Framework and Data

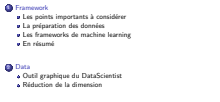
IML

Cédric Buche

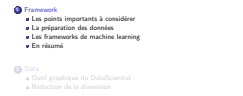
ENIB

1er juin 2018

Page 1 :



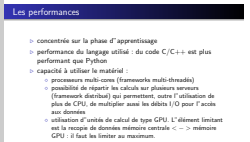
Page 2 :



Page 3 :

Les performances

- ▷ concentrée sur la phase d''apprentissage
- ▷ performance du langage utilisé : du code C/C++ est plus performant que Python
- ▷ capacité à utiliser le matériel :
  - ◇ processeurs multi-cores (frameworks multi-threadés)
  - ◇ possibilité de répartir les calculs sur plusieurs serveurs (framework distribué) qui permettent, outre l''utilisation de plus de CPU, de multiplier aussi les débits I/O pour l''accès aux données
  - ◇ utilisation d''unités de calcul de type GPU. L''élément limitant est la recopie de données mémoire centrale $< - >$ mémoire GPU : il faut les limiter au maximum.



Page 4 :

- Si vous vous attaquez à un problème nécessitant le traitement important de données (plusieurs dizaines de Gio ou plus) pour l''apprentissage, choisissez un framework qui gère la distribution, c''est le seul moyen de dépasser les limites matérielles. Pour le deep learning, tous les frameworks modernes le proposent, au vu de la nature même des calculs nécessaires. Un modèle pourra ne pas être calculable dans la pratique s''il faut des jours ou des semaines (ou plus !) pour le mener à terme sur une seule machine.
- Une carte de type nVidia GTX 1080 comporte 2 560 cudacores (les éléments de calcul), représentant pour la partie calcul numérique une centaine de cores d''un CPU moderne. Le GPU ne fonctionnant pas en mémoire centrale, il faut d''abord recopier les données en mémoire GPU (8 Gio pour le modèle cité) avant de lancer les calculs. Des opérations nécessitant la recopie de données à chaque opération n''amènent pas de gain réel. C''est pour cela que les frameworks orientés $\ll$ images $\gg$ proposent des paramètres appelés $\ll$ batch size $\gg$ qui permettent de traiter plusieurs images à la fois, en les recopiant d''abord par blocs et non pas image par image.

Features et Label

- les « features », ou variables explicatives en français : on peut les mesurer et c"est à partir d"elles que l"on va effectuer la modélisation et la prédiction.
- le « label » ou variable à expliquer : la donnée que l"on cherche à prédire : dans le cas de l"apprentissage supervisé, on dispose de la variable explicative dans les données d"apprentissage.

Features et Label

- les « features », ou variables explicatives en français : on peut les mesurer et c"est à partir d"elles que l"on va effectuer la modélisation et la prédiction.
- le « label » ou variable à expliquer : la donnée que l"on cherche à prédire : dans le cas de l"apprentissage supervisé, on dispose de la variable explicative dans les données d"apprentissage.

Page 5 :

Apache Spark : lecture de données, transformation et jointure

```
01: # reads input and apply processing for non-empty lines
02: # process_data is a previously defined function returning processed data as
     list
03: # sc is the spark context initialized when you invoke pyspark
04: input_file = sc.textfile("/my/rep/to/file.txt") \
05:     .filter(lambda x : x and x != u"") \
06:     .map ( process_data ) \
07:     .map(lambda x : (x[0], x) ) #creates key from first list item
08: #reads the second file.txt, which is already processed and saved as "key;
     data1;data2␣;..."
09: other_file = sc.textFile("/my/rep/to/second/file.txt")
10: #joining the two files on the key (first item of tuple)
11: result_file = input_file.join(other_file \
12:                               .map(lambda x : x.split(";") ) \
13:                               .map(lambda x : (x[0], x) ) #creates key
```


Apache Spark : lecture de données, transformation et jointure

Page 6 :

- La force de Spark est d"exécuter les traitements en distribué : les lignes 4 et 9 déclarent des fichiers de données en mode distribué sur lesquels on va appliquer des traitements, parallélisés par Spark. En complément, Spark ne va pas exécuter les traitements séquentiellement, mais va construire le plan d"exécution avant d"invoquer les traitements : les lignes 5,6 et 7 d"une part et 12 et 13 d"autre part seront exécutées en une seule fois. Pas la peine d"optimiser le code pour limiter les invocations de traitements distribués : il suffit d"utiliser l"API fonctionnelle (filter, map, ...) sur les données chargées par Spark.
- Si on ajoute que l"installation de Spark est très facile (un fichier tar à extraire), qu"il propose pour développer Java, Scala et Python (et R comme front-end), c"est un outil à recommander. Spark sera capable d"utiliser tous les cores de votre machine si vous utilisez son API. Le passage en mode distribué multi-machines ne nécessite pas de modification de code, mais il faudra disposer d"un moyen de stocker les données en mode distribué (comme Hadoop HDFS par exemple).

Préparation des données plus complexes

- voix (Automatic Speech Recognition ou Speech-To-Text) : Google en mode Cloud ou les solutions de Nuance pour un logiciel à installer
- images : Imagemagick, OpenCV2

Préparation des données plus complexes
- voix (Automatic Speech Recognition ou Speech-To-Text) : Google en mode Cloud ou les solutions de Nuance pour un logiciel à installer
- images : Imagemagick, OpenCV2

Page 7 :

Spark : mllib

```
01: #data is a dataframe holding the trainig set
02: #features are in a column named features - which is default name
03: #label is in a column named label - which is default name
04:
05: (trainingData, testData) = data.randomSplit([0.7, 0.3])
06: #keeping 30 % of data for validation
07: rf = RandomForestClassifier(numTrees=10)
08: model = rf.fit(trainingData) #voila !
09:
10: predictions = model.transform(testData)#to check quality of model
```

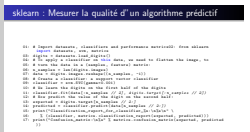

Spark : mllib

Page 8 :

La ligne 5 extrait un jeu de données (30%) pour la validation de l''algorithme du reste des données (70%) qui seront utilisées pour l''apprentissage. C'est la technique de base pour éviter l''over-fitting : on mesure la qualité de l'algorithme sur des données qui n''ont pas servi à l'apprentissage. La ligne 7 crée le modèle, et la ligne 8 effectue l'apprentissage d''une forêt aléatoire. La ligne 10 calcule la prédiction sur le jeu de données de validation pour vérifier la pertinence du modèle appris.

sklearn : Mesurer la qualité d"un algorithme prédictif

```
01: # Import datasets, classifiers and performance metrics02: from sklearn
     import datasets, svm, metrics
03: digits = datasets.load_digits()
04: # To apply a classifier on this data, we need to flatten the image, to
05: # turn the data in a (samples, feature) matrix:
06: n_samples = len(digits.images)
07: data = digits.images.reshape((n_samples, -1))
08: # Create a classifier: a support vector classifier
09: classifier = svm.SVC(gamma=0.001)
10: # We learn the digits on the first half of the digits
11: classifier.fit(data[:n_samples // 2], digits.target[:n_samples // 2])
12: # Now predict the value of the digit on the second half:
13: expected = digits.target[n_samples // 2:]
14: predicted = classifier.predict(data[n_samples // 2:])
15: print("Classification report for classifier %s:\n%s\n" \
16:     % (classifier, metrics.classification_report(expected, predicted)))
17: print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted
     ))
```

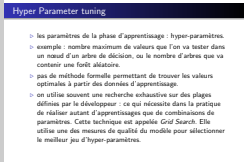



Page 9 :

Les lignes 15 et 16 affichent la synthèse des mesures mathématiques proposées par sklearn pour les 10 classes à prédire (les 10 chiffres possibles reconnus) ; la ligne 17 affiche la matrice de confusion, c"est-à-dire un tableau reprenant les mesures qui résument la qualité du modèle pour ces 10 classes. L"intérêt de ce tableau est qu"il montre très visuellement la proportion de bonnes prédictions, et la répartition par mauvaise prédiction effectuée.

Hyper Parameter tuning

- les paramètres de la phase d'apprentissage : hyper-paramètres.
- exemple : nombre maximum de valeurs que l'on va tester dans un nœud d'un arbre de décision, ou le nombre d'arbres que va contenir une forêt aléatoire.
- pas de méthode formelle permettant de trouver les valeurs optimales à partir des données d'apprentissage.
- on utilise souvent une recherche exhaustive sur des plages définies par le développeur : ce qui nécessite dans la pratique de réaliser autant d'apprentissages que de combinaisons de paramètres. Cette technique est appelée *Grid Search*. Elle utilise une des mesures de qualité du modèle pour sélectionner le meilleur jeu d'hyper-paramètres.



Page 10 :

- tester ces outils en tant qu'individuel ou dans une petite structure : sklearn (machine learning) + spark (traitement des données)
- monter un projet d'entreprise visant le traitement de données à forts volumes et avez la possibilité de déployer une plateforme de traitement de type multi-serveurs : déploiement Hadoop + Spark permettra la préparation distribuée des données, et la mllib de Spark fournit de plus en plus d'algorithmes de machine learning

- tester ces outils en tant qu'individuel ou dans une petite structure : sklearn (machine learning) + spark (traitement des données)
- monter un projet d'entreprise visant le traitement de données à forts volumes et avez la possibilité de déployer une plateforme de traitement de type multi-serveurs : déploiement Hadoop + Spark permettra la préparation distribuée des données, et la mllib de Spark fournit de plus en plus d'algorithmes de machine learning

Page 11 :

Pour la préparation et le traitement des données, vous pouvez utiliser Spark pour bénéficier du parallélisme, et utiliser tous les cores de votre machine si le volume de données est important. Il suffira d'écrire les données traitées avant de les lire côté framework machine learning pour la partie apprentissage proprement dite.

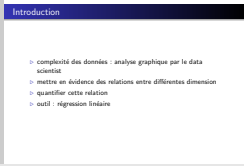
Framework
Les points importants à considérer
La préparation des données
Les frameworks de machine learning
En résumé
Data
Outil graphique du DataScientist
Réduction de la dimension

Page 12 :

Page 13 :

Introduction

- complexité des données : analyse graphique par le data scientist
- mettre en évidence des relations entre différentes dimension
- quantifier cette relation
- outil : régression linéaire



Page 14 :

NBA : relation taille / poids

- on pressent que le poids doit croître avec la taille, mais selon quelle mesure ?
- Est-il possible de prédire le poids d'un joueur connaissant sa taille ?

NBA : relation taille / poids
- on pressent que le poids doit croître avec la taille, mais selon quelle mesure ?
- Est-il possible de prédire le poids d'un joueur connaissant sa taille ?

Page 15 :

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()
```


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 16 :

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

NBA : relation taille / poids

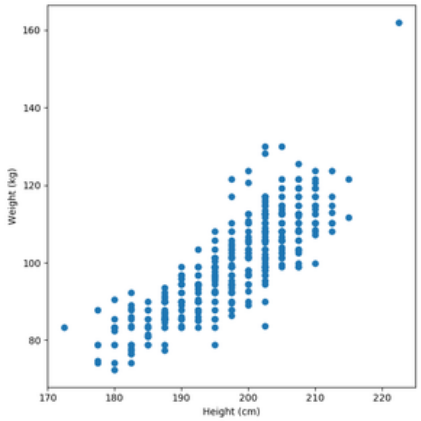
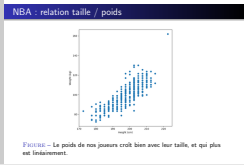


FIGURE – Le poids de nos joueurs croît bien avec leur taille, et qui plus est linéairement.



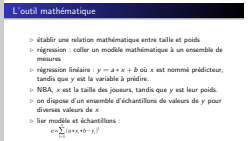
Page 17 :

Les lignes verticales sont artificielles, et tiennent à la résolution des tailles qui sont arrondies à 2.5 cm près. Quant à la tendance générale, il semble bien que le poids soit lié linéairement à la taille.

L'outil mathématique

- établir une relation mathématique entre taille et poids
- régression : coller un modèle mathématique à un ensemble de mesures
- régression linéaire : $y = a * x + b$ où x est nommé prédicteur, tandis que y est la variable à prédire.
- NBA, x est la taille des joueurs, tandis que y est leur poids.
- on dispose d'un ensemble d'échantillons de valeurs de y pour diverses valeurs de x
- lier modèle et échantillons :

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



Page 18 :

b est la valeur de la variable à prédire pour $x = 0$, soit l'ordonnée à l'origine.
Si l'on se représente mentalement l'opération, il s'agit de faire glisser une règle verticalement (ce qui change b), et de l'incliner plus ou moins (ce qui change a) jusqu'à ce que les points de notre échantillonnage semblent répartis régulièrement de part et d'autre de la règle.
En terme mathématique, nous allons dériver l'expression précédente par rapport à a et b, puis nous allons chercher à annuler cette dérivée. En effet, rappelez-vous vos cours de math du lycée : une fonction atteint ses extremum là où sa dérivée s'annule.

L'outil mathématique

Nous allons travailler sous forme matricielle :

$$e=(X*A-Y)^T(X*A-Y)=(X*A-Y)^2$$

Y est un vecteur colonne contenant les yi

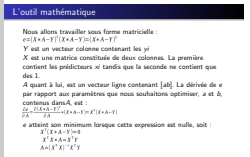
X est une matrice constituée de deux colonnes. La première contient les prédicteurs xi tandis que la seconde ne contient que des 1.

A quant à lui, est un vecteur ligne contenant $[ab]$. La dérivée de e par rapport aux paramètres que nous souhaitons optimiser, a et b, contenus dans$A$, est :

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

e atteint son minimum lorsque cette expression est nulle, soit :

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



Page 19 :

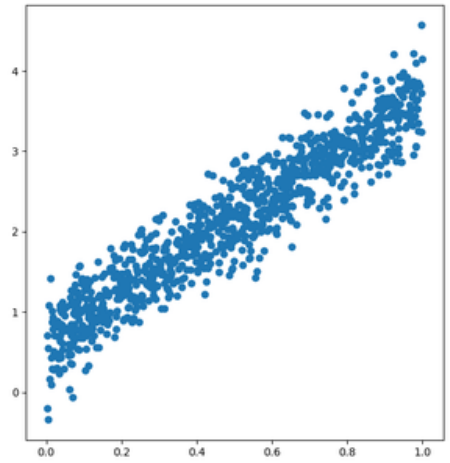
Un exemple de données linéaires distribuées selon une gaussienne.

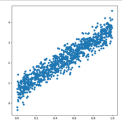
```
import numpy as np
importmath
importrandom
from numpy.linalg import inv
import matplotlib.pyplot as plt

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 20 :

voisinage d'une droite d'équation $y = 3 * x + 0.666$.
Nous avons ajouté un bruit gaussien, de moyenne nulle et avec une déviation standard de 0.1.

Un exemple de données linéaires distribuées selon une gaussienne.

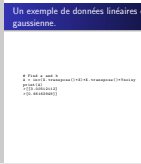




Page 21 :

Un exemple de données linéaires distribuées selon une gaussienne.

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

Page 22 :

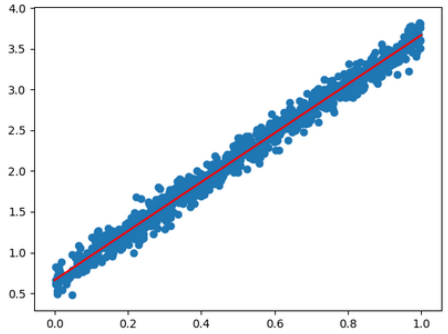
Voyons maintenant si nous retombons sur nos pieds, et si en appliquant notre formule, a = 3, b = 0.666 sortent du chapeau.

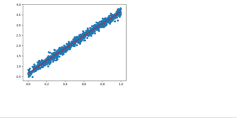
Un exemple de données linéaires distribuées selon une gaussienne.

```
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()
```


Page 23 :

Un exemple de données linéaires distribuées selon une gaussienne, et la régression linéaire associée.





Page 24 :

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 25 :

recharger les données de notre ensemble et les mettre en forme pour appliquer notre formule

NBA

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


Page 26 :

Nous obtenons ainsi A, qui contient les coefficients a, b. Reste à vérifier s'ils fournissent une bonne approximation de notre ensemble de données, en traçant la droite correspondante.

NBA

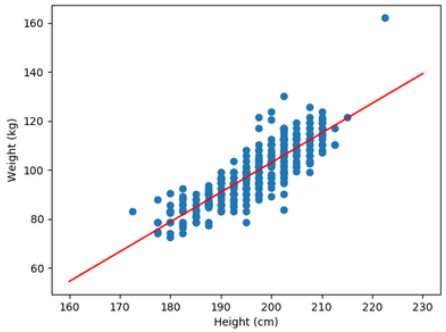
```
x =[160,230]
y =[[x[0],1],[x[1],1]]*A

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


NBA

Page 27 :

nous choisissons deux points au hasard, d'abscisse x = 160 et x1= 230 et nous calculons leurs ordonnées

NBA



La méthode des moindres carrés nous permet d'affirmer qu'un joueur de 2,10m doit peser pas loin de 116 kilos

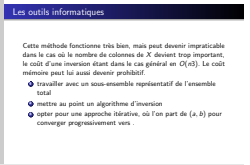
NBA
La méthode des moindres carrés nous permet d'affirmer qu'un joueur de 2,10m doit peser pas loin de 116 kilos

Page 28 :

Les outils informatiques

Cette méthode fonctionne très bien, mais peut devenir impraticable dans le cas où le nombre de colonnes de X devient trop important, le coût d'une inversion étant dans le cas général en $O(n3)$. Le coût mémoire peut lui aussi devenir prohibitif.

1. travailler avec un sous-ensemble représentatif de l'ensemble total
2. mettre au point un algorithme d'inversion
3. opter pour une approche itérative, où l'on part de (a, b) pour converger progressivement vers .



Page 29 :

traçons l'erreur en fonction de 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 30 :

on capture X et Y pour générer une fonction ne dépendant que de a.

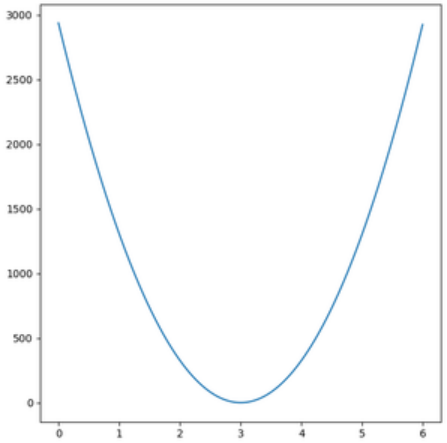
```
grad_err = grad(err)

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)
```

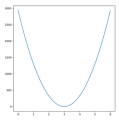

Page 31 :

L'idée est de partir d'une valeur de a, mettons a = 0. On calcule pour cette valeur err(a) ainsi que la dérivée de l'erreur en : $\frac{\partial err}{\partial a}(a_0)$
pour calculer la tangente, j'utilise une méthode peu connue, qui est la différentiation automatique, sans que nous ayons besoin de faire le calcul de la dérivée à la main. C'est ce que fait grad(), exporté du module autograd et qui génère une fonction de a donnant la dérivée en 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 32 :

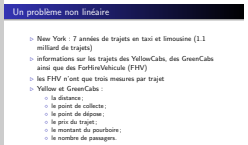




Page 33 :

Un problème non linéaire

- New York : 7 années de trajets en taxi et limousine (1.1 milliard de trajets)
- informations sur les trajets des YellowCabs, des GreenCabs ainsi que des ForHireVehicule (FHV)
- les FHV n'ont que trois mesures par trajet
- Yellow et GreenCabs :
  - la distance ;
  - le point de collecte ;
  - le point de dépose ;
  - le prix du trajet ;
  - le montant du pourboire ;
  - le nombre de passagers.



Page 34 :

Les données sont accessibles ici : http ://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

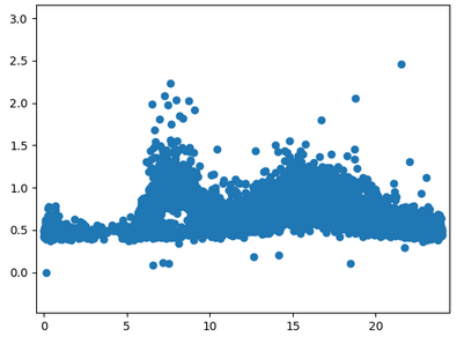
de l'aéroport JFK au quartier de l'UpperEastSide de Manhattan.

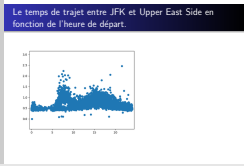
```
import pandas as pd
from dateutil importparser
import matplotlib.pyplot as plt

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)]

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 35 :

Dans notre base de trajets, les points de collecte et de dépôt sont identifiés par une ID. Un fichier CSV également en ligne précise que JFK a pour ID 132, tandis que Upper Manhattan est le 236.

Le temps de trajet entre JFK et Upper East Side en fonction de l'heure de départ.

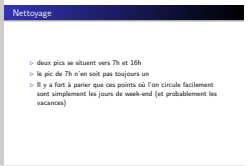




Page 36 :

Nettoyage

- deux pics se situent vers 7h et 16h
- le pic de 7h n'en soit pas toujours un
- Il y a fort à parier que ces points où l'on circule facilement sont simplement les jours de week-end (et probablement les vacances)

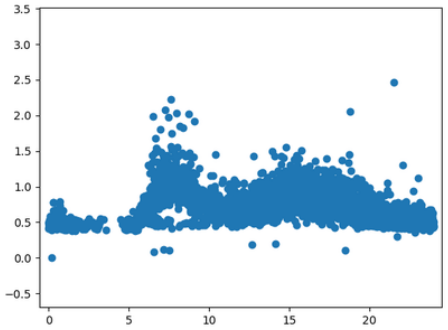


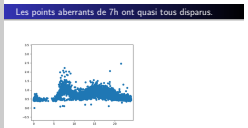
Page 37 :


```
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 38 :

Les points aberrants de 7h ont quasi tous disparus.

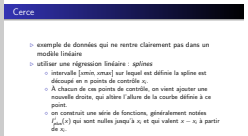




Page 39 :

Cerce

- exemple de données qui ne rentre clairement pas dans un modèle linéaire
- utiliser une régression linéaire : *splines*
  - intervalle $[xmin, xmax]$ sur lequel est définie la spline est découpé en n points de contrôle x_i.
  - À chacun de ces points de contrôle, on vient ajouter une nouvelle droite, qui altère l'allure de la courbe définie à ce point.
  - on construit une série de fonctions, généralement notées $I^i_{plus}(x)$ qui sont nulles jusqu'à x_i et qui valent $x - x_i$ à partir de x_i.



Page 40 :

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

Cela permet de faire commencer une nouvelle droite à chaque point de contrôle. Une fois cette fonction définie, le calcul de l'ordonnée de cette spline pour une abscisse donnée se fait sans détour :

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

Page 41 :

y s'exprime donc sous forme linéaire. Nous allons donc pouvoir réutiliser notre régression linéaire en l'étendant simplement à une dimension supérieure à 1. Concrètement, la matrice X, qui était jusqu'à présent constituée de deux colonnes, va dorénavant en contenir n + 1. La dernière colonne, correspondant au b est toujours remplie de 1. Les n premières contiennent pour leur part le résultat de l'application de liplus(x) sur l'abscisse de l'échantillon courant.


```
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 42 :

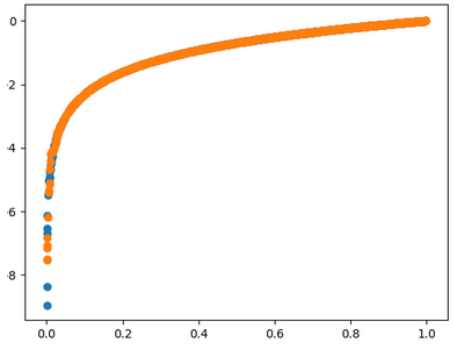
xMin et xMax précisent les bornes de l'intervalle comprenant toutes les valeurs de x, tandis que step spécifie la distance entre chaque nœud de notre spline. Sur un intervalle de [0, 1] et step = 0.1 la spline s'appuie sur 10 nœuds.

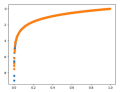
Cas d'école

```
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 43 :

Nous démarrons avec la construction de notre ensemble de données en remplissant X avec des valeurs tirées au hasard dans [0, 1]. De là, nous peuplons Y en appliquant la fonction logarithme.
Puis, de la même manière que dans le cas de la droite, nous remplissons Xm avec les termes de notre polynôme de degré 1, à l'aide de la fonction splinify(). A se calcule ensuite avec la même formule, et nous évaluons dans la foulée notre spline pour toutes les abscisses de notre échantillon à l'aide d'un simple produit scalaire. Le résultat est stocké dans Yreg.
Les trois dernières lignes génèrent la figure, où l'on voit que notre modélisation par une spline linéaire de notre ensemble de test fonctionne très bien. Il est possible de dégrader la qualité de cette modélisation en jouant sur le paramètre step de la fonction splinify().





Page 44 :

La fonction logarithme sur l'intervalle [0, 1], en bleu, et sa modélisation par la spline, en orange. Les deux se superposent quasi parfaitement.

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]
```

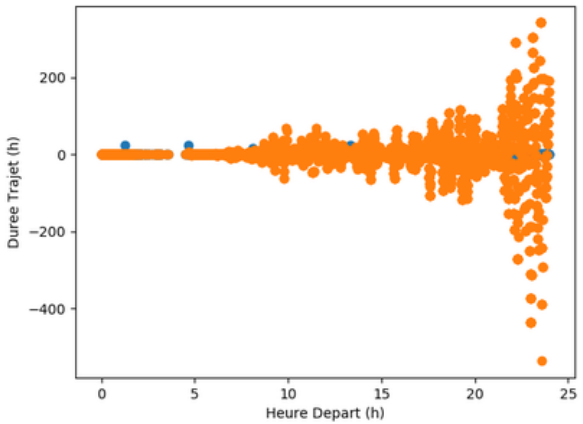

Page 45 :

JFK → Upper Manhattan

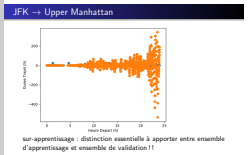
```
pu = [parser.parse(dt)for dt in JFK_MU['tpep_pickup_datetime'].values]
do = [parser.parse(dt)for dt in JFK_MU['tpep_dropoff_datetime'].values]
dur = [(b -a).total_seconds()/ 3600.0for a, b inzip(pu, do)]
startTime = [dt.hour+ dt.minute/ 60.0for dt in pu]

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 46 :

JFK → Upper Manhattan



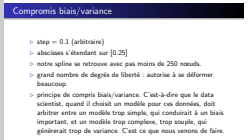
sur-apprentissage : distinction essentielle à apporter entre ensemble d'apprentissage et ensemble de validation ! !



Page 47 :

Compromis biais/variance

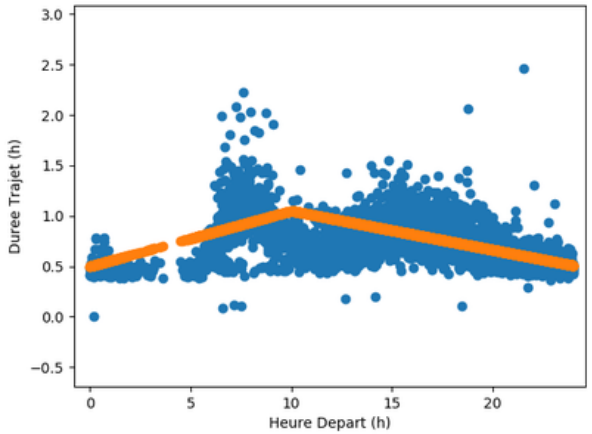
- step = 0.1 (arbitraire)
- abscisses s'étendant sur [0.25]
- notre spline se retrouve avec pas moins de 250 nœuds.
- grand nombre de degrés de liberté : autorise à se déformer beaucoup.
- principe de compris biais/variance. C'est-à-dire que le data scientist, quand il choisit un modèle pour ces données, doit arbitrer entre un modèle trop simple, qui conduirait à un biais important, et un modèle trop complexe, trop souple, qui générerait trop de variance. C'est ce que nous venons de faire.

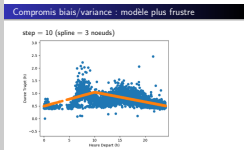


Page 48 :

Compromis biais/variance : modèle plus frustre

step = 10 (spline = 3 noeuds)

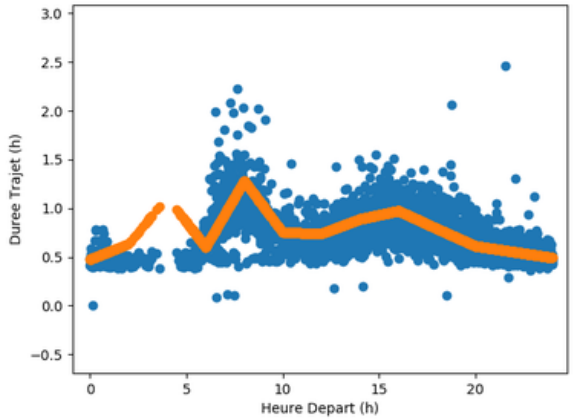


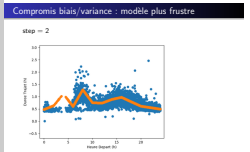


Page 49 :

Compromis biais/variance : modèle plus frustre

step = 2





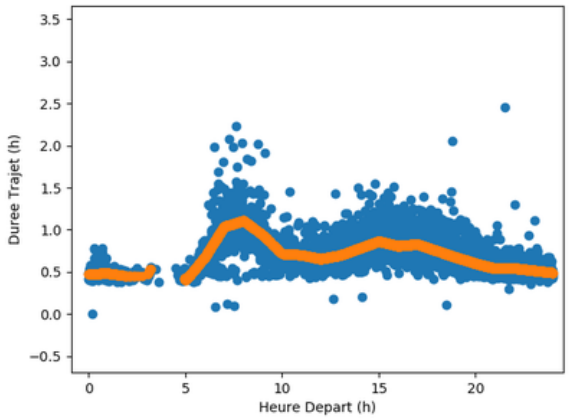
Page 50 :

Pour choisir la valeur de notre paramètre step, nous avons procédé par itération, en évaluant au jugé la qualité du résultat obtenu.
Cette méthode a le mérite de mettre à contribution l'expertise du data scientist, mais afin, une fois obtenu un premier résultat, de trouver le meilleur paramétrage, il convient de se baser sur des critères plus scientifiques.
Pour cela, il faut comme détaillé dans le précédent encadré, disposer d'un ensemble d'apprentissage et de validation. Doté de ces ensembles, il est alors possible de calculer plusieurs métriques, permettant de quantifier à quel point la solution modélise correctement la réalité.
Ces métriques sont souvent des mesures d'erreurs entre les valeurs réelles et leur prédiction à l'aide du modèle. On peut citer par exemple la Mean Absolute Error (MAE) ou encore la Root Mean Square Error (RMSE).

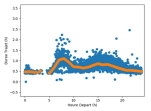
Points aberrants

erreur de modélisation aux alentours des 4h20. Cette erreur est due à la présence de points aberrants, qui sont soit des erreurs de mesures, soit des cas extraordinaires de bouchons, pannes, 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 51 :



Page 52 :

Le code a été développé sous Ubuntu, et est disponible sur GitHub : https ://github.com/kayhman/SmartLinearReg

Introduction

- nombre de variables d'un ensemble de données devient trop important.
- analyse précise dans chacune des dimensions, il faut un ensemble de mesures tout à fait gigantesque
- difficile pour un humain d'appréhender les relations entre autant de variables.

Introduction
- nombre de variables d'un ensemble de données devient trop important.
- analyse précise dans chacune des dimensions, il faut un ensemble de mesures tout à fait gigantesque
- difficile pour un humain d'appréhender les relations entre autant de variables.

Page 53 :

Exemple

3 espèces d'iris différentes, rassemble quatre mesures différentes :

- la longueur des sépales ;
- la largeur des sépales ;
- la longueur des pétales ;
- la largeur des pétales

Exemple
3 espèces d'iris différentes, rassemble quatre mesures différentes :
- la longueur des sépales ;
- la largeur des sépales ;
- la longueur des pétales ;
- la largeur des pétales

Page 54 :

Dimension 3

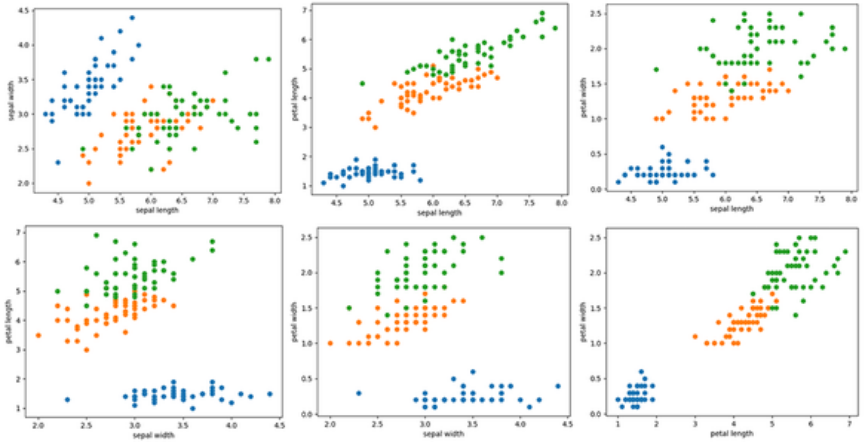
Comparaisons deux à deux des variables de l'ensemble

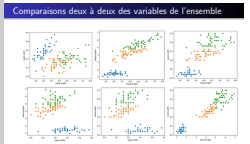
```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()

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 55 :

La dimension de cet ensemble, n = 4, étant réduite, le nombre de graphiques à tracer n'excède pas n(n-1)/2 = 6.
Cela reste analysable par un humain, et il est d'ailleurs facile de générer automatiquement ces analyses

Comparaisons deux à deux des variables de l'ensemble





Page 56 :

PCA

- Analyse en composantes principales : réduire la dimension de l'ensemble étudié en identifiant les dimensions qui portent le plus d'informations
- si l'un des prédicteurs a la même valeur pour tous les échantillons, alors il n'apporte aucune information
- identifier les axes qui portent le plus d'informations, et ce, de manière ordonnée
- Il s'agit quasi systématiquement de combinaison linéaire des prédicteurs.

PCA
- Analyse en composantes principales : réduire la dimension de l'ensemble étudié en identifiant les dimensions qui portent le plus d'informations
- si l'un des prédicteurs a la même valeur pour tous les échantillons, alors il n'apporte aucune information
- identifier les axes qui portent le plus d'informations, et ce, de manière ordonnée
- Il s'agit quasi systématiquement de combinaison linéaire des prédicteurs.

Page 57 :

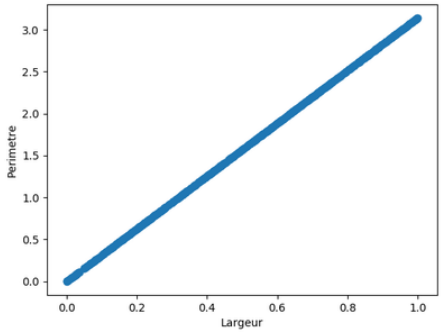
un simple cas 2D

```
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()
```

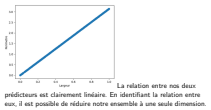

un simple cas 2D
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np
nbSamples =1000
X0 = [random()for x inrange(nbSamples)]
X1 = [3.1416*x for x in X0]
plt.scatter(X0, X1)
plt.show()

Page 58 :

Dans cet exemple, nous collectons la largeur d'un objet et son périmètre. Or, il s'avère que tous ces objets sont des disques. Étant connu leur largeur d, qui n'est autre que leur diamètre, il est facile de calculer leur périmètre $p = d * \pi$.



La relation entre nos deux prédicteurs est clairement linéaire. En identifiant la relation entre eux, il est possible de réduire notre ensemble à une seule dimension.



La relation entre nos deux prédicteurs est clairement linéaire. En identifiant la relation entre eux, il est possible de réduire notre ensemble à une seule dimension.

Page 59 :

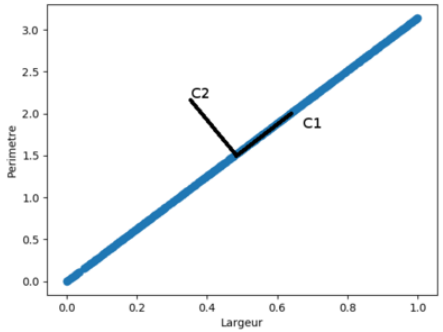

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np

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 60 :

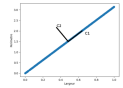
```
[[ 0.30331383 0.95289072]
[-0.95289072 0.30331383]]

[ 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 61 :

Notez que si l'on fait le ratio entre les deux coordonnées du premier vecteur, on retombe bien sur π.
Point très important aussi, si l'on s'amuse à faire le produit scalaire entre ces deux vecteurs, on découvre qu'ils sont orthogonaux



Le premier axe, celui avec la plus grande valeur propre, suffit seul pour capturer notre ensemble



Le premier axe, celui avec la plus grande valeur propre, suffit seul pour capturer notre ensemble

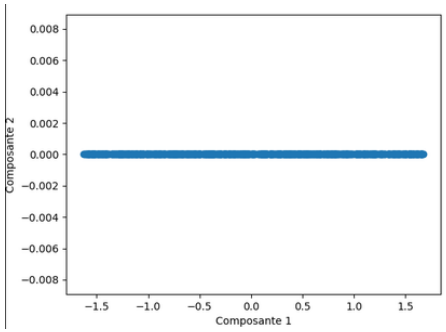
Page 62 :

la matrice présentée ci-dessus peut-être considérée, dans le cas 2D tout au moins, comme une matrice de rotation.

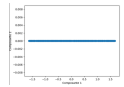
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from random importrandom
import numpy as np

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()
```


la matrice présentée ci-dessus peut-être considérée, dans le cas 2D tout au moins, comme une matrice de rotation.

Page 63 :



Plus de doute, la seconde dimension de notre cas 2D ne sert définitivement à rien.

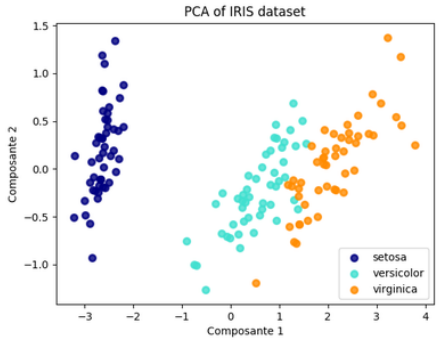


Plus de doute, la seconde dimension de notre cas 2D ne sert définitivement à rien.

Page 64 :

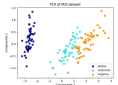
PCA et iris

```
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()
```

L'analyse en composante principale nous fournit automatiquement une représentation qui sépare bien les différents types d'iris.

Page 65 :



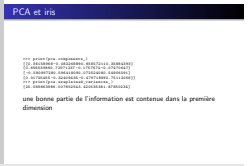
L'analyse en composante principale nous fournit automatiquement une représentation qui sépare bien les différents types d'iris.

Page 66 :

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]
```

une bonne partie de l'information est contenue dans la première dimension



Page 67 :

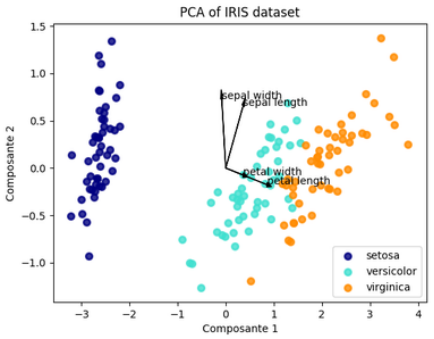
BIPLOT

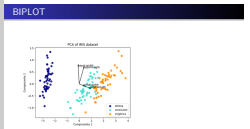
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA

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 68 :

biplot : afficher sur un graphique 2D le maximum d'informations sur toutes les dimensions du problème. Cet outil permet donc d'afficher plusieurs dimensions en seulement deux dimensions.
Pour cela, on repart de la projection de nos données dans le plan 2D décrit par les deux premières composantes principales de notre analyse. Puis on affiche sous forme de vecteurs 2D les dimensions initiales du problème considéré.

BIPLOT



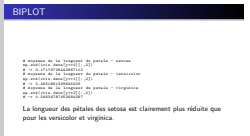


Page 69 :

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
```

La longueur des pétales des setosa est clairement plus réduite que pour les versicolor et virginica.



Page 70 :

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
```

Les valeurs sont dans ce cas très proches : ce n'est pas un bon paramètre pour distinguer les différentes espèces.

BIPLOT

Les valeurs sont dans ce cas très proches : ce n'est pas un bon paramètre pour distinguer les différentes espèces.

Page 71 :

Normalisation

- l'analyse en composantes principales fournit une série d'axes d'analyse qui capture la variabilité des données étudiées, et ce par ordre décroissant.
- Les données s'étalent donc largement suivant le premier axe, tandis qu'elles sont assez condensées autour du dernier.
- Si les données ne sont pas normalisées, c'est-à-dire si on ne les a pas retravaillées de telle manière que leurs moyennes soient nulles, et leurs écarts-types valent 1.0, alors l'analyse peut être biaisée par les différences d'unités utilisées.

Normalisation

- l'analyse en composantes principales fournit une série d'axes d'analyse qui capture la variabilité des données étudiées, et ce par ordre décroissant.
- Les données s'étalent donc largement suivant le premier axe, tandis qu'elles sont assez condensées autour du dernier.
- Si les données ne sont pas normalisées, c'est-à-dire si on ne les a pas retravaillées de telle manière que leurs moyennes soient nulles, et leurs écarts-types valent 1.0, alors l'analyse peut être biaisée par les différences d'unités utilisées.

Page 72 :

Données brutes

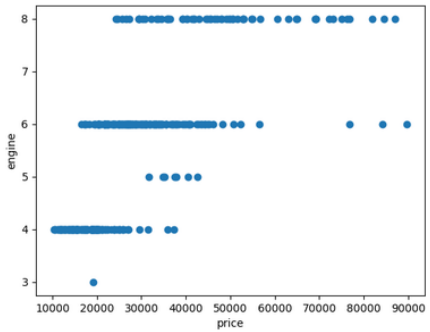
```
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt

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()
```


Données brutes

Page 73 :

Pour illustrer l'importance de la normalisation, nous allons nous intéresser à deux variables : le prix et la cylindrée. Le prix est donné en dollars, tandis que la cylindrée est en litre. Sans normalisation, on compare donc des données qui ont des échelles bien différentes, ce qui biaise le résultat.

BIPLOT



BIPLOT

Page 74 :

Brutes, ces données donnent l'impression que les voitures se distinguent surtout selon leur prix, puisque ce dernier varie de 10 000 à 90 000, tandis que la cylindrée est cantonnée à l'intervalle [3, 8].

Données normalisées

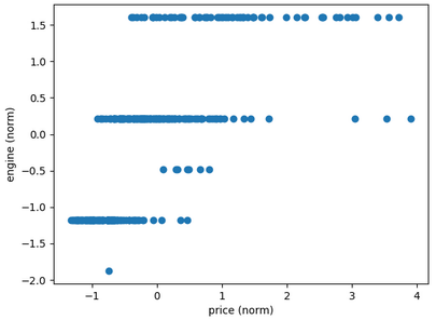
Normalisons nos données : une moyenne nulle et un écart type de 1

```
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()
```


Données normalisées

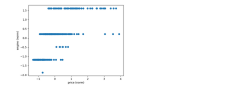
Page 75 :

Données normalisées



Prix et cylindrée des voitures, une fois normalisés. Ces deux axes semblent maintenant contenir de l'information.

Données normalisées



Prix et cylindrée des voitures, une fois normalisés. Ces deux axes semblent maintenant contenir de l'information.

Page 76 :

En substance, la normalisation rend les données adimensionnelles, ce qui permet de les comparer entre elles sans risque.

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)

# donnees brutes
pca.fit(pe)
print(pca.explained_variance_)

# donnees normalisee
pca.fit(pe_scaled)
print(pca.explained_variance_)
```


Page 77 :

PCA

```
# donnees brutes
[ 2.32179369e+08 1.08822536e+00]
# donnees normalisees
[ 1.69570742 0.31072345]
```


Page 78 :

L'analyse en composantes principales fait donc les mêmes observations que nous. Dans le premier cas, celui des données brutes, elle se laisse berner par la différence d'unité entre prix et cylindrée, et estime indûment que l'essentiel de l'information est porté par un unique axe.
Dans le second cas, l'analyse est plus mesurée, car la variance des données n'est pas clairement portée par un unique axe.

Matrice de corrélation

- en Python : `C = pe_scaled.transpose()*pe_scaled`
- Cette matrice contient une information de valeur : chaque élément *Cij* quantifie la relation entre les variables *i* et *j*. Si *Cij* est positive, alors lorsque *i* croît, alors *j* aussi. Si par contre, elle est négative, alors *j* décroît tandis que *i* croît.
- Dans le cas où *Cij* est nul, et c'est là que ça devient intéressant, alors les variables *i* et *j* ne sont pas corrélées. Elles varient donc indépendamment l'une de l'autre.
- Le cas particulier où la matrice est diagonale est donc particulièrement sympathique, car dans ce cas-là, les variables sont toutes indépendantes les unes des autres.

Matrice de corrélation
- en Python : C = pe_scaled.transpose()*pe_scaled
- Cette matrice contient une information de valeur : chaque élément Cij quantifie la relation entre les variables i et j. Si Cij est positive, alors lorsque i croît, alors j aussi. Si par contre, elle est négative, alors j décroît tandis que i croît.
- Dans le cas où Cij est nul, et c'est là que ça devient intéressant, alors les variables i et j ne sont pas corrélées. Elles varient donc indépendamment l'une de l'autre.
- Le cas particulier où la matrice est diagonale est donc particulièrement sympathique, car dans ce cas-là, les variables sont toutes indépendantes les unes des autres.

Page 79 :

C'est justement à ce cas particulier que se ramène l'analyse en composantes principales. En effet, la matrice de corrélation C est par construction symétrique, carrée, et contient des valeurs réelles. Il est donc possible de la diagonaliser, c'est-à-dire de trouver un changement de repère rendant les directions orthogonales.
C'est justement ce que fait l'analyse en composantes principales, en calculant les vecteurs propres et les valeurs propres de C. Les vecteurs propres sont alors les nouveaux axes d'analyse, tandis que les valeurs propres permettent de classer ces axes par variance décroissante.
À noter que pour des raisons de performances, on ne procède pas toujours directement à la diagonalisation de C, mais plutôt à la décomposition de en valeurs singulières.

Framework and Data

IML

Cédric Buche

ENIB

1er juin 2018

Framework and Data
IML
Cédric Buche
ENIB
1er juin 2018

Page 80 :