1. Elijo el ejercicio 1

2.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.neighbors import KNeighborsClassifier

clientes = pd.read\_csv("creditos.csv")

#graficar los datos

buenos = clientes[clientes["cumplio"]==1]

malos = clientes[clientes["cumplio"]==0]

plt.scatter(buenos["edad"], buenos["credito"],

            marker="\*", s=150, color="skyblue",

            label="Sí pagó (Clase: 1)")

plt.scatter(malos["edad"], malos["credito"],

            marker="\*", s=150, color="red",

            label="No pagó (Clase: 0)")

plt.ylabel("Monto del crédito")

plt.xlabel("Edad")

plt.legend(bbox\_to\_anchor=(1, 0.2))

#plt.show()

#Normalizar los datos

datos = clientes[["edad", "credito"]]

clase = clientes["cumplio"]

escalador = preprocessing.MinMaxScaler()

datos = escalador.fit\_transform(datos)

#Entrenar el clasificador

clasificador = KNeighborsClassifier(n\_neighbors=3)

clasificador.fit(datos, clase)

#nuevo solicitante

edad = 53

monto = 350000

#Escalar los datos del nuevo solicitante

solicitante = escalador.transform([[edad, monto]])

#Calcular clase y probabilidades

print("Clase:", clasificador.predict(solicitante))

print("Probabilidades por clase",

      clasificador.predict\_proba(solicitante))

#Código para graficar

plt.scatter(buenos["edad"], buenos["credito"],

            marker="\*", s=150, color="skyblue", label="Sí pagó (Clase: 1)")

plt.scatter(malos["edad"], malos["credito"],

            marker="\*", s=150, color="red", label="No pagó (Clase: 0)")

plt.scatter(edad, monto, marker="P", s=250, color="green", label="Solicitante")

plt.ylabel("Monto del crédito")

plt.xlabel("Edad")

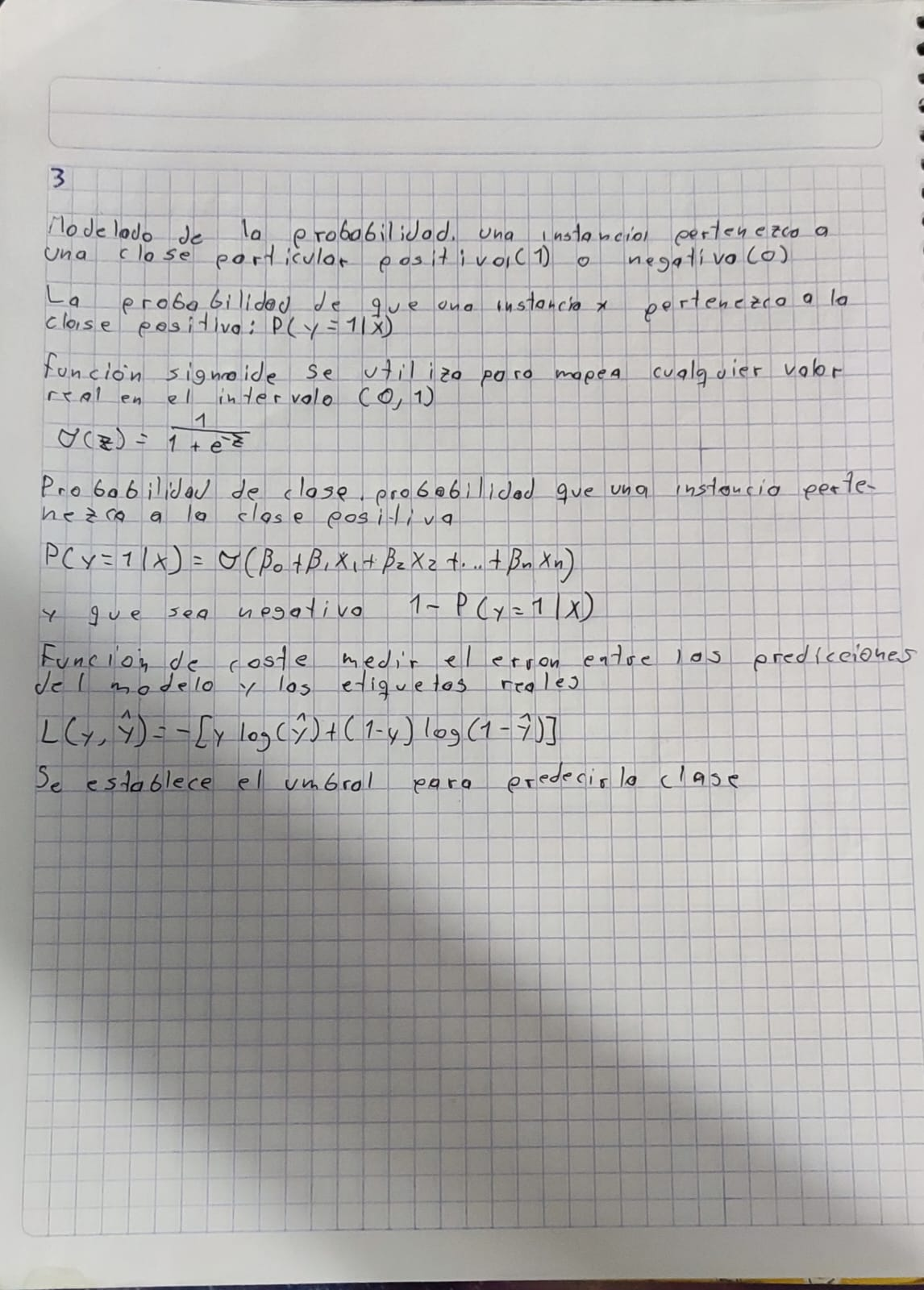
plt.legend(bbox\_to\_anchor=(1, 0.3))

#plt.show()

Interfaz de usuario gráfica, Gráfico, Gráfico de dispersión

Descripción generada automáticamente

3.



4. Elijo el ejercicio4

5. Elijo el ejercicio 5

6.

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm

from sklearn.datasets import make\_blobs

import matplotlib

# Generamos 50 muestras con dos características, asociadas a dos clases

X, y = make\_blobs(n\_samples=50, n\_features=2, centers=2, random\_state=21, center\_box=(0, 10.0))

# Creamos el modelo SVM para clasificación con kernel lineal y entrenamos el modelo

clf = svm.SVC(kernel='linear', C=100)

clf.fit(X, y)

# Graficamos los datos en el espacio de características

cmap = matplotlib.colors.ListedColormap(['blue', 'red'])

plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=cmap)

# Creamos un mesh para evaluar la función de decisión

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 = clf.decision\_function(xy).reshape(XX.shape)

# Graficamos el hiperplano y el margen

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

# Graficamos los vectores soporte

ax.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], s=100, facecolors='none', edgecolors='k')

# Predicción y gráfica de las fronteras de decisión con colores modificados

Z\_pred = clf.predict(np.c\_[XX.ravel(), YY.ravel()])

Z\_pred = Z\_pred.reshape(XX.shape)

cmap = matplotlib.colors.ListedColormap(['pink', 'yellow'])

plt.pcolormesh(XX, YY, Z\_pred, cmap=cmap, alpha=0.1)

# Predicción para un nuevo punto y gráfica del nuevo punto

new\_x = [[5, 0.5]]

new\_z = clf.predict(new\_x)

if new\_z[0] == 0:

    color = 'blue'

else:

    color = 'red'

plt.scatter(new\_x[0][0], new\_x[0][1], marker='+', color=color, s=300)

plt.grid()

plt.show()

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

7. Elijo el ejercicio 7