Logistics:

- HW0 graded, for regrade request submit it through GradeScope within 7 days from release of grade.
- HW1 due Tuesday Jan 25th midnight

Lecture 9:
Simple variable selection: LASSO for sparse regression

- Yet another hyper-parameter/family of model classes, but with a special property
  - # of features in polynomial regression
  - Regularization coefficient λ for ridge regression
  - Regularization coefficient λ for LASSO

Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

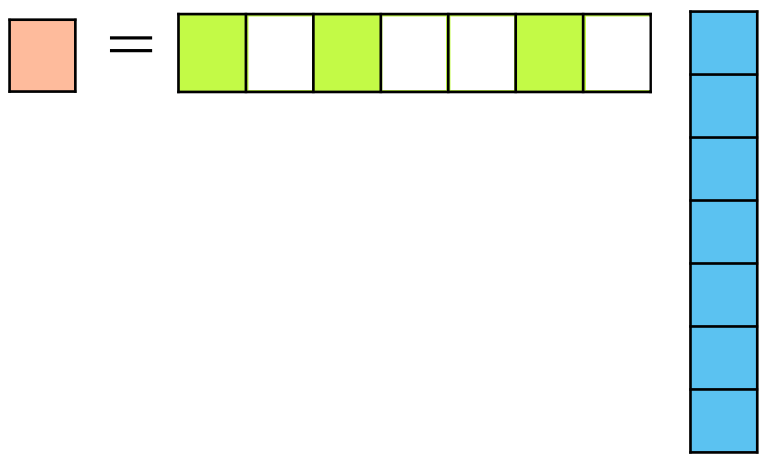
- Vector w is **sparse**, if many entries are zero
  - A vector w is said to be k-sparse if at most k entries are non-zero
  - We are interested in k-sparse w with $k \ll d$
  - Why do we prefer sparse vector w in practice?

Sparsity

$$\widehat{w}_{LS} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- Vector w is **sparse**, if many entries are zero
  - **Efficiency**: If size(w) = 100 Billion, each prediction $w^T x$ is expensive:
    - If w is sparse, prediction computation only depends on number of non-zeros in w

$$\widehat{y}_i = \widehat{w}_{\mathrm{LS}}^T x_i$$



$$= \sum_{j=1}^{d} \widehat{w}_{\mathrm{LS}}[j] \times x_i[j] = \sum_{j: w_{\mathrm{LS}}[j] \neq 0} \widehat{w}_{\mathrm{LS}}[j] \times x_i[j]$$

Computational complexity decreases from $2d$ to $2k$ for k-sparse $\widehat{w}_{\mathrm{LS}}$

Sparsity

$$\widehat{w}_{LS} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2$$

- Vector w is **sparse**, if many entries are zero
  - **Interpretability**: What are the relevant features to make a prediction?

Lot size
Single Family
Year built
Last sold price
Last sale price/sqft
Finished sqft
Unfinished sqft
Finished basement sqft
floors
Flooring types
Parking type
Parking amount
Cooling
Heating
Exterior materials
Roof type
Structure style

Dishwasher
Garbage disposal
Microwave
Range / Oven
Refrigerator
Washer
Dryer
Laundry location
Heating type
Jetted Tub
Deck
Fenced Yard
Lawn
Garden
Sprinkler System

- How do we find "best" subset of features useful in predicting the price among all possible combinations?

Finding best subset of features that explain the outcome/label: Exhaustive

- Try all subsets of size 1, 2, 3, … and one that minimizes validation error
  - Problem?
  - Any Ideas?

Finding best subset: Greedy

Forward stepwise:
Starting from simple model and iteratively add features most useful to fit

Forward Greedy

1: $T \leftarrow \varnothing$

2: **For** $j = 1, \ldots, k$ **do**

3: $\quad j^* \leftarrow \arg\min_{\ell} \min_{w} \sum_{i=1}^{n} \left(y_i - \sum_{j \in T \cup \{\ell\}} w[j] \times x_i[j] \right)^2$

4: $\quad T \leftarrow T \cup \{j^*\}$

Backward stepwise:
Start with full model and iteratively remove features least useful to fit

Combining forward and backward steps:
In forward algorithm, insert steps to remove features no longer as important

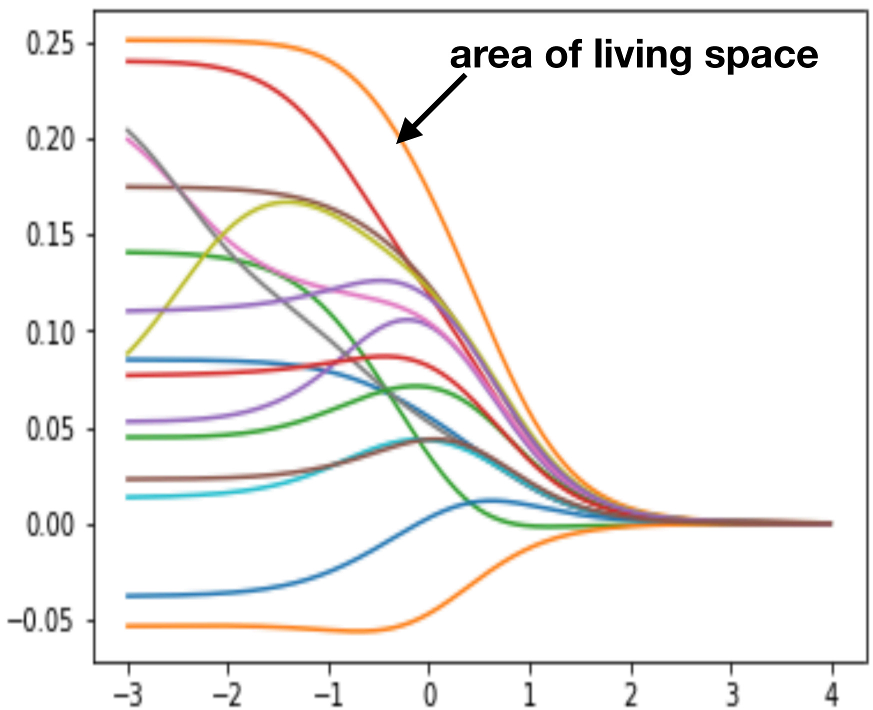
Lots of other variants, too.

Finding best subset: Regularize

Recall that Ridge regression makes coefficients small

$$\widehat{w}_{ridge} = \arg\min_w \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

w_i's

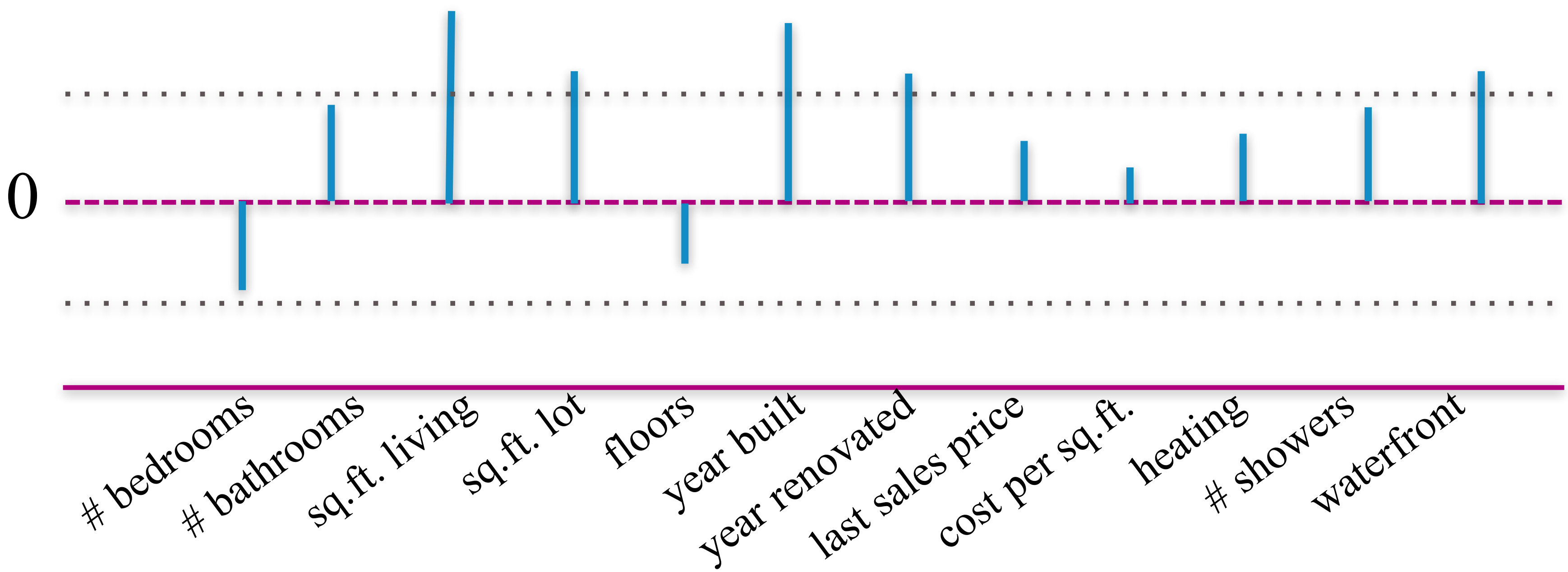


$\log_{10}(\lambda)$

Thresholded Ridge Regression

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

- Why don't we just set **small** ridge coefficients to 0?
    - Any issues?



Thresholded Ridge Regression

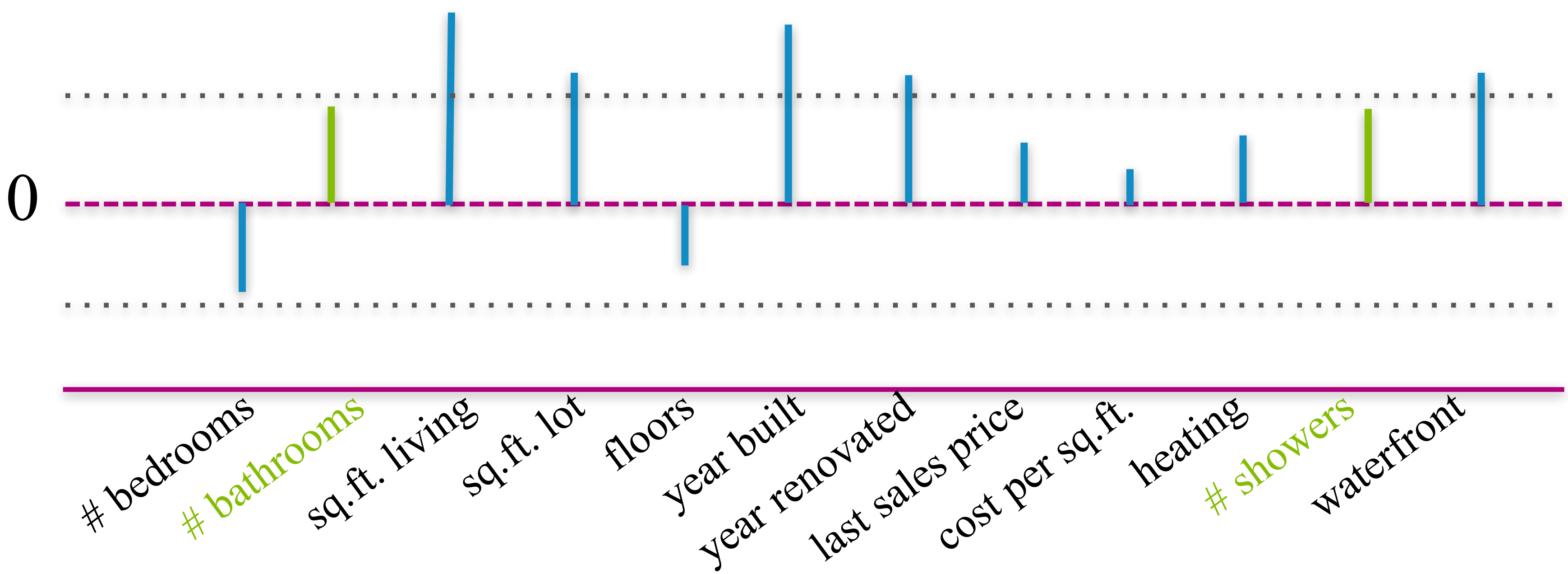
$$\widehat{w}_{ridge} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda||w||_2^2$$

- Consider two related features (bathrooms, showers)
- Consider $w[\text{bath}] = 1$ and $w[\text{shower}] = 1$, and

  $w[\text{bath}] = 2$ and $w[\text{shower}] = 0$,

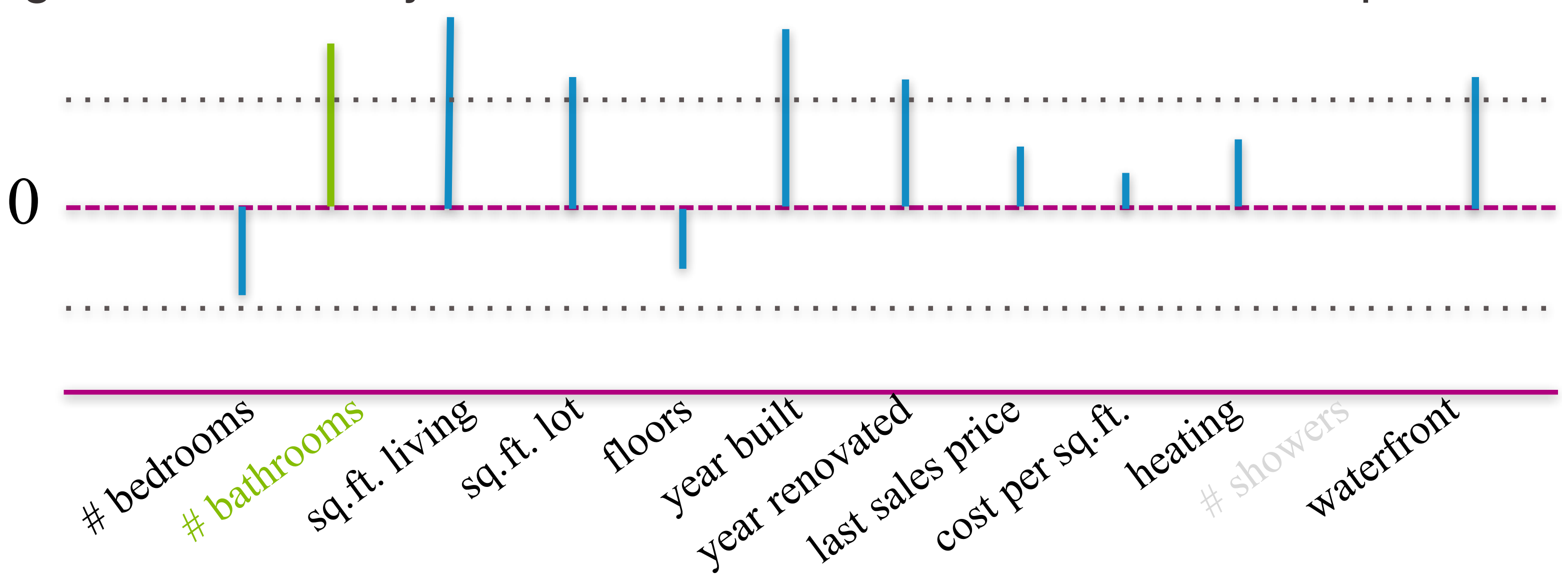
  which one does ridge regression choose?

  (assuming #bathroom=#showers in every house)



Thresholded Ridge Regression

- Consider two related features (bathrooms, showers)
- Issue with thresholded ridge regression is that
  ridge regression prefers balanced weights between similar features
- What if we **didn't** include showers? Weight on bathrooms increases, and it should have been selected.
- We want a feature selection scheme that selects one of (#bathroom) or (#showers) automatically,
  using the fact that if you delete #showers #bathroom is an important feature



- There is a better regularizer for sparse regression,
  that can perform the feature selection automatically.

Ridge vs. Lasso Regression

- Recall Ridge Regression objective:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

- sensitivity of a model w is measured in squared ℓ_2 norm $\|w\|_2^2$
- A principled method to get sparse model is **Lasso** with regularized objective:

$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$

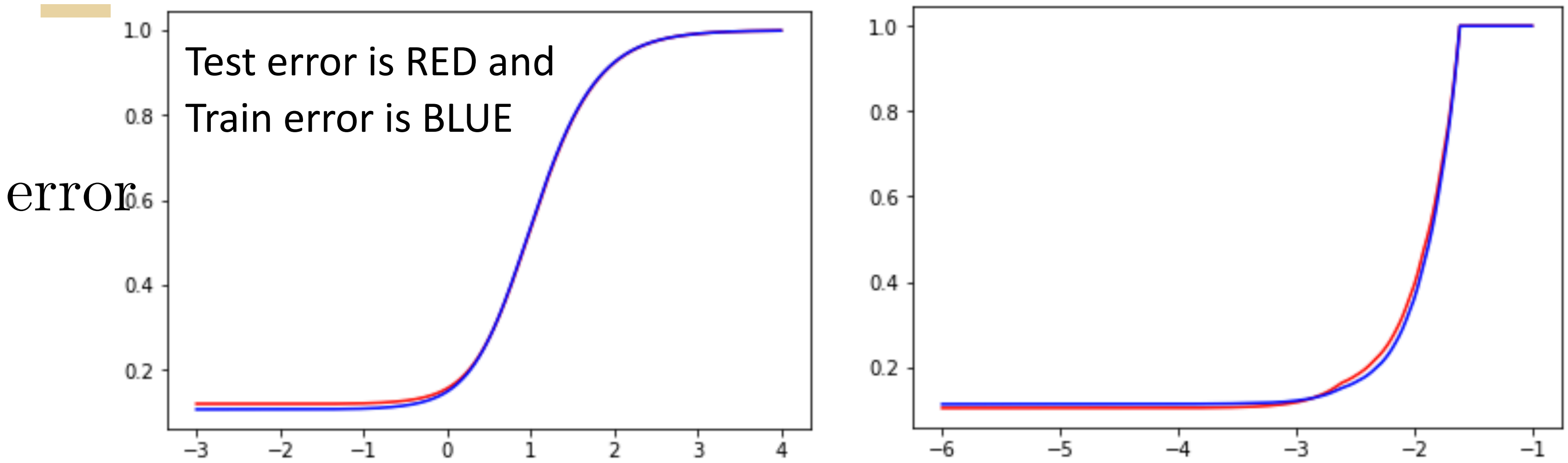
- sensitivity of a model w is measured in ℓ_1 norm:

$$\|w\|_1 = \sum_{j=1}^{d} \left| w[j] \right|$$

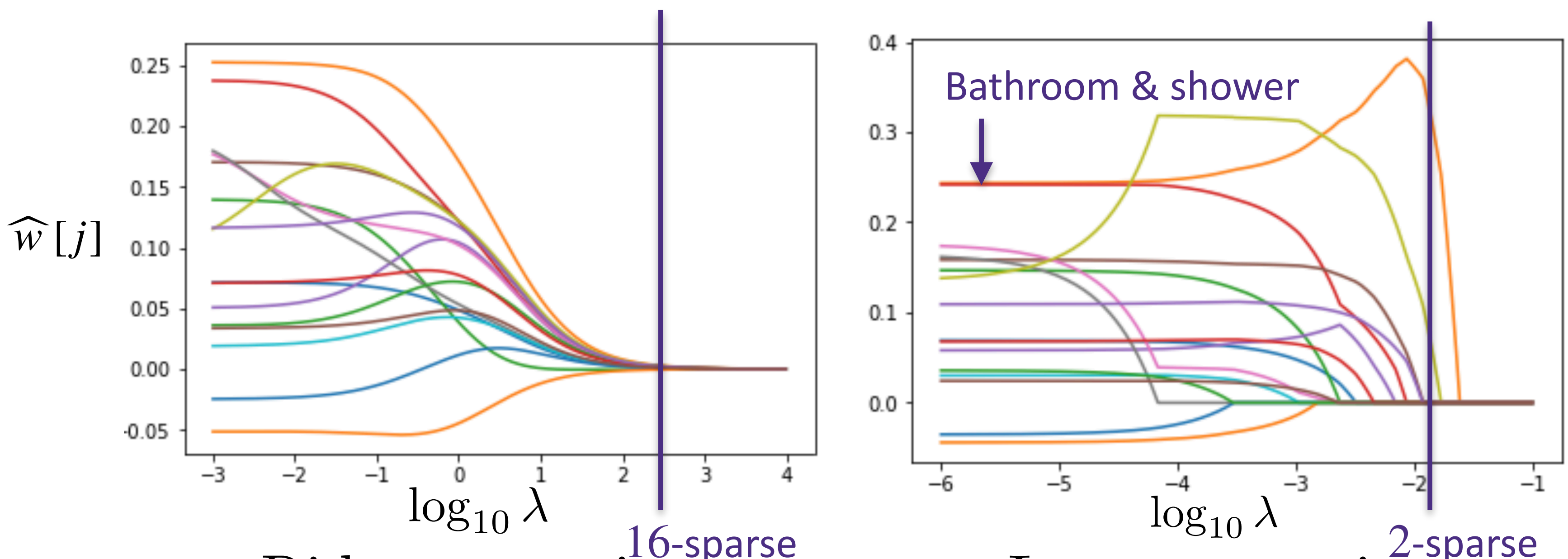
ℓ_p-norm of a vector $w \in \mathbb{R}^d$ is

$$\|w\|_p \triangleq \left(\sum_{j=1}^{d} |w[j]|^p\right)^{1/p}$$

Example: house price with 16 features



- Regularization path for Lasso shows that weights drop to exactly zero as λ increases



Ridge regression Lasso regression

Lasso regression naturally gives sparse features

- **feature selection** with Lasso regression
  1. **Model selection**: choose λ based on cross validation error
  2. **Feature selection**: keep only those features with non-zero (or not-too-small) parameters in w at optimal λ
  3. **retrain** with the sparse model and $\lambda = 0$

  why do we need to retrain?

Example: piecewise-linear fit

- We use Lasso on the piece-wise linear example

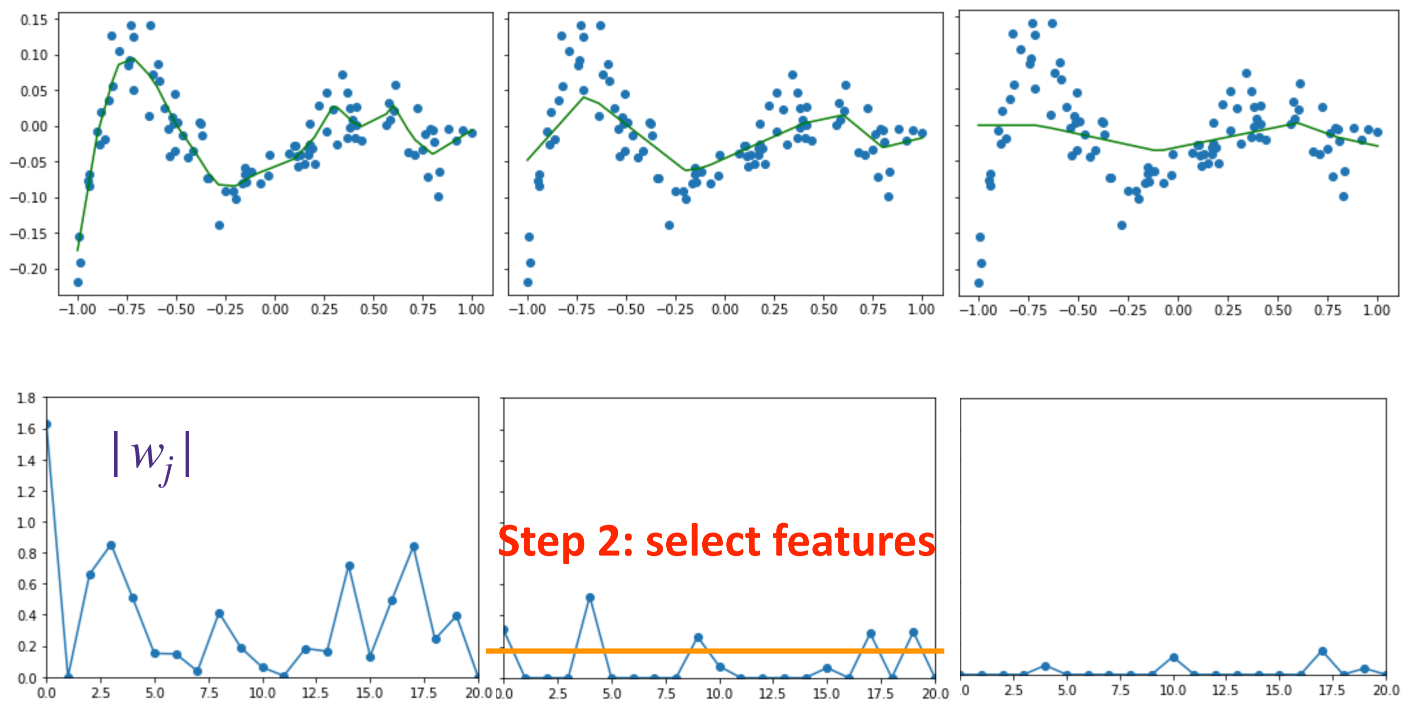
$$h_0(x) = 1$$
$$h_i(x) = [x + 1.1 - 0.1i]^+$$

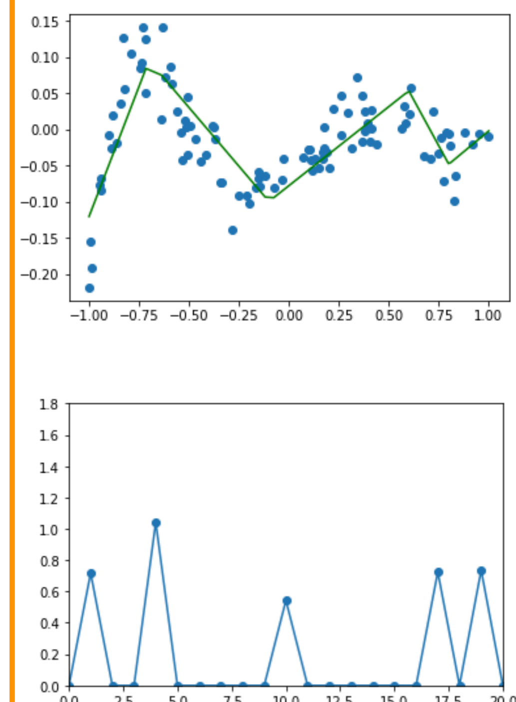
Step 1: find optimal $\lambda*$

$$\text{minimize}_w \;\; \mathscr{L}(w) + \lambda \|w\|_1$$

Step 3: retrain

$$\text{minimize}_w \;\; \mathscr{L}(w)$$





$\lambda = 10^{-8}$ $\quad$ $\lambda = 10^{-4}$ $\quad$ $\lambda = 2 \times 10^{-4}$ $\quad$ $\lambda = 0$

but only use selected features

- de-biasing (via re-training) is critical!

Penalized Least Squares

$$\text{Ridge} : r(w) = ||w||_2^2 \qquad \text{Lasso} : r(w) = ||w||_1$$

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

Penalized Least Squares

- Regularized optimization:

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda r(w)$$

$$\text{Ridge} : r(w) = ||w||_2^2$$

$$\text{Lasso} : r(w) = ||w||_1$$

- For any $\lambda^* \geq 0$ for which $\hat{w}_r$ achieves the minimum, there exists a $\mu^* \geq 0$ such that the solution of the constrained optimization, $\widehat{w}_c$, is the same as the solution of the regularized optimization , $\widehat{w}_r$, where

$$\widehat{w}_C = \arg\min_w \sum_{i=1}^n (y_i - x_i^T w)^2 \qquad \text{subject to } \; r(w) \leq \mu^*$$

- so there are pairs of (λ, μ) whose optimal solution $\widehat{w}_r$ are the same for the regularizes optimization and constrained optimization

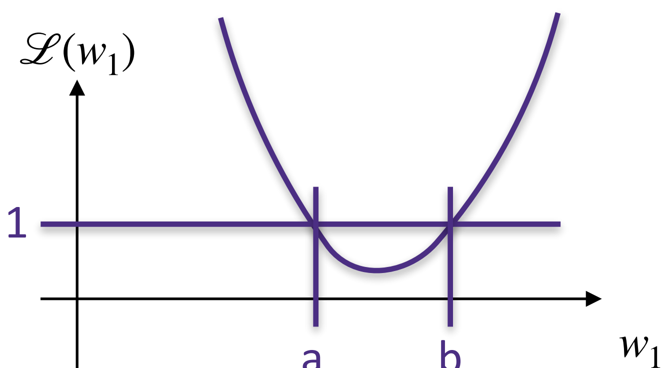
Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

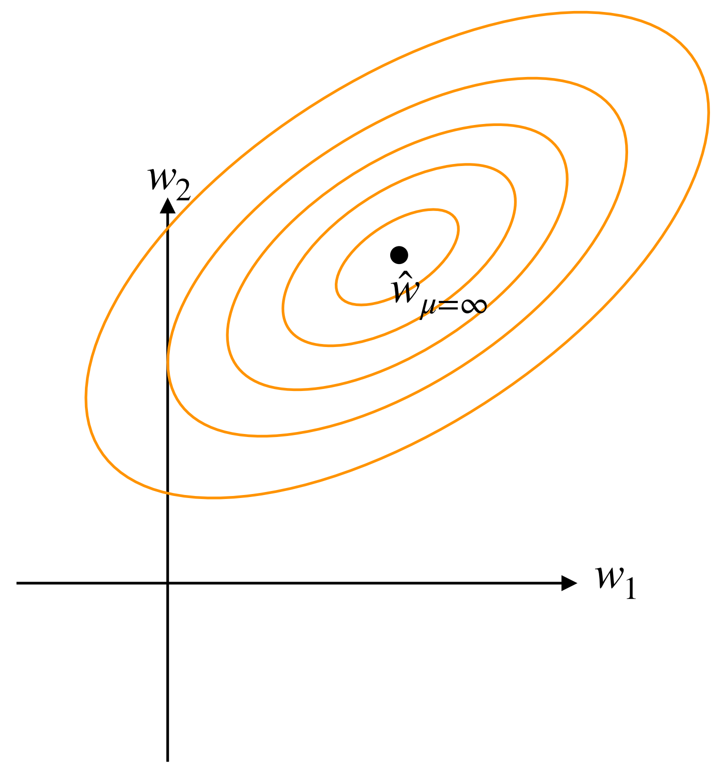
$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- the **level set** of a function $\mathscr{L}(w_1, w_2)$ is defined as the set of points (w_1, w_2) that have the same function value
- the level set of a quadratic function is an oval
- the center of the oval is the least squares solution $\hat{w}_{\mu=\infty} = \hat{w}_{\text{LS}}$

1-D example with quadratic loss



Level set of $\mathscr{L}(w_1)$ at value 1 is [a,b]

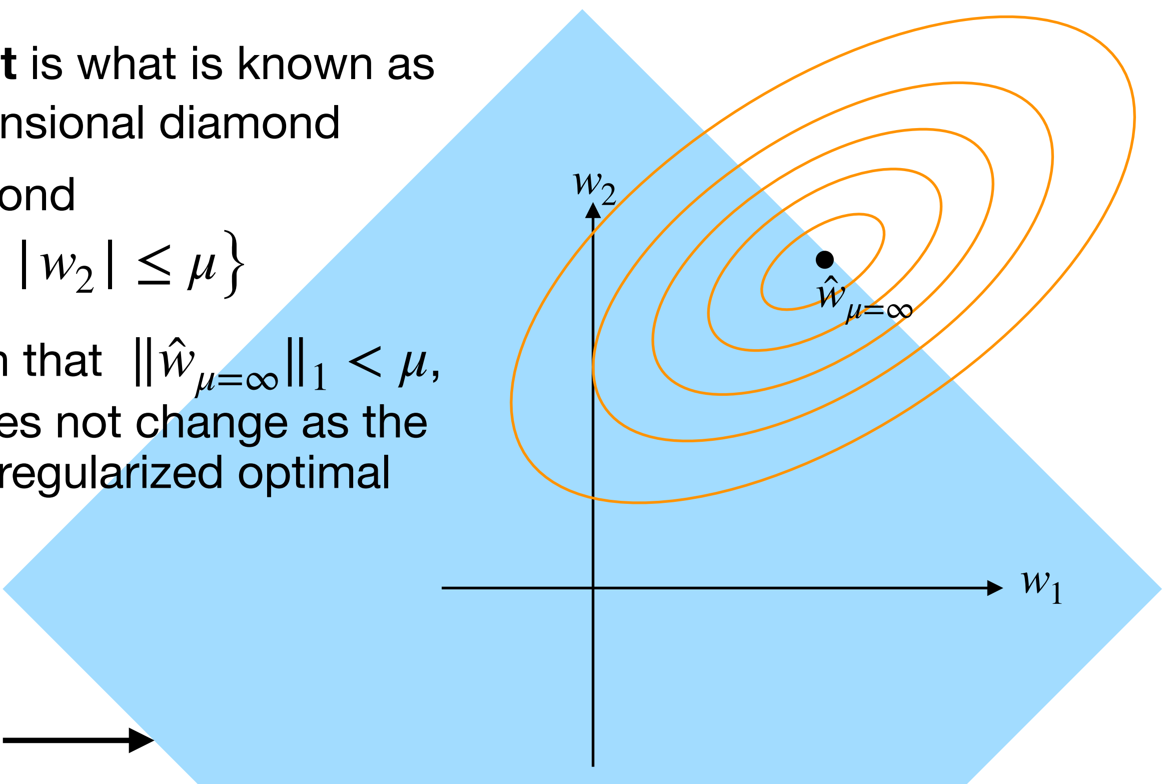


Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- as we decrease μ from infinity, the feasible set becomes smaller
- the shape of the **feasible set** is what is known as L_1 ball, which is a high dimensional diamond
- In 2-dimensions, it is a diamond
$$\left\{(w_1, w_2) \,\middle|\, |w_1| + |w_2| \leq \mu\right\}$$
- when μ is large enough such that $\|\hat{w}_{\mu=\infty}\|_1 < \mu$, then the optimal solution does not change as the feasible set includes the un-regularized optimal solution



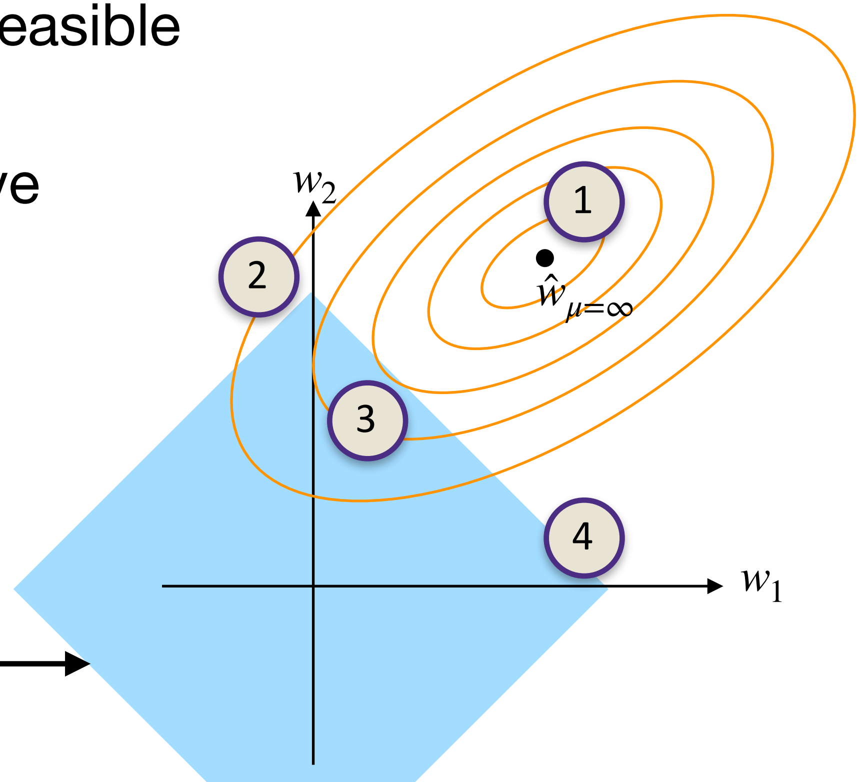
feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$

Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

- As μ decreases (which is equivalent to increasing regularization λ) the feasible set (blue diamond) shrinks
- The optimal solution of the above optimization is ?



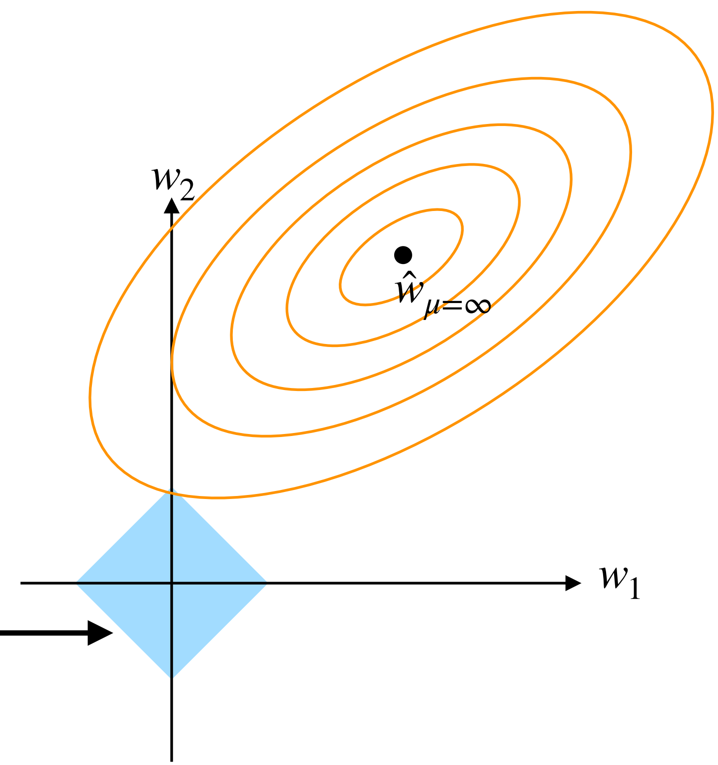
feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$

Why does Lasso give sparse solutions?

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \;\leq\; \mu$$

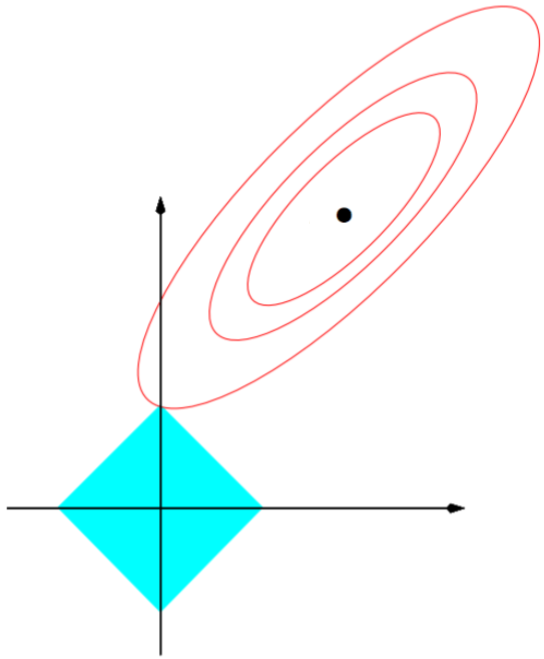
- For small enough μ, the optimal solution becomes **sparse**
- This is because the L_1-ball is "pointy",i.e., has sharp edges aligned with the axes

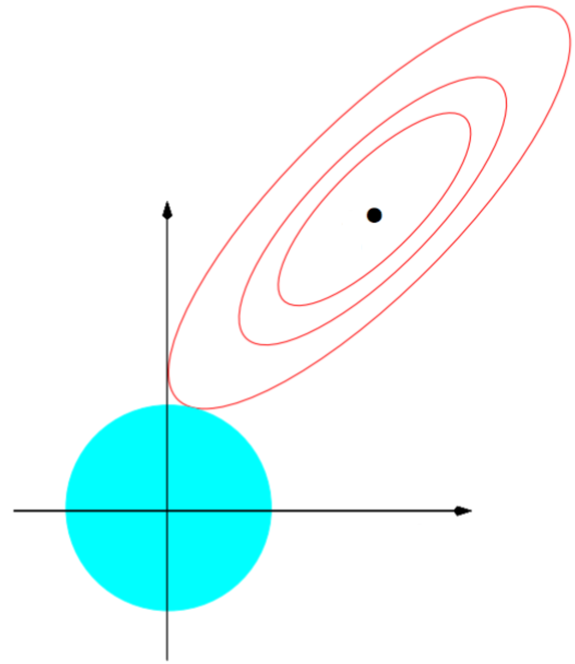


feasible set: $\{w \in \mathbb{R}^2 \mid \|w\|_1 \leq \mu\}$

Penalized Least Squares

- Lasso regression finds sparse solutions, as L_1-ball is "pointy"
- Ridge regression finds dense solutions, as L_2-ball is "smooth"





$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_1 \leq \mu$$

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$\text{subject to} \quad \|w\|_2^2 \leq \mu$$

Ridge vs. Lasso

- **Ridge**
  - Very fast:
    - Closed form solution if used with linear models
    - Even with non-linear and complex loss, optimization is fast for squared ℓ_2 regularization (to be taught later)
  - Gives regularized parameters that avoid overfitting

- **Lasso**
  - Slower than Ridge:
    - It is a non-smooth optimization which is slower (to be taught later)
  - Gives sparse parameters

Questions?

Logistics:
- HW2 is our and due Tuesday Feb 11th Friday,
  - it covers up to stochastic gradient descent
  - It is quite involved, so we are giving you more time, but start early!
- Return to in-person on Monday 1/31/2022
  - Sections will be in person starting next week and OHs will be hybrid

Lecture 10: Convexity

- When is an optimization (or learning) easy/fast to solve?

Recap: Ridge vs. Lasso

- **Ridge**

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2$$

  - Very fast:
    - Closed form solution if used with linear models
    - Even with other loss functions, optimization is fast for squared ℓ_2 regularization, because $\|w\|_2^2$ is **convex and smooth**

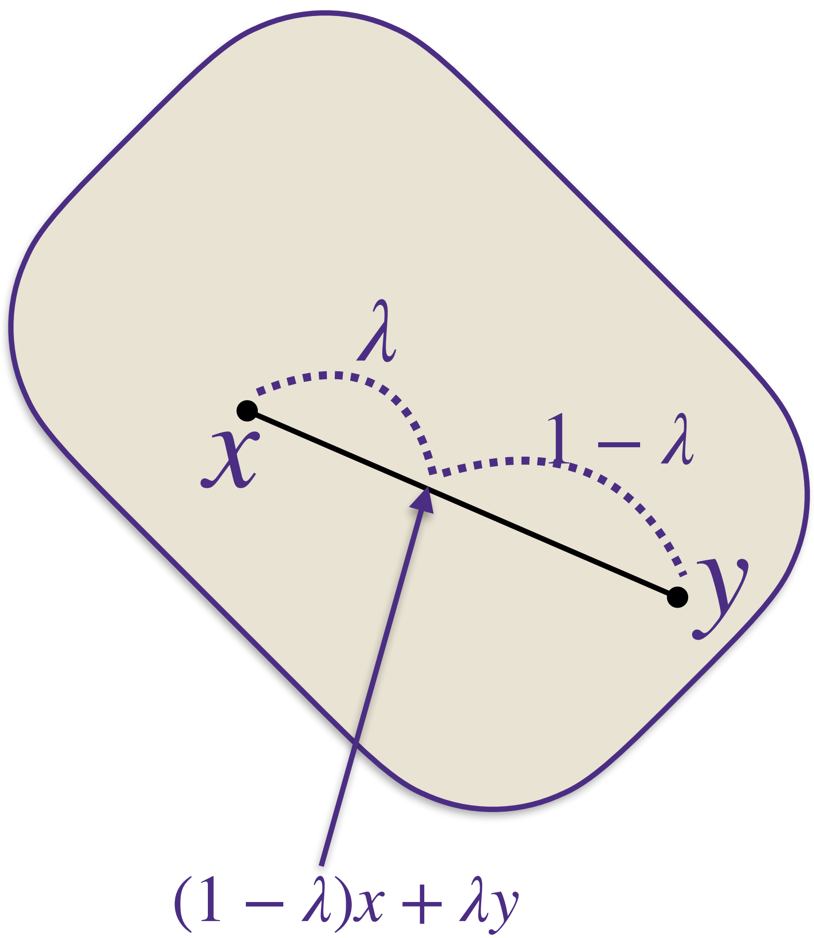
- **Lasso**

$$\text{minimize}_w \quad \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_1$$

  - Slower than Ridge:
    - Requires iterative optimization algorithm like sub-gradient descent
    - In particular, it is slower because $\|w\|_1$ is **convex but non-smooth**

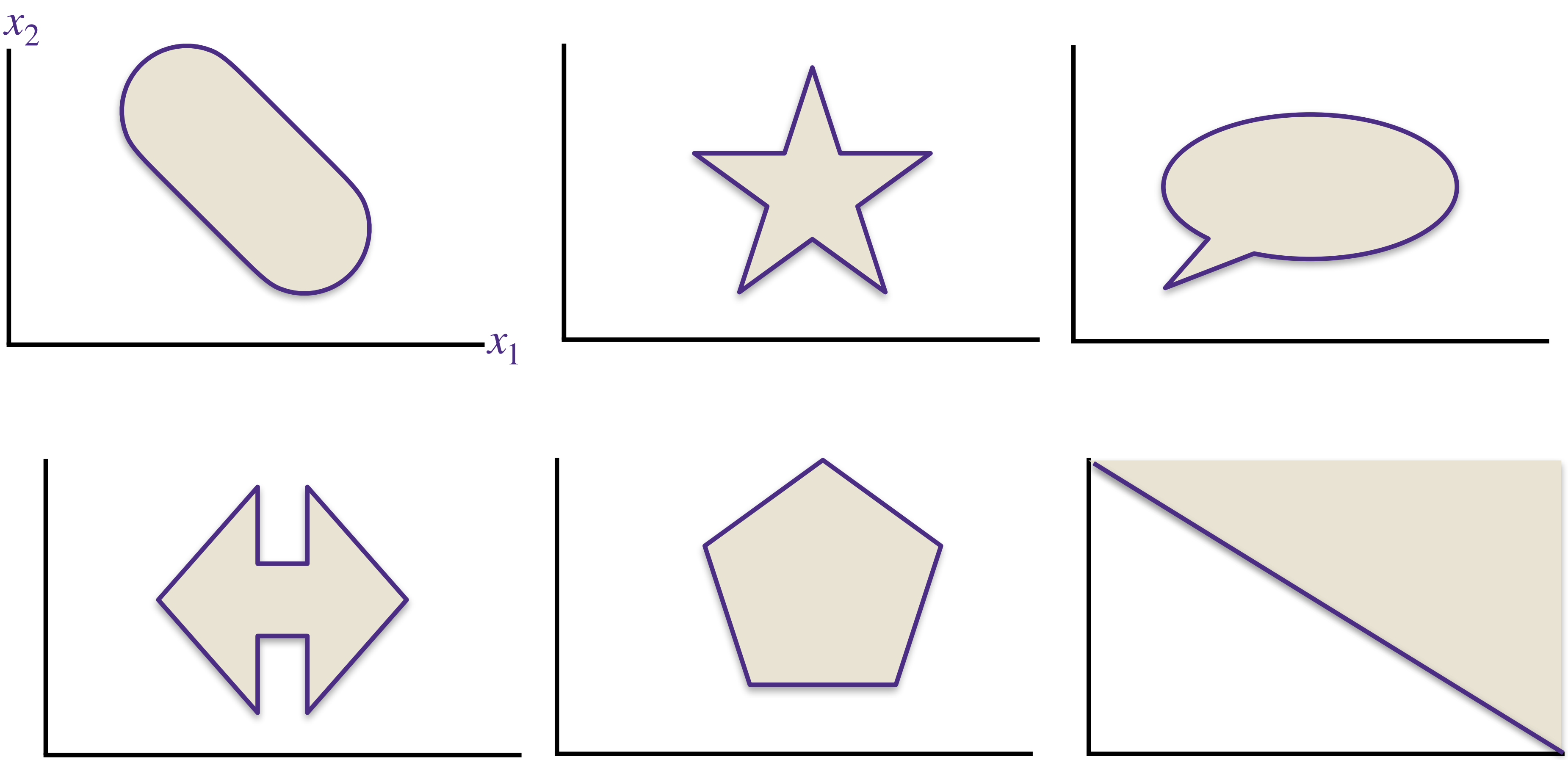
What is a convex set?

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$



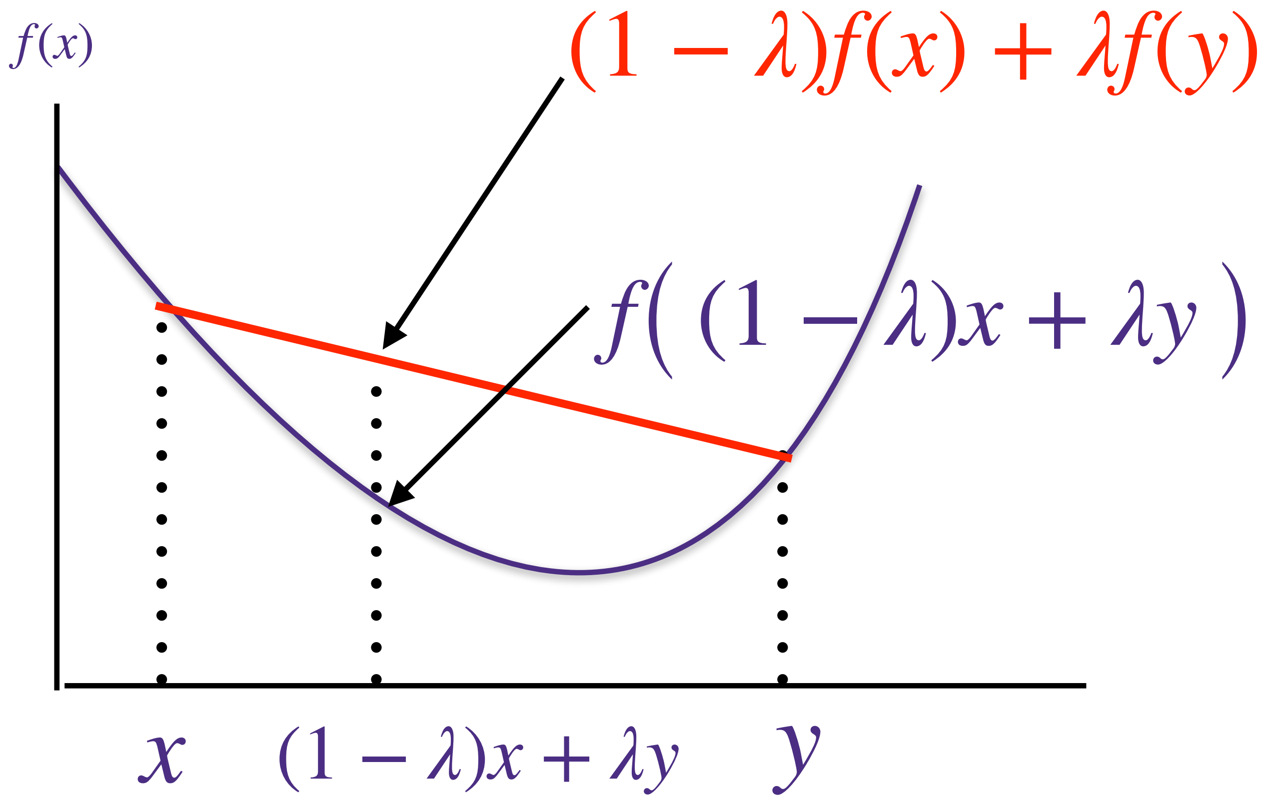
What is a convex set?

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$



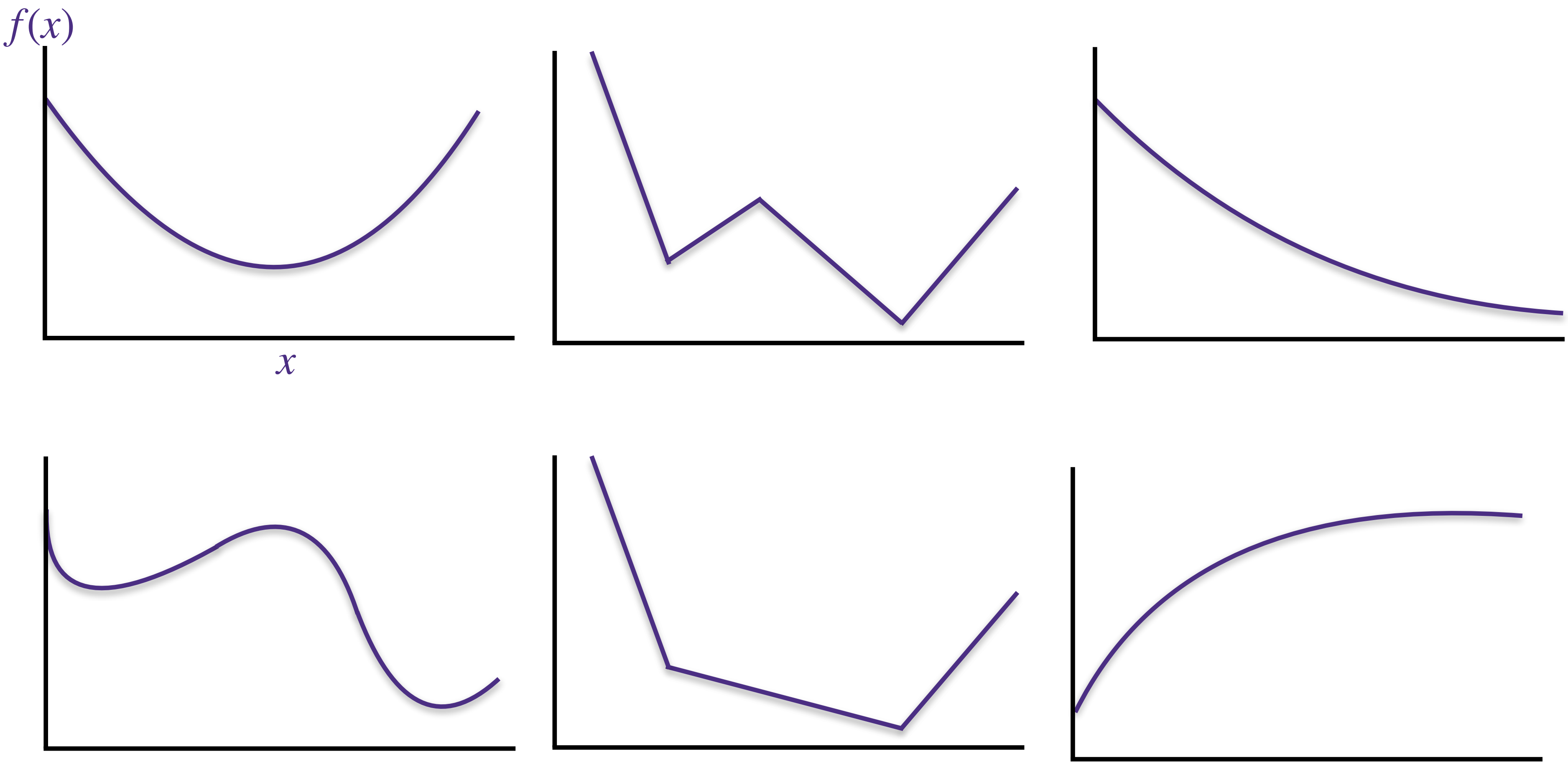
What is a convex function?

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$ for all $x, y \in \mathbb{R}^d$ and $\lambda \in [0, 1]$



What is a convex function?

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$ for all $x, y \in \mathbb{R}^d$ and $\lambda \in [0, 1]$



Convex functions and convex sets?

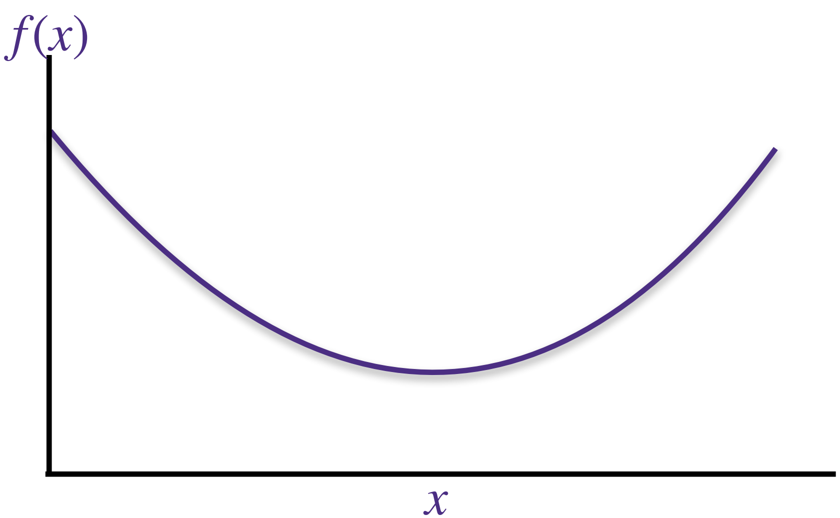
A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

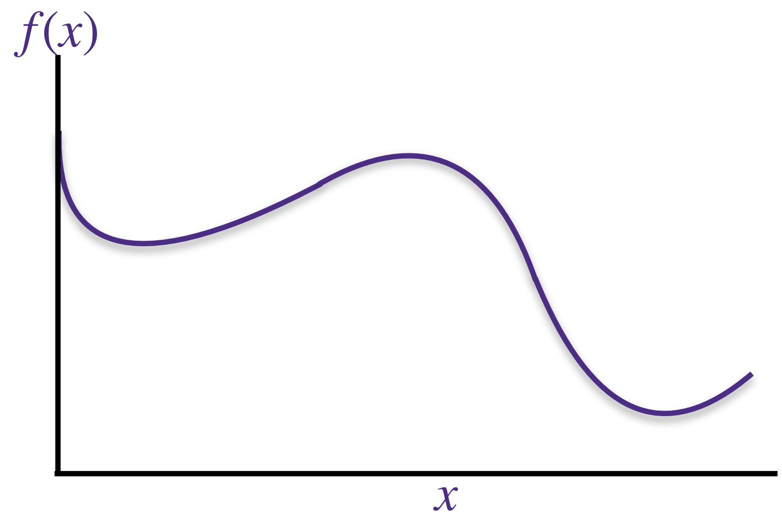
A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y)$ for all $x, y \in \mathbb{R}^d$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}$ is convex

Graph of f id defined as $\{(x,t) \;:\; f(x) = t\}$

Epigraph of f is defined as $\{(x,t) \;:\; f(x) \leq t\}$



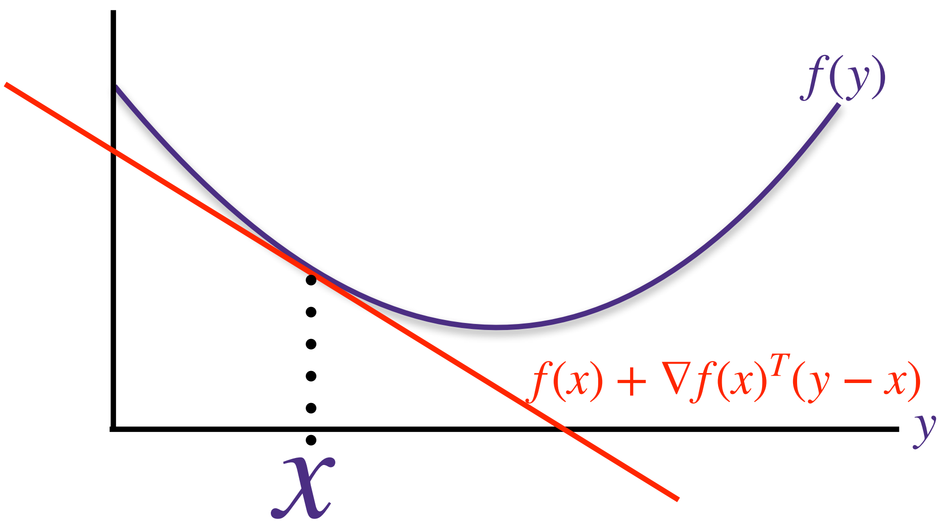


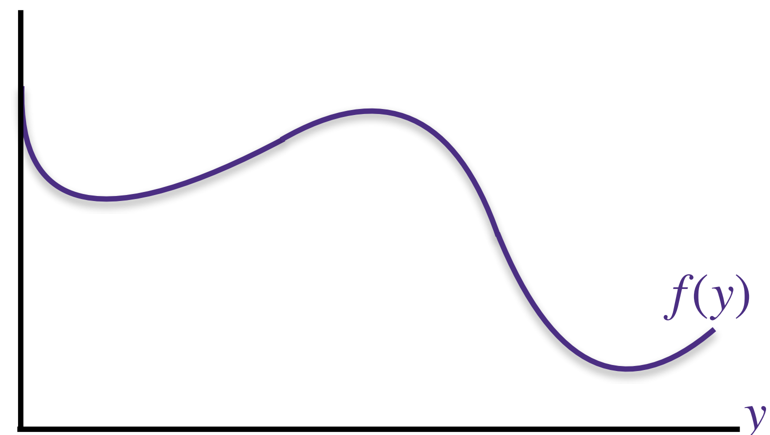
More definitions of convexity

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \le t\}$ is convex

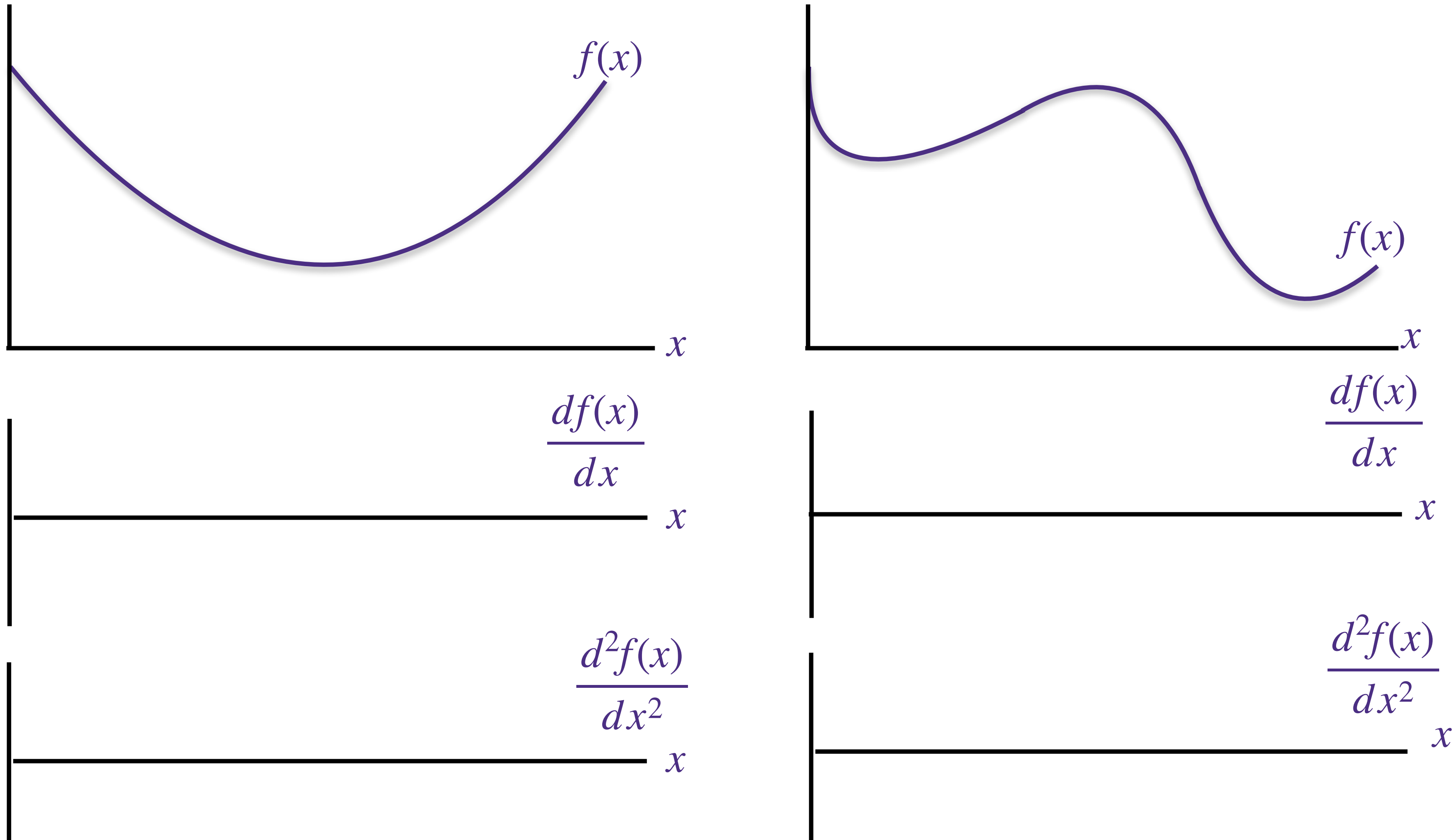
A function $f : \mathbb{R}^d \to \mathbb{R}$ that is differentiable everywhere is convex if $f(y) \ge f(x) + \nabla f(x)^\top (y-x)$ for all $x, y \in dom(f)$





More definitions of convexity

A function $f : \mathbb{R}^d \to \mathbb{R}$ that is twice-differentiable everywhere is convex if $\nabla^2 f(x) \succeq 0$ for all $x \in dom(f)$



More definitions of convexity

A set $K \subset \mathbb{R}^d$ is convex if $(1-\lambda)x + \lambda y \in K$ for all $x, y \in K$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if $f((1-\lambda)x + \lambda y) \leq (1-\lambda) f(x) + \lambda f(y)$ for all $x, y \in \mathbb{R}^d$ and $\lambda \in [0,1]$

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if the set $\{(x,t) \in \mathbb{R}^{d+1} : f(x) \leq t\}$ is convex

A function $f : \mathbb{R}^d \to \mathbb{R}$ that is differentiable everywhere is convex if $f(y) \geq f(x) + \nabla f(x)^\top (y-x)$ for all $x, y \in dom(f)$

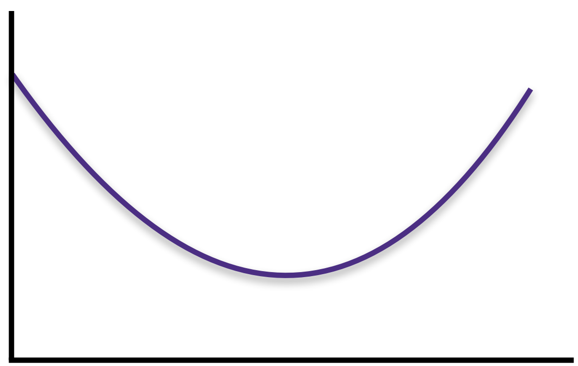
A function $f : \mathbb{R}^d \to \mathbb{R}$ that is twice-differentiable everywhere is convex if $\nabla^2 f(x) \succeq 0$ for all $x \in dom(f)$

Why do we care about convexity?

Convex functions

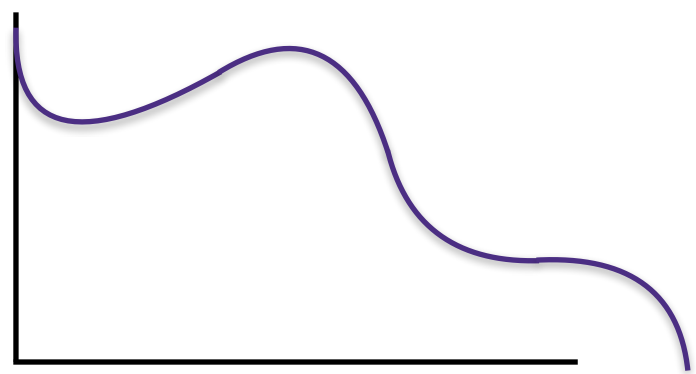
- All local minima are global minima
- Efficient to optimize (e.g., gradient descent)

Convex Function



We only need to find a point with $\nabla f(x) = 0$, which for convex functions implies that it is a local minima and a global minima

Non-convex Function



For non-convex functions, a stationary point with $\nabla f(x) = 0$ could be a local minima, a local maxima, or a saddle point

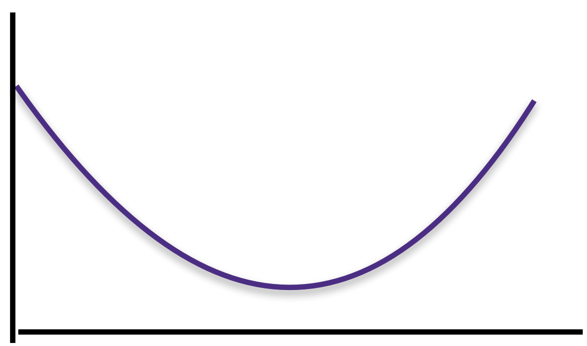
Gradient Descent on $\min_w f(w)$

$$\text{Initialize: } w_0 = 0$$

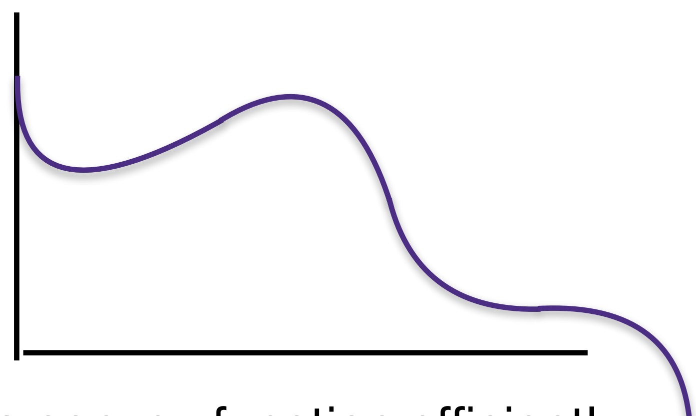
$$\text{for } t = 1, 2, \ldots$$

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

Convex Function



Non-convex Function



- Strength: Can find global minima of a convex function efficiently
- Weakness: Can only be applied to smooth functions
  - i.e., functions that is differentiable everywhere,
  - otherwise $\nabla f(x)$ is not defined and gradient descent cannot be applied

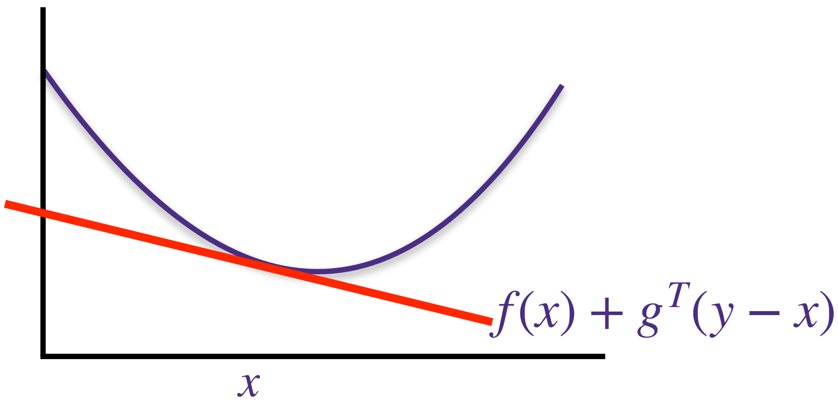
Sub-Gradient

Definition: a function is **non-smooth** if it is not differentiable everywhere

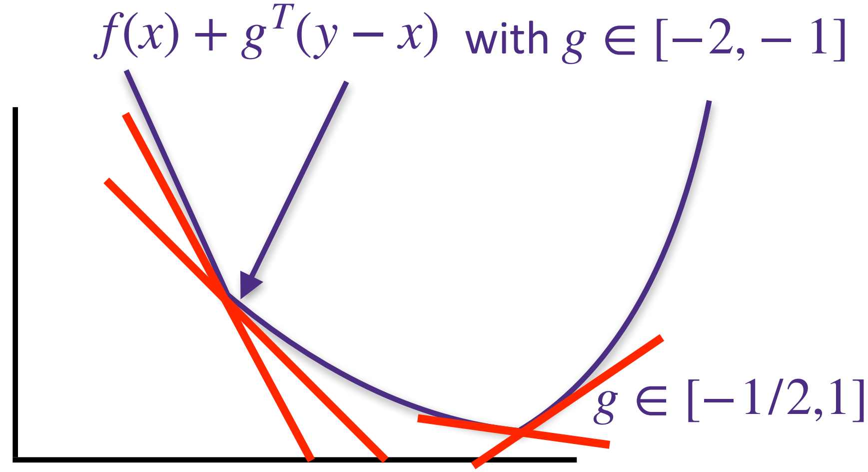
Definition: a vector $g \in \mathbb{R}^d$ is a **sub-gradient** at x if it satisfies

$$f(y) \geq f(x) + g^T(y - x) \text{ for all } y \in \mathbb{R}^d$$

Smooth Convex Function



Non-smooth Convex Function



- for smooth convex functions,
  - gradient is the unique sub-gradient, and
  - the global minimum is achieved at points where gradient is zero

- for non-smooth convex functions,
  - the minimum is achieved at points where sub-gradient set includes the zero vector

Sub-Gradient Descent for non-smooth functions

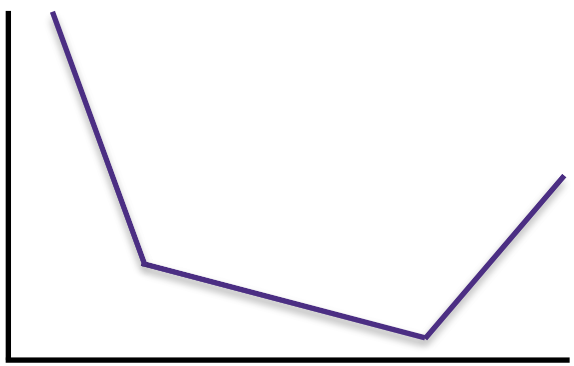
Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

Find any g_t such that $f(y) \geq f(w_t) + g_t^\top (y - w_t)$

$$w_{t+1} \leftarrow w_t - \eta_t g_t$$

Convex Function



Non-convex Function

- Strength: finds global minima for **non-smooth convex functions**
- Weakness: it is slower than gradient descent on convex smooth functions, because the gradient do not get smaller near the global minima
  - Instead of last iterate w_t, we use the best one we saw in all iterates
  - The stepsize needs to decrease with t

Coordinate descent

Initialize: $w_0 = 0$

for $t = 1, 2, \ldots$

Let $i_t = t \,\%\, d$

$$w_{t+1}[i_t] \leftarrow w_t[i_t] - \eta_t \frac{\partial f(w_t)}{\partial w[i_t]}$$

Optimization

- You can always run gradient descent whether f is convex or not. But you only have guarantees if f is convex
- Many bells and whistles can be added onto gradient descent such as momentum and dimension-specific step-sizes (Nesterov, Adagrad, ADAM, etc.)

Questions?

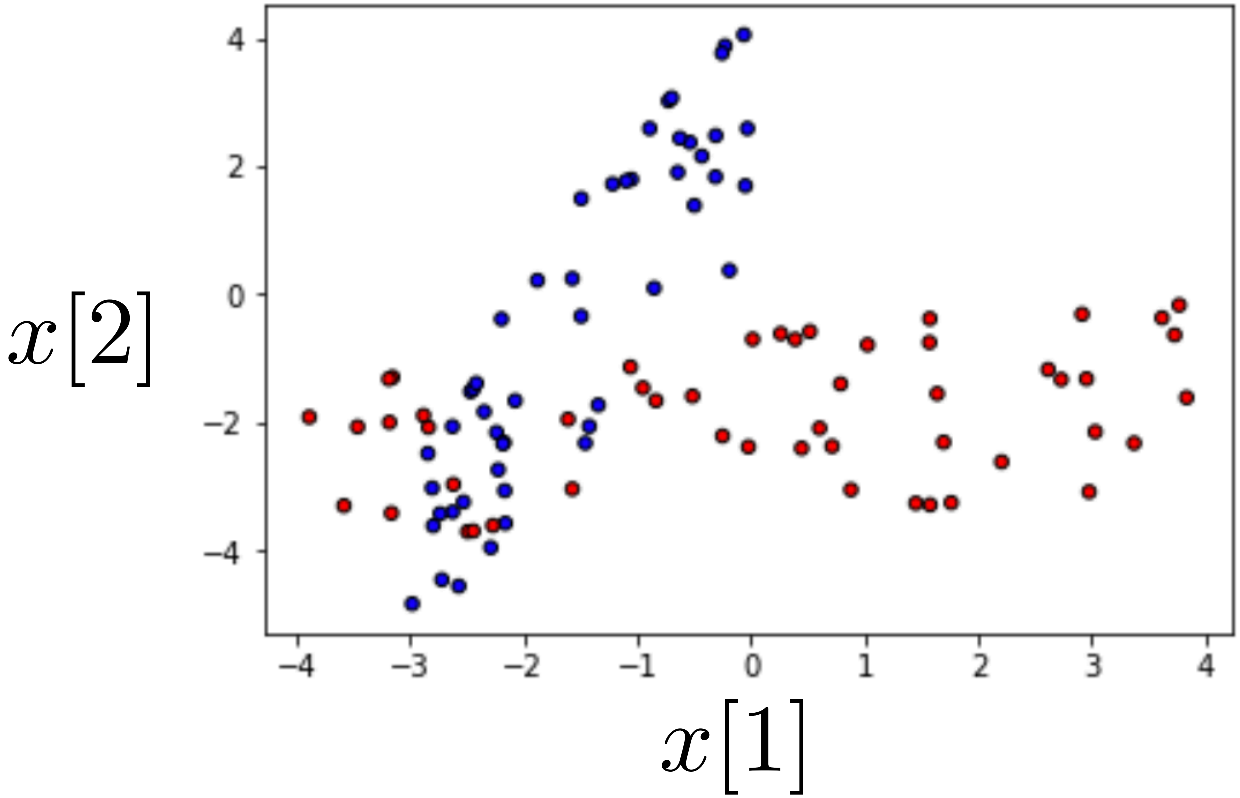
Logistics:

-

Lecture 11: Classification with logistic regression

- Regression: label is continuous valued
- Classification: label is discrete valued, e.g., {0,1}

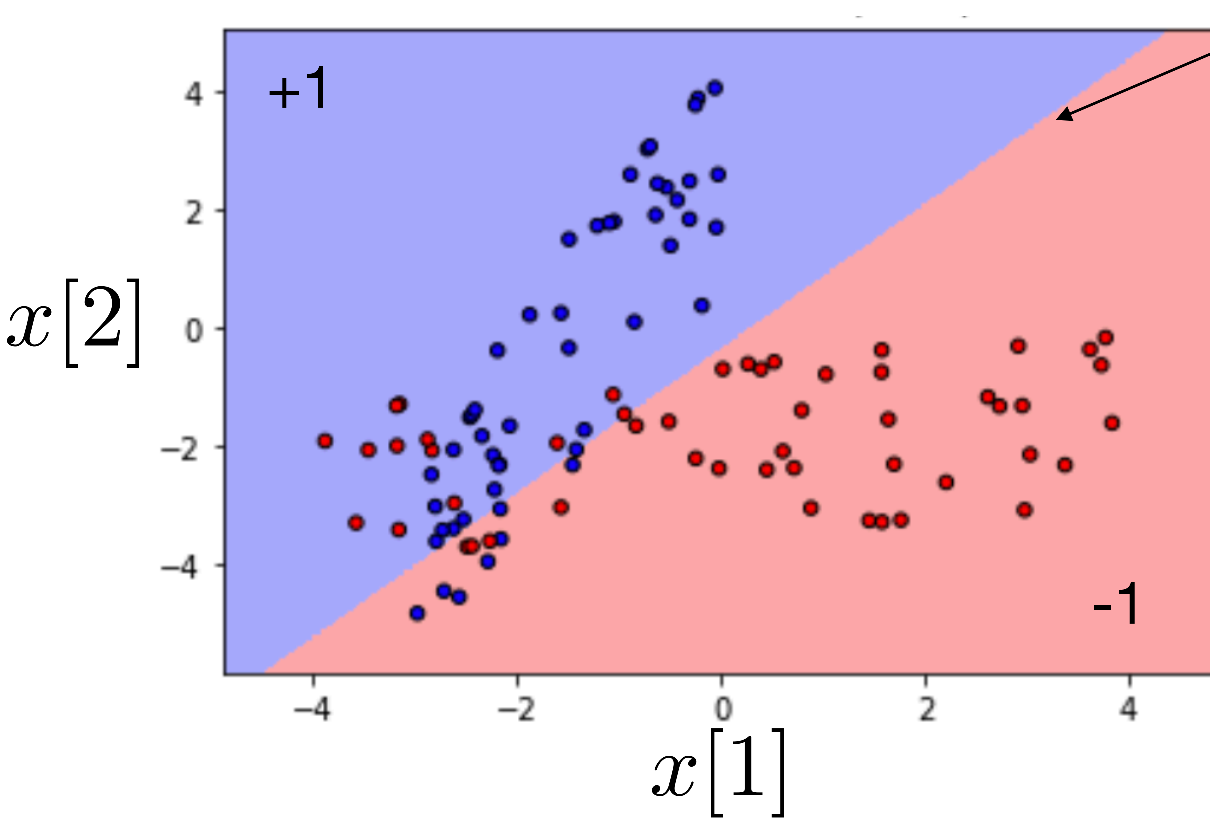
Training data for a binary classification problem



- in this example, each input is $x_i \in \mathbb{R}^2$
- Red points have label y_i=-1, blue points have label y_i=1
- We want a predictor that maps any $x \in \mathbb{R}^2$ to a prediction $\hat{y} \in \{-1, +1\}$

Example: linear classifier trained on 100 samples

simple decision boundary at $w^T x + b = 0$

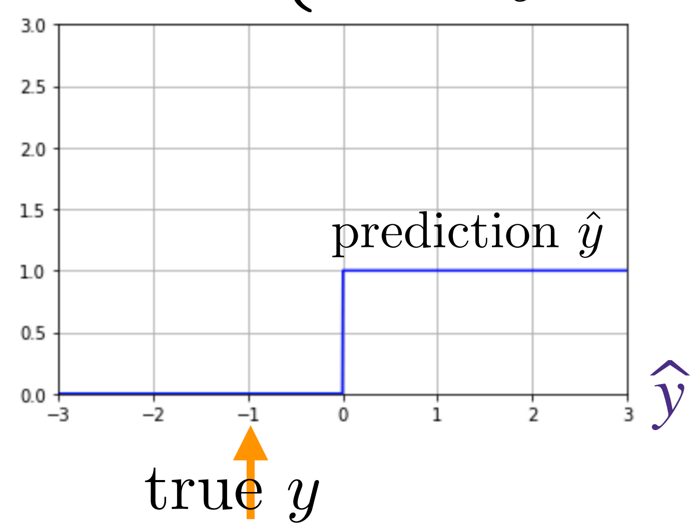


- linear model: $w_0 + w_1 x[1] + w_2 x[2]$
- predict using $\hat{y} = \text{sign}(b + x^T w)$
- How do we find such a linear classifier that fits the data?

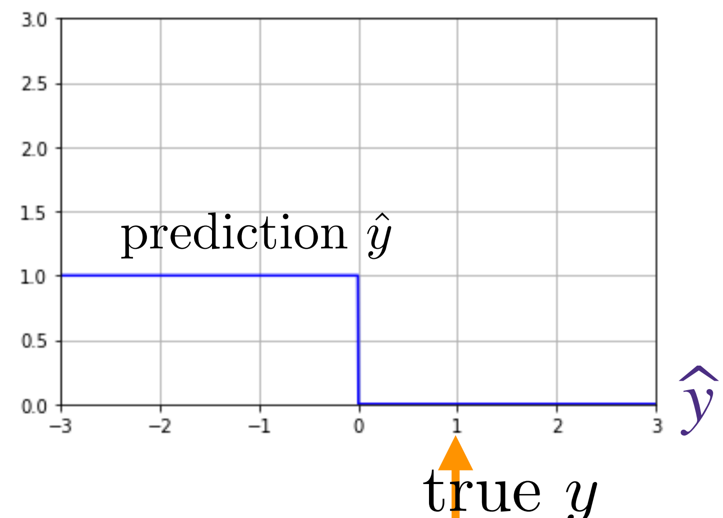
Binary Classification with 0-1 loss

- **Learn** a linear model: $f : x \mapsto y = b + x^T w$
  - x – input/features, $y \in \{-1, +1\}$ – label in target classes
  - Prediction: $\text{sign}(f(x))$
- **Ideal loss function** $\ell(\widehat{y}, y)$**:**
  - **0-1 loss**, because we care about how many were classified correctly
  - What are weaknesses?

$$\ell(\hat{y}, -1) = \begin{cases} 0 & \hat{y} < 0 \\ +1 & \hat{y} \geq 0 \end{cases}$$



$$\ell(\hat{y}, +1) = \begin{cases} 0 & \hat{y} > 0 \\ +1 & \hat{y} \leq 0 \end{cases}$$



Binary Classification with 0-1 loss

- If we know the underlying distribution, $(x, y) \sim P_{X,Y}$ and if we do not restrict ourselves to any function class, then we could find the optimal predictor called **Bayes optimal classifier**
  - $f_{\text{Bayes}}(x) = \arg\max_{\hat{y} \in \{-1,1\}} \mathbb{P}_{Y|X}(Y = \hat{y} \,|\, X = x)$
- Claim: Bayes optimal classifier achieves the minimum possible achievable **true error**
- True error: $\mathbb{E}_{X,Y}[\ell(f(X), Y)] = \mathbb{P}\Big(\operatorname{sign}(f(X)) \neq Y\Big)$
- Proof:
  We can write the true error of a classifier $f(\,\cdot\,)$ using chain rule as

  optimal classifier minimizes this true error, at every x

  $$f_{\text{opt}}(x) = \arg\min_{\hat{y} \in \{-1,1\}} \mathbb{P}_{Y|X}(Y \neq \hat{y} \,|\, x)$$
- But, we do not know $P_{X,Y}$ and 0-1 loss is cannot be optimizes (to be explained in lecture 11)

Binary Classification with 0-1 loss

- If we know the underlying distribution, $(x, y) \sim P_{X,Y}$ and if we do not restrict ourselves to any function class, then we could find the optimal predictor called **Bayes optimal classifier**
  - $f_{\text{Bayes}}(x) = \arg \max_{\hat{y} \in \{-1,1\}} \mathbb{P}_{Y|X}(Y = \hat{y} \,|\, X = x)$
- Claim: Bayes optimal classifier achieves the minimum possible achievable **true error**
- True error: $\mathbb{E}_{X,Y}[\ell(f(X), Y)] = \mathbb{P}\Big(\text{sign}(f(X)) \neq Y \Big)$
- Proof:

  We can write the true error of a classifier $f(\cdot)$ using chain rule as
  $\mathbb{E}_{X,Y}[\mathbb{I}\{Y \neq f(X)\}] = \mathbb{E}_X\Big[\mathbb{E}_{Y|X}[\mathbb{I}\{Y \neq f(x)\}] \,|\, X = x\Big] = \mathbb{E}_X\Big[\, \mathbb{P}_{Y|X}(Y \neq f(x) \,|\, X = x)\, \Big]$

  optimal classifier minimizes this true error, at every x
  $$f_{\text{opt}}(x) = \arg \min_{\hat{y} \in \{-1,1\}} \mathbb{P}_{Y|X}(Y \neq \hat{y} \,|\, x)$$
- But, we do not know $P_{X,Y}$ and 0-1 loss is cannot be optimizes (to be explained in lecture 11)

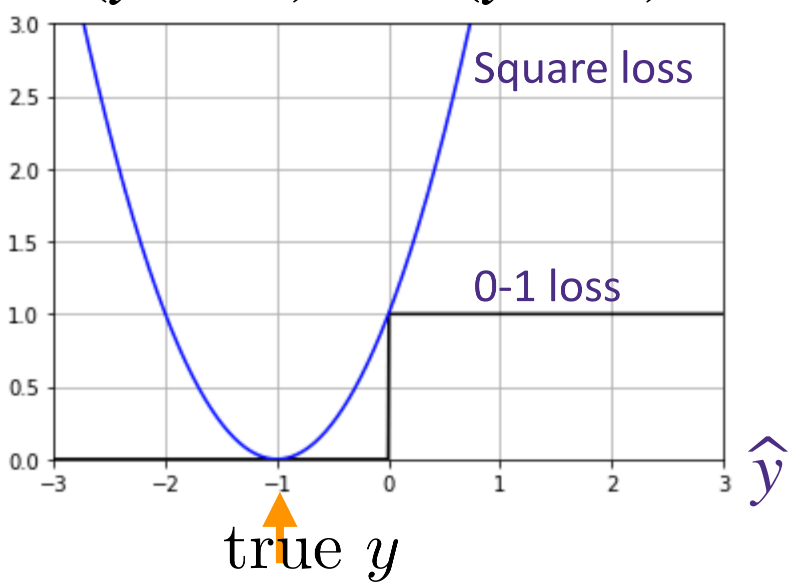
Binary Classification with square loss

- **Learn** a linear model: $f : x \mapsto y = b + x^T w$
  - x input/features, $y \in \{-1, +1\}$ label in target classes
  - Prediction: $\text{sign}(f(x))$
- **Square loss function** $\ell(b + x^T w, y) = (y - x^T w - b)^2$
  - This is the same as treating this as a linear regression problem

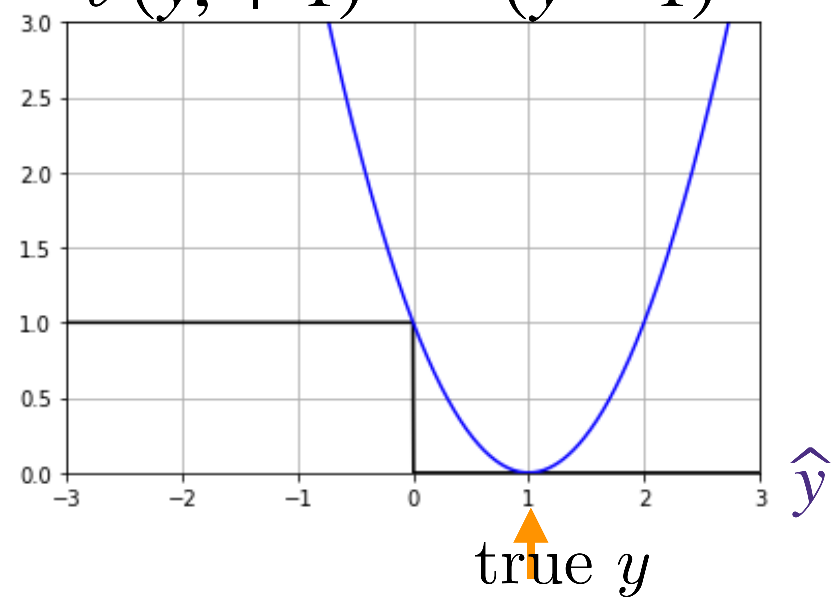
$$(\widehat{w}, \widehat{b}) = \arg\min_{b,w} \sum_{i=1}^{n} (y_i - (b + x_i^T w))^2$$

- What is the strengths and weaknesses?

$$\ell(\hat{y}, -1) = (\hat{y} + 1)^2$$



$$\ell(\hat{y}, +1) = (\hat{y} - 1)^2$$



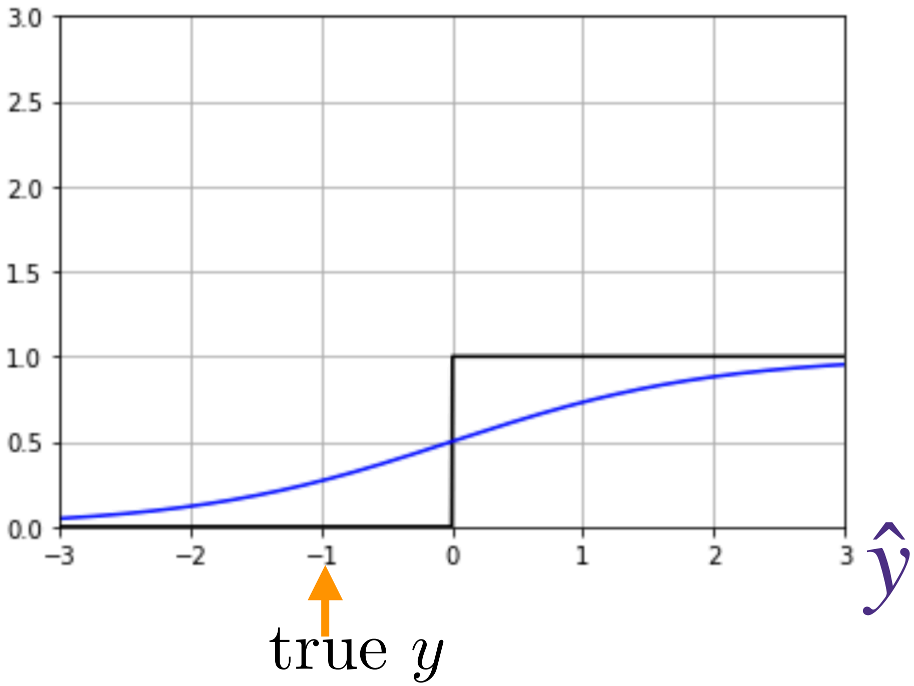
Looking for a better loss function

- we get better results using loss functions that
  - approximate, or captures the flavor of, the 0-1 loss
  - is more easily optimized (e.g. convex and/or non-zero derivatives)

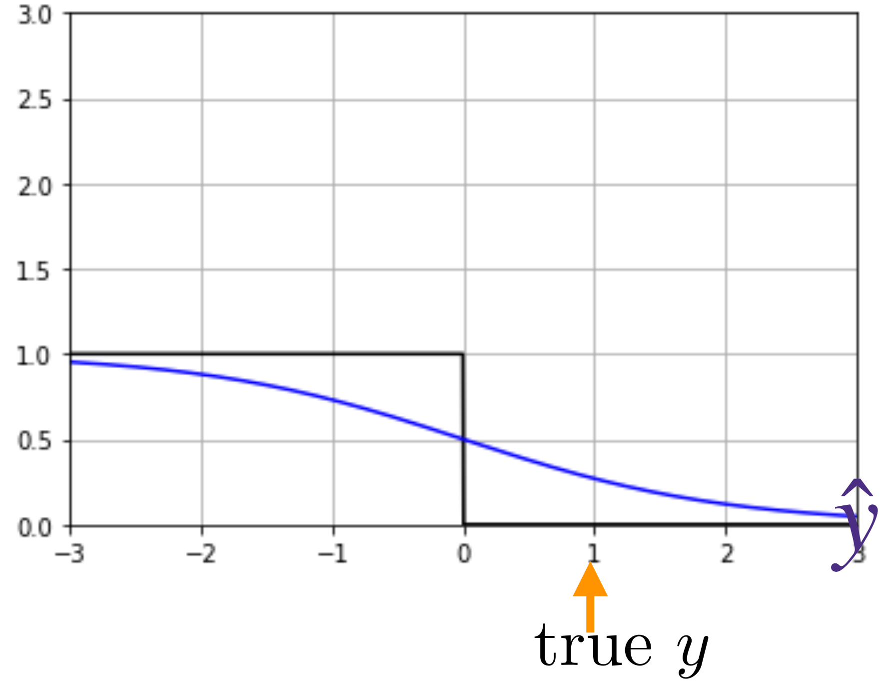
- concretely, we want a **loss function**
  - with $\ell(\hat{y}, -1)$ small when $\hat{y} < 0$ and larger when $\hat{y} > 0$
  - with $\ell(\hat{y}, 1)$ small when $\hat{y} > 0$ and larger when $\hat{y} < 0$
  - Which has other nice characteristics, e.g., differentiable or convex

Sigmoid loss $\ell(\hat{y}, y) = \dfrac{1}{1 + e^{y\hat{y}}}$

$$\ell(\hat{y}, -1) = \frac{1}{1+e^{-\hat{y}}}$$



$$\ell(\hat{y}, +1) = \frac{1}{1 + e^{\hat{y}}}$$



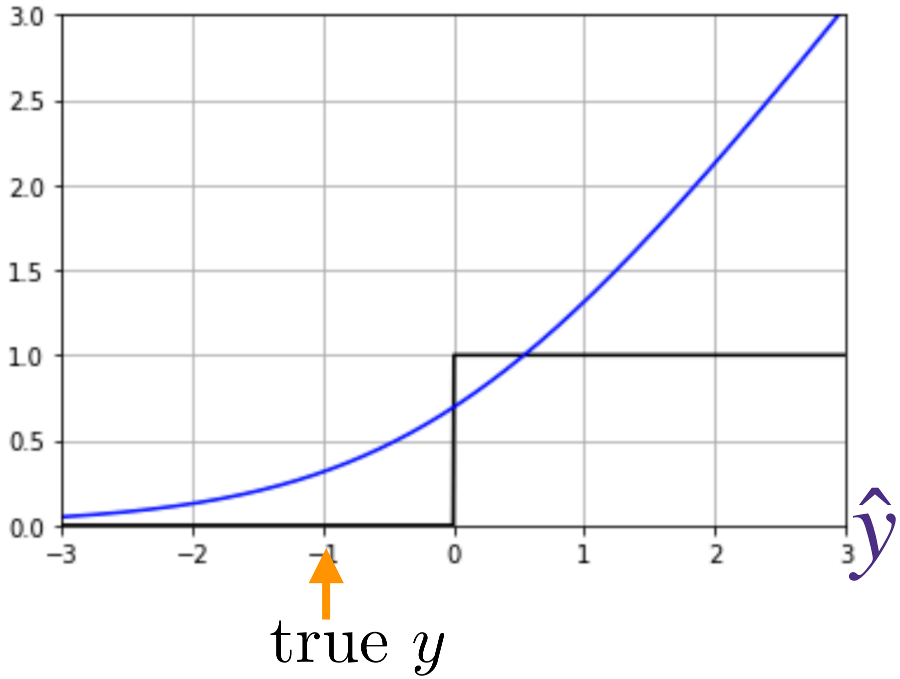
- differentiable approximation of 0-1 loss
- but not convex in $\hat{y}$
- the two losses sum to one

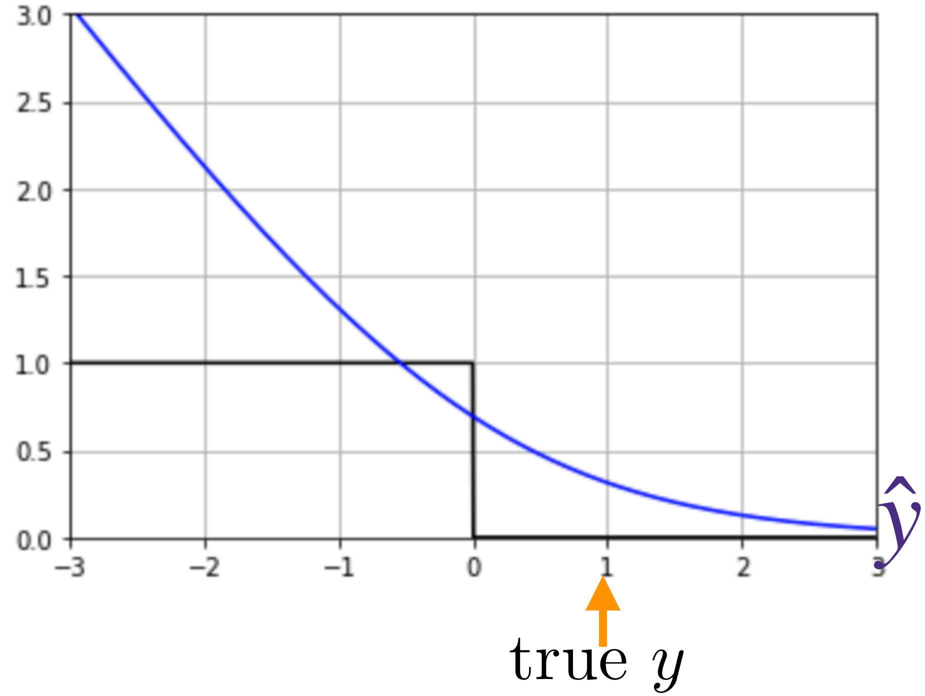
$$\frac{1}{1 + e^{-\hat{y}}} + \frac{1}{1 + e^{\hat{y}}} = \frac{e^{\hat{y}}}{e^{\hat{y}} + 1} + \frac{1}{1 + e^{\hat{y}}} = 1$$

- softer (or smoothed) version of the 0-1 loss

Logistic loss $\ell(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$

$\ell(\hat{y}, -1) = \log(1 + e^{\hat{y}})$ $\ell(\hat{y}, +1) = \log(1 + e^{-\hat{y}})$





- differentiable and convex in $\hat{y}$
- approximation of 0-1
- Most popular choice of a loss function for classification problems

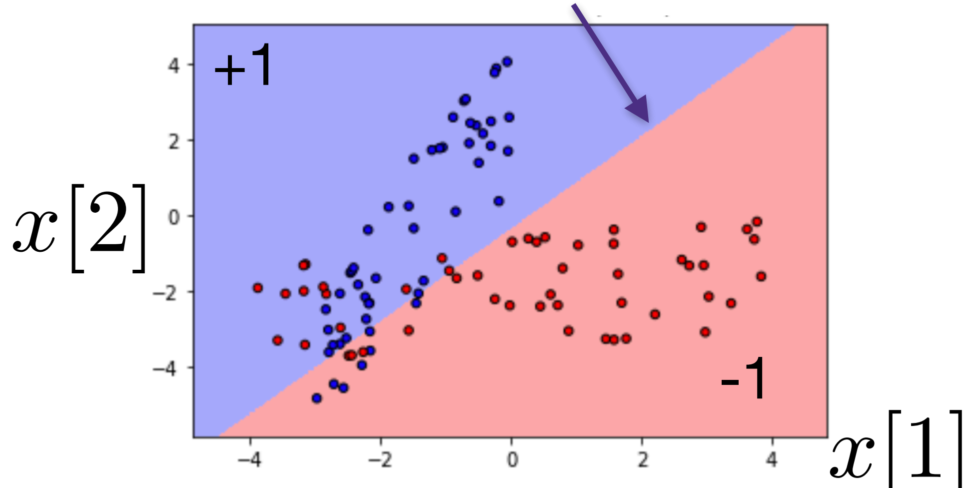
Logistic regression for binary classification

- Data $\mathcal{D} = \{(x_i \in \mathbb{R}^d, y_i \in \{-1, +1\})\}_{i=1}^n$
- Model: $y = x^T w + b$
- Loss function: logistic loss $\ell(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$
- Optimization: solve for

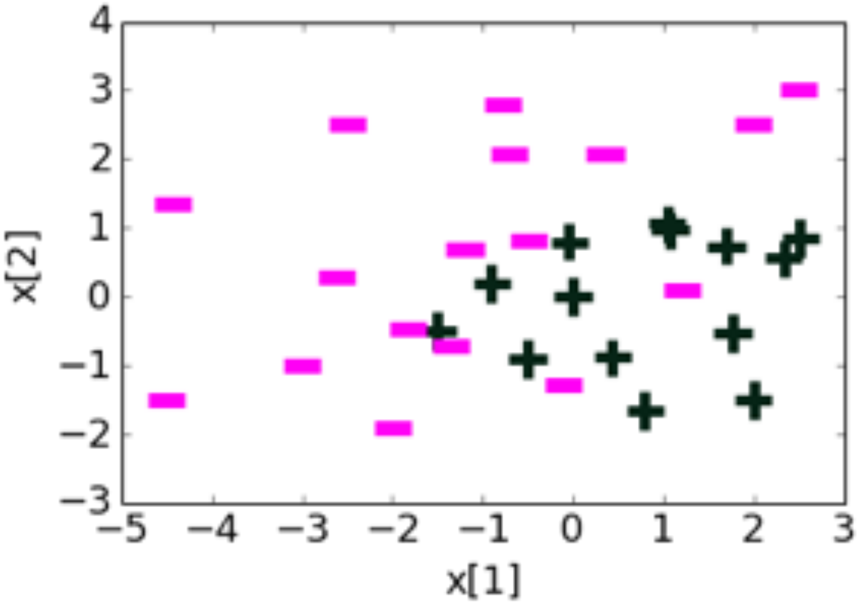
$$(\widehat{b}, \widehat{w}) = \arg\min_{b,w} \sum_{i=1}^{n} \log(1 + e^{-y_i(b + x_i^T w)})$$

- As this is a smooth convex optimization, it can be solved efficiently using gradient descent
- Prediction: $\text{sign}(b + x^T w)$

decision boundary at $w^T x + b = 0$



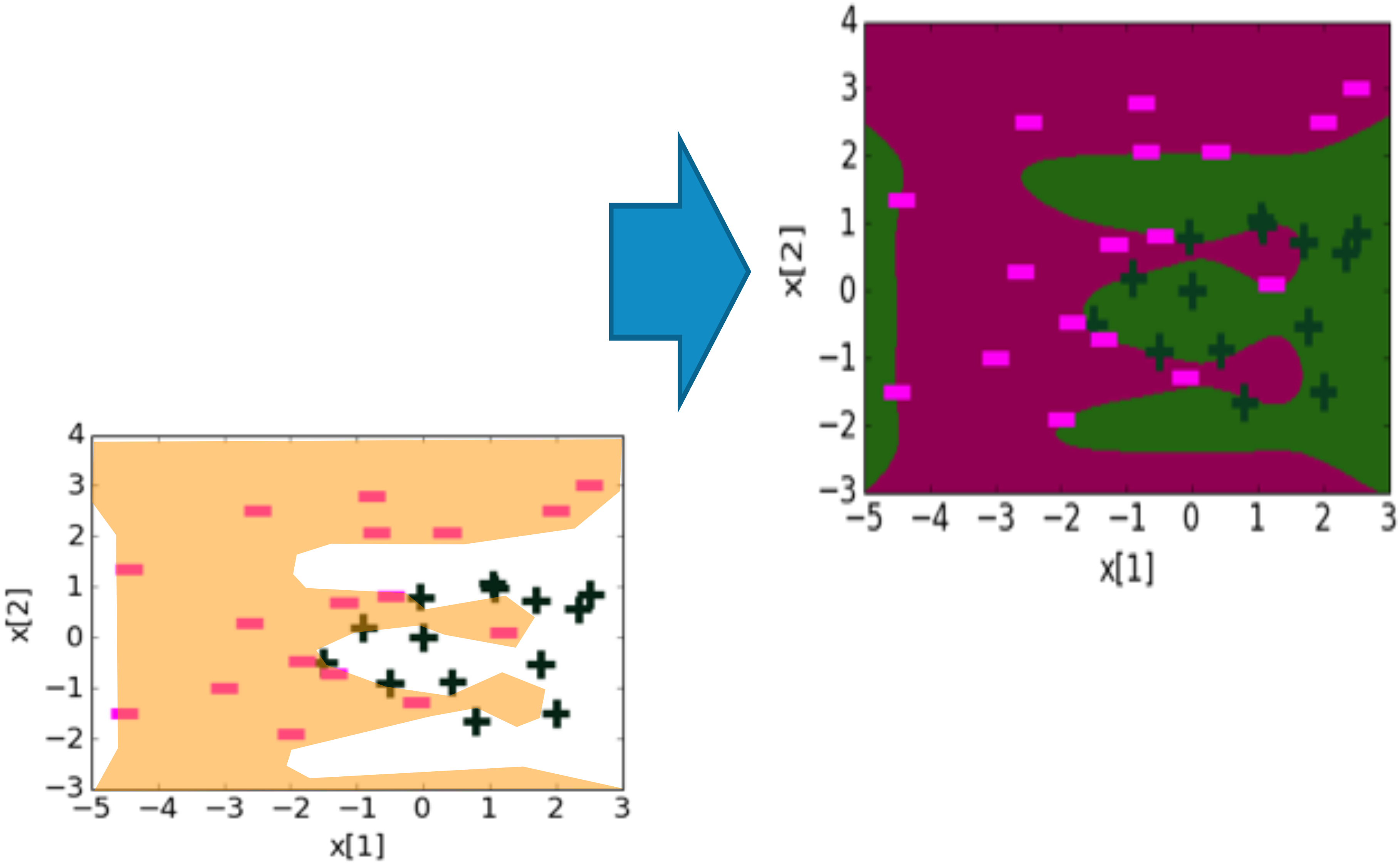
Example: adding more polynomial features



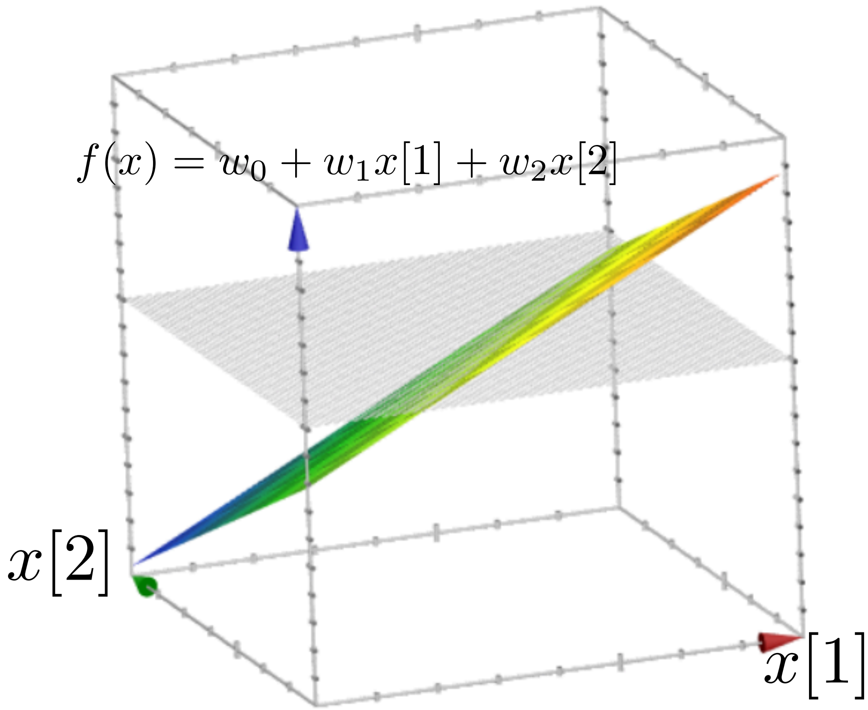
Polynomial features

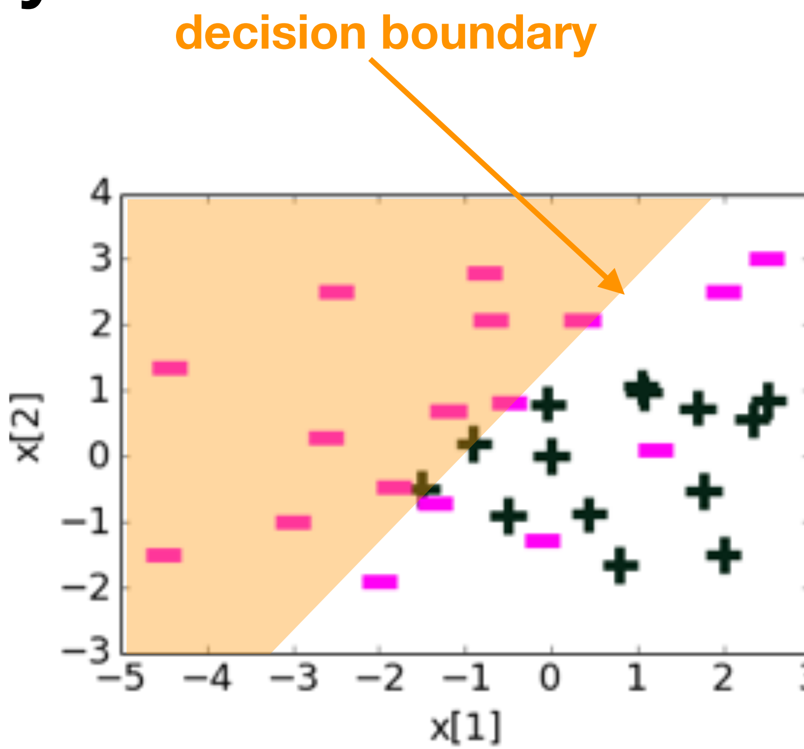
$$\begin{bmatrix} h_0(x) = 1 \\ h_1(x) = x[1] \\ h_2(x) = x[2] \\ h_3(x) = x[1]^2 \\ h_4(x) = x[2]^2 \\ \vdots \end{bmatrix}$$

- data: ***x*** in 2-dimensions, ***y*** in {+1,-1}
- features: polynomials
- model: linear on polynomial features
- $$f(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$


x[1]
x[2]
x[1]
x[2]

Learned decision boundary

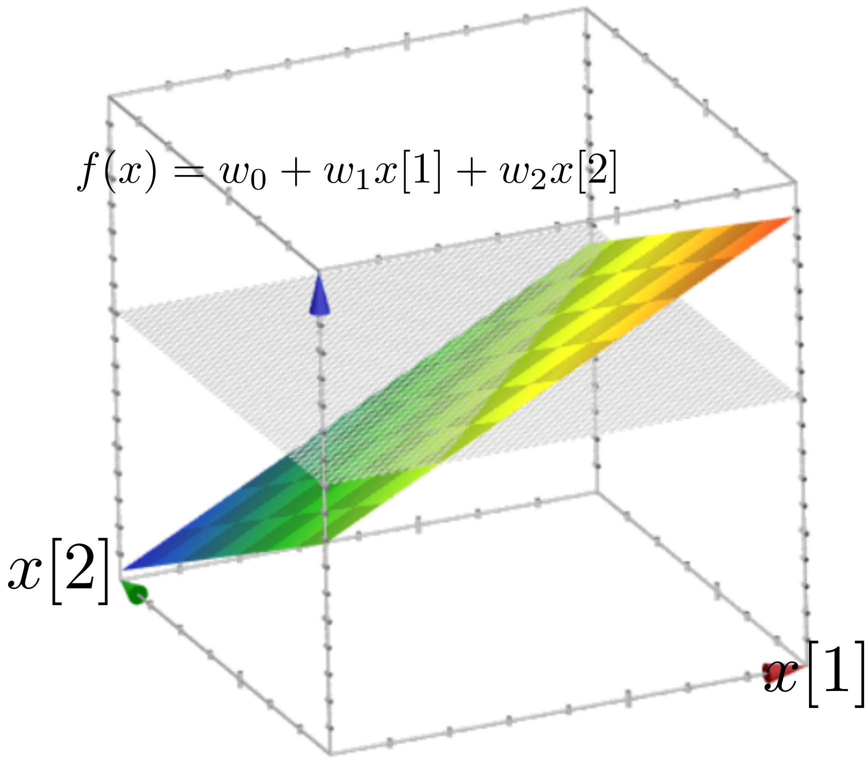


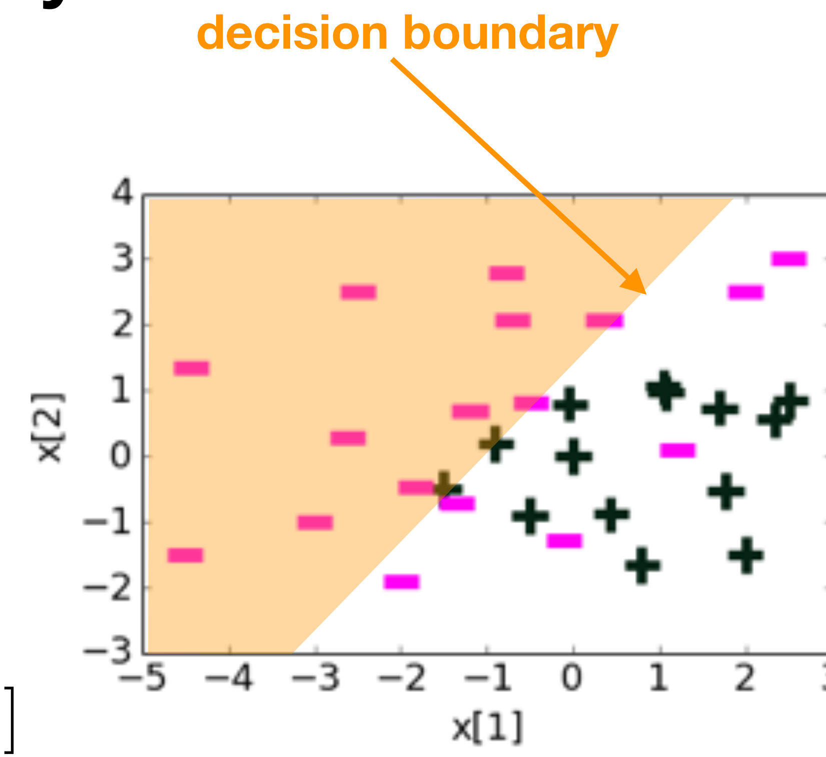


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary

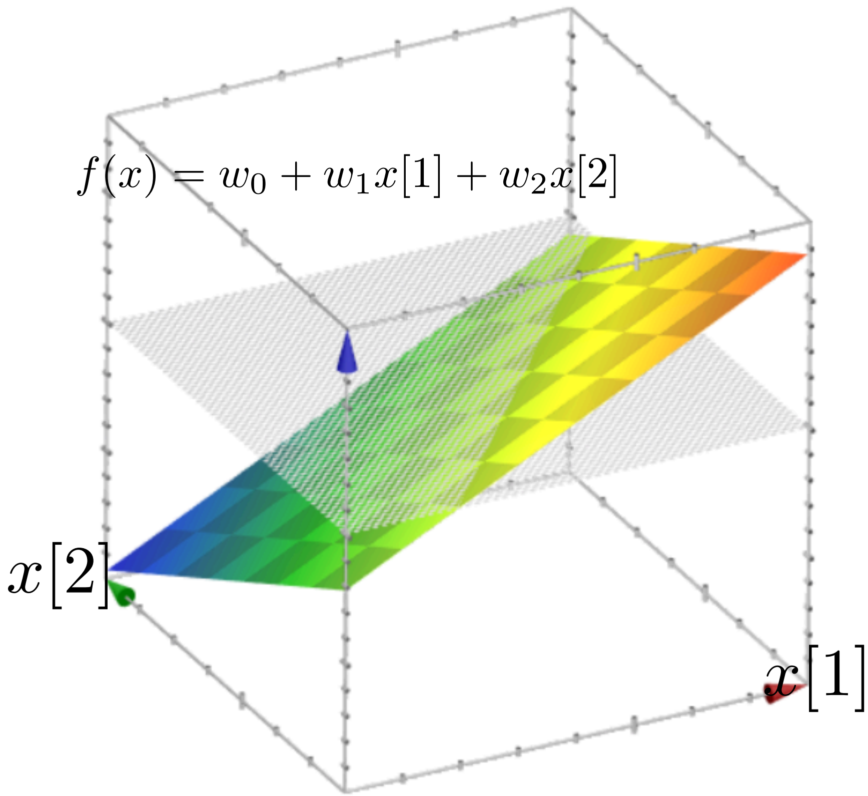


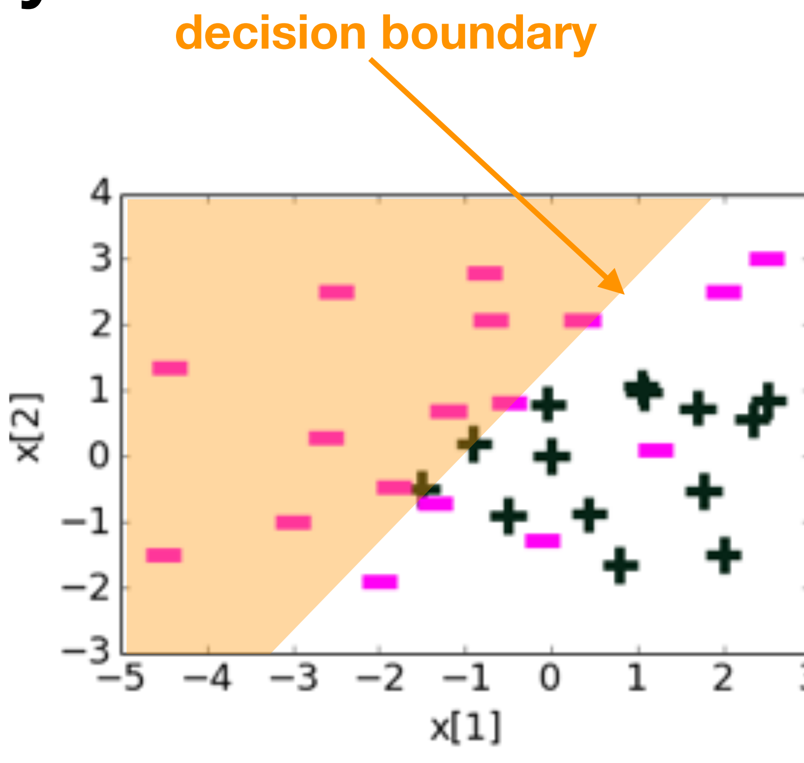


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary

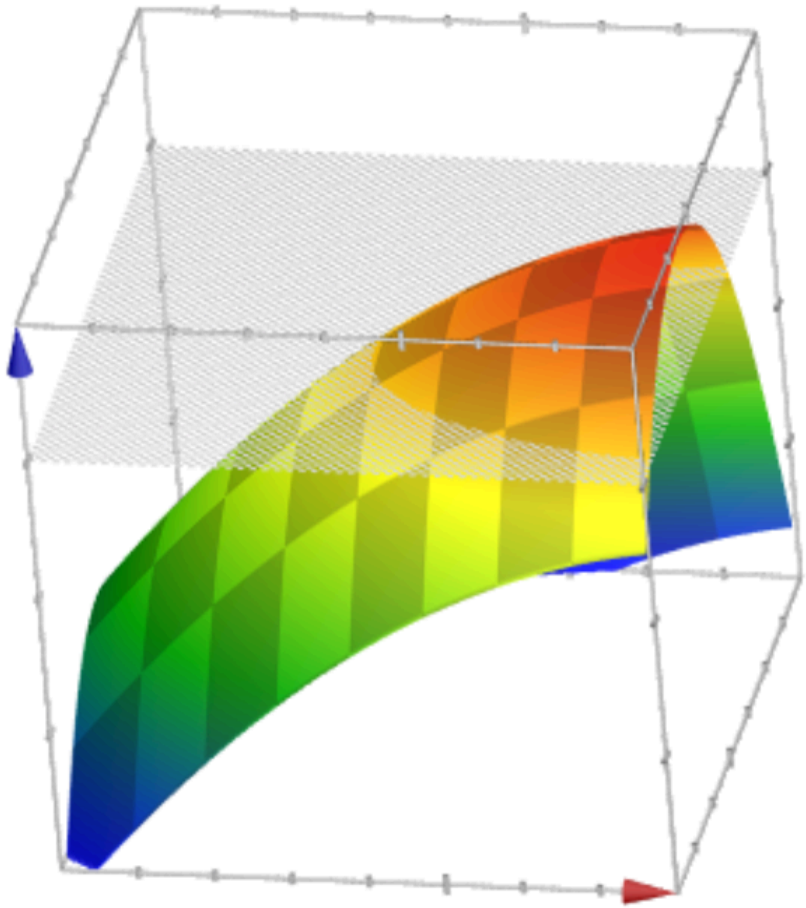


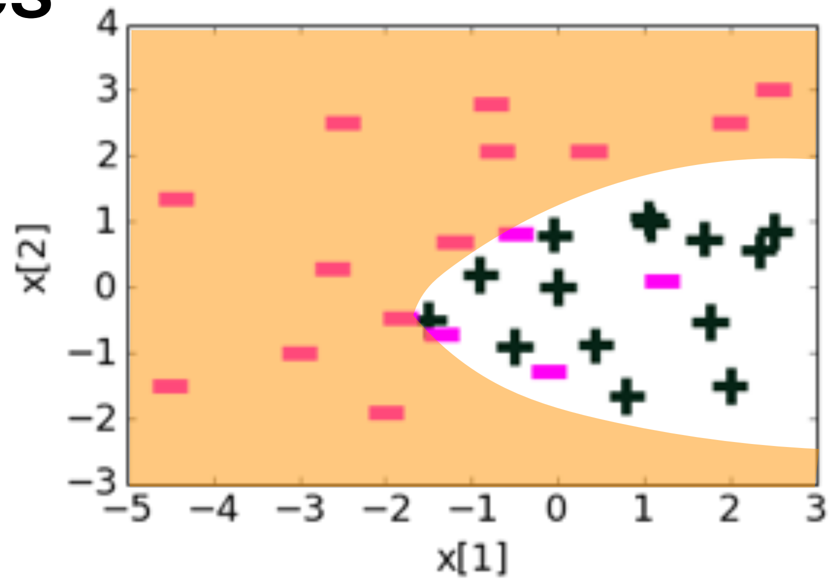


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Adding quadratic features

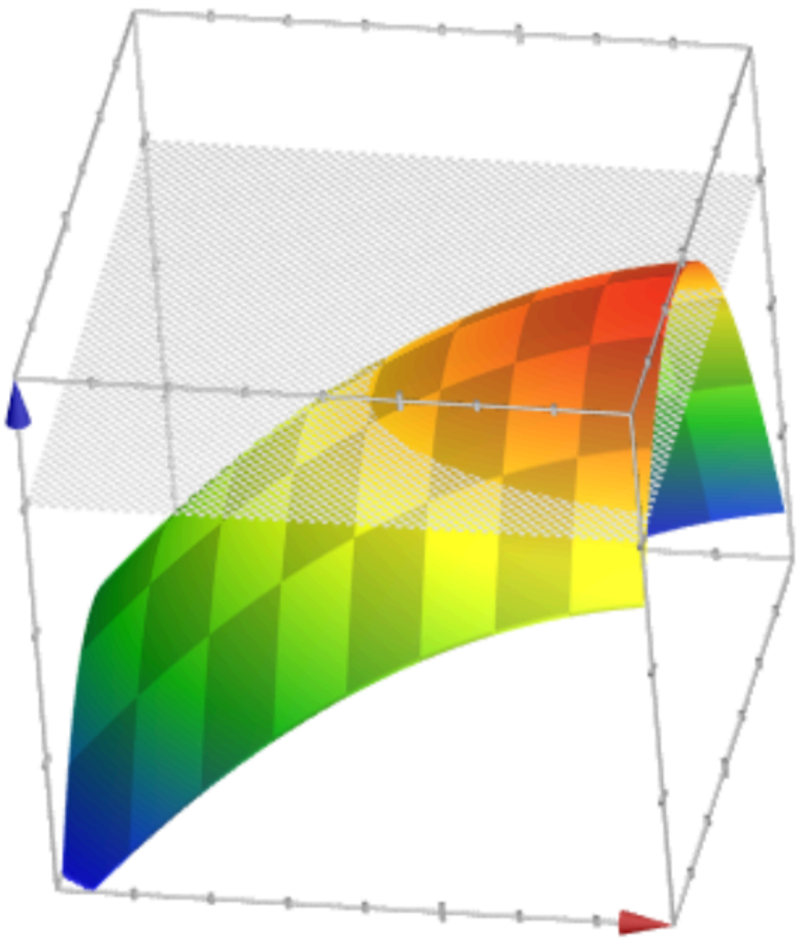


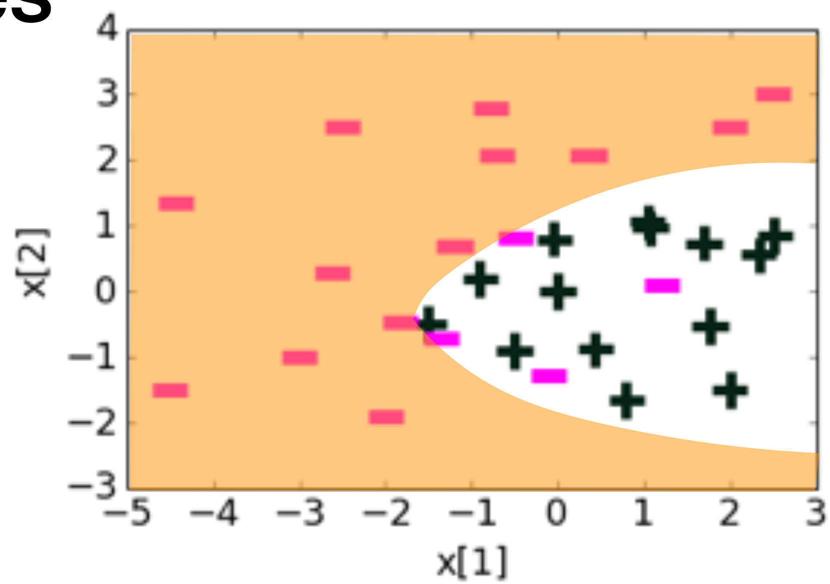


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features

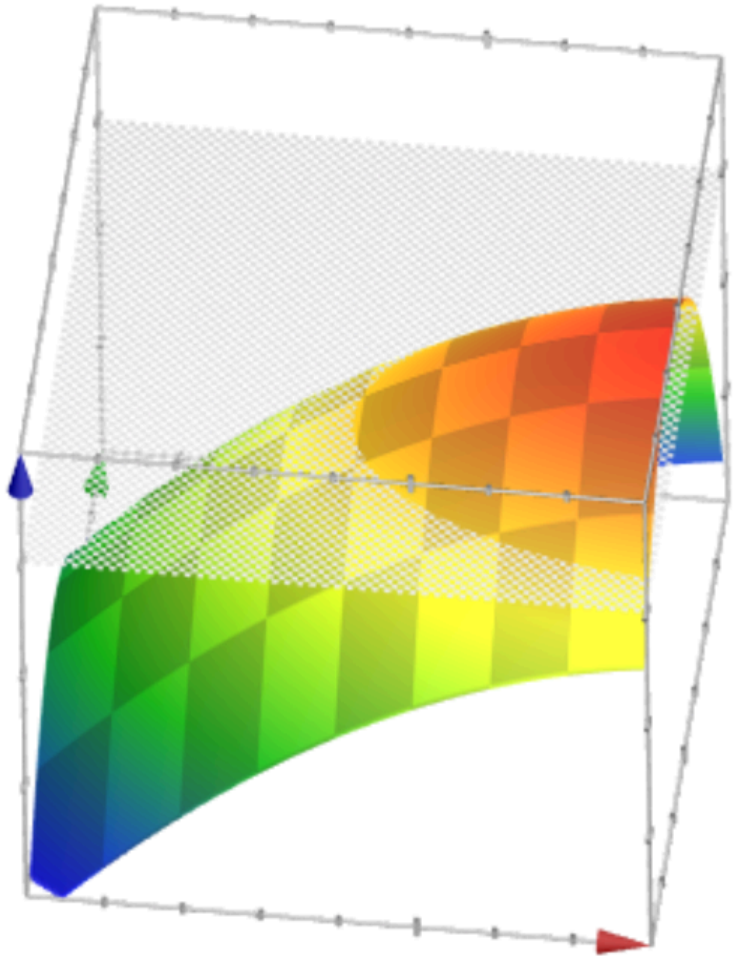


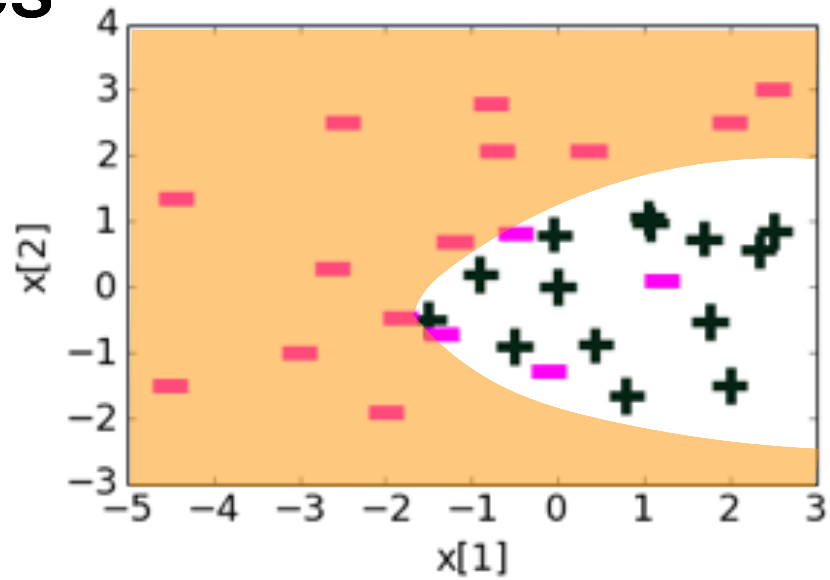


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features



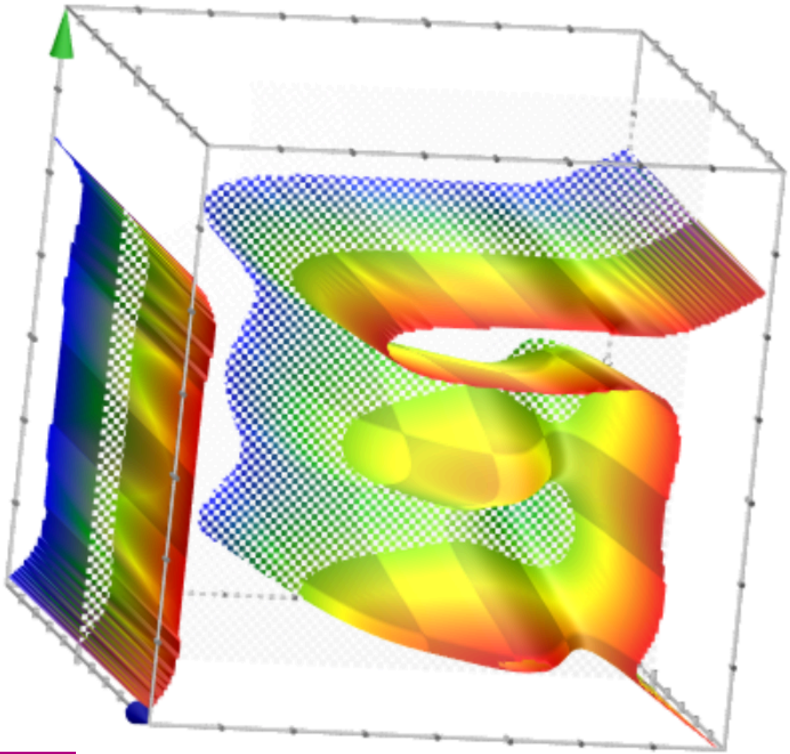


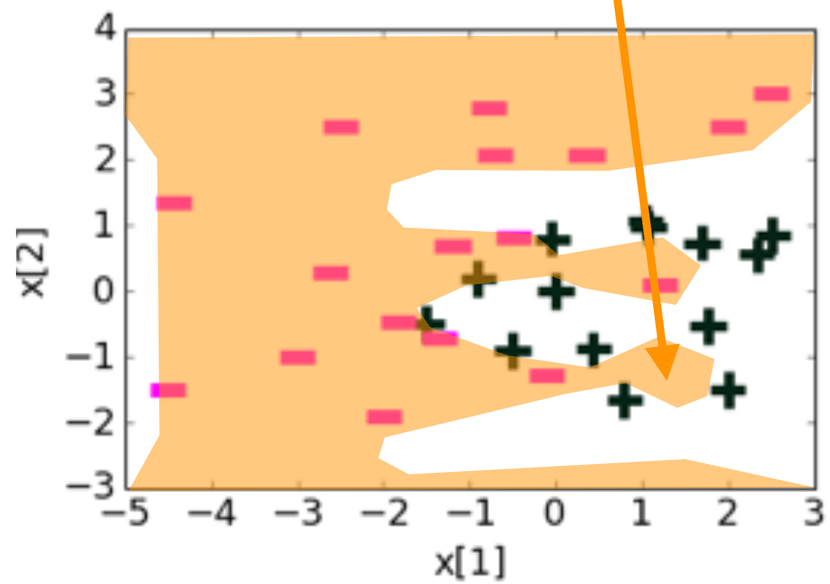
Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding higher degree polynomial features

Overfitting leads to non-generalization

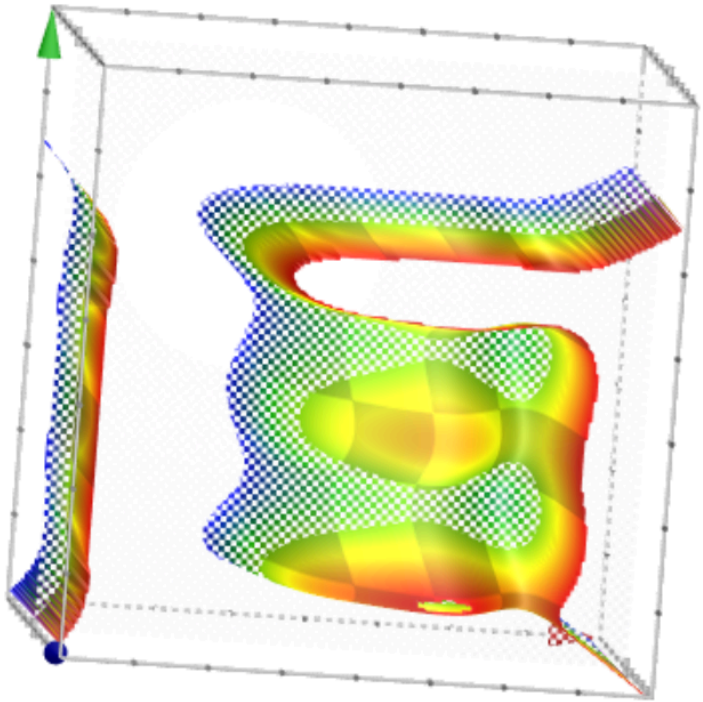




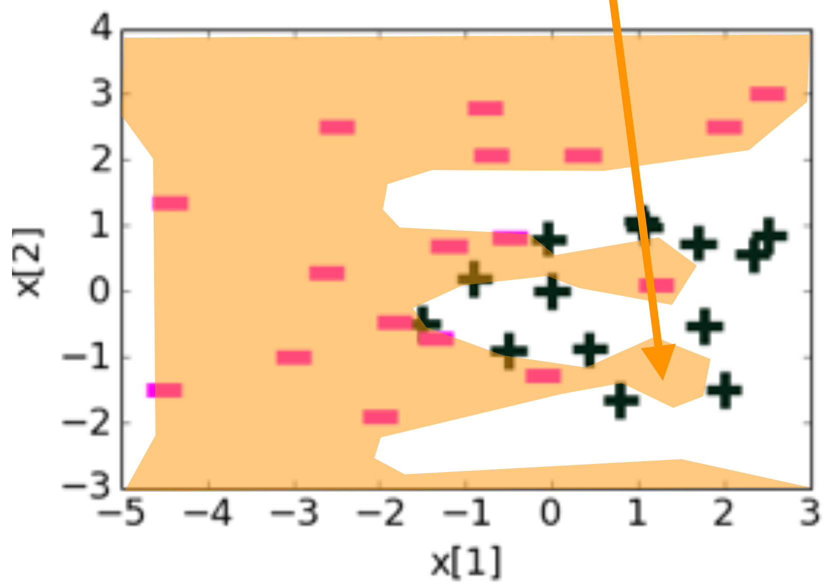
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features



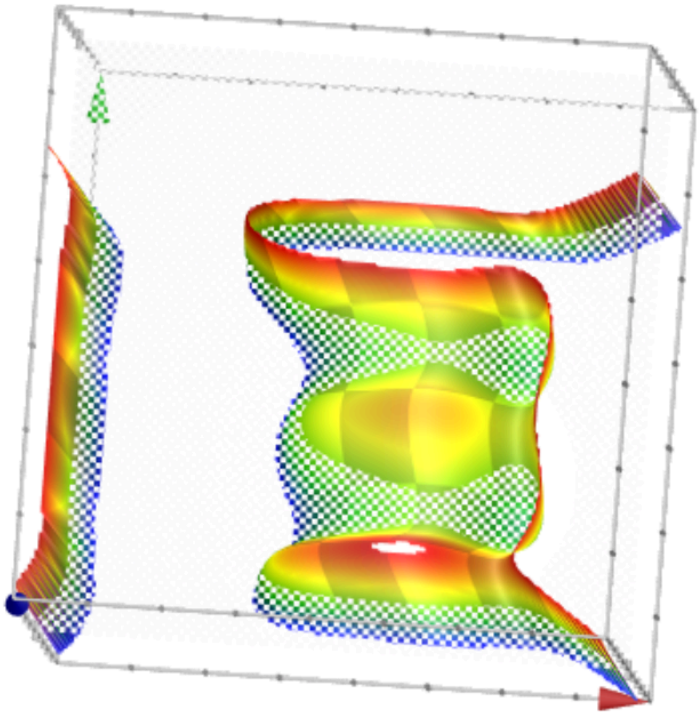
Overfitting leads to non-generalization



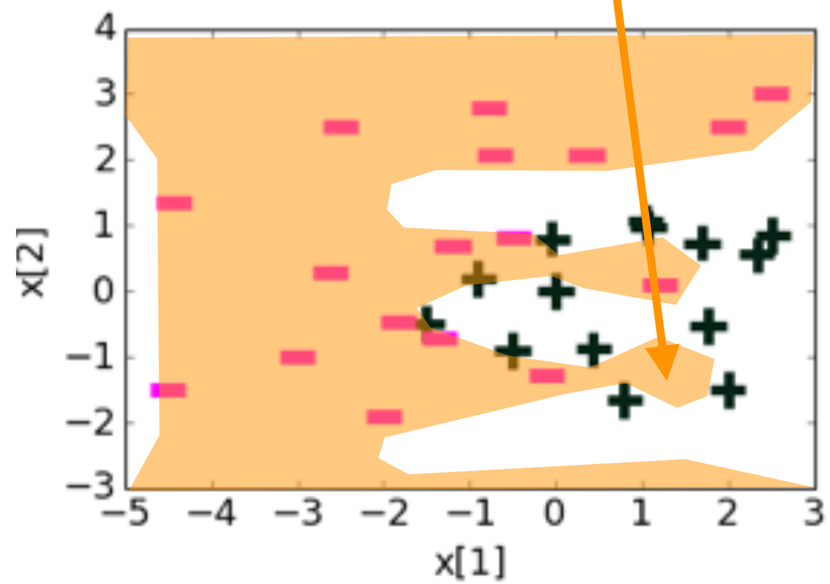
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features



Overfitting leads to non-generalization



Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

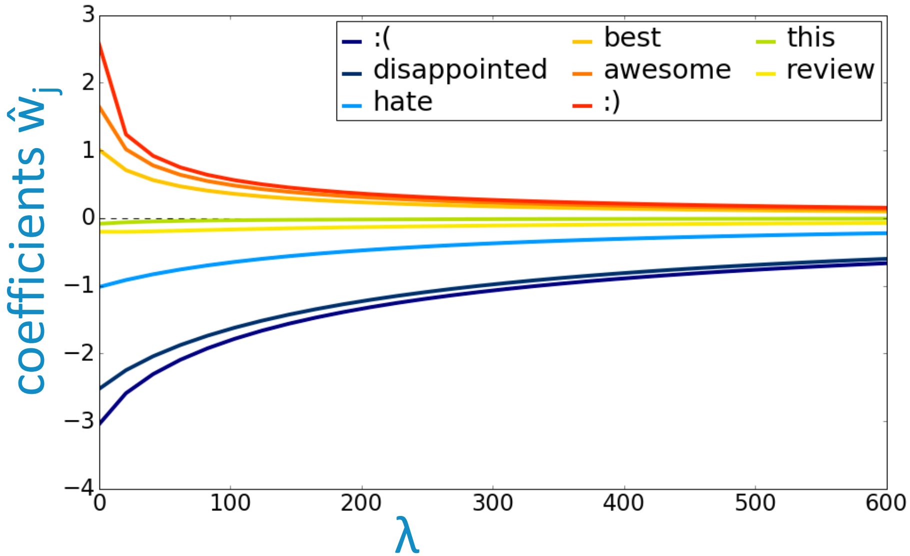
Coefficient values getting large

- Overfitting leads to very large values of

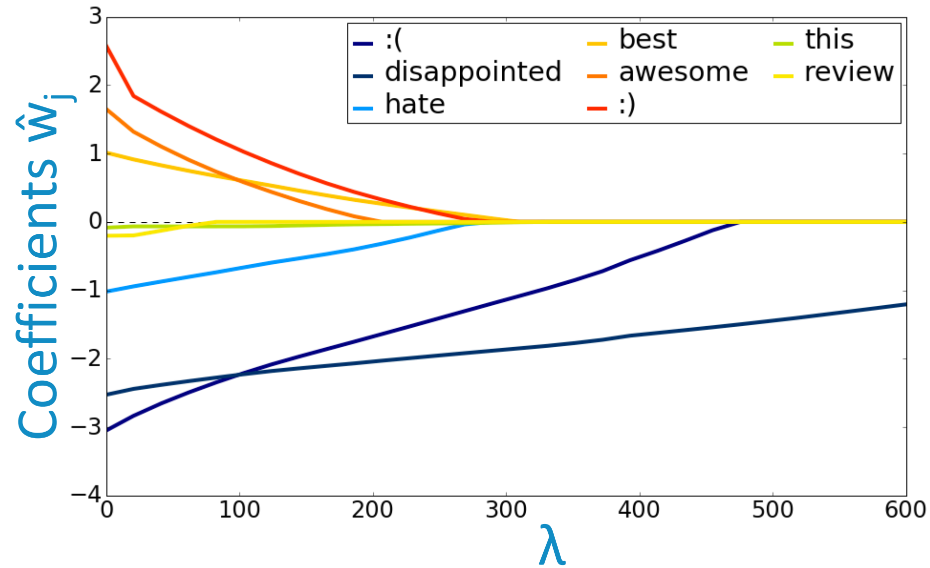
$$f(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

Regularization path

ℓ_2 regularizer: $\|W\|_2^2 = |w_1|^2 + \cdots + |w_d|^2$



ℓ_1 regularizer: $\|w\|_1 = |w_1| + \cdots + |w_d|$



- Absolute regularizer (a.k.a L1 regularizer) gives sparse parameters, which is desired for interpretability, feature selection, and efficiency

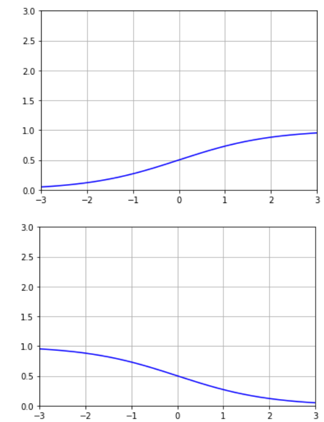
Probabilistic interpretation of logistic regression

- just as Maximum Likelihood Estimator (MLE) under linear model and additive Gaussian noise model recovers **linear least squares**,
- we study a particular noise model that recovers **logistic regression**

- a probabilistic noise model for Boolean labels:

$$\mathbb{P}(y_i = +1 \mid x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

$$\mathbb{P}(y_i = -1 \mid x_i) = \frac{1}{1 + e^{w^T x_i}}$$



$w^T x_i$

with a ground truth model parameter $w \in \mathbb{R}^d$

- this function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$ is called a **logistic function** (not to be confused with logistic loss, which is different) or a **sigmoid function**
- if we know that the data came from such a model, but do not know the ground truth parameter $w \in \mathbb{R}^d$, we can apply MLE to find the best w
- this MLE recovers the logistic regression algorithm, exactly

Maximum Likelihood Estimator (MLE)

- if the data came from a probabilistic model model:

$$(\underbrace{\frac{1}{1+e^{-w^T x}}}_{\mathbb{P}(y_i=+1|x_i)}, \underbrace{\frac{1}{1+e^{w^T x}}}_{\mathbb{P}(y_i=-1|x_i)})$$

- log-likelihood of observing a data point (x_i, y_i) is

$$\text{log-likelihood} = \log\Big(\mathbb{P}(y_i|x_i)\Big) = \begin{cases} \log\left(\frac{1}{1+e^{-w^T x_i}}\right) & \text{if } y_i = +1 \\ \log\left(\frac{1}{1+e^{w^T x_i}}\right) & \text{if } y_i = -1 \end{cases}$$

- Maximum Likelihood Estimator is the one that maximizes the sum of all log-likelihoods on training data points

$$\hat{w}_{\text{MLE}} = \arg\max_{w} \; \mathbb{P}(\{y_1, \ldots, y_n\} \,|\, \{x_1, \ldots, x_n\})$$

$$= \arg\max_{w} \prod_{i=1} \mathbb{P}(y_i \,|\, x_i)$$

(independence)

$$= \arg\max_{w} \sum_{i:y_i=-1} \log\Big(\frac{1}{1+e^{w^T x_i}}\Big) + \sum_{i:y_i=1} \log\Big(\frac{1}{1+e^{-w^T x_i}}\Big)$$

(substitution)

- notice that this is exactly the **logistic regression:**

$$\hat{w}_{\text{logistic}} = \arg\min_{w} \frac{1}{n}\Big(\sum_{i:y_i=-1} \log(1+e^{w^T x_i}) + \sum_{i:y_i=1} \log(1+e^{-w^T x_i}) \Big)$$

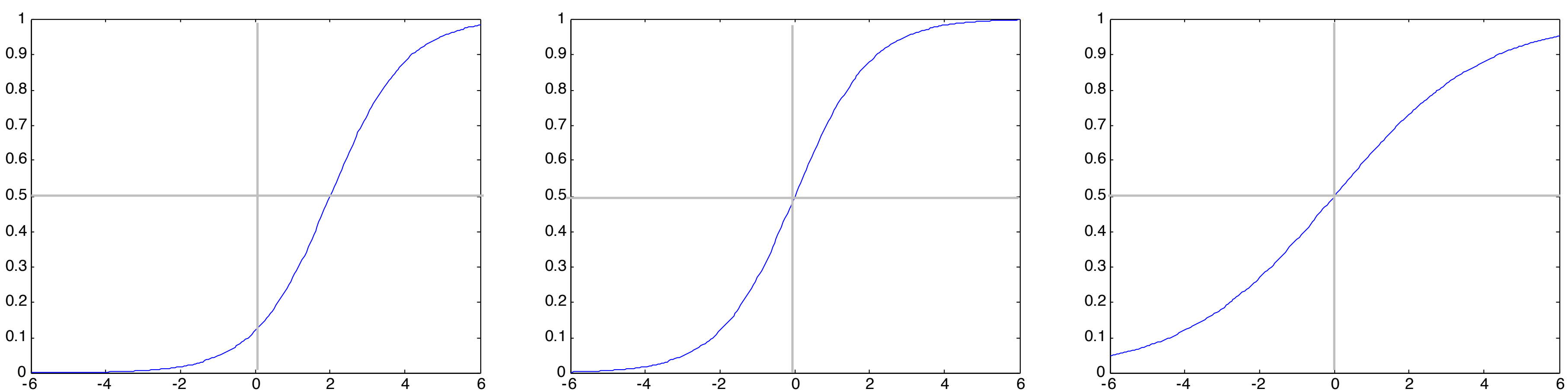
- once we have trained a model $\hat{w}_{\text{logistic}}$, we can make a hard prediction $\hat{v}$ of the label at an input example x

$$\begin{aligned}
\hat{v} &= \begin{cases} +1 & \text{if } \mathbb{P}(+1|x) \geq \mathbb{P}(-1|x) \\ -1 & \text{otherwise} \end{cases} \\
&= \begin{cases} +1 & \text{if } \frac{1}{1+e^{-w^T x}} \geq \frac{1}{1+e^{w^T x}} \\ -1 & \text{otherwise} \end{cases} \\
&= \begin{cases} +1 & \text{if } 1 \leq e^{2w^T x} \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign}(w^T x)
\end{aligned}$$

Understanding the sigmoid

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{w_0 + \sum_i w_i x_i}}$$

w_0=-2, w_1=-1 w_0=0, w_1=-1 w_0=0, w_1=-0.5

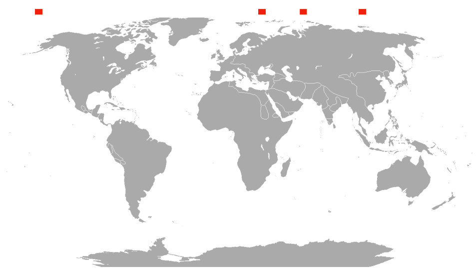


Multi-class regression

How do we encode categorical data *y*?

- so far, we considered Boolean case where there are two categories
- encoding y is simple: {+1,-1}, as there is not much difference

- multi-class classification predicts categorial y
- taking values in $C = \{c_1, \ldots, c_k\}$
- c_j's are called
- examples:



Country of birth
(Argentina, Brazil, USA,...)

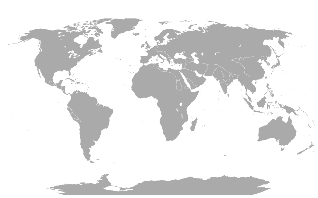
Zipcode
(10005, 98195,...)

All English words

- a **k-class classifier** predicts y given x

Embedding c_j's in real values

- for optimization we need to **embed** raw categorical c_j's into real valued vectors
- there are many ways to embed categorial data
  - True->1, False->-1
  - Yes->1, Maybe->0, No->-1
  - Yes->(1,0), Maybe->(0,0), No->(0,1)
  - Apple->(1,0,0), Orange->(0,1,0), Banana->(0,0,1)
  - Ordered sequence:
    (Horse 3, Horse 1, Horse 2) -> (3,1,2)
- we use **one-hot embedding** (a.k.a. **one-hot encoding**)
  - is a st dard basis vector in $k-$ dimension



Country of birth
(Argentina, Brazil, USA,...)

196 categories

1-hot encoding

x	$h_1(x)$	$h_2(x)$	…	$h_{195}(x)$	$h_{196}(x)$
Brazil		1			
Zimbabwe					1

196 features

Multi-class logistic regression

- data: categorical y in $\{c_1, \ldots, c_k\}$ with k categories

  we use one-hot encoding, s.t. $y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ implies that $y = c_1$

- model: linear vector-function makes a linear prediction $\hat{y} \in \mathbb{R}^k$

$$\hat{y}_i \;=\; f(x_i) \;=\; w^T x_i$$

with model parameter matrix $w \in \mathbb{R}^{d \times k}$ and sample $x_i \in \mathbb{R}^d$

$$f(x_i) \;=\; \begin{bmatrix} f_1(x_i) \\ f_2(x_i) \\ \vdots \\ f_k(x_i) \end{bmatrix} \;=\; \underbrace{\begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & \cdots \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots \\ \vdots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & \cdots \end{bmatrix}}_{w^T} \underbrace{\begin{bmatrix} 1 \\ x_i[1] \\ \vdots \\ x_i[d] \end{bmatrix}}_{x_i} \;=\; \begin{bmatrix} w_{1,0} + w_{1,1}x_i[1] + w_{1,2}x_i[2] + \cdots \\ w_{2,0} + w_{2,1}x_i[1] + w_{2,2}x_i[2] + \cdots \\ \vdots \\ w_{k,0} + w_{k,1}x_i[1] + w_{k,2}x_i[2] + \cdots \end{bmatrix}$$

$$w = \begin{bmatrix} w[:,1] & w[:,2] & \cdots & w[:,k] \end{bmatrix}$$

- Logistic regression

2 classes

$$\mathbb{P}(y_i = -1 \,|\, x_i) = \frac{1}{1 + e^{w^T x_i}}$$

$$\mathbb{P}(y_i = +1 \,|\, x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

k classes

$$\mathbb{P}(y_i = c_1 \,|\, x_i) = \frac{e^{w[:,1]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

$$\vdots$$

$$\mathbb{P}(y_i = c_k \,|\, x_i) = \frac{e^{w[:,k]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

Maximum Likelihood Estimator

$$\text{maximize}_w \ \frac{1}{n} \sum_{i=1}^{n} \log(\mathbb{P}(y_i \,|\, x_i))$$

$$\text{maximize}_{w \in \mathbb{R}^d} \ \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{1 + e^{-y_i w^T x_i}}\right)$$

$$\text{maximize}_{w \in \mathbb{R}^{d \times k}} \frac{1}{n} \sum_{i=1}^{n} \boxed{\sum_{j=1}^{k} \mathbf{I}\{y_i = c_j\} \log\left(\frac{e^{w[:,j]^T x_i}}{\sum_{j'=1}^{k} e^{w[:,j']^T x_i}}\right)}$$

$\mathbf{I}\{y_i = j\}$ is an indicator that is one only if $y_i = j$

Questions?