# CSCI 567: Machine Learning

Vatsal Sharan

Fall 2022

Lecture 3, Sep 8

USC University of Southern California

## Administrivia

- HW2 due in about 1 week (9/14 at 2pm).
- Each person gets 1 late day, if you want to use a late day we'll ask to fill a form (late day counts towards the group member filling form).
- Max 1 late day per HW.
- Please submit your groups by end of today (form on Ed Discussion post by Rachitha).

## Recap

## Supervised learning in one slide

| | |
|---|---|
| **Loss function:** | What is the right loss function for the task? |
| **Representation:** | What class of functions should we use? |
| **Optimization:** | How can we efficiently solve the empirical risk minimization problem? |
| **Generalization:** | Will the predictions of our model transfer gracefully to unseen examples? |

***All related! And the fuel which powers everything is data.***

## Summary: **Optimization** methods

- **GD/SGD** is a first-order optimization method.

- GD/SGM coverages to a stationary point. For convex objectives, this is all we need. For nonconvex objectives, it is possible to get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points).

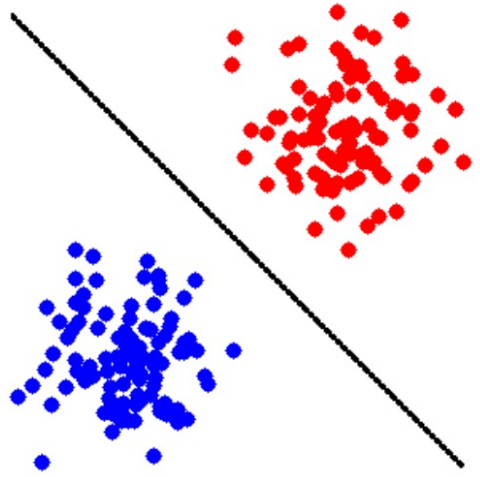
- **Newton’s method** is a second-order optimization method.

- Newton’s method has a much faster convergence rate, but each iteration also takes much longer. Usually for large scale problems, GD/SGD and their variants are the methods of choice.

# Linear classifiers

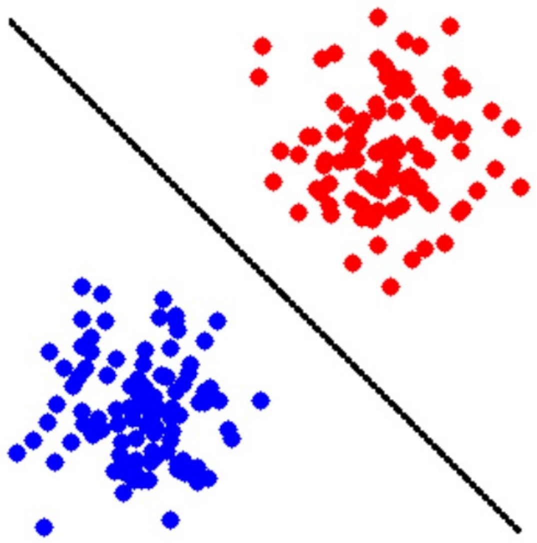
Binary classification:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{d}}$
- output (label): $y \in \{-1, +1\}$.
- goal: learn a mapping $f : \mathbb{R}^{\mathsf{d}} \to \{-1, +1\}$



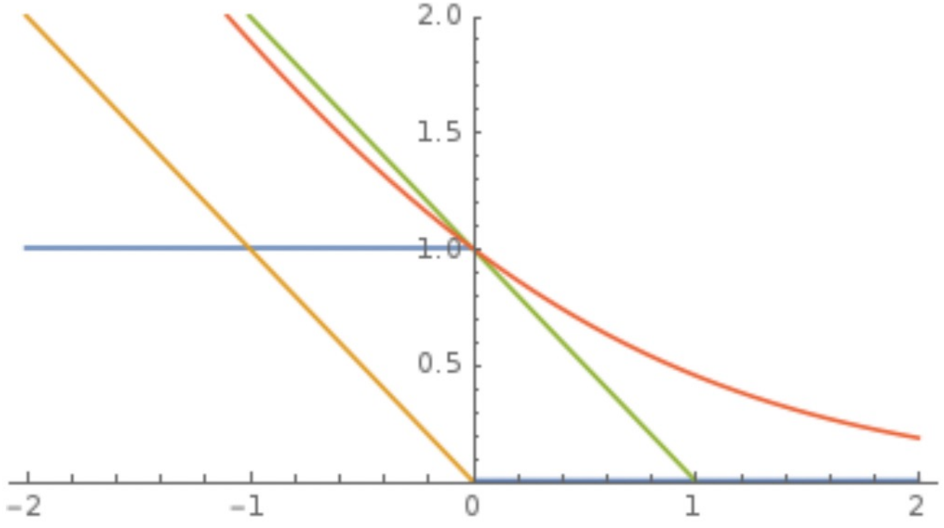
## Representation

Definition: The **function class of separating hyperplanes** is defined as
$\mathcal{F} = \{f(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x}) : \boldsymbol{w} \in \mathbb{R}^d\}$.



## Loss function

Use a **convex 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; the base of $\log$ doesn't matter)

## Optimization

Empirical risk minimization (ERM) problem:

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{d}}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)$$

Solve using a suitable optimization algorithm:

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

## **Maximum likelihood** estimation

*What we observe are labels, not probabilities.*

Take a **probabilistic view**

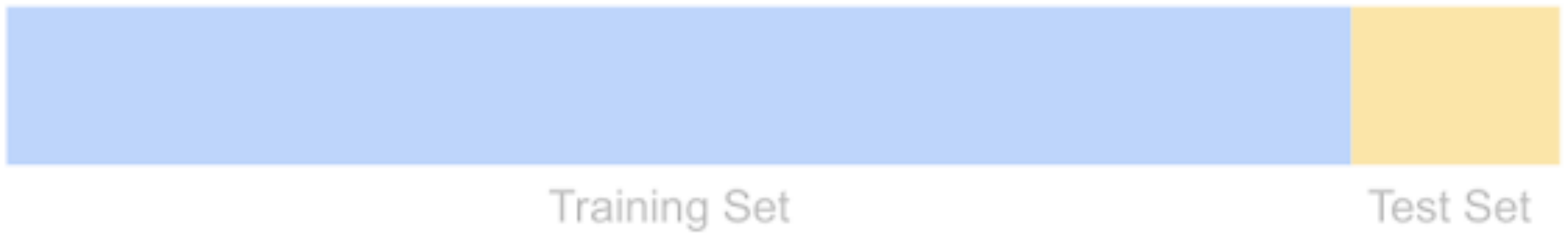
- assume data is independently generated in this way by some $\boldsymbol{w}$
- perform Maximum Likelihood Estimation (MLE)

Specifically, what is the probability of seeing labels $y_1, \cdots, y_i$ given $\boldsymbol{x_1}, \cdots, \boldsymbol{x_i}$, as a function of some $\boldsymbol{w}$?

$$P(\boldsymbol{w}) = \prod_{i=1}^{n} \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w})$$

**MLE**: find $\boldsymbol{w}^*$ that **maximizes the probability** $P(\boldsymbol{w})$

Minimizing logistic loss is exactly doing MLE for the sigmoid model!



## Generalization

## Reviewing definitions

- Input space: $\mathcal{X}$
- Output space: $\mathcal{Y}$
- Predictor: $f(\boldsymbol{x}) : \mathcal{X} \to \mathcal{Y}$
- Distribution $D$ over $(\boldsymbol{x}, y)$.
- Let $D^n$ denote the distribution of $n$ samples $\{(\boldsymbol{x}_i, y_i), i \in [n]\}$ drawn i.i.d. from $D$.
- Risk of a predictor $f(\boldsymbol{x})$ is $R(f) \ = \mathbb{E}_{(\boldsymbol{x},y)\sim D}\Big[\ell(f(\boldsymbol{x}), y)\Big]$
- Consider the 0-1 loss, $\ell(f(\boldsymbol{x}, y)) = \mathbb{1}(f(\boldsymbol{x}) \neq y)$.

*The analysis we'll do could also help you solve Problem 3 on HW1.*

## Assumptions for today’s theory

## Intuition: When does ERM generalize?

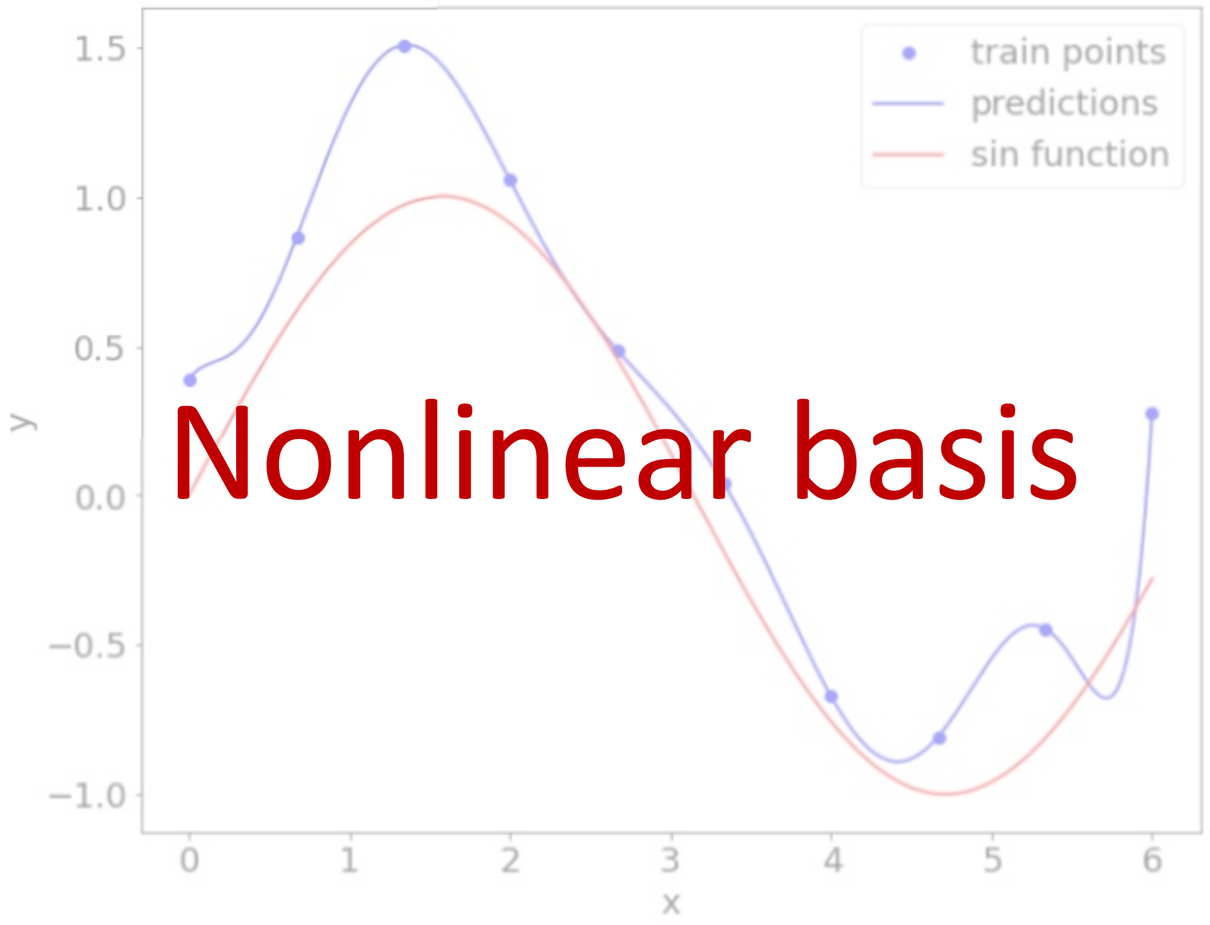
## Relaxing our assumptions

- We assumed that the function class is finite-sized. Results can be extended to **infinite function classes** (such as separating hyperplanes).
- We considered 0-1 loss. Can extend to **real-valued loss** (such as for regression).
- We assumed realizability. Can prove similar theorem which guarantees small generalization gap **without realizability** (but with an $\epsilon^2$ instead of $\epsilon$ in the denominator). This is called agnostic learning.

## Rule of thumb for generalization

Suppose the functions $f$ in our function class $\mathcal{F}$ have $d$ parameters which can be set.
Assume we discretize these parameters so they can take $W$ possible values each.
How much data do we need to have small generalization gap?

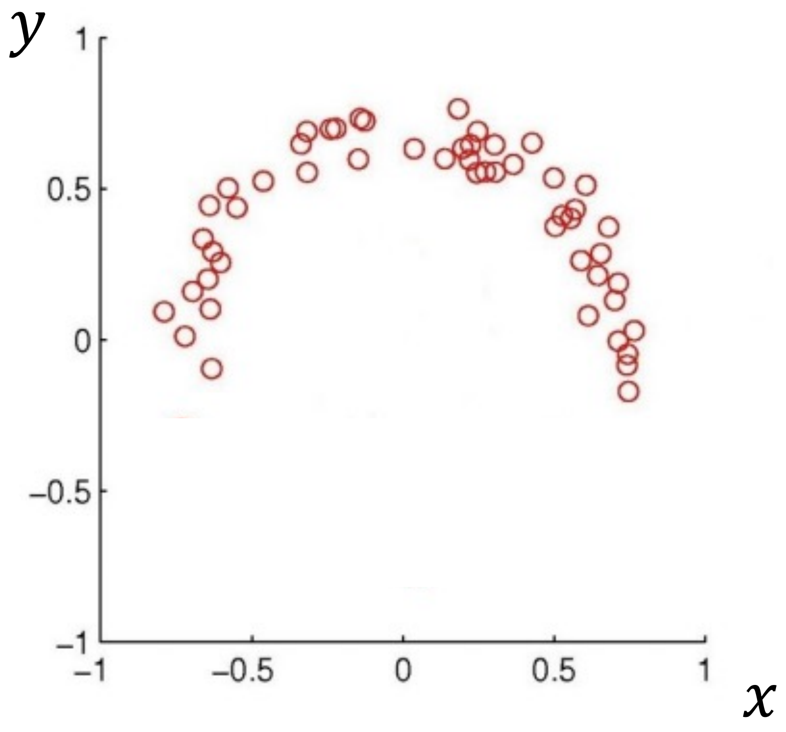
A useful rule of thumb: to guarantee generalization, make sure that your training data set size $n$ is at least linear in the number $d$ of free parameters in the function that you're trying to learn.



# Nonlinear basis

# What if a linear model is not a good fit?

Let's go back to the regression setup (output $\mathcal{Y} \in R$).
A linear model could be a bad fit for the following data:



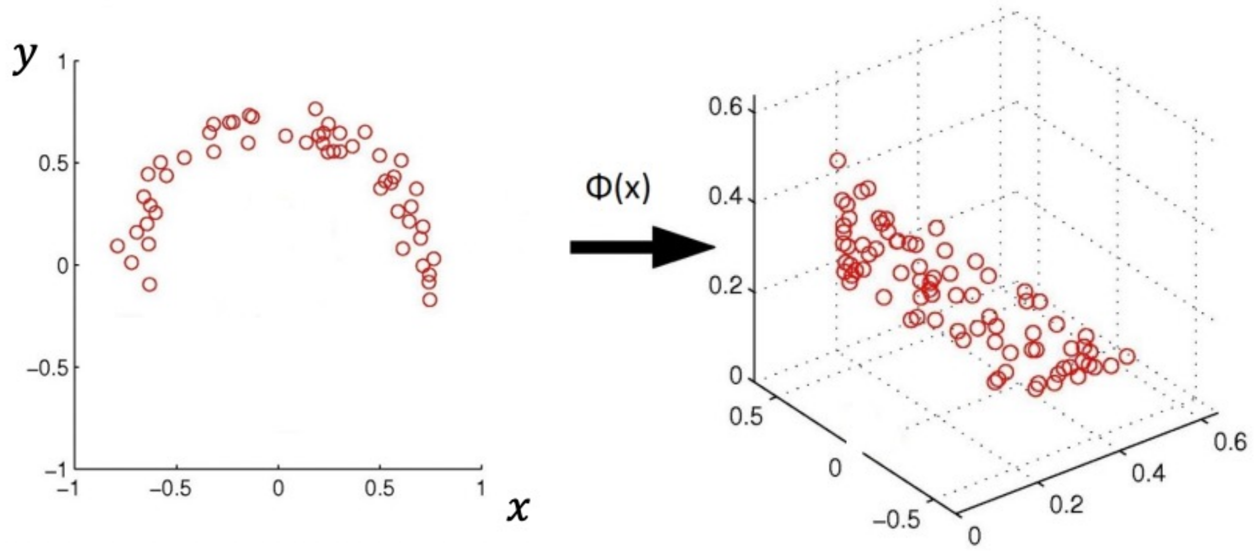
## A solution: nonlinearly transformed features

**1. Use a nonlinear mapping**

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

**2. Then apply linear regression** (hope: linear model is a better fit for the new feature space).



## A solution: nonlinearly transformed features

**1. Use a nonlinear mapping**

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

**2. Then apply linear regression** (hope: linear model is a better fit for the new feature space).

## Regression with nonlinear basis

**Model:** $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^{M}$

**Objective:**

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i\right)^2$$

**Similar least square solution:**

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \quad \text{where} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{n \times M}$$

## Example

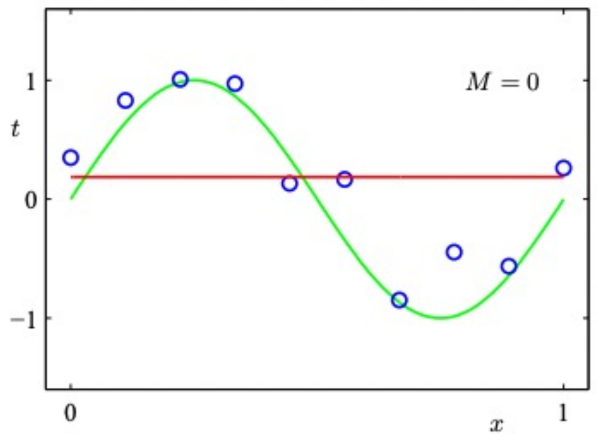
**Polynomial basis functions for $d = 1$**

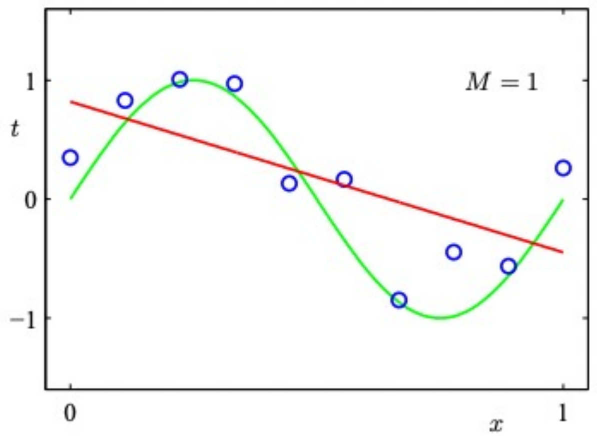
$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

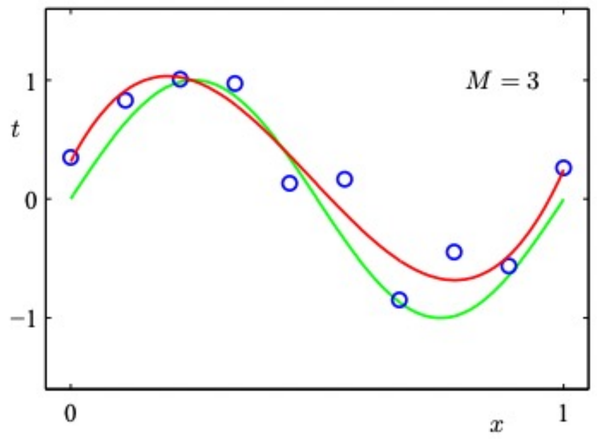
Learning a linear model in the new space
$=$ learning an *$M$-degree polynomial model* in the original space

## Example

**Fitting a noisy sine function with a polynomial ($M = 0, 1,$ or $3$):**







See Colab notebook

## Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times d}$$

No, it basically *does nothing* since

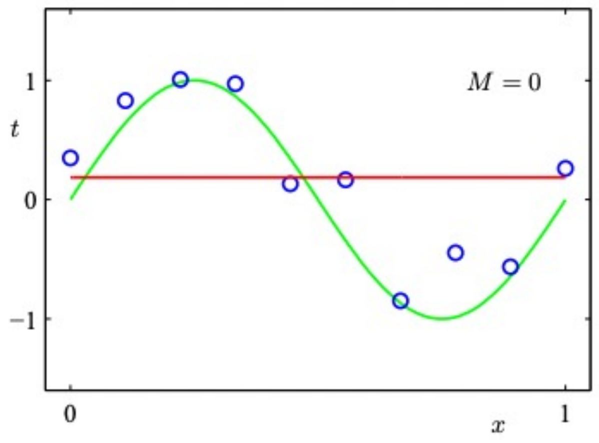
$$\min_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{M}}} \sum_i \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}_i - y_i\right)^2 = \min_{\boldsymbol{w}'\in\mathsf{Im}(\boldsymbol{A}^{\mathrm{T}})\subset\mathbb{R}^d} \sum_i \left(\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x}_i - y_i\right)^2$$

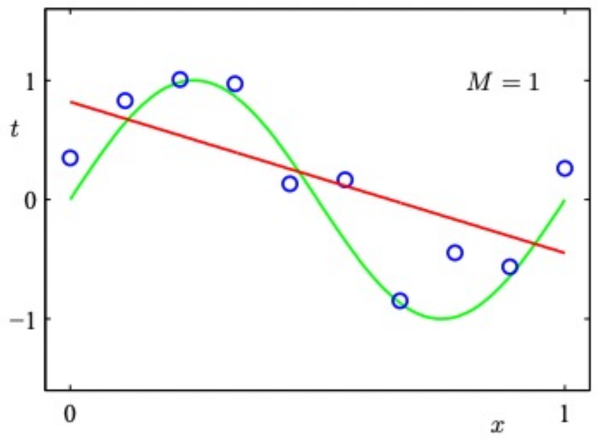
Training
Test
1
0.5
0
$E_{RMS}$
0
3
6
9
$M$

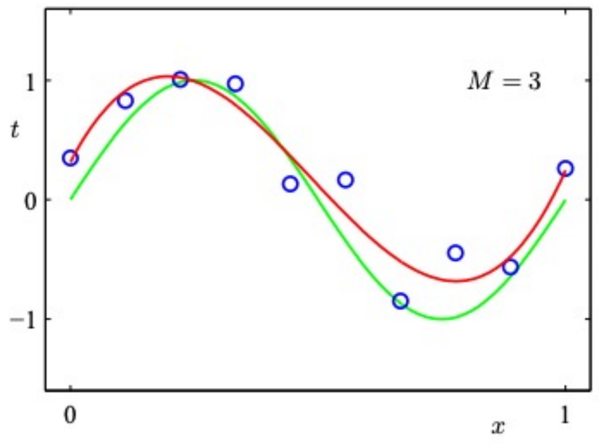
# Overfitting and Regularization

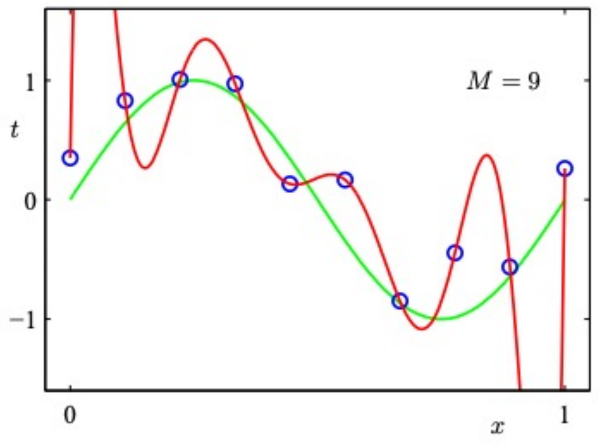
# Should we use a very complicated mapping?

**Ex: fitting a noisy sine function with a polynomial**:









See Colab notebook

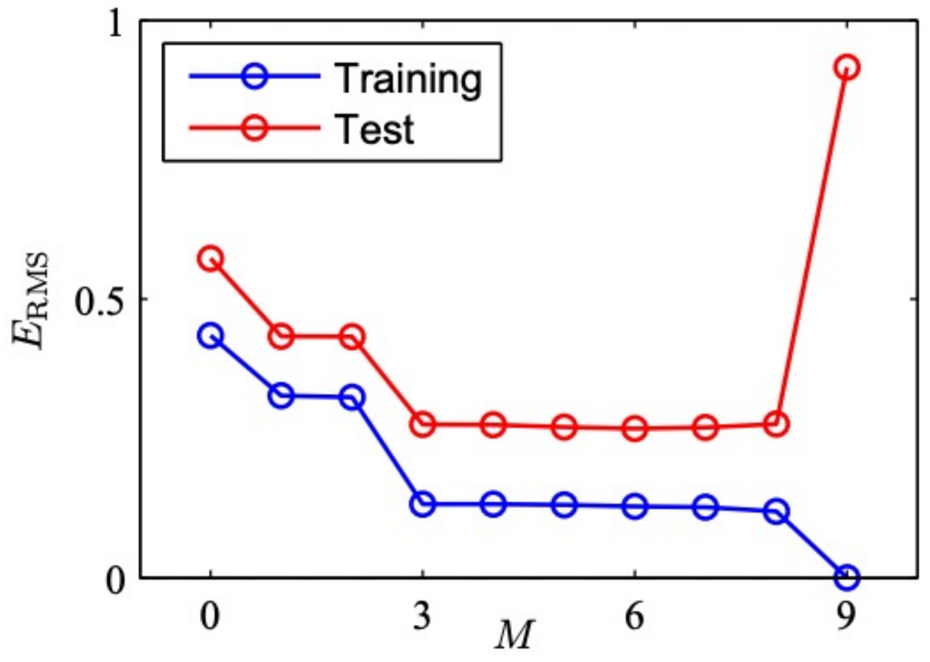
## Underfitting and overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**

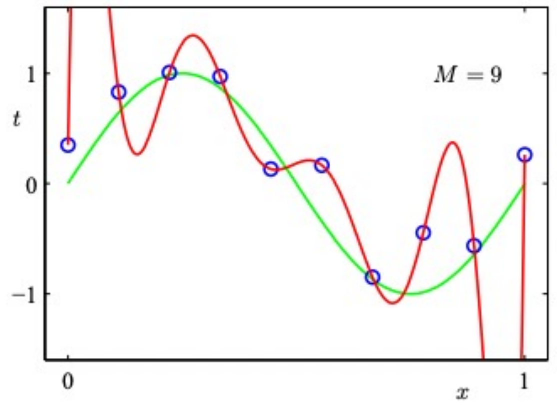


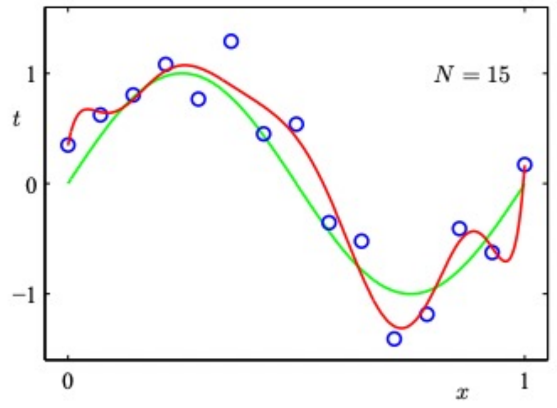
*More complicated models $\Rightarrow$ larger gap between training and test error*

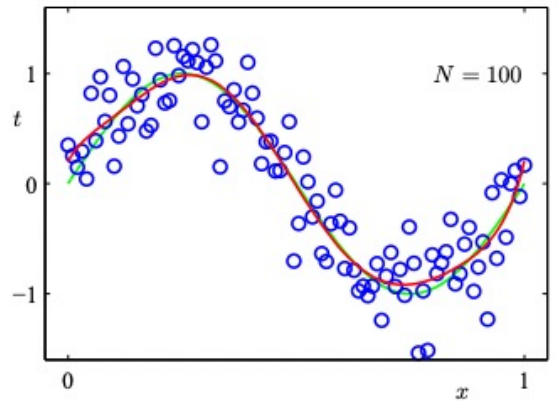
How to prevent overfitting?

See Colab notebook

## Method 1: **More data!!**







*More data $\Rightarrow$ smaller gap between training and test error*

See Colab notebook

## Method 2: Control model complexity

For polynomial basis, the **degree** $M$ clearly controls the complexity

- use **cross-validation** to pick hyper-parameter $M$

Cross-validation: Explored in HW1. Idea is to do a three-way split in addition to training set/test set, and tune hyperparameters on a *validation set*.

When $M$ or in general $\Phi$ is fixed, are there still other ways to control complexity?

## Magnitude of the weights

Least square solution for the polynomial example:

| | $M=0$ | $M=1$ | $M=3$ | $M=9$ |
|---|---|---|---|---|
| $w_0$ | 0.19 | 0.82 | 0.31 | 0.35 |
| $w_1$ | | -1.27 | 7.99 | 232.37 |
| $w_2$ | | | -25.43 | -5321.83 |
| $w_3$ | | | 17.37 | 48568.31 |
| $w_4$ | | | | -231639.30 |
| $w_5$ | | | | 640042.26 |
| $w_6$ | | | | -1061800.52 |
| $w_7$ | | | | 1042400.18 |
| $w_8$ | | | | -557682.99 |
| $w_9$ | | | | 125201.43 |

Intuitively, **large weights $\Rightarrow$ more complex model**

See Colab notebook

## How to make the weights small?

**Regularized linear regression**: new objective

$$G(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\psi(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \operatorname{argmin}_w G(\boldsymbol{w})$

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.

## How to make the weights small?

**Regularized linear regression**: new objective

$$G(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\psi(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \mathrm{argmin}_w\, G(\boldsymbol{w})$

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.
- $\lambda > 0$ is the *regularization coefficient*
  - $\lambda = 0$, no regularization
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \mathrm{argmin}_w\, \psi(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity

## $\ell_2$ regularization with non-linear basis: The effect of $\lambda$

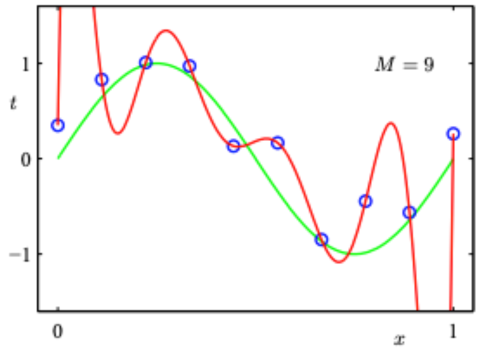
**when we increase regularization coefficient $\lambda$**

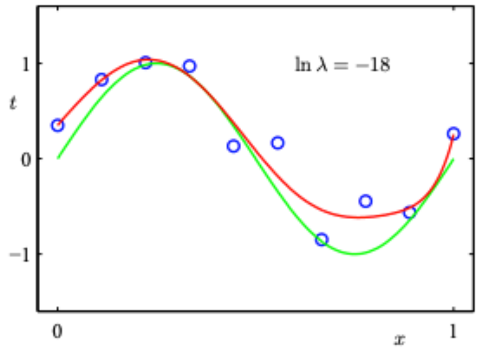
| | $\ln\lambda = -\infty$ | $\ln\lambda = -18$ | $\ln\lambda = 0$ |
|---|---|---|---|
| $w_0$ | 0.35 | 0.35 | 0.13 |
| $w_1$ | 232.37 | 4.74 | -0.05 |
| $w_2$ | -5321.83 | -0.77 | -0.06 |
| $w_3$ | 48568.31 | -31.97 | -0.06 |
| $w_4$ | -231639.30 | -3.89 | -0.03 |
| $w_5$ | 640042.26 | 55.28 | -0.02 |
| $w_6$ | -1061800.52 | 41.32 | -0.01 |
| $w_7$ | 1042400.18 | -45.95 | -0.00 |
| $w_8$ | -557682.99 | -91.53 | 0.00 |
| $w_9$ | 125201.43 | 72.68 | 0.01 |

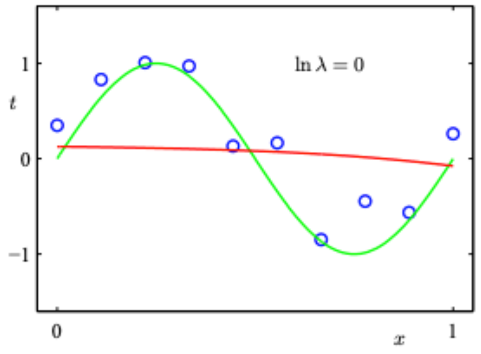
See Colab notebook

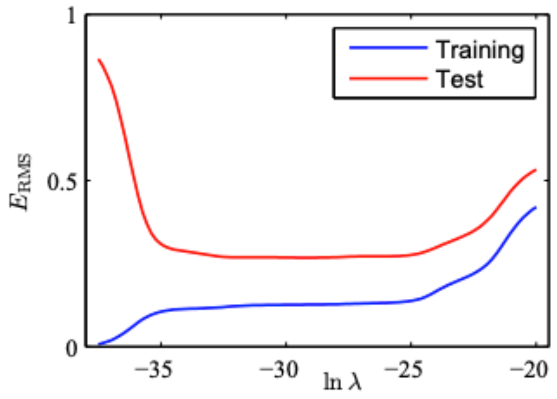
# $\ell_2$ regularization with non-linear basis : A **tradeoff**

**when we increase regularization coefficient $\lambda$**









See Colab notebook

## Why is regularization useful?

If you don’t have sufficient data to fit your more expressive model, then ERM will overfit. **Regularization helps with generalization**.

So should it not be useful in many practical settings, where we have enough data?

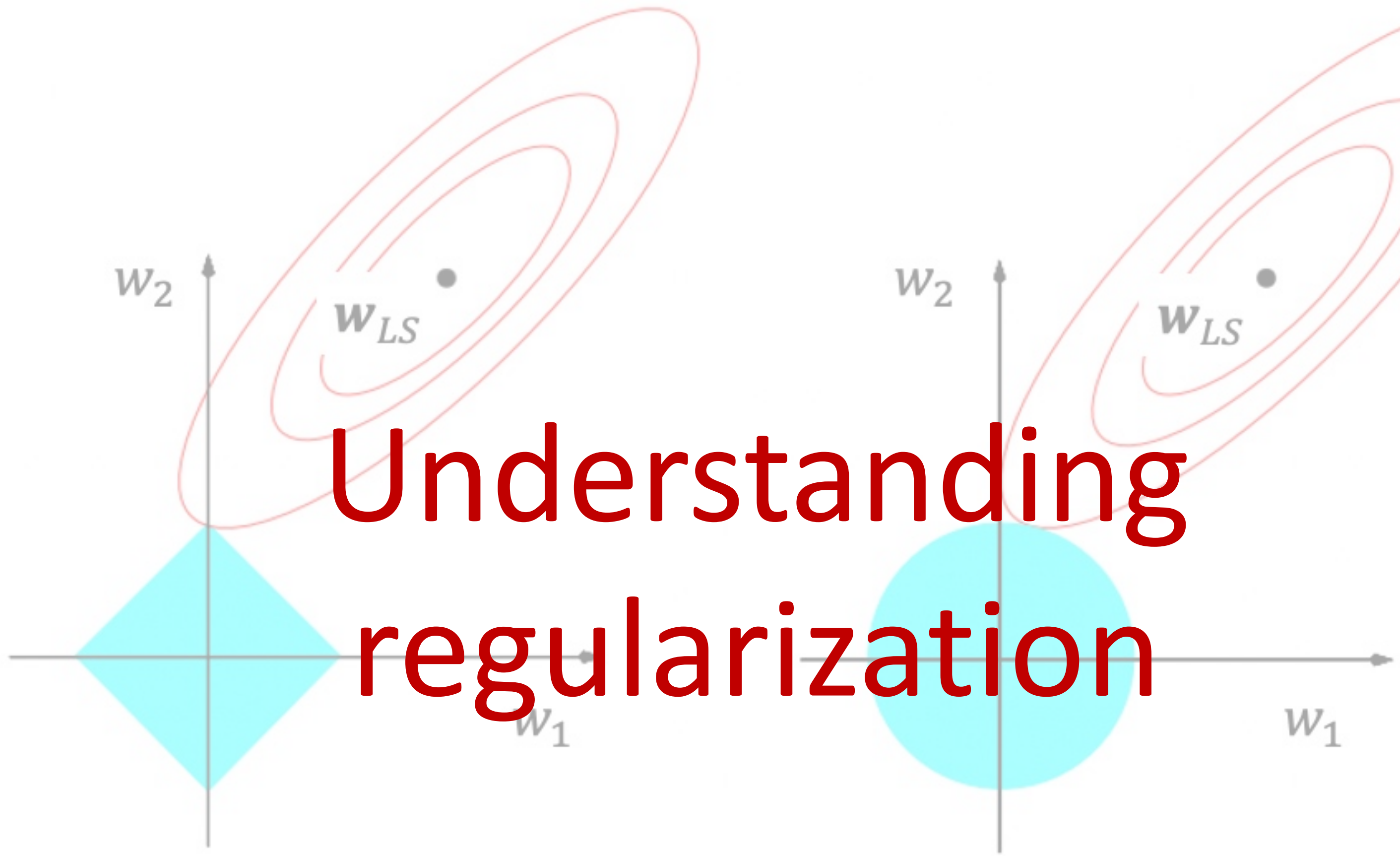
## Why is regularization useful?

If you don’t have sufficient data to fit your more expressive model, then ERM will overfit. **Regularization helps with generalization**.

So should it not be useful in many practical settings, where we have enough data?

In general, a viewpoint is that *we should always be trying to fit a more expressive model if possible*. We want our function class to be rich enough that we could almost overfit if we are not careful.

Since we’re often in this regime where the models we want to fit are more and more complex, regularization is very useful to help generalization (it’s also a relatively simple knob to control).



## Understanding regularization

## How to solve the regularized objective $G(\boldsymbol{w})$?

Let’s go back to the original linear model.

**Simple for $\ell_2$ regularization, $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:**

$$G(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\begin{aligned}
&\nabla G(\boldsymbol{w}) = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0 \\
&\Rightarrow \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \\
&\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}
\end{aligned}$$

Linear regression with $\ell_2$ regularization is also known as **ridge regression**.

For other regularizers, as long as it's **convex**, standard optimization algorithms can be applied.

## Aside: Least-squares when $\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$ is not invertible

## Aside: Least-squares when $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible

**Intuition:** what does inverting $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ do?

**eigendecomposition**: $$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_d & 0 \\ 0 & \cdots & 0 & \lambda_{d+1} \end{bmatrix} \boldsymbol{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{d+1} \geq 0$ are **eigenvalues**.

**inverse**: $$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_d} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{d+1}} \end{bmatrix} \boldsymbol{U}$$

*i.e. just invert the eigenvalues*

## Aside: Least-squares when $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible

Non-invertible $\Rightarrow$ some eigenvalues are 0.

**One natural fix: add something positive**

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 + \lambda & 0 & \cdots & 0 \\ 0 & \lambda_2 + \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_d + \lambda & 0 \\ 0 & \cdots & 0 & \lambda_{d+1} + \lambda \end{bmatrix} \boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix. Now it is invertible:

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1+\lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2+\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_d+\lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{d+1}+\lambda} \end{bmatrix} \boldsymbol{U}$$

# A “Bayesian view” of $\ell_2$ regularization

**Maximum a posteriori probability (MAP) estimation:** A Bayesian generalization of maximum likelihood estimation (MLE).

Let’s continue with the linear model, and Q3 from the practice problems for today.

## A “Bayesian view” of $\ell_2$ regularization

**Maximum a posteriori probability (MAP) estimation:** A Bayesian generalization of maximum likelihood estimation (MLE).

Let’s continue with the linear model, and Q3 from the practice problems for today.

## A “Bayesian view” of $\ell_2$ regularization

**Maximum a posteriori probability (MAP) estimation:** A Bayesian generalization of maximum likelihood estimation (MLE).

Bayesian view: A **prior** over $\boldsymbol{w}$

## A “Bayesian view” of $\ell_2$ regularization

**Maximum a posteriori probability (MAP) estimation:** A Bayesian generalization of maximum likelihood estimation (MLE).

Bayesian view: A **prior** over $\boldsymbol{w}$

## An equivalent form, and a "Frequentist view"

"Frequentist" approach to justifying regularization is to argue that if the true model has a specific property, then regularization will allow you to recover a good approximation to the true model. We this view, we can equivalently formulate regularization as:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \operatorname{RSS}(w) \qquad \textbf{subject to } \psi(\boldsymbol{w}) \leq \beta$$

where $\beta$ is some hyper-parameter.

Finding the solution becomes a *constrained optimization problem*.

Choosing either $\lambda$ or $\beta$ can be done by cross-validation.

## Encouraging sparsity: $\ell_0$ regularization

Continuing from the frequentist view, having small norm is one possible structure to impose on the model. Another very common one is **sparsity**.

**Sparsity of $w$:** Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_{\mathbf{0}}$

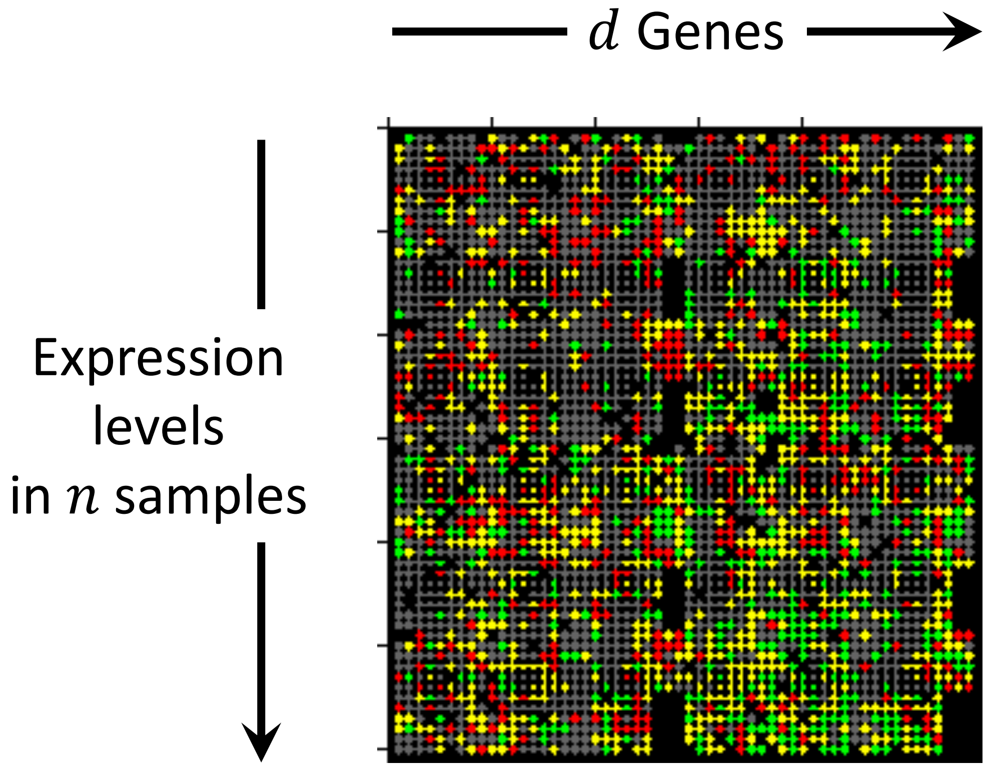
E.g. $\boldsymbol{w} = [1, 0, -1, 0, 0.2, 0, 0\ ]$ is 3-sparse

# Encouraging sparsity: $\ell_0$ regularization

**Sparsity of $w$:** Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.



Suppose we want to fit a linear models from gene expression to an outcome (disease, phenotype etc.).

$d$ is huge, but likely that only a few genes are related.

## Encouraging sparsity: $\ell_0$ regularization

**Sparsity of $w$:** Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_\mathbf{0}$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.
- Sparse models may also be more **interpretable**. They could narrow down a small number of features which carry a lot of signal.

E.g. $\boldsymbol{w} = [1.5, 0, -1.1, 0, 0.25, 0, 0\ ]$ is more interpretable than,
$\boldsymbol{w} = [1, 0.2, -1.3, 0.15, 0.2, 0.05, 0.12\ ]$

For a sparse model, it could be easier to understand the model. It is also easier to verify whether the features which have a high weight have a relation with the outcome (they are not spurious artifacts of the data).

## Encouraging sparsity: $\ell_0$ regularization

**Sparsity of $w$:** Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_{\mathbf{0}}$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.
- Sparse models may also be more **interpretable**. They could narrow down a small number of features which carry a lot of signal.
- Data required to learn sparse model maybe significantly less than to learn dense model.

We'll see more on the third point next.

## $\ell_0$ regularization: The good, the bad and the ugly

## $\ell_0$ regularization: The good, the bad and the ugly

## $\ell_0$ regularization: The good, the bad and the ugly

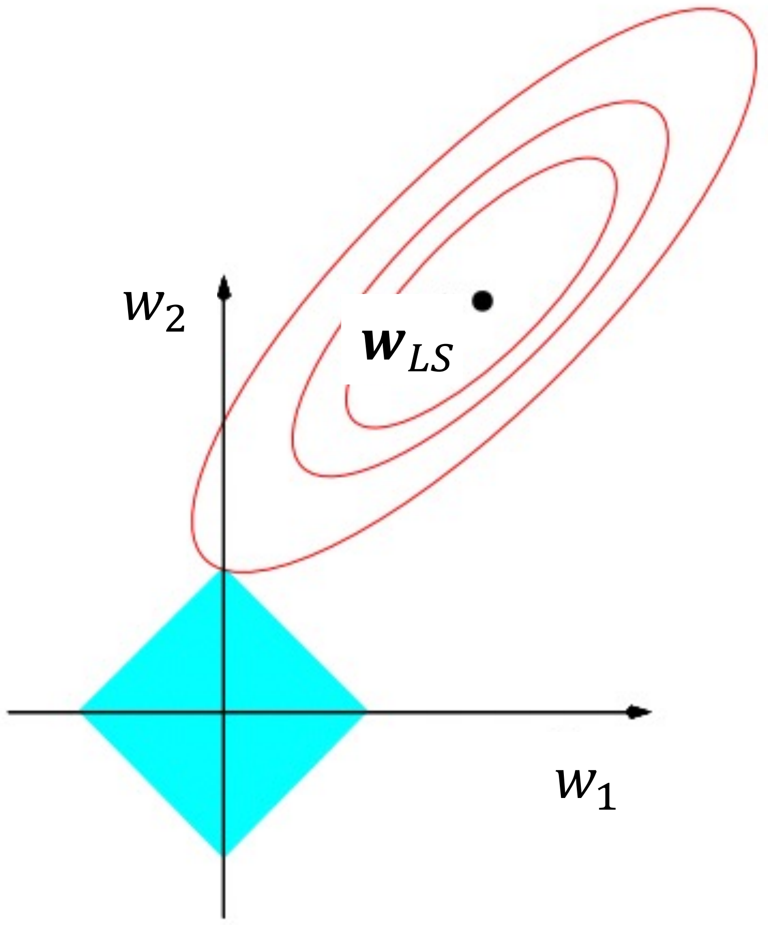
## $\ell_0$ regularization: The good, the bad and the ugly

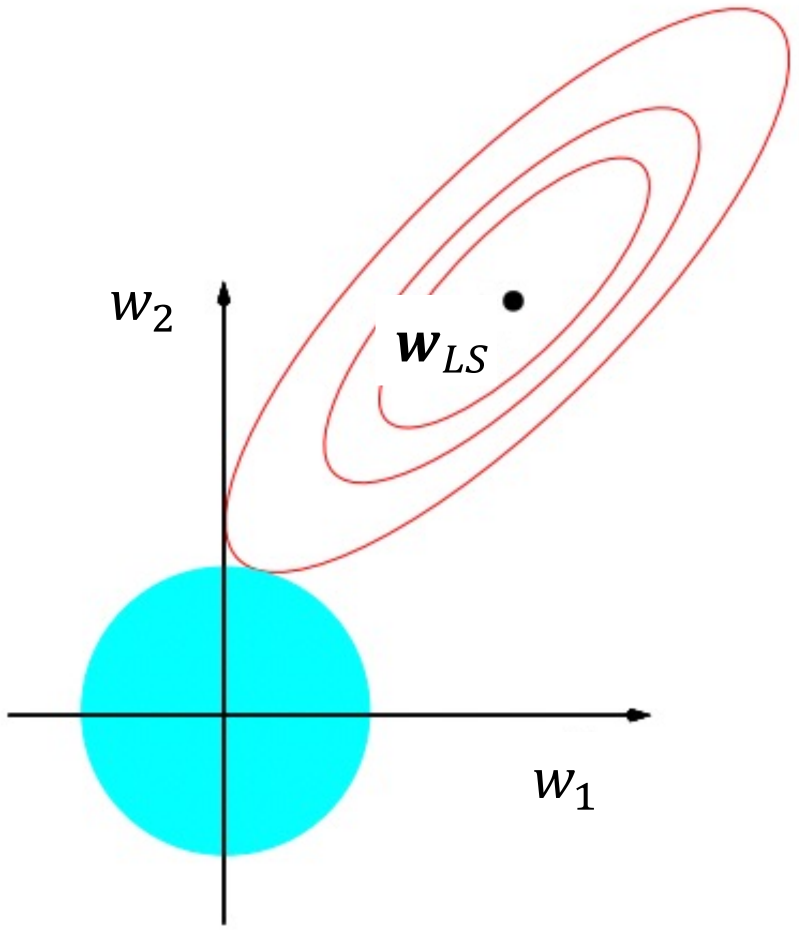
## $\ell_1$ regularization as a proxy for $\ell_0$ regularization

## $\ell_1$ regularization as a proxy for $\ell_0$ regularization

## Why does $\ell_1$ regularization encourage sparse solutions?

Optimization problem: $\text{argmin}_w \text{ RSS}(\boldsymbol{w})$ , subject to $\psi(\boldsymbol{w}) \leq \beta$





Adapted from ESL

## $\ell_1$ and $\ell_2$ regularization for the “isotropic” case

## $\ell_1$ and $\ell_2$ regularization for the "isotropic" case

## $\ell_1$ and $\ell_2$ regularization for the “isotropic” case

## $\ell_1$ and $\ell_2$ regularization for the “isotropic” case

## $\ell_1$ and $\ell_2$ regularization for the “isotropic” case

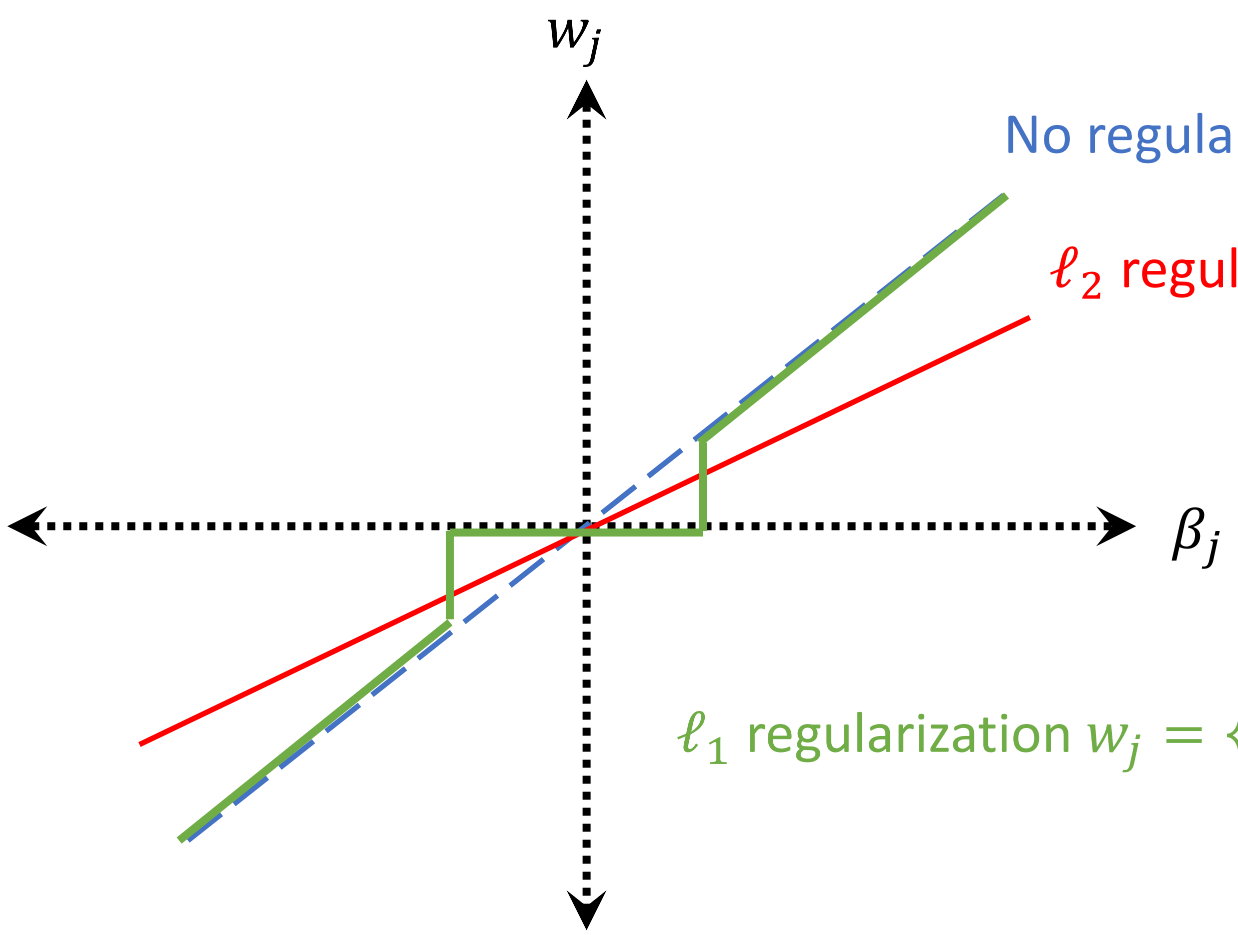
## $\ell_1$ and $\ell_2$ regularization for the “isotropic” case

## $\ell_1$ and $\ell_2$ regularization for the "isotropic" case

# $\ell_1$ and $\ell_2$ regularization for the "isotropic" case

Summary: Isotropic case ($\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}$).

Let $\beta_j = \boldsymbol{X}_{(i)}^T \boldsymbol{y}$



No regularization $w_j = \beta_j$

$\ell_2$ regularization $w_j = \beta_j/(1+\lambda)$

$\ell_1$ regularization $w_j = \begin{cases} \beta_j - \lambda/2, \beta_j > \lambda/2 \\ 0, |\beta_j| \le \lambda/2 \\ \beta_j + \lambda/2, \beta_j < -\lambda/2 \end{cases}$

## Implicit regularization

So far, we explicitly added a $\psi(\boldsymbol{w})$ term to our objective function to regularize.

In many cases, the optimization algorithm we use can themselves act as regularizers, favoring some solutions over others.

Currently a very active area of research, you'll see more in the homework.

# Multiclass classification

## Setup

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^d$
- output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- goal: learn a mapping $f : \mathbb{R}^d \to [\mathsf{C}]$

**Examples**:

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

## Linear models: 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 $\{1, 2\}$)

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

can be written as

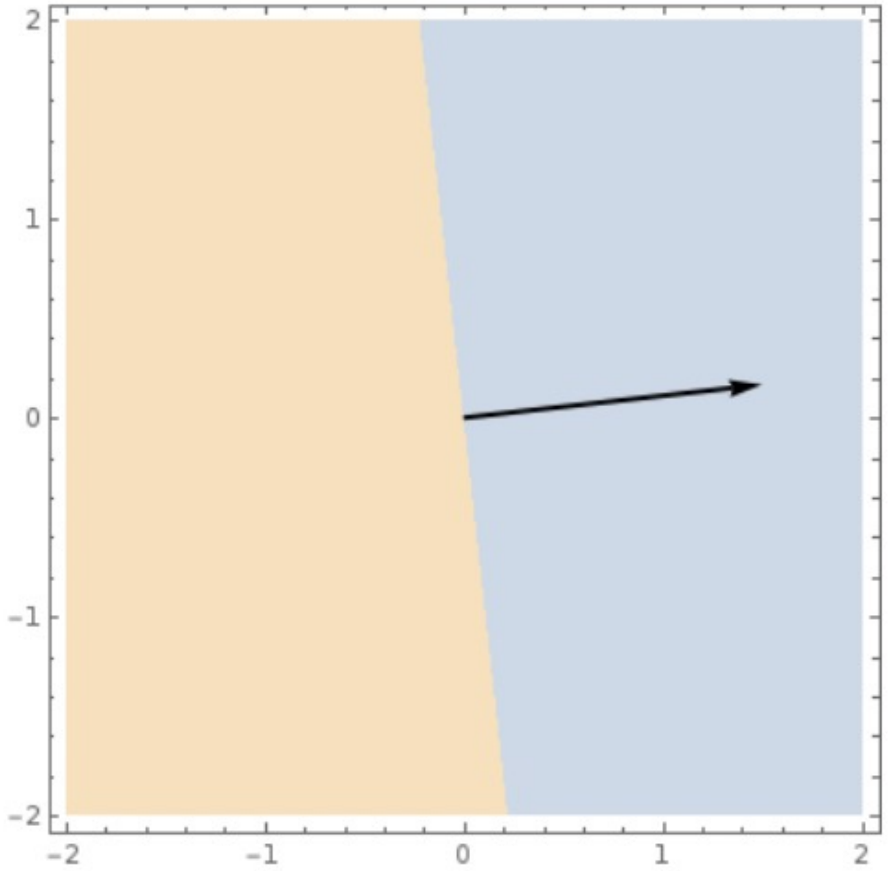
$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \geq \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} \\ 2 & \text{if } \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} > \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \end{cases}$$

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

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

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

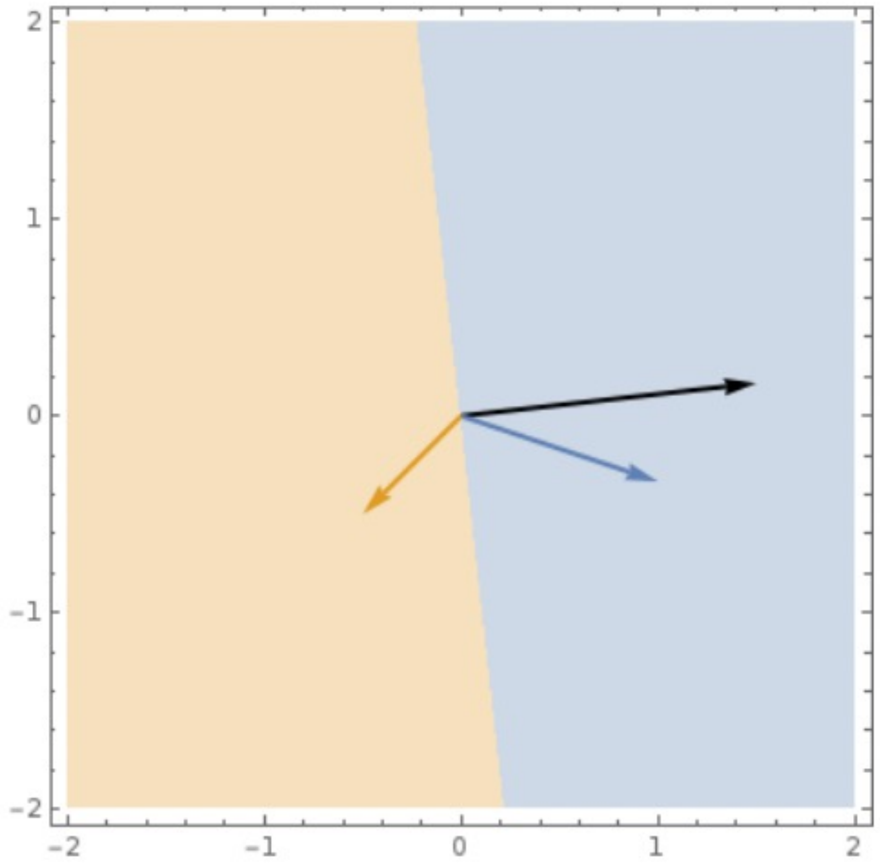
## Linear models: Binary to multiclass



$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6})$$

- Blue class: $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0\}$
- Orange class: $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} < 0\}$

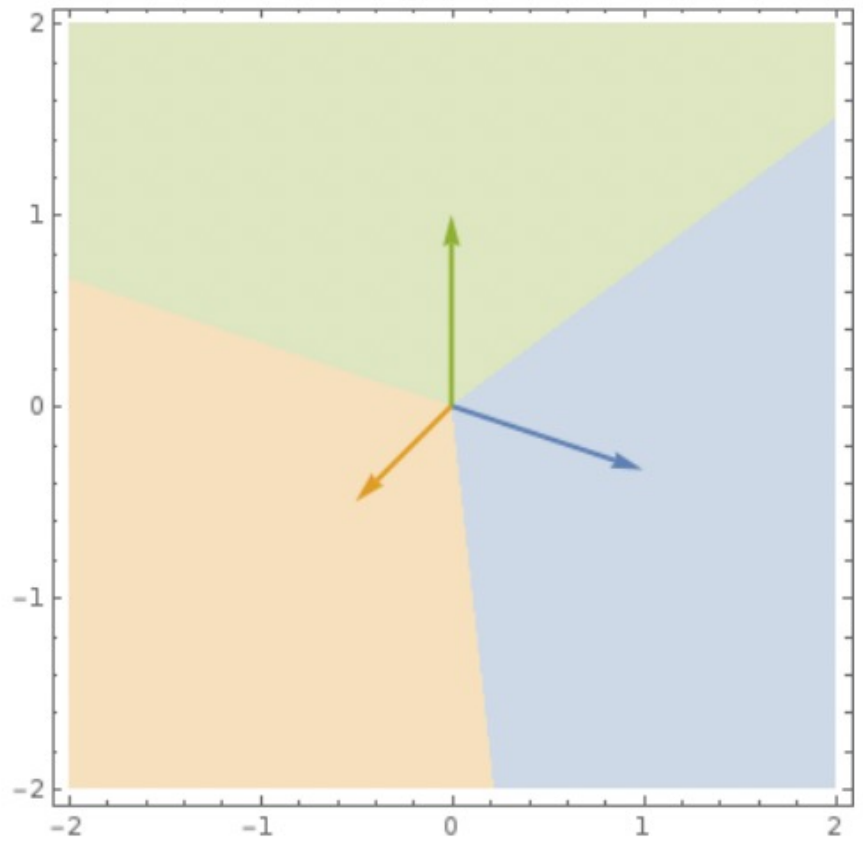
## Linear models: Binary to multiclass



$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6}) = \boldsymbol{w}_1 - \boldsymbol{w}_2$$
$$\boldsymbol{w}_1 = (1, -\tfrac{1}{3})$$
$$\boldsymbol{w}_2 = (-\tfrac{1}{2}, -\tfrac{1}{2})$$

- Blue class:
  $\{\boldsymbol{x} : 1 = \operatorname{argmax}_k \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \operatorname{argmax}_k \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

## Linear models: Binary to multiclass



$$\boldsymbol{w}_1 = (1, -\tfrac{1}{3})$$
$$\boldsymbol{w}_2 = (-\tfrac{1}{2}, -\tfrac{1}{2})$$
$$\boldsymbol{w}_3 = (0, 1)$$

- Blue class:
  $\{\boldsymbol{x} : 1 = \text{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \text{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Green class:
  $\{\boldsymbol{x} : 3 = \text{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

## Linear models: Function class

$$\mathcal{F} = \left\{ f(\boldsymbol{x}) = \underset{k \in [\mathsf{C}]}{\operatorname{argmax}} \; \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\mathsf{C}} \in \mathbb{R}^d \right\}$$

$$= \left\{ f(\boldsymbol{x}) = \underset{k \in [\mathsf{C}]}{\operatorname{argmax}} \; (\boldsymbol{W}\boldsymbol{x})_k \mid \boldsymbol{W} \in \mathbb{R}^{\mathsf{C} \times d} \right\}$$

Next, let’s try to generalize the loss functions. Focus on the logistic loss today.

## Multinomial logistic regression: a probabilistic view

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

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

Naturally, for multiclass:

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

This is called the *softmax function*.

## Let’s find the **MLE**

Maximize probability of seeing labels $y_1, \dots, y_n$ given $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$

$$P(\boldsymbol{W}) = \prod_{i=1}^{n} \mathbb{P}(y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) = \prod_{i=1}^{n} \frac{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i}}{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}}$$

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

$$F(\boldsymbol{W}) = \sum_{i=1}^{n} \ln\left(\frac{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}}{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i}}\right) = \sum_{i=1}^{n} \ln\left(1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}\right)$$

This is the *multiclass logistic loss*.

When $\mathsf{C} = 2$, this is the same as binary logistic loss.

## Let’s find the MLE

# Next, optimization

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

$$F(\boldsymbol{W}) = \ln\left(1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}\right)?$$

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

If $k \neq y_i$:

$$\nabla_{\boldsymbol{w}_k^{\mathrm{T}}} F(\boldsymbol{W}) = \frac{e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}}{1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = \mathbb{P}(k \mid \boldsymbol{x}_i; \boldsymbol{W}) \boldsymbol{x}_i^{\mathrm{T}}$$

else:

$$\nabla_{\boldsymbol{w}_k^{\mathrm{T}}} F(\boldsymbol{W}) = \frac{-\left(\sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}\right)}{1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = \left(\mathbb{P}(y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) - 1\right) \boldsymbol{x}_i^{\mathrm{T}}$$

## SGD for multinomial logistic regression

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

1. pick $i \in [n]$ uniformly at random
2. update the parameters

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \begin{pmatrix} \mathbb{P}(y = 1 \mid \boldsymbol{x}_i; \boldsymbol{W}) \\ \vdots \\ \mathbb{P}(y = y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) - 1 \\ \vdots \\ \mathbb{P}(y = \mathsf{C} \mid \boldsymbol{x}_i; \boldsymbol{W}) \end{pmatrix} \boldsymbol{x}_i^{\mathrm{T}}$$

Think about why the algorithm makes sense intuitively.

## Probabilities -> Prediction

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

- make a *deterministic* prediction $\operatorname{argmax}_{k\in[\mathsf{C}]}\ \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$
- make a *randomized* prediction according to $\mathbb{P}(k \mid \boldsymbol{x}; \boldsymbol{W}) \propto e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}$

In either case, **(expected) mistake is bounded by logistic loss**

- deterministic

$$\mathbb{I}[f(\boldsymbol{x}) \neq y] \leq \log_2\left(1 + \sum_{k\neq y} e^{(\boldsymbol{w}_k-\boldsymbol{w}_y)^{\mathrm{T}}\boldsymbol{x}}\right)$$

$\left(\log_2(1+e^x) \geq 1 \text{ for } x \geq 0\right)$

- randomized

$$\mathbb{E}\left[\mathbb{I}[f(\boldsymbol{x}) \neq y]\right] = 1 - \mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{W}) \leq -\ln \mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{W})$$