

Francisco Leonid Galvez Flores

A01174385

[Enlace al repositorio en GitHub](#)

## ▼ Explicacion del algoritmo:

En esta evidencia lo que se tenia que realizar era la implementacion de un algoritmo maquina sin la necesidad de un framework, asi tambien se debia implementar una tecnica de algoritmo maquina con un framwork.

Para este caso lo que hicimos fue escoger el modelo de regresion logistica para un dataset de deteccion sobre si un tumor es benigno o no.

La regresion logistica es un algoritmo maquina que se usa para la clasificacion binaria, esto significa que con este algoritmo solo obtendremos dos valores posibles: (1/0), (True/False), etc. Algo que tenemos que decir es que si graficamos los datos estos se encontraran en un intervalo entre uno y cero, cabe aclarar que para este caso si el valor es menor a 0.5 se toma como cero y si es mayor a 0.5 se toma como uno. Si por ejemplo trazamos una linea diagonal esta no se acoplara perfectamente al dataset, lo que hacemos en este caso es generar una cruva sigmoide la cual se acopla mas propiamente a nuestro dataset.

La formula de la curva sigmoide es la siguiente:

$$s(x) = \frac{1}{1+e^{-x}}$$

Basandonos en la funcion sigmoide, las formulas de nuestro algoritmo seran:

$$y = mx + b$$

$$y = \Theta_0 + \theta_1 * x$$

$$s(y) = s(mx + b)$$

$$s(y) = \frac{1}{1+e^{\Theta_0+\theta_1*x}}$$

## ▼ Dataset:

Importamos nuestro dataset el cual dispone de 32 columnas y un total de 569 registros, ninguno de

```
id 0
diagnosis 0
radius_mean 0
texture_mean 0
perimeter_mean 0
area_mean 0
smoothness_mean 0
compactness_mean 0
concavity_mean 0
concave points_mean 0
symmetry_mean 0
fractal_dimension_mean 0
radius_se 0
texture_se 0
perimeter_se 0
area_se 0
smoothness_se 0
compactness_se 0
concavity_se 0
concave points_se 0
symmetry_se 0
fractal_dimension_se 0
radius_worst 0
texture_worst 0
perimeter_worst 0
area_worst 0
smoothness_worst 0
compactness_worst 0
concavity_worst 0
concave points_worst 0
symmetry_worst 0
fractal_dimension_worst 0
dtype: int64
```

Cada una de estas columnas define lo siguiente:

- Id:  
Numero de Id.
- Diagnosis:  
Tumor maligno y benigno(M/B).
- radius\_mean:  
Radio medio del tumor.
- texture\_mean:  
Texture media del tumor.
- perimeter\_mean:

Media del perimetro del tumor.

- area\_mean:

Area media del tumor.

- smoothness\_mean:

Suavidad media del tumor.

- compactness\_mean:

Compacidad media del tumor.

- concavity\_mean:

Concavidad media del tumor.

- concave points\_mean:

Cantidad media de puntos concavos del tumor.

- symmetry\_mean:

Media de la simetria del tumor.

- fractal\_dimension\_mean:

Dimension fractal media del tumor.

- radius\_se:

Desviacion estandar del tamaño tumor.

- texture\_se:

Desviacion estandar de la textura del tumor.

- perimeter\_se:

desviacion estandar del perimetro del tumor.

- area\_se:

Desviacion estandar del area del tumor.

- smoothness\_se:

Desviacion estandar de la suavidad del tumor.

- compactness\_se:

Desviacion estandar de la compacidad del tumor.

- concavity\_se:

Desviacion estandar de la concavidad del tumor.

- concave points\_se:

Desviacion estandar de la cantidad de puntos concavos del tumor.

- symmetry\_se:

Desviacion estandar de la simetria del tumor.

- fractal\_dimension\_se:

Desviacion estandar de la dimension fractal del tumor.

- radius\_worst:

El peor radio.

- texture\_worst:

La peor textura.

- perimeter\_worst:

El peor perimetro.

- area\_worst:

La peor area.

- smoothness\_worst:

La peor suavidad.

- compactness\_worst:

La peor compacidad del tumor.

- concavity\_worst:

La peor concavidad del tumor.

- concave points\_worst:

La peor cantidad de puntos concavos del tumor.

- symmetry\_worst:

La peor simetria del tumor.

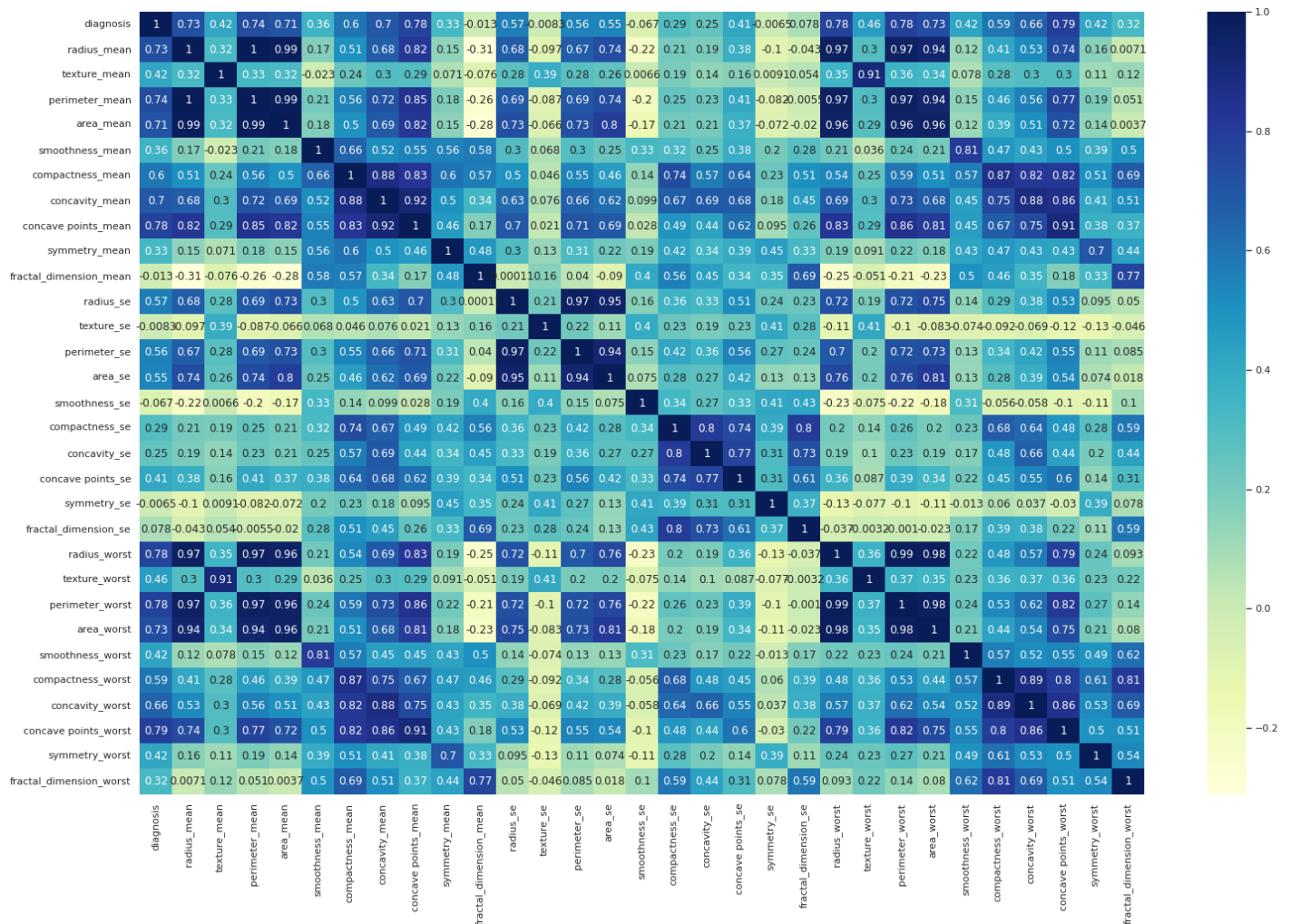
- fractal\_dimension\_worst:

La peor dimension fractal del tumor.

Posterior a la importacion de la base de datos realizamos algunos cambios necesarios, tales como la eliminacion de la columna 'id' asi como tambien cambiamos la columna de 'diagnosis' por

columnas binarias.

Tras esto realizamos un mapa de calor para ver que variables influian mas en la variable 'diagnosis':

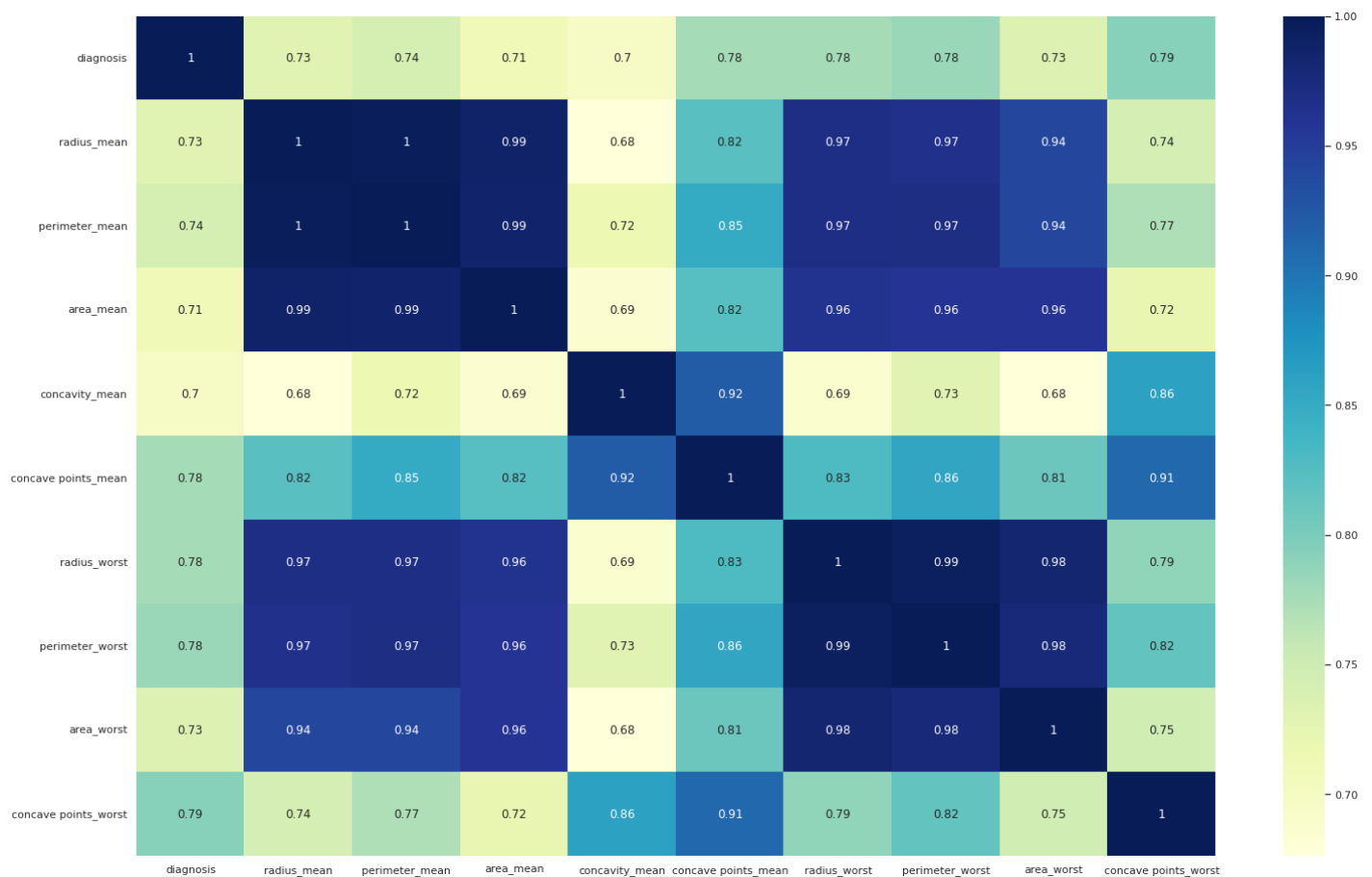


Con esto eliminamos las columnas que tengan minima relacion (menor a 70%) o relacion nula con nuestra variable a predecir:

- texture\_mean
- smoothness\_mean
- compactness\_mean
- symmetry\_mean
- fractal\_dimension\_mean
- radius\_se
- texture\_se
- perimeter\_se
- area\_se
- smoothness\_se

- compactness\_se
- concavity\_se
- concave points\_se
- symmetry\_se
- fractal\_dimension\_se
- texture\_worst
- smoothness\_worst
- compactness\_worst
- concavity\_worst
- symmetry\_worst
- fractal\_dimension\_worst

Tras esto volvemos a realizar un heatmap:



## ▼ Division del dataset:

Dividiremos el dataframe con ayuda de la libreria sklearn y su funcion 'train\_test\_split', se divide de manera aleatoria, del 100% del dataframe 10% es para testing, el 90% sobrante se convierte en

nuestro nuevo 100% y de ese 100% el 70% es para training y 30% para validating, eso se realiza de la siguiente manera.


Cabe aclarar que para este caso se tiene que dividir de una vez el dataset en la variable dependiente e independiente:

```
x = df.drop(["diagnosis"], axis = 1)
y = df.diagnosis
```


Posteriormente ya se realiza el split de los datos:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x.values, y.values, train_size=0.9, random_state=
```



```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, train_size=0.7, random_state=16)
```



Y en cuanto a cantidad de registros nos queda de la siguiente manera:

Datos: Registros		
0	Training	358
1	Validating	154
2	Testing	57

## ▼ Ahora nos toca hacer la implementacion del modelo:

En este caso como la implementacion del modelo es con un framework, en este caso 'sklearn' lo que hacemos es implementar el modelo con ayuda del framework, por lo que nos queda de la siguiente manera:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression(max_iter= 10000)
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_train)
```

Implementamos el algoritmo de regresion logistica con un numero de iteraciones maxima de 10,000.

## ▼ Accuracy:

Con la ayuda de 'sklearn' es mas facil calcular el accuracy:

Training:

```
score =accuracy_score(y_train,y_pred)
accuracy_training = score
print("Accuracy for the training set is: ", round(score*100, 4))
```

Accuracy for the training set is: 93.8547

Validating:

```
y_pred = logreg.predict(x_val)
score =accuracy_score(y_val,y_pred)
accuracy_validating = score
print("Accuracy for the training set is: ", round(score*100, 4))
```

Accuracy for the training set is: 96.1039

## ▼ Metricas(matriz de confusion y sus derivados):

Como 'sklearn' nos ayuda bastante, lo que hacemos ahora es simplemente llamar al paquete necesario para calcular la matriz de confusion:

Training:

```
from sklearn import metrics

y_pred = logreg.predict(x_train)
confusion_matrix = metrics.confusion_matrix(y_train, y_pred)
confusion_matrix
```



Matriz de confusion:

217---9

13---119

Para calcular el accuracy:

```
from sklearn.metrics import accuracy_score
acc = (accuracy_score(y_train, y_pred))*100
acc = round(acc, 2)
print("El accuracy es: ", acc, "%")
```

El accuracy es: 93.85 %

Para calcular el recall:

```
from sklearn.metrics import recall_score
rec = (recall_score(y_train, y_pred))*100
rec = round(rec, 2)
print("El recall es: ", rec, "%")
```

El recall es: 90.15 %

Para calcular la precision:

```
from sklearn.metrics import precision_score
pre = (precision_score(y_train, y_pred))*100
pre = round(pre, 2)
print("La precision es: ", pre, "%")
```

La precision es: 92.97 %

Ahora mandamos a llamar el paquete classification report, esto para tener todas las metricas que podemos sacar de la matriz de confusion:

```
from sklearn.metrics import classification_report

target_names = ['Sin tumor', 'Con tumor']
print(classification_report(y_val, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
Sin tumor	0.96	0.97	0.97	226
Con tumor	0.95	0.93	0.94	132
accuracy			0.96	358
macro avg	0.96	0.95	0.95	358
weighted avg	0.96	0.96	0.96	358

Validating:

```
from sklearn import metrics

y_pred = logreg.predict(x_val)
confusion_matrix = metrics.confusion_matrix(y_val, y_pred)
confusion_matrix
```

Matriz de confusion:

93---1

5---55

Para calcular el accuracy:

```
from sklearn.metrics import accuracy_score
acc = (accuracy_score(y_val, y_pred))*100
acc = round(acc, 2)
print("El accuracy es: ", acc, "%")
```

El accuracy es: 96.1 %

Para calcular el recall:

```
from sklearn.metrics import recall_score
rec = (recall_score(y_val, y_pred)*100)
rec = round(rec,2)
print("El recall es: ", rec, "%")
```

El recall es: 91.67 %

Para calcular la precision:

```
from sklearn.metrics import precision_score
pre = (precision_score(y_val, y_pred)*100)
```

```
pre = round(pre, 2)
print("La precision es: ", pre, "%")
```

La precision es: 98.21 %

Ahora mandamos a llamar el paquete classification report, esto para tener todas las metricas que podemos sacar de la matriz de confusion:

```
from sklearn.metrics import classification_report

target_names = ['Sin tumor', 'Con tumor']
print(classification_report(y_val, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
Sin tumor	0.95	0.99	0.97	94
Con tumor	0.98	0.92	0.95	60
accuracy			0.96	154
macro avg	0.97	0.95	0.96	154
weighted avg	0.96	0.96	0.96	154

## ▼ Explicacion dadas las metricas:

Para poder hacer el calculo de estas metricas de apoyo ocupamos la siguiente libreria:

```
from mlxtend.evaluate import bias_variance_decomp
```

Que contiene el metodo necesario para así poder obtener el bias, la varianza y el error cuadrado medio, para obtener esto solo le tenemos que pasar nuestro set de entrenamiento, el de validación y el tipo de algoritmo que estamos utilizando, que en nuestro caso es de regresión logística.

```
mse, bias, var = bias_variance_decomp(logreg, x_train, y_train, x_val, y_val, loss="mse", num_rounds=
```



## ▼ Bias:

Como bien sabemos este tipo de error es la diferencia entre la prediccion esperada de nuestro modelo y los valores verdaderos. Un bajo nivel de bias sugiere menos suposicion sobre la forma de la funcion objetivo mientras que un alto bias sugiere mas posiciones sobre la forma de la funcion objetivo.

```
print('Bias o sesgo: %.3f' % bias)
```

Bias o sesgo: 0.033

## ▼ Varianza:

Este error se refiere a la variabilidad o sensibilidad del modelo tran hacer modificaciones o cambios en los datos. Esto significa que los modelos de alta varianza generalmente se ajustaran bien a los datos de entrenamiento y por lo tanto lograran un rendimiento mucho mayo dentro de la muestra mientras que por el contrario este rendimiento sera mucho menor fuera de esta. Idealmente no deberia cambiar demasiado de un conjunto de datos de entrenamiento a otro. Los algoritmos que tiene un alto nivel de varianza usualmente son los que esta maytormente influenciados por los datos de entrenamiento.

```
print('Varianza: %.3f' % var)
```

Varianza: 0.011

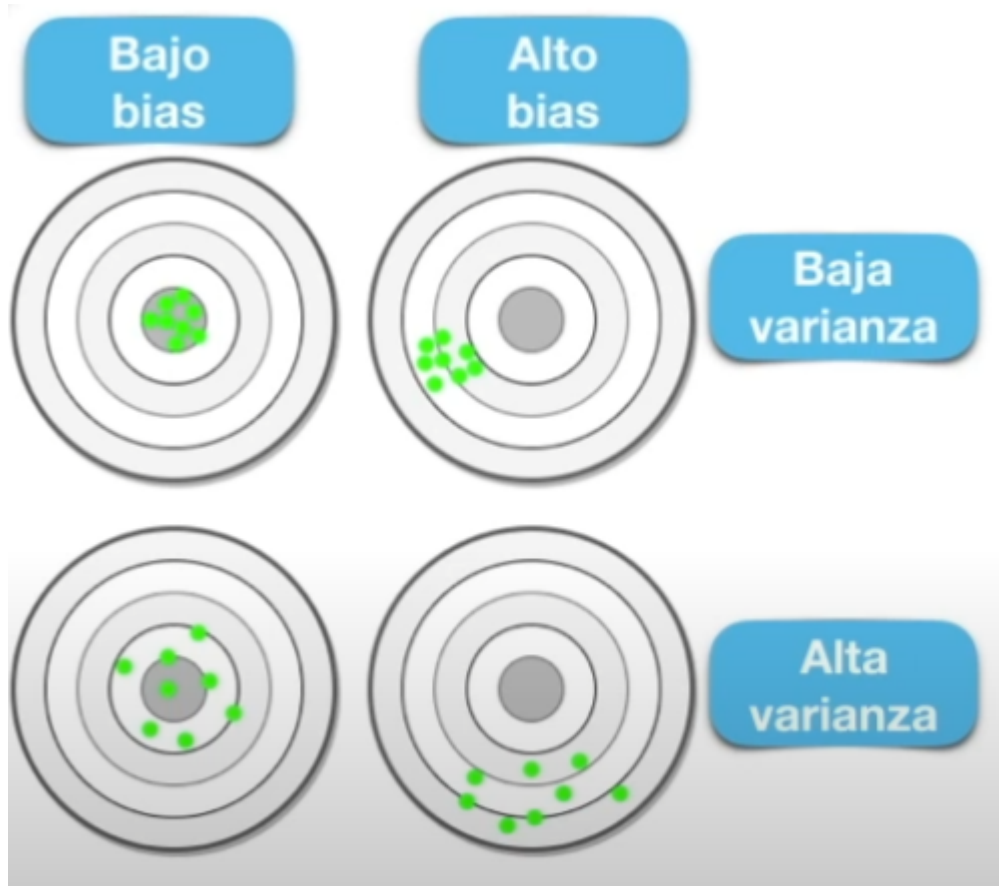
Dado que obtuvimos una varianza baja, se dice que se deben hacer ligeros cambios en la estimacion de la funcion objetivo con cambios en el conjunto de datos de capacitacion y esto era de esperarse porque estamos implementando un algoritmo de regresion logistica.

```
print('MSE: %.3f' % mse)
```

MSE: 0.045

## ▼ Nivel de ajuste del modelo:

El objetivo principal es lograr un bias bajo asi como una baja varianza, por lo que podemos ver anteriormente nuestro modelo dispone de ambas.



Dado lo analizado anteriormente y viendo el error aceptable en el set entrenamiento así como el error aceptable en el set de validación asumimos que nuestro modelo dispone de un **buen valance**.

## ▼ Predicciones:

Las predicciones las hicimos con el dataset de testing, algo que si generalizamos tanto para hacer esto a mano así como con la ayuda de un framework, lo que hicimos fue ver el tamaño del dataset de testing, posterior a esto generamos un grupo de 5 números aleatorios entre un número aleatorio entre el primer registro y el registro ubicado cinco posiciones antes del último:

```
l = np.random.randint(low=0, high=52)
```

Una vez explicado lo anterior, ahora con ayuda de la función 'predict' de 'sklearn' calculamos la variable a predecir de los cinco registros aleatorios:

```
y_pred = logreg.predict(x_pred)
```

Y obtenemos los siguientes resultados:

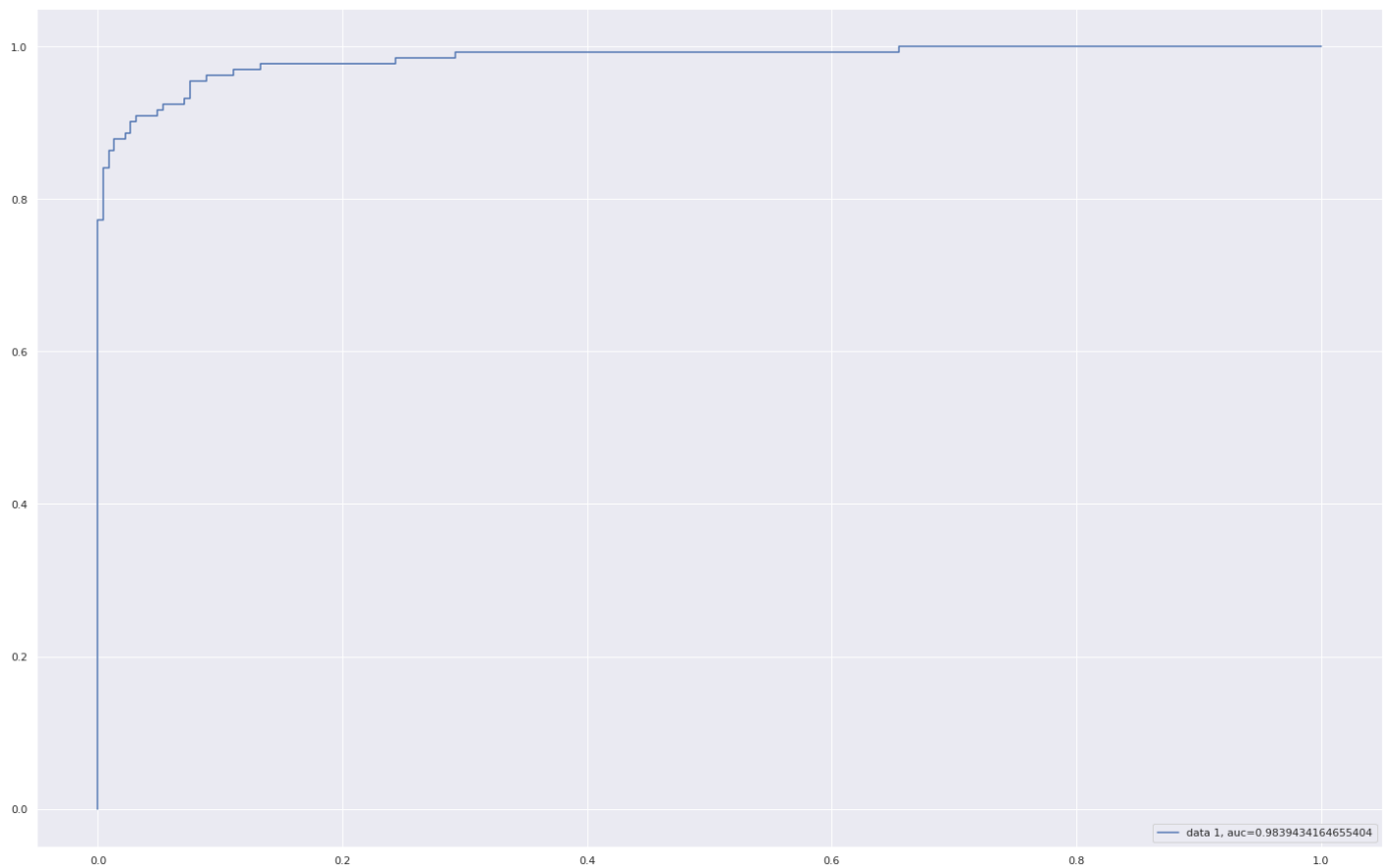
Numero de prediccion		Prediccion
0	1	1
1	2	1
2	3	0
3	4	1
4	5	0

## ▼ Curva ROC:

Tenemos que decir que la curva ROC es un grafico en el que se ponen los valores positivos y falsos en cada eje, y como podemos ver dieron un muy buen accuracy tanto en el dataset de training como en el dataset de validacion.

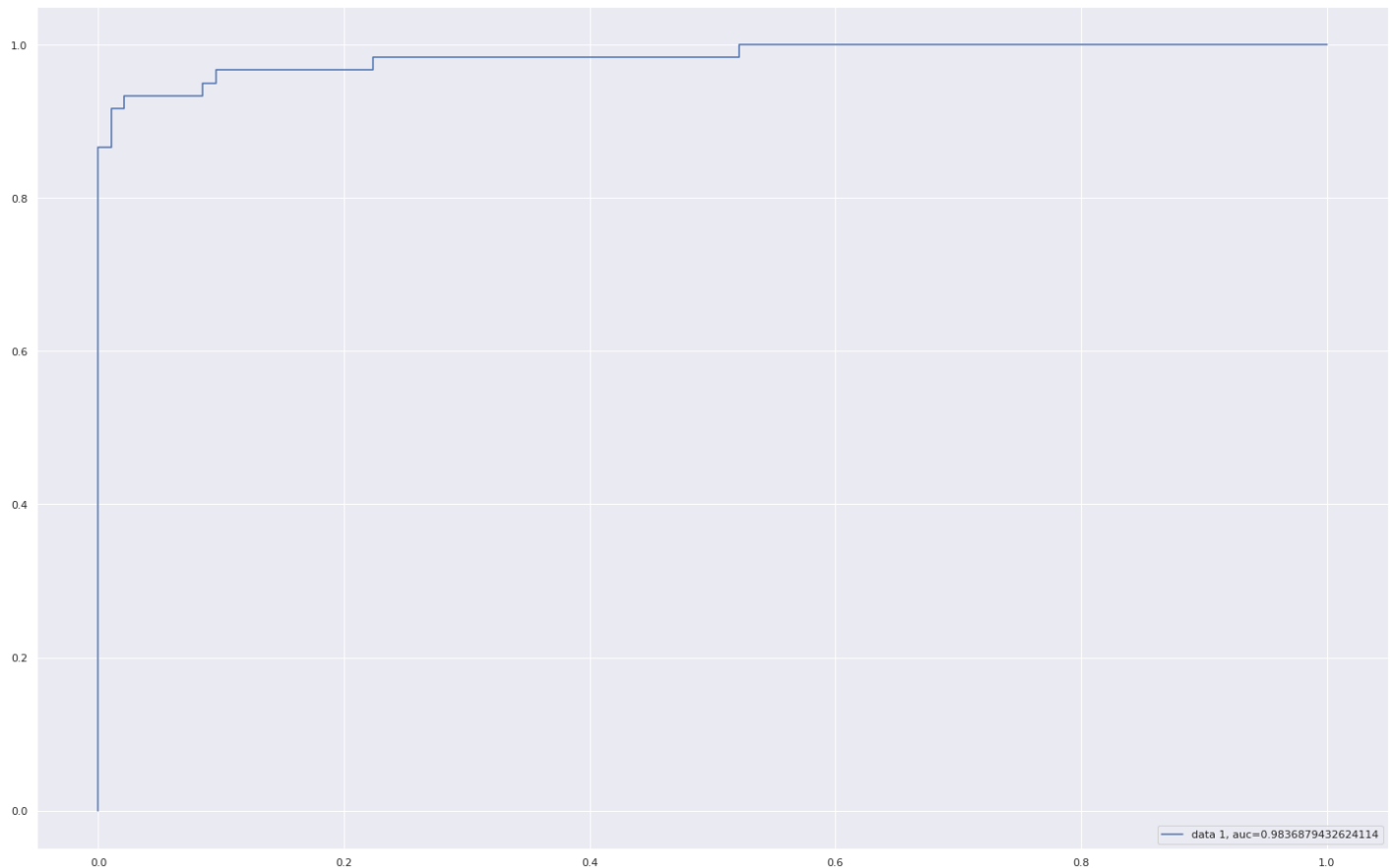
Training:

```
y_pred_proba = logreg.predict_proba(x_train)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_train, y_pred_proba)
auc = metrics.roc_auc_score(y_train, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Validating:

```
y_pred_proba = logreg.predict_proba(x_val)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_val, y_pred_proba)
auc = metrics.roc_auc_score(y_val, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



## Tecnicas de regularizacion para mejor desempeño del modelo:

Para esta parte lo que hicimos fue realizar regularizacion L2.

Antes que nada decimos que existe la probabilidad de que en la regresion logistica un modelo se sobreajuste, mas que nada cuando se trata de datos polinomicos, como es nuestro caso.

Por lo que, para regularizar el algoritmo y con esto hacer que nuestra toma de decision no disponga de un limite, lo que hacemos es hacer una penalizacion al sesgo del modelo, la cual seria la regularizacion L2, que se define matematicamente de la siguiente manera:



$$J(\theta) = \underbrace{\frac{1}{m} \sum_{i=1}^m \left[ -y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]}_{\text{Loss function}} + \underbrace{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2}_{\text{Ridge function}}$$

Como vemos lo que se hace es añadir una función de Ridge a la fórmula de gradiente descendente para así obtener una optimización más avanzada de nuestro algoritmo.

```
regularized_lr = LogisticRegression(penalty='l2', solver='newton-cg', max_iter=200)
regularized_lr.fit(x_train, y_train)
reg_pred = regularized_lr.predict(x_val)
```

```
print('Precision score', np.round(precision_score(y_val, reg_pred)*100, 2), "%")
print('Recall score', np.round(recall_score(y_val, reg_pred)*100, 2), "%")
```

Precision score 98.21 % Recall score 91.67 %

Dado los resultados podemos notar un ligero overfitting de nuestro modelo, si bien no es mucho se debe tomar en cuenta que esto sucedió.

```
confusion_matrix = metrics.confusion_matrix(y_val, reg_pred)
confusion_matrix
```

93--1

5--55

## ▼ Conclusion:

A mi criterio personal considero que la implementacion de un modelo de regresion logistica en este modelo fue una excelente idea, ya que la variable a predecir son binarias, los algoritmos nos dieron muy buenos 'accuracys', otro tipo de modelos que podrian funcionar en nuestro dataset podrian ser arboles de decision o redes neuronales.

Cabe aclarar que al implementar la tecnica de regularizacion, el algoritmo sufrio un ligero overfitting, asi que es de dudar si esto ayudo o no a nuestro algoritmo.

## Enlaces:

[Notebook del algoritmo](#)

[Archivo python del algoritmo](#)

[Dataset](#)

---

Productos pagados de Colab - [Cancela los contratos aquí](#)

