Análisis matemático para inteligencia artificial CEIA

TP 2
Optimización

Alumno: Pedro Perez
Docente: Ing. Magdalena Bouza

Descripción del problema:

Se busca resolver un problema de clasificación binaria para un conjunto de datos que se encuentran en $\mathbb{R}^2$. Se provee de dos sets de datos uno para entrenar el modelo y otro para testear el modelo. Los datos siguen la forma $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$ donde cada (x_i, y_i) se corresponde a una observación $x_i \in \mathbb{R}^2$, y su etiqueta corresponde a $y_i = \{-1,1\}$.

Se busca poder clasificar de la mejor manera posible las dos clases utilizando una SVM (Support Vector Machine) a través de una implementación utilizando gradiente descendiente y utilizando SVM de la librería scikit-learn.

Set de datos:

Se provee un set de datos de entrenamiento etiquetados con 319 observaciones (x_i, y_i). Se provee un set de datos para test etiquetados con 79 observaciones (x_i, y_i).

En la figura 1 se graficó el set de datos de entrenamiento para observar cómo se encuentran distribuidos, se observa que los datos con etiqueta 1 se encuentra distribuidos en la parte superior del grafico mientras que los datos con etiqueta -1 en la parte inferior.

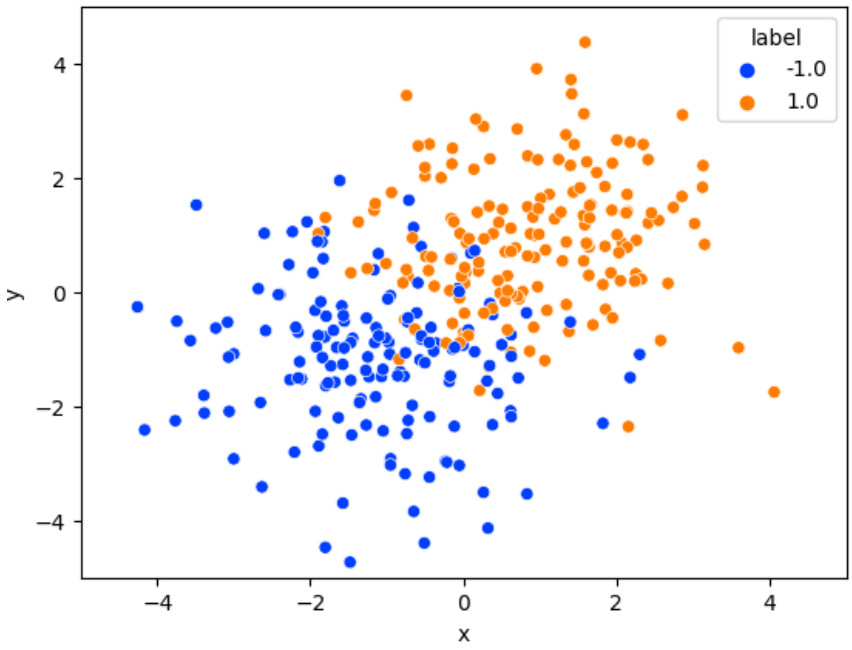

Figura 1. Distribución datos de entrenamiento según su clase.

Clasificación utilizando SVM y gradiente descendiente:

Las máquinas de vector de soporte o SVM (Support Vector Machine) es un algoritmo de machine learning supervisado utilizado para clasificar clases. El objetivo del algoritmo SVM es encontrar un hiperplano en el espacio N dimensional (donde N es el número de características o features del modelo) que permita clasificar los datos. En un espacio de dos dimensiones el hiperplano es una recta.

Cuando nos encontramos en dos dimensiones el objetivo de las SVM es encontrar una recta optima que separe a las dos clases, la recta será determinada por aquellos puntos que se encuentre más cercanos a la misma. La recta optima será aquella que posea la mayor distancia posible (margen) a los puntos más cercanos de las dos clases (vectores de soporte) en nuestro set de datos de entrenamiento.

Cuando los datos no son perfectamente separables a través de un hiperplano o recta si nos encontramos en $\mathbb{R}^2$, la idea es crear un clasificador que permita que ciertas observaciones se encuentren del lado incorrecto del margen que las separa, penalizando aquellos puntos que se encuentren del lado incorrecto. Por lo tanto, el objetivo de la SVM es encontrar el hiperplano en $\mathbb{R}^n$ de la forma $w^T x + b$ que separe a las dos clases con el mayor margen posible manteniendo la penalización por clasificación errónea lo más baja posible, lo que buscamos un $w \in \mathbb{R}^n$, b $\in \mathbb{R}$ optimo. La clasificación a través de SVM se fundamentan en un proceso de optimización convexa con restricciones que busca minimizar la siguiente ecuación:

$$\min_{w,b,\xi} \ \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{n} \xi_k$$

$$s.t. \qquad y_k(\langle w, x_k \rangle + b) \geq 1 - \ \xi_k \qquad k = 1, \ldots . n$$

$$\xi \geq 0$$

La constante C penaliza aquellas observaciones que fueron mal clasificadas. Si el valor de C es muy grande, no se permitirán ninguna violación del margen. Caso contrario, cuando C se aproxima a cero, menos se penalizan los errores y más observaciones pueden estar del lado incorrecto del margen, por lo cual el margen va a ser más ancho y más observaciones se convertirán en vectores de soporte. C es el hiperparámetro encargado de controlar el balance entre el sesgo y la varianza del modelo, cuando C está cerca de cero, el modelo estará sustentado por una cantidad mayor de vectores de soporte lo cual hará que aumente el sesgo, pero reduce la varianza. Cuando mayor es el valor de C, menor es el margen, menos observaciones será vectores de soporte y el clasificador tendrá un menor sesgo pero una varianza mayor.

De esta forma tratamos de penalizar aquellas muestras que fueron mal clasificadas en el entrenamiento adicionándole el factor $C\xi_k$. Para resolver el problema podemos utilizar la función hinge loss:

$$l_{hinge}(z) = \max\{0, 1 - z\}$$

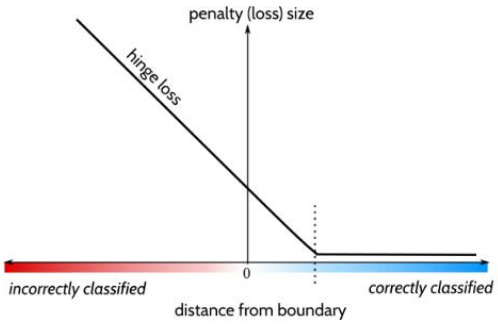

Figura 2. Higen loss.

De forma alternativa el problema primal se puede rescribir en términos de una función de costos que se desea minimizar a través de gradiente descendiente:

$$J(w, b) = \frac{1}{2}\|w\|^2 + C\frac{1}{n}\sum_{k=1}^{n}\underbrace{\max\{0, 1 - y_i(\langle w, x_i\rangle + b)\}}_{hinge\ loss}$$

Sabemos que la hinge loss es una función convexa y derivable en casi todos sus puntos, por lo que podemos solucionar el problema de SVM utilizando gradiente descendente:

$$\frac{\partial J}{\partial w} = \begin{cases} w & si \ \max\{0, 1 - y_i(\langle w, x_i\rangle + b)\} = 0 \\ w - Cy_i x_i & En\ cualquier\ otro\ caso \end{cases}$$

$$\frac{\partial J}{\partial b} = \begin{cases} 0 & si \ \max\{0, 1 - y_i(\langle w, x_i\rangle + b)\} = 0 \\ Cy_i & En\ cualquier\ otro\ caso \end{cases}$$

Implementación en Python:

Punto 1:

Se creó una función para calcular el gradiente de la función de costo con respecto a w y b:

```
def calculo_gradiente(W,b,X,Y,hyper_params):
    t = 1 - Y * (np.dot(X,W)+b)
    t[t<0] = 0

    dw_total = np.zeros(len(W))
    db_total = 0

    cont = 0
    for  i in enumerate (t):
        if i==0:
            dw = W
            db = 0
        else:
            dw = W - hyper_params['C'] * X[cont] * Y[cont]
            db = hyper_params['C'] * Y[cont]
        dw_total += dw
        db_total += db
        cont += 1
    dw_total = dw_total / cont
    db_total = db_total / cont


    grads = {'dw':dw_total, 'db':db_total}
    return grads
```

Punto 2:

Se realizó la implementación del algoritmo de gradiente descendiente, para cada uno de los epoch se calcula el gradiente y se actualizan los parámetros utilizando momentum con el fin de mejorar la optimización.

Para evitar el sobre entrenamiento, se utilizó como criterio de corte que la diferencia entre el costo del epoch anterior y el actual fuese menor a un delta:

```
for epoch in range(1, max_epochs):

    X, y = X_train, y_train
    X , y, numero_minibatch = creo_mini_batch(X_train,y_train,batch_size
    for i in range (numero_minibatch):
        grads = calculo_gradiente(W,b, X[i], y[i],hyper_params)
        W, b, states =  momentum_update(W,b, grads, states,hyper_params)
    costo = calculo_costo (W,b,X[i],y[i],hyper_params)
    lista_costo.append(costo)
    if abs(costo-costo_prev) < cost_delta_minimo * costo_prev:
        break
    costo_prev = costo
```

Se implementó la función que actualiza los valores de w y b utilizando momentum:

```
def momentum_update(W,b, grads, states, hyper_params):
  # hyper-param typical values: learning_rate=0.01, momentum=0.9

    W_ant = W
    b_ant = b
    W = W + hyper_params['momentum'] * states[0] - hyper_params['lr'] * grads['dw']
    b = b + hyper_params['momentum'] * states[1] - hyper_params['lr'] * grads['db']
    states = [W-W_ant, b-b_ant]
    return W,b, states
```

Punto 3:

Se cargaron los datos del archivo train.csv y se utilizaron para entrenar el modelo:

```
#Leo los datos de entrenamiento:
train = pd.read_csv('train.csv',  sep=',', names=["x", "y", "label"], header = 1)
```

Se utilizaron los siguientes hiperparametros para entranar el modelo:

```
hyper_params = {'lr': 0.01, 'C': 0.05, 'momentum': 0.7}
max_epochs = 1000
cost_delta_minimo = 0.0001
batch_size = 319
```

Se entreno el modelo utilizando diferentes tamaños de bach. En la figura 3 se observa que cuando el tamaño de minibatch es muy pequeño, el decenso del costo es mucho más rapido. Cuando se utiliza la totalidad de la muestra el decenso del costo es mucho más suave y uniforme.

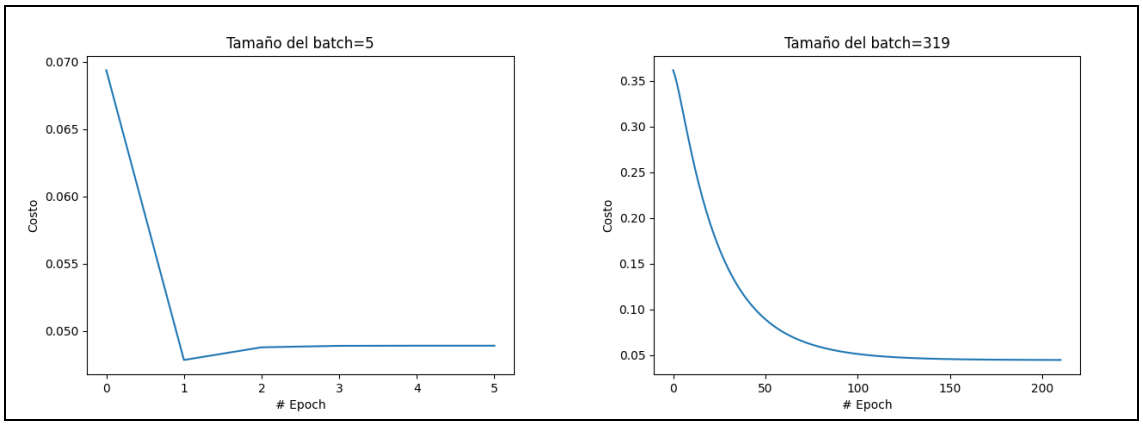

Figura 3. Entrenamiento utilizando diferentes tamaños de batch.

Para el resto del ejercicio se seleccionó un batch de tamaño 319 (tamaño total del set de datos de entrenamiento) para entrenar el modelo.

En la figura 4 se muestra la recta de decisión graficada con los valores de w y b obtenidos al minimizar la función de costo por gradiente descendente con el set de datos de entrenamiento:

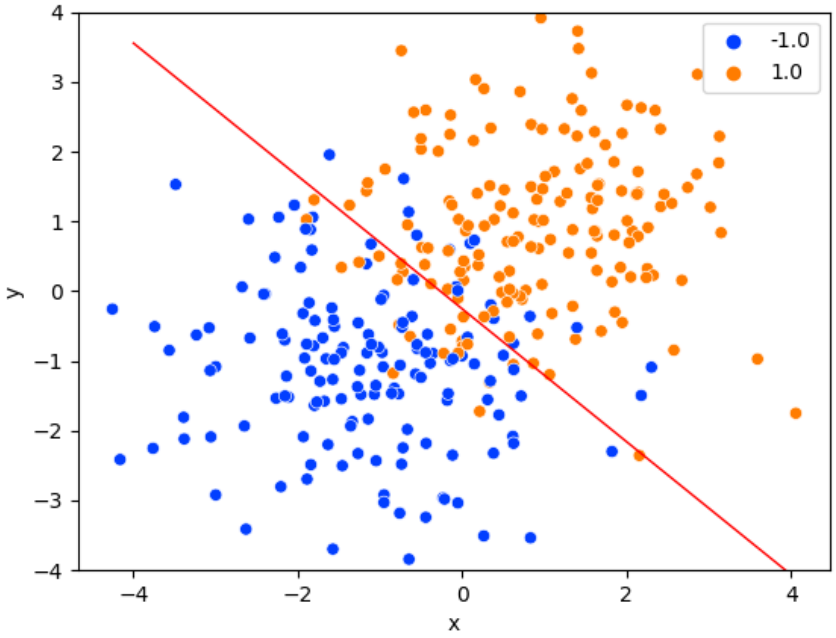

Figura 4. Set de datos de entrenamiento y línea de decisión por GD.

Punto 4:

Se implementó el algoritmo de clasificación SVM de la librería scikit-learn y se entrenó el modelo utilizando un kernel lineal, ya que nuestro problema es separable linealmente:

```
#Implementación con sklern
from sklearn.svm import SVC
model = SVC(C=0.5, kernel='linear')
clf = model.fit(X_train, y_train)
```

En la figura 5 se muestra la recta de decisión graficada con los valores de w y b óptimos obtenidos al entrenar el modelo con SVM de scikit-learn:

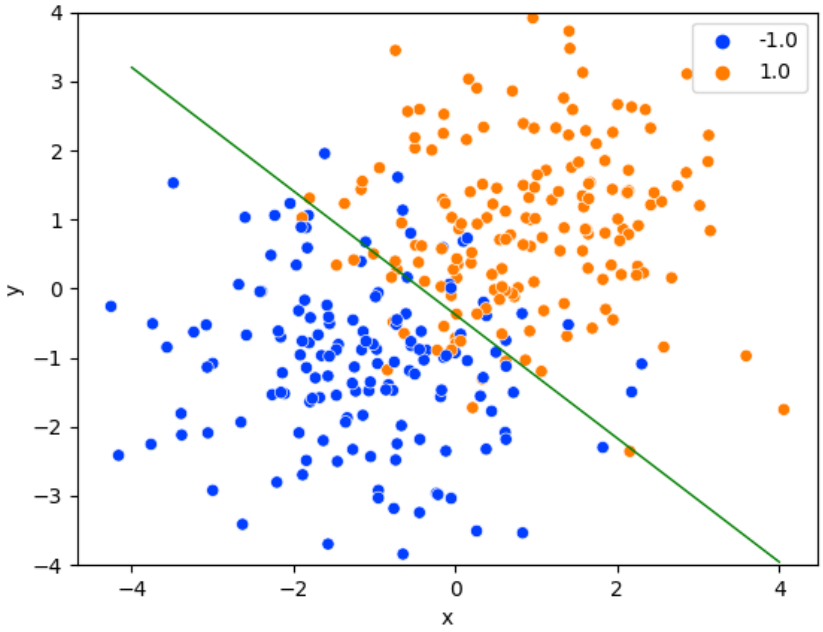

Figura 5. Set de datos de entrenamiento y línea de decisión scikit-learn.

Se llevaron a la forma canónica de la recta los valores de w y b para comparar las rectas obtenidas con ambos modelos.

$$w[0] * x + w[1] * y + intercept = 0$$

$$y = -\frac{w[0]x}{w[1]} - \frac{intercept}{w[1]}$$

Recta de decisión calculada por GD:

```
y= -0.9519805745509471 x - 0.25504543936538765
```

Recta de decisión utilizando scikit-learn:

```
y= -0.8942529546180036 x - 0.37771858462356794
```

Las dos rectas de decisión obtenidas por ambos modelos son muy parecidas. El modelo entrado utilizando gradiente descendente separa 13 observaciones de la clase -1 y 31 observaciones de la clase 1 de forma incorrecta. Mientras que el modelo entrenado utilizando la librería de scikit-learn separa 18 observaciones de la clase -1 y 18 observaciones de la clase 1 de forma incorrecta. En la figura 6, se muestra las dos rectas de decisión, la recta del modelo entrenado utilizando GD de color rojo y la recta de decisión obtenida utilizando la librería SVM de scikit-learn de color verde:

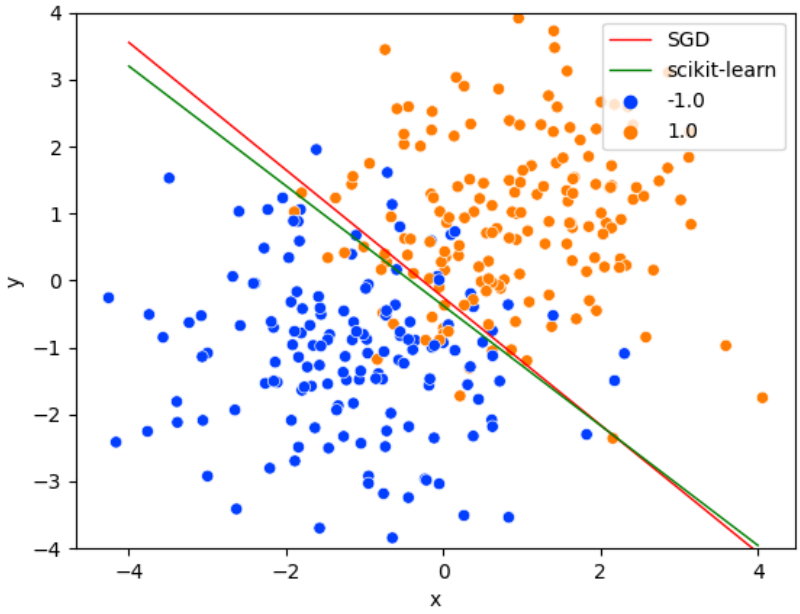

Figura 6. Rectas de decisión ambos modelos.

En la figura 7 se muestra la recta obtenida utilizando un valor de C de 0.9 en la cual se obtuvieron 89 vectores de soporte, y en la figura 8 un gráfico en la cual se utilizó un C=0.001 en la cual se obtuvieron 282 vectores de soporte y un margen mucho más amplio. Se observa que a medida que se disminuye el valor de C en el entrenamiento del modelo de scikit-learn aumenta el número de vectores de soporte, esto se debe a que se está penalizando cada vez menos aquellos puntos que fueron mal clasificados, por lo que cada vez más puntos podrán estar en el lado incorrecto, por lo que el margen será más grande.

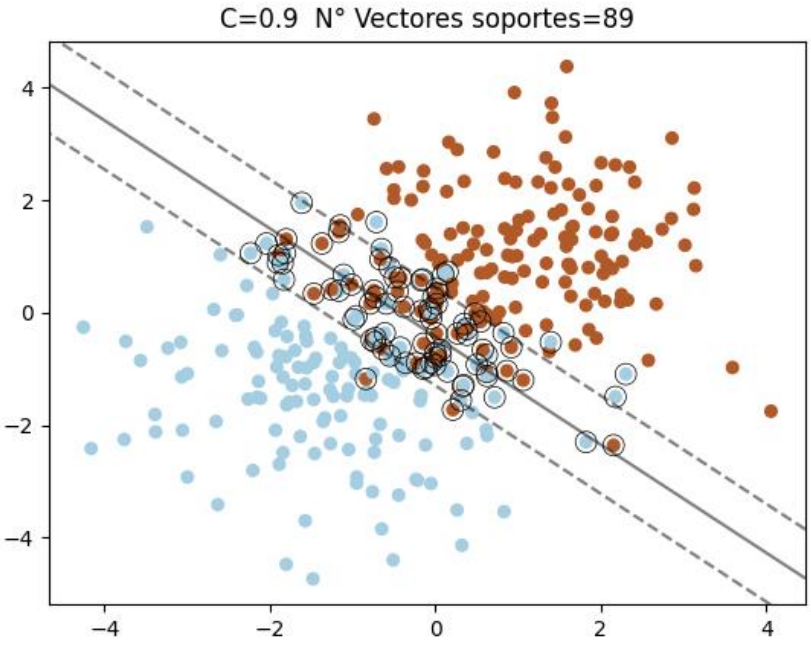

Figura 7. Vectores soporte C=0.9

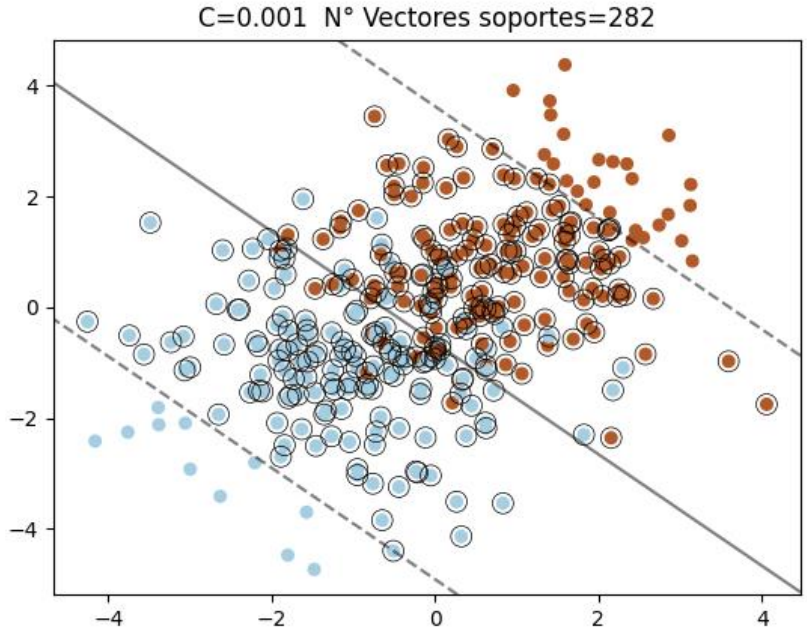

Figura 8. Vectores soporte C=0.001

Punto 5:

Se calculó la precisión del modelo entrenado utilizado gradiente descendiente sobre el set de datos de test. Se observó que para la clase y=-1, se lograron clasificar 34 observaciones de 35 de forma correcta, y para la clase y=1 se lograron clasificar 40 observaciones de 44 de forma correcta, por lo que podemos decir que el modelo tiene una precisión de 0.9367.

```
#Precisión set de test utilizando SGD:
y_predecida = []
a = -W[0] / W[1]
for i in range(X_test.shape[0]):
    y_linea = a * X_test[i][0] + b
    if X_test[i][1] > y_linea:
        y_predecida.append(1)
    else:
        y_predecida.append(-1)
print(confusion_matrix(y_test,y_predecida))
```

```
[[34  1]
 [ 4 40]]
```

```
precision = (79-5)/79
print("Precision del modelo="+ str(np.round (precision,decimals=4)))
```

```
Precision del modelo=0.9367
```

Se calculó la precisión del modelo entrenado utilizado SVM de scikit-learn sobre el set de datos de test. Se observó que para la clase y=-1, se lograron clasificar 32 observaciones de 35, y para la clase y=1 se lograron clasificar 41 observaciones de 44, por lo que podemos decir que el modelo tiene una precisión de 0.9241.

```
#Predicción utilizando scikit-learn:
#Implementación con sklern
model = SVC(C=0.5, kernel='linear')
clf = model.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test,y_pred))
```

```
[[32  3]
 [ 3 41]]
```

```
precision = (79-6)/79

print("Precision del modelo="+ str(np.round (precision,decimals=4)))
```

```
Precision del modelo=0.9241
```

Podemos decir que no existe diferencia significativa entre la presión de ambos modelos.

Punto 6:

Si se eliminan observaciones que son parte de los vectores de soporte de un modelo de SVM y se entrena nuevamente el modelo sin esas observaciones se obtiene un modelo diferente al modelo inicial, ya que los vectores de soporte son los que determinan cual va a ser la mejor recta que separe las observaciones. Por el contrario, si se eliminan observaciones que no forman parte de los vectores de soporte de un modelo SVM y se entrena nuevamente el modelo sin esas observaciones el modelo permanece igual.

Se eliminan observaciones que son vectores soportes:

Del modelo inicialmente entrenado se tomaron observaciones que fueron parte de los vectores de soportes y se eliminaron creando un nuevo set de datos sin esas observaciones. Como resultado se obtiene un modelo con menor número de vectores de soporte y con diferentes valores de w y b.

Recta de decisión obtenida con el set datos completos:

$$y= -0.8942529546180036\ x - 0.37771858462356794$$

Recta de decisión obtenida con el set datos sin algunos vectores de soporte utilizando scikit-learn:

$$y= -0.996106293411652\ x - 0.735408675521281$$

En la figura 9 se muestra un gráfico resultante luego de entrenar el modelo utilizando el nuevo set de datos de entrenamiento con un número menor de observaciones que fueron vectores de soporte.

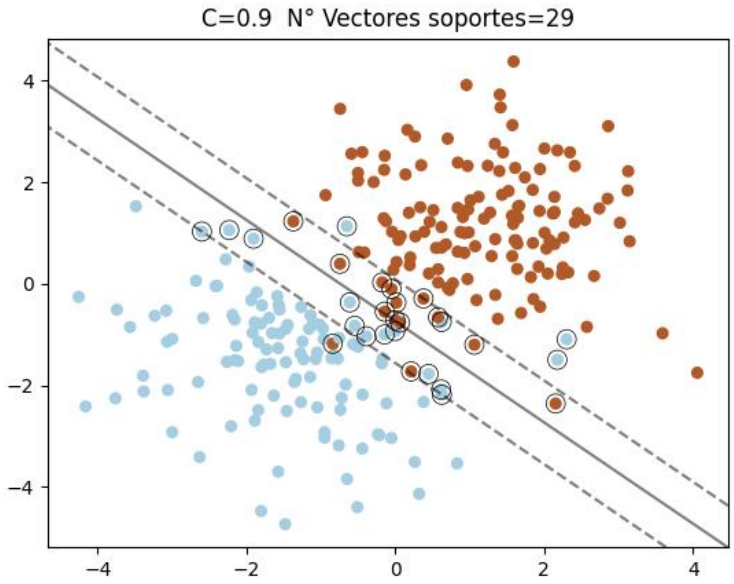

Figura 9. Modelo sin algunos vectores soporte.

Se eliminan observaciones que no son vectores soportes:

Del modelo inicialmente entrenado se tomaron observaciones que no fueron parte de los vectores de soportes y se eliminaron creando un nuevo set de datos sin esas observaciones.

Como resultado se obtiene un modelo con igual número de vectores de soporte y con iguales valores de w y b.

Recta de decisión obtenida con el set datos completos:

```
y= -0.8942529546180036 x - 0.37771858462356794
```

Recta de decisión obtenida con el set datos sin algunas observaciones que no fueron vectores de soporte:

```
y= -0.8941868963894747x + -0.37778717885801183
```

En la figura 10 se muestra un gráfico resultante luego de entrenar el modelo utilizando el nuevo set de datos de entrenamiento con un número menor de observaciones que no fueron vectores de soporte.

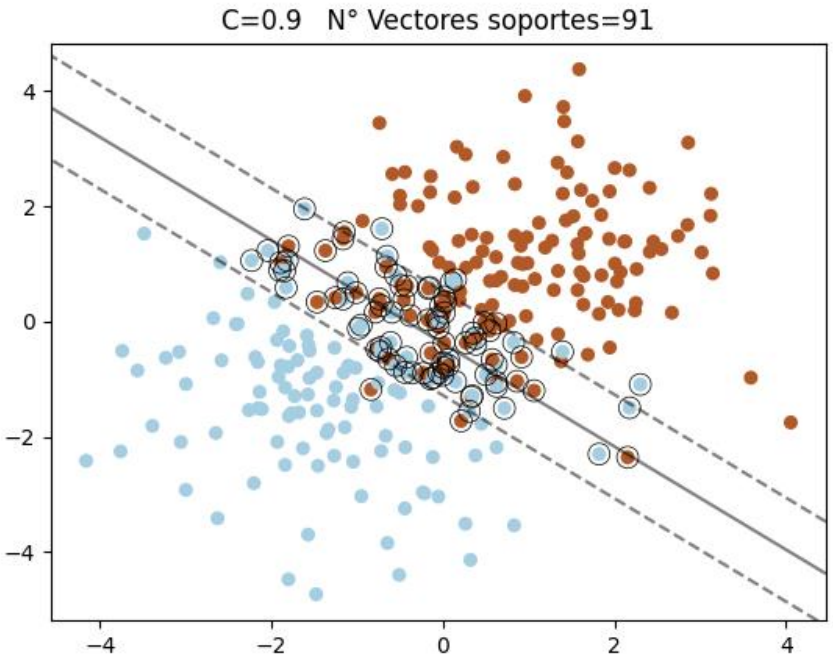

Figura 10. Modelo sin algunas observaciones.