Chapter 5
Supervised learning methods
Part I - Regression

Simon Fraser University
ECON 483
Summer 2023

Disclaimer

I do not allow this content to be published without my consent.

What is “machine learning (ML)”?

- There are many definitions that differ on several aspects
- Wikipedia says: “Machine learning (ML) is the study of computer algorithms that improve automatically through experience.[1] It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as”training data”, in order to make predictions or decisions without being explicitly programmed to do so”
- 1997 by Professor Tom M. Mitchel from Carnegie Mellon University, in his famous quote from (1997) “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E”.
- Self-driving cars use AI systems, the automatic vision system that identifies an imminent accident is ML

Outline

- **Linear models: shrinkage methods**
- **Regression trees**
  - **Principle**
  - **Trees and linear regression**
  - **Pruning**
  - **Boosting**
- **Random forests**
- **Neural networks**
- Suggested reading: Chapter 8 in **ISLR**

Supervised learning

Supervised learning

- Supervised problems are the most frequent ones for an economist
- Problems are said supervised when there is an outcome variable $\boldsymbol{Y_i}$, so methods can be "supervised" (we can assess the performance of the methods)
- We went over linear regression and some non parametric methods, for either the purpose of **estimation** or **prediction**. Both are supervised as well
- In this lecture, we will focus on **prediction** problems
- So **flexibility** is a major asset for finding the optimal bias-variance trade off
- For regression problems, we will cover
  - **Shrinkage methods**
  - **Trees** and **random forests**
  - **Neural networks**

Shrinkage methods in linear models

Shrinkage methods

- **Shrinkage methods** estimate full linear models (with all the variables), with a constraint on the value of the coefficients
- Shrinking the estimates can significantly decrease the variance
- We talk about **shrinkage** or **regularization**

Ridge regression

- **Ridge regression** constrains the sum of the squares of the coefficient estimates. In a linear model, the coefficients minimize the following:

$$\hat{\beta}_{Ridge} \equiv \underset{\{\beta\}}{\mathbf{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- λ is a **penalty term**. If $\sum_{j=1}^{p} \beta_j^2$ is high, the overall quantity is high
- In the process, ridge regression will set some $\hat{\beta}$ to low values, but not 0. So predictions are better, but model interpretation can be an issue
- λ should also be found. Use cross validation over a grid of values for λ. For each λ, the model is estimated. Chose the value of λ that yields the lowest cross validated error

LASSO

- **Least Absolute Shrinkage and Selection Operator** constrains the sum of the coefficient estimates. In a linear model, the coefficients minimize the following:

$$\hat{\beta}_{LASSO} \equiv \underset{\{\beta\}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - x_i'\beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- λ is a **penalty term**. If $\sum_{j=1}^{p} |\beta_j|$ is high, the overall quantity is high
- In the process, the LASSO will set some $\hat{\beta}$ to 0
- λ can be found via cross validation
- In practice, neither ridge nor LASSO dominate the other. Try both!

Elastic nets

- **Elastic nets** are a combination of LASSO and Ridge regressions. In a linear model, the coefficients minimize the following:

$$\hat{\beta}_{elastic} \equiv \underset{\{\beta\}}{\mathbf{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2)$$

- α tells us if we go for LASSO ($\alpha = 1$), Ridge ($\alpha = 0$) or a combination of both ($0 < \alpha < 1$)
- λ can be found via cross validation (see next sections)

Ridge, LASSO and elastic nets in Rstudio

- Consider the **Credit** data set from the **ISLR** library
- Reports the **Balance** (average credit card debt. Not lower than zero here) for a number of individuals with:
  - Quantitative predictors: **age, education, income**
  - Qualitative predictors: **gender, student, status, ethnicity**
- We want to predict **Balance**: what combination of variables to include?

```
data(Credit)
Credit
n_obs <- nrow(Credit)
Credit_1 <- Credit[1:floor(n_obs/3),]
Credit_2 <- Credit[(floor(n_obs/3)+1):floor(2*n_obs/3),]
Credit_3 <- Credit[(floor(2*n_obs/3)+1):n_obs,]
Credit_12 <- rbind(Credit_1, Credit_2) # Combine the first 2 subsets
K <- ncol(Credit_12)  # number of variables
```

Ridge, LASSO and elastic nets in Rstudio

```
# Ridge regression
cv_ridge <- cv.glmnet(x = data.matrix(Credit_12[ , 1:(K - 1) ]), y = Credit_12$Balance,
                      alpha = 0, family="gaussian")
ridge_lambda <- cv_ridge$lambda.min # optimal lambda for Ridge
ridge <- glmnet(y = Credit_12$Balance, x = Credit_12[ , 1:(K - 1) ],
                alpha = 0, family="gaussian")
# make predictions
ridge_fit <- predict(ridge, newx = data.matrix(Credit_3[ , 1:(K - 1)]), s = ridge_lambda )

# LASSO regression
cv_lasso <- cv.glmnet(x = data.matrix(Credit_12[ , 1:(K - 1) ]), y = Credit_12$Balance,
                      alpha = 1, family="gaussian")
lasso_lambda <- cv_lasso$lambda.min # optimal lambda for Lasso
lasso <- glmnet(y = Credit_12$Balance, x = Credit_12[ , 1:(K - 1) ],
                alpha = 1, family="gaussian"  )
# make predictions
lasso_fit <- predict(lasso, newx = data.matrix(Credit_3[ , 1:(K - 1)]), s = lasso_lambda)

# Elastic net
cv_net <- cv.glmnet(x = data.matrix(Credit_12[ , 1:(K - 1) ]), y = Credit_12$Balance,
                    alpha = 0.5, family = "gaussian")
net_lambda <- cv_net$lambda.min  # optimal lambda for elastic nets
elastic_net <- glmnet(y = Credit_12$Balance, x = Credit_12[ , 1:(K - 1) ],
                      alpha = 0.5, family = "gaussian"  )
# make predictions
elastic_fit <- predict(elastic_net, newx = data.matrix(Credit_3[ , 1:(K - 1)]), s = net_lambda)
```

Regression trees

What is a regression tree?

- One of the most straightforward, easy to implement machine learning algorithms
- Very visual, so easy to explain. Actually called **dendrograms**, i.e. tree diagrams
- Used for **prediction problems**, in particular with a lot of covariates ("Big data")
- Principle: make **binary** splits of the data according to covariate values (take an $\boldsymbol{X}$ and split the data in two according to some threshold value of $\boldsymbol{X}$)
- It creates different **regions** (or **terminal nodes** or **leaves**). Each observation falls in one region only
- The prediction for one observation is the average of all the observations in that region

Example

- Say we want to predict someone's income based on a bunch of characteristics
- We have a sample of data, where each observation is an individual for which we observe income, age, gender, and number of kids
- We want to predict income, so we build a tree based on the other characteristics
- Observations with $\boldsymbol{Age < 30}$ vs $\boldsymbol{Age \geq 30}$ are separated
- It opens two branches: Another binary split is then made in each branch, using another variable
  - Left branch: $\boldsymbol{nkids \leq 1}$ vs $\boldsymbol{nkids > 1}$
  - Right branch: $\boldsymbol{Male}$ vs $\boldsymbol{Female}$
- Final prediction of $\boldsymbol{Income}$ for individual $\boldsymbol{i}$: average income of all individuals in the same region as Mr. $\boldsymbol{i}$. So **one** prediction per region

Building a regression tree

- How do we build a regression tree?
  - How do we decide which variable to make a split over?
  - How do we decide of the value of the variable to make a split over?
  - How complex do we want our tree to be?
- Like for the OLS estimator, we want our predictions to be as close as possible to the actual data on average
- Define the regions $\boldsymbol{R_1(j,s) \equiv \{x | x_j < s\}}$ and $\boldsymbol{R_2(j,s) \equiv \{x | x_j \geq s\}}$
- Define the residual sum of squares

$$\boldsymbol{SSR \equiv \sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2}$$

Building a regression tree: Algorithm

Growing a tree: Algorithm

- Find the variable $\boldsymbol{X_j}$ and the threshold value $\boldsymbol{s}$ that minimizes $\boldsymbol{SSR}$ (loop over all possible covariates and values)
- Make the split according to the selected covariate and threshold
- **Repeat** the two previous steps in each newly created region until a criterion is satisfied (for instance, until each region has less than a given number of observations)
- **Compute** predictions by taking the average of the variable of interest for all observations in a terminal node (or leaf)

Regression trees in **Rstudio**

- There are several packages that build regression trees in **R**
- We are going with the ***rpart*** package, and will use the ***rpart*** function
- Let us consider the **Carseats** data set, which consists of child car seats sales, along with market related measures (population in an area, median income, ...)

Regression trees in Rstudio

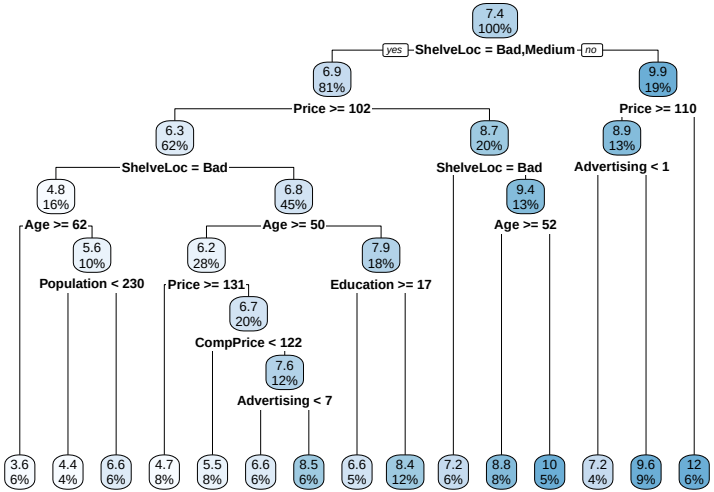
- **Sales**: Unit sales (in thousands) at each location
- **CompPrice**: Price charged by competitor at each location
- **Income**: Community income level (in thousands of dollars)
- **Advertising**: Local advertising budget for company at each location (in thousands of dollars)
- **Population**: Population size in region (in thousands)
- **Price**: Price company charges for car seats at each site
- **ShelveLoc**: A factor with levels Bad, Good and Medium indicating the quality of the shelving location for the car seats at each site
- **Age**: Average age of the local population
- **Education**: Education level at each location
- **Urban**: A factor with levels No and Yes to indicate whether the store is in an urban or rural location
- **US**: A factor with levels No and Yes to indicate whether the store is in the US or not

Regression trees in **Rstudio**

```
library(ISLR)
carseats <- Carseats  # Data set form the ISLR package

train_index <- sample (1: nrow(carseats ), nrow(carseats )/2)
test_sample <- carseats[- train_index, ]
train_sample <- carseats[train_index, ]
tree_carseats <- rpart(Sales ~ ., data = train_sample, method = "anova")
# summary(tree_carseats )
# plotcp(tree_carseats)
# printcp(tree_carseats)
rpart.plot(tree_carseats)
```

Regression tree in **Rstudio**

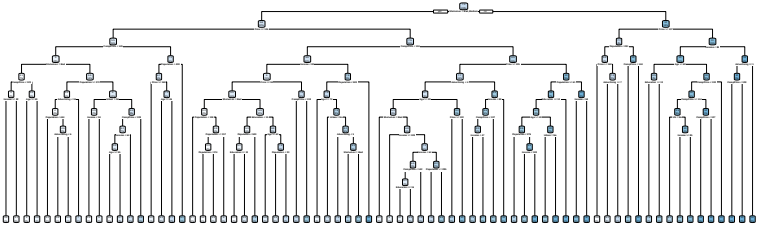


Predicting other samples: Illustration in Rstudio

```
# Predict the test sample
tree_fit <- predict(tree_carseats, newdata = test_sample)
head(tree_fit)
```

```
##         1         2         3         6         8        12
##  6.570000 12.024167  8.773125  7.150769  9.644444 12.024167
```

Regression trees in **Rstudio:** Overfitting



Trees and linear regression

- Trees can be assimilated to linear regressions with dummy variables denoting the different thresholds
- Let $\boldsymbol{R_m}$ be region $\boldsymbol{m}$
- Let $\boldsymbol{D_m = \mathbb{1}\{i \in R_m\}}$, $\boldsymbol{i = 1, \ldots, M}$ be a binary variable equal to 1 if $\boldsymbol{i}$ is in region $\boldsymbol{m}$, and 0 otherwise
- A regression tree can therefore be represented as

$$\boldsymbol{Y_i = \alpha_1 D_1 + \alpha_2 D_2 + \ldots + \alpha_M D_M + u_i}$$

- Estimated by OLS, we get

$$\boldsymbol{\hat{\alpha}_m = \frac{1}{n_m} \sum_{i \in R_m} Y_i}$$

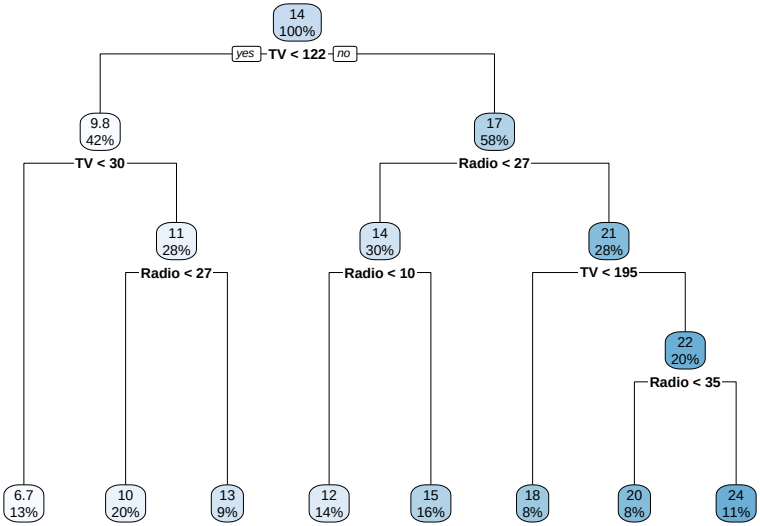
where $\boldsymbol{n_m}$ is the number of observations inside region $\boldsymbol{R_m}$

Trees and linear regression: Illusration

- Consider the ***Advertising*** data set, composed of 4 variables
  - ***Sales*** of a product in 200 different markets
  - ***TV***, ***Radio*** and ***Newspaper*** report budgets for the three different media

```
# Tree version
advertising <- read.csv("Advertising.csv")
tree_advertising <- rpart(Sales ~ ., data = advertising, method = "anova")
```

Trees and linear regression: Illustration



Trees and linear regression: Illustration

- Using the thresholds found in the tree, we can define the binary variables corresponding to each final region:
  - $D_1 = \mathbb{1}\{TV < 30\}$
  - $D_2 = \mathbb{1}\{TV < 122 \,\&\, TV > 30 \,\&\, Radio < 27\}$
  - $D_3 = \mathbb{1}\{TV < 122 \,\&\, TV > 30 \,\&\, Radio > 27\}$
  - $D_4 = \mathbb{1}\{TV > 122 \,\&\, Radio < 10\}$
  - $D_5 = \mathbb{1}\{TV > 122 \,\&\, Radio < 27 \,\&\, Radio > 10\}$
  - $D_6 = \mathbb{1}\{TV > 122 \,\&\, Radio > 27 \,\&\, TV < 195\}$
  - $D_7 = \mathbb{1}\{TV > 122 \,\&\, Radio > 27 \,\&\, TV > 195 \,\&\, Radio < 35\}$
  - $D_8 = \mathbb{1}\{TV > 122 \,\&\, Radio > 27 \,\&\, TV > 195 \,\&\, Radio > 35\}$

$$\begin{aligned} Y_i = \alpha_1 D_1 + \alpha_2 D_2 \\ + \alpha_3 D_3 + \alpha_4 D_4 \\ + \alpha_5 D_5 + \alpha_6 D_6 \\ + \alpha_7 D_7 + \alpha_8 D_8 + u_i \end{aligned}$$

Trees and linear regression: Illustration

```
# Linear regression version
# Left part of the tree
D_1 <- ifelse(advertising$TV < 30, 1, 0)
D_2 <- ifelse(advertising$TV < 122 & advertising$TV > 30
              & advertising$Radio < 27, 1, 0)
D_3 <- ifelse(advertising$TV < 122 & advertising$TV > 30
              & advertising$Radio > 27, 1, 0)
# Right part of the tree
D_4 <- ifelse(advertising$TV > 122 & advertising$Radio < 27
              & advertising$Radio < 10, 1, 0)
D_5 <- ifelse(advertising$TV > 122 & advertising$Radio < 27
              & advertising$Radio > 10, 1, 0)
D_6 <- ifelse(advertising$TV > 122 & advertising$Radio > 27
              & advertising$TV < 195, 1, 0)
D_7 <- ifelse(advertising$TV > 122 & advertising$Radio > 27
              & advertising$TV > 195 &  advertising$Radio < 35, 1, 0)
D_8 <- ifelse(advertising$TV > 122 & advertising$Radio > 27
              & advertising$TV > 195 &  advertising$Radio > 35, 1, 0)
```

Trees and linear regression: Illustration

```r
summary(lm(Sales ~ -1 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7 + D_8, data = advertising))
```

```
##
## Call:
## lm(formula = Sales ~ -1 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6 +
##     D_7 + D_8, data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.1423 -0.9685  0.0092  1.0449 12.6000
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## D_1   6.7423     0.3454   19.52   <2e-16 ***
## D_2  10.2641     0.2821   36.39   <2e-16 ***
## D_3  13.1000     0.4152   31.55   <2e-16 ***
## D_4  11.8357     0.3329   35.56   <2e-16 ***
## D_5  15.3818     0.3066   50.16   <2e-16 ***
## D_6  17.9588     0.4272   42.04   <2e-16 ***
## D_7  19.6438     0.4404   44.61   <2e-16 ***
## D_8  23.7227     0.3755   63.17   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.761 on 192 degrees of freedom
## Multiple R-squared:  0.9867, Adjusted R-squared:  0.9861
## F-statistic:  1779 on 8 and 192 DF,  p-value: < 2.2e-16
```

Refining trees: Pruning

- A highly complex tree (many splits, each leaf has a small amount of observations) will predict the training data well, but the test data poorly due to **overfitting**
- Extreme case: A tree where each region only has one observation in it
- That model (tree) will feature a high variance
- **Pruning** consists in trimming some branches of a very large tree. Call it $\boldsymbol{T_0}$
- How? By including a **penalty term**

Refining trees: Pruning

- Let α be a non negative number
- For each α, look for a tree T that minimizes

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

- The inner sum is the sum of squared residuals inside a given leaf R_m
- The outer sum adds all the squared residuals over all the leaves
- The last term penalizes complex trees, i.e. trees with a high number of terminal nodes $|T|$
- To each α, an optimal tree. Which α to choose? **Validation sets** or **CV**!

Pruning: Algorithm

Pruning a tree: Algorithm

- Grow a large tree $\boldsymbol{T_0}$
- Make a grid of values for $\boldsymbol{\alpha}$. For each $\boldsymbol{\alpha}$, find the best subtree of $\boldsymbol{T_0}$
- To find the optimal $\boldsymbol{\alpha}$:
  - Divide the sample into $\boldsymbol{K}$ folds. $\boldsymbol{K = 10}$ is standard
  - Repeat the first two steps on all but the $\boldsymbol{k^{th}}$ fold (grow a big tree, and prune it for each $\boldsymbol{\alpha}$)
  - Evaluate the test MSE on the fold left out
  - Repeat the CV steps by leaving out another fold. Choose $\boldsymbol{\alpha}$ that minimizes the average of the test MSEs
- Use the optimal $\boldsymbol{\alpha}$ to prune $\boldsymbol{T_0}$

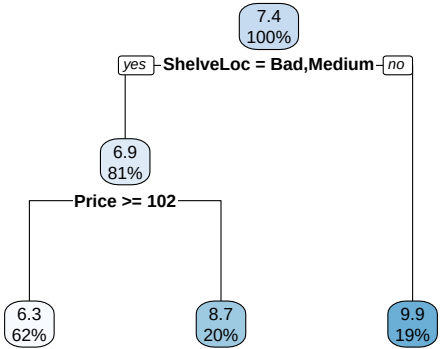
Pruning: Illustration in **Rstudio**

- In the ***rpart*** function, ***cp*** ("complexity parameter") allows to vary the amount of pruning
- Any split that does not decreases the overall lack of fit by a factor of ***cp*** is not attempted
- The bigger ***cp***, the more pruning there is, i.e. the less splits are made as it is harder for a split to improve the fit by that much
- The lower ***cp***, the less pruning there is, i.e. the more splits are made as small improvements in the fit are allowed

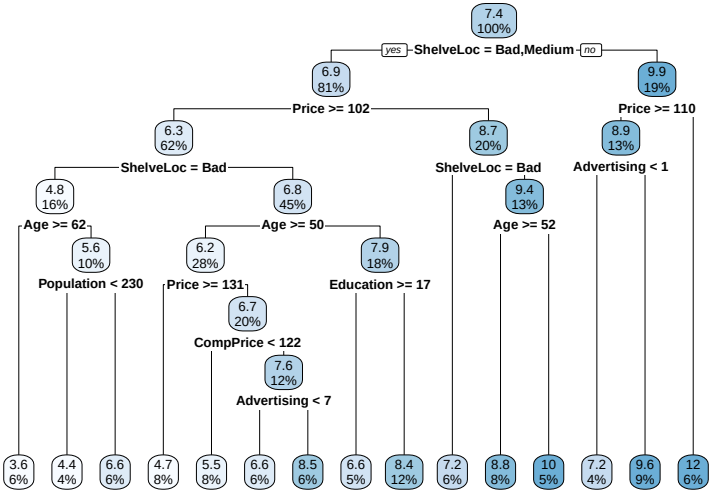
Pruning: Illustration in Rstudio

```
# prune the tree
# Extreme pruning
pfit_extreme <- prune.rpart(tree_carseats, cp = 0.1)
# almost no pruning
pfit_no_pruning <- prune.rpart(tree_carseats, cp = 0.001)
```

Extreme pruning: Illustration in **Rstudio**



Light pruning: Illustration in **Rstudio**



Boosting

- **Boosting** is another way to improve the performance of trees
- Idea: Grow trees based on previous trees

Boosting for trees: Algorithm

- Start with a simple tree (1 or 2 splits). Obtain $\hat{y}_i$ and compute the residuals $r_i = y_i - \lambda\hat{y}_i$. λ is a small value so trees improve slowly. Typically 0.01 or 0.001
- Grow a simple tree using the residuals as the dependent variable
- Repeat the process P times
- How to find P? CV!

- No resampling here, we just grow small trees based on what is left from the previous small tree

Bootstrap aggregating (aka Bagging)

- Growing one tree is nice, but it might still suffer from **high variance**
- We saw that averaging always reduces variance. Does not affect the bias though
- Let's apply this concept here: Grow trees (and prune them) using $\boldsymbol{B}$ separate training sets
- How to get $\boldsymbol{B}$ separate training sets? Use the **bootstrap**:
  - Re-sample your training data, with replacement (Note: some observations might appear twice)
  - Run your model on your new sample
  - Repeat $\boldsymbol{B}$ times. Typically $\boldsymbol{B}$ is between 100 and 500
- Since an estimator is random, we typically use its **asymptotic distribution** to conduct inference
- **Bootstrapping** is an alternative if the asymptotic distribution approximation of an estimator is suspicious: the $\boldsymbol{B}$ estimates can be used to approximate the finite sample distribution of the estimator

Random forests

- One can take advantage of **model averaging** and **bootstrapping** to improve trees
- **Random forests** take the average predictions of the $\boldsymbol{B}$ trees
- **Reduces the variance, for the same bias**
- Additional feature: Taking the average of correlated trees has a higher variance than uncorrelated trees
- So a random subset of $\boldsymbol{X}$'s is selected for each tree, so trees are less correlated (because they won't use the same covariates)
- The result is a **random forest**: Same bias as a tree, but lower variance, so better out-of-sample predictive power

Random forests in R

Advantages and disadvantages of trees

- Advantages:
  - Easy to explain, visually appealing. Anyone can get the gist of it by just looking!
  - Some say they more closely represent human decision-making than other types of regression
  - Trees can handle qualitative covariates easily, without the need to create many dummy variables
- Disadvantages:
  - Trees don't feature the same level of predictive accuracy as other methods (like neural networks)
  - Trees can be very non-robust: a small change in the data can have a big change on the structure of the tree (that caveat is offset by bagging, boosting and random forests)

Neural networks

Neural networks

- Ever hard of “deep learning” or “neural networks”?
- The idea is based on the human brain’s structure: Neurons are interconnected through **layers**
- Each **layer** contains multiple **nodes**
- Each node is a linear combination of the nodes from the previous layer
- In a regression setting, it consists in adding layers between $\boldsymbol{Y_i}$ (the **output layer**) and the $\boldsymbol{X_i}$’s (which are the nodes of the **input layer**). The layers in between are the **hidden layers**
- Thus we regress $\boldsymbol{Y_i}$ on **layers** of the $\boldsymbol{X_i}$’s (i.e. on the hidden layers), not just the $\boldsymbol{X_i}$’s

Neural networks

- Assume we have $\boldsymbol{p}$ different covariates
- Define a **linear combinations** of the $\boldsymbol{X_i}$'s called $\boldsymbol{Z_i}$ as follows:

$$\boldsymbol{Z_i \equiv \sum_{j=1}^{p} \alpha_j X_{i,j}}$$

- Now consider $\boldsymbol{p_1}$ such combinations (so there are $\boldsymbol{p_1}$ different $\boldsymbol{Z_i}$, each being a different linear combination of the $\boldsymbol{X_i}$'s) and apply a **nonlinear transformation** $\boldsymbol{g()}$ to each of them
(e.g. $\boldsymbol{g(Z_{i,k}) = \frac{1}{1+\exp(-Z_{i,k})}}$, $\boldsymbol{k = 1, ..., p_1}$)
- Estimate the following model, i.e. estimate $\boldsymbol{\alpha_{j,k}}$ and $\boldsymbol{\beta_k}$ in

$$\boldsymbol{Y_i = \sum_{k=1}^{p_1} \beta_k \, g\left(Z_{i,k}\right) + u_i}$$

Neural networks

- Estimate the following model, i.e. estimate $\boldsymbol{\alpha_{j,k}}$ and $\boldsymbol{\beta_k}$ in

$$\begin{aligned} \boldsymbol{Y_i} &= \sum_{\boldsymbol{k=1}}^{\boldsymbol{p_1}} \boldsymbol{\beta_k} \boldsymbol{g}\left(\boldsymbol{Z_{i,k}}\right) + \boldsymbol{\varepsilon_i} \\ &= \sum_{\boldsymbol{k=1}}^{\boldsymbol{p_1}} \boldsymbol{\beta_k} \boldsymbol{g}\left(\sum_{\boldsymbol{j=1}}^{\boldsymbol{p}} \boldsymbol{\alpha_{j,k}} \boldsymbol{X_{i,j}}\right) + \boldsymbol{u_i} \end{aligned}$$

- The model is **nonlinear** in the parameters, **nonlinear least squares** will estimate them
- This is called a **neural network with a single hidden layer with p_1 nodes**

Neural networks: Example

- Imagine there are **2** $\boldsymbol{X}$'s, and we make **1** hidden layer with **3** nodes
- We get

$$\begin{aligned}
\boldsymbol{Z_{i,1}} &= \boldsymbol{\alpha_{1,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{1,2}} \boldsymbol{X_{i,2}} \\
\boldsymbol{Z_{i,2}} &= \boldsymbol{\alpha_{2,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{2,2}} \boldsymbol{X_{i,2}} \\
\boldsymbol{Z_{i,3}} &= \boldsymbol{\alpha_{3,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{3,2}} \boldsymbol{X_{i,2}}
\end{aligned}$$

- So we estimate

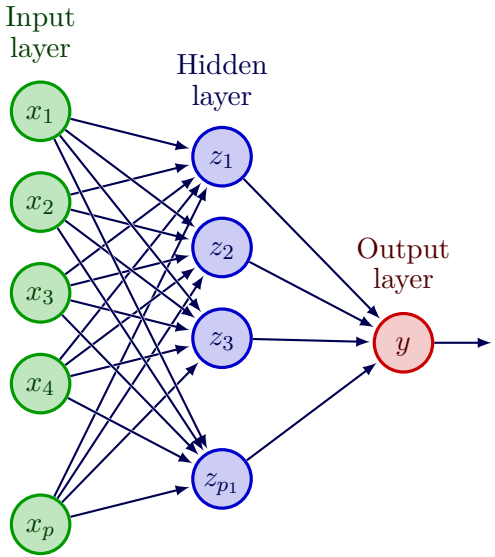
$$\begin{aligned}
\boldsymbol{Y_i} = & \boldsymbol{\beta_1} \boldsymbol{g} \left(\boldsymbol{\alpha_{1,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{1,2}} \boldsymbol{X_{i,2}} \right) \\
& + \boldsymbol{\beta_2} \boldsymbol{g} \left(\boldsymbol{\alpha_{2,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{2,2}} \boldsymbol{X_{i,2}} \right) \\
& + \boldsymbol{\beta_3} \boldsymbol{g} \left(\boldsymbol{\alpha_{3,1}} \boldsymbol{X_{i,1}} + \boldsymbol{\alpha_{3,2}} \boldsymbol{X_{i,2}} \right) \\
& + \boldsymbol{u_i}
\end{aligned}$$

Neural networks: Example

$$
\begin{aligned}
\boldsymbol{Y_i} = & \boldsymbol{\beta_1 g\left(\alpha_{1,1} X_{i,1} + \alpha_{1,2} X_{i,2}\right)} \\
& \boldsymbol{+ \beta_2 g\left(\alpha_{2,1} X_{i,1} + \alpha_{2,2} X_{i,2}\right)} \\
& \boldsymbol{+ \beta_3 g\left(\alpha_{3,1} X_{i,1} + \alpha_{3,2} X_{i,2}\right)} \\
& \boldsymbol{+ u_i}
\end{aligned}
$$

- 2 variables and 1 layer with 3 nodes makes $\boldsymbol{2(X's) \times 3(Z's) + 3(\beta \text{ per } Z) = 9}$ parameters to estimate
- It is common to add multiple hidden layers (so we would create $\boldsymbol{W_i}$ as a linear combinations of the $\boldsymbol{Z_i}$'s), thus increasing the **depth** of the neural network (and its flexibility)
- Say we add a second layer with 4 nodes, it makes $\boldsymbol{2(X's) \times 3(Z's) \times 4(W's) + 4(\beta \text{ per } W) = 28}$ parameters to estimate
- More layers increase the flexibility, hence producing better predictions

Neural networks



Neural networks: Takeaways

- Neural networks are used extensively in the machine learning literature due to their flexibility and predictive power
- Facial recognition and picture identification (Is it a bird? A plane? No wait! It is . . .)
- See **here** and **there**
- Fancier versions are coming out on a regular basis, but so do easy implementations on statistical software across the board

Conclusion

- We barely scratched the surface!
- A lot of research has improved the methods exposed in this lecture
- But the gist stays the same overall:
  - Machine learning methods are very suitable for **prediction problems**, **supervised** or **unsupervised**
  - **Overfitting** is one of the most important problems to deal with
  - Machine learning algorithms make use of **data driven** procedures (validation sets, cross validation) to fine tune the options in supervised problems (parameters for pruning, depth of neural networks)
  - For unsupervised problems, tuning of the parameter is less straightforward
- Many alternatives, it is always good to check more than 1 for robustness of your analysis (and model averaging?)
- Machine learning algorithms are the base of AI: Program an "error" function to minimize, and improve with experience (i.e. as data come in)