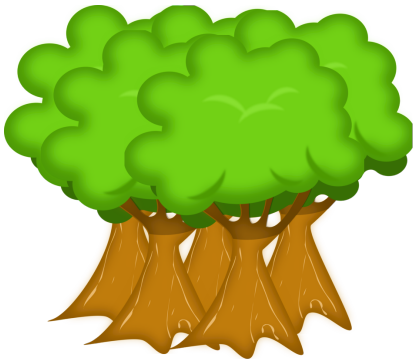
Random Forest

Nicolas Fontrodona

LBMC, ENS de Lyon, site Monod

December 5, 2018

Definition / Application



A Random Forest is a machine learning algorithm used to make predictions.

- Introduced by Leo Breiman (2001)

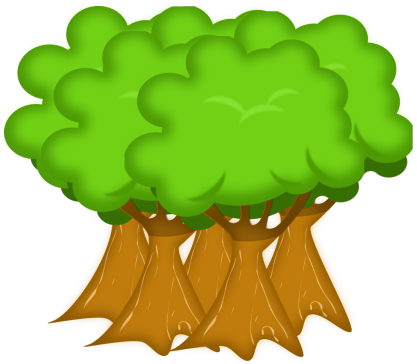
Applications

Medicine

Bank

Image processing

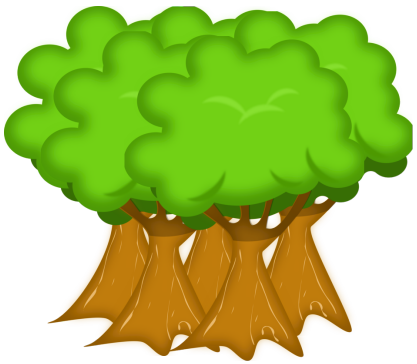
Definition / Application



A Random Forest is a machine learning algorithm used to make prediction.

- Introduced by Leo Breiman (2001)
- It works by constructing a multitude of decision trees and uses them to make predictions

Definition / Application



A Random Forest is a machine learning algorithm used to make prediction.

- Introduced by Leo Breiman (2001)
- It works by constructing a multitude of decision trees and uses them to make predictions

But what is exactly a decision tree ?

Decision Trees

Decision trees

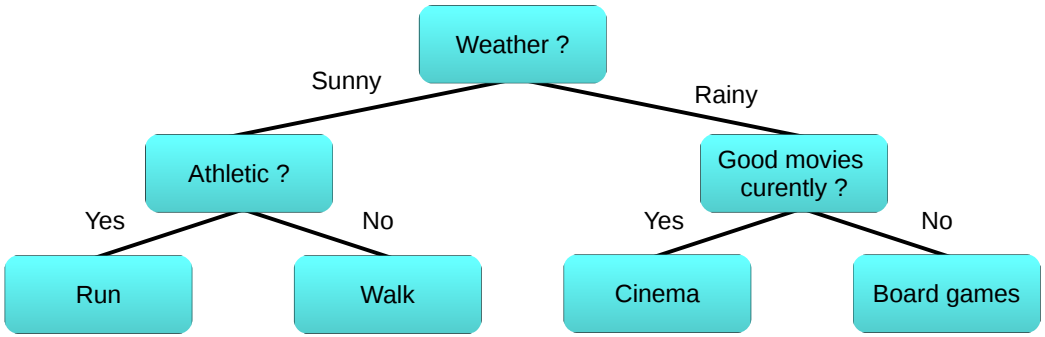
- Help to take decisions / to make predictions
- Tree-like graph
- Decisions are located on the leaves of the tree
- Are build with training data
  - Target variable : variable that we want to predict
  - Predictor variable(s) : variable(s) used for the predictions
- Used for data exploration in many fields

Decision Trees

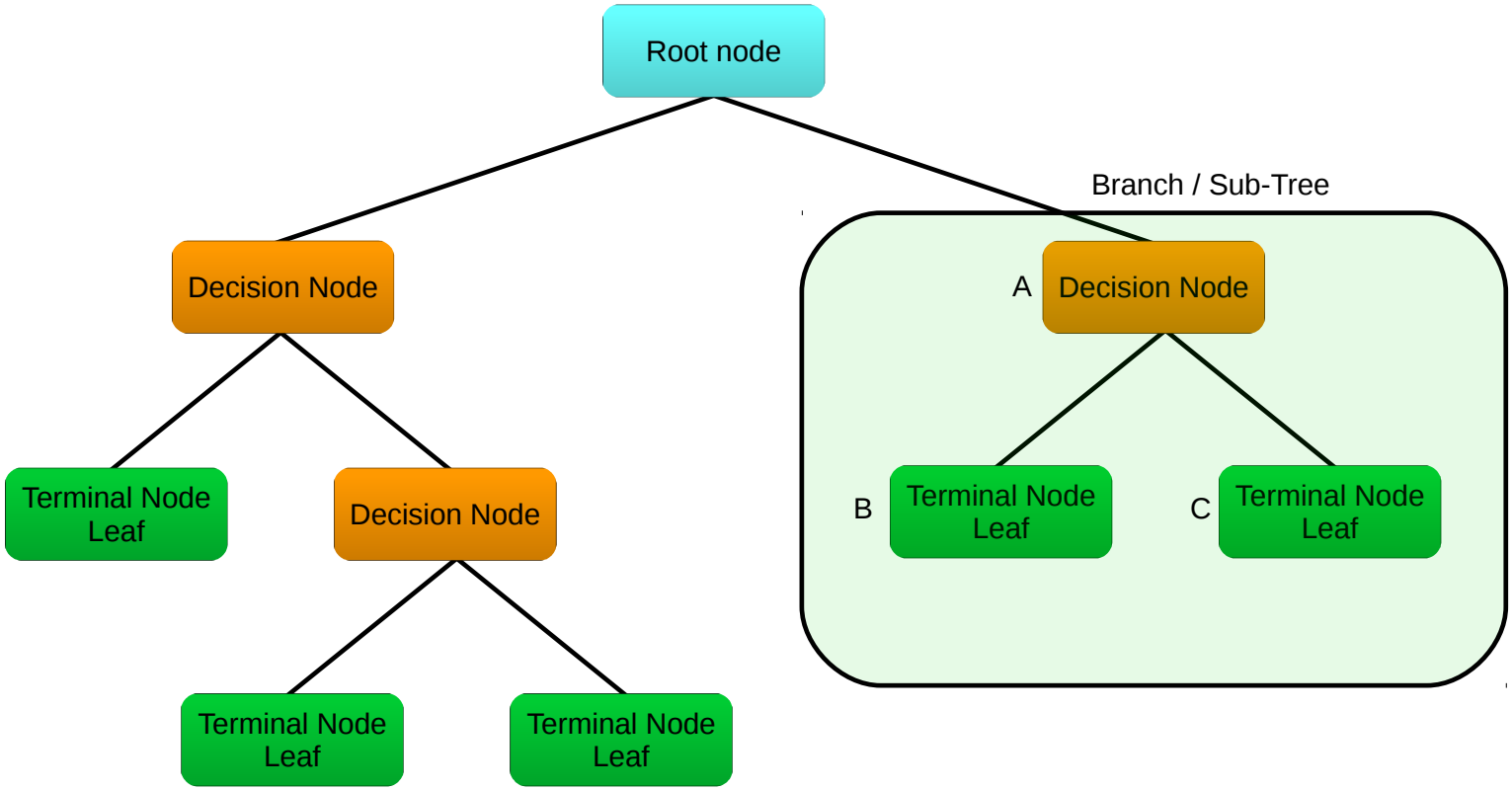
Decision trees

- Help to take decisions / to make predictions
- Tree-like graph
- Decisions are located on the leaves of the tree
- Are build with training data
  - Target variable : variable that we want to predict
  - Predictor variable(s) : variable(s) used for the predictions
- Used for data exploration in many fields

Or in every-day life ...



Decision trees
Terminology



- **B** and **C** are the children of **A**
- **A** is the parent of **B** and **C**

Decision Trees

Type of decision tree

- Classification Trees
  - Target variable : categorical variable
- Regression Trees
  - Target variable : continuous variable

Predictors, can be categorical or continuous variables

The trees can be built using different algorithms:

- Classification and Regression Tree algorithm (CART)
  - Introduced by Breiman in 1984

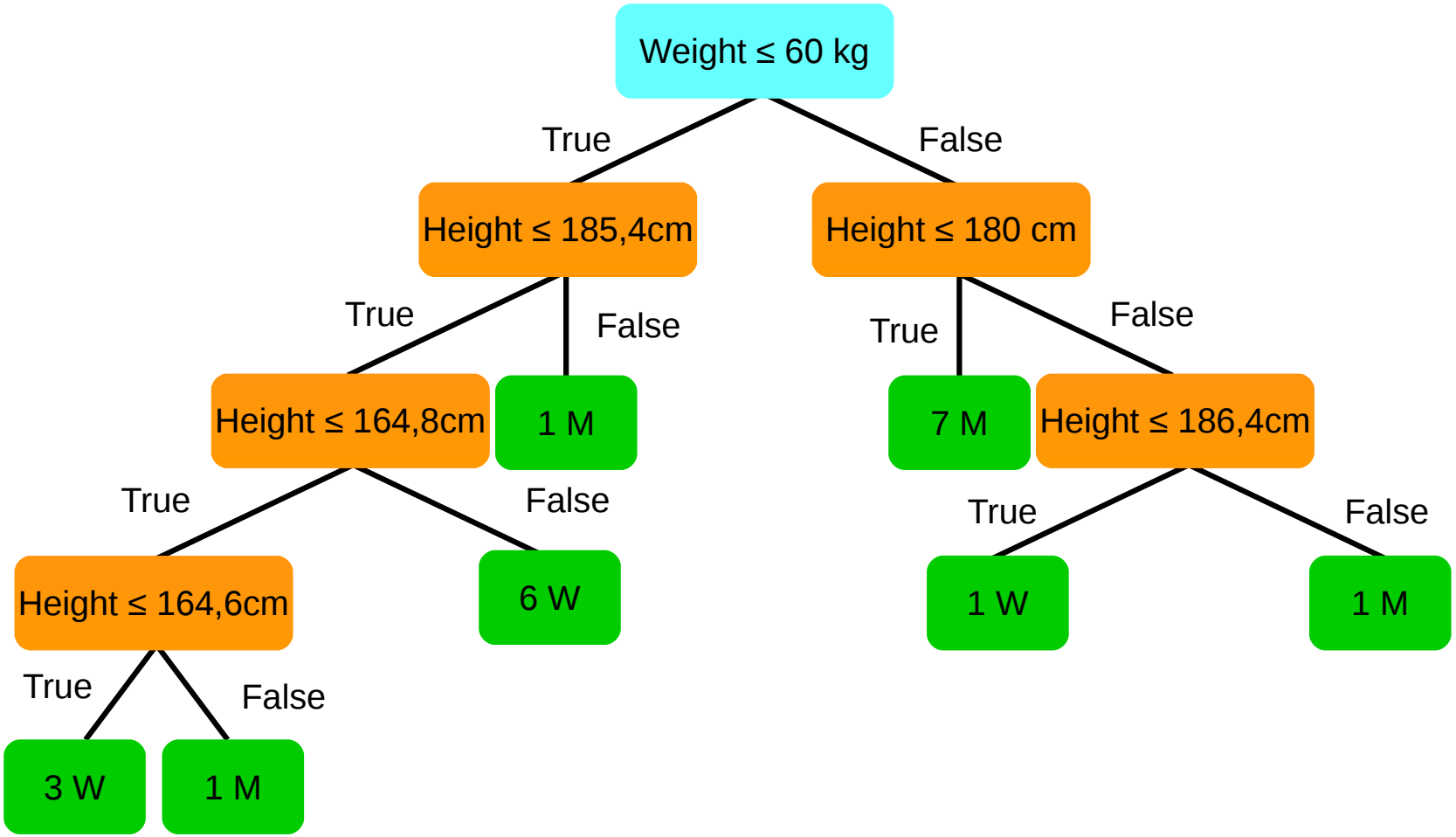
Classification Trees
Example

Let's consider a tree built (with the CART algorithm) from class composed of 20 students.

- Target variable : gender
- Predictors : size, weight of the students

Classification Trees
Example

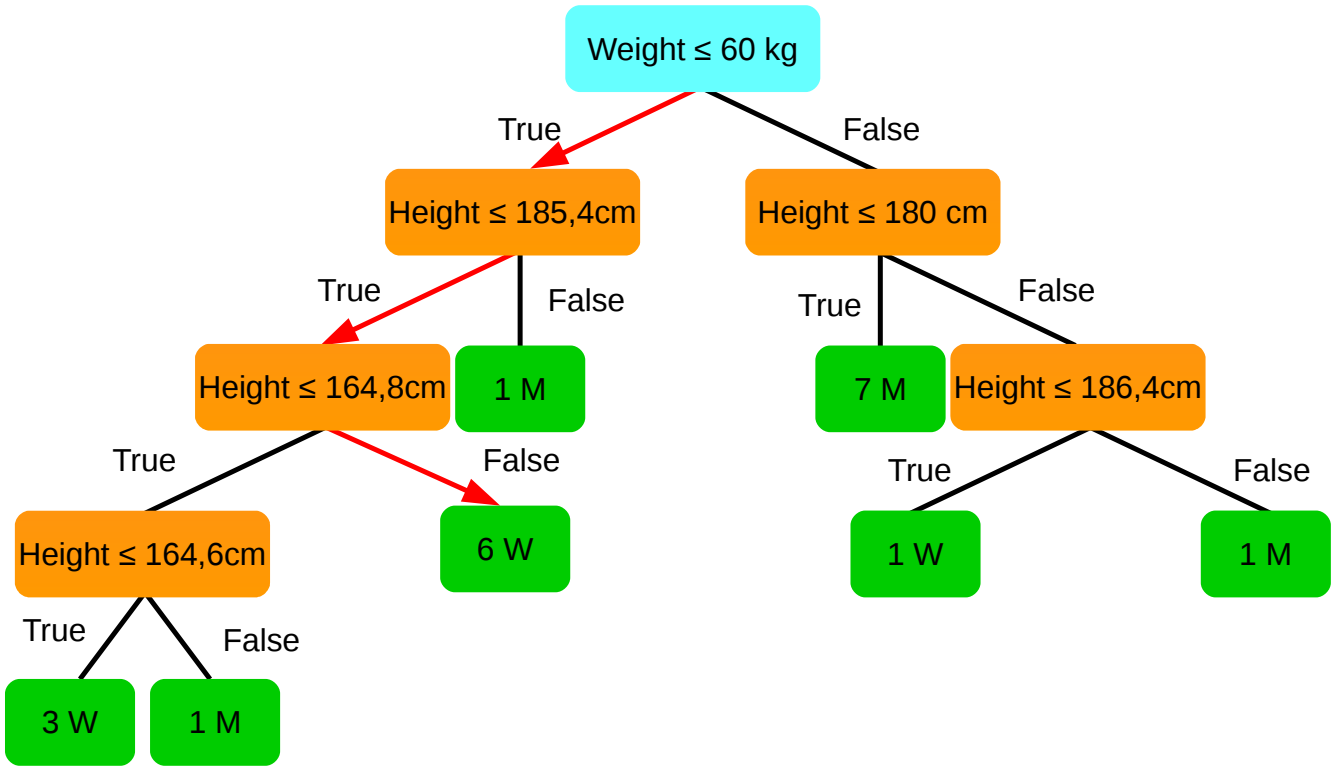
Tree built from our training set of students.



Prediction for a student with a weight of 59 kg and a size of 182 cm ?

Classification Trees
Example

Tree build from our training set of students



Prediction for a student with a weight of 59 kg and a size of 182 cm ? Woman

Prediction (classification tree) : modality of the majority of observations in the leaf where a test record fall.

How the classification tree is built ?

Main idea : Split node to increase homogeneity of the target variable.
Several algorithms are used to do this:

- Gini index : (default in CART algorithm) compute the impurity of a node
- And others (Entropy) ...

How the classification tree is built ?

Main idea : Split node to increase homogeneity of the target variable.
Several algorithms are used to do this:

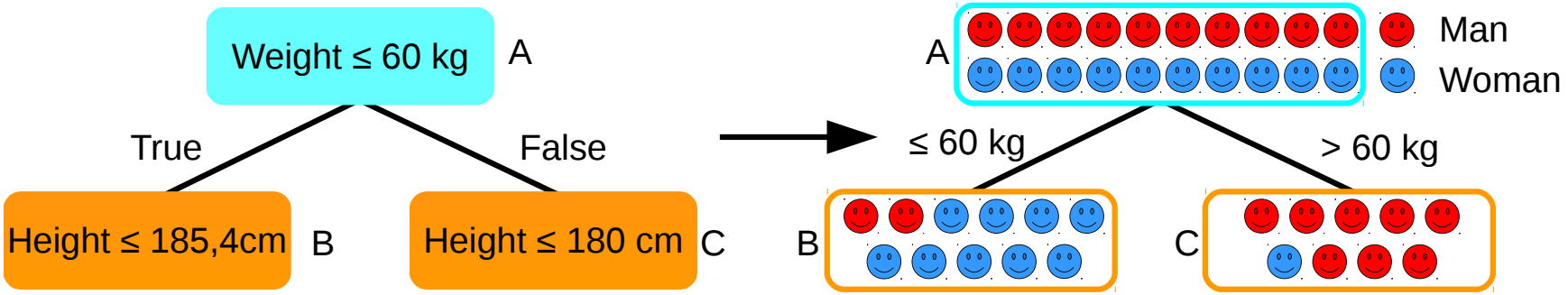
- Gini index : (default in CART algorithm) compute the impurity of a node
- And others (Entropy) ...

Gini formula (for a node): $G = 1 - \sum_{i=1}^{m} p_i^2$

Where :

- m is the number of modality of the target variable
- p_i frequency of the modality i in a node

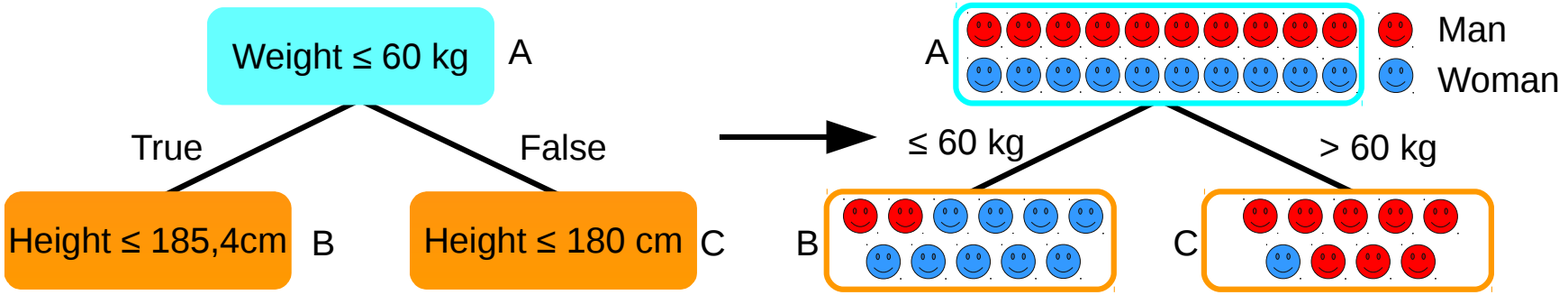
Classification Trees
Building the tree



Gini impurity of the 3 nodes

- $G = 1 - \sum_{i=1}^{m} p_i^2$

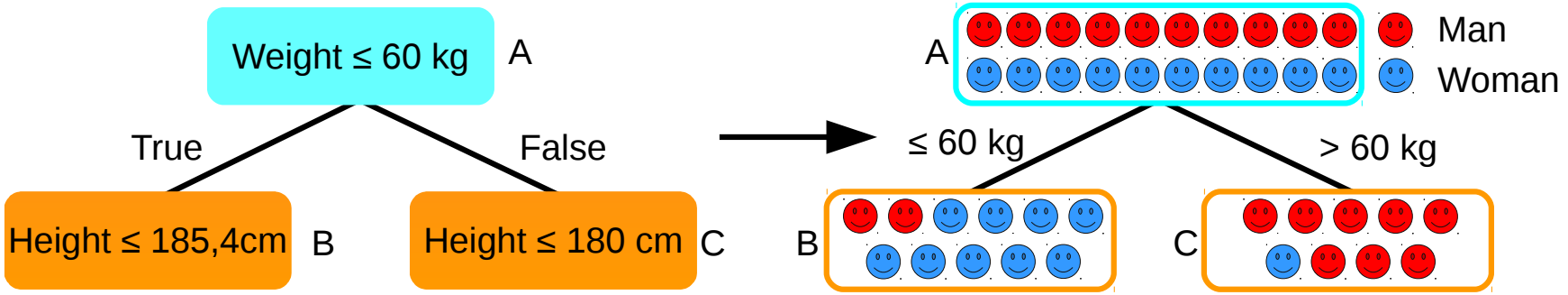
Classification Trees
Building the tree



Gini impurity of the 3 nodes

- $G = 1 - \sum_{i=1}^{m} p_i^2$
- $G_A = 1 - \left(\frac{10}{20}^2 + \frac{10}{20}^2\right) = 0.5$

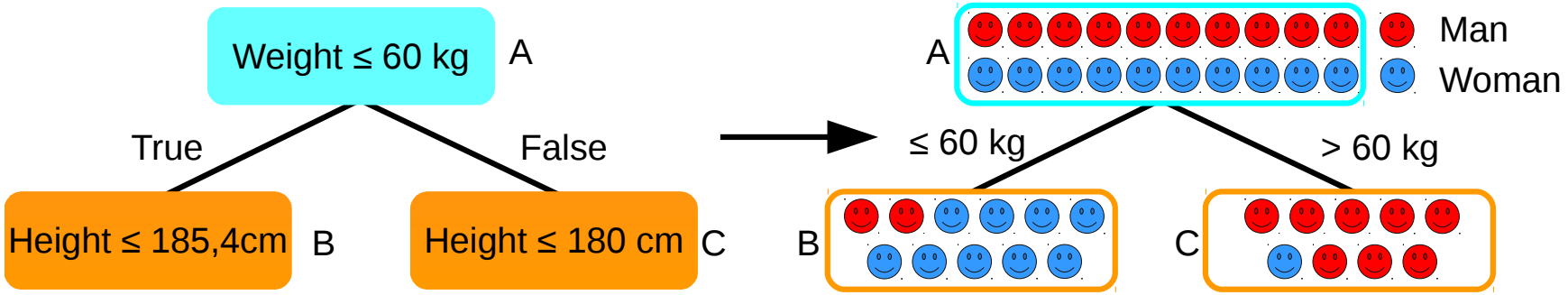
Classification Trees
Building the tree



Gini impurity of the 3 nodes

- $G = 1 - \sum_{i=1}^{m} p_i^2$
- $G_A = 1 - \left(\frac{10}{20}^2 + \frac{10}{20}^2\right) = 0.5$
- $G_B = 1 - \left(\frac{9}{11}^2 + \frac{2}{11}^2\right) = 0.298$

Classification Trees
Building the tree

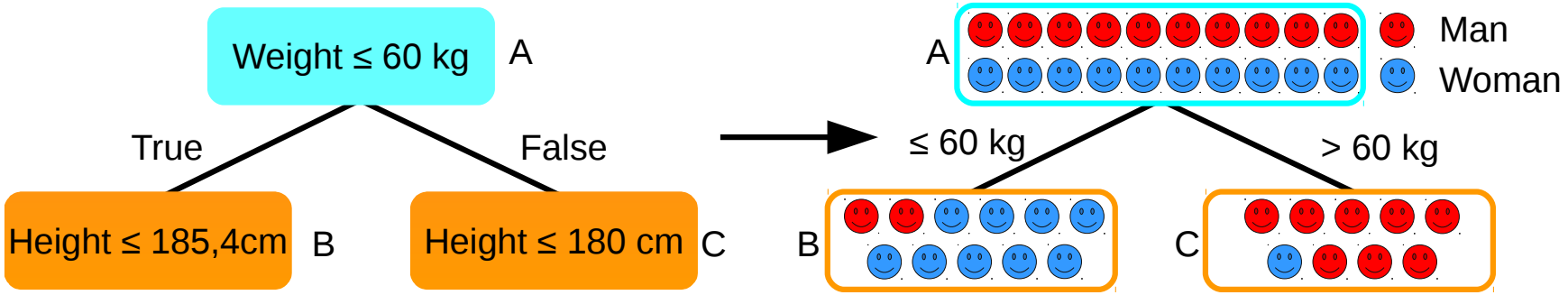


Gini impurity of the 3 nodes

- $G = 1 - \sum_{i=1}^{m} p_i^2$
- $G_A = 1 - \left(\frac{10}{20}^2 + \frac{10}{20}^2\right) = 0.5$
- $G_B = 1 - \left(\frac{9}{11}^2 + \frac{2}{11}^2\right) = 0.298$
- $G_C = 1 - \left(\frac{1}{9}^2 + \frac{8}{9}^2\right) = 0.198$

Classification Trees

Building the tree



To compute the information gain we use the following formula :

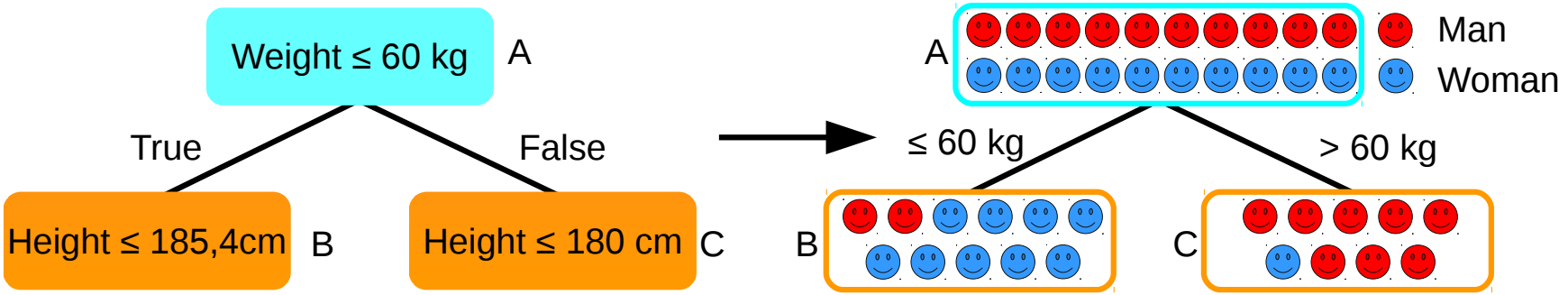
$$IG_{split\_A} = G_A - \frac{|Y_B|}{|Y_A|} \times G_B - \frac{|Y_C|}{|Y_A|} \times G_C$$

Where

- Y is the target variable
- $|Y_x|$ is the length of Y in the node x
- G_x is the gini impurity of the node x

Classification Trees

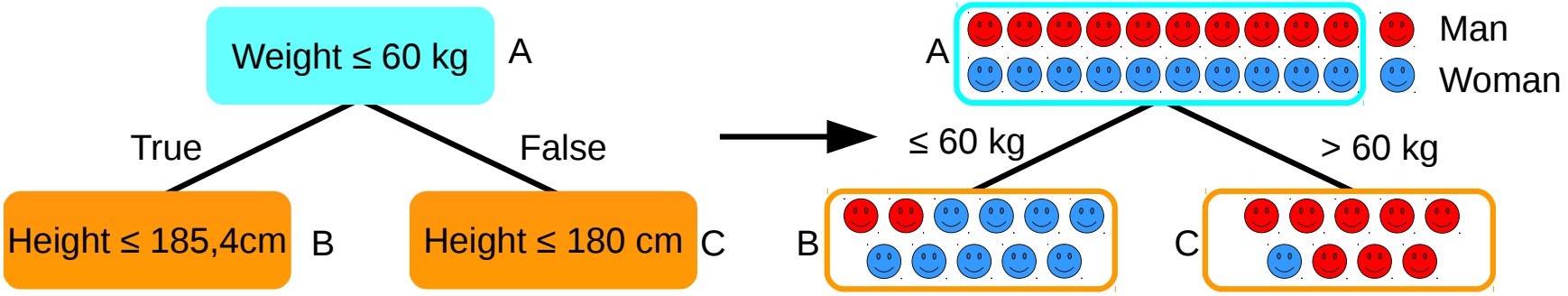
Building the tree



Information gain of the first split

- $G_A = 1 - \left(\frac{10}{20}^2 + \frac{10}{20}^2\right) = 0.5$
- $G_B = 1 - \left(\frac{9}{11}^2 + \frac{2}{11}^2\right) = 0.298$
- $G_C = 1 - \left(\frac{1}{9}^2 + \frac{8}{9}^2\right) = 0.198$
- $IG_{split\_A} = G_A - \frac{|Y_B|}{|Y_A|} \times G_B - \frac{|Y_C|}{|Y_A|} \times G_C$
- $IG_{split\_A} = 0.5 - \frac{11}{20} \times 0.298 - \frac{9}{20} \times 0.198 = 0.247$

Classification Trees
Building the tree



Information gain of the first split

- $IG_{split\_A} = 0.5 - \frac{11}{20} \times 0.298 - \frac{9}{20} \times 0.198 = 0.247$
- CART algorithm do this calculation for every values of every predictors in order to find the split with the biggest *IG* value.
- In our case : $Weight \leq 60kg$

Regression trees

- Globally same principle as classification tree
  - slightly variation in tree building
    - Other impurity index used : MSE (Mean Square index - default in CART algorithm) ...
    - The tree may grow until there is only leaves containing one sample (over-fit)
  - slightly variation in tree prediction
    - A sample falling in a leaf will take the mean value of the target variable of the training data in that leaf

Decision Trees

Classification and Regression Trees - Formalization

- Division of the predictor space into distinct and non-overlapping regions when building the tree
- Build by a top-down greedy approach / recursive binary splitting approach.
  - Note : Not all the algorithms use the binary splitting ...
  - But CART does.
- The splitting process goes on until a user defined stopping criteria is reached.
  - example : stop splitting when the leaves have less than 50 observations.

Decision trees

Advantages

- Easy to understand
- Useful in data exploration
- Little data cleaning required
- No constraint on predictor(s) / target variable type
- Non parametric method

Decision trees

Disadvantages

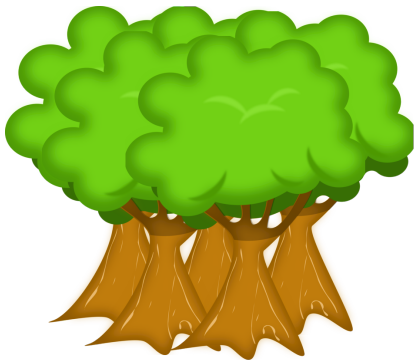
- No very well fitted for a continuous target variable
- Tree structure, locally greedy
- Computation of the impurity of the tree $\Rightarrow$ overfitting

Decision trees

Disadvantages

- No very well fitted for a continuous target variable
- Tree structure, locally greedy
- Computation of the impurity of the tree $\Rightarrow$ overfitting

Solution (overfitting): Create a multitude of decision trees $\Rightarrow$ Random Forest



Algorithm

1. From a training set T with N observations
   - Construct K samples of size N taken from T with replacement.
   - Each sampled training set will contain about 2/3 of the observations in T

Algorithm

1. From a training set T with N observations
   - Construct K samples of size N taken from T with replacement.
   - Each sampled training set will contain about 2/3 of the observations in T
2. Grow K trees
   - Build each tree with a different sampled training set.
   - If P are the predictors, p predictors (with $p < P$) are randomly selected to split each node of each tree.
     - This will induce a week correlation between the trees $\Rightarrow$ reduce the over-fitting

Algorithm

1. From a training set T with N observations
   - Construct K samples of size N taken from T with replacement.
   - Each sampled training set will contain about 2/3 of the observations in T
2. Grow K trees
   - Build each tree with a different sampled training set.
   - If P are the predictors, p predictors (with $p < P$) are randomly selected to split each node of each tree.
     - This will induce a week correlation between the trees $\Rightarrow$ reduce the over-fitting
3. Predict new data by aggregating the predictions of the K Trees. (i.e majority votes for classification, average for regression)

Random Forest

Advantages

- Inherits the advantages of decision trees
- No overfit
- Handle large data sets with large dimensions.

Random Forest

Advantages

- Inherits the advantages of decision trees
- No overfit
- Handle large data sets with large dimensions.

Disadvantages

- Not very well fitted for a continuous target variable.
- Time consuming for a forest with a lot of trees
- Very little control on what the model does

Application - Fisher's Iris data set

Ronald Fisher

Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in 1936

- The data set contains 50 records for each iris species (setosa, versicolor, virginica).
- For each flower the width and the length of the petal and the sepal were measured

Application - Fisher's Iris data set

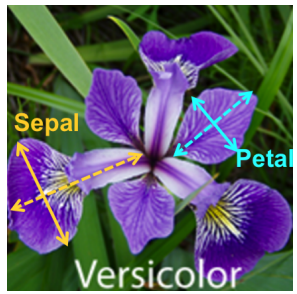
Ronald Fisher

Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in 1936

- The data set contains 50 records for each iris species (setosa, versicolor, virginica).
- For each flower the width and the length of the petal and the sepal were measured

Objectives

- Predict the species of the iris thanks to the width and the length of their petals and sepals
- We will use a random forest classifier



Application - Fisher's Iris data set

RandomForest package

Application - Fisher's Iris data set

RandomForest package

Objectives

- Target variable : Species
- Predictors : Width and length of petals and sepals

```
1  # import modules
2  from sklearn.ensemble import RandomForestClassifier
3  import sklearn.datasets # allow to load iris data set
4  import pandas as pd # dataframe management in python
5  import numpy as np # array management
6  # display of graphics
7  import seaborn as sns
8  from matplotlib import pyplot as plt
```

Loading the data set

```
1  # loading iris dataset
2  iris = sklearn.datasets.load_iris()
3  plant_df = {0: "setosa", 1: "versicolor", 2: "virginica"}
4  df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
5                    columns= iris['feature_names'] + ['species'])
6  df["species"] = df["species"].map(plant_df)
7  df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

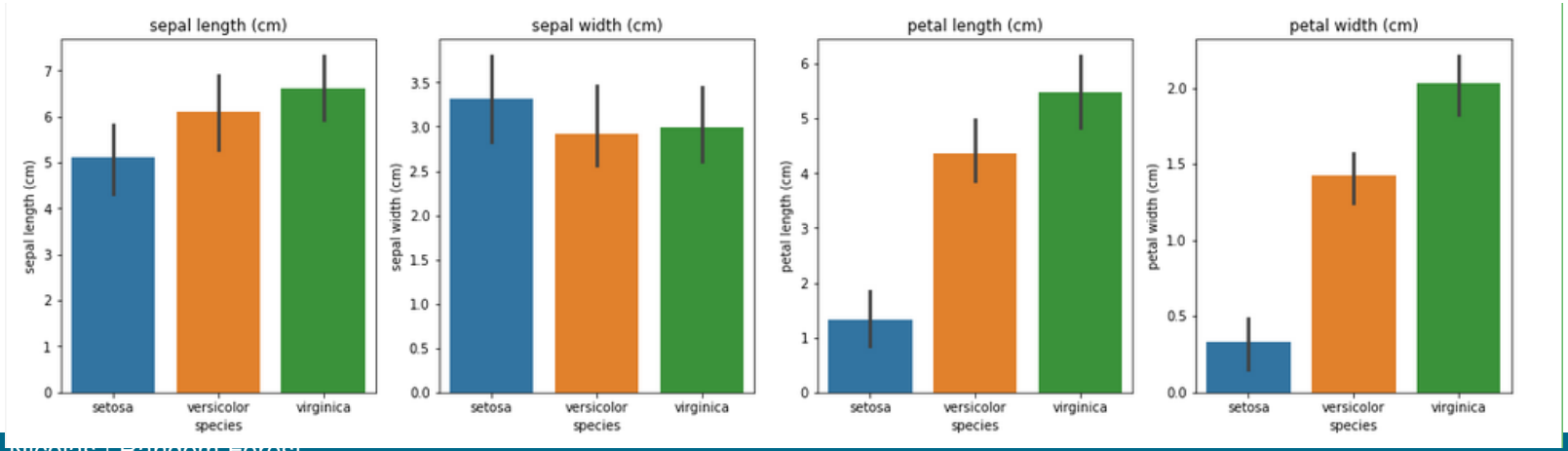
Exploration of the data set

```
1  # see the number of NaN values for the different columns ...
2  missing = df.isnull().sum()
3  unmissing = df.count()
4  percent_1 = missing / df.count() * 100
5  percent_2 = (round(percent_1, 1))
6  missing_data = pd.concat([missing, percent_2, unmissing], axis=1,
7                           keys=['missing', '%_missing', 'present'])
8  missing_data = missing_data.sort_values(by="missing", ascending=False)
9  missing_data.head(5)
```

	missing	%_missing	present
sepal length (cm)	0	0.0	150
sepal width (cm)	0	0.0	150
petal length (cm)	0	0.0	150
petal width (cm)	0	0.0	150
species	0	0.0	150

Exploration of the data set

```
# Adding noise
for i in range(len(df.columns[:-1])):
    my_mean = np.mean(df[df.columns[i]])
    df[df.columns[i]] = [val + ((np.random.random() * my_mean ) - my_mean / 2)
                         for val in  df[df.columns[i]]]
# Displaying a barplot of each predictor in function of the plant specie
ncol = len(df.columns[:-1])
fig, axes = plt.subplots(nrows=1, ncols=ncol, figsize=(20, 5))
for i in range(len(df.columns[:-1])):
    ax = sns.barplot(x=df["species"], y=df[df.columns[i]], ci=100, ax = axes[i])
    ax.set(ylabel = df.columns[i], title= df.columns[i])
```



Creation of a test / train set

```
1  # creation of a column is_train to select the data in the train or in the test dataset
2  df["is_train"] = np.random.uniform(0, 1, len(df)) <= .8
3  # creation of train and test dataframe
4  train, test = df[df['is_train']==True], df[df['is_train']==False]
5
6  # removing the column 'is_train'
7  train = train.drop('is_train', axis=1)
8  test = test.drop('is_train', axis=1)
9
10 # print the content of the selected dataset
11 print("train dataset %s flowers" % len(train))
12 print(train["species"].value_counts())
13 print("test dataset %s flowers" % len(test))
14 print(test["species"].value_counts())
```

```
train dataset 115 flowers
versicolor    41
setosa        39
virginica     35
Name: species, dtype: int64
test dataset 35 flowers
virginica     15
setosa        11
versicolor     9
Name: species, dtype: int64
```

Transform every non-numerical variable into numerical one

```
1  train.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	7.653718	4.415091	2.043026	0.430510	setosa
1	2.009672	2.303921	2.619738	0.361065	setosa
3	4.706728	4.406656	0.163423	0.093443	setosa
4	5.439364	4.639617	2.623645	0.740171	setosa
6	4.728835	2.078635	2.882080	0.732034	setosa

```
1  # Turn every non numeric variable into numeric one.
2  to_num = {"setosa": 0, "versicolor": 1, "virginica" : 2}
3  train["species"] = train["species"].map(to_num)
4  train.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	7.653718	4.415091	2.043026	0.430510	0
1	2.009672	2.303921	2.619738	0.361065	0
3	4.706728	4.406656	0.163423	0.093443	0
4	5.439364	4.639617	2.623645	0.740171	0
6	4.728835	2.078635	2.882080	0.732034	0

Building and training the classifier

```
1  p = train.drop("species", axis=1) # predictors
2  y = train["species"] # target
3  # Building the classifier
4  clf = RandomForestClassifier(n_jobs=3, random_state=1, n_estimators=100,
5                               criterion='gini', oob_score=True)
6  # Training the classifier
7  clf.fit(p, y)
```

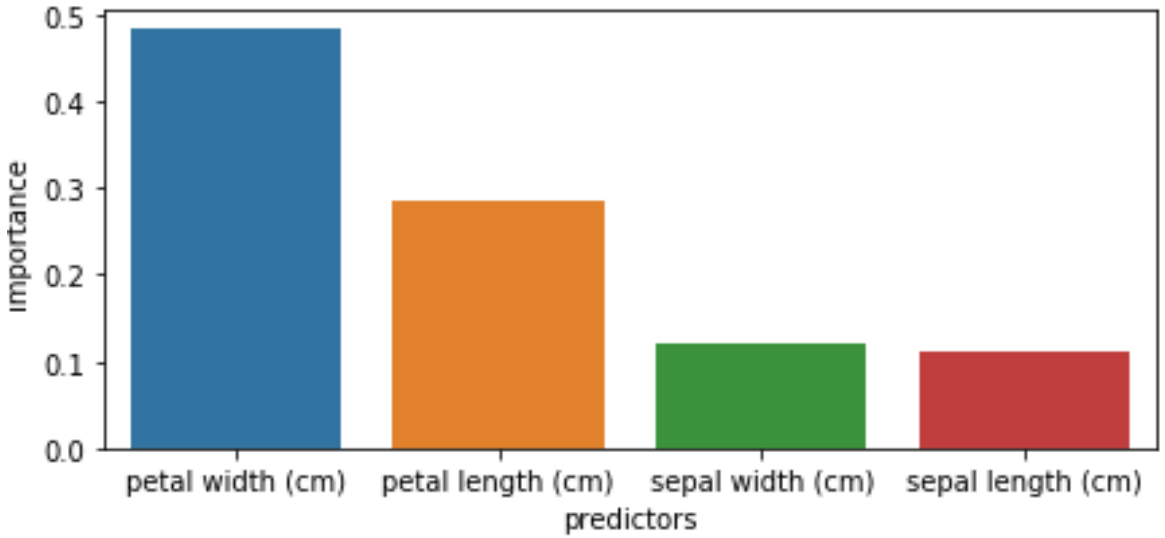
```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini', max_depth=None, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=3,
oob_score=True, random_state=1, verbose=0, warm_start=False)
```

Above between parenthesis : hyper-parameters of the random forest

classifier

Importance of predictors

```
# Display the importance of predictors
importances = pd.DataFrame({'predictors':p.columns,
                            'importance':np.round(clf.feature_importances_,10)})
importances = importances.sort_values('importance',ascending=False)
                            .set_index('predictors')
plt.subplots(figsize = (7, 3))
sns.barplot(x=importances.index, y=importances["importance"])
plt.show()
```



Prediction of the classifier

```
1  # Apply the Classifier to the test data (which, remember, it has never seen before)
2  preds = list(map(lambda x: plant_df[x], clf.predict(test.drop("species", axis=1))))
3  real = test["species"]
4  # creation of the confusion matrix
5  pd.crosstab(np.array(real), np.array(preds),
6              rownames=['Actual species'], colnames=['Predicted species'])
```

Predicted species	setosa	versicolor	virginica
Actual species			
setosa	10	1	0
versicolor	0	6	3
virginica	0	0	15

Accuracy of the classifier

```
1  # Accuracy of the model with out of bag score
2  print("oob score:", round(clf.oob_score_, 4)*100, "%")
3  from sklearn.metrics import precision_score, recall_score
4  print("Precision:", round(precision_score(real, preds, average="micro"), 2) * 100, "%")
5  print("Recall:", round(recall_score(real, preds, average="micro"), 2) * 100, "%")
```

```
oob score:  77.39 %
Precision:  89.0 %
Recall:  89.0 %
```

Conclusion

Conclusion

Random forest :

- Machine learning algorithm. Training set needed to build a forest.
- Useful for data exploration of large data sets
- Easy to use
- Don't overfit the model thanks to random sampling of predictors and training records to build each tree.
- Time consuming when building a forest with a lot of trees on large data sets.