Data Analysis and Machine Learning: Nearest Neighbors and Decision Trees

Morten Hjorth-Jensen[1,2]

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

Nov 1, 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.

Nearest Neighbors

```
import mglearn
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier

# Generate sample data
X = np.sort(5*np.random.rand(40,1), axis=0)
y = X**3
y=y.ravel()

# Add noise to targets
X[::4] +=(0.5 - np.random.rand(1))
y[::5] +=(0.5 - np.random.rand(8))

a=np.array(X)
b=np.array(y)

X_train=a[:19]
X_test=a[19:]
y_train=b[:19]
```

```
y_test=b[19:]

model=Pipeline([('poly', PolynomialFeatures(degree=3)),('linear', LinearRegression(fit_intercept=F
model=model.fit(X_train, y_train)
pred=model.predict(X_test)


poly=PolynomialFeatures(degree=3)
poly.fit_transform(X_train, y_train)
plt.scatter(X_test, y_test)
plt.plot(X_test, pred, color='green')
plt.show()

print (model.score(X_test,y_test))

print ("---------K-Nearest Neighbors-------")
"""neighbors_settings=range(1,11)
for n_neighbors in neighbors_settings:
    clf=KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    training_accuracy.append(clf.score(X_train, y_train))
    test_accuracy.append(clf.score(X_test, y_test))


print (mglearn.plots.plot_knn_regression(n_neighbors=3))"""

from sklearn.neighbors import KNeighborsRegressor

X, y=mglearn.datasets.make_wave(n_samples=40)
reg = KNeighborsRegressor(n_neighbors=3)
reg.fit(X_train, y_train)
```

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 Data")

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)

lin_reg=LinearRegression()
poly_fit=lin_reg.fit(X_poly,distance_list)
b=lin_reg.coef_
c=lin_reg.intercept_
print ("2nd degree coefficients:")
print ("zero power: ",c)
print ("first power: ", b[0])
print ("second power: ",b[1])

z = np.arange(0, steps, .01)
z_mod=b[1]*z**2+b[0]*z+c

fit_mod=b[1]*X**2+b[0]*X+c
plt.plot(z, z_mod, color='r', label="2nd Degree Fit")
plt.title("Polynomial Regression")

plt.xlabel("Steps")
plt.ylabel("Distance")

#Degree 10
poly_features10=PolynomialFeatures(degree=10, include_bias=False)
X_poly10=poly_features10.fit_transform(X)

poly_fit10=lin_reg.fit(X_poly10,distance_list)

y_plot=poly_fit10.predict(X_poly10)
plt.plot(X, y_plot, color='black', label="10th Degree Fit")

plt.legend()
plt.show()


#Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
regr_1=DecisionTreeRegressor(max_depth=2)
regr_2=DecisionTreeRegressor(max_depth=5)
regr_3=DecisionTreeRegressor(max_depth=7)
regr_1.fit(X, distance_list)
regr_2.fit(X, distance_list)
regr_3.fit(X, distance_list)

X_test = np.arange(0.0, steps, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
y_3=regr_3.predict(X_test)

# Plot the results
plt.figure()
plt.scatter(X, distance_list, s=2.5, c="black", label="data")
plt.plot(X_test, y_1, color="red",
         label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="green", label="max_depth=5", linewidth=2)
plt.plot(X_test, y_3, color="m", label="max_depth=7", linewidth=2)

plt.xlabel("Data")
plt.ylabel("Darget")
plt.title("Decision Tree Regression")
```

```
plt.legend()
plt.show()

# Program to test the Metropolis algorithm with one particle at given temp in
# 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)*VelocityStep
    EnergyChange = 0.5*(TrialVelocity*TrialVelocity -CurrentVelocity*CurrentVelocity);
    if random.random() <= exp(-beta*EnergyChange):
        CurrentVelocity = TrialVelocity
        Energy += EnergyChange
        VelocityValues[i] = CurrentVelocity
    AverageEnergy += Energy
    AverageEnergy2 += Energy*Energy
#Final averages
AverageEnergy = AverageEnergy/MCcycles
AverageEnergy2 = AverageEnergy2/MCcycles
Variance = AverageEnergy2 - AverageEnergy*AverageEnergy
print(AverageEnergy, Variance)
n, bins, patches = plt.hist(VelocityValues, 400, facecolor='green')

plt.xlabel('$v$')
plt.ylabel('Velocity distribution P(v)')
plt.title(r'Velocity histogram at $k_BT=2$')
plt.axis([-5, 5, 0, 600])
plt.grid(True)
from collections import Counter

#print (Counter(VelocityValues))

print (VelocityValues[:20])
VelocityValues=list(Counter(VelocityValues).keys())
d=list(Counter(VelocityValues).values())

VelocityValues=np.asarray(VelocityValues)[:, np.newaxis]
d=np.asarray(d)
print (VelocityValues.shape, d.shape)
```

```
plt.scatter(VelocityValues, d)
plt.show()

#2nd Degree Polynomial
poly_feat=PolynomialFeatures(degree=20, include_bias=False)
X_poly=poly_feat.fit_transform(VelocityValues)
lin_reg=LinearRegression()
poly_fit=lin_reg.fit(X_poly,d)

y_plot=poly_fit.predict(X_poly)
plt.title("Polynomial Fit")
plt.plot(VelocityValues, y_plot, color='black', label="Fit")
plt.show()

#Decision Trees

from sklearn.tree import DecisionTreeRegressor
regr_1=DecisionTreeRegressor(max_depth=2)
regr_2=DecisionTreeRegressor(max_depth=5)
regr_3=DecisionTreeRegressor(max_depth=7)
regr_1.fit(VelocityValues, d)
regr_2.fit(VelocityValues, d)
regr_3.fit(VelocityValues, d)

X_test = np.arange(0.0, MCcycles, 0.01)[:, np.newaxis]
y_1=regr_1.predict(X_test)
y_2=regr_2.predict(X_test)
y_3=regr_3.predict(X_test)

plt.title("Decision Tree")
plt.plot(X_test, y_1, color="red", label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="green", label="max_depth=5", linewidth=2)
plt.plot(X_test, y_3, color="m", label="max_depth=7", linewidth=2)
plt.show()

#Separate each frequency not in one specific velocity, but in a range of values,
#i.e. frequency of all velocities in range -5 to -4.9, -4.9 to -4.8, etc...
```