Decision Trees and Random Forests
Data science Certificate

Aymeric DIEULEVEUT

May 2020

1. Decision Trees

2. Random Forests

Outline

Random Forests

- Task: do these pictures contain individuals that look nice (yes or no), suspicious (yes or no)?
- Train algorithms using data (a lot of these):

- Pictures: $X_i \in \mathbb{R}^d$, Labels $Y_i \in \{0, 1, 2, 3\}$
- Features: categorical and numerical features (is there a smile? what is the age of the person ?)
- Size of the dataset: n for training $+t$ for test.

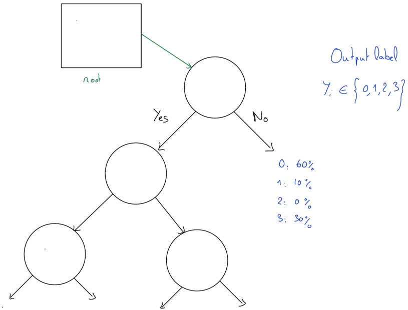
Goals of Decision Trees

1. Prediction: what is the class of this picture (Classification) ? What is the age of the person (Regression) ?

2. Provide some understanding: what on this picture allows me to classify it? Ex: presence of a face, human, smiling, etc ...

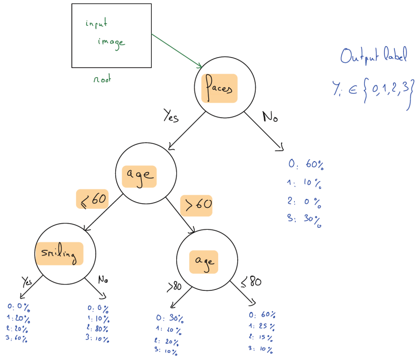
Decision Tree

- Nodes = tests
- Branches = possible outcomes
- Leaves = final decision.

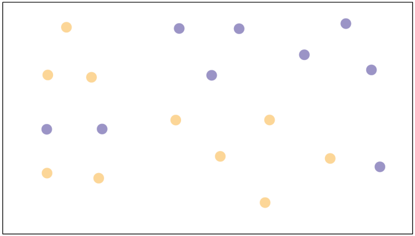


Decision Tree

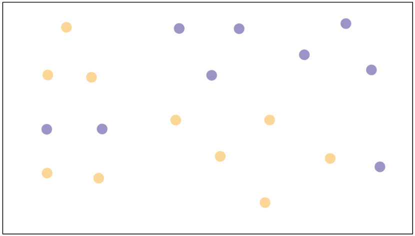
- Nodes = tests
- Branches = possible outcomes
- Leaves = final decision.



Building trees : CART



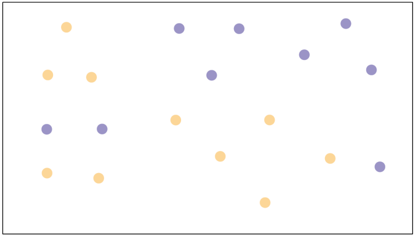
Building trees : CART



How to chose the best cut ?

- e.g., Given a possible cut, measure the reduction of « impurity »:
  - in regression: mean-squared-error $\sum_{i \in C} (Y_i - \bar{Y}_C)^2$
  - in classification: Gini's impurity.
- Find dimension and threshold with optimal impurity reduction.
- Iterate until stopping criterion is met.

Gini's impurity



Gini impurity measures **how often a randomly chosen element from the set** would be **incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset**.

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} p_i(1-p_i) = 1 - \sum_{i=1}^{J} {p_i}^2$$

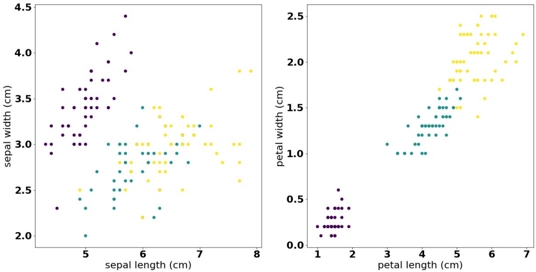
For two classes, related to the variance if classified as Bernoulli r.v..

Example: Iris Dataset

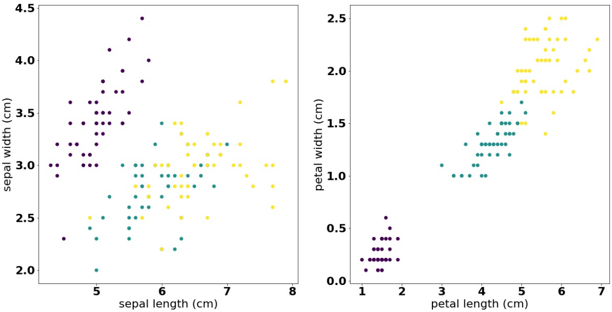
Figure: Iris: setosa, versiocolor, virginica

One of the most widely used toy dataset in classification:

- 3 classes : *['setosa', 'versicolor', 'virginica']*
- 50 points per class.
- 4 features: *['sepal length', 'sepal width', 'petal length', 'petal width']*



Exercise: Guess Classification tree for Iris Dataset



In pratice : Scikit Learn

Universal Python library for Machine Learning:

- Very easy to use
- Very well documented
- Open Source

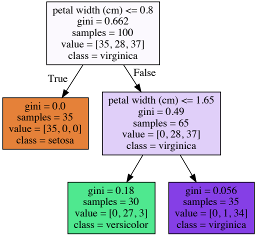
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Example: Iris Dataset

What do we need to specify ?

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split

# Choose the Learning algorithm
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state=0)

#Load the dataset
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.33, random_state=43)

#Check accuracy
cross_val_score(clf, iris.data, iris.target, cv=10)

# Fit the model
clf.fit(X_train,y_train)

# Plot tree
plot_tree(clf, feature_names = fn,
              class_names=cn,
              filled = True)
plt.show()
```

Output:



Example:

```python
# Check
X_test[0,:], iris.target_names[y_test[0]]
```

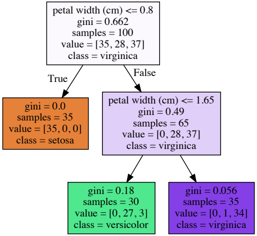
Returns:

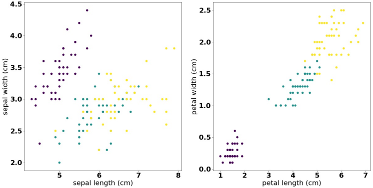
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')

Code: https://colab.research.google.com/drive/1dXYIjacNAeASgl54vIWyWjMQlfgigSVA?usp=sharing#scrollTo=VeJHkJd7RsDw

Example: Iris Dataset

Output:





Example:

```
# Check
X_test[0,:], iris.target_names[y_test[0]]
```

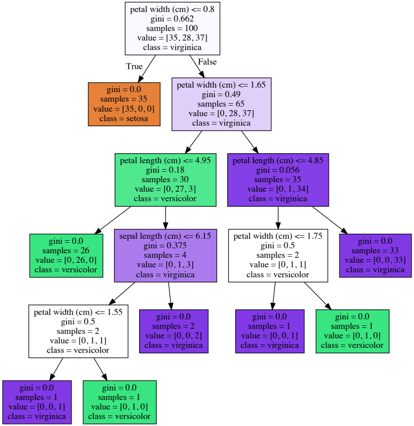
Returns:
(array([4.8, 3.1, 1.6, 0.2]), 'setosa')

Code: https://colab.research.google.com/drive/1dXYIjacNAeASgl54vIWyWjMQlfgigSVA?usp=sharing#scrollTo=VeJHkJd7RsDw

What if I use a deeper tree ? Or if I do not specify the depth in the model definition ?

Deeper trees

```
# Choose the Learning algorithm
clf = DecisionTreeClassifier(random_state=0)
```



- When does the algorithm stop then ?
- What do you think about it ?

Stopping / Pruning

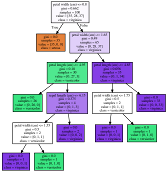
Trees that achieve perfect classification may suffer from over-fitting. (see example in the lab)

To avoid this problem, we can reduce the complexity of the trees:

1. Stopping rules
   - Set a minimum number of samples inside each leaf.
   - Set a maximum depth.
2. Pruning
   - Reduced error pruning.
   - Cost complexity pruning.

Pruning : example of Cost complexity pruning.

We create a sequence of Trees by changing one subtree at each step into a leaf, until we get only the root: the subtree is chosen to minimize the error made.[1] Then the test error is evaluated along the sequence, to choose the optimal pruning.



[1] *more details here*

Cart: pros and cons

Pros:

- Simple to understand, interprete, visualize
- Implicitly perform features/variable selection
- can handle both categorical and numerical data
- little effort for data preparation
- Non linear relationships do not affect performance
- Can work with large number of observations.

Cons:

1. Can create over-complex trees: overfitting
2. Unstability: small variations of data can generate completely different trees
3. Cannot guarantee global optimal tree
4. Biased if some classes dominate

Solution : Using Random Forests !

Outline

Random Forests

To put it in simple words: « random forests builds multiple decision trees and merges them together to get a more accurate and stable predictions ».
↬key idea: create variable trees + aggregate them

How to create variability ?

- Instead of the most important features, search the best feature among a random subset (ntry).
- Work on subsets of data (bootstrap=True).
- More `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`

```
# Creating and fitting a Random Forest
from sklearn.ensemble import RandomForestClassifier

plt.figure(figsize=(20,10))
clf = RandomForestClassifier(n_estimators = 100,
                             max_depth=3, bootstrap = True,
                             random_state =43)

clf.fit(X_train,y_train)
```

How to aggregate ? How to interpret a Random Forest ?

Aggregation

We using a voting or averaging process !

Feature importance

- how much tree nodes using a particular feature reduce impurity along the trees.
- suggests feature selection rule: drop out features with low importance to avoid overfitting.
- Attribute **feature_importance_** in RandomForestRegressor of Sklearn.

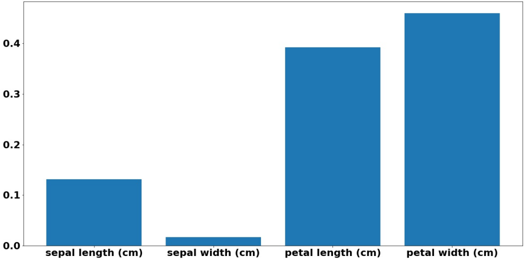


Figure: Feature importane output for a Random Forest Classifier in Python

Comparison with decision trees

Cons:

- Less interpretable
- Slower to run

Pros:

- Better in higher dimension
- More stable outputs
- Feature selection

Improving predictions - more arguments.

1. **n_estimators** : number of trees in the forest
   - improves prediction / stability
   - slows down the algorithm
2. **max_features** : maximum number of features considered to split a node
3. **min_sample_leaf** : minimum number of samples that should remain in a leaf node.

RF pros and cons

RF Pros:

1. works for classification and regression
2. default hyperparameters often produce reasonable prediction -> easy to use.
3. avoid overfitting if some good trees in the forest and easy feature selection -> high dimensional pb.
4. hard to beat in performance.

RF Cons:

1. Fast to train but slow to provide new predictions -> ineffective for « real-time predictions ».
2. Good for prediction but bad for description.

What about Regression ?

Questions?

Boosting

Another import technique (very powerful !)

Main idea:

- Give more importance to difficult point iteratively
- Incrementally building an ensemble by training each new instance to emphasize the training instances previously mis-modeled.

Example: AdaBoost

See: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`

Conclusion

1. Trees are one of the simplest method (the most intuitive / human like)
2. Random Forests give excellent results in many applications.

Lab :

- Example on a synthetic dataset of time series
- Application to inflation prediction in Brazil.