Computer Science

CSC580: Principles of Machine Learning

Linear Models

Jason Pacheco

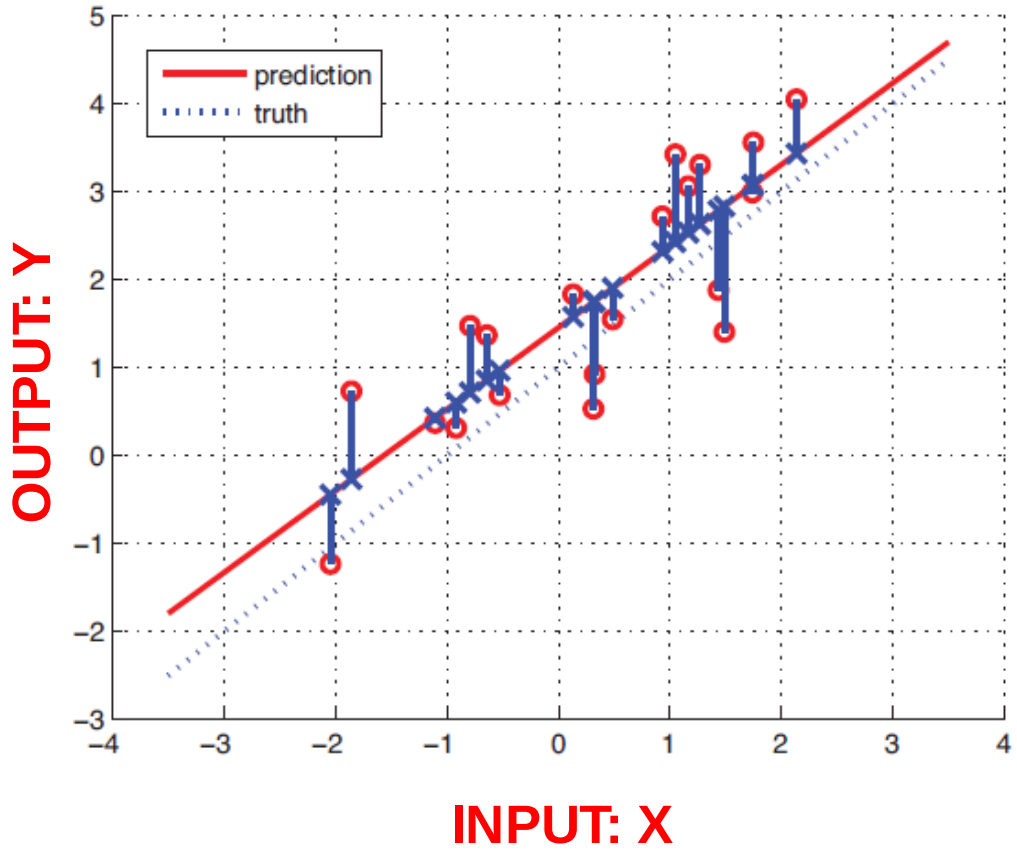
Outline

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

Outline

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

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs y are real-valued

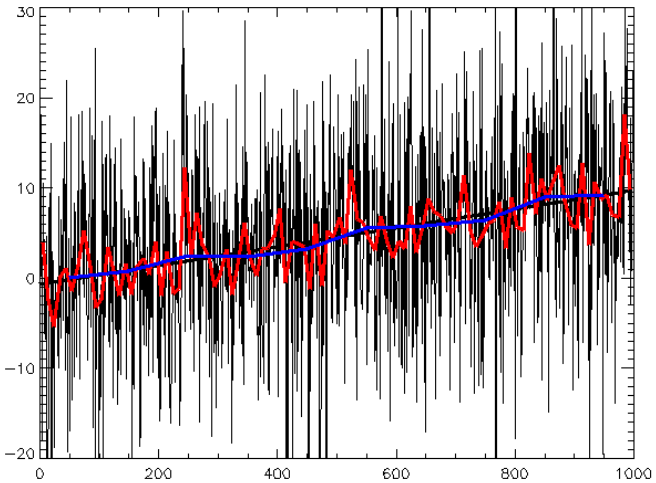
Linear Regression As the name suggests, uses a *linear function*:

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

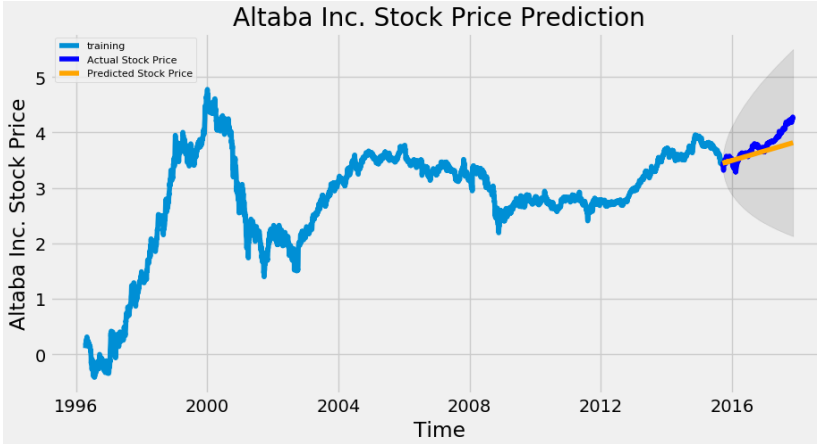
We will add noise later…

Linear Regression

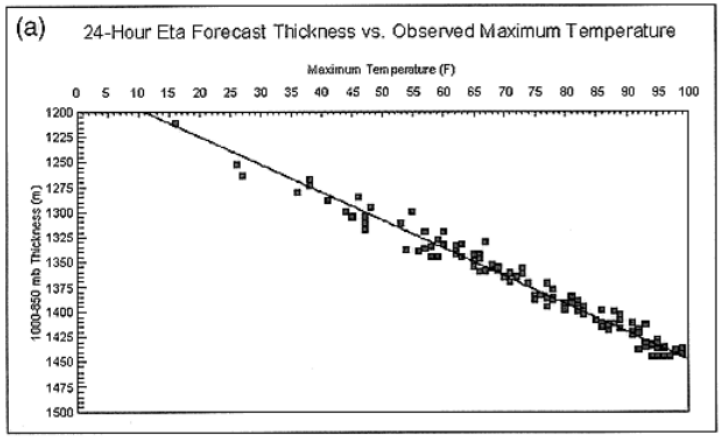
Where is linear regression useful?



Trendlines



Stock Prediction

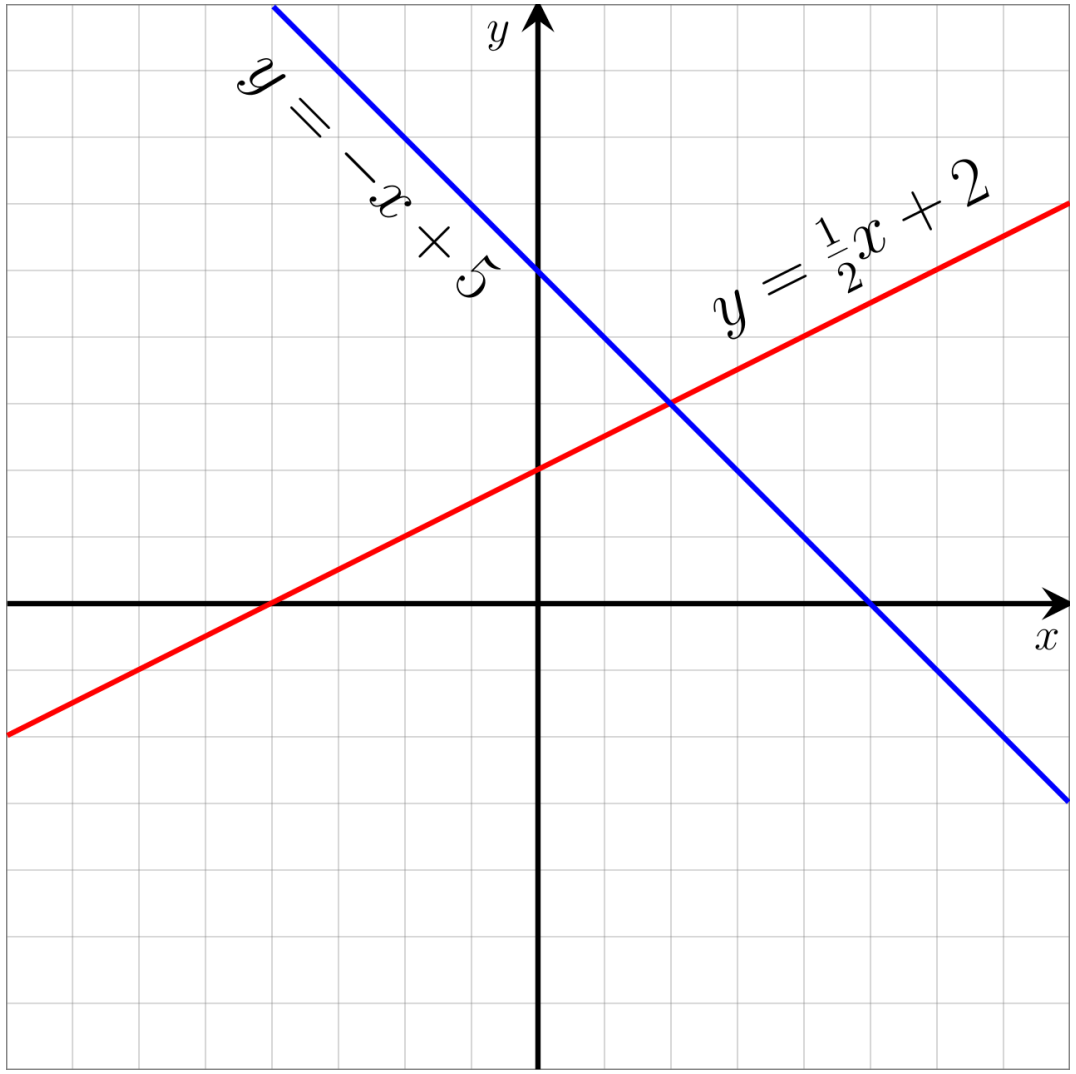


Climate Models

Massie and Rose (1997)

Used anywhere a linear relationship is assumed between continuous inputs / outputs

Line Equation



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

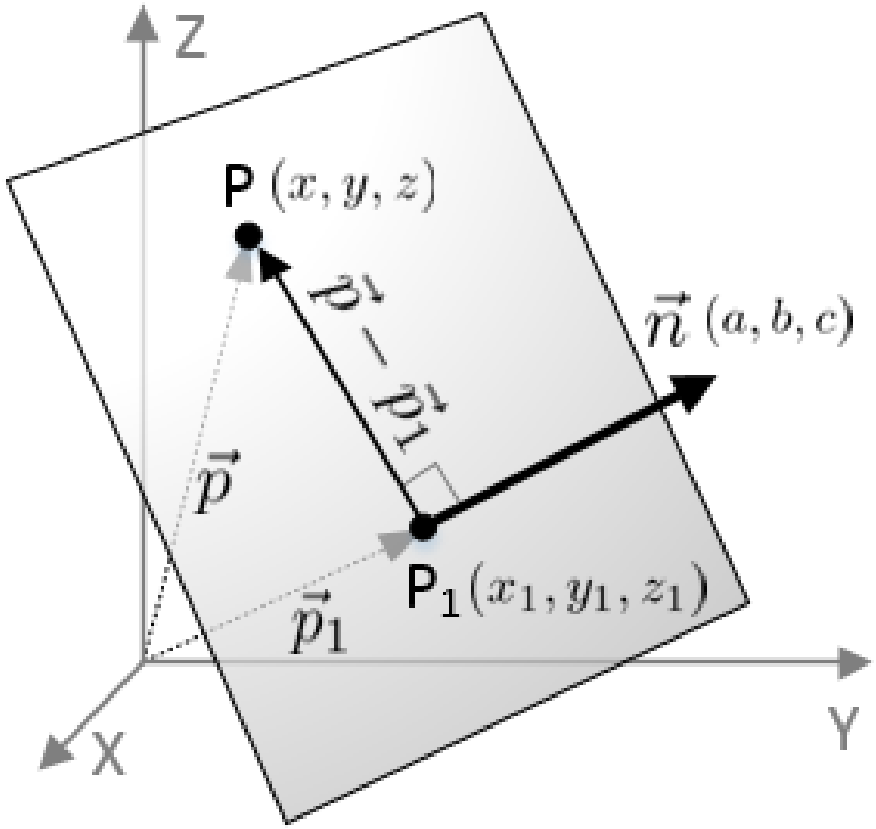
$$y = w \cdot x + b$$

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…

In higher dimensions Line → Plane



Multiple ways to define a plane, we will use:

$$n^T(p - p_1) = 0$$

Normal Vector (controls orientation)

In-Plane Vector (handles offset)

Regression weights will take place of normal vector

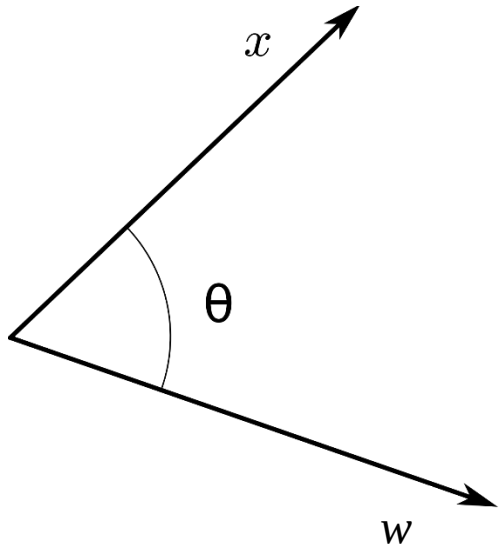
Source: http://www.songho.ca/math/plane/plane.html

Inner Products

Recall the definition of an *inner product*:

$$w^T x = w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$$

$$= \sum_{d=1}^{D} w_d x_d$$



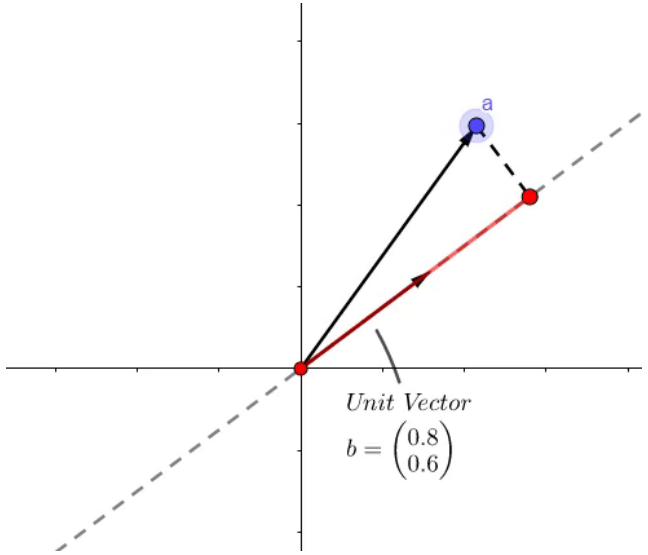
Projection of one vector onto another,

$$w^T \hat{x} = |w| \cos\theta$$

where

$$\hat{x} = \frac{x}{|x|} = \frac{x}{\sqrt{\sum_d x_d^2}}$$

Unit Vector



Linear Regression

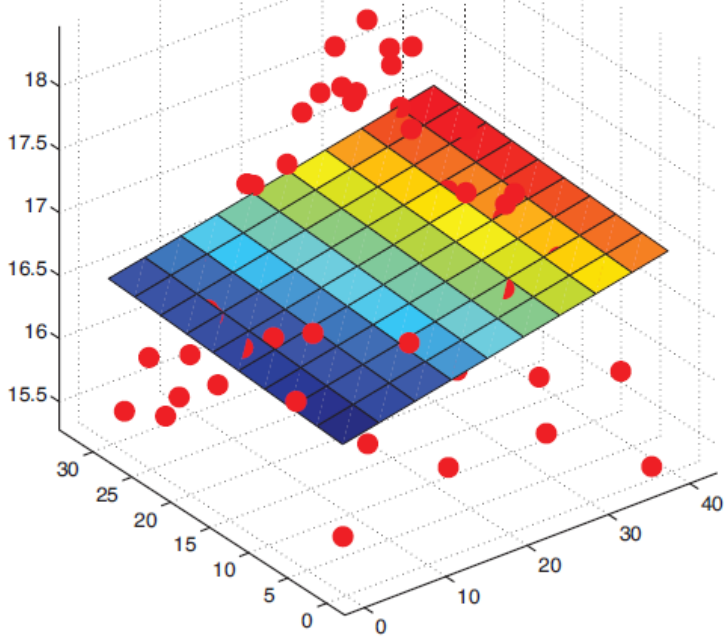
For D-dimensional input vector $x \in \mathbb{R}^D$ the plane equation,

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

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

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

[Image: Murphy, K. (2012)]



Since:

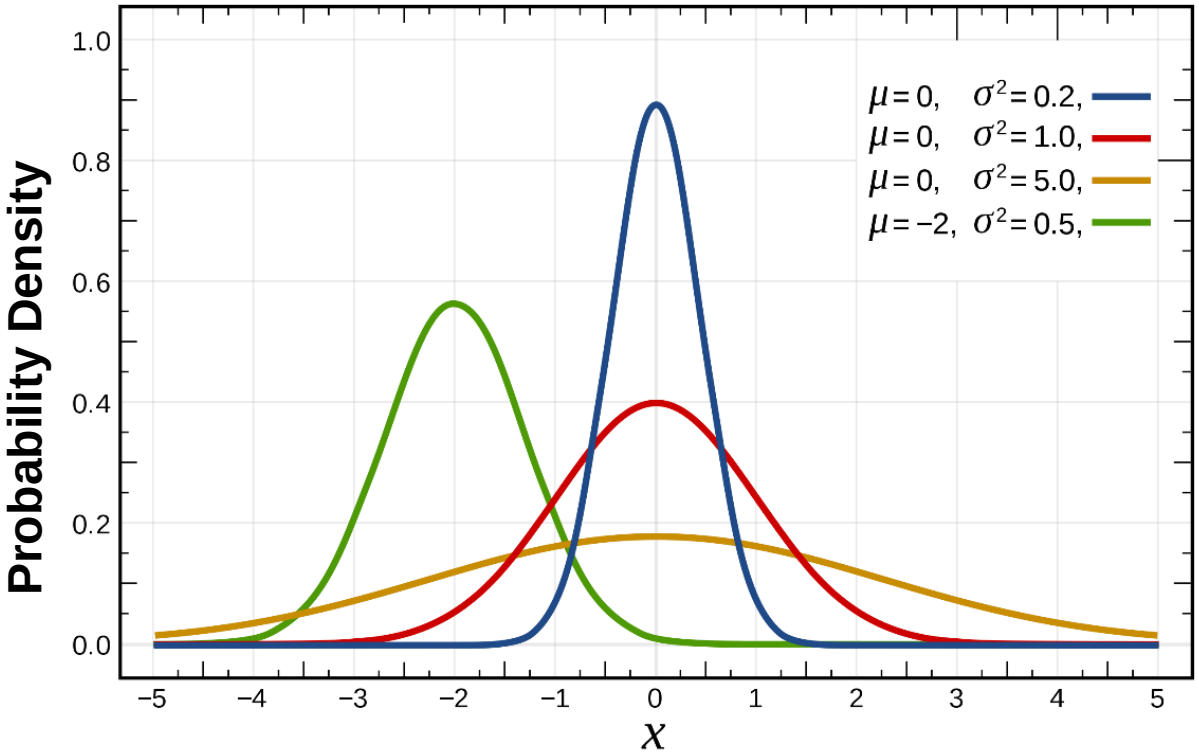
$$\widetilde{w}^T \widetilde{x} = \sum_{d=1}^{D} w_d x_d + b \cdot 1$$
$$= w^T x + b$$

Adding Noise

Gaussian (a.k.a. Normal) distribution with mean (location) μ and variance (scale) σ^2 parameters,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{1}{2}(x-\mu)^2/\sigma^2$$

We say $X \sim \mathcal{N}(X \mid \mu, \sigma^2)$



Useful Properties

- Closed under additivity:

$$X \sim \mathcal{N}(\mu_x, \sigma_x^2) \qquad Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$$

$$X + Y \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$$

- Closed under linear functions (a and b constant):

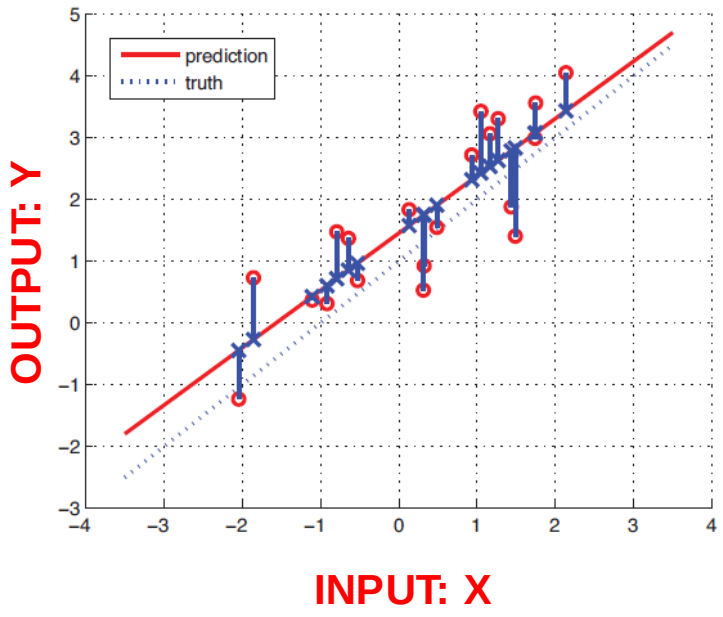
$$aX + b \sim \mathcal{N}(a\mu_x + b, a^2\sigma_x^2)$$

Linear Regression

Input-output mapping is not exact, so we will add zero-mean Gaussian noise,

Multivariate Normal (uncorrelated)

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



This is equivalent to the likelihood function,

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

Because Adding a constant to a Normal RV is still a Normal RV,

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

In the case of linear regression $z \to \epsilon$ and $c \to w^T x$

Great, we're done right?

We need to fit it to data by learning the regression weights

How to do this? What makes *good* weights?

Data – We have this

$$y = w^T x + \epsilon$$

Random; Can't do anything about it

Don't know these; need to learn them

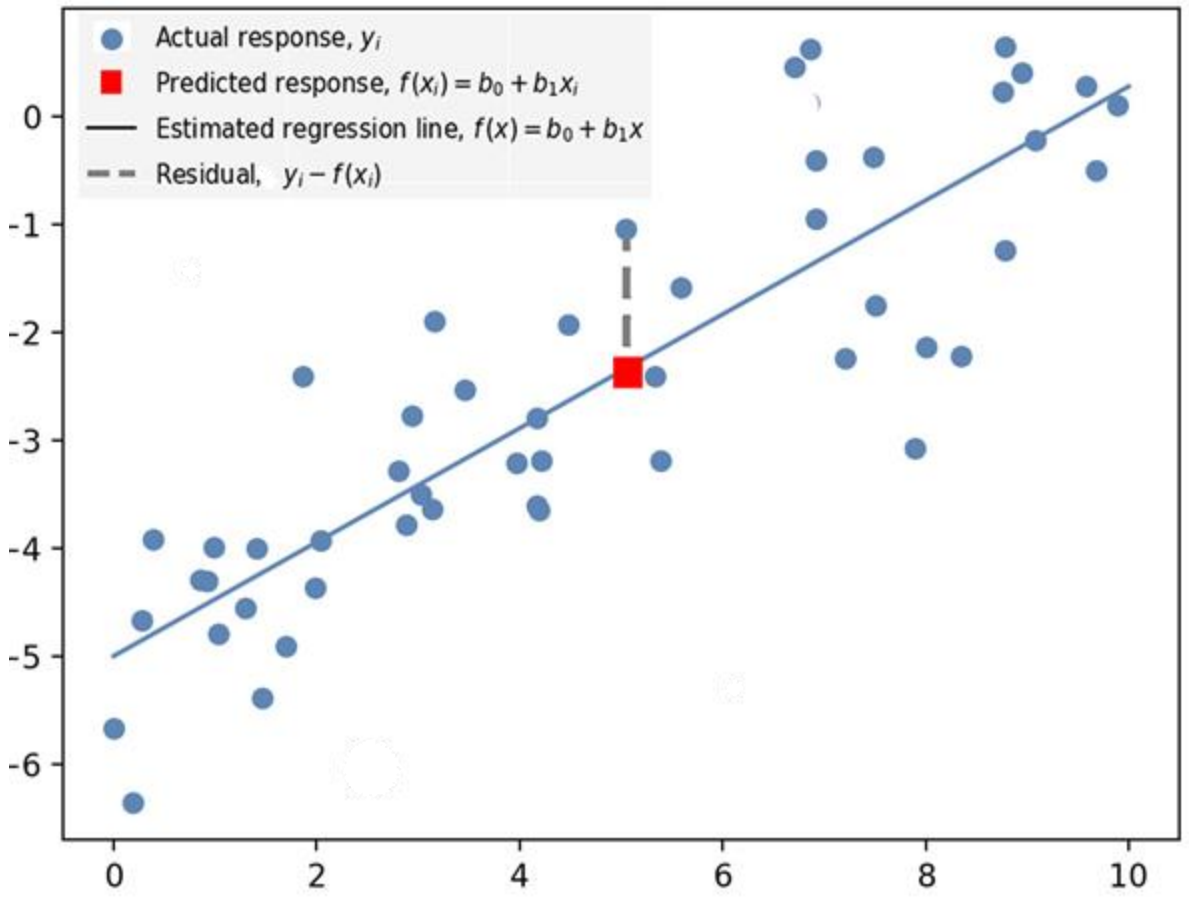
Learning Linear Regression Models

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**

$$y - w^T x$$

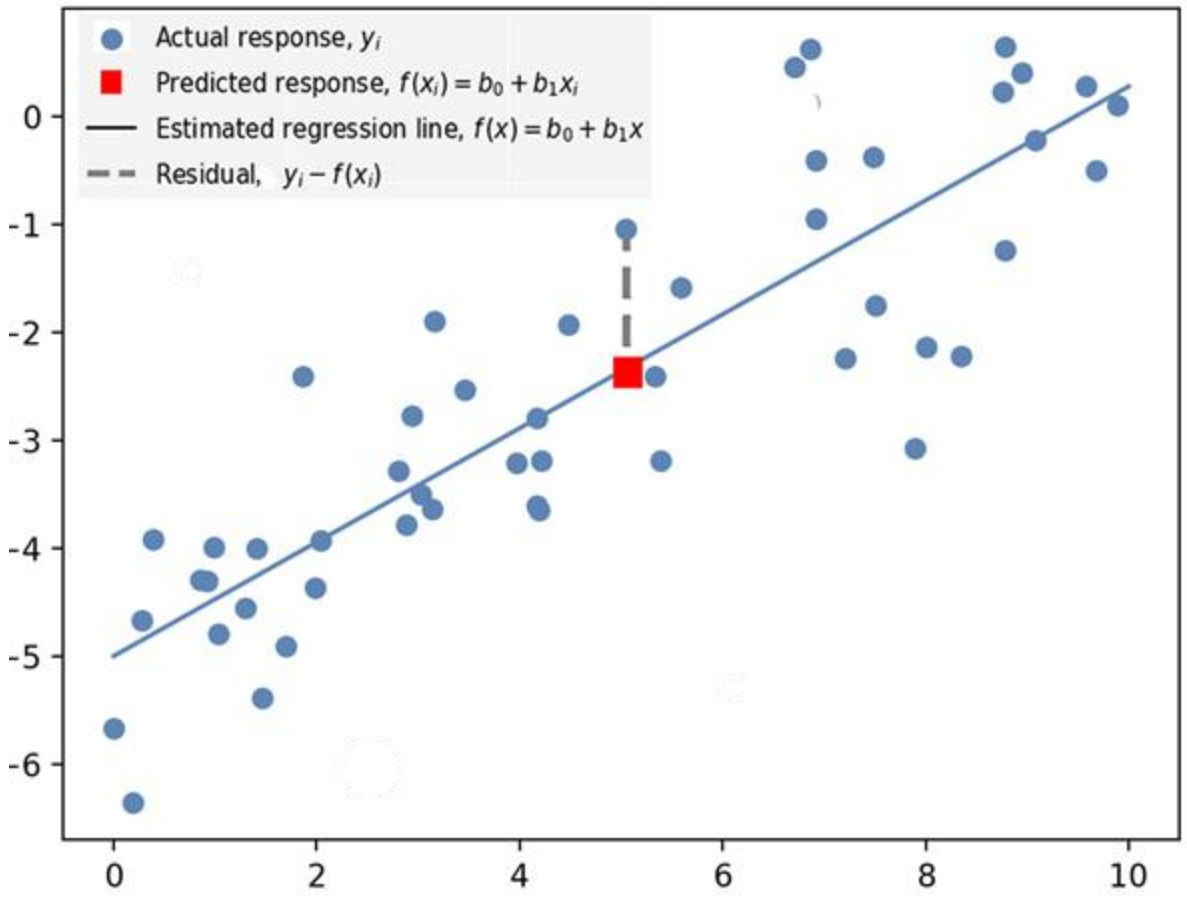
Training Output **Prediction**

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Outline

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

Least Squares Solution



Functional Find a line that minimizes the sum of squared residuals

$$w^* = \arg\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Over all the training data,

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

Least squares regression

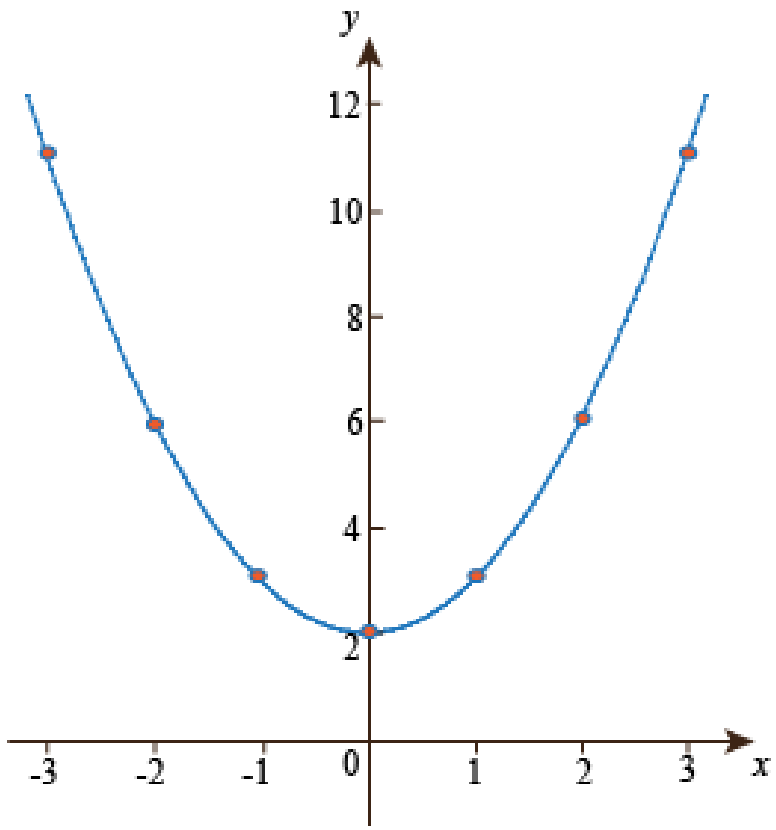
https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Least Squares

$$\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

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

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

Derivative (+ chain rule)

$$= \sum_{i=1}^{N} 2(y_i - wx_i)(-x_i) = 0 \Rightarrow$$

Distributive Property

$$0 = \sum_{i=1}^{N} y_i x_i - w\sum_{j=1}^{N} x_j^2$$

Algebra

$$w = \frac{\sum_i y_i x_i}{\sum_j x_j^2}$$

Least Squares in Higher Dimensions

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

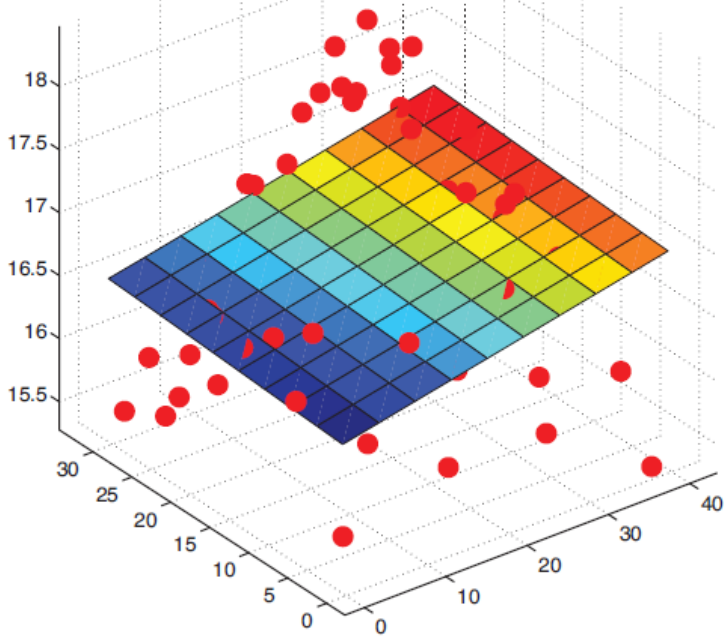
$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \dots & x_{ND} \end{pmatrix}$$

Design Matrix
(each training input on a column)

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

Vector of
Training labels

[Image: Murphy, K. (2012)]



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

$$\mathbf{y} = \mathbf{X}w$$

Nx1 Vector

Least Squares in Higher Dimensions

Least squares can also be written more compactly,

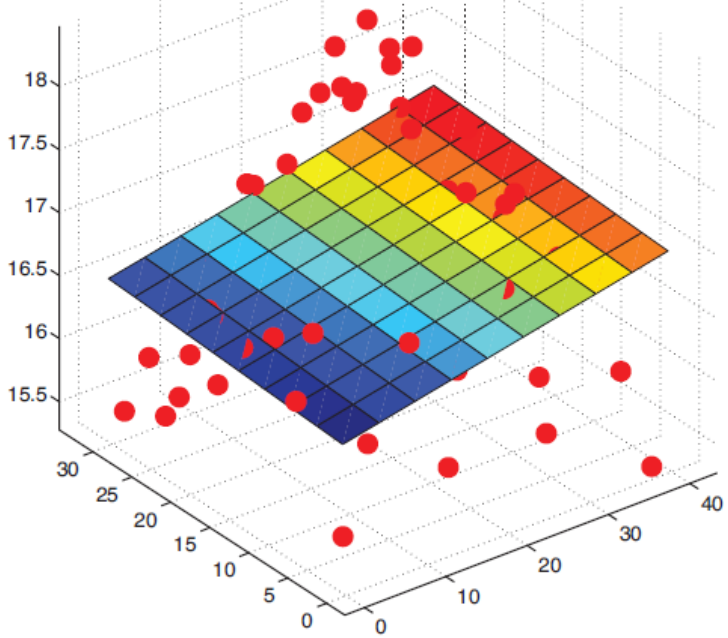
$$\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

Some slightly more advanced linear algebra gives us a solution,

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

Ordinary Least Squares (OLS) solution

[Image: Murphy, K. (2012)]



Derivation a bit involved for lecture but…

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

Learning Linear Regression Models

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…

Learning Linear Regression Models

There are several ways to think about fitting regression:

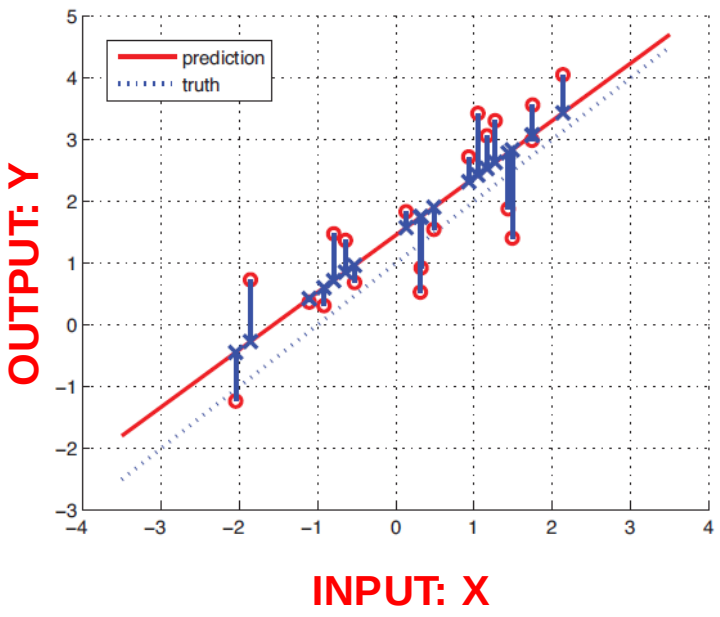
- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

MLE for Linear Regression

Given training data $\{(x_i, y_i)\}_{i=1}^N$ likelihood function is given by,

$$\log \prod_{i=1}^{N} p(y_i \mid x_i, w) = \sum_{i=1}^{N} \log p(y_i \mid x_i, w)$$



Recall that the likelihood is Gaussian:

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

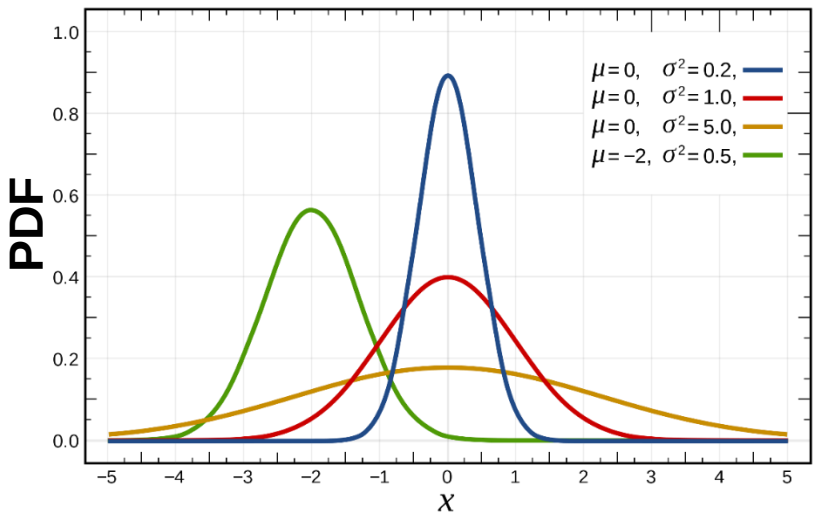
So MLE maximizes the log-likelihood over the whole data as,

$$w^{\mathrm{MLE}} = \arg\max_{w} \sum_{i=1}^{N} \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2)$$

Univariate Gaussian (Normal) Distribution

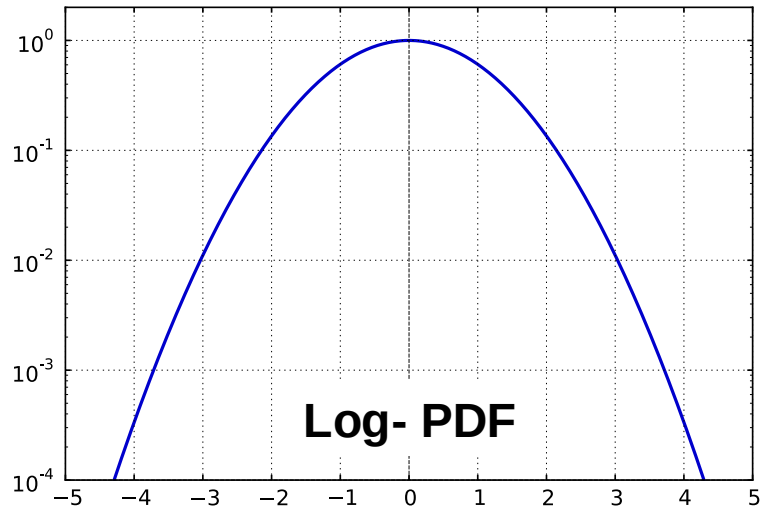
Gaussian (a.k.a. Normal) distribution with mean (location) μ and variance (scale) σ^2 parameters,

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{1}{2}(x-\mu)^2/\sigma^2$$



The logarithm of the PDF if just a negative quadratic,

$$\log \mathcal{N}(x \mid \mu, \sigma^2) = \underbrace{-\frac{1}{2}\log 2\pi - \log\sigma}_{\text{Constant in mean}} \underbrace{- \frac{1}{2\sigma^2}(x-\mu)^2}_{\text{Quadratic Function of mean}}$$



Notation

Likelihood of linear basic regression model…

$$p(y \mid w, x) = \mathcal{N}(y \mid wx, \sigma^2)$$

$$p(y \mid \mu) = \mathcal{N}(y \mid \mu, \sigma^2)$$

…we will just look at learning mean parameter for now

MLE of Gaussian Mean

Assume data are i.i.d. univariate Gaussian,

Variance is known

$$p(\mathcal{Y} \mid \mu) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mu, \sigma^2)$$

Log-likelihood function:

$$\mathcal{L}(\mu) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}(y_i - \mu)^2 \sigma^{-2}\right)\right)$$

Constant doesn't depend on mean

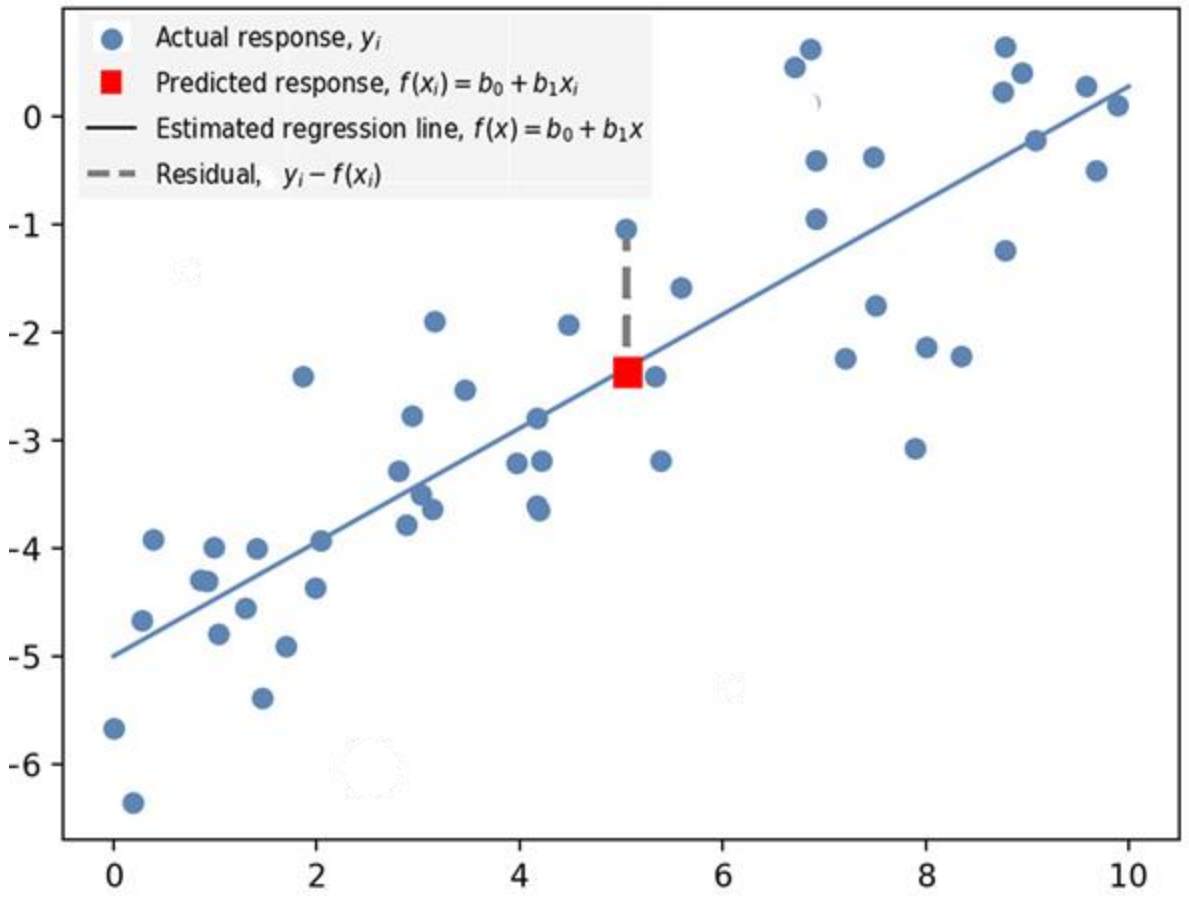
$$= \text{const.} - \frac{1}{2}\sum_{i=1}^{N}\left((y_i - \mu)^2 \sigma^{-2}\right)$$

MLE doesn't change when we:
1) Drop constant terms (in μ)
2) Minimize negative log-likelihood

MLE estimate is *least squares estimator*:

$$\mu^{\text{MLE}} = -\frac{1}{2\sigma^2} \arg\max_{\mu} \sum_{i=1}^{N}(y_i - \mu)^2 = \arg\min_{\mu} \sum_{i=1}^{N}(y_i - \mu)^2$$

MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\min_{w} \sum_{i=1}^{N} (y_i - w x_i)^2$$

So for Linear Regression, MLE = Least Squares Estimation

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Multivariate Gaussian Distribution

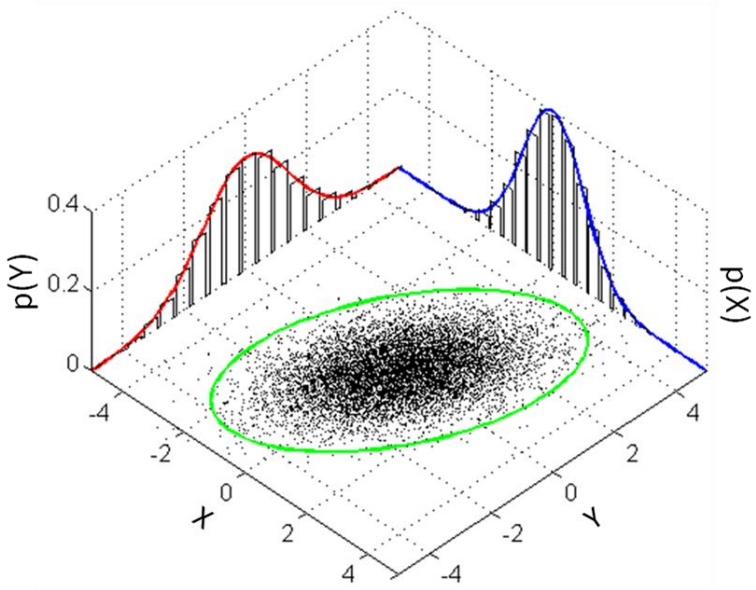
We have only seen scalar (1-dimensional) X, but MLE is still least squares for higher-dimensional X…

Let $X \in \mathcal{R}^d$ with mean $\mu \in \mathcal{R}^d$ and <u>positive semidefinite</u> covariance matrix $\Sigma \in \mathcal{R}^{d \times d}$ then the PDF is,

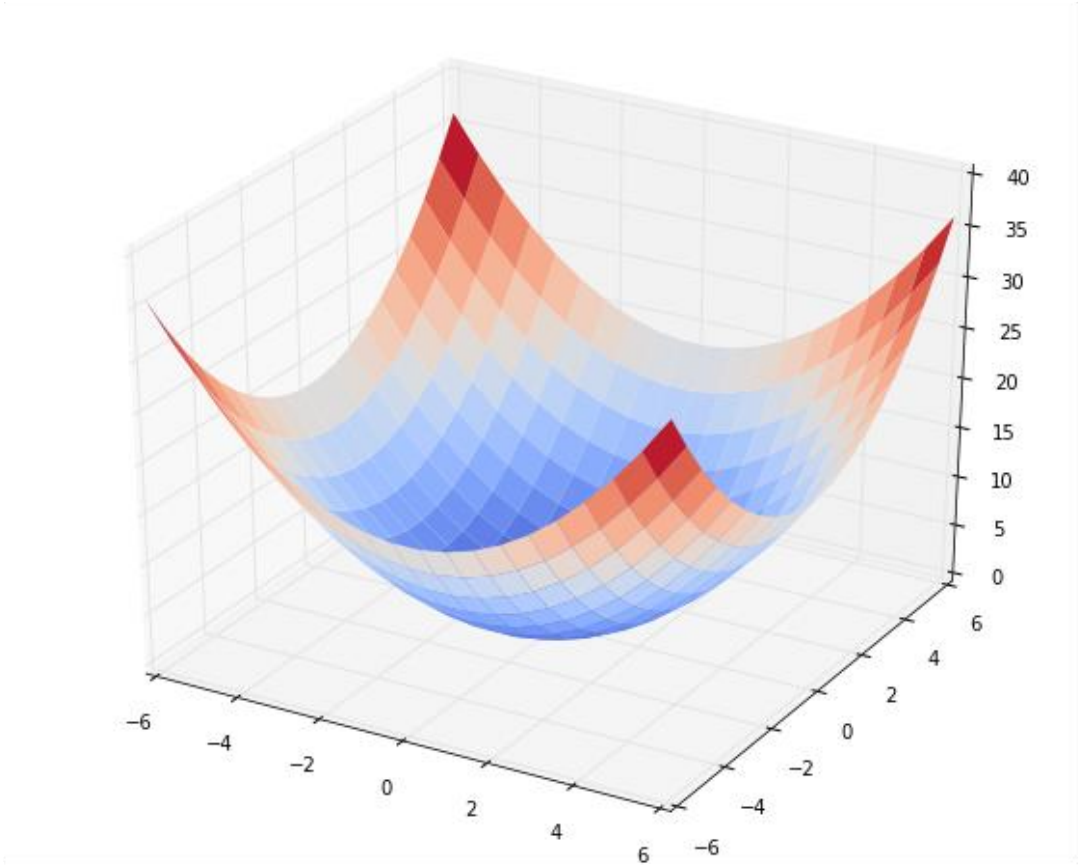
$$\mathcal{N}(x \mid \mu, \Sigma) = |2\pi\Sigma|^{-1/2} \exp -\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)$$

Again, the logarithm is a negative quadratic form,

$$\underbrace{\log |2\pi\Sigma|^{-1/2}}_{\text{Constant (in mean)}} - \underbrace{\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}_{\text{Quadratic Function of mean}}$$



Multivariate Quadratic Form



Quadratic form for vectors is given by inner product,

$$\frac{1}{2\sigma^2}(y-\mu)^T(y-\mu)$$

For iid data MLE of Gaussian mean is once-again least squares,

$$\min_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2$$

- Strongly convex
- Differentiable
- Unique optimizer at zero gradient

Notation

Substitute multi-dimensional linear regression…

$$p(y \mid \mu) = \mathcal{N}(y \mid \mu, \sigma^2)$$

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

…brings us back to the least squares solution

MLE of Linear Regression

Using previous results, MLE is equivalent to minimizing squared residuals,

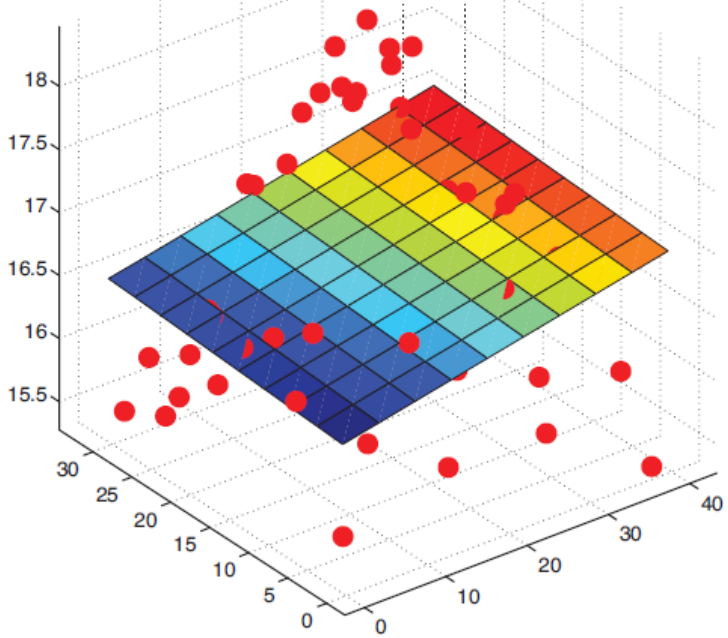
$$\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

Some slightly more advanced linear algebra gives us a solution,

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

Ordinary Least Squares (OLS) solution

[Image: Murphy, K. (2012)]



Derivation a bit involved for lecture but…

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

Linear Regression Summary

1. Definition of linear regression model,

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

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

$$w^{\mathrm{OLS}} = \arg\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

3. Equivalent to maximum likelihood solution

Linear Regression Summary

Ordinary least squares solution

$$w^{\mathrm{OLS}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Is solved in closed-form using the Normal equations,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & \dots & x_{ND} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \qquad w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Design Matrix
(each training input on a column)

Vector of
Training labels

QUESTIONS?

A word on matrix inverses…

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

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)

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

→ **Small numerical errors in input can lead to large errors in solution**

Pseudoinverse

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

The *Moore-Penrose pseudoinverse* is denoted,

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

- Generalization of the standard matrix inverse
- Exists even for non-invertible X^TX
- Directly computable in most libraries
- In Numpy it is: `linalg.pinv`

Linear Regression in Scikit-Learn

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]
```

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

Train / Test Split:

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

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

Linear Regression in Scikit-Learn

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)
```

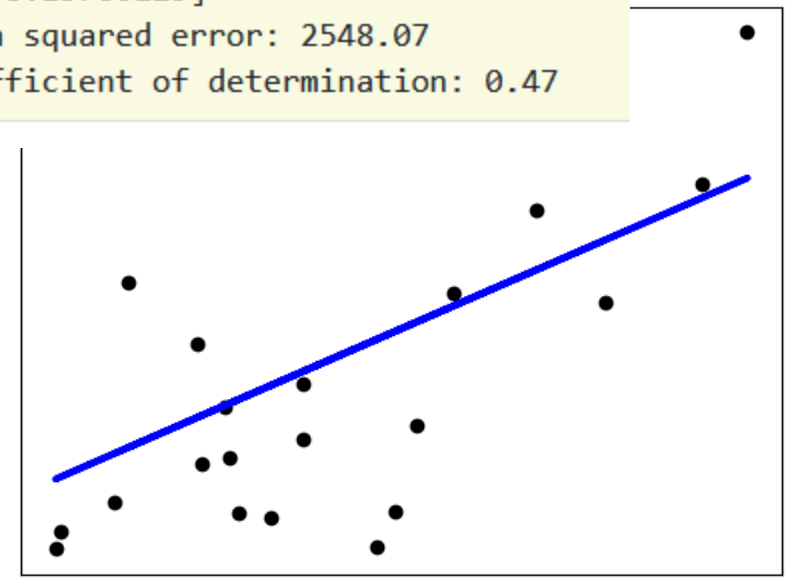
Plot regression line with the test set,

```
# 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()
```

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

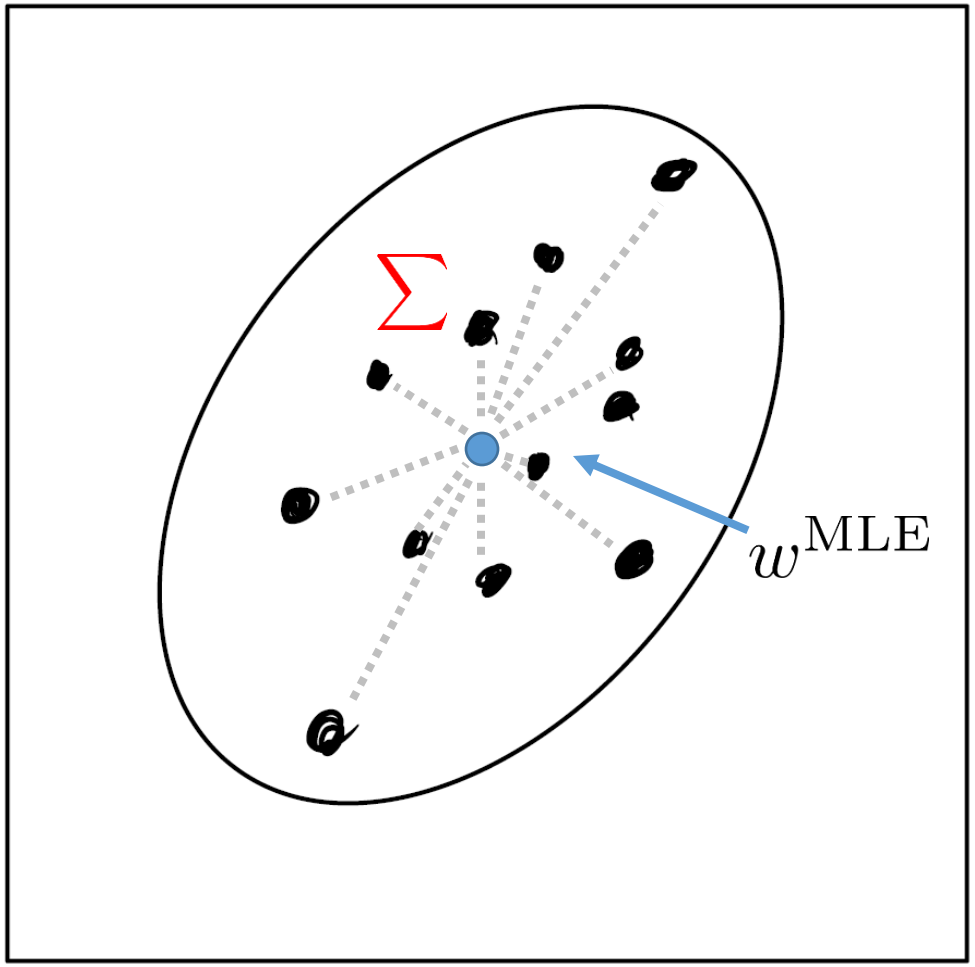


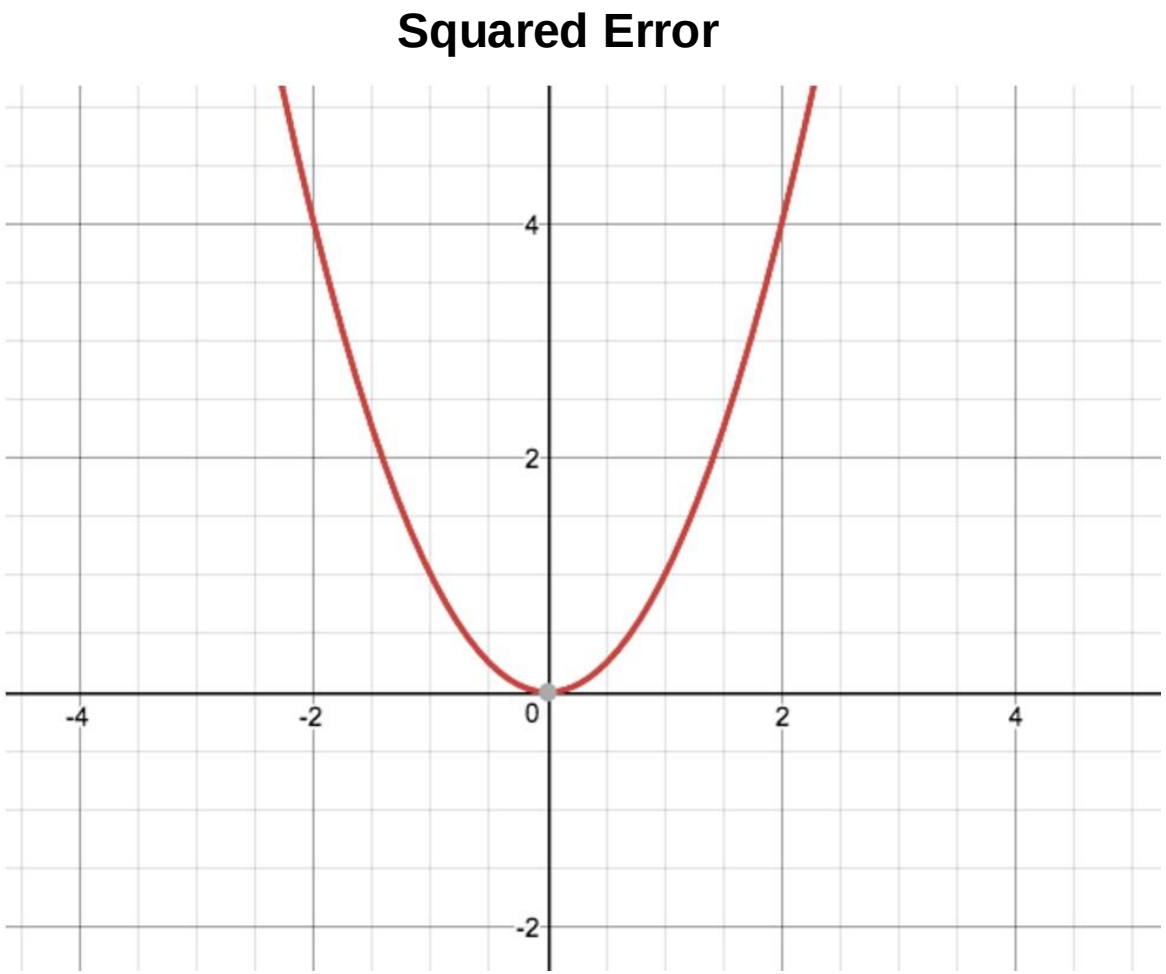
Outline

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

Outliers

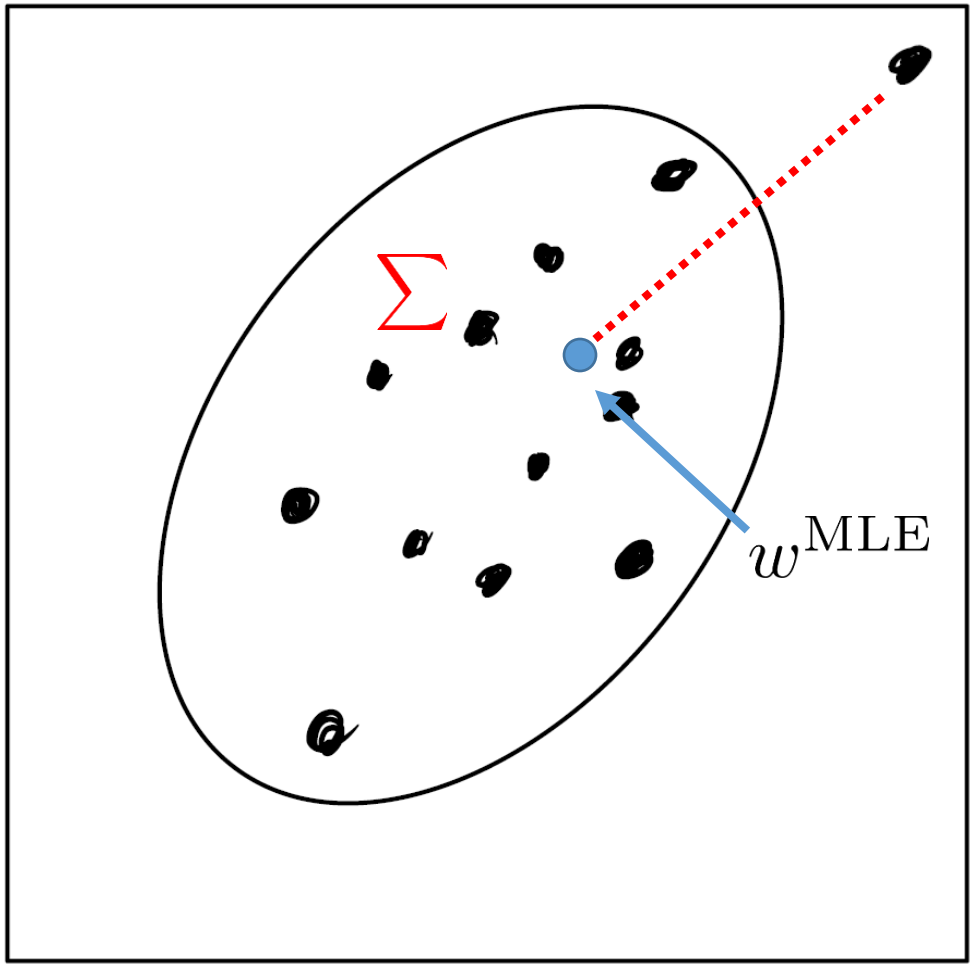
How does an outlier affect the estimator?

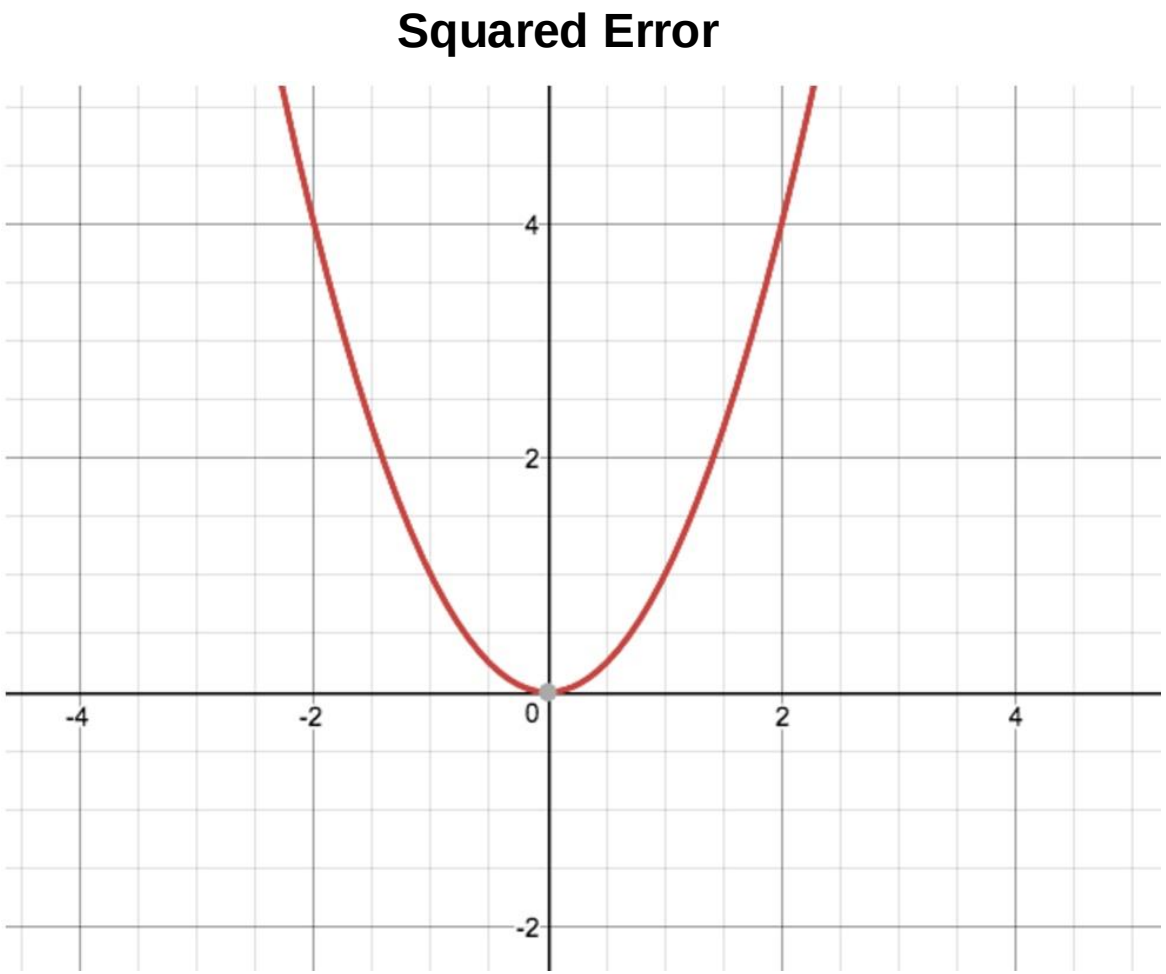




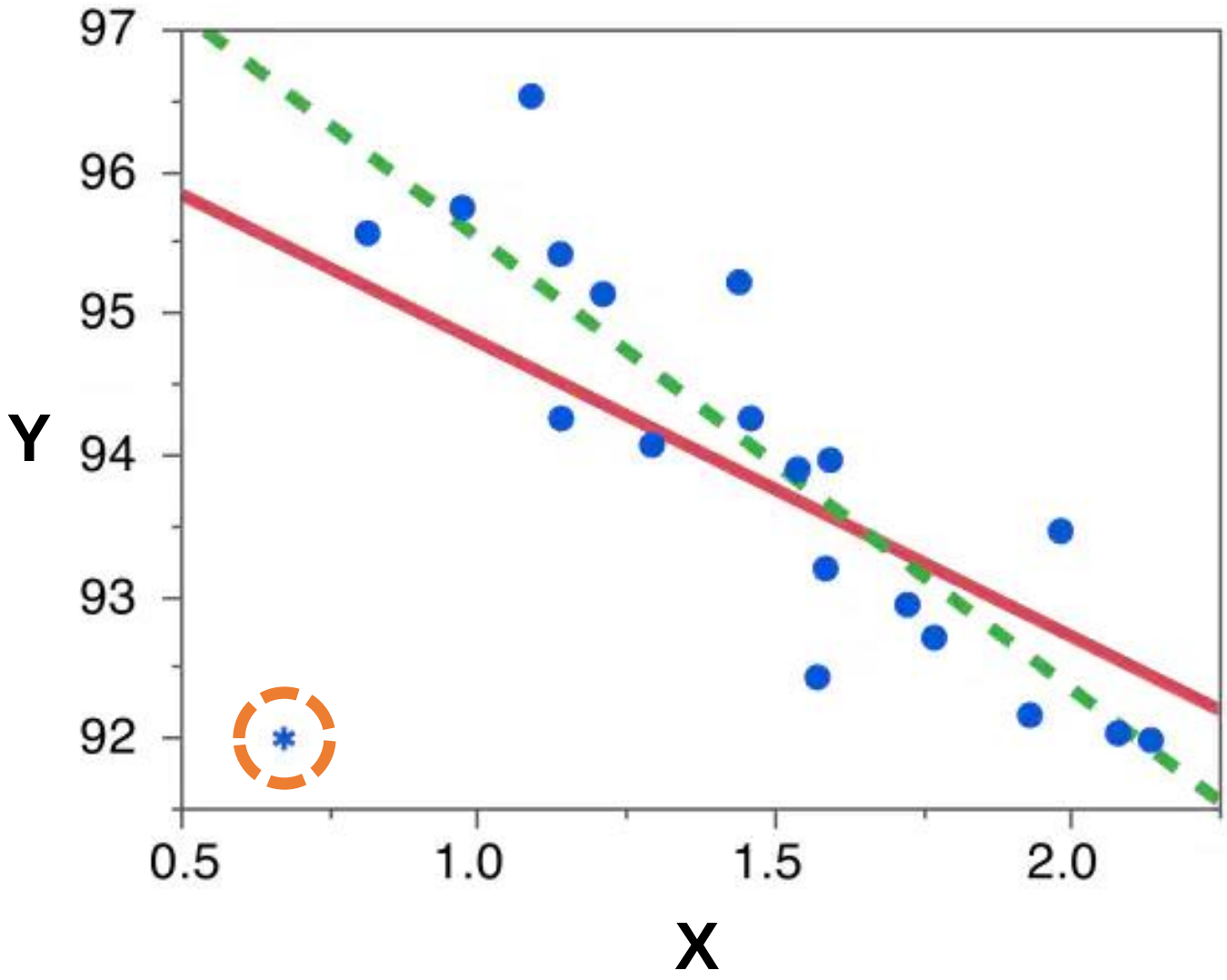
Outliers

How does an outlier affect the estimator?





Outliers in Linear Regression



Outlier “pulls” regression line away from inlier data

Need a way to *ignore* or to *down-weight* impact of outlier

https://www.jmp.com/en_us/statistics-knowledge-portal/what-is-multiple-regression/mlr-residual-analysis-and-outliers.html

Dealing with Outliers

Too many outliers can indicate many things: non-Gaussian (heavy-tailed) data, corrupt data, bad data collection, …

A few ways to handle outliers…

1. Use a heavy-tailed noise distribution (Student's T)

Fitting regression becomes difficult

2. Identify outliers and discard them

NP-Hard and throwing away data is generally bad

3. Penalize extreme weights to avoid overfitting (Regularization)

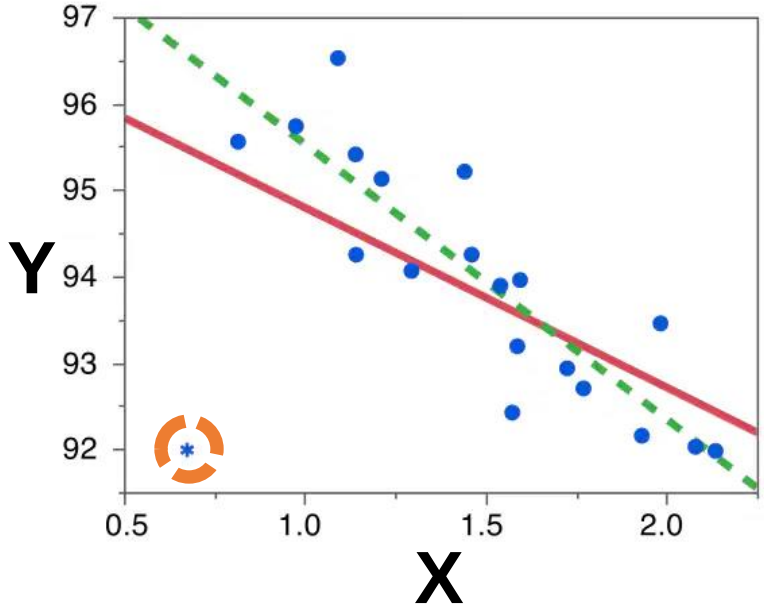
Regularization

Regularization helps avoid overfitting training data…

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

Regularization Strength

Regularization Penalty



Red model is without regularization

Green model includes regularization

Regularized Least Squares

A couple regularizers are so common they have specific names

L2 Regularized Linear Regression

- Ridge Regression
- Tikhonov Regularization

L1 Regularized Linear Regression

- LASSO
- Stands for: Least Absolute Shrinkage and Selection Operator

Regularized Least Squares

Ordinary least-squares estimation (no regularizer),

Already know how to solve this…

$$w^{\mathrm{OLS}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

L2-regularized Least-Squares (Ridge)

Quadratic Penalty

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \frac{\lambda}{2} \|w\|^2$$

L1-regularized Least-Squares (LASSO)

Absolute Value (L1) Penalty

$$w^{\mathrm{L1}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \lambda |w|$$

A word on vector norms…

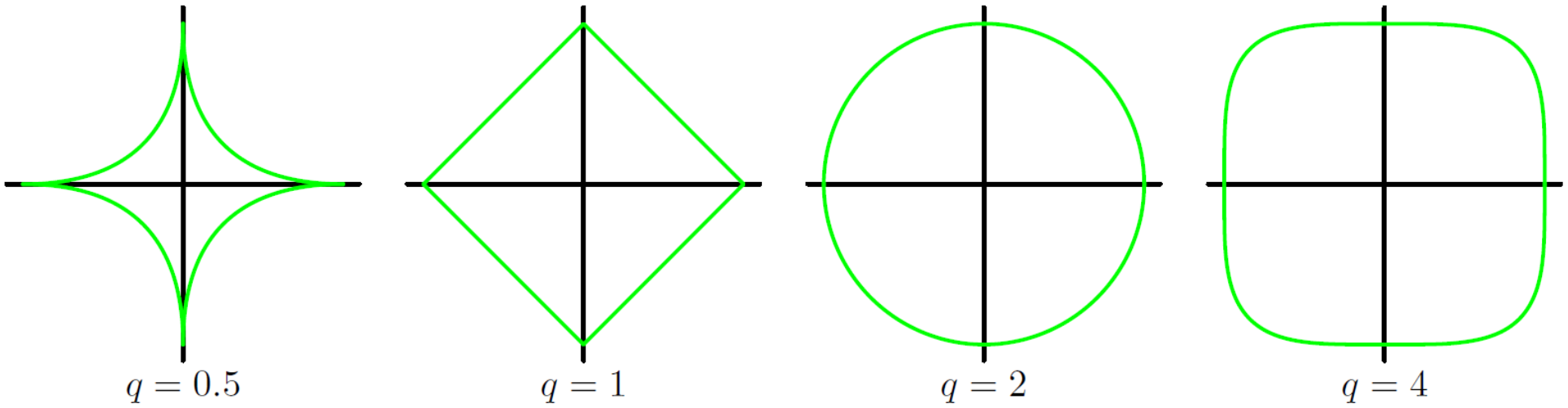
The *L2-norm* (Euclidean norm) of a vector w is,

$$\|w\| = \sqrt{w^T w} = \sqrt{\sum_{d=1}^{D} w_d^2} \qquad \|w\|^2 = \sum_{d=1}^{D} w_d^2$$

The *L1-norm* (absolute value) of a vector w is,

$$|w| = \sum_{d=1}^{D} |w_d|$$

Other Regularization Terms



q<1 is not a norm, and thus not convex

L1 is non-differentiable

L2 Regularization

A more general regularization penalty,

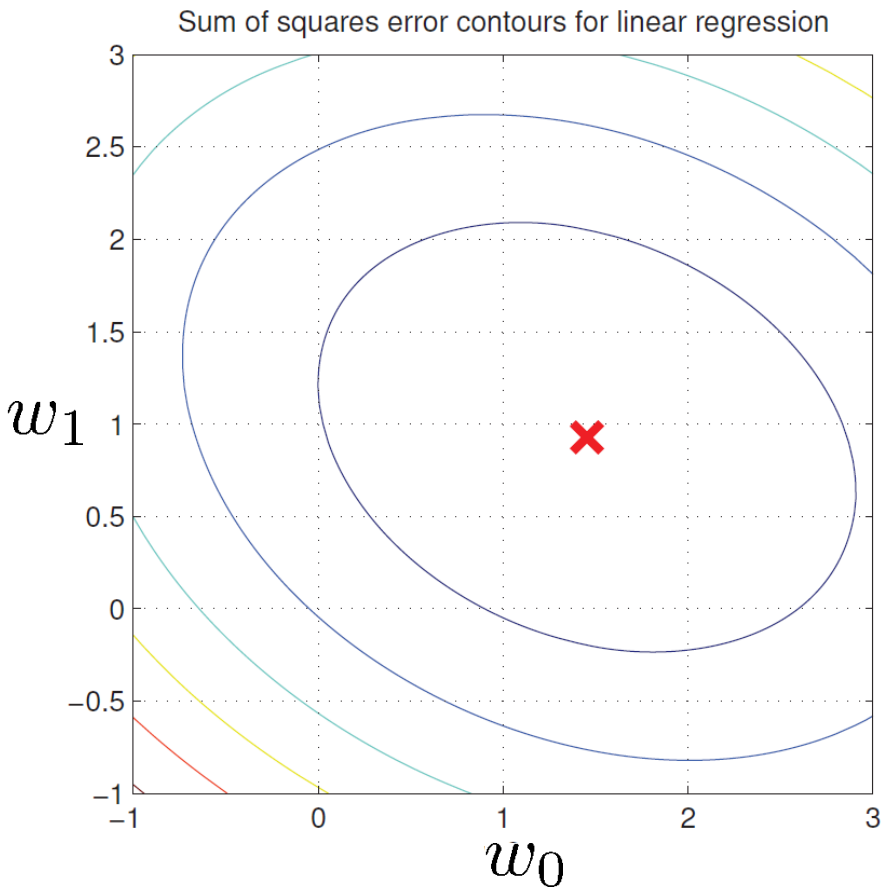
$$\hat{\theta} = \arg\min_{\theta} \frac{1}{2} \sum_{i=1}^{N} (y_i - \theta)^2 + \frac{\lambda}{2} |\theta|^q$$

L2 Regularized Least Squares

$$w^{\mathrm{L2}} = \arg\min_{w} \underbrace{\sum_{i=1}^{N} (y_i - w^T x_i)^2}_{\text{Quadratic}} + \overbrace{\frac{\lambda}{2}\|w\|^2}^{\text{Quadratic}}$$

Quadratic + Quadratic = Quadratic

- Differentiable
- Convex
- Unique optimum
- Closed form solution



L2 Regularized Least Squares : Simple Case

$$\frac{d}{dw}\frac{1}{2}\sum_{i=1}^{N}(y_i - wx_i)^2 + \frac{\lambda}{2}\frac{d}{dw}w^2 =$$

Derivative (+ chain rule)

$$= \sum_{i=1}^{N}(y_i - wx_i)(-x_i) + \lambda w \quad = 0 \Rightarrow$$

Distributive Property

$$0 = \sum_{i=1}^{N} y_i x_i - w\sum_{j=1}^{N} x_j^2 - \lambda w$$

Algebra

$$w = \frac{\sum_i y_i x_i}{\lambda + \sum_j x_j^2}$$

L2 Regularized Linear Regression – Ridge Regression

Source: Kevin Murphy's Textbook

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \frac{\lambda}{2} \|w\|^2$$

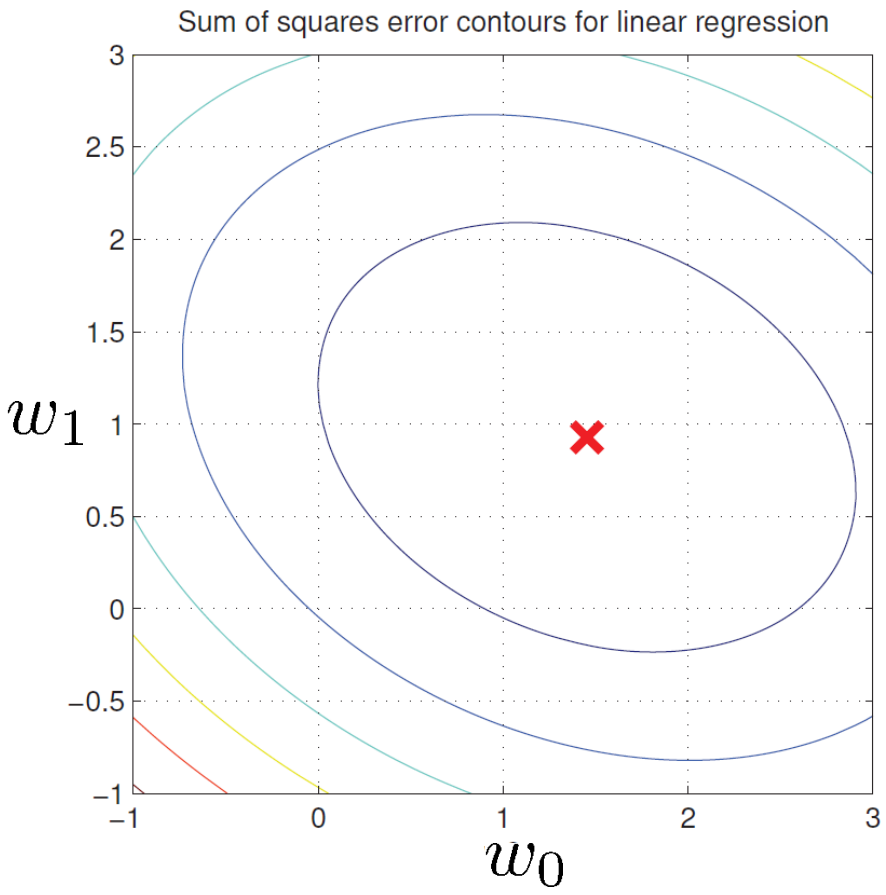
After some algebra…

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

Compare to ordinary least squares:

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

Regularized least-squares includes pseudocount in weighting similar to Gaussian mean estimator



Notes on L2 Regularization

- Feature weights are “shrunk” towards zero (and each other) – statisticians often call this a “shrinkage” method
- Typically do **not** penalize bias (y-intercept, w_0) parameter,

$$\min_w \sum_i (y_i - w^T x_i - w_0)^2 + \lambda \sum_{d=1}^{D} w_d^2$$

- Penalizing w_0 would make solution depend on origin for Y – adding a constant c to Y would **not** add a constant to solution weights
- Can fit bias in a two-step procedure, by *centering* features $x_{ij} - \bar{x}$ then bias estimate is $w_0 = \bar{y}$
- Solutions are **not** invariant to scaling, so typically we standardize (e.g. Z-score) features before fitting model (Sklearn `StandardScaler`)

Scikit-Learn : L2 Regularized Regression

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]

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.

Alpha is what we have been calling λ

Scikit-Learn : L2 Regularized Regression

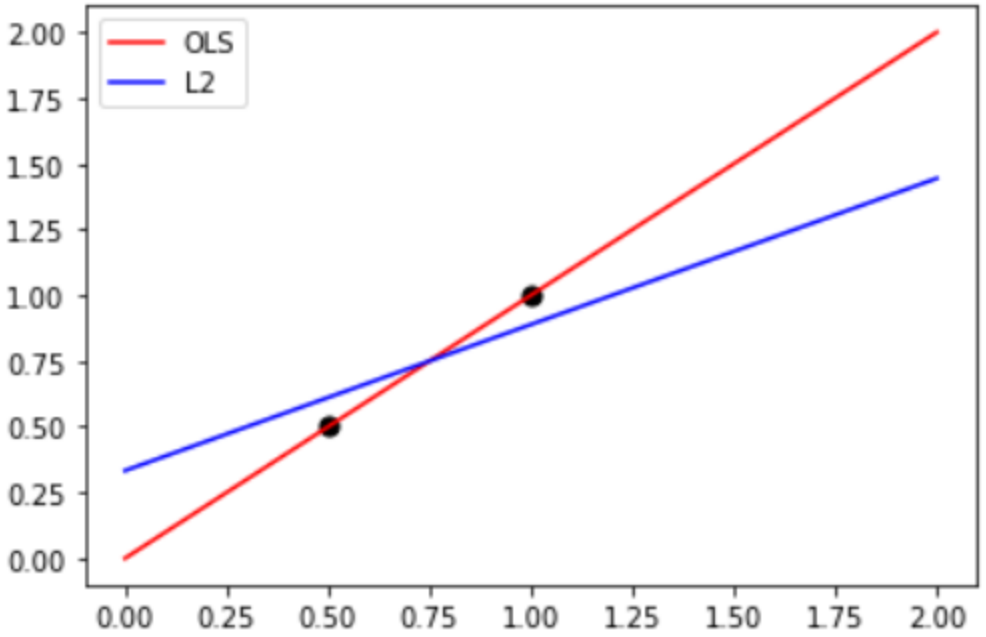
Define and fit OLS and L2 regression,

```
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()
```

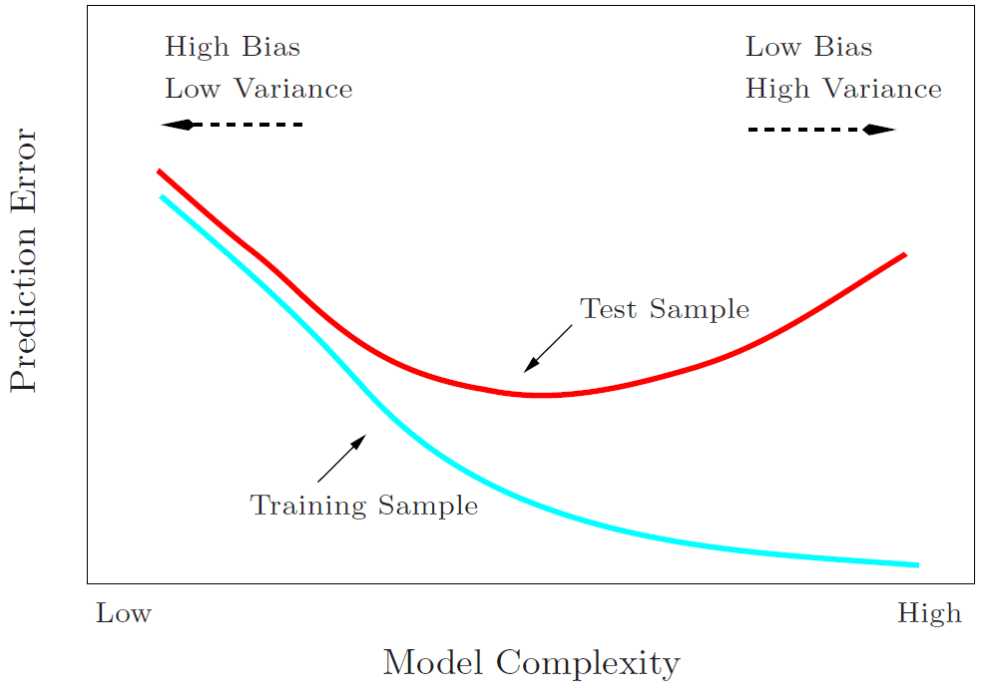


L2 (Ridge) reduces impact of any single data point

Choosing Regularization Strength

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

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \frac{\lambda}{2}\|w\|^2$$



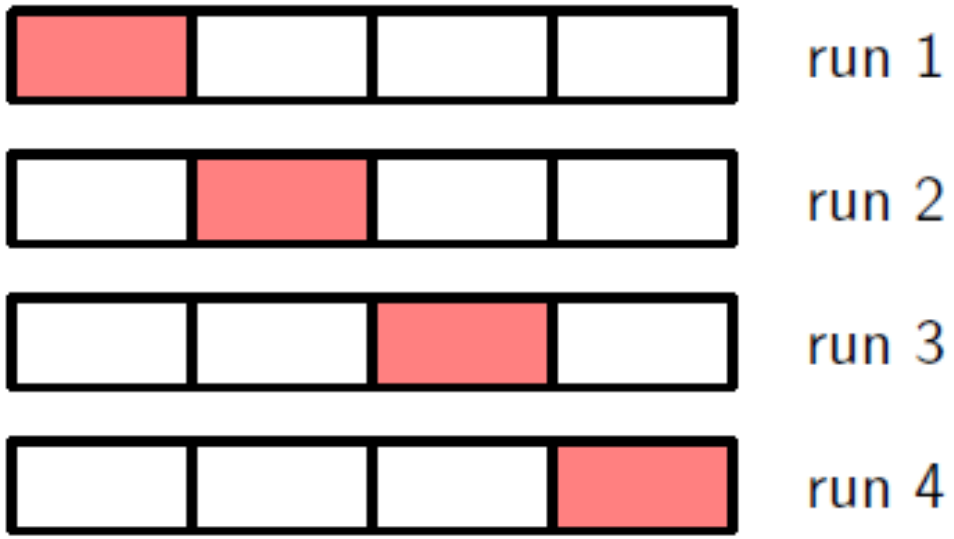
Recall bias/variance tradeoff

Error = Irreducible error + Bias^2 + Variance

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

How should we properly tune λ?

Cross-Validation



N-fold Cross Validation Partition training data into N “chunks” and for each run select one chunk to be validation data

For each run, fit to training data (N-1 chunks) and measure accuracy on validation set. Average model error across all runs.

Drawback Need to perform training N times.

Source: Bishop, C. PRML

Model Selection for Linear Regression

A couple of common metrics for model selection…

Residual Sum-of-squared Errors The total squared residual error on the held-out validation set,

$$\mathrm{RSS} = \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Coefficient of Determination Also called R-squared or R^2. Fraction of variation explained by the model.

Model selection metrics are known as "goodness of fit" measures

Coefficient of Determination R^2

Predicted Variance

Residual Sum-of-Squares

$$R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{SS}} = 1 - \frac{\sum_{i=1}^{N}(y_i - w^T x_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Total variance in dataset

Variance using avg. prediction

Where: $\bar{y} = \frac{1}{N}\sum_i y_i$ is the average output

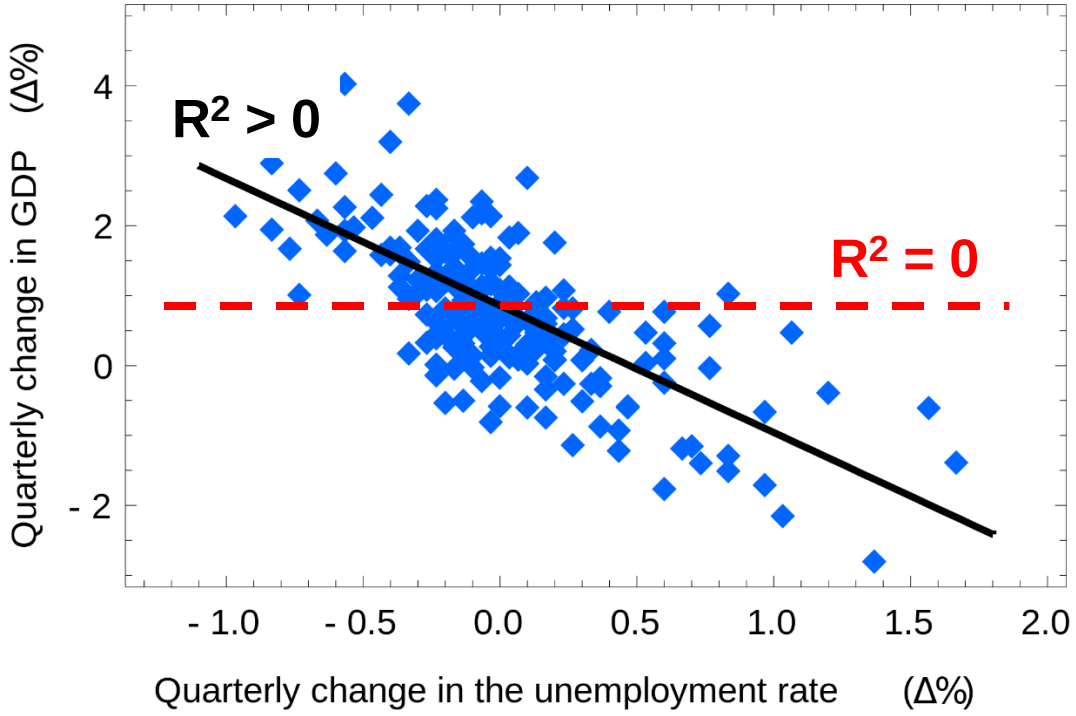
Coefficient of Determination R^2

$$R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{SS}} = 1 - \frac{\sum_{i=1}^{N}(y_i - w^T x_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

Value R^2=1.0 means model explains *all variation* in the data

Value R^2=0 means model is as good as predicting average response

R^2<0 means model worse than predicting average output



“Shrinkage” Feature Selection

Down-weight features that are not useful for prediction…

Quadratic penalty $\lambda\|w\|^2$ down-weights (shrinks) features that are not useful for prediction

Term	LS	Ridge
Intercept	2.465	2.452
lcavol	0.680	0.420
lweight	0.263	0.238
age	−0.141	−0.046
lbph	0.210	0.162
svi	0.305	0.227
lcp	−0.288	0.000
gleason	−0.021	0.040
pgg45	0.267	0.133

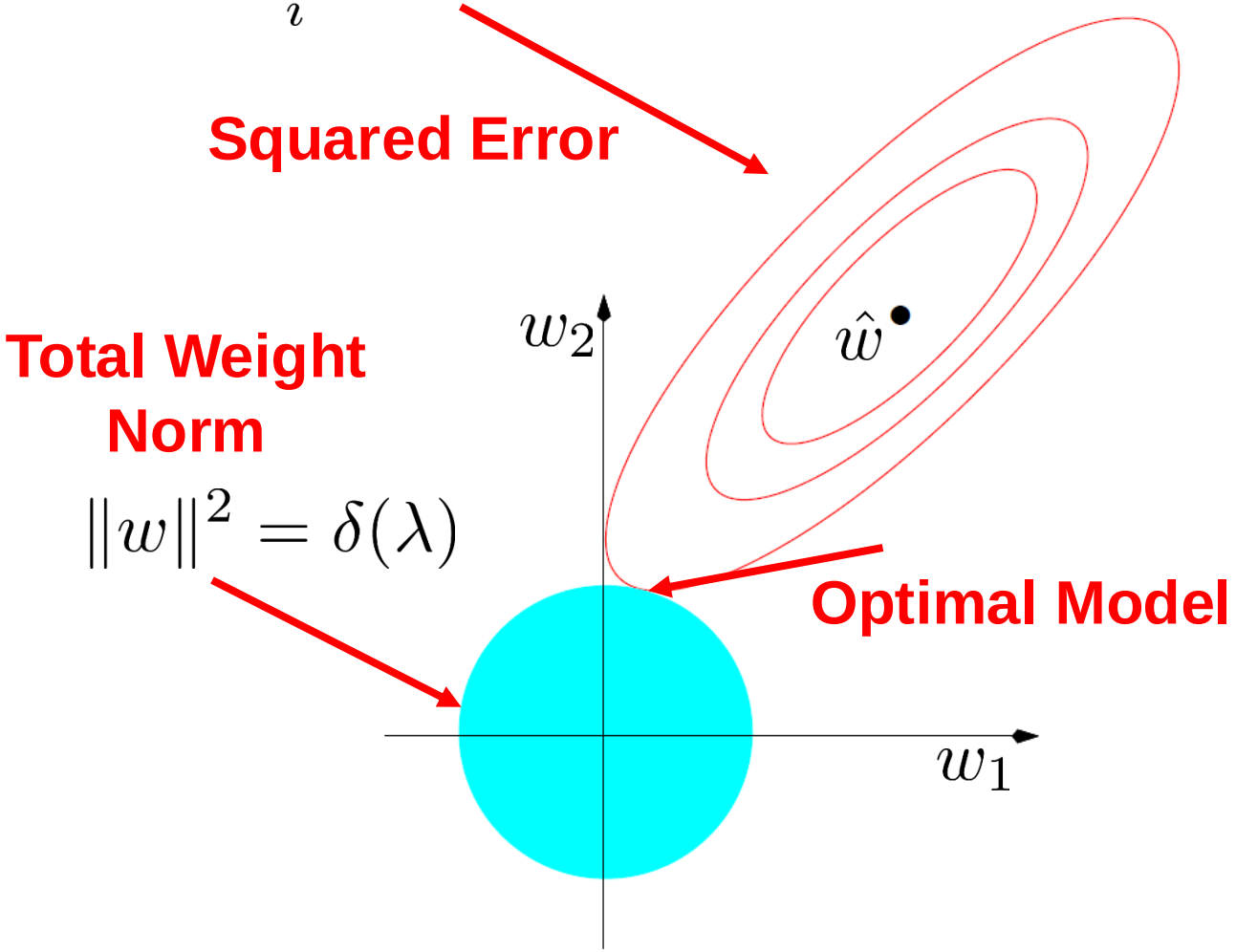
Example *Prostate Cancer Dataset* measures prostate-specific cancer antigen with features: age, log-prostate weight (lweight), log-benign prostate hyperplasia (lbph), Gleason score (gleason), seminal vesical invasion (svi), etc.

L2 regularization learns zero-weight for log capsular penetration (lcp)

[Source: Hastie et al. (2001)]

Constrained Optimization Perspective

$$\min_w \sum_i (y_i - w^T x)^2$$



Intuition Find best model (lowest RSS) given constraint on total feature weights…

There exists a mathematically equivalent formulation for some function $\delta(\lambda)$

L2 penalized regression rarely learns feature weight that are *exactly zero…*

[Source: Hastie et al. (2001)]

Regularized Least Squares

Ordinary least-squares estimation (no regularizer),

$$w^{\mathrm{OLS}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

L2-regularized Least-Squares (Ridge)

Quadratic Penalty

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \frac{\lambda}{2} \|w\|^2$$

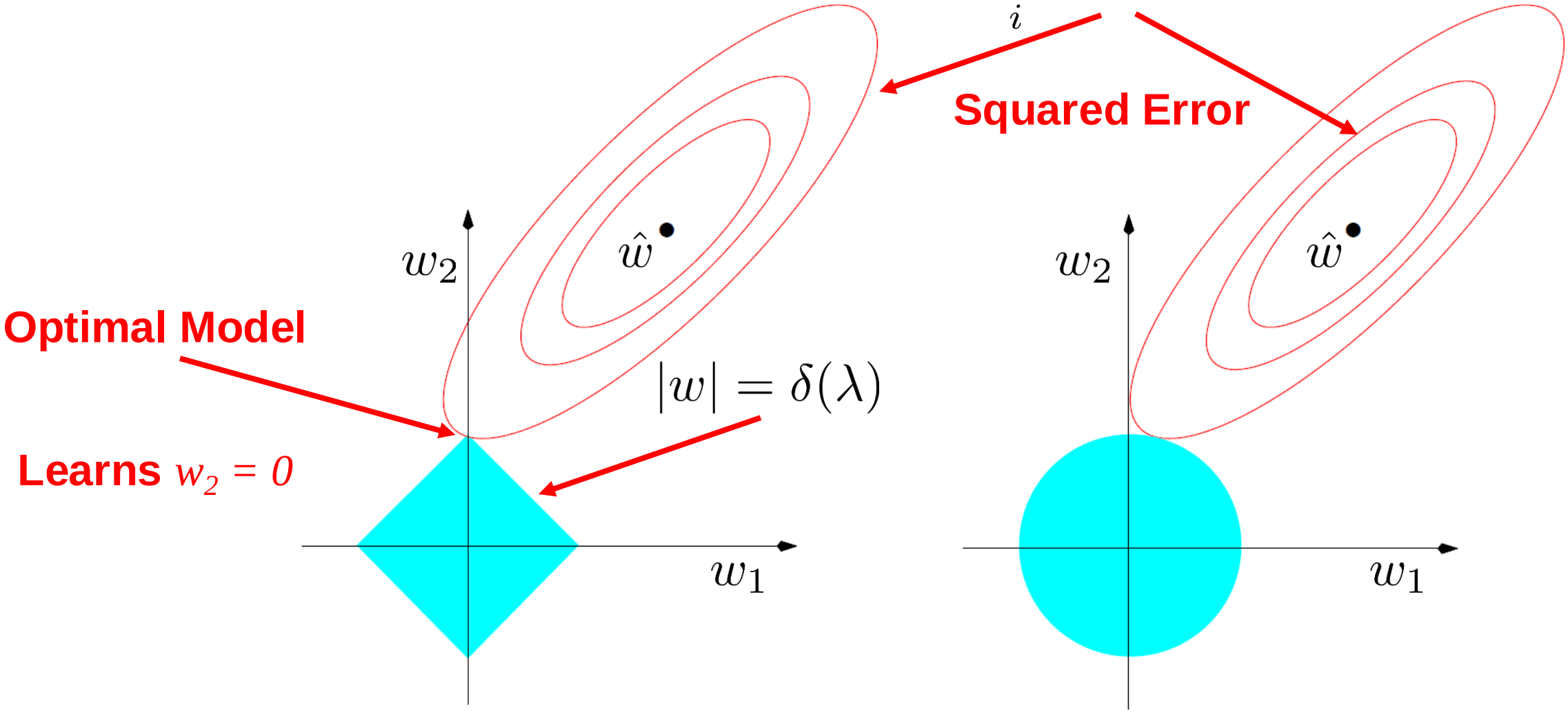
L1-regularized Least-Squares (LASSO)

Absolute Value (L1) Penalty

$$w^{\mathrm{L1}} = \arg\min_{w} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \lambda |w|$$

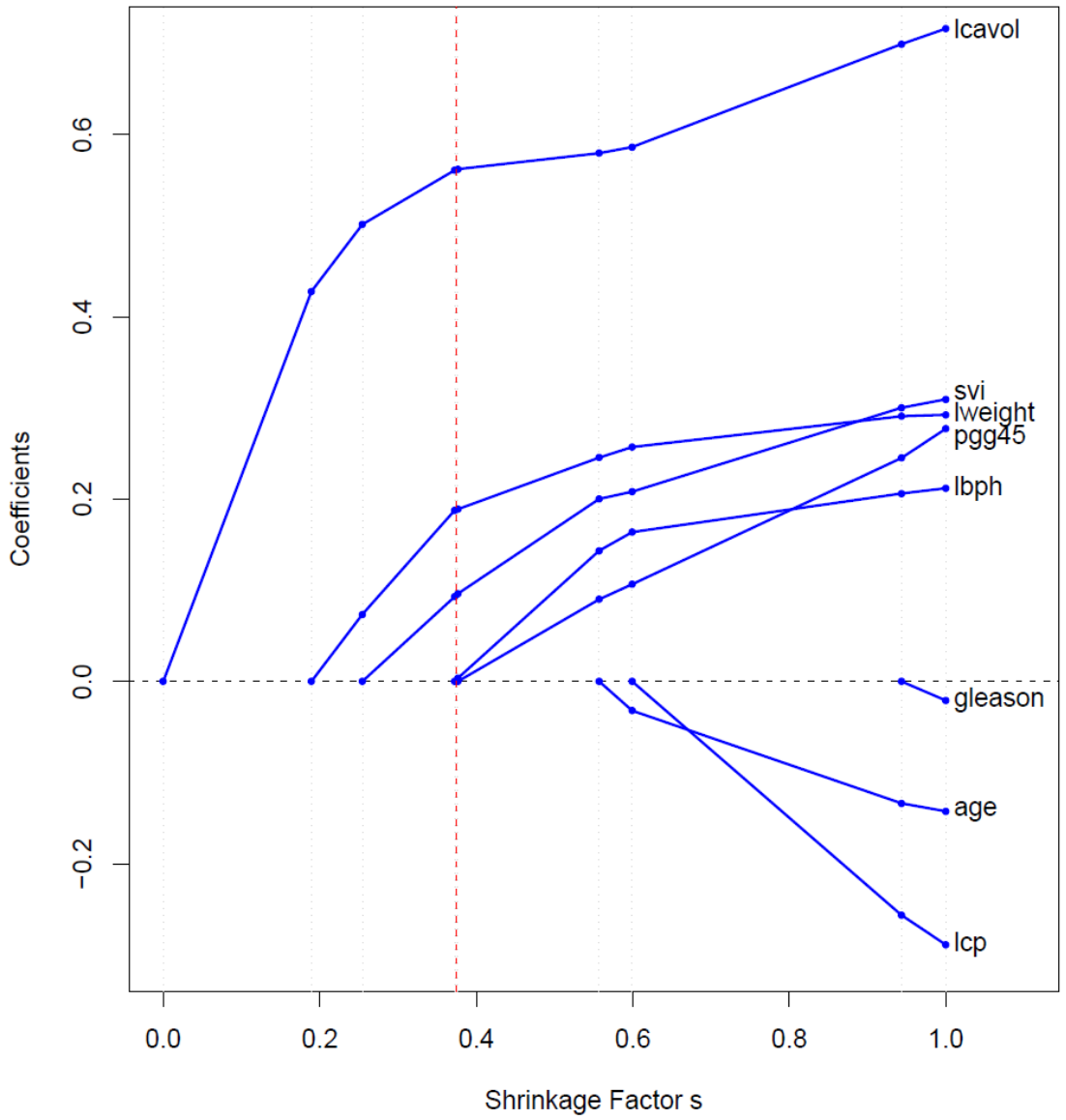
L1 Regularized Least-Squares

$$\min_w \sum_i (y_i - w^T x)^2$$



Able to zero-out weights that are not predictive…

Feature Weight Profiles

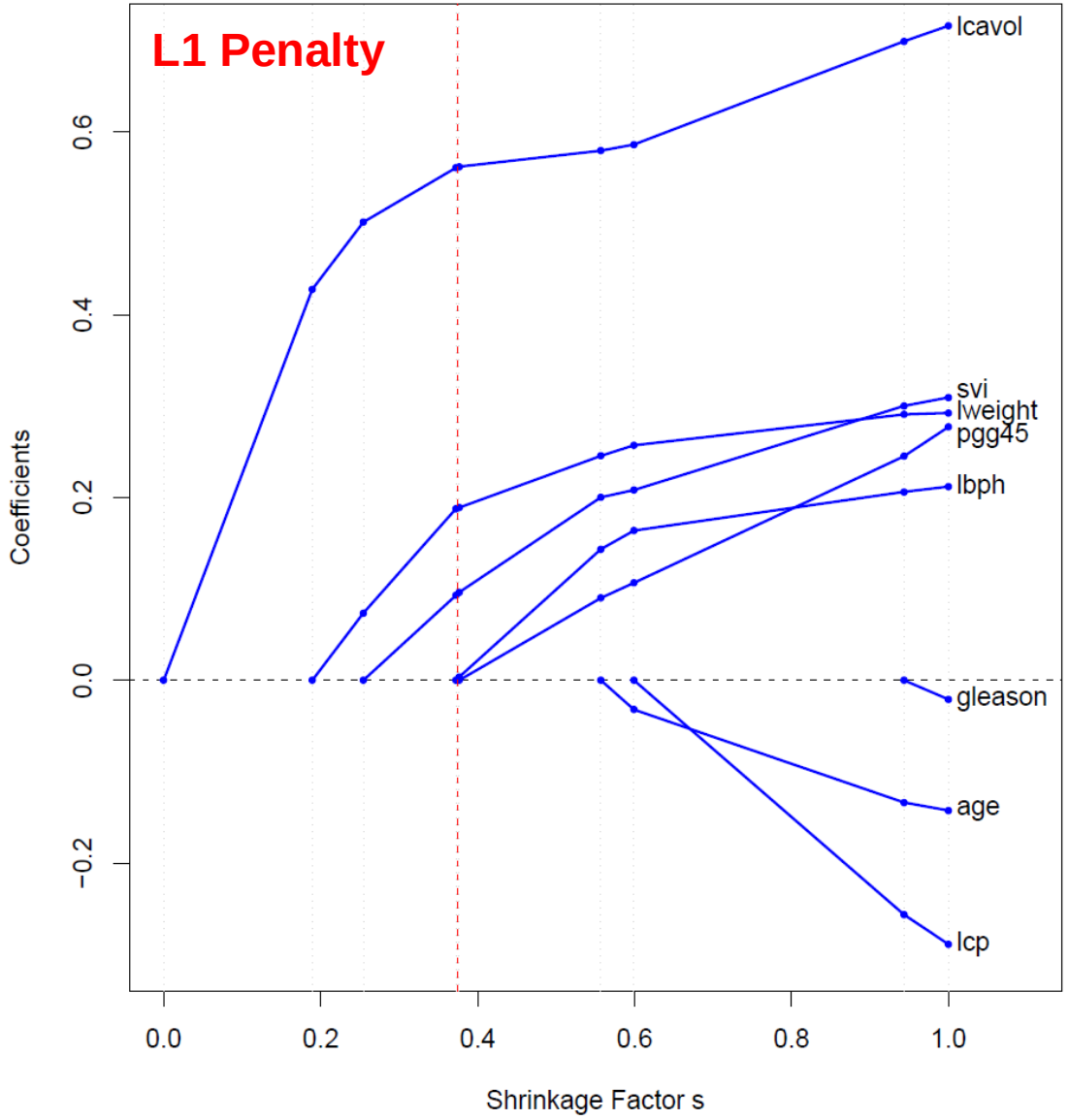


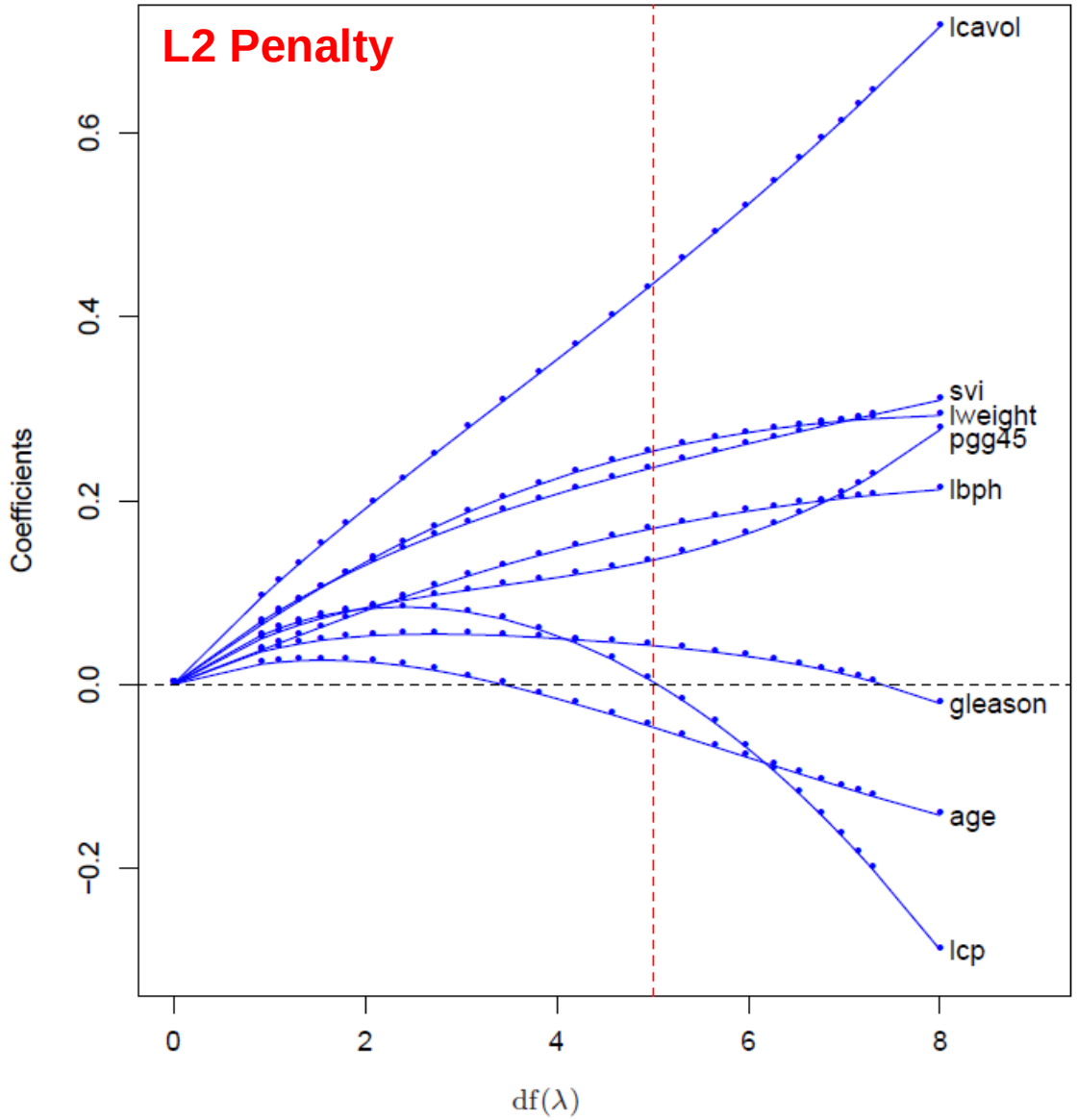
Varying regularization parameter moderates *shrinkage factor*

For moderate regularization strength weights for many features go to zero

- Induces *feature sparsity*
- Ideal for high-dimensional settings
- Gracefully handles p>N case, for p features and N training data

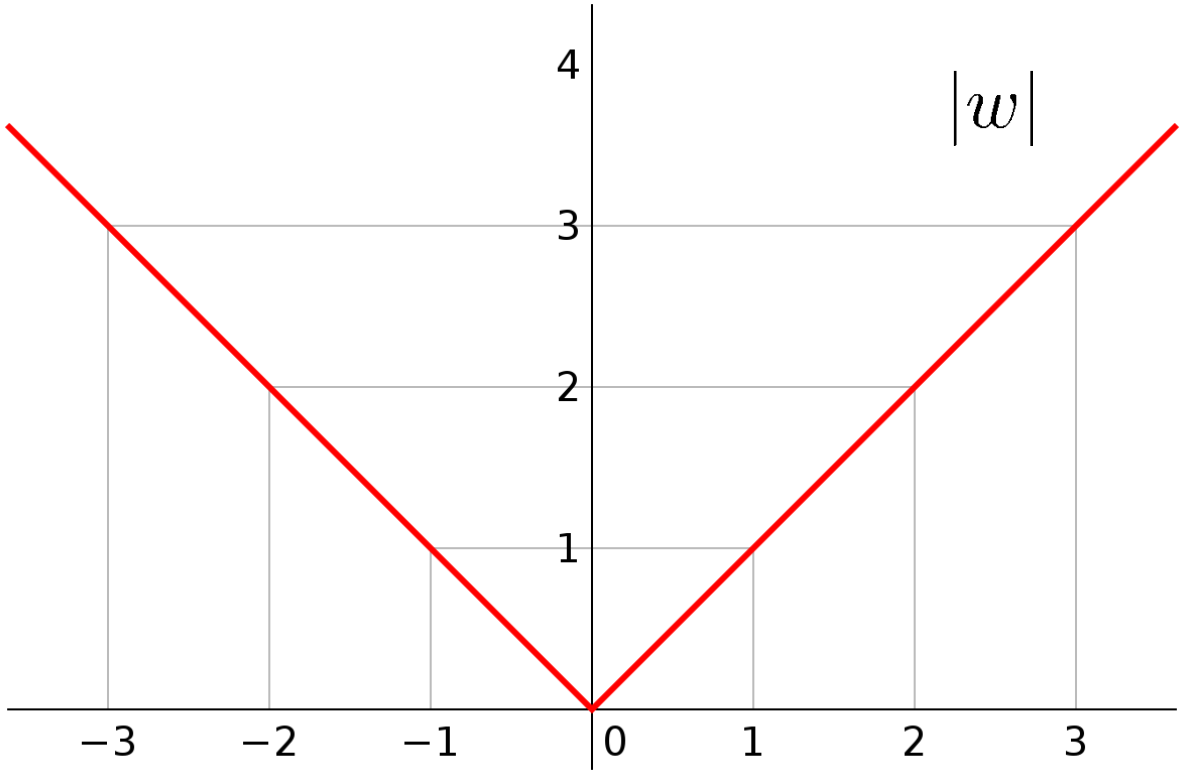
Feature Weight Profiles





Learning L1 Regularized Least-Squares

$$w^{\mathrm{L1}} = \arg\min_{\theta} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \lambda |w|$$



Not differentiable…

$$\frac{d}{dw}|w|$$

…doesn't exist at w=0

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

Learning L1 Regularized Least-Squares

- Not differentiable, no closed-form solution
- But it is convex! Can be solved by *quadratic programming* (beyond the scope of this class…)
- Efficient optimization algorithms exist
- *Least Angle Regression* (LAR) computes full solution path for a range of values λ
- Can be solved as efficiently as L2 regression

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*

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]

Computes solution using coordinate descent

sklearn.linear_model.LassoLarsCV

class sklearn.linear_model.LassoLarsCV(*, *fit_intercept=True, verbose=False, max_iter=500, normalize='deprecated', precompute='auto', cv=None, max_n_alphas=1000, n_jobs=None, eps=2.220446049250313e-16, copy_X=True, positive=False*) ¶ [source]

Uses *least angle regression* (LARS) to compute solution path

L1 Regression Cross-Validation

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

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

or

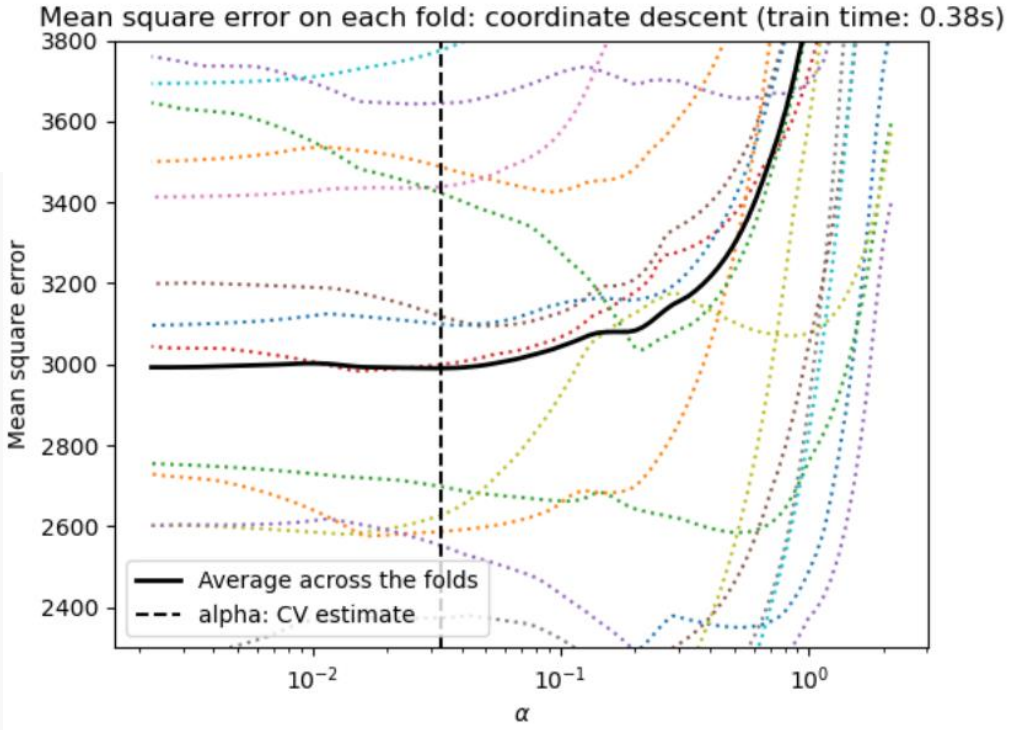
```
model = LassoLarsCV(cv=20, normalize=False).fit(X, y)
```

Plot solution path 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 alphas_

Learned alpha_ (no “s”… annoying…)

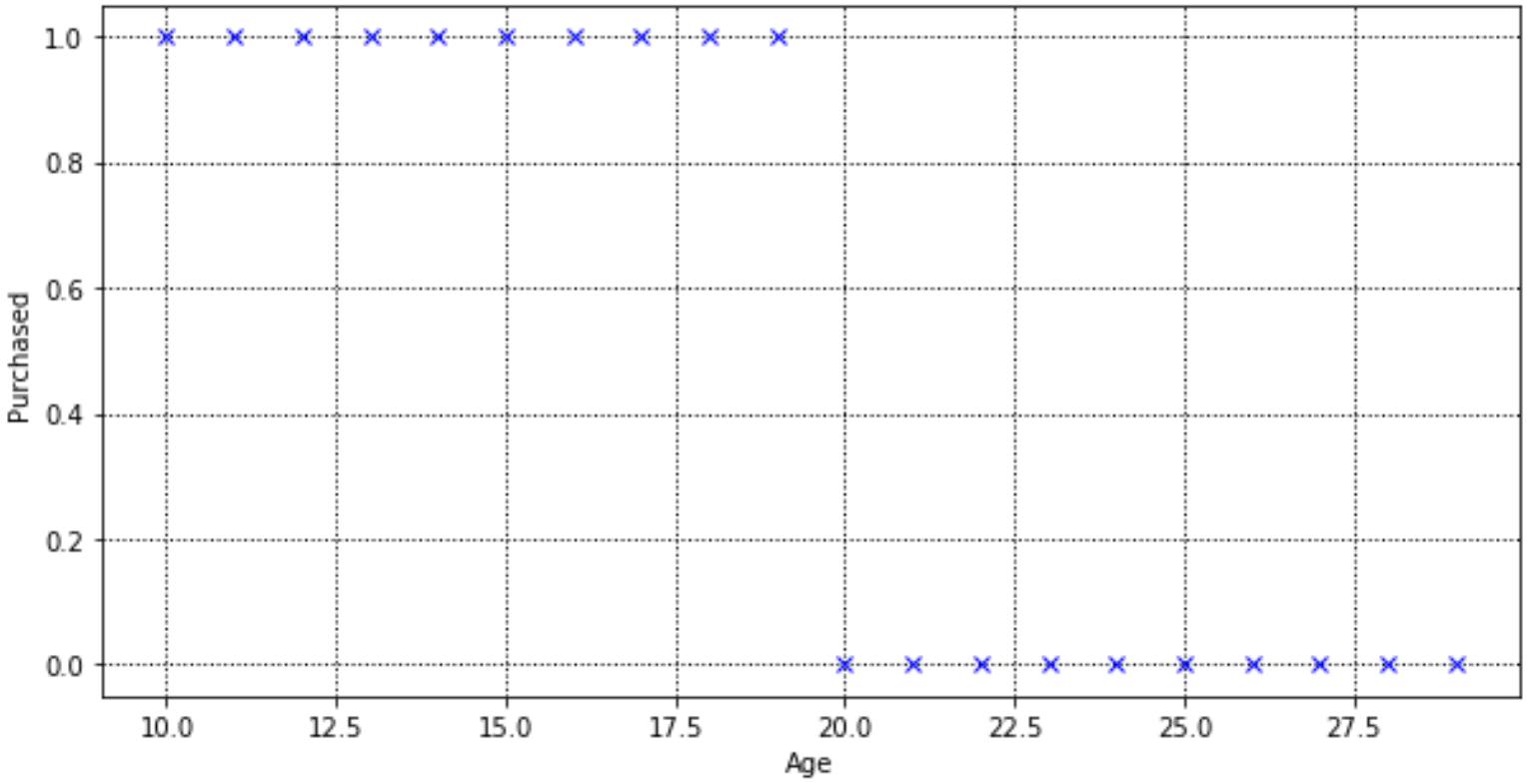


Outline

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

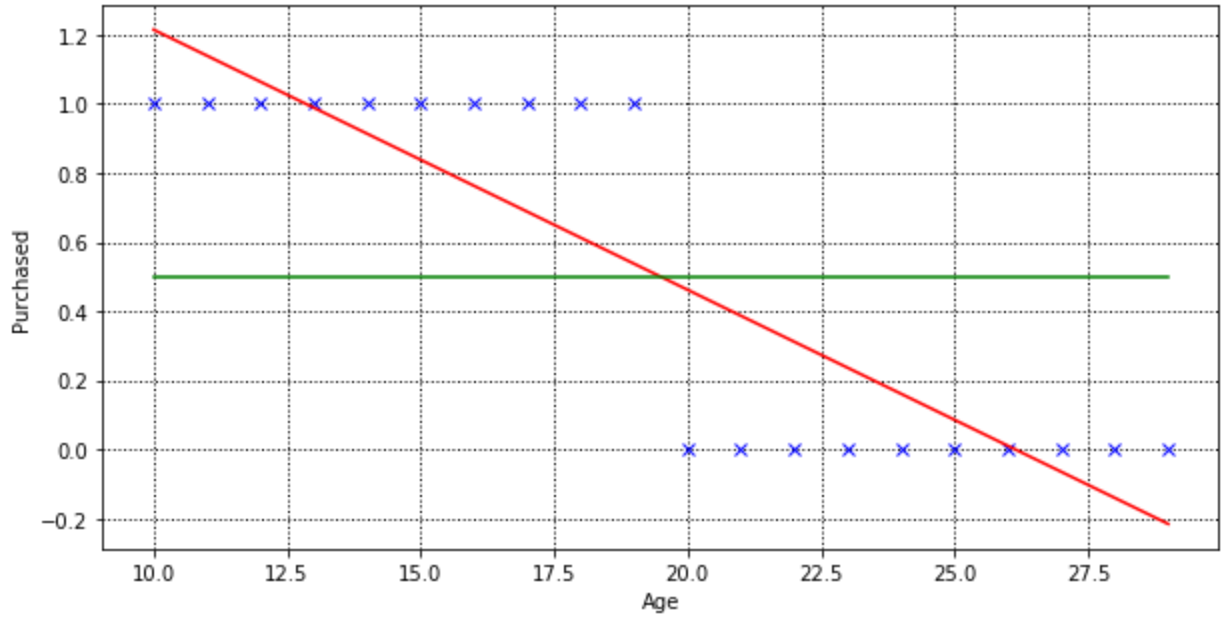
Classification as Regression

Suppose our response variables are binary y={0,1}. How can we use linear regression ideas to solve this classification problem?



https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Classification as Regression



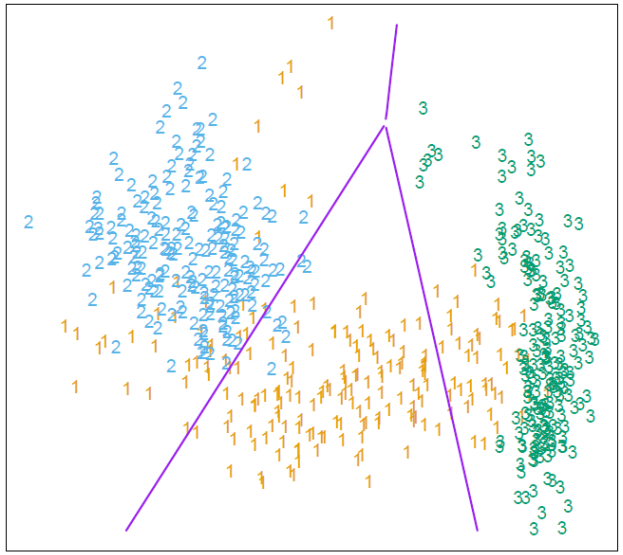
Idea Fit a regression function to the data (red). Classify points based on whether they are *above* or *below* the midpoint (green).

$$\text{Class} = \begin{cases} 0 & \text{if } w^T x < 0.5 \\ 1 & \text{if } w^T x >= 0.5 \end{cases}$$

- This is a *discriminant* function, since it discriminates between classes
- It is a linear function and so is a *linear discriminant*
- Green line is the *decision boundary* (also linear)

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Multiclass Classification as Regression



Suppose we have K classes. Training outputs for each class are a set of *indicator vectors*,

$$Y = (Y_1, \ldots, Y_K)$$

With $Y_k = 1$ if class k, e.g. Y=(0,0,…,1,0,0).

For N training inputs create NxK matrix of outputs $\mathbf{Y}$ and solve,

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

W is DxK matrix of K linear regression models, one for each class

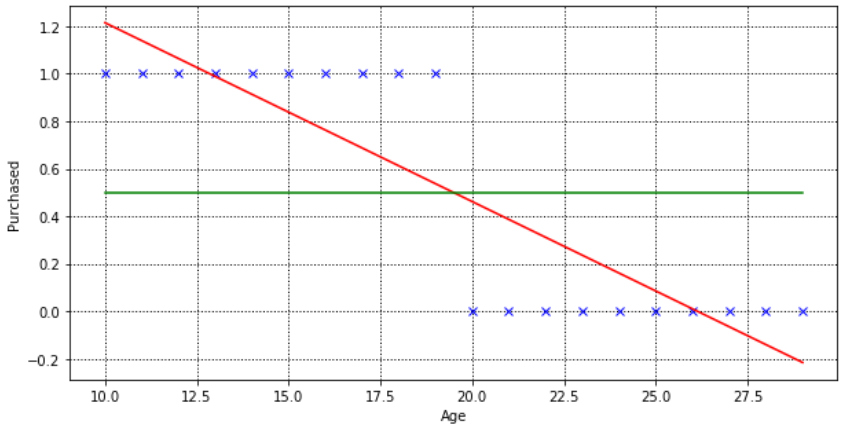
- Compute fitted output $f(x) = [x^T\mathbf{W}]^T$ a K-vector
- Identify largest component and classify as,

$$C = \arg\max_k f_k(x)$$

This is an instance of multi-output linear regression

[Image: Hastie et al. (2001)]

Linear Probability Models



$$\text{Class} = \begin{cases} 0 & \text{if } w^T x < 0.5 \\ 1 & \text{if } w^T x >= 0.5 \end{cases}$$

Binary Classification Linear model approximates probability of class assignment,

$$y(x) = w^T x \approx p(\text{Class} = 1 | w, x)$$

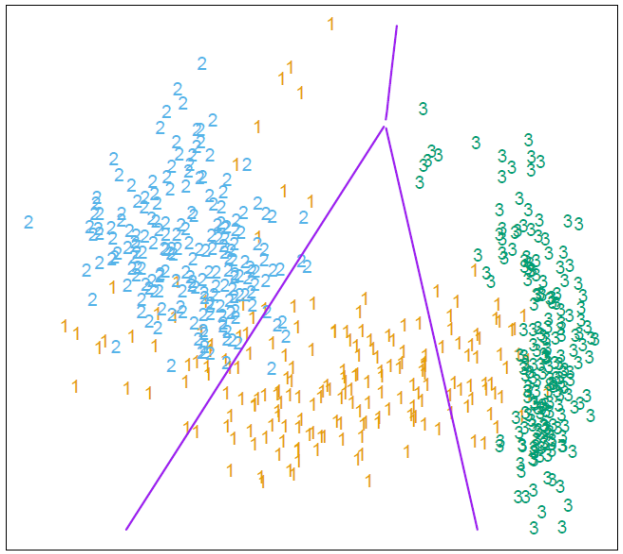
Multiclass Classification Multiple decision boundaries, each approximated by the class-specific linear model,

$$\hat{f}_k(x) = W_{k:} x$$

Where $W_{k:}$ is kth row

Approximates probability of class assignment,

$$\hat{f}_k(x) \approx p(\text{Class} = k \mid x)$$



What's the rationale?

Recall the linear regression model,

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

So linear regression models the expected value,

$$\mathbf{E}[y \mid x] = w^T x$$

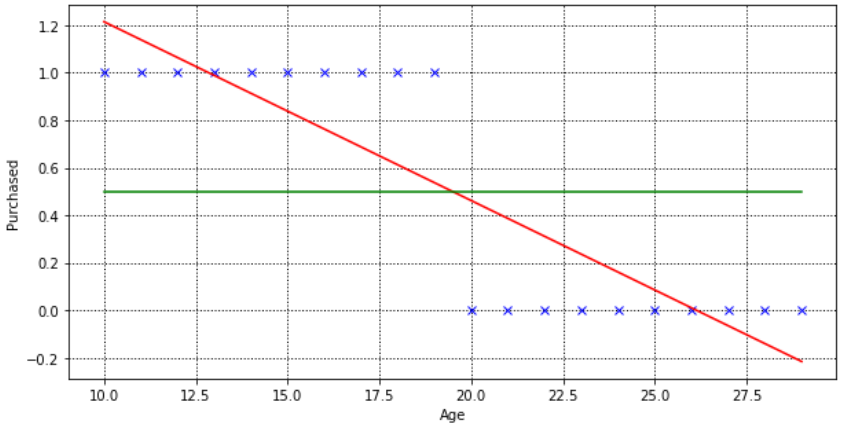
We can call this approach **least squares classification**

For *discrete* values we have that,

$$\mathbf{E}[y_k \mid x] = f_k(x) = p(\text{Class} = k \mid x)$$

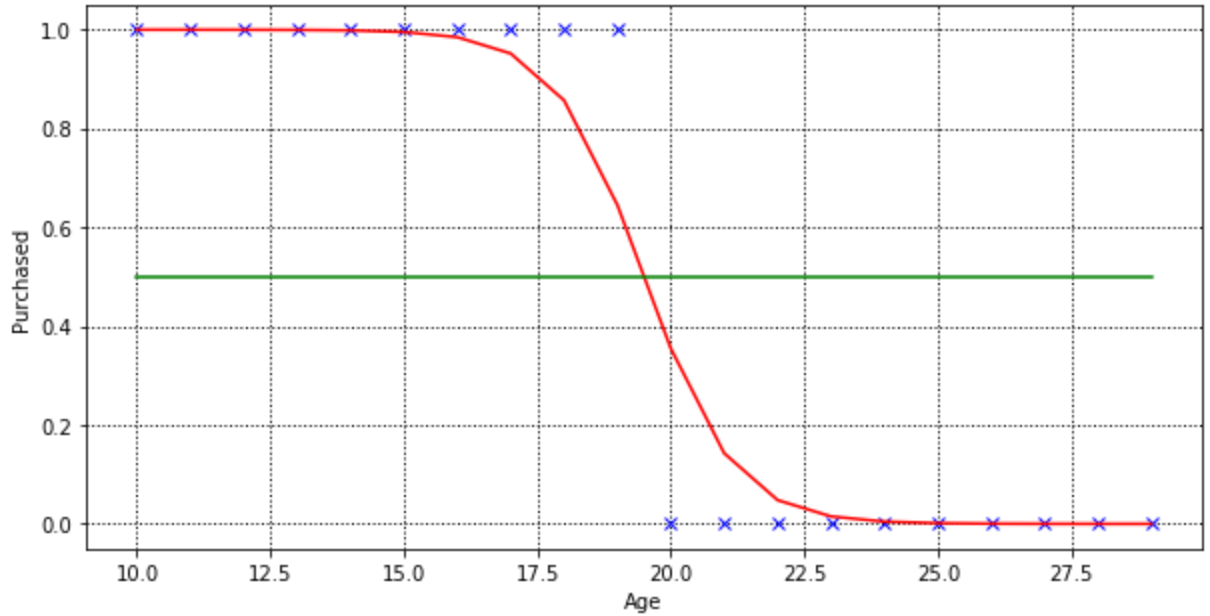
Can easily verify that they sum to 1,

$$\sum_{k=1}^{K} f_k(x) = 1$$



But they are not guaranteed to be positive!

Logistic Regression



Idea Distort the response variable in some way to map to [0,1] so that it is actually a probability:

$$y(x) = \sigma(w^T x)$$

Uses the *logistic* function,

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

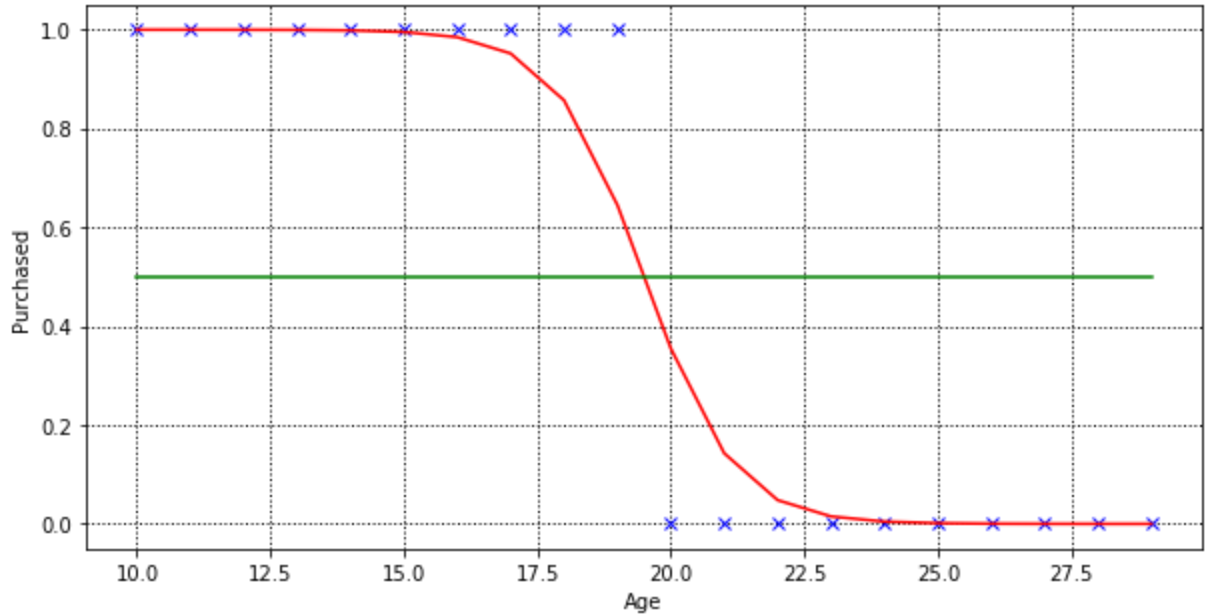
- Logistic function is a type of *sigmoid* or *squashing function*, since it maps any value to the range [0,1]

- Predictor variable now actually maps to a valid probability mass function (PMF),

$$y(x) = \sigma(w^T x) = p(Class = 1|w, x)$$

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Logistic Regression : Decision Boundary



Binary classification decisions are based on the *posterior odds ratio,*

$$\frac{p(C=1 \mid x)}{p(C=0 \mid x)}$$

If this ratio is greater than 1.0 then classify as C=1, otherwise C=0

In practice, we use the (natural) logarithm of the posterior odds ratio,

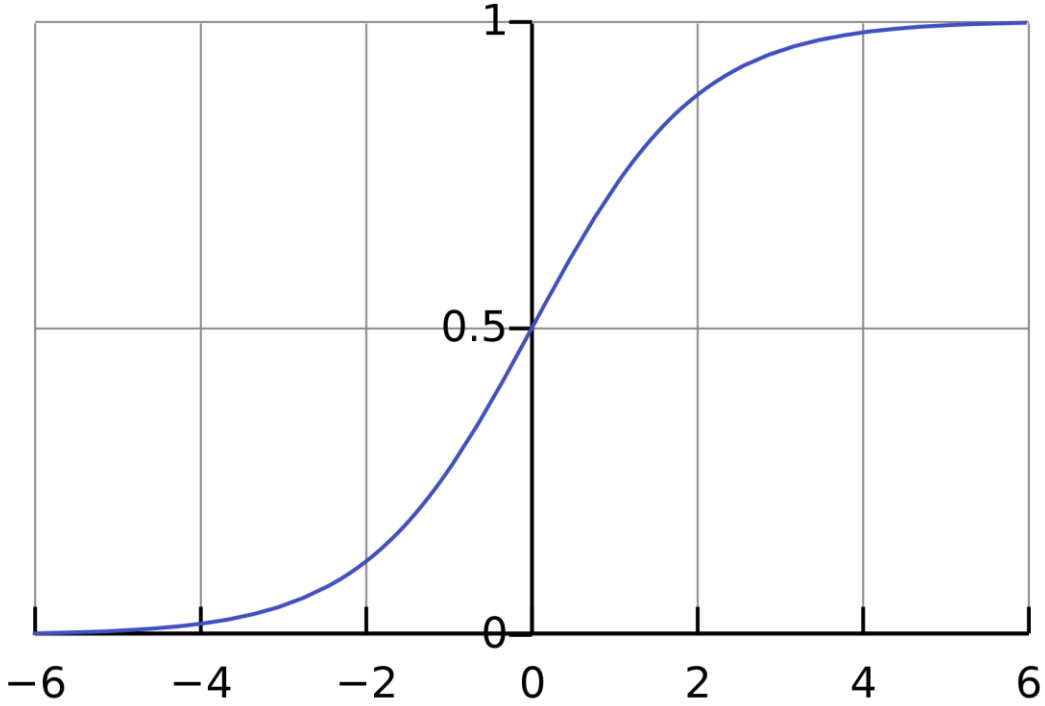
$$\log \frac{p(C=1 \mid x)}{p(C=0 \mid x)} = w^T x$$

This is a *linear decision boundary*

Logistic regression is a *linear classifier*

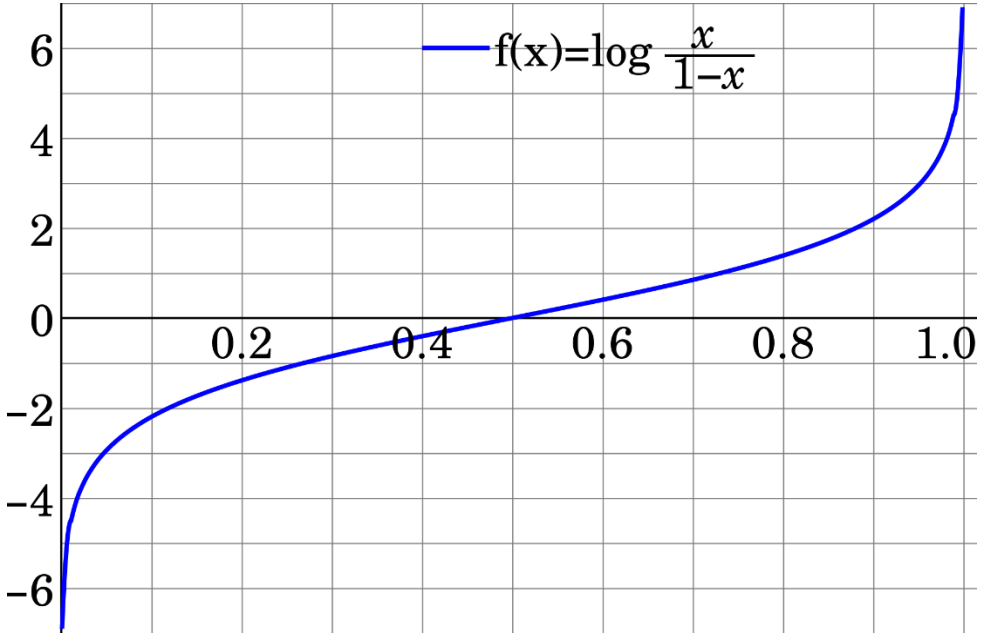
Logistic vs. Logit Transformations

Logistic Function



Maps $(-\infty, \infty)$ to [0,1]

Logit Function



Maps [0,1] to $(-\infty, \infty)$

Logistic also widely used in Neural Networks – for classification last layer is typically just a logistic regression

Logistic vs. Logit Transformations

Logistic function maps the linear regression to the interval [0,1],

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

Logit function is defined for probability values p in [0,1] as,

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

Logit is the *inverse* of the logistic function,

$$\text{logit}(\sigma(w^T x)) = w^T x$$

Logit is also the log-likelihood ratio, and thus decision boundary for our binary classifier

Multiclass Logistic Regression

Classification decision based on log-ratio compared to final class,

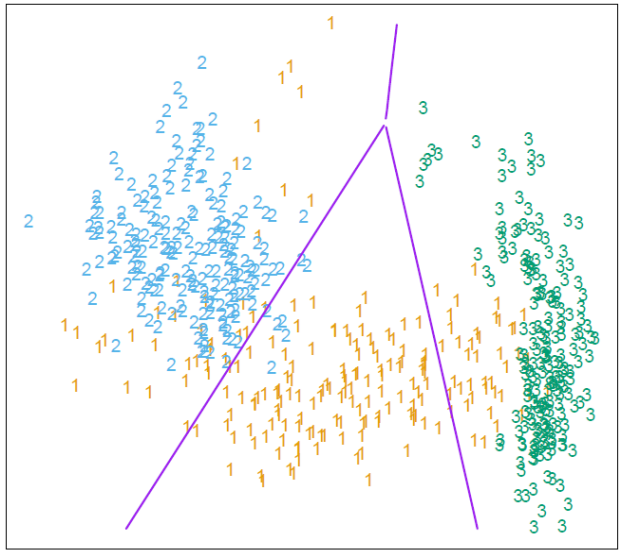
K-1 log-odds (or logit) transformations ensures probabilities sum to 1

$$\log \frac{p(C=1 \mid x)}{p(C=K \mid x)} = w_1^T x$$

$$\log \frac{p(C=2 \mid x)}{p(C=K \mid x)} = w_2^T x$$

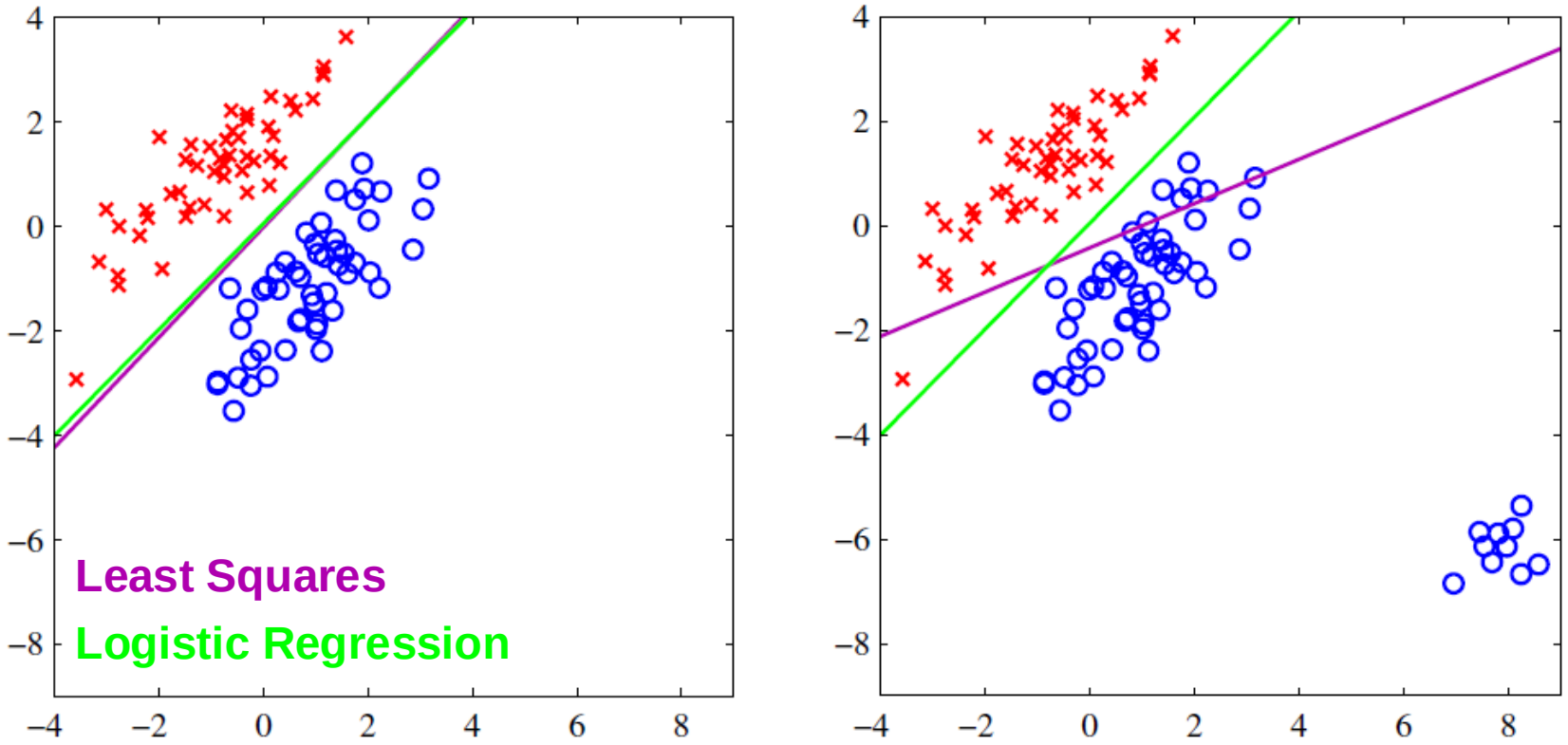
$$\vdots$$

$$\log \frac{p(C=K-1 \mid x)}{p(C=K \mid x)} = w_{K-1}^T x$$



Choice of denominator class is arbitrary, but use K by convention

Least Squares vs. Logistic Regression

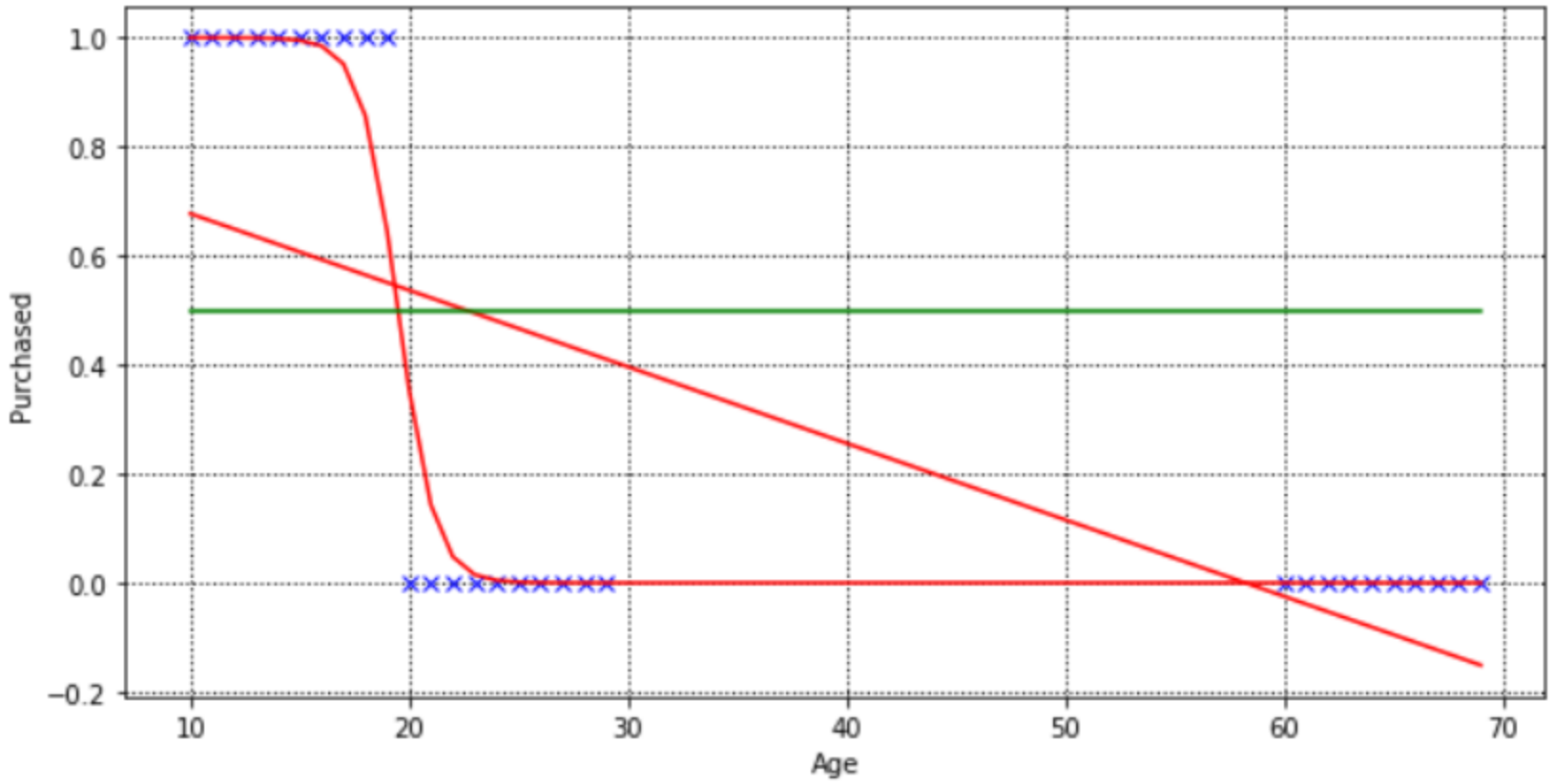


- Both models learn a linear decision boundary
- Least squares can be solved in closed-form (convex objective)
- Least squares is sensitive to outliers (need to do regularization)

[Source: Bishop “PRML”]

Least Squares vs. Logistic Regression

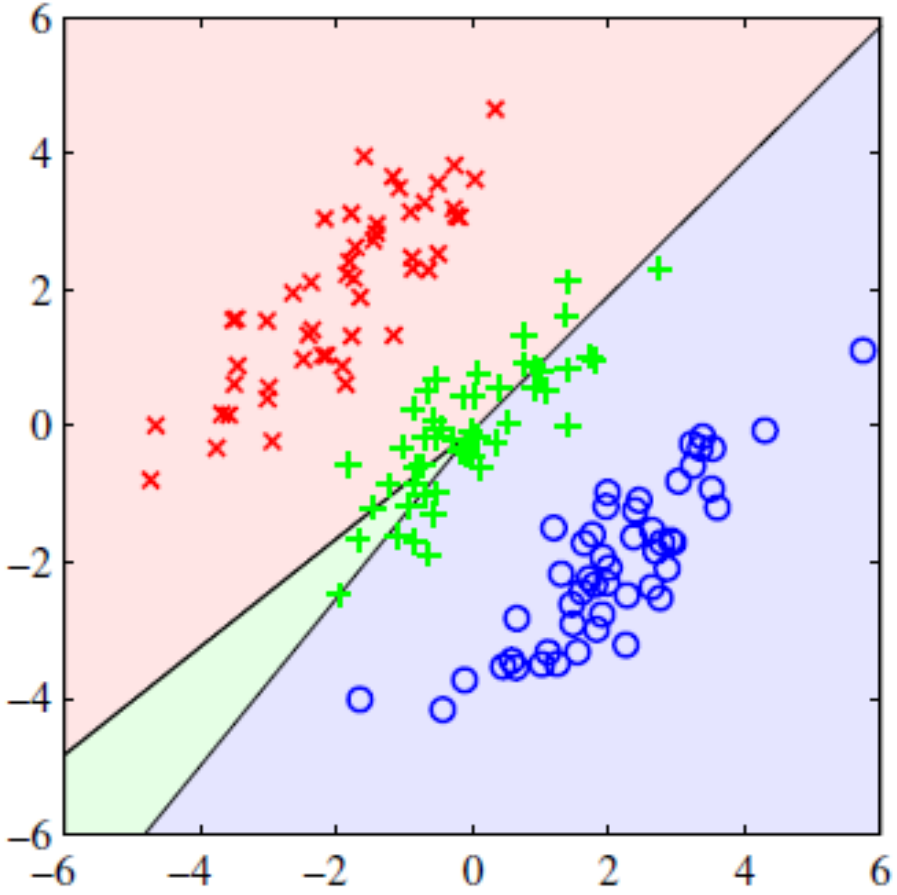
Similar results in 1-dimension



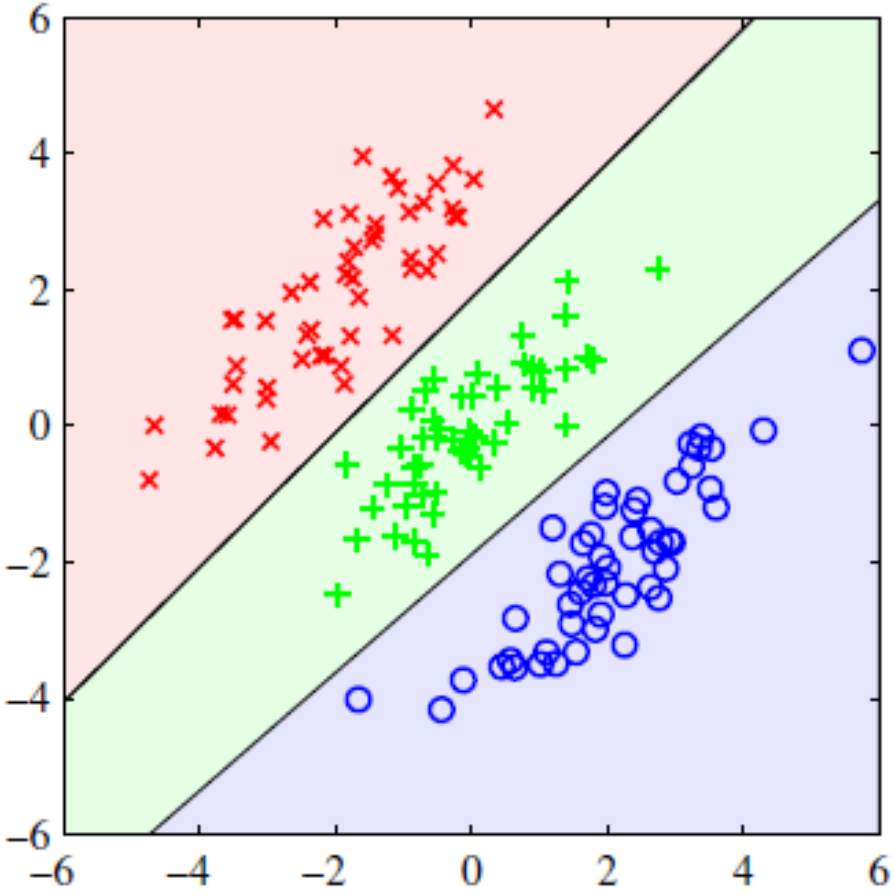
https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Least Squares vs. Logistic Regression

Least Squares



Logistic Regression



[Source: Bishop “PRML”]

Fitting Logistic Regression

Fit by maximum 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,

$$\begin{aligned}
\mathcal{L}_N(w) &= \sum_{i=1}^{N} \log p(y_i \mid x_i, w) \\
&= \sum_{i} \{y_i \log p(y_i = 1 \mid x_i, w) + (1 - y_i) \log p(y_i = 0 \mid x_i, w)\} \\
&= \sum_{i} \left\{ y_i w^T x_i - \log \left(1 + e^{w^T x_i}\right) \right\}
\end{aligned}$$

Fitting Logistic Regression

$$w^{\mathrm{MLE}} = \arg\max_{w} \sum_{i} \left\{ y_i w^T x_i - \log\left(1 + e^{w^T x_i}\right) \right\}$$

Computing the derivatives with respect to each element w_d,

$$\frac{\partial \mathcal{L}}{\partial w_d} = \sum_{i} x_{di} \left(y_i - \frac{e^{w^T x_i}}{1 + e^{w^T x_i}} \right) = 0$$

- For D features this gives us D equations and D unknowns
- But equations are nonlinear and can't be solved directly
- Need to use gradient-based optimization to solve (Newton's method)
- Beyond scope of this class; but know that it is an iterative process

Iteratively Reweighted Least Squares

Given some estimate of the weights w^{old} update by solving,

$$w^{\text{new}} = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z}$$

Design Matrix (NxD)

NxN Diagonal Weight matrix

Where z is the gradient direction,

$$\mathbf{z} = \mathbf{X} w^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$$

P(y=1|x) for each training point

Essentially solving a *reweighted* version of least squares,

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

Each iteration changes W and p so need to resolve

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.

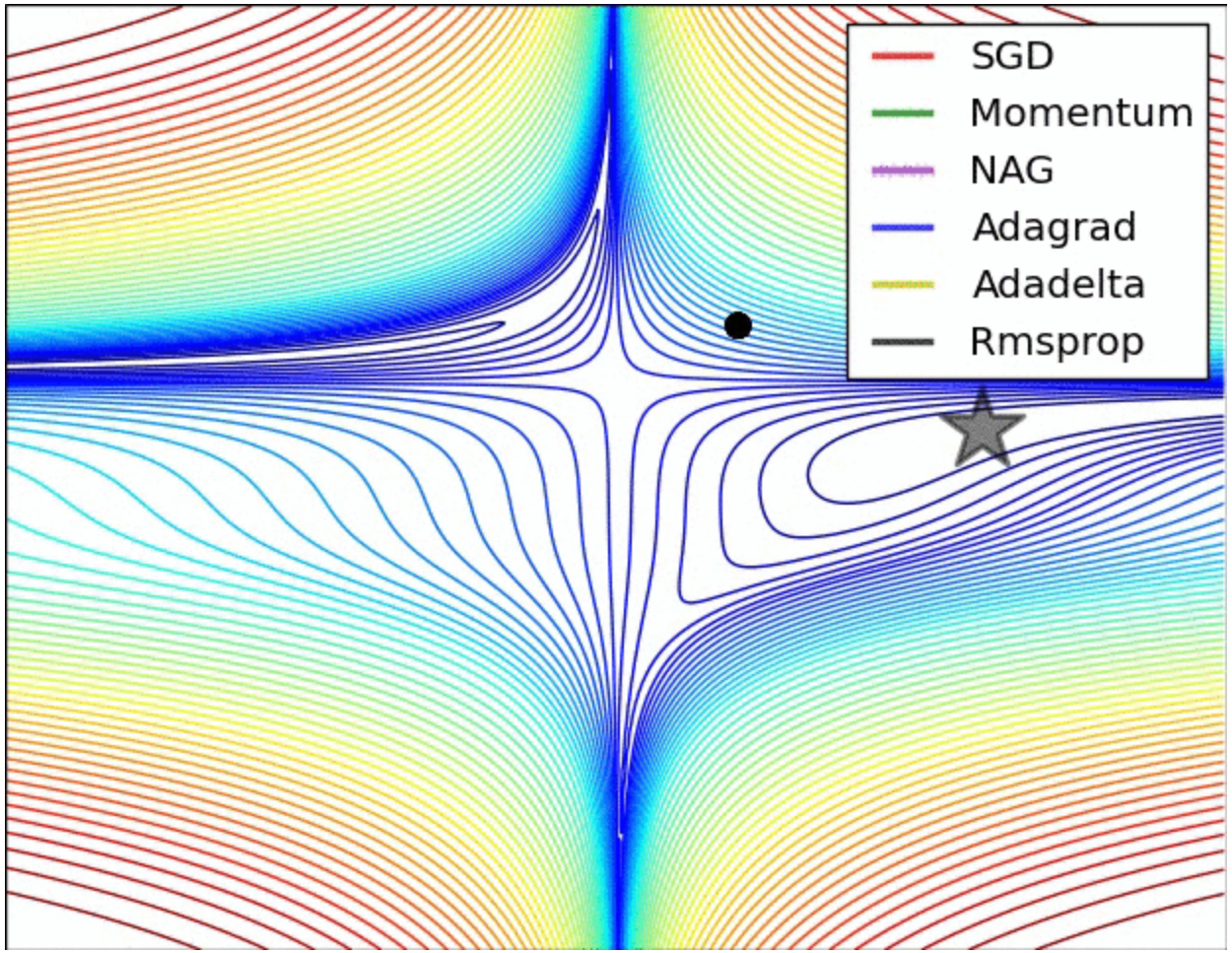
tol : *float, default=1e-4*

Tolerance for stopping criteria.

C : *float, default=1.0*

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

Choice of Optimizer



Since Logistic regression requires an optimizer, there are more parameters to consider

The choice of optimizer and parameters can effect time to fit model (especially if there are many features)

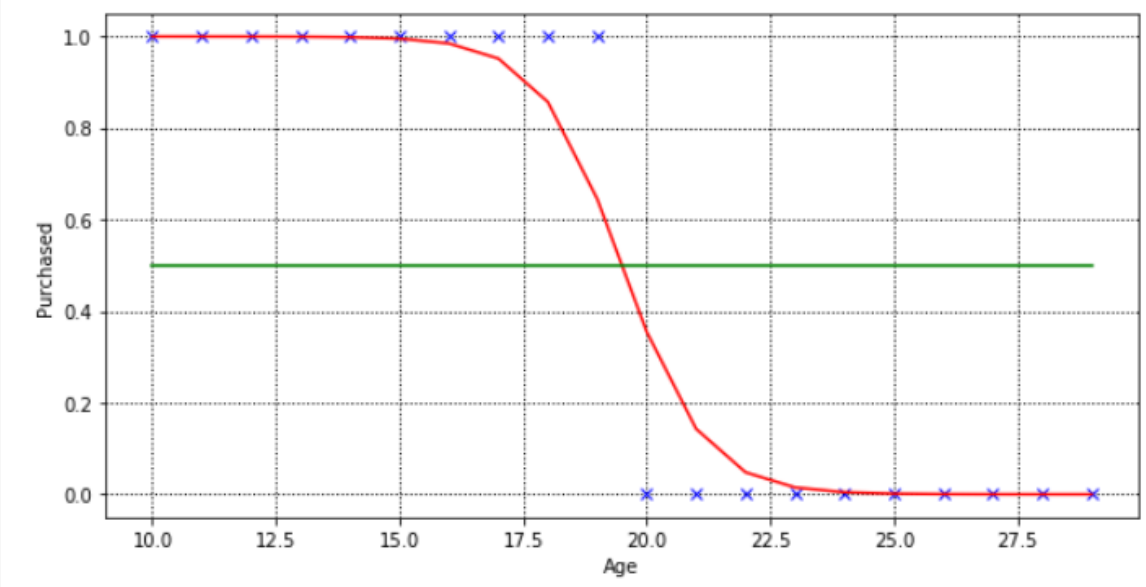
https://www.datasciencecentral.com/profiles/blogs/an-overview-of-gradient-descent-optimization-algorithms

Scikit-Learn Logistic Regression

```
_ = 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 (just a number in binary case)

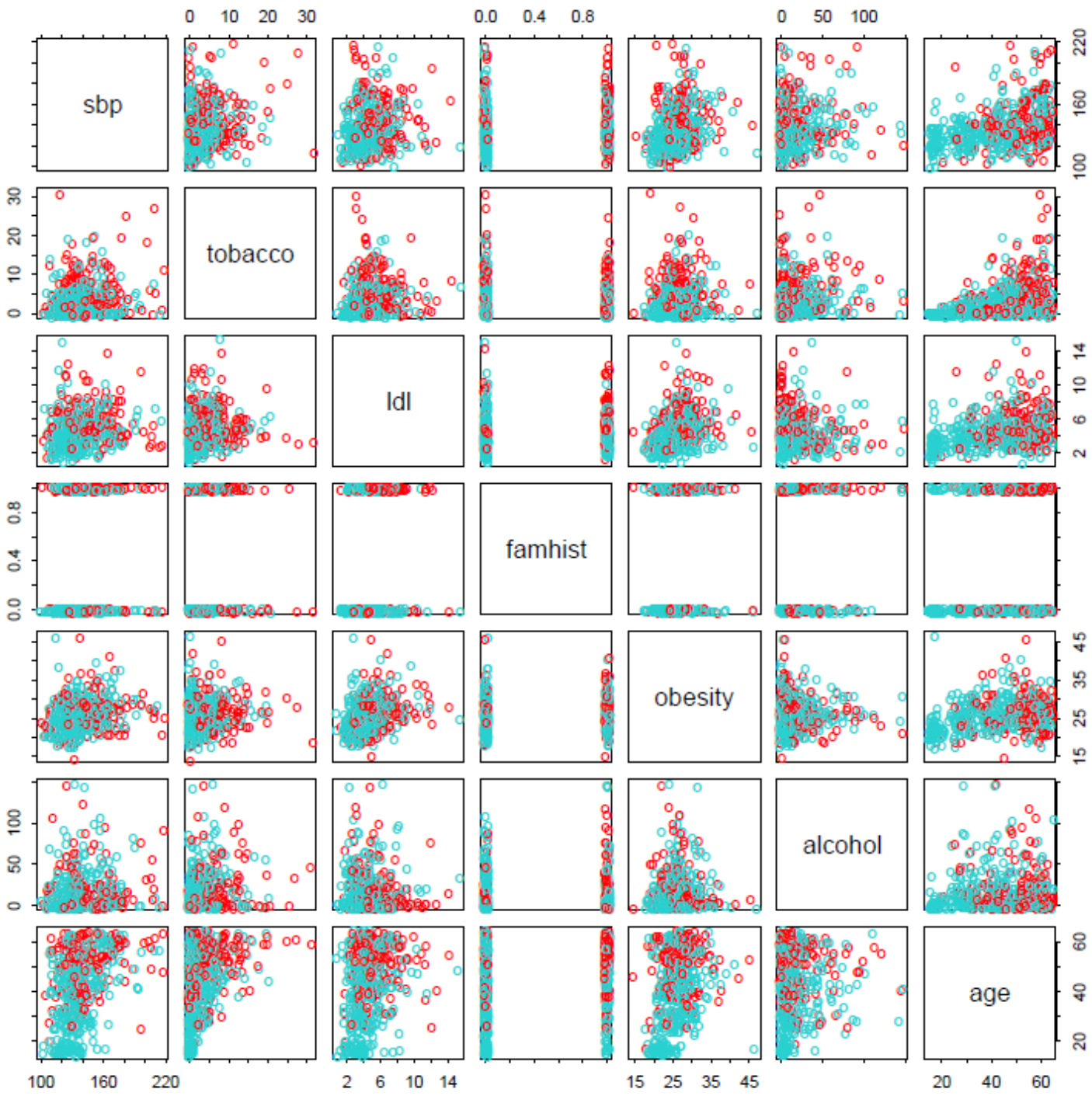
https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Using Logistic Regression

The role of Logistic Regression differs in ML and Data Science,

- In Machine Learning we use Logistic Regression for building predictive classification models
- In Data Science we use it for understanding how features relate to data classes / categories

Example South African Heart Disease (Hastie et al. 2001)
Data result from Coronary Risk-Factor Study in 3 rural areas of South Africa. Data are from white men 15-64yrs and response is presence/absence of *myocardial infraction (MI).* How predictive are each of the features?



Looking at Data

Each scatterplot shows pair of risk factors. Cases **with** MI (**red**) and **without** (**cyan**)

Features

- Systolic blood pressure
- Tobacco use
- Low density lipoprotein (ldl)
- Family history (discrete)
- Obesity
- Alcohol use
- Age

[Source: Hastie et al. (2001)]

Example: African Heart Disease

	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

Fit logistic regression to the data using MLE estimate via iteratively reweighted least squares

Standard error is estimated standard deviation of the learned coefficients

Recall, Z-score of weights is a random variable from standard Normal,

$$w_d \div \mathrm{SE}(w_d) \sim \mathcal{N}(0, 1)$$

Thus anything with Z-score > 2 is significant at 5% confidence level

Example: African Heart Disease

	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

Finding Systolic blood pressure (sbp) is **not a significant predictor**

Obesity is not significant and negatively correlated with heart disease in the model

Remember All correlations / significance of features are based on presence of *other features*. We must always consider that features are strongly correlated.

Example: African Heart Disease

	Coefficient	Std. Error	Z score
(Intercept)	−4.204	0.498	−8.45
tobacco	0.081	0.026	3.16
ldl	0.168	0.054	3.09
famhist	0.924	0.223	4.14
age	0.044	0.010	4.52

Doing some feature selection we find a model with 4 features: tobacco, ldl, family history, and age

How to interpret coefficients? (e.g. tobacco → 0.081)

- Tobacco is measured in total lifetime usage (in kg)
- Thus, increase of 1kg of lifetime tobacco yields

$$\exp(0.081) = 1.084$$

Or 8.4% increase in odds of coronary heart disease

- 95% CI is 3% to 14% since $\exp(0.081 \pm 2 \times 0.026) = (1.03, 1.14)$