INTRODUCTION À LIBSVM SOUS PYTHON

version 0.1 – 2008-10-30

Jérôme Landré - jerome.landre@univ-reims.fr
CReSTIC - Univ. de Reims-Champagne-Ardenne - IUT de Troyes

1 Introduction

Le classifieur "Séparateur à Vaste Marge" (*Support Vector Machine*) permet de classifier un ensemble de données de même type en plusieurs familles grâce à une phase d'apprentissage.

1.1 Quelques questions et leur réponse

– Comment ça marche ?

Le principe des classifieurs SVM est assez simple. Si on n'arrive pas à séparer de façon claire des données dans un espace de dimension n on considère que les données sont issues de la projection d'un espace de dimension supérieure k (avec $k > n$) sur l'espace de dimension n. On tente donc de trouver un espace de dimension k qui va permettre de séparer les données.

– Comment on sépare les données dans cet espace de dimension k ?

C'est la phase d'apprentissage et une fonction d'apprentissage ϕ (le noyau) qui vont nous permettre de séparer correctement les données. Dans la phase d'apprentissage, on va donner des étiquettes aux données d'apprentissage pour dire au SVM à quelle famille quelle donnée appartient.

La fonction ϕ va alors se "débrouiller" pour trouver un modèle de transformation pour arriver à séparer les données étiquetées dans des familles séparée dans l'espace de dimension k avec une séparation (une marge) maximale.

– Comment le SVM classifie les données réelles ?

Le classifieur va appliquer le modèle de transformation défini pour les données d'apprentissage sur les données à classer et les projeter dans l'espace de dimension k pour les étiqueter et reprojeter le résultat sur l'espace de dimension n pour donner le résultat de la classification (l'étiquetage).

La section suivante va nous permettre de mieux comprendre comment cela se passe...

1.2 Exemple en dimension 2

1.3 Python

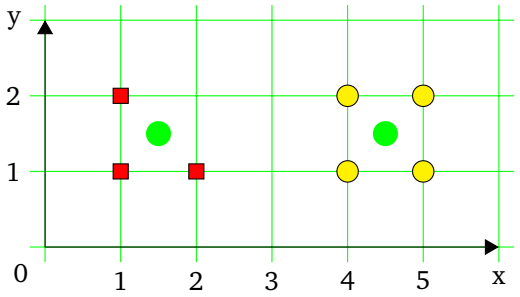
Python est un langage de programmation simple et évolutif pour lequel de nombreuses librairies et extensions spécialisées existent. Cet article décrit l'utilisation de libSVM [2] pour effectuer des classifications de données par apprentissage avec Python.

Pour utiliser libSVM avec Python ou tout autre environnement basé sur Python (comme Sage [1]), il faut mettre le fichier svm.py et le fichier svmc.so (librairie dynamique) dans le répertoire de travail.

2 Utilisation de libSVM

2.1 Un exemple simple pour commencer

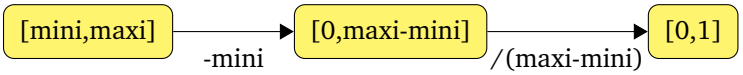
On va débuter avec un exemple simple : on définit un problème de classification de points dans un espace à deux dimensions. On considère les points d'apprentissage en deux dimensions de la figure ci-dessous. Les carrés rouges sont affectés à la classe 0 et les disques jaunes sont affectés à la classe 1 qui sont disjointes pour le premier exemple.



Dans cette figure, les trois points en forme de carré sont les points d'apprentissage de la classe d'étiquette 0 et les quatre points en forme de cercle sont les points d'apprentissage de la classe d'étiquette 1. Les points en forme d'ellipse verte sont les deux points à classer. Je rappelle que l'exemple est volontairement simpliste...

Les SVM utilisent n vecteurs K_i avec $1 < i < n$ de nombres réels k_{ij} de longueur fixe l. Les vecteurs K_i sont constitués de caractéristiques k_{ij} avec $1 < j < l$. Afin de garantir que les données apporteront toutes la même contribution aux calculs des SVM, il FAUT les normaliser. En d'autres termes, si les valeurs d'une caractéristique k_{ij} sont comprises dans l'intervale [m,M] (m : minimum et M : maximum de la plage de valeurs), il faut les ramener dans l'intervalle [0,1]. Il en est ainsi pour chaque valeur de j.

La figure suivante donne les étapes pour passer d'un ensemble de valeurs dans une plage [m,M] à un ensemble de valeurs dans une plage [0,1].



Autrement dit, la normalisation opère ainsi : $k_{ij} = (k_{ij} - mini_j)/(maxi_j - mini_j)$. Attention, il faut normaliser toutes les données du problème en même temps : données d'apprentissage et données à classer.

```
"""
Test des SVM avec Python
"""
from svm import *

# definition de la liste des donnees
liste_donnees = [ [1.0,1.0],[2.0,1.0],[1.0,2.0],[4.0,1.0],[4.0,2.0],
[5.0,1.0],[5.0,2.0],[1.5,0.5],[5.5,2.5] ]
```

On définit ensuite la fonction de normalisation de l'ensemble des vecteurs du problème : vecteurs d'apprentissage et vecteurs à classer.

```
def normalise(liste):
        """
        Fonction de normalisation d'un ensemble de vecteurs
        """
        for j in range(len(liste[0])):
                mini=1e6 # definition du minimum temporaire
                maxi=-1e6 # definition du maximum temporaire
                for i in range(len(liste)):
                        if liste[i][j]<mini:
                                mini=liste[i][j]
                        if liste[i][j]>maxi:
                                maxi=liste[i][j]
                for i in range(len(liste)):
                        liste[i][j]=(liste[i][j]-mini)/(maxi-mini)
        return liste
```

On classe toutes les données en même temps (données d'apprentissage et données à classer).

```
liste_donnees = normalise(liste_donnees)
print "liste des donnees normalisees : ",
print liste_donnees
```

Pour la phase d'apprentissage, on définit une liste d'étiquette et une liste de points. Ici, il y a deux étiquettes 0 et 1 pour les classes et on compte sept points d'apprentissage. On définit donc une liste `liste_etiq` et une liste `liste_points_apprentissage` qui contiennent respectivement les étiquettes des points et les points d'apprentissage.

```
#definition de la liste des points d'apprentissage
# on prend les 7 premiers elements des donnees
liste_pts_apprentissage = liste_donnees[:7]
print "liste des points d'apprentissage : ", liste_pts_apprentissage

# definition de la liste des etiquettes
# il y en a 7 comme les points d'apprentissage
# les 3 premiers points sont dans la classe 0
# les 4 derniers points sont dans la classe 1
liste_etiq = [0,0,0,1,1,1,1]
```

Ensuite, on passe à la phase d'apprentissage du SVM. On définit les paramètres du SVM qu'on souhaite entraîner (fonction *svm_parameter*), on définit le problème à résoudre pour l'apprentissage (*svm_problem*) et enfin on calcule le modèle SVM du classifieur (*svm_model*) à partir du problème et des paramètres.

```
# definition des parametres SVM
param = svm_parameter(kernel_type = RBF)

# definition des etiquettes et des donnees d'apprentissage
prob = svm_problem(liste_etiq , liste_pts_apprentissage)

# calcul du modele de classifieur
mon_modele = svm_model(prob, param)
```

Une fois que l'on obtient le modèle du classifieur, on peut demander une validation croisée avec la fonction *cross_validation*. Cette fonction permet de tester le classifieur SVM obtenu par validation croisée des données.

```
# validation croisee des donnees
valid_croisee = cross_validation(prob, param, 3)
```

Enfin, on peut demander au classifieur de remplir sont rôle et de classer des données qui ne sont pas dans l'ensemble d'apprentissage avec le modèle calculé.

```
# prediction pour le point r
r = mon_modele.predict( liste_donnees[7] )
print "r : "+str(liste_donnees[7])+" - modele de r : "+str(r)

# prediction pour le point p
p = mon_modele.predict( liste_donnees[8] )
print "p : "+str(liste_donnees[8])+" - modele de p : "+str(p)
```

Voici le programme complet pour notre premier exemple.

```
"""
Test des SVM avec Python
"""
# fichier test-svm1.py
from svm import *

# definition de la liste des donnees
liste_donnees = [ [1.0,1.0],[2.0,1.0],[1.0,2.0],[4.0,1.0],[4.0,2.0],[5.0,
1.0],[5.0,2.0],[1.5,1.5],[4.5,1.5] ]

print "liste des donnees : ", liste_donnees

def normalise(liste):
        """
        Fonction de normalisation d'un ensemble de vecteurs
        """
        for j in range(len(liste[0])):
                mini=1e6 # definition du minimum temporaire
                maxi=-1e6 # definition du maximum temporaire
                for i in range(len(liste)):
                        if liste[i][j]<mini:
                                mini=liste[i][j]
                        if liste[i][j]>maxi:
                                maxi=liste[i][j]
                for i in range(len(liste)):
                        liste[i][j]=(liste[i][j]-mini)/(maxi-mini)
        return liste
```

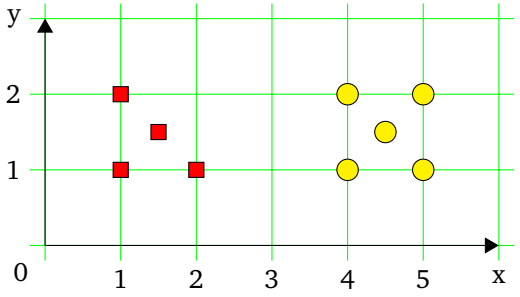
```
liste_donnees = normalise(liste_donnees)
print "liste des donnees normalisees : ", liste_donnees

#definition de la liste des points d'apprentissage, on prend les 7 premiers elements des donnees
liste_pts_apprentissage = liste_donnees[:7]
print "liste des points d'apprentissage : ", liste_pts_apprentissage

# definition de la liste des etiquettes, il y en a 7 comme les points d'apprentissage
# les 3 premiers points sont dans la classe 0, les 4 derniers points sont dans la classe 1
liste_etiq = [0,0,0,1,1,1,1]

# definition des parametres SVM
param = svm_parameter(kernel_type = RBF)

# definition des etiquettes et des donnees d'apprentissage
prob = svm_problem(liste_etiq, liste_pts_apprentissage)

# calcul du modele de classifieur
print "calcul du modele..."
mon_modele = svm_model(prob, param)
print "...modele calcule, OK."
# validation croisee des donnees
valid_croisee = cross_validation(prob, param, 3)

# prediction pour le point r
r = mon_modele.predict( liste_donnees[7] )
print "r : "+str(liste_donnees[7])+" - modele de r : "+str(r)

# prediction pour le point p
p = mon_modele.predict( liste_donnees[8] )
print "p : "+str(liste_donnees[8])+" - modele de p : "+str(p)
```

Et voici le résultat obtenu en lançant la commande "python test-svm-1.py".

```
liste des donnees :  [[1.0, 1.0], [2.0, 1.0], [1.0, 2.0], [4.0, 1.0],
 [4.0, 2.0], [5.0, 1.0], [5.0, 2.0],
 [1.5, 1.5], [4.5, 1.5]]
liste des donnees normalisees :  [[0.0, 0.0], [0.25, 0.0], [0.0, 1.0],
 [0.75, 0.0], [0.75, 1.0], [1.0, 0.0], [1.0, 1.0],
 [0.125, 0.5], [0.875, 0.5]]
liste des points d'apprentissage :  [[0.0, 0.0], [0.25, 0.0], [0.0, 1.0],
 [0.75, 0.0], [0.75, 1.0], [1.0, 0.0], [1.0, 1.0]]
calcul du modele...
*
optimization finished, #iter = 4
nu = 0.857143
obj = -4.250999, rho = 0.220378
nSV = 7, nBSV = 5
Total nSV = 7
...model calcule, OK.
*
optimization finished, #iter = 2
nu = 0.800000
obj = -3.212285, rho = 0.511725
nSV = 4, nBSV = 4
Total nSV = 4
*
optimization finished, #iter = 2
nu = 0.800000
obj = -3.237081, rho = 0.511725
nSV = 4, nBSV = 4
Total nSV = 4
*
optimization finished, #iter = 2
```

```
nu = 1.000000
obj = -2.905946, rho = 0.042179
nSV = 4, nBSV = 4
Total nSV = 4
r : [0.125, 0.5] - modele de r : 0.0
p : [0.875, 0.5] - modele de p : 1.0
```

On constate que le point r (de coordonnées [1.5, 1.5] avant normalisation et [0.125, 0.5] après) est bien dans la classe 0 et que le point p ([4.5, 1.5] puis [0.875, 0.5] après normalisation est bien dans la classe 1.



2.2 Un autre exemple

3 Conclusion

L'interface Python de libSVM permet d'utiliser les classifieurs SVM avec beaucoup de facilité. Il y a très peu de commandes à connaître et à mettre en œuvre pour parvenir à entraîner et à utiliser un classifieur.

On peut aussi utiliser la librairie libsvm sous Sage puisque ce dernier est écrit entièrement en Python.

Références

[1] MANY CONTRIBUTORS, Sage : opensource mathematics software, *http ://www.sagemath.org*, 2008

[2] CHIH-CHUNG CHANG AND CHIH-JEN LIN, LIBSVM – A Library for Support Vector Machines, *http ://www.csie.ntu.edu.tw/˜cjlin/libsvm*, 2008