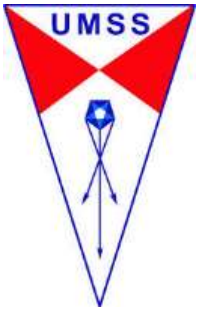



UNIVERSIDAD MAYOR DE SAN SIMÓN

POSGRADO FACULTAD DE CIENCIAS Y TECNOLOGÍA


# APLICACIÓN DE MODELOS MACHINE LEARNING

# PARA

# DETECCION DE FRAUDE CON TARJETA DE CREDITO

Presentado por:

Eguivar Vilca Shirley Carminia

COCHABAMBA - BOLIVIA

Cochabamba 29 de Diciembre de 2019

# INDICE

# 1 INTRODUCCION

*"Machine learning is a core, transformative way by which we're rethinking everything we're doing".* – Sundar Pichai

Este proyecto busca aplicar técnicas de machine learning a un set de datos que corresponde a transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeos. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de 284,807 transacciones.

# 2 OBJETIVO

## 2.1 Objetivo General

Aplicar técnicas de Machine Learning *(*Logistic Regression y Decision Tree*) para detectar* si hubo un fraude o no sobre el set de datos escogido.

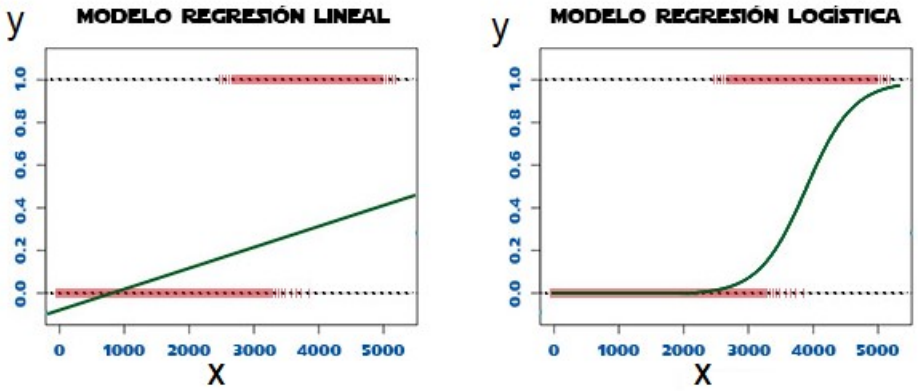
## 2.2 Objetivos Específicos

- Limpiar/Depurar los datos.
- Aplicar modelos de Machine Learning.
- Graficar ls resultados finales
- comparar los modelos para evaluar qué modelo funcionó mejor.

# 3 ALGORITMOS USADOS

## 3.1 Regresion Logistica

La regresión logística es un método de clasificación supervisado que devuelve la probabilidad de variable dependiente binaria que se predice a partir de la variable independiente del conjunto de datos, es decir la regresión logística predice la probabilidad de un resultado que tiene dos valores, ya sea cero o uno, falso o verdader o como en nuestro caso estos valores nos ayudaran a definir si hay fraude o no. La regresión logística tiene similitudes con la regresión lineal,pero, en la regresión lineal se obtiene una línea recta, la regresión logística muestra una curva. La regresión logística produce curvas logísticas que trazan los valores entre cero y uno.,la regresión logística es un modelo de regresión donde la variable dependiente es categórica y analiza la relación entre múltiples variables independientes.




### 3.2 Arbol de desicion

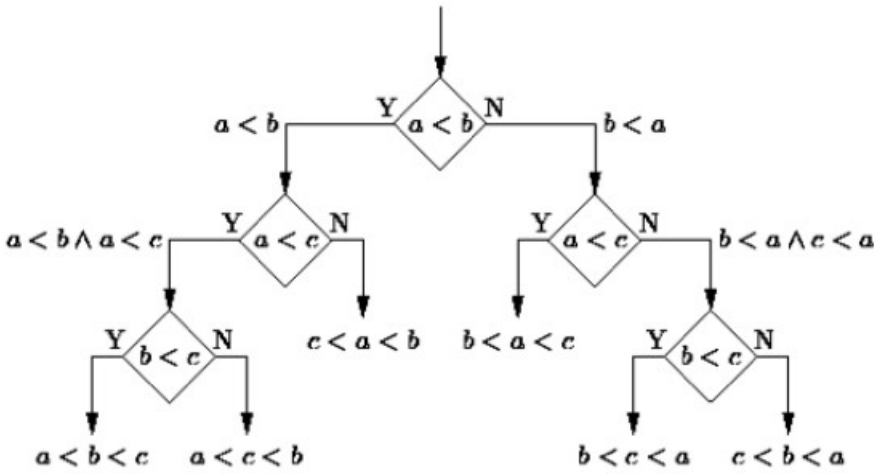
El árbol de decisiones es un algoritmo que utiliza un árbol como gráfico o modelo de decisiones y sus posibles resultados para predecir la decisión final, este algoritmo usa control condicional. Un árbol de decisión es un algoritmo para acercarse a funciones objetivo de valores discretos,en el que el árbol de decisión se denota por una función aprendida.

En nuestro caso daremos la etiqueta a una nueva transacción que es si es legítima o fraudulenta para qué clase la etiqueta es desconocida y luego el valor de la transacción se prueba contra el árbol de decisión, y después de eso desde el nodo raíz hasta la etiqueta de salida / clase para esa transacción, se rastrea una ruta.

Las reglas de decisión determinan el resultado del contenido del nodo hoja. En general las reglas tienen la forma de "Si la condición 1 y la condición 2 pero no la condición 3, entonces el resultado". Árbol de decisión ayuda a determinar los peores, mejores y esperados valores para diferentes escenarios, simplificado a comprender e interpretar y permite la adición de nuevos escenarios posibles.

Los pasos para tomar un árbol de decisión son:

- calcular la entropía de cada atributo utilizando conjunto de datos en problema,
- entonces, el conjunto de datos se divide en subconjuntos utilizando el atributo para el cual se obtiene ganancia máximo o entropía es mínimo,
- después de eso, para tomar un nodo de árbol de decisión que contenga ese atributo y,
- por último, la recursión se realiza en subconjuntos utilizando los atributos restantes para crear un árbol de decisión.




# 4 DESARROLLO

A continuación detallamos los pasos que se siguieron para completar la practica final del módulo.

## 4.1 Dataset

Los conjuntos de datos contienen transacciones realizadas con tarjetas de crédito en septiembre de 2013. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de 284.807 transacciones. El conjunto de datos está altamente desequilibrado, la clase positiva (fraudes) representan el 0.172% de todas las transacciones.

Contiene solo variables de entrada numéricas las columnas V1, V2, ... V28.

- La columna 'Tiempo' contiene los segundos transcurridos entre cada transacción y el primera transacción en el conjunto de datos.
- La columna 'Amount' es la Cantidad de la transacción, esta función puede ser utilizado para el aprendizaje dependiente de los costos dependiente del ejemplo.
- La columna 'Class' es la variable de respuesta y toma el valor 1 en caso de fraude y 0 en caso contrario.

```
In [2]: dataset = pd.read_csv('creditcard.csv')

        dataset.describe()
        dataset.head()
```

Out[2]:

| V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |

## 4.2 Analisis exploratorio de Datos

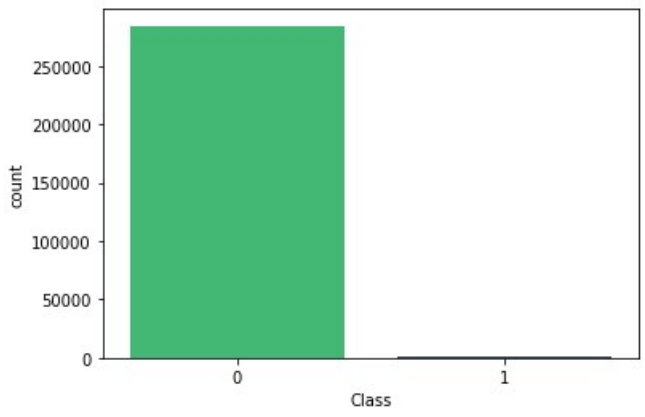
Se realizo un análisis exploratorio de datos para evaluar la información como tipos de columnas, cantidad de valores , etc.

Se identifica la columna Class como el objetivo(target)

```
In [556]: print('No Fraude :', round(dataset['Class'].value_counts()[0]/len(dataset) * 100,2), '%')
          print('Fraude :', round(dataset['Class'].value_counts()[1]/len(dataset) * 100,2), '%')
          sns.countplot('Class',data=dataset, palette=["#2ecc71","#34495e"])

          No Fraude : 99.83 %
          Fraude : 0.17 %
```

En este análisis se ha identificado que dada la relación de desequilibrio para la columna Class.

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1b4010eb1d0>
```




NO Fraude 99.83%

Fraude 0.17%

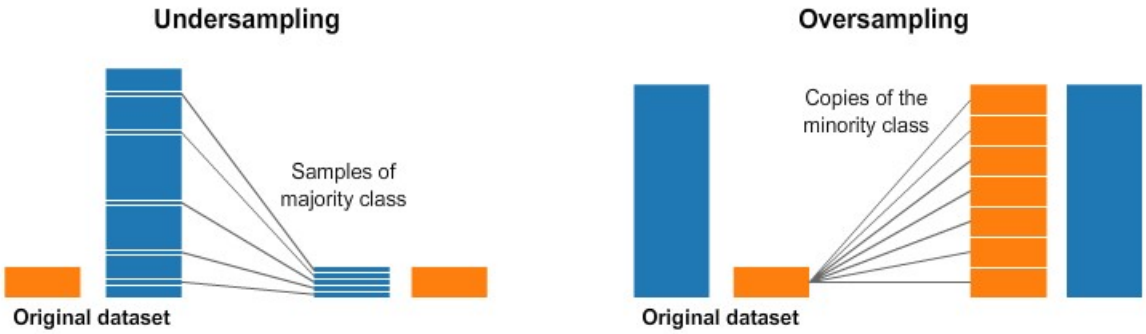
Dado que la mayoría de las transacciones son de no fraude tenemos un dataset no balancedo, si utilizamos el dataset sin modificación alguna como base para nuestros modelos y análisis predictivos, podríamos enfentrarnos al "***overfitting***" que es el efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado

Por lo tanto crearemos una submuestra del dataset original para tener la misma cantidad de casos de Fraude y No Fraude. (Undersampling)

## 4.3 Limpieza de datos

### 4.3.1 Resampling

Una técnica ampliamente adoptada para tratar con conjuntos de datos altamente desequilibrados se llama remuestreo(Resampling). Consiste en eliminar muestras de la clase mayoritaria (Undersampling) y / o agregar más ejemplos de la clase minoritaria (Oversampling).

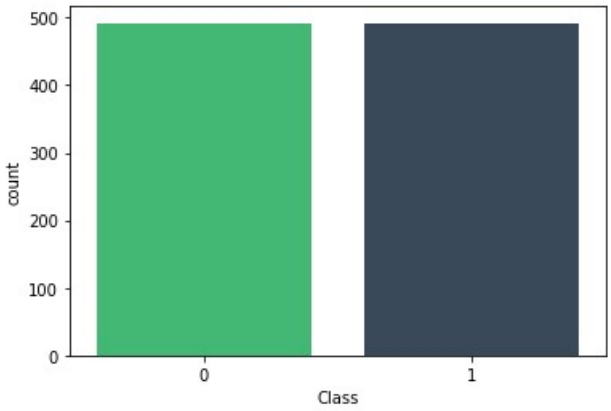



En este caso en particular aplicaremos Undersampling para balancear el dataset original que básicamente consistio en remover datos para tener un dataset mas balanceado y evitar el overfitting.

```
dataset = dataset.sample(frac=1)

# Divide by class: amount of fraud classes 492 rows.
fraud_df = dataset.loc[dataset['Class'] == 1]
no_fraud_df = dataset.loc[dataset['Class'] == 0]#[:492]

# Random under-sampling
df_class_0_under = no_fraud_df.sample(492)
df_test_under = pd.concat([df_class_0_under, fraud_df], axis=0)

#new dataframe
new_df = df_test_under

print('Random under-sampling:')
#print(new_df.Class.value_counts())
target_count = new_df.Class.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')
```

```
Random under-sampling:
Class 0: 492
Class 1: 492
Proportion: 1.0 : 1
```

En la siguiente grafica podemos apreciar el resultado del balanceo del dataset:




## 4.4 Aplicación de modelos de Machine Learning

Se aplicaron dos modelos de ML para ver cuál era más preciso para predecir si hubo fraude o no. Se uso de la librería scikit-learn.

### 4.4.1 Regresion Logistica

In [637]:
```
model_LogisticRegression = LogisticRegression()
```

In [638]:
```
#Trainning
model_LogisticRegression.fit(X_train, y_train)
```

```
C:\Users\carminia.eguivar\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\linear_model\_logist
ic.py:939: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html.
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[638]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [639]:
```
# Predicting
y_pred = model_LogisticRegression.predict(X_test)
result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
result.head(15)
```

Out[639]:

| | Actual | Predicted |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 0 |
| **2** | 1 | 1 |
| **3** | 1 | 1 |
| **4** | 1 | 1 |
| **5** | 1 | 0 |
| **6** | 0 | 0 |
| **7** | 0 | 0 |
| **8** | 1 | 1 |
| **9** | 1 | 1 |
| **10** | 1 | 1 |

### 4.4.2 Arbol de decision

In [644]:
```
model_DecisionTree = DecisionTreeClassifier()
```

In [645]:
```
#Trainning
model_DecisionTree.fit(X_train, y_train)
```

Out[645]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [646]:
```
# Predicting
y_pred = model_DecisionTree.predict(X_test)
result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
result.head(15)
```

Out[646]:

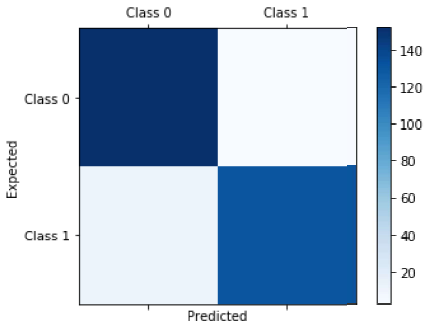
| | Actual | Predicted |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 0 |
| **2** | 1 | 0 |
| **3** | 1 | 1 |
| **4** | 1 | 1 |
| **5** | 1 | 0 |
| **6** | 0 | 0 |
| **7** | 0 | 0 |
| **8** | 1 | 1 |
| **9** | 1 | 1 |
| **10** | 1 | 1 |

## 4.5 Resultados finales

Se aplicó métodos de visualización sobre el resultado final para obtener conocimiento sobre el modelo creado.

```
#Confusion matrix
print("Confusion matrix: Logistic Regression")
conf_mat=confusion_matrix(y_test, y_pred)
print(conf_mat)

labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Confusion matrix: Logistic Regression
[[152   3]
 [ 11 131]]
```

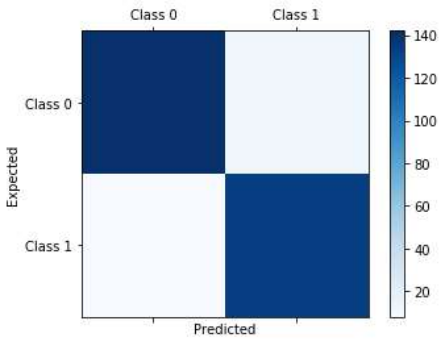



```
# Metrics by Training and Test
print ('Metrics with train dataset: ')
train_pred = model_LogisticRegression.predict(X_train)
print (classification_report(y_train, train_pred))
print ('Metrics with test dataset: ')
test_pred = model_LogisticRegression.predict(X_test)
print (classification_report(y_test, test_pred))
```

```
Metrics with train dataset:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       342
           1       0.96      0.91      0.93       350

    accuracy                           0.93       692
   macro avg       0.94      0.94      0.93       692
weighted avg       0.94      0.93      0.93       692

Metrics with test dataset:
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       155
           1       0.98      0.92      0.95       142

    accuracy                           0.95       297
   macro avg       0.96      0.95      0.95       297
weighted avg       0.95      0.95      0.95       297
```

```
#Confusion matrix
print("Confusion matrix: Decision Tree")
conf_mat=confusion_matrix(y_test, y_pred)
print(conf_mat)

labels = ['Class 0', 'Class 1']
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat, cmap=plt.cm.Blues)
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Expected')
plt.show()
```

```
Confusion matrix: Decision Tree
[[142  13]
 [  8 134]]
```

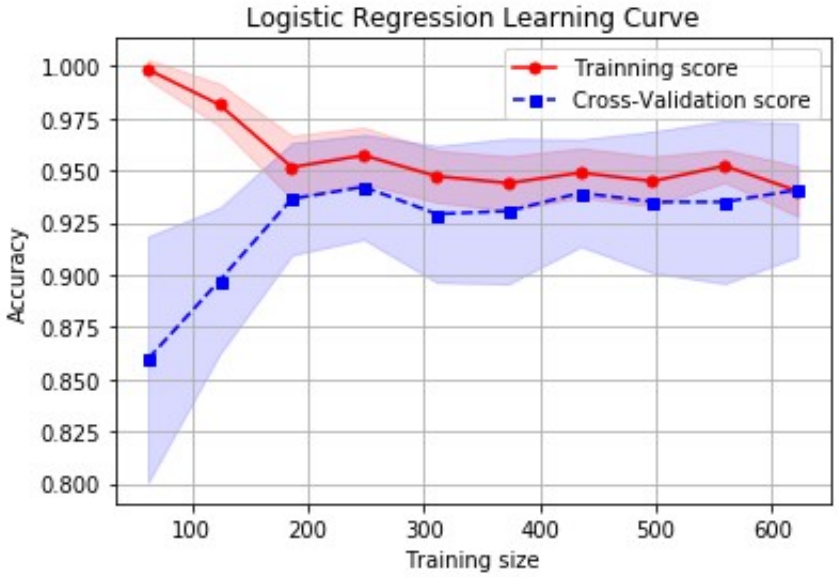


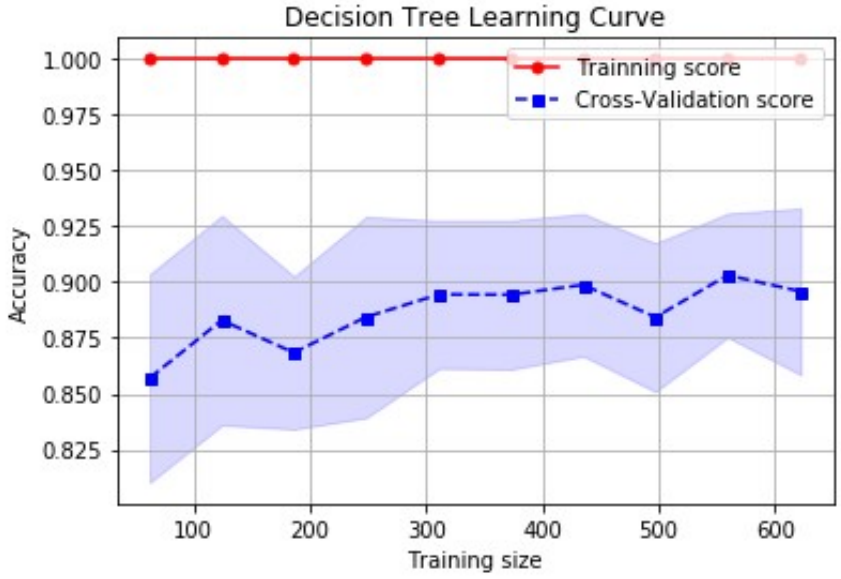

```
# Metrics by Training and Test
print ('Metrics with train dataset: ')
train_pred = model_DecisionTree.predict(X_train)
print (classification_report(y_train, train_pred))
print ('Metrics with test dataset: ')
test_pred = model_DecisionTree.predict(X_test)
print (classification_report(y_test, test_pred))
```

```
Metrics with train dataset:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       342
           1       1.00      1.00      1.00       350

    accuracy                           1.00       692
   macro avg       1.00      1.00      1.00       692
weighted avg       1.00      1.00      1.00       692

Metrics with test dataset:
              precision    recall  f1-score   support

           0       0.95      0.92      0.93       155
           1       0.91      0.94      0.93       142

    accuracy                           0.93       297
   macro avg       0.93      0.93      0.93       297
weighted avg       0.93      0.93      0.93       297
```

# 5 CONCLUSIONES

**Curva de Aprendizaje:**



**Metricas :**

**Logistic Regression**

```
accuracy = metrics.accuracy_score(y_pred,y_test)
print("accuracy:",accuracy)
precision = metrics.precision_score(y_pred,y_test)
print("precision:",precision)
f1_score = metrics.f1_score(y_pred,y_test)
print("f1_score:",f1_score)
recall_score = metrics.recall_score(y_pred,y_test)
print("recall_score/sensitivity:",recall_score)
```

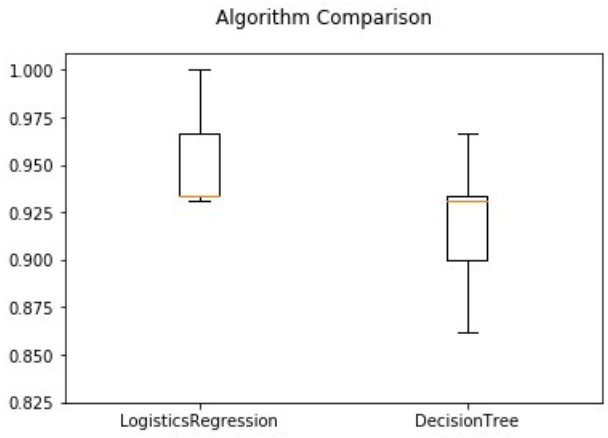
```
accuracy: 0.9528619528619529
precision: 0.9225352112676056
f1_score: 0.9492753623188407
recall_score/sensitivity: 0.9776119402985075
```

**Decision Tree**

```
accuracy = metrics.accuracy_score(y_pred,y_test)
print("accuracy:",accuracy)
precision = metrics.precision_score(y_pred,y_test)
print("precision:",precision)
f1_score = metrics.f1_score(y_pred,y_test)
print("f1_score:",f1_score)
recall_score = metrics.recall_score(y_pred,y_test)
print("recall_score/sensitivity:",recall_score)
```

```
accuracy: 0.9124579124579124
precision: 0.9507042253521126
f1_score: 0.9121621621621622
recall_score/sensitivity: 0.8766233766233766
```

Se concluye que la regresión logística tiene una precisión del 95.2%, mientras que el árbol de decisión muestra un 91.2%. Por lo tanto, concluimos que el algorithmo the Regresion Logistica tiene un mejor desempeño como se puede apreciar en la siguiente grafica comparativa.



# 6 BIBLIOGRAFÍA

Seaborn Displot - Recuperado 23 diciembre de 2019 de : https://seaborn.pydata.org/generated/seaborn.distplot.html

Confusion Matrix – Recuperado 23 diciembre de 2019 de: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

API reference- Recuperado 25 diciembre 2019 de :https://scikit-learn.org/stable/modules/classes.html

Boxplot con Python – Recuperado 25 diciembre de 2019: https://italoffvv.wordpress.com/2013/12/19/graficos-boxplot-con-python/

Boxplot - Recuperado 27 diciembre de:https://es.stackoverflow.com/questions/231468/generar-gr%C3%A1fico-boxplot-con-matplotlib-python-3-6

Compare algorithms- Recuperado 28 Diciembre de:https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/

SobreAjuste-Recuperado 25 diciembre 2019 de :https://relopezbriega.github.io/blog/2016/05/29/machine-learning-con-python-sobreajuste/

Metricas-Recuperado 29 Diciembre 2019 de: http://ligdigonzalez.com/metricas-de-evaluacion-clasificacion-con-scikit-learn-machine-learning/

Metrics-Recuperado 29 Diciembre 2019 de:https://towardsdatascience.com/metrics-for-evaluating-machine-learning-classification-models-python-example-59b905e079a5