Data Analysis and Machine Learning: Trees, forests and all that

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

Nov 2, 2018

Decision trees, overarching aims

Decision trees are supervised learning algorithms used for both, classification and regression tasks where we will concentrate on classification in this first part of our decision tree tutorial. Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning. We can use decision trees for issues where we have continuous but also categorical input and target features.

Nodes, leafs, roots and branches

The main idea of decision trees is to find those descriptive features which contain the most **information** regarding the target feature and then split the dataset along the values of these features such that the target feature values for the resulting sub_datasets are as pure as possible.
The descriptive feature which leaves the target feature most purely is said to be the most informative one. This process of finding the **most informative** feature is done until we accomplish a stopping criteria where we then finally end up in so called **leaf nodes**.
The leaf nodes contain the predictions we will make for new query instances presented to our trained model. This is possible since the model has kind of learned the underlying structure of the training data and hence can, given some assumptions, make predictions about the target feature value (class) of unseen query instances.
A decision tree mainly contains of a **root node**, **interior nodes**, and **leaf nodes** which are then connected by **branches**.

How do we set it up?

In simplified terms, the process of training a decision tree and predicting the target features of query instances is as follows:

1. Present a dataset containing of a number of training instances characterized by a number of descriptive features and a target feature
2. Train the decision tree model by continuously splitting the target feature along the values of the descriptive features using a measure of information gain during the training process
3. Grow the tree until we accomplish a stopping criteria create leaf nodes which represent the *predictions* we want to make for new query instances
4. Show query instances to the tree and run down the tree until we arrive at leaf nodes

Then we are essentially done!

Decision trees and Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

steps=250

distance=0
x=0
distance_list=[]
steps_list=[]
while x<steps:
    distance+=np.random.randint(-1,2)
    distance_list.append(distance)
    x+=1
    steps_list.append(x)
plt.plot(steps_list,distance_list, color='green', label="Random Walk D

steps_list=np.asarray(steps_list)
distance_list=np.asarray(distance_list)

X=steps_list[:,np.newaxis]

#Polynomial fits

#Degree 2
poly_features=PolynomialFeatures(degree=2, include_bias=False)
X_poly=poly_features.fit_transform(X)
```

Maxwell-Boltzmann velocity distribution

```
# Program to test the Metropolis algorithm with one particle at given
# one dimension
#!/usr/bin/env python
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import random
from math import sqrt, exp, log
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
# initialize the rng with a seed
random.seed()
# Hard coding of input parameters
MCcycles = 100000
Temperature = 2.0
beta = 1./Temperature
InitialVelocity = -2.0
CurrentVelocity = InitialVelocity
Energy = 0.5*InitialVelocity*InitialVelocity
VelocityRange = 10*sqrt(Temperature)
VelocityStep = 2*VelocityRange/10.
AverageEnergy = Energy
AverageEnergy2 = Energy*Energy
VelocityValues = np.zeros(MCcycles)
# The Monte Carlo sampling with Metropolis starts here
for i in range (1, MCcycles, 1):
    TrialVelocity = CurrentVelocity + (2.0*random.random() - 1.0)*Velo
    EnergyChange = 0.5*(TrialVelocity*TrialVelocity -CurrentVelocity*C
```