Aprendizaje automático y minería de datos

Práctica 6 – Support Vector Machines

Ramón Arjona Quiñones
Celia Castaños Bornaechea

Contenido

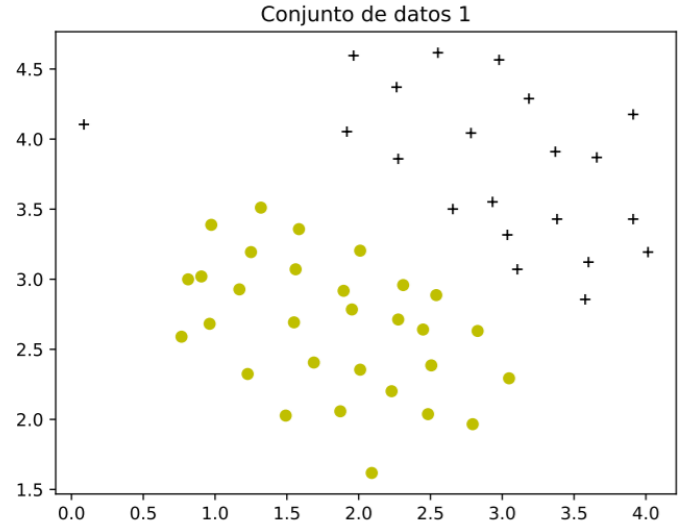
1. Support Vector Machines

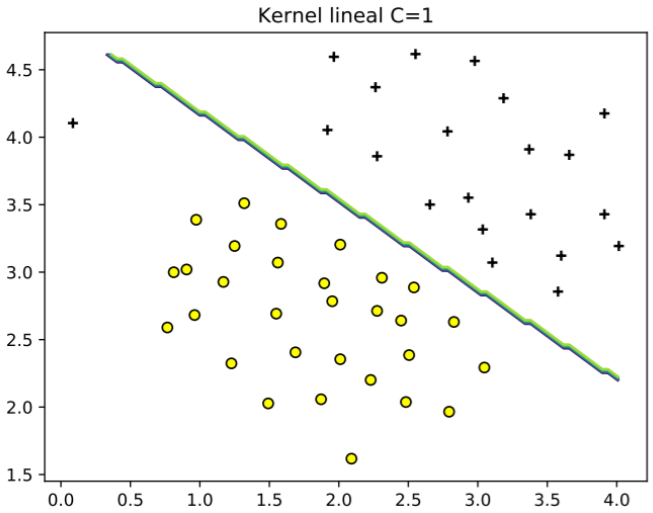
Resumen

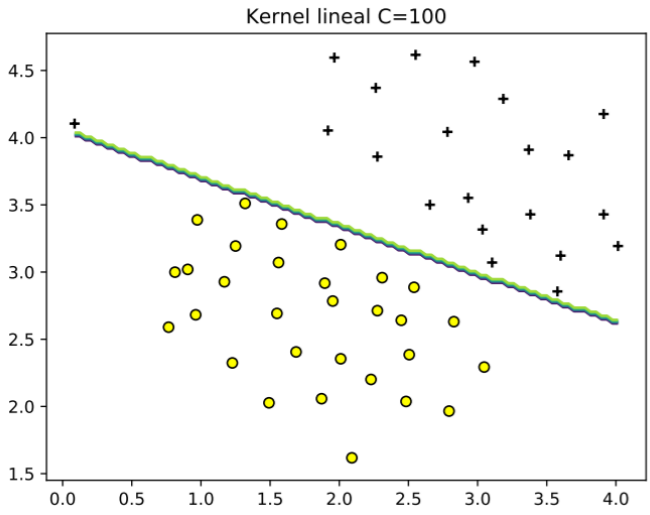
Familiarización con el clasificador SVM de *sckit-learn*.

1.1 Kernel lineal

Se instancia un clasificador SVM con un parámetro C de regularización. Los conjuntos son separables mediante una recta.








```
def Ejericio1(reg):
    '''
    Con kernel lineal
    '''
    # 1. Cargamos los datos
    data = loadmat('ex6data1.mat')
    X = data['X']
    y = data['y'].ravel()

    # Hacemos el SVM con kernel lineal (el conjunto es linalmente separab
le)
```

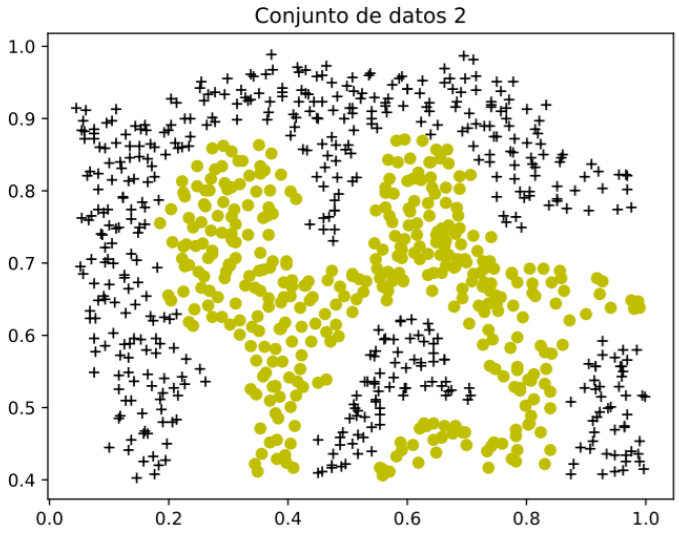
```
    svm = SVC(kernel='linear', C=reg)
    svm.fit(X, y)

    # Pintamos la frontera
    pintaFrontera(X, y, svm, "Kernel lineal C=" + str(reg))

    # Guardamos la gráfica y cerramos
    #plt.show()
    plt.savefig("SVMLineal_C=" + str(reg) + ".pdf")
    plt.close()
```

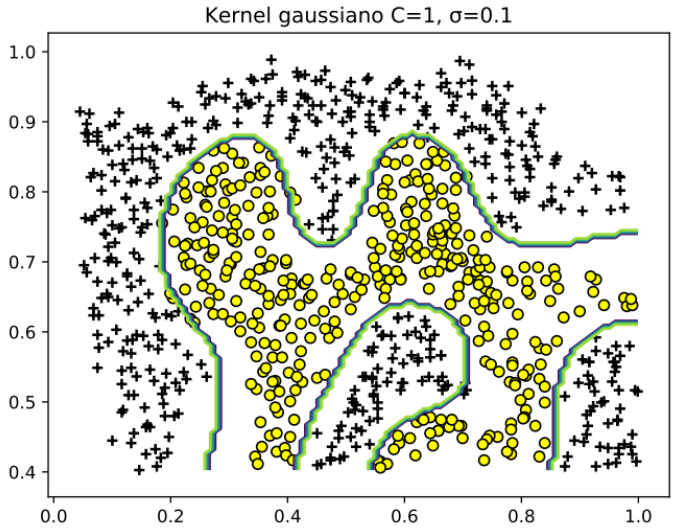
1.2 Kernel gaussiano

Este kernel entrena una SVM que clasifique correctamente un conjunto de datos que no es linealmente separable. Para ello se basa en la siguiente ecuación:

$$K_{gauss}(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{k=1}^{n}(x_k^{(i)} - x_k^{(j)})^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x^{(i)} - x^{(j)}\|^2\right)$$

Al programarlo se utiliza el kernel RBF, que es equivalente al gaussiano pero sustituyendo la constante $1/2\sigma^2$ por γ.





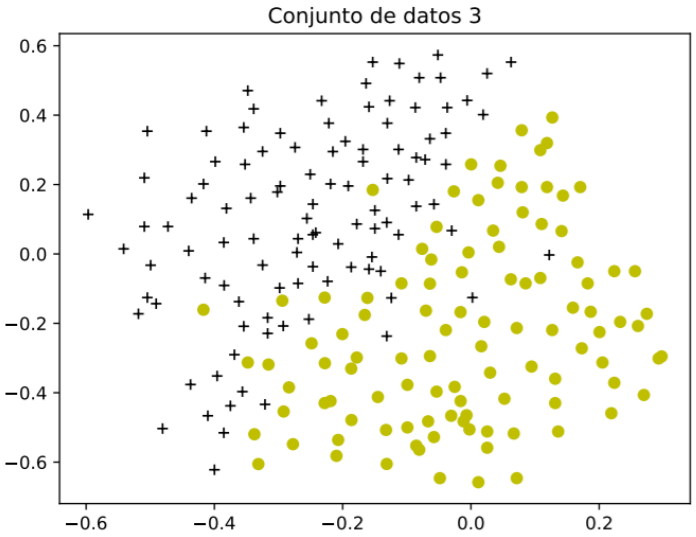

```
ef Ejericio2(reg, sigma):
    '''
    Con kernel gaussiano
    '''
    # 1. Cargamos los datos
    data = loadmat('ex6data2.mat')
    X = data['X']
```

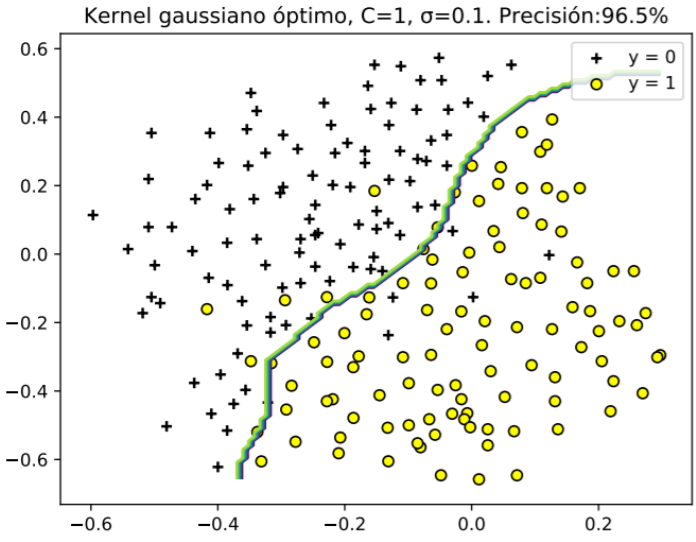
```
    y = data['y'].ravel()

    # Hacemos el SVM con kernel gaussiano
    svm = SVC(kernel='rbf', C=reg, gamma=1 / (2 * sigma ** 2))
    svm.fit(X, y)

    # Pintamos la frontera
    pintaFrontera(X, y, svm, ("Kernel gaussiano C=" + str(reg) + ", σ=" +
 str(sigma)))

    # Guardamos la gráfica y cerramos
    #plt.show()
    plt.savefig("SVMGaussiano_C=" + str(reg) + "_sigma" + str(sigma) + ".
pdf")
    plt.close()
```

1.3 Elección de los parámetros C y σ

En este último apartado el objetivo es encontrar los mejores valores de C y σ para clasificar el conjunto. Además de datos de entrenamiento también tenemos datos de validación con los que se calcula el porcentaje de ejemplos que se clasifican correctamente.

```
Mejor porcentaje:96.5% con C=1, σ= 0.1
```






```
def Ejercicio3(values):
    '''
    Con kernel gaussiano y eligiendo bien los parámetros C y σ
    '''
    # 1. Cargamos los datos
    data = loadmat('ex6data3.mat')
    X = data['X']
    y = data['y'].ravel()
    Xval = data['Xval']
```

```
    Yval = data['yval']

    #Lista de modelos
    svm = []

    mejor = 0 #El mejor modelo con la validación
    mejor_porc = 0

    cOPt = values[0]
    sigmaOpt = values[0]

    #Todas las posibles combinaciones de modelos con los valores dados pa
ra C y sigma
    actual = 0
    for i in range (len(values)):
        for j in range (len(values)):
            # Hacemos el SVM con kernel gaussiano
            svm.append(SVC(kernel='rbf', C=values[i], gamma=1 / (2 * valu
es[j] ** 2)))
            svm[actual].fit(X, y)

            #Vemos el porcentaje de aciertos
            h = svm[actual].predict(Xval)
            porc = calcula_porcentaje(Yval.ravel(), h, 4)

            #Si el porcentaje es mejor que el máximo, actualizamos
            if(porc > mejor_porc):
                mejor = actual
                mejor_porc = porc
                cOpt = values[i]
                sigmaOpt = values[j]

            actual+=1 #Avanzamos

    print("Mejor porcentaje:" + str(mejor_porc) + "% con C=" + str(cOpt)+
 ", σ=", str(sigmaOpt))


    # Pintamos la frontera
    pintaFrontera(X, y, svm[mejor], "Kernel gaussiano óptimo, C=" + str(c
Opt) + ", σ=" + str(sigmaOpt) + ". Precisión:" + str(mejor_porc) + "%")
    plt.savefig("SVMGaussianoOptimo_C=" + str(cOpt) + "_sigma" + str(sigm
aOpt) + ".pdf")
    plt.close()
```

2. Detección de spam

Resumen

Utilizar las funciones para el cálculo de modelos SVM del apartado anterior para detectar correos *spam.*

2.1 Procesamiento de los correos

Se procesan los correos para generar los datos de entrenamiento y validación. Para facilitar el aprendizaje se transforma el texto de los mensajes a una lista de elementos. A partir de esta lista se representan las palabras que aparecen en un vector de 0s y 1s, el cual tiene como longitud el número de palabras del diccionario.

```
def MakeTrainData(folder, size, Yvalue):
    '''
    Lee los archivos de la carpeta correspondiente, los guarda en las X
    y pone el valor de las Y correspondiente
    '''
     #Leemos los de spam
    for i in range(size):
        email_contents = open('{0}/{1:04d}.txt'.format(folder, i+1), 'r',
 encoding='utf-8', errors='ignore').read()
        email = email2TokenList(email_contents)

        # Palabras del diccionario
        vocab = getVocabDict()
        words = np.zeros([len(vocab)], dtype=int) #Será la X

        # Vemos cuáles aparecen en el correo y rellenamos con 1's y 0's
        for i in range (len(email)):
            token = email[i]
            # Comprobamos que no es una palabra incompleta o mal escrita
            if token in vocab:
                pos = vocab[token] #Posición que ocupa la palabra en el
                                   diccionario
                words[pos - 1] = 1 #Ponemos esa posición a 1 (empezando
                                   en 0)
        X.append(words)
        y.append(Yvalue)
```

2.2 Uso de SVM

Tras cargar y generar todos los datos los separamos entre los de entrenamiento, validación y los de test. Una vez separados, entrenamos el SVM. Para ello utilizamos el kernel RBF.

```
def SpamDetector(values):
    '''
    Detector de spam usando SVM
    '''
    # 1. Leemos los correos y los añadimos a la X e Y
    # Spam
    MakeTrainData("spam", NUM_SPAM, 1)
    # No spam
    MakeTrainData("easy_ham", NUM_EASY_HAM, 0)
    MakeTrainData("hard_ham", NUM_HARD_HAM, 0)

    # 2. Hacemos la DIVISIÓN de entrenamiento, validación y test
    #Dividimos en entrenamiento / test (80% - 20%)
    X_train, X_test, y_train, y_test = train_test_split(np.asarray(X), np
.asarray(y), test_size=0.2, random_state=42)
    #Volvemos a dividir en entrenamiento / validación, para tener un tota
l de 60% (train), 20% (val), 20% (test)
    X_train, X_val, y_train, y_val = train_test_split(np.asarray(X_train)
, np.asarray(y_train), test_size=0.25, random_state=42)
    # 2217, 660, 661

    # 3. ENTRENAMOS el SVM y nos quedamos con los parámetros que den mejo
r resultado sobre el conjunto de validación
    svm, C, sigma = getBestSVMModel('rbf', values, X_train, y_train, X_va
l, y_val)

    # 4. Vemos el porcentaje sobre los ejemplos de test
    h = svm.predict(X_test)
    porc = calcula_porcentaje_Y(y_test.ravel(), h, 4)

    print("Precisión del clasificador de spam: " + str(porc) + "%, C=" +
str(C) + ", σ=" + str(sigma))
```

```
Precisión del clasificador de spam: 86.233%, C=0.1, σ=1
```