Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Inteligencia Artificial Avanzada I. Módulo 2

Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución.

Enrique Maldonado Chavarría

Profesor:

Dr. Jorge Adolfo Ramirez Uresti

Fecha de entrega:

12 de septiembre del 2022

Bitácora:

Decidí implementar nuevamente la técnica de regresión logística con el dataset de semillas de calabaza extraído de Kaggle para poder obtener una comparación de los resultados que ofrece mi versión hecha de forma manual contra la que ofrece la librería de *scikit-learn* la cual es un estándar en la industria.

La preparación del dataset fue la misma:

- Conversión de la variable a predecir de categórica a variable
- Sortear el dataset para evitar sesgo en la información al crear subsets de entrenamiento y prueba.
- Normalizar los datos con la técnica *min-max*

Con la excepción de que la división en sets de entrenamiento y prueba fue ahora hecho con el comando "*train_test_split"* proveniente de igual forma de la librería *scikit-learn.*

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
2
3 print(X_train.shape,y_train.shape, X_test.shape, y_test.shape)
```

```
(2000, 12) (2000,) (500, 12) (500,)
```

Primera corrida:
Una vez preparado el dataset ,sin modificar ningún parámetro default, entrené el modelo de regresión logística de *scikit-learn* con los subsets "*X_train"* y *"y_train"* y este fue el resultado obtenido:

- Accuracy para el subset de prueba: ***84.8%***
    - El buen resultado inicial se puede atribuir (parcialmente) a que dentro de los procesos internos de la librería está el mejoramiento iterativo del modelo creando un subset de validación.

```
1 model = LogisticRegression().fit(X_train, y_train)
2 model.score(X_test, y_test)
```

```
0.848
```

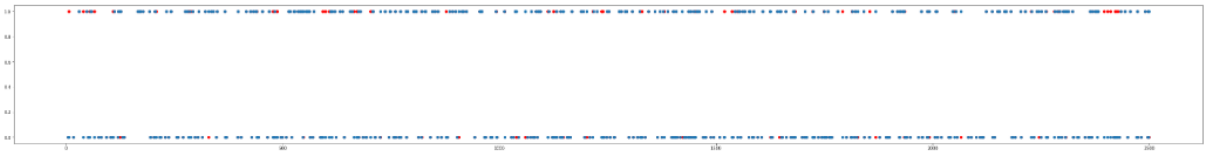
```
1 model.get_params()
```

```
{'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}
```

- *Bias: Bajo*

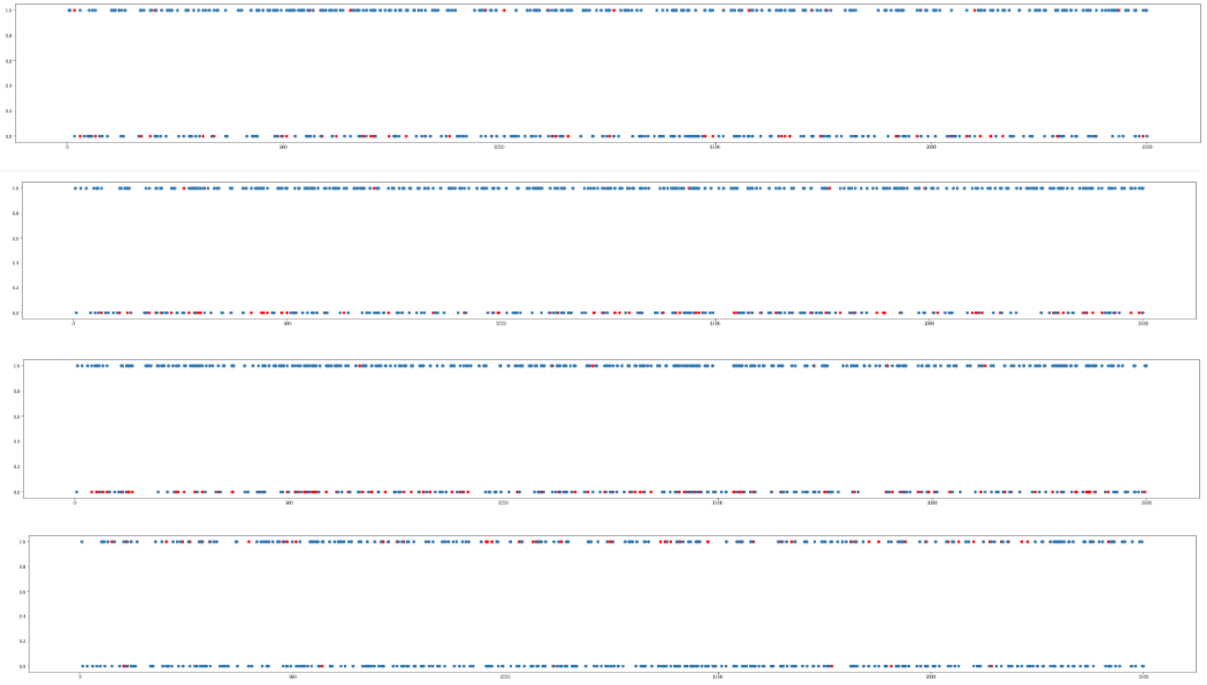
```
y_pred = model.predict(X_test)
test_id = X_test.index
```

```
plt.figure(figsize=(50, 6))
plt.scatter(test_id,y_test, c = 'red')
plt.scatter(test_id,y_pred)
plt.show()
```



Está gráfica muestra las instancias predecidas de forma correcta (puntos azules) y de forma incorrecta (puntos rojos). Gracias a la misma podemos argumentar que el sesgo del modelo es bajo.

- Varianza: Baja

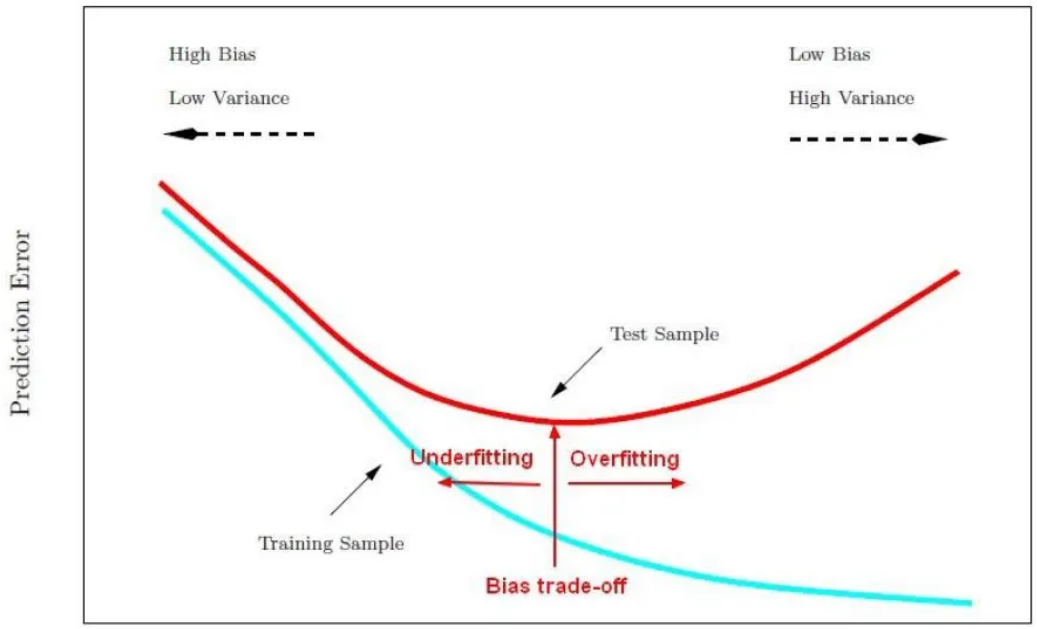


Al correr varias veces el mismo código, este nos ofrece la misma gráfica de comparación con ligeras diferencias: la cantidad de puntos rojos y su posición varía con cada ejecución. Sin embargo, las diferencias son sutiles y nos permiten argumentar que el modelo varía a un nivel bajo.

- Ajuste del modelo: Desconocido

Al trabajar con un modelo de librería “cerrada” como lo es *scikit-learn* y tras leer la documentación de la función de regresión logística, llego a la conclusión qué no es posible observar cómo se va ajustando el modelo cada iteración que sucede dentro del mismo.

Sin embargo, de ser posible esperaría poder obtener el valor del error que actualiza cada iteración e incluso graficarlo para obtener una figura parecida a la siguiente y con ella argumentar si el modelo sufre de *underfitting, overfitting* o se acerca más a un ajuste “adecuado”.



Regularización para mejorar el desempeño del modelo

Decidí cambiar los siguientes parámetros uno por uno:

- Fit_intercept: True
- Solver: Liblinear
- Dual: True

La esperanza era lograr un *accuracy* mayor al logrado con valores *default* y estos fueron los resultados:

Parámetros	***Accuracy***	***¿Mejor que el original?***
{'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}	**81.4%**	

{'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': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}	**83.2%**	
{'C': 1.0, 'class_weight': None, 'dual': True, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': None, 'solver': 'liblinear', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}	**87.4%**	

```
1 model = LogisticRegression(dual = True, fit_intercept=True, solver='liblinear').fit(X_train, y_train)
2 model.score(X_test, y_test)
```

```
0.874
```

Conclusiones:

El ajuste individual de un parámetro puede no sólo no mejorar un modelo sino empeorarlo. Sin embargo, si se entiende lo que cada uno modifica y cómo funcionan en conjunto se puede encontrar una combinación que ofrezca mejores resultados.

Por otro lado, es útil conocer cómo funciona un modelo para poder ajustarlo después pero esa habilidad no debe estar en conflicto con permitirse utilizar los modelos ya implementados en librerías de confianza que han demostrado ofrecer resultados igual de buenos a una velocidad mucho mejor (2s en promedio).