Examen Final

January 31, 2021

1 Examen Final - Edgar Gerardo Alarcón González

1.1 Número de cuenta: 31210231-9

[1]:
```
# Librerías

# General
import pandas as pd
import numpy as np
from scipy import integrate
from sklearn.metrics import mean_squared_error, mean_squared_log_error,␣
↪mean_absolute_error, median_absolute_error
from scipy.integrate import odeint
from scipy.optimize import differential_evolution, minimize
import matplotlib.pyplot as plt

# Ejercicio 1
import numpy.polynomial.polynomial as poly
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score

# Ejercicio 2
from keras.models import Sequential
from keras.layers import Dense

# Ejercicio 3
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# Ejercicio 4
import random
from mpl_toolkits import mplot3d
```

```python
import math
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

1.2 Ejercicio 1

Usando el data set llamado problem1.csv "(x_training, y_training)":

a) Enceuntra el polinomio que mejor ajusta los datos de entrenamiento.

b) Usando el criterio del AIC, encuentra el mejor polinomio que que puede ajustar los datos.

c) Realiza validación cruzada del polinomio con los datos llamados "problem1.csv (x_test, y_test)"

1.2.1 Solución

Comenzamos haciendo una lectura de los datos

[2]:
```python
# Letura de datos
datos = pd.read_csv('problem1.csv')
datos.head()
```

[2]:

	X_training	Y_training	X_test	Y_test
0	-2.00	22.067387	2.00	6.024049
1	-1.97	19.944915	2.05	6.885408
2	-1.94	18.062490	2.10	7.578968
3	-1.91	16.384313	2.15	8.439467
4	-1.88	14.567798	2.20	9.554611

Vamos ahora a guardar los datos de entrenamiento y de prueba

[3]:
```python
# Separar datos
train = datos[["X_training","Y_training"]]
train.head()
```

[3]:

	X_training	Y_training
0	-2.00	22.067387
1	-1.97	19.944915
2	-1.94	18.062490
3	-1.91	16.384313
4	-1.88	14.567798

[4]:
```python
test  = datos[["X_test","Y_test"]]
test.head()
```

[4]:

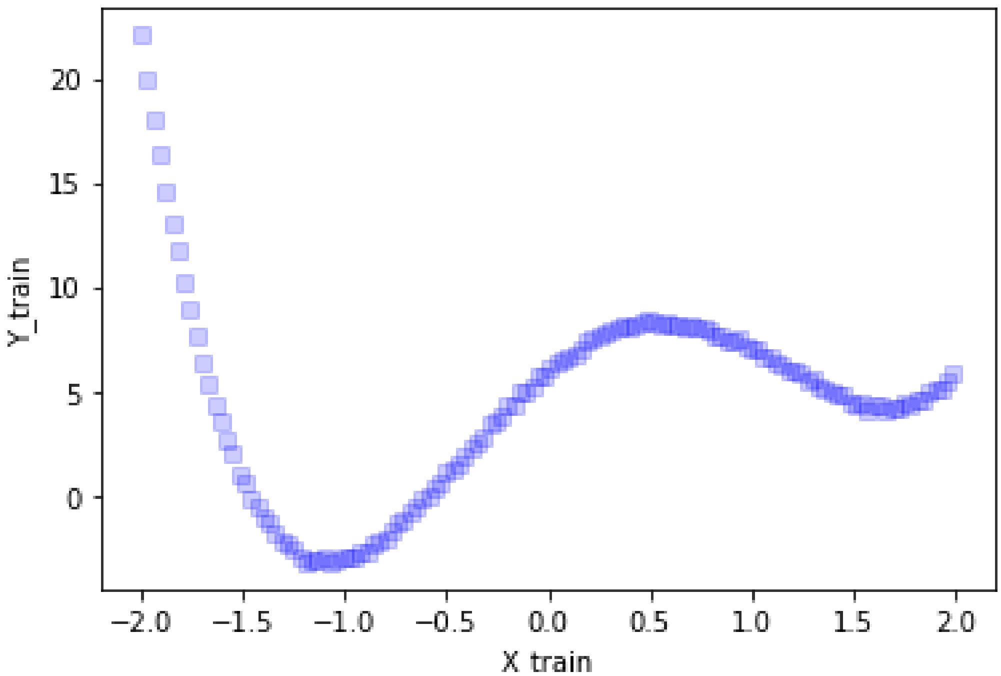
	X_test	Y_test
0	2.00	6.024049
1	2.05	6.885408
2	2.10	7.578968

```
3    2.15  8.439467
4    2.20  9.554611
```

Ahora hacemos un gráfico de los datos.

[5]:
```
plt.plot(train.X_training,train.Y_training,'bs',alpha=0.2,)
plt.xlabel('X_train')
plt.ylabel('Y_train')
#plt.savefig('fdx.pdf', format='pdf', dpi=1200, bbox_inches="tight")
#plt.show()
```

[5]:
```
Text(0, 0.5, 'Y_train')
```



1.2.2 Ajuste polinomial

Primero, vamos a proponer un ajuste polinomial de grado 8.

[6]:
```
coef = poly.polyfit(train["X_training"], train["Y_training"],␣
 ↪8,rcond=None,w=None)
```

Veamos los coeficientes del modelo propuesto.

[7]:
```
model = poly.Polynomial(coef)
model
```

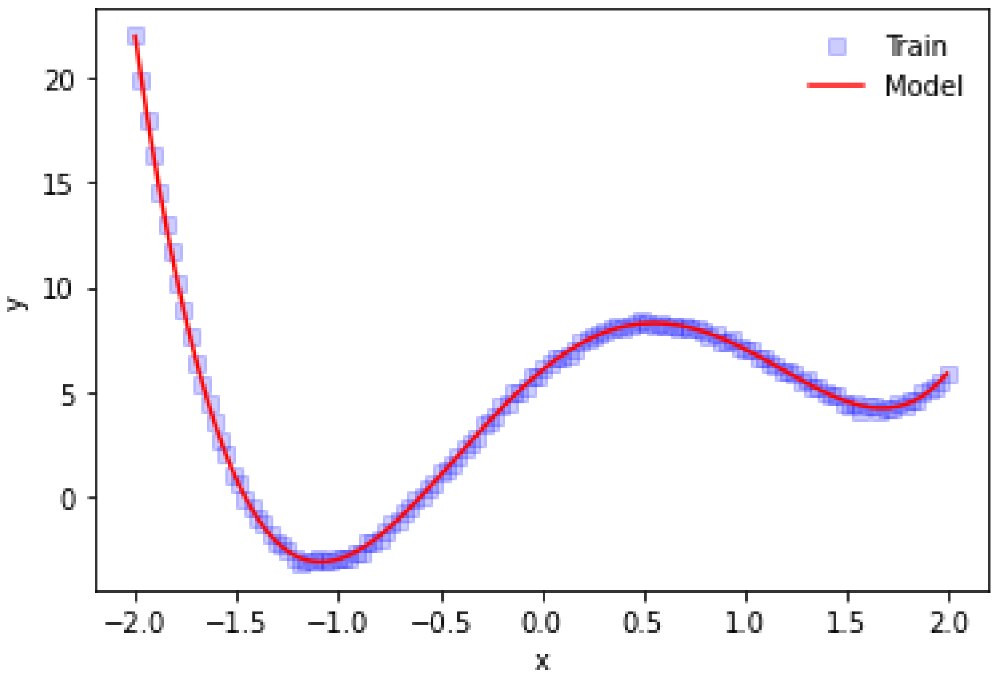
[7]:

$x \mapsto 6.007938949185448 + 8.037968100328241\,x - 6.063208668293313\,x^2 - 3.0250349986092737\,x^3 + 2.0950040571004287\,x^4 + 0.008001051821489162\,x^5 - 0.04294316587789839\,x^6 - 0.001121137939318037\,x^7 + 0.005735129230567836\,x^8$

> Notemos que los coeficientes del polinomio a partir del grado 5 parecen ser bastante pequeños.

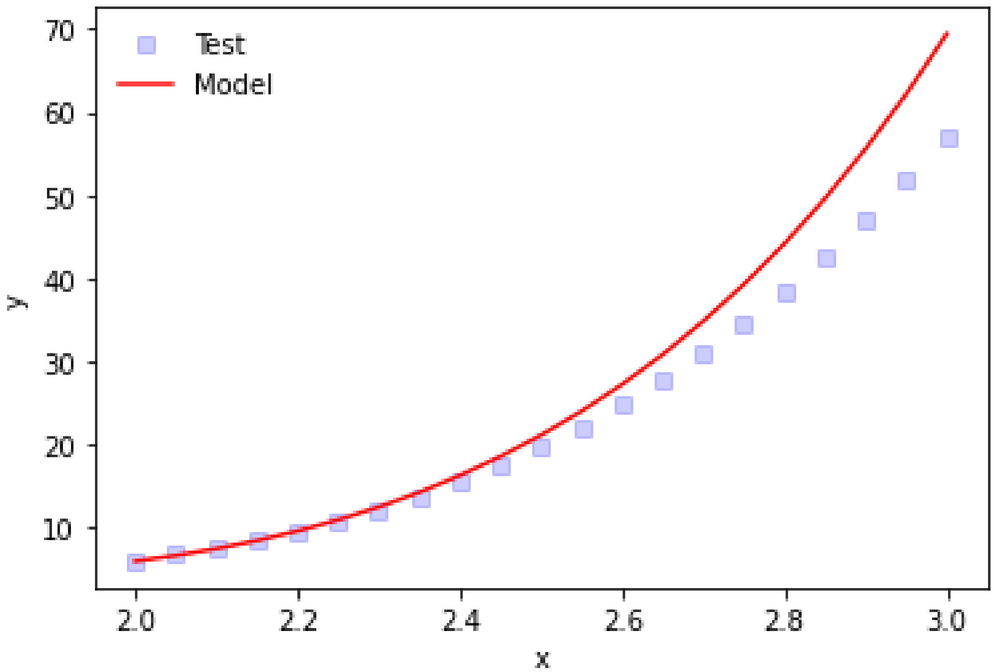
Veamos el ajuste del polinomio y nuestros datos.

[8]:
```
plt.plot(train["X_training"], train["Y_training"],'bs',alpha=0.2, label='Train')
plt.plot(train["X_training"], model(train["X_training"]),'r', label='Model')
plt.legend(loc='best', frameon=False)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Ahora, ajustando este polinomio a los datos de prueba

[9]:
```
plt.plot(test["X_test"], test["Y_test"],'bs',alpha=0.2, label='Test')
plt.plot(test["X_test"], model(test["X_test"]),'r', label='Model')
plt.legend(loc='best', frameon=False)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



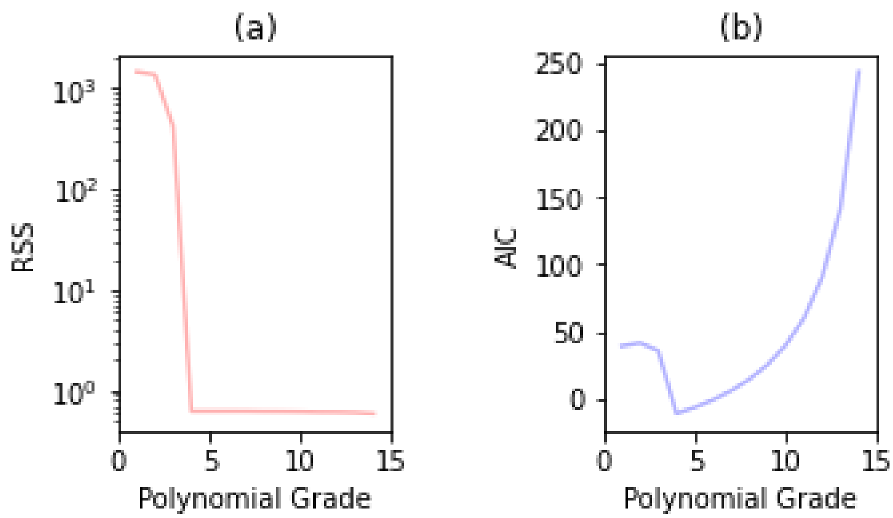
Donde vemos que el ajuste es bueno pero sí presenta errores en los valores alejados de los datos de entrenamiento.

1.2.3 Selección de Modelo

Realizaremos un ajuste de modelo usando los datos de entrenamiento y con base en el AIC y en la suma de los cuadrados de los residuales (RSS). Esto lo haremos asumiendo un grado máximo de polinomio de 15 y mostrando un gráfico para seleccionar aquel modelo que tenga un menor AIC y menor RSS.

[10]:
```
Pol_Max = 15 # Highest degree polynomial we are going to check.
# empty arrays
RSSv = []
RSSv = np.zeros(Pol_Max-1)
AICv = []
AICv = np.zeros(Pol_Max-1)
# Cycle to check different polynomials
x = np.arange(1,10,0.5)
for i in np.arange(0,Pol_Max-1,1):
        coef = poly.polyfit(train["X_training"], train["Y_training"], i+1)
        model = poly.Polynomial(coef)
        RSSv[i]=0.5*np.sum((train["Y_training"]-model(train["X_training"]))**2)
        AICv[i]=len(x)*math.log10(RSSv[i]/len(x))+ 2*len(coef)*len(x)/
 ↪(len(x)-len(coef)-1)
# Plotting RSS and AIC
```

```python
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(np.arange(0,Pol_Max-1,1)+1, RSSv,'r',alpha=0.3)
ax1.set_yscale('log')
ax1.set(xlabel='Polynomial Grade', ylabel='RSS')
ax1.set(xlim=(0, Pol_Max))
ax1.set_title('(a)')
# Second #figure
fig.tight_layout(pad=5.0)
ax2.plot(np.arange(0,Pol_Max-1,1)+1, AICv,'b',alpha=0.3)
ax2.set(xlabel='Polynomial Grade', ylabel='RSS')
ax2.set(xlabel='Polynomial Grade', ylabel='AIC')
ax2.set(xlim=(0, Pol_Max))
ax2.set_title('(b)')
plt.show()
```



Donde podemos observar que el mínimo de los AIC se alcanza cuando el grado es 4 y justo ahí es donde el RSS parece estabilizarse. Algo que coincide con lo que mencionamos en el primer ajuste de modelo.

[11]:
```python
grade = np.argmin(AICv)+1
grade
```

[11]: 4

Donde ahora continuamos haciendo un ajuste de modelo con este grado

[12]:
```python
coef = poly.polyfit(train["X_training"], train["Y_training"], grade)
```

Veamos los coeficientes del modelo propuesto.

[13]:
```python
model = poly.Polynomial(coef)
model
```

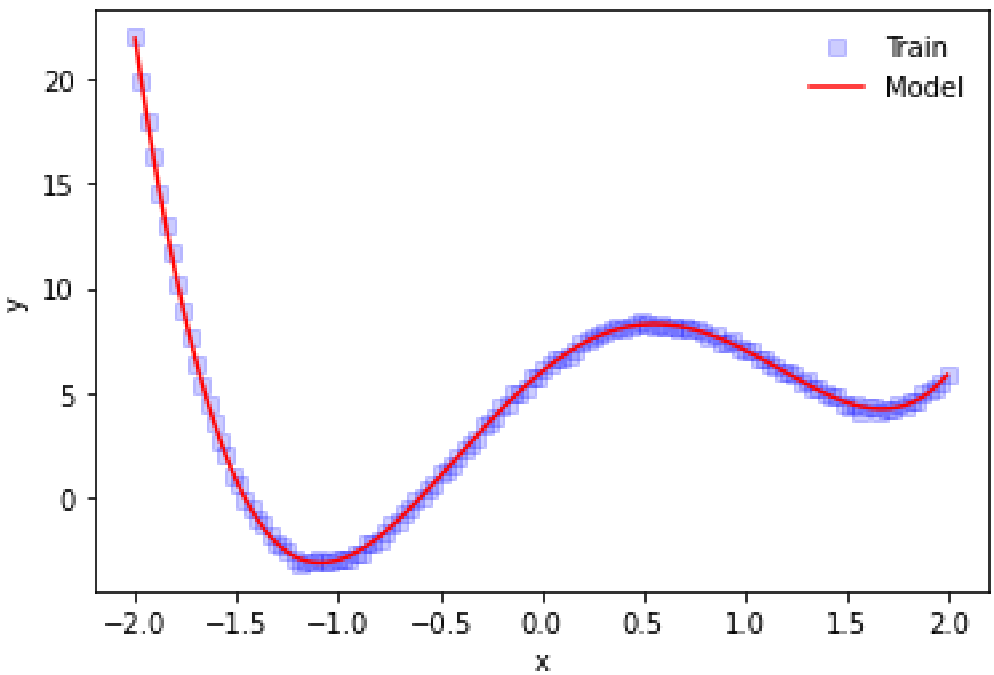
[13]:

$x \mapsto 6.000059202668212 + 8.029567412979475\,x - 5.991414146668381\,x^2 - 3.0086947131582313\,x^3 + 1.9958810906630031\,x^4$

Notemos que los coeficientes del polinomio ya lucen todos diferentes de cero.

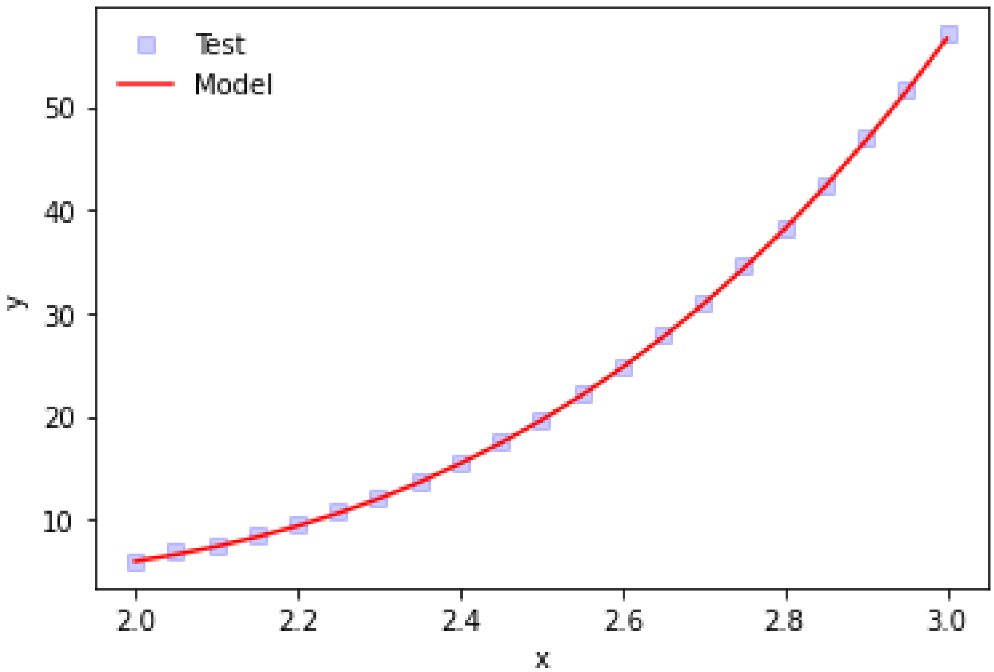
Veamos el ajuste del polinomio y nuestros datos.

[14]:
```python
plt.plot(train["X_training"], train["Y_training"],'bs',alpha=0.2, label='Train')
plt.plot(train["X_training"], model(train["X_training"]),'r', label='Model')
plt.legend(loc='best', frameon=False)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



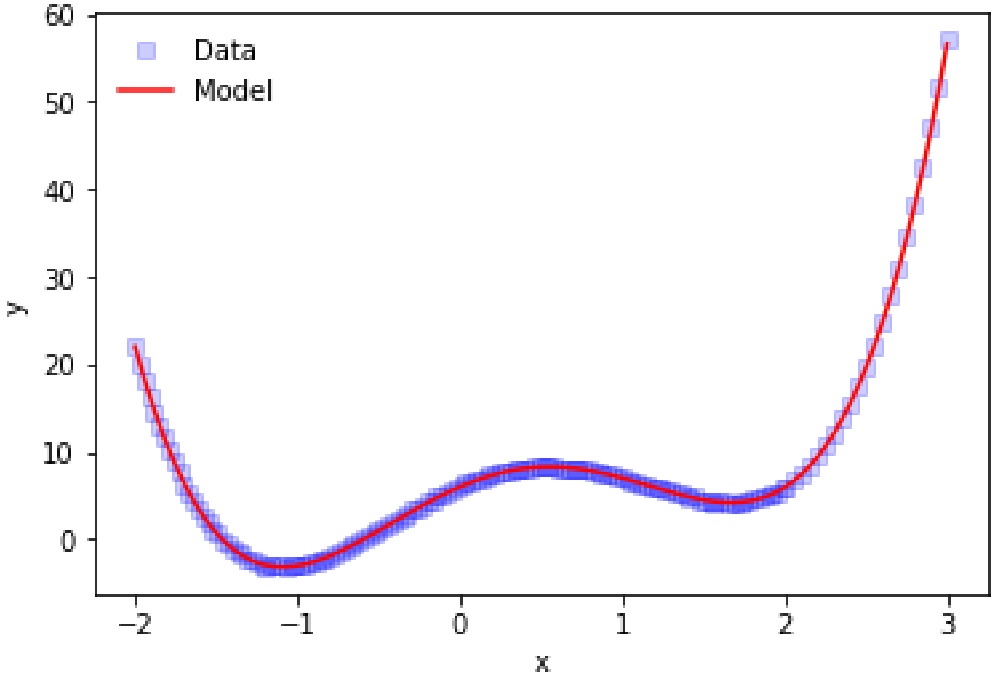
Ahora, ajustando este polinomio a los datos de prueba

[15]:
```python
plt.plot(test["X_test"], test["Y_test"],'bs',alpha=0.2, label='Test')
plt.plot(test["X_test"], model(test["X_test"]),'r', label='Model')
plt.legend(loc='best', frameon=False)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Donde vemos que el ajuste es muy superior que el visto anteriormente y además sí predice los valores alejados de los datos de entrenamiento. Finalmente, veamos la figura completa de los datos que tenemos y su ajuste.

[16]:
```
x = np.concatenate((train["X_training"], test["X_test"]))
y = np.concatenate((train["Y_training"], test["Y_test"]))
# Gráfico
plt.plot(x, y,'bs',alpha=0.2, label='Data')
plt.plot(x, model(x),'r', label='Model')
plt.legend(loc='best', frameon=False)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



1.2.4 Validación Cruzada

Vamos a evaluar la calificación de validación cruzada de nuestro modelo

[17]:
```
# C-V para el modelo que tenemos.
test = test.dropna()
poly_features = PolynomialFeatures(degree=grade)
X_poly = poly_features.fit_transform(test)
poly = LinearRegression()
np.mean(cross_val_score(poly, X_poly, test["Y_test"], cv=5))
```

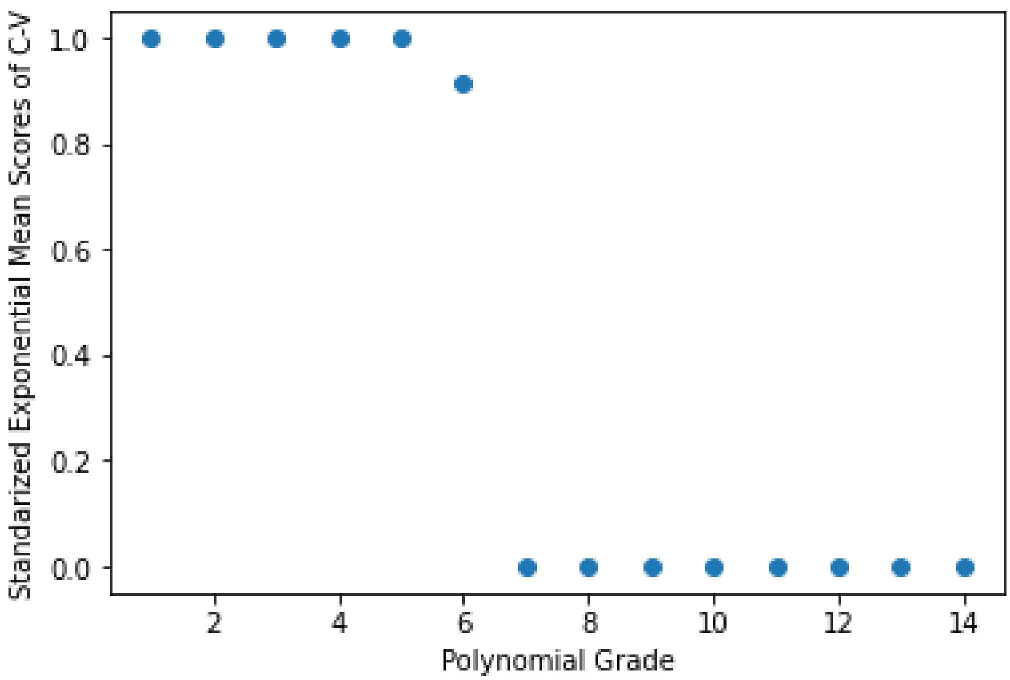
[17]:
```
1.0
```

Esto nos indica que el modelo es muy bueno. Finalmente comparémoslo contra otros grados del polinomio

[18]:
```
def create_polynomial_regression_model(degree):
 poly_features = PolynomialFeatures(degree=degree)
 X_poly = poly_features.fit_transform(test)
 poly = LinearRegression()
 return np.exp(np.mean(cross_val_score(poly, X_poly, test["Y_test"], cv=5)))/np.
 ↪exp(1)
poly_cv = []
for i in range(1,15):
 poly_cv.append(create_polynomial_regression_model(i))
```

```
plt.scatter(range(1,15),poly_cv)
plt.xlabel('Polynomial Grade')
plt.ylabel('Standarized Exponential Mean Scores of C-V')
```

[18]: Text(0, 0.5, 'Standarized Exponential Mean Scores of C-V')



Donde vemos que justo el grado elegido es adecuado.

1.3 Ejercicio 2

Tener la siguiente tabla lógica

x_1	x_2	$x_1 \& x_2$
0	0	1
1	0	1
0	1	1
1	1	0

x_1	x_2	$x_1 \$ x_2$
0	0	0
1	0	0
0	1	0
1	1	1

Usando Keras, construye un operador & y $ con un perceptron simple. Si A=[1.001 0 0.001 1], B=[0 1 0 1] y C=[0 1 1 0]. Calcula la operación (A&B)$C

1.3.1 Solución

1.3.2 Operador &

[19]:
```python
# The input and output, i.e. truth table, of a NAND gate
x_train = np.array([[0,0],[1,0],[0,1],[1,1]], "uint8")
y_train = np.array([[1],[1],[1],[0]], "uint8")

# Create neural networks model
model_Ampersand = Sequential()
# Add layers to the model
model_Ampersand.add(Dense(5, activation='relu', input_dim=2))      # first␣
 ↪hidden layer
#model_Ampersand.add(Dense(3, activation='relu'))                    # second␣
 ↪hidden layer
model_Ampersand.add(Dense(1, activation='sigmoid'))                 # output␣
 ↪layer

# Compile the neural networks model
model_Ampersand.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
# Train the neural networks model
model_Ampersand.fit(x_train, y_train, epochs=5000,verbose=0)
```

[19]:
```
<tensorflow.python.keras.callbacks.History at 0x1889ad645e0>
```

[20]:
```python
# Test the output of the trained neural networks based NAND gate
y_predict = model_Ampersand.predict(x_train)
print(y_predict)
```

```
[[0.99889034]
 [0.99345964]
 [0.9940161 ]
 [0.06992993]]
```

1.3.3 Operador $

[21]:
```python
# The input and output, i.e. truth table, of a NAND gate
x_train = np.array([[0,0],[1,0],[0,1],[1,1]], "uint8")
y_train = np.array([[0],[0],[0],[1]], "uint8")

# Create neural networks model
model_Dollar = Sequential()
# Add layers to the model
```

```
model_Dollar.add(Dense(5, activation='relu', input_dim=2))       # first hidden␣
 ↪layer
#model_Ampersand.add(Dense(3, activation='relu'))                     # second␣
 ↪hidden layer
model_Dollar.add(Dense(1, activation='sigmoid'))                 # output layer

# Compile the neural networks model
model_Dollar.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
# Train the neural networks model
model_Dollar.fit(x_train, y_train, epochs=5000,verbose=0)
```

[21]: <tensorflow.python.keras.callbacks.History at 0x1889b0181f0>

[22]:
```
# Test the output of the trained neural networks based NAND gate
y_predict = model_Dollar.predict(x_train)
print(y_predict)
```

```
[[5.0193071e-04]
 [2.7458072e-03]
 [2.6729107e-03]
 [9.9726784e-01]]
```

1.3.4 Componiendo los operadores

$A\&B$

[23]:
```
A = np.array([1.001, 0, 0.001, 1])
B = np.array([0, 1, 0, 1])
AB = np.dstack((A,B))
A_Amp_B = model_Ampersand.predict(AB)
print(A_Amp_B)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 2) for input
Tensor("dense_input:0", shape=(None, 2), dtype=float32), but it was called on an
input with incompatible shape (None, 4, 2).
[[[0.99347866]
  [0.9940161 ]
  [0.9988851 ]
  [0.06992993]]]
```

$(A\&B)\$C$

[24]:
```
C = np.array([0,1,1,0])
AAmpBC = np.dstack((A_Amp_B,C))
Result = model_Dollar.predict(AAmpBC)
print(Result)
```

```
WARNING:tensorflow:Model was constructed with shape (None, 2) for input
Tensor("dense_2_input:0", shape=(None, 2), dtype=float32), but it was called on
```

```
an input with incompatible shape (None, 4, 2).
[[[2.7094483e-03]
  [9.9706805e-01]
  [9.9723172e-01]
  [5.6514144e-04]]]
```

1.4 Ejercicio 3

De un ensayo clínico, tenemos 12 pacientes con infección por VIH. Después del tratamiento, la enfermedad progresó en 6 pacientes (1) y en 6 pacientes la infección no progresó (0). Cuatro medidas se toman en los 12 pacientes (edad, niveles de azúcar, niveles de células T y colesterol).

¿Qué medida se puede utilizar como marcador para describir la progresión de la enfermedad?

¿Cuáles serán los criterios a predecir la progresión? Los datos se pueden encontrar en“problema3.csv (x_age, x_sugar, x_Tcell, x_cholesterol, Salir)”?.

Ordene los datos y explique brevemente sus resultados. La variable “y” (objetivo) es un vector de 0 y 1 para representar la progresión.

1.4.1 Solución

[25]:
```
# Letura de datos
datos = pd.read_csv('problem3.csv')
datos.head()
```

[25]:

	x_age	x_cholesterol	x_sugar	x_Tcell	y
0	35	220	80	550	0
1	18	240	120	600	0
2	22	260	55	580	0
3	23	220	75	575	0
4	28	180	100	620	0

1.4.2 Probando la variable ‘age’

Definimos el modelo logístico

[26]:
```
model = LogisticRegression(C=1.0, solver='lbfgs', multi_class='ovr')
y_resp = datos["y"]
x_expl = datos[["x_age"]]
```

Separamos en covariable y respuesta pero de un subconjunto de datos

[27]:
```
# Separar datos
x_train, x_test, y_train, y_test = train_test_split(x_expl, y_resp, test_size=0.
↪3, random_state=5)
```

[28]:
```
model.fit(x_train, y_train)
model.score(x_train, y_train)
```

[28]: 0.5

Mostramos los coeficientes del modelo

[29]:
```
model.coef_
```

[29]: array([[0.07509664]])

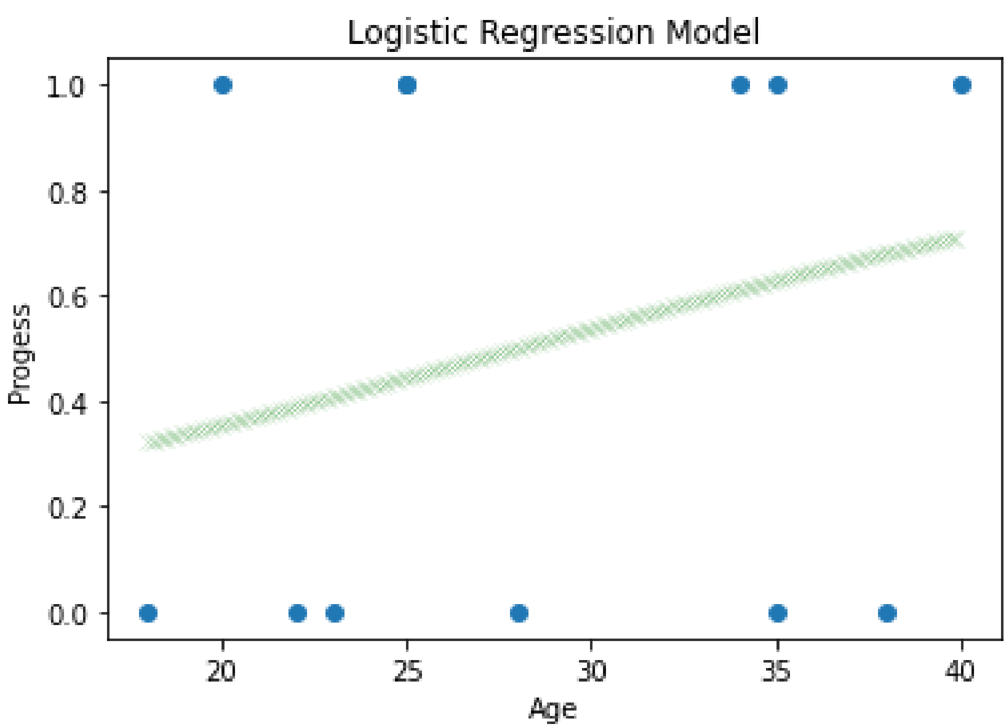
[30]:
```
model.intercept_
```

[30]: array([-2.09950044])

Graficando el Modelo

[31]:
```
X = np.arange(18, 40, 0.1)
X = X.reshape(-1, 1)
plt.scatter(x_expl, y_resp)
plt.scatter(X,model.predict_proba(X)[:,1],marker='x',color='g',linewidth=.1)
plt.title("Logistic Regression Model")
plt.xlabel('Age')
plt.ylabel('Progess')
```

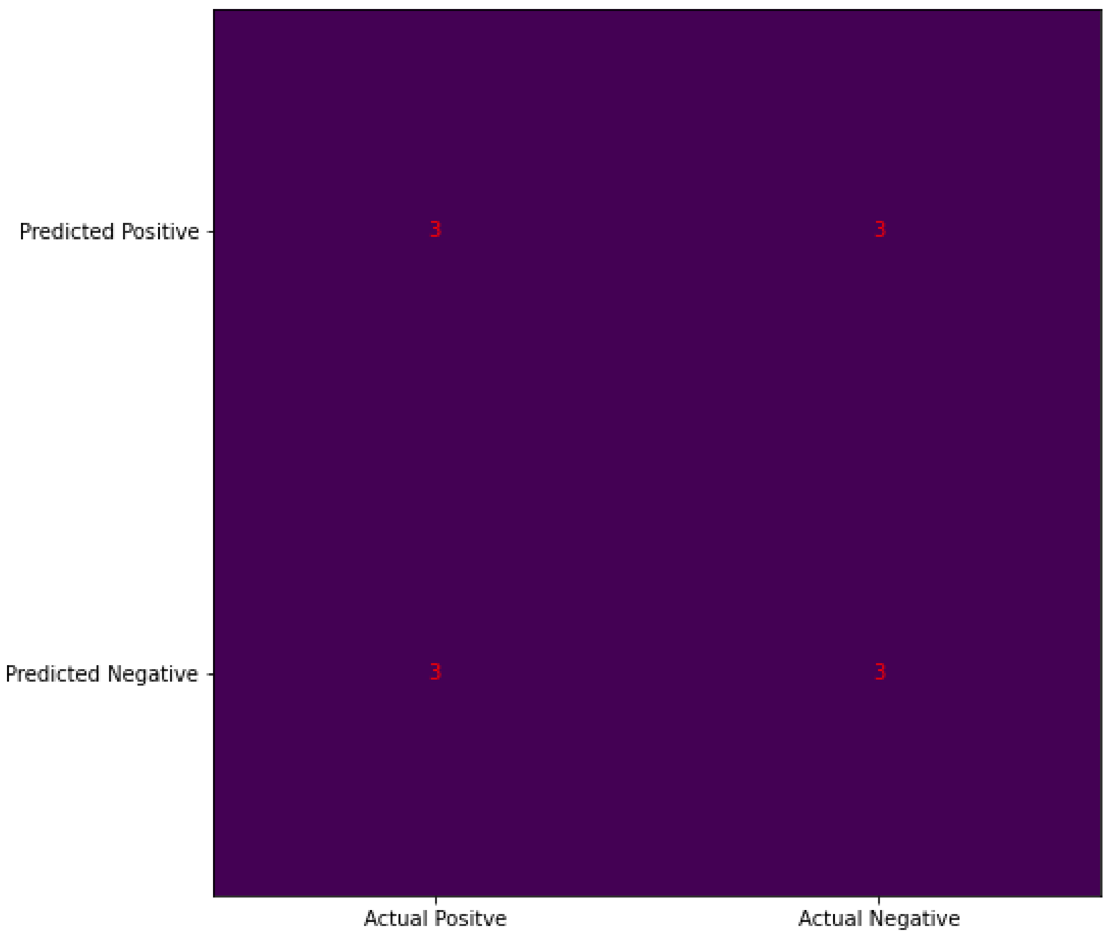
[31]: Text(0, 0.5, 'Progess')



Matriz de Confusión

[32]:
```
cm = confusion_matrix(y_resp, model.predict(x_expl))
```

[33]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.yaxis.set(ticks=(0, 1), ticklabels=('Predicted Positive', 'Predicted␣
↪Negative'))
ax.xaxis.set(ticks=(0, 1), ticklabels=('Actual Positve', 'Actual Negative'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



ROC

[34]:
```
# Genrate a Diagonal(Random Guess)
ns_probs = [0 for _ in range(len(y_resp))]
```

```
# predict probabilities
lr_probs = model.predict_proba(x_expl)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_resp, ns_probs)
lr_auc = roc_auc_score(y_resp, lr_probs)
```

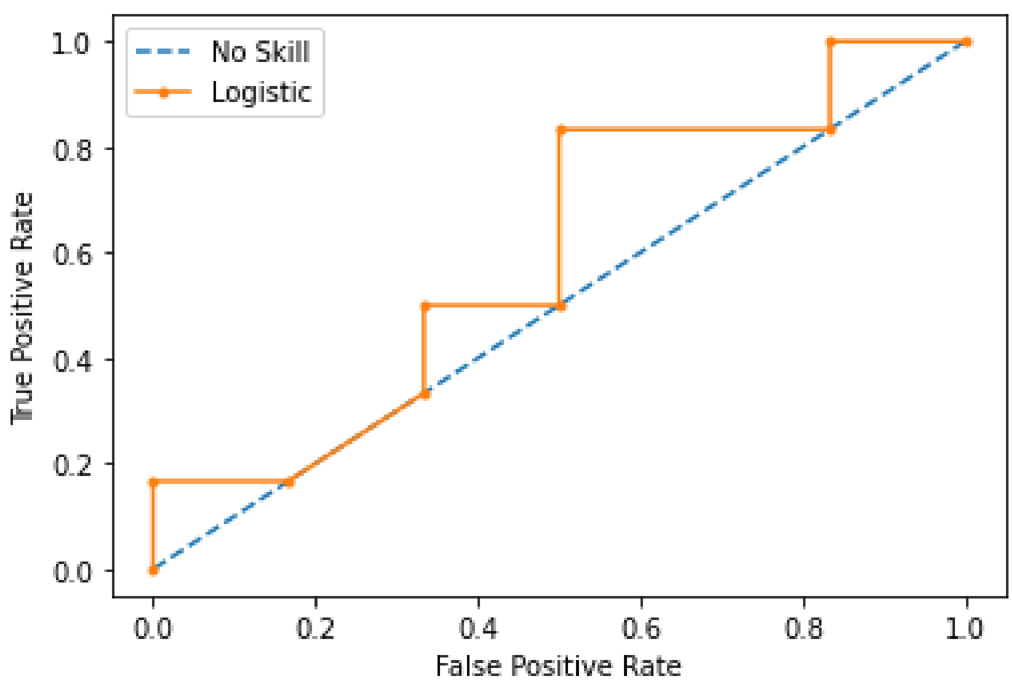
[35]:

```
# summarize scores
print('ROC AUC for Logistic Model =%.3f' % (lr_auc))
```

```
ROC AUC for Logistic Model =0.597
```

[36]:

```
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_resp, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_resp, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



Esta variable no nos parece ayudar pues parece que predice prácticamente la certeza de un volado.

1.4.3 Probando la variable 'cholesterol'

Definimos el modelo logístico

[37]:
```
model = LogisticRegression(C=1.0, solver='lbfgs', multi_class='ovr')
y_resp = datos["y"]
x_expl = datos[["x_cholesterol"]]
```

Separamos en covariable y respuesta pero de un subconjunto de datos

[38]:
```
# Separar datos
x_train, x_test, y_train, y_test = train_test_split(x_expl, y_resp, test_size=0.
↪3, random_state=5)
```

[39]:
```
model.fit(x_train, y_train)
model.score(x_train, y_train)
```

[39]: 0.625

Mostramos los coeficientes del modelo

[40]:
```
model.coef_
```

[40]: array([[-0.00722093]])

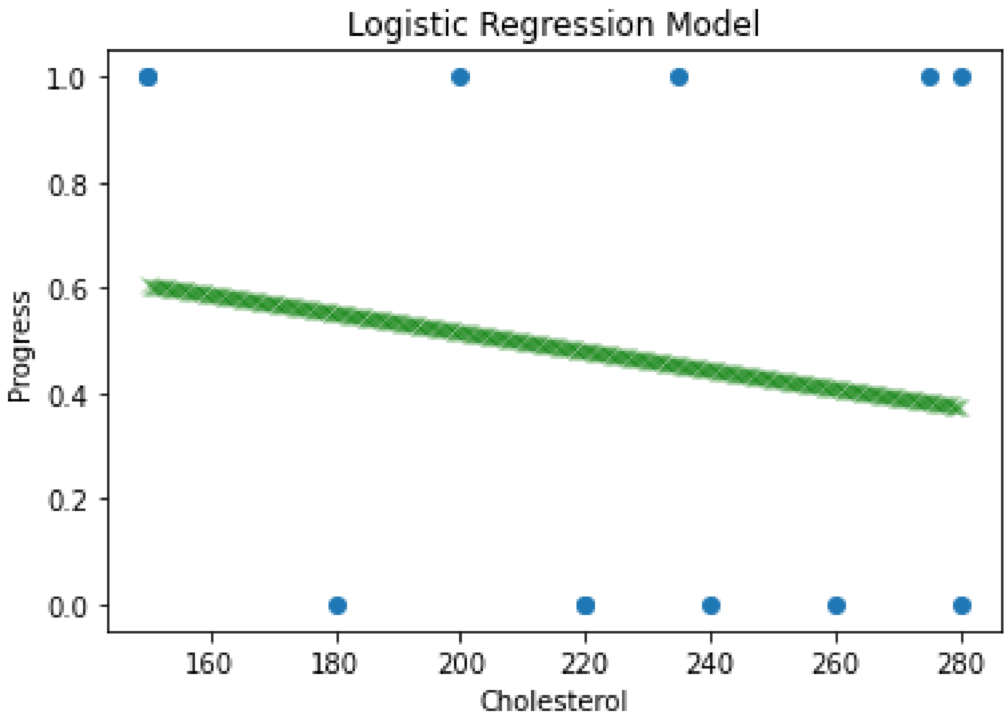
[41]:
```
 model.intercept_
```

[41]: array([1.50771544])

Graficando el Modelo

[42]:
```
X = np.arange(150, 280, 0.1)
X = X.reshape(-1, 1)
plt.scatter(x_expl, y_resp)
plt.scatter(X,model.predict_proba(X)[:,1],marker='x',color='g',linewidth=.1)
plt.title("Logistic Regression Model")
plt.xlabel('Cholesterol')
plt.ylabel('Progress')
```

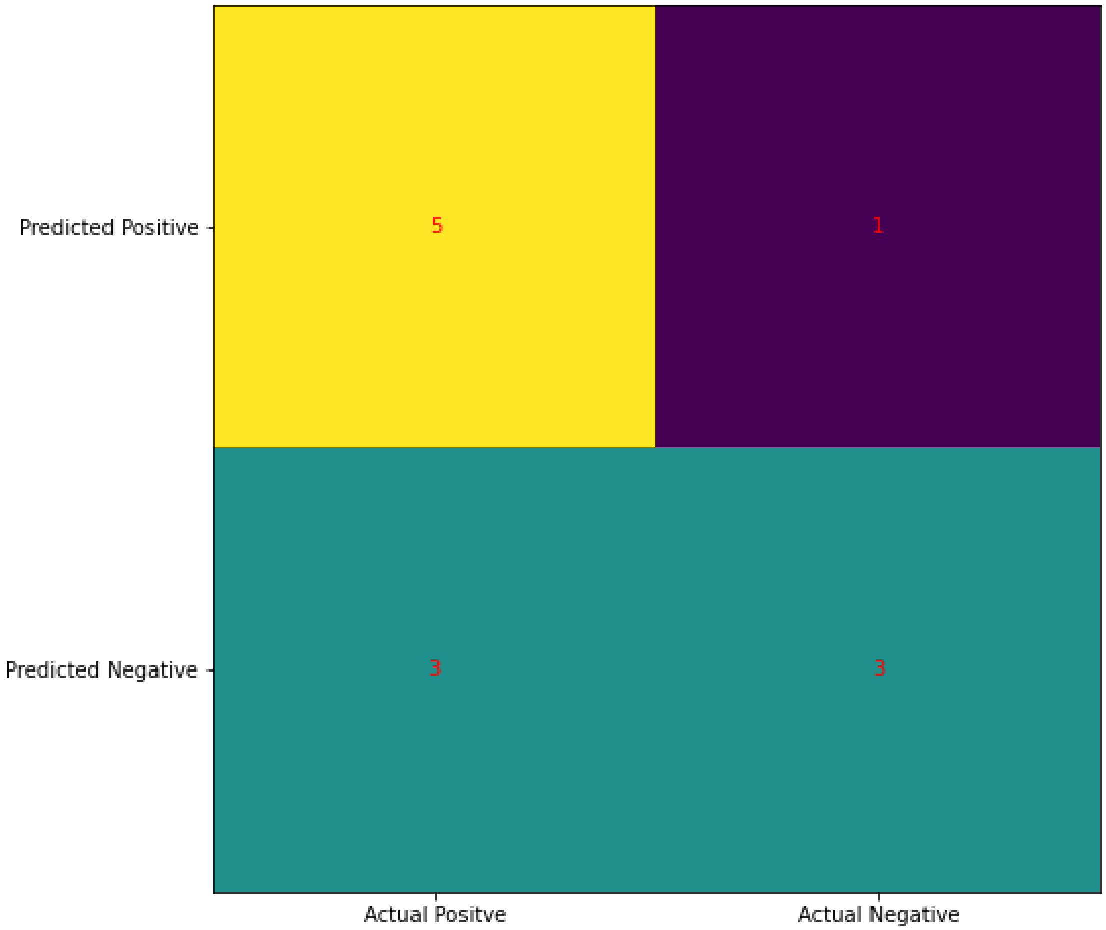
[42]: Text(0, 0.5, 'Progress')



Matriz de Confusión

[43]:
```
cm = confusion_matrix(y_resp, model.predict(x_expl))
```

[44]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.yaxis.set(ticks=(0, 1), ticklabels=('Predicted Positive', 'Predicted␣
 ↪Negative'))
ax.xaxis.set(ticks=(0, 1), ticklabels=('Actual Positve', 'Actual Negative'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



ROC

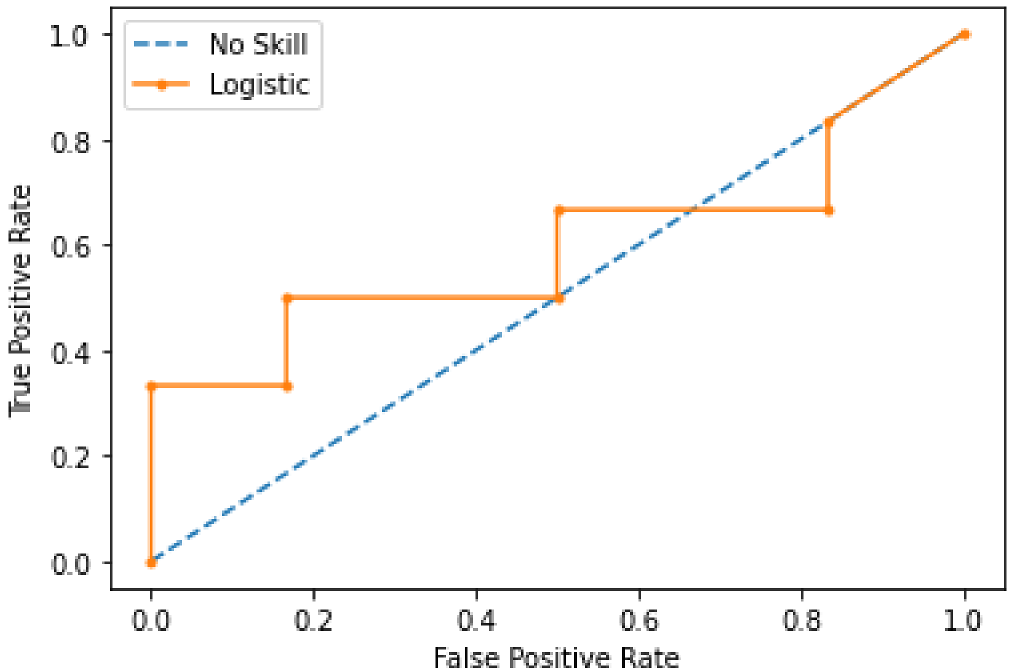
[45]:
```
# Genrate a Diagonal(Random Guess)
ns_probs = [0 for _ in range(len(y_resp))]
# predict probabilities
lr_probs = model.predict_proba(x_expl)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_resp, ns_probs)
lr_auc = roc_auc_score(y_resp, lr_probs)
```

[46]:
```
# summarize scores
print('ROC AUC for Logistic Model =%.3f' % (lr_auc))
```

```
ROC AUC for Logistic Model =0.597
```

[47]:
```
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_resp, ns_probs)
```

```
lr_fpr, lr_tpr, _ = roc_curve(y_resp, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



Esta variable mejora con respecto a la anterior, pero aún su poder de predicción no parece bueno.

1.4.4 Probando la variable 'sugar'

Definimos el modelo logístico

[48]:
```
model = LogisticRegression(C=1.0, solver='lbfgs', multi_class='ovr')
y_resp = datos["y"]
x_expl = datos[["x_sugar"]]
```

Separamos en covariable y respuesta pero de un subconjunto de datos

[49]:
```
# Separar datos
```

```
x_train, x_test, y_train, y_test = train_test_split(x_expl, y_resp, test_size=0.
 ↪3, random_state=5)
```

[50]:
```
model.fit(x_train, y_train)
model.score(x_train, y_train)
```

[50]: 1.0

Mostramos los coeficientes del modelo

[51]:
```
model.coef_
```

[51]: array([[0.7157536]])

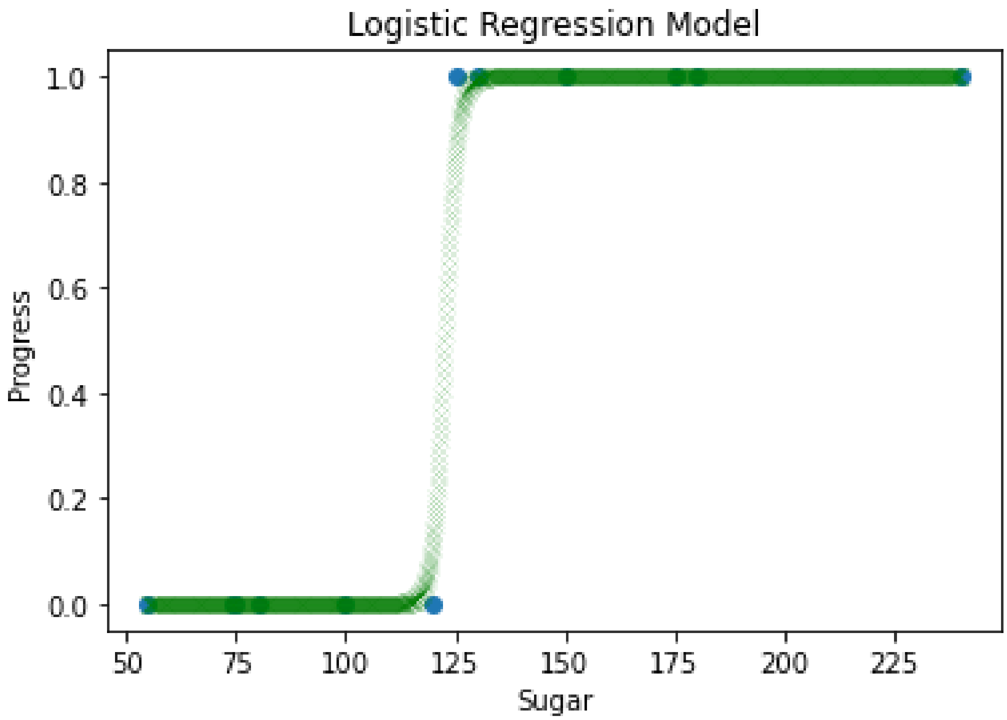
[52]:
```
 model.intercept_
```

[52]: array([-87.67981786])

Graficando el Modelo

[53]:
```
X = np.arange(55, 240, 0.1)
X = X.reshape(-1, 1)
plt.scatter(x_expl, y_resp)
plt.scatter(X,model.predict_proba(X)[:,1],marker='x',color='g',linewidth=.1)
plt.title("Logistic Regression Model")
plt.xlabel('Sugar')
plt.ylabel('Progress')
```

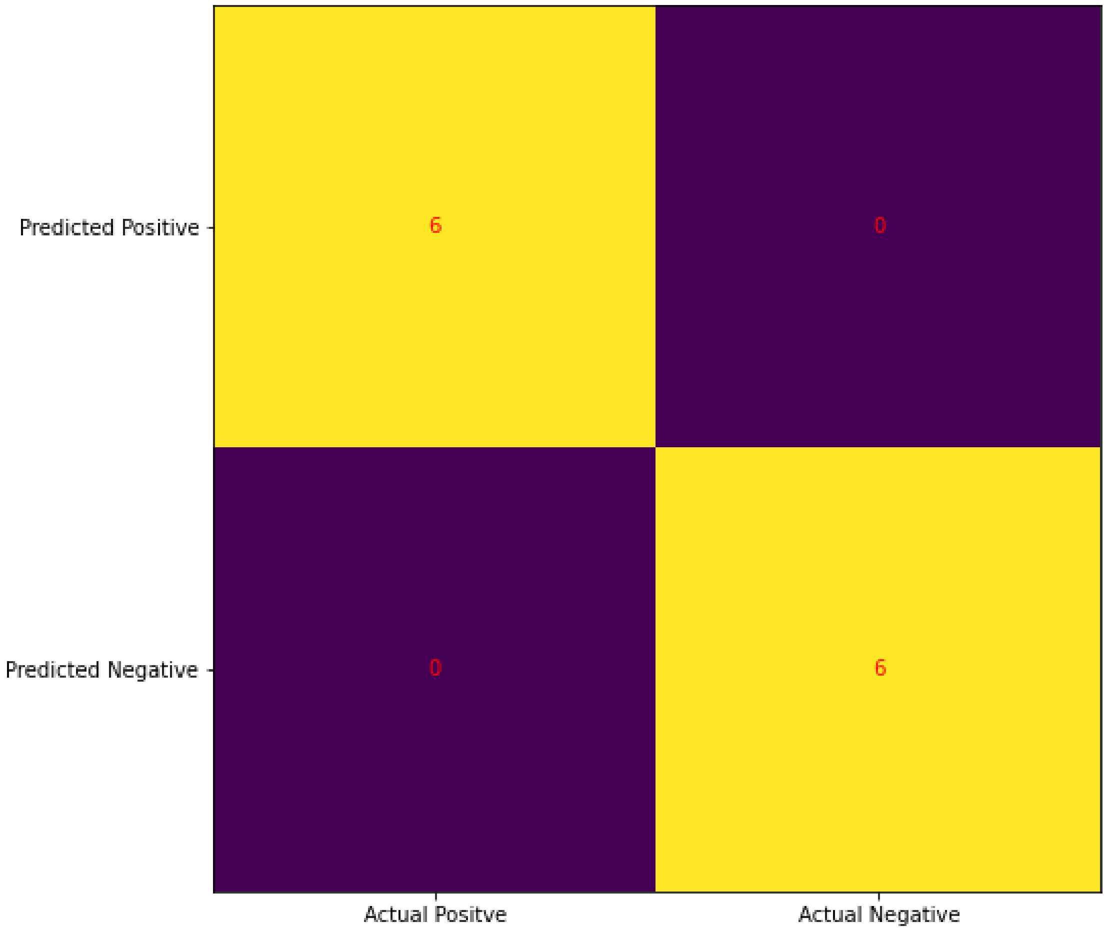
[53]: Text(0, 0.5, 'Progress')



Matriz de Confusión

[54]:
```
cm = confusion_matrix(y_resp, model.predict(x_expl))
```

[55]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.yaxis.set(ticks=(0, 1), ticklabels=('Predicted Positive', 'Predicted␣
↪Negative'))
ax.xaxis.set(ticks=(0, 1), ticklabels=('Actual Positve', 'Actual Negative'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



ROC

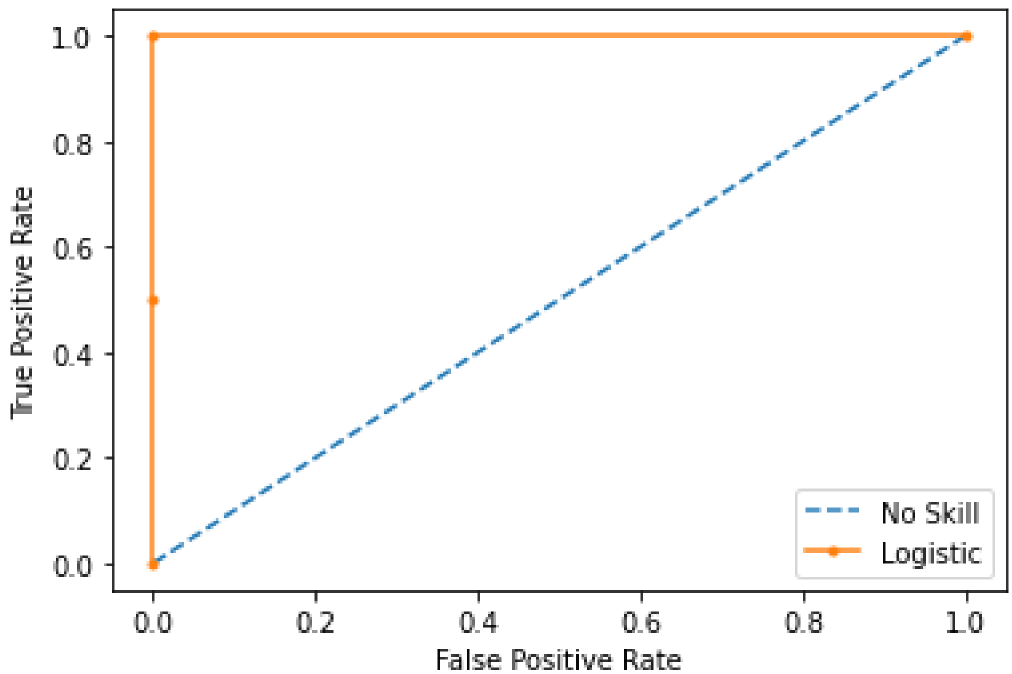
[56]:
```python
# Genrate a Diagonal(Random Guess)
ns_probs = [0 for _ in range(len(y_resp))]
# predict probabilities
lr_probs = model.predict_proba(x_expl)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_resp, ns_probs)
lr_auc = roc_auc_score(y_resp, lr_probs)
```

[57]:
```python
# summarize scores
print('ROC AUC for Logistic Model =%.3f' % (lr_auc))
```

```
ROC AUC for Logistic Model =1.000
```

[58]:
```python
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_resp, ns_probs)
```

```
lr_fpr, lr_tpr, _ = roc_curve(y_resp, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



Esta variable predice demasiado bien, de hecho ajusta perfectamente a los datos que nosotros tenemos.

1.4.5 Probando la variable 'Tcell'

Definimos el modelo logístico

[59]:
```
model = LogisticRegression(C=1.0, solver='lbfgs', multi_class='ovr')
y_resp = datos["y"]
x_expl = datos[["x_Tcell"]]
```

Separamos en covariable y respuesta pero de un subconjunto de datos

[60]:
```
# Separar datos
x_train, x_test, y_train, y_test = train_test_split(x_expl, y_resp, test_size=0.
 ↪3, random_state=5)
```

[61]:
```
model.fit(x_train, y_train)
model.score(x_train, y_train)
```

[61]:
```
1.0
```

Mostramos los coeficientes del modelo

[62]:
```
model.coef_
```

[62]:
```
array([[-0.07262875]])
```

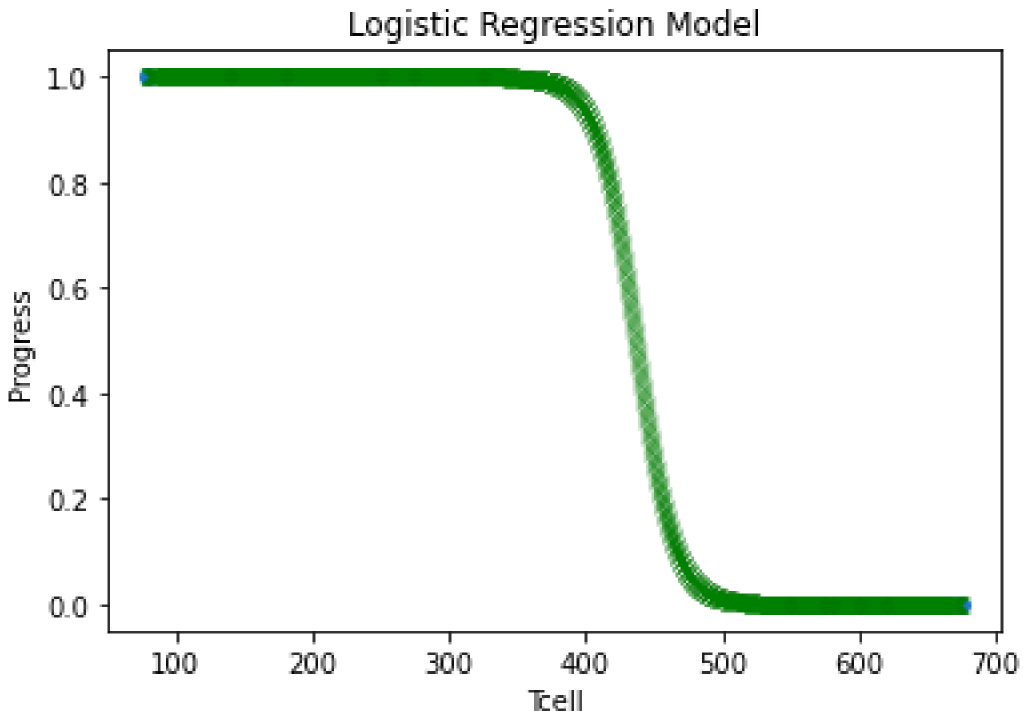
[63]:
```
 model.intercept_
```

[63]:
```
array([31.70077079])
```

Graficando el Modelo

[64]:
```
X = np.arange(80, 674, 0.1)
X = X.reshape(-1, 1)
plt.scatter(x_expl, y_resp)
plt.scatter(X,model.predict_proba(X)[:,1],marker='x',color='g',linewidth=.1)
plt.title("Logistic Regression Model")
plt.xlabel('Tcell')
plt.ylabel('Progress')
```

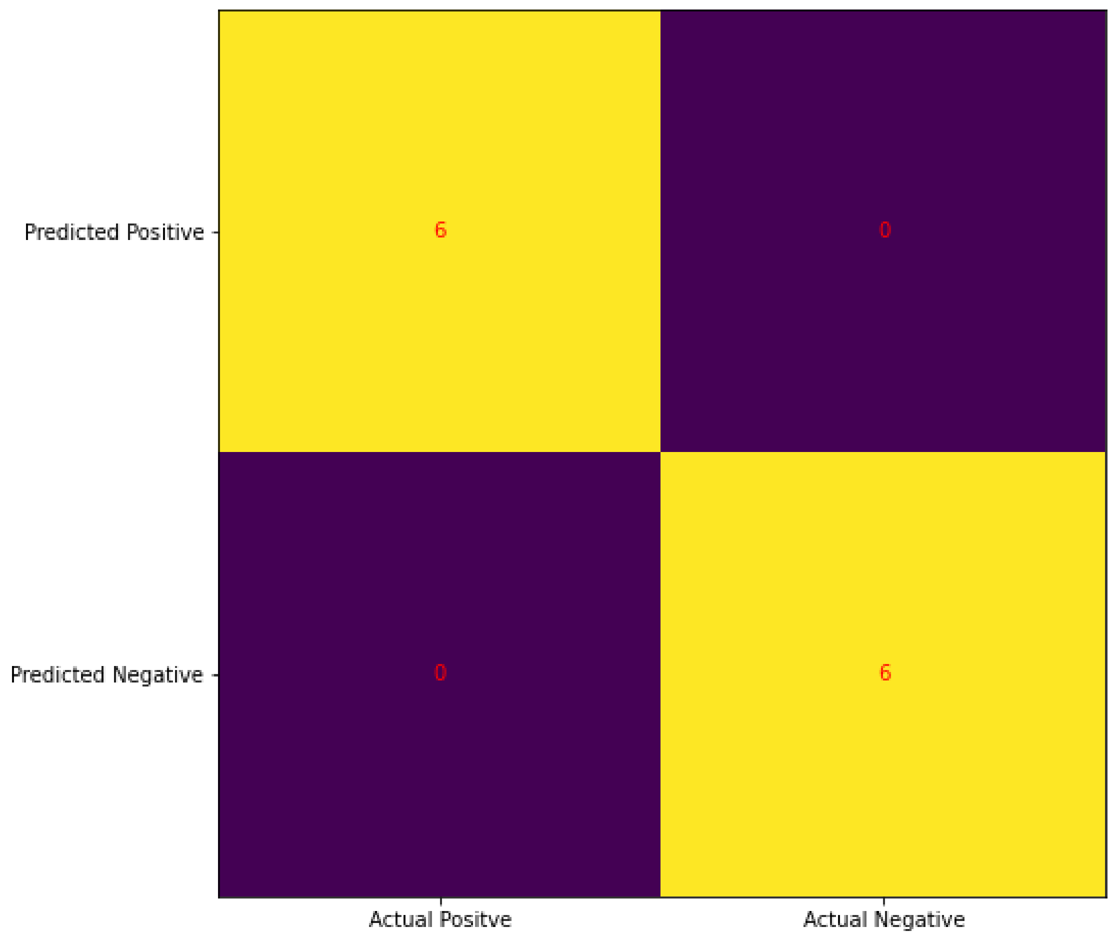
[64]:
```
Text(0, 0.5, 'Progress')
```



Matriz de Confusión

[65]:
```
cm = confusion_matrix(y_resp, model.predict(x_expl))
```

[66]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.yaxis.set(ticks=(0, 1), ticklabels=('Predicted Positive', 'Predicted␣
↪Negative'))
ax.xaxis.set(ticks=(0, 1), ticklabels=('Actual Positve', 'Actual Negative'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



ROC

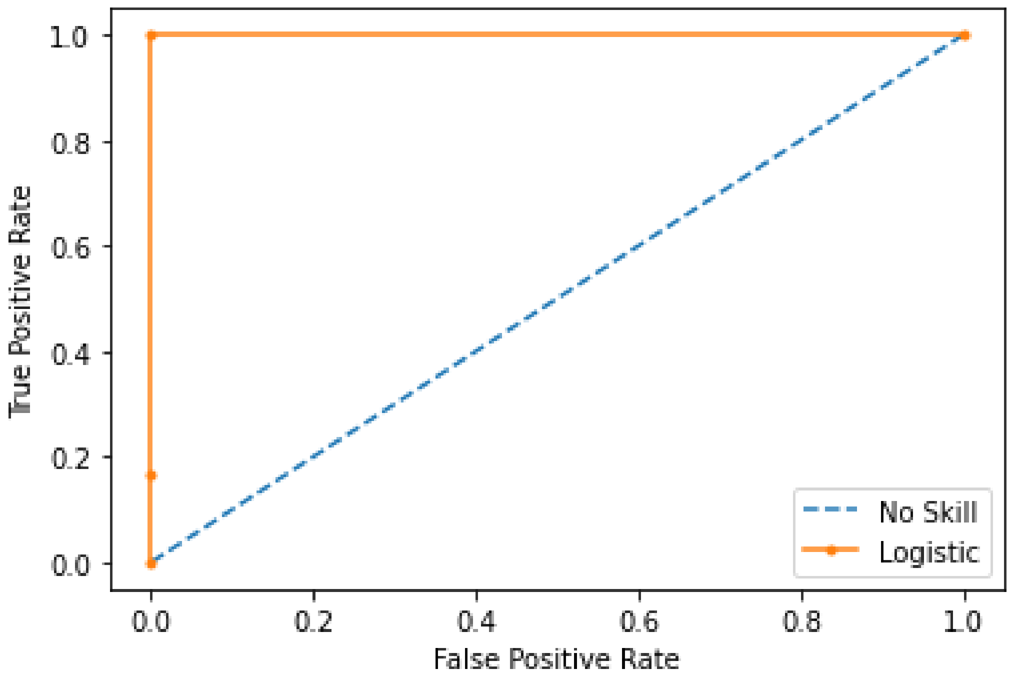
[67]:
```
# Genrate a Diagonal(Random Guess)
ns_probs = [0 for _ in range(len(y_resp))]
# predict probabilities
lr_probs = model.predict_proba(x_expl)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_resp, ns_probs)
lr_auc = roc_auc_score(y_resp, lr_probs)
```

[68]:
```
# summarize scores
print('ROC AUC for Logistic Model =%.3f' % (lr_auc))
```

```
ROC AUC for Logistic Model =1.000
```

[69]:
```
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_resp, ns_probs)
```

```python
lr_fpr, lr_tpr, _ = roc_curve(y_resp, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



Esta variable al igual que la anterior ajusta muy bien a los datos. Con base en estas dos podemos darnos una buena idea de propuesta de modelo.

1.4.6 Conclusión

Lo que podemos proponer para lograr predecir la variable respuesta es usar las variables `sugar` y `Tcell` con el criterio de tomar como 1 la variable respuesta `y` si `sugar` $>=175$ o bien `Tcell` $<=325$. Esta es la información inmediata que puede proporcionarnos nuestros datos.

1.5 Ejercicio 4

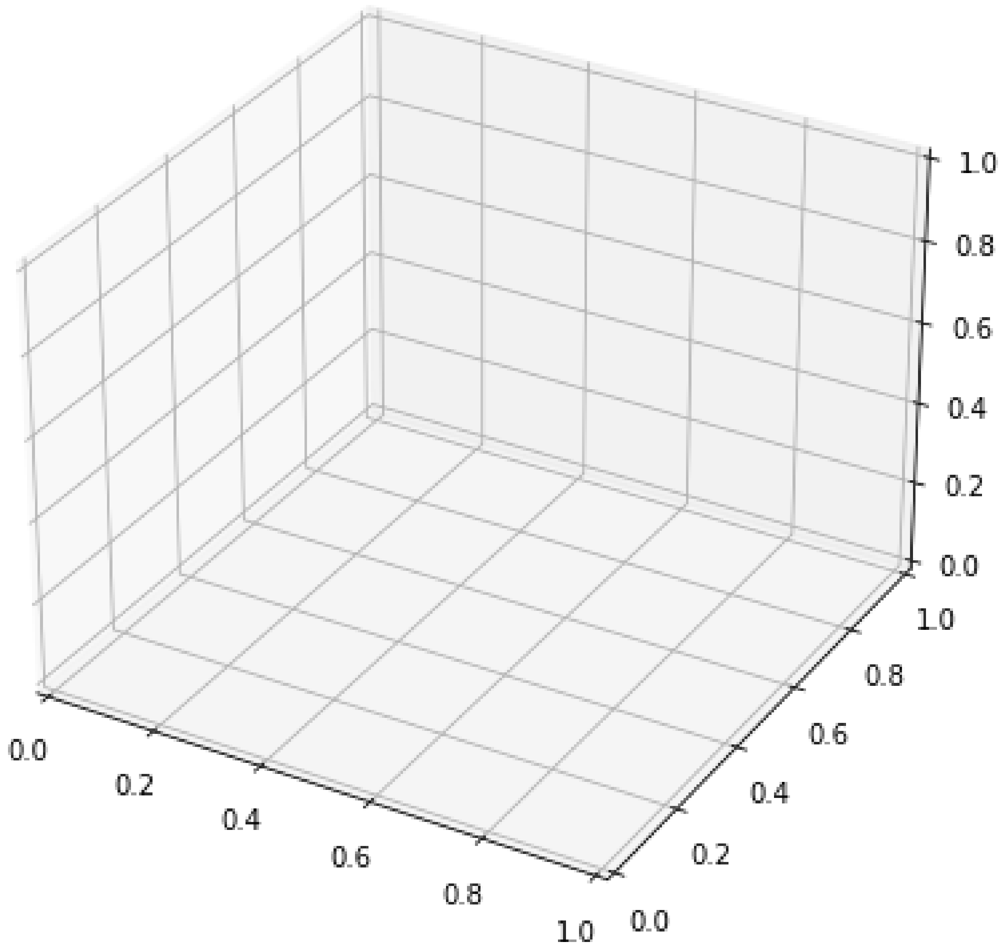
Usando un perceptron multicapa con Keras, produ ce un cono de la 'dimensión' de tu elección. Nota: En otras palabras, usa la ecuación de un cono y luego genera datos artificiales para generar X, Y y Z. Luego, usa los datos X, Y y Z para entrenar una red neuronal, y crea la forma de un cono pero producida por la red neuronal. En tu reporte debes mencionar la ecuación del cono que seleccionaste, y la figura creada por la red neuronal.

1.5.1 Solución

Ecuación del cono que trabajaremos es

$$z = \sqrt{x^2 + y^2}$$

[70]:
```
random.seed(10)
a = np.random.uniform(-1,1,1000)
b = np.random.uniform(-1,1,1000)
z = math.sqrt(2)-np.sqrt(a**2+b**2)

#from mpl_toolkits import mplot3d
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")

# Creating plot
#ax.scatter3D(a, b, z, color = "green")
#plt.title("simple 3D scatter plot")

# show plot
#plt.show()




fig = go.Figure(data=[go.Scatter3d(
    x=a,
    y=b,
    z=z,
    mode='markers',
    marker=dict(
        size=5,
        color=z,                # set color to an array/list of desired values
        colorscale='Viridis',   # choose a colorscale
        opacity=0.8
    )
)])


# tight layout
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0))
```



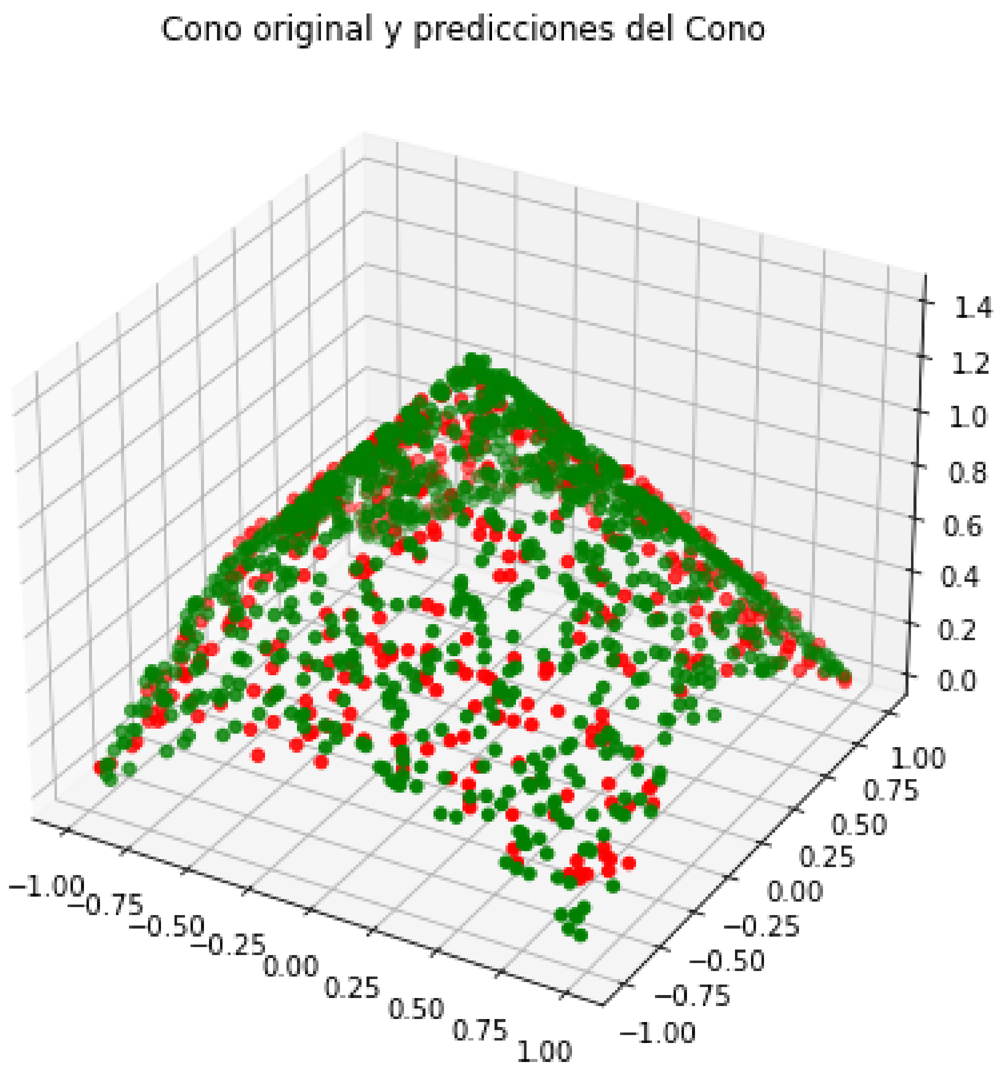
[71]:
```
# The input and output, i.e. truth table, of a NAND gate
x_train = np.array(np.dstack((a,b)))
x_train = np.reshape(x_train, (1000, 2))

# Create neural networks model
model_cono = Sequential()
# Add layers to the model
model_cono.add(Dense(3, activation='relu', input_dim=2))      # first hidden␣
 ↪layer
model_cono.add(Dense(5, activation='relu'))                   # second hidden␣
 ↪layer
model_cono.add(Dense(1, activation='relu'))                   # output layer

# Compile the neural networks model
```

```
model_cono.compile(optimizer='sgd', loss='mse', metrics=['accuracy'])
# Train the neural networks model
model_cono.fit(x_train, z, epochs=2500,verbose=0)
```

[71]: <tensorflow.python.keras.callbacks.History at 0x1889f72b3a0>

[72]:
```
random.seed(15)
xt = np.random.uniform(-1,1,500)
yt = np.random.uniform(-1,1,500)
x_test = np.array(np.dstack((xt,yt)))
x_test = np.reshape(x_test, (500, 2))
zt = model_cono.predict(x_test)

# Para el gráfico
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
# Creating plot
ax.scatter3D(a, b, z, color = "green")
ax.scatter3D(xt, yt, zt, color = "red")
plt.title("Cono original y predicciones del Cono")

# show plot
plt.show()
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001889F7884C0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



[73]:
```
zt = np.reshape(zt, 500)
```

[74]:
```
fig = make_subplots(
    rows=1, cols=2,
    specs=[[{"type": "scatter3d"}, {"type": "scatter3d"}],],
)



fig.add_trace(
    go.Scatter3d(
    x=a,
    y=b,
    z=z,
```

```python
    mode='markers',
    marker=dict(
        size=5,
        color=z,                # set color to an array/list of desired values
        colorscale='Viridis',   # choose a colorscale
        opacity=0.8
    )),
row=1, col=1)

fig.add_trace(
    go.Scatter3d(
    x=xt,
    y=yt,
    z=zt,
    mode='markers',
    marker=dict(
        size=5,
        color=z,                # set color to an array/list of desired values
        #colorscale='Viridis',   # choose a colorscale
        opacity=0.8
    )),
row=1, col=2)
```

2 Referencias

- https://medium.com/analytics-vidhya/using-cross-validation-to-evaluate-different-models-regression-5f61ec89531
- http://www.w3big.com/es/python/func-number-seed.html
- https://mathinsight.org/parametrized_surface_examples#:~:text=Parametrize%20the%20single%20cone%
- https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590#:~:text=A%20single%2Dlayer%20perceptron%20is,weighted%20sum%20and%20activation%2
- https://www.deeplearningedge.com/implementing-logic-gates-with-a-neural-network/build-keras-model-of-a-nand-gate/
- https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/