mardi 21 janvier 2020

Rapport de

Apprentissage statistique et sélection de modèle

Sommaire

J'ai travaillé sur ce projet en utilisant un git. Voici le lien de celui-ci:

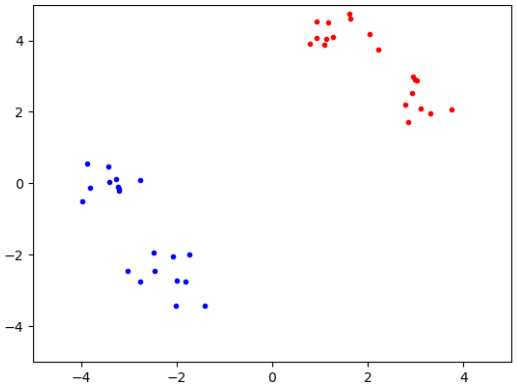
https://github.com/lukyky414/Projet_Assm

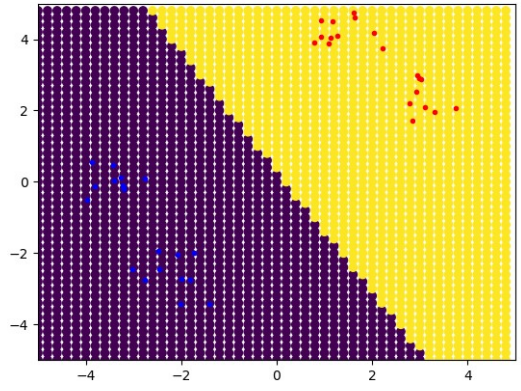
A chaque question où du code est donné, je donne l'ID du commit pour pouvoir facilement suivre.

I. Classification SVM

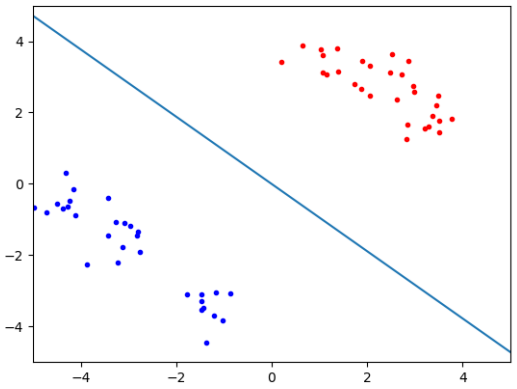
1. Cas séparable

2) Complétez la fonction monprogramme pour apprendre l'hyperplan de séparation optimal sur lesdonnées et visualiser la classification du plan obtenue au travers d'une grille de points de test.

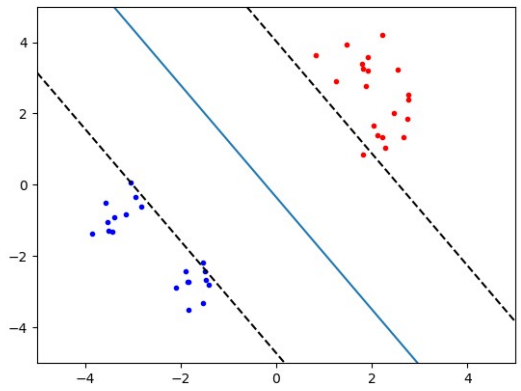




3) Tracez la droite de séparation



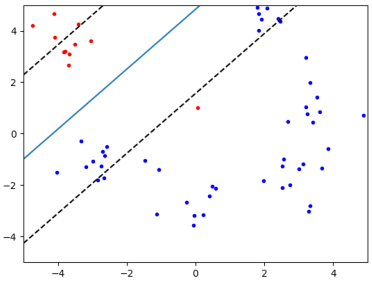
4) Calculez la marge Δ et tracez en tirets les frontières de la marge.



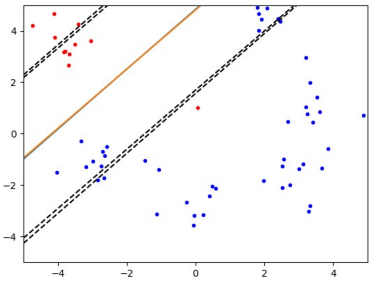
2. Cas non séparable

2) Utilisez maintenant une SVM "à marge souple", avec une valeur de C finie et tracez les mêmes droites que précédemment.

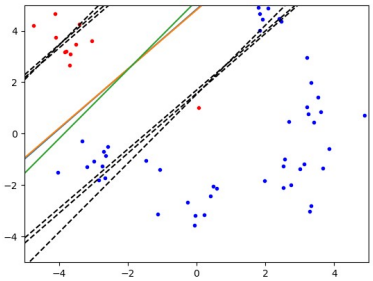
C=1



C=2

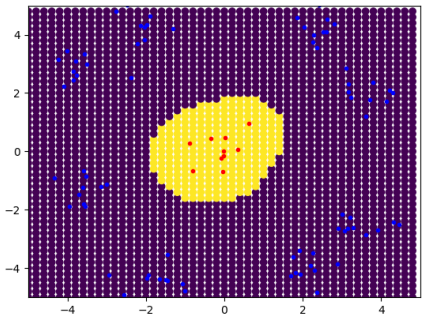


C=64



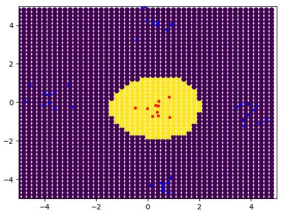
3. Classification non linéaire

1) Utilisez maintenant un noyau gaussien.

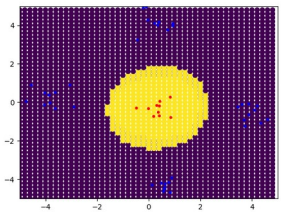


Le C ne fait varier que très peu la patate. Par contre, pour sigma:

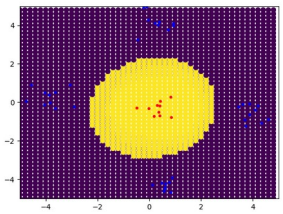
Sigma = 1



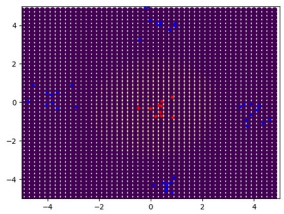
Sigma = 2



Sigma = 4



Sigma = 8



Arrivé à une certaine valeur, la patate pour la classe rouge disparaît entièrement. Avant cela elle grossit rapidement avec sigma (comparé au changement de ‘c’ qui ne fait pas grand-chose)

II. Problème multi-classe et techniques de validation croisée : applica-tion à la classification de défauts de rails

1. Classifieurs binaires

1 & 2) Écrivez une boucle pour créez les 4 classifieursbinaires SVM linéaires et calculer, pour chaque classifieur binaire séparément, son taux de reconnaissance sur la base d'apprentissage

J'ai crée une classe "Mon Classifieur" qui garde les données neccéssaires et facilite l'écriture des boucles. Voici ma classe à cet étape du TP: (git checkout e5c3)

```
class Mon_Classifieur():
        def __init__(self, Xapp, Yapp, nb_classes):
                self.Xapp = Xapp
                self.Yapp = Yapp
                self.nb_classes = nb_classes
                self.nb_point = Xapp.shape[0]
                self.classifieurs = []


        def apprentissage(self):
                Y = np.ones((self.nb_classes, self.nb_point))
                for i in range(self.nb_classes - 1):
                        self.classifieurs.append(svm.LinearSVC(C = 1, max_iter = 10000))

                        Y[i,self.Yapp!=(i+1)] = -1

                        self.classifieurs[i].fit(self.Xapp, Y[i])
```

2. Combinaison des classifieurs binaires

J'ai ajouté à ma classe une méthode prédiction:

```
        def prediction(self, Xtest):
                all_pred = np.zeros((Xtest.shape[0], self.nb_classes))

                for i in range(self.nb_classes - 1):
                        all_pred[:,i] = self.classifieurs[i].decision_function(Xtest)

                        Ypred = np.argmax(all_pred, axis=1)+1

                        return Ypred
```

3. Estimation de l'erreur de généralisation par validation croisée

Voici ma boucle pour le LOO (git checkout df7f)

```
for j in range(self.nb_classes):
        #Créer un classifieur
        self.classifieurs.append(svm.LinearSVC(C = 1, max_iter = 10000))

err = 0
all_err = np.zeros((self.nb_classes,1))
for one_out in range(self.nb_point):
        #Afficher une barre de chargement[...]

        Xone_out = np.delete(self.Xapp, one_out, axis=0)
        Yone_out = np.delete(self.Yapp, one_out, axis=0)



        for j in range(self.nb_classes):
                #Tous les points sont la classe à rechercher
                Y = np.ones((self.nb_classes, self.nb_point-1))
                #Sauf ceux dont le Yapp != numéro du classifieur
                Y[j,Yone_out!=(j+1)] = -1

                #Lancer l'apprentissage
                self.classifieurs[j].fit(Xone_out, Y[j])

                Ypred = self.classifieurs[j].predict(self.Xapp)
                if (Ypred[one_out] == 1 and self.Yapp[one_out] != j+1) or (Ypred[one_out] == 0 and self.Yapp[one_out]
                                                                                                        == j+1) :
                        all_err[j] += 1

        Ypred = self.prediction(self.Xapp)



        if Ypred[one_out] != self.Yapp[one_out] :
                err += 1

err = err/self.nb_point
all_err = np.divide(all_err, self.nb_point)
print()
print("L'erreur du LOO est en moyenne de "+str(int(err*100))+"%")
for i in range(self.nb_classes) :
        print("L'erreur du classifieur " + str(i+1) + " du LOO est en moyenne de "+str(int(all_err[i]*100))+"%")
```

Je n'ai plus les résultats, mais je sais que le classifieurs [0] (donc qui classifie la classe 1) a plus d'erreur que les autres. Ceci s'explique par le fait qu'il y a autant de défaut de type 1, que de défauts des autres types réunis. Plus de cas donc plus d'erreur.

4. Sélection de modèle

1) Proposez une méthode permettant d'effectuer la sélection de modèle.

On peut changer le paramètre C lors de l'apprentissage. Je propose donc que pour chaque étape du LOO, on choisisse la meilleure valeur de C possible. Pour vérifier quelle est la meilleure valeur de C possible, il faut tester chacune des valeurs. Et pour tester, il nous faut un 2e LOO. (git checkout 4130)

```
def apprentissage(self):
    my_file = open("log.txt", "w+")
    err_gen = 0

    #Pour chaque points de la base
    for one_out in range(self.nb_point):

        #Base sans le one out
        Xone_out = np.delete(self.Xapp, one_out, axis=0)
        Yone_out = np.delete(self.Yapp, one_out, axis=0)

        #Permet de séparer la donnée out dans une array 2D
        Xtest_one = np.array([self.Xapp[one_out,:]])
        Ytest_one = self.Yapp[one_out]

        #Choix du meilleur C
        best_c = 0
        best_err = self.nb_point
        for c in [0.01, 0.1, 1, 10, 100]:
            self.classifieurs = []
            for _ in range(nb_classes):
                self.classifieurs.append(svm.LinearSVC(C = c, max_iter = 10000))
            err = 0
            for two_out in range(self.nb_point-1):
                #barre de chargement
                print("Training pour "+str(one_out+1)+"/"+str(self.nb_point)+" [", end='')
                for i in range(20):
                    if i/20 < two_out/(self.nb_point-1) :
                        print("#", end='')
                    else:
                        print(' ', end='')
                print("] C="+str(c)+"      ", end='\r')

                Xtwo_out = np.delete(Xone_out, two_out, axis=0)
                Ytwo_out = np.delete(Yone_out, two_out, axis=0)

                Xtest_two = np.array([Xone_out[two_out,:]])
                Ytest_two = Yone_out[two_out]

                #Classification
                for j in range(self.nb_classes):
                    #Tous les points sont la classe à rechercher
                    Y = np.ones((self.nb_classes, self.nb_point-2))
                    #Sauf ceux dont le Yapp != numéro du classifieur
                    Y[j,Ytwo_out!=(j+1)] = -1

                    #Lancer l'apprentissage
                    self.classifieurs[j].fit(Xtwo_out, Y[j])

                #Calcul de l'erreur multiclasse
                Ypred = self.prediction(Xtest_two)

                if Ypred[0] != Ytest_two :
                    err += 1

            #Garder le meilleur c
            if err < best_err :
                best_c = c
                best_err = err

        my_file.write("i:"+str(one_out)+" - c:"+str(best_c)+" - e:"+str(best_err)+"/"+str(self

        #On a choisi le meilleur C, maintenant il faut apprendre dessus avec le one out
        self.classifieurs = []
        for _ in range(nb_classes):
            self.classifieurs.append(svm.LinearSVC(C = best_c, max_iter = 10000))


        #Classification
        for j in range(self.nb_classes):
            #Tous les points sont la classe à rechercher
            Y = np.ones((self.nb_classes, self.nb_point-1))
            #Sauf ceux dont le Yapp != numéro du classifieur
            Y[j,Yone_out!=(j+1)] = -1

            #Lancer l'apprentissage
            self.classifieurs[j].fit(Xone_out, Y[j])

        #Calcul de l'erreur multiclasse
        Ypred = self.prediction(Xtest_one)

        if Ypred[0] != Ytest_one :
            err_gen += 1

    #Affichage des erreures
    err_gen = err_gen/self.nb_point
    print()
    print("L'erreur du LOO est en moyenne de "+str(int(err*100))+"%")
    my_file.write("L'erreur du LOO est en moyenne de "+str(err)+"")

    my_file.close()
```

J'ai gardé dans un fichier tous les C qui ont étés choisis. C'est majoritairement la valeur 1 qui sort le plus souvent, avec 11 erreurs sur les 139 points du LOO. C'est seulement à la 7^{e} itération que le classifieur trouve un meilleur C à 0,1.

2) Modifiez votre programme pour effectuer la sélection de modèle indépendamment pour chaque catégorie.

Il suffit d'ajouter quelques boucles pour séparer les choix des différents C. Seulement, en étant à un total de 140 x 5 x 139, le programme s'éxecute en un peu plus d'une heure. Là on est à 140 x 625 x 139 (625 = 5^4), et je ne suis même pas arrivé au bout d'une seule itération du 1e LOO, j'ai donc abandonné l'espoir d'avoir un quelconque résultat à cette étape.

3) Considérez simultanément pour optimiser les performances du classifieurmulti-classe global.

Ici on réutilise la boucle dans la question 1 (donc 140 x 5 x 139). Seulement, on garde le meilleur c séparément pour chaque classifieur comme dans la question 2. Je retrouve une éxecution supérieur à 1h mais reste en temps raisonnable.

On peut utiliser ce raisonnement seulement parce que chacun des classifieurs fait du un contre tous. Les paramètres d'un classifieur binaire n'impacte en rien les résultats des autres classifieurs binaires. Il suffit donc d'apprendre une seule fois quel est le meilleur *c*, et non pas plusieurs fois à chaque fois que l'on change le *c* d'un autre classifieur

4) Implémentez cette dernière approche.

(git checkout master)

Mon code à cette étape est assez commenté et clair. Je ne vais donc pas l'expliquer avec des phrases dans ce rapport. A la suite d'un soucis dans ma planification de travail, je n'ai pu faire d'autres questions après celle-ci. Je tiens à m'excuser du retard avec lequel je rend ce projet. Voici donc l'état finale de mon classifieur:

```
def apprentissage(self):
        my_file = open("log.txt", "w+")
        err_gen = 0
        possible_c = [0.01, 0.1, 1, 10, 100]
        #Pour chaque points de la base
        for one_out in range(self.nb_point):
                #Base sans le one out
                Xone_out = np.delete(self.Xapp, one_out, axis=0)
                Yone_out = np.delete(self.Yapp, one_out, axis=0)

                #Permet de séparer la donnée out dans une array 2D
                Xtest_one = np.array([self.Xapp[one_out,:]])
                Ytest_one = self.Yapp[one_out]

                #Choix du meilleur C
                best_c = [0, 0, 0, 0]
                best_err = [self.nb_point, self.nb_point, self.nb_point, self.nb_point]
                counter = 0
                for c in possible_c:
```

```
                counter += 1
                err = [0, 0, 0, 0]
                #Reset les classifieurs
                self.classifieurs = []
                for _ in range(self.nb_classes):
                        self.classifieurs.append(svm.LinearSVC(C = c, max_iter = 100000))
                #Calcul de l'erreur
                for two_out in range(self.nb_point-1):
                        #barre de chargement [... ]
                        Xtwo_out = np.delete(Xone_out, two_out, axis=0)
                        Ytwo_out = np.delete(Yone_out, two_out, axis=0)
                        Xtest_two = np.array([Xone_out[two_out,:]])
                        Ytest_two = Yone_out[two_out]

                        #Classification
                        for j in range(self.nb_classes):
                                #Tous les points sont la classe à rechercher
                                Y = np.ones((self.nb_classes, self.nb_point-2))
                                #Sauf ceux dont le Yapp != numéro du classifieur
                                Y[j,Ytwo_out!=(j+1)] = -1

                                #Lancer l'apprentissage
                                self.classifieurs[j].fit(Xtwo_out, Y[j])
                                #Calcul de l'erreur séparément pour chaque classifieur
                                Ypred = self.classifieurs[j].predict(Xtest_two)
                                if ( Ypred[0] == 1 and Ytest_two != (j+1) ) or ( Ypred[0] == 0 and Ytest_two
                                                                                 == (j+1) ) :
                                        err[j] += 1
                #Garder le meilleur c pour chaque classifieurs
                for j in range(self.nb_classes):
                        if err[j] < best_err[j] :
                                best_c[j] = c
                                best_err[j] = err[j]

        my_file.write("i:"+str(one_out)+" - c:"+str(best_c)+" - e:"+str(best_err)+"\n")
        #On a choisi le meilleur C, maintenant il faut apprendre dessus avec le one out
        self.classifieurs = []
        for j in range(nb_classes):
        self.classifieurs.append(svm.LinearSVC(C = best_c[j], max_iter = 10000))

        #Classification
        for j in range(self.nb_classes):
        #Tous les points sont la classe à rechercher
        Y = np.ones((self.nb_classes, self.nb_point-1))
        #Sauf ceux dont le Yapp != numéro du classifieur
        Y[j,Yone_out!=(j+1)] = -1

        #Lancer l'apprentissage
        self.classifieurs[j].fit(Xone_out, Y[j])

        #Calcul de l'erreur multiclasse
        Ypred = self.prediction(Xtest_one)

        if Ypred[0] != Ytest_one :
        err_gen += 1

#Affichage des erreures
err_gen = err_gen/self.nb_point
print()
print("L'erreur du LOO est en moyenne de "+str(int(err*100))+"%")
my_file.write("L'erreur du LOO est en moyenne de "+str(err)+"")

my_file.close()
```