Instituto Tecnológico y Estudios Superiores de Monterrey

Grecia Pacheco Castellanos A01366730

Materia: Inteligencia artificial avanzada para la ciencia de datos
Fecha: 11 de septiembre de 2022

ANÁLISIS Y REPORTE SOBRE EL DESEMPEÑO DE LOS MODELOS

INTRODUCCIÓN

Durante el módulo de aprendizaje máquina se pudieron analizar diversos tipos de modelos para el análisis de comportamiento de algún fenómeno con el objetivo de configurarlos y entrenarlos para poder realizar predicciones y con ellas tomar decisiones o interpretar para el entendimiento más a profundidad de un suceso.

En este caso, de manera individual se escogió un set de datos para poder adaptar algunos de las técnicas de aprendizaje máquina que ayudaran a entender los datos. De forma particular se usó el set de datos del Titanic, tratando de modelar el comportamiento de la supervivencia de una persona a la catástrofe únicamente con relación a su edad y clase en la que se encontraba.

Este proceso se realizó en dos etapas distintas, la primera de ellas con la implementación manual de un método, y la segunda haciendo uso de librerías ya existentes.

Para el momento de retroalimentación 1se realizó la actividad con la implementación manual de una regresión logística; mientras que para el segundo momento de retroalimentación se realizó la implementación con uso de librerías, específicamente “sklearn” , de manera que se pudieran comparar los modelos.

Con ambos casos implementados, se realizó una comparación del nivel de exactitud del modelo, y para poder mejorar el desempeño del modelo se debieron de modificar los hiperparámetros del mismo.

ANÁLISIS DEL MODELO

El modelo seleccionado es una regresión logística debido a que el resultado de si sobrevive o no se codifica como un 1 y 0 que es una variable binaria, lo cual es el comportamiento que busca modelar este tipo de regresión. Se usó a diferencia de una regresión lineal ya que de esta forma siempre se tendrá un valor binario y no rangos comprendidos entre estos.

Para poder hacer la configuración y entrenamiento de modelo fue necesaria la separación de los datos en una proporción 80 % de entrenamiento y 20% de prueba, esto se realizó sobre el mismo archivo de información del que se puede saber más dentro del apartado de anexos.

Para realizar el análisis y la comparación entre modelos se usaron distintos indicadores como el ajuste, sesgo y precisión de ambas formas de implementación del modelo, la siguiente tabla nos

indica las principales diferencias que hubo entre la implementación manual y la implementación con el uso de librerías.

Momento de retroalimentación 1 (sin framework)	Momento de retroalimentación 2 (con framework)

La implementación de la regresión se hizo a través de una clase donde se realizó la implementación de la función sigmoide:
`def fun_sigmoide(self,x):`
`  r = 1 / (1 + np.exp(-x))`
`  return r`
 	
En este caso la división de los datos se hace de manera manual:
`#División de dataset`
`pr= 0.8`
`n_train = math.floor(pr * x.shape[0])`
`n_test = math.ceil((1-pr) * x.shape[0])`
`x_train = x[:n_train]`
`y_train = y[:n_train]`
`x_test = x[n_train:]`
`y_test = y[n_train:]`
 	
Para hacer el entrenamiento de y la prueba del modelo se hizo uso de las funciones creadas por la clase RegresionLogistica creada.
`def fitting(self, x, y):`
`  samples, features = x.shape`
`  self.weights = np.zeros(features)`
`  self.bias = 0`
`  for i in range(self.itr):`
`  linear_m = np.dot(x, self.weights) + self.bias`
`  y_expected = self.fun_sigmoide(linear_m)`
`  derivative_w = (1/ samples) * np.dot(x.T , (y_expected -y))`
`  #derivative_w = (1/ samples) *(np.mat(x) * np.mat(y))`
`  derivative_b = (1/ samples) * np.sum(y_expected - y)`
`  self.weights -= self.rate * derivative_w`
`  self.bias = self.rate * derivative_b`
 	
Dicha función se mandaba llamar a través de la creación de un objeto de la clase.
`log_reg = RegresionLogistica(0.0001, 100)`
`log_reg.fitting(x_train, y_train)`
	En este caso para el uso de la regresión únicamente se importó el modelo de la librería correspondiente.
	`from sklearn.linear_model import LogisticRegression`
	
	Ya que se tenía permitido el uso de librerías, para la separación de los datos se utilizó la función de train_test_split de sklearn.
	`x_train , x_test, y_train, y_test = train_test_split(x,y, test_size= 0.2,random_state=1234)`
	
	Para hacer el entrenamiento en este caso se usa la función de fit que ya viene preestablecida.
	`rf_model = LogisticRegression()`
	`rf_model.fit(x_train,y_train)`
	
	Para hacer las pruebas se realizan las predicciones con los datos y la evaluación del modelo se usan nuevamente las funciones dadas por sklearn.
	`print("\n Train")`
	`y_predicted=rf_model.predict(x_train)`
	`print("score : " ,rf_model.score(x_train,y_train))`
	`print("bias  : " ,rf_model.intercept_)`
	`print(classification_report(y_train,y_predicted))`
	`# Testing del modelo`
	`print("\n Test")`
	`y_predicted=rf_model.predict(x_test)`
	`print("score : " ,rf_model.score(x_test,y_test))`
	`print("bias  : " ,rf_model.intercept_)`
	`print(classification_report(y_test,y_predicted))`
	
	En este caso se obtuvieron los datos correspondientes al ajuste del modelo con respecto a los datos de entrenamiento y de los datos de prueba.

Para poder realizar las predicciones es este caso se creo la función y se realiza la función que hace la comparación con las mismas y los valores esperados:

```python
def predictions(self, x):
    linear_m = np.dot(x, self.weights) + self.bias
    y_expected = self.fun_sigmoide(linear_m)
    y_predicted_cls = [1 if i > 0.5 else 0 for i in y_expected]
    return np.array(y_predicted_cls)
```

Finalmente, al evaluar esta primera prueba se obtuvo como resultado una precisión de: 0.61

```
 Train
score :  0.6095505617977528
bias  :  4.490167766681183e-06

 Test
score :  0.6424581005586593
bias  :  4.490167766681183e-06
```

```
 Train
score:  0.7008426966292135
              precision    recall  f1-score   support

           0       0.72      0.84      0.78       440
           1       0.65      0.47      0.55       272

    accuracy                           0.70       712
   macro avg       0.68      0.66      0.66       712
weighted avg       0.69      0.70      0.69       712


 Test
score :  0.6815642458100558
              precision    recall  f1-score   support

           0       0.69      0.87      0.77       109
           1       0.66      0.39      0.49        70

    accuracy                           0.68       179
   macro avg       0.67      0.63      0.63       179
weighted avg       0.68      0.68      0.66       179
```

Como primer análisis podemos observar que el modelo propuesto por la librería utilizada tiene una mejor precisión, donde la diferencia es de 0.682 a 0.642 para los datos de prueba.

Por otro lado, también se puede obtener el sesgo, que se muestra en la siguiente tabla:

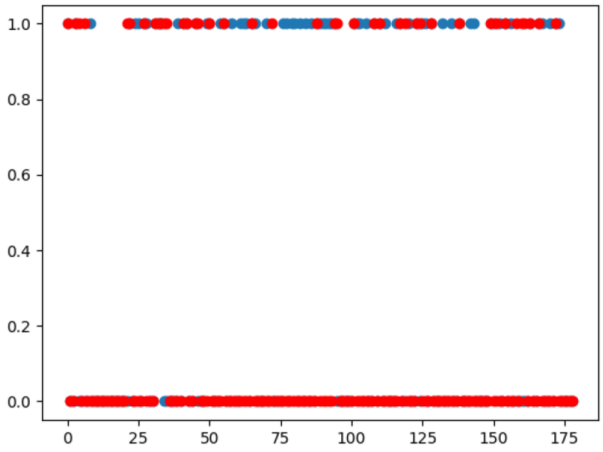
Momento de retroalimentación 1 (sin framework)	Momento de retroalimentación 2 (con framework)
El bias obtenido es de	En este caso los resultados son

```
 Train
score :  0.6095505617977528
bias  :  4.490167766681183e-06

 Test
score :  0.6424581005586593
bias  :  4.490167766681183e-06
```

```
 Train
score :  0.7008426966292135
bias  :  [2.02726046]
              precision    recall  f1-score   support

           0       0.72      0.84      0.78       440
           1       0.65      0.47      0.55       272

    accuracy                           0.70       712
   macro avg       0.68      0.66      0.66       712
weighted avg       0.69      0.70      0.69       712


 Test
score :  0.6815642458100558
bias  :  [2.02726046]
              precision    recall  f1-score   support

           0       0.69      0.87      0.77       109
           1       0.66      0.39      0.49        70

    accuracy                           0.68       179
   macro avg       0.67      0.63      0.63       179
weighted avg       0.68      0.68      0.66       179
```

En este caso podemos determinar que tenemos un sesgo positivo, lo que indica que el modelo mide por encima del valor real.

Comparando gráficamente los valores de las predicciones con los valores esperados se obtiene la siguiente gráfica:



Con la cual podemos observar que las predicciones (rojo) siguen un comportamiento muy similar al de las observaciones (azul) por lo cual podríamos determinar que en los casos en los que no coinciden son aquel porcentaje delimitado por el accuracy en el que el modelo continúa fallando.

También se puede observar como el nivel de exhaustividad es muy bajo, lo que indica que, si bien el modelo puede de representar el fenómeno, necesita de ser mejorado para poder ajustarse de mejor manera, o simplemente no es la alternativa más adecuada para este sistema.

TÉCNICAS DE REGULARIZACIÓN Y AJUSTE DE PARÁMETROS

Considerando de que se tuvo una precisión baja para el modelo con ambas propuestas, fue necesario el poder modificar los hiperparámetros de la regresión logística, para lo cual se deben de saber cuales son estos y por ende poder modificarlos.

Los hiperparámetros de el modelo de regresión logística son:

```
{'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}
```

Sabiendo los hiperparámetros involucrados se puede hacer una búsqueda de los mejores para poder tener un mejor ajuste del modelo , para ello podemos emplear un GridSearchCV con los parámetros más importantes del modelo y sus posibles valores.

El proceso anterior nos da los valores ideales de cada uno de los parámetros, los cuales son:

```
  warnings.warn(
{'C': 0.615848211066026, 'max_iter': 100, 'n_jobs': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
```

Dichos parámetros deben de ser utilizados para la configuración del modelo, en este caso se utilizó con el modelo realizado con el framework ya que de los parámetros que se establecieron en la implementación manual, el único que podía ser modificado era el máximo número de iteraciones, pero el valor recomendado era aquel con el que inicialmente se hicieron las pruebas.

```
params ={
    'C': 0.615848211066026,
    'penalty': 'l2',
    'solver': 'lbfgs',
    'max_iter': 100,
    'n_jobs' : 1

}

rf_model = LogisticRegression(**params)
```

Los parámetros modificados representan lo siguiente:

Hiperparámetros	Definición	Valor asignado y significado
penalty	Especifica la norma de la penalización	L2: Cuadrado de la magnitud de los de los coeficientes.
C	Inverso de la fuerza de regularización	0.616 – se usa para controlar el nivel de penalización en dicha proporción
solver	Algoritmo usado para la optimización	Lbggs: algoritmo default, se usa cuando se trabaja con problemas multiclase
max_iter	Número máximo de iteraciones que le toma para converger	100
N_jobs	Número de hilos usados para hacer la parelización entre clases	1

Una vez que se tienen este modelo, podemos volver a comparar el rendimiento con los nuevos parámetros:

```
 Train
score :  0.7008426966292135
bias  :  [2.00930471]
              precision    recall  f1-score   support

           0       0.72      0.84      0.78       440
           1       0.65      0.47      0.55       272

    accuracy                           0.70       712
   macro avg       0.68      0.66      0.66       712
weighted avg       0.69      0.70      0.69       712


 Test
score :  0.6815642458100558
bias  :  [2.00930471]
              precision    recall  f1-score   support

           0       0.69      0.87      0.77       109
           1       0.66      0.39      0.49        70

    accuracy                           0.68       179
   macro avg       0.67      0.63      0.63       179
weighted avg       0.68      0.68      0.66       179
```

Con esto podemos de concluir que aún teniendo los hiperparámetros de mejor ajuste del modelo, el sistema no muestra la mejor eficiencia para poder predecir el comportamiento, lo cual nos puede llevar a conclusiones como el hecho de que puede que no se están considerando todas las variables relevantes en el suceso y por ende el modelo no se está ajustando de la mejor manera al fenómeno.

CONCLUSIONES

A través del proceso de la implementación de un modelo de aprendizaje máquina manualmente se puede entender de mejor manera la matemática detrás del mismo, así como la importancia de los parámetros y las funciones necesarias para poder realizar las predicciones.

Al poder comparar un modelo propio con el establecido en una librería se puede notar cómo el poder configurar una mayor cantidad de hiperparámetros influye de gran manera en el ajuste que tiene el modelo sobre el fenómeno.

Asimismo, al usar el mismo set de datos que en la situación problema general del bloque, pude notar las diferencias en el ajuste teniendo más variables involucradas a través del tratamiento previo de los datos, por lo cual resalto la importancia de ese proceso antes de realizar el ajuste con la regresión.

Por último, con respecto al caso, podemos decir que si bien las variables modelan de gran manera y nos permiten entender el fenómeno, es importante la consideración de otras características que influyeran en el nivel de supervivencia.

BIBLIOGRAFÍA

Rodrigo, J. A. (s. f.). Regresión logística simple y múltiple. Recuperado 17 de septiembre de 2022, de https://www.cienciadedatos.net/documentos/27_regresion_logistica_simple_y_multiple

sklearn.linear_model.LogisticRegression. (s. f.). scikit-learn. Recuperado 17 de septiembre de 2022, de https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Glen, S. (2020, 7 julio). Regularization: Simple Definition, L1 & L2 Penalties. Statistics How To. Recuperado 17 de septiembre de 2022, de https://www.statisticshowto.com/regularization/

ANEXOS

La actualización de los momentos de retroalimentación se realiza en el mismo repositorio, cuya liga es la siguiente:

https://github.com/GrePaC/Modulo-2-Aprendizaje-maquina

Donde la estructura de este se compone de :

Nombre de archivo	Descripción
MomentoRetroalimentacion.py	Implementación del modelo desde cero, sin el uso de sklearn.
MomentoRetroalimentacion.py	Implementación del modelo con el uso de librerías.
Train.csv	Set de datos con el que se están realizando las pruebas.
M2_Reporte.docx	El presente documento donde se analizan y se concluyen los resultados obtenidos