Momento de Retroalimentación: Módulo 2 - Análisis y Reporte sobre el desempeño del modelo

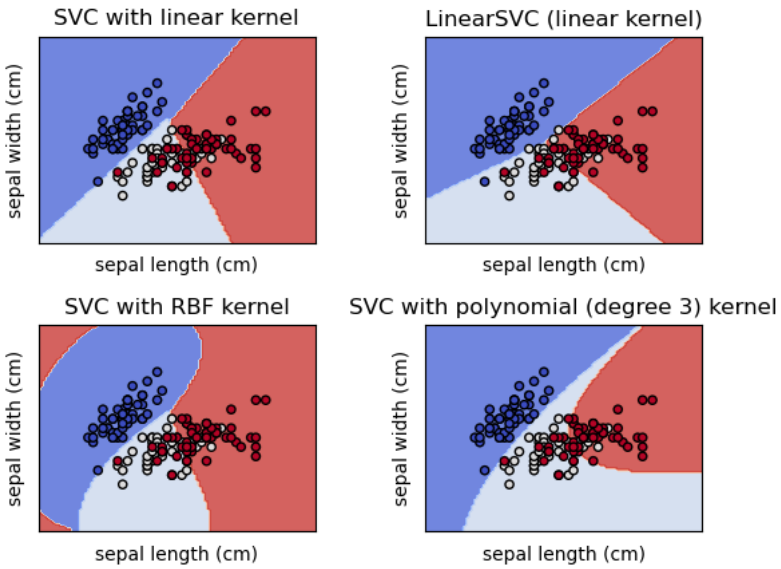
Raúl Youthan Irigoyen Osorio A01750476

13 de septiembre del 2022

Inteligencia artificial avanzada para la ciencia de datos

Este documento tiene el propósito de analizar la implementación del modelo de **regresión logística**, así como los procesos y elementos que giran en torno a la misma, es decir, los datos utilizados, cómo fueron utilizados para el entrenamiento y pruebas del modelo, cuáles fueron los resultados del mismo y posibles técnicas de ajuste de parámetros para mejorar su desempeño.

El modelo implementado tiene por objetivo tomar en cuenta un determinado número de atributos para clasificar sus valores y establecer así una predicción sobre la clase a la que la combinación de estos valores puede pertenecer. Este tipo de regresión puede utilizar diferentes algoritmos para realizar los cálculos necesarios, como se mostró en entregable anterior (https://docs.google.com/document/d/1q17RL7s4p2ZV3by8xAgSZ5nV0y9Ur0Vio2oSV6x6Qbw/edit), sin embargo, para el caso de este análisis se trabajará con el solucionador de gran escala (*implementado en lsklearn como* ***liblinear***), el cual se apoya en algoritmos como **SVM** (*Support Vector Machines*), que es efectivo en espacios multidimensionales, o bien, con un número alto de atributos a considerar pues construye un conjunto de **hiperplanos** como se muestra a continuación de manera bidimensional:



De esta manera, es posible determinar de manera más precisa la clase a la que un conjunto de valores pertenece, dado que no se presenta algún tipo de limitación por el número de dimensiones a la que el modelo esté sujeto.

Separación de conjuntos (Train/Test):

El dataset utilizado puede ser encontrado en este link. Los datos están constituidos por valores provenientes de un estudio de múltiples imágenes de células potencialmente malignas, separados en 9 columnas distintas que representan un atributo particular del estudio (*radio, área, perímetro, suavidad, forma, etc.*) mientras que la última columna representa el diagnóstico final (*benigna o maligna*).

Los datos fueron divididos de tal manera que el **80%** del dataset conforme al **training set** mientras el **20%** restante al **testing set**, de esta manera es posible comprobar qué tan alejadas/acercadas están las predicciones de los valores reales de los datos, como se puede observar a continuación:

```
        # Split into training and testing datasets

        self.train = self.df.sample(frac=0.8, random_state=25)

        self.test = self.df.drop(self.train.index)

        # Generate csv files with training and testing data

        self.train.to_csv(os.path.join(os.path.split(sys.path[0])[0], "data" "train.csv"), index=False)

        self.test.to_csv(os.path.join(os.path.split(sys.path[0])[0], "data", "test.csv"), index=False)
```

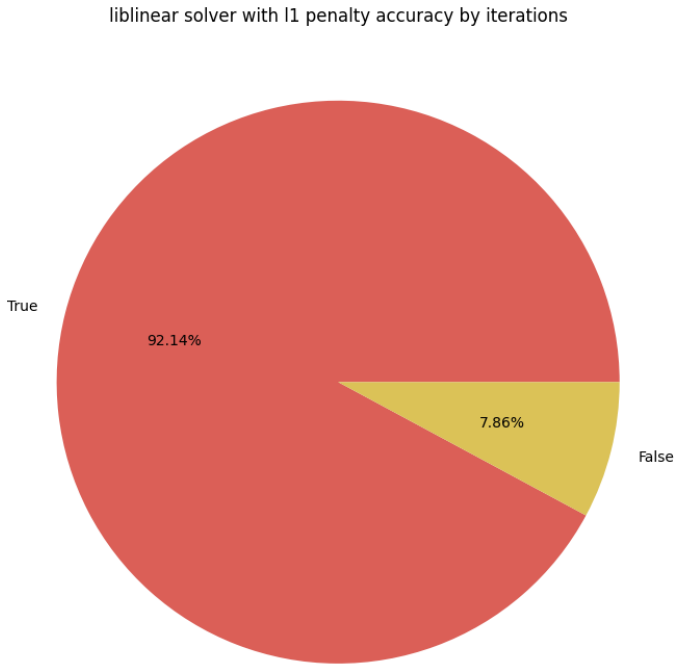
Diagnóstico de la varianza y el sesgo (*bias*):

Para todo modelo es esencial encontrar un balance prácticamente perfecto entre varianza y sesgo, esto es, considerar los errores de predicción y el rango de los valores para aquellos datos clave. El modelo debe analizar datos y hallar patrones en dicha información, usando estos patrones se

pueden establecer generalizaciones sobre determinadas instancias de los datos para aplicarlas en el **testing set** y realizar las predicciones correspondientes.

Cuando el **sesgo** es **alto**, lo que el modelo asume es demasiado básico pues no está capturando atributos importantes de los datos pues no ha hallado los patrones suficientes para hacer mejores estimaciones; cuando la **varianza** es **alta**, el modelo es altamente sensible a los cambios en los datos, haciéndolo sensible también al ruido.

En el caso del modelo de regresión logística implementado, la precisión de las predicciones luce de la siguiente manera (*True = Predicción acertada, False = Predicción equivocada*):



Para el cálculo de varianza, se implementaron las siguientes líneas de código:

```
results = run_model(x, y, x_test, 'l1', 'liblinear', -1, 1000)

variance = np.var(results)

print(variance)
```

Su resultado fue:

```
PS D:\irigx\Documents\School\7mo\Evaluación M2> & C:/Python310/python.exe
"d:/irigx/Documents/School/7mo/Evaluación M2/LogisticRegression (Deliverable
2)/processing/logistic_regression.py"

0.8163265306122448
```

Esto es un indicador de que la varianza es relativamente alta tomando en cuenta que el rango de los valores esperados es de **0 a 1**, por lo que el modelo es altamente sensible a los datos del **training set**.

Para el caso del **bias**, las siguientes líneas de código se implementaron:

```
sse = np.mean((np.mean(results) - y_test) ** 2)

bias = sse - variance

print(bias)
```

El resultado fue el siguiente:

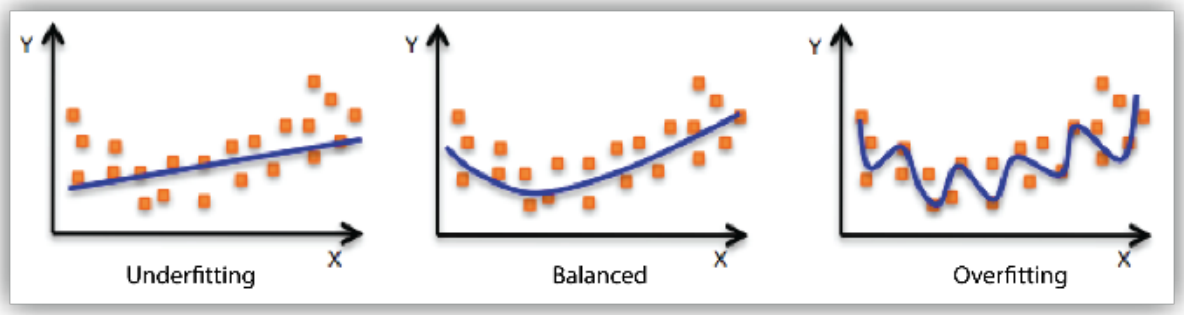
```
PS D:\irigx\Documents\School\7mo\Evaluación M2> & C:/Python310/python.exe
"d:/irigx/Documents/School/7mo/Evaluación M2/LogisticRegression (Deliverable
2)/processing/logistic_regression.py"

-0.006938775510204165
```

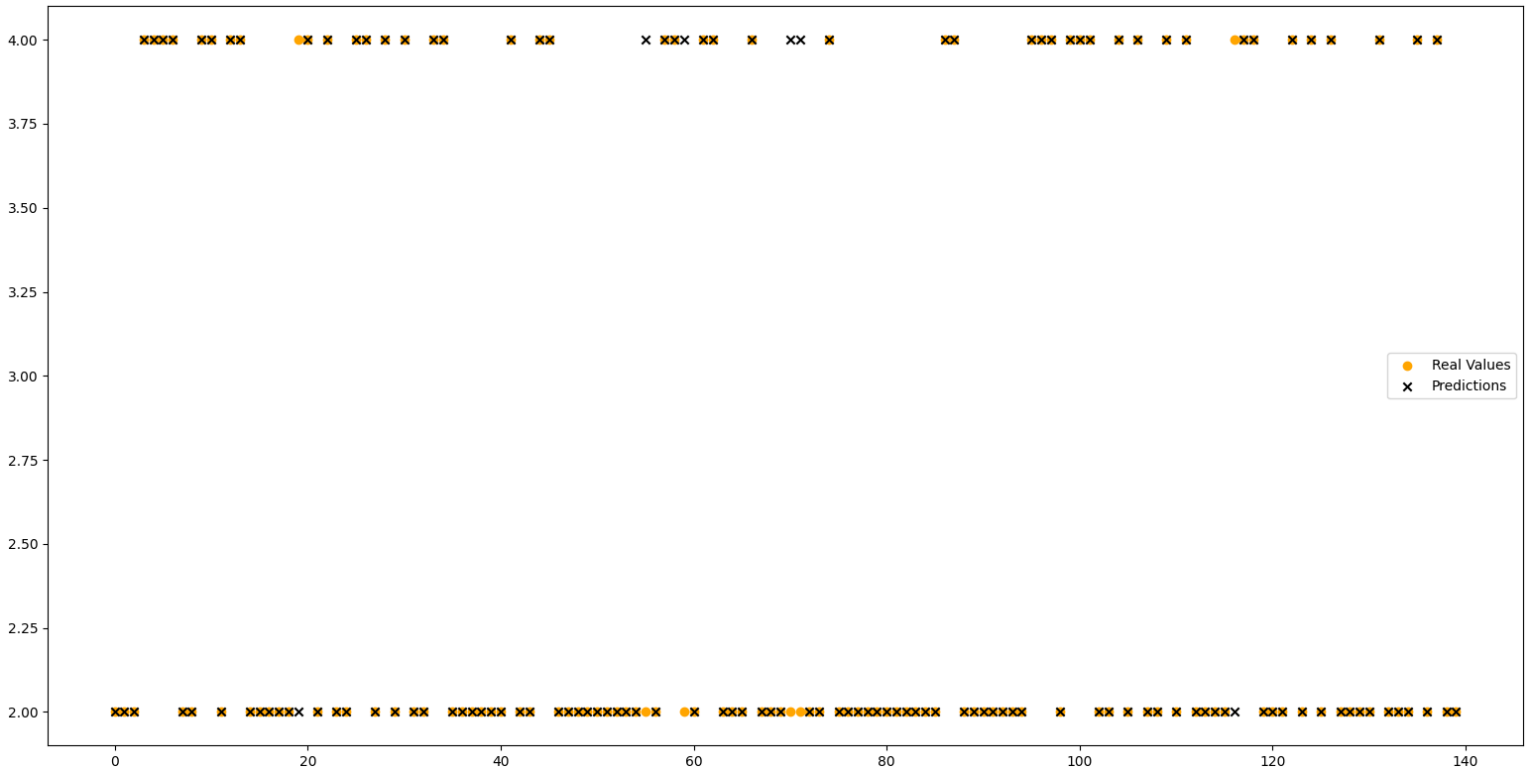
Es evidente que el **sesgo** tiene un valor bajo, lo que da a entender que el modelo aprendió los patrones suficientes sobre los datos del **training set**.

Explicación del nivel de ajuste del modelo:

Dado que la **varianza** es relativamente **alta** y el **sesgo** es bajo, es posible llegar a la conclusión de que el modelo corre el riesgo de presentar **over-fitting**, es decir, que el modelo siga demasiado el patrón de los datos del **training set** y que además es altamente sensible a datos que podrían ser incluso **ruido** dentro del mismo set, como se muestra en el tercer recuadro de la siguiente figura:



En el caso de nuestro modelo, las predicciones lucen de la siguiente manera:



Técnicas de regularización y ajuste de parámetros:

Para obtener un mejor resultado del modelo, se implementó el **escalamiento** de datos tomando en cuenta que el rango de cada atributo siempre debe ser entre **0 y 10**. Por esta razón, se redujo dicho rango para que sus valores estén entre **0 y 1**.

Además, es posible añadir **cross-validation**, que entrena al modelo múltiples veces con diferentes permutaciones de los **training** y **testing sets** con el fin de hacer más independientes las predicciones del **training set** y así encontrar un mejor equilibrio entre la **varianza** y el **sesgo**.

Para el ajuste de parámetros puede utilizarse un algoritmo de **pathfinding** como **Breadth-First Search** al considerar dichos parámetros como los ejes de una matriz que contiene diferentes valores para cada uno, utilizando como heurística la precisión de las predicciones del modelo con la selección actual y de sus vecinos.