# Aprendizaje Automático y Big Data, Práctica 6

Romain Contrain y Guillermo Martín Sánchez

Mayo 2020

## 1 Objetivo

El objetivo de esta práctica es estudiar el algoritmo de *Support Vector Machines*. Para ello, en una primera parte, estudiamos la clase *SVC* de la librería *sklearn.svm*. Luego hacemos un caso de uso en la clasificación de *emails* en *spam* y *ham*.

## 2 Método

### 2.1 Estudio de SVC

Primero, estudiamos el valor del parámetro $C$ sobre la frontera de decisión en un primer conjunto de datos linealmente separable.

En segundo lugar, al clasificar un segundo conjunto de datos no linealmente separables, usamos un kernel gaussiano en vez de lineal.

Finalmente, para un tercer conjunto de datos, divididos en datos de entrenamiento y de validación, buscamos los valores óptimos de $C$ y $\sigma$ para el modelo. Para ello, entrenamos el SVM para cada posible par de valores de $C, \sigma \in \{0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30\}$ sobre los datos de entrenamiento. Después, calculamos el error de este modelo sobre los datos de validación y nos quedamos con la elección de los hiperparámetros que lo hace mínimo.

### 2.2 ¿Spam o ham?

A continuación estudiamos el uso de SVM sobre un dataset de emails que están clasificados en *spam*, *easy ham* y *hard ham*. Para ello primero, usando el código facilitado, cargamos cada uno de los ejemplos. Con la función *email2TokenList* procesamos cada email, haciendo una transformación que simplifica el mensaje a un conjunto de *tokens* formados por las palabras que aparecen. Luego, obtenemos el diccionario de 1899 palabras con las que vamos a trabajar con la función *getVocabDict* que lo carga del fichero *vocab.txt*. Finalmente, recorremos cada ejemplo, creando un vector $y^{(i)}$ donde $y_j^{(i)} = 1$ si el email $i$ contiene el *token* $vocab(j)$ y $y_j^{(i)} = 0$ si no, $\forall j = 1...1899$.

Una vez tenemos los datos cargados, entrenamos dos modelos.

En el primero, usamos tanto los ejemplos de *spam* como de *easy ham* para entrenar el modelo. Dividimos los ejemplos en 60% de entrenamiento, 30% de validación y 10% de test. Usando kernel gaussiano, buscamos los hiperparámetros $C, \sigma \in \{1, 3, 10, 14, 20, 30\}$ que minimizan el error en los ejemplos de validación y calculamos el *F1-Score* sobre los datos de test. Luego, vemos el error del modelo sobre los datos de test que son *easy ham* y sobre los ejemplos que tenemos de *hard ham*.

En el segundo, hacemos lo mismo pero usando también los ejemplos de *hard ham*. Además de calcular *F1-Score* sobre los datos de test, el error se estudia como el error del modelo sobre los datos de test que son *easy ham* y sobre los que son *hard ham* por separado.

# 3 Resultados

## 3.1 Estudio de SVC

En la Figura 1 vemos el resultado de usar un kernel lineal y $C = 1$ mientras que en la Figura 2 usamos $C = 100$. Como se puede observar cuánto mayor sea la $C$ menos regularización tenemos y se puede sufrir de *overfitting* (como vemos en la figura, la separación artificialmente se inclina para incluir el ejemplo de entrenamiento que se encuentra muy alejado de su categoría).

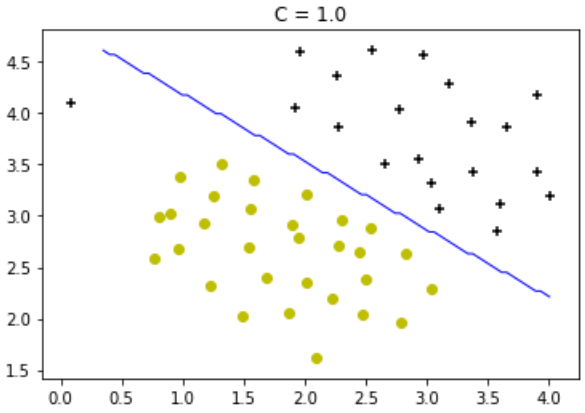


Figure 1: Frontera de decisión para $C = 1$

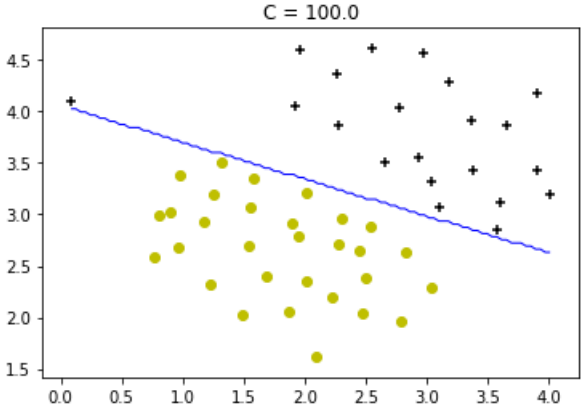


Figure 2: Frontera de decisión para $C = 100$

En la Figura 3 vemos el resultado de aplicar el algoritmo con un kernel gaussiano con $C = 1$ y $\sigma = 0.1$. Como se pude observar, gracias a este kernel podemos separar datos que no son linealmente separables.

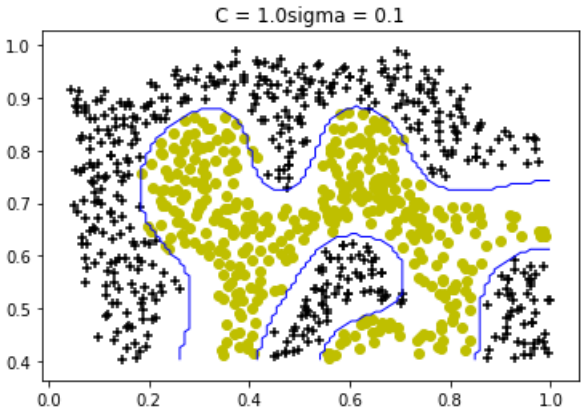


Figure 3: Frontera de decisión con kernel gaussiano para $C = 1$ y $\sigma = 0.1$

Finalmente, respecto al tercer conjunto de valores, en la Figura 4 vemos el resultado de usar un kernel gaussiano con los valores óptimos encontrados: $C = 1$ y $\sigma = 0.1$ para separar los ejemplos de validación.

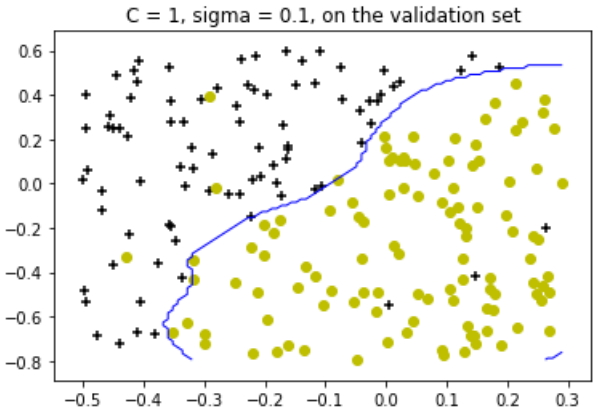


Figure 4: Frontera de decisión con kernel gaussiano para $C = 1$ y $\sigma = 0.1$ sobre los datos de validación

### 3.2 ¿Spam o ham?

Para el primer caso, en el que solo entrenamos con *spam* y con *easy ham*, obtenemos que los valores óptimos de los que probamos son $C = 30$ y $\sigma = 10$. Para estos valores, obteneos un *F1-Score* del $\approx 94\%$. Sin embargo, se ve claro que se ha entrenado sólo sobre *easy ham* puesto que el error a ejemplos de test de *easy ham* es $\approx 0.76\%$ mientras que sobre ejemplos de *hard ham* es 51.2%, mucho mayor (de hecho, el modelo no es mejor que tirar una moneda al aire para decidir).

En el segundo caso, entrenando también con *hard ham*, obtenemos que los valores óptimos son $C = 10$ y $\sigma = 10$. Aunque nuestro *F1-Score* es casi igual ($\approx 93\%$) y el error a ejemplos de test de *easy ham* también ($\approx 0.78\%$); el error a ejemplos de test de *hard ham* ha disminuido considerablemente a un $\approx 14.29\%$.

## 4 Conclusiones

*Support Vector Machines* es una técnica de clasificación supervisada muy potente para separar tanto datos linealmente separables (usando un kernel lineal), como datos que no lo son (usando otro kernel). Para modular los problemas de *high bias* y *high variance* ahora tenemos el parámetro de regularización $C$ que funciona al contrario que $\lambda$.

Hemos usado lo que hemos aprendido de este método, el estudio de hiperparámetros óptimos usando los datos de validación, y el estudio de errores usando los datos de test para entrenar un modelo de clasificación que funciona bien para clasificar *spam* de *ham*

# 5 Código

```
"""
Coursework 6: Support Vector Machines
"""

#%%
"""
Imports and definitions
"""



import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from get_vocab_dict import getVocabDict
from process_email import email2TokenList
from sklearn.svm import SVC
import codecs
import time

def load_dat(file_name):
    """
    carga el fichero dat (Matlab) especificado y lo
    devuelve en un array de numpy
    """
    data = loadmat(file_name)
    y = data['y']
    X = data['X']
    return X,y

def load_email_data(folder_name, numExamples):
    """
    Loads the data from a folder containing email examples
    and transforms them in a vectorized form suitable for sum training
    parameters :
        - folder_name string with the name of the folder containing the examples
        - numExamples the number of examples in the folder
    returns:
        - X a matrix of numExamples rows containing a 1 at coefficient (i,j)
          iff the (i-1)th example contained word (j-1) of the dictionnary
    """
    X = np.zeros((numExamples,nwords))
    for i in range(numExamples) :
        email_contents = codecs.open(folder_name + '/{:0>4d}.txt'.format(i+1),
                                     'r', encoding='utf-8', errors='ignore').read()
```

```
        email = email2TokenList(email_contents)
        for word in email :
            index = dictionnary.get(word)
            if index != None :
                X[i,index-1] = 1

    return X

def f1score(prediction,ytest):
    """
    Computes the f1 score of a model
    given the predictions it makes on a set
    and the true values on that set
    """
    truepos = np.sum(ytest[prediction == 1])
    predpos = np.sum(prediction)
    actpos = np.sum(ytest)
    precision = truepos/predpos
    recall = truepos/actpos
    return 2*precision*recall/(precision + recall)


#%%
"""
part 1.1 : linear kernel
"""

# Loading the data and setting the parameters
X,Y = load_dat("ex6data1.mat")
Y = Y.ravel()
pos = np.where(Y==1)
not_pos = np.where(Y==0)
C = 100.

# Trainig the model with a linear kernel
svm = SVC(kernel='linear', C=C)
svm.fit(X,Y)

# Computing the decision border
x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
x2_min, x2_max = X[:, 1].min(), X[:, 1].max()
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max,100),
                       np.linspace(x2_min, x2_max,100))
h = svm.predict(np.c_[xx1.ravel(),xx2.ravel()])
h = h.reshape(xx1.shape)
```

```
# Drawing the decision border along with the data
plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k')
plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y')
plt.title('C = '+ str(C))
plt.show()

#%%
"""
part 1.2 : gaussian kernel
"""

# Loading the data and setting the parameters
X,Y = load_dat("ex6data2.mat")
Y = Y.ravel()
pos = np.where(Y==1)
not_pos = np.where(Y==0)
C = 1.
sigma = 0.1

# Trainig the model with a gaussian kenel
svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
svm.fit(X,Y)

# Computing the decision border
x1_min, x1_max = X[:, 0].min(), X[:, 0].max()
x2_min, x2_max = X[:, 1].min(), X[:, 1].max()
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max,100),
                       np.linspace(x2_min, x2_max,100))
h = svm.predict(np.c_[xx1.ravel(),xx2.ravel()])
h = h.reshape(xx1.shape)

# Drawing the decision border along with the data
plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
plt.scatter(X[pos, 0],X[pos, 1], marker='+', c='k')
plt.scatter(X[not_pos, 0], X[not_pos, 1], marker='o', c='y')
plt.title('C = '+ str(C) + 'sigma = ' + str(sigma))
plt.show()

#%%
"""
part 1.3 : Choice of C and sigma
"""

# Loading the data
X,Y = load_dat("ex6data3.mat")
```

```
Y = Y.ravel()
data = loadmat("ex6data3.mat")
Xval = data['Xval']
Yval = data['yval']
Yval = Yval.ravel()
mval = np.size(Yval)


# Looking for the best values for C and sigma
hpvalues = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

minerrorval = float('Inf')

for C in hpvalues :
    for sigma in hpvalues :
        svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
        svm.fit(X,Y)

        prediction = svm.predict(Xval)
        error = 1 - np.sum(prediction == Yval) / mval
        # Choosing the parameters with the smallest validation error
        if (error < minerrorval) :
            minerrorval = error
            Cmin = C
            sigmamin = sigma
            svmmin = svm

print("Best value for C : " + str(Cmin))
print("Best value for sigma : " + str(sigmamin))

pos = np.where(Yval==1)
not_pos = np.where(Yval==0)

# Computing the decision border
x1_min, x1_max = Xval[:, 0].min(), Xval[:, 0].max()
x2_min, x2_max = Xval[:, 1].min(), Xval[:, 1].max()
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max,100),
                       np.linspace(x2_min, x2_max,100))
h = svmmin.predict(np.c_[xx1.ravel(),xx2.ravel()])
h = h.reshape(xx1.shape)

# Drawing the decision border along with the data of the validation set
plt.contour(xx1, xx2, h, [0.5], linewidths=1, colors='b')
plt.scatter(Xval[pos, 0],Xval[pos, 1], marker='+', c='k')
plt.scatter(Xval[not_pos, 0], Xval[not_pos, 1], marker='o', c='Y')
plt.title('C = '+ str(Cmin) + ', sigma = ' + str(sigmamin) + ", on the validation set")
```

```
plt.show()

#%%
"""
part 2 : Spam detection
"""

# Loading the data
dictionnary = getVocabDict()
nwords = len(dictionnary)

Xspam = load_email_data('spam',500)
yspam = np.ones((500,1))
espam = np.zeros((500,1))
Xeham = load_email_data('easy_ham',2551)
yeham = np.zeros((2551,1))
eeham = np.ones((2551,1))
Xhham = load_email_data('hard_ham',250)
yhham = np.zeros((250,1))
ehham = np.ones((250,1))*2


#%%
"""
Training a model on spam and easy ham
"""

print("Training on spam and easy ham")

np.random.seed(0) # To get consistent results while debugging

# Shuffling the datasets
X = np.concatenate((Xspam,Xeham), axis = 0)
Y = np.concatenate((yspam,yeham), axis = 0)
mat = np.concatenate((X,Y), axis = 1)
np.random.shuffle(mat)

# Dividing the set into training, validation and test sets
m = np.shape(mat)[0]
train = int(np.floor(0.6*m))
val = int(train + np.floor(0.3*m))

Xtrain = mat[0:train,:-1]
ytrain = mat[0:train,-1]
Xval = mat[train:val,:-1]
yval = mat[train:val,-1]
```

```
Xtest = mat[val::,:-1]
ytest = mat[val::,-1]


#%%

# Looking for the best values for C and sigma

mval = np.shape(Xval)[0]

hpvalues = [1, 3, 10, 14, 20, 30] # In previous tests the result was 10 or 30

minerrorval = float('Inf')

for C in hpvalues:
    for sigma in hpvalues:
        before = time.process_time()
        svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
        svm.fit(Xtrain,ytrain)

        prediction = svm.predict(Xval)
        error = 1 - np.sum(prediction == yval) / mval
        # Choosing the hyperparameters with the smallest validation error
        if (error < minerrorval) :
            minerrorval = error
            Cmin = C
            sigmamin = sigma
            svmmin = svm
        after = time.process_time()
        #print(after-before)

print("Best value for C : " + str(Cmin))
print("Best value for sigma : " + str(sigmamin))

#%%

# Giving C and sigma the values the previous cell returned
# manually so that we don't have to rerun it all the time,
# as it takes a long time
C = 30
sigma = 10

svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
svm.fit(Xtrain,ytrain)

prediction = svm.predict(Xtest)
```

```
f1s = f1score(prediction,ytest)
print("F1-Score of test values: " + str(f1s))

# Testing if the hard ham is harder than the easy one
predeham = svm.predict(Xtest[ytest == 0])
error = np.sum(predeham)/np.size(predeham)
print("Error on easy ham: " + str(error))

predhham = svm.predict(Xhham)
error = np.sum(predhham)/np.size(predhham)
print("Error on hard ham: " + str(error))


#%%
"""
Training a model on spam, easy ham and hard ham
"""

print("Training on spam, easy ham and hard ham")

# Shuffling the datasets
X = np.concatenate((Xspam,Xeham,Xhham), axis = 0)
Y = np.concatenate((yspam,yeham,yhham), axis = 0)
E = np.concatenate((espam,eeham,ehham), axis = 0)
mat = np.concatenate((X,Y,E), axis = 1)
np.random.shuffle(mat)

# Dividing the set into training, validation and test sets
m = np.shape(mat)[0]
train = int(np.floor(0.6*m))
val = int(train + np.floor(0.3*m))

Xtrain = mat[0:train,:-2]
ytrain = mat[0:train,-2]
Xval = mat[train:val,:-2]
yval = mat[train:val,-2]
Xtest = mat[val::,:-2]
ytest = mat[val::,-2]
eashard = mat[val::,-1]

#%%

# Looking for the best values for C and sigma

mval = np.shape(Xval)[0]
```

```
hpvalues = [1, 3, 10, 14, 20, 30] # In previous tests the result was 10 or 30

minerrorval = float('Inf')

for C in hpvalues:
    for sigma in hpvalues:
        before = time.process_time()
        svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
        svm.fit(Xtrain,ytrain)

        prediction = svm.predict(Xval)
        error = 1 - np.sum(prediction == yval) / mval
        # Choosing the hyperparameters with the smallest validation error
        if (error < minerrorval) :
            minerrorval = error
            Cmin = C
            sigmamin = sigma
            svmmin = svm
        after = time.process_time()
        #print(after-before)

print("Best value for C : " + str(Cmin))
print("Best value for sigma : " + str(sigmamin))

#%%

# Giving C and sigma the values the previous cell returned
# manually so that we don't have to rerun it all the time,
# as it takes a long time
C = 10
sigma = 10

svm = SVC(kernel='rbf', C=C, gamma = 1/(2*sigma**2))
svm.fit(Xtrain,ytrain)

prediction = svm.predict(Xtest)
f1s = f1score(prediction,ytest)
print("F1-Score of test values: " + str(f1s))

# Testing if the effect of including hard ham on false positives
predeham = svm.predict(Xtest[eashard == 1])
error = np.sum(predeham)/np.size(predeham)
print("Error on easy ham: " + str(error))

predhham = svm.predict(Xtest[eashard == 2])
error = np.sum(predhham)/np.size(predhham)
```

```
print("Error on hard ham: " + str(error))
```