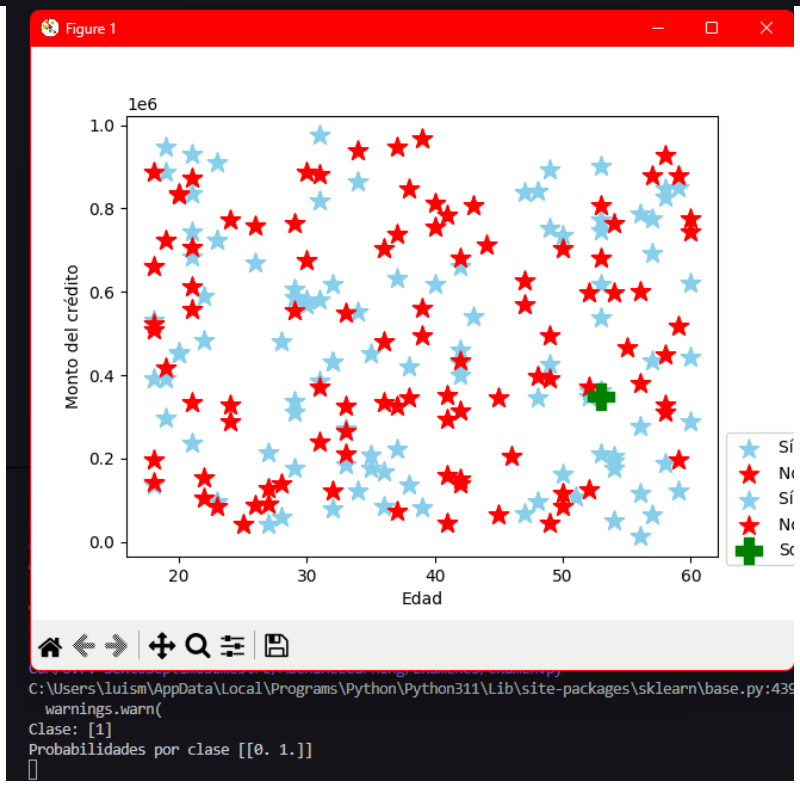
Torres Abonce Luis Miguel Examen Parcial 2

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()
```



3.

3

Modelado de la probabilidad. una instancia pertenezca a una clase particular positiva(1) o negativa(0)

La probabilidad de que una instancia x pertenezca a la clase positiva: $P(y=1|x)$

función sigmoide se utiliza para mapea cualquier valor real en el intervalo (0, 1)

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Probabilidad de clase. probabilidad que una instancia pertenezca a la clase positiva

$$P(y=1|x) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)$$

y que sea negativo $1 - P(y=1|x)$

Función de coste medir el error entre las predicciones del modelo y los etiquetas reales

$$L(y, \hat{y}) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Se establece el umbral para predecir la clase

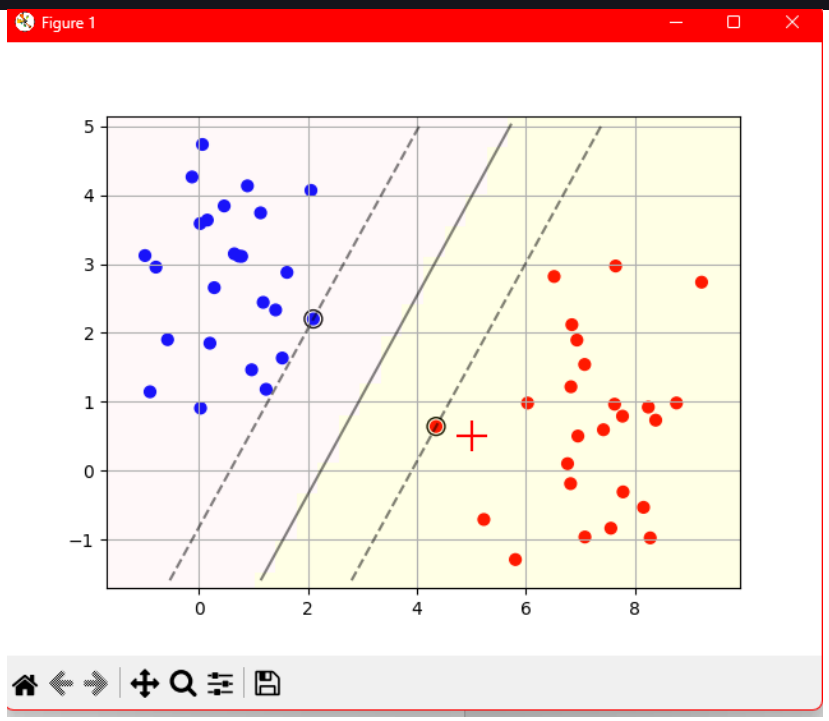
4. Elijo el ejercicio4

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()
```



7. Elijo el ejercicio 7