**Python for Data Science - Cours 1**

Clustering -> partionnement des données [clustering de données](https://www.google.com/search?q=clustering&hl=fr&sxsrf=ACYBGNT12M0fMzemgxD4zjX3xXyH8O8cGA:1568270898156&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjW-a6u2MrkAhWoxYUKHQmLDhgQ_AUIFSgE&biw=1440&bih=688" \t "_blank).

Elle vise à diviser un ensemble de données en différents « paquets » homogènes, en ce sens que les données de chaque sous-ensemble partagent des caractéristiques communes, qui correspondent le plus souvent à des critères de proximité (similarité informatique) que l'on définit en introduisant des mesures et classes de distance entre objets.

**Trouver un use case pour le ML dans un domaine**

| **Application** | **Data** | **KPI/Am** | **Diff** |
| --- | --- | --- | --- |
| Generation du parole au karaoké | Sound Vidéo | \_ | \_ |
| Génération de photo/face | Photo | \_ | \_ |
| Email filtering | Text | % des spams détectés | 2-3 |
| Automatic traduction | Text | \_ | 4 |
| Generer design system (materials/solar panels) | Tabular data | \_ | 4 |
| Customer changing | \_ | \_ | \_ |
| Maintenance prédictive | Tabular data | \_ | \_ |
| Fraud détection | \_ | \_ | \_ |

Erreur = prédiction - réalité

ML = trouver des fonctions en minimisant ses erreurs

**Gradient descendant** utilise le minimum d'une fonction

**Generalisation overfilling** = apprendre par coeur

**Python for Data Science - Cours 2**

**Pourquoi python ?**

* Facile
* Outils pour les datasciences
  + Numpy
  + Matplotlib
  + Pandas
  + Scikit learn : ML
  + Pyspark : big data
* Typés dynamiquement
* Procédural

**ML** = Apprendre des règles depuis des données

**Deux champs d'apprentissage du ML**

1. Classification

Predire des variables discretes.

exemple : X(i), the feature of the problem

X(i) : 0 ? 1 ? 2 ? ..... => Y Target variable label

2. Regression

Predire une variable continue (numérique).

exemple : Predire le prix d'une maison dépendamment des features

**Trouver deux exemples de regression/classification**

| **Regression** | **Classification** |
| --- | --- |
| Prix d'une maison = f(size, nb\_room, city...) | Type des films = f(titre, description, actors) |
| Temperature global = f(CO2, sunlight) | effet d'une molecule (1 = effective, 0 = not effective) |
| Demography | Fonctionnement d'une machine (0 = pas cassé, 1 = cassé) |
| Turn over | \_ |

**2.1 Regression linéaire**

**But** : trouver une fonction qui est la plus proche du dataset (minimiser la distance moyenne entre les points et une fonction et trouver les bons 𝜃θ0 , 𝜃θ1... qui sont compatibles avec les données.

=> Pour ca il faut trouver le min J𝜃θ.

Pour trouver le minmin, il faut caluler le **Gradient descendant**. Le GD est un algorithme.  
Gradient = vecteur d'une dérivée partiel d'une fonction (variable multiple). Direction vers laquelle tu as à déplacer l'entrée de la fonction pour l'améliorer le plus possible.

Etape 0 : initialiser aléatoirement 𝜃θ0 , 𝜃θ1

Etape 1 : LOOP.  
Pour chaque exemple, calculer la prediction de f𝜃θ(X(i)) = Y(i). Calculer le gradient de J.  
Mettre à jour les paramètres 𝜃θ.  
𝜃θnext = 𝜃θcurrent - 𝛼α \* grad J𝜃θ

𝛼α est le taux d'apprentissage.

exemple:: ^y = f(x) = 𝜃θ0 + 𝜃θ1 \* X1 + ... + 𝜃θn \* Xn  
𝜃θ = parametres

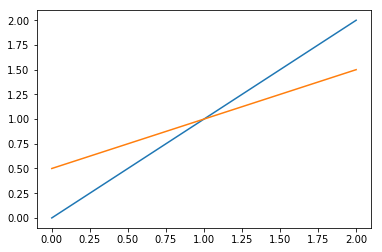
Entrée [8]:

**import** matplotlib.pyplot **as** plt

plt.plot([0, 1, 2])

plt.plot([0.5, 1, 1.5])

plt.show()



**Notation 2D**

table = mes données, 1 ligne = 1 exemple, 1 colone = 1 variable

X = Maison 1, Maison 2, Maison 3

Xi = ligne i, Xi = colone i

Pour calculer la marge d'erreur entre la courbe actuelle et celle qu'on a calculé, on utilise le **MSE ( Mesure Symbol Error)**.

MSE = 1𝑚1m ×× ( 𝛿δ1 + 𝛿δ2 + ... + 𝛿δm)

Exercice : f(x) = x2 - 2x + 1. Calculer f'(x).

pour x = 2 →→ f'(x) = 2x - 2 donc f'(2) = 2.

Calculer f(x), f(x + 0.1 \* f'(x)) et f(x - 0.1 \* f'(x))

pour x = 2 →→ f(x + 0.1 \* f'(x)) = 1,44 et f(x - 0.1 \* f'(x)) = 0,64

Récap TP:

* Data Science:
* \* Analyse (Data analyst, BI)
* \* Processing/Storage (Data engineering)

\* Prediction (ML, datascience)

* Python:
* \* Langage basic
* \* Numpy (vectorisation)
* \* Matplotlib

\* Pandas (manipulé and process tabular data)

Model = fonction prédictive  
X feature/predicting variable -> f -> prediction/target variable

**Logistic Regression**:

When you "fit" to find the parameters of Log.Reg you do not minimise the MSE with gradient descente. You minimise croos entropy loss

"Linear Regression + sigmoid function" for classification

signoid(x) = 𝜃θ(x) = 11+𝑒𝑥11+ex

**Cross entropy loss / Binary cross entropy loss**

Rq: Loss function measure difference between prediction and "Label".

L = 1𝑛𝑏𝐸𝑥1nbEx ∑𝑛𝑏𝑖=0∑i=0nb yi loglog ^yi + (1 - yi) loglog(1 - yi)

**Metrics**: R2 : coefficient of determination

Log.Reg(Classification):

* Accuracy  
  rate/ percentage of good predictions. It is a trap when data is imbalanced.
* Precision/Recall Data X -> Label = 0  
  model f(x) = 1 -> False positive model (The model tells you have cancer (^y=1) but you have (y=0)  
  Confusion matrix avec True Nagetive, True Positive, False Positive, False Negative.

Recall : its the proportion of the class the model identified.  
𝑇𝑃𝑇𝑃+𝐹𝑁TPTP+FN

Prediction : error percentage when a prediction on a class.  
𝑇𝑃𝑇𝑃+𝐹𝑃TPTP+FP

f1 harmonics = 2∗𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛∗𝑅𝑒𝑐𝑎𝑙𝑙𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛+𝑅𝑒𝑐𝑎𝑙𝑙2∗Precision∗RecallPrecision+Recall

/!\ Metrics on both train and test.

* F1 score
* AUC

**Python for Data Science - Cours 3**

**Feature engineering**: creer/transferer la variable de notre dataset pour avoir de la performance

**Categorical variable**:

- Gender

- Color

- House Orientation

- City

-> Ordinal  
-> Non Ordinal

**One-Hot encoding**: Transform un categorical variable in several 0 ou 1 variables

Exercice :

Load the following data set in pandas.

Do a linear model with scikit learn

Display the prediction function and the scatter datapoints on a plot

Does it sounds well ?

Transform the variable (use cosinus, exponential function, log, and so ) And test the a model each time with the new feature

what is the best transformation ?

Entrée [1]:

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** numpy **as** np

**import** math

​

**from** sklearn.linear\_model **import** LinearRegression

Entrée [2]:

df **=** pd.read\_csv("./feature\_engineering1.csv")

Entrée [20]:

model **=** LinearRegression()

xp **=** df[['experience']]

sal **=** df[['salary']]

Entrée [21]:

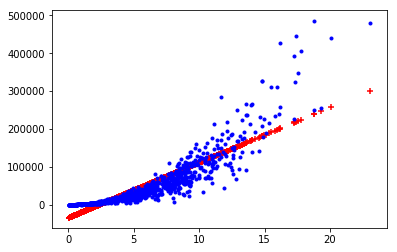
model.fit(xp, sal)

prediction\_salary **=** model.predict(xp)

plt.scatter(xp, prediction\_salary, c**=**'red', marker**=**'+')

plt.scatter(xp, sal, c**=**'blue', marker**=**'.')

plt.show()



Entrée [32]:

salcos **=** np.cos(sal)

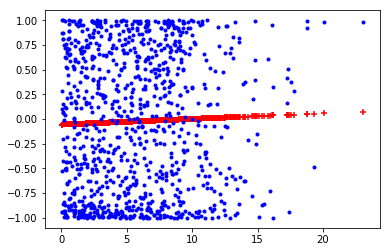
model.fit(xp, salcos)

prediction\_salary **=** model.predict(xp)

plt.scatter(xp, prediction\_salary, c**=**'red', marker**=**'+')

plt.scatter(xp, salcos, c**=**'blue', marker**=**'.')

plt.show()



Entrée [38]:

salsqrt **=** np.sqrt(sal)

model.fit(xp, salsqrt)

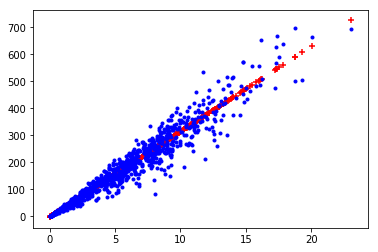
prediction\_salary **=** model.predict(xp)

plt.scatter(xp, prediction\_salary, c**=**'red', marker**=**'+')

plt.scatter(xp, salsqrt, c**=**'blue', marker**=**'.')

plt.show()

*# best transformation*



Entrée [35]:

sallog **=** np.log(sal)

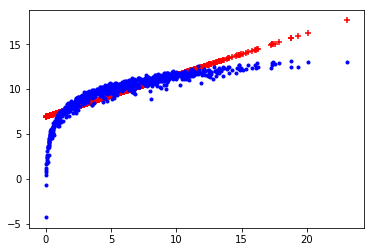
model.fit(xp, sallog)

prediction\_salary **=** model.predict(xp)

plt.scatter(xp, prediction\_salary, c**=**'red', marker**=**'+')

plt.scatter(xp, sallog, c**=**'blue', marker**=**'.')

plt.show()



Entrée [37]:

salexp **=** np.exp(sal)

model.fit(xp, salexp)

prediction\_salary **=** model.predict(xp)

plt.scatter(xp, prediction\_salary, c**=**'red', marker**=**'+')

plt.scatter(xp, salexp, c**=**'blue', marker**=**'.')

plt.show()

*# marche po*

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-37-b820443dca5e> in <module>

**1** salexp = np.exp(sal)

----> 2 model.fit(xp, salexp)

**3** prediction\_salary = model.predict(xp)

**4** plt.scatter(xp, prediction\_salary, c='red', marker='+')

**5** plt.scatter(xp, salexp, c='blue', marker='.')

~/anaconda3/lib/python3.7/site-packages/sklearn/linear\_model/base.py in fit(self, X, y, sample\_weight)

**456** n\_jobs\_ = self.n\_jobs

**457** X, y = check\_X\_y(X, y, accept\_sparse=['csr', 'csc', 'coo'],

--> 458 y\_numeric=True, multi\_output=True)

**459**

**460** if sample\_weight is not None and np.atleast\_1d(sample\_weight).ndim > 1:

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in check\_X\_y(X, y, accept\_sparse, accept\_large\_sparse, dtype, order, copy, force\_all\_finite, ensure\_2d, allow\_nd, multi\_output, ensure\_min\_samples, ensure\_min\_features, y\_numeric, warn\_on\_dtype, estimator)

**757** if multi\_output:

**758** y = check\_array(y, 'csr', force\_all\_finite=True, ensure\_2d=False,

--> 759 dtype=None)

**760** else:

**761** y = column\_or\_1d(y, warn=True)

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in check\_array(array, accept\_sparse, accept\_large\_sparse, dtype, order, copy, force\_all\_finite, ensure\_2d, allow\_nd, ensure\_min\_samples, ensure\_min\_features, warn\_on\_dtype, estimator)

**571** if force\_all\_finite:

**572** \_assert\_all\_finite(array,

--> 573 allow\_nan=force\_all\_finite == 'allow-nan')

**574**

**575** shape\_repr = \_shape\_repr(array.shape)

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in \_assert\_all\_finite(X, allow\_nan)

**54** not allow\_nan and not np.isfinite(X).all()):

**55** type\_err = 'infinity' if allow\_nan else 'NaN, infinity'

---> 56 raise ValueError(msg\_err.format(type\_err, X.dtype))

**57**

**58**

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Entrée [41]:

df1 **=** pd.read\_csv("./feature\_engineering2.csv")

Entrée [42]:

model **=** LinearRegression()

stck **=** df1[['stock']]

to **=** df1[['turn\_over']]

Entrée [65]:

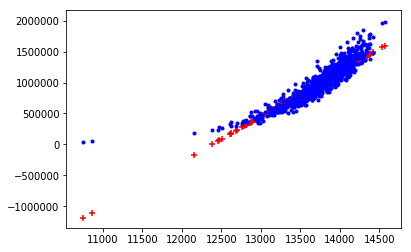
model.fit(stck, to)

prediction\_to **=** model.predict(stck)

plt.scatter(stck, prediction\_to, c**=**'red', marker**=**'+')

plt.scatter(stck, to, c**=**'blue', marker**=**'.')

plt.show()



Entrée [66]:

tocos **=** np.cos(to)

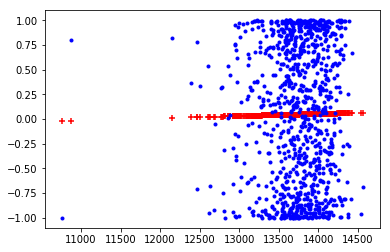
model.fit(stck, tocos)

prediction\_to **=** model.predict(stck)

plt.scatter(stck, prediction\_to, c**=**'red', marker**=**'+')

plt.scatter(stck, tocos, c**=**'blue', marker**=**'.')

plt.show()



Entrée [67]:

toexp **=** np.exp(to)

model.fit(stck, toexp)

prediction\_to **=** model.predict(stck)

plt.scatter(stck, prediction\_to, c**=**'red', marker**=**'+')

plt.scatter(stck, toexp, c**=**'blue', marker**=**'.')

plt.show()

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-67-8966bca43f53> in <module>

**1** toexp = np.exp(to)

----> 2 model.fit(stck, toexp)

**3** prediction\_to = model.predict(stck)

**4** plt.scatter(stck, prediction\_to, c='red', marker='+')

**5** plt.scatter(stck, toexp, c='blue', marker='.')

~/anaconda3/lib/python3.7/site-packages/sklearn/linear\_model/base.py in fit(self, X, y, sample\_weight)

**456** n\_jobs\_ = self.n\_jobs

**457** X, y = check\_X\_y(X, y, accept\_sparse=['csr', 'csc', 'coo'],

--> 458 y\_numeric=True, multi\_output=True)

**459**

**460** if sample\_weight is not None and np.atleast\_1d(sample\_weight).ndim > 1:

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in check\_X\_y(X, y, accept\_sparse, accept\_large\_sparse, dtype, order, copy, force\_all\_finite, ensure\_2d, allow\_nd, multi\_output, ensure\_min\_samples, ensure\_min\_features, y\_numeric, warn\_on\_dtype, estimator)

**757** if multi\_output:

**758** y = check\_array(y, 'csr', force\_all\_finite=True, ensure\_2d=False,

--> 759 dtype=None)

**760** else:

**761** y = column\_or\_1d(y, warn=True)

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in check\_array(array, accept\_sparse, accept\_large\_sparse, dtype, order, copy, force\_all\_finite, ensure\_2d, allow\_nd, ensure\_min\_samples, ensure\_min\_features, warn\_on\_dtype, estimator)

**571** if force\_all\_finite:

**572** \_assert\_all\_finite(array,

--> 573 allow\_nan=force\_all\_finite == 'allow-nan')

**574**

**575** shape\_repr = \_shape\_repr(array.shape)

~/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py in \_assert\_all\_finite(X, allow\_nan)

**54** not allow\_nan and not np.isfinite(X).all()):

**55** type\_err = 'infinity' if allow\_nan else 'NaN, infinity'

---> 56 raise ValueError(msg\_err.format(type\_err, X.dtype))

**57**

**58**

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Entrée [68]:

tolog **=** np.log(to)

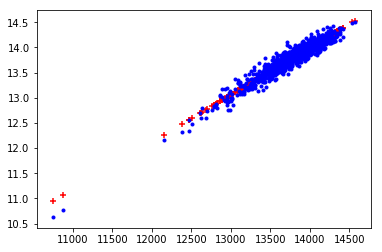
model.fit(stck, tolog)

prediction\_to **=** model.predict(stck)

plt.scatter(stck, prediction\_to, c**=**'red', marker**=**'+')

plt.scatter(stck, tolog, c**=**'blue', marker**=**'.')

plt.show()



Entrée [69]:

tosqrt **=** np.sqrt(to)

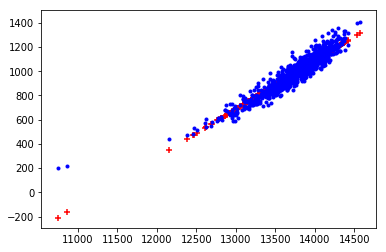
model.fit(stck, tosqrt)

prediction\_to **=** model.predict(stck)

plt.scatter(stck, prediction\_to, c**=**'red', marker**=**'+')

plt.scatter(stck, tosqrt, c**=**'blue', marker**=**'.')

plt.show()



Find how to to one hot encoding of this variable and then make a model with the one\_hot encoded variable and the size to predict the price

Entrée [61]:

df2 **=** pd.read\_csv('./TP1/house.csv')

Entrée [63]:

df2\_ohe **=** pd.get\_dummies(df2,'Orientation')

**On training models**:  
When you do a fit on a linear regression, le paramètre 𝜃θ est trouvé automatiquement.  
GD to minimise the loss function (MSE).

Problem:  
if the features do not have the same scale. GD will not work well.

ex: predict if a land ill be sold in 6 months  
Dataset = size (big number) type (one hot encoding)  
Size is much bigger than Type and it will prevent gradient descent working well -> Scale Down Numerical values

Ways to normalize variable:

* standard scaler X -> X - mean(X)  
  Most values (but not all) will be close to 0
* Min Max Scaler X -> 𝑋−𝑚𝑖𝑛(𝑋)𝑚𝑎𝑥(𝑋)−𝑚𝑖𝑛(𝑋)X−min(X)max(X)−min(X)  
  all values will be between 0 and 1

Remember to scale feature. For gradient based algorithm.

**TP 4**

**Unsupervised learning (no label)**:

* Clustering : Kmeans, Spectral clustering, Dbscan
* Dimensionnality Reduction (PCA/Autoencoders)
  + Tsne and Umap (plot data in 2D)

K-mean -> you have to choose the number of clusters

**Ensembling** : combine several models to improve the final performance.  
"Wisdom of the crowd". Random Forest = Ensembling of Desicion Tree.

3 types of ensembling (mainly):

1. Bagging -> Random forest
2. Bosting -> Xg boost, lightGBM
3. Stacking

Bagging : def sampling without remplacing = national lotery  
you have a dataset D = (X,Y)  
Step 1: generate datasets by doing Sampling with replacing  
Step 2: Train models on each dataset  
Step 3: combine the predictions.  
Comment ?  
=> regression -> average  
=> classification -> majority voting

Boosting:  
Step 1: Train the model on D  
Step 2: Train a second model and make it pay more attention to the exemple where M1 failed.  
ect...

Stacking:  
Train n models M1, M2..., Mn. Combine prédictions by training a model Mstack which combines M1, M2..., Mn.  
Mstack will use the predictions of M1, M2..., Mn as features.

**Intro computer vision**

Computer science field. The goal is to give machines the sense of sight (to be able to see). Related to Image Processing and also video. Old field -> Has been revolutionized by deep learning.

What is an image ?  
For computers : array of value (black and white image 2D matrix/array, colored image 3D array). Arrays contain the pixel intensities

Tools:

* openCV (C/C++ lib) python Bidings
* scikit image
* pillow

**What are convultions ?**

Like rolling mean in 2D.  
You want to compute a locale weighted average.  
Weighted average = 1919 (w1P2,2 + w2P2,3 + w3P2,4 + w5P3,2 + ...)

𝐖𝐞𝐢𝐠𝐡𝐭𝐬=𝑊1𝑊4𝑊7𝑊2𝑊5𝑊8𝑊3𝑊6𝑊9Weights=[W1W2W3W4W5W6W7W8W9]

Convolutions : compute the weighted average of all windows in the img

**Deep Learnig**

Sub field of ML. DL = models with a lot of parameters.  
Model = functions. X -> f𝜃θ -> 𝑌̂ Y^ (predictions)

𝑌̂ Y^DL = f𝜃θ(X) = very complicated

Nerone (mathematical one) = function = small model

The NonLinear function is called = activation function  
ex ReLU (rectified linear unit)  
ReLU(X) = max(0,𝑥)max(0,x)

**Convonutional neural network**

basic idea : each neurone is a convulution + non linear function

sconv = ReLU(Conv𝜃θ(x))  
The weights of the filter are the parameters of the model

**Introduction to NLP**

Text processing

Text data is text data, it's not number. You can't use ML on

First: you have to find a numerical representations of your text.  
What is a good numerical representation ? -> It is a representation which carry the meaning of the text.

ex of bad one: one hot encoding "chicken"=[1,0,0,0,0,0] "beef"=[0,1,0,0,0,0]  
<chicken|beef>=0 => bad :(

Better word representation: Word2Vec  
-> (computed with neural network) V("Queen")-V("King")+V("Women") = V("Man")

Doc representations:

* TFIDF Represent doc as vector

Important rq You have also a preprocessing step to do.  
-> Tokenisation  
-> Lemmalization | Stemming -> Removing Stopwords

Use case:

* Sentiment analysis for customer satisfaction
* NLP for legal doc CHATBOTS

=> Spacy

**TDIDF**: Represent document as vectors

1. Count the frequencies of each word in the document (TF -> team frequency)
2. IDF: inverse document frequency Document frequency : count in how many document the word appears

TFIDF = 𝑇𝐹𝐼𝐷𝐹