MARIN Thomas 3_CI

Machine learning
Compte-rendu Scikit learn

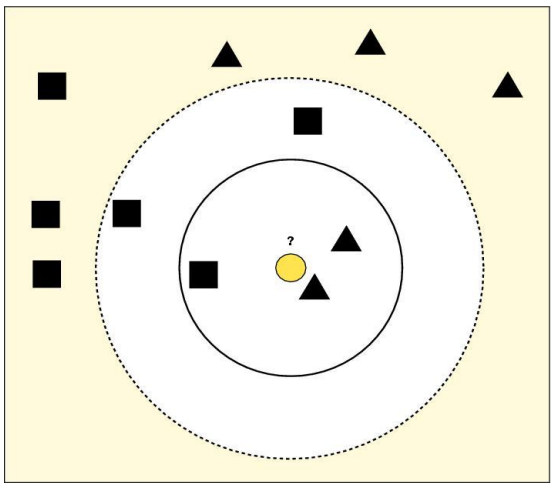
Le machine learning peut se résumer à la création d'une fonction de prédiction. Cette fonction est le plus souvent le résultat d'une minimisation de l'erreur.

On peut classer les algorithmes en deux catégories :
-une première catégorie où l'algorithme se contente d'appliquer une série de règles, sans vraiment apprendre des cas précédents (un algorithme dit « DecisionTrees » par exemple) ;
-une seconde catégorie où l'algorithme est en apprentissage automatique à partir de données précédemment intégrées (les « Support Vector Machine » en font partie).

I/ Nearest Neighbors

```
from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors=7)
#import de l'algorithme knn avec un k=7
```

L'algorithme « Nearest Neighbors » (kNN) peut servir aussi bien pour la classification que pour la régression linéaire. Le principe de cet algorithme consiste à choisir les « k » données les plus proches du point à déterminer pour en prédire sa valeur. Le « k » signifie que lorsque l'on ajoutera une donnée en entrée (dans un nuage de points rouges et bleus par exemple), si l'on attribut k=7, l'algorithme va observer les 7 points les plus proches de la nouvelle donnée afin d'en prédire la couleur. Toutefois, k n'est pas une donnée d'apprentissage, l'algorithme ne pourra pas l'apprendre automatiquement.



Sur le graphique de droite, un exemple de kNN pour un k=3 est représenter via le premier cercle.
Un deuxième exemple de kNN avec k=5 est présent.
Le but est ici de faire varier k pour obtenir une erreur la plus faible possible.

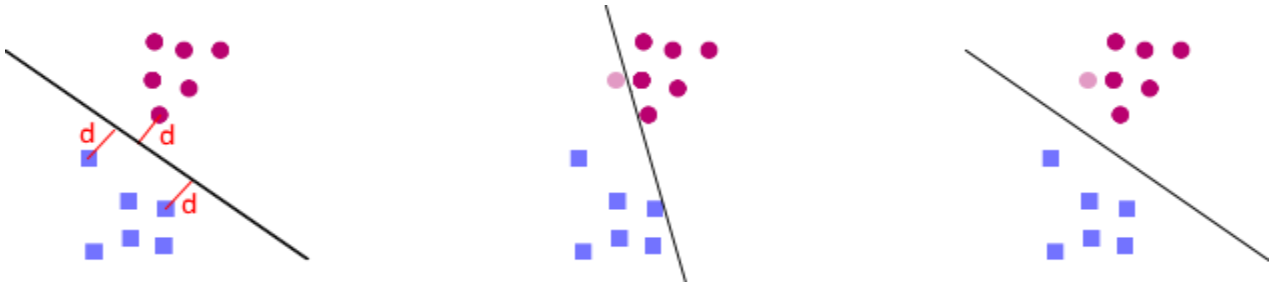
II/ Support Vector Machines

```
from sklearn import svm
svm = svm.SVC(kernel = 3)
```

Les « Support Vector Machines » (SVM) sont des algorithmes utilisés pour des problèmes de classification. C'est-à-dire, que l'on doit séparer deux catégories de points dans un nuages de points, par exemple. Ici, le but du SVM est, à chaque nouveau point que l'on ajoutera, le mettre dans une catégorie, tout en ne connaissant pas la frontière entre ces deux catégories.

La SVM la plus simple fait partie des classificateurs dit linéaires, la séparation des deux groupes d'individus dans le plan sera une droite. Toutefois, des SVM plus élaborées existent comme une SVM linéaire avec noyau.

Phase d'apprentissage :
-en donnant des données d'entrainement a la SVM, en connaissant la répartition des groupes, l'algorithme va alors estimer l'emplacement de la frontière entre les deux groupes. Il sera alors ensuite capable de prédire l'appartenance d'une nouvelle entrée.



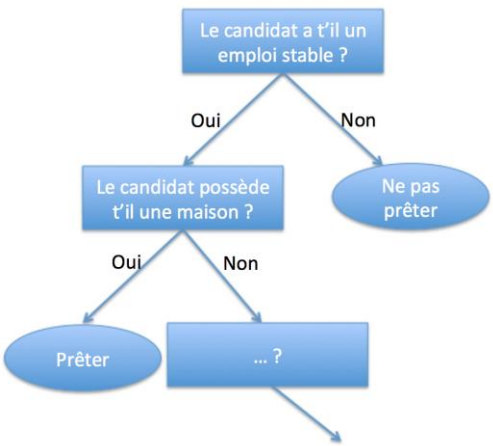
Définition d'une frontière généraliste

Comme choisir la bonne frontière ?
Sur le graphique précédent, on note que la figure centrale n'est pas la mieux adaptée, dans ce cas, l'algorithme classera un rond rouge parmi les carrés bleus. Pour être le plus généraliste possible dans la classification, la SVM doit placer la frontière entre les deux groupes le plus loin de chacun des individus.

III/ Decision Trees

```
from sklearn import tree
tree = tree.DecisionTreeClassifier()
```

L'arbre de décision est une représentation visuelle d'un algorithme de classification de données suivant différents critères que l'on nomme décisions ou nœuds.
Pour un algorithme de ce type, nous aurons en entrée un set de données représentant de nombreux objets ayant chacun des propriétés. Pour un set d'entrainement, ces données seront déjà classées.
L'algorithme va ensuite tester les différentes propriétés pour les répartir dans des sets distincts puis testera les sets ainsi décomposés.



Dans cet exemple d'arbre de décision, l'algorithme va d'abord déterminer la branche à emprunter en étudiant le premier. De cette décision dépendra la suite du test, si la réponse est positive, alors il passera au second nœud, si la réponse est négative alors il arrêtera (nœud oval).

IV/ Regression logistique

```
from sklearn linear_model
logisticr =linear_model.LogisticRegression(C = 1e7)
```

La régression logistique permet de traiter des cas où la variable réponse est de type binaire (oui/non, 0/1). Un modèle de régression logistique permet de prédire la probabilité qu'un événement arrive ou non (valeur allant de1 à 0).

VI/ Discriminant Analysis

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lineard = LinearDiscriminantAnalysis()
```

Algorithme utilisé pour déterminer des variables qui permettent de discriminer deux à plusieurs groupes. Un exemple est de mesurer la taille dans un échantillon aléatoire d'homme et de femme. Les femmes étant, en moyenne, plus petites que les hommes, la différence va être visible dans le calcul de la moyenne et par conséquent la taille va permettre de mieux discriminer les hommes et les femmes. Pour résumer le fonctionnement de cet algorithme, l'appartenance à un groupe ou un autre va se faire par rapport à la moyenne sur une variable particulière et donc utiliser cette dernière pour déterminer l'appartenance à un groupe.

```
Fonction knn permettant d'afficher un graphique de « précision » des erreurs en fonction de k
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 10 20:30:00 2017

@author: thomas
"""

from sklearn.datasets import fetch_mldata
import matplotlib.pyplot as plt
mnist = fetch_mldata('MNIST original')
import numpy as np

# Le dataset principal qui contient toutes les images
print (mnist.data.shape)

# Le vecteur d'annotations associé au dataset (nombre entre 0 et 9)
print (mnist.target.shape)

# Echantillonage du dataset
sample = np.random.randint(70000, size=2500)
data = mnist.data[sample]
target = mnist.target[sample]

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(data, target, train_size=0.8)



from sklearn import neighbors

knn = neighbors.KNeighborsClassifier(n_neighbors=7)
knn.fit(xtrain, ytrain)

error = 1 - knn.score(xtest, ytest)
print('Erreur: %f' % error)


errors = []
for k in range(2,15):
    knn = neighbors.KNeighborsClassifier(k)
    errors.append(100*(1 - knn.fit(xtrain, ytrain).score(xtest, ytest)))
plt.plot(range(2,15), errors, 'o-')
plt.show()
```

```
Différentes fonctions décrites précédemment
# -*- coding: utf-8 -*-
"""
Created on Sun Dec 10 20:30:00 2017

@author: thomas
"""

import numpy as np
from sklearn import linear_model, tree, neighbors, datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

mnist = fetch_mldata('MNIST original')


# Le dataset principal qui contient toutes les images
print (mnist.data.shape)

# Le vecteur d'annotations associé au dataset (nombre entre 0 et 9)
print (mnist.target.shape)

# Echantillonage du dataset
sample = np.random.randint(70000, size=2500)
data = mnist.data[sample]
target = mnist.target[sample]

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(data, target, train_size=0.8)

def knn():
    knn = neighbors.KNeighborsClassifier(n_neighbors=7)
    knn.fit(xtrain, ytrain)

def svm():
    svm = svm.SVC(kernel = 3)
    svm.fit(xtrain, ytrain)

def tree():
    tree = tree.DecisionTreeClassifier()
    tree.fit(xtrain, ytrain)

def logisticr():
    logisticr =linear_model.LogisticRegression(C=1e7)
    logisticr.fit(xtrain, ytrain)

def lineard():
    lineard = LinearDiscriminantAnalysis()
    lineard.fit(xtrain, ytrain)


knn()
svm()
tree()
logisticr()
lineard()
```