Is the test error unbiased for these programs?

```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
mu = np.mean(X, axis=0)
X = X - mu

idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                     np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

No. Preprocessing by de-meaning using whole (TEST) set.

```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
idx = np.random.permutation(1000)
TRAIN = idx[0:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
Xtrain_avg = np.mean(Xtrain, axis=0)
Xtrain = Xtrain - Xtrain_avg

# Solve for argmin_w ||Xtrain*w - ytrain||_2
w = np.linalg.solve( np.dot(Xtrain.T, Xtrain),
                     np.dot(Xtrain.T, ytrain) )
b = np.mean(ytrain)

ytest = y[TEST]
Xtest = X[TEST,:]
~~Xtest_avg = np.mean(Xtest, axis=0)~~
Xtest = Xtest - ~~Xtest_avg~~ Xtrain_avg

train_error = np.dot( np.dot(Xtrain, w)+b - ytrain,
                np.dot(Xtrain, w)+b - ytrain )/len(TRAIN)
test_error = np.dot( np.dot(Xtest, w)+b - ytest,
                np.dot(Xtest, w)+b - ytest )/len(TEST)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

Is the test error unbiased for this program?

```python
# Given dataset of 1000-by-50 feature
# matrix X, and 1000-by-1 labels vector
mu = np.mean(X, axis=0)
X = X - mu

idx = np.random.permutation(1000)
TRAIN = idx[0:800]
VAL = idx[800:900]
TEST = idx[900::]

ytrain = y[TRAIN]
Xtrain = X[TRAIN,:]
yval = y[VAL]
Xval = X[VAL,:]

err = np.zeros(50)
for d in range(1,51):
    w, b = fit(Xtrain[:,0:d], ytrain)
    yval_hat = predict(w, b, Xval[:,0:d])
    err[d-1] = np.mean((yval_hat-yval)**2)
d_best = np.argmin(err)+1
w, b = fit(Xtrain[:,0:d_best], ytrain)

Xtot = np.concatenate((Xtrain, Xval), axis=0)
ytot = np.concatenate((ytrain, yval), axis=0)

ytest = y[TEST]
Xtest = X[TEST,:]

ytot_hat = predict(w, b, Xtot[:,0:d_best])
tot_train_error = np.mean((ytot_hat-ytot)**2)
ytest_hat = predict(w, b, Xtest[:,0:d_best])
test_error = np.mean((ytest_hat-ytest)**2)

print('Train error = ',train_error)
print('Test error = ',test_error)
```

tot tot

```python
def fit(Xin, Yin):
    mu = np.mean(Xin, axis=0)
    Xin = Xin - mu
    w = np.linalg.solve( np.dot(Xin.T, Xin),
                         np.dot(Xin.T, Yin) )
    b = np.mean(Yin) - np.dot(w, mu)
    return w, b

def predict(w, b, Xin):
    return np.dot(Xin, w)+b
```

$$c = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$f(x) = (x-\mu)^T w + c$$
$$= x^T w \underbrace{- \mu^T w + c}_{b}$$

(see non-annotated slides for correct example)

Simple Variable Selection LASSO: Sparse Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 9, 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?

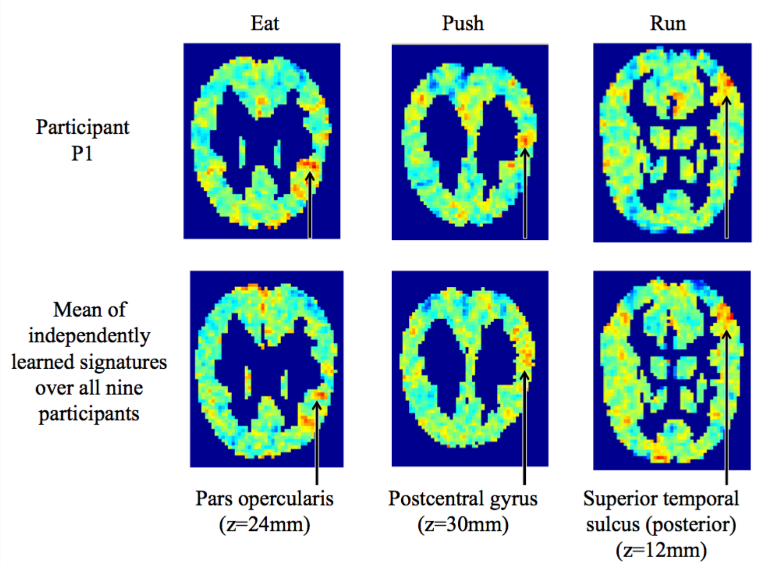


Figure from Tom Mitchell

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?

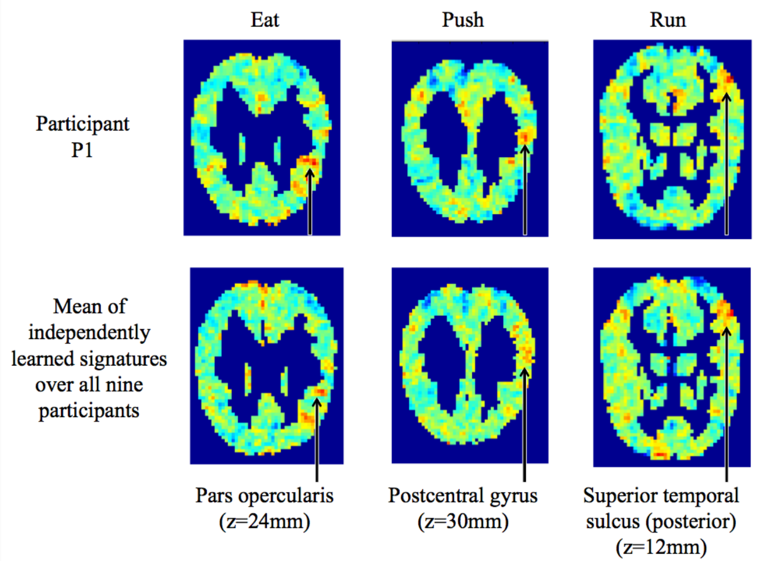


Figure from Tom Mitchell

- 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 $h_i(x)^*$**
    - 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 $\mathbf{X_i^*}$**
    - E.g. $h_j(x)$ that results in lowest training error learner when using $F_t + \{h_j(x)^*\}$
  - Recurse

When do you stop???

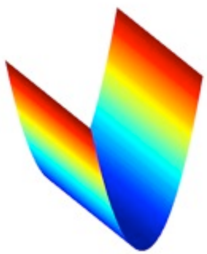
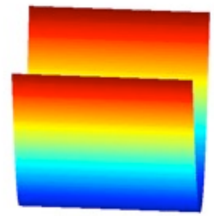
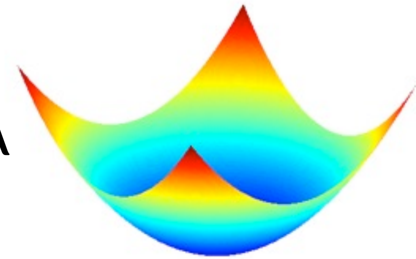
- When training error is low enough?
- When test set error is low enough?
- Using cross validation?

Is there a more principled approach?

Recall Ridge Regression

- Ridge Regression objective:

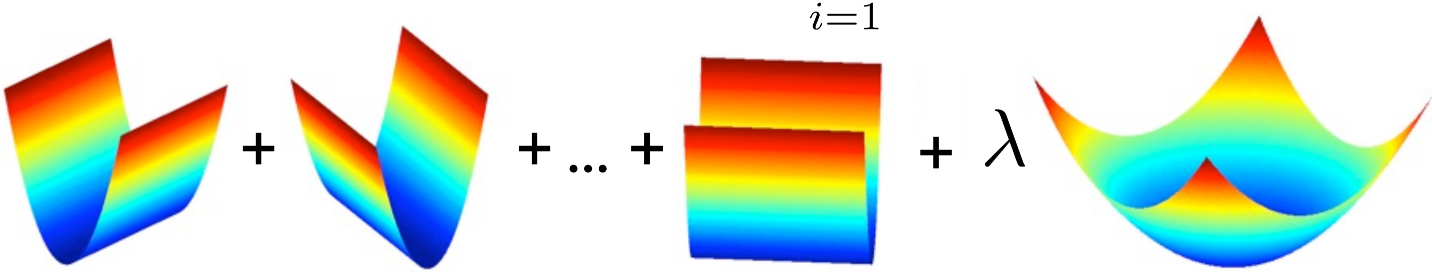
$$\widehat{w}_{ridge} = \arg\min_w \sum_{i=1}^n \left(y_i - x_i^T w\right)^2 + \lambda ||w||_2^2$$

 +  + ... +  + λ 

Ridge vs. Lasso Regression

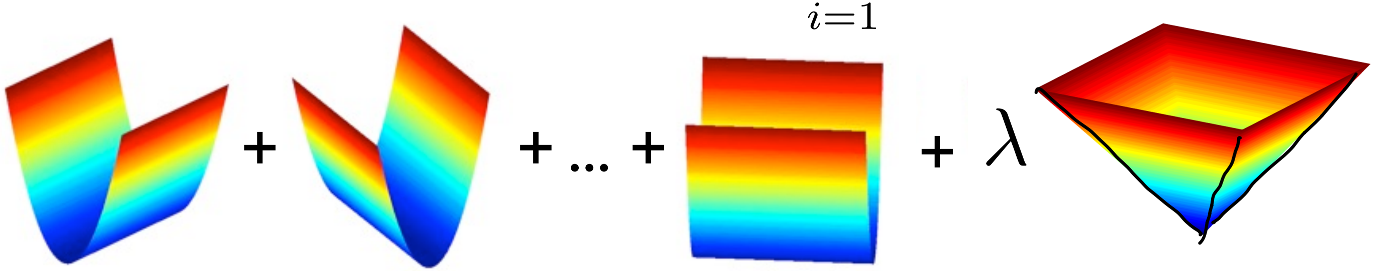
- Ridge Regression objective:

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



- Lasso objective:

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



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

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

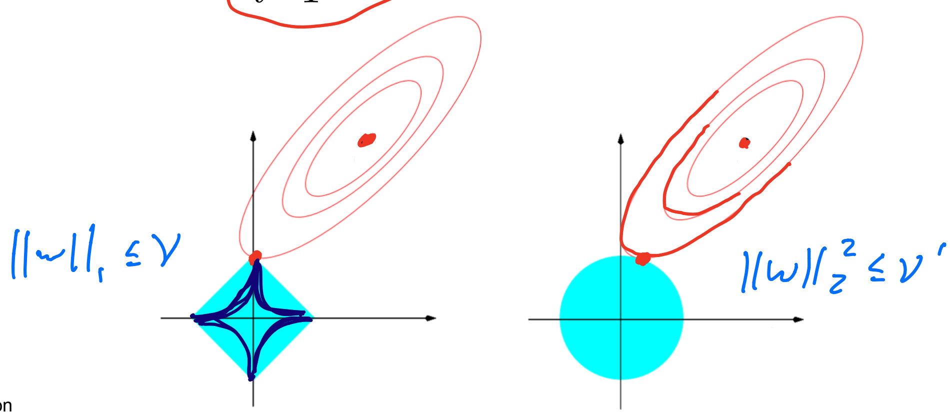
Penalized Least Squares

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

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

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

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



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 $\frac{1}{n}\sum_{i=1}^{n} y_i = 0, \frac{1}{n}\sum_{i=1}^{n} x_i = \mathbf{0}$

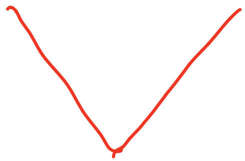
so we don't have to worry about an offset.

Optimizing the LASSO Objective

- LASSO solution:

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

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



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

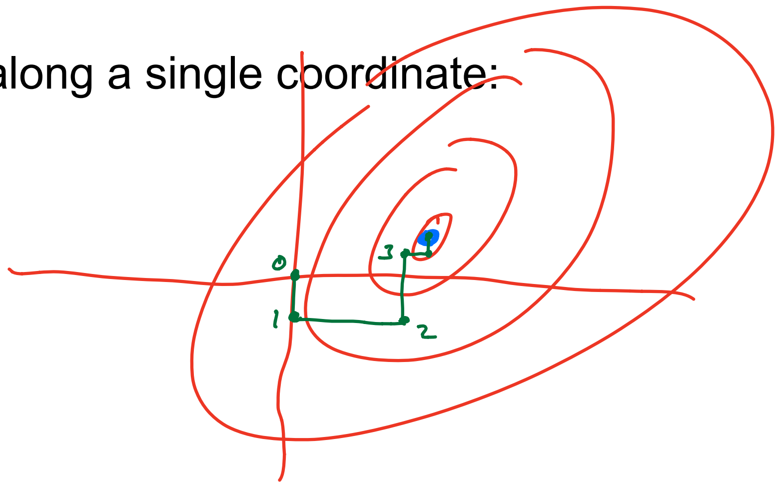
so we don't have to worry about an offset.

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

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?

  - Randomly
  - Round Robin

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

Optimizing LASSO Objective One Coordinate at a Time

Fix any $j \in \{1, \ldots, d\}$

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

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

Optimizing LASSO Objective One Coordinate at a Time

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

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

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

Initialize $\widehat{w}_k = 0$ for all $k \in \{1, \dots, d\}$

Loop over $j \in \{1, \dots, n\}$:

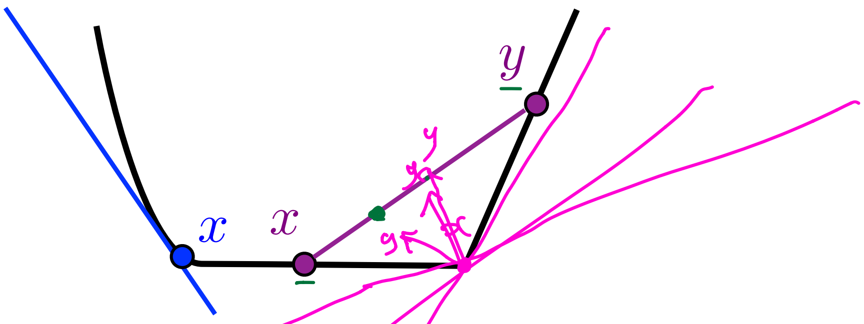
$$r_i^{(j)} = y_i - \sum_{k \neq j} x_{i,j} \widehat{w}_k$$

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

Convex Functions

f is convex $\iff \{(x, z) : x \in \mathbb{R}^d, z \geq f(x)\}$ is a convex set.

- Equivalent definitions of convexity:



f convex:

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

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

- **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

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

Taking the Subgradient

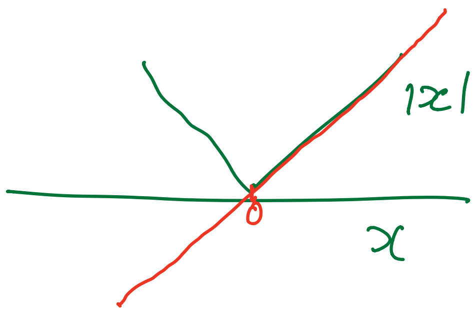
$$\widehat{w}_j = \arg\min_{w_j} \sum_{i=1}^{n} \left(r_i^{(j)} - x_{i,j}\, w_j\right)^2 + \lambda |w_j|$$

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

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

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

$|y| \geq |0| + g(y-0)$
$= gy$



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

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

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

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

a_j

Setting Subgradient to 0

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

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

$$w_j = \frac{c_j + \lambda}{a_j} < 0$$

$$\lambda < -c_j$$

$$|\lambda| \le |c_j|$$

Setting Subgradient to 0

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

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

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

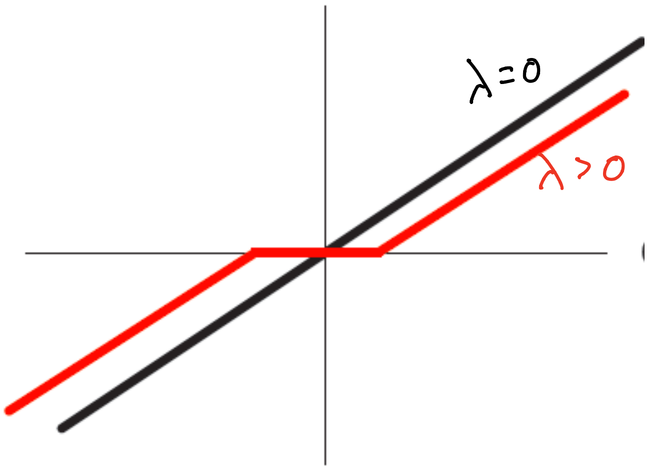
w is a minimum if
0 is a sub-gradient at w

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

Soft Thresholding

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

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



Coordinate Descent for LASSO (aka Shooting Algorithm)

- Repeat until convergence (initialize w=0)
  - Pick a coordinate *l* at (random or sequentially)
    - Set:

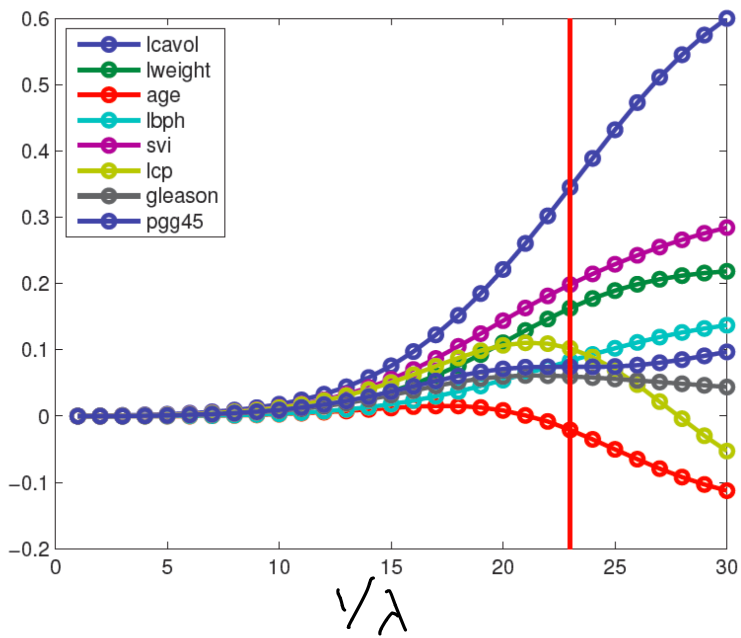
$$\widehat{w}_j = \begin{cases} (c_j + \lambda)/a_j & \text{if } c_j < -\lambda \\ 0 & \text{if } |c_j| \leq \lambda \\ (c_j - \lambda)/a_j & \text{if } c_j > \lambda \end{cases}$$

    - Where:

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

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

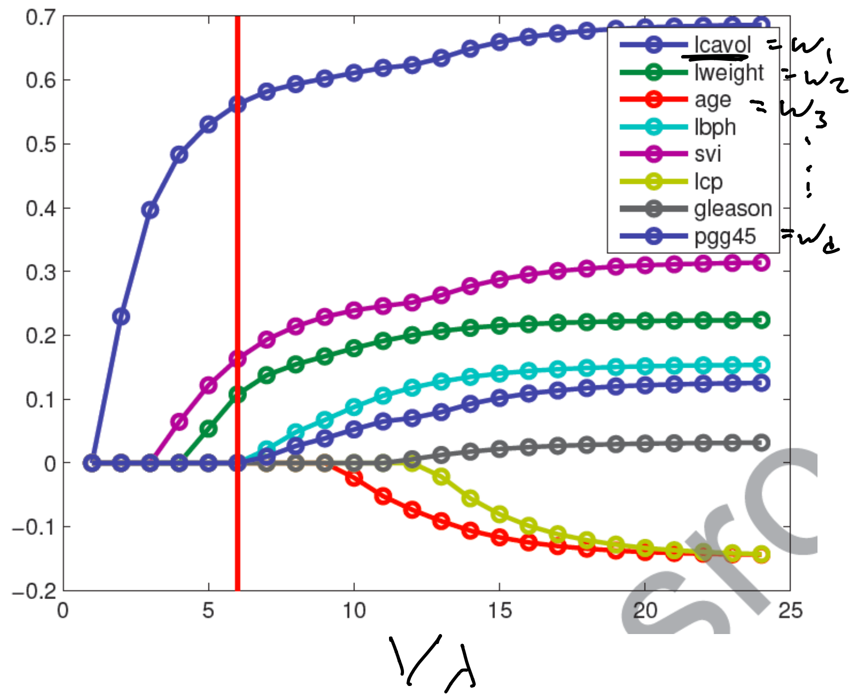
Recall: *Ridge Coefficient Path*



From
Kevin Murphy
textbook

- Typical approach: select λ using cross validation

Now: *LASSO Coefficient Path*



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

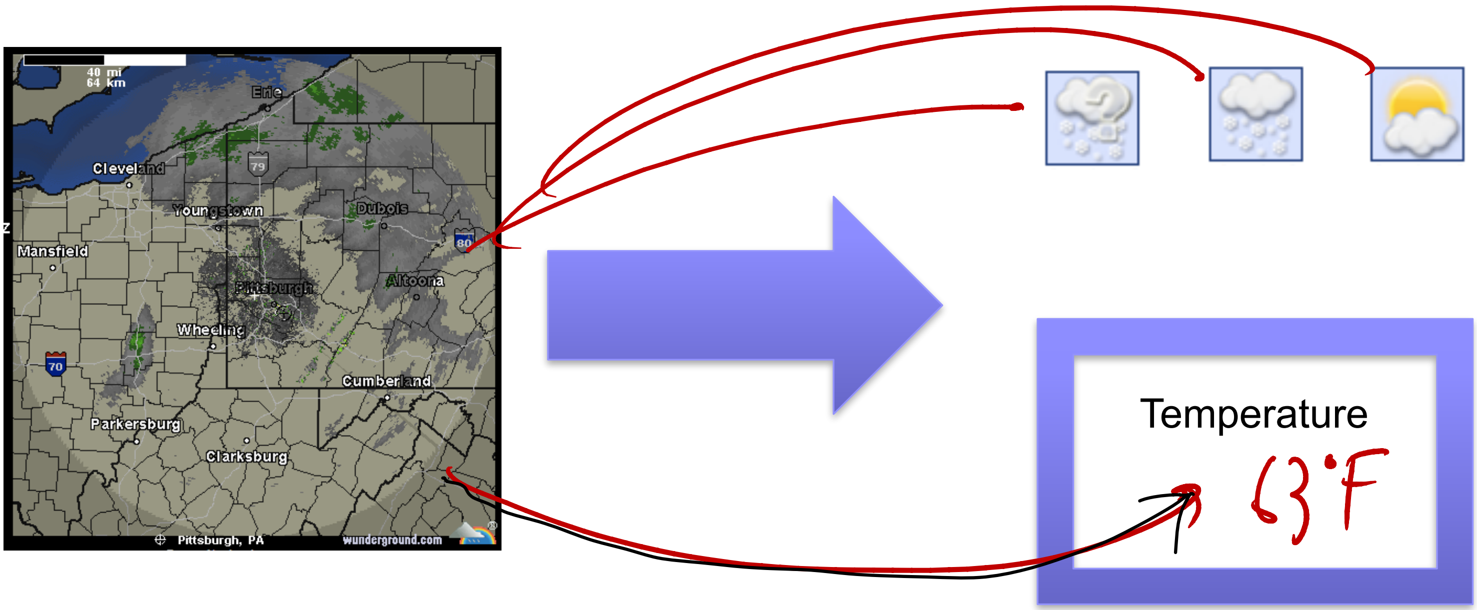
Classification
Logistic Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October ~~9~~ 11, 2016

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

Weather prediction revisted

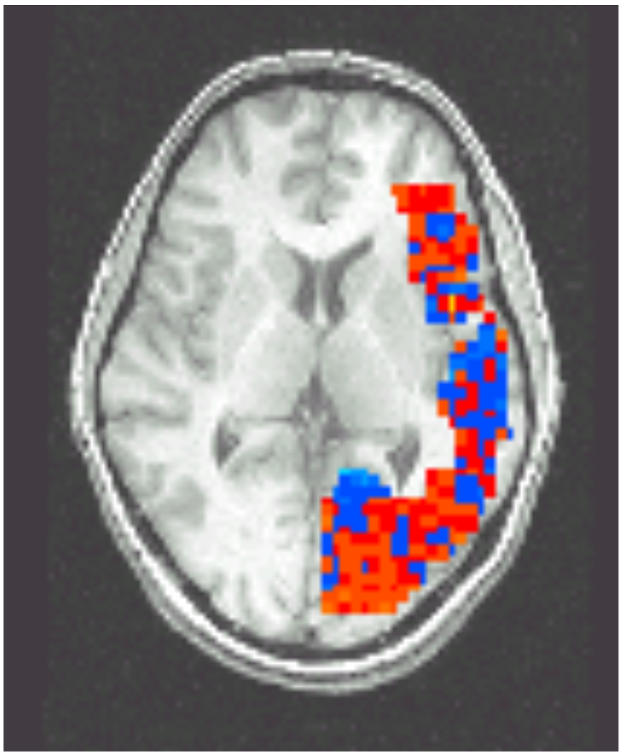


Reading Your Brain, Simple Example

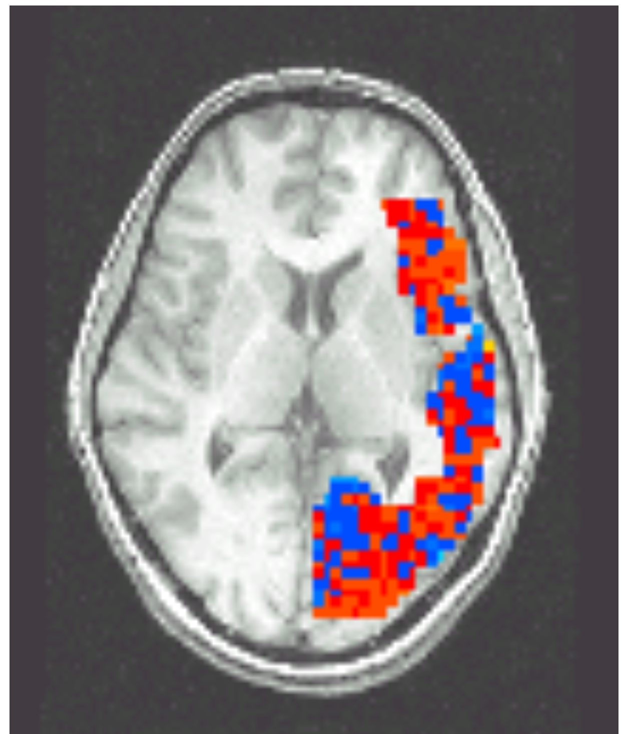
[Mitchell et al.]

Pairwise classification accuracy: 85%

Person



Animal



Binary Classification

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

    $Y \in \{0,1\}$

- **Loss function:** $\mathbb{1}\{f(X) \neq Y\}$ "0/1 Loss"

- **Expected loss of f:**

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

$$\mathbb{1}\{f(x)=1\}\mathbb{P}(Y=0 \mid X=x) + \mathbb{1}\{f(x)=0\}\mathbb{P}(Y=1 \mid X=x)$$

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

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

Binary Classification

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

$$Y \in \{0, 1\}$$

- **Loss function:** $\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$

- **Expected loss of f:**

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

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = \mathbf{1}\{f(x) = 1\}\mathbb{P}(Y = 0|X = x) + \mathbf{1}\{f(x) = 0\}\mathbb{P}(Y = 1|X = x)$$

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

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

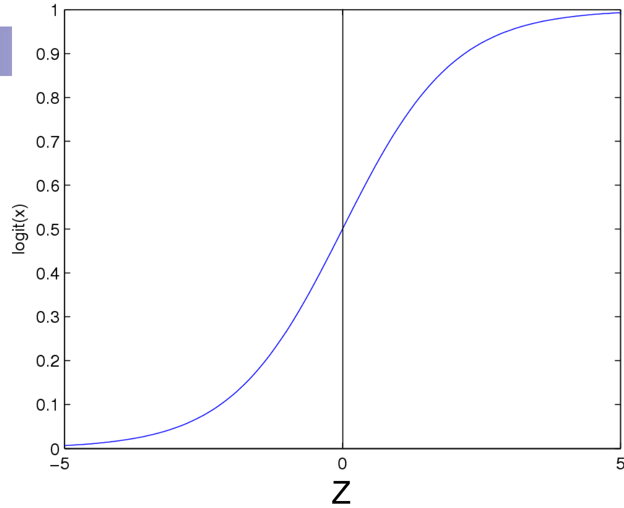
Link Functions

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

- 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:

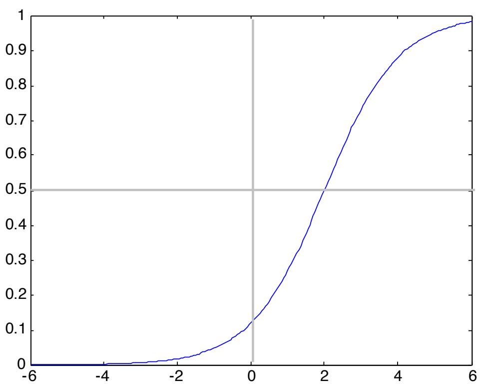
$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

Features can be discrete or continuous!

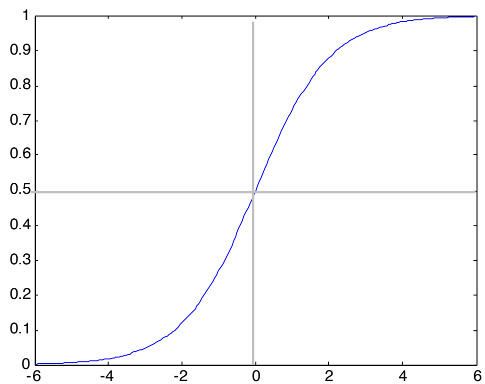
Understanding the sigmoid

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

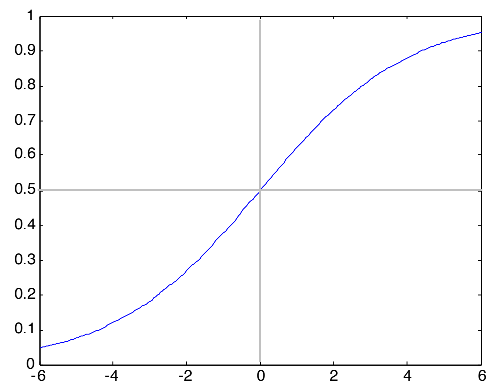
w_0=-2, w_1=-1



w_0=0, w_1=-1



w_0=0, w_1=-0.5



Sigmoid for binary classes

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

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

$$\frac{\mathbb{P}(Y=1|w,X)}{\mathbb{P}(Y=0|w,X)} =$$

Sigmoid for binary classes

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

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

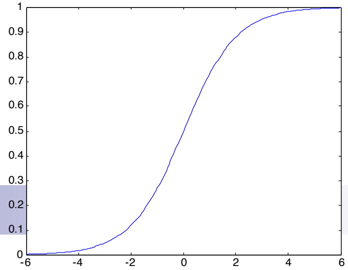
$$\frac{\mathbb{P}(Y=1|w,X)}{\mathbb{P}(Y=0|w,X)} = \exp(w_0 + \sum_k w_k X_k)$$

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

Linear Decision Rule!

Logistic Regression – a Linear classifier

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



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

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

Loss function: Conditional Likelihood

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

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

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

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

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

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

Loss function: Conditional Likelihood

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

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

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

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)

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: $J(\mathbf{w})$ is convex function of **w**, no local optima problems

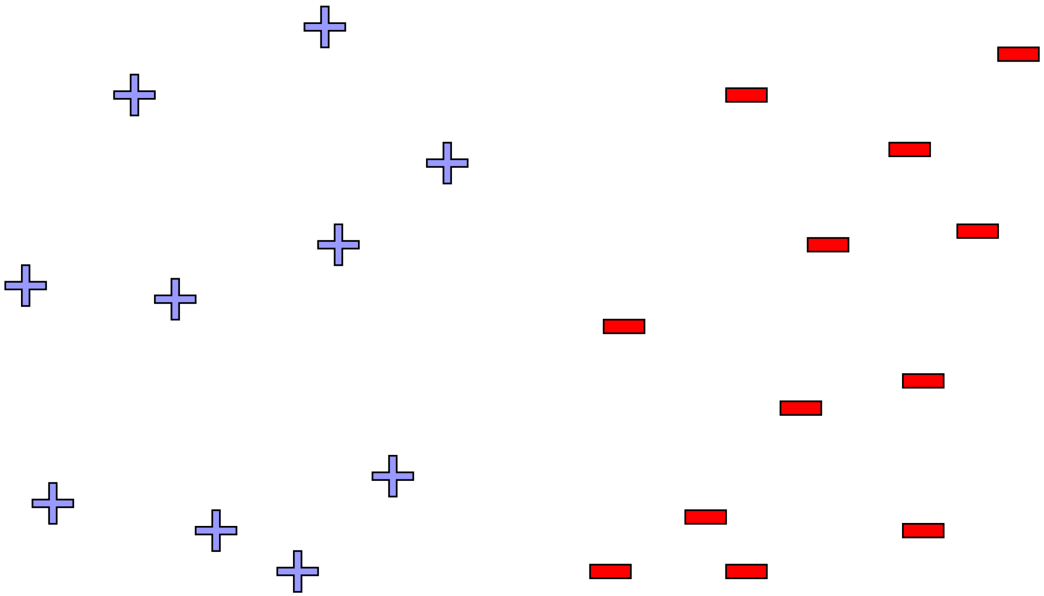
Bad news: no closed-form solution to maximize $J(\mathbf{w})$

Good news: convex functions easy to optimize

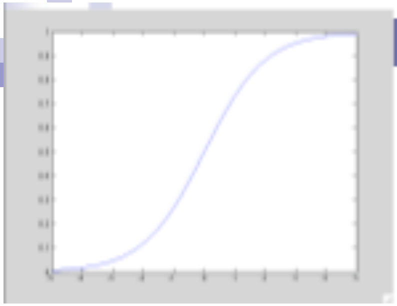
Linear Separability

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

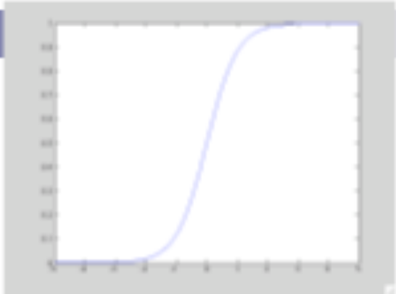
When is this loss small?



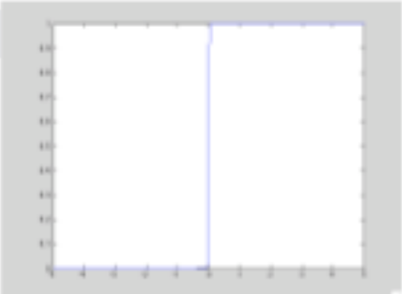
Large parameters → Overfitting



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



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



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

- 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:

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

Be sure to not regularize the offset b!

Gradient Descent

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 11, 2016

Machine Learning Problems

- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\displaystyle\sum_{i=1}^n \ell_i(w)$

  Each $\ell_i(w)$ is convex.

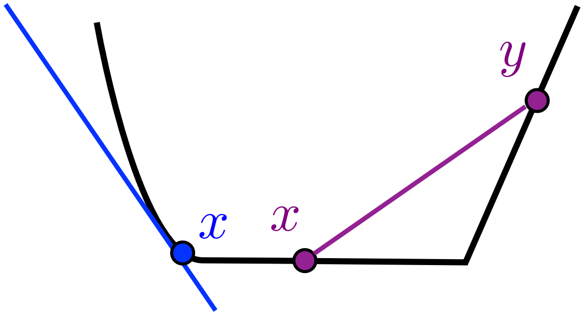
Machine Learning Problems

- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\sum_{i=1}^n \ell_i(w)$

Each $\ell_i(w)$ is convex.



g is a subgradient at x if

$$f(y) \geq f(x) + g^T(y - x)$$

f convex:

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

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

Machine Learning Problems

- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters:

  Each $\ell_i(w)$ is convex.

$$\sum_{i=1}^n \ell_i(w)$$

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

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

Least squares

- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters:

$$\sum_{i=1}^n \ell_i(w)$$

Each $\ell_i(w)$ is convex.

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

How does software solve: $\frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

Least squares

- Have a bunch of iid data of the form:

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\sum_{i=1}^n \ell_i(w)$

Each $\ell_i(w)$ is convex.

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

How does software solve: $\frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

…its complicated:
(LAPACK, BLAS, MKL…)

Do you need high precision?
Is X column/row sparse?
Is $\widehat{w}_{LS}$ sparse?
Is $\mathrm{X}^T\mathrm{X}$ "well-conditioned"?
Can $\mathrm{X}^T\mathrm{X}$ fit in cache/memory?

Taylor Series Approximation

- Taylor series in one dimension:

$$f(x+\delta) = f(x) + f'(x)\delta + \frac{1}{2}f''(x)\delta^2 + \dots$$

- Gradient descent:

Taylor Series Approximation

- Taylor series in **d** dimensions:

$$f(x+v) = f(x) + \nabla f(x)^T v + \tfrac{1}{2} v^T \nabla^2 f(x) v + \dots$$

- Gradient descent:

Gradient Descent $f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

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

$$\nabla f(w) =$$

Gradient Descent $f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

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

$$\begin{aligned}(w_{t+1} - w_*) &= (I - \eta \mathrm{X}^T \mathrm{X})(w_t - w_*) \\ &= (I - \eta \mathrm{X}^T \mathrm{X})^{t+1}(w_0 - w_*)\end{aligned}$$

Example: $\mathrm{X} = \begin{bmatrix} 10^{-3} & 0 \\ 0 & 1 \end{bmatrix}$ $\mathrm{y} = \begin{bmatrix} 10^{-3} \\ 1 \end{bmatrix}$ $w_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $w_* =$

Taylor Series Approximation

- Taylor series in one dimension:

$$f(x+\delta) = f(x) + f'(x)\delta + \tfrac{1}{2}f''(x)\delta^2 + \dots$$

- Newton's method:

Taylor Series Approximation

- Taylor series in **d** dimensions:

$$f(x+v) = f(x) + \nabla f(x)^T v + \tfrac{1}{2} v^T \nabla^2 f(x) v + \ldots$$

- Newton's method:

Newton's Method $f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

$$\nabla f(w) =$$

$$\nabla^2 f(w) =$$

$$v_t \text{ is solution to} : \nabla^2 f(w_t) v_t = -\nabla f(w_t)$$

$$w_{t+1} = w_t + \eta v_t$$

Newton's Method $f(w) = \frac{1}{2}||\mathrm{X}w - \mathrm{y}||_2^2$

$$\nabla f(w) = \mathrm{X}^T(\mathrm{X}w - \mathrm{y})$$

$$\nabla^2 f(w) = \mathrm{X}^T\mathrm{X}$$

$$v_t \text{ is solution to} : \nabla^2 f(w_t)v_t = -\nabla f(w_t)$$

$$w_{t+1} = w_t + \eta v_t$$

For quadratics, Newton's method converges in one step! (Not a surprise, why?)

$$w_1 = w_0 - \eta(\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T(\mathrm{X}w_0 - y) = w_*$$

General case

In general for Newton's method to achieve $f(w_t) - f(w_*) \leq \epsilon$:

So why are ML problems overwhelmingly solved by gradient methods?

Hint: $v_t \text{ is solution to} : \nabla^2 f(w_t) v_t = -\nabla f(w_t)$

General Convex case $f(w_t) - f(w_*) \leq \epsilon$

Clean converge nice proofs: Bubeck

Newton's method:

$$t \approx \log(\log(1/\epsilon))$$

Gradient descent:

- f is *smooth* and *strongly convex:* $aI \preceq \nabla^2 f(w) \preceq bI$
- f is *smooth*: $\nabla^2 f(w) \preceq bI$
- f is potentially non-differentiable: $||\nabla f(w)||_2 \leq c$

Nocedal +Wright, Bubeck

Other: BFGS, Heavy-ball, BCD, SVRG, ADAM, Adagrad,…

Revisiting… Logistic Regression

Machine Learning – CSE546
Kevin Jamieson
University of Washington

October 16, 2016

Loss function: Conditional Likelihood

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

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

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

$$\nabla f(w) =$$