Review: Cross-Validation

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 12, 2016

Use *k*-fold cross validation

- Randomly **divide training data into *k* equal parts**
  - $D_1, \ldots, D_k$
- For each i
  - **Learn classifier $f_{D \backslash D_i}$ using data point not in D_i**
  - **Estimate error of $f_{D \backslash D_i}$ on validation set D_i**:

1	2	3	4	5
Train	Train	Validation	Train	Train

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} (y_j - f_{\mathcal{D} \backslash \mathcal{D}_i}(x_j))^2$$

- ***k*-fold cross validation error is average** over data splits:

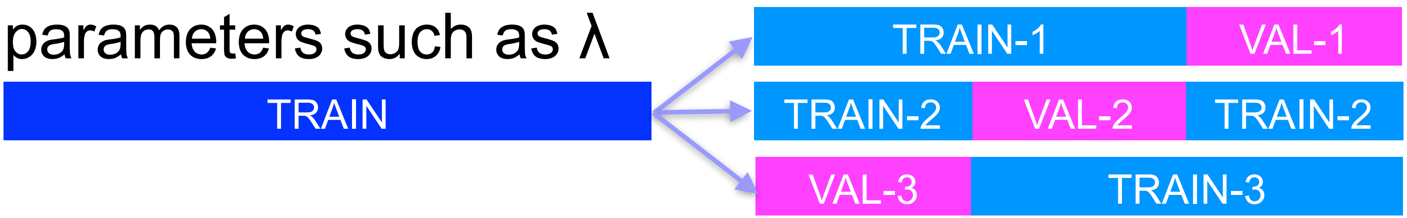
$$error_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- *k*-fold cross validation properties:
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only *n(k-1)/k*
  - Usually, **k = 10**

Recap

- Given a dataset, begin by splitting into

- **Model selection**: Use k-fold cross-validation on TRAIN to train predictor and choose magic parameters such as λ



- **Model assessment**: Use TEST to assess the accuracy of the model you output
  - Never ever ever ever ever train or choose parameters based on the test data

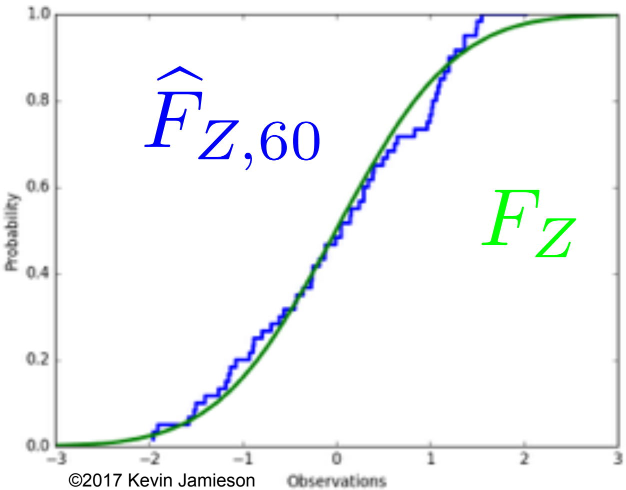
Bootstrap: basic idea

Given dataset drawn iid samples with CDF F_Z:

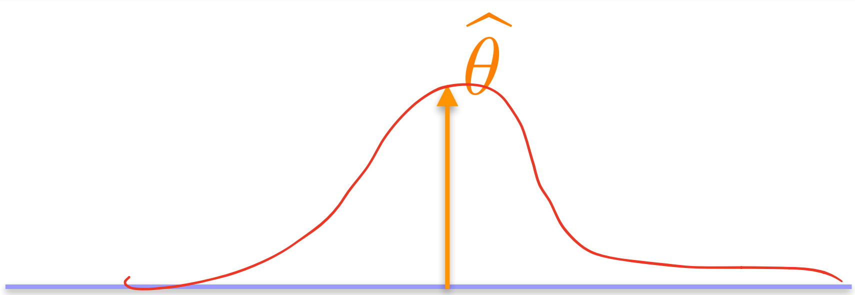
$$\mathcal{D} = \{z_1, \dots, z_n\} \overset{i.i.d.}{\sim} F_Z \qquad \widehat{\theta} = t(\mathcal{D})$$

For b=1,…,B, samples sampled **with replacement** from *D*

$$\mathcal{D}^{*b} = \{z_1^{*b}, \dots, z_n^{*b}\} \overset{i.i.d.}{\sim} \widehat{F}_{Z,n} \qquad \theta^{*b} = t(\mathcal{D}^{*b})$$



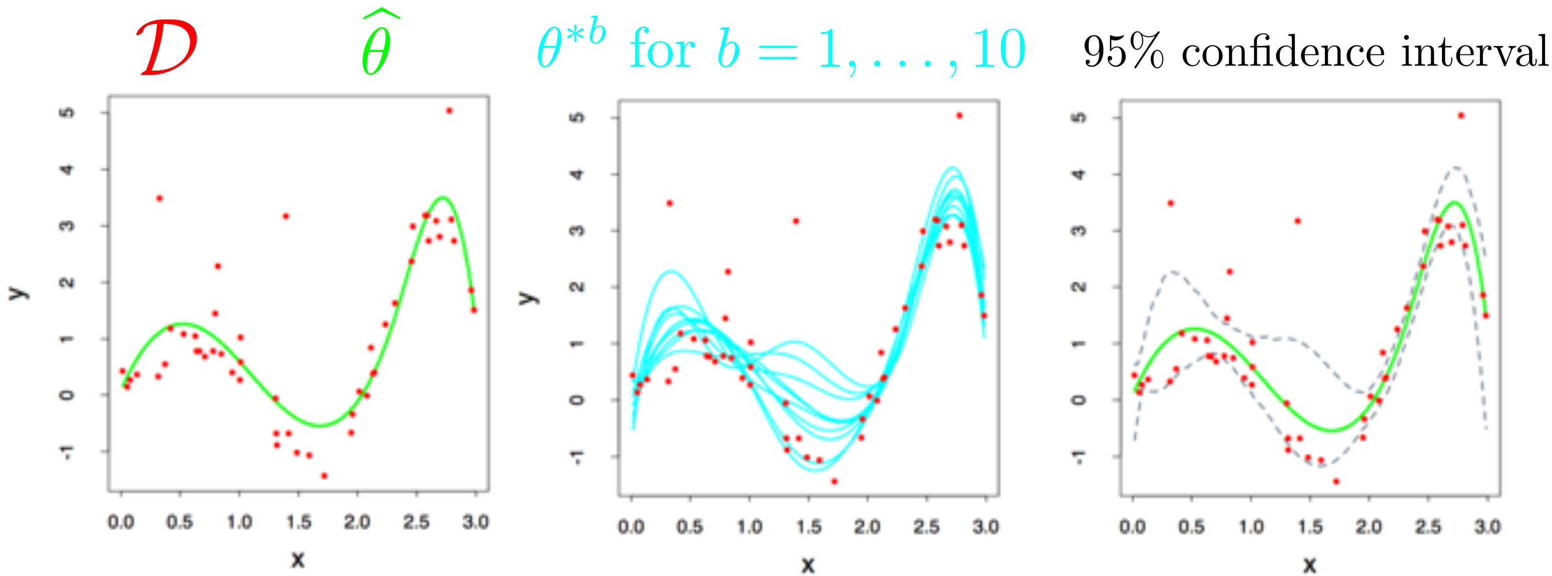
$$\boxed{\sup_x |\widehat{F}_n(x) - F(x)| \to 0 \quad \text{as } n \to \infty}$$



Applications

Common applications of the bootstrap:

- Estimate parameters that escape simple analysis like the variance or median of an estimate
- Confidence intervals
- Estimates of error for a particular example:



Figures from Hastie et al

Takeaways

Advantages:

- Bootstrap is **very** generally applicable. Build a confidence interval around ***anything***
- **Very** simple to use
- Appears to give meaningful results even when the amount of data is very small
- Very strong **asymptotic theory** (as num. examples goes to infinity)

Disadvantages

- Very few meaningful finite-sample guarantees
- Potentially **computationally intensive**
- Reliability relies on test statistic and rate of convergence of empirical CDF to true CDF, which is unknown
- Poor performance on "extreme statistics" (e.g., the max)

Not perfect, but better than nothing.

Recap

- Learning is…
  - Collect some data
    - E.g., housing info and sale price
  - Randomly split dataset into TRAIN, VAL, and TEST
    - E.g., 80%, 10%, and 10%, respectively
  - Choose a hypothesis class or model
    - E.g., linear with non-linear transformations
  - Choose a loss function
    - E.g., least squares with ridge regression penalty on TRAIN
  - Choose an optimization procedure
    - E.g., set derivative to zero to obtain estimator, cross-validation on VAL to pick num. features and amount of regularization
  - Justifying the accuracy of the estimate
    - E.g., report TEST error with Bootstrap confidence interval

Simple Variable Selection
LASSO: Sparse Regression

Machine Learning – CSE546

Kevin Jamieson
University of Washington

October 11, 2016

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, each prediction is expensive:
    - If part of an online system, too slow
    - If **w** is sparse, prediction computation only depends on number of non-zeros

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, each prediction is expensive:
    - If part of an online system, too slow
    - If **w** is sparse, prediction computation only depends on number of non-zeros
  - **Interpretability**: What are the relevant dimension to make a prediction?
    - E.g., what are the parts of the brain associated with particular words?

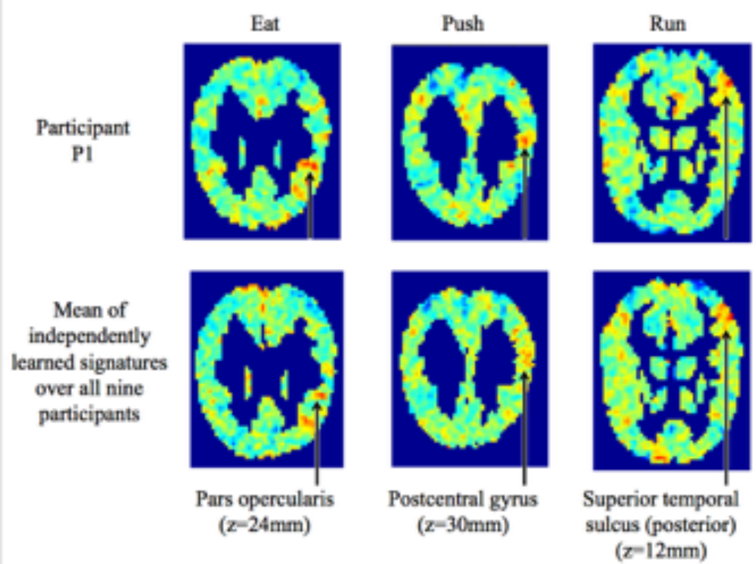


Figure from Tom Mitchell

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

- Very useful for many tasks, e.g.,
  - **Efficiency**: If size(**w**) = 100 Billion, each prediction is expensive:
    - If part of an online system, too slow
    - If **w** is sparse, prediction computation only depends on number of non-zeros
  - **Interpretability**: What are the relevant dimension to make a prediction?
    - E.g., what are the parts of the brain associated with particular words?

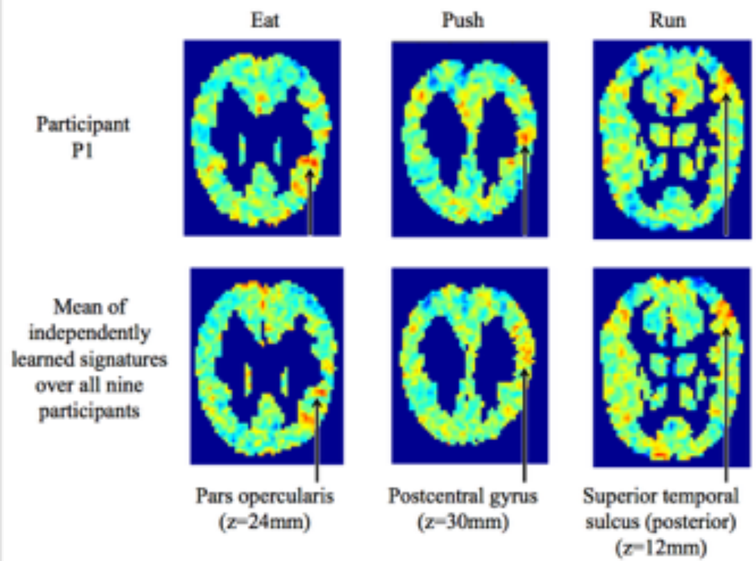


Figure from Tom Mitchell

- How do we find "best" subset among all possible?

Greedy model selection algorithm

- Pick a dictionary of features
  - e.g., cosines of random inner products
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain weights for these features
  - Select **next best feature $\mathbf{h_i(x)^*}$**
    - e.g., $h_j(x)$ that results in lowest training error learner when using $F_t + \{h_j(x)^*\}$
  - $F_{t+1} \leftarrow F_t + \{h_i(x)^*\}$
  - Recurse

Greedy model selection

- Applicable in many other settings:
  - Considered later in the course:
    - Logistic regression: Selecting features (basis functions)
    - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
    - Decision trees: Selecting leaves to expand
- Only a heuristic!
  - **Finding the best set of k features is computationally intractable!**
  - Sometimes you can prove something strong about it…

When do we stop???

- Greedy heuristic:
  - …
  - Select **next best feature X_i^***
    - E.g. $h_j(x)$ that results in lowest training error learner when using $F_t + \{h_j(x)^*\}$
  - Recurse

When do you stop???

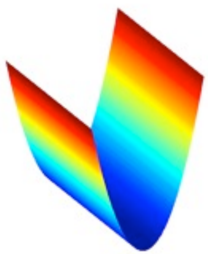
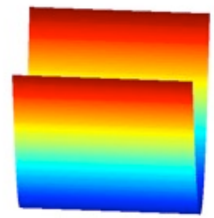
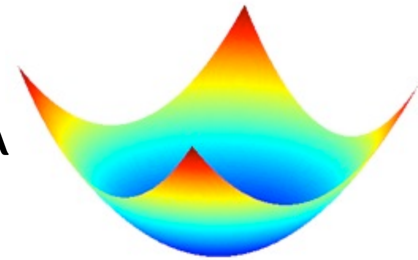
- When training error is low enough?
- When test set error is low enough?
- Using cross validation?

Is there a more principled approach?

Recall Ridge Regression

- Ridge Regression objective:

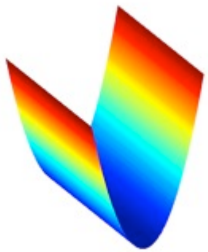
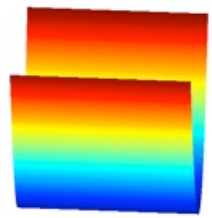
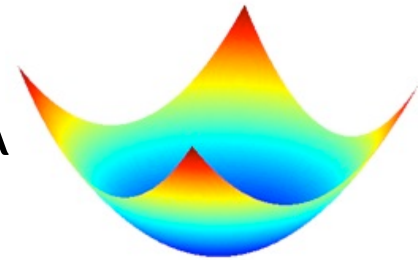
$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$


+

+ ... +

+ λ


Ridge vs. Lasso Regression

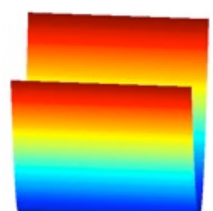
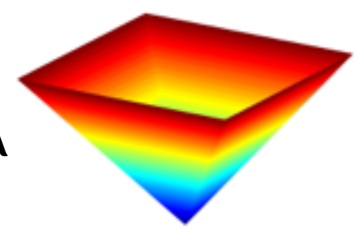
- Ridge Regression objective:

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

 +  + ... +  + λ 

- Lasso ~~Ridge~~ objective:

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

 +  + ... +  + λ 

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

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

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

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(\lambda) \leq \nu$$

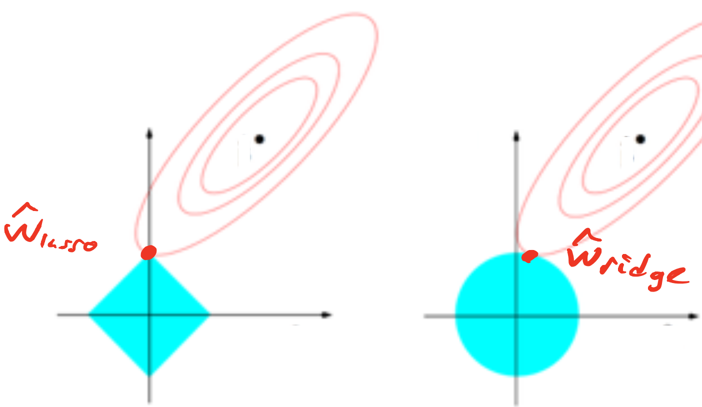
Penalized Least Squares

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

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

For any $\lambda \geq 0$ for which $\widehat{w}_r$ achieves the minimum, there exists a $\nu \geq 0$ such that

$$\widehat{w}_r = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 \qquad \text{subject to } r(\cancel{\lambda}\, w) \leq \nu$$



Optimizing the LASSO Objective

- LASSO solution:

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

$$\widehat{b}_{lasso} = \arg\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^T \widehat{w}_{lasso})\right)$$

Optimizing the LASSO Objective

- LASSO solution:

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

$$\widehat{b}_{lasso} = \arg\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i^T \widehat{w}_{lasso})\right)$$

So as usual, preprocess to make sure that $\frac{1}{n}\sum_{i=1}^{n} y_i = 0, \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbf{0}$

so we don't have to worry about an offset.

Optimizing the LASSO Objective

- LASSO solution:

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

$$\widehat{b}_{lasso} = \cancel{\arg\min_{w,b}} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \widehat{w}_{lasso}))$$

$\widehat{w}_{LS} = (X^TX)^{-1}X^Ty$

So as usual, preprocess to make sure that $\frac{1}{n}\sum_{i=1}^{n} y_i = 0, \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbf{0}$

so we don't have to worry about an offset.

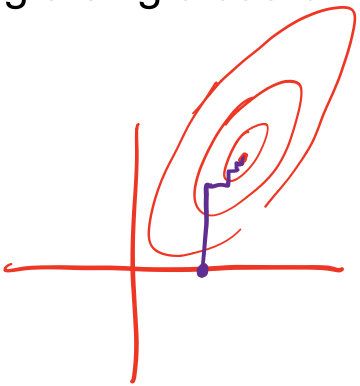
$$\widehat{w}_{lasso} = \arg\min_{w} \sum_{i=1}^{n} \left(y_i - x_i^T w\right)^2 + \lambda ||w||_1$$

How do we solve this?

Coordinate Descent

- Given a function, we want to find minimum

- Often, it is easy to find minimum along a single coordinate:



- How do we pick next coordinate?

- Super useful approach for *many* problems
  - Converges to optimum in some cases, such as LASSO

Optimizing LASSO Objective One Coordinate at a Time

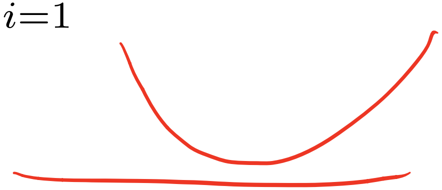
$$\sum_{i=1}^{n}\left(y_i - x_i^T w\right)^2 + \lambda ||w||_1 = \sum_{i=1}^{n}\left(y_i - \sum_{k=1}^{d} x_{i,k}\, w_k\right)^2 + \lambda \sum_{k=1}^{d} |w_k|$$

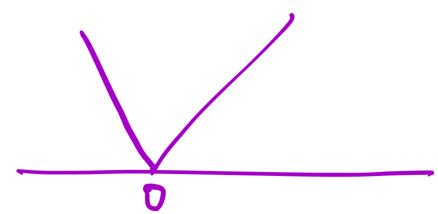
Fix $j \in \{1, \dots, d\}$

$$= \sum_{i=1}^{n}\left(\underbrace{\left(y_i - \sum_{k\neq j} x_{i,k}\, w_k\right)}_{r_i^{(j)}} - x_{i,j}\, w_j\right)^2 + \lambda \sum_{k \neq j} |w_k| + \lambda |w_j|$$

Equivalently:

$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\, w_j\right)^2 + \lambda |w_j|$$

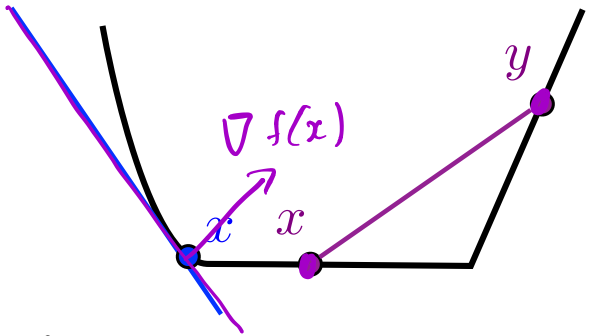


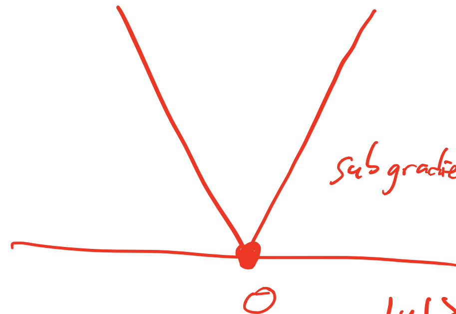


Convex Functions

- Equivalent definitions of convexity:

$f(x) = |x|$





$$\text{subgradients} = \begin{cases} 1 & x>0 \\ [-1,1] & x=0 \\ -1 & x<0 \end{cases}$$

$|y| \geq gy$

f convex:

$$f\left(\lambda x + (1-\lambda)y\right) \leq \lambda f(x) + (1-\lambda)f(y) \qquad \forall x, y, \lambda \in [0,1]$$

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) \qquad \forall x, y$$

- **Gradients** lower bound convex functions and are unique at **x** iff function differentiable at **x**
- **Subgradients** generalize gradients to non-differentiable points:
  - Any supporting hyperplane at **x** that lower bounds entire function

$$g \text{ is a subgradient at } x \text{ if } f(y) \geq f(x) + g^T(y-x)$$

Taking the Subgradient

$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n} \left(r_i^{(j)} - x_{i,j}\, w_j\right)^2 + \lambda |w_j|$$

- Convex function is minimized at w if 0 is a sub-gradient at w.

$$\partial_{w_j} |w_j| = \begin{cases} 1 & \text{if } w_j > 0 \\ \in [-1, 1] & \text{if } w_j = 0 \\ -1 & \text{if } w_j < 0 \end{cases}$$

$$\partial_{w_j} \sum_{i=1}^{n} \left(r_i^{(j)} - x_{i,j}\, w_j\right)^2 = -2 \sum_{i=1}^{n} x_{i,j} r_i^{(j)} + 2 \sum_{i=1}^{n} x_{i,j}^2 w_j$$

Setting Subgradient to 0

$$\partial_{w_j}\left(\sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\,w_j\right)^2 + \lambda|w_j|\right) = \begin{cases} a_j w_j - c_j - \lambda & \text{if } w_j < 0 \\ [-c_j - \lambda, -c_j + \lambda] & \text{if } w_j = 0 \\ a_j w_j - c_j + \lambda & \text{if } w_j > 0 \end{cases}$$

$$a_j = (\sum_{i=1}^{n} x_{i,j}^2) \qquad c_j = 2(\sum_{i=1}^{n} r_i^{(j)} x_{i,j})$$

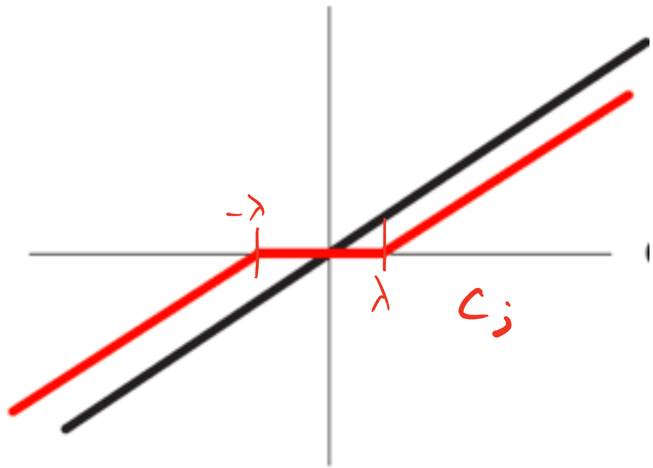
$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n}\left(r_i^{(j)} - x_{i,j}\,w_j\right)^2 + \lambda|w_j|$$

$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

Soft Thresholding

$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

$$a_j = \sum_{i=1}^{n} x_{i,j}^2 \qquad c_j = 2\sum_{i=1}^{n}\Big(y_i - \sum_{k\neq j} x_{i,k}\, w_k\Big)x_{i,j}$$



From
Kevin Murphy
textbook

Coordinate Descent for LASSO (aka Shooting Algorithm)

- Repeat until convergence
  - Pick a coordinate *l* at (random or sequentially)
    - Set:

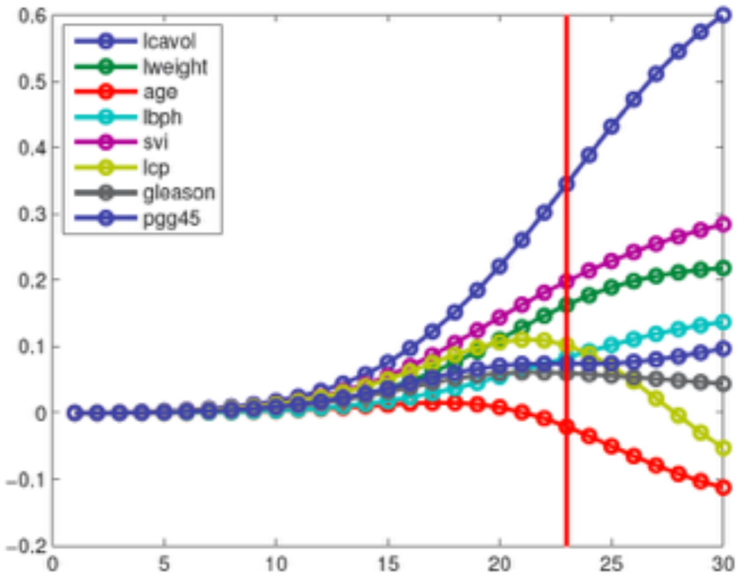
$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

    - Where:

$$a_j = \sum_{i=1}^{n} x_{i,j}^2 \qquad c_j = 2\sum_{i=1}^{n}\Big(y_i - \sum_{k \neq j} x_{i,k}\, w_k\Big)x_{i,j}$$

  - For convergence rates, see Shalev-Shwartz and Tewari 2009
- Other common technique = LARS
  - Least angle regression and shrinkage, Efron et al. 2004

Recall: *Ridge Coefficient Path*

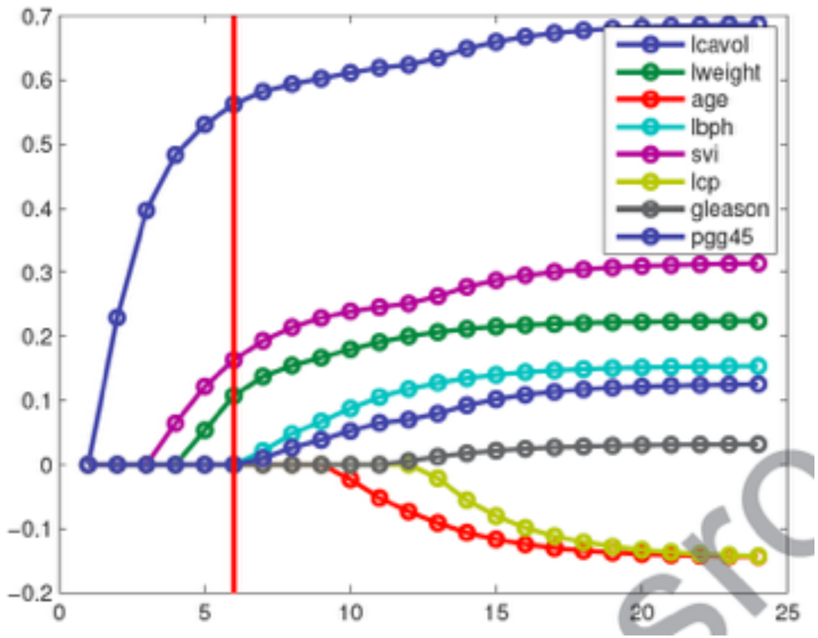


$\lambda \to 0$

From
Kevin Murphy
textbook

- Typical approach: select λ using cross validation

Now: *LASSO Coefficient Path*



$\lambda \to 0$

From
Kevin Murphy
textbook

What you need to know

- Variable Selection: find a sparse solution to learning problem
- L_1 regularization is one way to do variable selection
  - Applies beyond regression
  - Hundreds of other approaches out there
- LASSO objective non-differentiable, **but convex** ➔ Use subgradient
- No closed-form solution for minimization ➔ Use coordinate descent
- Shooting algorithm is simple approach for solving LASSO

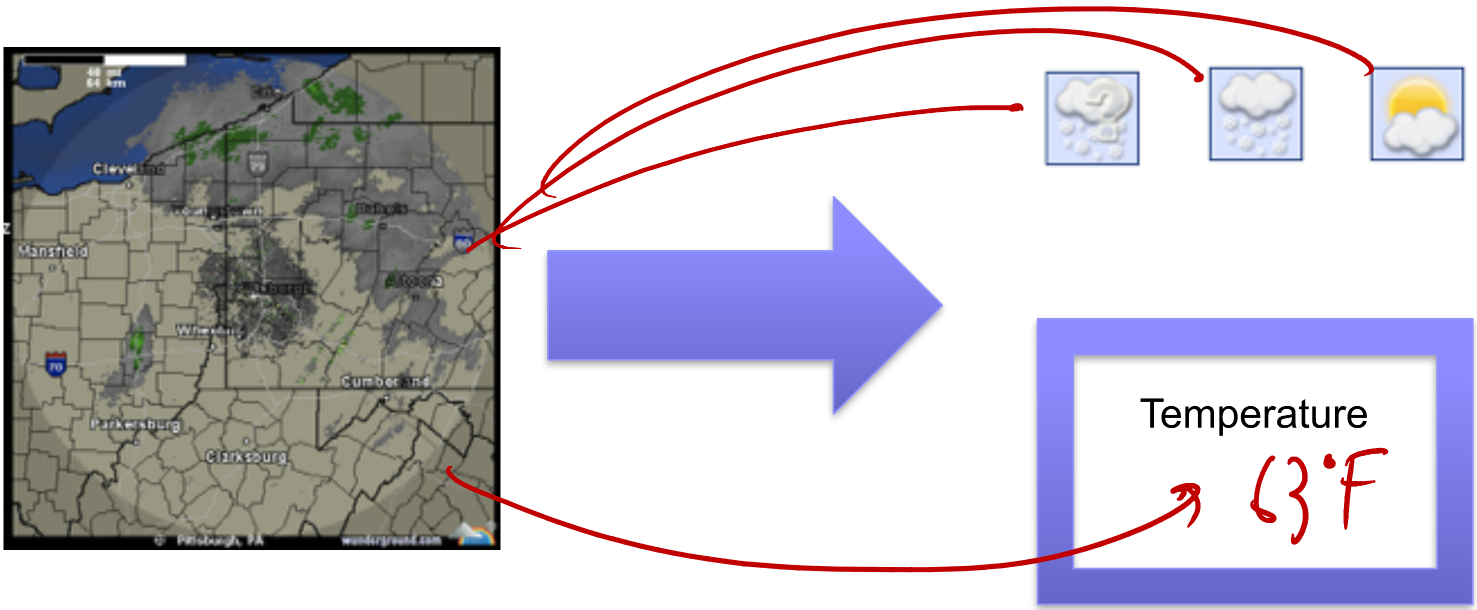
Classification
Logistic Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 12, 2016

THUS FAR, REGRESSION: PREDICT A CONTINUOUS VALUE GIVEN SOME INPUTS

Weather prediction revisted

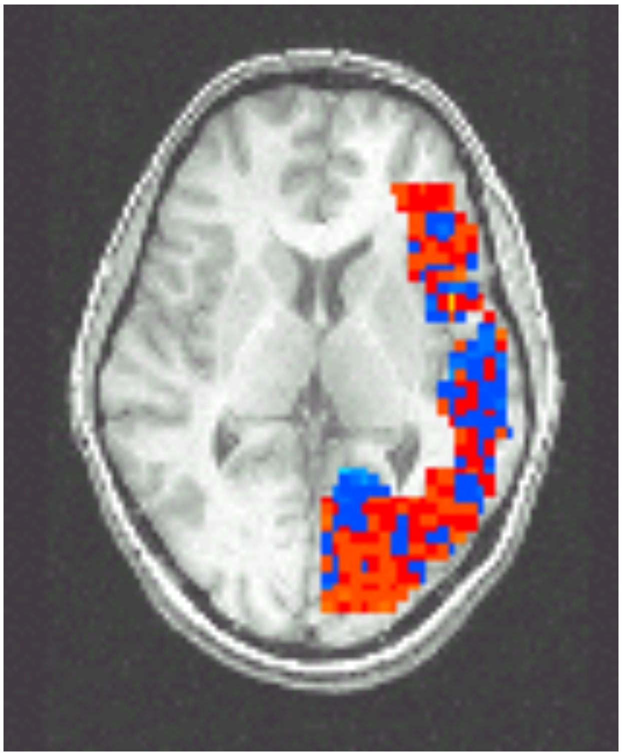


Reading Your Brain, Simple Example

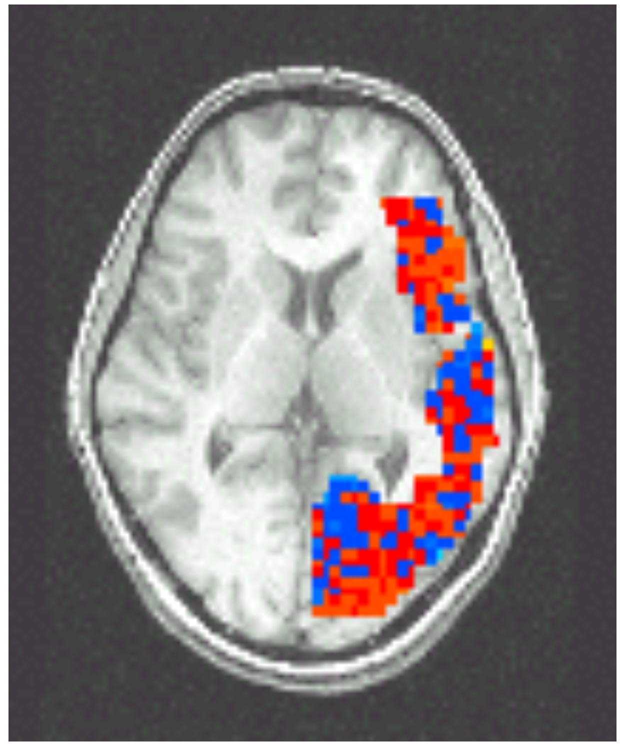
[Mitchell et al.]

Pairwise classification accuracy: 85%

Person



Animal



Classification

- **Learn**: f:**X** —>Y
  - **X** – features
  - Y – target classes

- Conditional probability: P(Y|**X**)

- Suppose you know P(Y|**X**) exactly, how should you classify?
  - Bayes optimal classifier:

- **How do we estimate P(Y|X)?**

Link Functions

- Estimating P(Y|**X**): Why not use standard linear regression?

$\cancel{P(Y|x) \approx w^T x} \; > 1$

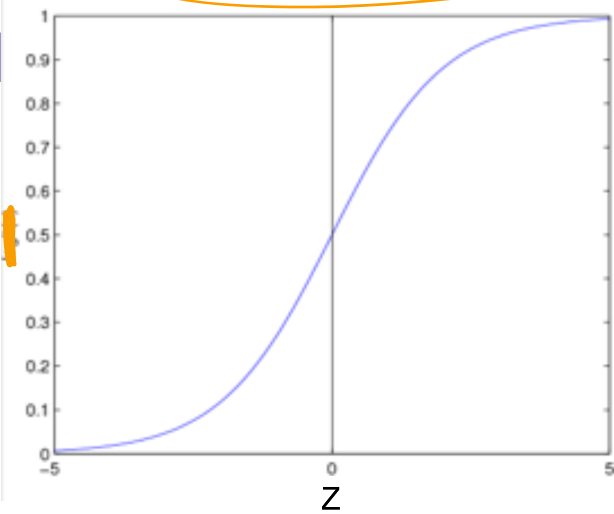
- Combining regression and probability?
  - Need a mapping from real values to [0,1]
  - A link function!

Logistic Regression

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$

- Learn P(Y|**X**) directly
  - Assume a particular functional form for link function
  - Sigmoid applied to a linear function of the input features:

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

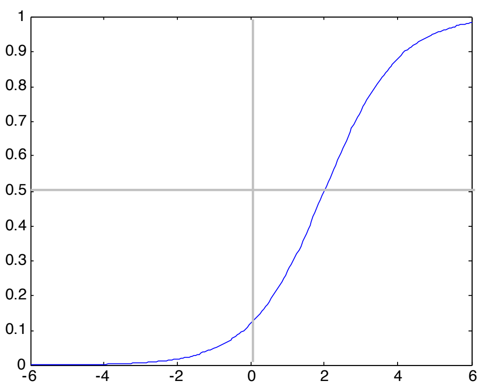


Features can be discrete or continuous!

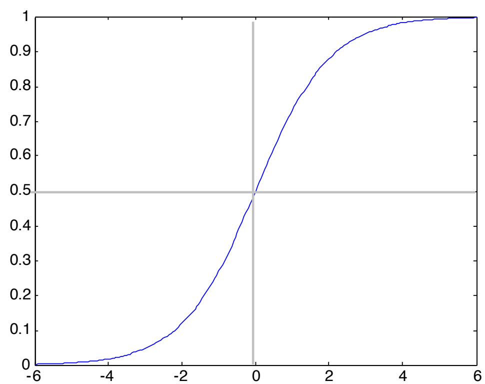
Understanding the sigmoid

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

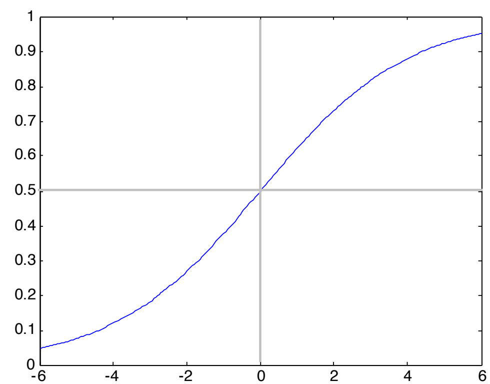
w_0=-2, w_1=-1



w_0=0, w_1=-1



w_0=0, w_1=-0.5



Very convenient!

$$P(Y=0|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=1|X=<X_1,...X_n>) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

Very convenient!

$$P(Y=0|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=1|X=<X_1,...X_n>) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

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

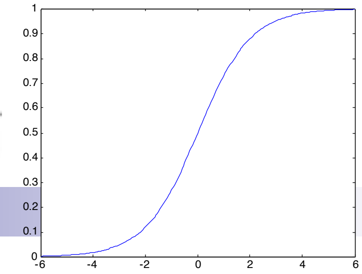
linear classification rule!

implies

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

Logistic Regression – a Linear classifier

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



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

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

Loss function: Conditional Likelihood

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

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

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

- 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 of the form: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

$= \arg\min_w -\log(\quad)$

$= \arg\min_w -\sum$

Loss function: Conditional Likelihood

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

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

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

Loss function: Conditional Likelihood

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

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

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

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

$$\text{Squared error Loss: } \ell_i(w) = (y_i - x_i^T w)^2$$ (MLE for Gaussian noise)

Loss function: Conditional Likelihood

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

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

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

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

Loss function: Conditional Likelihood

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

$$\widehat{w}_{MLE} = \arg\max_w \prod_{i=1}^n P(y_i|x_i, w) \qquad P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

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

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

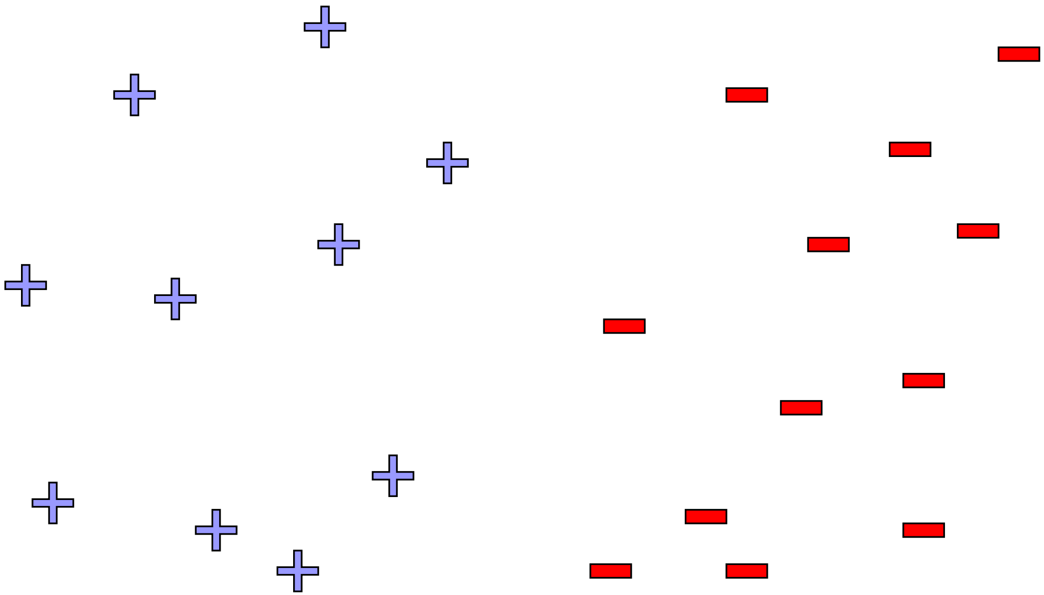
Bad news: no closed-form solution to maximize *J*(**w**)

Good news: convex functions easy to optimize (next time)

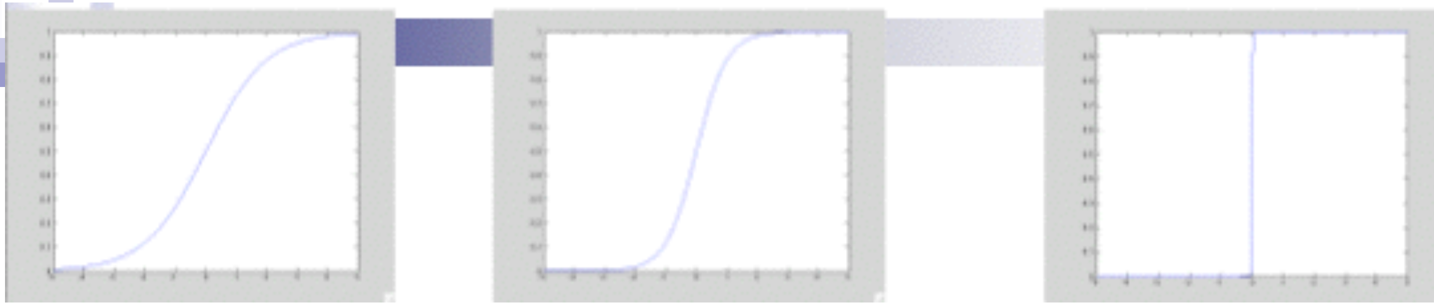
Linear Separability

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

When is this loss small?



Large parameters $\rightarrow$ Overfitting



$$\frac{1}{1+e^{-x}} \qquad \frac{1}{1+e^{-2x}} \qquad \frac{1}{1+e^{-100x}}$$

- If data is linearly separable, weights go to infinity
  - In general, leads to overfitting:
- Penalizing high weights can prevent overfitting…

Regularized Conditional Log Likelihood

- Add regularization penalty, e.g., L_2:

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

- Practical note about w_0: