Universidad Complutense de Madrid

Facultad de Informática

Aprendizaje Automático y Big Data

Memoria Práctica 6:

Support Vector Machines

Profesor:

- Alberto Díaz Esteban.

Alumnos:

- Marina de la Cruz López.

- Diego Alejandro Rodríguez Pereira.

**Introducción.**

El objetivo de esta práctica es familiarizarse con los SVM (Support Vector Machine) que incorpora la librería de Python sckit-learn y utilizar sus métodos para la elaboración de la resolución de dos problemas. Implementaremos un kernel lineal, luego utilizaremos un kernel gaussiano, finalizando con la elección de los parámetros de entrada del algoritmo. Por último, aplicaremos todas estas funciones implementadas en esta parte para resolver el problema de detección de correos electrónicos de spam.

**1. Implementación Support Vector Machine**

El objetivo de este primer apartado de la práctica es el aprender y familiarizarse con el entorno de sckit-learn y el uso de los clasificadores que implementan SVM.

En primer lugar implementamos el kernel lineal, en el que se crea una instancia del clasificador SVM de la clase *sklearn.svm.SVC*, pasando como parámetros una constante de regularización *C* con valor igual a 1. Luego también se pasa como parámetro la función kernel RBF, a su vez se considera que dos números puntos flotantes son iguales si estos tiene una diferencia menor de 0.001 que la pasaremos con el parámetro tol, y a su vez se le indicará un máximo de 1.

Este clasificador se ejecuta posteriormente con los datos de entrenamiento, que son la variable X que se leen del fichero *ex6data1.mat,* etiquetados con un vector de Y que contiene 0s y 1s.

Chart, scatter chart

Description automatically generated

Figura 1

Chart, scatter chart

Description automatically generated

Figura 2

En estas dos figuras mostradas, figuras 1 y 2, se pueden observar con claridad que los datos del primer conjunto son separables linealmente. Luego de acatar a esta observación se comprueba el efecto del parámetro C en el ajuste de los datos de entrenamiento utilizando el kernel Lineal. La figura 1 pertenece a C = 1, y la figura 2 pertenece a C = 2.

Para visualizar estos dos gráficos se ha usado la siguiente función proporcionada por el profesor:

Text

Description automatically generated

**Kernel Gaussiano**

Después de esta implementación pasamos a implementar el clasificador SVM que utiliza un kernel Guassiano. Para esta implementación se ha usado un fichero distinto, en este caso se ha utilizado el *ex6data2.mat*, que funciona para datos que no son linealmente separables, como lo vamos a ver en las siguientes gráficas.

Chart, scatter chart

Description automatically generated

Figura 3

Para esta implementación se ha utilizado la función de kernel gaussiano que calcula la distancia entre dos ejemplos de entrenamiento. Para este caso no tenemos que calcular ni implementar la función, ya que la propia librería de sckit-learn lo implementa y lo calcula por nosotros.

A picture containing table

Description automatically generated

Por último, para la implementación del clasificador SVM con el kernel gaussiano, hemos utilizados los siguientes parámetros: C = 1, sigma = 0.1.

Se ha obtenido el siguiente resultado:

Scatter chart, qr code

Description automatically generated

Figura 4

**Elección de los parámetros C y sigma**

En este apartado se seleccionaran los valores de C y sigma para un modelo de SVM con kernel guassiano que clasifica un tercer conjunto de datos que se leerán en el conjutno de datos *ex6data3.mat*. En este conjunto de datos, además de los ejemplos de entrenamiento que hemos estado trabajando X e Y, se incluyen los datos de los ejemplos de validación Xval e Yval que servirán para evaluar el modelo aprendido por nuestro algoritmo.

Para leer el fichero se utiliza la función de la librería de *spicy: spicy.io.loadmat* en el que so obtendrá un diccionario que contiene los valores de las claves X, y, Xval e yval.

Se generarán distintos modelos para los diferentes valores de C y sigma del conjunto: 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, generando entonces 64 distintos modelos. Cada modelo se evualua sobre el conjunto de datos de validación Xval e yval, calculando el porcentaje de los ejemplos de entrenamiento clasificados correctamente.

Chart, scatter chart

Description automatically generated

Figura 5

Se han obtenido como resultado los siguientes porcentajes de aciertos, para cada uno de los valores de C y sigma indicados:

41.23222748815166 C = 0.01 Sigma 0.01

41.23222748815166 C = 0.01 Sigma 0.03

41.23222748815166 C = 0.01 Sigma 0.1

41.23222748815166 C = 0.01 Sigma 0.3

41.23222748815166 C = 0.01 Sigma 1

41.23222748815166 C = 0.01 Sigma 3

41.23222748815166 C = 0.01 Sigma 10

41.23222748815166 C = 0.01 Sigma 30

41.23222748815166 C = 0.03 Sigma 0.01

41.23222748815166 C = 0.03 Sigma 0.03

42.65402843601896 C = 0.03 Sigma 0.1

81.51658767772511 C = 0.03 Sigma 0.3

58.767772511848335 C = 0.03 Sigma 1

41.23222748815166 C = 0.03 Sigma 3

41.23222748815166 C = 0.03 Sigma 10

41.23222748815166 C = 0.03 Sigma 30

41.23222748815166 C = 0.1 Sigma 0.01

41.23222748815166 C = 0.1 Sigma 0.03

89.57345971563981 C = 0.1 Sigma 0.1

86.25592417061611 C = 0.1 Sigma 0.3

78.19905213270142 C = 0.1 Sigma 1

41.23222748815166 C = 0.1 Sigma 3

41.23222748815166 C = 0.1 Sigma 10

41.23222748815166 C = 0.1 Sigma 30

41.23222748815166 C = 0.3 Sigma 0.01

71.56398104265402 C = 0.3 Sigma 0.03

90.99526066350711 C = 0.3 Sigma 0.1

87.67772511848341 C = 0.3 Sigma 0.3

84.36018957345972 C = 0.3 Sigma 1

70.14218009478674 C = 0.3 Sigma 3

41.23222748815166 C = 0.3 Sigma 10

41.23222748815166 C = 0.3 Sigma 30

57.345971563981045 C = 1 Sigma 0.01

85.78199052132702 C = 1 Sigma 0.03

91.4691943127962 C = 1 Sigma 0.1

91.4691943127962 C = 1 Sigma 0.3

87.67772511848341 C = 1 Sigma 1

80.09478672985783 C = 1 Sigma 3

41.23222748815166 C = 1 Sigma 10

41.23222748815166 C = 1 Sigma 30

58.767772511848335 C = 3 Sigma 0.01

84.36018957345972 C = 3 Sigma 0.03

91.4691943127962 C = 3 Sigma 0.1

89.57345971563981 C = 3 Sigma 0.3

88.15165876777252 C = 3 Sigma 1

84.36018957345972 C = 3 Sigma 3

68.24644549763033 C = 3 Sigma 10

41.23222748815166 C = 3 Sigma 30

58.767772511848335 C = 10 Sigma 0.01

84.36018957345972 C = 10 Sigma 0.03

89.0995260663507 C = 10 Sigma 0.1

90.52132701421802 C = 10 Sigma 0.3

88.62559241706161 C = 10 Sigma 1

87.20379146919431 C = 10 Sigma 3

80.09478672985783 C = 10 Sigma 10

41.23222748815166 C = 10 Sigma 30

58.767772511848335 C = 30 Sigma 0.01

84.36018957345972 C = 30 Sigma 0.03

89.0995260663507 C = 30 Sigma 0.1

90.99526066350711 C = 30 Sigma 0.3

87.67772511848341 C = 30 Sigma 1

87.67772511848341 C = 30 Sigma 3

84.36018957345972 C = 30 Sigma 10

70.14218009478674 C = 30 Sigma 30

El mejor ha obtenido un porcentaje de aciertos del:

91.4691943127962

Chart, scatter chart

Description automatically generated

Figura 6

**2. Detección de spam**

Para este segundo apartado de la práctica se utilizarán las funciones implementadas en el apartado anterior para el cálculo de los modelos SVM probados para así implementar un algoritmo clasificador de correos electrónicos de spam y no spam.

Con la práctica se ha proporcionado varios ficheros. Un fichero que contiene correos de spam que se llama spam.zip, luego otros dos ficheros que contienen palabras de correos de no spam como es easy\_ham.zip que contiene palabras más fáciles de identificar como correo no spam, y el otro se llama hard\_ham.zip que contiene palabras que son más difíciles de distinguir de las palabras en correo spam. Estos ficheros han sido proporcionados por el profesor y han sido extraídos del SpamAssassin Public Corpus.

En primer lugar se deben de procesar los correos electrónicos para generar los datos de entrenamiento, test y validación. En los sistems de detección de creo spam se tiene que realizar un procesamiento previo que transforma el texto de los mensajes para poder facilitar el proceso de aprendizaje de nuestro algoritmo clasificador. Para esto obtenemos ayuda de una función de un fichero Python que se nos ha proporcionado junto con la práctica, la función se llama email2TokenList que se encuentra en el archivo *process\_email.py*. Esta función se encarga de: eliminar la cabecera del mensaje, pasar todo el texto a minúsculas, eliminar las etiquetas HTML, normalizar las URLs, sustituyéndolas todas por el texto “httpaddr”, normalizar las direcciones de correo, sustituyéndolas todas por el texto “emailaddr”, sustituir todos los números por el texto “number”, sustituir todas las apariciones del signo $ por el texto “dollar”, sustituir cada palabra por su raíz, utilizando para ello el algoritmo de Porter que se importa del toolkit NLTK para procesamiento de lenguaje natural, eliminar todos los signos de puntuación.

Como ejemplo, se nos ha proporcionado con la práctica el siguiente gráfico que muestra la transformación.

Graphical user interface, text, application, Word

Description automatically generated

A picture containing table

Description automatically generated

El siguiente paso que hemos realizado es una conversión del texto del mensaje en un vector de atributos. Para esto se ha seleccionado un conjunto de palabras que se usarán para describir los correos. Este conjunto de palabras se nos ha proporcionado con el archivo *vocab.txt* que contiene ordenado alfabéticamente las 1899 palabras que aparecen al menos 100 veces en el corpus de los correos. Para esto se utiliza una función de un fichero de Python, *getVocabDict* que se proporciona junto con la práctica. Esta función se encarga de leer el fichero y devolver el contenido de este como un diccionario, donde cada palabra es usada como clave del diccionario y cada una de estas se les asocia un número distinto.

Cada correo electrónico se clasifica con 0 si este no es spam y con 1 si este es spam.

Los resultados obtenidos son:

84.84848484848484 C = 0.01 Sigma 0.01

84.84848484848484 C = 0.01 Sigma 0.03

84.84848484848484 C = 0.01 Sigma 0.1

84.84848484848484 C = 0.01 Sigma 0.3

84.84848484848484 C = 0.01 Sigma 1

84.84848484848484 C = 0.01 Sigma 3

84.84848484848484 C = 0.01 Sigma 10

84.84848484848484 C = 0.01 Sigma 30

84.84848484848484 C = 0.03 Sigma 0.01

84.84848484848484 C = 0.03 Sigma 0.03

84.84848484848484 C = 0.03 Sigma 0.1

84.84848484848484 C = 0.03 Sigma 0.3

84.84848484848484 C = 0.03 Sigma 1

84.84848484848484 C = 0.03 Sigma 3

84.84848484848484 C = 0.03 Sigma 10

84.84848484848484 C = 0.03 Sigma 30

84.84848484848484 C = 0.1 Sigma 0.01

84.84848484848484 C = 0.1 Sigma 0.03

84.84848484848484 C = 0.1 Sigma 0.1

84.84848484848484 C = 0.1 Sigma 0.3

84.84848484848484 C = 0.1 Sigma 1

85.60606060606061 C = 0.1 Sigma 3

85.15151515151516 C = 0.1 Sigma 10

84.84848484848484 C = 0.1 Sigma 30

84.84848484848484 C = 0.3 Sigma 0.01

84.84848484848484 C = 0.3 Sigma 0.03

84.84848484848484 C = 0.3 Sigma 0.1

84.84848484848484 C = 0.3 Sigma 0.3

85.60606060606061 C = 0.3 Sigma 1

86.66666666666667 C = 0.3 Sigma 3

91.06060606060606 C = 0.3 Sigma 10

84.84848484848484 C = 0.3 Sigma 30

86.2121212121212 C = 1 Sigma 0.01

86.2121212121212 C = 1 Sigma 0.03

86.2121212121212 C = 1 Sigma 0.1

86.2121212121212 C = 1 Sigma 0.3

86.81818181818181 C = 1 Sigma 1

88.63636363636364 C = 1 Sigma 3

94.0909090909091 C = 1 Sigma 10

88.93939393939394 C = 1 Sigma 30

86.2121212121212 C = 3 Sigma 0.01

86.2121212121212 C = 3 Sigma 0.03

86.2121212121212 C = 3 Sigma 0.1

86.2121212121212 C = 3 Sigma 0.3

87.72727272727273 C = 3 Sigma 1

90.45454545454545 C = 3 Sigma 3

95.0 C = 3 Sigma 10

92.72727272727272 C = 3 Sigma 30

86.2121212121212 C = 10 Sigma 0.01

86.2121212121212 C = 10 Sigma 0.03

86.2121212121212 C = 10 Sigma 0.1

86.2121212121212 C = 10 Sigma 0.3

87.72727272727273 C = 10 Sigma 1

90.6060606060606 C = 10 Sigma 3

94.6969696969697 C = 10 Sigma 10

95.3030303030303 C = 10 Sigma 30

86.2121212121212 C = 30 Sigma 0.01

86.2121212121212 C = 30 Sigma 0.03

86.2121212121212 C = 30 Sigma 0.1

86.2121212121212 C = 30 Sigma 0.3

87.72727272727273 C = 30 Sigma 1

90.6060606060606 C = 30 Sigma 3

95.60606060606061 C = 30 Sigma 10

95.0 C = 30 Sigma 30

Mejor resultado obtenido con validacion: C = 30 Sigma = 10

95.60606060606061

Resultado de aciertos con test: 96.2178517397882

**Código**

#Imports

import numpy as np

from scipy.io import loadmat

import scipy.optimize as opt

import matplotlib.pyplot as plt

from scipy.optimize import minimize

from sklearn import svm

from get\_vocab\_dict import getVocabDict

import process\_email as mail

import codecs

import glob

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)

#yp = svm.predict(np.array([x1, x2]))

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(file\_name)

plt.show()

plt.close()

def main1():

datafile = 'p6/ex6data1.mat'

mat = loadmat(datafile)

X = mat.get("X")

y = mat.get("y")

print(X.shape[0])

#print(y)

y = y.ravel()

l = [1,100]

for i in l:

svm1 = svm.SVC(kernel='linear' , C=i)

svm1.fit(X, y)

visualize\_boundary(X, y, svm1, "eeeee")

main1()

def main2():

datafile = 'p6/ex6data2.mat'

mat = loadmat(datafile)

X = mat.get("X")

y = mat.get("y")

print(X.shape[0])

#print(y)

yr = y[:, -1]

l = [1]

sigma =0.1

svm1 = svm.SVC(kernel='rbf' , C=1, gamma = (1 / (2 \* sigma\*\*2)))

svm1.fit(X, yr)

visualize\_boundary(X, y, svm1, "eeeee")

main2()

def main3():

datafile = 'p6/ex6data3.mat'

mat = loadmat(datafile)

X = mat.get("X")

y = mat.get("y")

Xval = mat.get("Xval")

Yval = mat.get("yval")

print(X.shape[0])

#print(y)

y = y.ravel()

Yval = Yval.ravel()

C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

sigma = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

c = 0

sig = 0

accAux = 0

for i in C:

for j in sigma:

svm1 = svm.SVC(kernel='rbf' , C=i, gamma = (1 / (2 \* j\*\*2)))

svm1.fit(X, y)

#visualize\_boundary(X, y, svm1, "eeeee")

acc = (np.sum(svm1.predict(Xval) == Yval)/X.shape[0])\*100

if (acc > accAux):

accAux = acc

c = i

sig = j

print(acc, "C =", i, "Sigma", j)

svm1 = svm.SVC(kernel='rbf' , C=c, gamma = (1 / (2 \* sig\*\*2)))

svm1.fit(X, y)

visualize\_boundary(X, y, svm1, "eeeee")

print(accAux)

main3()

**Deteccion de spam**

def read\_file(filename, vocab):

email\_content = codecs.open(filename, 'r', encoding='utf-8', errors='ignore').read()

email = mail.email2TokenList(email\_content)

mail\_arr = np.zeros(len(vocab))

for word in email:

if word in vocab.keys():

mail\_arr[vocab[word] - 1] = 1

return mail\_arr

def read\_folder(foldername, vocab):

files\_in\_folder = glob.glob(foldername)

files\_in\_folder = sorted(files\_in\_folder)

num\_of\_files = len(files\_in\_folder)

vocab\_len = len(vocab)

emails = np.empty((num\_of\_files, vocab\_len))

i = 0

for filename in files\_in\_folder:

emails[i] = read\_file(filename, vocab)

i = i + 1

return emails

def cross\_validation(X\_spam, y\_spam, X\_easyHam, y\_easyHam, X\_hardHam, y\_hardHam):

percTrain = 0.6

percVal = 0.2

percTest = 0.2

m\_spam = X\_spam.shape[0]

m\_easyHam = X\_easyHam.shape[0]

m\_hardHam = X\_hardHam.shape[0]

X\_train = np.vstack((X\_spam[:(int)(percTrain \* m\_spam)],

X\_easyHam[:(int)(percTrain \* m\_easyHam)],

X\_hardHam[:(int)(percTrain \* m\_hardHam)]))

y\_train = np.hstack((y\_spam[:(int)(percTrain \* m\_spam)],

y\_easyHam[:(int)(percTrain \* m\_easyHam)],

y\_hardHam[:(int)(percTrain \* m\_hardHam)]))

X\_val = np.vstack((X\_spam[(int)(percTrain \* m\_spam):(int)((percTrain + percVal) \* m\_spam)],

X\_easyHam[(int)(percTrain \* m\_easyHam):(int)((percTrain + percVal) \* m\_easyHam)],

X\_hardHam[(int)(percTrain \* m\_hardHam):(int)((percTrain + percVal) \* m\_hardHam)]))

y\_val = np.hstack((y\_spam[(int)(percTrain \* m\_spam):(int)((percTrain + percVal) \* m\_spam)],

y\_easyHam[(int)(percTrain \* m\_easyHam):(int)((percTrain + percVal) \* m\_easyHam)],

y\_hardHam[(int)(percTrain \* m\_hardHam):(int)((percTrain + percVal) \* m\_hardHam)]))

X\_test = np.vstack((X\_spam[(int)((percTrain + percVal) \* m\_spam):],

X\_easyHam[(int)((percTrain + percVal) \* m\_easyHam):],

X\_hardHam[(int)((percTrain + percVal) \* m\_hardHam):]))

y\_test = np.hstack((y\_spam[(int)((percTrain + percVal) \* m\_spam):],

y\_easyHam[(int)((percTrain + percVal) \* m\_easyHam):],

y\_hardHam[(int)((percTrain + percVal) \* m\_hardHam):]))

return X\_train, y\_train, X\_val, y\_val, X\_test, y\_test

def main():

"""

Function to read in the supplied vocab list text file into a dictionary.

Dictionary key is the stemmed word, value is the index in the text file

If "reverse", the keys and values are switched.

"""

vocab = getVocabDict()

spam\_emails = read\_folder("p6/spam/\*", vocab)

X\_spam = spam\_emails

y\_spam = np.ones(X\_spam.shape[0])

easyHam\_emails = read\_folder("p6/easy\_ham/\*", vocab)

X\_easyHam = easyHam\_emails

y\_easyHam = np.zeros(X\_easyHam.shape[0])

hardHam\_emails = read\_folder("p6/hard\_ham/\*", vocab)

X\_hardHam = hardHam\_emails

y\_hardHam = np.zeros(X\_hardHam.shape[0])

print(X\_spam.shape[0])

print(X\_easyHam.shape[0])

print(X\_hardHam.shape[0])

X\_train, y\_train, X\_val, y\_val, X\_test, y\_test = cross\_validation(X\_spam, y\_spam, X\_easyHam, y\_easyHam, X\_hardHam, y\_hardHam)

C = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

sigma = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

c = 0

sig = 0

accAux = 0

for i in C:

for j in sigma:

svm1 = svm.SVC(kernel='rbf' , C=i, gamma = (1 / (2 \* j\*\*2)))

svm1.fit(X\_train, y\_train)

#visualize\_boundary(X\_val, y\_val, svm1, "eeeee")

acc = (np.sum(svm1.predict(X\_val) == y\_val)/X\_val.shape[0])\*100

if (acc > accAux):

accAux = acc

c = i

sig = j

print(acc, "C =", i, "Sigma", j)

svm1 = svm.SVC(kernel='rbf' , C=c, gamma = (1 / (2 \* sig\*\*2)))

svm1.fit(X\_train, y\_train)

print("\nMejor resultado obtenido con validacion: C =", c, "Sigma =", sig)

print(accAux)

acc = (np.sum(svm1.predict(X\_test) == y\_test)/X\_test.shape[0])\*100

print("\nResultado de aciertos con test:", acc)

main()