Data Science(Prediction)

Presented By:

Sri Krishnamurthy, CFA, CAP

www.QuantUniversity.com

sri@quantuniversity.com

Tree-based Regression

- ✓ Prediction via stratification of the feature space
- ✓ Tree pruning
- ✓ Example using R
- ✓ Example using python

Regression trees

- Prediction via stratification of the feature space
  - Roughly speaking, there are two steps of building a regression tree.

1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \ldots, X_p$—into *J* distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Regression trees

- Prediction via stratification of the feature space
  - In theory, the regions $R_1, R_2, \ldots, R_J$ could have any shape.
  - However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
  - The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS, given by

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the *j*th box.

Regression trees

- Prediction via stratification of the feature space
  - Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into *J* boxes.
  - For this reason, we take a *top-down*, *greedy* approach that is known as *recursive binary splitting*.
  - In order to perform recursive binary splitting, we first select the predictor X_j and the cutpoint *s* such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.

Regression trees

- Prediction via stratification of the feature space
  - In greater detail, for any *j* and *s*, we define the pair of half-planes

$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\}$$

and we seek the value of *j* and *s* that minimize the equation

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

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j,s)$

Regression trees

- Prediction via stratification of the feature space
  - we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
  - we split one of the two previously identified regions. Again, we look to split one of these three regions further, so as to minimize the RSS.
  - The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Regression trees

- Tree pruning
  - The process described above is likely to over fit the data.
  - Therefore, a better strategy is to grow a very large tree T_0, and then *prune* it back in order to obtain a *subtree*.
  - *Cost complexity pruning*—also known as *weakest link pruning*—gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

Example using R

- Here we fit a regression tree to the Boston data set. First, we create a training set, and fit the tree to the training data.

```
> library (tree)
> library (MASS)
> library (ISLR)
> set.seed (1)
> train = sample (1:nrow(Boston), nrow(Boston)/2)
> tree.boston = tree(medv~.,Boston,subset=train)
> summary (tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

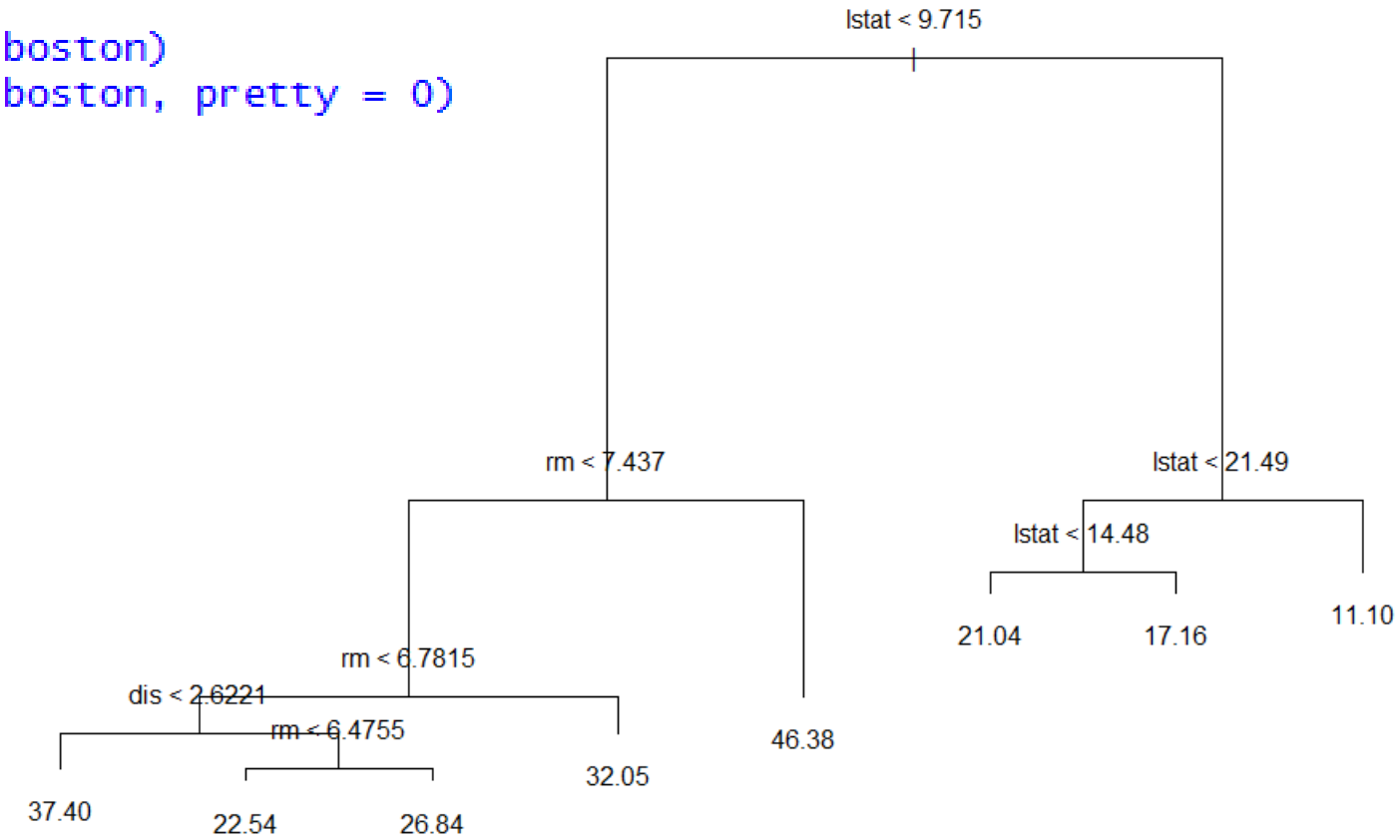
only three of the variables have been used in constructing the tree.

See *Supervised learning-5.R > Regression trees*

Example using R

- We now plot the tree.

```
> plot (tree.boston)
> text (tree.boston, pretty = 0)
```

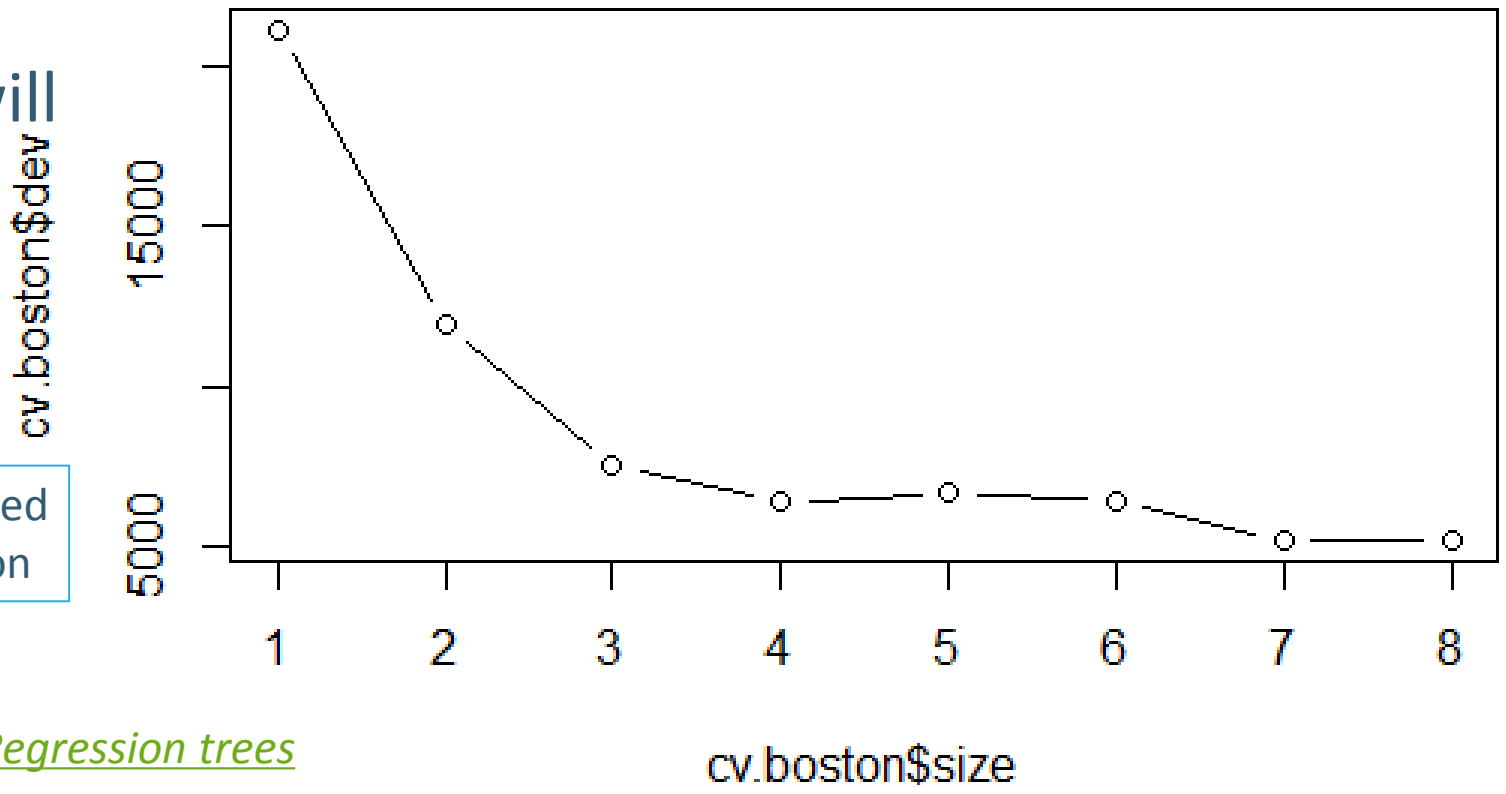


See *Supervised learning-5.R > Regression trees*

Example using R

- Now we use the *Cross-validation for Choosing Tree Complexity* function: **cv.tree()** to see whether pruning the tree will improve performance.

```
> cv.boston = cv.tree (tree.boston)
> plot (cv.boston$size, cv.boston$dev, type='b')
```



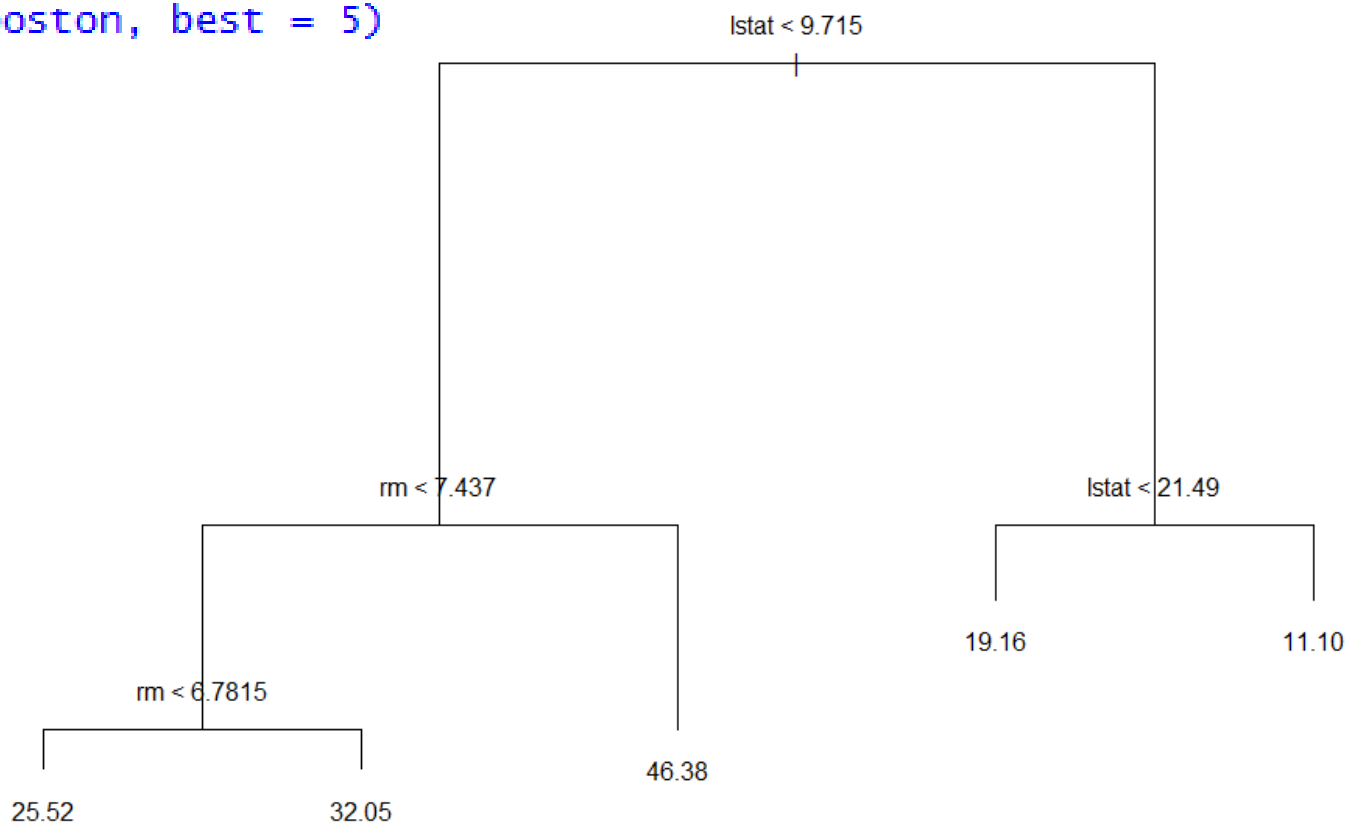
Deviance produced by cross validation

See *Supervised learning-5.R > Regression trees*

Example using R

- if we wish to prune the tree, we could use the **prune.tree()** function:

```
> prune.boston =prune.tree(tree.boston, best = 5)
> plot(prune.boston)
> text(prune.boston, pretty = 0)
```

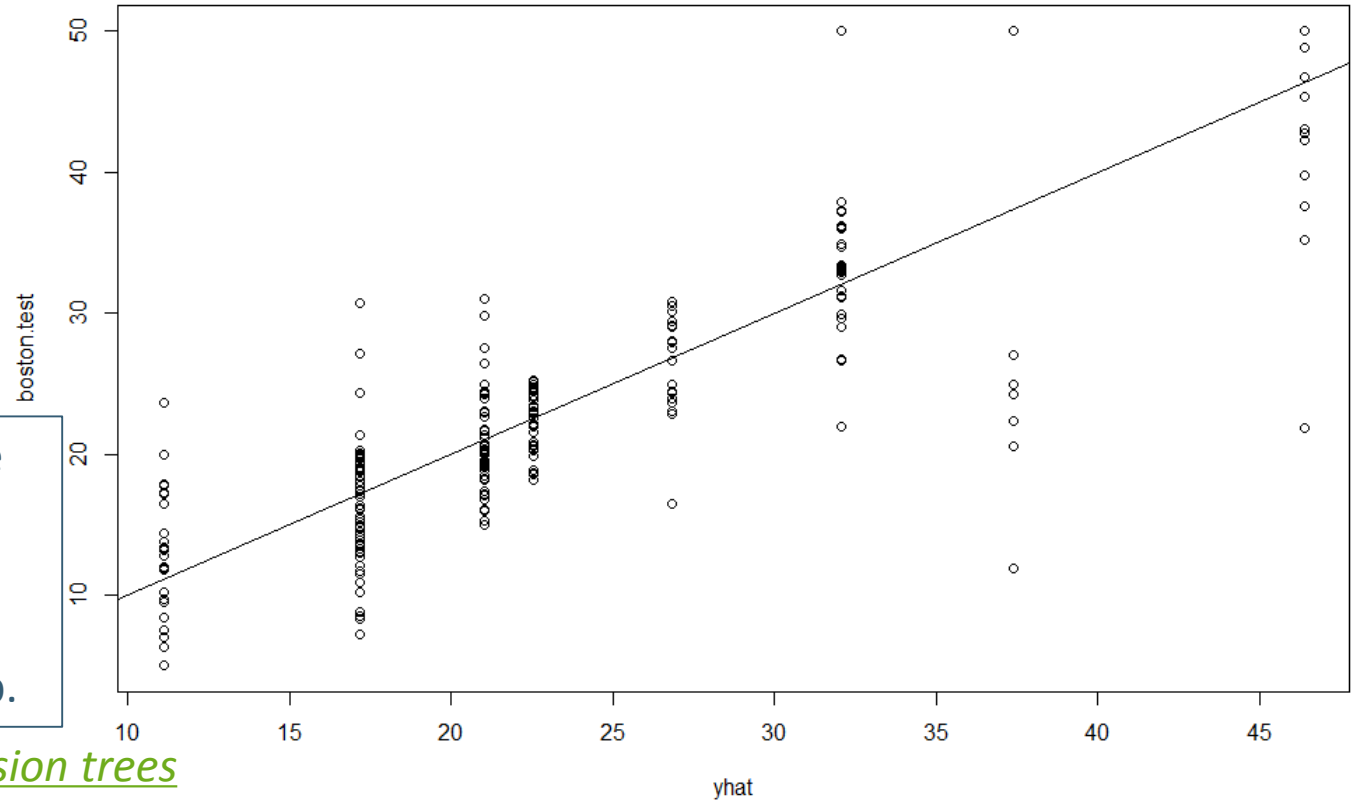


See *Supervised learning-5.R > Regression trees*

Example using R

- In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

```
> yhat=predict (tree.boston, newdata =Boston [-train,])
> boston.test=Boston [-train,"medv"]
> plot(yhat,boston.test)
> abline (0,1)
> mean((yhat -boston.test)^2)
[1] 25.04559
```



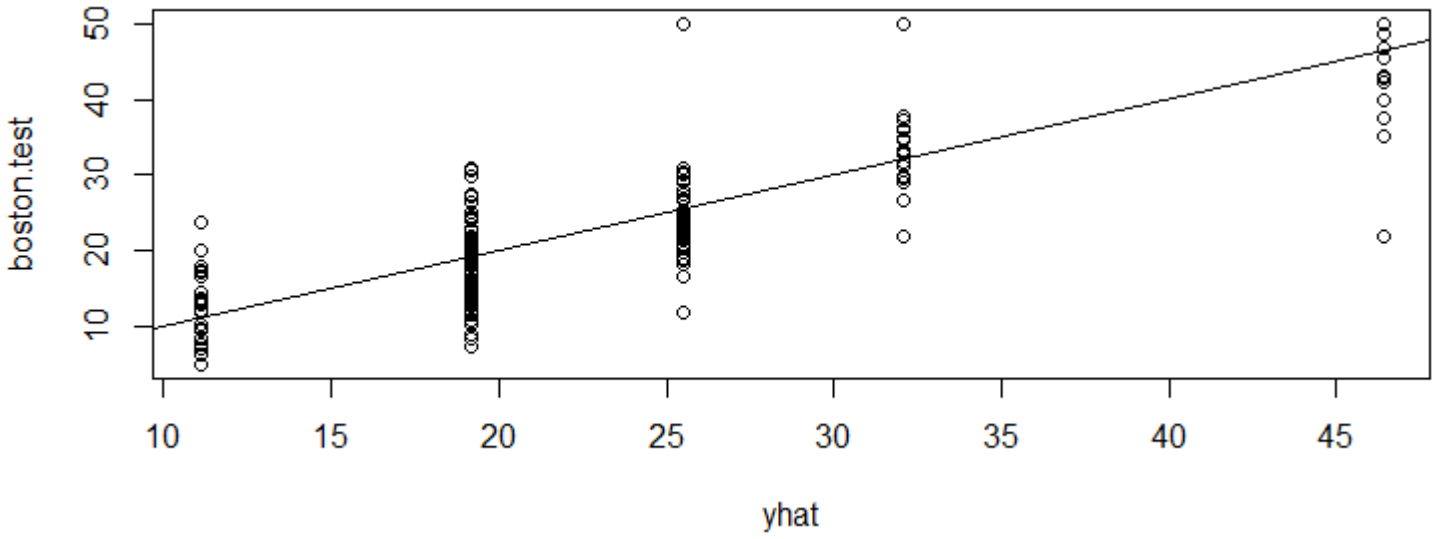
MSE is 25.05, therefore the square root of the MSE is around 5.005, means the test predictions are within around $5, 005 of the true median home value for the suburb.

See *Supervised learning-5.R > Regression trees*

Example using R

- We can also see the prediction given by the pruned tree:

```
> yhat=predict (prune.boston, newdata =Boston [-train,])
> boston.test=Boston [-train,"medv"]
> plot(yhat,boston.test)
> abline (0,1)
> mean((yhat -boston.test)^2)
[1] 26.83413
```



See *Supervised learning-5.R > Regression trees*

After removing the least important splits, the MSE only increases a little

Example using python

In [1]:
```
# Import the necessary modules and libraries
import numpy as np
from sklearn import datasets
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
from pandas import Series, DataFrame
import pandas as pd
```

In [2]:
```
# Import the boston dataset
boston=datasets.load_boston()
frame=DataFrame(boston.data, columns=[boston.feature_names])
frame['MEDV']=boston.target
frame2=frame.sort(columns='LSTAT')
data = np.array(frame2)
X = data[:,12:13]
y = np.array(frame2['MEDV'])
```

See *Supervised learning-6.ipynb > Tree regression using python*

Example using python

In [3]:
```
# Fit regression model
clf_1 = DecisionTreeRegressor(max_depth=2)
clf_2 = DecisionTreeRegressor(max_depth=5)
clf_1.fit(X, y)
clf_2.fit(X, y)
```

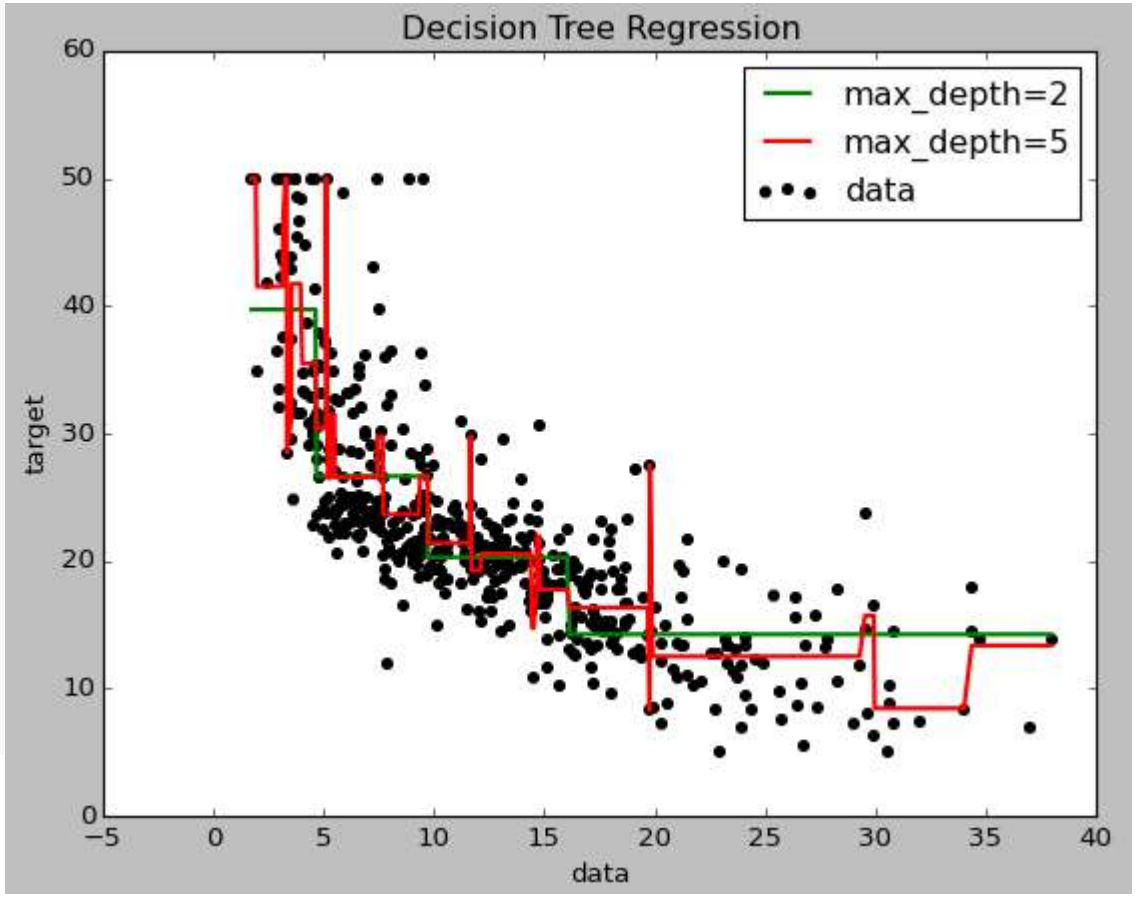
Out[3]:
```
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best')
```

In [4]:
```
# Predict
y_1 = clf_1.predict(X)
y_2 = clf_2.predict(X)
```

See *Supervised learning-6.ipynb > Tree regression using python*

Example using python

```
In [*]: # Plot the results
plt.figure()
plt.scatter(X, y, c="k",
            label="data")
plt.plot(X, y_1, c="g",
         label="max_depth=2",
         linewidth=2)
plt.plot(X, y_2, c="r",
         label="max_depth=5",
         linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```



See *Supervised learning-6.ipynb > Tree regression using python*

Artificial Neural Network

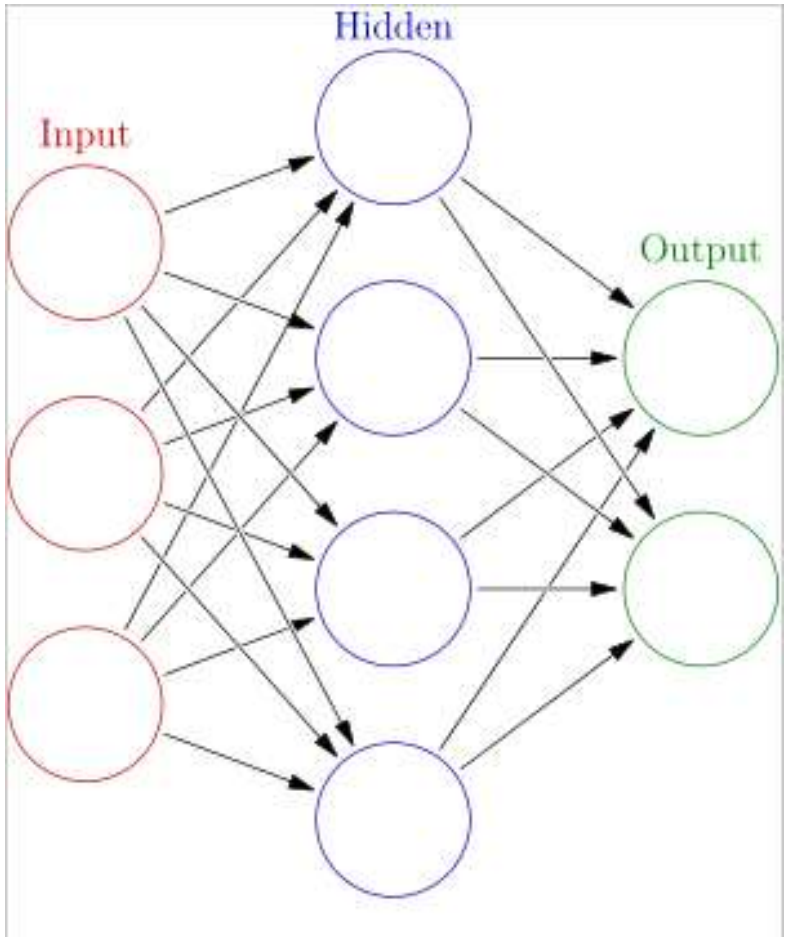
- ✓ Background of Neural Network
- ✓ Example using R
- ✓ Example using python

Background of Neural Network

- **Artificial neural networks** (**ANNs**) are a family of statistical learning models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown.
- Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other.

Background of Neural Network

- An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.



Neural network using R

```
> library (MASS)
> library (grid)
> library (neuralnet)
>
> #Generate 50 random numbers uniformly distributed between 0 and 100
> #And store them as a dataframe
> traininginput <-  as.data.frame(runif(50, min=0, max=100))
> trainingoutput <- sqrt(traininginput)
>
> #Column bind the data into one variable
> trainingdata <- cbind(traininginput,trainingoutput)
> colnames(trainingdata) <- c("Input","Output")
>
> #Train the neural network
> #Going to have 2 hidden layers, each layer has 4 neutrons
> #Threshold is a numeric value specifying the threshold for the partial
> #derivatives of the error function as stopping criteria.
> net.sqrt <- neuralnet(Output~Input,trainingdata, hidden=c(4,4), thresh
old=0.01)
```

See *Supervised learning-7.R > Neural network using R*

Set number of neutrons for each hidden layer

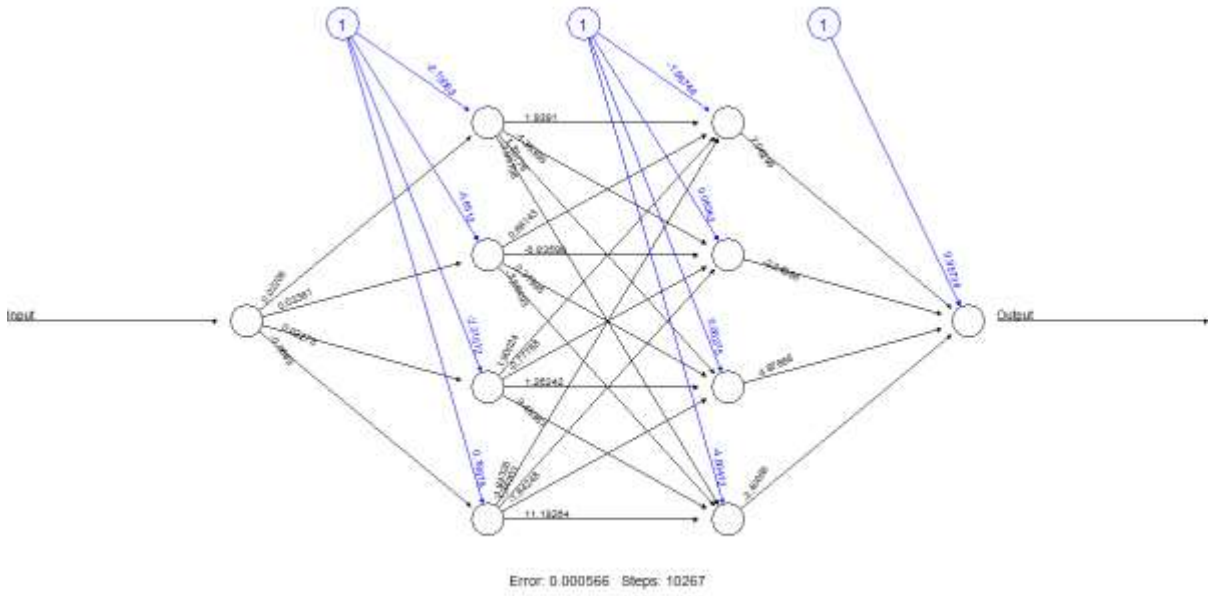
Neural network using R

```
> print(net.sqrt)
Call: neuralnet(formula = Output ~ Input, data = trainingdata, hidden =
c(4,     4), threshold = 0.01)

1 repetition was calculated.

            Error Reached Threshold Steps
1 0.0005660667653    0.009932495116 10267

>
> #Plot the neural network
> plot(net.sqrt)
```



See *Supervised learning-7.R > Neural network using R*

Neural network using R

```
> #Test the neural network on some training data
> testdata <- as.data.frame((1:10)^2) #Generate some squared numbers
> net.results <- compute(net.sqrt, testdata) #Run them through the neura
l network
>
> #Lets see what properties net.sqrt has
> ls(net.results)
[1] "net.result" "neurons"
>
> #Lets see the results
> print(net.results$net.result)
              [,1]
 [1,] 1.329957121
 [2,] 2.018365650
 [3,] 3.004037487
 [4,] 4.002608378
 [5,] 4.996541607
 [6,] 6.005105579
 [7,] 6.999796805
 [8,] 7.994645516
 [9,] 9.004832546
[10,] 9.978830807
```

See *Supervised learning-7.R > Neural network using R*

Neural network using python

```
In [1]: from pybrain.tools.shortcuts import buildNetwork

In [2]: net = buildNetwork(1, 4, 4, 1)

In [3]: net.activate([1])
Out[3]: array([ 0.41121249])

In [4]: from pybrain.datasets import SupervisedDataSet

In [5]: ds = SupervisedDataSet(1, 1)
```

Set number of neutrons for input layer, hidden layers and output layer

See *Supervised learning-8.ipynb > Neural network using python*

Neural network using python

```
In [6]: import numpy as np

In [7]: s = np.random.uniform(size=50)

In [8]: trainingoutput = np.array(10*s)

In [9]: traininginput = trainingoutput*trainingoutput

In [10]: trainingdata = np.array([traininginput,trainingoutput])

In [11]: trainingdata = trainingdata.T
```

See *Supervised learning-8.ipynb > Neural network using python*

Neural network using python

```
In [11]: trainingdata = trainingdata.T

In [12]: from pybrain.supervised.trainers import BackpropTrainer

In [13]: for i in range(50):
             ds.addSample((trainingdata[i,0]), (trainingdata[i,1]))

In [14]: len(ds)
Out[14]: 50

In [15]: trainer = BackpropTrainer(net, ds)

In [16]: trainer.train()
Out[16]: 6.498825013213471 3
```

See *Supervised learning-8.ipynb > Neural network using python*

Neural network using python

```
In [18]: print net
```

```
FeedForwardNetwork-11
   Modules:
    [<BiasUnit 'bias'>, <LinearLayer 'in'>, <SigmoidL
ayer 'hidden0'>, <SigmoidLayer 'hidden1'>, <LinearLay
er 'out'>]
   Connections:
    [<FullConnection 'FullConnection-10': 'in' -> 'hi
dden0'>, <FullConnection 'FullConnection-5': 'hidden
1' -> 'out'>, <FullConnection 'FullConnection-6': 'hi
dden0' -> 'hidden1'>, <FullConnection 'FullConnectio
n-7': 'bias' -> 'out'>, <FullConnection 'FullConnecti
on-8': 'bias' -> 'hidden0'>, <FullConnection 'FullCon
nection-9': 'bias' -> 'hidden1'>]
```

See *Supervised learning-8.ipynb > Neural network using python*

Neural network using python

- Here, we can test the network using **.activate.**

```
In [126]: net.activate([1])
Out[126]: array([ 1.27486969])

In [127]: net.activate([4])
Out[127]: array([ 2.29150977])

In [128]: net.activate([9])
Out[128]: array([ 3.60072216])

In [129]: net.activate([16])
Out[129]: array([ 4.29900189])

In [130]: net.activate([25])
Out[130]: array([ 4.88355796])
```

See *Supervised learning-8.ipynb > Neural network using python*

Summary

We have covered	Key functionality
Tree-based regression	✓ Two steps of building a regression tree. ✓ Prune the regression tree to create feasible prediction ✓ How to use tree-based regression in R and python.
Artificial Neural Network	✓ Background of Artificial Neural Network ✓ How the artificial neural network works ✓ How to use artificial neural network in R and python

Reference

- An Introduction to Statistical Learning. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani.
- brms: An R Package for Bayesian Generalized Linear Mixed Models using Stan. Paul-Christian Burkner.
- scikit-learn.org
- pybrain.org

Q&A

QuantUniversity, LLC

www.quantuniversity.com

Thank you!

Contact

Sri Krishnamurthy, CFA, CAP
Founder and CEO
QuantUniversity LLC.

LinkedIn srikrishnamurthy

www.QuantUniversity.com