Computer Science

CSC380: Principles of Data Science

Basic machine learning 1

Xinchen Yu

- Probability
- Statistics

- Data Visualization
- Predictive modeling

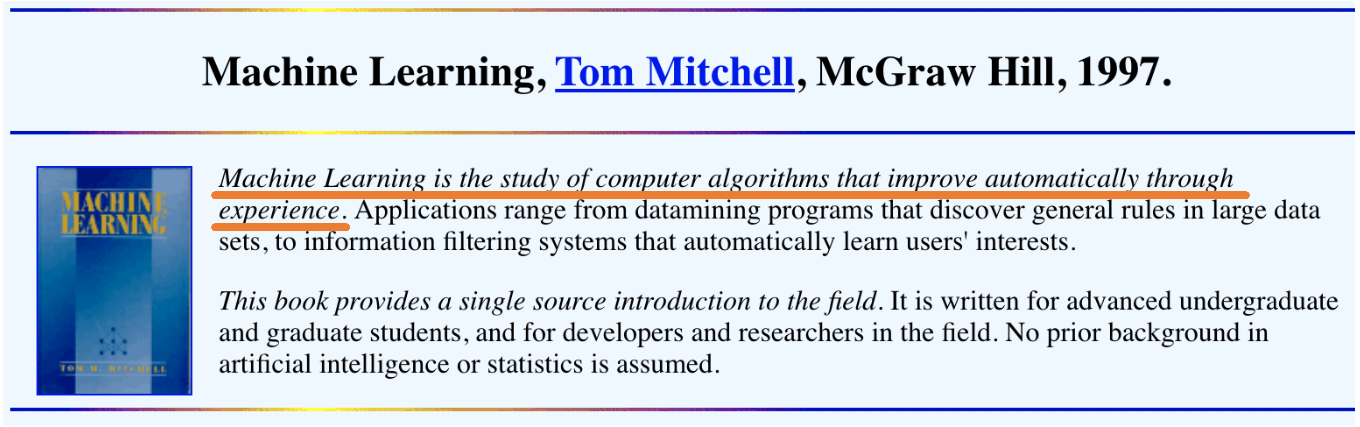
Outline

- Introduction to Machine Learning
- Supervised Learning: Linear Regression
- Overfitting and underfitting
- Regularization in regression
- Feature Selection

Introduction to Machine Learning

What is machine learning (ML)?

- **Tom Mitchell** established Machine Learning Department at CMU (2006).



Machine Learning, Tom Mitchell, McGraw Hill, 1997.

Machine Learning is the study of computer algorithms that improve automatically through experience. Applications range from datamining programs that discover general rules in large data sets, to information filtering systems that automatically learn users' interests.

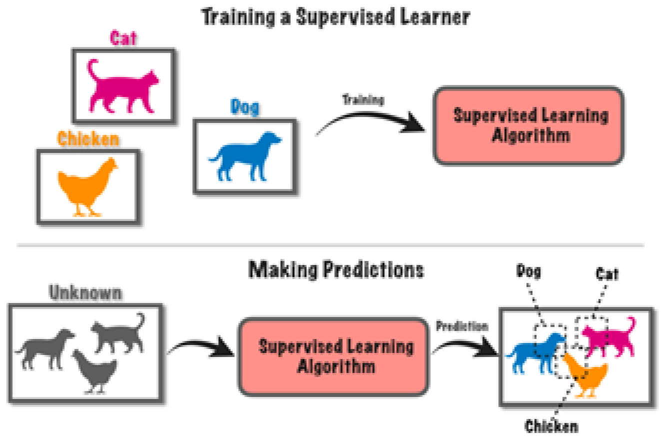
This book provides a single source introduction to the field. It is written for advanced undergraduate and graduate students, and for developers and researchers in the field. No prior background in artificial intelligence or statistics is assumed.

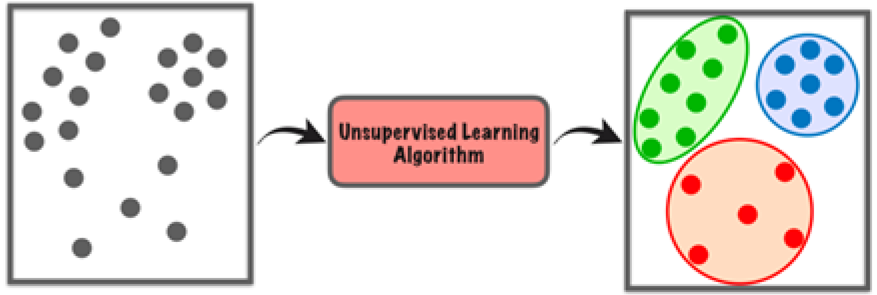
- In short: algorithms adapt to data
- A subfield of **Artificial Intelligence (AI)** – computers perform “intelligent” tasks.
- Classical AI vs ML: rule-driven approaches vs. data-driven approaches

Supervised vs Unsupervised Learning

- **Supervised Learning** - Training data consist of inputs and outputs
  - Classification, regression, translation, …

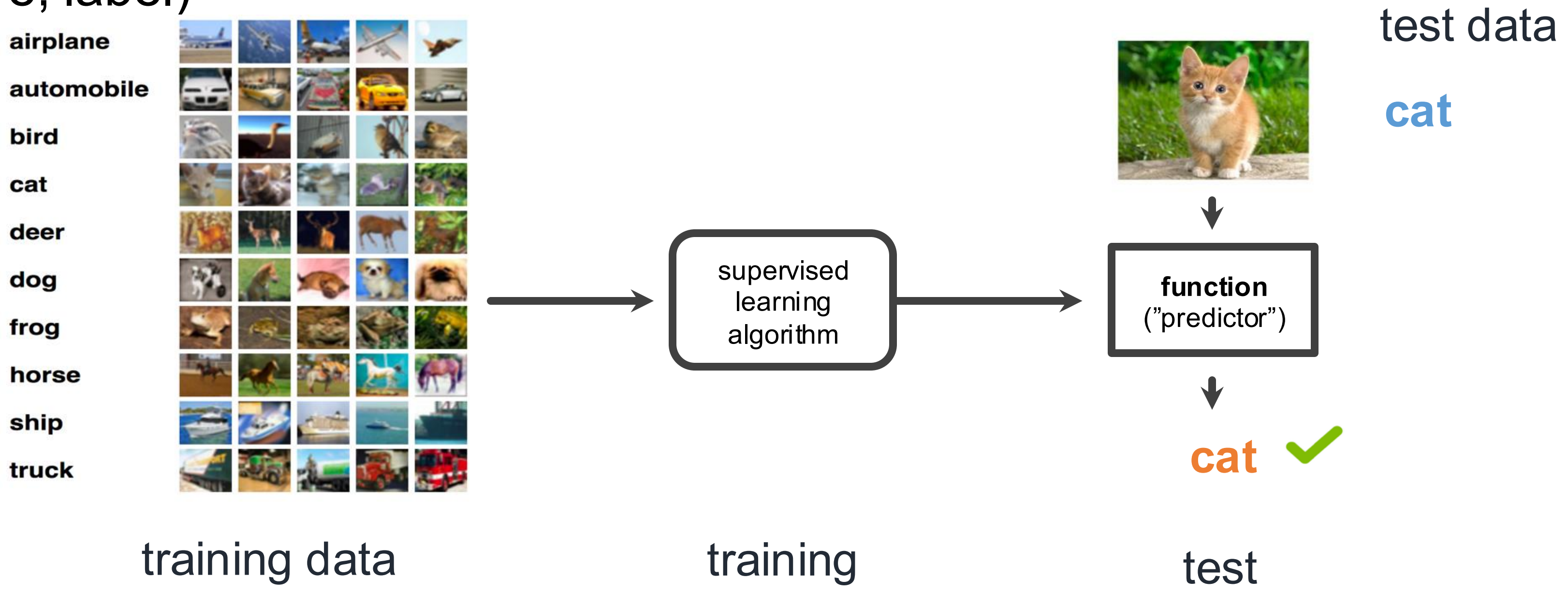
- **Unsupervised Learning** – Training data only contain inputs
  - Clustering, dimensionality reduction, segmentation, …


Training a Supervised Learner
Cat
Dog
Chicken
Training
Supervised Learning Algorithm
Making Predictions
Unknown
Supervised Learning Algorithm
Prediction
Dog
Cat
Chicken



Supervised learning

- Training / test data: datasets comprised of *labeled examples*: pairs of (feature, label)

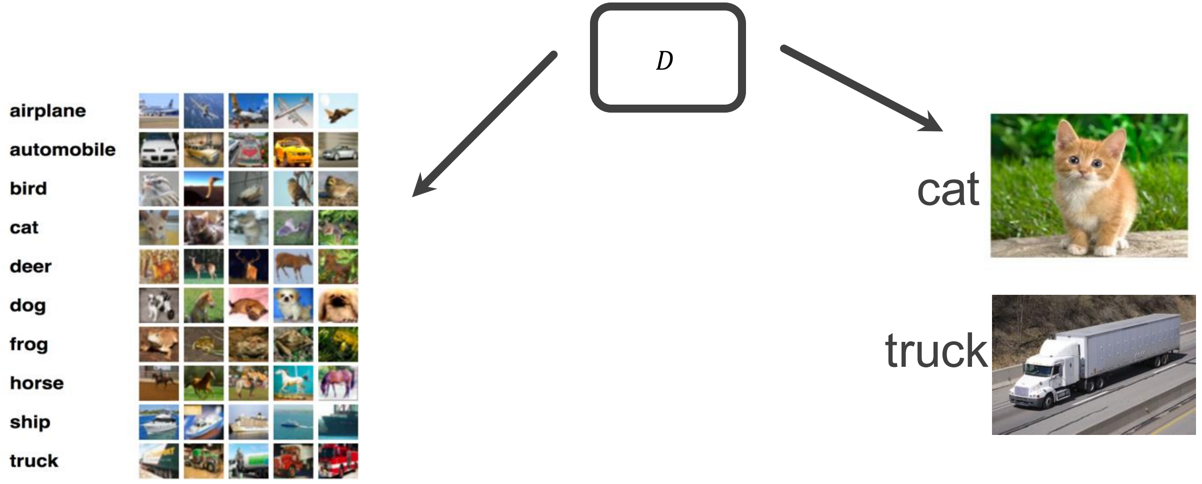


How should test data be chosen?

- Test data cannot be identical to training data.
- Test data cannot be just ONE data point.

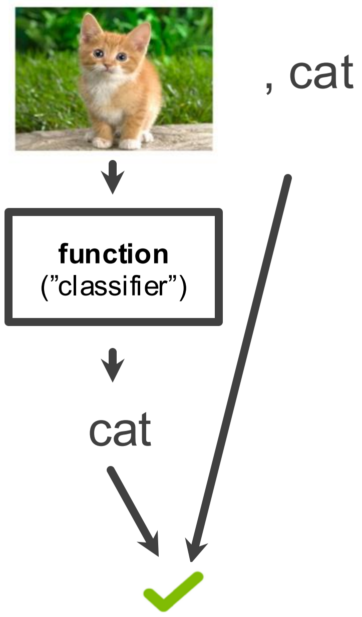
Supervised learning setup

- Key assumption: training and test data are drawn from the same *population*, or *data generating distribution* D
  - They are assumed to be IID samples: independent and identically distributed
- Training and test data are independent



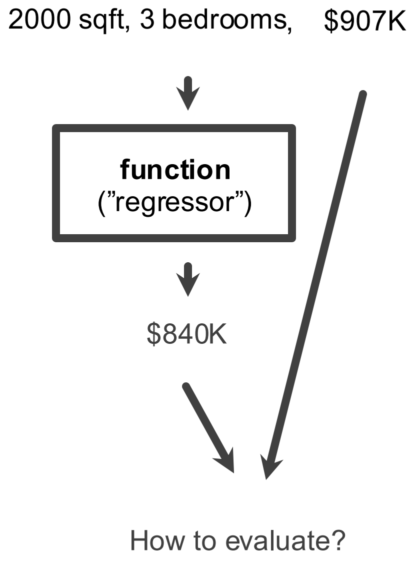
Supervised learning setup

- Scenario 1: classification



- Scenario 2: regression

(e.g. house price prediction)



- Loss function ℓ: measures the quality of prediction $\hat{y}$ respect to true label y

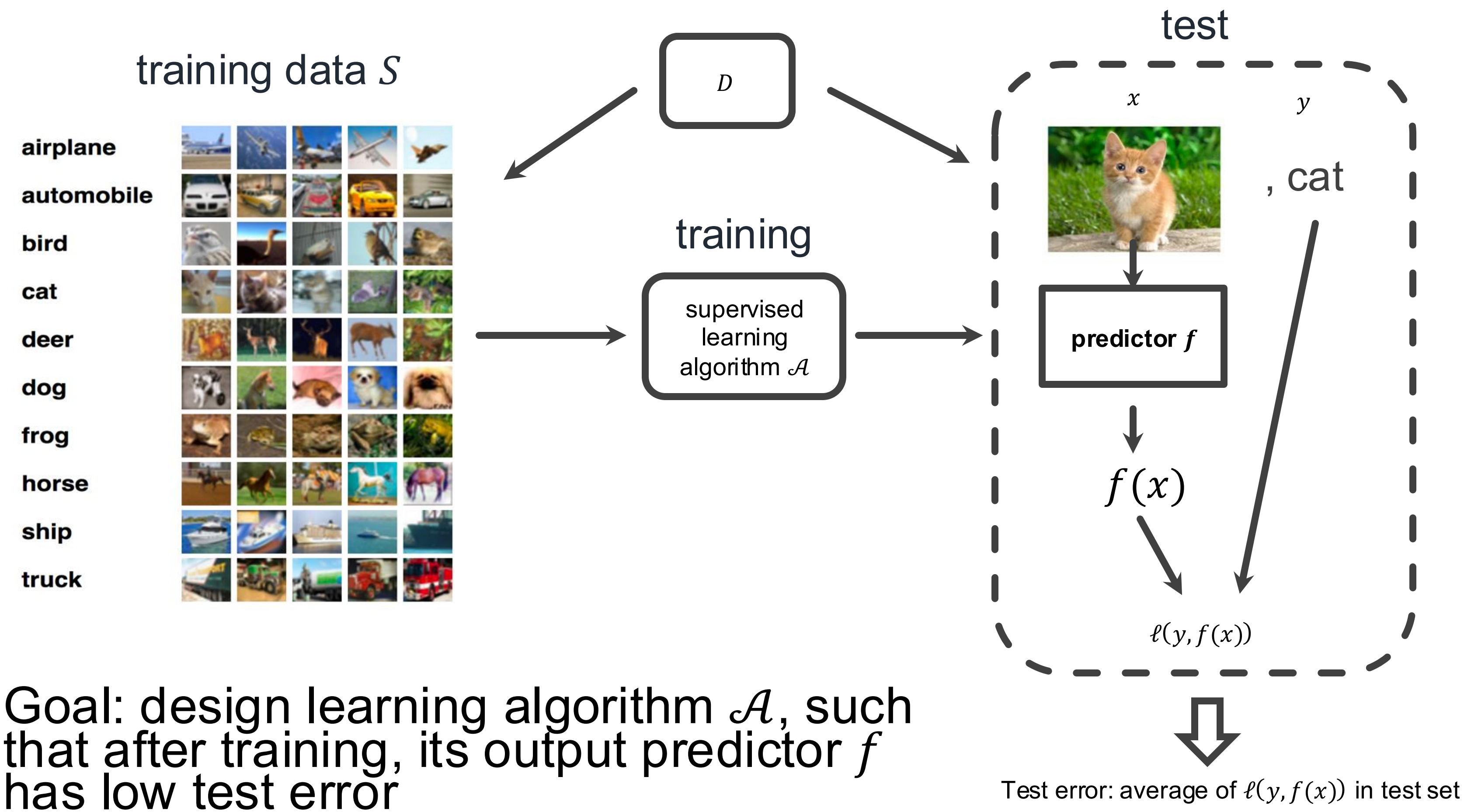
- Examples:
  - Classification error

    $\ell(y, \hat{y}) = 1$ if $y \neq \hat{y}$, and zero otherwise

  - Square loss

    $\ell(y, \hat{y}) = (y - \hat{y})^2$ - regression

Supervised learning setup in one figure

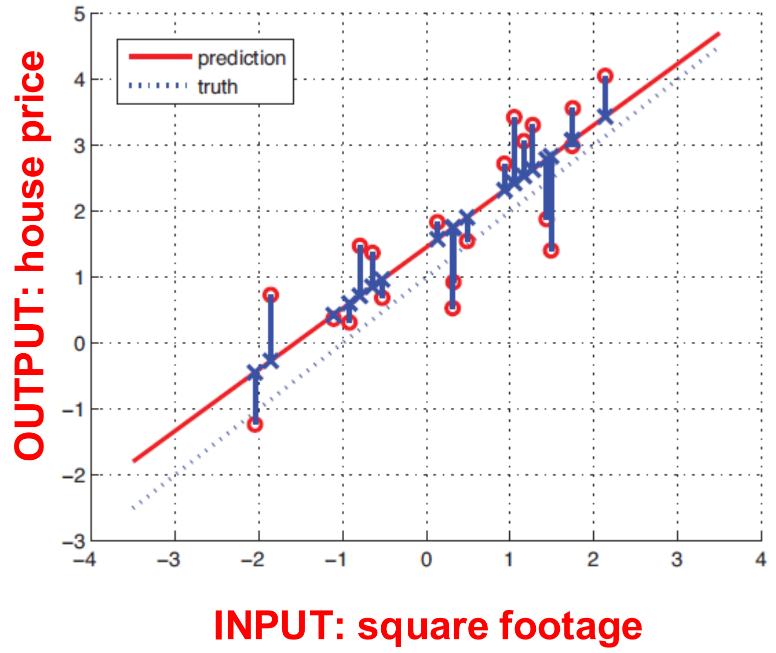


Test error: average of $\ell(y, f(x))$ in test set

- Goal: design learning algorithm $\mathcal{A}$, such that after training, its output predictor f has low test error

Supervised Learning: Linear Regression

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

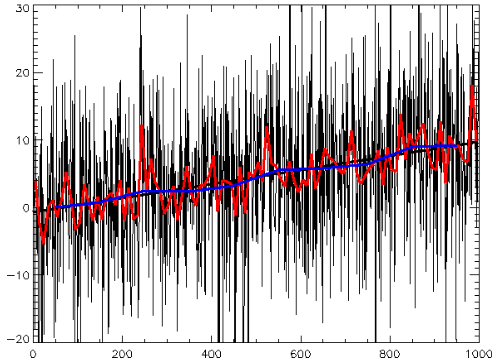
Outputs y are real-valued

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

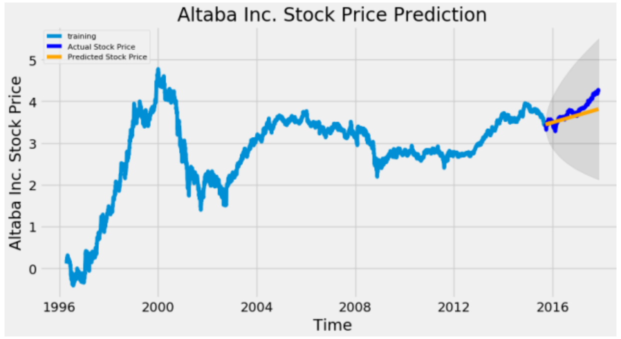
$$y = w^T x + b$$

Linear Regression

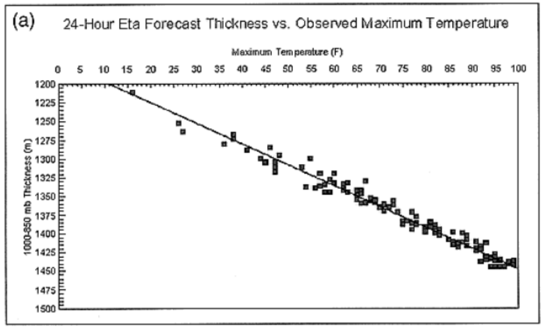
Where is linear regression useful?



Trendlines



Stock Prediction

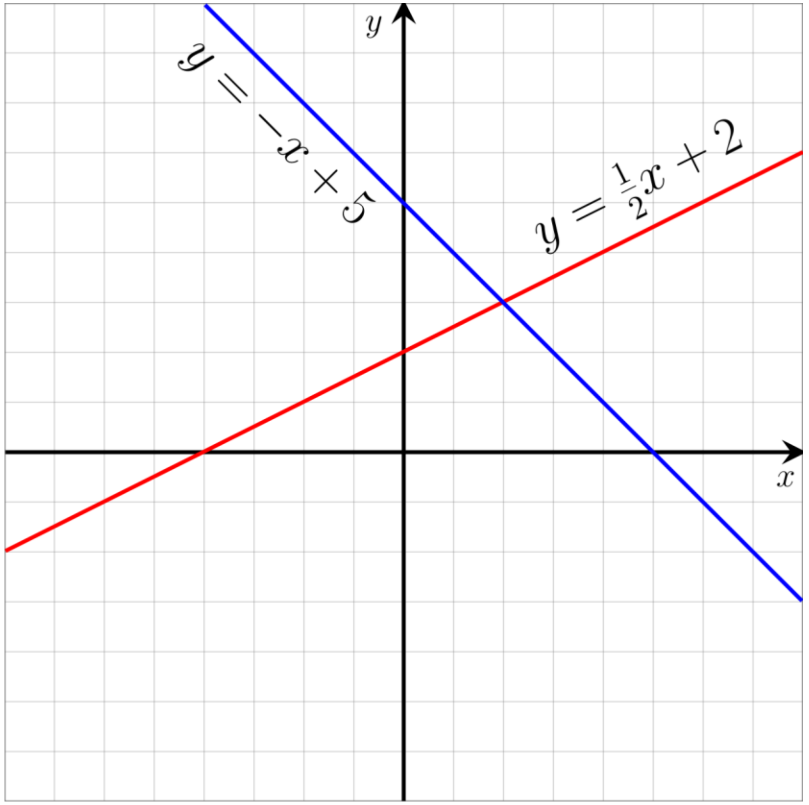


Climate Models

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*,

$$y = \underbrace{w}_{\text{Slope}} \cdot x + \underbrace{b}_{\text{Intercept}}$$

- 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

Math Interlude: inner product

Two vectors:

$$\vec{x} = \langle 2, -3 \rangle \qquad \mathbf{x} = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

$$\vec{y} = \langle 5, 1 \rangle \qquad \mathbf{y} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

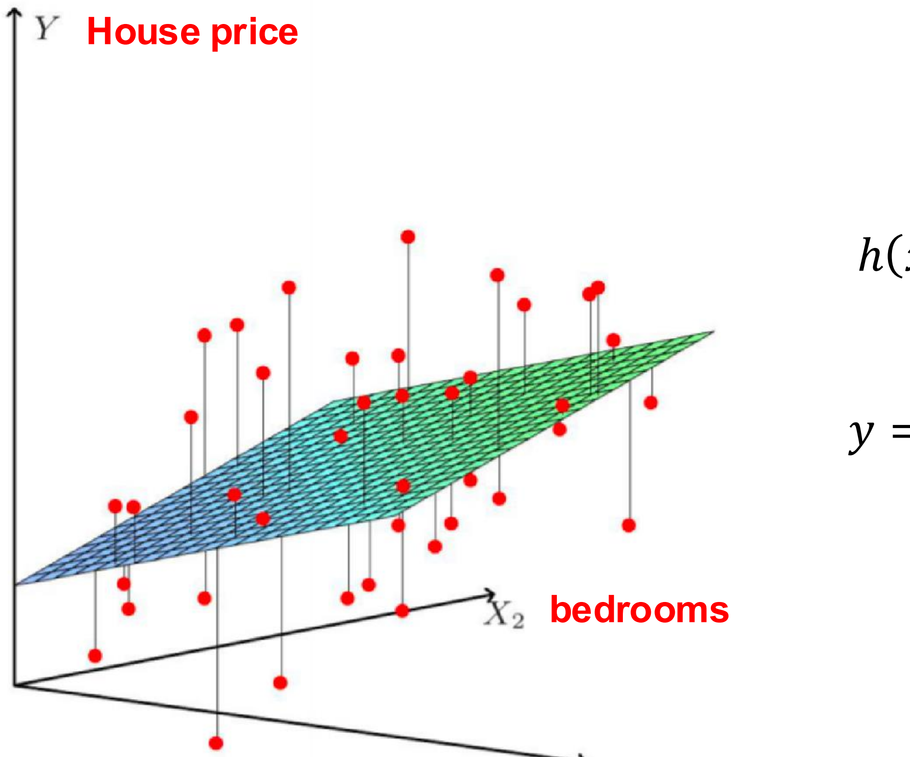
How to compute $\vec{x} \cdot \vec{y}$?

Multiply corresponding entries and add:

$$\vec{x} \cdot \vec{y} = \langle 2, -3 \rangle \cdot \langle 5, 1 \rangle = (2)(5) + (-3)(1) = 7$$

$$\mathbf{x}^T \mathbf{y} = \begin{bmatrix} 2 & -3 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \end{bmatrix} = [7] \quad \text{(or just 7)} \quad (\text{so } \vec{x} \cdot \vec{y} \text{ becomes } \mathbf{x}^T \mathbf{y})$$

Linear regression in dimension more than 2



$$h(x) = w_1 \cdot x_1 + w_2 \cdot x_2 + b = w \cdot x + b$$

$y = w \cdot x + b$ is a hyperplane

Linear Regression

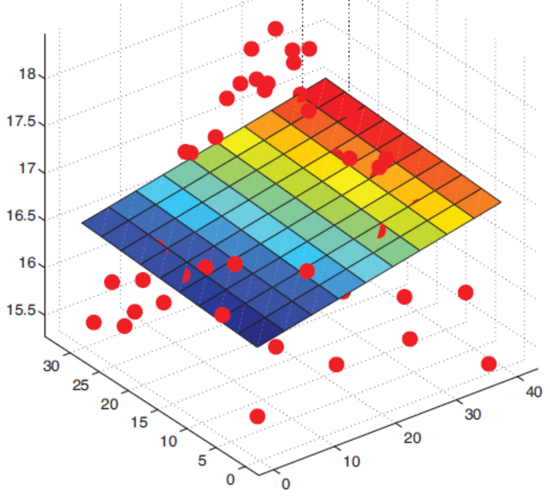
For D-dimensional input vector $x \in \mathbb{R}^D$ the plane equation,

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

Sometimes 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)]

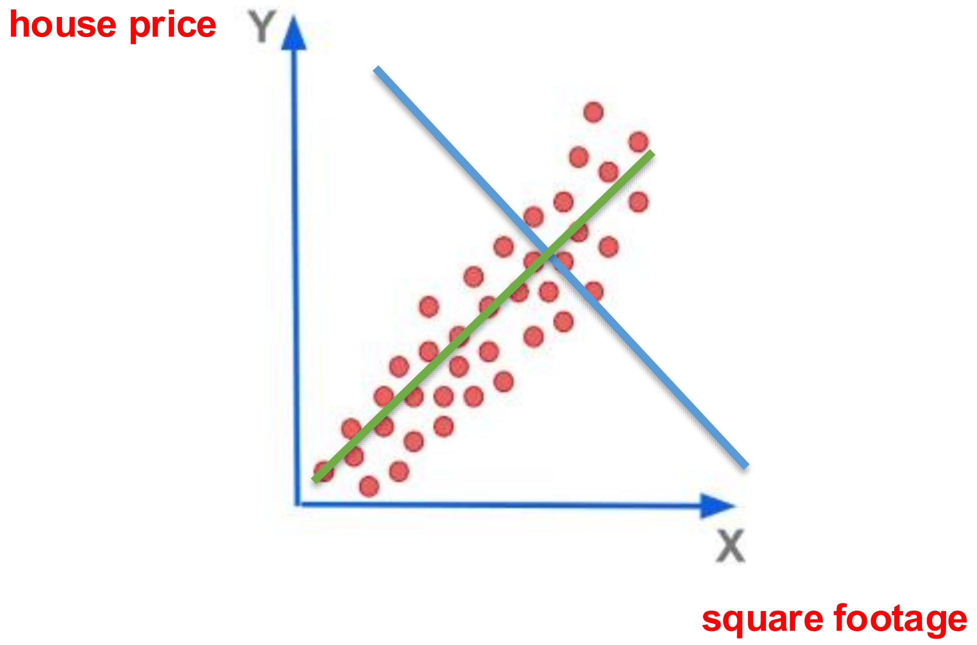


Since:

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

Learning Linear Regression Models

- Which line is a better predictor, blue or green?



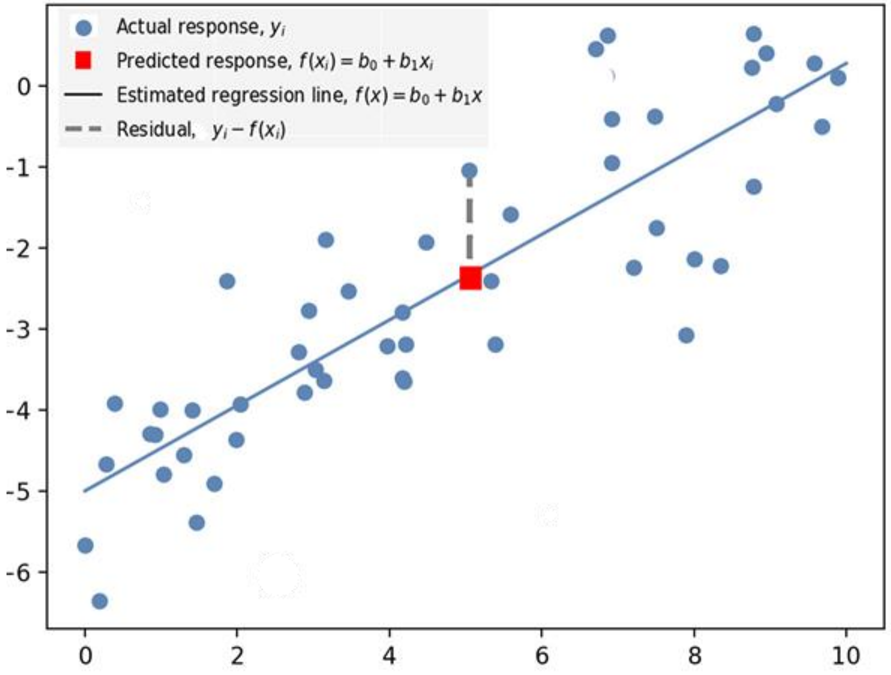
- green

Learning Linear Regression Models

There are at least two ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *square* loss

Fitting Linear Regression



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

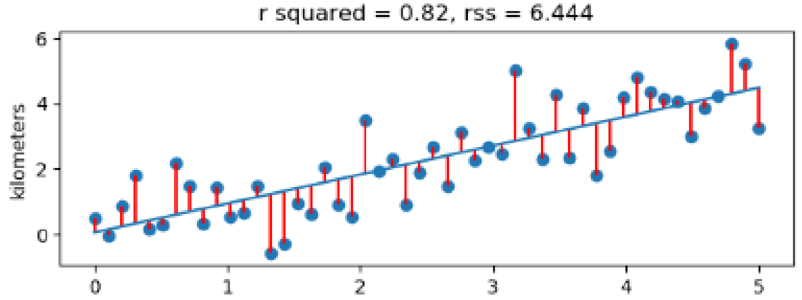
The distance from each point to the line is the **residual**

$$\underset{\textbf{Label}}{y} - \underset{\textbf{Prediction}}{w^T x}$$

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

Fitting linear regression

- Each point i induces a separate residual value $y_i - w \cdot x_i$



- We'd like to find w such that all $y_i - w \cdot x_i$ are small
- We can convert this to an optimization problem: find w that minimizes

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

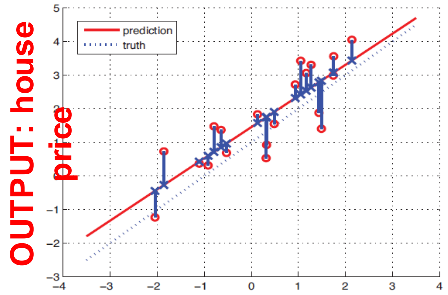
- This is called the ***least squares solution***

Math Interlude: optimization problems

$$\operatorname{argmin}_{w \in \mathrm{R}^d} \sum_{i=1}^{n} (y_i - w \cdot x_i)^2$$

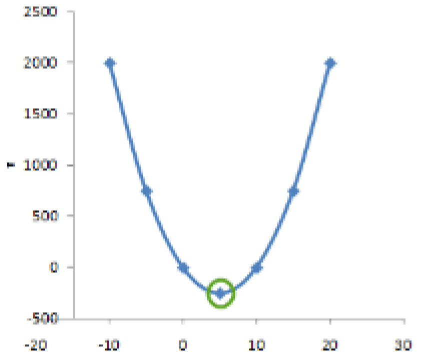
w: Optimization variable

Objective function



INPUT: square footage

- **Example** Find $\operatorname{argmin}_x ax^2 + bx + c \ (a > 0)$
  - $x = -\frac{b}{2a}$



Math Interlude: optimization problems

Example Suppose we have 2 data points (x=1, y=0) and (x=-1, y=1), we fit $y = w\,x$ without intercept, find the least squares solution $\hat{w}$

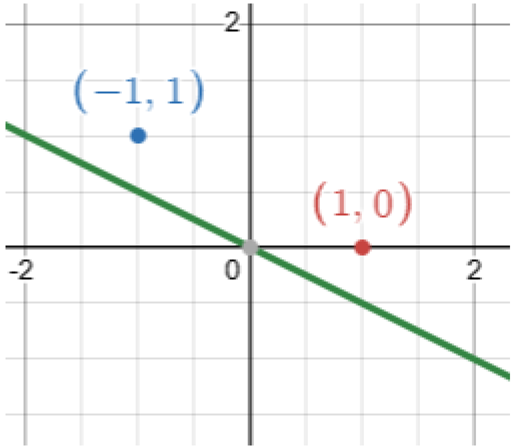
Solution the objective function of least squares is

$$(y_1 - w\,x_1)^2 + (y_2 - w\,x_2)^2$$

which is

$$w^2 + (1+w)^2 = 2w^2 + 2w + 1$$

the minimizer is $\hat{w} = -\frac{b}{2a} = -\frac{1}{2}$



Why cannot the line fit perfectly?

- Here, we only consider $y = w\,x$ without intercept

In-class exercise: training and test loss

- We have the following training data

Study hours (x)	Exam score (y)
1	2
3	6

- Q1: We fit a linear regression model $y = w \cdot x$ that minimizes mean square error. What is this model $\hat{w}$?

- Q2: What is the average loss of model $\hat{w}$ on training and test data?

Study hours (x)	Exam score (y)
4	7
5	10

In-class exercise: training and test loss

Solution

$\hat{w} = \text{argmin}_w \, (1w - 2)^2 + (3w - 6)^2$

Minimizer: $\hat{w} = -\frac{b}{2a} = 2$

$10w^2 - 40w + 40$

Study hours (x)	Exam score (y)
1	2
3	6

Training loss of $\hat{w}$:

$\frac{1}{2}((2-2)^2 + (6-6)^2) = 0$

size of training set

Study hours (x)	Exam score (y)	Predicted score
1	2	2
3	6	6

Test loss of $\hat{w}$:

$\frac{1}{2}((8-7)^2 + (10-10)^2) = 0.5$

size of test set

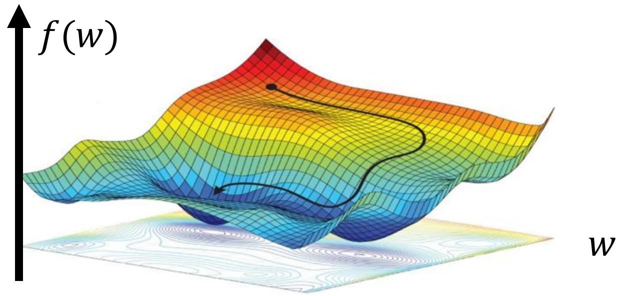
Study hours (x)	Exam score (y)	Predicted score
4	7	8
5	10	10

Usually, a trained model has smaller training loss than test loss

Math Interlude: optimization problems

- Unconstrained optimization problem: find

$$\operatorname{argmin}_{w \in \mathrm{R}^d} f(w)$$



- Solutions can oftentimes be found in one of two ways:
  1. Closed form solutions
  2. Open-source or commercial optimization libraries (e.g. `cvxpy`, `scipy.optimize.minimize`)

Linear Regression in Scikit-Learn

For Evaluation

Load your libraries,

```
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)
```

```
Coefficients: [998.57768914]

Intercept: 152.00335421448167

Mean squared error: 4061.83

Coefficient of determination (R^2): 0.23
```

Scale sensitive, a bit hard to interpret ->

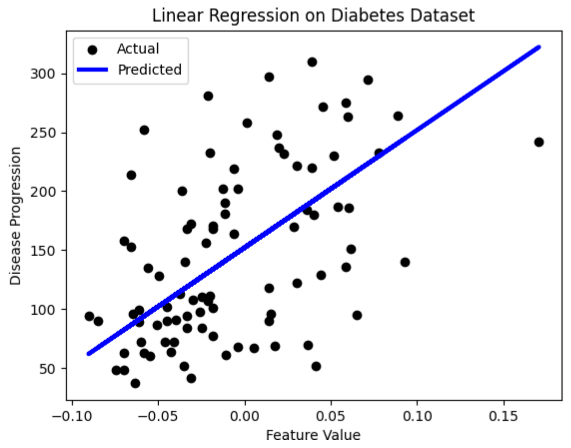
More interpretable->

Plot regression line with the test set,

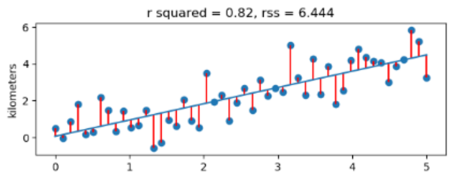
```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

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

plt.show()
```



Coefficient of Determination R^2



Variance unexplained by Regression model

Residual Sum-of-Squares

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

Total variance in dataset

Variance using avg. prediction

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

R^2 represents the proportion of the variance in y that is predictable from a x_i.

Coefficient of Determination R^2

$$R^2 = 1 - \frac{\text{RSS}}{\text{SS}}$$

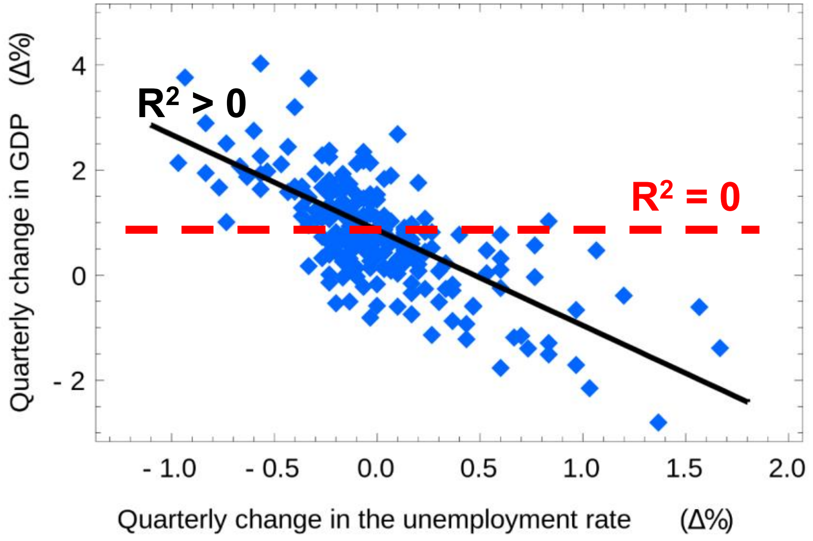
Variance unexplained by Regression model

Variance using avg. prediction

Maximum value $R^2=1.0$ means model explains *all variation* in the data

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

$R^2<0$ means model is worse than predicting average output (rare)



Overfitting and underfitting

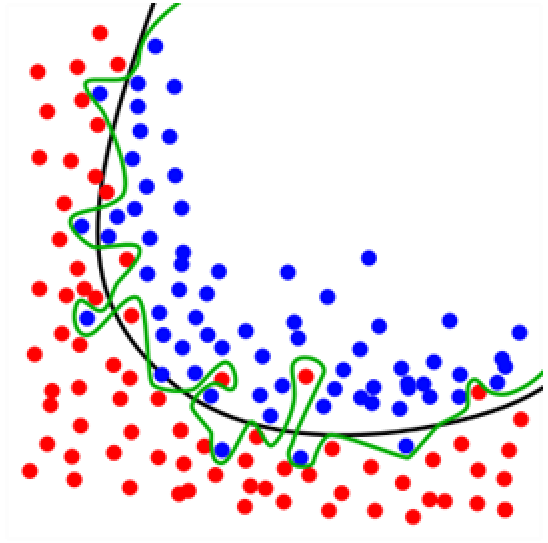
Challenge in machine learning: generalization

Why not learn the most complex predictor that can work flawlessly for the training data and be done with it? (i.e., predicts every training data point correctly)

Problem: may not generalize to unseen data – called *overfitting* the training data.

In other words, memorization is not generalization

Mitigation: Fit the training set but don't "over-do" it -- **regularization**.

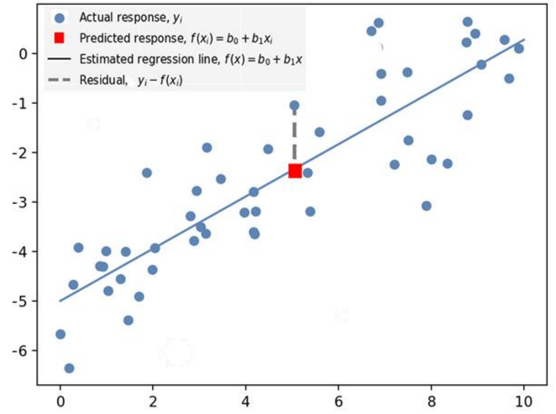


green: may be sensitive to noise in training data
black: more robust and can generalize better

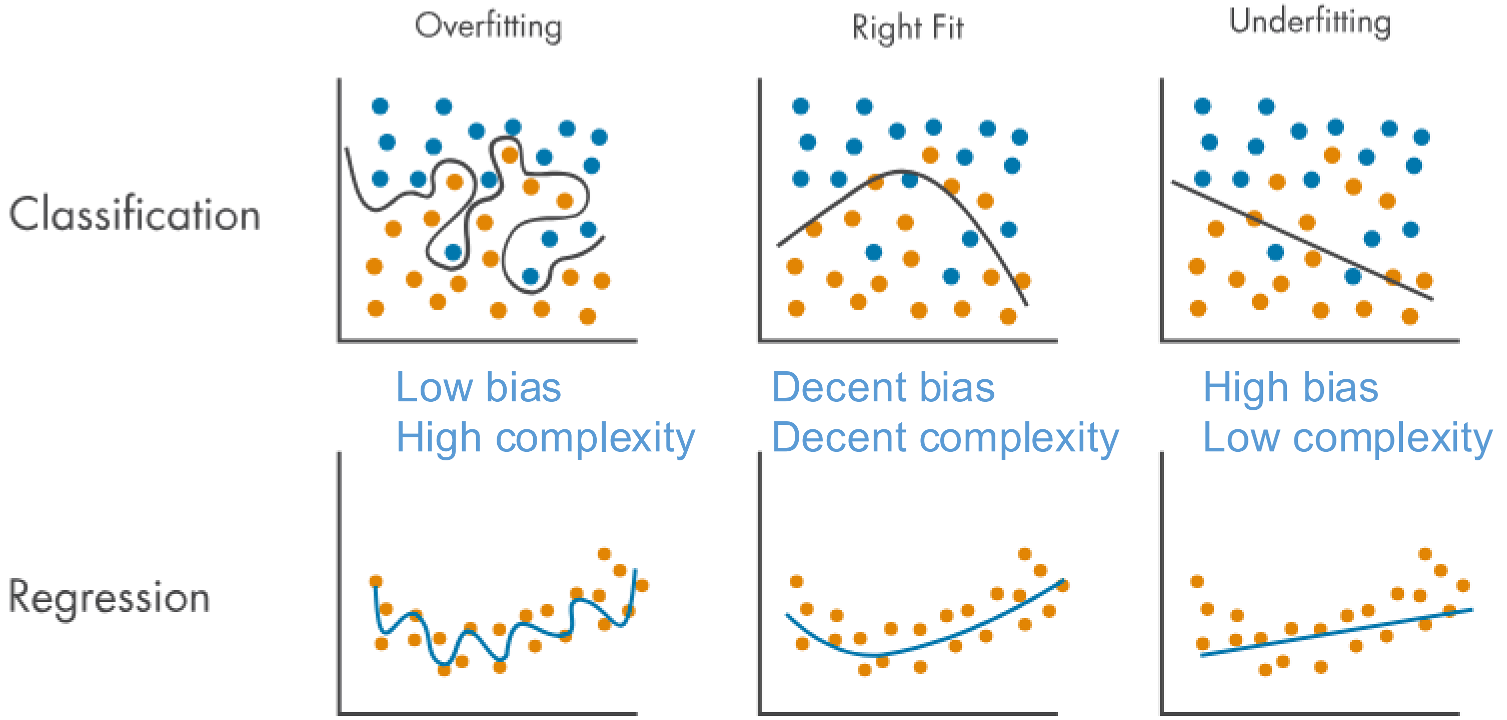
Recap

- Linear regression
  - Train the model on the training set
  - Training objective: find the w that minimize the mean squared error
  - $\text{argmin}_{w \in \mathrm{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - w \cdot x_i)^2$

- Evaluate the model on the testing set
  - Two metrics:
  - Mean squared error (smaller the better)
  - R squared (larger the better)



Overfitting and Underfitting



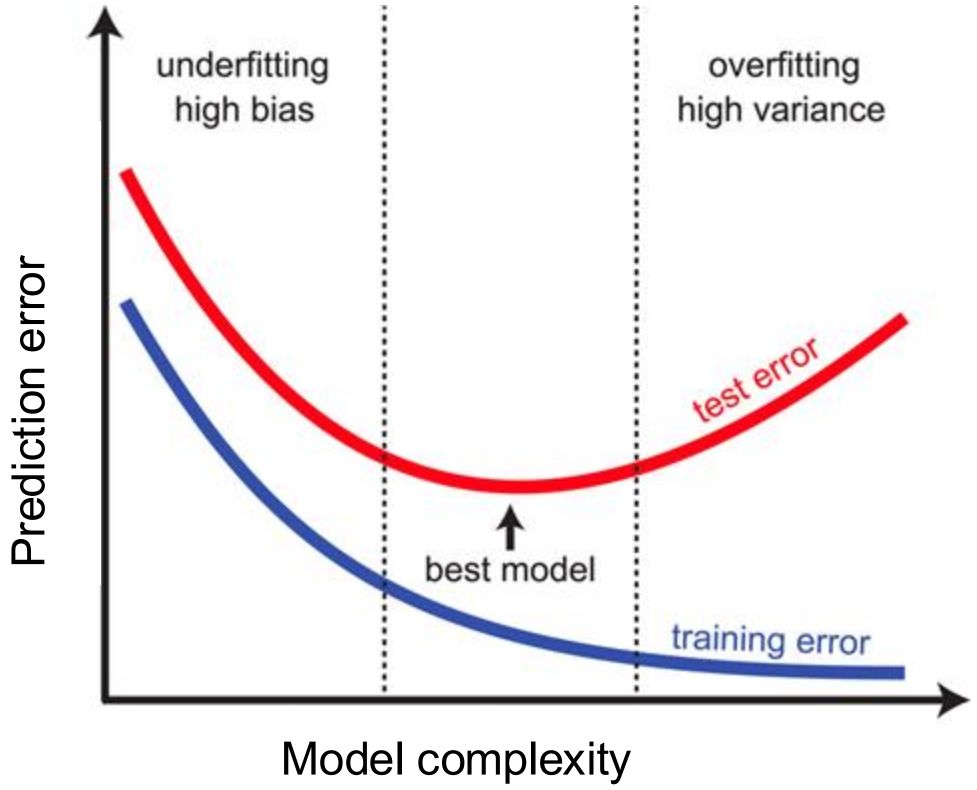
Ideal: select a model that trades off bias & complexity, i.e.,
- sophisticated enough to capture meaningful patterns for accurate predictions,
- yet not so intricate that it overfits the data. **“just right” complexity with low bias**

Relationship of Complexity and Model selection

The model is more complex when:

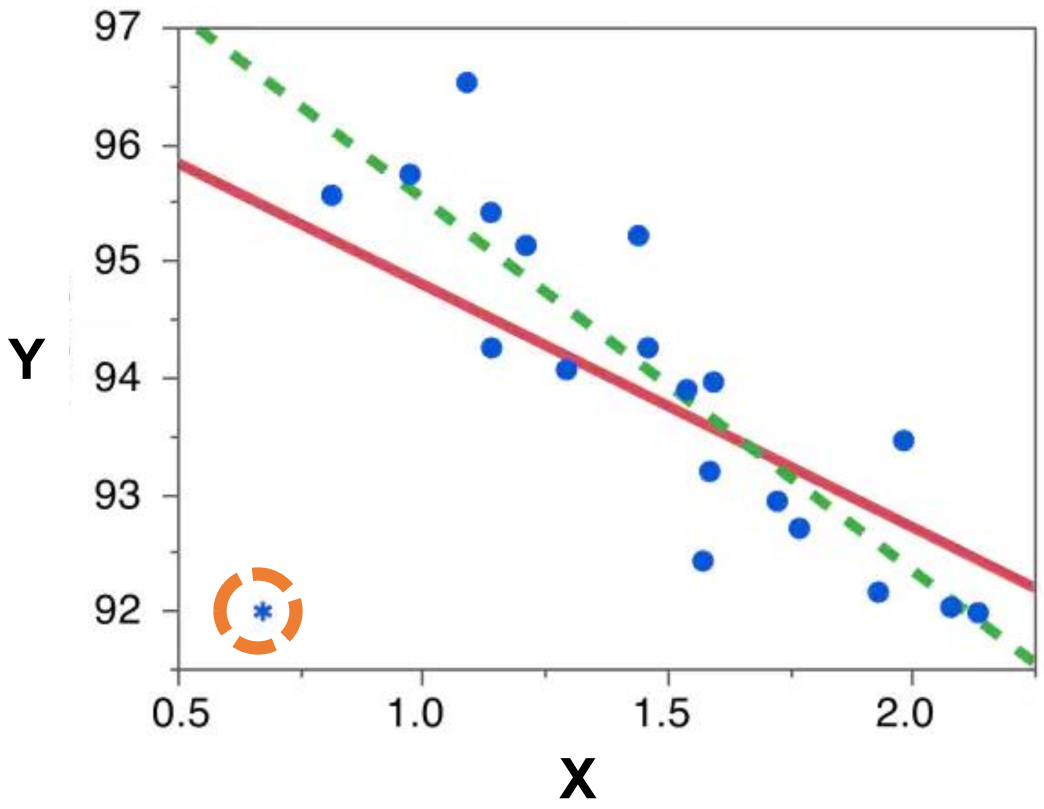
- There are more features used for prediction, and
- The weight of the predictors used for prediction is higher

Model selection: choosing model with “just right” complexity for data



Model Selection: Regularization

Outliers in Linear Regression



Outlier “pulls” regression line (red) away from inlier data, which results in overfitting

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

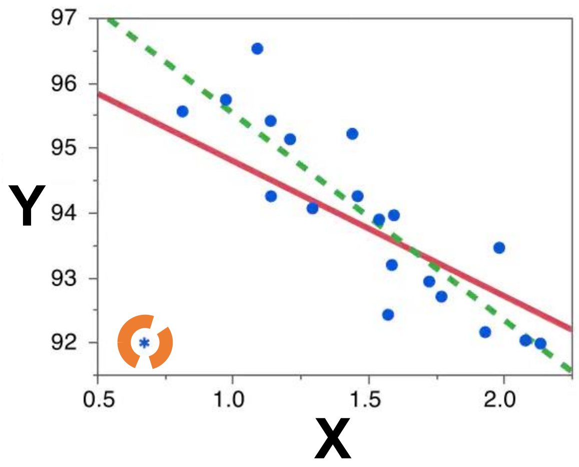
Regularization

Regularization helps avoid overfitting to training data by penalizing extreme weights

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

Regularization Strength

Regularization Penalty



Red model is without regularization

Green model is with regularization

Regularized Least Squares

A couple common regularizers:

L2 Regularized Linear Regression

- Ridge Regression

L1 Regularized Linear Regression

- LASSO -- “Least Absolute Shrinkage and Selection Operator”

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

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

$$||w||^2 = \sum_{j=1}^{d} w_j^2$$

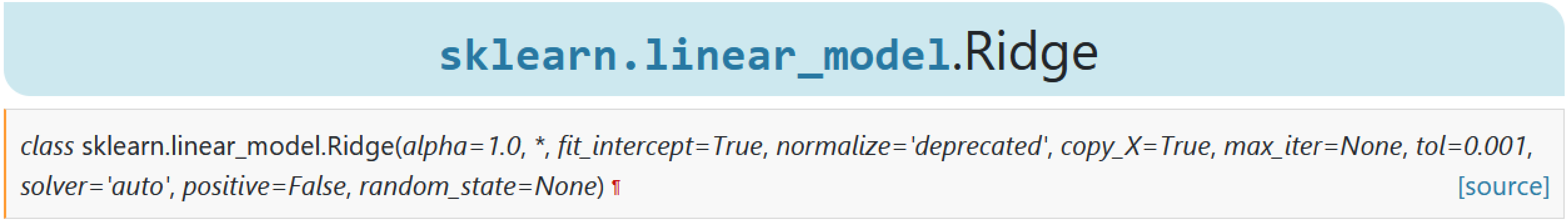
L1-regularized Least-Squares (LASSO)

Absolute Value (L1) Penalty

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

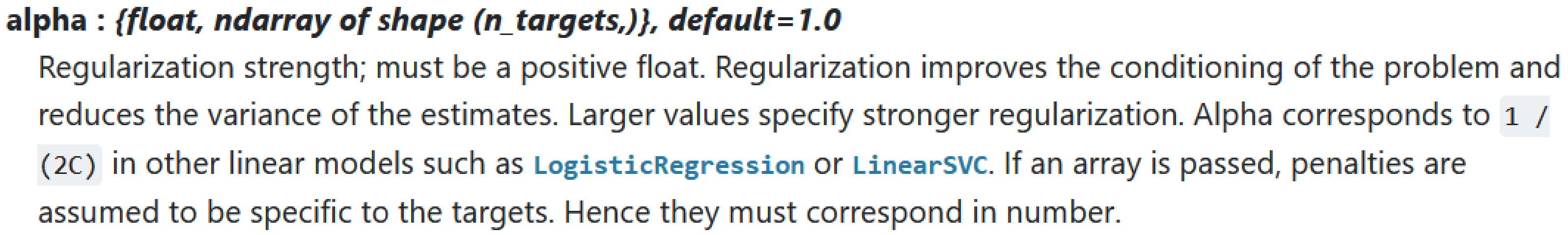
$$|w| = \sum_{j=1}^{d} |w_j|$$

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 **Regularization Strength**

Scikit-Learn: L2 Regularized Regression

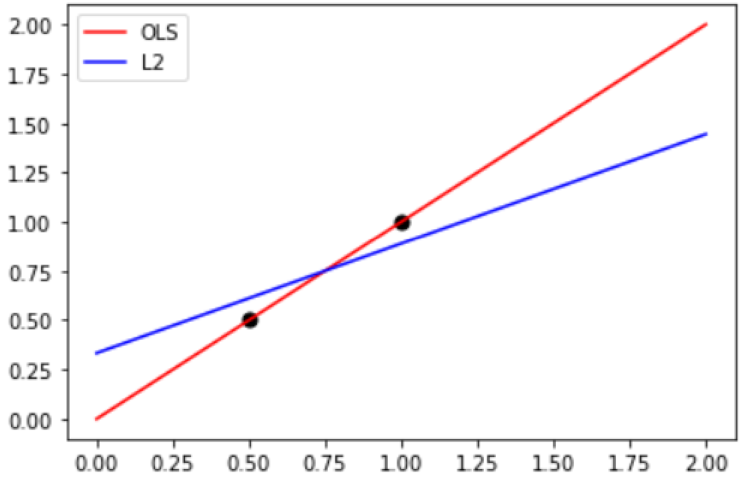
Define and fit OLS and L2 regression,

```
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()
```

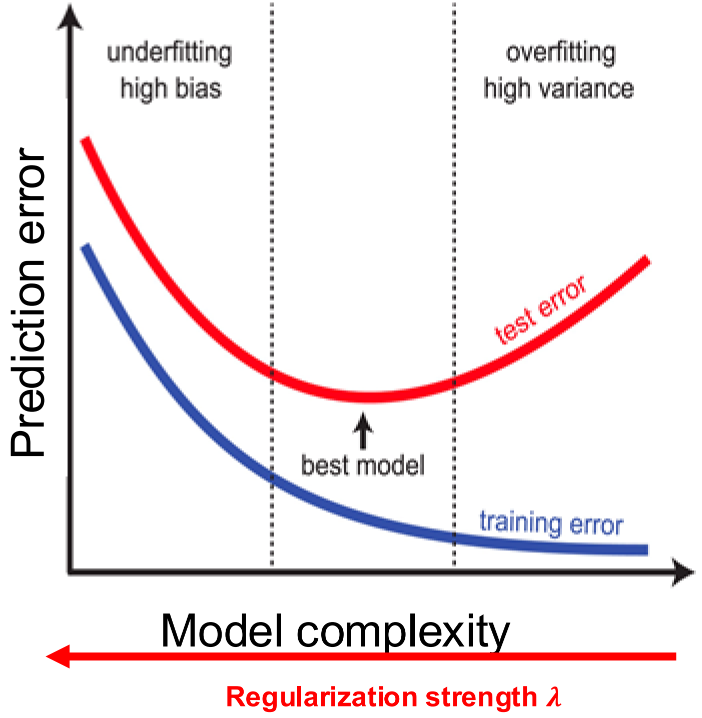


L2 (Ridge) reduces impact of any single data point

Choosing Regularization Strength

We need to tune regularization strength to get the best performance…

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



High λ => learned w has small weights

- *increases* bias & *decreases complexity*

Low λ => learned w has large weights

- *Increases test* bias & *increases complexity*

Model selection: How should we properly tune λ?

Naïve idea: using training loss to choose regularization

How to choose a good λ?

First, we need set of candidate λ's

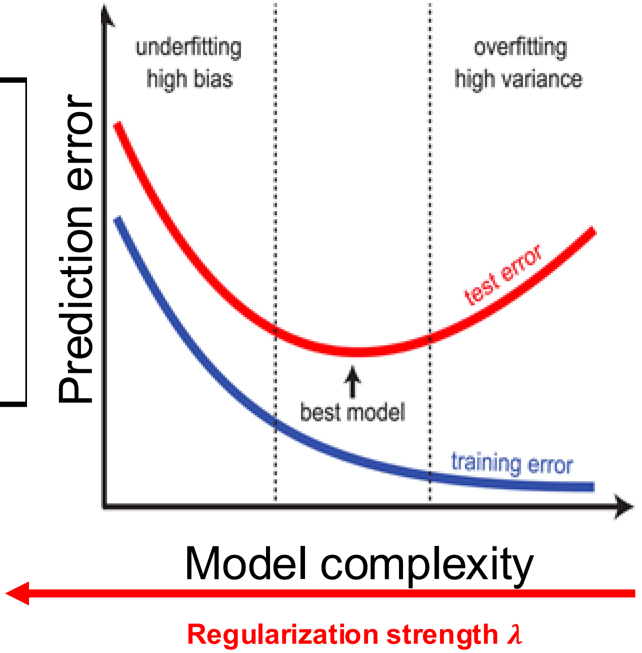
- e.g., geometric grid $\Lambda = \{0.1, 0.2, 0.4, .., 1000\}$

Is the following a good approach?

```
For each $\lambda \in \Lambda$:
    Train ridge estimator $w_\lambda$ with regularization $\lambda$

Return: $w_\lambda$ with the smallest training loss
```

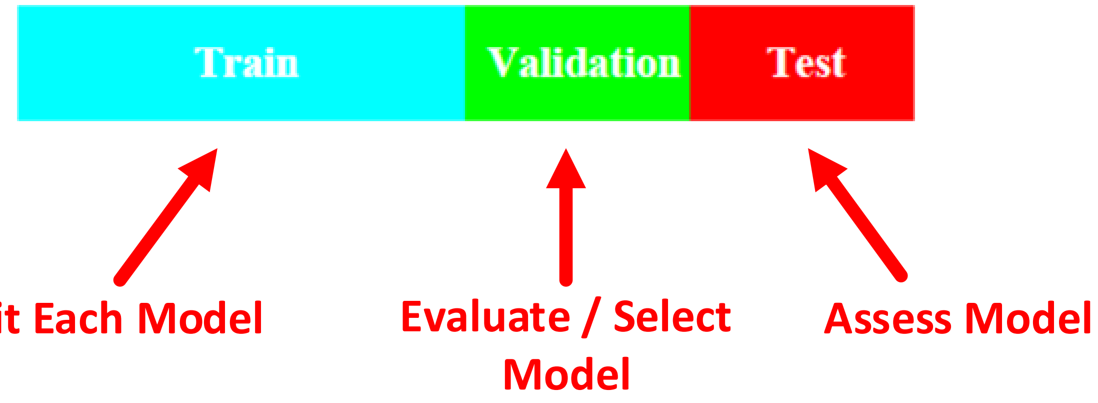
No – this likely always chooses the smallest λ which is prone to overfitting



Model Selection approach 1: hold-out validation set

How to choose a good $\lambda \in \Lambda$?

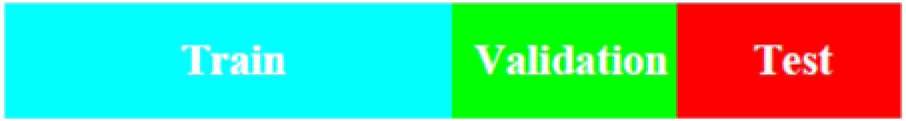
Partition data into Train-Validation-Test sets



- Ideally, Test set is kept in a "vault" and **only peek at it once final predictor is selected**
- Small dataset: 50% Training, 25% Validation, 25% Test (rule of thumb by statisticians)
- For large data (say a few thousands), 80-10-10 is usually fine.

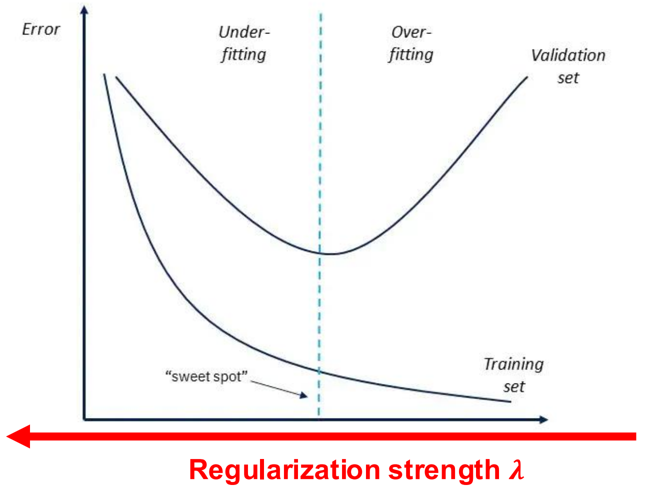
Model Selection approach 1: hold-out validation set

Key idea: use validation performance as a proxy of test performance



```
For each $\lambda \in \Lambda$:
    • Train ridge estimator $w_\lambda$ with training set with regularization $\lambda$
    • measure performance $e_\lambda$ of $w_\lambda$ on validation set

Return $w_{\hat{\lambda}}$, $\hat{\lambda}$: the $\lambda$ with the best $e_\lambda$ value
```



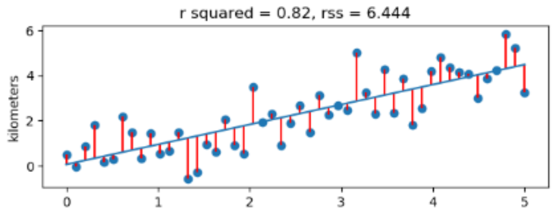
Model Selection for Linear Regression

A couple of common metrics for model selection…

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

Lower the better

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



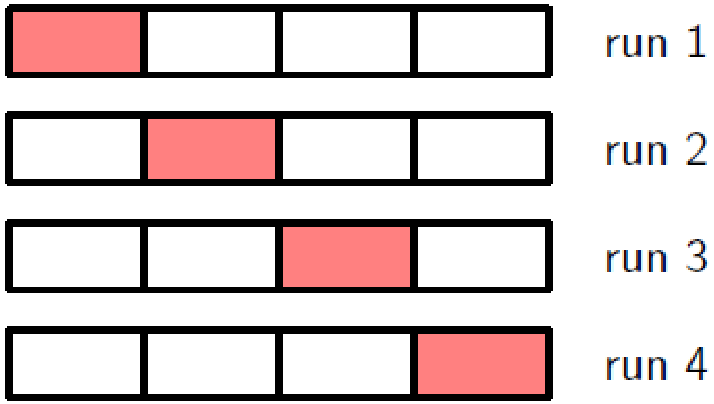
Coefficient of Determination Also called R-squared or R^2.

Higher the better

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

Model Selection approach 2: cross-validation

Main idea: improve data efficiency by splitting the training / validation data in multiple ways



K-fold Cross Validation: Partition training data into K "chunks" and for each run select one chunk to be validation data

For each run, fit to training data (K-1 chunks) and measure performance on validation set. Average performance across all runs.

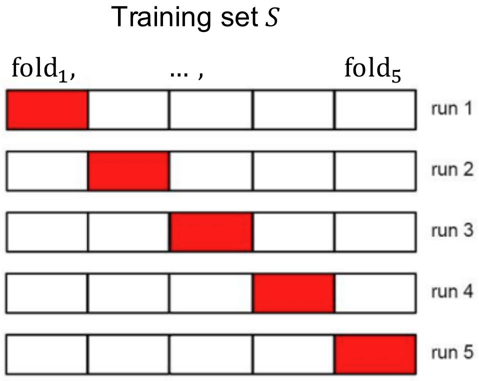
$K = 5, 10$ are typical good choices

Source: Bishop, C. PRML

Cross-validation: formal description

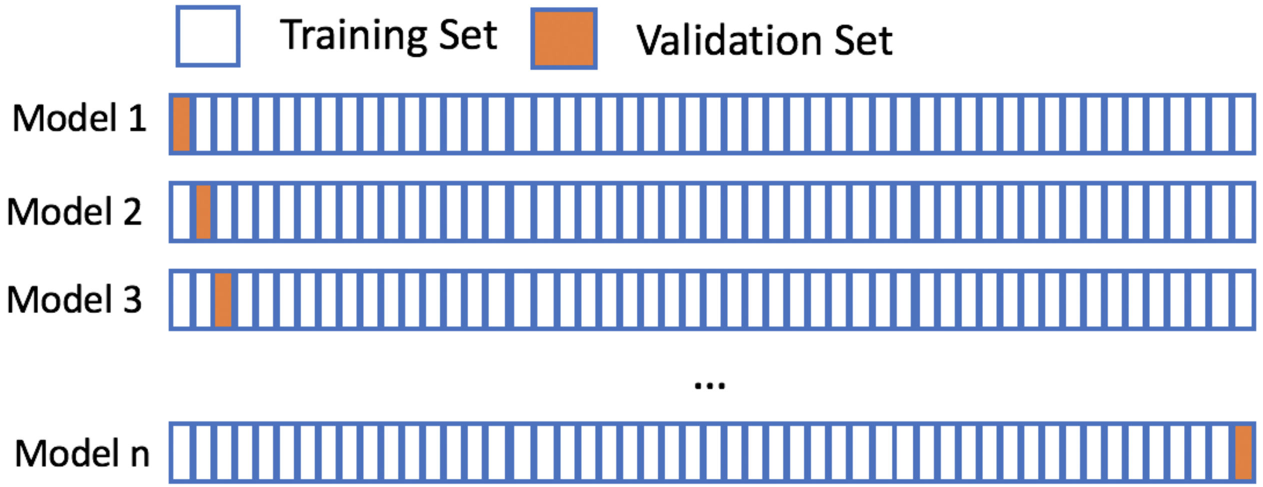
```
For each $\lambda \in \Lambda$:
    For $k \in \{1, \dots, K\}$:
        • Train ridge estimator $f$ with $S \setminus \mathrm{fold}_k$
        • measure performance $e_{\lambda,k}$ of $f$ on $\mathrm{fold}_k$
    Compute average performance: $E_\lambda = \frac{1}{K}\sum_{k=1}^{K} e_{\lambda,k}$

Choose $\hat{\lambda}$ := best $\lambda$ according to $E_\lambda$
Train $\hat{f}$ using $S$ with hyperparameter $\hat{\lambda}$
```



Leave one out cross-validation

What is the largest possible value of K?
$K = |S|$ -- this is called leave-one-out cross validation (LOOCV)



"Shrinkage" Feature Selection

Regularization 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

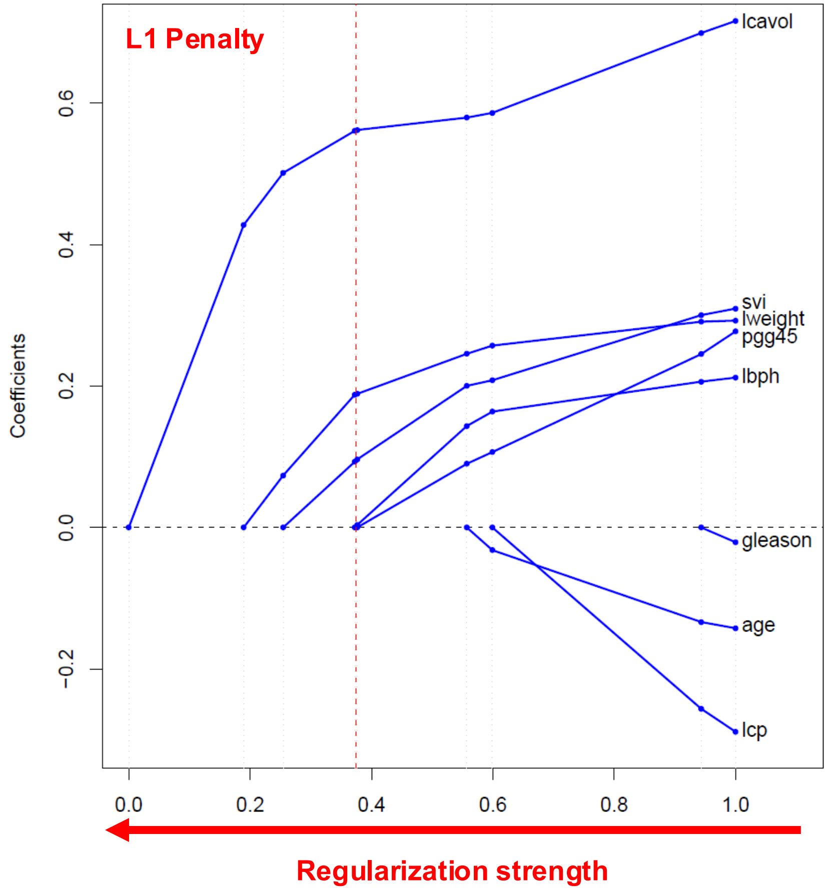
Example *Prostate Cancer Dataset* predicts prostate-specific cancer antigen with features: age, log-prostate weight (lweight), log-benign prostate hyperplasia (lbph), Gleason score (gleason), seminal vesical invasion (svi), etc.

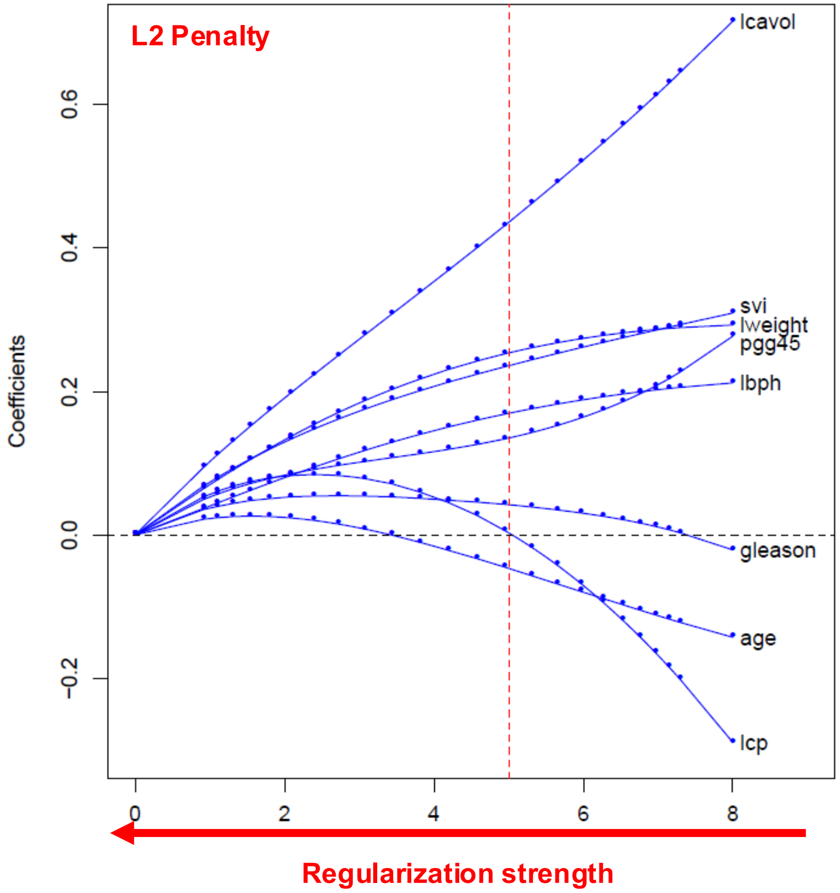
$\longrightarrow$ **L2 regularization learns nearly 0 weight for log capsular penetration (lcp)**

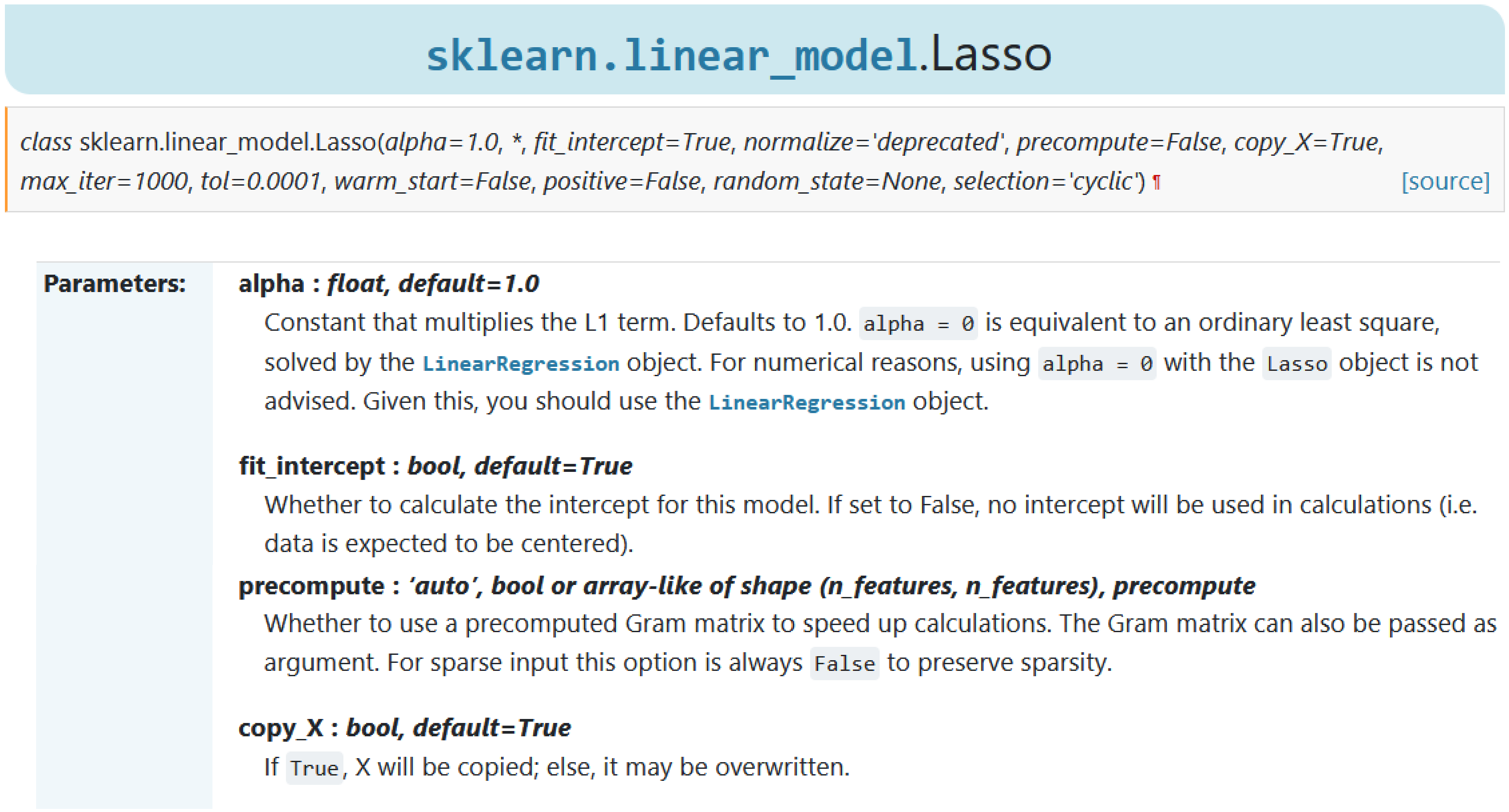
[Source: Hastie et al. (2001)]

Feature Weight Profiles

L1 penalty more likely learns coefficients that are zero, thus induces sparsity







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).

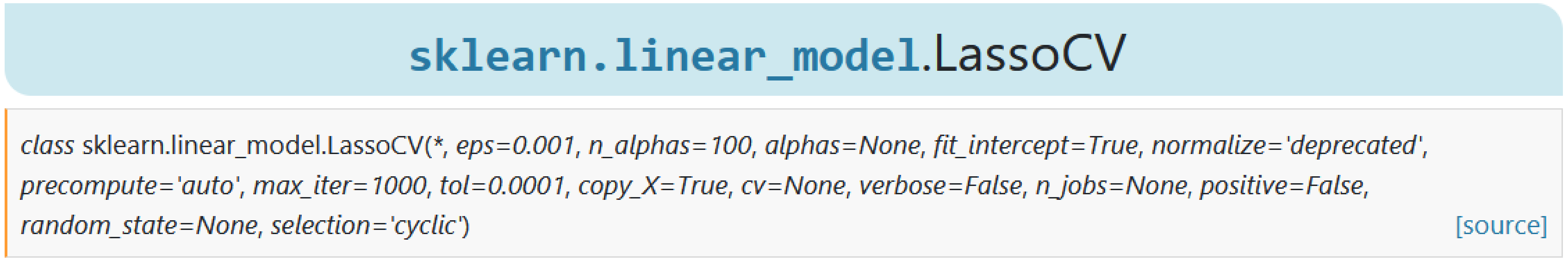
precompute : *'auto', bool or array-like of shape (n_features, n_features), precompute*

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.

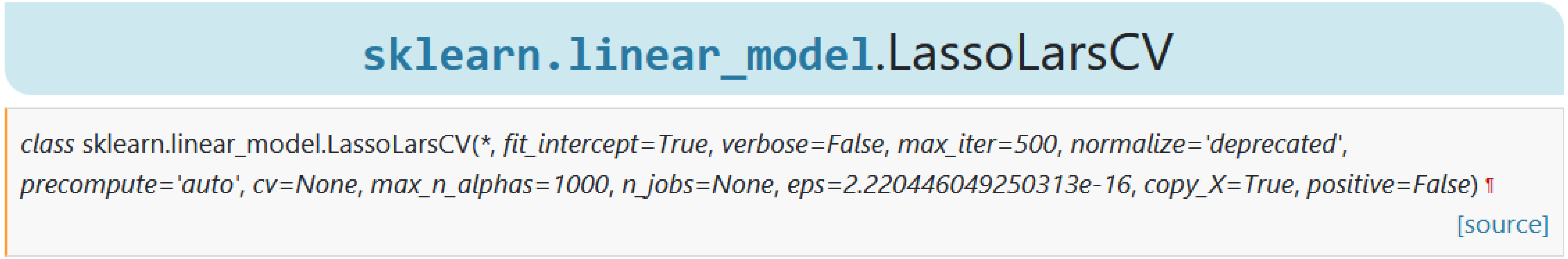
Specialized methods for cross-validation…



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

Their results are similar; LassoCV may be more stable

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)
```

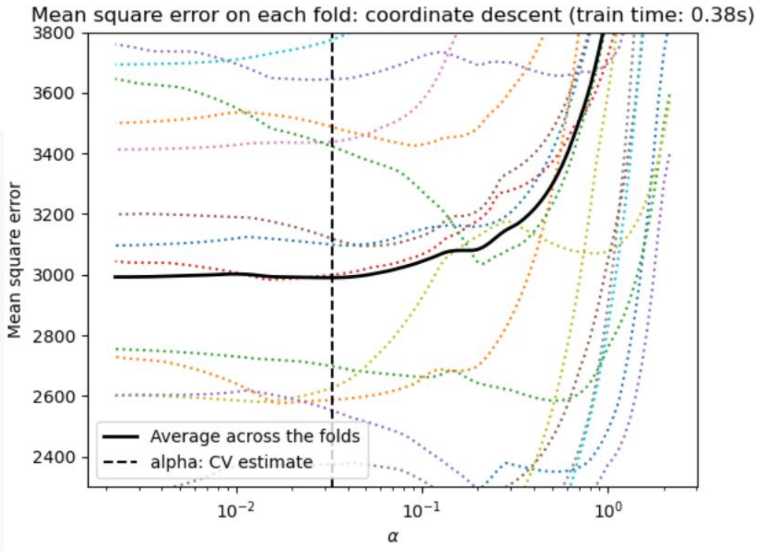
Plot solution path for range of alphas,

```
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"
)
```

20 validation error curves (dashed)

mean curve (solid)

alpha_ value chosen by cross validation



Feature Selection

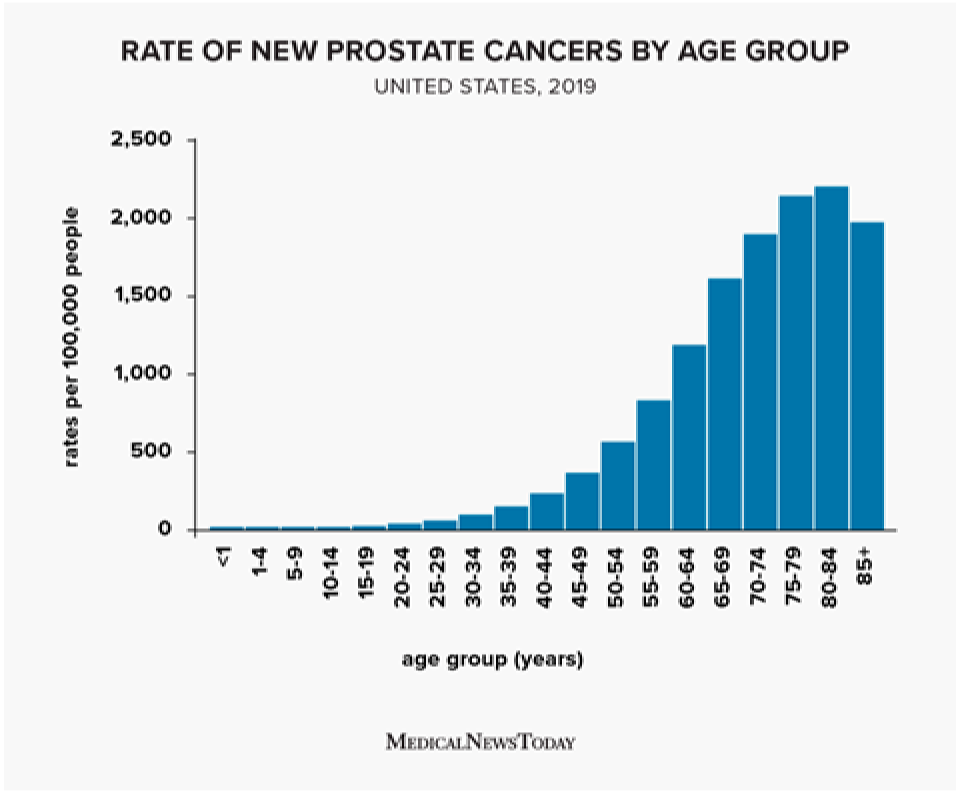
Feature selection

- Select only a few features to make predictions

- Benefits of using only a few features:
    - Model selection – trades off between bias and complexity

    - Interpretability – makes the model trustworthy by e.g. doctors and policy makers

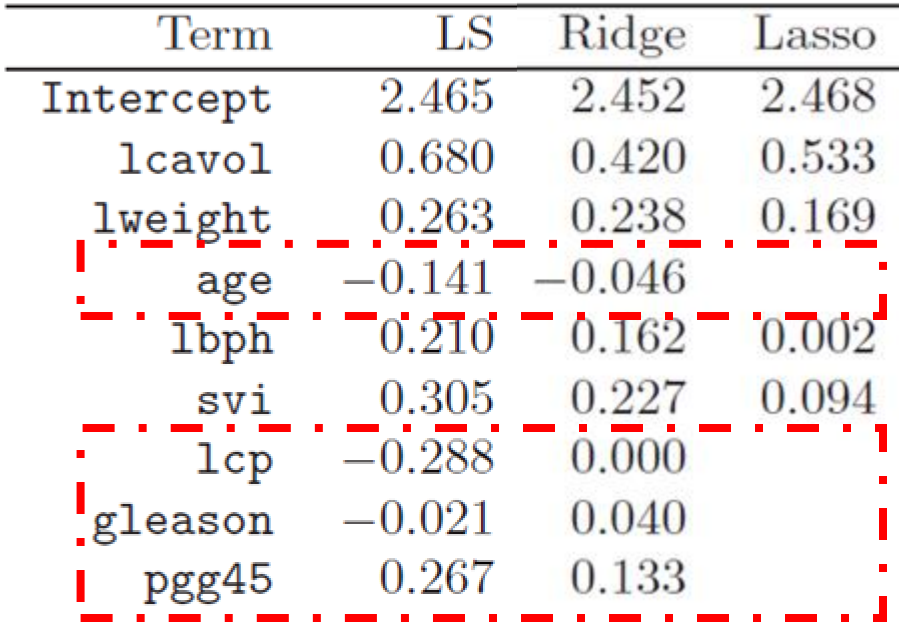
Rate of Prostate Cancer



https://www.medicalnewstoday.com/articles/age-range-for-prostate-cancer

Example: Prostate Cancer Dataset

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



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).

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

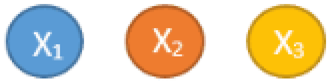
Age is highly correlated with other factors and thus *not significant* in the presence of those factors

Best-Subset Selection

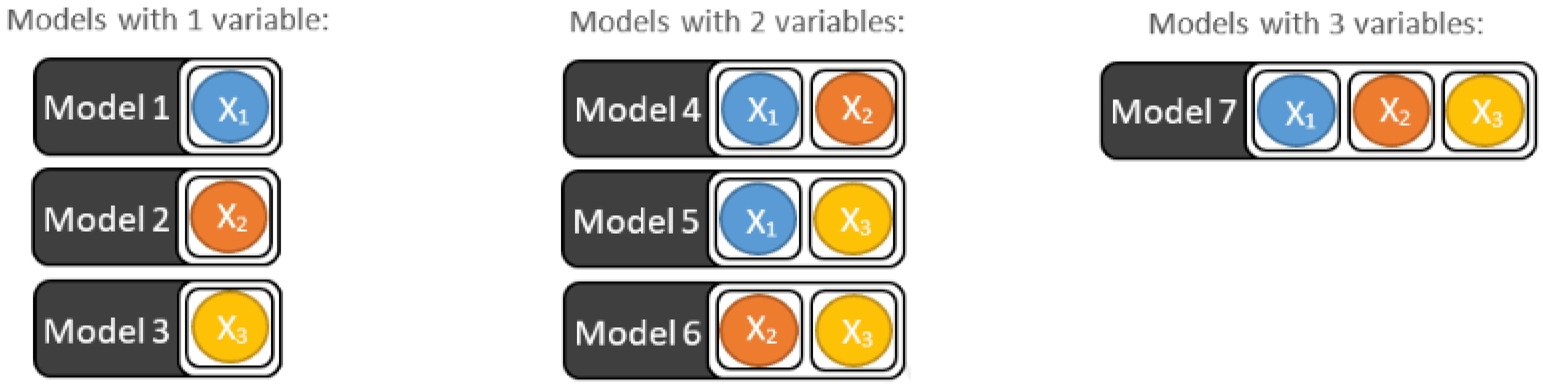
The optimal strategy for p features looks at models over *all possible combinations* of features,

```
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
```

Best-Subset Selection



age Log prostrate weight Log cancer volume



Feature Selection: Prostate Cancer Dataset

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

[Source: Hastie et al. (2001)]

Best-Subset Selection : Prostate Cancer Dataset

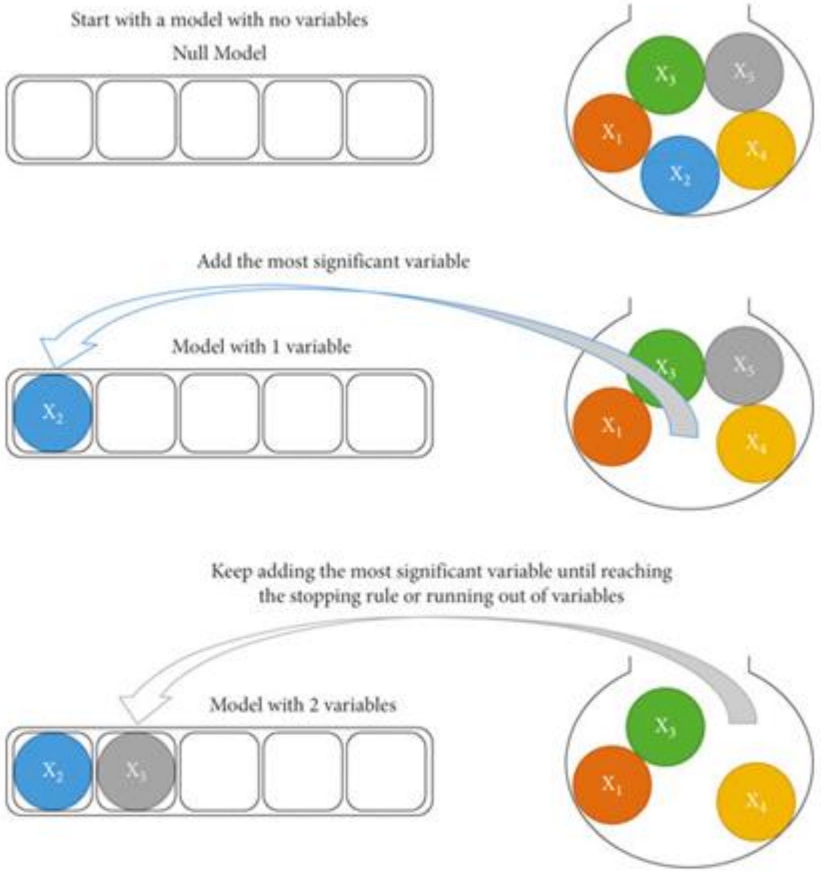
Time complexity

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

This is undesirable even for moderate p (e.g. $p = 20$)

Instead, we can use greedy algorithms to reduce time cost

Forward Sequential Selection

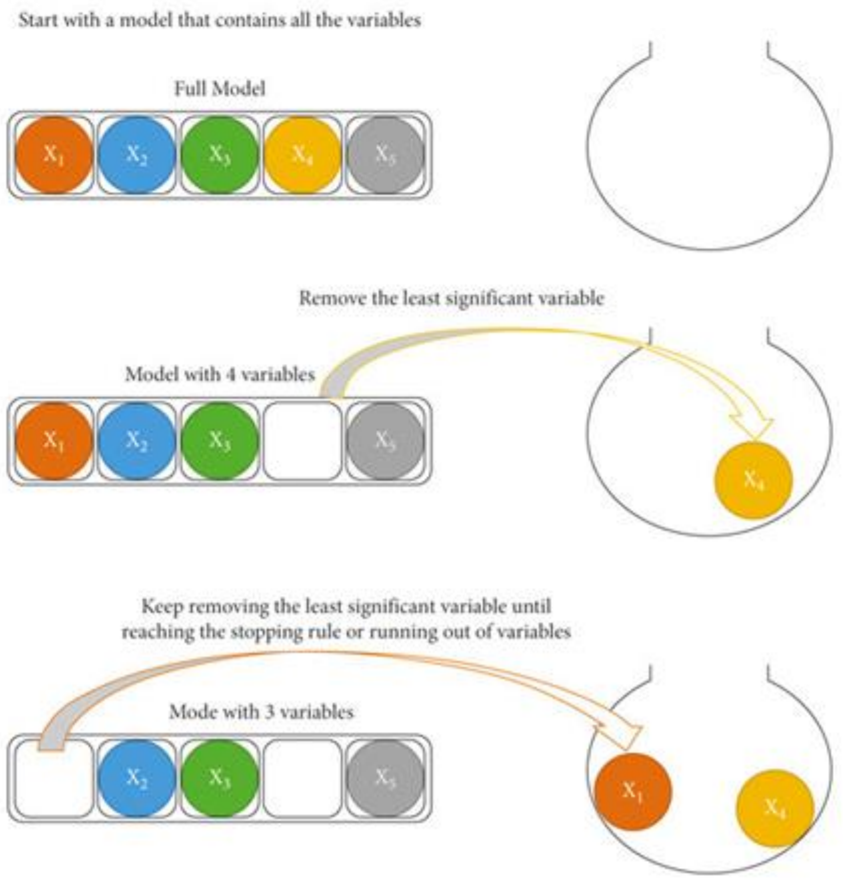

Start with a model with no variables
Null Model
X_1
X_2
X_3
X_4
X_5
Add the most significant variable
Model with 1 variable
Keep adding the most significant variable until reaching the stopping rule or running out of variables
Model with 2 variables

Forward Sequential Selection

An efficient method that 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
```

Backward Sequential Selection


Start with a model that contains all the variables
Full Model
X1
X2
X3
X4
X5
Remove the least significant variable
Model with 4 variables
X1
X2
X3
X5
X4
Keep removing the least significant variable until reaching the stopping rule or running out of variables
Mode with 3 variables
X2
X3
X5
X1
X4

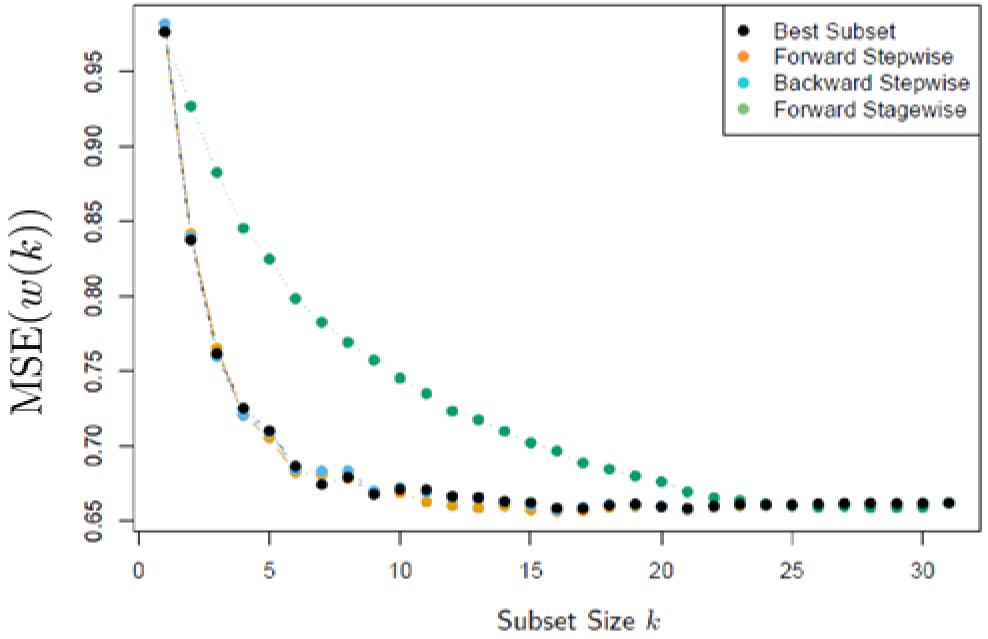
Backward Sequential Selection

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
```

Comparing Feature Selection Methods

Sequential selection is greedy, but often performs well…



Example Feature selection on data with p=30 features. 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 $O(2^p)$ for best subset

Sequential feature selection available in Scikit-Learn under:

`feature_selection.SequentialFeatureSelector`