Aprendizaje Automático y minería de datos: Práctica 6

Autores: *Tatiana Duarte Balvís, Miguel Mur Cortés*

1. Objetivo:

El objetivo de la práctica es aplicar el clasificador SVM que incorpora sckit-learn.

2. Kernel lineal:

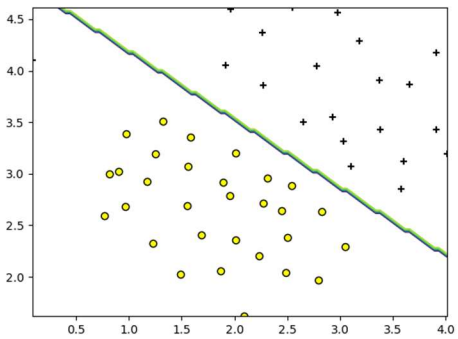
En este apartado, utilizamos el clasificador SVM utilizando un parámetro de regularización C, aplicando una función de kernel sobre un conjunto de datos de entrenamiento X.

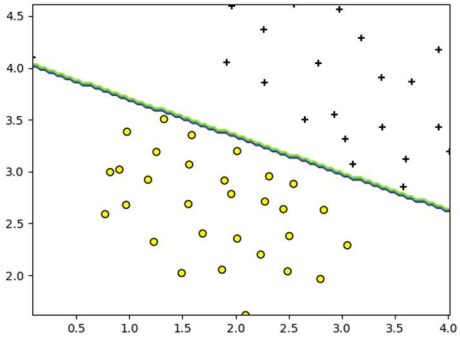
```
def kernel_lineal(c):
    datos = loadmat("ex6data1.mat")
    X, y = datos["X"], datos["y"].ravel()

    svm = SVC(kernel="linear", C=c)
    svm.fit(X, y)

    nombreFigura = f"kernelLineal_c{c}"
    visualize_boundary(X, y, svm, nombreFigura)
```

Para representar la separación, lo visualizamos de esta manera:

```
def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)

    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(X[neg, 0], X[neg, 1], color='yellow', edgecolors='black',
marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(f"imgs/{file_name}.png")
    plt.close()
```

Con lo que obtenemos estas figuras para c = 1 y c = 100 respectivamente:



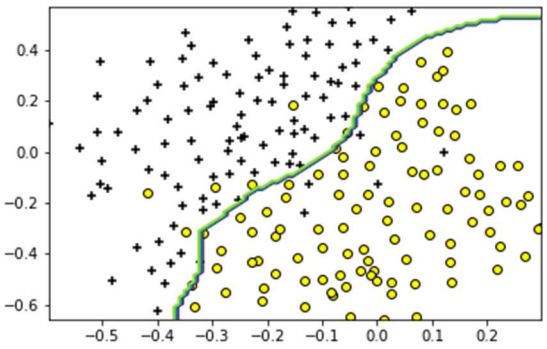


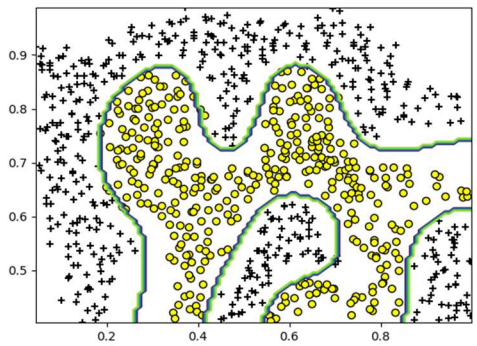
2.2 Kernel lineal:

Para conjuntos de datos más complejos que no se pueden separar de forma lineal, utilizamos el kernel gaussiano.

```
def kernel_gaussiano(c, sigma):
    datos = loadmat("ex6data2.mat")
    X, y = datos["X"], datos["y"].ravel()

    svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
    svm.fit(X, y)

    nombreFigura = f"kernelGaussiano_c{c}_s{sigma}"
    visualize_boundary(X, y, svm, nombreFigura)
```

Y utilizando la función auxiliar, se obtiene esta figura:





2.2 Elección de parámetros C y σ

A continuación, elaboramos un método para seleccionar los valores óptimos C y σ para un modelo SVM con kernel gaussiano para un conjunto de datos.

Elegiremos los valores de C y σ entre los valores del conjunto $0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30$ Además, este conjunto de datos presenta unos valores Xval e yval de validación con lo que calcularemos el porcentaje de ejemplos que clasifica correctamente.

```
def eleccion_params():
    datos = loadmat("ex6data3.mat")
    X, y = datos["X"], datos["y"].ravel()
    Xval, yval = datos["Xval"], datos["yval"].ravel()

    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))

    i = 0
    for c in C_vec:
        j = 0
        for sigma in sigma_vec:
            svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
            svm.fit(X, y)
            scores[i,j] = svm.score(Xval, yval)
            j += 1
        i += 1

    cOptima = C_vec[scores.argmax() // len(sigma_vec)]
    sigmaOptima = sigma_vec[scores.argmax() % len(sigma_vec)]
    minError = 1 - scores.max()
    print(f"Eleccion de parametros kernel gaussiano: min error =
{str(minError)[:5]}")
    print(f"cOptima = {str(cOptima)}. sigmaOptima = {str(sigmaOptima)}")

    svm = SVC(kernel="rbf", C=cOptima, gamma=1 / (2 * sigmaOptima**2))
    svm.fit(X, y)

    nombreFigura = f"kernelGaussiano_ElecParam_c{cOptima}_s{sigmaOptima}"
    visualize_boundary(X, y, svm, nombreFigura)
```

Con lo que obtenemos estos resultados:

```
Eleccion de parametros kernel gaussiano: min error = 0.035
cOptima = 1. sigmaOptima = 0.1
```

3. Detección de spam:

En esta parte, realizaremos experimentos en la detección de correo de spam utilizando las funciones para el cálculo de modelos SVM.

Primero, cargamos los datos.

```
def generar_datos():
    diccionario = getVocabDict()
    X_spam, aux = procesar_emails("spam", diccionario)
    y_spam = np.ones_like(aux)
    X_easy, aux = procesar_emails("easy_ham", diccionario)
    y_easy = np.zeros_like(aux)
    X_hard, aux = procesar_emails("hard_ham", diccionario)
    y_hard = np.zeros_like(aux)

    X = np.concatenate((X_spam, X_easy, X_hard), axis=0)
    y = np.concatenate([y_spam, y_easy, y_hard])

    #generar datos de entrenamiento, validacion y test (0.6/0.2/0.2)
    Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2,
random_state=1)
    Xtrain, Xval, ytrain, yval = train_test_split(Xtrain, ytrain,
test_size=0.2, random_state=1)

    return Xtrain, ytrain, Xval, yval, Xtest, ytest
```

Y los procesamos utilizando la función proporcionada **email2TokenList**

```
def procesar_emails(nombreArchivo, diccionario):
    aux = "emails/" + nombreArchivo + "/*.txt"
    #cualquier archivo que esté en el direcctorio "emails/nombreArchivo" y
tenga extension .txt
    mensajes = glob.glob(aux)
    X = np.zeros((len(mensajes), len(diccionario)))
    y_template = np.zeros(len(mensajes))
    i = 0
    for m in mensajes:
        email_contents = codecs.open(m, "r", encoding="utf-8", errors="ig-
nore").read()
        tokens = email2TokenList(email_contents)
        words = filter(None, [diccionario.get(x) for x in tokens])

        for w in words: #se marcan las palabras que aparecen en la matriz X
con un 1
            X[i, w-1] = 1
        i+=1
    return X, y_template
```

Y por último, aplicamos el método de selección de parámetros C y σ y el kernel gaussiano.

```
def deteccion_spam():
    Xtrain, ytrain, Xval, yval, Xtest, ytest = generar_datos()

    #seleccion de parametros C y sigma
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))

    i = 0
    for c in C_vec:
        j = 0
        for sigma in sigma_vec:
            svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
            svm.fit(Xtrain, ytrain)
            scores[i,j] = svm.score(Xval, yval)
            j += 1
        i += 1

    cOptima = C_vec[scores.argmax() // len(sigma_vec)]
    sigmaOptima = sigma_vec[scores.argmax() % len(sigma_vec)]
    minError = 1 - scores.max()
    print()
    print(f"Detección de spam: min error = {str(minError)[:5]}")
    print(f"C óptima: {str(cOptima)} ; sigma óptima: {str(sigmaOptima)}")

    #kernel gaussiano
    svm = SVC(kernel="rbf", C=cOptima, gamma=1 / (2 * sigmaOptima**2))
    svm.fit(Xtrain, ytrain)
    resFinal = svm.score(Xtest, ytest)
    print(f"Spam clasificado correctamente: {str(resFinal*100)[:5]}%")
    print()
```

El resultado que obtenemos al ejectuarlo es:

```
Detección de spam: min error = 0.015
C óptima: 10 ; sigma óptima: 10
Spam clasificado correctamente: 98.94%
```

4. Código:

```
from scipy.io import loadmat
import scipy.optimize as opt
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC

from get_vocab_dict import getVocabDict
from process_email import email2TokenList

import glob
import codecs
from sklearn.model_selection import train_test_split


def visualize_boundary(X, y, svm, file_name):
    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)

    pos = (y == 1).ravel()
    neg = (y == 0).ravel()
    plt.figure()
    plt.scatter(X[pos, 0], X[pos, 1], color='black', marker='+')
    plt.scatter(X[neg, 0], X[neg, 1], color='yellow', edgecolors='black',
marker='o')
    plt.contour(x1, x2, yp)
    plt.savefig(f"imgs/{file_name}.png")
    plt.close()

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def kernel_lineal(c):
    datos = loadmat("ex6data1.mat")
    X, y = datos["X"], datos["y"].ravel()

    svm = SVC(kernel="linear", C=c)
    svm.fit(X, y)

    nombreFigura = f"kernelLineal_c{c}"
    visualize_boundary(X, y, svm, nombreFigura)


def kernel_gaussiano(c, sigma):
    datos = loadmat("ex6data2.mat")
    X, y = datos["X"], datos["y"].ravel()
```

```
    svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
    svm.fit(X, y)

    nombreFigura = f"kernelGaussiano_c{c}_s{sigma}"
    visualize_boundary(X, y, svm, nombreFigura)


def eleccion_params():
    datos = loadmat("ex6data3.mat")
    X, y = datos["X"], datos["y"].ravel()
    Xval, yval = datos["Xval"], datos["yval"].ravel()

    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))

    i = 0
    for c in C_vec:
        j = 0
        for sigma in sigma_vec:
            svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
            svm.fit(X, y)
            scores[i,j] = svm.score(Xval, yval)
            j += 1
        i += 1

    cOptima = C_vec[scores.argmax() // len(sigma_vec)]
    sigmaOptima = sigma_vec[scores.argmax() % len(sigma_vec)]
    minError = 1 - scores.max()
    print(f"Eleccion de parametros kernel gaussiano: min error = {str(mi-
nError)[:5]}")
    print(f"cOptima = {str(cOptima)}. sigmaOptima = {str(sigmaOptima)}")

    svm = SVC(kernel="rbf", C=cOptima, gamma=1 / (2 * sigmaOptima**2))
    svm.fit(X, y)

    nombreFigura = f"kernelGaussiano_ElecParam_c{cOptima}_s{sigmaOptima}"
    visualize_boundary(X, y, svm, nombreFigura)

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def procesar_emails(nombreArchivo, diccionario):
    aux = "emails/" + nombreArchivo + "/*.txt"
    #cualquier archivo que esté en el direcctorio "emails/nombreArchivo" y tenga
extension .txt
    mensajes = glob.glob(aux)

    X = np.zeros((len(mensajes), len(diccionario)))
```

```
    y_template = np.zeros(len(mensajes))

    i = 0
    for m in mensajes:
        email_contents = codecs.open(m, "r", encoding="utf-8", errors="ig-
nore").read()
        tokens = email2TokenList(email_contents)
        words = filter(None, [diccionario.get(x) for x in tokens])

        for w in words: #se marcan las palabras que aparecen en la matriz X con un
1
            X[i, w-1] = 1

        i+=1

    return X, y_template


def generar_datos():
    diccionario = getVocabDict()

    X_spam, aux = procesar_emails("spam", diccionario)
    y_spam = np.ones_like(aux)
    X_easy, aux = procesar_emails("easy_ham", diccionario)
    y_easy = np.zeros_like(aux)
    X_hard, aux = procesar_emails("hard_ham", diccionario)
    y_hard = np.zeros_like(aux)

    X = np.concatenate((X_spam, X_easy, X_hard), axis=0)
    y = np.concatenate([y_spam, y_easy, y_hard])

    #generar datos de entrenamiento, validacion y test (0.6/0.2/0.2)
    Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, ran-
dom_state=1)
    Xtrain, Xval, ytrain, yval = train_test_split(Xtrain, ytrain, test_size=0.2,
random_state=1)

    return Xtrain, ytrain, Xval, yval, Xtest, ytest


def deteccion_spam():
    Xtrain, ytrain, Xval, yval, Xtest, ytest = generar_datos()

    #seleccion de parametros C y sigma
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))

    i = 0
```

```
    for c in C_vec:
        j = 0
        for sigma in sigma_vec:
            svm = SVC(kernel="rbf", C=c, gamma=1 / (2 * sigma**2))
            svm.fit(Xtrain, ytrain)
            scores[i,j] = svm.score(Xval, yval)
            j += 1
        i += 1

    cOptima = C_vec[scores.argmax() // len(sigma_vec)]
    sigmaOptima = sigma_vec[scores.argmax() % len(sigma_vec)]
    minError = 1 - scores.max()
    print()
    print(f"Detección de spam: min error = {str(minError)[:5]}")
    print(f"C óptima: {str(cOptima)} ; sigma óptima: {str(sigmaOptima)}")

    #kernel gaussiano
    svm = SVC(kernel="rbf", C=cOptima, gamma=1 / (2 * sigmaOptima**2))
    svm.fit(Xtrain, ytrain)
    resFinal = svm.score(Xtest, ytest)
    print(f"Spam clasificado correctamente: {str(resFinal*100)[:5]}%")
    print()

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def main():
    # kernel_lineal(1.0)
    # kernel_lineal(100.0)
    # kernel_gaussiano(1.0, 0.1)
    # eleccion_params()

    # spam.zip: 500 mensajes
    # easy_ham.zip: 2551 mensajes
    # hard_ham.zip: 250 mensajes
    deteccion_spam()


main()
```