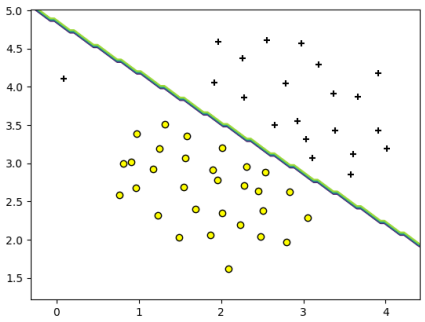
Práctica 6: Support Vector Machines

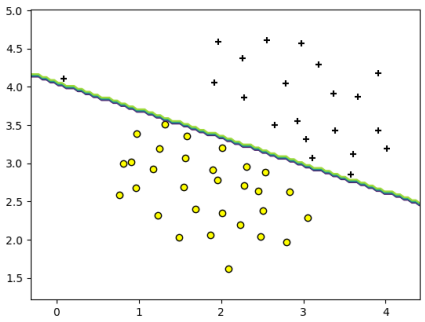
Ana Martín Sánchez, Nicolás Pastore Burgos

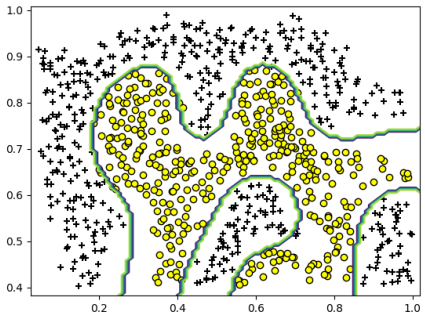
21/09/2021

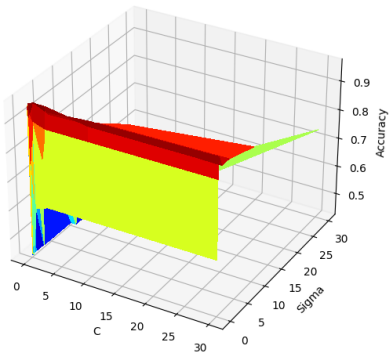
1 Solución propuesta

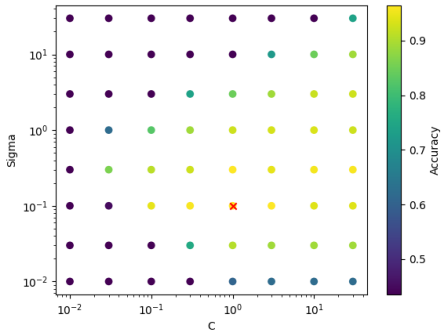
1.1 Resultados obtenidos

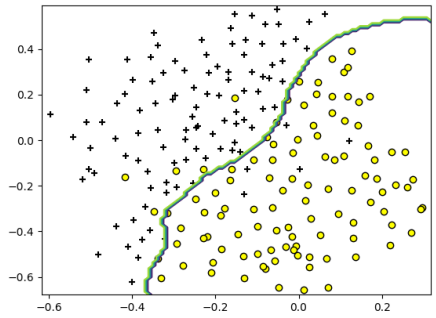


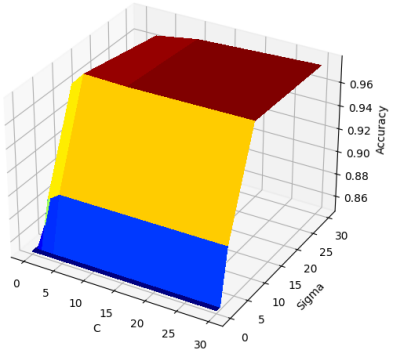


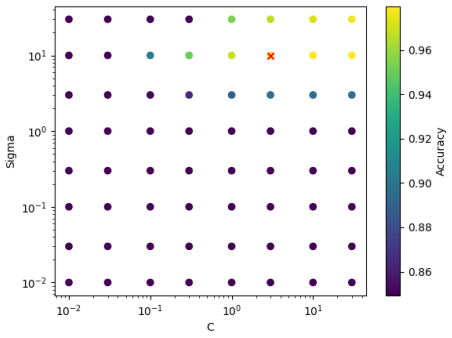












1.2 Implementación

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm as svm
import codecs
import os, os.path

from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from scipy.io import loadmat
from sklearn.metrics import accuracy_score

from Data.Parte2.process_email import email2TokenList
from Data.Parte2.get_vocab_dict import getVocabDict

def scatter_mat(x,y):
    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')

def visualize_boundary(x, y, svm, zoom):
    x1 = np.linspace(x[:, 0].min() - zoom, x[:, 0].max() + zoom, 100)
    x2 = np.linspace(x[:, 1].min() - zoom, x[:, 1].max() + zoom, 100)

    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape
    )

    scatter_mat(x,y)

    plt.contour(x1, x2, yp)
    plt.show()

def parte1(x,y):
    s1 = svm.SVC(kernel='linear', C=1.0)
    s1.fit(x,y)

    s2 = svm.SVC(kernel='linear', C=100.0)
    s2.fit(x,y)

    zoom = 0.4
    visualize_boundary(x,y,s1,zoom)
    visualize_boundary(x,y,s2,zoom)

def parte1_2(x, y, C, sigma):
    s = svm.SVC(kernel='rbf', C=C, gamma= 1 / (2 * sigma ** 2))
    s.fit(x,y)

    visualize_boundary(x,y,s, 0.02)
    plt.show()

def searchBestCandSigma(x, xVal, y, yVal):
    cs = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])
    sigmas = np.array([0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30])

    numCs = cs.shape[0]
    numSigmas = cs.shape[0]

    res = np.zeros(numCs * numSigmas).reshape(numCs, numSigmas)

    bestAcc = 0
    bestC = -1
    bestSigma = -1
    bestSVM = {}

    for i in np.arange(numCs):
        for j in np.arange(numSigmas):
            s = svm.SVC(kernel='rbf', C=cs[i], gamma= 1 / (2 * sigmas[j] **
     2))
            s.fit(x,y)

```

```
            res[i,j] = accuracy_score(yVal, s.predict(xVal))
            if bestAcc < res[i,j]:
                bestAcc = res[i,j]
                bestC = cs[i]
                bestSigma = sigmas[j]
                bestSVM = s
            print('Tested sigma ' + str(sigmas[j]))

        print('Tested c ' + str(cs[i]))

    print("Best accuracy: " + str(bestAcc))
    print("C: " + str(bestC))
    print("Sigma: " + str(bestSigma))

    fig = plt.figure()

    cc, ss = np.meshgrid(sigmas, cs)

    ax = Axes3D(fig, auto_add_to_figure=False)
    ax.set_xlabel('C')
    ax.set_ylabel('Sigma')
    ax.set_zlabel('Accuracy')

    fig.add_axes(ax)

    surf = ax.plot_surface(ss,cc,res, cmap=cm.jet, linewidth=0, antialiased
    =False)
    plt.show()

    plt.scatter(ss, cc, c=res)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('C')
    plt.ylabel('Sigma')
    plt.clim(np.min(res), np.max(res))
    plt.colorbar().set_label('Accuracy')
    plt.scatter(bestC, bestSigma, marker='x', color='r')
    plt.show()

    return bestSVM, bestAcc, bestC, bestSigma

def parte1_3(x, xVal, y, yVal):
    s, acc, c, sigma = searchBestCandSigma(x, xVal, y, yVal)

    visualize_boundary(x,y,s, 0.02)
    plt.show()

"""
    ==========================================================
    ==========================================================
    ==========================================================
    ==========================================================
"""
```

```
1123
1124 def filterEmailWithDictionary(dict, email):
1125     emailDict = np.zeros([len(dict)])
1126     for word in email:
1127         if word in dict:
1128             emailDict[dict[word]-1] = 1
1129
1130     return emailDict
1131
1132 def readEmails(dict, route, folder, format, size):
1133     emails = np.empty((size,len(dict)))
1134     for i in np.arange(size):
1135         email_contents = codecs.open(route + folder + '{0:04d}'.format(i+1)
      + format, 'r', encoding='utf 8', errors='ignore').read()
1136         emails[i] = filterEmailWithDictionary(dict, email2TokenList(
     email_contents))
1137     print('Done reading and preparing ' + folder)
1138     return emails
1139
1140 def createXandY(spam, easyHam, hardHam, ySpam, yEasy, yHard):
1141     return np.append(np.append(spam, easyHam, axis=0), hardHam, axis=0), np
     .append(np.append(ySpam, yEasy, axis=0), yHard, axis=0)
1142
1143 def parte2():
1144     dict = getVocabDict()
1145
1146     route = 'Data/Parte2/'
1147     folders = ['spam/', 'easy_ham/', 'hard_ham/']
1148     format = '.txt'
1149
1150     print('Comencing to read data')
1151
1152     spam = readEmails(dict, route, folders[0], format, len([name for name
     in os.listdir('./' + route + folders[0])]))
1153     spamSize = len(spam)
1154     ySpam = np.ones(spamSize)
1155
1156     easy = readEmails(dict, route, folders[1], format, len([name for name
     in os.listdir('./' + route + folders[1])]))
1157     easySize = len(easy)
1158     yEasy = np.zeros(easySize)
1159
1160     hard = readEmails(dict, route, folders[2], format, len([name for name
     in os.listdir('./' + route + folders[2])]))
1161     hardSize = len(hard)
1162     yHard = np.zeros(hardSize)
1163
1164     trainPerc = 0.6
1165     valPerc = 0.3 + trainPerc
1166     # rest for test
1167
1168     trainSpam = int(trainPerc * spamSize)
1169     trainEasy = int(trainPerc * easySize)
```

```
    trainHard = int(trainPerc * hardSize)

    valSpam = int(valPerc * spamSize)
    valEasy = int(valPerc * easySize)
    valHard = int(valPerc * hardSize)

    x, y = createXandY(spam[:trainSpam],
                       easy[:trainEasy],
                       hard[:trainHard],
                      ySpam[:trainSpam],
                      yEasy[:trainEasy],
                      yHard[:trainHard])

    xVal, yVal = createXandY(spam[trainSpam:valSpam],
                             easy[trainEasy:valEasy],
                             hard[trainHard:valHard],
                            ySpam[trainSpam:valSpam],
                            yEasy[trainEasy:valEasy],
                            yHard[trainHard:valHard])

    xTest, yTest = createXandY(spam[valSpam:],
                           easy[valEasy:],
                           hard[valHard:],
                          ySpam[valSpam:],
                          yEasy[valEasy:],
                          yHard[valHard:])

    s, acc, c, sigma = searchBestCandSigma(x, xVal, y, yVal)

    print('Accuracy over Test sample: ' + str(accuracy_score(yTest, s.
    predict(xTest)) * 100) + '%')

def main():
    data1 = loadmat("Data/Parte1/ex6data1.mat")

    x1 = data1['X']
    y1 = data1['y']
    y1R = np.ravel(y1)

    data2 = loadmat("Data/Parte1/ex6data2.mat")

    x2 = data2['X']
    y2 = data2['y']
    y2R = np.ravel(y2)

    data3 = loadmat("Data/Parte1/ex6data3.mat")

    x3 = data3['X']
    y3 = data3['y']
    y3R = np.ravel(y3)

    x3Val = data3['Xval']
    y3Val = data3['yval']
```

```
    y3ValR = np.ravel(y3Val)

    parte1(x1, y1R)
    parte1_2(x2, y2R, 1.0, 0.1)
    parte1_3(x3, x3Val, y3R, y3ValR)
    parte2()

if __name__ == "__main__":
    main()
```

main.py