Support Vector Machines

Código

SVM

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

data1 = loadmat('p6/p6/ex6data1.mat')
y1 = data1['y']
X1 = data1['X']

data2 = loadmat('p6/p6/ex6data2.mat')
y2 = data2['y']
X2 = data2['X']

data3 = loadmat('p6/p6/ex6data3.mat')
y3 = data3['y']
X3 = data3['X']
yval3 = data3['yval']
Xval3 = data3['Xval']

for i in range(1, 120, 20):
    svm = SVC(kernel='linear', C=i)
    svm.fit(X1, y1)

    positive = plt.scatter(X1[np.where(y1 == 1), 0], X1[np.where(y1 == 1), 1],
marker='+', c='k')
    negative = plt.scatter(X1[np.where(y1 == 0), 0], X1[np.where(y1 == 0), 1],
marker='o', c='b')

    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = svm.decision_function(xy).reshape(XX.shape)

    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
    plt.title('SVM with C = ' + str(i))
    plt.show()

svm = SVC(kernel='rbf', C=1, gamma=1 / (2 * 0.1 ** 2))
svm.fit(X2, y2)

positive = plt.scatter(X2[np.where(y2 == 1), 0], X2[np.where(y2 == 1), 1],
marker='+', c='k')
negative = plt.scatter(X2[np.where(y2 == 0), 0], X2[np.where(y2 == 0), 1],
marker='o', c='b')
```

```
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = svm.decision_function(xy).reshape(XX.shape)

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
       linestyles=['--', '-', '--'])
plt.title('SVM with gaussian kernel')
plt.show()

values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
for C in values:
    for sigma in values:
        svm = SVC(kernel='rbf', C=C, gamma=float(1)/(2 * sigma ** 2))
        svm.fit(X3, y3)
        predictions = svm.predict(Xval3)
        number = 0
        for i in range(len(yval3)):
            if predictions[i] == yval3[i]:
                number = number + 1
        accuracy = (float(number)/len(yval3))*100
        print("Accuracy for C = " + str(C) + " and sigma = " + str(sigma) + " = " +
str(accuracy) + "%")

svm = SVC(kernel='rbf', C=1, gamma=float(1)/(2 * 0.1 ** 2))
svm.fit(X3, y3)

positive = plt.scatter(X3[np.where(y3 == 1), 0], X3[np.where(y3 == 1), 1],
marker='+', c='k')
negative = plt.scatter(X3[np.where(y3 == 0), 0], X3[np.where(y3 == 0), 1],
marker='o', c='b')

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = svm.decision_function(xy).reshape(XX.shape)

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
       linestyles=['--', '-', '--'])
plt.title('SVM with best parameters')
plt.show()
```

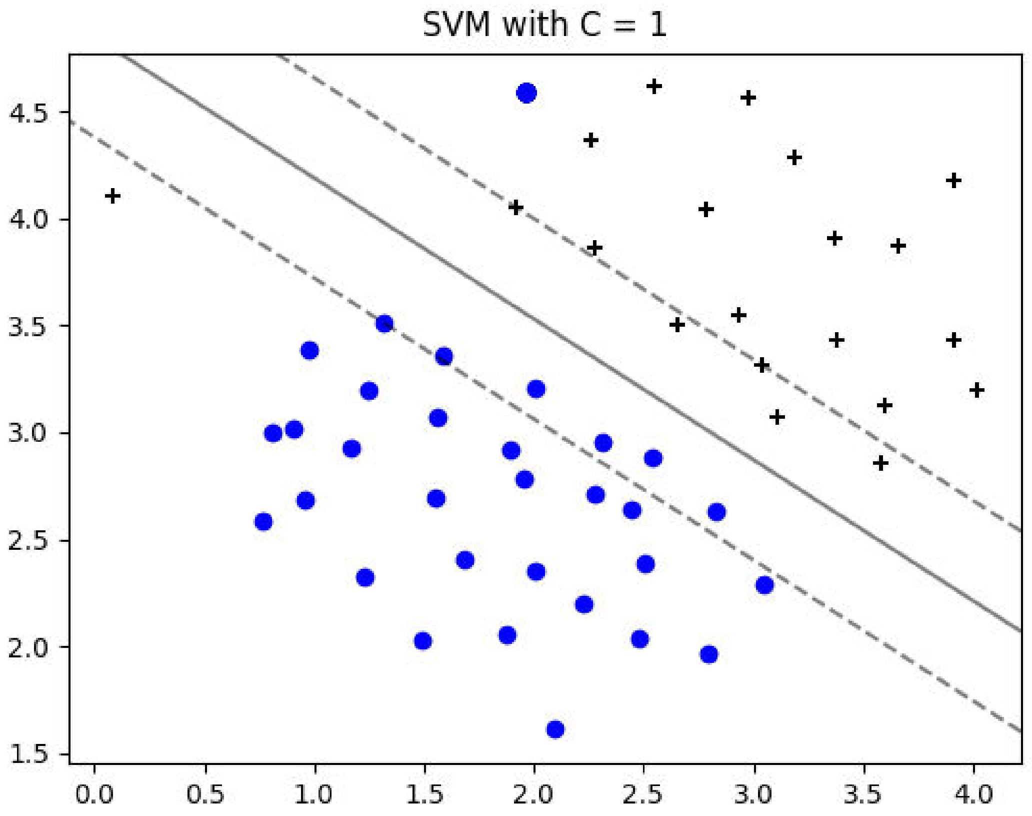
Spam detection

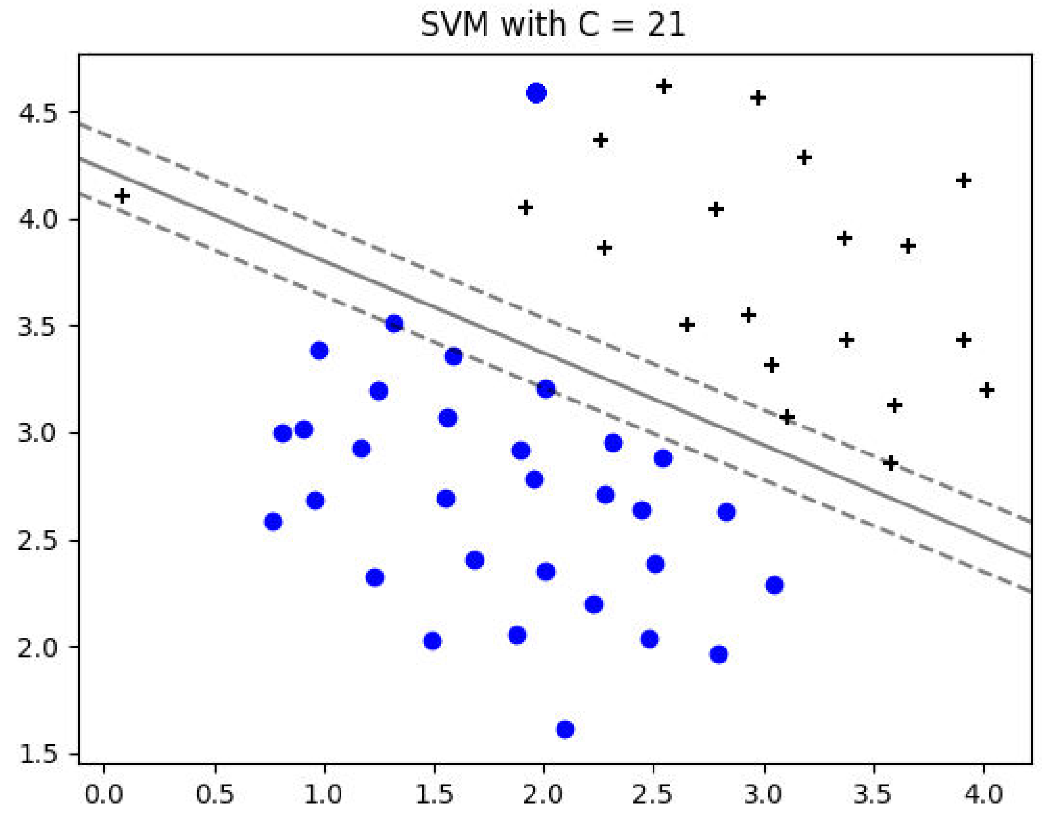
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import re
import nltk
import nltk.stem.porter
import os
import io
from sklearn.model_selection import train_test_split

def preProcess(email):

    hdrstart = email.find("\n\n")
    if hdrstart != -1:
        email = email[hdrstart:]

    email = email.lower()
    # Strip html tags. replace with a space
    email = re.sub('<[^<>]+>', ' ', email)
    # Any numbers get replaced with the string 'number'
    email = re.sub('[0-9]+', 'number', email)
    # Anything starting with http or https:// replaced with 'httpaddr'
    email = re.sub('(http|https)://[^\s]*', 'httpaddr', email)
    # Strings with "@" in the middle are considered emails --> 'emailaddr'
    email = re.sub('[^\s]+@[^\s]+', 'emailaddr', email)
    # The '$' sign gets replaced with 'dollar'
    email = re.sub('[$]+', 'dollar', email)
    return email

def email2TokenList(raw_email):
    """
    Function that takes in a raw email, preprocesses it, tokenizes it,
    stems each word, and returns a list of tokens in the e-mail
    """

    stemmer = nltk.stem.porter.PorterStemmer()
    email = preProcess(raw_email)

    # Split the e-mail into individual words (tokens)
    tokens = re.split('[ \@\$\/\#\.\-\:\&\*\+\=\[\]\?\!\(\)\{\}\,\'\"\>\_\<\;\%]',
                      email)

    # Loop over each token and use a stemmer to shorten it
    tokenlist = []
    for token in tokens:

        token = re.sub('[^a-zA-Z0-9]', '', token)
        stemmed = stemmer.stem(token)
        #Throw out empty tokens
        if not len(token):
            continue
        # Store a list of all unique stemmed words
        tokenlist.append(stemmed)

    return tokenlist

def getVocabDict(reverse=False):
    """
    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_dict = {}
    with open("p6/p6/vocab.txt") as f:
        for line in f:
            (val, key) = line.split()
            if not reverse:
                vocab_dict[key] = int(val)
            else:
                vocab_dict[int(val)] = key

    return vocab_dict

spamFiles = []
for i in os.listdir('p6/p6/spam'):
    spamFile = io.open('p6/p6/spam/' + str(i), mode='r', encoding='utf-8',
errors='ignore').read()
    spamFiles.append(email2TokenList(spamFile))

easyHamFiles = []
for i in os.listdir('p6/p6/easy_ham'):
    easyHamFile = io.open('p6/p6/easy_ham/' + str(i), mode='r', encoding='utf-8',
errors='ignore').read()
    easyHamFiles.append(email2TokenList(easyHamFile))

hardHamFiles = []
for i in os.listdir('p6/p6/hard_ham'):
    hardHamFile = io.open('p6/p6/hard_ham/' + str(i), mode='r', encoding='utf-8',
errors='ignore').read()
    hardHamFiles.append(email2TokenList(hardHamFile))

vocabDict = getVocabDict()

spamFilesEncoded = []
for email in spamFiles:
    encoded = np.zeros((1, 1900))
    for word in email:
        if word in vocabDict:
            encoded[0, vocabDict[word]] = 1
    #Label (spam)
    encoded[0, 0] = 1
    spamFilesEncoded.append(encoded)

easyHamFilesEncoded = []
for email in easyHamFiles:
    encoded = np.zeros((1, 1900))
    for word in email:
        if word in vocabDict:
            encoded[0, vocabDict[word]] = 1
    #Label (non-spam)
    encoded[0, 0] = 0
    easyHamFilesEncoded.append(encoded)

hardHamFilesEncoded = []
for email in hardHamFiles:
    encoded = np.zeros((1, 1900))
    for word in email:
        if word in vocabDict:
            encoded[0, vocabDict[word]] = 1
    #Label (non-spam)
    encoded[0, 0] = 0
    hardHamFilesEncoded.append(encoded)
```

```
spamFilesEncoded = np.vstack(spamFilesEncoded)
easyHamFilesEncoded = np.vstack(easyHamFilesEncoded)
hardHamFilesEncoded = np.vstack(hardHamFilesEncoded)

data = np.vstack((spamFilesEncoded, easyHamFilesEncoded, hardHamFilesEncoded))
np.random.shuffle(data)

X = data[:, 1:]
y = data[:, 0]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=1)

values = [0.01, 0.1, 1, 10]
for C in values:
    for sigma in values:
        svm = SVC(kernel='rbf', C=C, gamma=float(1) / (2 * sigma ** 2))
        svm.fit(X_train, y_train)
        predict = svm.predict(X_val)
        number = 0
        for i in range(len(y_val)):
            if predict[i] == y_val[i]:
                number = number + 1
        accuracy = (float(number)/len(y_val))*100
        print("Accuracy for C = " + str(C) + " and sigma = " + str(sigma) + " = " +
str(accuracy) + "%")
```

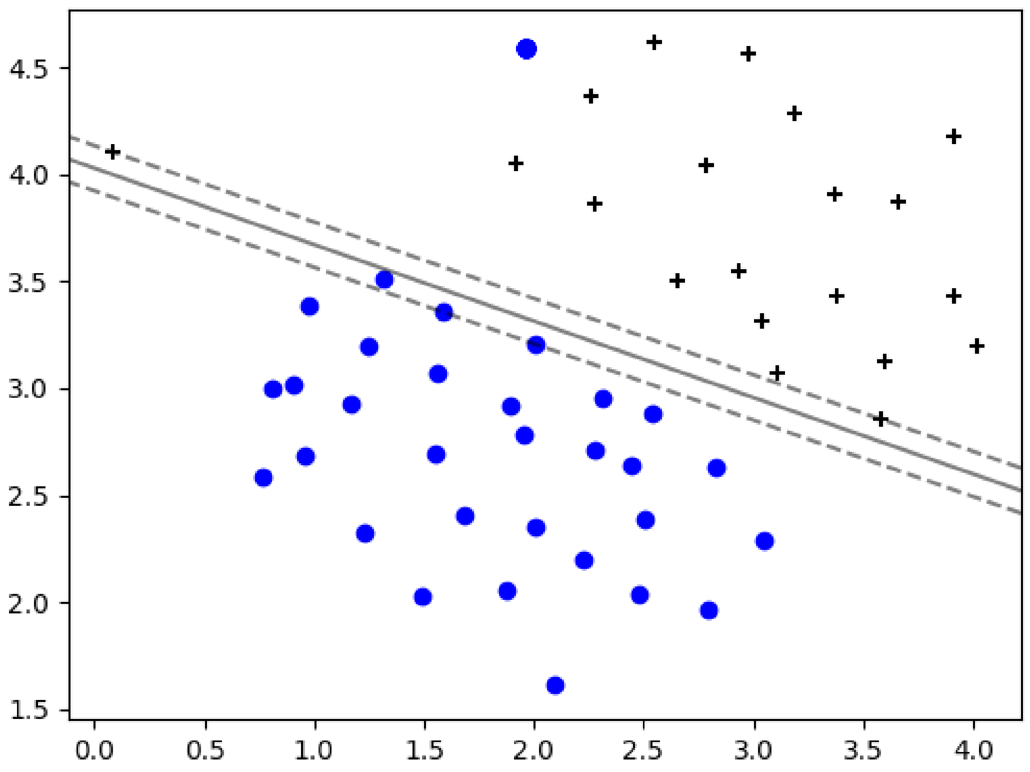
Resultados

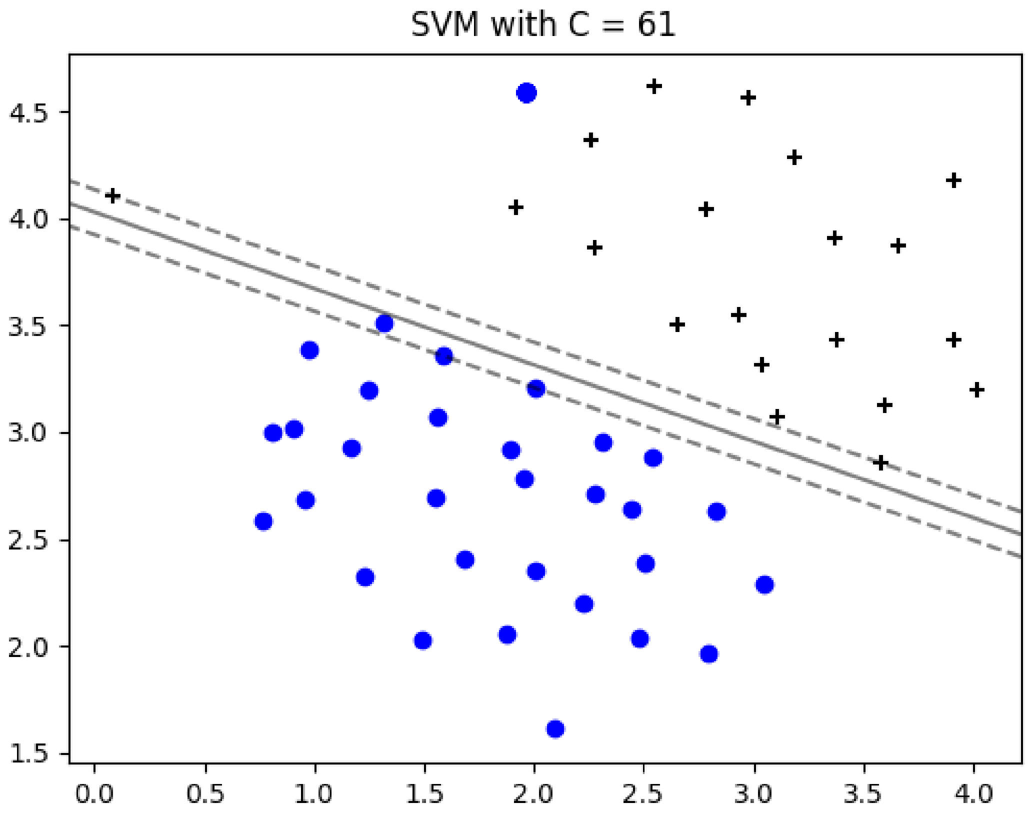
SVM

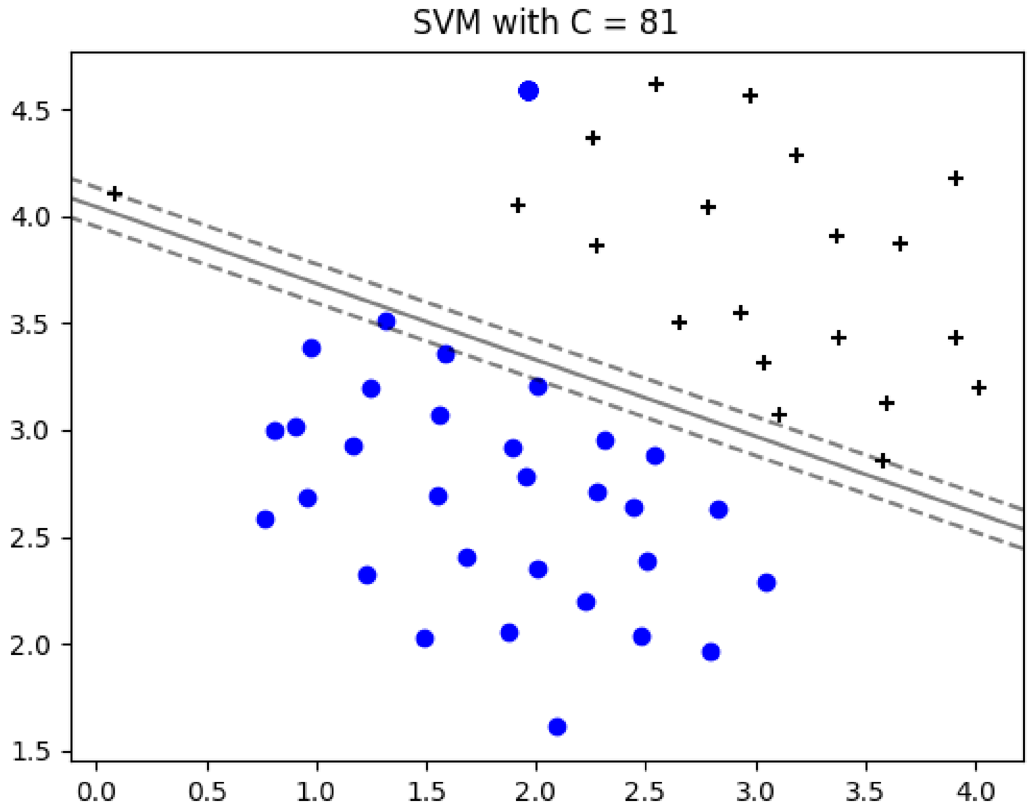

SVM with C = 1
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

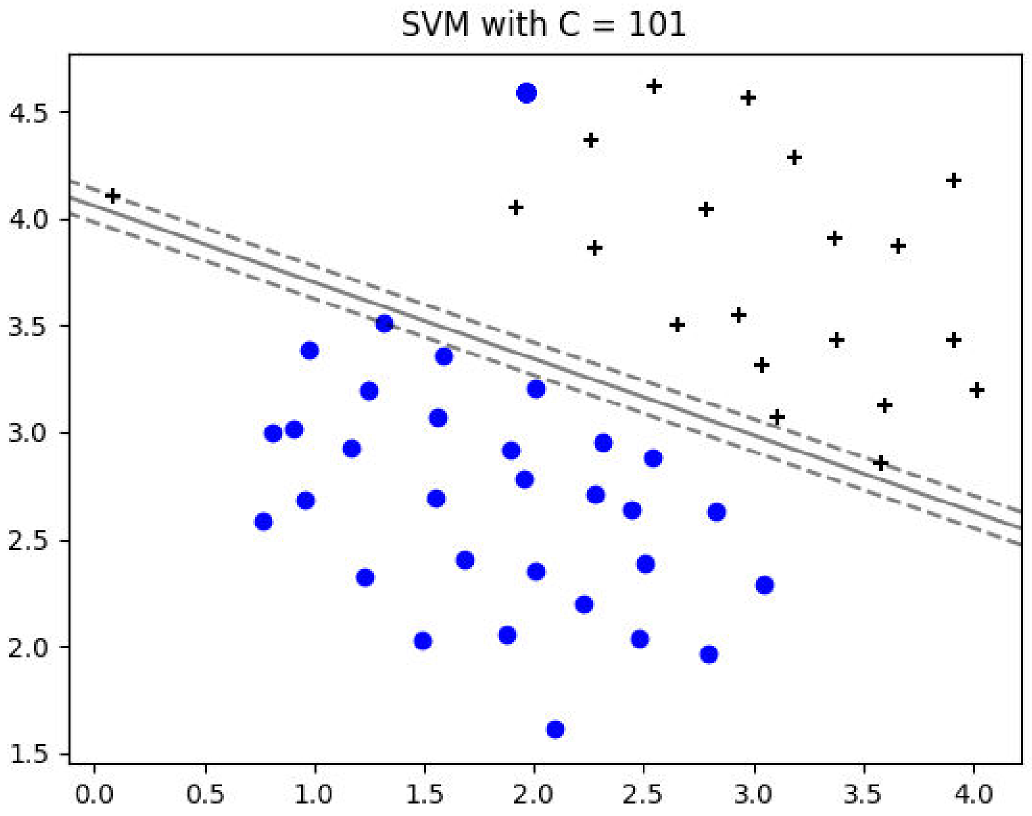

SVM with C = 21
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

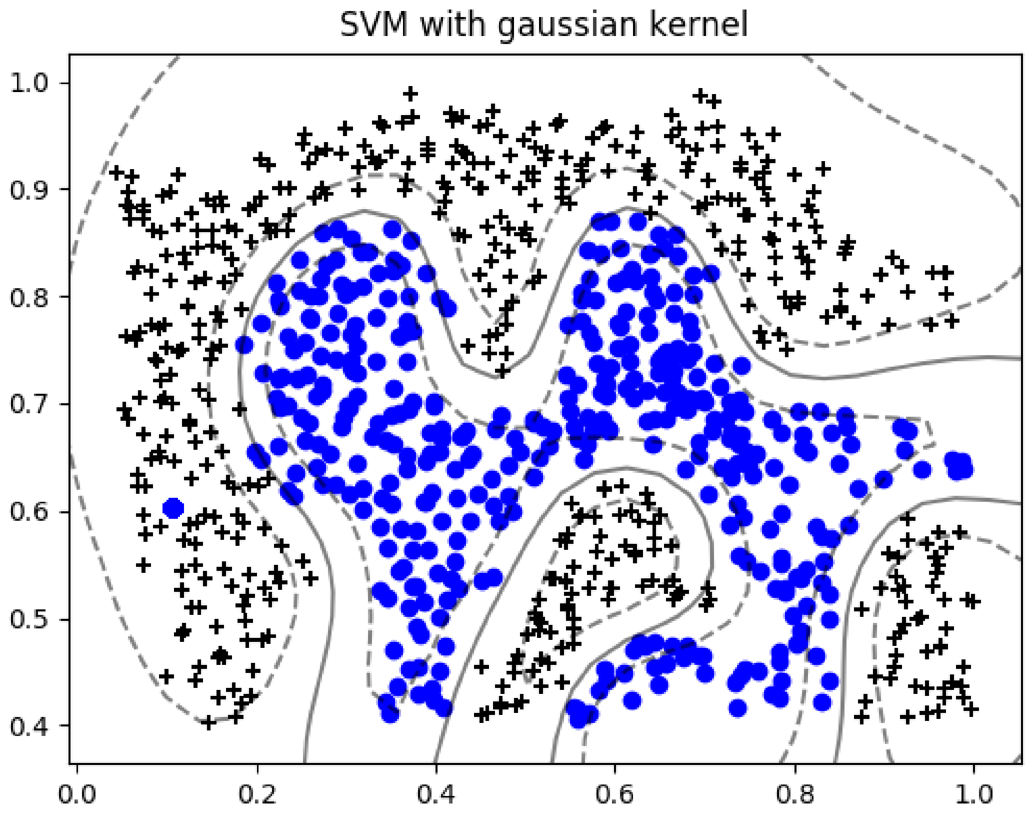
SVM with C = 41



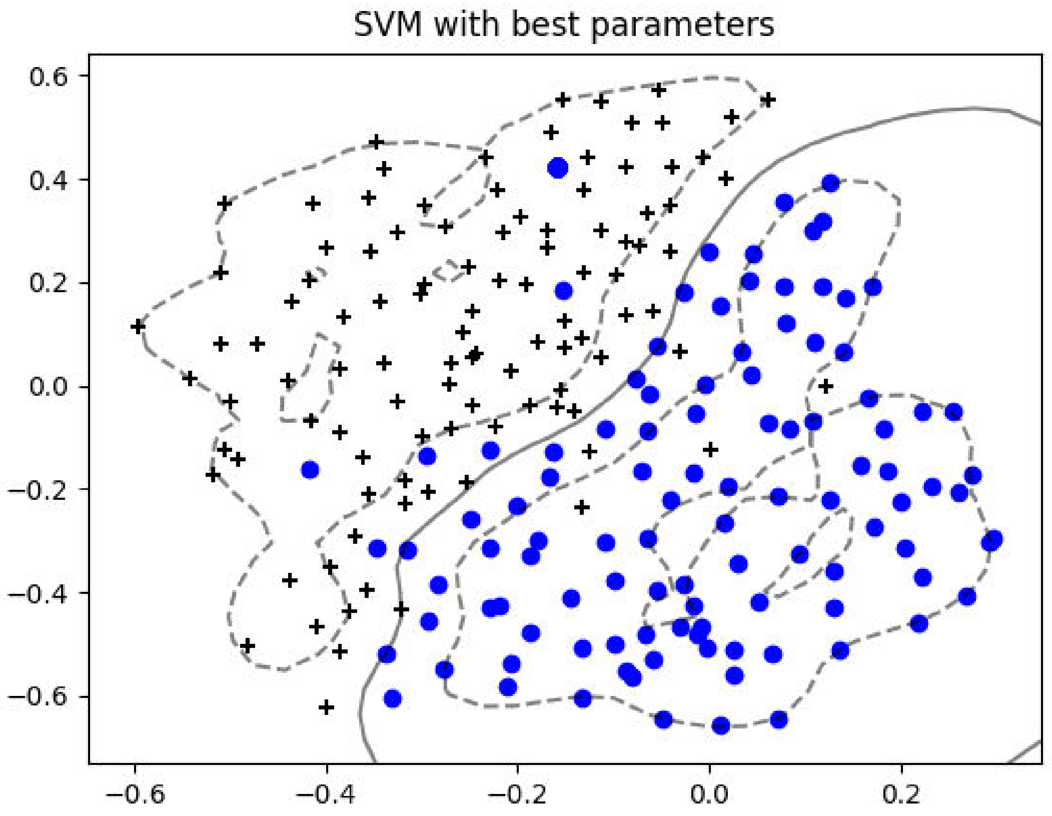

SVM with C = 61
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


SVM with C = 81
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


SVM with C = 101
4.5
4.0
3.5
3.0
2.5
2.0
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



```
Accuracy for C = 0.01 and sigma = 0.01 = 43.5%
Accuracy for C = 0.01 and sigma = 0.03 = 43.5%
 Accuracy for C = 0.01 and sigma = 0.1 = 43.5%
 Accuracy for C = 0.01 and sigma = 0.3 = 43.5%
   Accuracy for C = 0.01 and sigma = 1 = 43.5%
   Accuracy for C = 0.01 and sigma = 3 = 43.5%
  Accuracy for C = 0.01 and sigma = 10 = 43.5%
  Accuracy for C = 0.01 and sigma = 30 = 43.5%
Accuracy for C = 0.03 and sigma = 0.01 = 43.5%
Accuracy for C = 0.03 and sigma = 0.03 = 43.5%
 Accuracy for C = 0.03 and sigma = 0.1 = 45.0%
 Accuracy for C = 0.03 and sigma = 0.3 = 86.0%
   Accuracy for C = 0.03 and sigma = 1 = 62.0%
   Accuracy for C = 0.03 and sigma = 3 = 43.5%
  Accuracy for C = 0.03 and sigma = 10 = 43.5%
  Accuracy for C = 0.03 and sigma = 30 = 43.5%
 Accuracy for C = 0.1 and sigma = 0.01 = 43.5%
 Accuracy for C = 0.1 and sigma = 0.03 = 43.5%
  Accuracy for C = 0.1 and sigma = 0.1 = 94.5%
  Accuracy for C = 0.1 and sigma = 0.3 = 91.0%
    Accuracy for C = 0.1 and sigma = 1 = 82.5%
    Accuracy for C = 0.1 and sigma = 3 = 43.5%
   Accuracy for C = 0.1 and sigma = 10 = 43.5%
   Accuracy for C = 0.1 and sigma = 30 = 43.5%
 Accuracy for C = 0.3 and sigma = 0.01 = 43.5%
 Accuracy for C = 0.3 and sigma = 0.03 = 75.5%
  Accuracy for C = 0.3 and sigma = 0.1 = 96.0%
  Accuracy for C = 0.3 and sigma = 0.3 = 92.5%
Accuracy for C = 0.3 and sigma = 1 = 89.0%
Accuracy for C = 0.3 and sigma = 3 = 74.0%
Accuracy for C = 0.3 and sigma = 10 = 43.5%
Accuracy for C = 0.3 and sigma = 30 = 43.5%
Accuracy for C = 1 and sigma = 0.01 = 60.5%
Accuracy for C = 1 and sigma = 0.03 = 90.5%
Accuracy for C = 1 and sigma = 0.1 = 96.5%
Accuracy for C = 1 and sigma = 0.3 = 96.5%
Accuracy for C = 1 and sigma = 1 = 92.5%
Accuracy for C = 1 and sigma = 3 = 84.5%
Accuracy for C = 1 and sigma = 10 = 43.5%
Accuracy for C = 1 and sigma = 30 = 43.5%
Accuracy for C = 3 and sigma = 0.01 = 62.0%
Accuracy for C = 3 and sigma = 0.03 = 89.0%
Accuracy for C = 3 and sigma = 0.1 = 96.5%
Accuracy for C = 3 and sigma = 0.3 = 94.5%
Accuracy for C = 3 and sigma = 1 = 93.0%
Accuracy for C = 3 and sigma = 3 = 89.0%
Accuracy for C = 3 and sigma = 10 = 72.0%
Accuracy for C = 3 and sigma = 30 = 43.5%
Accuracy for C = 10 and sigma = 0.01 = 62.0%
Accuracy for C = 10 and sigma = 0.03 = 89.0%
Accuracy for C = 10 and sigma = 0.1 = 94.0%
Accuracy for C = 10 and sigma = 0.3 = 95.5%
Accuracy for C = 10 and sigma = 1 = 93.5%
Accuracy for C = 10 and sigma = 3 = 92.0%
Accuracy for C = 10 and sigma = 10 = 84.5%
Accuracy for C = 10 and sigma = 30 = 43.5%
Accuracy for C = 30 and sigma = 0.01 = 62.0%
Accuracy for C = 30 and sigma = 0.03 = 89.0%
Accuracy for C = 30 and sigma = 0.1 = 94.0%
Accuracy for C = 30 and sigma = 0.3 = 96.0%
Accuracy for C = 30 and sigma = 1 = 92.5%
Accuracy for C = 30 and sigma = 3 = 92.5%
Accuracy for C = 30 and sigma = 10 = 89.0%
Accuracy for C = 30 and sigma = 30 = 74.0%
```



Spam detection

```
Accuracy for C = 0.01 and sigma = 0.01 = 85.4765506808%
Accuracy for C = 0.01 and sigma = 0.1 = 85.4765506808%
Accuracy for C = 0.01 and sigma = 1 = 85.4765506808%
Accuracy for C = 0.01 and sigma = 10 = 85.4765506808%
Accuracy for C = 0.1 and sigma = 0.01 = 85.4765506808%
Accuracy for C = 0.1 and sigma = 0.1 = 85.4765506808%
Accuracy for C = 0.1 and sigma = 1 = 85.4765506808%
Accuracy for C = 0.1 and sigma = 10 = 87.1406959153%
Accuracy for C = 1 and sigma = 0.01 = 90.9228441755%
Accuracy for C = 1 and sigma = 0.1 = 90.9228441755%
Accuracy for C = 1 and sigma = 1 = 90.9228441755%
Accuracy for C = 1 and sigma = 10 = 97.1255673222%
Accuracy for C = 10 and sigma = 0.01 = 90.9228441755%
Accuracy for C = 10 and sigma = 0.1 = 90.9228441755%
Accuracy for C = 10 and sigma = 1 = 92.1331316188%
Accuracy for C = 10 and sigma = 10 = 97.8819969743%
```

Comentarios

Cuando el valor de C se aumenta, el borde de SVM se disminuye para SVM con kernel lineal. Los mejores parámetros para la primera parte de la práctica son C = 1 y sigma = 0.1. Los mejores valores para C y sigma para spam detection son C = 10 y sigma = 10 con el valor de accuracy igual a casi 98%.