Data

Cédric Buche

ENIB

August 27, 2019

Data
Cédric Buche
ENIB
August 27, 2019

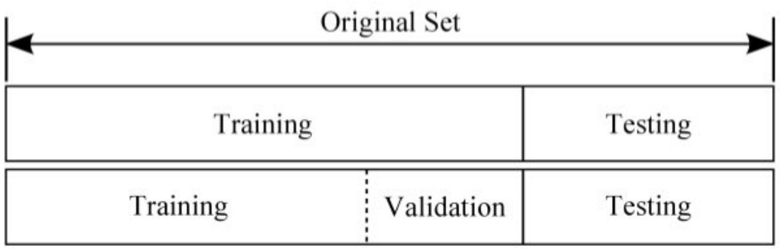
Page 1 :

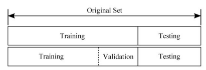
1 Dataset rules
2 Hyper Parameter tuning
3 Data preparation
4 Graphic tool for DataScientist
Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem
5 Reduction of dimension
Iris
The theory behind principal component analysis

Page 2 :

Page 3 :

Dataset





Page 4 :

How do you think about data? Think of a spreadsheet. You have columns, rows, and cells.
The statistical perspective of machine learning frames data in the context of a hypothetical function (f) that the machine learning algorithm aims to learn. Given some input variables ($Input$) the function answer the question as to what is the predicted output variable ($Output$).
$Output = f(Input)$
The inputs and outputs can be referred to as variables or vectors.
The computer science perspective uses a row of data to describe an entity (like a person) or an observation about an entity. As such, the columns for a row are often referred to as attributes of the observation and the rows themselves are called instances.

```
def split_data(data,prob):
        # split data into fractions [prob, 1 - prob]
        results=[],[]
        for row in data:
                results[0 if random.random() < prob else 1].append(row)
        return results



def train_test_split(x,y,test_pct):
        # pair corresponding values
        data = zip ( x , y )
        # split the data set of pairs
        train , test = split_data ( data , 1 - test_pct )
        x_train , y_train = zip ( * train )
        x_test , y_test = zip ( * test )
        return x_train , x_test , y_train , y_test



model = SomeKindOfModel ()
x_train , x_test , y_train , y_test = train_test_split ( xs , ys , 0.33
     )
model . train ( x_train , y_train )
performance = model . test ( x_test , y_test )
```


Page 5 :

Page 6 :

Hyper Parameter tuning

- the parameters of the learning phase: hyper-parameters.
- example: maximum number of values that will be tested in a node of a decision tree, or the number of trees that will contain a random forest.
- no formal method to find the optimal values from the training data.
- often use exhaustive search on ranges defined by the developer: this requires in practice to make as many learnings as combinations of parameters. This technique is called *Grid Search*. It uses one of the model's quality metrics to select the best set of hyper-parameters.

Hyper Parameter tuning
the parameters of the learning phase: hyper-parameters.
example: maximum number of values that will be tested in a node of a decision tree, or the number of trees that will contain a random forest.
no formal method to find the optimal values from the training data.
often use exhaustive search on ranges defined by the developer: this requires in practice to make as many learnings as combinations of parameters. This technique is called *Grid Search*. It uses one of the model's quality metrics to select the best set of hyper-parameters.

Page 7 :

Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Reduction of dimension

Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem
Reduction of dimension
Iris
The theory behind principal component analysis

Page 8 :

Features and Label

- the "features": we can measure them and it is from them that we will perform modeling and prediction.
- the "label": the data that we are trying to predict: in the case of supervised learning, we have the explanatory variable in the learning data.

Features and Label
the "features": we can measure them and it is from them that we will perform modeling and prediction.
the "label": the data that we are trying to predict: in the case of supervised learning, we have the explanatory variable in the learning data.

Page 9 :

Preparation of more complex data

- voice (Automatic Speech Recognition or Speech-To-Text) : Google cloud mode or Nuance solutions
- images : Imagemagick, OpenCV2

Preparation of more complex data
voice (Automatic Speech Recognition or Speech-To-Text) : Google cloud mode or Nuance solutions
images : Imagemagick, OpenCV2

Page 10 :

Page 11 :

Introduction

- ▷ data complexity: graphical analysis by data scientist
- ▷ highlight relationships between different dimensions
- ▷ quantify this relationship
- ▷ tool: linear regression

Page 12 :

NBA: size / weight relationship

- it is hinted that the weight must increase with size, but to what extent?
- Is it possible to predict the weight of a player who knows his size?

Page 13 :

Pandas

```
import pandas as pd
import matplotlib.pyplot as plt
from numpy.linalg import inv
import numpy as np

df = pd.read_csv('players_stats.csv')
height = df.dropna()['Height']
weight = df.dropna()['Weight']

plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')

plt.scatter(height, weight)
plt.show()
```


Page 14 :

https://www.kaggle.com/drgilermo/nba-players-stats-20142015.
Demo : players_scatter.py

NBA: size / weight relationship

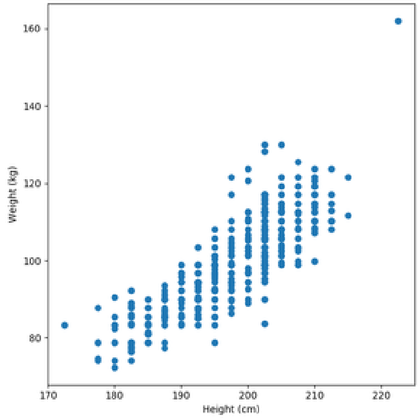
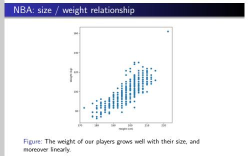


Figure: The weight of our players grows well with their size, and moreover linearly.



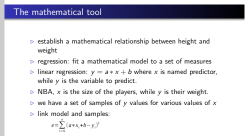
Page 15 :

The vertical lines are artificial, and hold to the resolution of the sizes which are rounded to 2.5 cm. As for the general trend, it seems that weight is linearly related to size.

The mathematical tool

- establish a mathematical relationship between height and weight
- regression: fit a mathematical model to a set of measures
- linear regression: $y = a * x + b$ where x is named predictor, while y is the variable to predict.
- NBA, x is the size of the players, while y is their weight.
- we have a set of samples of y values for various values of x
- link model and samples:

$$e=\sum_{i=0}^{n}(a*x_i+b-y_i)^2$$



Page 16 :

b is the value of the variable to predict for $x = 0$, which is the intercept.
If one represents the operation mentally, it is a question of sliding a rule vertically (which changes b), and of inclining it more or less (which changes a) until the points of our sampling seems to be regularly distributed on both sides of the rule.
In mathematical terms, we will derive the previous expression with respect to a and b, then we will seek to cancel this derivative. Indeed, remember your high school math class: a function reaches its extremum where its derivative vanishes.

The mathematical tool

We will work in matrix form:
$e=(X*A-Y)^T(X*A-Y)=(X*A-Y)^2$

Y is a column vector containing yi
X is a matrix consisting of two columns. The first contains the predictors xi while the second contains only 1.
A meanwhile, is a line vector containing $[ab]$. The derivative of e with respect to the parameters we wish to optimize, a and b, contained in A, is:

$$\frac{\partial e}{\partial A}=\frac{\partial(X*A-Y)^T}{\partial A}*(X*A-Y)=X^T(X*A-Y)$$

e reaches its minimum when this expression is null, that is:

$$X^T(X*A-Y)=0$$
$$X^T X*A=X^T Y$$
$$A=(X^T X)^{-1}X^T Y$$

Page 17 :

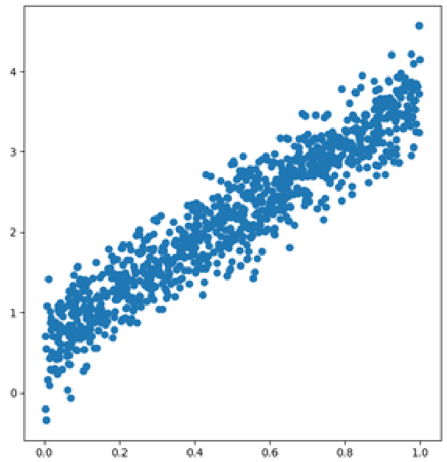
An example of linear data distributed according to a Gaussian

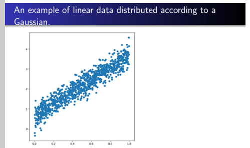
```
import numpy as np
importmath
importrandom
from numpy.linalg import inv
import matplotlib.pyplot as plt

nbSamples =1000

X = np.matrix([[random.random(), 1]for x inrange(nbSamples)])
Y = np.matrix([3*x[0].item(0)+ 0.666for x in X]).transpose()

Gnoise = np.random.normal(0.0,0.1,len(Y))
Ynoisy = np.matrix([Y[i].item(0)+ Gnoise[i]for i inrange(len(Y))]).transpose()

plt.scatter(np.asarray(X[:,0]), np.asarray(Ynoisy))
plt.show()
```


Page 18 :

neighborhood of an equation line $y = 3 * x + 0.666$.
We added a Gaussian noise, of zero mean and with a standard deviation of 0.1.

An example of linear data distributed according to a Gaussian.

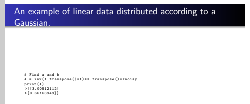




Page 19 :

An example of linear data distributed according to a Gaussian.

```
# Find a and b
A = inv(X.transpose()*X)*X.transpose()*Ynoisy
print(A)
>[[3.00512112]
>[0.66163949]]
```

Page 20 :

Let's see now if we fall back on our feet, and if by applying our formula, $a = 3$, $b = 0.666$ come out of the hat.

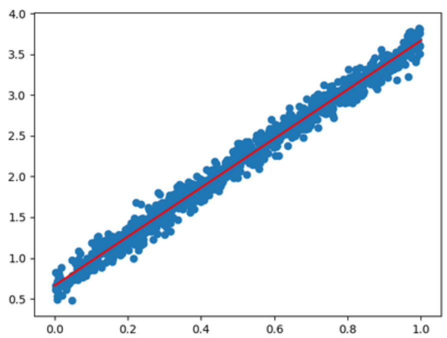
An example of linear data distributed according to a Gaussian.

```
x =[0,1]
y =[[x[0],1],[x[1],1]]*A
plt.scatter(np.asarray(X[:,0]), np.asarray(Ynoisy))
plt.plot(x, y, color='r')
plt.show()
```

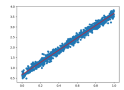

An example of linear data distributed according to a Gaussian.

Page 21 :

An example of linear Gaussian distributed data, and the associated linear regression.



An example of linear Gaussian distributed data, and the associated linear regression.



Page 22 :

NBA

```
import pandas as pd
import matplotlib.pyplot as plt
from numpy.linalg import inv
import numpy as np

df = pd.read_csv('players_stats.csv')
height = df.dropna()['Height']
weight = df.dropna()['Weight']

X = np.zeros((len(height),2))
X[:,0]= height
X[:,1]=1
Xm = np.matrix(X)
```


Page 23 :

reload the data of our set and put them in shape to apply our formula

Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Reduction of dimension
Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem

NBA

```
Y = np.matrix(weight.as_matrix())
A = inv(Xm.transpose()*Xm)*Xm.transpose()*Y.transpose()
```


2019-08-27
IML
└Graphic tool for DataScientist
└Tell me everything, and I'll tell you who you are
└NBA

Page 24 :

We thus obtain A, which contains the coefficients a, b. It remains to be seen if they provide a good approximation of our data set, by drawing the corresponding line.

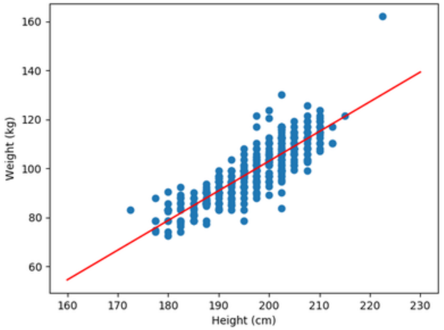
NBA

```
x =[160,230]
y =[[x[0],1],[x[1],1]]*A

plt.xlabel('Height␣(cm)')
plt.ylabel('Weight␣(kg)')
plt.scatter(height, weight)
plt.plot(x, y, color='r')
plt.show()
```

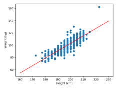

Page 25 :

we choose two random points, with abscissa x = 160 and x1 = 230 and we calculate their ordinates

NBA



The least squares method allows us to say that a player of 2.10m must weigh not far from 116 kilos



Page 26 :

Demo : players.py

Computer tools

This method works very well, but may become impractical if the number of columns of X becomes too large, the cost of an inversion being in the general case in $O(n3)$. The memory cost can also become prohibitive.

1. work with a subset representative of the total ensemble
2. develop an inversion algorithm
3. opt for an iterative approach, where we start from (a, b) to converge progressively to.

Page 27 :

Let's plot the error according to a

```
import autograd.numpy as np
from autograd import grad
import math
import random
from numpy.linalg import inv
import matplotlib.pyplot as plt

nbSamples =1000
X = np.matrix([[random.random(), 1]for x inrange(nbSamples)])
Y = np.matrix([3*x[0].item(0)+ 0.666for x in X]).transpose()

def error(X, Y, a):
   a = np.matrix([[a],[0.666]])
   e = X*a - Y
   return(e.transpose()* e).item(0)

def genError(X, Y):
   return lambda a : error(X, Y, a)

err = genError(X, Y)
xs = [x *6.0/ nbSamples for x inrange(nbSamples)]
e = [err(x)for x in xs]
plt.plot(xs, e)
```


Page 28 :

we capture X and Y to generate a function depending only on a.

```
grad_err = grad(err)

def newtonStep(f0, df, x0):
   df0 = df(x0)
   x1 = x0 - f0/ df0
   return x1

def newtonSolver(f, df, x0):
   count =0
   f0 = f(x0)
   whileTrue:
      x0 = newtonStep(f0, df, x0)
      print("iter␣%d␣:␣%f"%(count, x0))
      count +=1
      f0 = f(x0)
      if f0< 1e-6:
         break
   return x0

newtonSolver(err, grad_err,0)
```

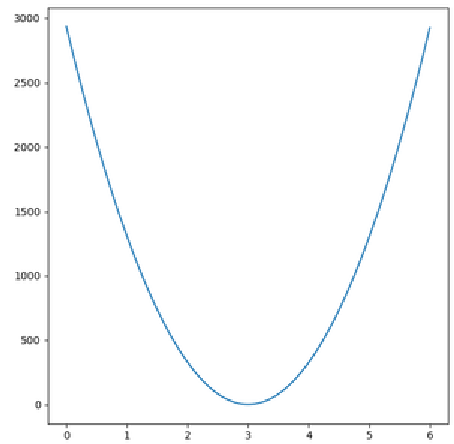

Page 29 :

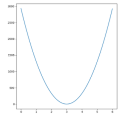
The idea is to start from a value of a, say $a = 0$. We calculate for this value $err(a)$ as well as the derivative of the error in: $\frac{\partial err}{\partial a}(a_0)$
to calculate the tangent, I use a little known method, which is the automatic differentiation, without us having to do the calculation of the derivative by hand. This is done by $grad()$, exported from the autograd module and generating a function of a giving the derivative in a.


```
iter 0 : 1.500000
iter 1 : 2.250000
iter 2 : 2.625000
iter 3 : 2.812500
iter 4 : 2.906250
iter 5 : 2.953125
iter 6 : 2.976562
iter 7 : 2.988281
iter 8 : 2.994141
iter 9 : 2.997070
iter 10 : 2.998535
iter 11 : 2.999268
iter 12 : 2.999634
iter 13 : 2.999817
iter 14 : 2.999908
iter 15 : 2.999954
```


Page 30 :

Demo : autodiff.py (ATTENTION MARCHE PAS)

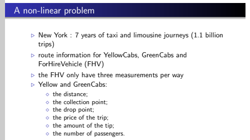




Page 31 :

A non-linear problem

- New York : 7 years of taxi and limousine journeys (1.1 billion trips)
- route information for YellowCabs, GreenCabs and ForHireVehicle (FHV)
- the FHV only have three measurements per way
- Yellow and GreenCabs:
  - the distance;
  - the collection point;
  - the drop point;
  - the price of the trip;
  - the amount of the tip;
  - the number of passengers.



Page 32 :

Data are available here: `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`

from JFK Airport to Manhattan's UpperEastSide.

```
import pandas as pd
from dateutil importparser
import matplotlib.pyplot as plt

cols =['PULocationID','DOLocationID','tpep_pickup_datetime','
      tpep_dropoff_datetime','trip_distance']

dfJ = pd.read_csv('yellow_tripdata_2017-01.csv', usecols=cols)
dfF = pd.read_csv('yellow_tripdata_2017-02.csv', usecols=cols)
dfM = pd.read_csv('yellow_tripdata_2017-03.csv', usecols=cols)
dfA = pd.read_csv('yellow_tripdata_2017-04.csv', usecols=cols)
dfMy = pd.read_csv('yellow_tripdata_2017-05.csv', usecols=cols)

df = dfJ.append(dfF).append(dfM).append(dfA).append(dfMy)

#236 manhattan upper east side
JFK_MU = df[(df['PULocationID']==132)&(df['DOLocationID']==236)]


JFK_MU.to_csv("JFKraw.csv", columns=cols)

pu = [parser.parse(dt)for dt in JFK_MU['tpep_pickup_datetime'].values]
do = [parser.parse(dt)for dt in JFK_MU['tpep_dropoff_datetime'].values]
dur = [(b -a).total_seconds()/ 3600.0for a, b inzip(pu, do)]
startTime = [dt.hour+ dt.minute/ 60.0for dt in pu]

plt.scatter(startTime, dur)
```

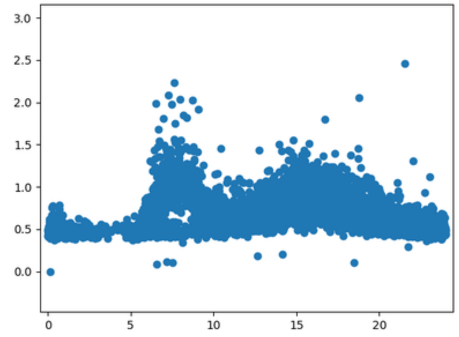

2019-08-27

IML
└Graphic tool for DataScientist
  └A non-linear problem
    └from JFK Airport to Manhattan's UpperEastSide.

Page 33 :

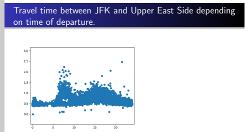
In our trip database, collection and deposit points are identified by an ID. A CSV file also online specifies that JFK has for ID 132, while Upper Manhattan is 236.

Travel time between JFK and Upper East Side depending on time of departure.



2019-08-27

IML
└Graphic tool for DataScientist
  └A non-linear problem
    └Travel time between JFK and Upper East Side depending on time of departure.



Page 34 :

Demo : JFK.py

Cleaning

- two peaks are around 7am and 4pm
- the peak of 7am is not always a real one
- It's a safe bet that these easy-going points are just weekend days (and probably holidays)

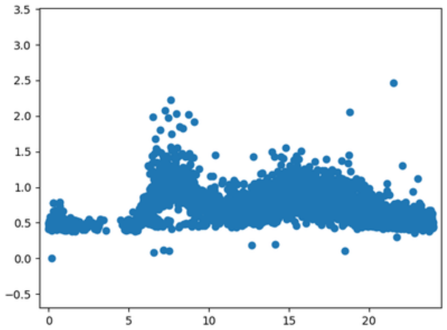
Cleaning
- two peaks are around 7am and 4pm
- the peak of 7am is not always a real one
- It's a safe bet that these easy-going points are just weekend days (and probably holidays)

Page 35 :


```
import pandas as pd
from dateutil importparser
import matplotlib.pyplot as plt

cols =['PULocationID','DOLocationID','tpep_pickup_datetime','
     tpep_dropoff_datetime','trip_distance']

dfJ = pd.read_csv('yellow_tripdata_2017-01.csv', usecols=cols)
dfF = pd.read_csv('yellow_tripdata_2017-02.csv', usecols=cols)
dfM = pd.read_csv('yellow_tripdata_2017-03.csv', usecols=cols)
dfA = pd.read_csv('yellow_tripdata_2017-04.csv', usecols=cols)
dfMy = pd.read_csv('yellow_tripdata_2017-05.csv', usecols=cols)

df = dfJ.append(dfF).append(dfM).append(dfA).append(dfMy)

JFK_MU = df[(df['PULocationID']==132)&(df['DOLocationID']==236)]

JFK_MU['weekday']= JFK_MU['tpep_pickup_datetime'].apply(lambda x :parser.parse(x
     ).weekday())

JFK_MU = JFK_MU[JFK_MU['weekday']<5]

pu = [parser.parse(dt)for dt in JFK_MU['tpep_pickup_datetime'].values]
do = [parser.parse(dt)for dt in JFK_MU['tpep_dropoff_datetime'].values]
dur = [(b -a).total_seconds()/ 3600.0for a, b inzip(pu, do)]
startTime = [dt.hour+ dt.minute/ 60.0for dt in pu]

plt.scatter(startTime, dur)
plt.show()
```


Page 36 :

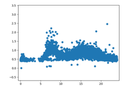
All aberrations (7am) are almost disappeared.



IML
└Graphic tool for DataScientist
 └A non-linear problem
  └All aberrations (7am) are almost disappeared.

2019-08-27

All aberrations (7am) are almost disappeared.



Page 37 :

Cerce

- example of data that clearly does not fit into a linear model
- use a linear regression: *splines*
  - interval $[xmin, xmax]$ on which the spline is defined is divided into n control points x_i.
  - At each of these points of control, we add a new line, which alters the pace of the curve defined at this point.
  - we build a series of functions, generally noted $I^i_{plus}(x)$ which are zero until x_i and the value is $x - x_i$ from x_i.

IML
└Graphic tool for DataScientist
 └A non-linear problem
  └Cerce

2019-08-27

Page 38 :

```
def Iplus(xi, x):
        if x>= xi: return x - xi
        else: return 0.0
```

This allows you to start a new line at each control point. Once this function has been defined, the calculation of the ordinate of this spline for a given abscissa is straightforward:

$$y=S(x)=\sum_{i=0}^{n-1} a_i I^i_{plus}(x)+b$$

Page 39 :

y is therefore expressed in linear form. We will therefore be able to reuse our linear regression by simply extending it to a dimension greater than 1. Concretely, the matrix X, which up to now consisted of two columns, will henceforth contain $n+1$. The last column, corresponding b is always filled with 1. The first n, for their part, contain the result of the application of *Iiplus*(x) on the abscissa of the current sample.


```
def splinify(xMin, xMax, step, x):
   a = [Iplus(xMin + i *step, x)for i inrange(int((xMax - xMin) / step))]
   a.reverse()
   return a +[1]

np.dot(x, A)
```


Page 40 :

xMin and xMax specify the bounds of the interval including all values of x, while step specifies the distance between each node of our spline. On an interval of [0, 1] and step = 0.1 the spline is based on 10 nodes.

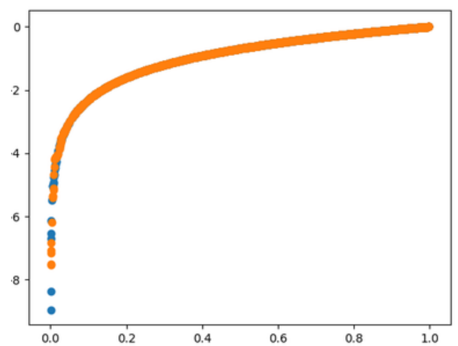
Case study

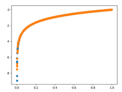
```
import numpy as np
import math
import random
from numpy.linalg import inv
import matplotlib.pyplot as plt

nbSamples =1000

X = np.matrix([[random.random(), 1]for x inrange(nbSamples)])
Y = np.matrix([math.log(x[0].item(0))for x in X]).transpose()

def Iplus(xi, x):
        if x>= xi: return x - xi
        else: return 0.0

def splinify(xMin, xMax, step, x):
        a = [Iplus(xMin + i *step, x)for i inrange(int((xMax - xMin) / step))]
        a.reverse()
        return a +[1]

Xm = np.matrix([splinify(0.0, 1.0, 0.01, x[0].item(0))for x in X])
A = inv(Xm.transpose()*Xm)*Xm.transpose()*Y
Yreg = np.matrix([[np.dot(x, A).item(0)]for x in Xm])

plt.scatter(np.asarray(X[:,0]), np.asarray(Y))
plt.scatter(np.asarray(X[:,0]), np.asarray(Yreg))
plt.show()
```


Page 41 :

We start with the construction of our dataset by filling in X with values drawn at random in [0, 1]. From there, we populate Y by applying the logarithm function.
Then, in the same way as in the case of the right, we fill *Xm* with the terms of our polynomial of degree 1, using the *splinify*() function. A is then calculated with the same formula, and we evaluate in the process our spline for all the abscissae of our sample using a simple scalar product. The result is stored in Yreg.
The last three lines generate the figure, where we see that our modeling by a linear spline of our test set works very well. It is possible to degrade the quality of this modeling by playing on the step parameter of the *splinify*() function.

Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Reduction of dimension
Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem





Page 42 :

The logarithm function on the interval [0, 1], in blue, and its modeling by the spline, in orange. The two overlap almost perfectly.
Demo : spline.py

JFK → Upper Manhattan

```
import numpy as np
import math
import random

from numpy.linalg import inv
import pandas as pd
from dateutil importparser
import matplotlib.pyplot as plt

cols =['PULocationID','DOLocationID','tpep_pickup_datetime','
      tpep_dropoff_datetime','trip_distance']

df = pd.read_csv('JFKraw.csv', usecols=cols)


#236 manhattan upper east side
JFK_MU = df[(df['PULocationID']==132)&(df['DOLocationID']==236)]

JFK_MU['weekday']= JFK_MU['tpep_pickup_datetime'].apply(lambda x :parser.parse(x
      ).weekday())

JFK_MU = JFK_MU[JFK_MU['weekday']<5]
```

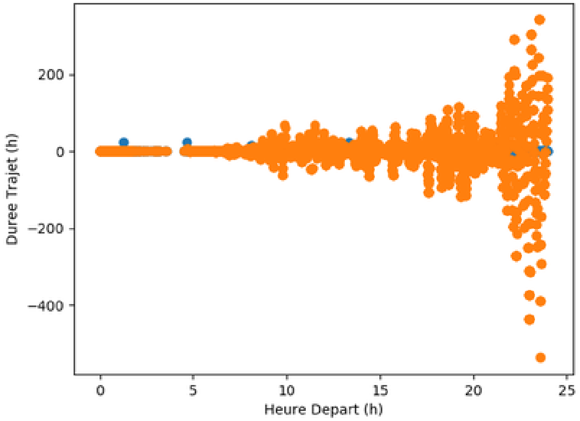

Page 43 :

JFK → Upper Manhattan

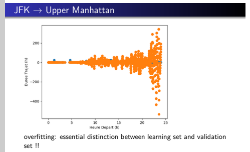
```
pu = [parser.parse(dt)for dt in JFK_MU['tpep_pickup_datetime'].values]
do = [parser.parse(dt)for dt in JFK_MU['tpep_dropoff_datetime'].values]
dur = [(b -a).total_seconds()/ 3600.0for a, b inzip(pu, do)]
startTime = [dt.hour+ dt.minute/ 60.0for dt in pu]

X = startTime
Y = dur

def Iplus(xi, x):
        if x>= xi:       return x - xi
        else:    return 0.0

def splinify(xMin, xMax, step, x):
        a = [Iplus(xMin + i *step, x)for i inrange(int((xMax - xMin) / step))]
        a.reverse()
        return a +[1]

Xm = np.matrix([[Iplus(0.5, x), Iplus(0, x), 1]for x in X])

# Find a and b
Xm = np.matrix([splinify(np.min(X), np.max(X), 0.1, x)for x in X])
A = inv(Xm.transpose()*Xm)*Xm.transpose()*np.matrix(Y).transpose()
Yreg = np.matrix([[np.dot(x, A).item(0)]for x in Xm])

plt.scatter(X, np.asarray(Y))
plt.scatter(X, np.asarray(Yreg))
plt.show()
```


Page 44 :

JFK → Upper Manhattan



overfitting: essential distinction between learning set and validation set !!

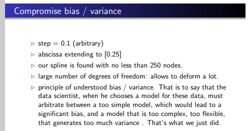


Page 45 :

Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Reduction of dimension
Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem

Compromise bias / variance

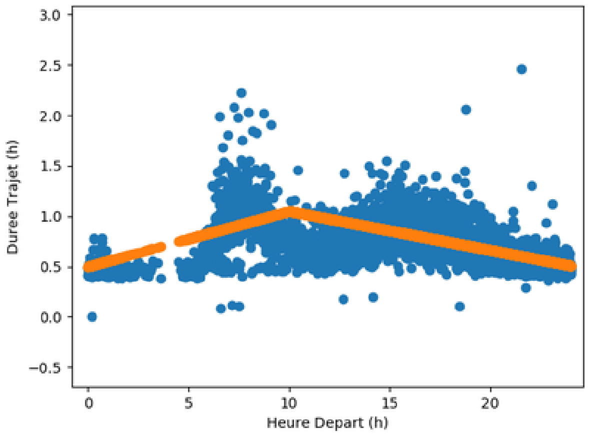
- ▷ step = 0.1 (arbitrary)
- ▷ abscissa extending to [0.25]
- ▷ our spline is found with no less than 250 nodes.
- ▷ large number of degrees of freedom: allows to deform a lot.
- ▷ principle of understood bias / variance. That is to say that the data scientist, when he chooses a model for these data, must arbitrate between a too simple model, which would lead to a significant bias, and a model that is too complex, too flexible, that generates too much variance . That's what we just did.

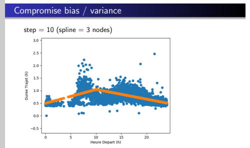


Page 46 :

Compromise bias / variance

step = 10 (spline = 3 nodes)





Page 47 :

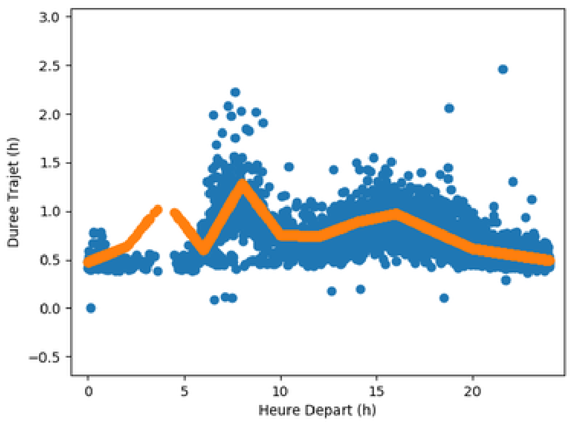
Demo : splineJFK.py splineJFKVar.py

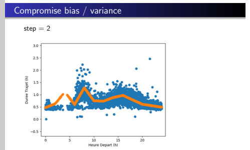
Dataset rules
Hyper Parameter tuning
Data preparation
Graphic tool for DataScientist
Reduction of dimension

Introduction
Tell me everything, and I'll tell you who you are
A non-linear problem

Compromise bias / variance

step = 2





Page 48 :

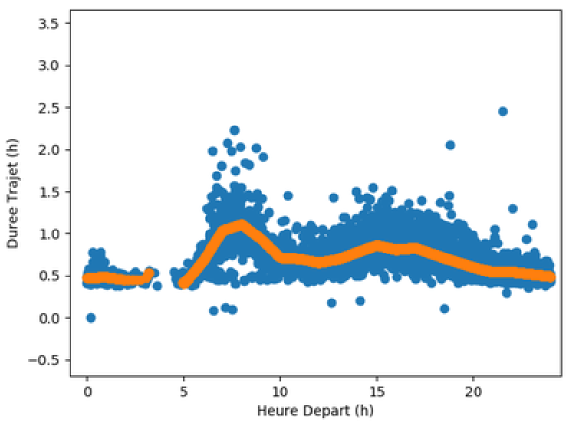
To choose the value of our step parameter, we proceeded by iteration, evaluating the quality of the obtained result. This method has the merit of making use of the expertise of the data scientist, but in order to obtain the best result, it is necessary to rely on more scientific criteria.
For this, it is necessary as detailed in the previous box, to have a set of learning and validation. Given these sets, it is then possible to calculate several metrics, to quantify how well the solution models the reality.
These metrics are often measures of errors between the actual values and their prediction using the model. We can cite for example the Mean Absolute Error (MAE) or the Root Mean Square Error (RMSE).

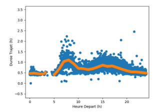
Aberrant points

modeling error around 4:20. This error is due to the presence of outliers, which are either measurement errors or extraordinary cases of plugs, failures, etc.

```
# Find a and b
Xm = np.matrix([splinify(np.min(X), np.max(X), 1.0, x)for x in X])
A = inv(Xm.transpose()*Xm)*Xm.transpose()*np.matrix(Y).transpose()
Yreg = np.matrix([[np.dot(x, A).item(0)]for x in Xm])

Yfiltered = [Y[i]for i in range(len(Y)) if ((math.fabs((Y[i]-Yreg[i]) / Y[i]) <
      0.9) and (Y[i] > 0.2) and(Y[i]<2.5))]
Xfiltered = [X[i]for i in range(len(Y)) if ((math.fabs((Y[i]-Yreg[i]) / Y[i]) <
      0.9) and (Y[i] > 0.2) and(Y[i]<2.5))]

Xm = np.matrix([splinify(np.min(Xfiltered), np.max(X), 1.0, x)for x in Xfiltered
      ])
A = inv(Xm.transpose()*Xm)*Xm.transpose()*np.matrix(Yfiltered).transpose()
Yfilteredreg = np.matrix([[np.dot(x, A).item(0)]for x in Xm])

plt.xlabel('Heure␣Depart␣(h)')
plt.ylabel('Duree␣Trajet␣(h)')
plt.scatter(X, np.asarray(Y))
plt.scatter(Xfiltered, np.asarray(Yfilteredreg))
plt.show()
```


Page 49 :





Page 50 :

The code was developed under Ubuntu, and is available on GitHub: https://github.com/kayhman/SmartLinearReg

Page 51 :

Introduction

- ▷ number of variables in a dataset becomes too large.
- ▷ precise analysis in each of the dimensions, it takes a set of measures quite gigantic
- ▷ difficult for a human to understand the relationships between so many variables.

Page 52 :

Example

3 different iris species, brings together four different measures:

- the length of the sepals;
- the width of the sepals;
- the length of the petals;
- the width of the petals

Page 53 :

Dimension 3

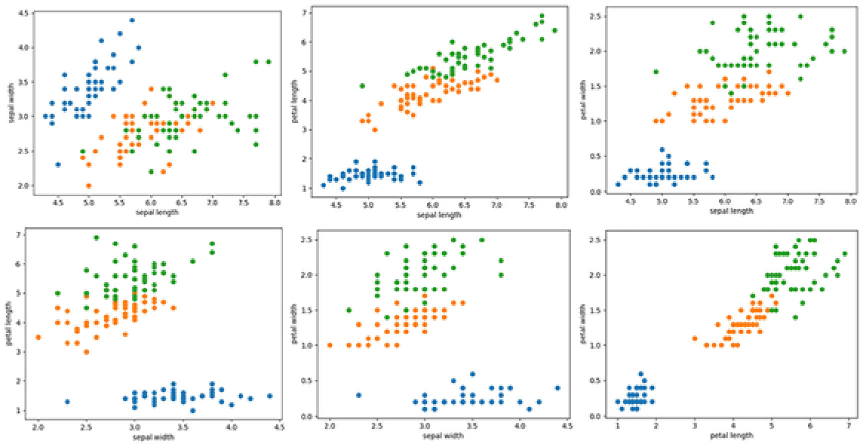
Comparisons two by two of the variables of the set

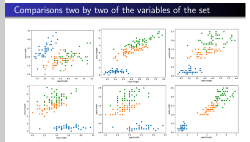
```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()

labels =['sepal␣length','sepal␣width','petal␣length','petal␣width']
for xx inrange(4):
        for yy inrange(4):
                if yy> xx:
                print xx, yy
                plt.xlabel(labels[xx])
                plt.ylabel(labels[yy])
                plt.scatter(iris.data[y==0][:, xx], iris.data[y==0][:,yy])
                plt.scatter(iris.data[y==1][:, xx], iris.data[y==1][:,yy])
                plt.scatter(iris.data[y==2][:, xx], iris.data[y==2][:,yy])
                plt.show()
```


Page 54 :

The dimension of this set, $n = 4$, being reduced, the number of graphs to be plotted does not exceed n (n-1) / 2 = 6. This remains analysable by a human, and it is also easy to generate automatically these analyzes

Comparisons two by two of the variables of the set

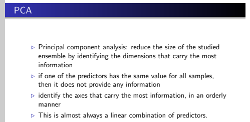




Page 55 :

PCA

- ▷ Principal component analysis: reduce the size of the studied ensemble by identifying the dimensions that carry the most information
- ▷ if one of the predictors has the same value for all samples, then it does not provide any information
- ▷ identify the axes that carry the most information, in an orderly manner
- ▷ This is almost always a linear combination of predictors.



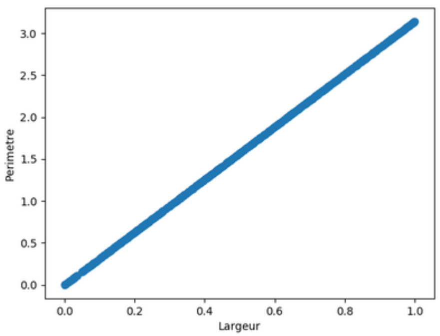
Page 56 :

a simple 2D case

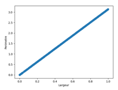
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np

nbSamples =1000
X0 = [random()for x inrange(nbSamples)]
X1 = [3.1416*x for x in X0]

plt.scatter(X0, X1)
plt.show()
```


Page 57 :

In this example, we collect the width of an object and its perimeter. Now, it turns out that all these objects are disks. Given their width d, which is none other than their diameter, it is easy to calculate their perimeter $p = d * \pi$.



The relationship between our two predictors is clearly linear. By identifying the relationship between them, it is possible to reduce our set to one dimension.



Page 58 :

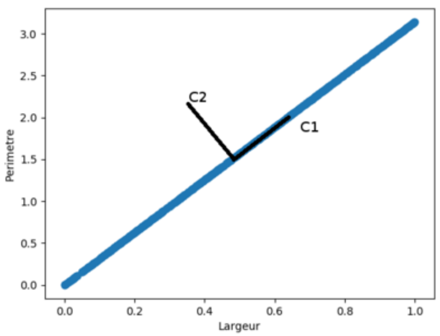
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np

nbSamples =1000
X0 = [random()for x inrange(nbSamples)]
X1 = [3.1416*x for x in X0]

X = np.matrix((X0, X1)).transpose()
pca = PCA(n_components=2)
pca.fit(X)
print(pca.components_[0])
print(pca.explained_variance_)
```


Page 59 :

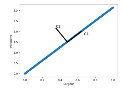

```
[[ 0.30331383 0.95289072]
[-0.95289072 0.30331383]]

[ 3.04402295e+01 2.11846137e-15]


>>> pca.components_[0][1]/ pca.components_[0][0]
3.1416000000000022

>>> np.dot(pca.components_[0], pca.components_[1])
0.0


>>> np.linalg.norm(pca.components_[0])
1.0

>>> np.linalg.norm(pca.components_[1])
1.0
```


Page 60 :

Note that if we do the ratio between the two coordinates of the first vector, we fall well on π.
Very important point too, if we have fun making the scalar product between these two vectors, we discover that they are orthogonal



The first axis, the one with the greatest eigenvalue, is enough to capture our whole



The first axis, the one with the greatest eigenvalue, is enough to capture our whole

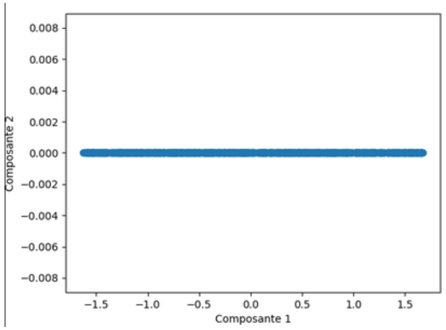
Page 61 :

the matrix presented above can be considered, in the case 2D at least, as a rotation matrix.

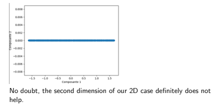
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np

nbSamples =1000
X0 = [random()for x inrange(nbSamples)]
X1 = [3.1416*x for x in X0]

X = np.matrix((X0, X1)).transpose()
pca = PCA(n_components=2)
X_r = pca.fit(X).transform(X)
print(pca.components_)
print(pca.singular_values_)

plt.scatter(X_r[:,0], X_r[:,1])
plt.xlabel("Composante␣1")
plt.ylabel("Composante␣2")
plt.show()
```


the matrix presented above can be considered, in the case 2D at least, as a rotation matrix.

Page 62 :



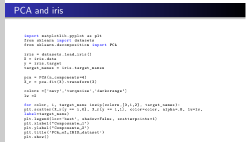
No doubt, the second dimension of our 2D case definitely does not help.



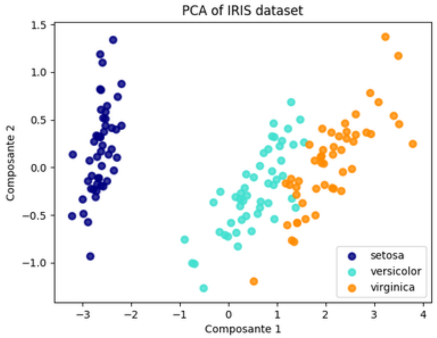
Page 63 :

PCA and iris

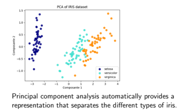
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA

iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

pca = PCA(n_components=4)
X_r = pca.fit(X).transform(X)

colors =['navy','turquoise','darkorange']
lw =2

for color, i, target_name inzip(colors,[0,1,2], target_names):
plt.scatter(X_r[y == i,0], X_r[y == i,1], color=color, alpha=.8, lw=lw,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.xlabel("Composante␣1")
plt.ylabel("Composante␣2")
plt.title('PCA␣of␣IRIS␣dataset')
plt.show()
```

Page 64 :



Principal component analysis automatically provides a representation that separates the different types of iris.

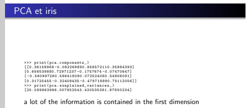


Page 65 :

PCA et iris

```
>>> print(pca.components_)
[[0.36158968-0.082268890.856572110.35884393]
[0.656539880.72971237-0.1757674-0.07470647]
[-0.580997280.596418090.072524080.54906091]
[0.31725455-0.32409435-0.479718990.75112056]]
>>> print(pca.exaplained_variances_)
[25.089863986.007852543.420535381.87850234]
```

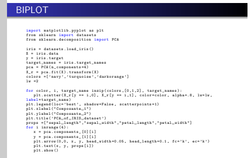
a lot of the information is contained in the first dimension



Page 66 :

BIPLOT

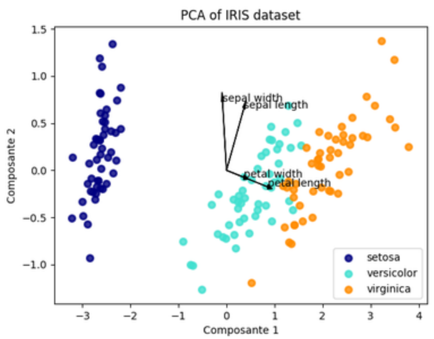
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA

iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
pca = PCA(n_components=4)
X_r = pca.fit(X).transform(X)
colors =['navy','turquoise','darkorange']
lw =2

for color, i, target_name inzip(colors,[0,1,2], target_names):
   plt.scatter(X_r[y == i,0], X_r[y == i,1], color=color, alpha=.8, lw=lw,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.xlabel("Composante␣1")
plt.ylabel("Composante␣2")
plt.title('PCA␣of␣IRIS␣dataset')
props =["sepal␣length","sepal␣width","petal␣length","petal␣width"]
for i inrange(4):
   x = pca.components_[0][i]
   y = pca.components_[1][i]
   plt.arrow(0,0, x, y, head_width=0.05, head_length=0.1, fc='k', ec='k')
   plt.text(x, y, props[i])
   plt.show()
```

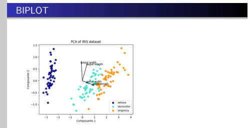



Page 67 :

biplot: display on a 2D graph the maximum of information on all dimensions of the problem. This tool allows to display several dimensions in only two dimensions.
For this, we start from the projection of our data in the 2D plane described by the first two main components of our analysis. Then we display in the form of 2D vectors the initial dimensions of the problem considered.

BIPLOT





Page 68 :

BIPLOT

```
# moyenne de la longueur du petale - setosa
np.std(iris.data[y==0][:,2])
# -> 0.17176728442867112
# moyenne de la longueur du petale - versicolor
np.std(iris.data[y==1][:,2])
# -> 0.4651881339845203
# moyenne de la longueur du petale - virginica
np.std(iris.data[y==2][:,2])
# -> 0.54634787452684397
```

The length of the petals of the setosa is clearly smaller than for versicolor and virginica.

Page 69 :

BIPLOT

```
# moyenne de la Largeur du sepale - setosa
np.std(iris.data[y==0][:,1])
# -> 0.37719490982779713
# moyenne de la Largeur du sepale - versicolor
np.std(iris.data[y==1][:,1])
# -> 0.31064449134018135
# moyenne de la Largeur du sepale - virginica
np.std(iris.data[y==2][:,1])
# -> 0.31925538366643091
```

In this case, the values are very close: it is not a good parameter to distinguish the different species.

Page 70 :

Normalization

- ▷ Principal component analysis provides a series of analysis axes that capture the variability of the data studied, in descending order.
- ▷ The data thus spread widely along the first axis, while they are fairly condensed around the last one.
- ▷ If the data are not normalized, that is, if they have not been reworked in such a way that their averages are zero, and their standard deviations are 1.0, then the analysis may be skewed by differences in units used.

Page 71 :

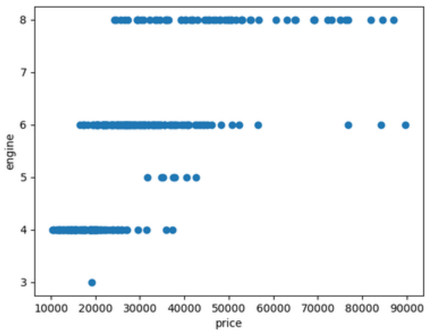
Raw data

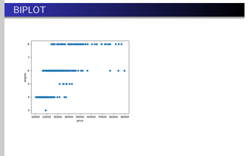
```
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt

cols =['price','invoice␣price','dealer␣cost','engine','cylinders','horsepower','
     weight','wheel','length','width','cm␣per␣gallons','hm␣per␣gallons']
df = pd.read_csv('04cars.dat.txt', usecols=cols)

pe = df[df['price']>1000][df['engine']<10][['price','engine']]
plt.scatter(pe['price'], pe['engine'])
plt.xlabel('price')
plt.ylabel('engine')
plt.show()
```


Page 72 :

To illustrate the importance of standardization, we will focus on two variables: price and engine capacity. The price is given in dollars, while the cubic capacity is in liters. Without normalization, we compare data that have very different scales, which bias the result.

BIPLOT





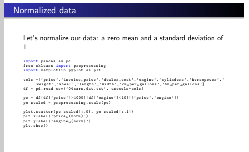
Page 73 :

These data, raw, give the impression that cars are distinguished mainly by their price, since the latter varies from 10,000 to 90,000, while the cubic capacity is confined to the interval [3, 8].

Normalized data

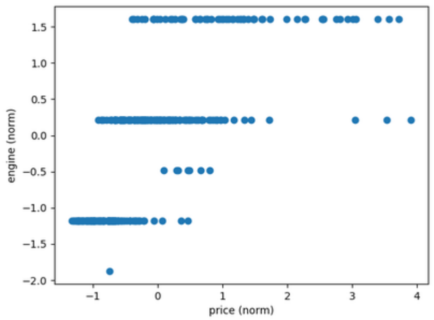
Let's normalize our data: a zero mean and a standard deviation of 1

```
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt

cols =['price','invoice␣price','dealer␣cost','engine','cylinders','horsepower','
     weight','wheel','length','width','cm␣per␣gallons','hm␣per␣gallons']
df = pd.read_csv('04cars.dat.txt', usecols=cols)

pe = df[df['price']>1000][df['engine']<10][['price','engine']]
pe_scaled = preprocessing.scale(pe)

plot.scatter(pe_scaled[:,0], pe_scaled[:,1])
plt.xlabel('price␣(norm)')
plt.ylabel('engine␣(norm)')
plt.show()
```

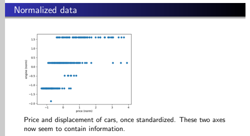



Page 74 :

Normalized data



Price and displacement of cars, once standardized. These two axes now seem to contain information.



Page 75 :

In essence, standardization makes the data dimensionless, allowing them to be compared without risk.

PCA

```
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt

cols =['price','invoice␣price','dealer␣cost','engine','cylinders','horsepower','
     weight','wheel','length','width','cm␣per␣gallons','hm␣per␣gallons']
df = pd.read_csv('04cars.dat.txt', usecols=cols)
X_scaled = preprocessing.scale(df[cols].replace('*',float('nan')).dropna().
     as_matrix())
pe = df[df['price']>1000][df['engine']<10][['price','engine']]
pe_scaled = preprocessing.scale(pe)
pca = PCA(n_components=2)

# raw data
pca.fit(pe)
print(pca.explained_variance_)

# normalized data
pca.fit(pe_scaled)
print(pca.explained_variance_)
```


Page 76 :

PCA

```
# raw data
[ 2.32179369e+08 1.08822536e+00]

# normalized data
[ 1.69570742 0.31072345]
```


Page 77 :

The principal component analysis therefore makes the same observations as we do. In the first case, that of raw data, it is fooled by the difference in unity between price and engine capacity, and unduly believes that most of the information is carried by a single axis.
In the second case, the analysis is more measured because the variance of the data is not clearly driven by a single axis.

Correlation matrix

- in Python: `C = pe_scaled.transpose()*pe_scaled`
- This matrix contains valuable information: each element *Cij* quantifies the relationship between the variables *i* and *j*. If *Cij* is positive, then when *i* grows, then *j* as well. If, on the other hand, it is negative, then *j* decreases while *i* increases.
- In the case where *Cij* is zero, and that's where it gets interesting, then the variables *i* and *j* are not correlated. They therefore vary independently of each other.
- The particular case where the matrix is diagnonal is therefore particularly sympathetic, because in this case, the variables are all independent of each other.

Page 78 :

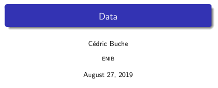
It is precisely in this particular case that the analysis in principal components is reduced. Indeed, the correlation matrix C is symmetric, square, and contains real values. It is therefore possible to diagonalize it, that is to say to find a change of reference making the directions orthogonal.
This is precisely what principal component analysis does, by calculating the eigenvectors and the eigenvalues of C. The eigenvectors are then the new axes of analysis, while the eigenvalues make it possible to classify these axes by variance. decreasing.
Note that for reasons of performance, we do not always proceed directly to the diagonalization of C, but rather to the decomposition of singular values.

Data

Cédric Buche

ENIB

August 27, 2019



Page 79 :