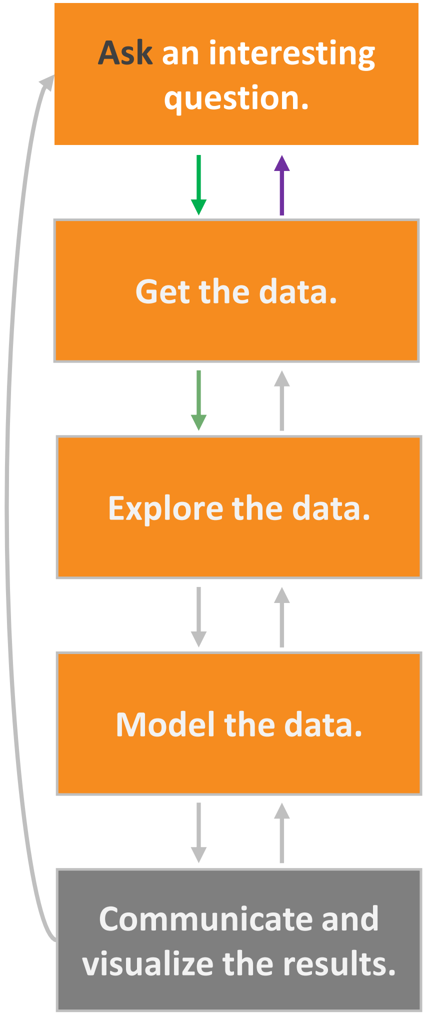
Data Analysis I: Simple Model Application

Introduction to Data Science

efl Data Science Course

Micha Bender

Step 4 – Model the data



How do we compute the interest rate automatically from a given datapoint?
What is the data we need to train a debt-class-classifier?

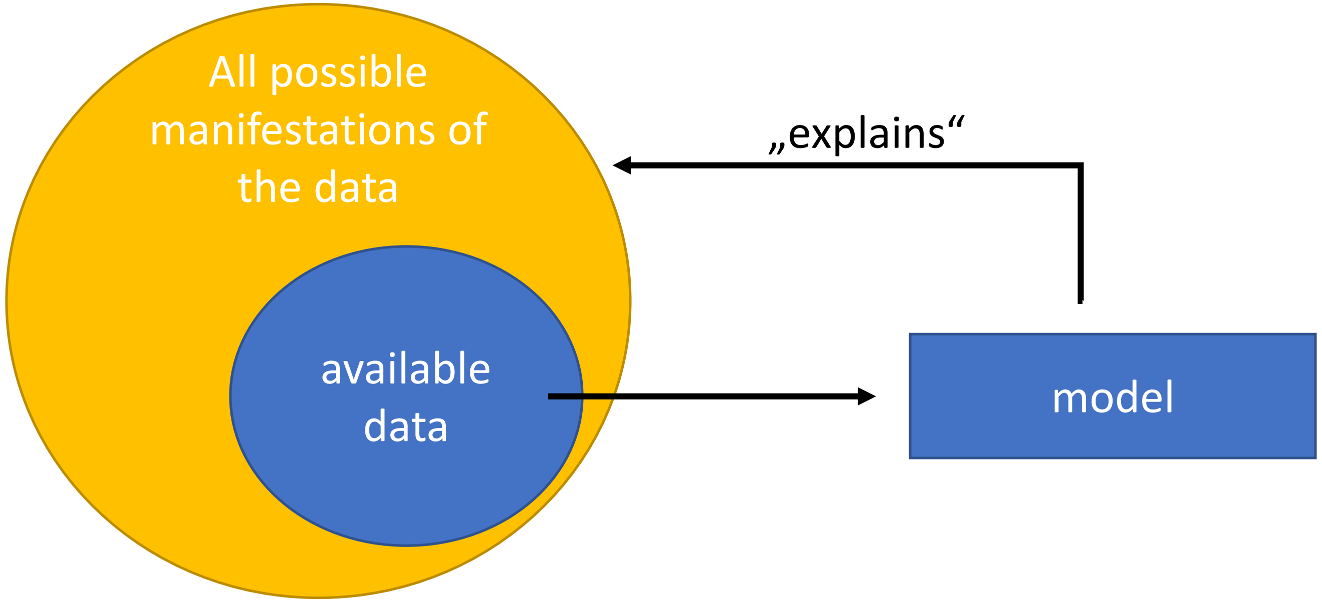
We got the data from our firm.
No privacy problems.
Let's go and explore!

Plot the data.
Are there anomalies?
Are there patterns?

Build a model.
Fit the model.
Validate the model.

What did we **learn**?
Do the results make **sense**?
Can we tell a **story**?

What is a model?



How do we model data / learn from data?

Example machine learning: supervised learning vs. unsupervised learning

Unsupervised Learning: The data has no target attribute

- We want to explore the data to find some intrinsic structures in it

Supervised Learning: Discover patterns in the data that relate data attributes with a target (class) attribute

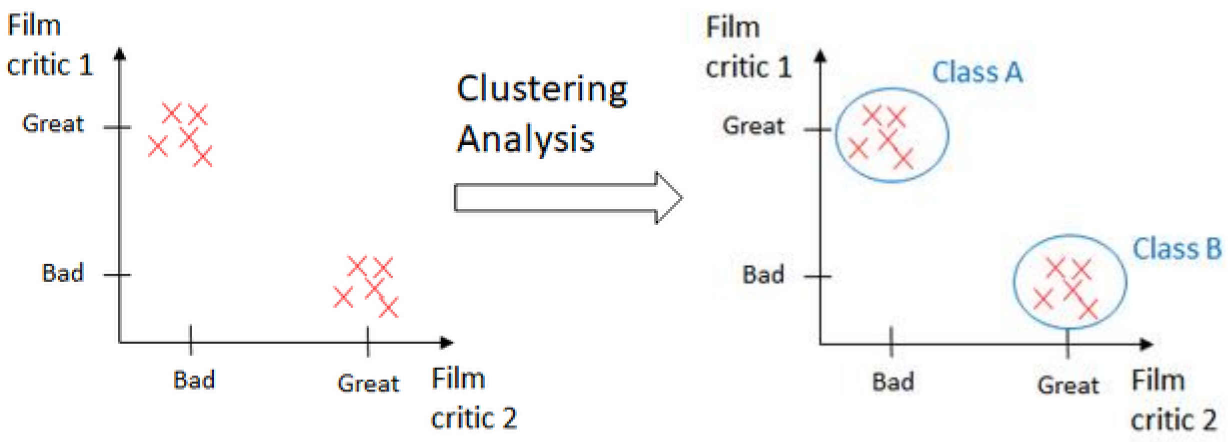
- These patterns are then utilized to predict the values of the target attribute in future data instances

Unsupervised Learning Example: Clustering

- No target value
- **Goal:** Identify associations, i.e. grouping data (to previously unknown classes)
- **Applications**:
  - Anomaly detection
  - Recommending systems
  - Documents grouping
  - Finding customers with common interests

Example

- Imagine we have a dataset of movies and want to classify them. We have the following reviews of films.



- The machine learning model will infer that there are two different classes without knowing anything else from the data

Source: https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e

Supervised Learning Example 1: Classification

- Target value is **categorical** (a set of known classes), e.g. the genre of a video game
- **Goal**: Train a model that detects patterns in the data in order to assign observations to one of theses classes

Example

- Find the genre of a game based on its name

Action
Sports
Misc
Role-Playing
Shooter
Adventure
Racing
Platform
Simulation
Fighting
Strategy
Puzzle

Genres

Name	Platform	Year	Genre
Wii Sports	Wii	2006	Sports
Super Mario Bros.	NES	1985	Platform
Mario Kart Wii	Wii	2008	Racing
Wii Sports Resort	Wii	2009	Sports
Pokemon Red/Pokemon Blue	GB	1996	Role-Playing
Tetris	GB	1989	Puzzle
New Super Mario Bros.	DS	2006	Platform
Wii Play	Wii	2006	Misc
New Super Mario Bros. Wii	Wii	2009	Platform
Duck Hunt	NES	1984	Shooter
Nintendogs	DS	2005	Simulation
Mario Kart DS	DS	2005	Racing

Games

→ Games having the word „Kart“ in their name are highly likely to be in the „Racing“ genre

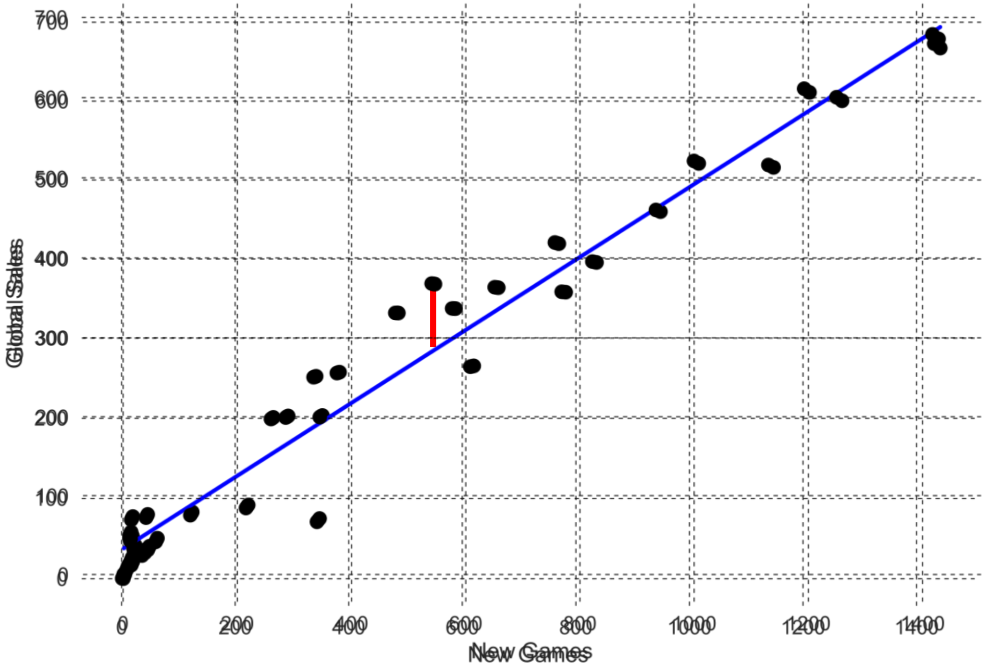
Supervised Learning Example 2: Regression

- Target value is **continuous**,
  e.g. Global Sales in $

- **Goal**: Find a function of the explaining variables that minimize the sum of squared errors

- In a two-dimensional problem the function has the form
  - $y_i = ß_0 + ß_1 \mathrm{x_i} + \varepsilon_i$
  - $\min \sum \left[y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right) \right]^2$

    where $\varepsilon_i = y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i \right)$

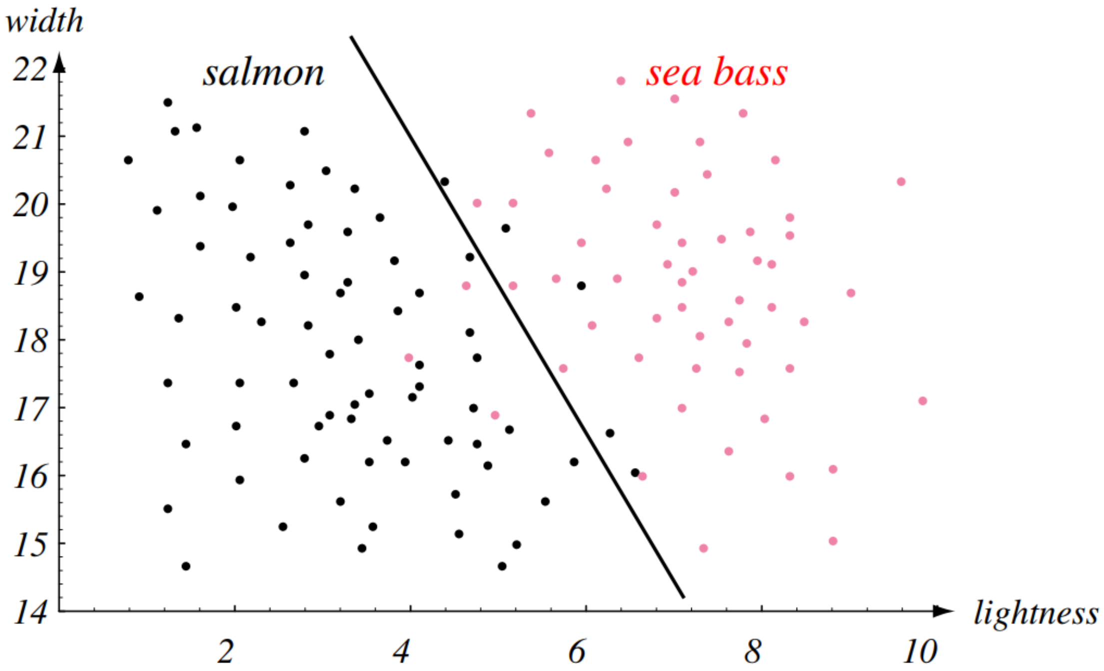


Example

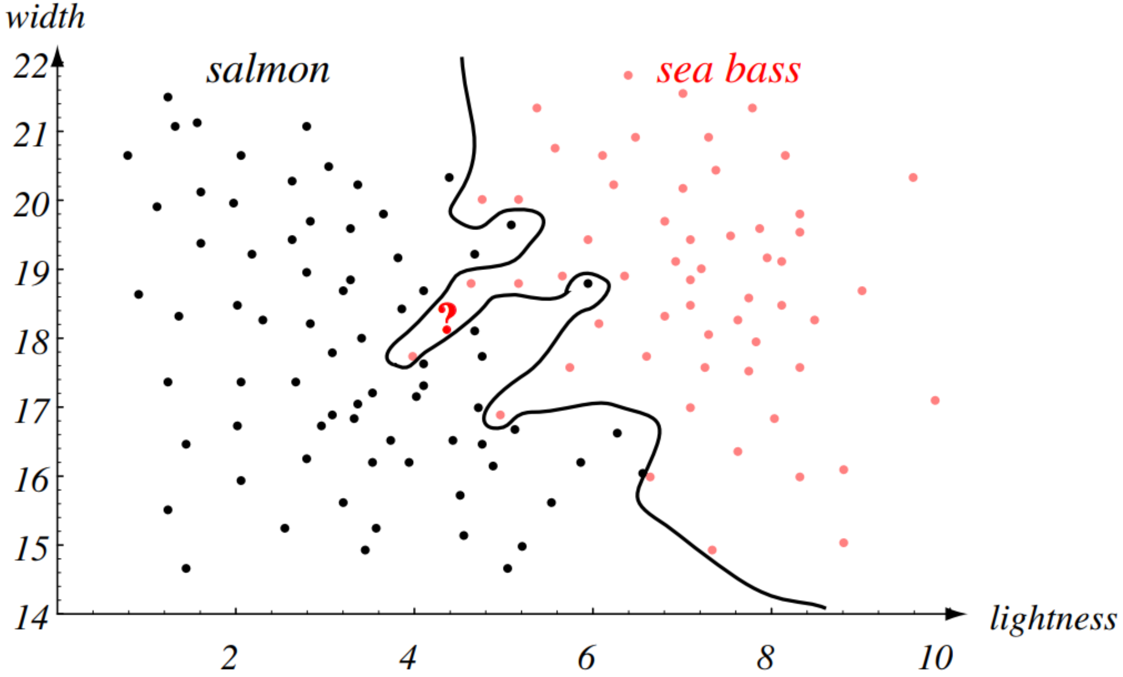
- Explain the global video game sales in a year with the number of new game releases in a year

$$\widehat{GlobalSales}_i = \hat{\beta}_0 + \hat{\beta}_1 NewGames_i$$

Problems with modelling: Overfitting



Linear classification



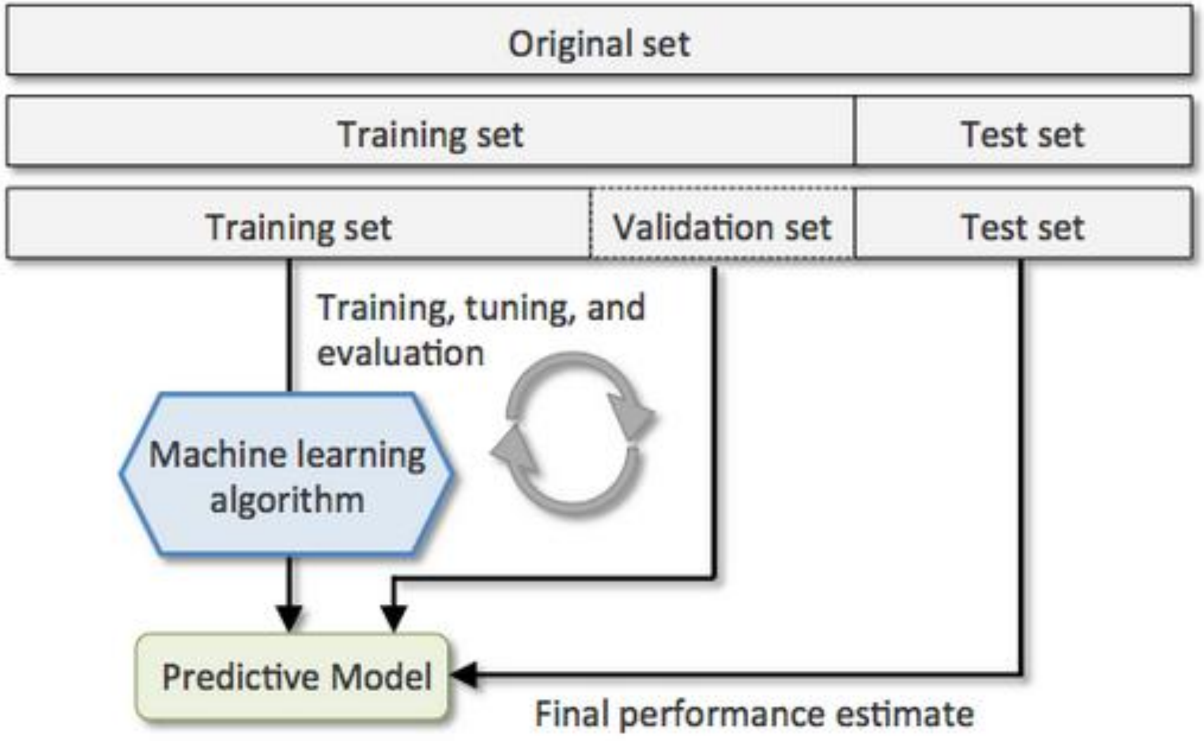
Overfitted model

Overfitting? So what?! my model performs good on my data!

- Yet, the produced model corresponds too closely to the training set and may therefore fail to fit unseen data or predict future observations reliably → We want a generalizing model!

Source: J. M. Buhmann – Lectures on Machine Learning

How to overcome problems of overfitting?



1. Train: we train the model
2. Validation: we validate and adjust model parameters
3. Test: unseen data. We get an unbiased final estimate

Source: https://stats.stackexchange.com/questions/410118/cross-validation-vs-train-validation-test

Recap and what this session is about

- You learnt how to read, preprocess, and describe data using visualizations and summary statistics

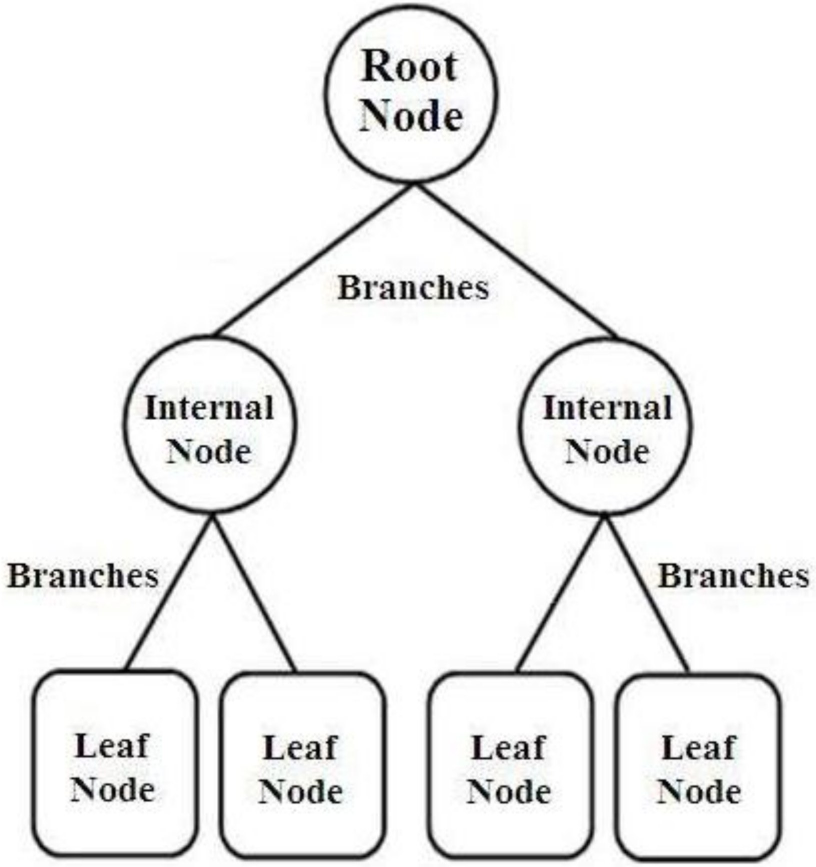
This session: First simple model application

- **Decision Trees** are popular non-parametric **supervised** learning techniques for **regression** and classification problems which are able to handle **complex relations** within data in an **accessible** and **interpretable** way

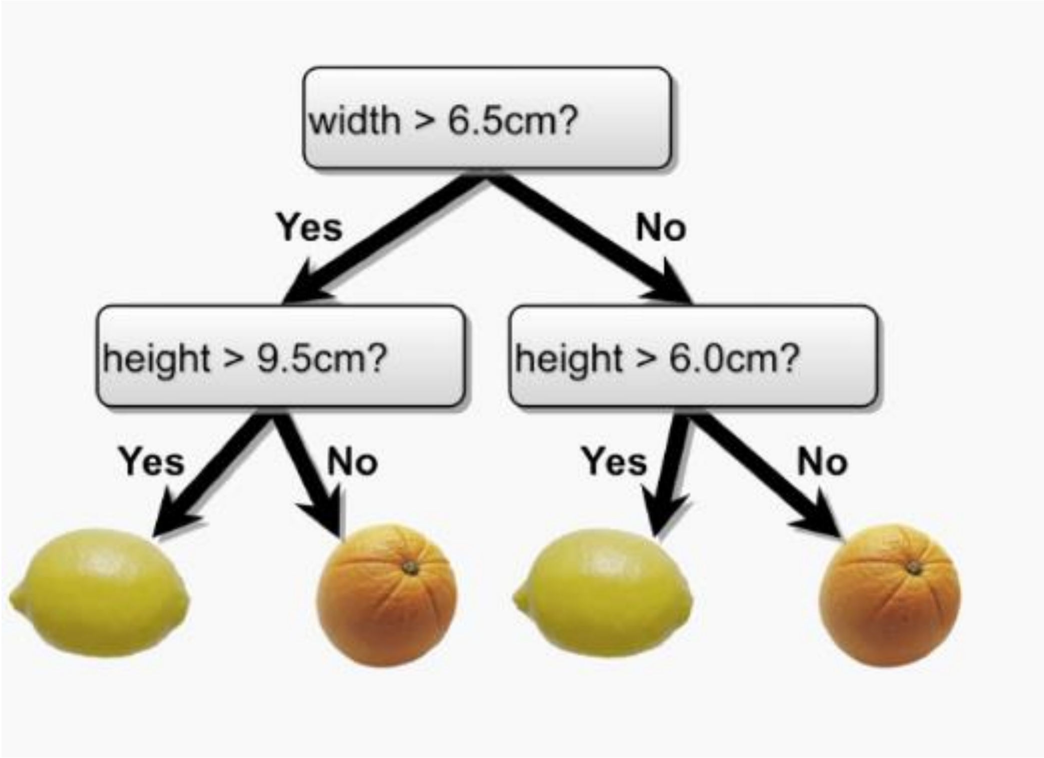
- **Random Forests** overcome problems of Decision Trees with relatively little extra work by growing a set of roughly independent tree models

Table of Contents

What is a decision tree?



General Structure of a Decision Tree



Example of a Classification Tree

Introductory example and problem formulation

Example data set:
- Major League Baseball Data from the 1986 and 1987 seasons
- 322 observations of major league players
- The color indicates the amount of salary (The darker the higher)

Goal:
- Predict the salary of major league players in 1987 based on
  - Number of years in the major leagues
  - Number of hits in 1986

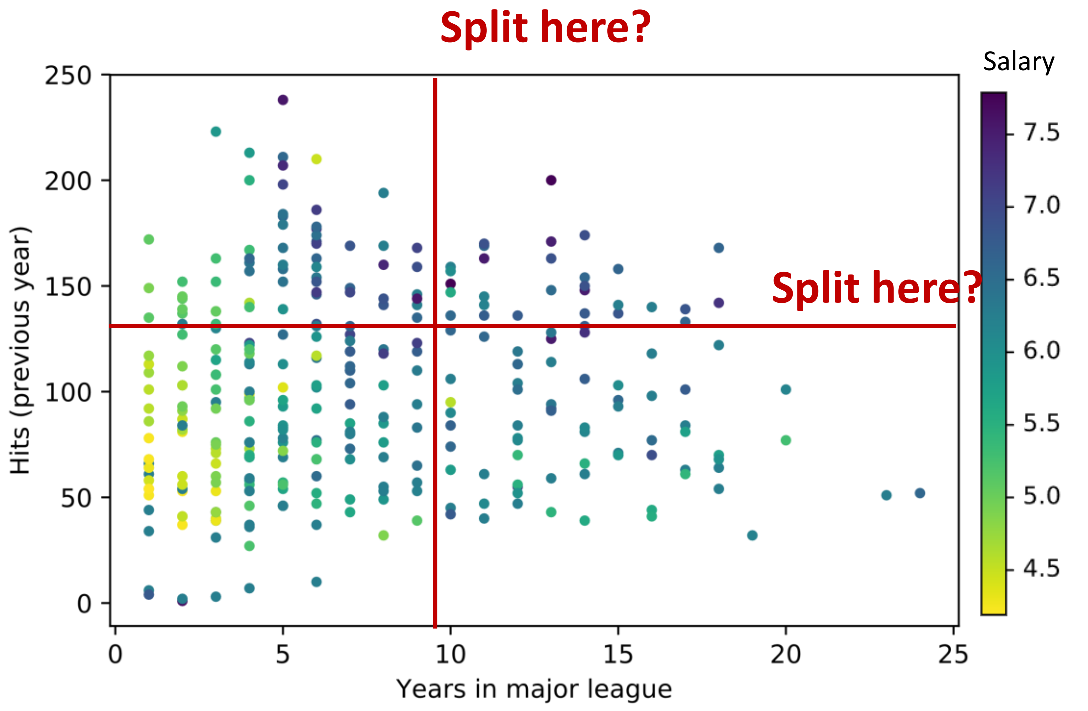
General goal of a Decision Tree:
- Find binary splits in the data
- Minimize overall error

General learning algorithm of a Decision Tree:

1. Start with empty decision tree (undivided feature space)

2. Choose optimal predictor on which to split and choose the optimal threshold value for splitting by applying a **splitting criterion**

3. Recurs on each new node until **stopping condition** (e.g. maximum depth) is met



The concept of the Regression Tree

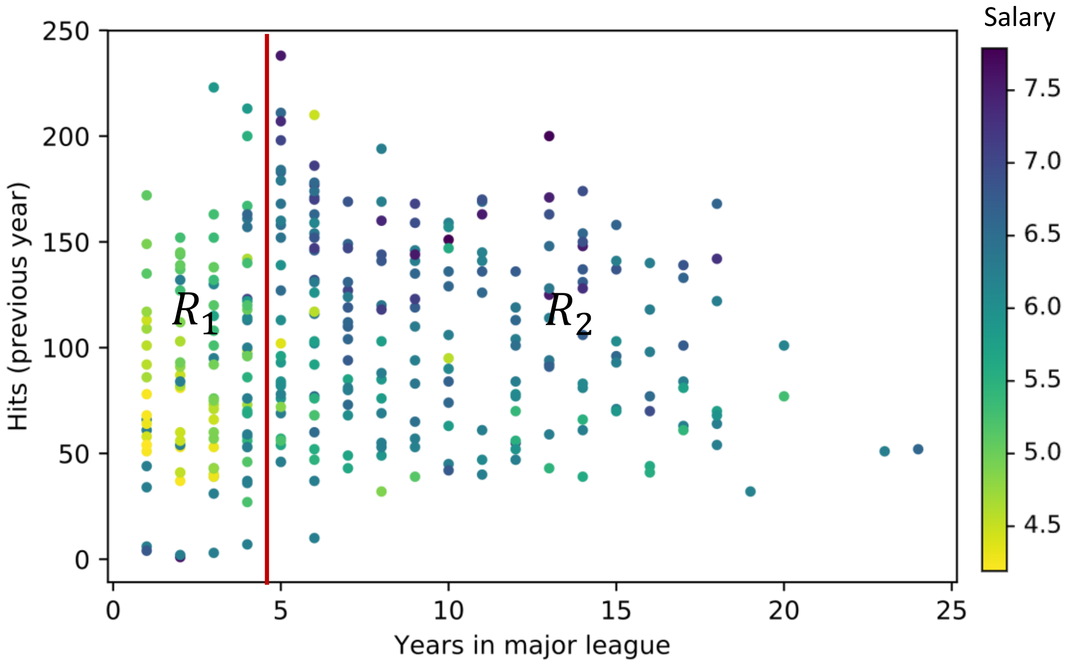
- Splitting criterion shall promote splits that **improve** the **predictive accuracy** of the model → for y in $\mathbb{R}$ use **Mean Squared Error** (MSE)

- Having an output in $\mathbb{R}$, each region in the model should be labeled with a real number – typically the average of the output values of the training points contained in the region

➢ **Objective function** to be **minimized** (MSE):

$$H(Y|X) = \frac{1}{m} \sum_{j=1}^{k} \sum_{i=1}^{m_j} (y_i - \mu_j|_{x_i \in X_j})^2$$

where the outer sum goes over all k data areas $R_j = \{Y_j, X_j\}$ of size m_j, and the inner sum is the Sum of Squared Errors (SSE) for each data area R_j



Learning algorithm of a Regression Tree

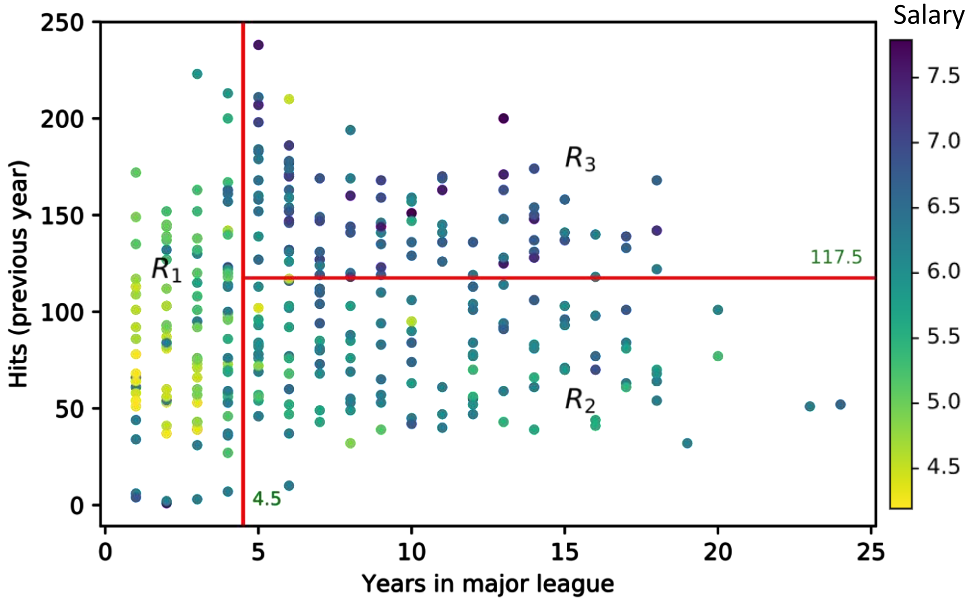
1. Start with empty decision tree (undivided feature space)

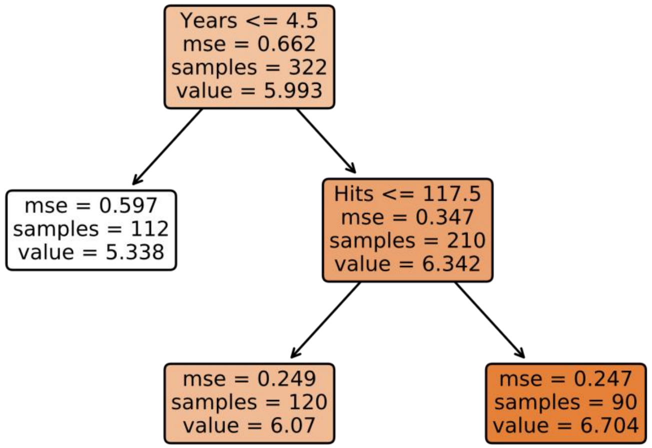
2. Choose a predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j \leq s\}$ and $\{X|X_j > s\}$ leads to the greatest possible reduction in SSE:

   In detail, for any j and s, we define a pair of half-planes
   $R_1(j,s) = \{X|X_j \leq s\}$ and $R_2(j,s) = \{X|X_j > s\}$

   and seek the value of j and s that minimize the equation
   $\sum_{i:\, x_i \in R_1(j,s)} (y_i - \mu_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \mu_{R_2})^2$

3. Recurs on each new node until **stopping condition** is met





Stopping conditions for Regression Trees

Stopping conditions

- Max depth – the maximum depth of the tree
- Accuracy gain - Further splits give less than some minimal amount of extra information
- Further splits would results in nodes containing less than a certain percentage of the total data

Using the Regression Tree for prediction

Prediction of the Regression Tree

For any data point x_i

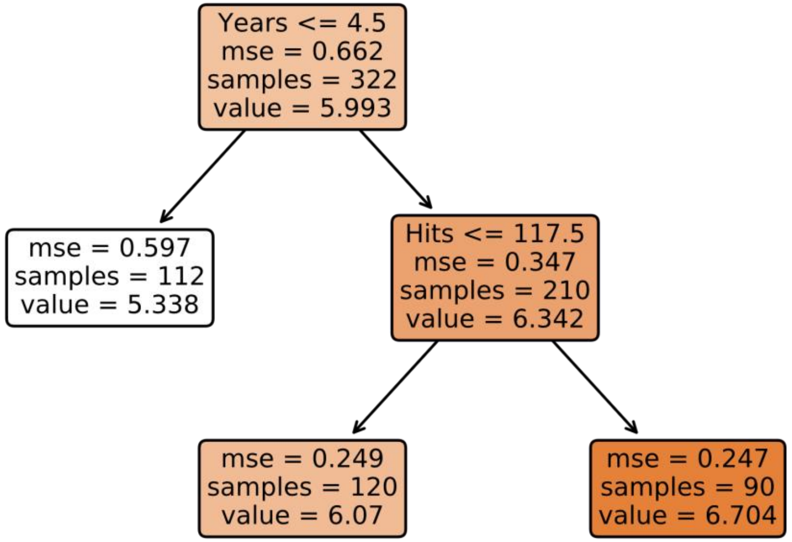
1. Traverse tree until we reach a leaf node

2. Averaged value of the response variable in the leaf (from the training set) is the prediction value $\hat{y}$

Example

The baseball player Steve Johnson has played **6 years** in the major league and had **80 hits** in the last year

→ According to our Regression Tree he has a salary of $\$1000e^{6.07}$ = $432,680.68



DecisionTreeRegressor in Python (not complete)

We use the DecisionTreeRegressor from the Scikit-learn package:

```
from sklearn.tree import DecisionTreeRegressor
```

Parameters for model configuration(we concentrate on two at first):

- criterion: *string,* function to measure the quality of a split, default = `'mse'`
- max_depth: *int,* the maximum depth of the tree, default = None. If None, nodes are expanded until all leaves are pure.

Syntax:

```
regr = DecisionTreeRegressor(max_depth = 2)
```

Methods:

- `fit(self, X, y)`: Build a decision tree regressor from the training set (X, y), X = training input samples, y = target values, returns the trained model

- `predict(self, X)`: Predict regression value for X, X = input samples, returns predicted values based on X

Python example

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Data
X_train # training sample (independent variables)
y_train # training sample (dependent variable)
X_test # testing sample (independent variables)
y_test # testing sample (dependent variable)

# Configure model
regr = DecisionTreeRegressor(max_depth = 2)

# Fit regression model
regr.fit(X_train, y_train)

# Predict
y_pred = regr.predict(X_test)

# evaluation
mse = mean_squared_error(y_test, y_pred)
```

Plotting the Tree

1. **Exporting the tree as text**

Use scikit-learn package export_text:

```
from sklearn.tree import
     export_text
```

plot_tree only requires a trained Tree object

Syntax:
`export_text(decision_tree)`

But you can use the following parameters to configure the output (not complete):

- feature_names: *list of strings,* Names of each of the features

2. **Plotting the tree**

Use scikit-learn package plot_tree:

```
from sklearn.tree import
     plot_tree
```

plot_tree only requires a trained Tree object

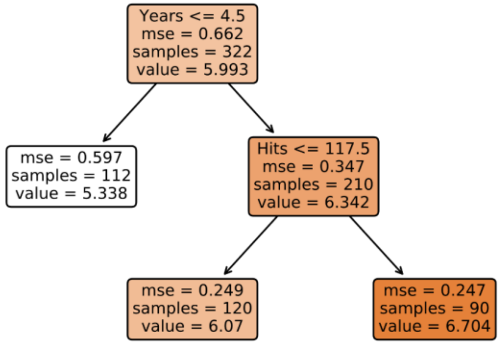
Syntax:
`plot_tree(decision_tree)`

But you can use the following parameters to configure the plot (not complete):

- feature_names: *list of strings,* Names of each of the features
- filled: *bool,* paint nodes to indicate extremity of values for regression
- fontsize: *int,* Size of text font

Python example

```
from sklearn.tree import DecisionTreeRegressor,
            export_text, plot_tree
import matplotlib.pyplot as plt

# export tree to text (String)
sTree = export_text(regr, feature_names =
            list(X_train.columns))

# plot tree and save it to pdf
plt.figure()
plot_tree(regr, filled = True,
feature_names=list(X_train.columns),
            fontsize = 9)
plt.savefig('tree.pdf')
```



Case study: Lending Club
Motivation for using Decision Trees for our data set

Assumption:

- Lending club uses different features for assigning specific interest rates to a customer
- Based on specific thresholds for each feature an interest rate will be assigned
- The decision tree helps us to understand the decision process for assigning interest rates to customers by rebuilding the decision process in an easy-to-interpret tree structure

Exercise 1: Decision Tree Regression

1. Use your new knowledge about Decision Trees to train a tree according to the following specifications:
   1. Select features ‘dti’ and ‘loan_amnt’
   2. Use a maximum tree depth of 2

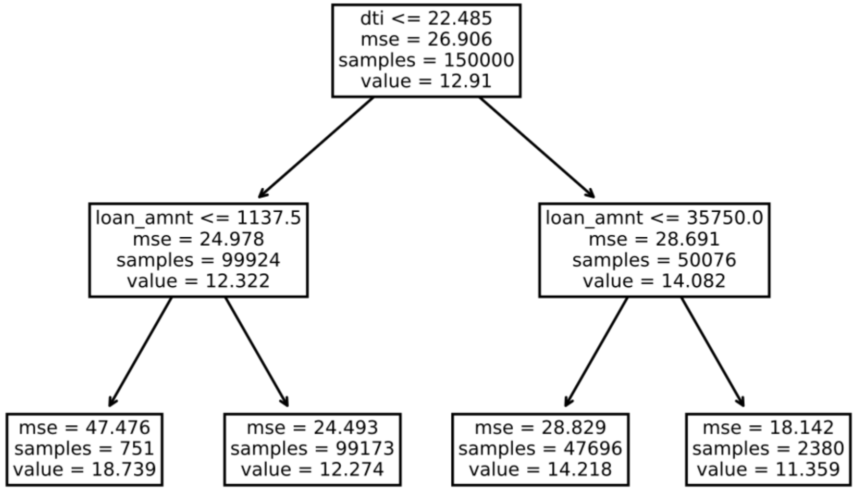
2. Plot the tree using the following specifications
   1. Font size = 7
   2. No fill
   3. Include feature names

3. Evaluate your results and print the Mean Squared Error

Code skeleton

```
# import packages
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# load data set
data = pd.read_csv('dataset_small.csv’)

# fill missing values
data["dti"]=data["dti"].fillna(data["dti"].mean())

# select X and y from data set
y = ...
X = ...

# get random train and test data from data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
             test_size=0.25, random_state=42)

# Configure model
...
# Fit regression model
...
# Predict
...
# evaluation
...
# plot tree
...
```

Exercise 1: Decision Tree Regression - Results



MSE = 25.73871286007019

Code result

```
# import packages
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# load data set
data = pd.read_csv('dataset_small.csv’)

# fill missing values
data["dti"]=data["dti"].fillna(data["dti"].mean())

# select X and y from data set
y = data['int_rate’]
X = data[['dti','loan_amnt’]]

# get random train and test data from data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
             test_size=0.25, random_state=42)

# Configure model
regr = DecisionTreeRegressor(max_depth = 2)
# Fit regression model
regr.fit(X_train, y_train)
# Predict
y_1 = regr.predict(X_test)
# evaluation
mse1 = mean_squared_error(y_test, y_1)
print('DT: mse = '+ str(mse1))
# plot tree
plot_tree(regr, filled = False, feature_names=list(X.columns),
             fontsize = 7)
```

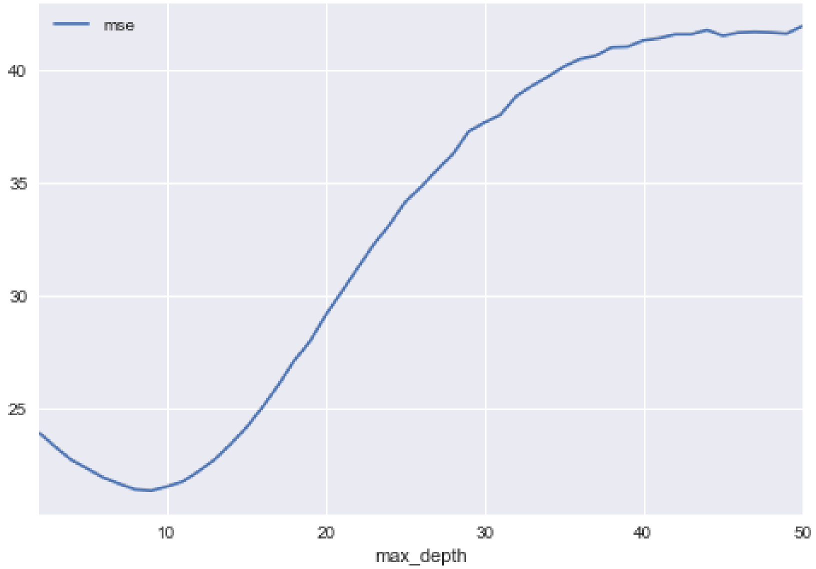
Limitations of Decision Trees

Decision Trees

Advantages:
- easy to understand, easy to interpret, and fast to train

Disadvantages:
- to capture a complex decision boundary, we need to use large trees which are prone to overfitting and therefore underperform compared to other regression methods

➢ **Random Forests** overcome problems of Decision Trees with relatively little extra work



Decision Tree using 12 features from the Lending Club data set

Table of Contents

1. Decision Trees

2. Random Forests

The concept of Random Forests

General idea
Grow a set of roughly independent tree models, which jointly perform better than a single tree model

Algorithm for growing random forests

1. Randomly select a sample X' of size $m' < m$ from X, with about $70\% - 90\%$ of m.[1]

2. Grow a decision tree using X' as described in the Decision Tree section with the difference that:
    1. the split feature at each branching is chosen from a random sample S_r of $n' < n$ features.
    2. S_r is chosen anew at each split point.

    A standard value for $n' = \sqrt{n}$

3. The Regression values are then given by the average prediction of all single trees

[1] Therefore, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set

RandomForestRegressor in Python (not complete)

We use the RandomForestRegressor from the Scikit-learn package:

```
from sklearn.ensemble import RandomForestRegressor
```

Parameters are the same as for the Decision Tree, but we have two new Parameters:

- n_estimators: *int,* number of trees in the forest, default = 100
- max_features: *int, float, or string,* number of features to consider when looking for the best split, default = n, we choose $n' = \sqrt{n}$

Syntax:

```
regr = RandomForestRegressor(max_depth = 2,
       n_estimators = 10, max_features = 'sqrt')
```

Methods (Same as for Decision Tree)**:**

- `fit(self, X, y)`: Build a forest of trees from the training set (X, y), X = training input samples, y = target values, returns the trained model
- `predict(self, X)`: Predict regression value for X, X = input samples, returns predicted values based on X

Python example

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Data
X_train # training sample (independent variables)
y_train # training sample (dependent variable)
X_test # testing sample (independent variables)
y_test # testing sample (dependent variable)

# Configure model
regr = RandomForestRegressor(max_depth = 2 ,
          n_estimators = 10, max_features = 'sqrt')

# Fit regression model
regr.fit(X_train, y_train)

# Predict
y_pred = regr.predict(X_test)

# evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

Exercise 2: Random Forest Regression

1. Use your new knowledge about Random Forests and adjust your code from Exercise 1 to train a Random Forest Regression model
   - Start with 50 trees in the forest and use the other parameters from Exercise 1

2. Use different values for max_depth and n_estimators and try to minimize the MSE

3. Include other features from the data set and try to improve your model

Code skeleton

```
# import packages
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# load data set
data = pd.read_csv('dataset_small.csv')

# fill missing values
data["dti"]=data["dti"].fillna(data["dti"].mean())

# select X and y from data set
y = ...
X = ...

# get random train and test data from data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
             test_size=0.25, random_state=42)

# Configure model
...
# Fit regression model
...
# Predict
...
# evaluation
...
# plot tree
...
```

Exercise 2: Random Forest Regression - Results

the Data Science Institute

Code result

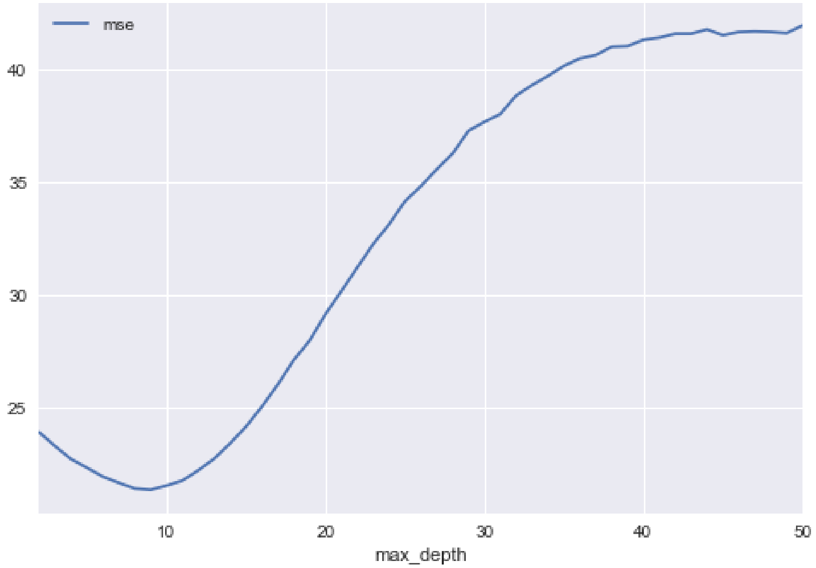
```
# import packages
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# load data set
data = pd.read_csv('dataset_small.csv')

# fill missing values
data["dti"]=data["dti"].fillna(data["dti"].mean())

# select X and y from data set
y = data['int_rate']
X = data[['dti','loan_amnt']]

# get random train and test data from data set
X_train, X_test, y_train, y_test = train_test_split(X, y,
             test_size=0.25, random_state=42)

# Configure model
regr = RandomForestRegressor(max_depth = 2, n_estimators=50,
             max_features='sqrt')

# Fit regression model
regr.fit(X_train, y_train)

# Predict
y_1 = regr.predict(X_test)

# evaluation
mse1 = mean_squared_error(y_test, y_1)
r2_1 = r2_score(y_test, y_1)
print('RF: mse = '+ str(mse1) + ' r2 = '+ str(r2_1))
```

MSE = 25.552141676526368

R^2 = 0.04206088718725787

Outlook

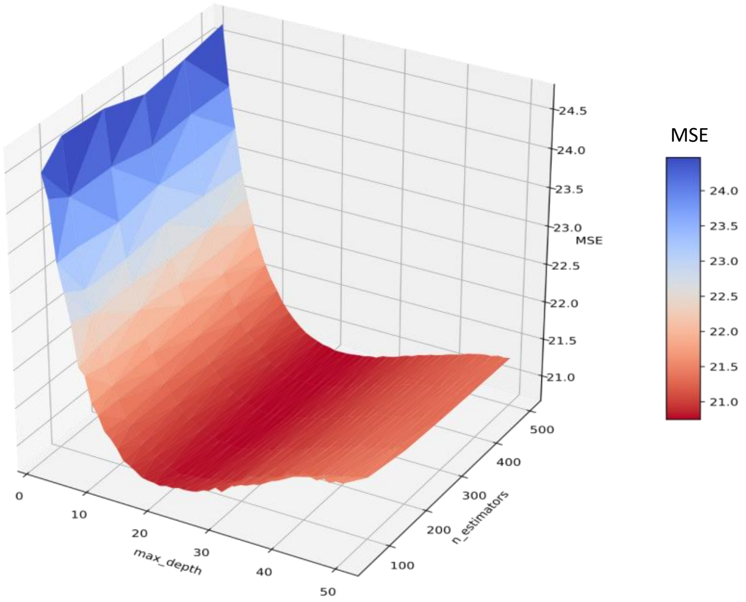
Hyper parameter tuning for Decision Tree optimizing 1 hyper parameter



Decision Tree using 12 features from the Lending Club data set

- Minimum MSE of 21.3699 at max_depth = 9

Hyper parameter tuning for Random Forest optimizing 2 hyper parameters



Random Forest using 12 features from the Lending Club data set

- Minimum MSE of 20.7431 at max_depth = 21, n_estimators = 500

References

James, G.; Witten, D.; Hastie, T.; Tibshirani, R.: An Introduction to statistical learning with Applications in R. 2013.

Chakraborty, C.; Joseph, A.: Machine learning at central banks. Staff Working Paper No. 674. Bank of England. URL: https://www.bankofengland.co.uk/-/media/boe/files/working-paper/2017/machine-learning-at-central-banks.pdf?la=en&hash=EF5C4AC6E7D7BDC1D68A4BD865EEF3D7EE5D7806

Protopapas, P.; Rader, K.; Dave, R.; Levine, M.: Data Science 1, Lecture #15: Regression Trees & Random Forests. Lecture Notes. URL: https://harvard-iacs.github.io/2017-CS109A/lectures/lecture15/presentation/lecture15_RandomForest.pdf

Scikit learn homepage:
https://scikit-learn.org/stable/modules/tree.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_text.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html
https://scikit-learn.org/stable/modules/ensemble.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html