

# CSC380: Principles of Data Science

## Linear Models

Xinchen Yu

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

# Outline

## ➤ Linear Regression

we will first focus on what is a linear function

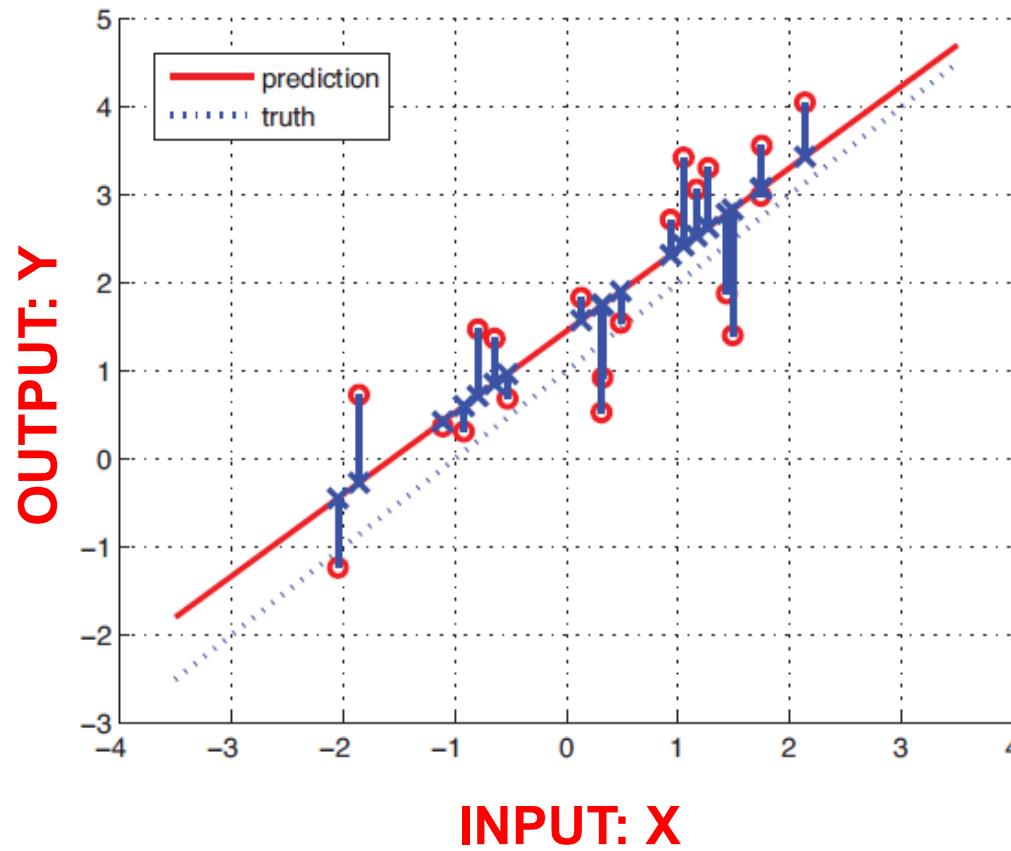
## ➤ Least Squares Estimation

then learn how to train a linear function from data

## ➤ Regularized Least Squares

## ➤ Logistic Regression

# Linear Regression



**Regression** Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs  $y$  are real-valued

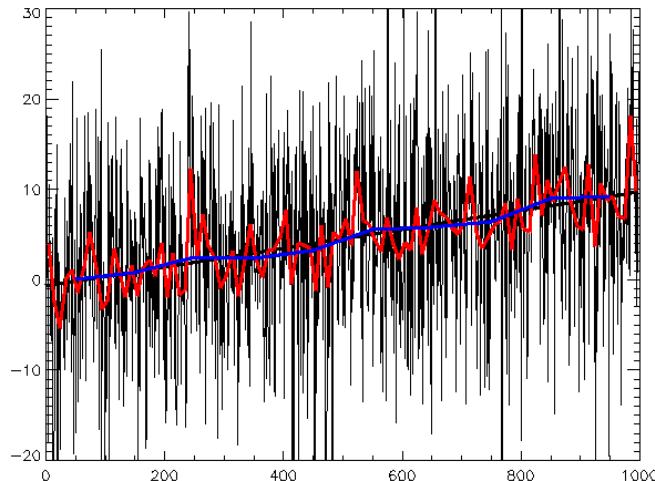
**Linear Regression** As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

$$w^T x := \sum_{d=1}^D w_d x_d$$

# Linear Regression

## When is linear regression useful?

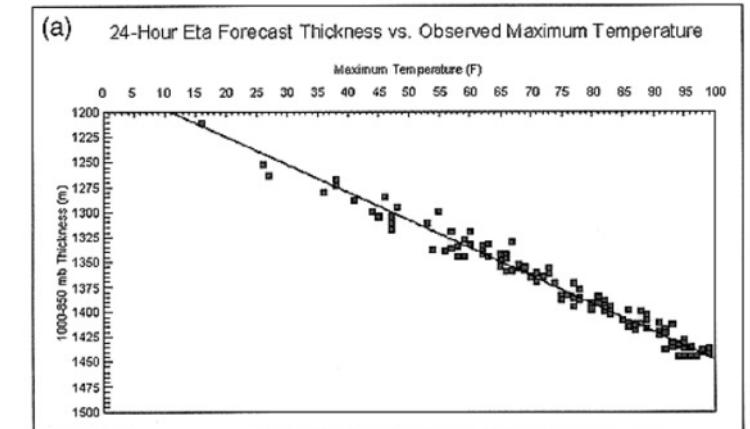
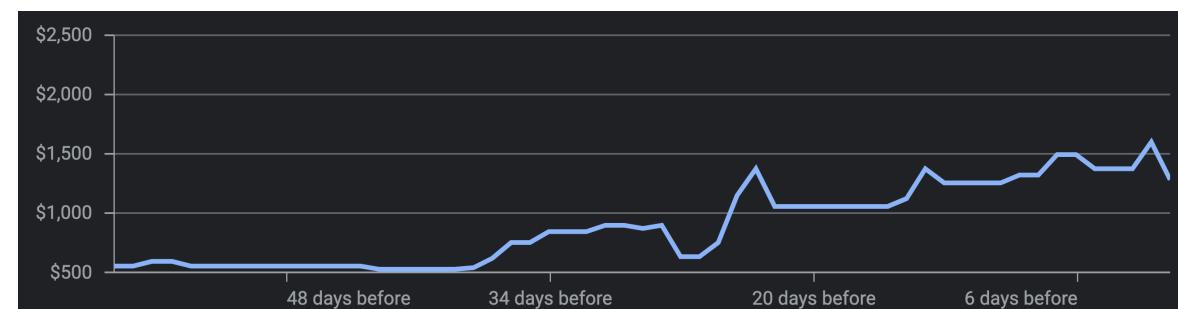


Trendlines

Price of an airline ticket:



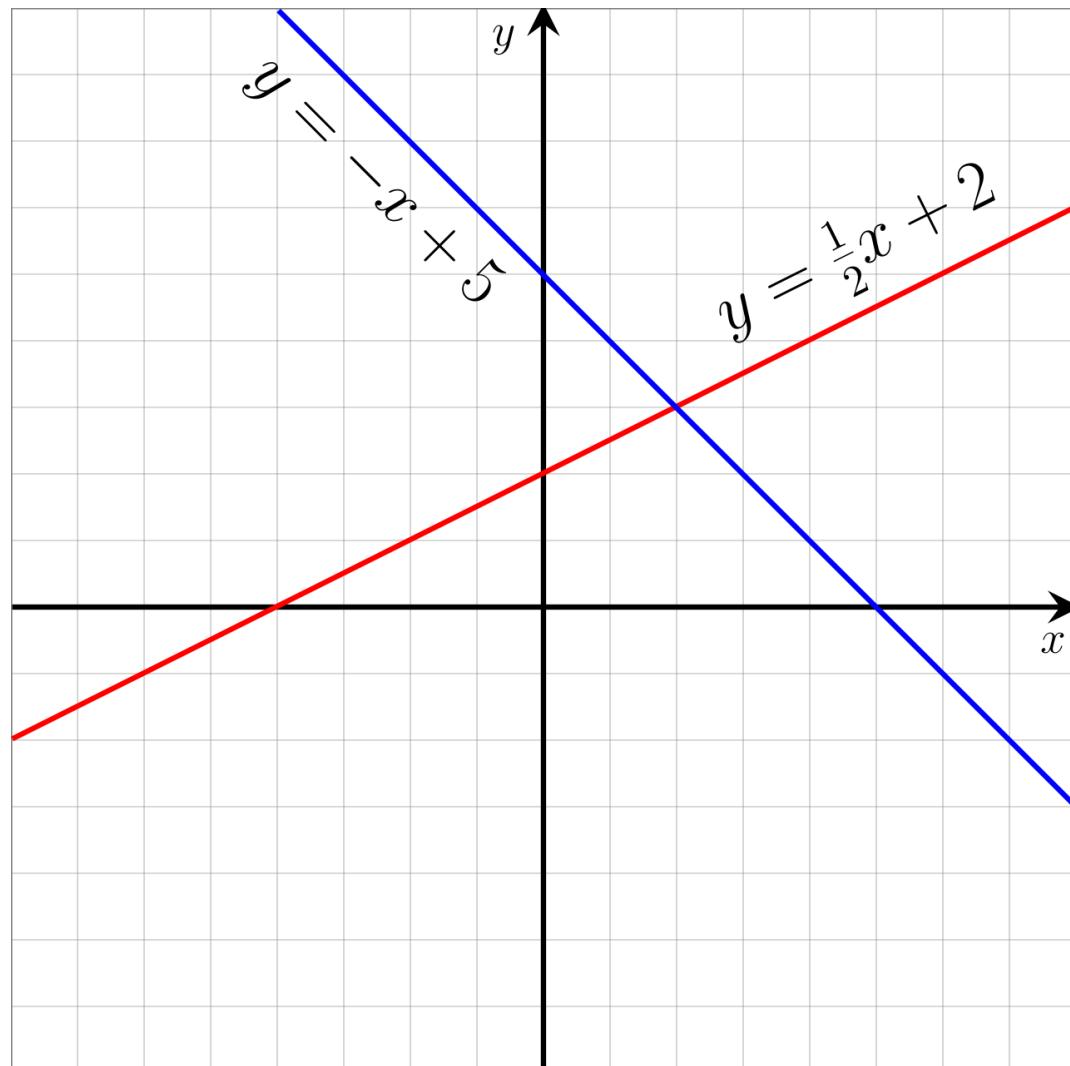
Stock Prediction



Climate Models  
Massie and Rose (1997)

*Used anywhere a linear relationship is assumed between inputs / (real-valued) outputs*

# Line Equation



Recall the equation for a line has a *slope* and an *intercept*,

$$y = w \cdot x + b$$

Slope      Intercept

- Intercept ( $b$ ) indicates where line crosses  $y$ -axis
- Slope controls angle of line
- Positive slope ( $w$ ) → Line goes up left-to-right
- Negative slope → Line goes down left-to-right

# Moving to higher dimensions...

7

- **1d regression**: regression with 1d input:

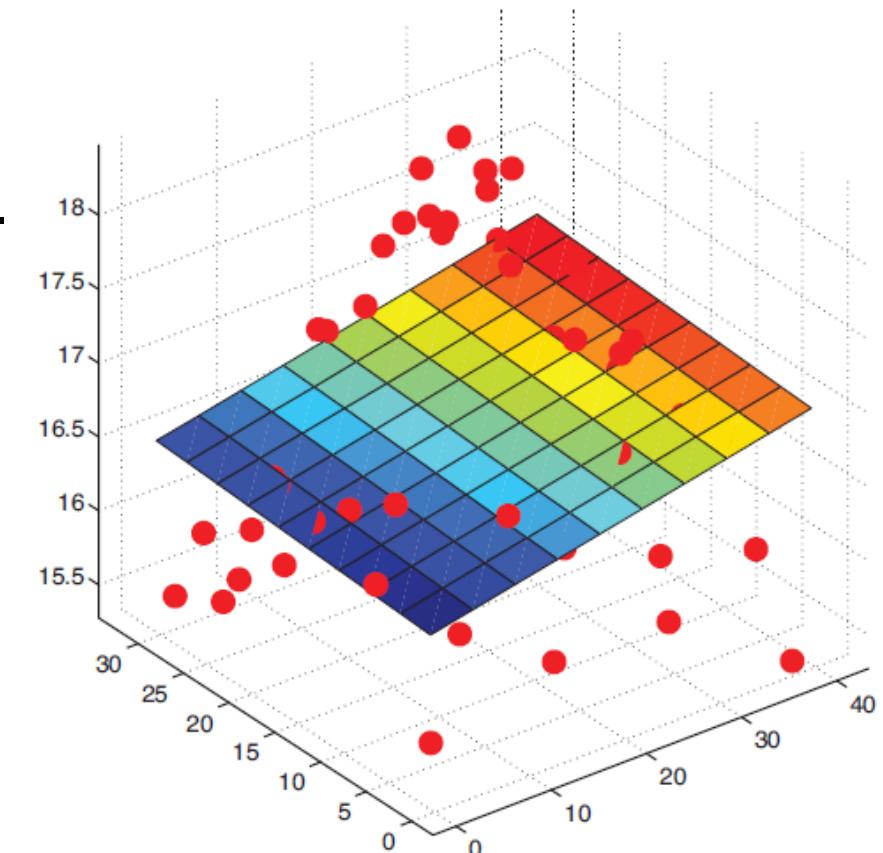
$$y = wx + b$$

- **D-dimensional regression**: input vector is  $x \in \mathbb{R}^D$ .

Recall the definition of an *inner product*:

$$w^T x = w_1 x_1 + w_2 x_2 + \dots + w_D x_D = \sum_{d=1}^D w_d x_d$$

The model is  $y = w^T x + b$



[ Image: Murphy, K. (2012) ]

# Moving to higher dimensions...

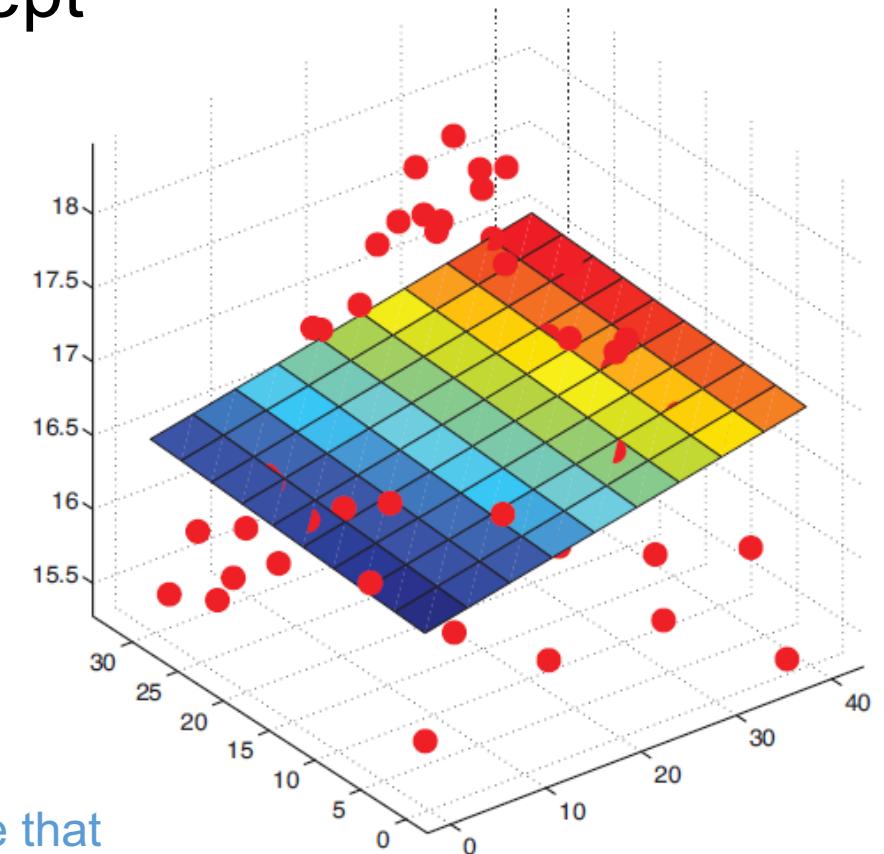
Often we simplify this by including the intercept into the weight vector,

$$\tilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_D \\ b \end{pmatrix} \quad \tilde{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_D \\ 1 \end{pmatrix} \quad y = \tilde{w}^T \tilde{x}$$

Since:

$$\begin{aligned} \tilde{w}^T \tilde{x} &= \sum_{d=1}^D w_d x_d + b \cdot 1 \\ &= w^T x + b \end{aligned}$$

from now on, we assume that  $w \in \mathbb{R}^D$  and  $x \in \mathbb{R}^D$  already has  $b$  and  $1$  in the last coordinate respectively.



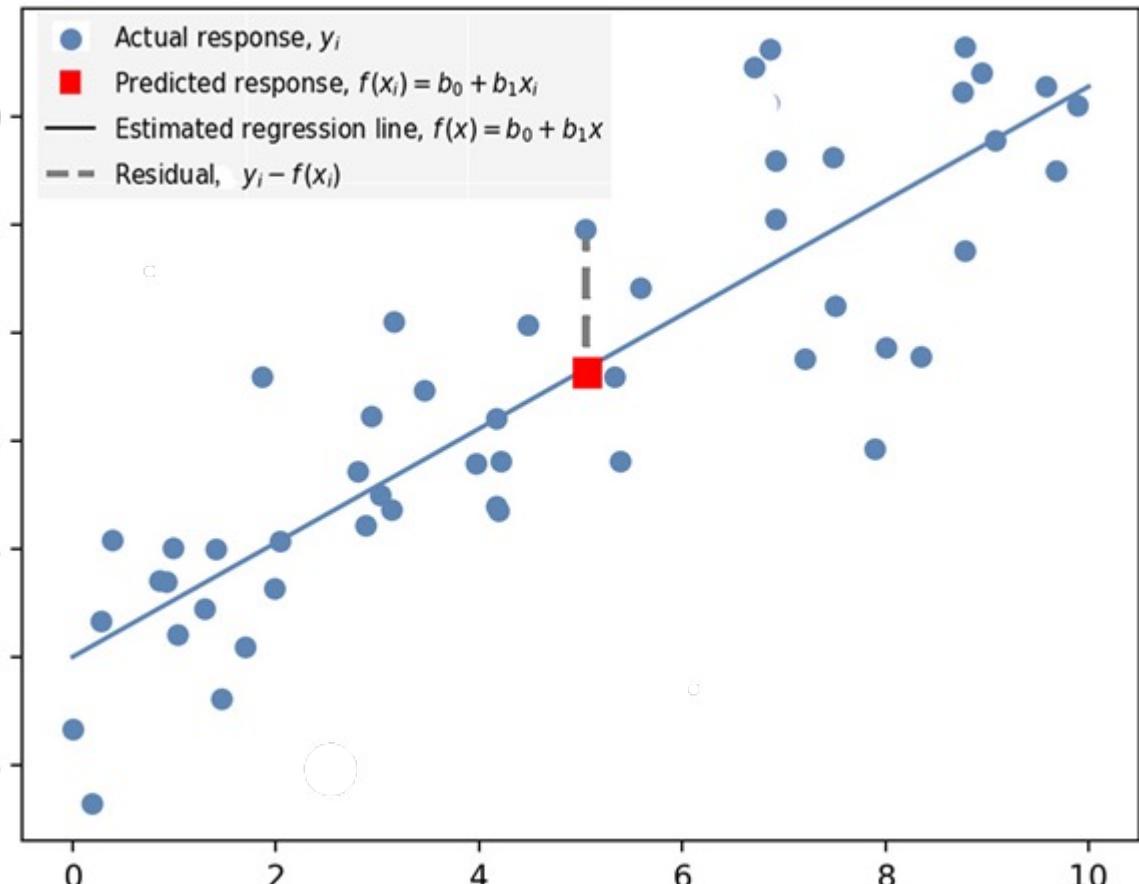
[ Image: Murphy, K. (2012) ]

**There are several ways to think about fitting regression:**

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

*They are all the same thing...*

# Fitting Linear Regression



**Intuition** Find a line that is as close as possible to every training data point

The distance from each point to the line is the **residual**

$$y - w^T x$$

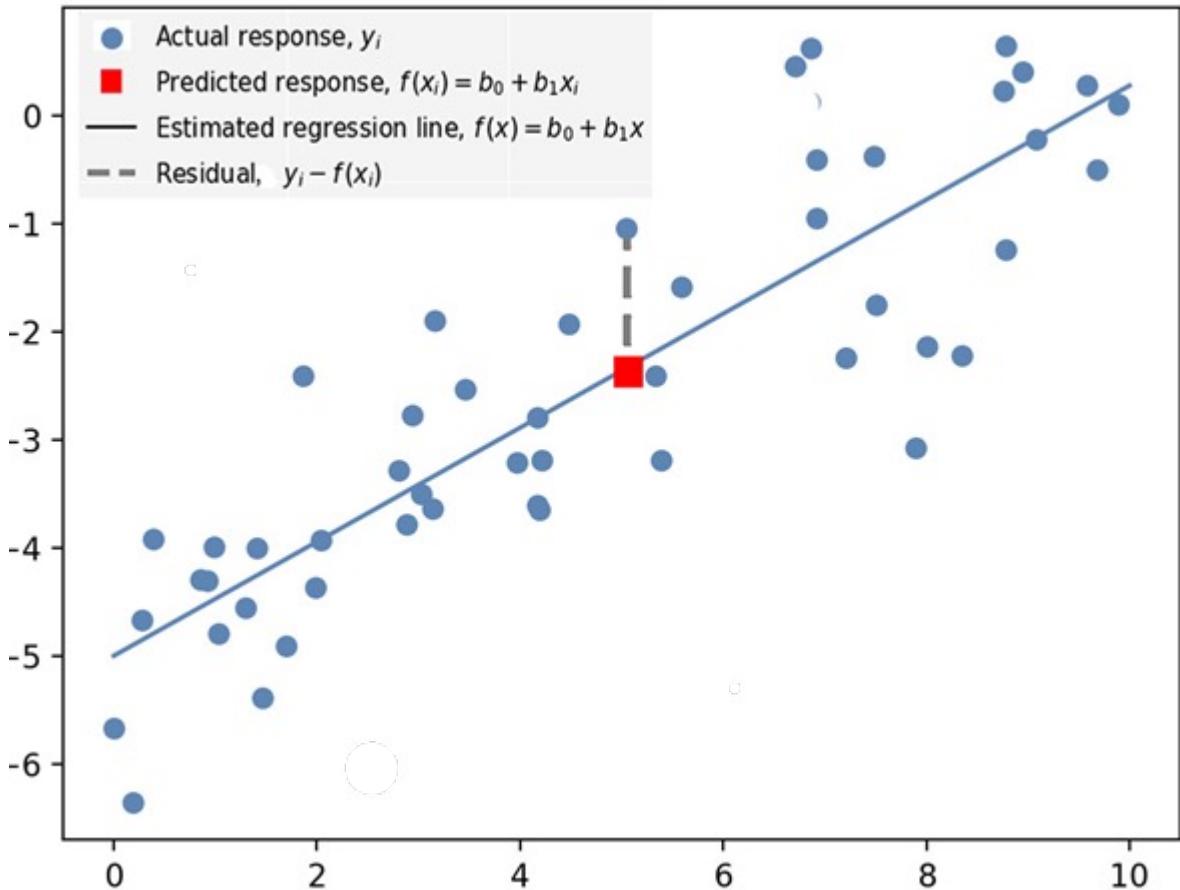
↑ Training Output      ↑ Prediction

*Let's find  $w$  that will minimize the residual!*

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

# Least Squares Solution

**Functional** Find a line that minimizes the sum of squared residuals!



Given:  $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$

Compute:

$$w^* = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2$$

*Least squares regression*

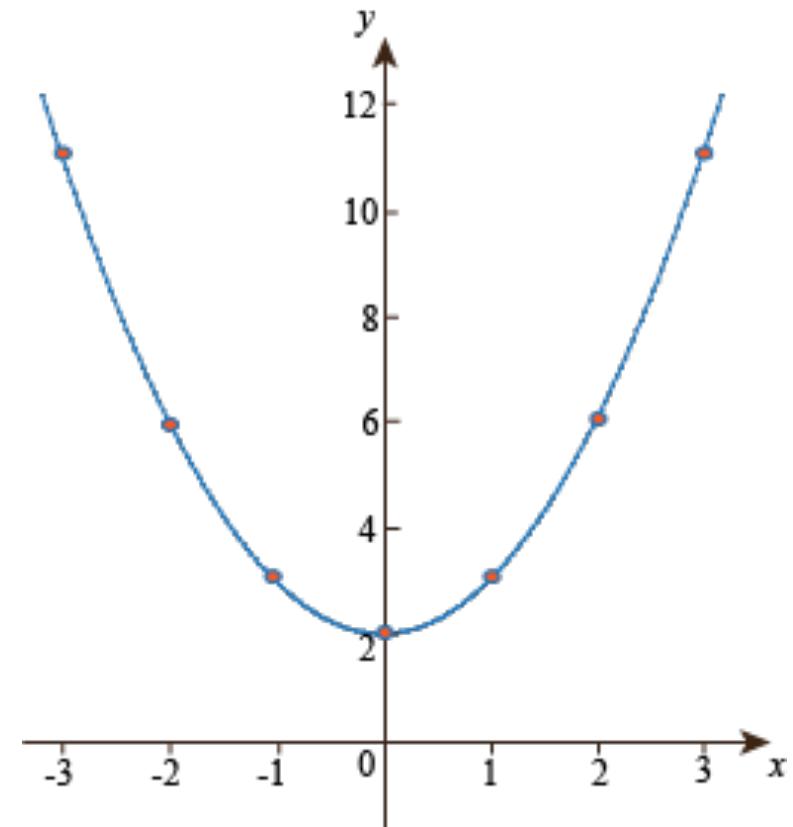
$$\min_w \sum_{i=1}^N (y^{(i)} - w^T x^{(i)})^2$$

**This is just a quadratic function...**

- Convex, unique minimum
- Minimum given by zero-derivative
- Can find a closed-form solution

Let's see for scalar case with no bias,

$$y = wx$$



# Least Squares : Simple Case

14

$$\frac{d}{dw} \sum_{i=1}^N (y^{(i)} - wx^{(i)})^2 =$$

**Derivative (+ chain rule)**

$$= \sum_{i=1}^N 2(y^{(i)} - wx^{(i)})(-x^{(i)}) = 0 \Rightarrow$$

**Distributive Property  
(and multiply -1 both sides)**

$$0 = \sum_{i=1}^N y^{(i)}x^{(i)} - w \sum_{j=1}^N (x^{(j)})^2$$

**Algebra**

$$w = \frac{\sum_i y^{(i)}x^{(i)}}{\sum_j (x^{(j)})^2}$$

# Least Squares in Higher Dimensions

15

Things are a bit more complicated in higher dimensions and involve more linear algebra,

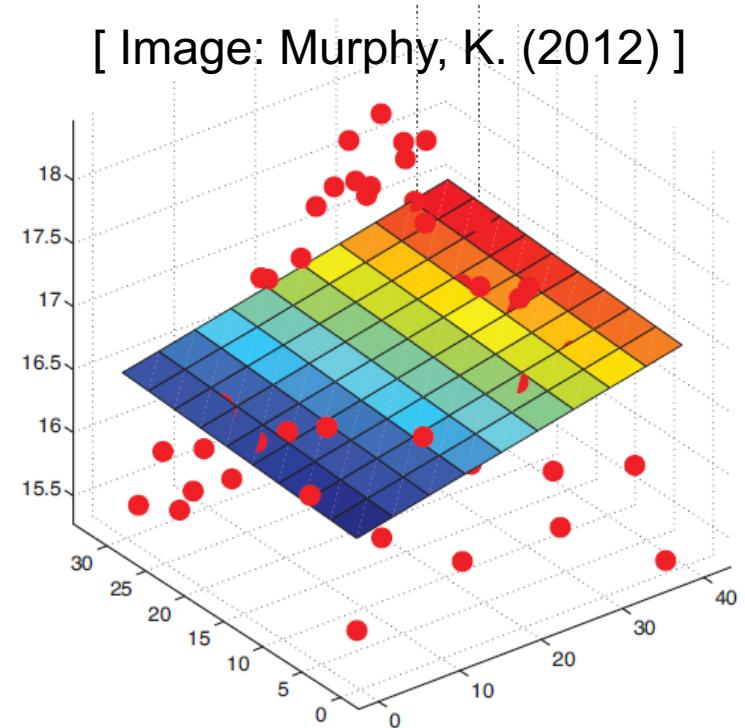
$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & \dots & x_D^{(1)} & 1 \\ x_1^{(2)} & \dots & x_D^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(m)} & \dots & x_D^{(m)} & 1 \end{pmatrix}$$

**Design Matrix**  
( each row is a data point)

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

**Vector of  
labels**

[ Image: Murphy, K. (2012) ]



Can write regression over *all training data* more compactly...

$$\mathbf{y} \approx \mathbf{X}\mathbf{w} \quad \xleftarrow{\text{mx1 Vector}}$$

$$= \begin{pmatrix} (x^{(1)})^\top \mathbf{w} \\ \dots \\ (x^{(m)})^\top \mathbf{w} \end{pmatrix}$$

# Least Squares in Higher Dimensions

16

Least squares can also be written more compactly,

$$\min_w \sum_{i=1}^N (y^{(i)} - w^T x^{(i)})^2 = \|\mathbf{y} - \mathbf{X}w\|^2$$

Some slightly more advanced linear algebra gives us a solution,

$$w = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \left( \sum_j x^{(j)} (x^{(j)})^T \right)^{-1} \sum_i y^{(i)} x^{(i)}$$

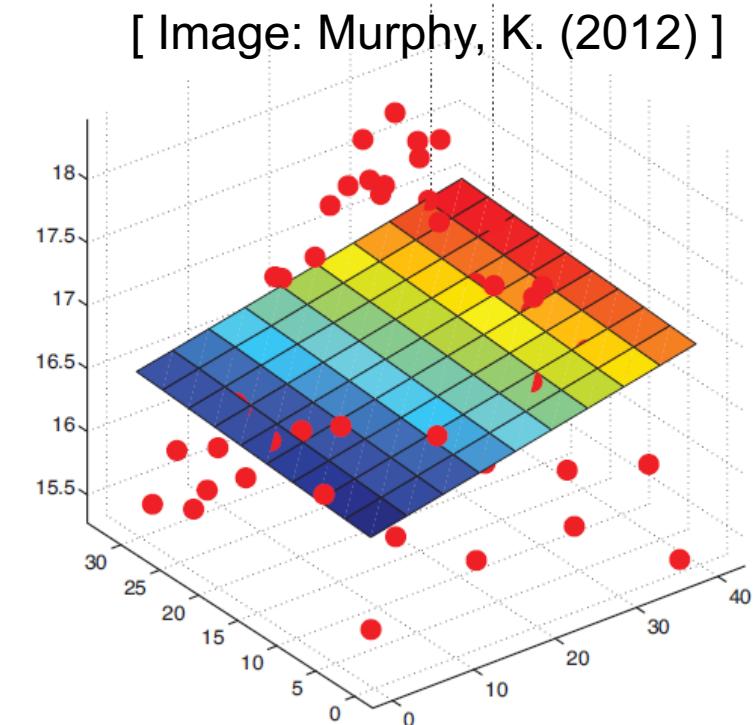
compare with the 1d version:  $w = \frac{\sum_i y^{(i)} x^{(i)}}{\sum_j (x^{(j)})^2}$

## Ordinary Least Squares (OLS) solution

Derivation a bit advanced for this class, but enough to know

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from.

[ Image: Murphy, K. (2012) ]



# Least Squares Implementation in Numpy

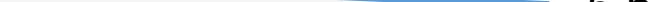
17

```
import numpy as np  
import numpy.linalg as la
```

```
y = np.array([5,4,3,2,1])
```

`m,D = X_orig.shape` ← .shape returns the dimensions of the array

```
X = np.hstack((X_orig, np.ones((m,1))))
```



✓ 0.2s

```
w_hat = la.inv(X.T@X) @ (X.T@y)
```

✓ 0.2s

```
array([-0.5,  0.,  5.5])
```

- .T for transpose
  - la.inv() is the inversion operator
  - @ operator for matrix-matrix, matrix-vector, or vector-vector product.

**There are several ways to think about fitting regression:**

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

*They are all the same thing...*

**There are several ways to think about fitting regression:**

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

*They are all the same thing...*

# Probabilistic Assumptions

20

- Assume  $x \sim \mathcal{D}_X$  from some distribution. We then assume that

$$y = \mathbf{w}^T \mathbf{x} + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Equivalently,

$$p(y|x; w) = \mathcal{N}(\mathbf{w}^T \mathbf{x}, \sigma^2)$$

**Why?** Adding a constant to a Normal RV is still a Normal RV,

$$z \sim \mathcal{N}(m, P)$$

$$z + c \sim \mathcal{N}(m + c, P)$$

for our case, linear regression  $z \leftarrow \epsilon$       and  $c \leftarrow \mathbf{w}^T \mathbf{x}$

# MLE for Linear Regression

21

Given training data  $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ , maximize the likelihood!

$$\hat{w} = \arg \max_w \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}; w)$$

$$= \arg \max_w \log \prod_{i=1}^m p(x^{(i)}) p(y^{(i)}|x^{(i)}; w)$$

$$= \arg \max_w \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; w)$$

note  $p(x^{(i)})$  does not depend on  $w$ !

subtracting a constant w.r.t.  $w$  does not affect the solution  $w$ !

note model assumption!  $p(y|x; w) = \mathcal{N}(w^T x, \sigma^2)$

# Univariate Gaussian (Normal) Distribution

22

Let's focus on 1d case.

Let  $\mu = w^T x$  for now.

**Gaussian** (a.k.a. Normal) distribution with mean (location)  $\mu$  and variance (squared scale)  $\sigma^2$  parameters,

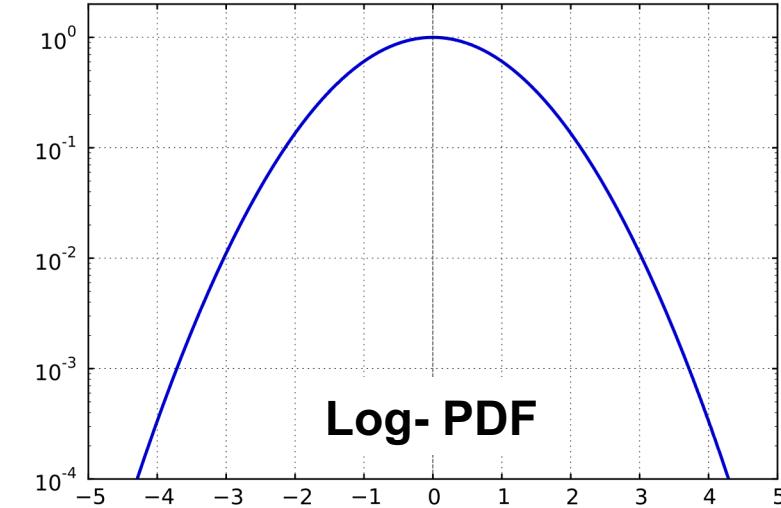
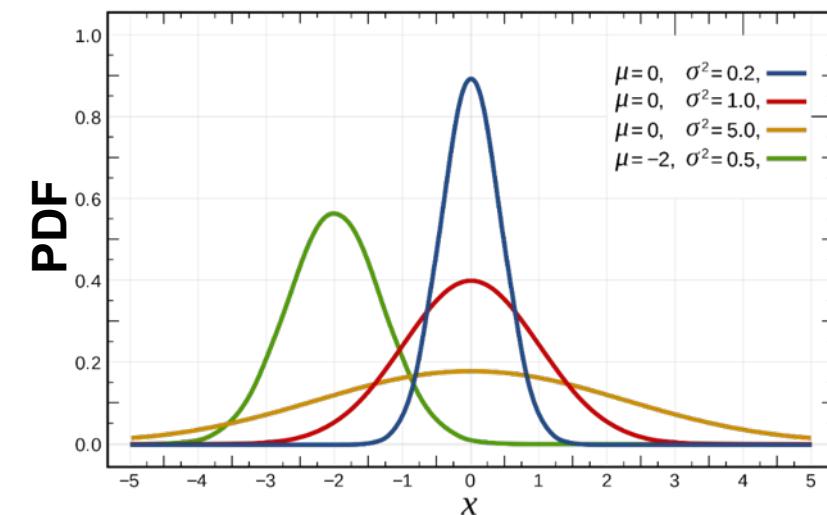
$$\mathcal{N}(y; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}(y - \mu)^2/\sigma^2\right)$$

The logarithm of the PDF is just a negative quadratic,

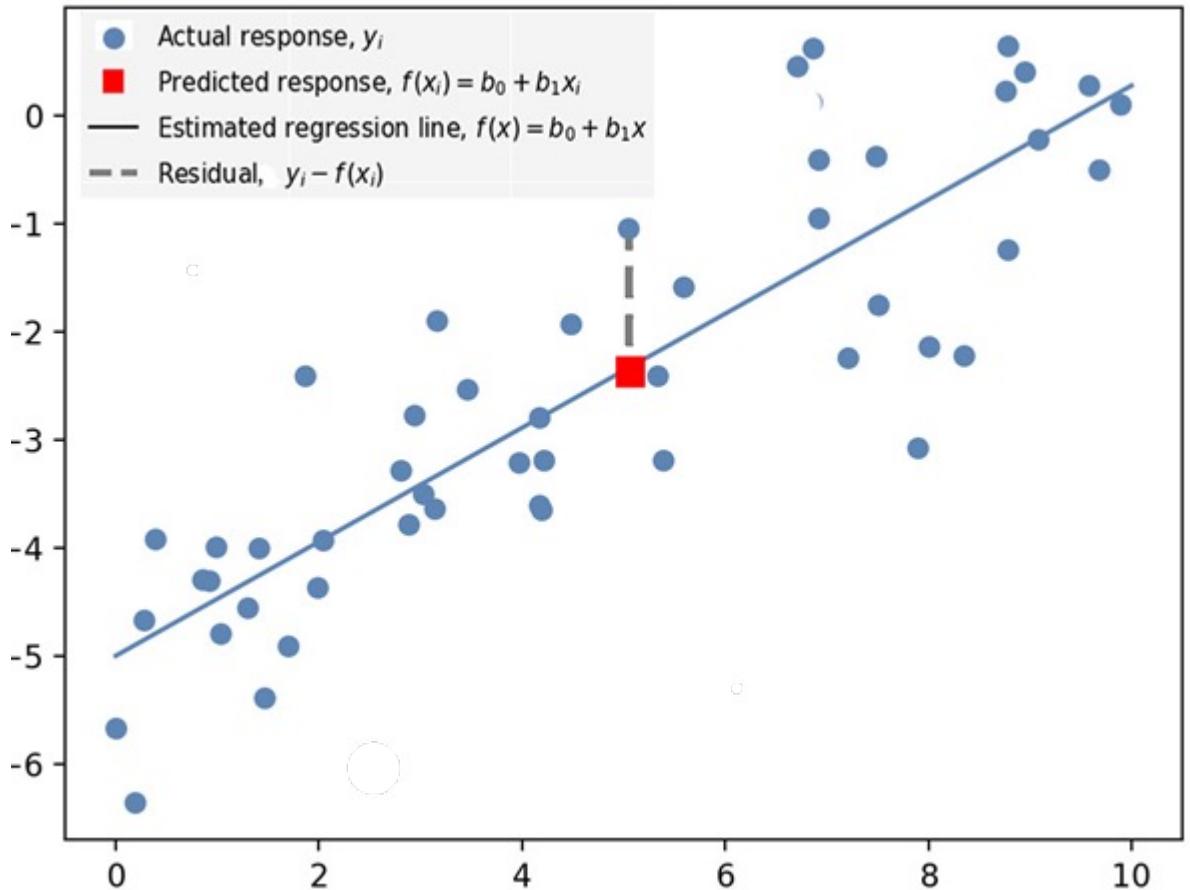
$$\log \mathcal{N}(y; \mu, \sigma^2) = -\frac{1}{2} \log 2\pi - \log \sigma - \frac{1}{2\sigma^2}(y - \mu)^2$$

Constant w.r.t. mean

Quadratic Function of mean



# MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2$$

So for Linear Regression,  
MLE = Least Squares Estimation

# Linear Regression Summary

1. The linear regression model (assumption),

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

2. For N iid training data fit using least squares,

$$w^{\text{OLS}} = \arg \min_w \sum_{i=1}^N (y^{(i)} - w^T x^{(i)})^2$$

3. Equivalent to maximum likelihood solution

# A word on matrix inverses...

$$w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Least squares solution requires inversion of the term,

$$(\mathbf{X}^T \mathbf{X})^{-1}$$

*What are some issues with this?*

1. Requires  $\mathcal{O}(D^3)$  time for D input features
2. May be numerically unstable (or even non-invertible)

$$(x + \epsilon)^{-1} = \frac{1}{x + \epsilon} \quad \rightarrow$$

if  $x$  is small, then small numerical errors in input can lead to large errors in solution

$$w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Use *Moore-Penrose pseudoinverse* ('dagger' notation)

$$w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^\dagger \mathbf{X}^T \mathbf{y}$$

- Generalization of the standard matrix inverse for noninvertible matrices.
- Directly computable in most libraries
- In Numpy it is: `linalg.pinv`

# Linear Regression in Scikit-Learn

27

Load your libraries,

For Evaluation

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```



Load data,

```
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

^: same as diabetes\_X[:,[2]]

Train / Test Split:

```
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

Samples total	442
Dimensionality	10
Features	real, -.2 < x < .2
Targets	integer 25 - 346

```
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

Train (fit) and predict,

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

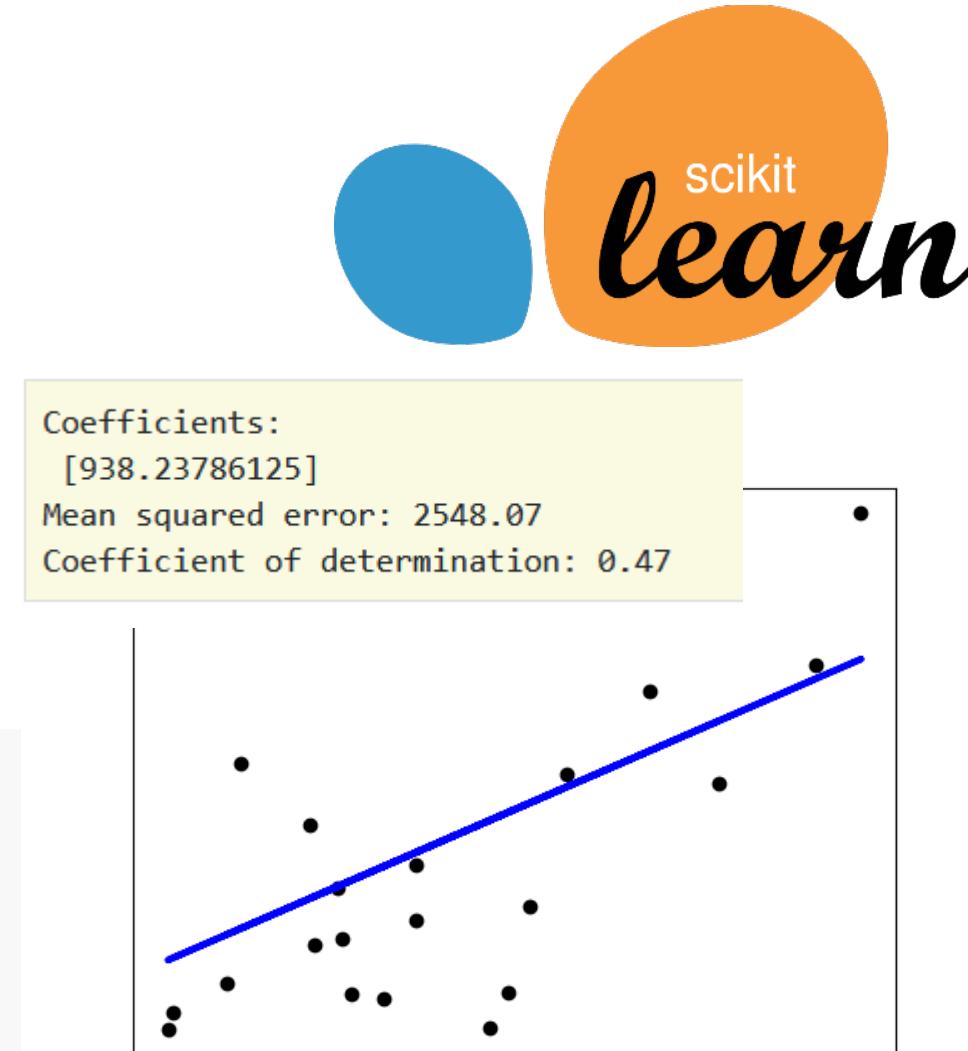
Plot regression line with the test set,

```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

regr.coef\_ : coefficient (array)  
regr.intercept\_ : intercept (float)



- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

# Alternatives to Ordinary Least Squares (OLS)

30

Recall: OLS solution may not

$$w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Use *Moore-Penrose pseudoinverse* ('dagger' notation)

$$w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^\dagger \mathbf{X}^T \mathbf{y}$$

Or, use L2 Regularized Least Squares (RLS)

$$w^{\text{L2}} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

why is this called regularized least squares?

# Regularization

$$w^{\text{L2}} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

Turns out,  $w^{\text{L2}}$  is the solution of

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

recall:  $\|w\| = \sqrt{\sum_{d=1}^D w_d^2}$

$\lambda$ : Regularization Strength
 $\|w\|^2$ : Regularization Penalty

Prefers smaller magnitudes for  $w$ !

$\lambda$  very small: almost OLS

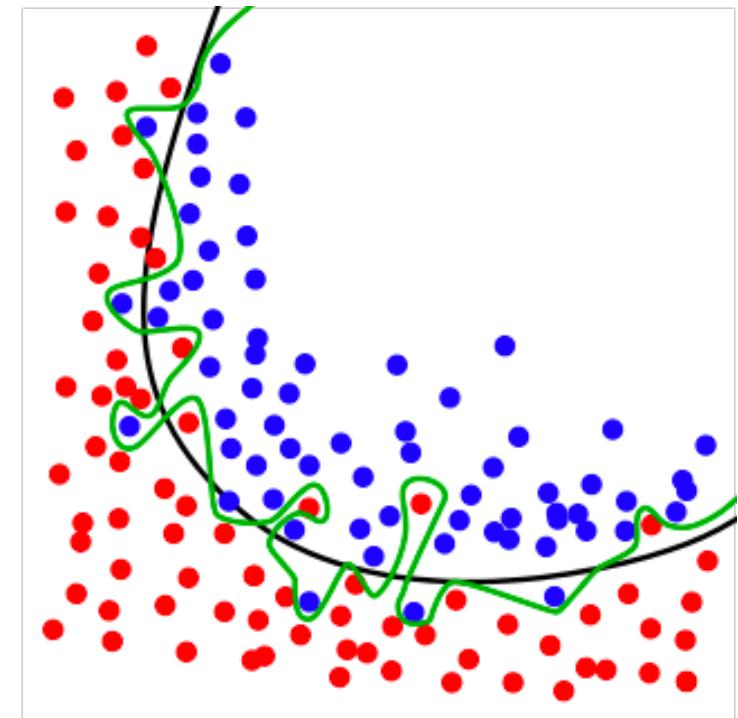
$\lambda$  very large:  $w \approx 0$  and high trainset error

Okay, we have a training data. Why not learn the most complex function that can work flawlessly for the training data and be done with it? (i.e., classifies every data point correctly)

**Extreme example:** Let's memorize the data. To predict an unseen data, just follow the label of the closest memorized data.

Doesn't generalize to unseen data – called *overfitting* the training data.

**Solution:** Fit the train set but don't "over-do" it. This is called **regularization**.



green: almost memorization  
black: true decision boundary

# Regularization

$$w^{\text{RLS}} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

Turns out,  $w^{\text{RLS}}$  is the solution of

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

recall:  $\|w\| = \sqrt{\sum_{d=1}^D w_d^2}$

$\lambda$ : Regularization Strength
 $\|w\|^2$ : Regularization Penalty

In short, the benefits of L2-RLS

- No need to worry about the estimator being undefined (due to matrix inversion)
- Avoid overfitting (if  $\lambda$  is chosen well)!

# Notes on L2 Regularization

34

- Feature weights are “shrunk” towards zero – statisticians often call this a “shrinkage” method
- Common practice: Do **not** penalize bias ( $y$ -intercept,  $w_D$ ) parameter,

$$\min_w \sum_i (y^{(i)} - w^T x^{(i)})^2 + \frac{\lambda}{2} \sum_{d=1}^{D-1} w_d^2$$

Recall: we enforced  
 $x_D^{(i)} = 1$  so that  $w_D$  encodes the intercept

- Penalizing intercept will make solution depend on origin for  $\mathbf{Y}$ .  
i.e., add a constant  $c$  to  $y^{(i)}$ 's  $\Rightarrow$  the solutions changes!
- Typically, we standardize (e.g. Z-score) features before fitting model ( Sklearn StandardScaler )
  - Why? If we multiply 0.1 to feature  $d$ ,  $w_d$  needs to be much larger to compensate for it. But the regularization would prevent that!

As a general rule, if you are unsure, do standardization for every ML algorithm anyways

## sklearn.linear\_model.Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True, max_iter=None, tol=0.001, solver='auto', positive=False, random_state=None) \[source\]
```

Minimizes the objective function:

$$\|y - Xw\|^2_2 + \alpha * \|w\|^2_2$$

Alpha is what we have been calling  $\lambda$

**alpha : {float, ndarray of shape (n\_targets,)}, default=1.0**

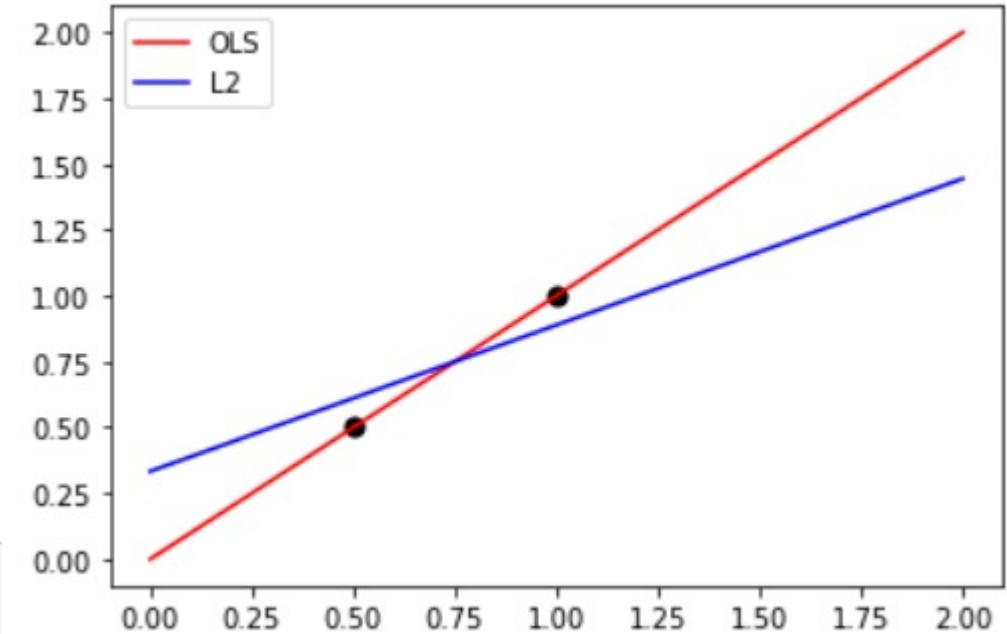
Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to  $1 / (2c)$  in other linear models such as `LogisticRegression` or `LinearSVC`. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

Define and fit OLS and L2 regression,

```
ols=linear_model.LinearRegression()  
ols.fit(X_train, y_train)  
ridge=linear_model.Ridge(alpha=0.1)  
ridge.fit(X_train, y_train)
```

Plot results,

```
fig, ax = plt.subplots()  
ax.scatter(X_train, y_train, s=50, c="black", marker="o")  
ax.plot(X_test, ols.predict(X_test), color="red", label="OLS")  
ax.plot(X_test, ridge.predict(X_test), color="blue", label="L2")  
  
plt.legend()  
plt.show()
```



quiz candidate

Q: why L2 has a lower slope?

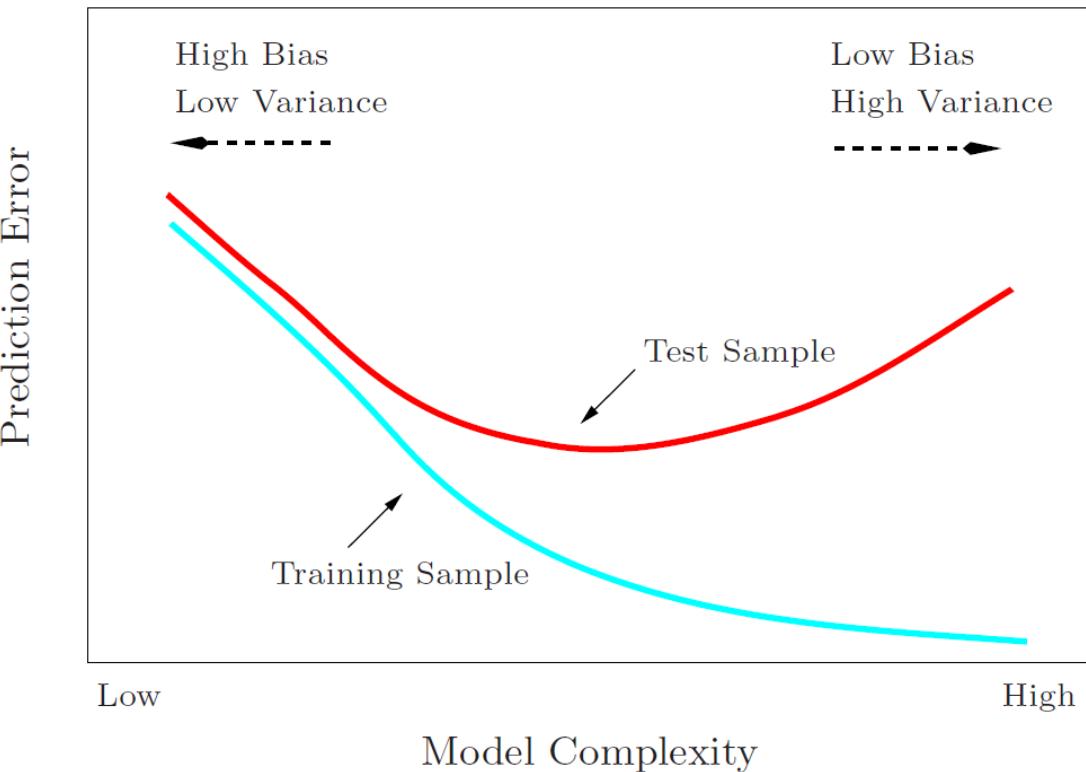
*L2 (Ridge) reduces impact of any single data point*

# Choosing Regularization Strength

37

We need to tune regularization strength to avoid over/under fitting...

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$



## Recall bias/variance tradeoff

High regularization reduces model complexity: *increases bias / decreases variance*

Q: How should we properly tune  $\lambda$  ?

cross validation!

# CSC380: Principles of Data Science

## Linear Models 2

Xinchen Yu

# Regularized Least Squares

Ordinary least-squares (OLS) estimation (no regularizer),

$$w^{\text{OLS}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2$$

L2 norm:  $\|w\| = \sqrt{\sum_{d=1}^D w_d^2}$

L1 norm:  $\|w\|_1 = \sum_{d=1}^D |w_d|$

## L2-regularized Least-Squares (Ridge)

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

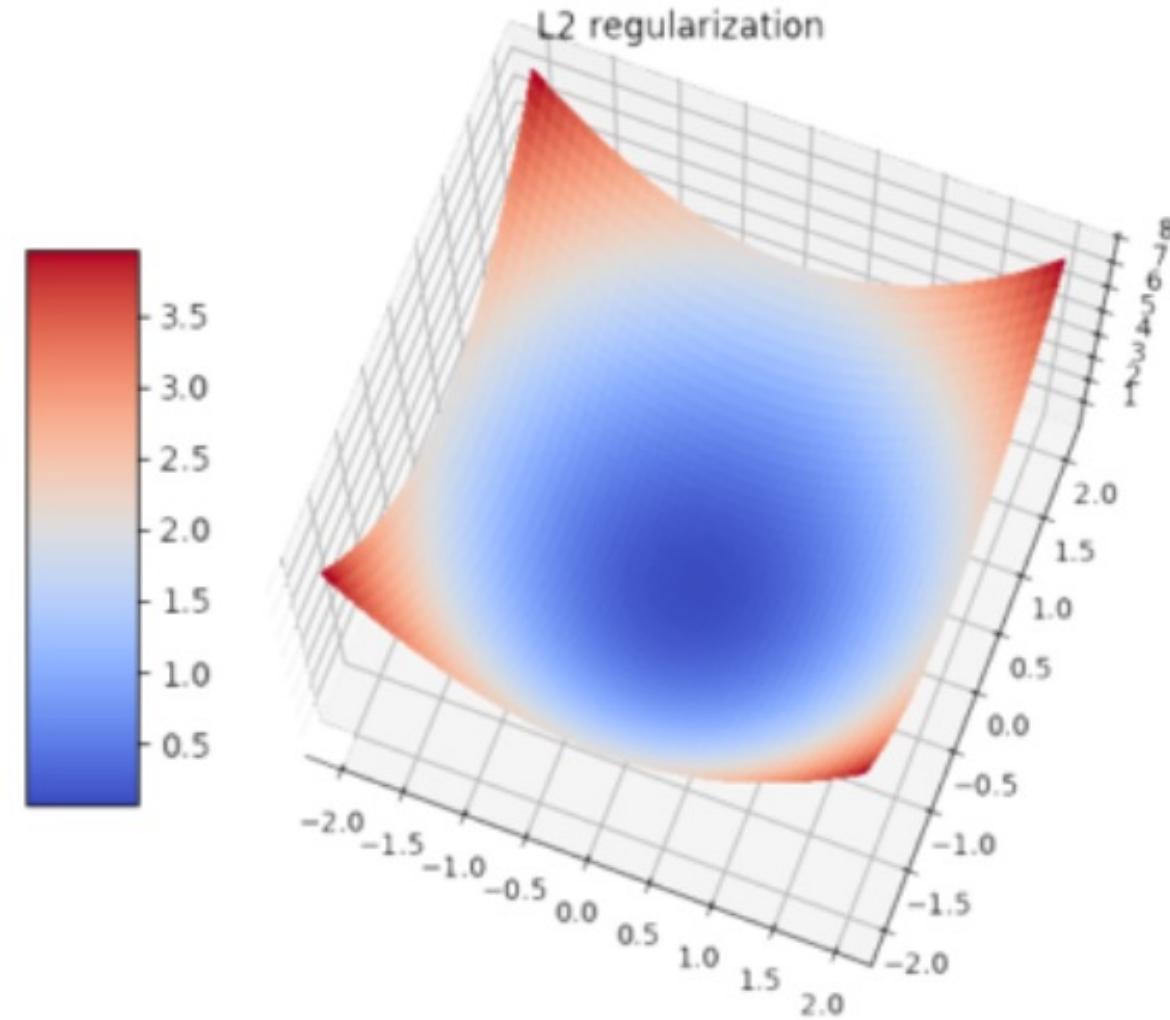
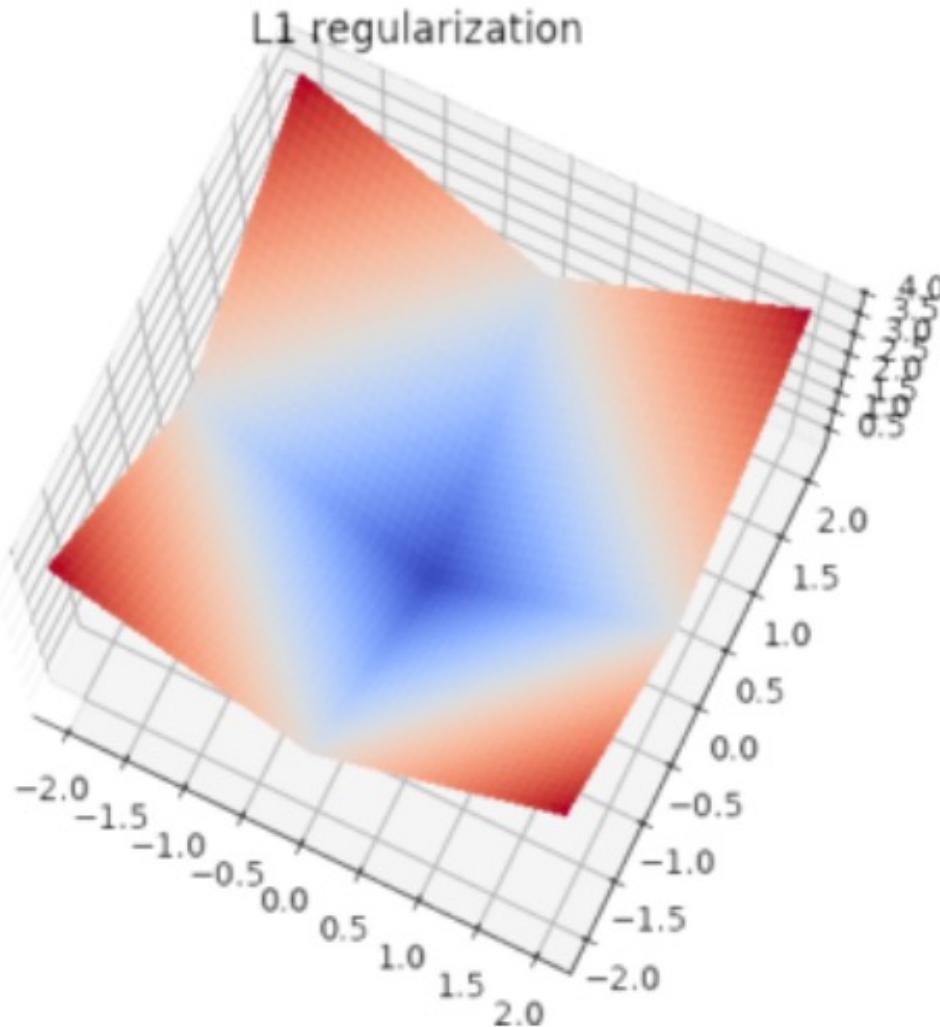
Convention: Just saying 'RLS' means L2-RLS

## L1-regularized Least-Squares (LASSO)

LASSO: Least Absolute Shrinkage and Selection Operator

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|_1$$

# L1 vs L2

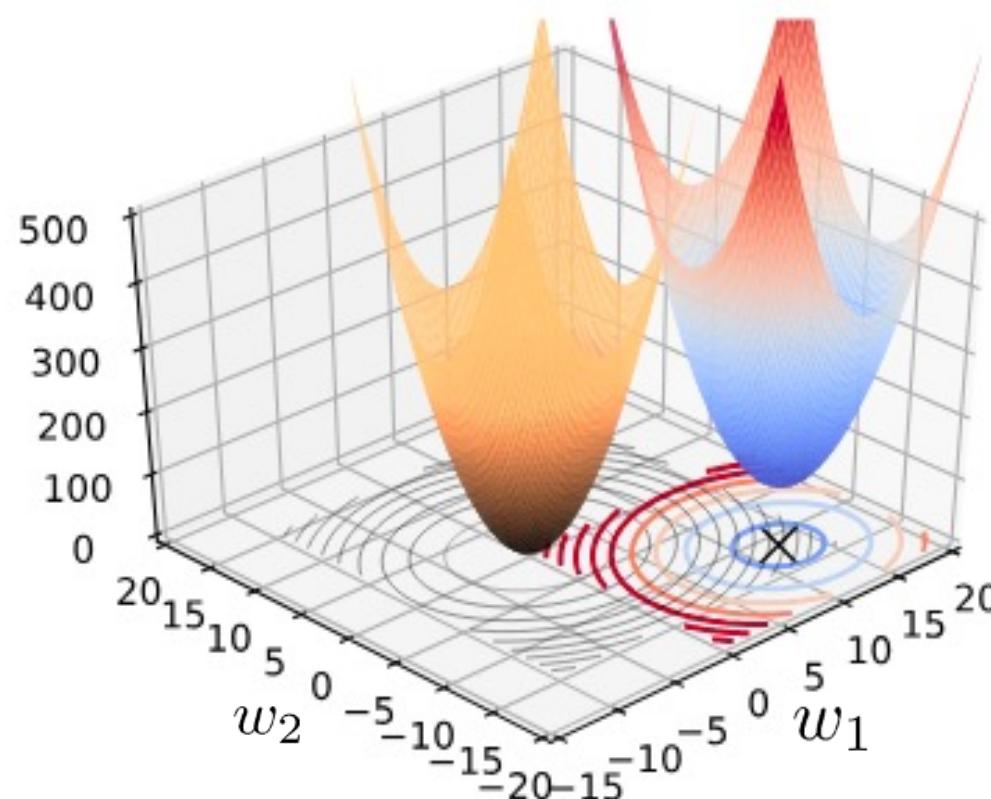


Looks subtle but L2-RLS and L1-RLS are often quite different!

# L2 Constrained Optimization Viewpoint

Orange: non-regularized loss function  $\sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2$

Blue-red: penalty  $\|w\|^2$



# Constrained Optimization Viewpoint

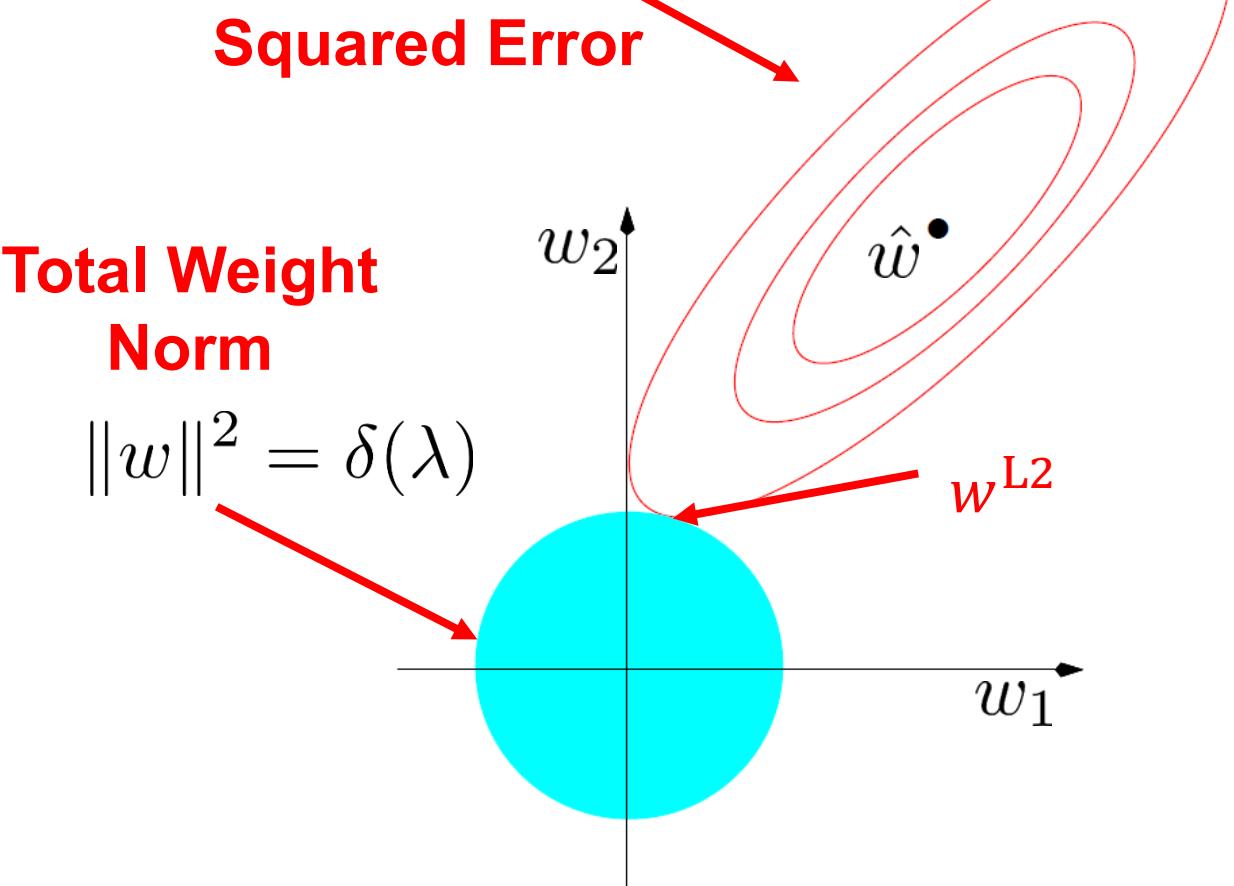
(Theorem) If

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

then there exists a function  $\delta(\lambda)$  s.t.

$$\begin{aligned} w^{\text{L2}} &= \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 \\ \text{subject to } \|w\|^2 &\leq \delta(\lambda) \end{aligned}$$

$$\hat{w} = \arg \min_w \sum_i (y^{(i)} - w^T x^{(i)})^2$$

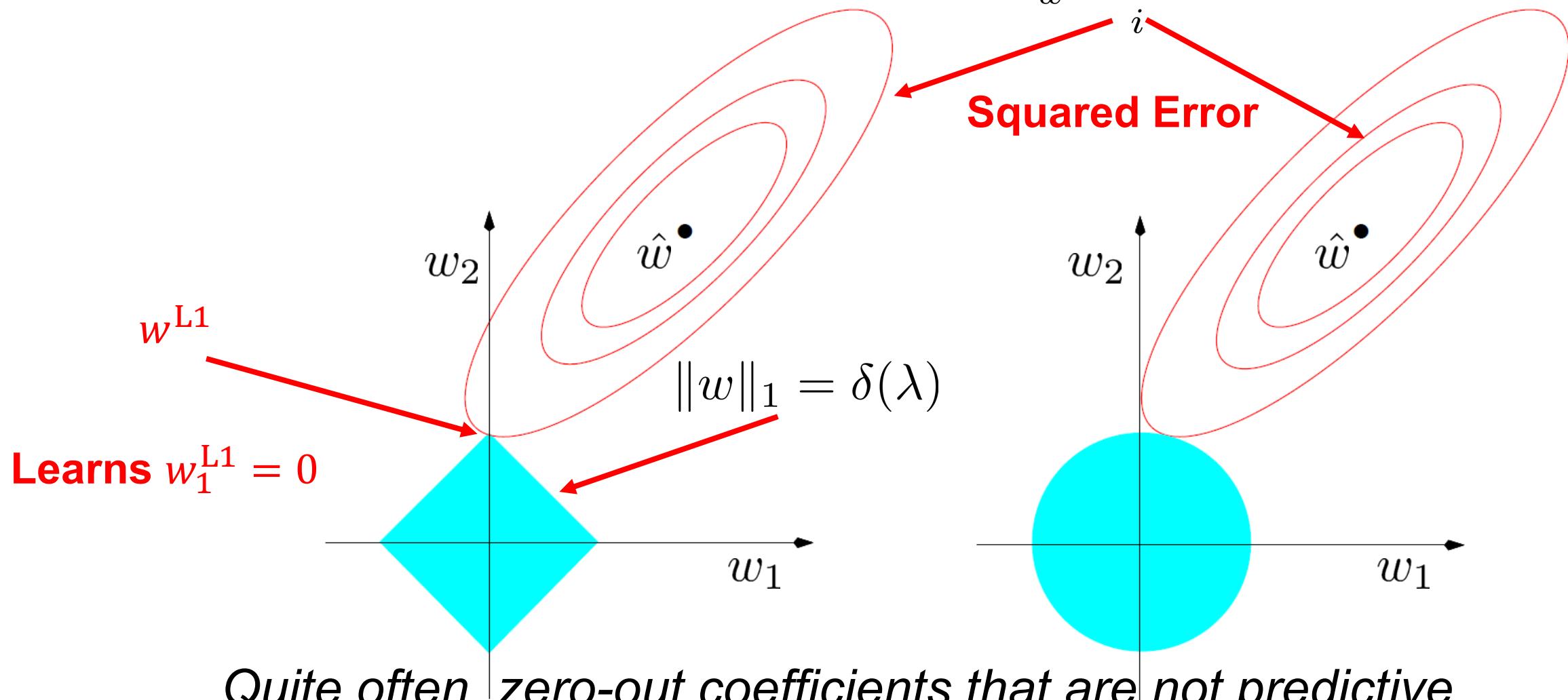


[ Source: Hastie et al. (2001) ]

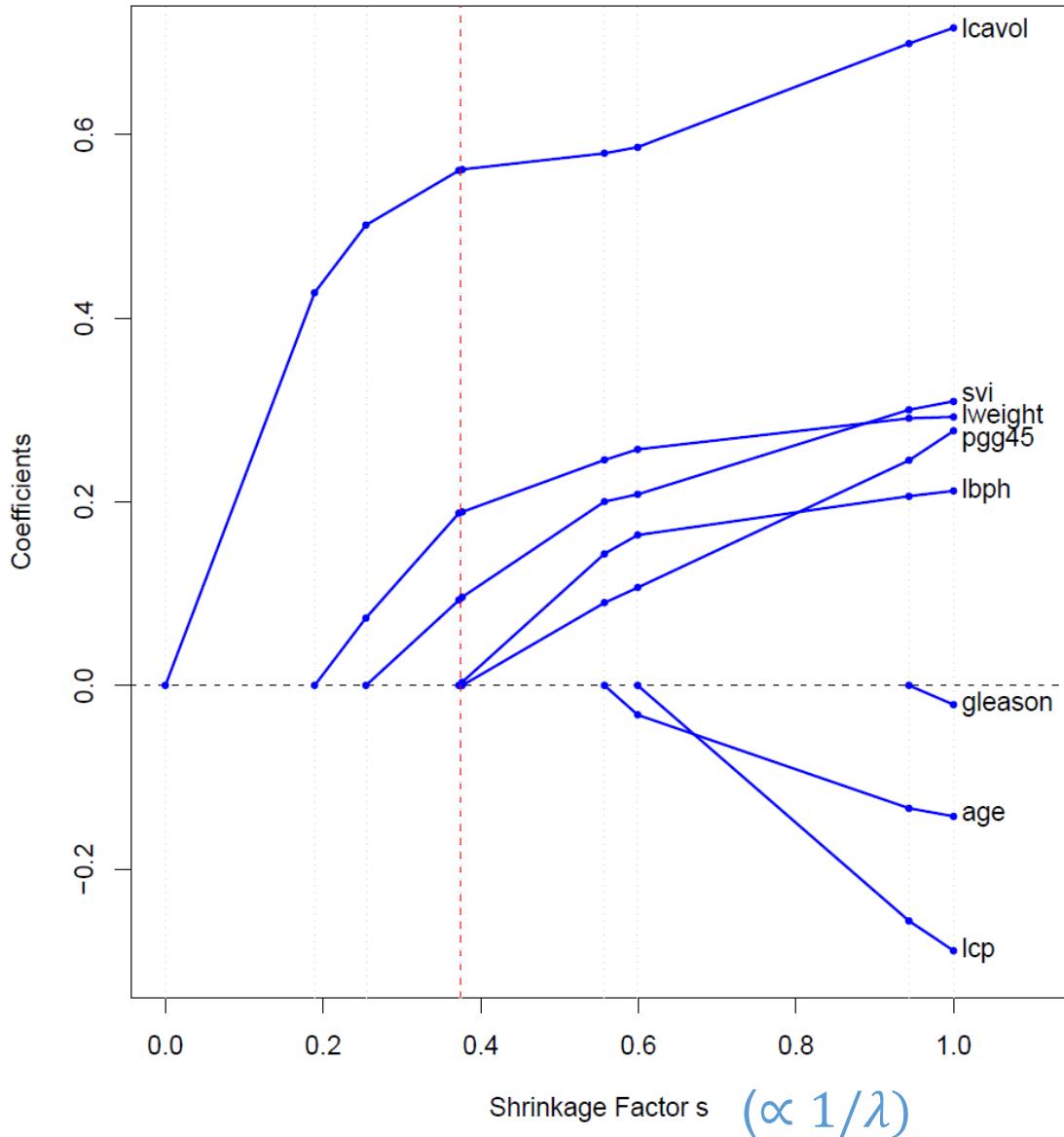
# L1 Regularized Least-Squares

43

$$\hat{w} = \arg \min_w \sum (y^{(i)} - w^T x^{(i)})^2$$



# Feature Weight Profiles

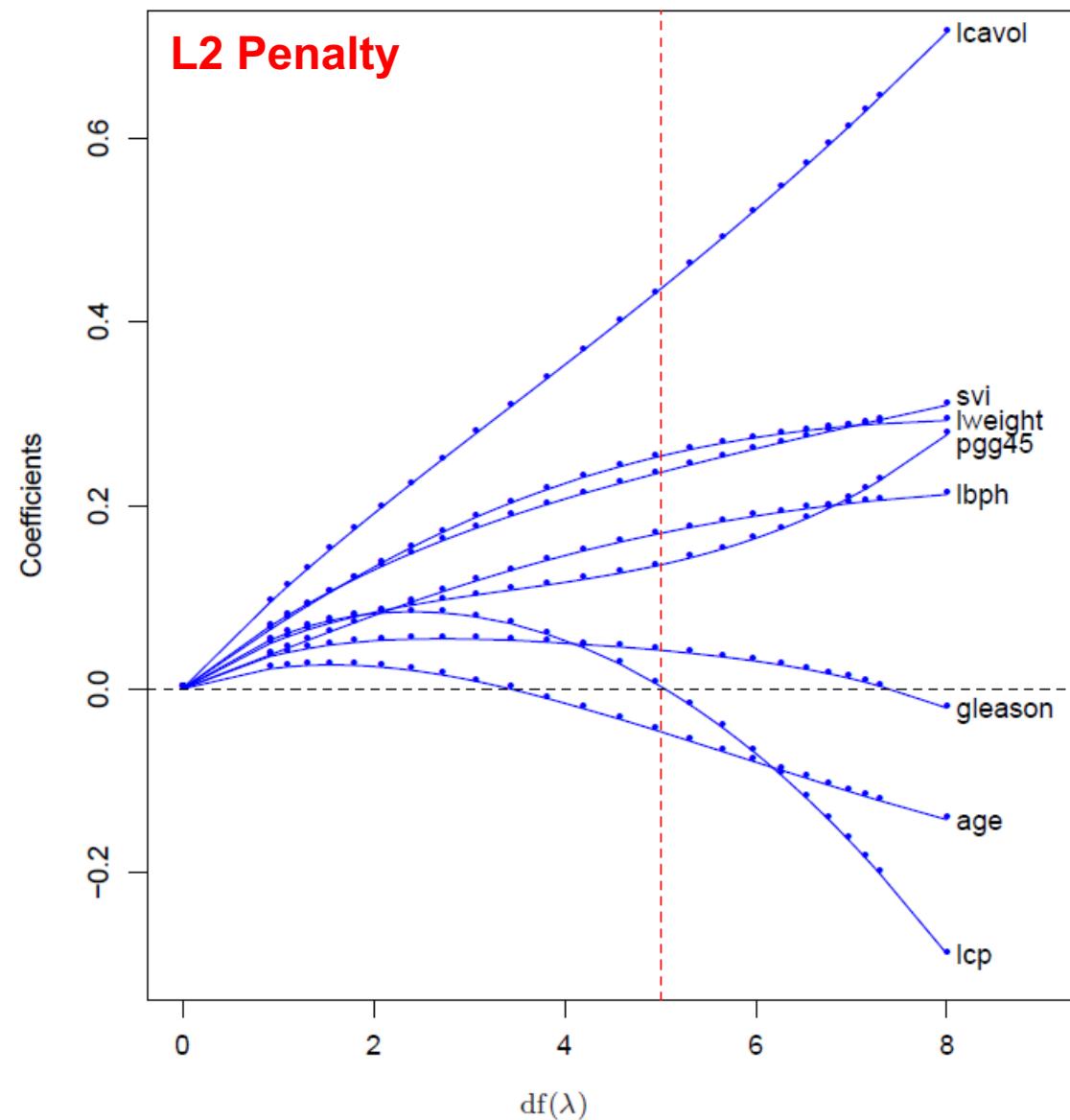
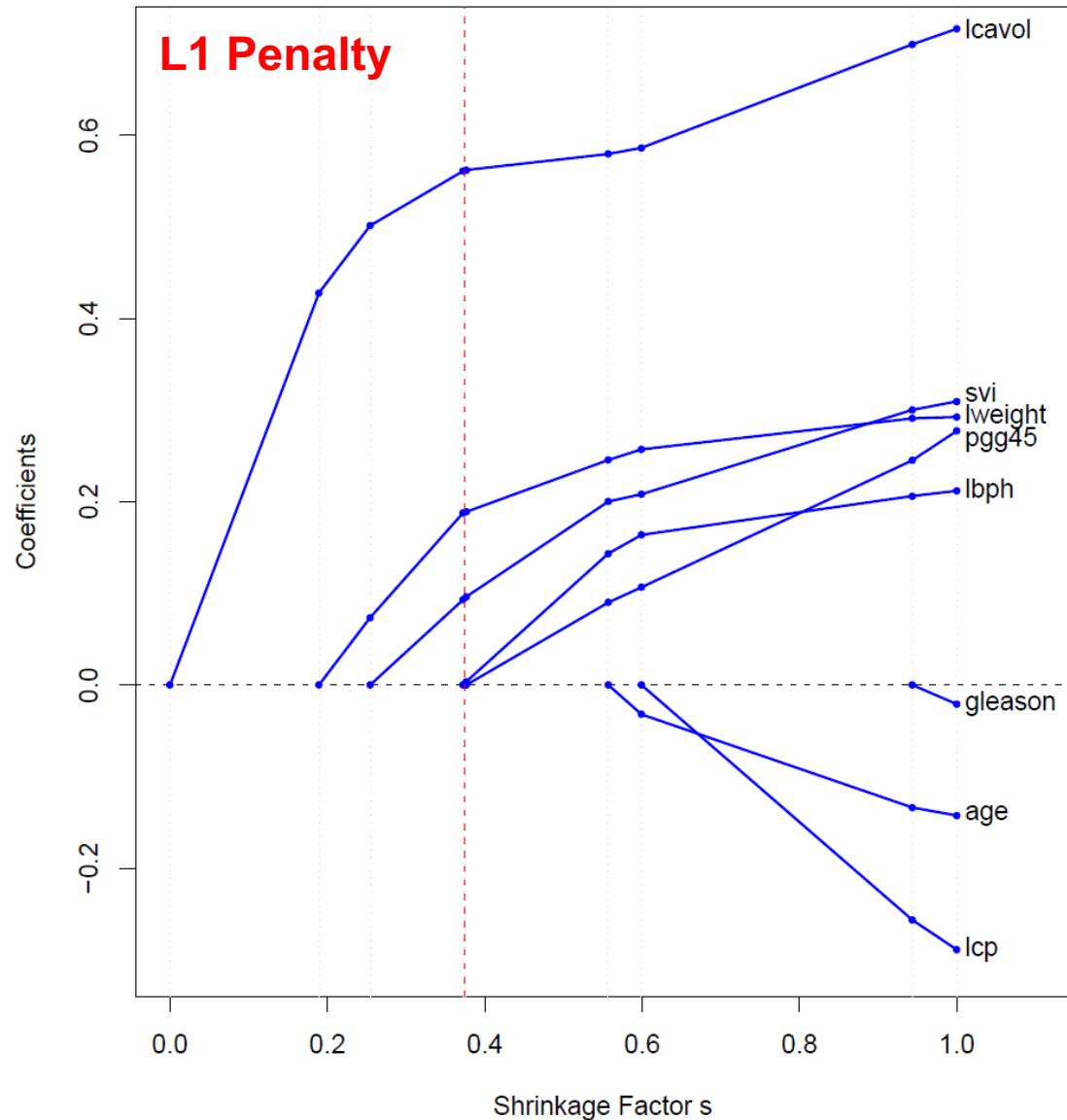


Varying regularization parameter  
adjusts *shrinkage factor*

For moderate regularization  
strength weights for many features  
go to zero

- Induces *feature sparsity*
- Ideal for high-dimensional settings since it reduced variance from having too many features!
- Gracefully handles  $D > m$  case, for  $D$  features and  $m$  training data

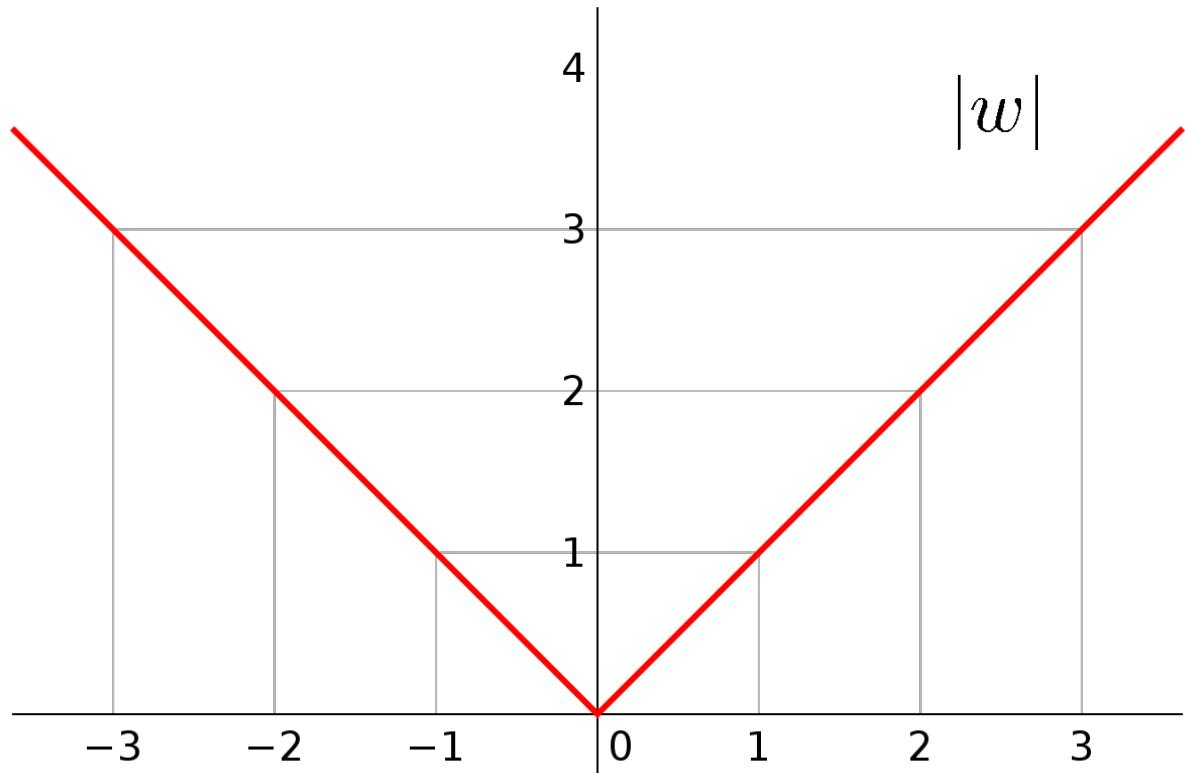
# Feature Weight Profiles



# Learning L1 Regularized Least-Squares

46

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|_1$$



Not differentiable...

$$\frac{d}{dx}|x|$$

...doesn't exist at  $x=0$

Can't set derivatives to zero as  
in the L2 case!

- **Not differentiable**, no closed-form solution. => Need to use iterative methods
- But it is **convex**!
  - Global minimum can be found!
  - Efficient optimization algorithms exist
- *Least Angle Regression* (LAR) computes full solution path for a range of values  $\lambda$

## sklearn.linear\_model.Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize='deprecated', precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic') ¶
```

[\[source\]](#)

**Parameters:** **alpha : float, default=1.0**  $\leftarrow$  this is  $\lambda$

Constant that multiplies the L1 term. Defaults to 1.0. `alpha = 0` is equivalent to an ordinary least square, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Given this, you should use the `LinearRegression` object.

**fit\_intercept : bool, default=True**

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

**precompute : 'auto', bool or array-like of shape (n\_features, n\_features), precompute**

Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always `False` to preserve sparsity.

**copy\_X : bool, default=True**

If `True`,  $X$  will be copied; else, it may be overwritten.

# Specialized methods for cross-validation...

## `sklearn.linear_model.LassoCV`

```
class sklearn.linear_model.LassoCV(*, eps=0.001, n_alphas=100, alphas=None, fit_intercept=True, normalize='deprecated',  
    precompute='auto', max_iter=1000, tol=0.0001, copy_X=True, cv=None, verbose=False, n_jobs=None, positive=False,  
    random_state=None, selection='cyclic')
```

[\[source\]](#)

Tries out a range of  $\alpha$  values and reports the best, but maintains other values of  $\alpha$  as well.

# L1 Regression Cross-Validation

50

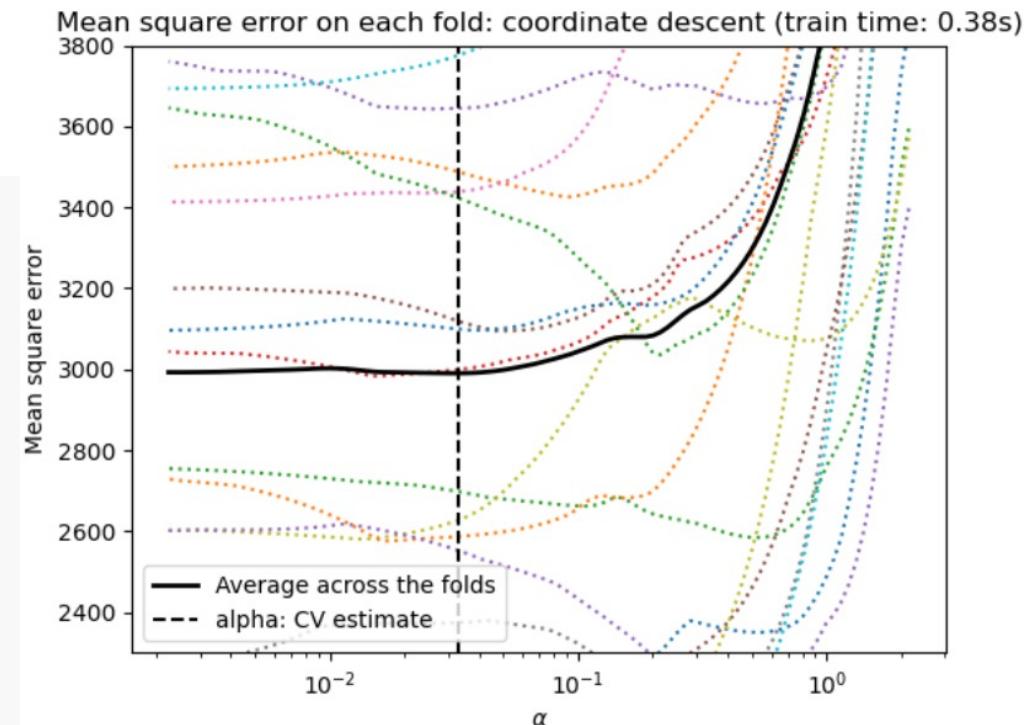
Perform L1 Least Squares (LASSO) 20-fold cross-validation,

```
model = LassoCV(cv=20).fit(X, y)      or
```

Plot the error for range of alphas,

```
plt.figure()
ymin, ymax = 2300, 3800
plt.semilogx(model.alphas_ + EPSILON, model.mse_path_, ":")
plt.plot(
    model.alphas_ + EPSILON,
    model.mse_path_.mean(axis=-1),
    "k",
    label="Average across the folds",
    linewidth=2,
)
plt.axvline(
    model.alpha_ + EPSILON, linestyle="--", color="k", label="alpha: CV estimate"
)
```

all these colored dotted lines for each test fold  
all alphas\_  
adds vertical line  
the best alpha



# Example: Prostate Cancer Dataset

51

Term	LS	Ridge	Lasso
Intercept	2.465	2.452	2.468
lcavol	0.680	0.420	0.533
lweight	0.263	0.238	0.169
age	-0.141	-0.046	
lbph	0.210	0.162	0.002
svi	0.305	0.227	0.094
lcp	-0.288	0.000	
gleason	-0.021	0.040	
pgg45	0.267	0.133	

**Task:** predict logarithm of prostate specific antigen (PSA).

Best LASSO model learns to ignore several features (age, lcp, gleason, pgg45).

Wait...Is **age** really not a significant predictor of prostate cancer? What's going on here?

Age is highly correlated with other factors and thus *not significant* in the presence of those factors

- **A big open problem in ML:** being able to throw in all the possible features, but still perform as good as knowing the truly relevant features ahead of time (i.e., not affected by irrelevant features)
  - Recall irrelevant features can be harmful.

The optimal strategy for  $p$  features looks at models over *all possible combinations of features*,

```
For k in 1,...,p:  
    subset = Compute all subset of k-features (p-choose-k)  
    For kfeat in subset:  
        model = Train model on kfeat features  
        score = Evaluate model using cross-validation  
    Choose the model with best cross-validation score
```

## Time complexity

- Data have 8 features, there are 8-choose-k subsets for each  $k=1,\dots,8$  for a total of 255 models
- Using 10-fold cross-val requires  $10 \times 255 = 2,550$  training runs!
- In general,  $O(2^p)$  time complexity

... who can afford exponential time complexity?

Best subset works well!  
reasonably good test error, low standard deviation, and only  
based on two features!

Term	LS	Best Subset	Ridge	Lasso
Intercept	2.465	2.477	2.452	2.468
lcavol	0.680	0.740	0.420	0.533
lweight	0.263	0.316	0.238	0.169
age	-0.141		-0.046	
lbph	0.210		0.162	0.002
svi	0.305		0.227	0.094
lcp	-0.288		0.000	
gleason	-0.021		0.040	
pgg45	0.267		0.133	
Test Error	0.521	0.492	0.492	0.479
Std Error	0.179	0.143	0.165	0.164

# Forward Sequential Selection

An efficient method adds the most predictive feature one-by-one

```
featSel = empty
featUnsel = All features
For iter in 1,...,p:
    For kfeat in featUnsel:
        thisFeat = featSel + kfeat
        model = Train model on thisFeat features
        score = Evaluate model using cross-validation
        featSel = featSel + best scoring feature
        featUnsel = featUnsel - best scoring feature
Choose the model with best cross-validation score
```

Backwards approach starts with *all* features and removes one-by-one

```
featSel = All features  
For iter in 1,...,p:  
    For kfeat in featSel:  
        thisFeat = featSel - kfeat  
        model = Train model on thisFeat features  
        score = Evaluate model using cross-validation  
        featSel = featSel - worst scoring feature  
Choose the model with best cross-validation score
```

# Backward Sequential Selection

58

```
featSel = All features  
For iter in 1,...,p:  
    For kfeat in featSel:  
        thisFeat = featSel - kfeat  
        model = Train model on thisFeat features  
        score = Evaluate model using cross-validation  
        featSel = featSel - worst scoring feature  
Choose the model with best cross-validation score
```

1

Start with original feature set of size  $n=29$ 

2

Iteration 1. Generate all possible feature subsets of size  $n - 1=28$ .

Subset 1



Subset 2



Subset 28



Remove the feature that is absent from the subset with the highest evaluation score.

3

Iteration 2. (Suppose Subset 2 corresponded to the highest evaluation score.)  
Generate all possible feature subsets of size  $(n - 1) - 1=27$ .Repeat steps ③ and ④ until the feature subset contains only single feature.  
(Given an initial feature set of size  $n$ , there are  $n-1$  iterations in total.)

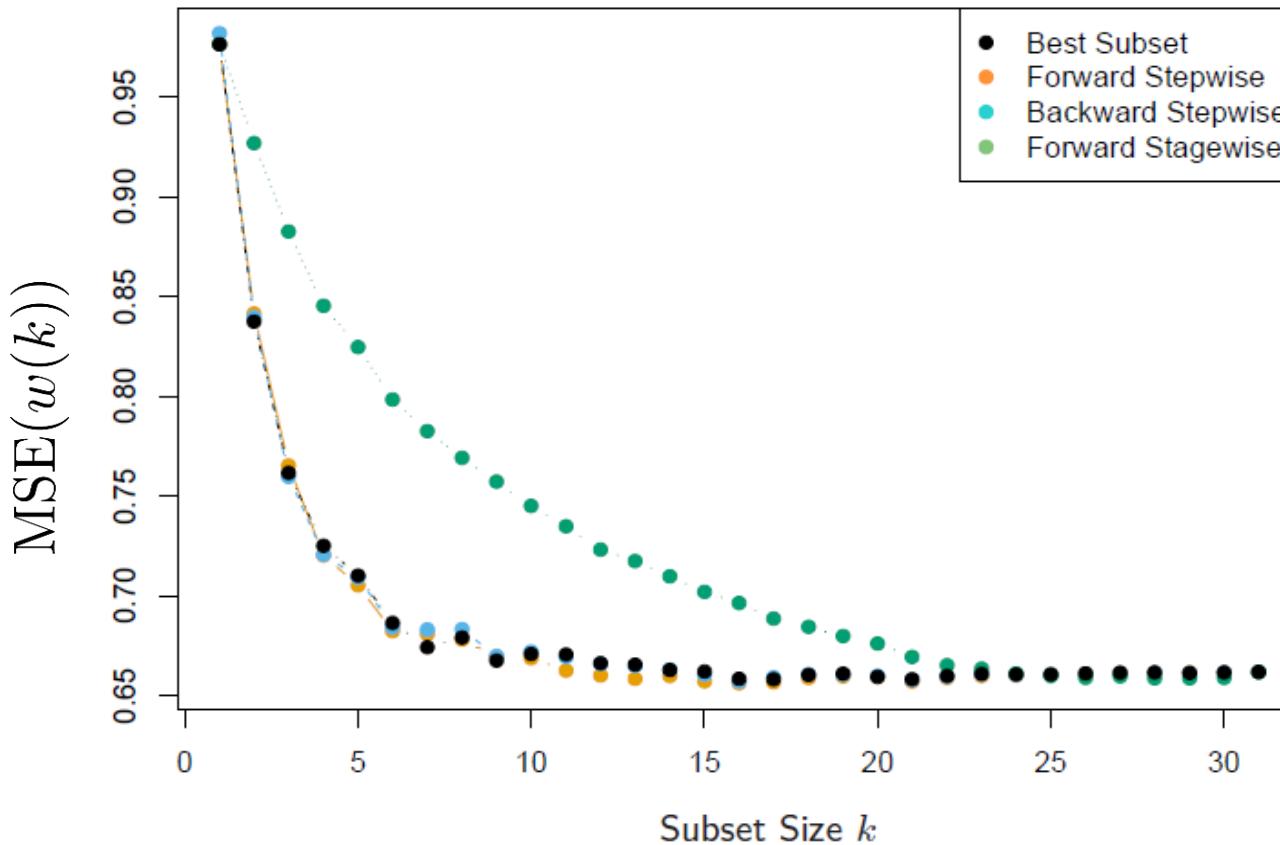
4

Considering all iterations 1 ...  $n - 1$ , the subset with the highest evaluation score is selected as the final feature subset. In case of a tie, the smallest feature subset is selected.

# Comparing Feature Selection Methods

59

Sequential selection is greedy, but often performs well...



**Example** Feature selection on synthetic model with  $p=30$  features with pairwise correlations (0.85). True feature weights are all zero except for 10 features, with weights drawn from  $N(0, 6.25)$ .

Sequential selection with  $p$  features takes  $O(p^2)$  time, compared to exponential time for best subset

Sequential feature selection available in Scikit-Learn under:  
`feature_selection.SequentialFeatureSelector`

# General Principles of Regularization

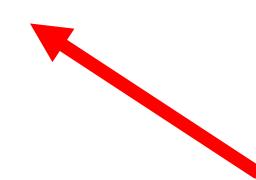
61

- From the loss function point of view

$$\text{Model} = \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$



**Regularization  
Strength**



**Regularization Penalty**

- We will see more examples of loss functions going forward.

# CSC380: Principles of Data Science

## Linear Models 3

Xinchen Yu

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

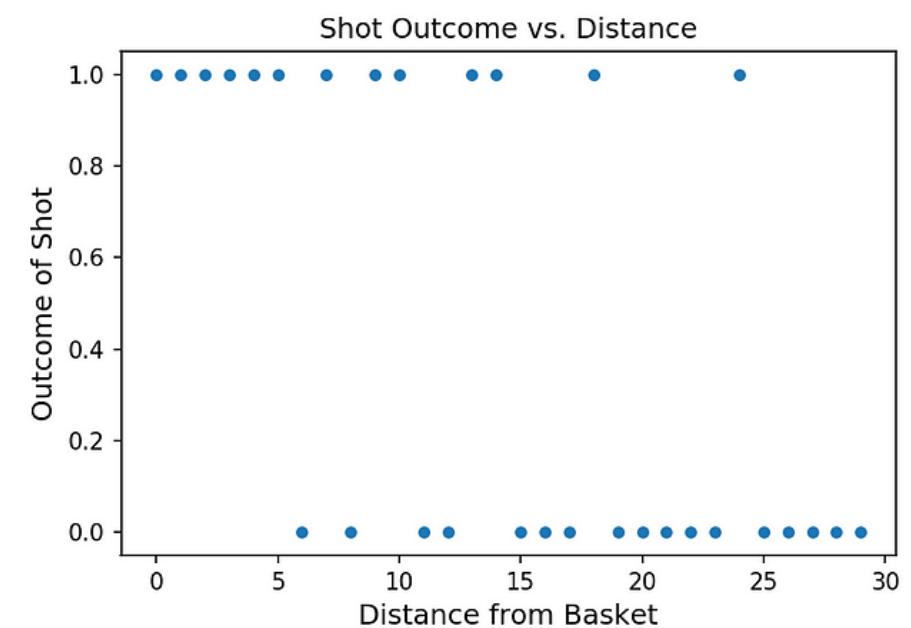
Examine the relationship between

- shooting accuracy:  $P(Y = 1|x)$
- the distance:  $x$

The further from the basket ( $x$  greater), less accurate:  $P(Y = 1|x)$  smaller,  $P(Y = 0|x)$  greater.

Q1: Can we directly use  $w^T x$  to present the probability?

Q2: How?

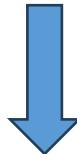


# Odds Ratios in Logistic Regression

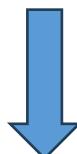
65

$$\text{Odds Ratio} = \frac{p}{1 - p}$$

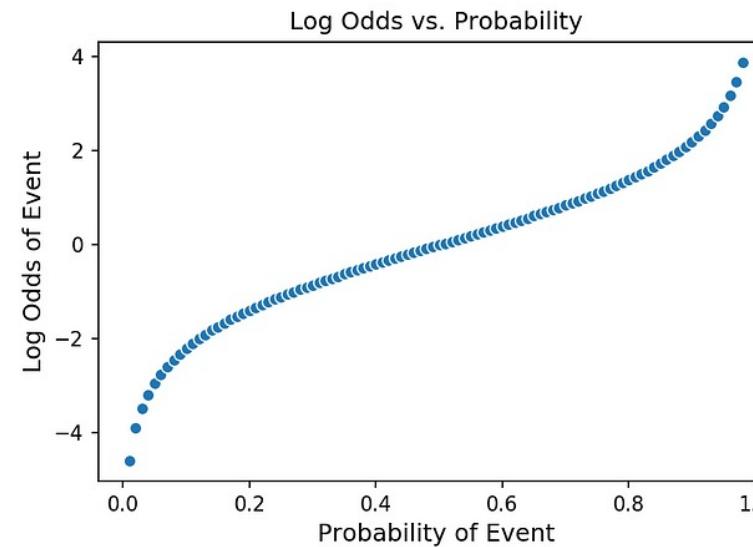
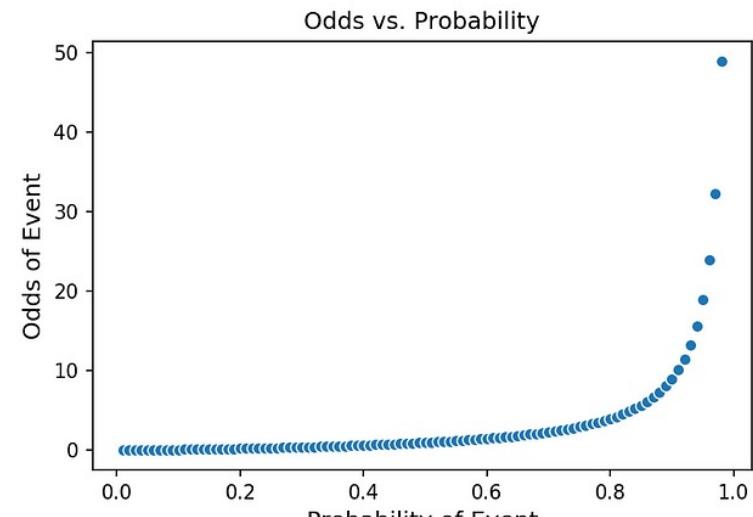
- Another way to present probability.
- Issue:  $w^T x$  needs to be no less 0



$$\text{Logit}(p) = \log \frac{p}{1 - p}$$



$$\log \frac{P(Y = 0|x)}{1 - P(Y = 0|x)} = w^T x$$



# The inverse of Logistic Regression

66

$$\log \frac{P(Y = 0|x)}{1 - P(Y = 0|x)} = w^\top x$$

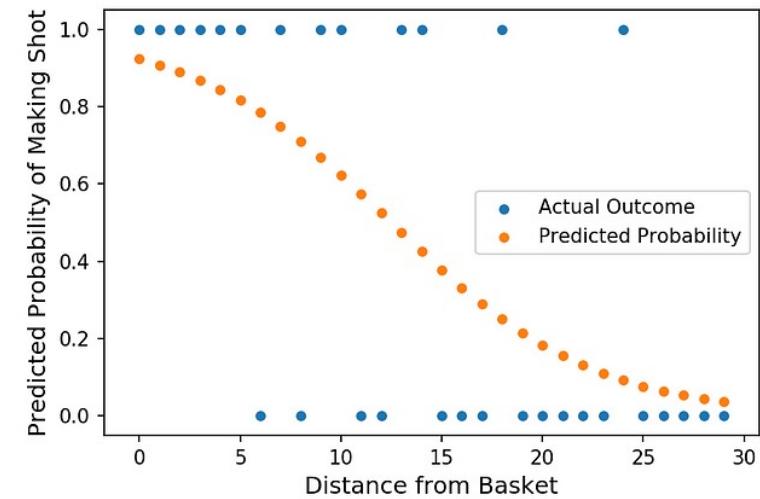
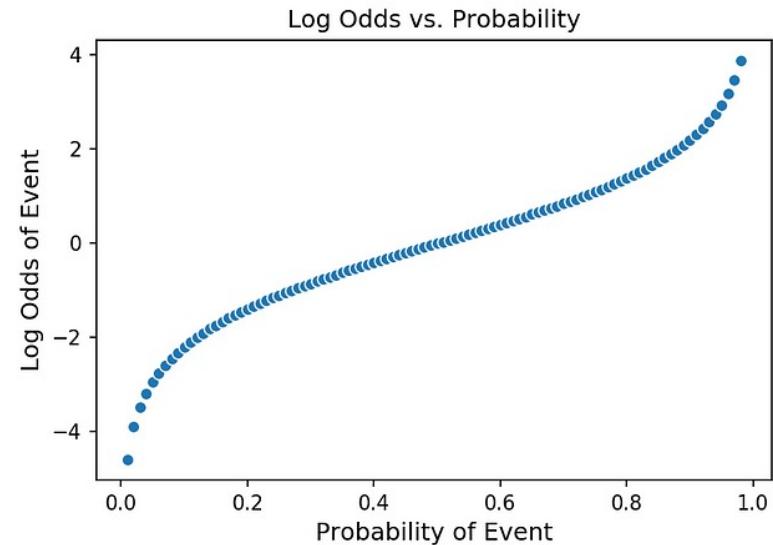


$$P(Y = 0|x) = \frac{e^{w^\top x}}{1 + e^{w^\top x}}$$

$$P(Y = 1|x) = 1 - \frac{e^{w^\top x}}{1 + e^{w^\top x}} = \frac{1}{1 + e^{w^\top x}}$$

Sigmoid  
function

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



# Logistic Regression

67

Idea Distort the prediction  $w^\top x$  in some way to map to  $[0,1]$  so that it is always a probability.

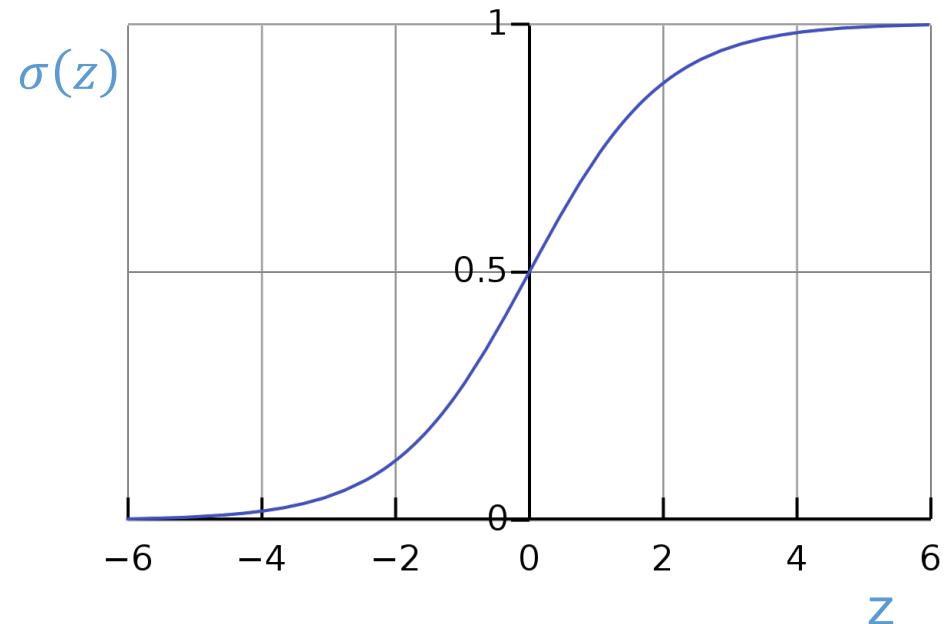
$\sigma(w^\top x)$  instead of  $w^\top x$

where

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

That is, assume

$$y \sim \text{Bernoulli}(p = \sigma(w^\top x))$$



- Logistic function is a type of *sigmoid function*, since it maps any value to the range  $[0,1]$
- Logistic also widely used in Neural Networks – for classification last layer is typically just a logistic regression

# Logistic Regression

68

Model:

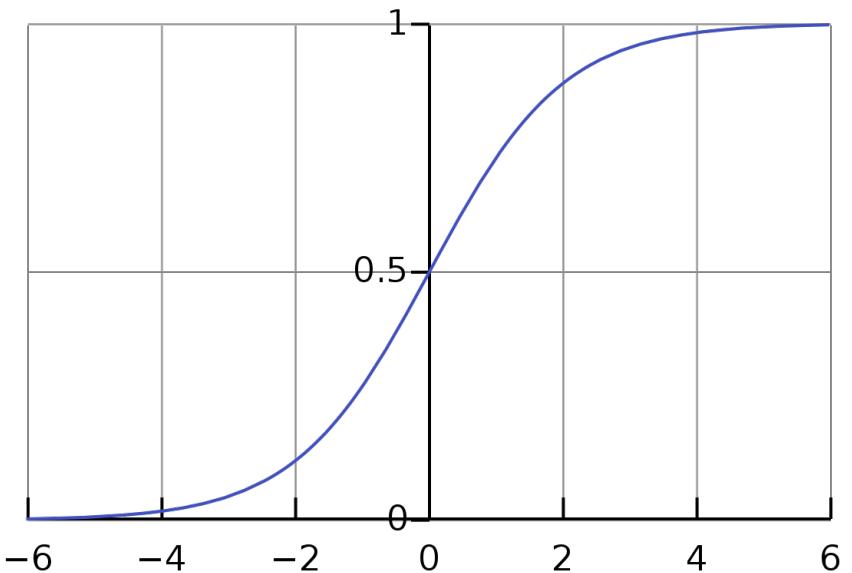
$$y \sim \text{Bernoulli}(p = \sigma(w^\top x))$$

Train: compute the MLE  $\hat{w}$

Test: Given test point  $x^*$  compute

$$y^* = \arg \max_{v \in \{-1, 1\}} p(y = v \mid x^*; \hat{w})$$

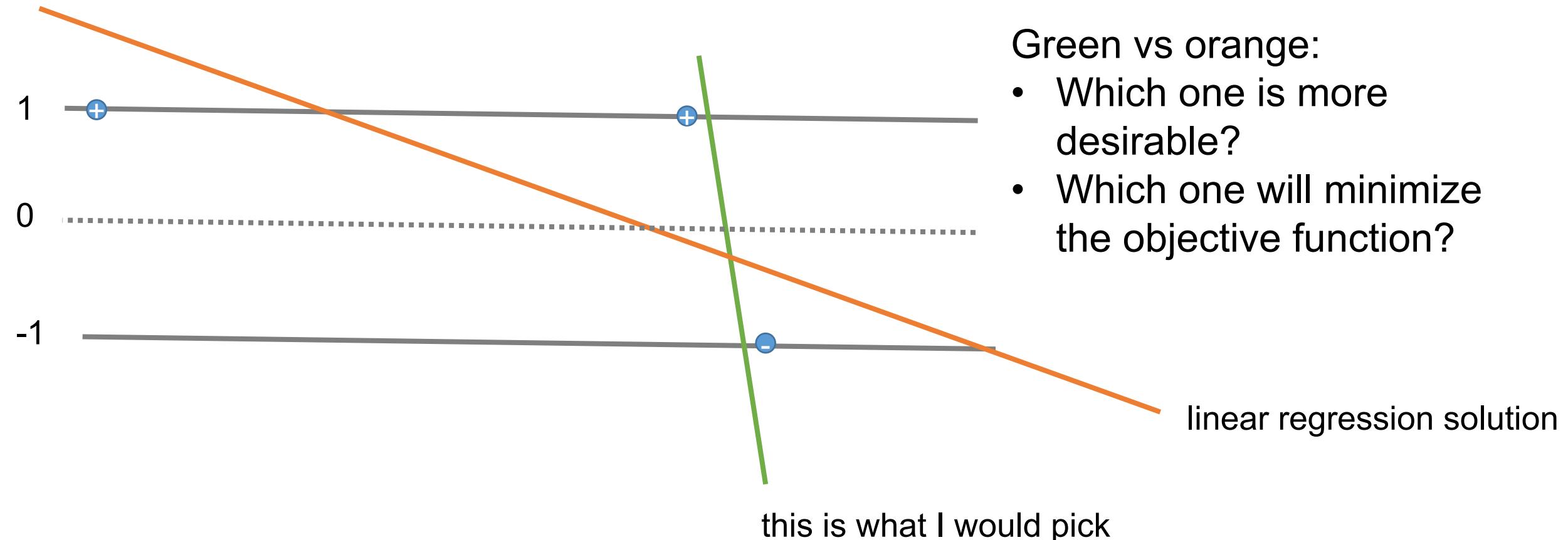
- Equivalent to  $y^* = \mathbf{I}\{\hat{w}^\top x^* \geq 0\}$



# Why Classification as Regression is Not Desirable

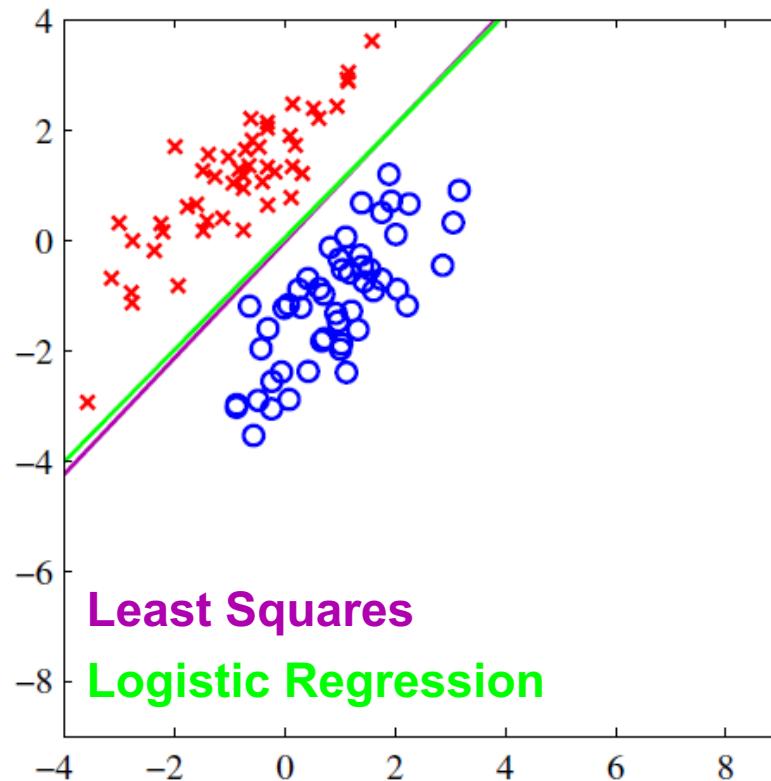
69

Recall:  $w^{\text{L2}} = \arg \min_w \sum_{i=1}^m (y^{(i)} - w^T x^{(i)})^2$



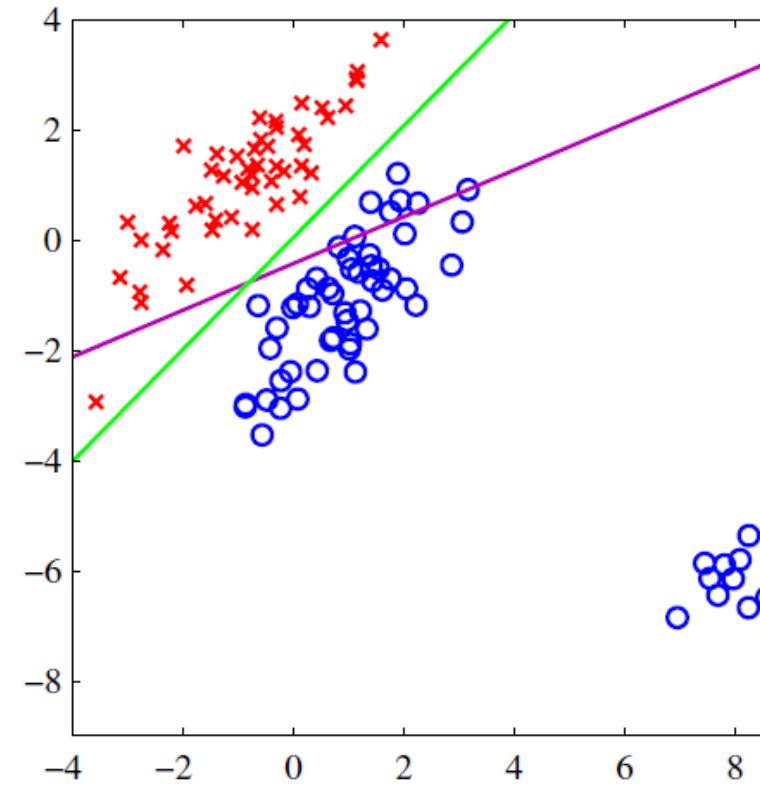
# Least Squares vs. Logistic Regression

70



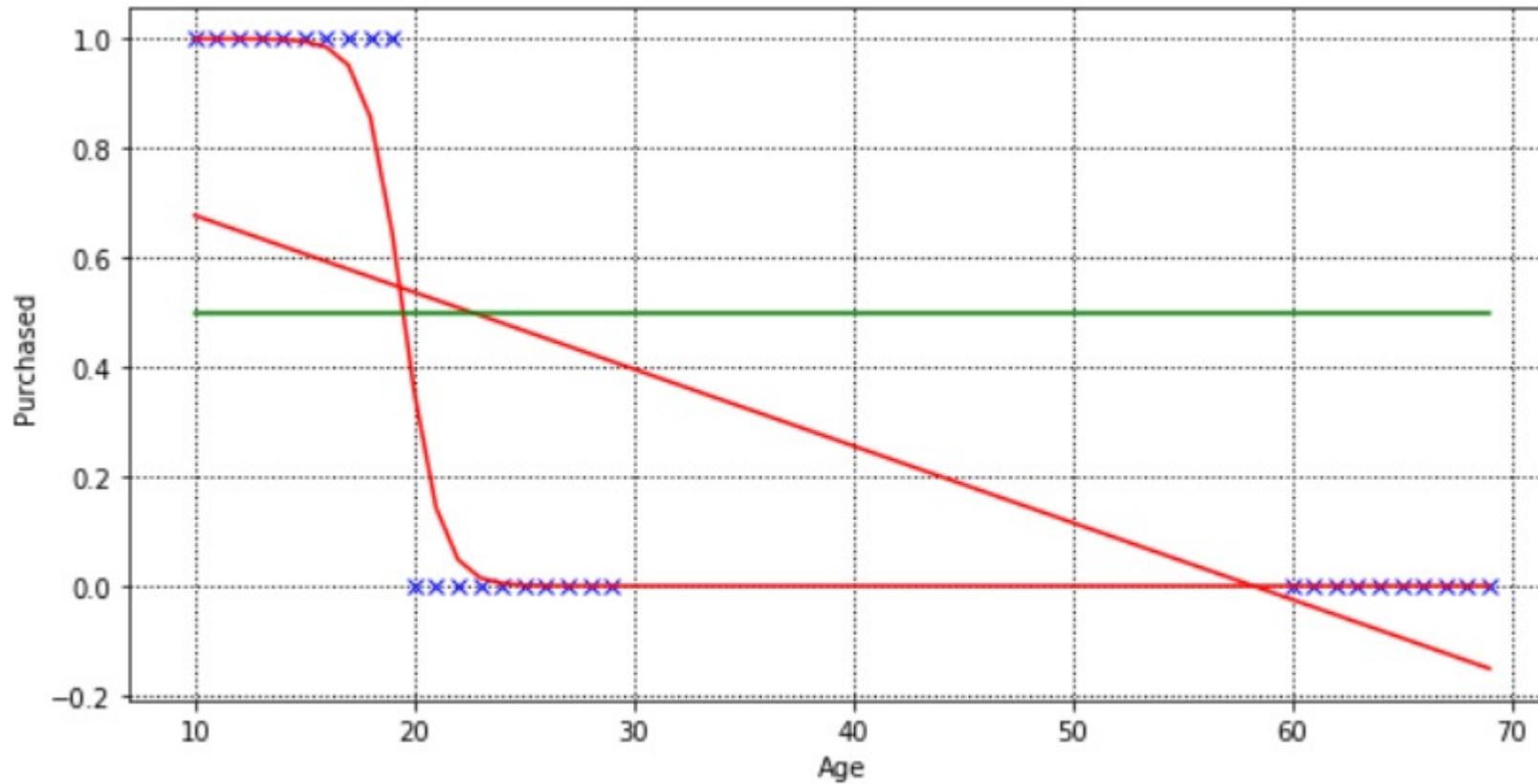
Least Squares

Logistic Regression



- Both models learn a linear decision boundary
- 😊 Least squares can be solved in closed-form (convex objective)
- 😞 Least squares is sensitive to outliers

Similar results in 1-dimension



# Linear Regression vs. Logistic regression

72

Assume  $x \sim \mathcal{D}_X$  from some distribution. We then assume that

Linear regression:

- $y = w^T x + \epsilon$  where  $\epsilon \sim \mathcal{N}(0, \sigma^2)$

Logistic regression:

- $p(y|x; w) = \sigma(w^T x)$

- Equivalently,

$$p(y|x; w) = \mathcal{N}(w^T x, \sigma^2)$$

# Fitting Logistic Regression

73

Fit by maximizing likelihood—start with the *binary* case

Posterior probability of class assignment is Bernoulli,

$$p(y \mid x; w) = p(y = 1 \mid x; w)^y (1 - p(y = 1 \mid x; w))^{(1-y)}$$

Given N iid training data pairs the log-likelihood function is,

$$\begin{aligned}\mathcal{L}_m(w) &= \sum_{i=1}^m \log p(y_i \mid x_i; w) \\ &= \sum_i \{y_i \log p(y_i = 1 \mid x_i; w) + (1 - y_i) \log p(y_i = 0 \mid x_i; w)\} \\ (\text{algebra}) \quad &= \sum_i \left\{ y_i w^T x_i - \log \left( 1 + e^{w^T x_i} \right) \right\}\end{aligned}$$

# Fitting Logistic Regression

$$w^{\text{MLE}} = \arg \max_w \sum_i \left\{ y^{(i)} w^T x^{(i)} - \log \left( 1 + e^{w^T x^{(i)}} \right) \right\}$$

Computing the derivatives with respect to each element  $w_d$ ,

$$\frac{\partial \mathcal{L}}{\partial w_d} = \sum_i x_d^{(i)} \left( y^{(i)} - \frac{e^{w^T x^{(i)}}}{1 + e^{w^T x^{(i)}}} \right) = 0$$

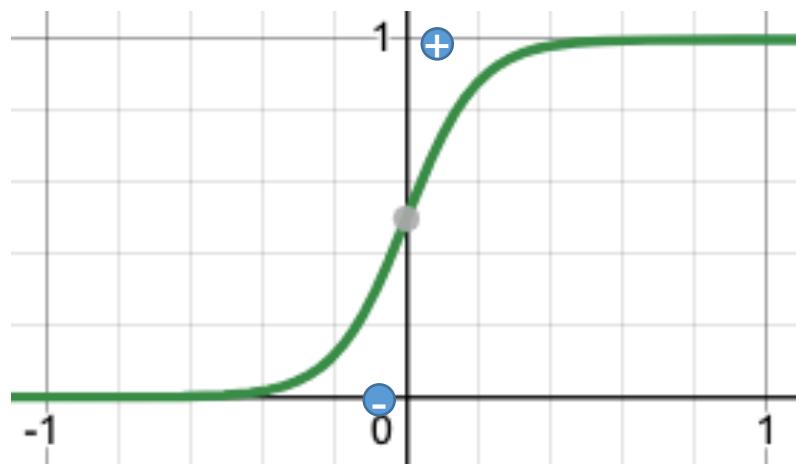
- For D features this gives us D equations and D unknowns
- Does not give a closed-form solution.
- Need to use iterative methods to solve it
- The objective function is concave => global solution can be found!

# Potential Issues

- Imagine that we adjust  $w$  manually to make this sigmoid curve to be as close as possible to training point.
  - We will have to make it a step function!
  - This can be achieved by taking  $w$  to be infinity!
- In fact, this is what happens with logistic regression in this dataset.
  - In general, happens when the dataset is **linearly separable**
- Issue: **iterative algorithms will never converge**

$$w^{\text{MLE}} = \arg \max_w \sum_i \left\{ y^{(i)} w^T x^{(i)} - \log \left( 1 + e^{w^T x^{(i)}} \right) \right\}$$

**linearly separable:**  
there exists  $w$  that can have 0 train error



Q: Solution?

$$\text{1d. } \sigma(w^T x) = \frac{e^{wx}}{1+e^{wx}}$$

# Need for Regularization

Solution: add a regularization!

$$\begin{aligned} w^{\text{L2}} &= \arg \max_w \sum_i \left\{ y^{(i)} w^T x^{(i)} - \log \left( 1 + e^{w^T x^{(i)}} \right) \right\} - \lambda \|w\|^2 \\ &= \arg \min_w \sum_i \left\{ -y^{(i)} w^T x^{(i)} + \log \left( 1 + e^{w^T x^{(i)}} \right) \right\} + \lambda \|w\|^2 \end{aligned}$$

L1 regularization also possible

- Shares the same ‘feature selection’ property!

$$w^{\text{L1}} = \arg \min_w \sum_i \left\{ -y^{(i)} w^T x^{(i)} + \log \left( 1 + e^{w^T x^{(i)}} \right) \right\} + \lambda \|w\|_1$$

Recall: linear regression has two viewpoints: (1) loss function (2) probabilistic model

Reformulation: take positive label as +1 and negative label as -1

- Loss function  $\ell(w, x, y)$  for  $x, y$

$$\begin{aligned}\ell(w, x, y) &= -yw^\top x + \log(1 + e^{w^\top x}) \\ &= \log(1 + e^{-y' w^\top x})\end{aligned}$$

- Set  $y' = 2y - 1 \in \{-1, 1\}$ . Then,

exercise:

try plugging in  $y=0$  above and  $y'=-1$  above and verify the equality. try again with  $y=1$  and  $y'=1$

- So, if the label is encoded as  $y = \pm 1$ , then

$$w^{\text{L2}} = \arg \min_w \sum_i \left\{ \log \left( 1 + e^{-y^{(i)} w^\top x^{(i)}} \right) \right\} + \lambda \|w\|^2$$

- Black: classification error (aka zero-one loss)

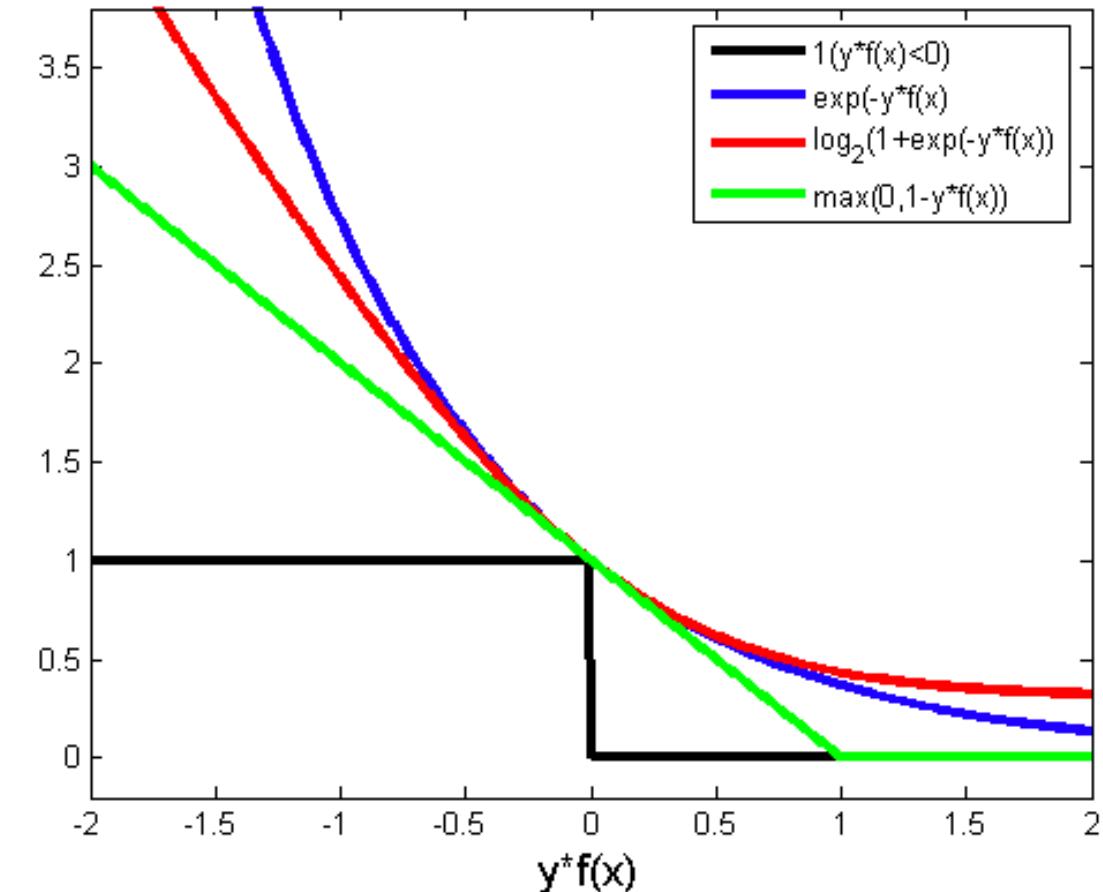
$$\ell(w, x, y') = \mathbf{I}\{y' \cdot w^\top x < 0\} \text{ with } y' \in \pm 1$$

- Red: “logistic loss”

$$\ell(w, x, y') = \log(1 + e^{-y'w^\top x})$$

## Interpretation

- The zero-one loss is not convex and hard to optimize
  - in fact, proven to be NP-hard)
- Logistic loss is a convex upper bound on the zero-one loss. => easier to optimize!



## sklearn.linear\_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None) \[source\]
```

**penalty : {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=‘l2’**

Specify the norm of the penalty:

- ‘none’ : no penalty is added;
- ‘l2’ : add a L2 penalty term and it is the default choice;
- ‘l1’ : add a L1 penalty term;
- ‘elasticnet’ : both L1 and L2 penalty terms are added.

**tol : float, default=1e-4**

Tolerance for stopping criteria.

**C : float, default=1.0**

$$C = 1/\lambda$$

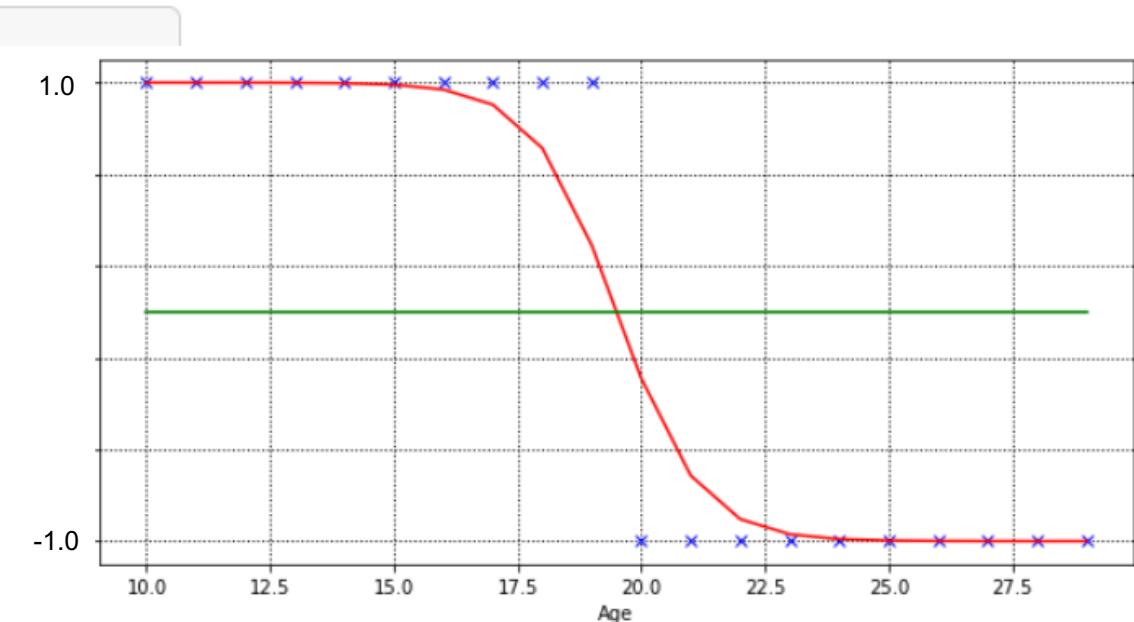
Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

# Scikit-Learn Logistic Regression

```
log_regression = sklearn.linear_model.LogisticRegression()
_ = log_regression.fit(pd.DataFrame(x), y)

y_pred = log_regression.predict_proba(pd.DataFrame(x))
log_y_pred_1 = [item[1] for item in y_pred]

fig = plt.figure(figsize=(10,5))
xlabel = 'Age'
ylabel = 'Purchased'
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.grid(color='k', linestyle=':', linewidth=1)
plt.plot(x, y, 'xb')
plt.plot(x, log_y_pred_1, '-r')
_ = plt.plot(x, line_point_5, '-g')
```



Function `predict_proba(X)` returns prediction of class assignment **probabilities** for each class. It returns n by C matrix if n data points were provided as argument.

# Extension: Multiclass

- What if we have more than 2 classes?
- For  $C$  classes,

$$y \mid x \sim \text{Categorical}(\pi) \quad \text{with} \quad \pi_j = \frac{\exp(w^{(j)\top} x)}{\sum_{c=1}^C \exp(w^{(c)\top} x)} \quad \text{CD}$$

- Alternatively, one can use

$$\pi_j = \frac{\exp(w^{(j)\top} x)}{1 + \sum_{c=1}^{C-1} \exp(w^{(c)\top} x)} \quad \text{for } j = 1, \dots, C-1, \text{ and then define } \pi_C = \frac{1}{1 + \sum_{c=1}^{C-1} \exp(w^{(c)\top} x)} \quad (\text{C-1D})$$

Q: Number of parameters for the top one and the bottom one (say D features)?

- Given movie review text, choose positive (+) or negative (-)
- 6 features :

Var	Definition
$x_1$	count(positive lexicon words $\in$ doc)
$x_2$	count(negative lexicon words $\in$ doc)
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
$x_4$	count(1st and 2nd pronouns $\in$ doc)
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
$x_6$	$\ln(\text{word count of doc})$

- Values for 6 features  $X$ :

It's hokey. There are virtually no surprises , and the writing is second-rate.  
So why was it so enjoyable ? For one thing , the cast is  
great . Another nice touch is the music I was overcome with the urge to get off  
the couch and start dancing . It sucked me in , and it'll do the same to you .

$x_1=3$        $x_2=2$        $x_3=1$        $x_4=3$        $x_5=0$        $x_6=4.19$

- 6 weights for each features  $w$ :  $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$
- $b = 0.1$

$$\begin{aligned} p(+|x) &= P(y = 1|x) \\ &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(0.833) \\ &= 0.70 \end{aligned}$$

Positive!

$$p(-|x) = 1 - 0.79 = 0.30$$