Causal Inference (6.S059/15.C08/17.C08)

Recitation, Week 7.

Topic: Supervised Machine Learning

Licheng Liu

April 5, 2024

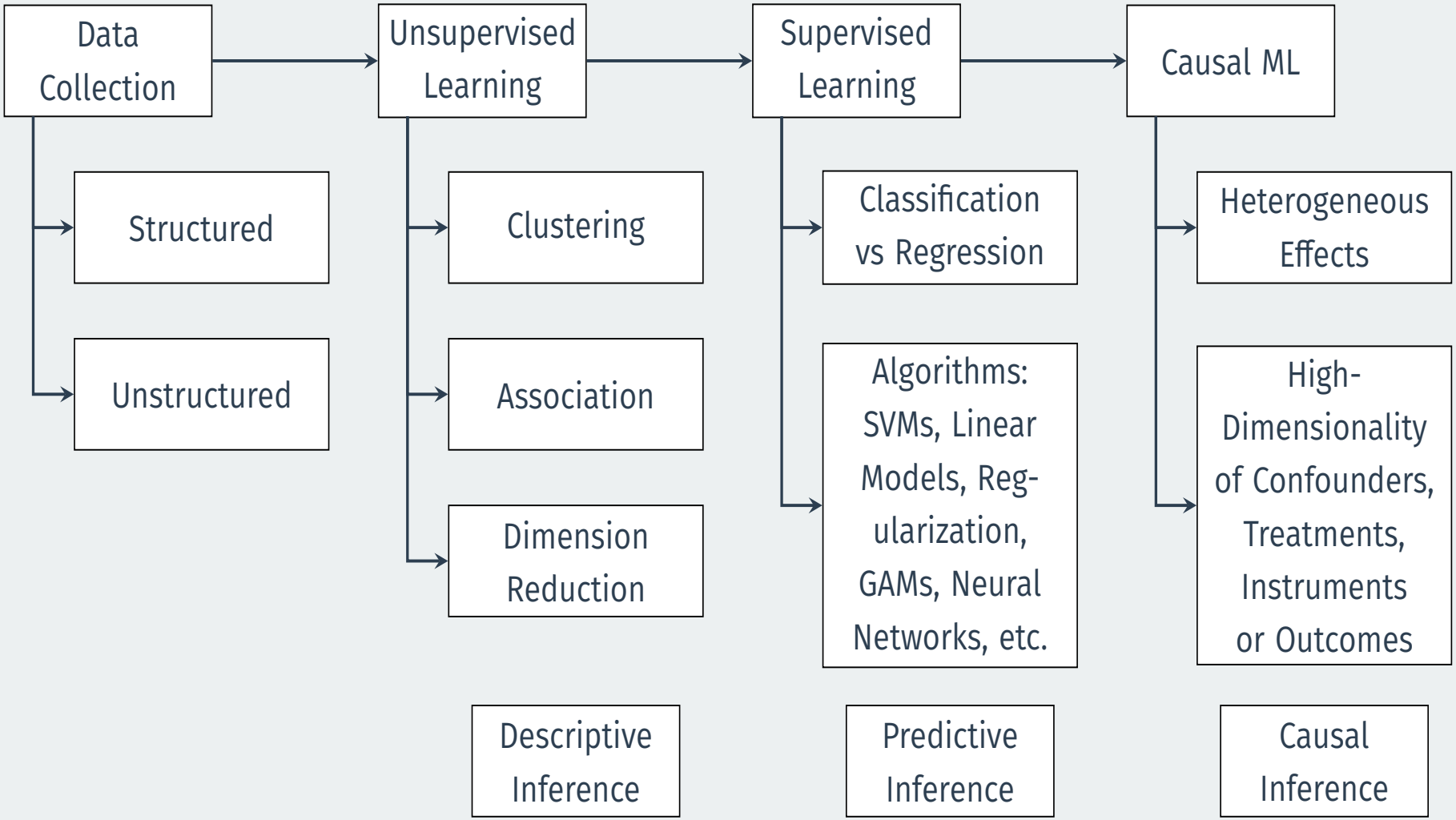
MIT

Massachusetts Institute of Technology

Table of contents

1/ Overview of Concepts

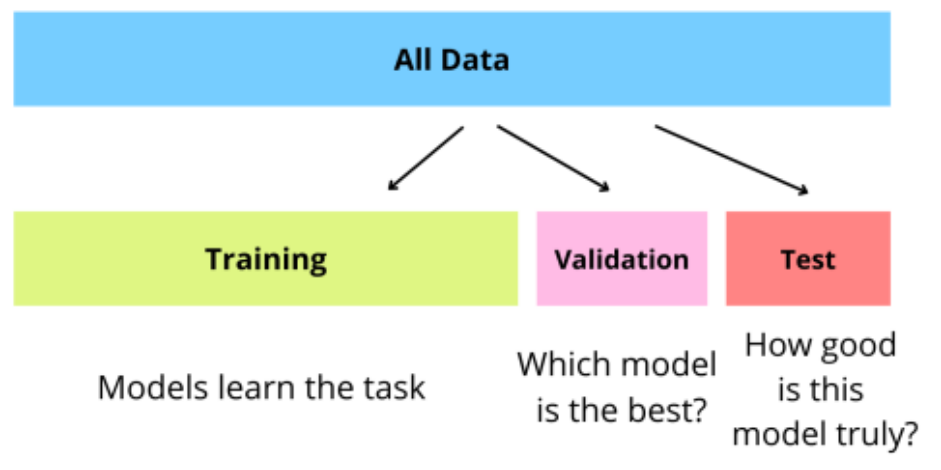
Machine Learning and Causal Inference



Supervised Learning

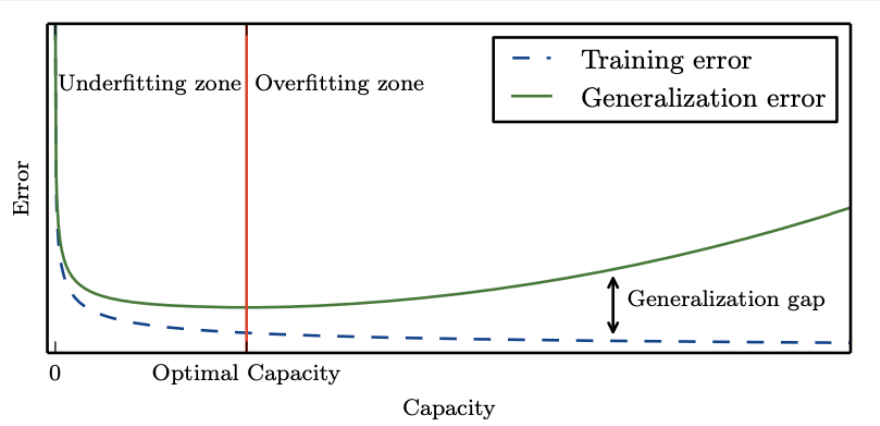
- Algorithms that experience "features" and a supervision signal (target or label). **Labeled Set** of input-output pairs ($D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$).
  - Using this data we build a **prediction** model or learner ($p(Y|\boldsymbol{X})$).
  - Good Learner: accurately predicts the target (*Performance Measure*).
  - Learner: Function $f(X)$ for predicting Y given the input vector $\boldsymbol{X}$. This requires a **Loss Function** ($L(Y, f(X))$) for penalizing errors in prediction.
- **Type of Outcome:** Regression vs Classification.

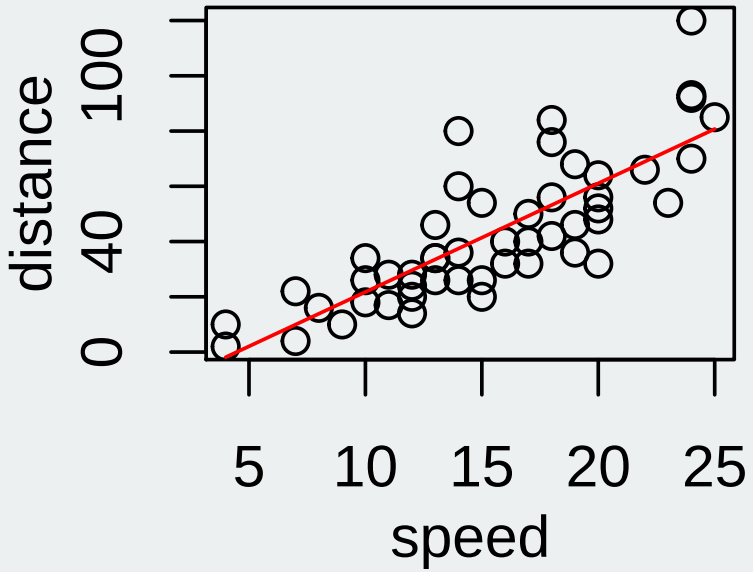
Workflow



- **Generalization:** Algorihm must perform well on *new, previously unseen* inputs.
- Minimization of Training Error $\longrightarrow$ Optimization Problem.
- Minimization of Generalization (Test) Error is the goal.

Model's Capacity, Bias-Variance Decomposition and Overfitting

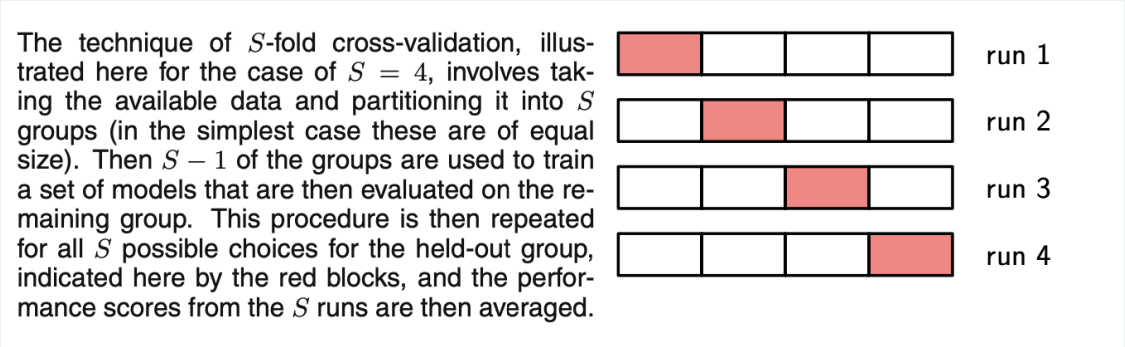




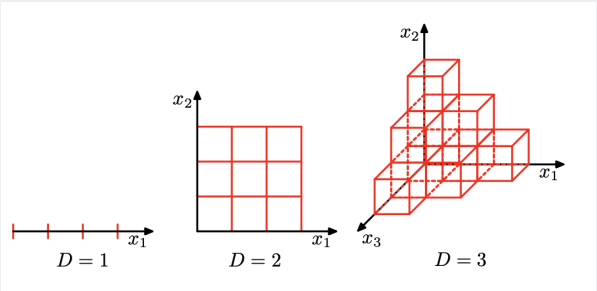
- Ability to fit a wide variety of functions:
  - Underfitting.
  - Overfitting.
- Another Tradeoff: Flexibility (Prediction Accuracy) and Interpretability.

Other Relevant Concepts

- **Cross-Validation:** resampling method that uses different portions of the data to test and train a model on different iterations.



- **Curse of Dimensionality:** increasing data dimensions and its explosive tendencies. We would need an exponentially large quantity of training data (and computational power) to explore the high-dimensional feature space.

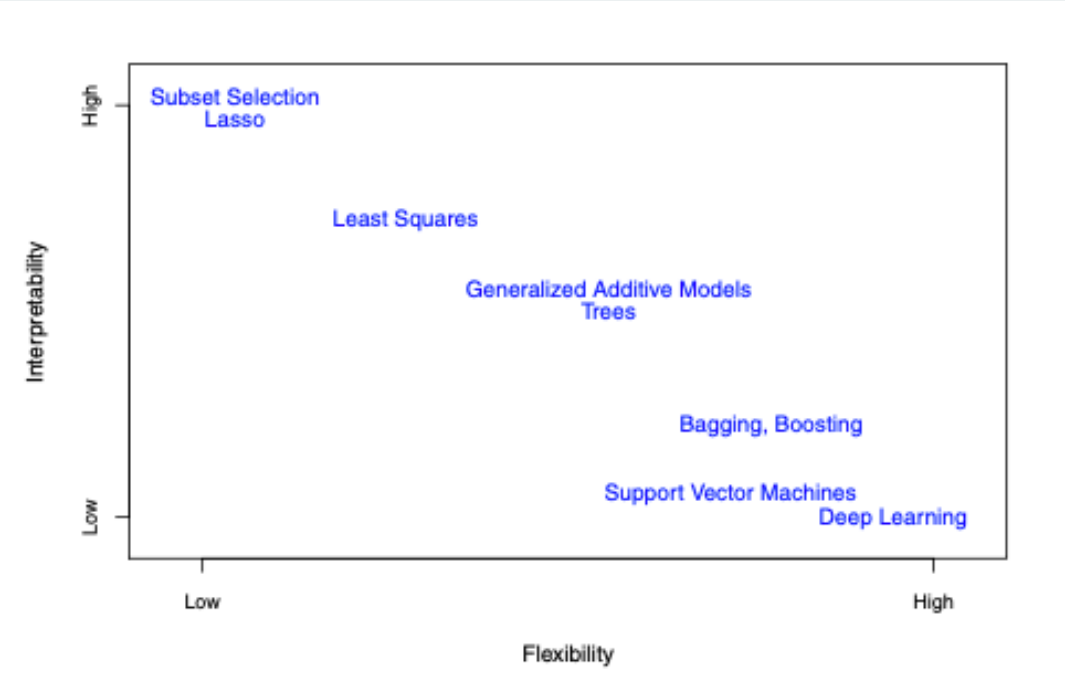


2/ Supervised Learning Models and Algorithms

Algorithms

- **No Free Lunch Theorem:** averaged over all possible DGPs, every algorithm has the same error rate when classifying previously unobserved points.
- No ML algorithm is universally any better than any other.
- Best performance: capacity is appropriate for the true complexity of the task and the amount of training data.

Algorithms


Subset Selection
Lasso
Least Squares
Generalized Additive Models
Trees
Bagging, Boosting
Support Vector Machines
Deep Learning
High
Low
Interpretability
Low
High
Flexibility

Regularization

- Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
- We can regularize a model that learns a function f(x) by adding a penalty (regularizer) to the loss function (controls model complexity $\longrightarrow$ overfitting).

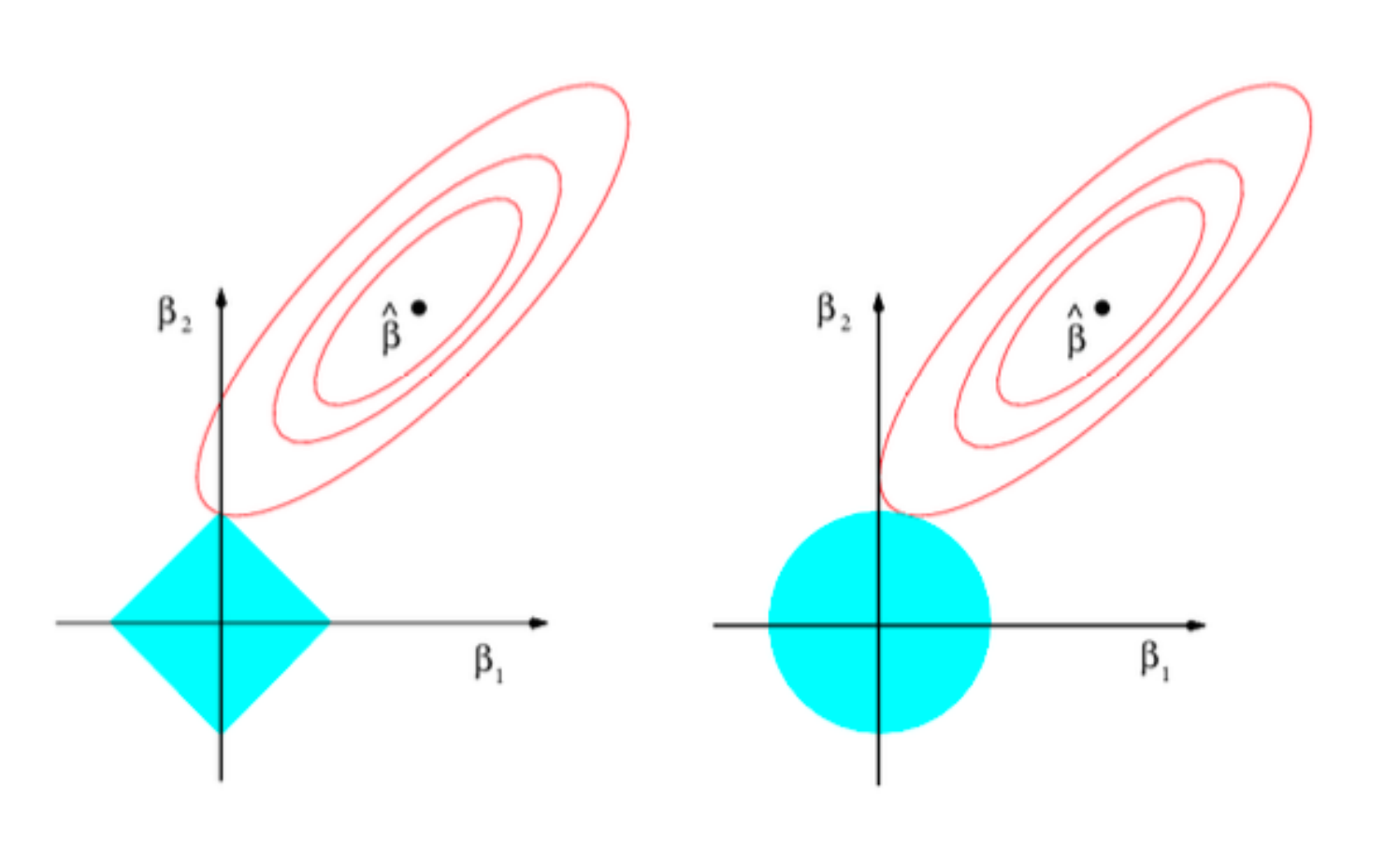
$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq s \tag{6.8}$$

and

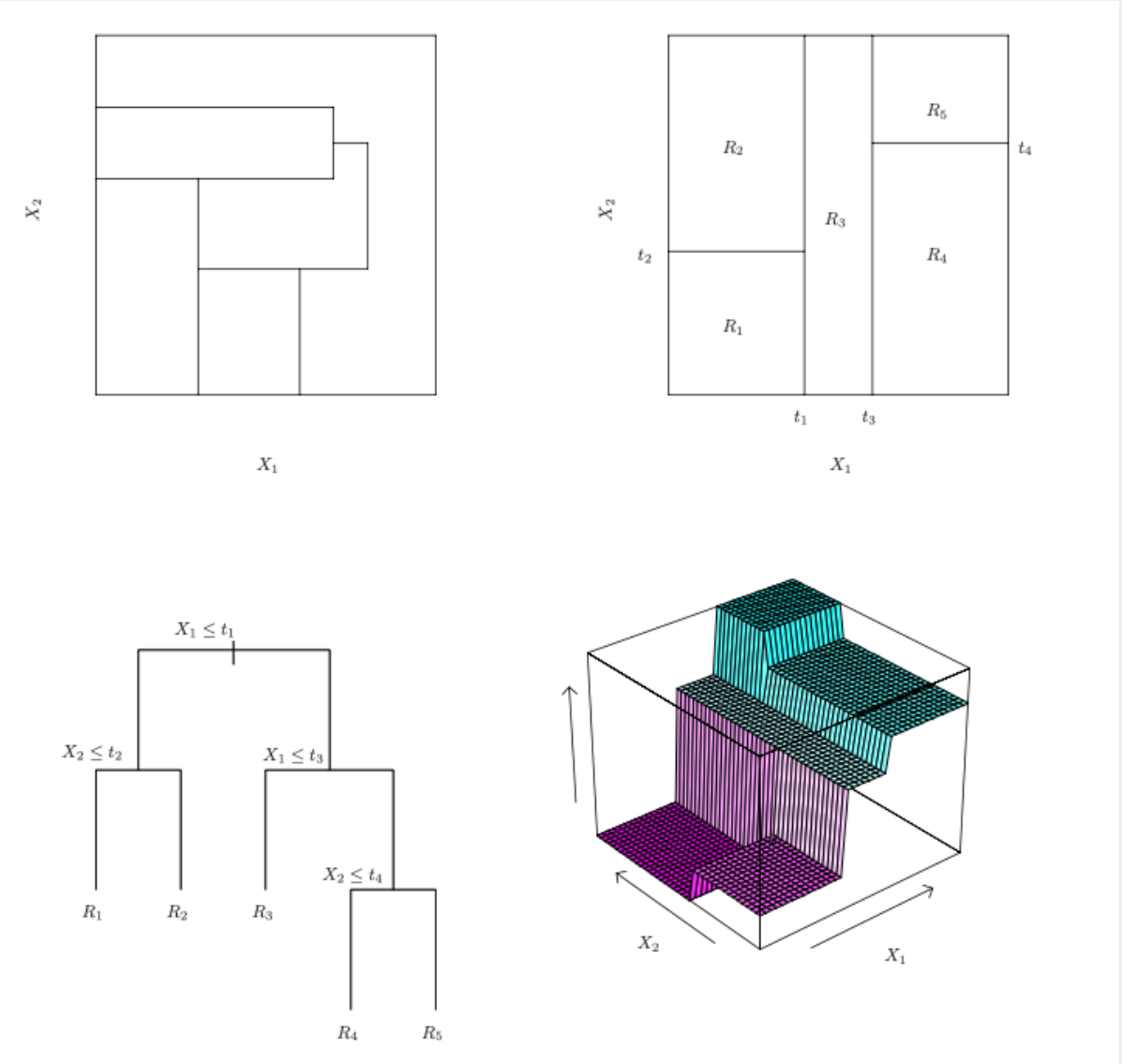
$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq s, \tag{6.9}$$

Regularization

- Ridge, LASSO, Elastic Net.
- LASSO for variable selection ($\beta_j^2 \longrightarrow |\beta_j|$).



Non-Linear (more Flexible) Algorithms



3/ Practice in R and Python

Cross-Validation

- How to split data?
- Approach 1: randomly assign folds to each observation.
- This does not ensure that the size of each fold is equal.
- Practically ok if your N is large enough compared to K.

R Code

```
set.seed(02139)
# 5-folds cross validation with 10 observations
K <- 5; N <- 10; x <- rnorm(N); y <- rnorm(N)
folds <- sample(1:K, size = N, replace = TRUE) # assign folds
x[folds == 5]; y[folds == 5] # fifth folds: four observation
## [1] -0.07963674 -0.81726793 -0.16561194 0.36658112
## [1] 0.64319207 0.18791807 -0.05517906 0.63358473
x[folds == 2]; y[folds == 2] # second folds: one observations
## [1] -0.232987
## [1] 1.295322
```

Cross-Validation

- How to make the fold size equal?
- Approach 2: randomly assign IDs to each observation and select these observations.
- By doing so, you can ensure that the size is equal.

R Code

```
# assign folds
IDs <- sample(1:N, size = N, replace = FALSE)
# there should be 2 obs. in each fold
# ID 1:2 belongs to the first fold, 3:4 belongs to the second fold ...
x[IDs %in% c(1:2)] # first fold
## [1] 0.7720908 1.7165340
x[IDs %in% c(3:4)] # second fold
## [1] 0.9728744 0.3665811
```

Indicator function and Polynomials

- e.g., 4-th degree (order) polynomial regression

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \epsilon$$

- Terms x and x^2 will be highly correlated where $x > 0$
- `poly` function creates a "curved" set of variables from quadratic terms and makes sure that each term is not highly correlated to each other
- Technically, `poly` with the default setting computes "orthogonal polynomials" where each polynomial is orthogonal

R Code

```
#Load data
data(cars)

#Run Model
m1 <- lm(dist ~ speed + I(speed^2), data = cars)
m2 <- lm(dist ~ poly(speed, degree = 2, raw = FALSE), data = cars)
m3 <- lm(dist ~ poly(speed, degree = 2, raw = TRUE), data = cars)

# Correlations
cor(model.matrix(m1)[, 2], model.matrix(m1)[, 3])
## [1] 0.9794765
cor(model.matrix(m2)[, 2], model.matrix(m2)[, 3])
## [1] 4.686464e-17
cor(model.matrix(m3)[, 2], model.matrix(m3)[, 3])
## [1] 0.9794765
```

LASSO

- Use the package `glmnet`.
  - `x:` input matrix.
  - `y:` target vector.
  - `family:` type of model (“binomial” for binary outcome) .
  - `alpha:` mixing parameter (elastic net) .
  - `lambda:` penalty parameter.
  - `standardize:` standardization of X (default = TRUE).
  - `exclude:` exclude variables from regularization/variable selection.
  - `maxit:` number of iterations.

R Code

```
#Load package
library(glmnet)

#Load data
data(QuickStartExample)
x <- QuickStartExample$x
y <- QuickStartExample$y

#Run learner
l1 <- glmnet(x=x, y=y, family = "gaussian", lambda = 0) #Ridge
l2 <- glmnet(x=x, y=y, family = "gaussian", lambda = 1) #LASSO
```

Random Forest

- Use the package `randomForest` or `ranger`.
  - `x:` input matrix.
  - `y:` target vector.
  - `ntree:` number of trees.
  - `mtry:` number of variables randomly sampled.
  - `replace:` sampling with or without replacement.
  - `cutoff:` threshold point (only for classification).
  - `nodesize:` maximum number of terminal nodes.

R Code

```
#Load package
library(randomForest)

data(iris)
set.seed(71)

iris.rf <- randomForest(
    x = as.matrix(iris[, c("Sepal.Length", "Sepal.Width",  "Petal.Length", "Petal.Width")]),
    y = iris$Species, ntree = 500)
```

Neural Networks

- Use the package `nnet`.
  - `x`: input matrix.
  - `y`: target vector.
  - linout: switch for linear output units. Default logistic output units.
  - entropy: switch for entropy (= maximum conditional likelihood) fitting.
  - `softmax`: switch for softmax (log-linear model).
  - censored: A variant on softmax, in which non-zero targets mean possible classes. linout, entropy, softmax and censored are mutually exclusive.

R Code

```
#Load package
library(randomForest)

data(iris)
set.seed(71)

iris.nn <- nnet(
    x = as.matrix(iris[, c("Sepal.Length", "Sepal.Width",  "Petal.Length", "Petal.Width")]),
    y = iris$Species, data=iris, maxit = 200)
```

Implementing ML Algorithms in R

- All the ML algorithms have easy-to-implement R packages/functions!

R Code

```
# LASSO
lasso <- glmnet(x = xtrain, y = ytrain, family = 'binomial', lambda = 0.1)
# check out the cv.glmnet also!

# random forest
rf <- randomForest(x = xtrain, y = factor(ytrain))

# neural network
nn <- nnet(x = xtrain, y = ytrain, size = 10, entropy = TRUE, trace = FALSE)

# support vector machine
library(e1071)

# svm linear kernel
svm_lin <- svm(x = xtrain, y = factor(ytrain), kernel = "linear")

# svm radial kernel
svm_rbf <- svm(x = xtrain, y = factor(ytrain), kernel = "radial")
```

Implementing ML Algorithms in Python

- All the ML algorithms have easy-to-implement Python libraries/functions!
- Popular Python libraries for machine learning: PyTorch, keras, scikit-learn, TensorFlow, etc.

Python Code

```
# LASSO

from glmnet import LogitNet
# Initialize and fit the LASSO model
lasso = LogitNet(alpha=1, lambda_path=np.array([0.1]))
lasso = lasso.fit(xtrain, ytrain)

# random forest

from sklearn.ensemble import RandomForestClassifier
# Initialize the RandomForestClassifier
rf = RandomForestClassifier()

# Fit the model to the data
rf.fit(xtrain, ytrain)
```

Implementing ML Algorithms in Python (Cont')

Python Code

```
# neural networks

from sklearn.neural_network import MLPClassifier

# Initialize the MLPClassifier
nn = MLPClassifier(hidden_layer_sizes=(10,), activation='relu', solver='adam',
                   max_iter=200, verbose=False, random_state=1)

# Fit the model to the data
nn.fit(xtrain, ytrain)


# support vector machine

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline


# SVM with Linear Kernel
svm_lin = make_pipeline(StandardScaler(), SVC(kernel='linear'))
svm_lin.fit(xtrain, ytrain)

# SVM with Radial Basis Function (RBF) Kernel
svm_rbf = make_pipeline(StandardScaler(), SVC(kernel='rbf'))  # Note: 'rbf' is the default kernel
svm_rbf.fit(xtrain, ytrain)
```