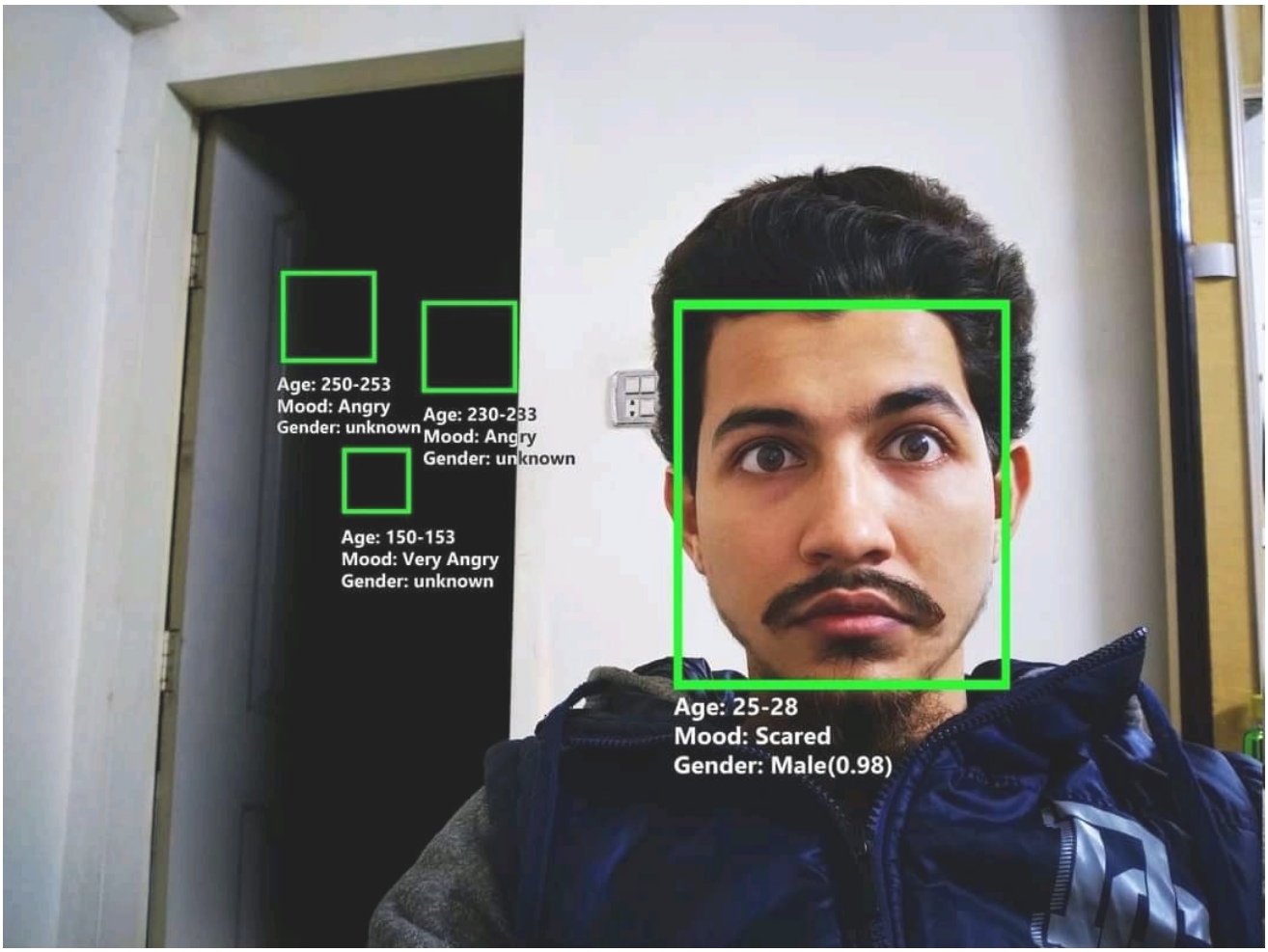
# Supervised Learning

by **Ivan Alducin**




## Exploratory Data Analysis

Vamos a trabajar con el conjunto de datos de `Heart Attack`, el objetivo es predecir bajo que escenario es más probable que un paciente pueda tener un ataque al corazón Un experto en medicina cardiovasuclar puede predecir esto sin hacer uso de *Machine Learning*, pero probablemente no instantáneamente, ¡y ciertamente no si estamos tratando con cientos o miles de muestras!.

A continuación una breve explicación de las variables del dataset:

- **age:** Age of the patient
- **sex:** Sex of the patient
- **cp:** Chest pain type ~ 0 = Typical Angina, 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
- **trtbps:** Resting blood pressure (in mm Hg)
- **chol:** Cholestoral in mg/dl fetched via BMI sensor
- **fbs:** (fasting blood sugar > 120 mg/dl) ~ 1 = True, 0 = False
- **restecg:** Resting electrocardiographic results ~ 0 = Normal, 1 = ST-T wave normality, 2 = Left ventricular hypertrophy
- **thalachh:** Maximum heart rate achieved
- **oldpeak:** Previous peak
- **slp:** Slope
- **caa:** Number of major vessels
- **thall:** Thalium Stress Test result ~ (0,3)
- **exng:** Exercise induced angina ~ 1 = Yes, 0 = No
- **output:** Target variable

In [17]:
```
# Archivo Heart Attack.csv - ¿Cuales son los factores que pueden incrementar o disminuir la probabilidad de un a

import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import os
os.chdir('/Users/Lenovo/Desktop/EBAC/Proyecto_EBAC')
```

```
df = pd.read_csv('Heart Attack.csv')
y = df.output.values
X = df.drop(['output'], axis = 1)
X
```

Out[17]:

| | age | sex | cp | trtbps | chol | fbs | restecg | thalachh | exng | oldpeak | slp | caa | thall |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 |

303 rows × 13 columns

In [5]:
```
# Hacer EDA (Exploratory Data Analysis) suele ser un tanto laborioso dependiendo del detalle al que se quiera l
#!pip install dataprep
#from dataprep.eda import create_report

#create_report(df)
```

In [13]:
```
import pandas as pd
import sweetviz as sv

report = sv.analyze(df)
report.show_html('reporte_sweetviz.html')
```

```
                                             |          | [  0%]   00:00 -> (? left)
Report reporte_sweetviz.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, th
e report IS saved in your notebook/colab files.
```

## k-Nearest Neighbors

Habiendo hecho un Análisis Exploratorio de los factores que pueden o no tener más posibilidad de un ataque al corazón, es hora de crear tu primer clasificador!!! usando el algoritmo de k-NN.

**Nota**: es importante garantizar que los datos esten en el formato requerido por la librería de `scikit-learn`. La información debe estar en una matriz en la que cada columna sea una variable y cada fila una observación diferente, en este caso, el registro de análisis clinico por paciente. Y la variable objetivo debe ser una sola columna con el mismo número de observaciones.

In [19]:
```
# Importa la librería para un clasificador k-NN de sklearn
from sklearn.neighbors import KNeighborsClassifier

# Crea dos arreglos "X", "y" que contengan los valores de las variables independientes y la variable objetivo
y = df.output.values
X = df.drop(['output'], axis = 1)

# Crea un clasificador k-NN con 6 vecinos
knn = KNeighborsClassifier(n_neighbors=6)

# Ajusta el clasificador a las variables
knn.fit(X, y)
```

Out[19]:

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

## Predicción

Una vez que entrenamos al clasificador k-NN, ahora lo podemos usar para predecir un nuevo registro. Para este caso, no hay datos sin clasificar disponibles ya que todos se usaron para entrenar al modelo. Para poder calcular una predicción, vamos a usar el método `.predict()` pero, para esto vamos a simular una observación completamente nueva

In [21]:
```
import numpy as np
```

```
# Crea un arreglo simulando una observación
X_new = np.array([[63, 1, 3, 145, 233, 1, 0, 150, 0, 2.3, 0, 0, 1]])

# Predice la clasificación para el arreglo que creaste
y_new_pred = knn.predict(X_new)
print("Prediction: {}".format(y_new_pred))
```

```
Prediction: [1]
```

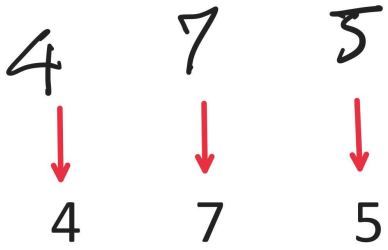
# Conclusion

Dado el resultado, podemos concluir que el paciente es propenso de un ataque al corazon.

## Reconocimiento de digitos

Hasta ahora, solo hemos realizado una clasificación binaria, ya que la variable objetivo tenía dos resultados posibles. En los siguientes ejercicios, trabajarás con el conjunto de datos de reconocimiento de dígitos MNIST, que tiene 10 clases, ¡los dígitos del 0 al 9! Una versión reducida del conjunto de datos MNIST es uno de los conjuntos de datos incluidos en `scikit-learn`

Cada muestra de este conjunto de datos es una imagen de 28x28 que representa un dígito escrito a mano. Cada píxel está representado por un número entero en el rango de 1 a 784, lo que indica niveles variables de negro.




In [32]:
```
# Importa el archivo de MNIST
digits = pd.read_csv('MNIST.csv')
digits
```

Out[32]:

| | label | pixel0 | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 | ... | pixel774 | pixel775 | pixel776 | pixel777 | pixel778 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **3** | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **41995** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **41996** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **41997** | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **41998** | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |
| **41999** | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 |

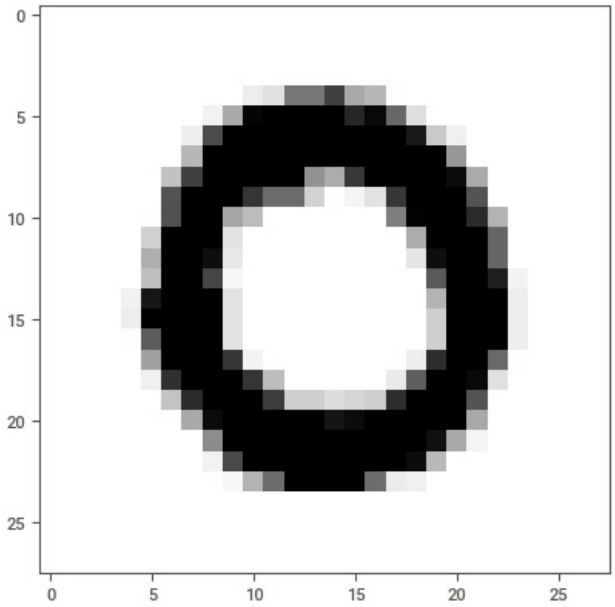
42000 rows × 785 columns

In [26]:
```
# Crea una variable 'cols' para hacer referencia a todas las columnas que contienen la palabra 'pixel'
cols = [col for col in digits.columns if "pixel" in col]
```

In [36]:
```
import matplotlib.pyplot as plt
# Vamos a imprimir un digito
i = 1
print("El número es:", digits.loc[i, 'label'])
plt.imshow(digits.loc[i, cols].values.reshape((28,28)).astype(float), cmap=plt.cm.gray_r, interpolation='neares
```

```
El número es: 0
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x1f86527fb60>
```




## Train/Test

Una de las principales diferencias entre la Estadística Clasica y el *Machine Learning* es la división del conjunto de datos en conjuntos de entrenamiento y prueba, con el objetivo de medir y cuantificar la precisión y el nivel de error en los datos que de alguna manera el modelo *"No ha visto"*. A continuación crearemos nuestros conjuntos de entrenamiento y prueba con el método `train_test_split` y mediremos cual es el nivel de precisión de nuestro modelo. El objetivo es **predecir cual es el digito dada una imagen**!!!. Para lo cual entrenaremos un clasificador *k-NN* a los datos de entrenamiento y luego calcularemos su precisión usando el método `accuracy_score()` en los datos de prueba ¿Como crees que en un modelo de Clasificación se calcule su precisión?. Parece bastante dificil, pero no lo es ;)

In [40]:
```
# Importa la librería para entrenamiento y prueba de datos y la librería para calcular la precisión
from sklearn.model_selection import train_test_split
```

In [50]:
```
# Crea los arreglos para las variables independientes y la variable objetivo
X = digits.drop(['label'], axis = 1)
y = digits.label.values

# Divide los arreglos anteriores en conjuntos de training y test en una proporcion del 70/30
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state= 1)

# Instancia un clasificador k-NN con 14 vecinos
knn = KNeighborsClassifier(n_neighbors= 14)

# Ajusta (Entrenamiento) el clasificador en el conjunto de entrenamiento
knn.fit(X, y)

# Calcular las predicciones sobre el conjunto de prueba
y_pred = knn.predict(X_test)

# Verificar la precisión del modelo
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

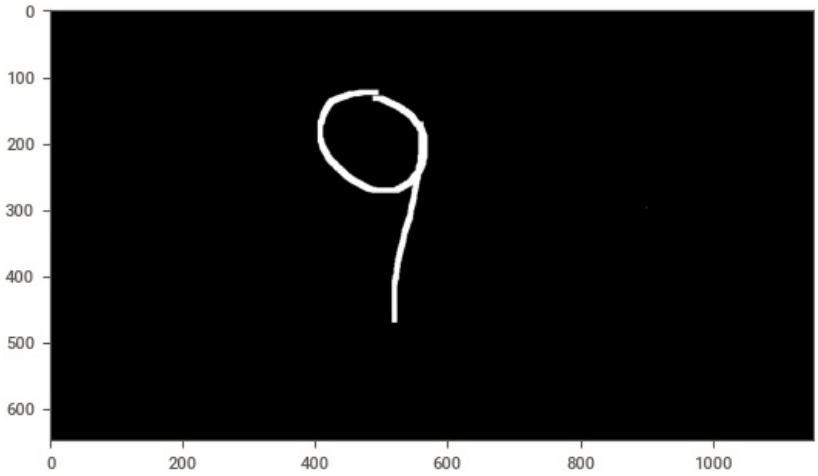
```
0.97
```

## Reconocimiento de tu imagen

Con todo lo anterior, podemos hacer el reconocimiento de cualquier digito que dibujes, ¿Estás list@?

In [54]:
```
# Vamos a visualizar la imagen de un número que vas a crear en tu computador con la aplicación de paint, ésta i
image = plt.imread('C:/Users/Lenovo/Desktop/EBAC/Proyecto_EBAC/EbacPaint.jpg') # Coloca aquí la ruta de la imag
plt.imshow(image)
```

Out[54]:
```
<matplotlib.image.AxesImage at 0x1f801dbcc20>
```



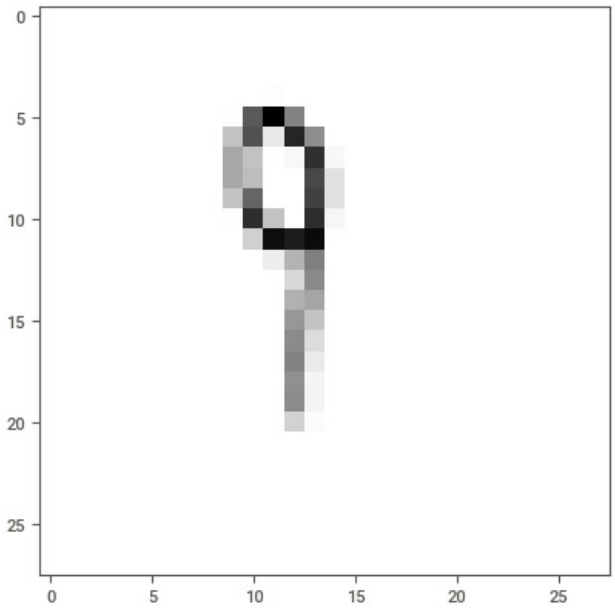

In [56]:
```
# Con esta libreria transformaremos la imagen creada a un formato de 28x28 pixeles
from PIL import Image
pil = Image.open('C:/Users/Lenovo/Desktop/EBAC/Proyecto_EBAC/EbacPaint.jpg')
image_resize = pil.resize((28, 28))

# Vamos transformar la nueva imagen en un array donde se almacenara la información de los pixeles
pixels = np.asarray(image_resize)
```

In [58]:
```
# Necesitamos hacer algunas configuraciones a la imagen debido al formato de datos que esta alimentando al mode
arr = pixels.transpose(2, 0, 1).reshape(-1, pixels.shape[1])[0:28]

image_final = arr.ravel().reshape(1, -1)
```

In [60]:
```
# Calcula la predicción del modelo con el número que creaste, ¿Fue la clasificación correcta? :O
print("El número es:", knn.predict(image_final))
plt.imshow(arr, cmap=plt.cm.gray_r, interpolation='nearest')
```

```
El número es: [1]
```

Out[60]:
```
<matplotlib.image.AxesImage at 0x1f8071673e0>
```

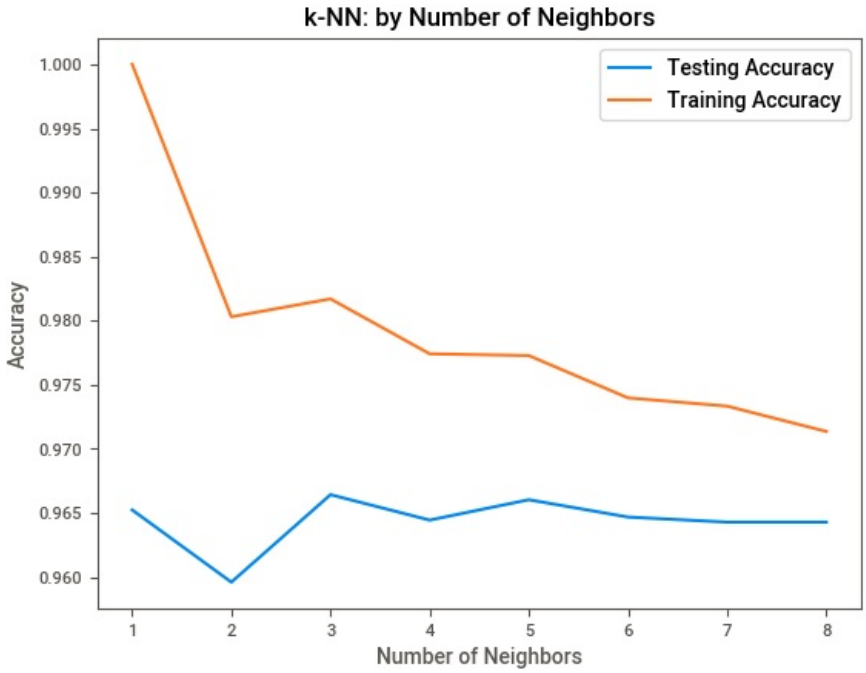



## Overfit and Underfit

¿Cual es mi numero ideal para elegir el parametro *k*? Vamos a calcular los valores de precisión para los conjuntos e entrenamiento y prueba para una rango de valores k. Al observar cómo difieren estos valores podremos observar cual es el mejor parametro sin caer en un *Overfit* o un *Underfit*.

In [68]:
```
# Coniguración de arreglos iniciales
neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
# Loop para diferentes valores de k
for i, k in enumerate(neighbors):
    # Clasificador k-NN para el parametro k
    knn = KNeighborsClassifier(n_neighbors = k)

    # Ajuste del clasificador al dataset de entrenamiento
    knn.fit(X_train, y_train)

    # Calculo de precision sobre el dataset de entrenamiento
    train_accuracy[i] = knn.score(X_train, y_train)

    # Calculo de precision sobre el dataset de prueba
    test_accuracy[i] = knn.score(X_test, y_test)

# Grafico para encontrar un valor optimo de k
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training Accuracy')
plt.title('k-NN: by Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




## Regresión Logística

Haz la predicción de tu imagen, pero esta vez por medio de una Regresión Logística, ¿Cuál de los dos modelos te da mejores resultados?

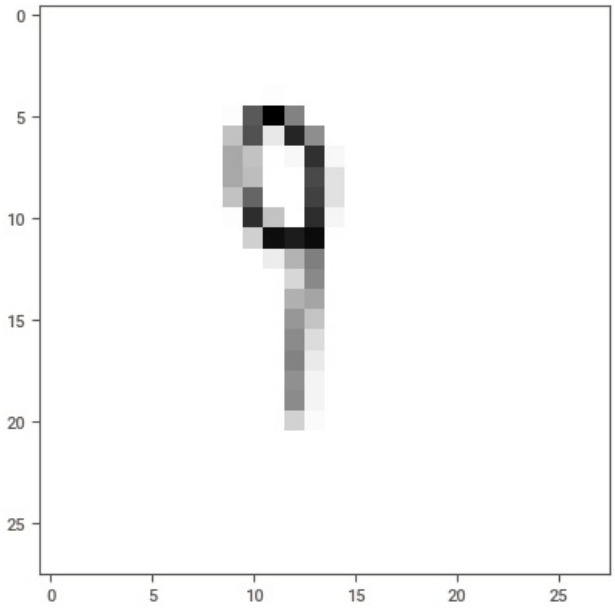
In [70]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver = 'sag')
clf = model.fit(X_train, y_train)

# Evaluacion de la precision del modelo
score = model.score(X_test, y_test)
print('Score de precision: ', score)
```

```
Score de precision:  0.915
```

In [72]:
```
# Calcula la predicción del modelo con el número que creaste, ¿Fue la clasificación correcta? :O
print("El número es:", model.predict(image_final))
plt.imshow(arr, cmap=plt.cm.gray_r, interpolation='nearest')
```

```
El número es: [8]
```

Out[72]:
```
<matplotlib.image.AxesImage at 0x1f82a9113a0>
```




# Conclusion

Utilizando ambos modelos, podemos llegar a la conclusion que ambos tienen acertividad muy elevada (por encima del 90%), por lo que podemos considerar que son muy buenos. Ahora, en relacion a la prediccion de la imagen, ambos fallaron, ya que KNN predijo que era el 1, y Regresion Logistica predijo que era 8.

In [ ]: