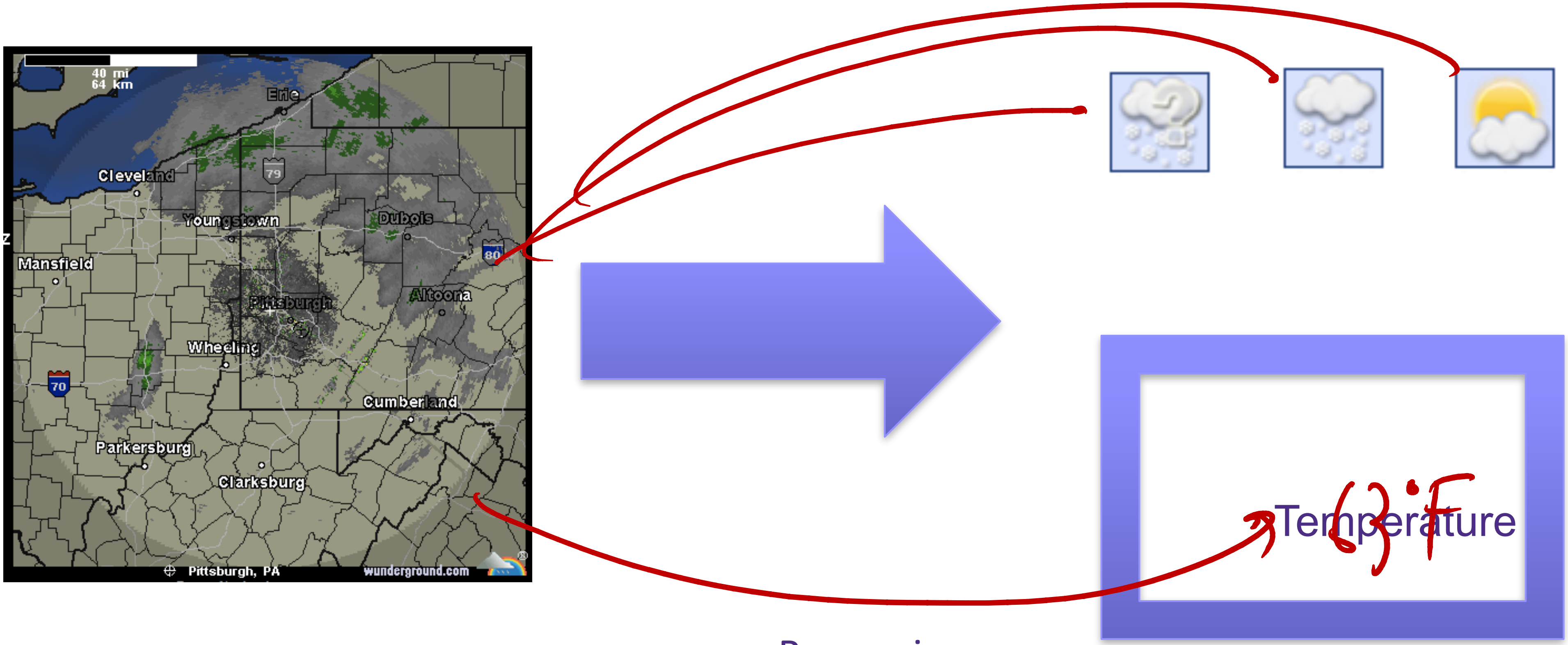
Classification
Logistic Regression

UNIVERSITY *of* WASHINGTON

Thus far, regression:

predict a continuous value given some inputs

Weather prediction revisted



Classification

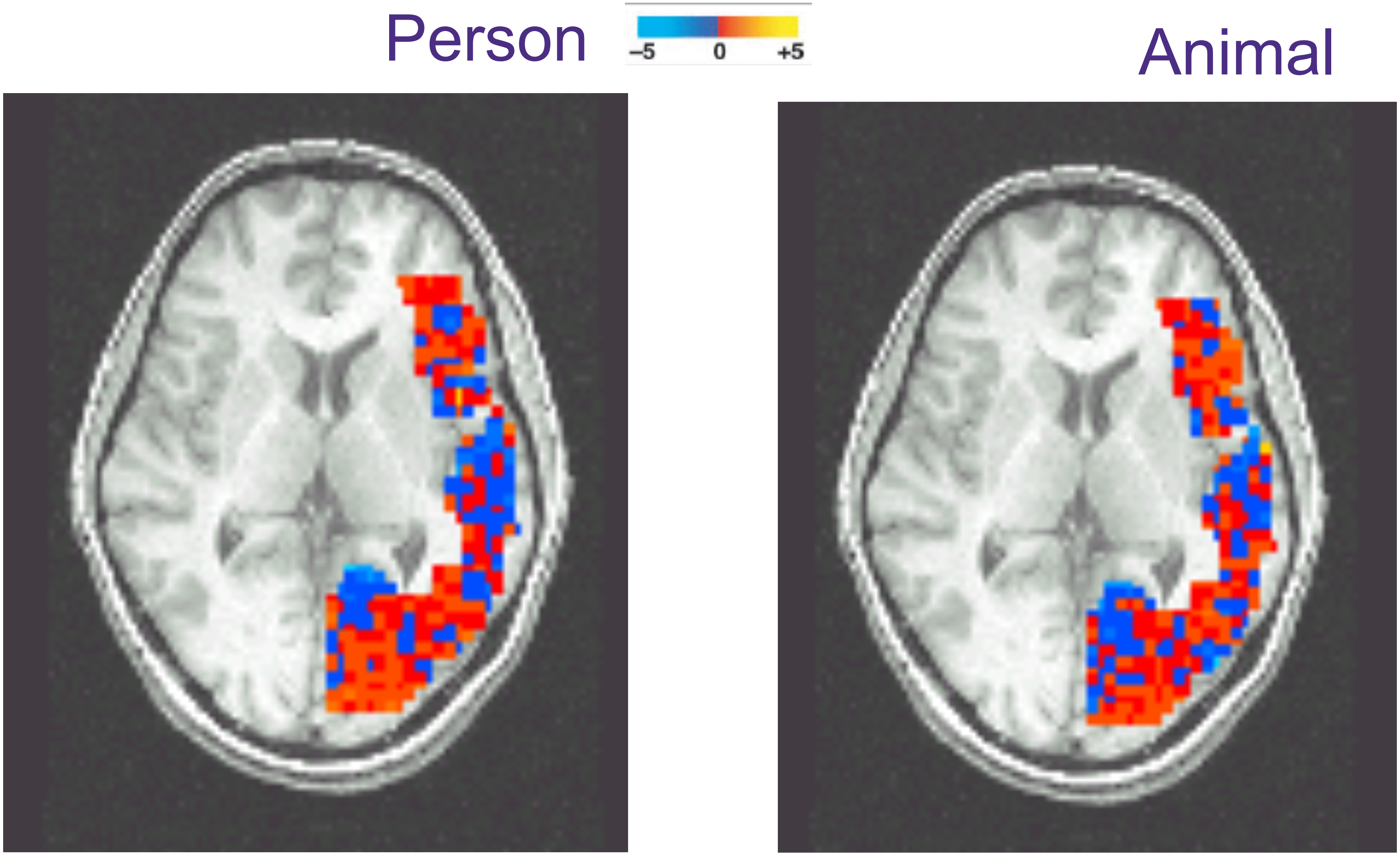
Temperature (?)°F

Regression

Reading Your Brain, Simple Example

[Mitchell et al.]

Pairwise classification accuracy: 85%


Person

Animal

Classification

$(X,Y) \overset{iid}{\sim} \mathcal{D}$

- **Learn** $f : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X} \subset \mathbb{R}^d$ **- features**
  - $\mathcal{Y} = \{1, \ldots, k\}$ **- target classes**

- **Loss Function** $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$
- **Expected loss of f:**

$$\mathbb{E}_{XY}\left[\ell(f(X),Y)\right] = \mathbb{E}_{XY}\left[\mathbb{1}\{f(X) \neq Y\}\right]$$

$$= \mathbb{E}_{X}\left[\underbrace{\mathbb{E}\left[\mathbb{1}\{f(X) \neq Y\} \mid X=x\right]}_{\mathbb{P}(f(x) \neq Y \mid X=x)}\right]$$

Classification

- **Learn** $f : \mathscr{X} \to \mathscr{Y}$
  - $\mathscr{X} \subset \mathbb{R}^d$ **- features**
  - $\mathscr{Y} = \{1, \ldots, k\}$ **- target classes**
- **Loss Function** $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$
- **Expected loss of f:**

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = \sum_i P(Y = i|X = x)\mathbf{1}\{f(x) \neq i\} = \sum_{i \neq f(x)} P(Y = i|X = x)$$

$$= 1 - P(Y = f(x)|X = x)$$

- **Suppose you knew P(YIX) exactly, how should you classify?**

Classification

- Learn $f : \mathscr{X} \rightarrow \mathscr{Y}$
  - $\mathscr{X} \subset \mathbb{R}^d$ - features
  - $\mathscr{Y} = \{1, \ldots, k\}$ - target classes

- Loss Function $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$
- Expected loss of f:

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = \sum_i P(Y = i|X = x)\mathbf{1}\{f(x) \neq i\} = \sum_{i \neq f(x)} P(Y = i|X = x)$$

$$= 1 - P(Y = f(x)|X = x)$$

- Suppose you knew P(YlX) exactly, how should you classify?
- Bayes-Optimal classifier:

$$f(x) = \arg\max_y \mathbb{P}(Y = y|X = x)$$

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- Suppose we don't know $P(Y = y\,|\,X = x)$, but have n iid examples

$$\{(x_i, y_i)\}_{i=1}^n \qquad Y \in \{0, 1\}$$

- Suppose $\mathcal{X}$ is discrete so that $X \in \{1,2,\ldots,m\}$. What is a natural estimator for $\underline{P(Y = y\,|\,X = x)}$?

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- **Suppose we don't know $P(Y = y \mid X = x)$, but have n iid examples**

$$\{(x_i, y_i)\}_{i=1}^n \qquad Y \in \{0, 1\}$$

- **Suppose $\mathcal{X}$ is discrete so that $X \in \{1,2,\ldots,m\}$. What is a natural estimator for $P(Y = y \mid X = x)$?**

$$\hat{f}(x) = \arg\max_{y \in \{0,1\}} \frac{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x, y_i = y}]}{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i = x}]}$$

What if $\mathcal{X}$ is continuous? That is, what if $X \in \mathbb{R}^d$?

Bayes Optimal Binary Classifier

- **Bayes-Optimal classifier:** $f(x) = \arg\max_y \mathbb{P}(Y = y | X = x)$
- Suppose we don't know $P(Y = y \,|\, X = x)$, but have n iid examples

$$\{(x_i, y_i)\}_{i=1}^n \qquad Y \in \{0, 1\}$$

- Suppose $\mathscr{X}$ is discrete so that $X \in \{1,2,...,m\}$. What is a natural estimator for $P(Y = y \,|\, X = x)$?

$$\hat{f}(x) = \arg\max_{y \in \{0,1\}} \frac{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i} = \mathbf{x}, \mathbf{y_i} = \mathbf{y}]}{\sum_{i=1}^n \mathbf{1}[\mathbf{x_i} = \mathbf{x}]}$$

What if $\mathscr{X}$ is continuous? That is, what if $X \in \mathbb{R}^d$?

We need a model to explain observations

Logistic Regression

Recall linear regression:

- We assumed that for any x, we have $p(Y = y \,|\, X = x) = \frac{1}{\sqrt{2\pi}} e^{(y - w^T x)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^n$ we then computed the MLE for w.

Logistic Regression

Recall linear regression:

- We assumed that for any x, we have $p(Y=y\,|\,X=x) = \frac{1}{\sqrt{2\pi}}e^{(y-w^Tx)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^{n}$ we then computed the MLE for w.

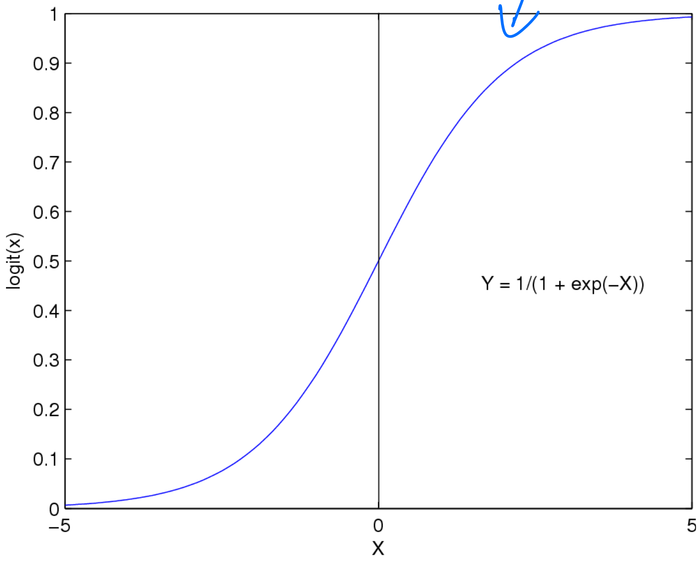
$\sigma(z) = \frac{1}{1+e^{-z}}$

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y=1|X=x,w] = \sigma(w^Tx) = \frac{1}{1+\exp(-w^Tx)}$$

$$\mathbb{P}[Y=0|X=x,w] = 1-\sigma(w^Tx) = \frac{\exp(-w^Tx)}{1+\exp(-w^Tx)}$$

$$= \frac{1}{1+\exp(w^Tx)}$$



Logistic Regression

Recall linear regression:

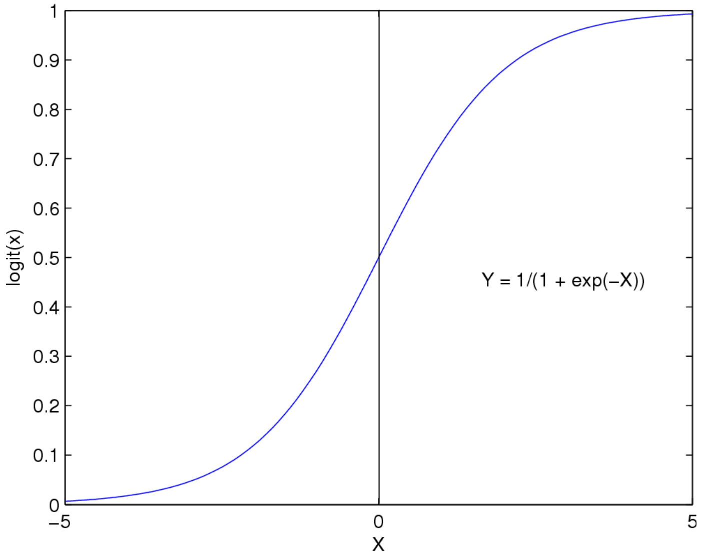
- We assumed that for any x, we have $p(Y = y \,|\, X = x) = \frac{1}{\sqrt{2\pi}} e^{(y - w^T x)^2/2}$.
- Given data $\{(x_i, y_i)\}_{i=1}^n$ we then computed the MLE for w.

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y = 1 | X = x, w] = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

$$\mathbb{P}[Y = 0 | X = x, w] = 1 - \sigma(w^T x) = \frac{\exp(-w^T x)}{1 + \exp(-w^T x)}$$

$$= \frac{1}{1 + \exp(w^T x)}$$

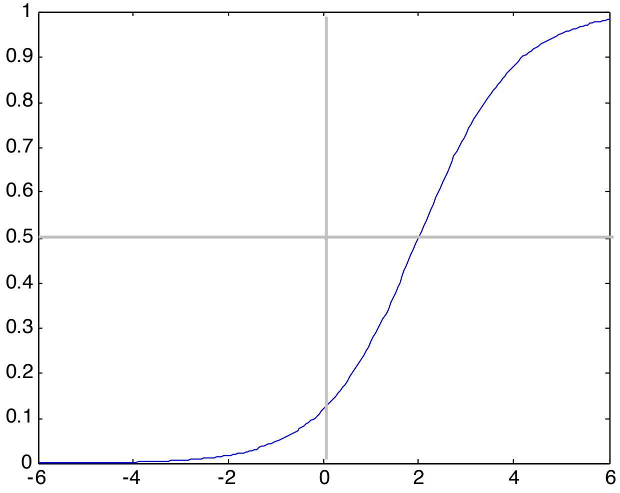


Features can be discrete or continuous!

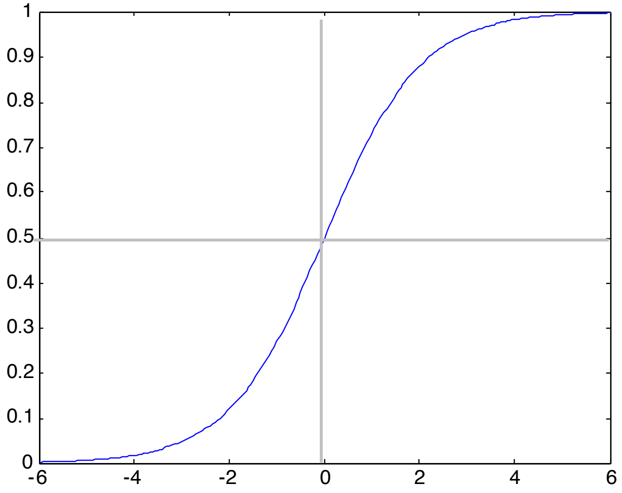
Understanding the sigmoid

$$\sigma(w_0 + \sum_k w_k x_k) = \frac{1}{1 + e^{w_0 + \sum_k w_k x_k}}$$

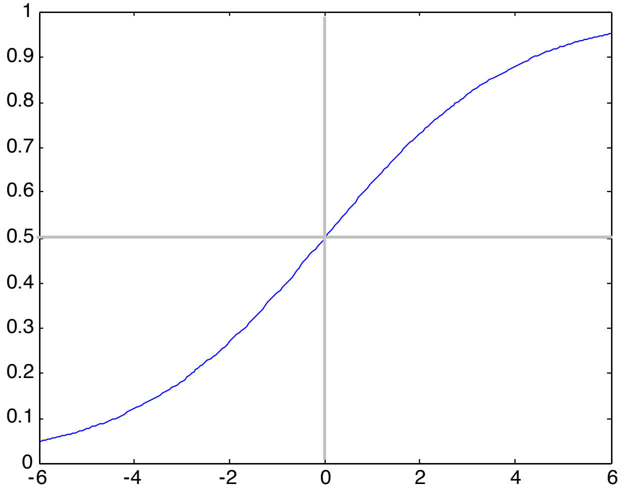
w_0=-2, w_1=-1



w_0=0, w_1=-1



w_0=0, w_1=-0.5



Sigmoid for binary classes

$$\mathbb{P}(Y=0|w,X) = \frac{1}{1+\exp(w_0+\sum_k w_k X_k)}$$

$$\mathbb{P}(Y=1|w,X) = 1-\mathbb{P}(Y=0|w,X) = \frac{\exp(w_0+\sum_k w_k X_k)}{1+\exp(w_0+\sum_k w_k X_k)}$$

$$\frac{\mathbb{P}(Y=1|w,X)}{\mathbb{P}(Y=0|w,X)} = \exp\left(w_0 + \sum_n w_n X_n\right) = \exp(w_0 + w^T X)$$

Sigmoid for binary classes

$$\mathbb{P}(Y = 0|w, X) = \frac{1}{1 + \exp(w_0 + \sum_k w_k X_k)}$$

$$\mathbb{P}(Y = 1|w, X) = 1 - \mathbb{P}(Y = 0|w, X) = \frac{\exp(w_0 + \sum_k w_k X_k)}{1 + \exp(w_0 + \sum_k w_k X_k)}$$

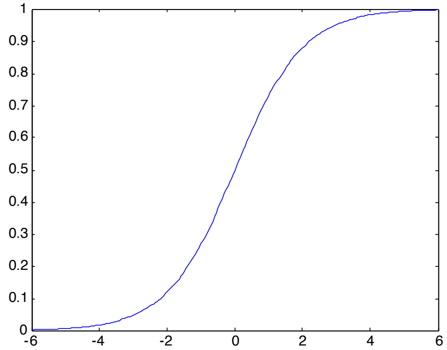
$$\frac{\mathbb{P}(Y = 1|w, X)}{\mathbb{P}(Y = 0|w, X)} = \exp(w_0 + \sum_k w_k X_k)$$

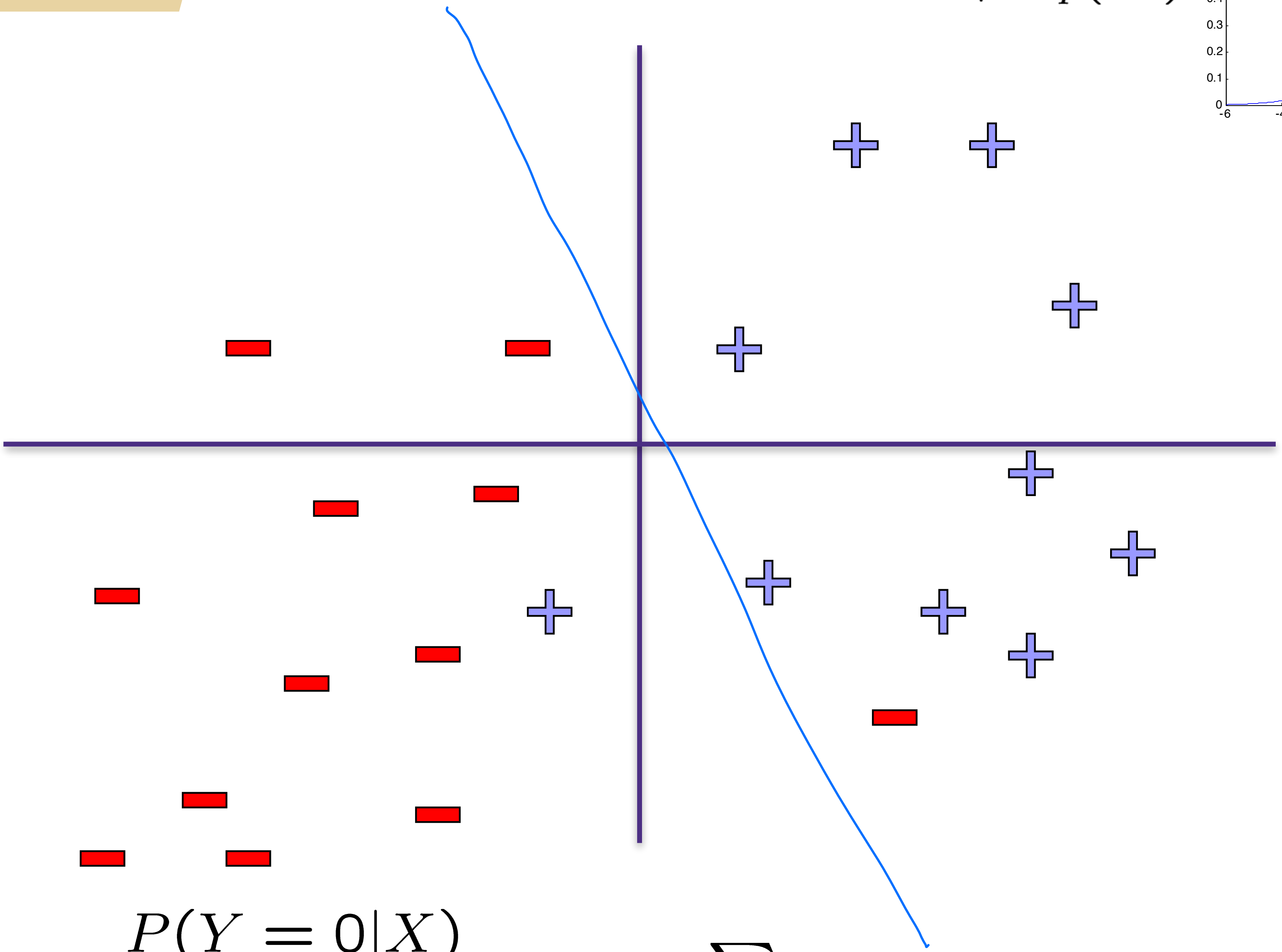
Linear Decision Rule!

$$\log \frac{\mathbb{P}(Y = 1|w, X)}{\mathbb{P}(Y = 0|w, X)} = w_0 + \sum_k w_k X_k$$

Logistic Regression – a Linear classifier

$$\frac{1}{1+exp(-z)}$$





$$\ln \frac{P(Y=0|X)}{P(Y=1|X)} = w_0 + \sum_i w_i X_i$$

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = -1|x, w) = \frac{1}{1 + \exp(w^T x)}$$

$$P(Y = 1|x, w) = \frac{\exp(w^T x)}{1 + \exp(w^T x)}$$

- **This is equivalent to:**

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

- **So we can compute the maximum likelihood estimator:**

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w)$$

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w)$$

$$= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w))$$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

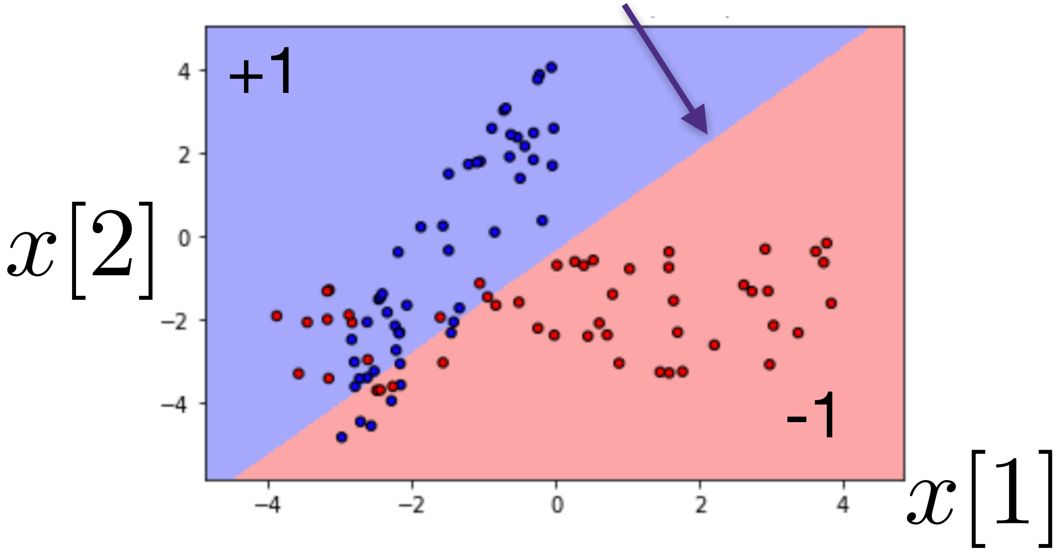
(MLE for Gaussian noise)

Logistic regression for binary classification

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

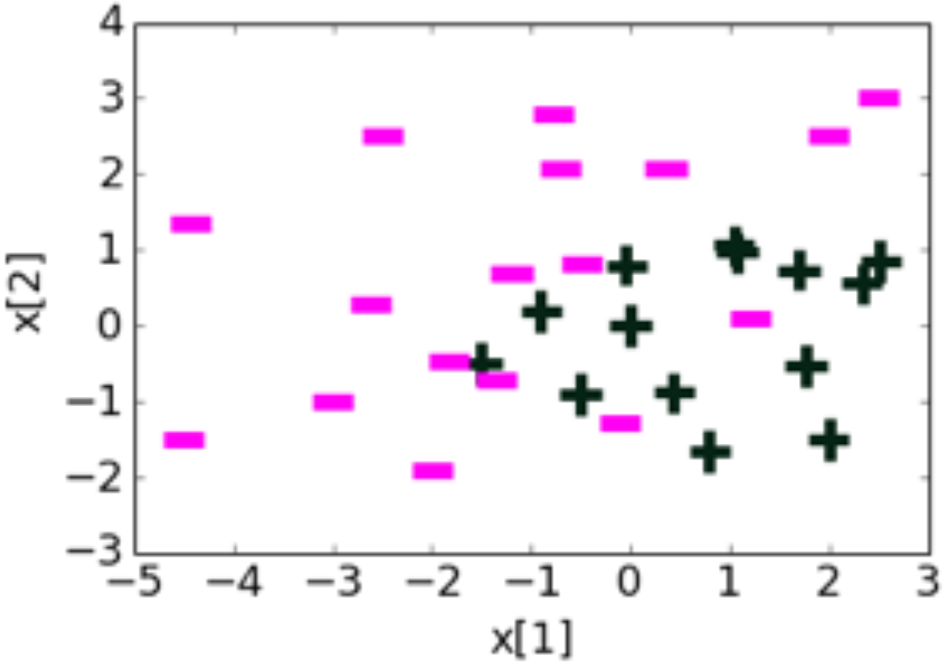
$$(\widehat{b}, \widehat{w}) = \arg\min_{b,w} \sum_{i=1}^{n} \log(1 + e^{-y_i(b + x_i^T w)})$$

- 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

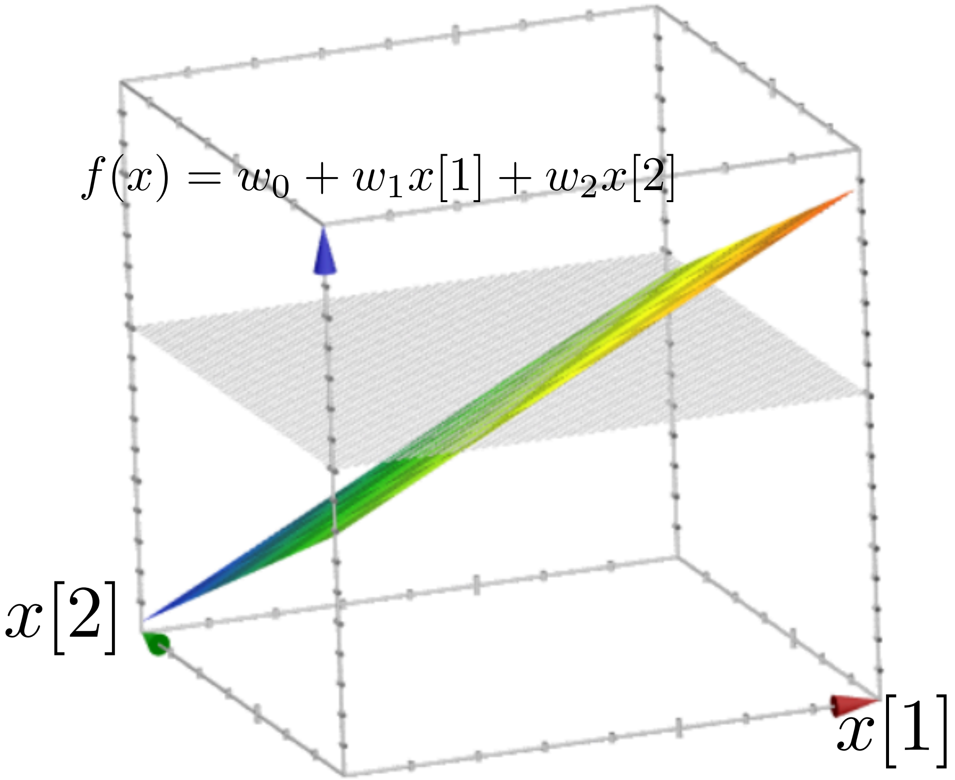


Polynomial features

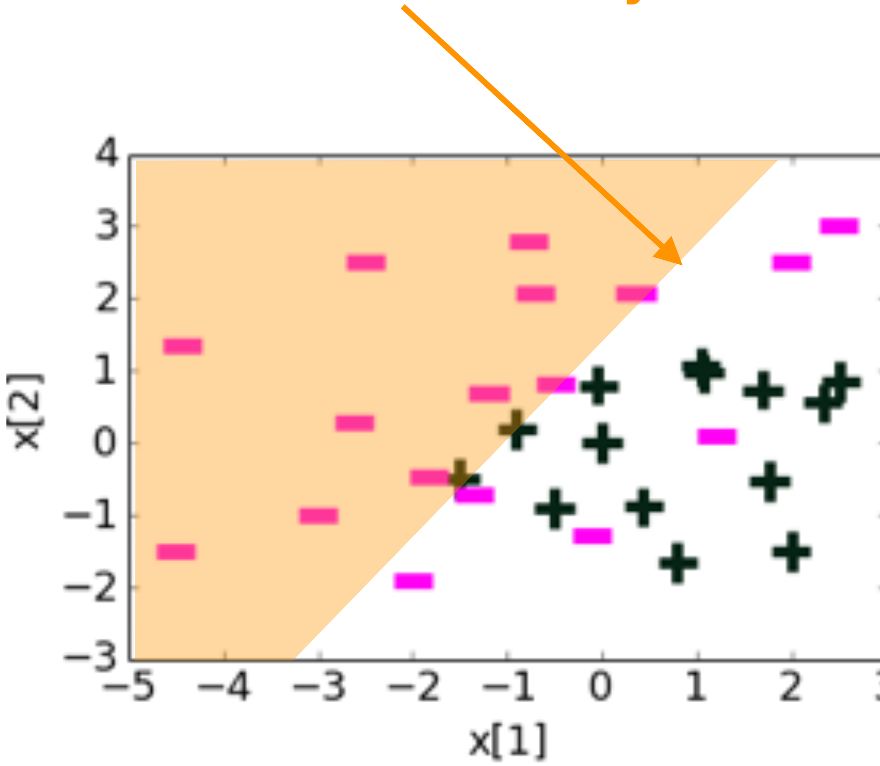
$$\begin{bmatrix} h_0(x) = 1 \\ h_1(x) = x[1] \\ h_2(x) = x[2] \\ h_3(x) = x[1]^2 \\ h_4(x) = x[2]^2 \\ \vdots \end{bmatrix}$$

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

Learned decision boundary



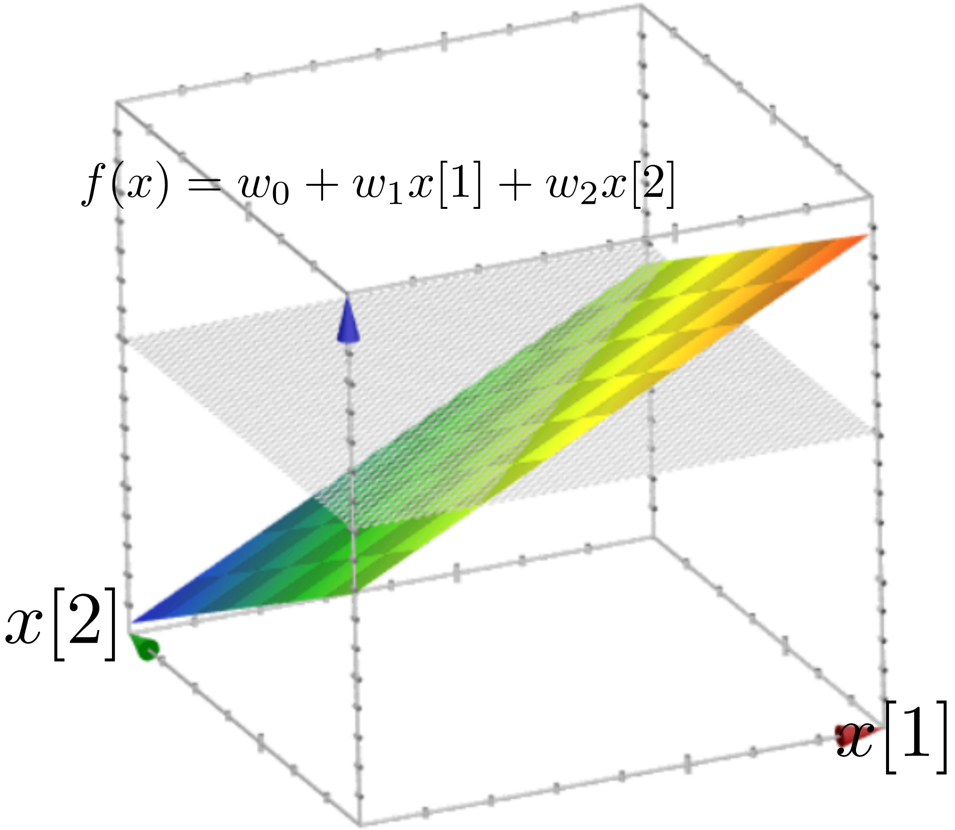
decision boundary

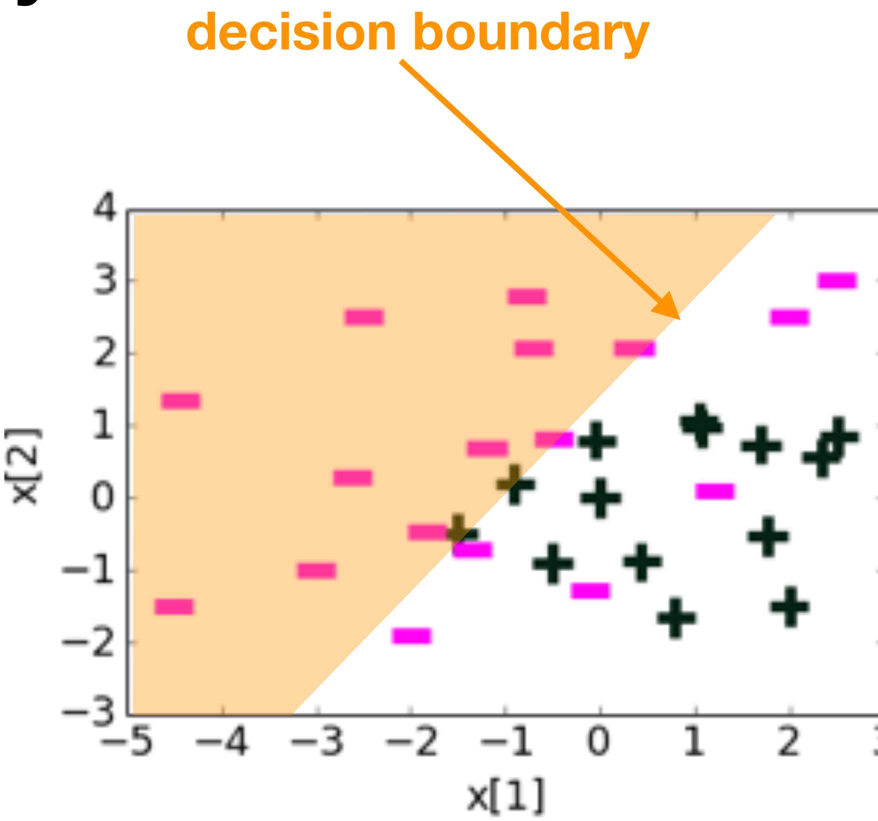


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

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

Learned decision boundary

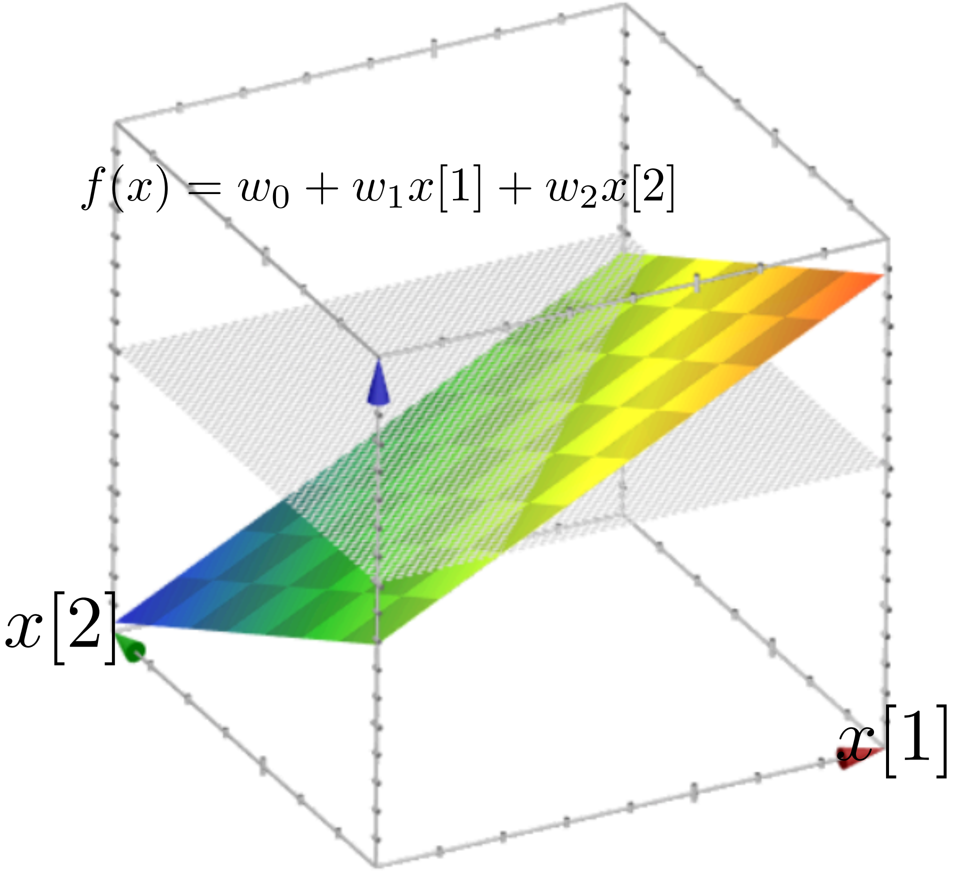


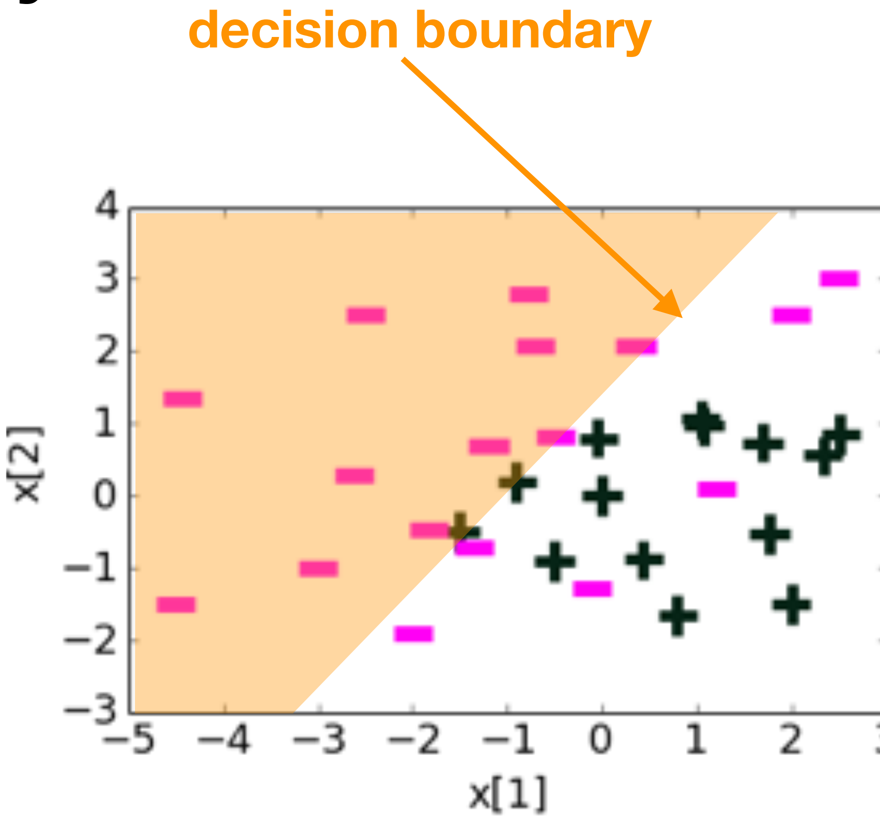

decision boundary

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

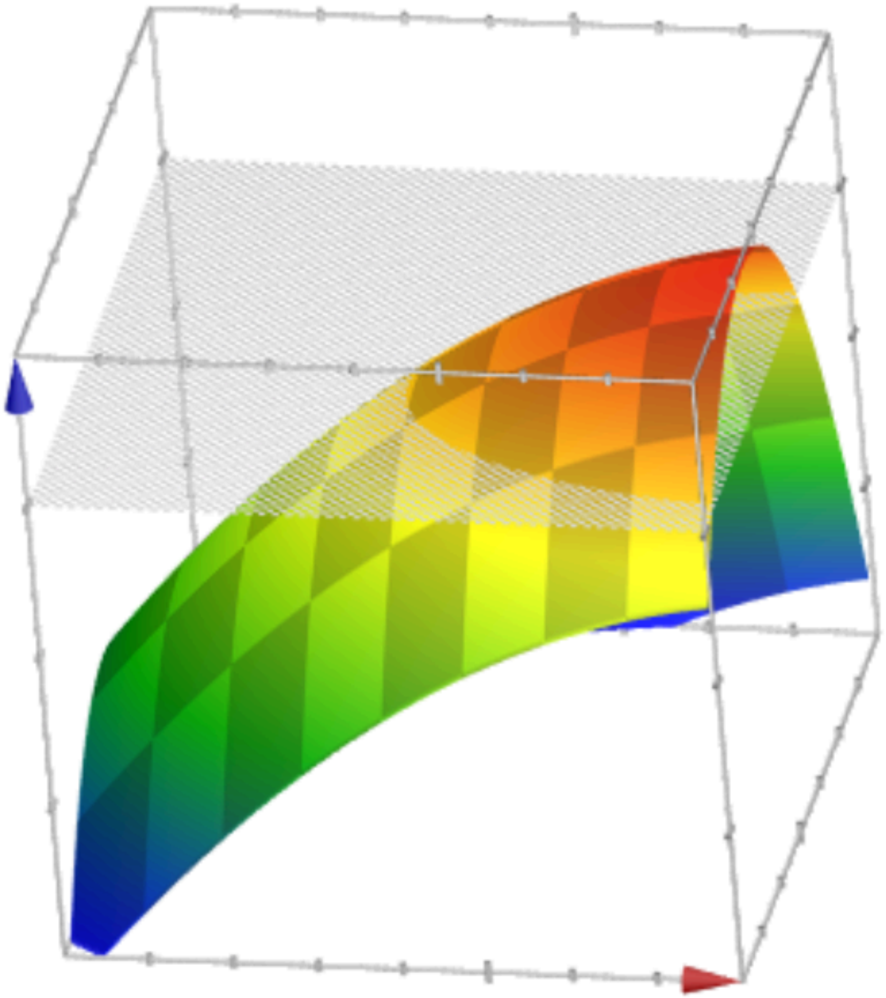


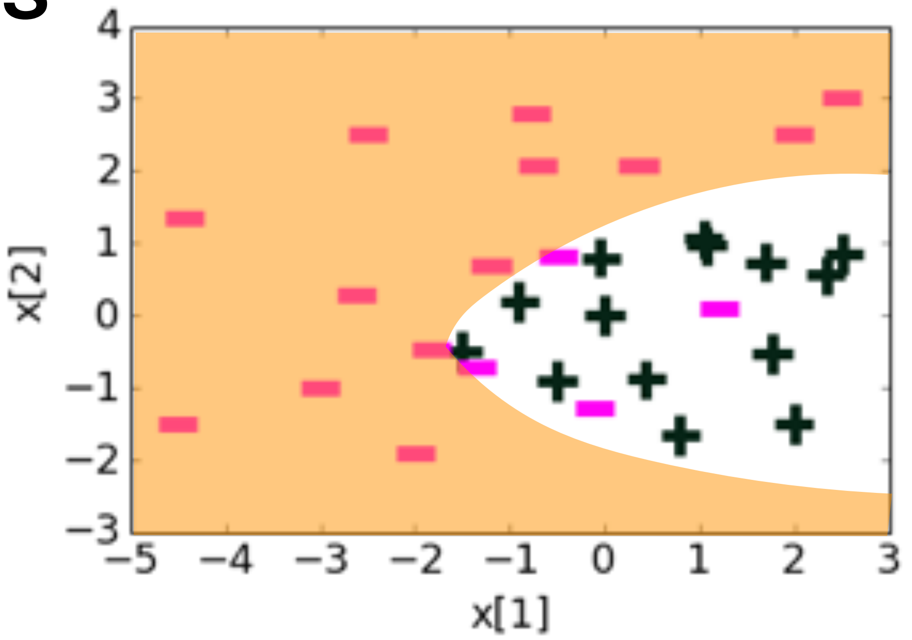

decision boundary

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

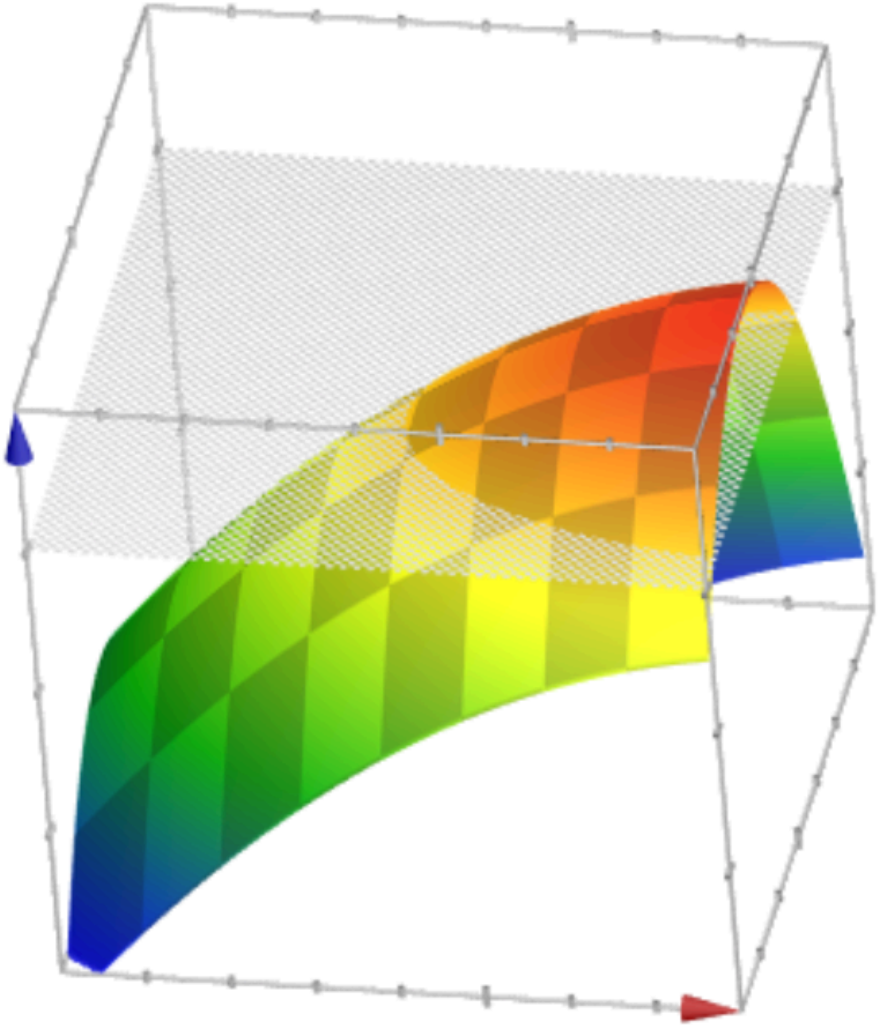


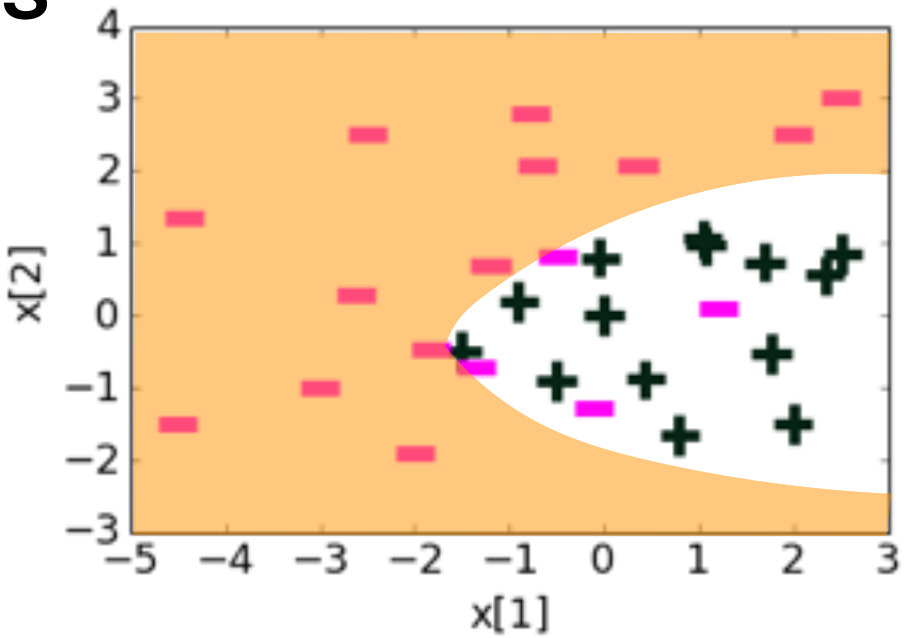


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

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

Adding quadratic features

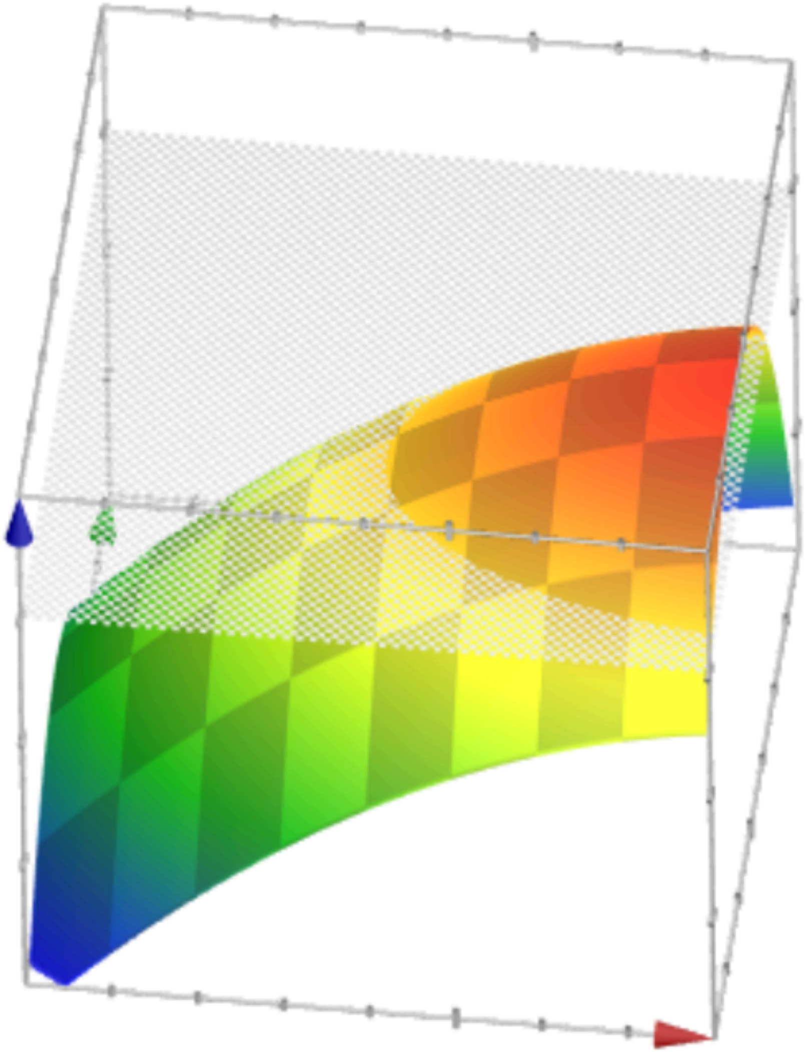


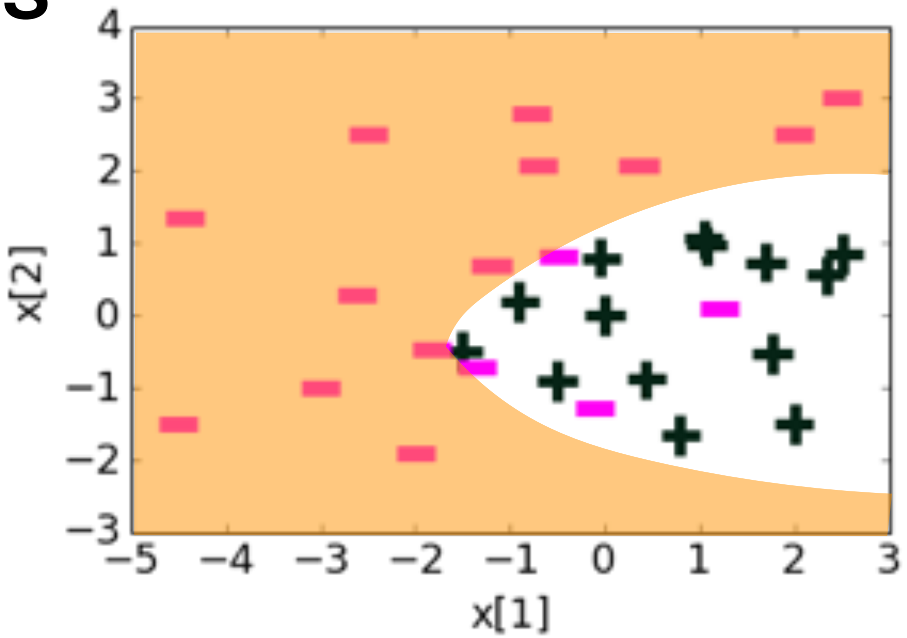


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

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

Adding quadratic features

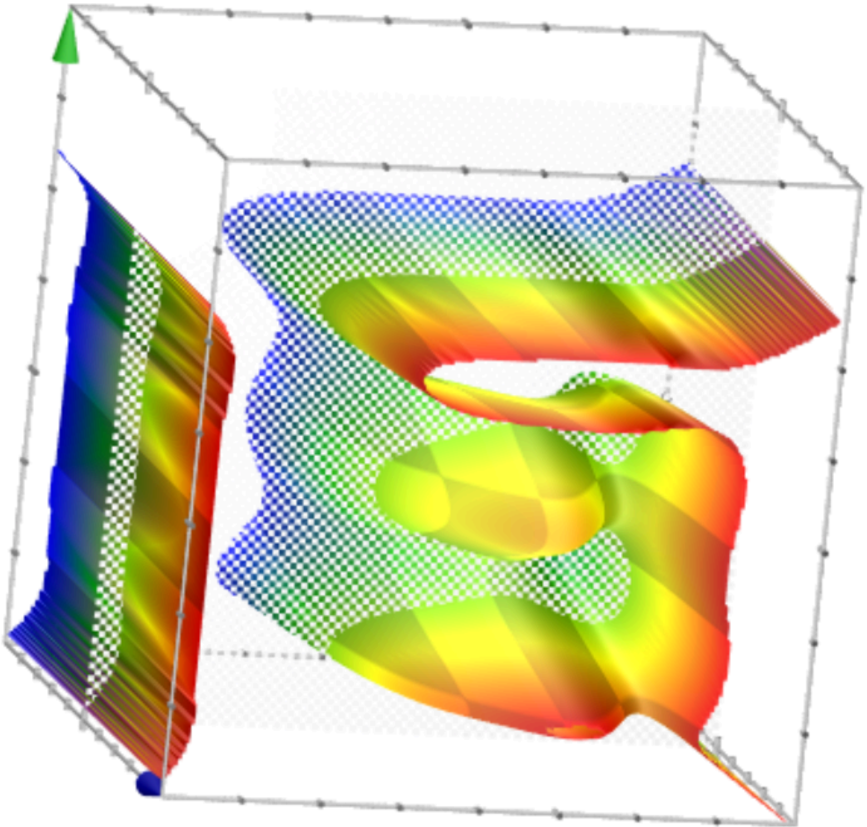




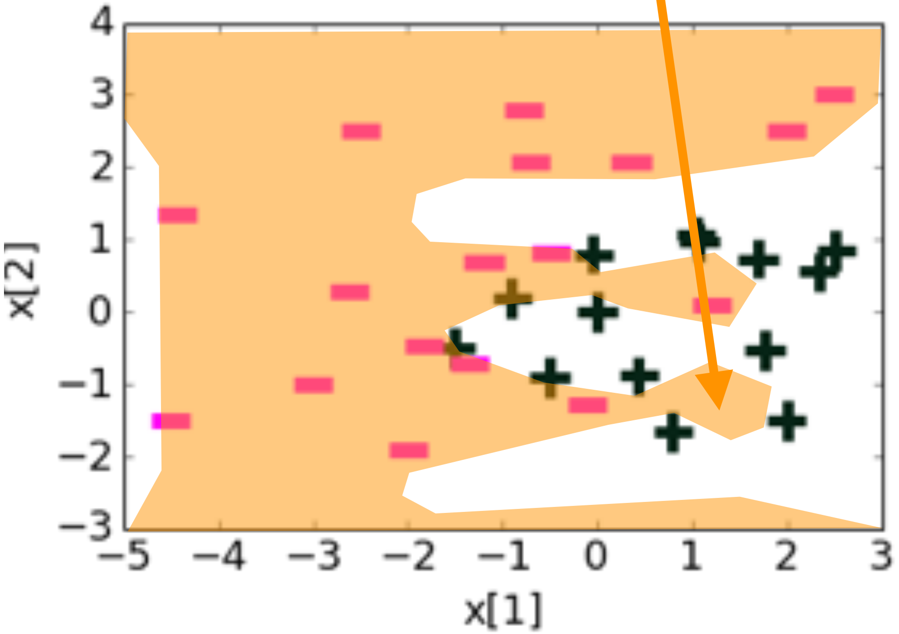
Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

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

Adding higher degree polynomial features



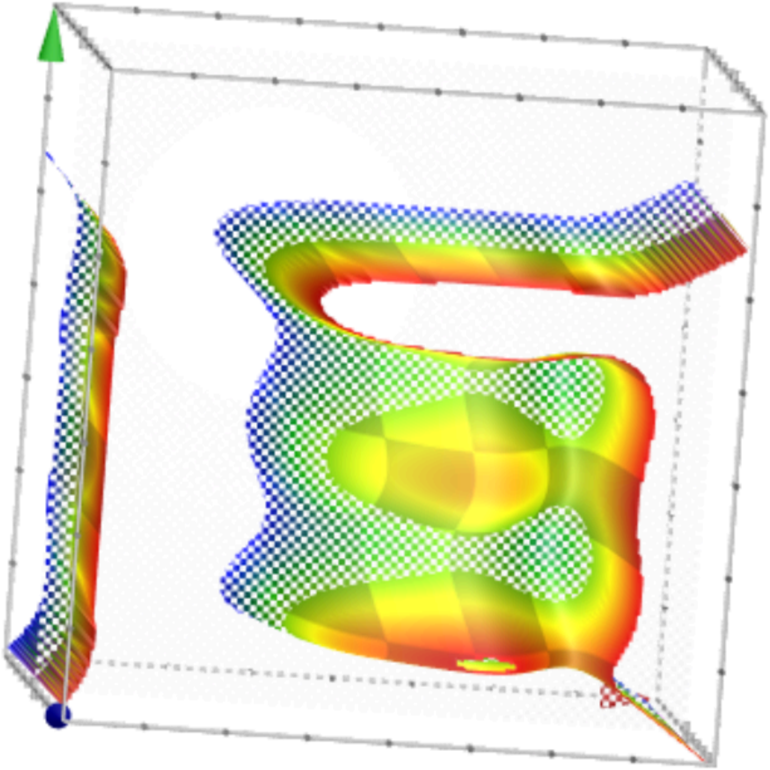
Overfitting leads to non-generalization



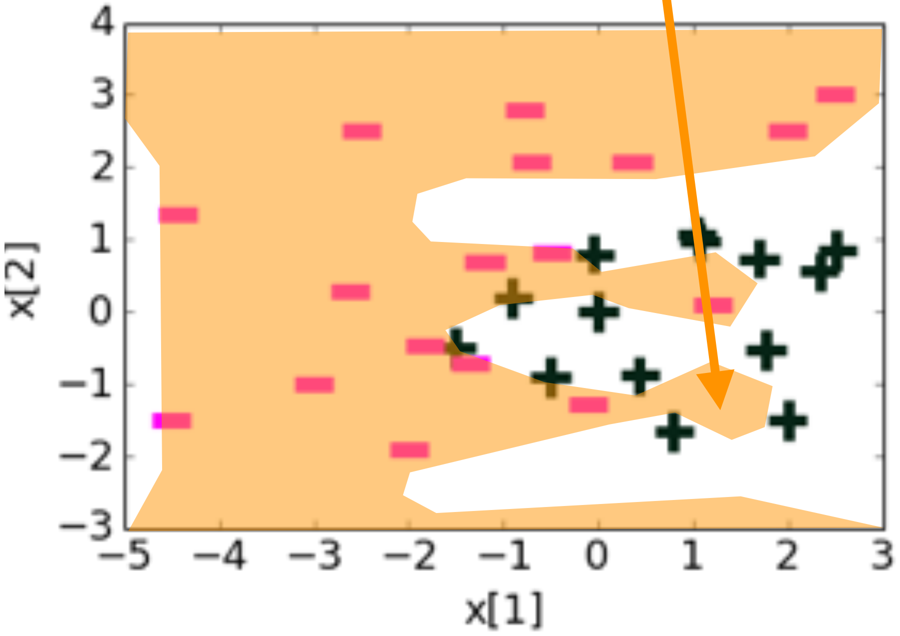
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$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

Coefficient values getting large

Adding higher degree polynomial features



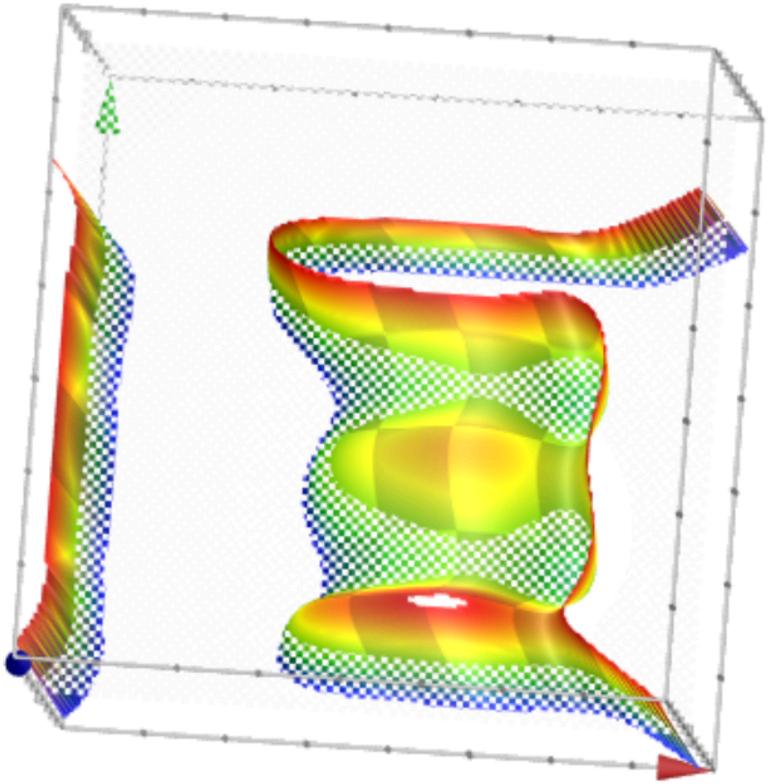
Overfitting leads to non-generalization



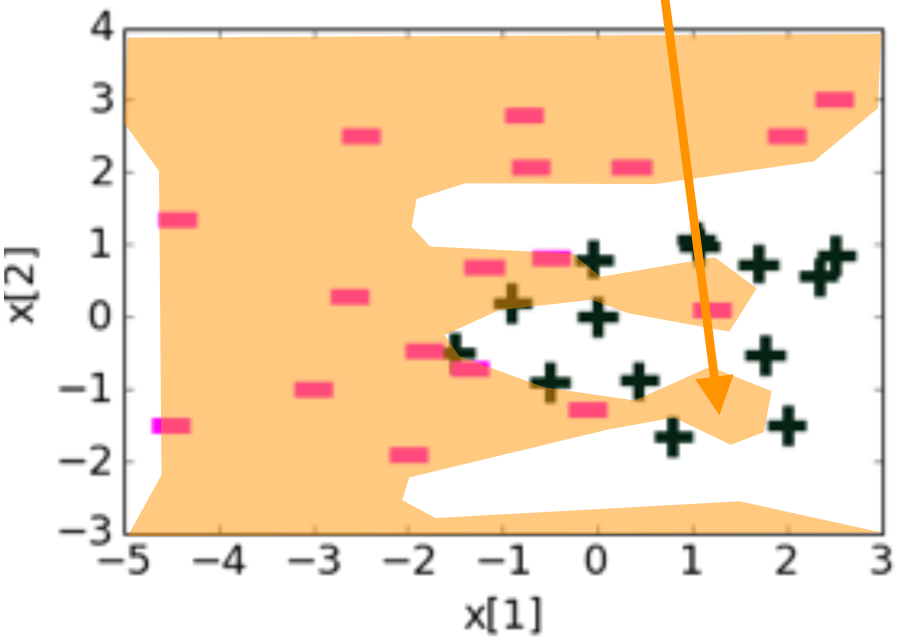
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$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

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

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$$\begin{aligned} \widehat{w}_{MLE} &= \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \\ &= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w)) = J(w) \end{aligned}$$

What does $J(w)$ look like? Is it convex?

Loss function: Conditional Likelihood

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$$\begin{aligned}\widehat{w}_{MLE} &= \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \\ &= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w)) = J(w)\end{aligned}$$

Good news: *J*(**w**) is convex function of **w**, no local optima problems

Bad news: no closed-form solution to maximize *J*(**w**)

Good news: convex functions easy to optimize

One other concern… overfitting.

- **Have a bunch of iid data:** $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

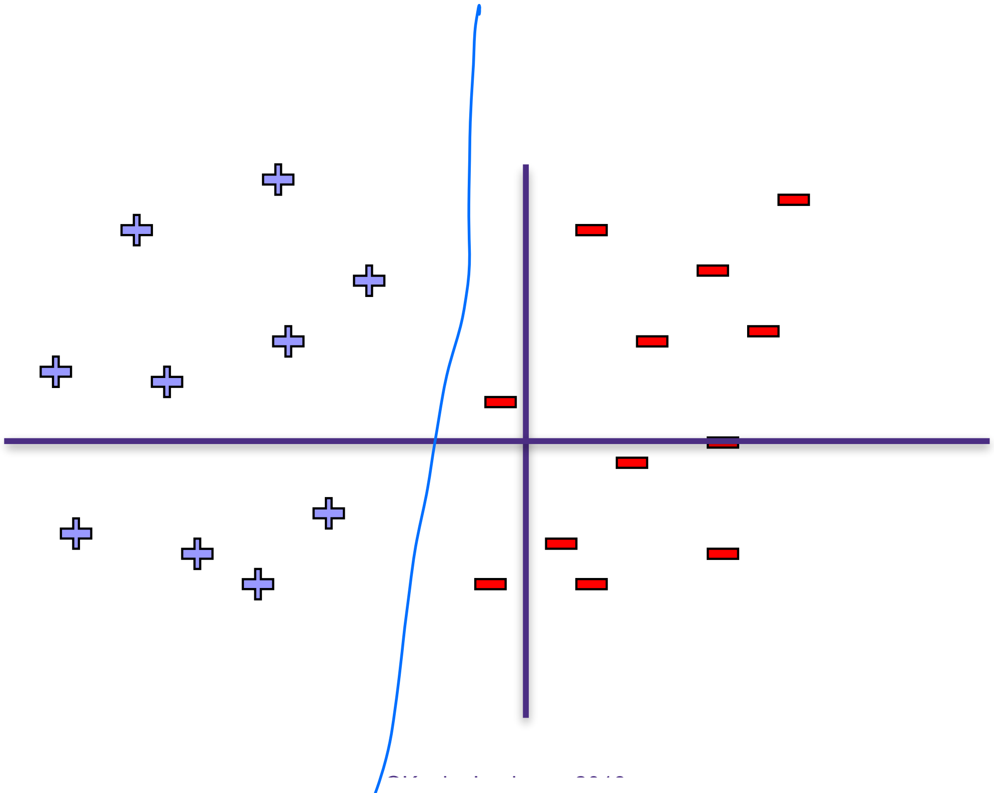
$$\begin{aligned}\widehat{w}_{MLE} &= \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \\ &= \arg\min_w \sum_{i=1}^n \log(1 + \exp(-y_i\, x_i^T w))\end{aligned}$$

Does anyone see a situation when this minimization might overfit?

Overfitting and Linear Separability

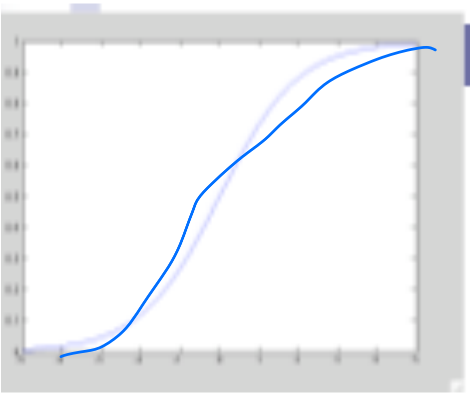
$$\arg\min_{w} \sum_{i=1}^{n} \log(1 + \exp(-y_i \, x_i^T \, w))$$

When is this loss small?

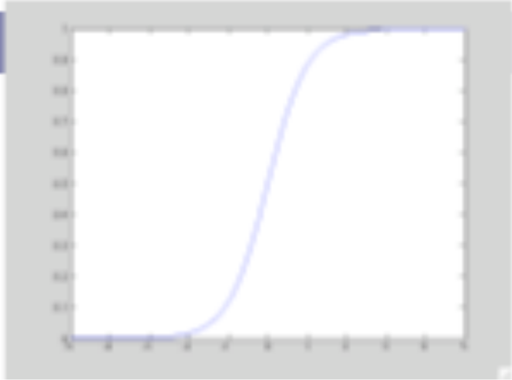


Large parameters → Overfitting

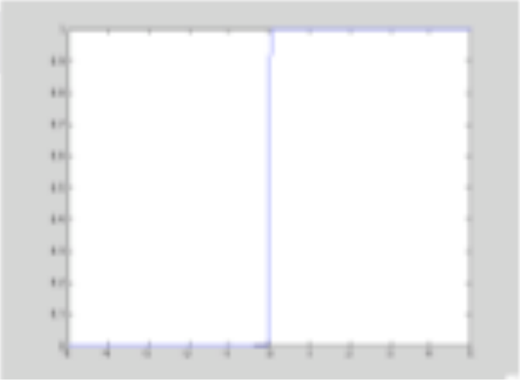
When data is linearly separable, weights $\Rightarrow \infty$



$$\frac{1}{1+e^{-x}}$$



$$\frac{1}{1+e^{-2x}}$$



$$\frac{1}{1+e^{-100x}}$$

Overfitting

Penalize high weights to prevent overfitting?

Regularized Conditional Log Likelihood

Add a penalty to avoid high weights/overfitting?:

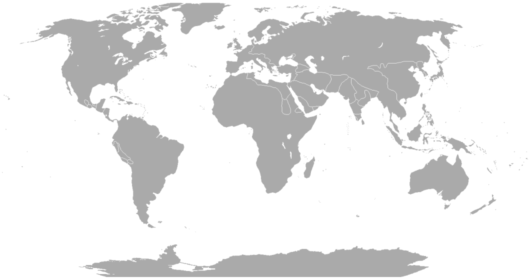
$$\arg\min_{w,b} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i \left(x_i^T w + b\right))\right) + \lambda ||w||_2^2$$

Be sure to not regularize the offset b!

How do we encode categorical data $\boldsymbol{y}$?

- so far, we considered Binary case where there are two categories
- encoding y is simple: {+1,-1}

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



Country of birth

(Argentina, Brazil, USA,...)

Zipcode

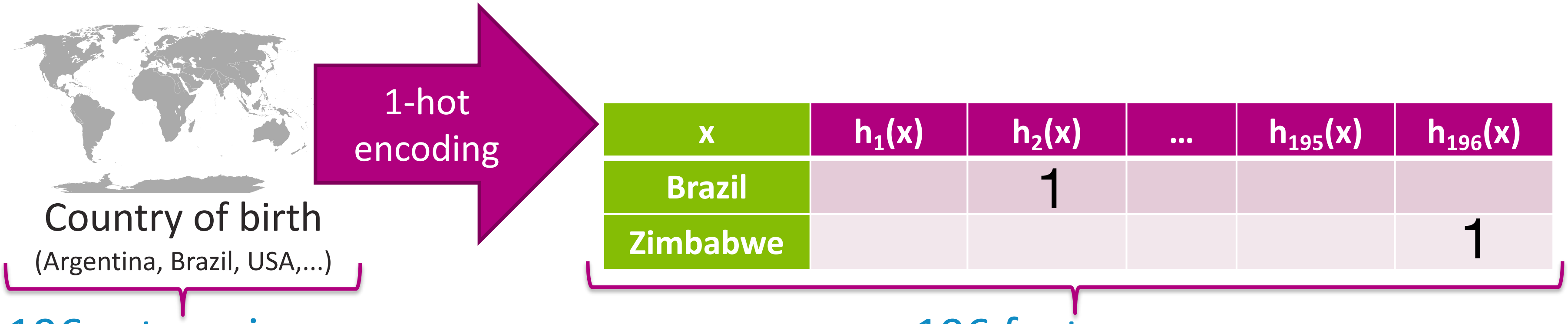
(10005, 98195,...)

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**)
  - each class is a standard basis vector in $k-$dimension



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

196 categories

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 $\hat{y} \in \mathbb{R}^k$

$$\hat{y}_i \;=\; f(x_i) \;=\; w^T x_i \quad \in \mathbb{R}^k$$

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

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

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

k classes

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

$$\vdots$$

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

Without loss of generality setting w[:,1]=0 when $k = 2$ recovers the original binary class case

Maximum Likelihood Estimator

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

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

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

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

Kernels

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors $\{u_j\}_{j=1}^{p} \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?

How do we deal with high-dimensional lifts/data?

The kernel trick:

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi : \mathbb{R}^d \to \mathbb{R}^p$ if $K(x, x') = \langle \phi(x), \phi(x') \rangle$ for all x, x'

Big idea: if we can represent our

- training algorithms and
- decision rules for prediction

as functions of dot products of feature maps (i.e. $\{\langle \phi(x), \phi(x') \rangle\}$) and we can find a kernel for our feature map such that

$$K(x \,.\, x') = \langle \phi(x), \phi(x') \rangle$$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^n$ and instead only work with the kernel matrix of the training data $\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

Recap: Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$

Recap: Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$

- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn
  - The features are **implicit** and accessed only via kernels, making it efficient

Examples of popular Kernels

- Polynomials of degree exactly$k$

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

Examples of popular Kernels

- Polynomials of degree exactly k

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

- Gaussian (squared exponential) kernel
  (a.k.a RBF kernel for Radial Basis Function)

$$K(x, x') = \exp\left(- \frac{\|x - x'\|_2^2}{2\sigma^2} \right)$$

Examples of popular Kernels

- Polynomials of degree exactly k

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

- Gaussian (squared exponential) kernel
  (a.k.a RBF kernel for Radial Basis Function)

$$K(x, x') = \exp\Big(- \frac{\|x - x'\|_2^2}{2\sigma^2} \Big)$$

- Sigmoid

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

- All these kernels are efficient to compute, but the corresponding features are in high-dimensions

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$ = $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$ = $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$ $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$

$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$ = $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$
$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

- Hence, to make prediction on any future data points, all we need to know is
  - $\mathbf{X}x_{\text{new}} = \begin{bmatrix} x_1^T x_{\text{new}} \\ \vdots \\ x_n^T x_{\text{new}} \end{bmatrix} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, and $\mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$
- **Key idea**: Now consider $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - w^T \phi(x_i)\right)^2 + \lambda \|w\|_2^2$ and use an *any* kernel $K(x, x') = \phi(x)^T \phi(x')$!

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\widehat{w} = \sum_{i=1}^n \alpha_i x_i} \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

   Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
3. Substitute $x_i^T x_j$ with $K(x_i, x_j)$, and find α using the above algorithm from step 2.
4. Make prediction with $\widehat{y}_{\text{new}} = \sum_{i=1}^n \alpha_i K(x_i, x_{\text{new}})$

   (replacing $x_i^T x_{\text{new}}$ with $K(x_i, x_{\text{new}})$)

The Kernel Trick for regularized least squares

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

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. We will prove it later)

The Kernel Trick for regularized least squares

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

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. We will prove it later)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

The Kernel Trick for regularized least squares

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

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ (Step 1. We will prove it later)

$$\widehat{\alpha} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_\alpha ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

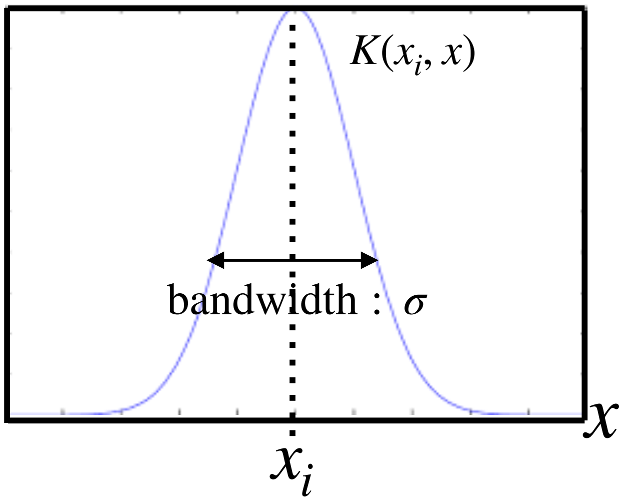
(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{K} + \lambda \mathbf{I}_{n\times n})^{-1} \mathbf{y}$

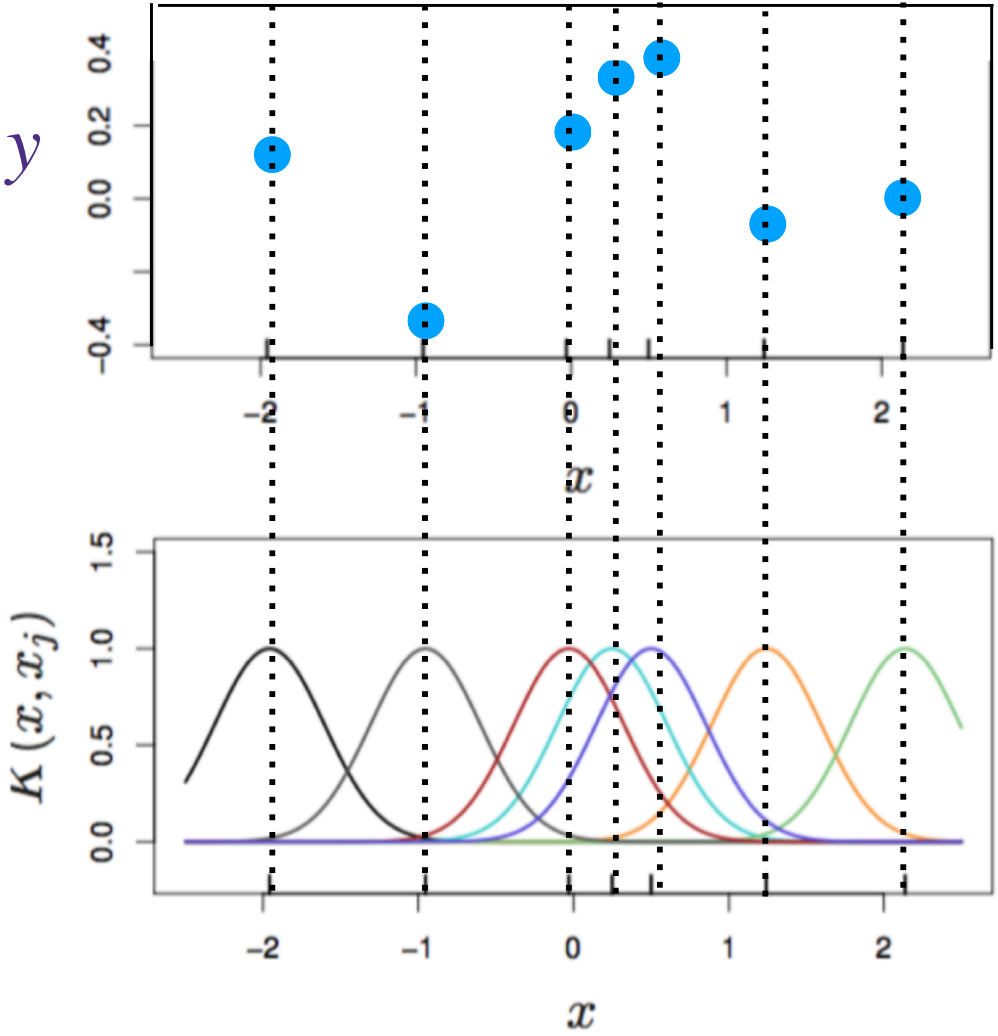
Why do we need regularization when using kernels?

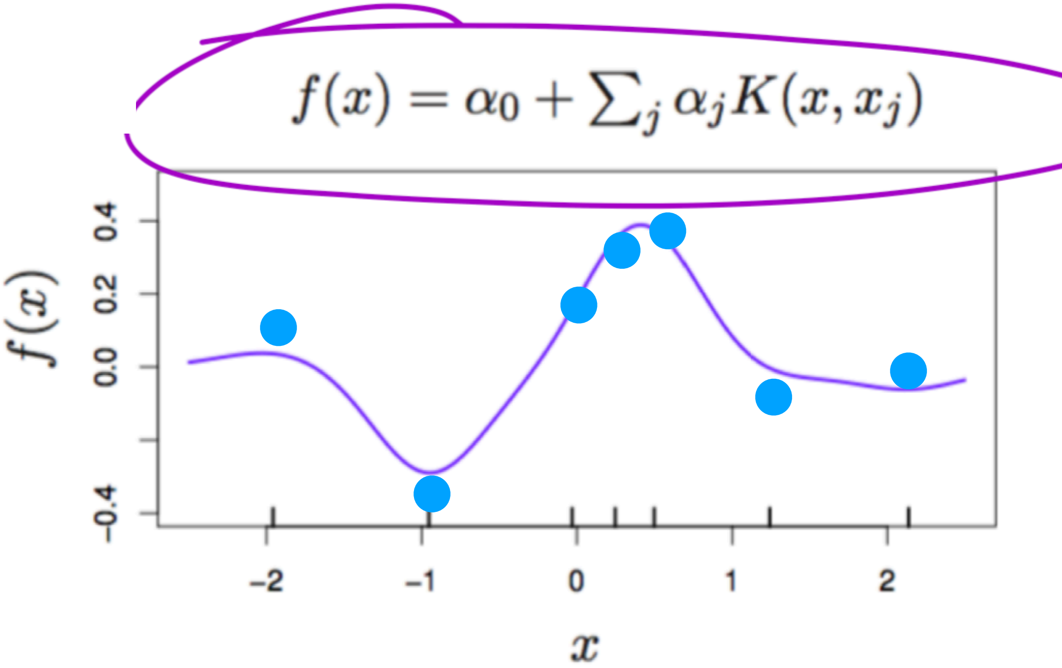
- $\widehat{\alpha} = \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda\alpha^T\mathbf{K}\alpha$
- Typically, $\mathbf{K}$ is invertible so that $\widehat{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_{n\times n})^{-1}\mathbf{y}$ is well defined.
- What if $\lambda = 0$? What goes wrong?

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$



samples $\{(x_i, y_i)\}_{i=1}^n$



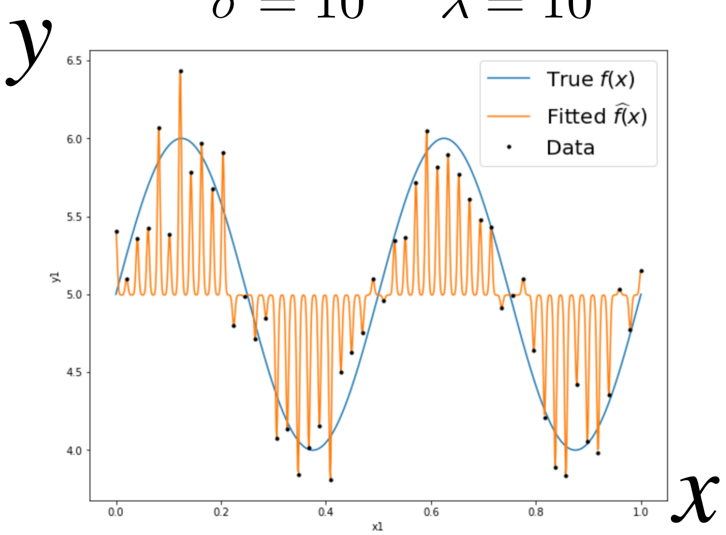

$f(x) = \alpha_0 + \sum_j \alpha_j K(x, x_j)$

- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

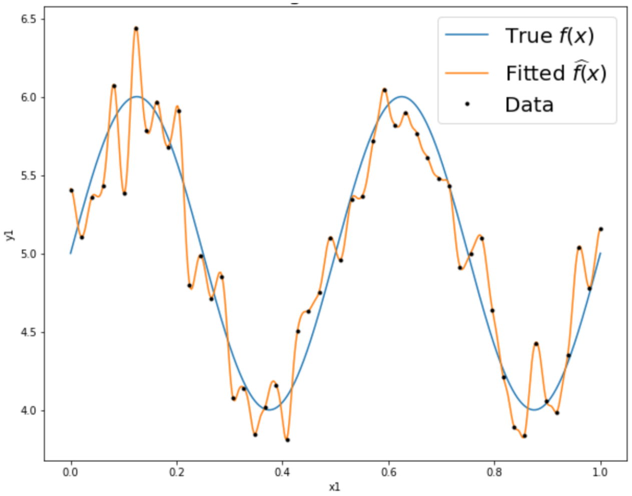
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathcal{L}(\alpha) \; = \; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 \; + \; \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

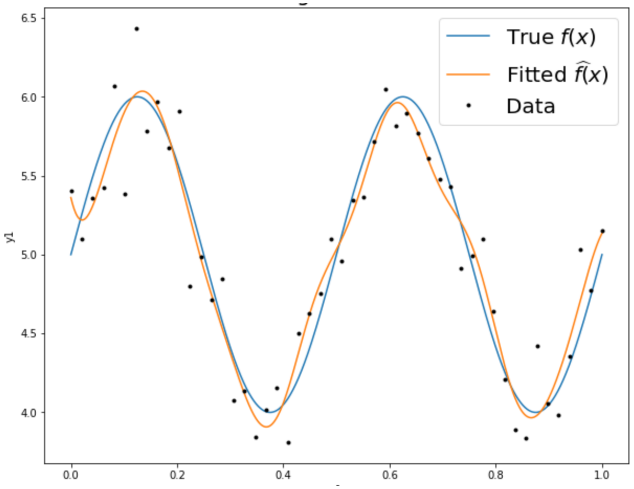
$\sigma = 10^{-3} \;\; \lambda = 10^{-4}$



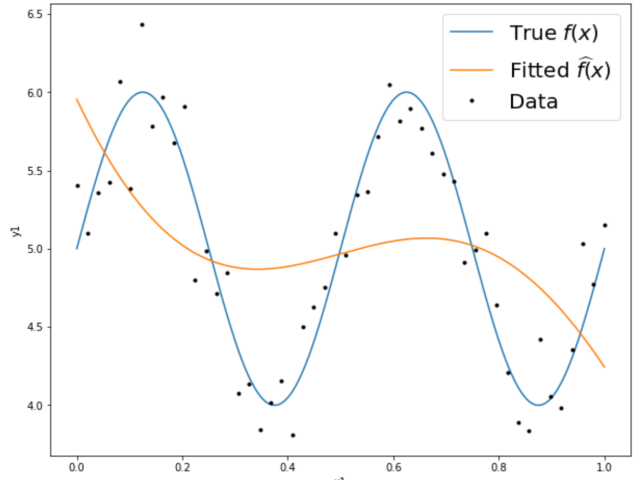
$\sigma = 10^{-2} \;\; \lambda = 10^{-4}$



$\sigma = 10^{-1} \;\; \lambda = 10^{-4}$

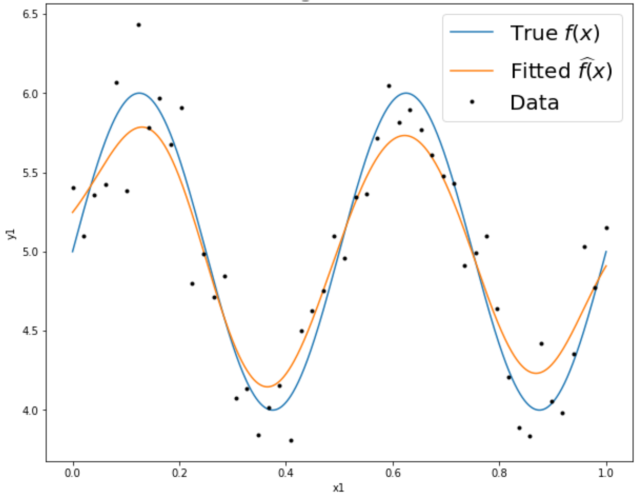


$\sigma = 10^{-0} \;\; \lambda = 10^{-4}$



$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

$\sigma = 10^{-1} \;\; \lambda = 10^{-0}$

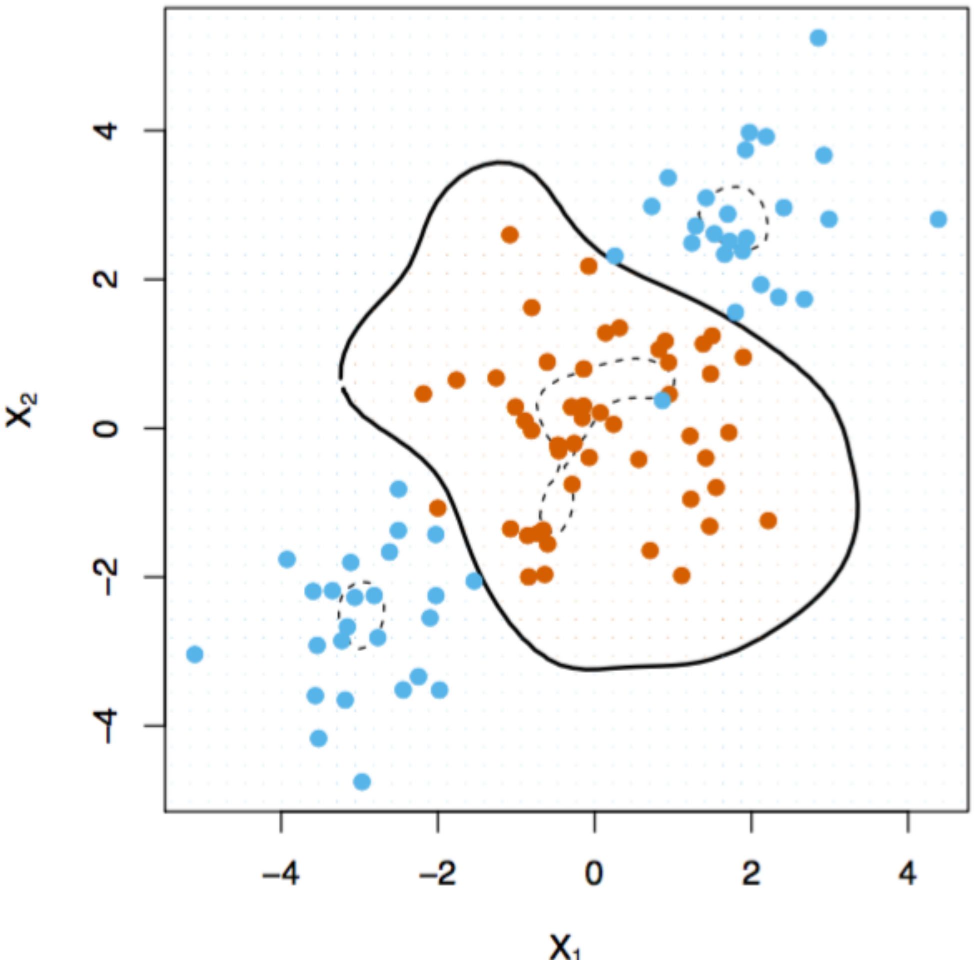


RBF kernel and random features

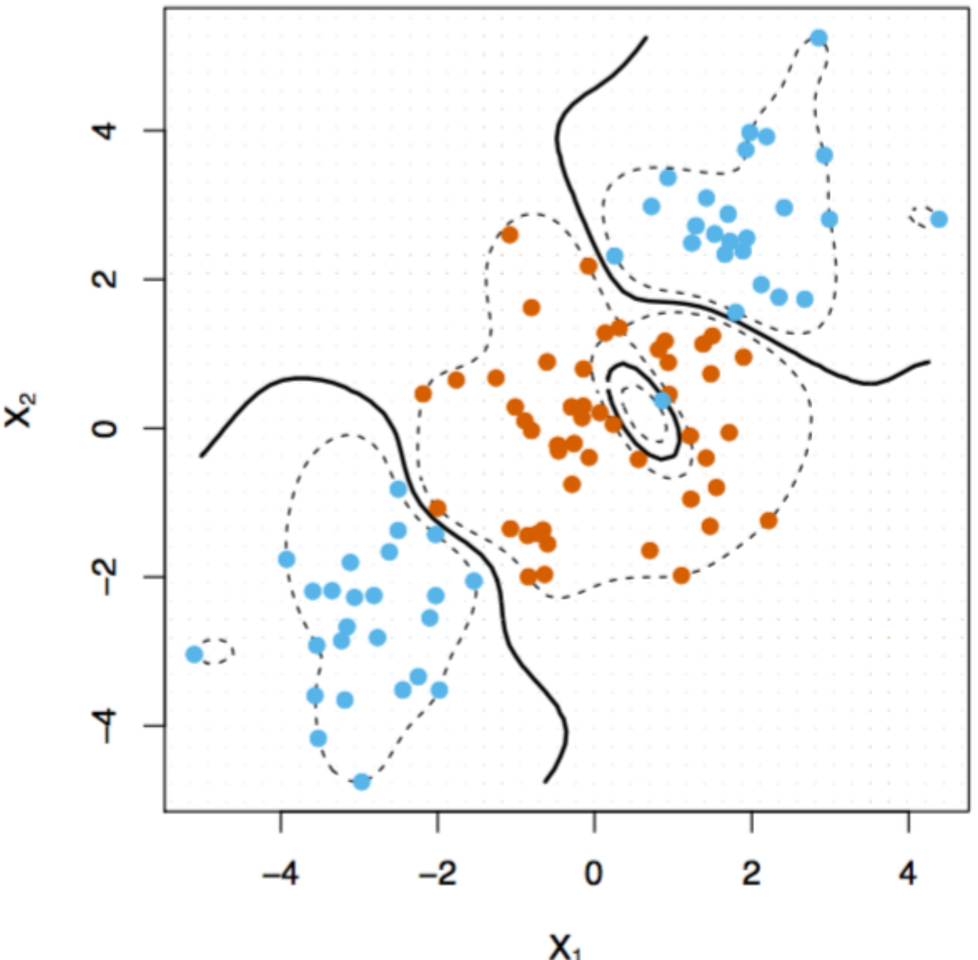
$$\widehat{w} = \arg\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i(b+w^T x_i))) + \lambda\|w\|_2^2$$

$$\widehat{\alpha}, \widehat{b} = \arg \min_{\alpha\in\mathbb{R}^n, b} \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i(b+\sum_{j=1}^{n}\alpha_j K(x_j, x_i)))) + \lambda \sum_{i=1,j=1}^{n} \alpha_i\alpha_j K(x_i, x_j)$$

Bandwidth σ is large enough



Bandwidth σ is small



Features vs. RBF kernel vs. random features

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||_2^2}{2\sigma^2}\right)$$

If n is very large, allocating an n-by-n matrix is tough.

Instead, consider generating random feature maps of the form:

$$\phi(x) = \begin{bmatrix} \sqrt{2}\cos(w_1^T x + b_1) \\ \vdots \\ \sqrt{2}\cos(w_p^T x + b_p) \end{bmatrix} \qquad \begin{aligned} w_k &\sim \mathcal{N}(0, 2\gamma\, I) \\ b_k &\sim \text{uniform}(0, \pi) \end{aligned}$$

with $p \ll n$

One can show that

$$\mathbb{E}_{w,b}\Big[\frac{1}{p}\phi(x)^T\phi(x')\Big] \;=\; \exp(-\gamma\|x - x'\|_2^2)$$

So this choice of random features approximate the desired RBF kernel with $\gamma = \dfrac{1}{2\sigma^2}$

[Rahimi, Recht NIPS 2007]
“NIPS Test of Time Award, 2018”

Fixed Feature V.S. Learned Feature

Can we learn the feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ from data also?

Questions?