Installation de l’environnement :

docker run -v `pwd`:/home/guest/host -p 4040:4040 -p 8888:8888 -p 23:22 -ti --privileged yannael/kafka-sparkstreaming-cassandra

lancement de jupyter : notebook --allow-root

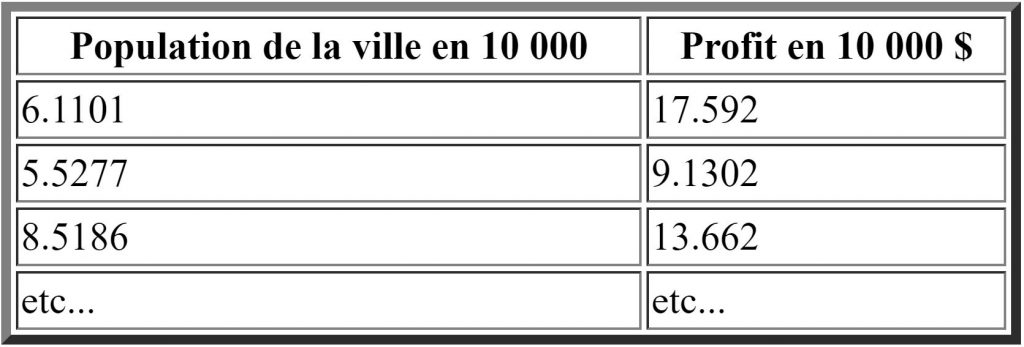
Chapitre régression :

TP régression linéaire :

Supposons que vous soyez le chef de direction d’une franchise de camions ambulants (Food Trucks). Vous envisagez différentes villes pour ouvrir un nouveau point de vente. La chaîne a déjà des camions dans différentes villes et vous avez des données pour les bénéfices et les populations des villes.

Vous souhaitez utiliser ces données pour vous aider à choisir la ville pour y ouvrir un nouveau point de vente.

Les données sont en CSV :



**Etape 1:**

Chargement des diverses librairies utiles pour ce notebook

# chargement de bibliothèques

"""

instruction specifique pour utiliser matplotlib dans un notebook

quand on utilise les notebooks Jupyter pour utiliser Matplotlib

"""

%matplotlib inline

import pandas as pd

import matplotlib.pyplot as plt

from scipy import stats

**Etape 2 :**

Chargement du jeu de données

La fonction read\_csv(), renvoie un DataFrame. Il s’agit d’un tableau de deux dimensions contenant, respectivement, la taille de population et les profits effectués. Pour pouvoir utiliser les librairies de régression de Python, il faudra séparer les deux colonnes dans deux variables Python.

df = pd.read\_csv("univariate\_linear\_regression\_dataset.csv")

X = df.iloc[:,0] #selection de la première colonne de notre dataset (indice 0)

Y = df.iloc[:,1] #selection de la première colonne de notre dataset (indice 1)

**Etape 3 :**

Visualisation du jeu de données

axes = plt.axes()

axes.grid()

plt.scatter(X,Y)

plt.show()

**Etape 4 :**

Entrainement d'un modèle de régression linéaire

La fonction est => H(x)=intercept+slope\*x

slope, intercept, r\_value, p\_value, std\_err = stats.linregress(X, Y)

**Etape 5 :**

Ecriture de la fonction de prédiction avec Python

# définition de quatre observations

def predict(x):

return slope \* x + intercept

**Etape 6 :**

Dessiner la fonction de prédiction

axes = plt.axes()

axes.grid()

plt.scatter(X,Y)

fitLine = predict(X)

plt.plot(X, fitLine, c='r')

plt.show()

**Etape 7 :**

Prédiction d'une nouvelle observation

On voit que pour pour la valeur x = 22.5, la valeur de y pour est environ 25. Utilisons la fonction predict pour trouver une estimation de H(x = 22.5)

predict(22.5)

le résultat est 23.093309207513975

Assez proche ! 22.5 => 23.09

TP régression linéaire Multi-variables :

Le jeu de données comporte 47 enregistrements.

Note : Un pieds² fait environ 0,092 m²

Etape 1 : chargement des librairies:

import pandas as pd

import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

Etape 2 :

Chargement des données

df = pd.read\_excel("Multivariate\_Linear\_Regression\_dataset.xlsx")

# df.head permet de voir les premières lignes chargées de notre fichier Excel

print df.head()

Etape 3 :

Nous devons ensuite séparer la variable cible (Y) des variables prédictives x

#Récupérer le prix : les valeurs observées pour la variable Cible

Y = df["prix"]

#Récupérer les variables prédictives : La superficie en pieds² et le nb chambre

X = df[['taille\_en\_pieds\_carre','nb\_chambres']]

Etape 4 :

Représentation des données

Pour mieux comprendre les données, il est souvent utile de les visualiser. Ici, nous avons un jeu de données de 3 dimensions. Nos données seront éparpillées dans un espace 3D (chaque ligne du dataset sera un couple (x,y,z)). On peut représenter les données dans un espace 3D avec la librairie matplotlib :

import matplotlib.pyplot as plt

fig = plt.figure()

ax = fig.add\_subplot(1,2,1, projection='3d')

ax.scatter(df["taille\_en\_pieds\_carre"], df["nb\_chambres"], df["prix"], c='r', marker='^')

ax.set\_xlabel('surface en pieds\_carre')

ax.set\_ylabel('nb\_chambres')

ax.set\_zlabel('prix en $')

plt.show()

Etape 5 :

Normalisation des données et feature Scaling

Vous l’avez peut être remarqué, notre exemple comporte des variables prédictives avec des ordres de grandeurs très différents. En effet, le nombre de chambre d’une maison est généralement compris entre 1 et 10 alors que la superficie se compte en quelques milliers de pieds2.

Pour appliquer l’algorithme Multivariate Regression, il est nécessaire que les variables prédictives faisant partie du modèle prédictif soient du même ordre de grandeur. Généralement, il faut que la valeur de chaque variable prédictive soient compris (approximativement) entre -1 et 1. Si certaines valeurs dépassent un peu (par exemple -2 , 1.5…) ce n’est pas très grave.

Pour ramener nos variables prédictives au même ordre de grandeur, nous appliquerons un procédé qui s’appelle : features scaling

La librairie Scikit learn de Python propose plusieurs classes et méthodes pour faire de la préparation de données (Data pre-processing) pour les algorithmes de Machine Learning. Le package sklearn.preprocessing propose la classe StandardScaler qui permettra de faire du features scaling sur toutes nos variables prédictives.

scale = StandardScaler()

X\_scaled = scale.fit\_transform(X[['taille\_en\_pieds\_carre', 'nb\_chambres']].as\_matrix())

#print X\_scaled

Etape 6 :

Apprentissage de l’algorithme par les données

On y est ! tous les ingrédients sont là pour pouvoir appliquer notre algorithme de régression linéaire multivariée. Parmi les procédés mathématiques pour faire de la régression linéaire, il y a le Ordinary Least Squares (OLS).

OLS est une méthode pour estimer une variable cible dans un modèle de régression linéaire. Pour y parvenir, OLS va minimiser la somme des carrés des différences entre les réponses observées (de notre Training Set) et ceux prédits par la fonction linéaire appliquée à nos variables prédictives de notre ensemble de données.

est = sm.OLS(Y, X).fit()

Ceci nous affiche un résumé sur le modèle prédictif produit :

print est.summary()

Etape 7 :

Visualisation de la fonction de prédiction

"prix\_estimé = F(superficie, nb\_chambres) = ε + α \* superficie + β \* nb\_chambres"

frst\_col\_surface = df.iloc[0:len(df),0] #selection de la première colonne de notre dataset

scnd\_col\_nb\_chambre = df.iloc[0:len(df),1]

third\_col\_prix = df.iloc[0:len(df),2]

def predict\_price\_of\_house(taille\_maison, nb\_chambre):

return 140.8611 \* taille\_maison + 1.698e+04 \* nb\_chambre # not scaled

#return 1.094e+05 \* taille\_maison + (6578.3549 \* nb\_chambre) # scaled

def predict\_all(lst\_sizes, lst\_nb\_chmbres):

predicted\_prices = []

for n in range(0, len(Y)):

predicted\_prices.append(predict\_price\_of\_house(lst\_sizes[n], lst\_nb\_chmbres[n]))

return predicted\_prices

#print predict\_all(df["taille\_en\_pieds\_carre"], df["nb\_chambres"])

# set up the axes for the second plot

fig=plt.figure()

ax = fig.add\_subplot(1, 2, 2, projection='3d')

ax.plot\_trisurf(df["taille\_en\_pieds\_carre"], df["nb\_chambres"], predict\_all(df["taille\_en\_pieds\_carre"], df["nb\_chambres"]))

plt.show()

Etape 8 :

Test de prédiction

print predict\_price\_of\_house(4500,5)

TP régression polynomiale:

Supposons, qu’un site e-commerce souhaite savoir si le temps de chargement des pages web de son site impacte le montant du panier moyen de l’internaute.

On souhaite trouver une fonction de prédiction qui modélisera cette corrélation.

Etape 1 :

Importation des librairies

import numpy as np

import matplotlib.pyplot as plt

Etape 2 :

Création d’un jeu de données

np.random.seed(2)

Génération de 1000 nombres aléatoires distribués selon la loi normale X ~ N(3, 1), les pages chargeront en moyenne en 3 secondes et un ecart-type de 1 seconde.

tempsChargementPages = np.random.normal(3.0, 1.0, 1000)

génération aléatoires de montants d'achat corrélés aux temps de chargement

montantAchat = np.random.normal(50.0, 10.0, 1000) / (tempsChargementPages \* tempsChargementPages)

le temps de chargement de la page est notre variable prédictive

x = np.array(tempsChargementPages)

montantAchat est la variable cible (qu'on cherche à prédire)

y = np.array(montantAchat)

Etape 3 :

Visualisation des données

axes = plt.axes()

axes.grid()

plt.xlabel('temps de chargement en secondes')

plt.ylabel('montant des achats euros')

plt.scatter(tempsChargementPages, montantAchat)

plt.show()

Etape 4 :

Retrouver une fonction polynomiale

Plus on rajoute un degré au polynôme, plus ce dernier devient sensible aux données et s’y adapte mieux.

trouvons un modèle avec un polynôme de 4 degrés.

x = np.array(tempsChargementPages)

y = np.array(montantAchat)

p4 = np.poly1d(np.polyfit(x, y, 4))

Le polynôme que nous retrouve python est le suivant :

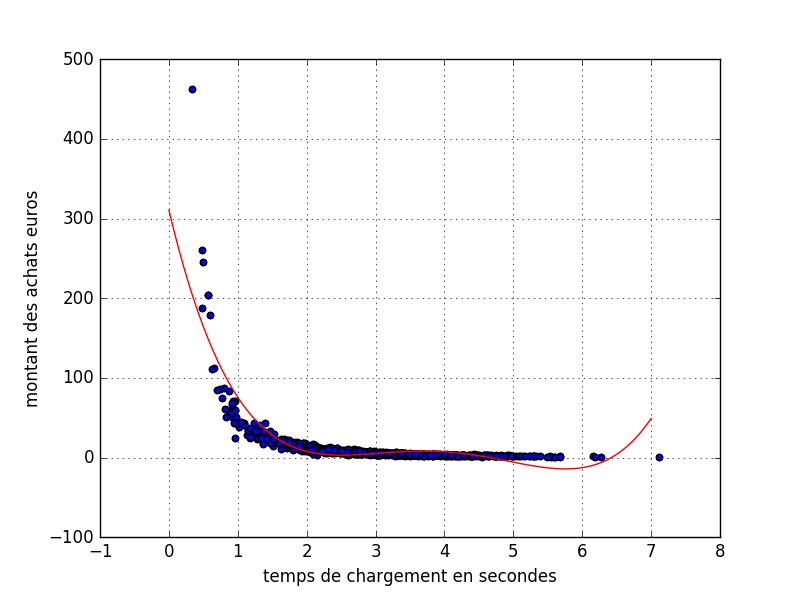


xp = np.linspace(0, 7, 100)

plt.scatter(x, y)

plt.plot(xp, p4(xp), c='r')

plt.show()



TP K-NN

Pour ce TP nous allons utiliser le module pyspark sous python 3

docker run -it --rm -p 8889:8888 jupyter/pyspark-notebook

La donnée est le nerf de la guerre quand il s’agit de Machine Learning. Pour ce tutoriel, on utilisera le célèbre jeu de données **MNIST.**Ce dernier est une base de donnée de chiffres manuscrits très utilisée en Machine Learning, notamment pour l’entrainement et le test de modèles prédictifs. Wikipedia fournit des [informations abondantes](https://en.wikipedia.org/wiki/MNIST_database) sur ce *dataset.*



**MNIST dataset**

**Représentation des données**

MNIST est une base de données étiquetée propice pour un apprentissage supervisé. Dans l’image ci-dessus, pour chaque chiffre, on a sa représentation sous forme d’image ainsi que son étiquette. Par exemple, pour le dernier chiffre en bas à droit, l’étiquette vaut 9 vu qu’il s’agit du chiffre 9. La représentation de ces chiffres est normalisée à travers tout le jeu de données MNIST. Ainsi, chaque chiffre est codé dans un format 8 pixels \* 8 pixels. En plus, chaque pixel peut prendre une valeur de 0 à 255. Cette plage de valeurs représente le niveau de gris [Grayscale](https://fr.wikipedia.org/wiki/Niveau_de_gris). En d’autres terme, chaque représentation **d’une image est une matrice** de dimension .Le jeu de données MNIST présent par défaut dans la librairie [Scikit Learn](http://scikit-learn.org/stable/), comporte un sous-ensemble de la “vraie” base de données MNIST. Le sous-ensemble comporte  chiffres que nous diviserons par la suite en deux sous ensembles : d’*entrainement* et de *test.*

Etape 1 :

Chargement des diverses librairies utiles pour ce notebook

**from** **sklearn.datasets** **import** \* *# chargement du package datasets contenant plusieurs jeu de données*

**import** **pandas** **as** **pd** *# Chargement de Pandas*

%matplotlib inline *# instruction spécifique pour quand on utilise les notebooks Jupyter pour utiliser Matplotlib*

**import** **matplotlib.pyplot** **as** **plt** *# import de Matplotlib*

**from** **sklearn.model\_selection** **import** train\_test\_split *# classe utilitaire pour découper les jeux de données*

**from** **sklearn.neighbors** **import** KNeighborsClassifier *# import de la classe de K-NN*

*#Méthode displayImage pour afficher des données images (méthode optionnelle)*

**def** displayImage(i):

plt.imshow(digit['images'][i], cmap='Greys\_r')

plt.show()

Affichage des jeux de données

*#Affichage de jeu de données*

digit = load\_digits() *# chargement du dataset MNIST*

dig = pd.DataFrame(digit['data'][0:1700]) *# Création d'un dataframe Panda*

dig.head() *# affiche le tableau ci-dessous*

Etape 2 :

Affichez une image du jeu de données!

displayImage(0) *# affichage de la première image du jeu de données MNIST*

digit.keys()

train\_x = digit.data

train\_y = digit.target

Etape 3 :

Découpage du jeu de données MNIST en Training set et Testing Set.

Avec :

* 75% en Training set
* 25% en Testing set

x\_train,x\_test,y\_train,y\_test=train\_test\_split(train\_x,train\_y,test\_size=0.25)

Etape 4 :

Instantiation et entrainement d'un classifieur K-NN avec K = 7

KNN = KNeighborsClassifier(7)

KNN.fit(x\_train, y\_train)

Etape 5 :

Calcul du Scoring de performane de notre modèle 7-NN

*#la précision par rapport aux données de test*

print(KNN.score(x\_test,y\_test))

Etape 6 :

Test de prédiction de notre modèle sur un chiffre non encore vu

*#Afficher un élement de la matrice format image*

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

test = np.array(digit['data'][1726])

test1 = test.reshape(1,-1)

displayImage(1726)

*#prédiction*

KNN.predict(test1)

TP K-Means

Le groupe examiné comprenait des noyaux appartenant à trois variétés différentes de blé: Kama, Rosa et Canadian, de 70 éléments chacun, choisis au hasard pour l'expérience.

L'ensemble de données peut être utilisé pour les tâches de classification et d'analyse de cluster.

sept paramètres géométriques des grains de blé ont été mesurés:

zone A,

périmètre P,

compacité C = 4piA / P ^ 2,

longueur du noyau,

largeur du noyau,

coefficient d'asymétrie

longueur de la gorge du noyau. Tous ces paramètres étaient des valeurs réelles continues.

en les regroupant en 3 groupes avec K-means

Etape 1 :

Initialisation du projet sur spark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('cluster').getOrCreate()

from pyspark.ml.clustering import KMeans

dataset = spark.read.csv("seeds\_dataset.csv",header=True,inferSchema=True)

Etape 2 :

Visualisation d’un jet de données

dataset.head()

dataset.describe().show()

Etape 3 :

Formatage de données

from pyspark.ml.linalg import Vectors

from pyspark.ml.feature import VectorAssembler

dataset.columns

vec\_assembler = VectorAssembler(inputCols = dataset.columns, outputCol='features')

final\_data = vec\_assembler.transform(dataset)

from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler

scalerModel = scaler.fit(final\_data)

# Normaliser chaque caractéristique pour avoir un écart type d'unité.

final\_data = scalerModel.transform(final\_data)

Etape 4 :

Entrainement du model et son evaluation

# Training a k-means model.

kmeans = KMeans(featuresCol='scaledFeatures',k=3)

model = kmeans.fit(final\_data)

# Evaluate clustering by computing Within Set Sum of Squared Errors.

wssse = model.computeCost(final\_data)

print("Within Set Sum of Squared Errors = " + str(wssse))

centers = model.clusterCenters()

print("Cluster Centers: ")

for center in centers:

print(center)

model.transform(final\_data).select('prediction').show()

TP Naive Bayes

Notre jeu d’apprentissage sera une collection de mails. Pour chacun d’eux, on indiquera s’il s’agit d’un SPAM ou non.



Notre *Training Set* est composé de :

* 500 mails Spam
* 2500 mails Non spam (Ham)

Tout d’abord, on chargera nos mails en utilisant les modules « OS » et “IO” de Python

Etape 1 : chargement de données

import os

import io

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

# quelques variables globales utiles

PATH\_TO\_HAM\_DIR = "/home/guest/host/emails/ham"

PATH\_TO\_SPAM\_DIR = "/home/guest/host/emails/spam"

SPAM\_TYPE = "SPAM"

HAM\_TYPE = "HAM"

#les tableaux X et Y seront de la meme taille et ordonnes

X = [] # represente l'input Data (ici les mails)

#indique s'il s'agit d'un mail ou non

Y = [] #les etiquettes (labels) pour le training set

def readFilesFromDirectory(path, classification):

os.chdir(path)

files\_name = os.listdir(path)

for current\_file in files\_name:

message = extract\_mail\_body(current\_file)

X.append(message)

Y.append(classification)

#fonction de lecture du contenu d'un fichier texte donne.

#ici, on fait un peu de traitement pour ne prendre en compte que le "corps du mail".

# On ignorer les en-tetes des mails

def extract\_mail\_body(file\_name\_str):

inBody = False

lines = []

file\_descriptor = io.open(file\_name\_str,'r', encoding='latin1')

for line in file\_descriptor:

if inBody:

lines.append(line)

elif line == '\n':

inBody = True

message = '\n'.join(lines)

file\_descriptor.close()

return message

readFilesFromDirectory(PATH\_TO\_HAM\_DIR, HAM\_TYPE)

readFilesFromDirectory(PATH\_TO\_SPAM\_DIR, SPAM\_TYPE)

A l’exécution de ce code, on chargera 3000 mails (ce qui constitue notre *Training Set*).

* *Pensez à modifier le chemin vers le répertoire contenant les mails lors de l’exécution du script sur votre machine.*

Etape 2 :

Préparation du classifieur Naïve Bayes

*Pour classifier nos emails avec Naïve Bayes, on****calcule le nombre d’occurrence des mots****dans chaque mail*

training\_set = pd.DataFrame({'X': X, 'Y': Y})

vectorizer = CountVectorizer()

counts = vectorizer.fit\_transform(training\_set['X'].values)

classifier = MultinomialNB()

targets = training\_set['Y'].values

classifier.fit(counts, targets)

Etape 3 :

Test de l’algo

examples = [' free viagra now!!!', "Hi Bob, how about a game of golf tomorrow?"]

example\_counts = vectorizer.transform(examples)

predictions = classifier.predict(example\_counts)

print predictions

TP Arbre de décision

from sklearn import tree

# You may hard code your data as given or to use a .csv file import csv then fetch your data from .csv file

# Assume we have two dimensional feature space with two classes we like distinguish

dataTable = [[2,9],[4,10],[5,7],[8,3],[9,1]]

dataLabels = ["Class A","Class A","Class B","Class B","Class B"]

# Declare our classifier

trained\_classifier = tree.DecisionTreeClassifier()

# Train our classifier with data we have

trained\_classifier = trained\_classifier.fit(dataTable,dataLabels)

# We are done with training, so it is time to test it!

someDataOutOfTrainingSet = [[9,10]]

label = trained\_classifier.predict(someDataOutOfTrainingSet)

# Show the prediction of trained classifier for data [9,10]

print label[0]

TP plus avancé :

Etape 1 : chargement des librairies

import numpy as np

import pandas as pd

from sklearn import datasets

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.grid\_search import GridSearchCV

from sklearn.cross\_validation import train\_test\_split

import warnings

warnings.filterwarnings('ignore')

get\_ipython().magic(u'matplotlib inline')

Etape 2 : chargement des jeux de données d’une source externe

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data',

sep= ',', header= None)

Etape 3 : création des jeux de données et des prédicats

#create your X and y

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:,0]

Etape 4 : échantillonnage et teste des valeurs

#split your X and y

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.3, random\_state = 100)

#Run decision tree - create instance of class, fit, score, predict

#l’indice de GINI : il mesure avec quelle fréquence un élément aléatoire de l'ensemble serait mal classé si son étiquette était choisie aléatoirement selon la distribution des étiquettes dans le sous-ensemble.

#L'indice de diversité de Gini peut être calculé en sommant la probabilité pour chaque élément d'être choisi,

clf\_gini = DecisionTreeClassifier(criterion = "gini", random\_state = 100,

max\_depth=5, min\_samples\_leaf=6)

clf\_gini.fit(X\_train, y\_train)

Etape 5 : vérification des valeurs de resultats et de traitements

score =clf\_gini.score(X\_train, y\_train)

print X\_train

print score

print X\_test

predicted= clf\_gini.predict(X\_test)

#how do i get the score from the predicted values now?

print predicted

TP libre : prédiction obtention crédit

La particularité de ce TP c’est qu’il inclus plusieurs étapes de préparation et d’évaluation des données avant d’entrainer le modèle avec des données de testes avant d’évaluer l’obtention de crédit par des nouveaux clients en utilisant l’arbre de décision

<https://github.com/harshitahluwalia7895/Loan_Prediction>

TP : SVM

L’objectif de cet exercice est de créer une classification et une seconde non linéaire permettant de regrouper les éléments regroupés dans les mêmes zones ou reparties dans les différentes zones du graph.

Etape 1 :

Importation de la librairie SVM de scikitlearn

import numpy as np

import pandas as pd

import sklearn

import sklearn.datasets as ds

import sklearn.model\_selection as ms

import sklearn.svm as svm

import matplotlib.pyplot as plt

%matplotlib inline

Etape 2 :

Génération de jeux de données en 2 dimensions

X = np.random.randn(200, 2)

y = X[:, 0] + X[:, 1] > 1

Etape 3 :

Faire le fitting du support vectoriel lineaire classifié

# We train the classifier.

est = svm.LinearSVC()

est.fit(X, y)

Etape 4 :

On definit une fonction qui affichera les frontières et les decisions en fonction des données d’entrainement classifiés

# We generate a grid in the square [-3,3 ]^2.

xx, yy = np.meshgrid(np.linspace(-3, 3, 500),

np.linspace(-3, 3, 500))

# This function takes a SVM estimator as input.

def plot\_decision\_function(est, title):

# We evaluate the decision function on the grid.

Z = est.decision\_function(np.c\_[xx.ravel(),

yy.ravel()])

Z = Z.reshape(xx.shape)

cmap = plt.cm.Blues

# We display the decision function on the grid.

fig, ax = plt.subplots(1, 1, figsize=(5, 5))

ax.imshow(Z,

extent=(xx.min(), xx.max(),

yy.min(), yy.max()),

aspect='auto',

origin='lower',

cmap=cmap)

# We display the boundaries.

ax.contour(xx, yy, Z, levels=[0],

linewidths=2,

colors='k')

# We display the points with their true labels.

ax.scatter(X[:, 0], X[:, 1],

s=50, c=.5 + .5 \* y,

edgecolors='k',

lw=1, cmap=cmap,

vmin=0, vmax=1)

ax.axhline(0, color='k', ls='--')

ax.axvline(0, color='k', ls='--')

ax.axis([-3, 3, -3, 3])

ax.set\_axis\_off()

ax.set\_title(title)

Etape 5 :

On visualise la separation lineaire du SVC (support vectoriel classifié)

ax = plot\_decision\_function(

est, "Linearly separable, linear SVC")

Etape 6 :

On modifie les labels avec une fonction XOR. Dans le cas la classification est non linéaire ce qui rendra le jeu de données non exploitable en SVC.

y = np.logical\_xor(X[:, 0] > 0, X[:, 1] > 0)

# We train the classifier.

est = ms.GridSearchCV(svm.LinearSVC(),

{'C': np.logspace(-3., 3., 10)})

est.fit(X, y)

print("Score: {0:.1f}".format(

ms.cross\_val\_score(est, X, y).mean()))

# We plot the decision function.

ax = plot\_decision\_function(

est, "XOR, linear SVC")

Etape 7 :

Il est possible d’utiliser une SVC non linéaire en utilisant un noyau non linéaire. Le noyau spécifie une transformation non linéaire des points en un espace dimensionnel plus grand. Les points transformés sont séparés de manière linaires.

y = np.logical\_xor(X[:, 0] > 0, X[:, 1] > 0)

est = ms.GridSearchCV(

svm.SVC(), {'C': np.logspace(-3., 3., 10),

'gamma': np.logspace(-3., 3., 10)})

est.fit(X, y)

print("Score: {0:.3f}".format(

ms.cross\_val\_score(est, X, y).mean()))

plot\_decision\_function(

est.best\_estimator\_, "XOR, non-linear SVC")

`

Tp Final de validation des acquis :

A partir du fichier zoo.data.csv disponible dans le dossier TP, on va essayer de classifier les animaux de ce dernier en plusieurs groupe d’animaux. Une fois la prédiction faite, on récupèrera le résultat et on l’affichera dans un graph sur Tableau Software.

Il est nécessaire de se baser sur la libraire pyspark

Pour rappel les étapes nécessaires :

* Importer la librairie KMeans de spark
* Construire un vecteur sans les champs avec du texte vu que le kmeans n’accepte que les valeurs numériques
* Transformer le dataset en vecteur
* Rescaler les valeurs
* Réaliser le fitting necessaire
* Transformer le dataset
* Entrainer le model
* Evaluer le coût du clustering !
* Définir les centroids
* À partir du model transformer le dataset vers le nouveau dataset
* Exporter en CSV le dataset
* Sur tableau afficher les regroupements d’animaux par catégorie et extraire les indicateurs nécessaires.