Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo. (Portafolio Análisis)

Facundo Vecchi A01283666

Introduccion

En el portafolio de implementacion se intento encontrar cual seria el mejor de 8 modelos para predecir el tipo de vino del Wine dataset. El modelo que obtuvo el mejor resultado fue el de Random Forest Classifier de Sklearn. Debido a que este obtuvo un accuracy de 100% sobre el dataframe de prueba, no me hizo mucho sentido utilizarlo para este analisis ya que no estaria buscando mejorar el modelo si no que empeorarlo para ver las diferencias en la varianza y el sesgo. Por esta razon, decidi utilizar el modelo de Regression logistica de Sklearn el cual obtuvo una precision de 98%, aunque no hay mucho espacio para mejorar, el proposito de este analisis sera buscar cuales son los hiperparametros que reduzcan la varianza y el sesgo lo mas posible.

Modelo base

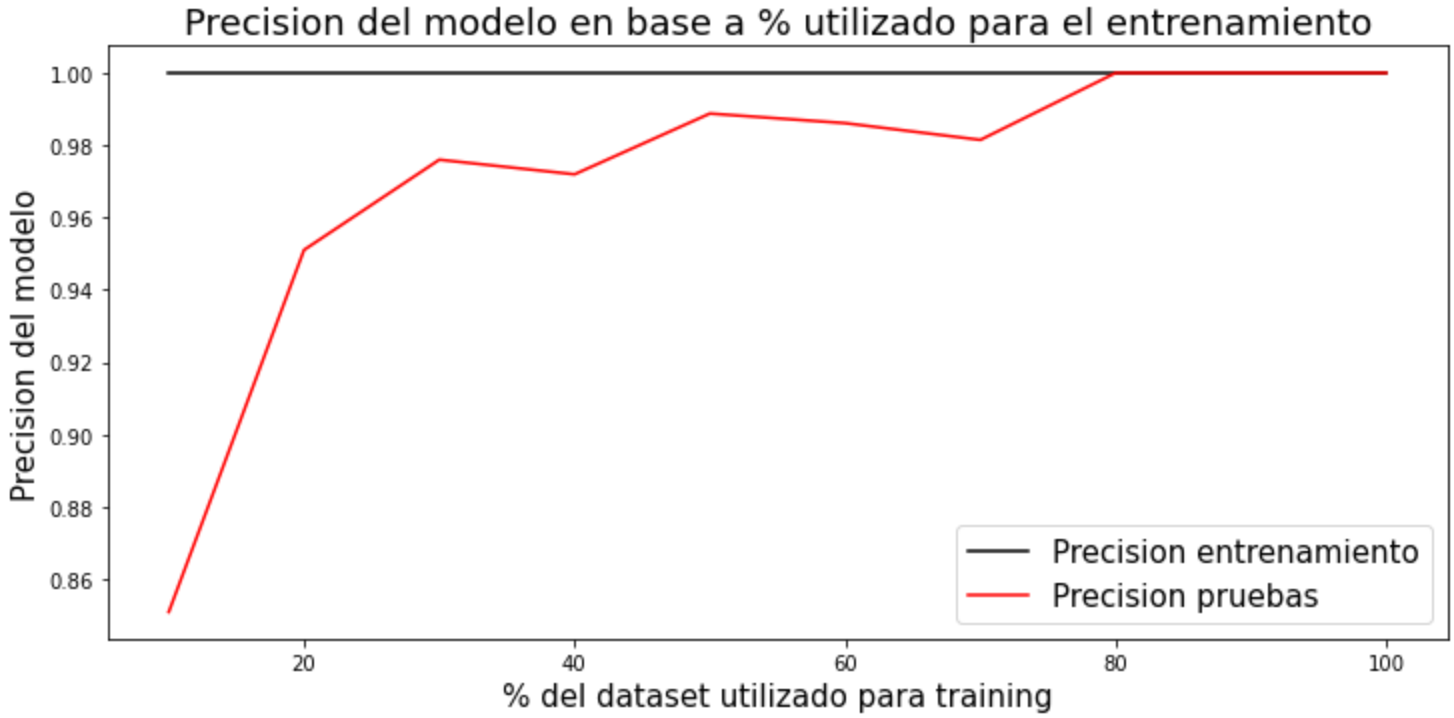
Primero vamos a observar el modelo base, sin modificar ningun parametro y calcular su varianza y sesgo para determinar si este se esta ajustando correctamente al dataset.

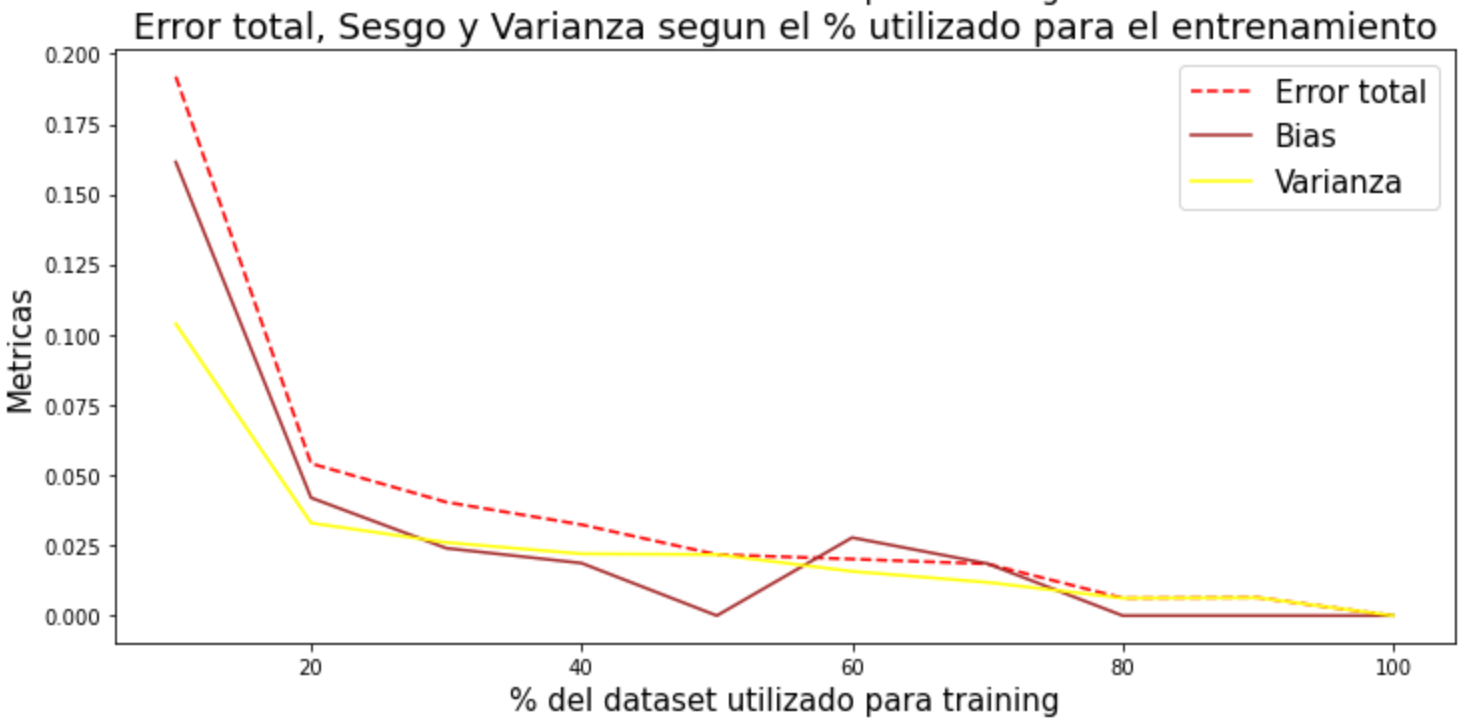
```
Precision                       0.981481
Error total esperado promedio   0.018426
Sesgo promedio                  0.018519
Varianza promedio               0.011852
dtype: float64
```

Como mencione anteriormente y se puede observar en los resultados, la regresion logistica con los parametros por defecto ya tiene una precision muy buena sobre el dataset utilizando un 70-30 split para entrenar y probar. Se puede observar tambien como tanto el sesgo como la varianza son bastante bajos.
Partiendo de estos resultados se probaran entrenar el modelo con diferentes parametros para tratar de reducir tanto la varianza como el sesgo, teniendo en cuenta que estos comunmente son proporcionalmente inversos.

Cambiando el split de entrenamiento y prueba

Antes de empezar a modificar los parametros, veremos cuanto cambia la precision del modelo, junto con el sesgo y su varianza cuando se utilizan diferentes porcentajes para dividir el dataset.
Para lograr esto, se va a calcular y graficar la precision del modelo, su error total, varianza y sesgo utilizando un porcentaje de 10% a 100% de training en pasos de 10%. (En total se entrenarian 10 modelos)





Observando estos resultados, a simple vista se puede concluir que cuanto mas porcentaje del dataset utilicemos para el entrenamiento, mejor es el modelo. En este caso, eso seria incorrecto, el data set de Wine tiene un numero de entradas muy pequeño, con tan solo 178 entradas, cuanto mas incrementamos el porcentaje de entrenamiento, menos datos tenemos con que validar el modelo. Esto signfica que, al utilizar arriba de 80% del dataset para entrenamiento, tan solo estamos utilizando 35 datos para validar, y el numero baja cada porcentaje que subamos arriba de eso.
Dirigiendo la atencion hacia la grafica de error total, sesgo y varianza, podemos observar que en este caso, la precision y (el sesgo, varianza y error total) parecen ser proporcionalmente inversos. Cuando la precision sube, estas 3 metricas bajan. Lo curioso es que podemos ver una anomalia alrededor del 50%-70% de training, aqui podemos observar como la precision baja tras haber subido significativamente en el lapso anterior. Justo en ese mismo brinco, del 50% al 60% podemos observar como el sesgo sube repentinamente tambien.
Aqui pueden estar pasando varias cosas:

- Realmente lo que esta pasando es que a partir del 60% de training, se empieza a tener suficientes datos para probar la verdadera precision del modelo, lo que signfica que los valores de precision obtenidos estan inflados y no son los

verdaderos

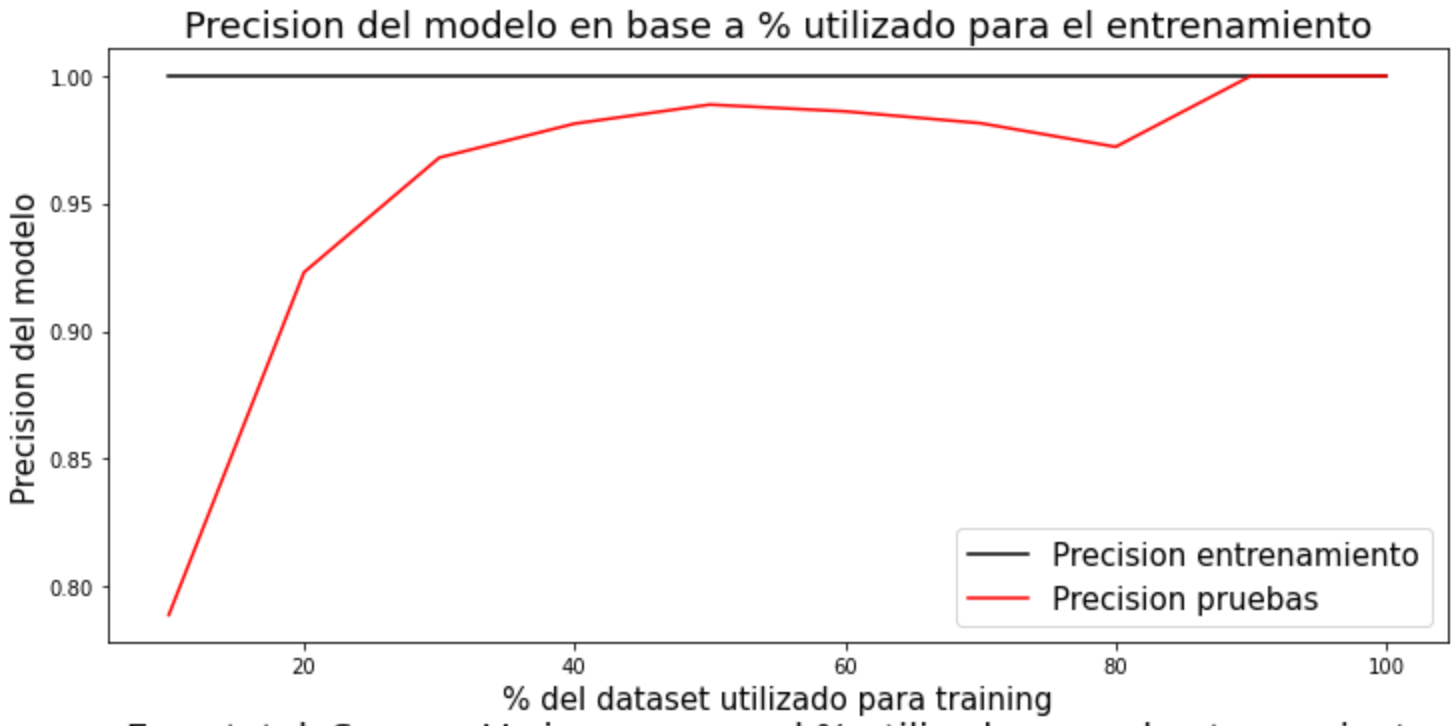
- El random_state que se esta utilizando para el train_test_split causa que al utilizar algun porcentaje alrededor de 50%-70% el modelo no este siendo entrenado con los datos optimos

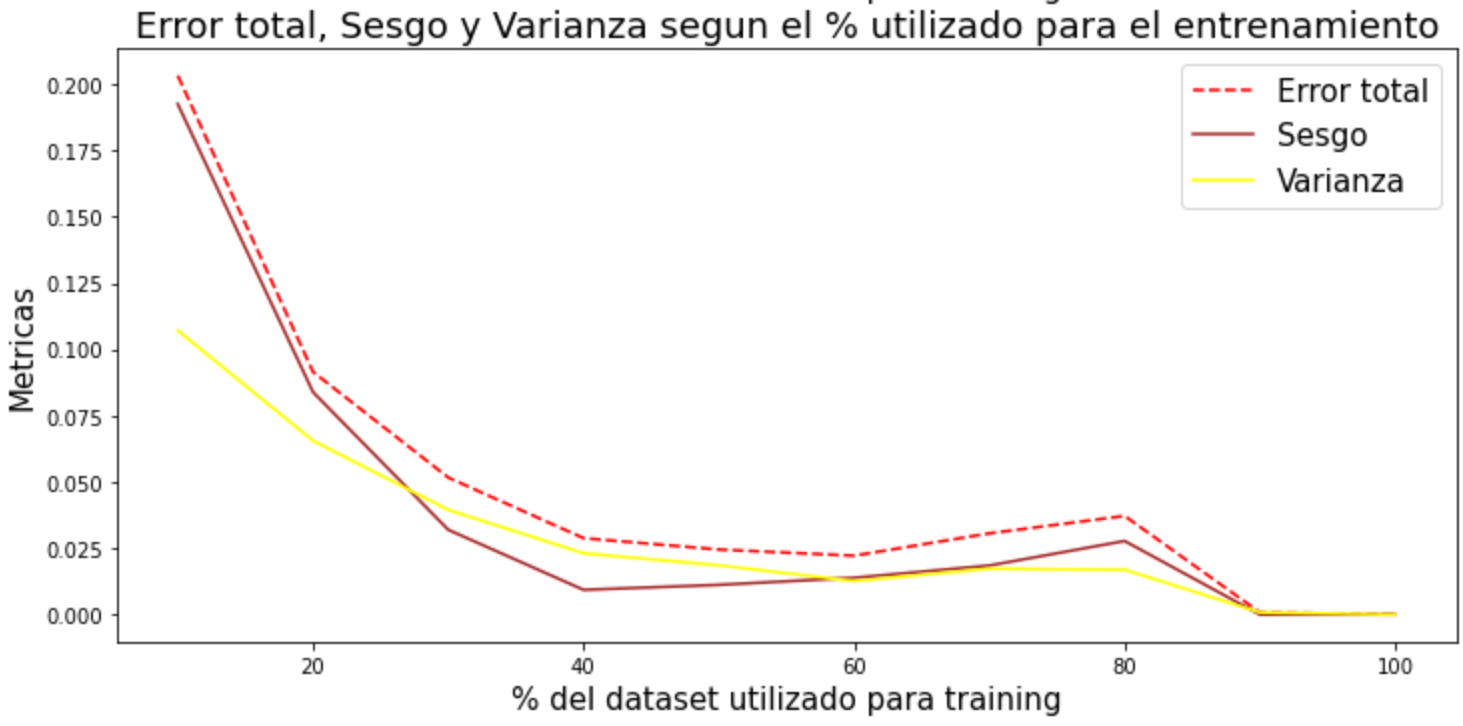
Modificando el parametro de random_state

Para eliminar la posibilidad de que el random_state este causando anomalias en los resultados, se hara el mismo procedimiento pero con un random_state diferente

- En este caso se utilizara 123 en vez de 42

(No hay una razon especifica por la cual usar 123, para mantener los resultados deterministas, simplemente se tiene que asignar un random_state manualmente)





Sorprentemente la anomalia que apenas y era notable utilizando un random_state de 42, con un random_state de 123 se vuelve aun mas evidente. Podemos observar como alrededor del 60%, el error, el sesgo y la varianza empiezan a subir y la precision empieza a bajar. Esto signfica que podemos excluir el factor de la aleatoridad como la causa de la anomalia en estas metricas ya que aunque se vuelve mas evidente, no es exclusivo a un solo random_state, solo puede ser mas prevalente en uno que en otro.
En cuanto a por que a partir del 90% la precision sube a 100% y el error, sesgo y varianza bajan a casi cero, se debe por lo mencionado anteriormente, los resultados estan inflados ya que no hay suficientes datos para probar la precision del modelo correctamente.

Conclusiones acerca del % de entrenamiento y prueba

Debido a lo descubierto, se utilizara a partir de este punto un split de 60-40, ya que aunque la precision bajo ligeramente comparada a la del 50%, el error y varianza tambien bajaron, el sesgo y varianza tienen casi el mismo valor y el error se encuentra en lo mas bajo posible siempre y cuando se excluyan los resultados del 90% y 100%.
Asimismo en las graficas se muestra claramente como en los porcentajes bajos se tiene un sesgo muy alto, indicando underfitting. Observando el otro extremo a partir del 80%, si tuvieramos mas datos para probar correctamente, la varianza probablemente empezaria a incrementar ligeramente indicando un poco de overfitting.

Parametros del modelo

Ya que se sabe cual es el mejor split de entrenamiento y prueba para el dataset se empezara a probar con diferentes parametros del modelo con el proposito de reducir aun mas la varianza y el sesgo e incrementar la precision simultaneamente.
Para lograr esto, vamos a concentrarnos en 4 parametros del modelo de regresion logistica de Sklearn:

- solver : Algoritmo utilizado en la optimización
- C : Inverso de la fuerza de regularización
- max_iter : Número máximo de iteraciones necesarias para que los solucionadores converjan
- penalty : Especificar la norma del penalty

Con el proposito de facilitar la busqueda de estos parametros, se hara uso de un GridSearchCV para encontrar la mejor combinacion de parametros

```
Mejores parametros: {'C': 0.00026366508987303583, 'max_iter': 100, 'penalty': 'none', 'solver': 'sag', 'ver
bose': 0}
```

Tras correr el GridSearchCV, este nos indica que, de los parametros que le dimos para probar, la mejor combinacion es la siguiente:

- solver : newton-cg
- C : 0.08858667904100823
- max_iter : 100
- penalty : l2

Utilizando estos parametros para sacar la precision, ya podemos observar que hubo un ligero aumento en comparacion a la precision inicial del modelo base.

Resultados y Conclusiones

Tras obtener los supuestos mejores parametros para el modelo, se realizara una comparativa con el modelo base con el que empezamos

```
LogisticRegression(C=0.08858667904100823, solver='newton-cg')
```

		base	nuevo
0	Precision	0.981481	0.986111
1	Error total esperado promedio	0.018426	0.021944
2	Sesgo promedio	0.018519	0.013889
3	Varianza promedio	0.011852	0.014444

Como se puede observar, se logro una ligera mejora en la precision, subio alrededor de un 0.5%, como se menciono al inicio del analisis, el modelo ya tiene una precision bastante alta y dificilmente se iba a lograr una mejora signficativa. Sin embargo, cabe mencionar que aun que se obtuvo una mayor precision sobre los datos de prueba y el sesgo haya bajado, el error total esperado subio, y la varianza tambien. Siguen estando signficativamente bajos, lo que no perjudicara tanto al rendimiento del modelo, pero para los propositos de este analisis, se puede considerar como un ligero sacrificio el incremento en el error y la varianza para obtener mejor precision dependiendo del caso.