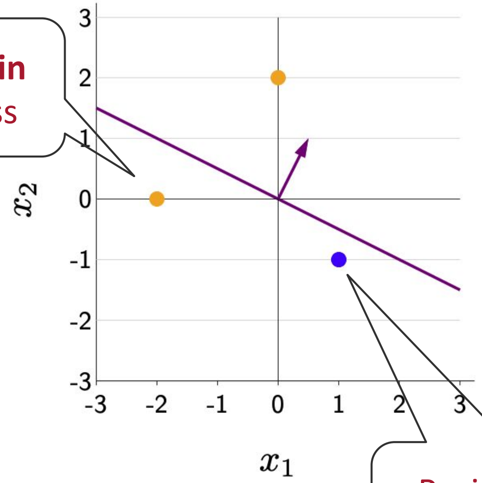
1 if it's incorrect, and
0 if correct.

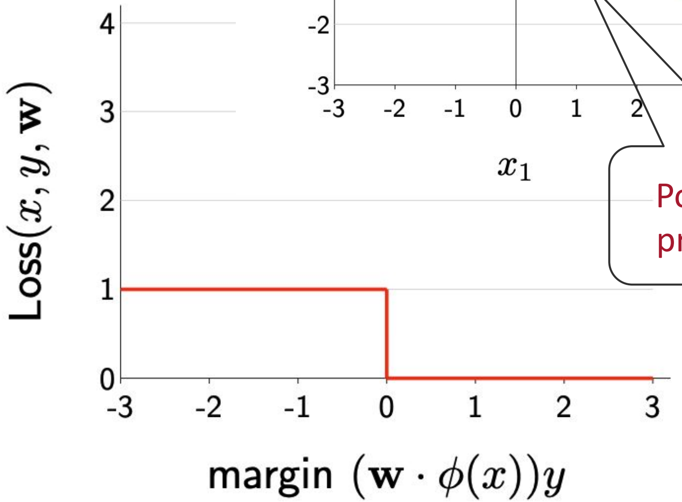
Rewriting the 0-1 loss in terms of **margin**

Definition: zero-one loss

$$\begin{aligned}\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) &= \mathbf{1}[f_{\mathbf{w}}(x) \neq y] \\ &= \mathbf{1}[\underbrace{(\mathbf{w} \cdot \phi(x))y}_{\text{margin}} \leq 0]\end{aligned}$$

Negative **margin** produces 1 loss





Positive **margin** produces 0 loss

Gradient descent: Problem with zero-one loss

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[f_{\mathbf{w}}(x) \neq y]$$
$$= \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$

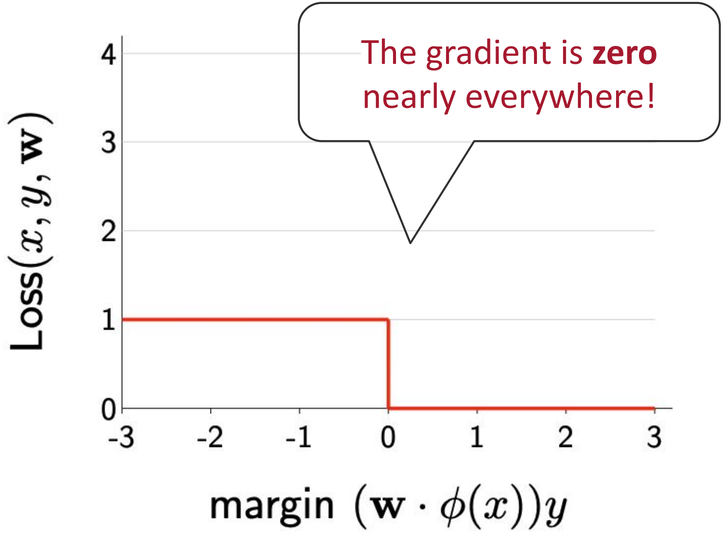
If you do this, you will not converge to a specific value.
This loss function does not give a nice differentiable function.

Goal: $\min_{\mathrm{w}}$ TrainLoss(**w**)
To run gradient descent, compute the gradient:

$$\nabla_{\mathbf{w}} \text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \nabla \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$

Gradient of training loss: average over the sample losses.

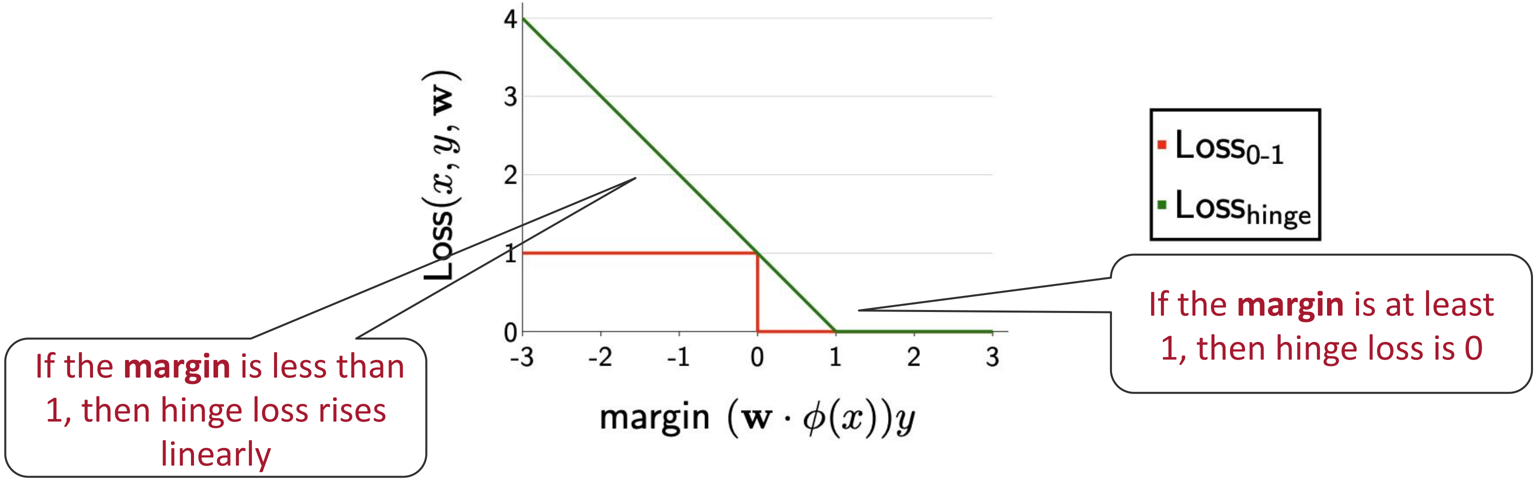
$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \nabla \text{Loss}_{0\text{-}1}(x, y, \mathbf{w})$$



Hinge Loss

What does hinge loss represent?

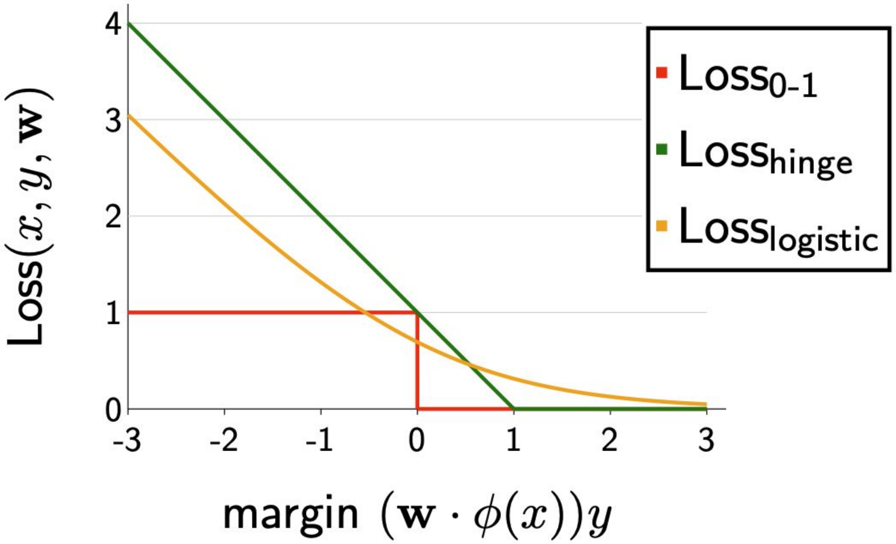
$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$



Intuition: The 1 is to provide buffer to predict not only correctly but with a positive margin of safety. The 1 ~ regularization strength.

Logistic regression

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w} \cdot \phi(x))y})$$



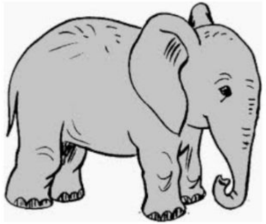
Intuition: Try to increase margin even when it already exceeds 1

Supervised Learning

Stochastic Gradient Descent

Gradient Descent is Slow

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Problem: each iteration requires going over all training examples — expensive when have lots of data!

Stochastic gradient descent

Review stochastic gradient descent.

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Stochastic gradient descent gets to a solution quicker since
your weight parameter decreases more quickly, reaching a gradient of zero more quickly

What is the difference between this algorithm and the previous algorithm for gradient descent?

You still go over all the samples, no?

Algorithm: stochastic gradient descent

You end up reusing the same data points. How does this help?

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

For $(x, y) \in \mathcal{D}_{\text{train}}$:

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

The step size adjusts how much the loss function affects the weight?

Step size

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Question: what should η be?

$$\underset{\text{conservative, more stable}}{0} \longrightarrow \underset{\text{aggressive, faster}}{1} \quad \eta$$

Strategies:

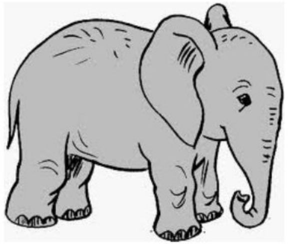
- Constant: $\eta = 0.1$
- Decreasing: $\eta = 1/\sqrt{\#\text{ updates made so far}}$

Start with larger step sizes, and decrease over time

Is this to avoid overshooting the minimum?

Summary

$$\textsf{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\textsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\textsf{train}}} \textsf{Loss}(x, y, \mathbf{w})$$



gradient descent

stochastic gradient descent

Key idea: stochastic updates

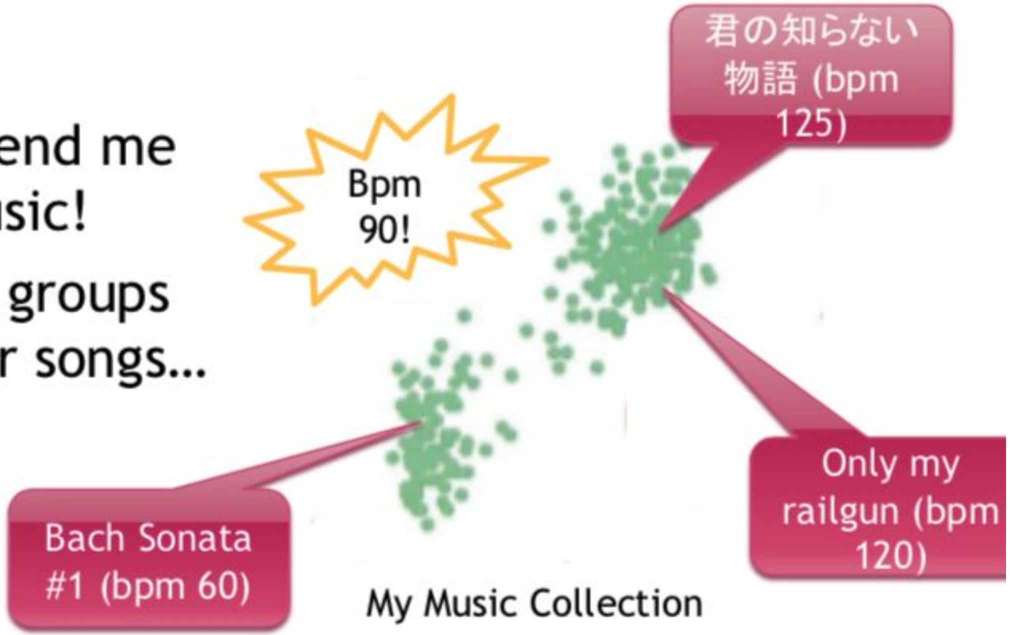
It's not about **quality**, it's about **quantity**.

Unsupervised Learning

K-Means Clustering

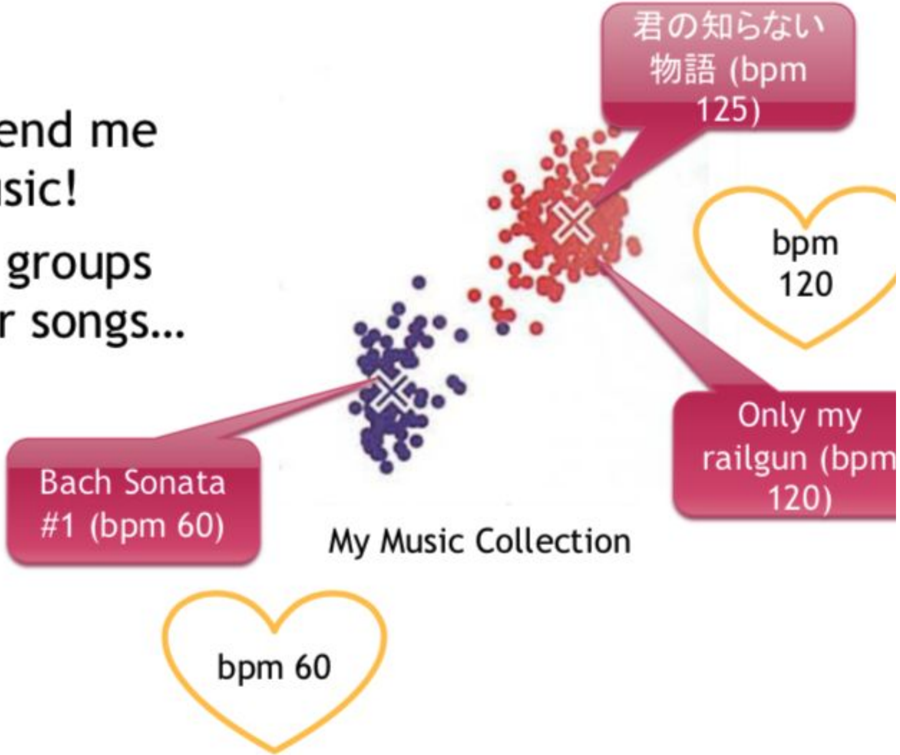
A Clustering Problem

- Recommend me some music!
- Discover groups of similar songs…



A Clustering Problem

- Recommend me some music!
- Discover groups of similar songs…

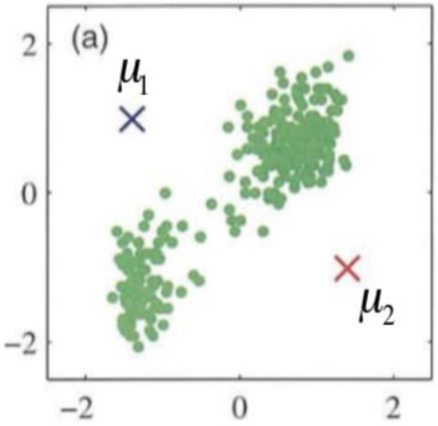


Unsupervised Learning

K-Means Clustering

K-Means Algorithm

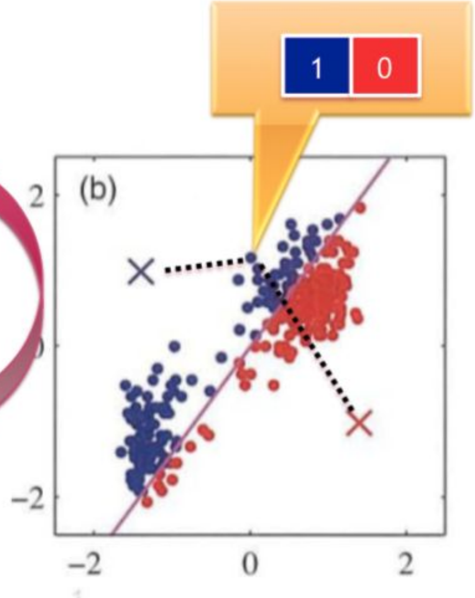
1. Initialize K "means" μ_k, one for each class
   - Eg. Use random starting points, or choose k random points from the set



K=2

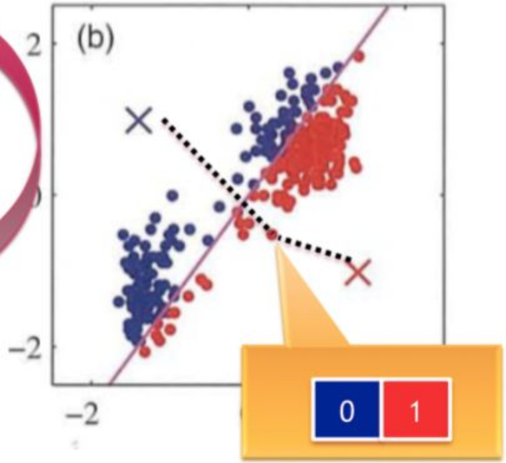
K-Means Algorithm

2. **Phase 1: Assign each point to closest mean** μ_k
3. Phase 2: Update means of the new clusters



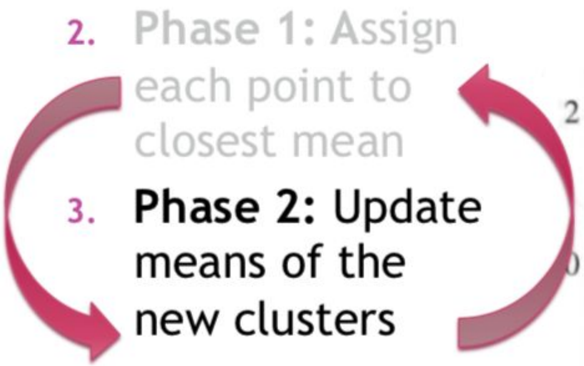
K-Means Algorithm

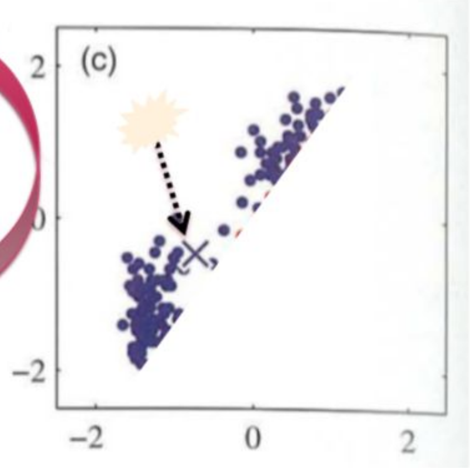
2. **Phase 1: Assign each point to closest mean μ_k**
3. Phase 2: Update means of the new clusters



When you update the means of the new clusters, how are we doing this?

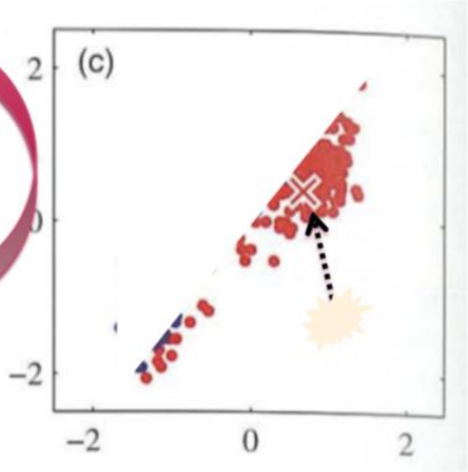
K-Means Algorithm





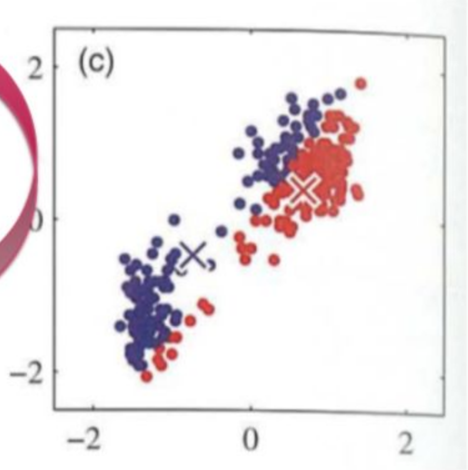
K-Means Algorithm

2. **Phase 1:** Assign each point to closest mean
3. **Phase 2:** Update means of the new clusters



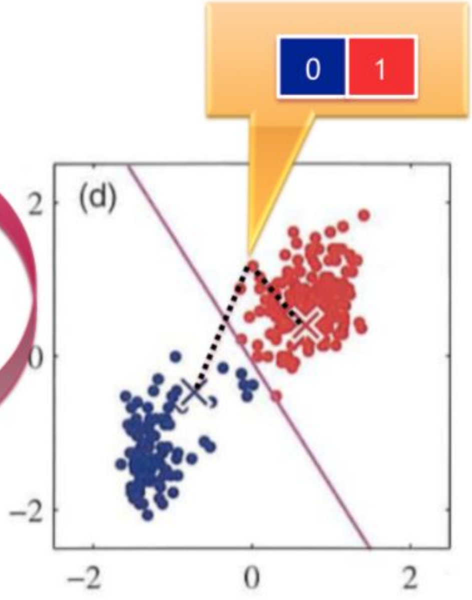
K-Means Algorithm

2. Phase 1: Assign each point to closest mean
3. **Phase 2:** Update means of the new clusters



K-Means Algorithm

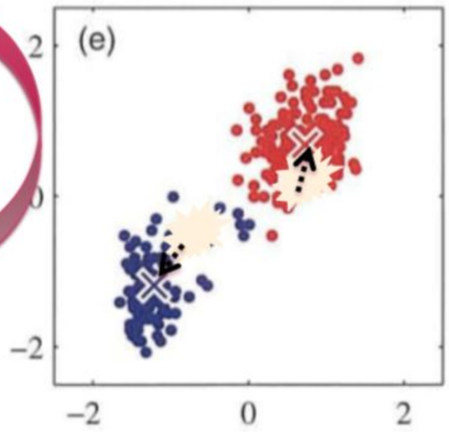
2. **Phase 1: Assign each point to closest mean μ_k**
3. Phase 2: Update means of the new clusters



Review K-means algorithm

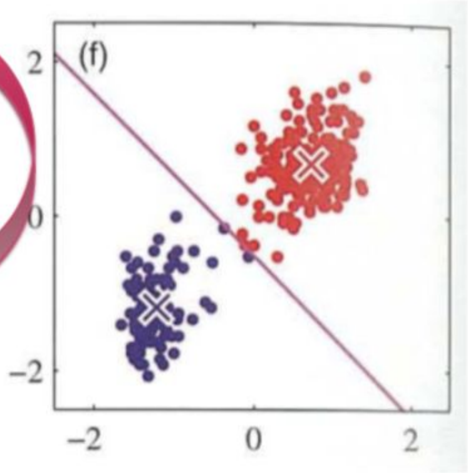
K-Means Algorithm

2. Phase 1: Assign each point to closest mean
3. **Phase 2:** Update means of the new clusters



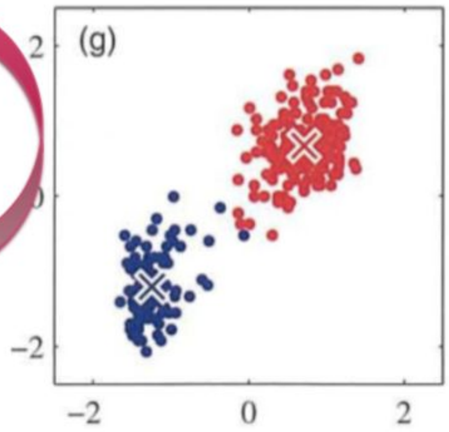
K-Means Algorithm

2. **Phase 1: Assign** each point to closest mean μ_k
3. **Phase 2: Update** means of the new clusters



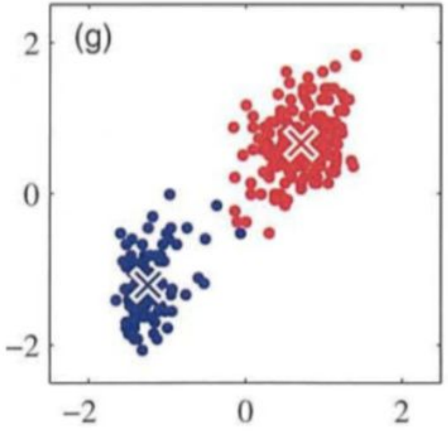
K-Means Algorithm

2. **Phase 1:** Assign each point to closest mean
3. **Phase 2:** Update means of the new clusters



K-Means Algorithm

4. When means do not change anymore → clustering DONE.



Problem with K-Means

- In K-means, a point can only have 1 class
- But what about points that lie in between groups? eg. Jazz + Classical