Sebastian Rodriguez Salinas
A00827463
Sep 9, 2022

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

Intro

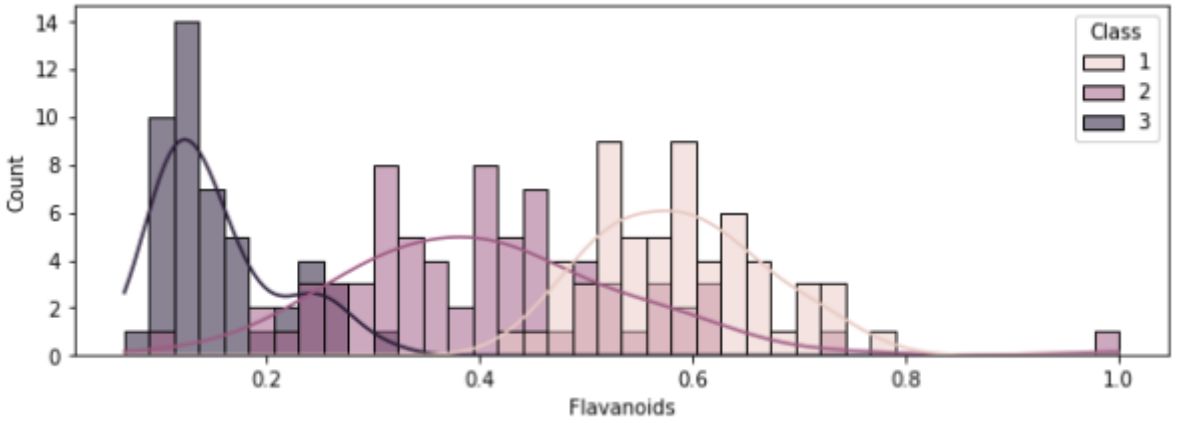
Para llevar a cabo este reporte, estaré considerando la elaboración de la práctica en la que se implementó el modelo de "*Neural Network Multi-layer Perceptron classifier*" de la librería Sklearn. Ahora bien, para implementar este modelo se hizo uso de el dataser "*wine.data*" el cual contiene información de 178 vinos diferentes. Entre los atributos se encuentran campos como:

1. La clase a la que pertenece (1, 2, 3)
2. Porcentaje de alcohol
3. Ácido málico
4. Ceniza
5. Alcalinidad de ceniza
6. Magnesio
7. Fenoles totales
8. Flavonoides
9. Fenoles no flavonoides
10. Proantocianidinas
11. Intencidad de color
12. Matiz
13. OD280/OD315 de vinos diluidos
14. Prolina

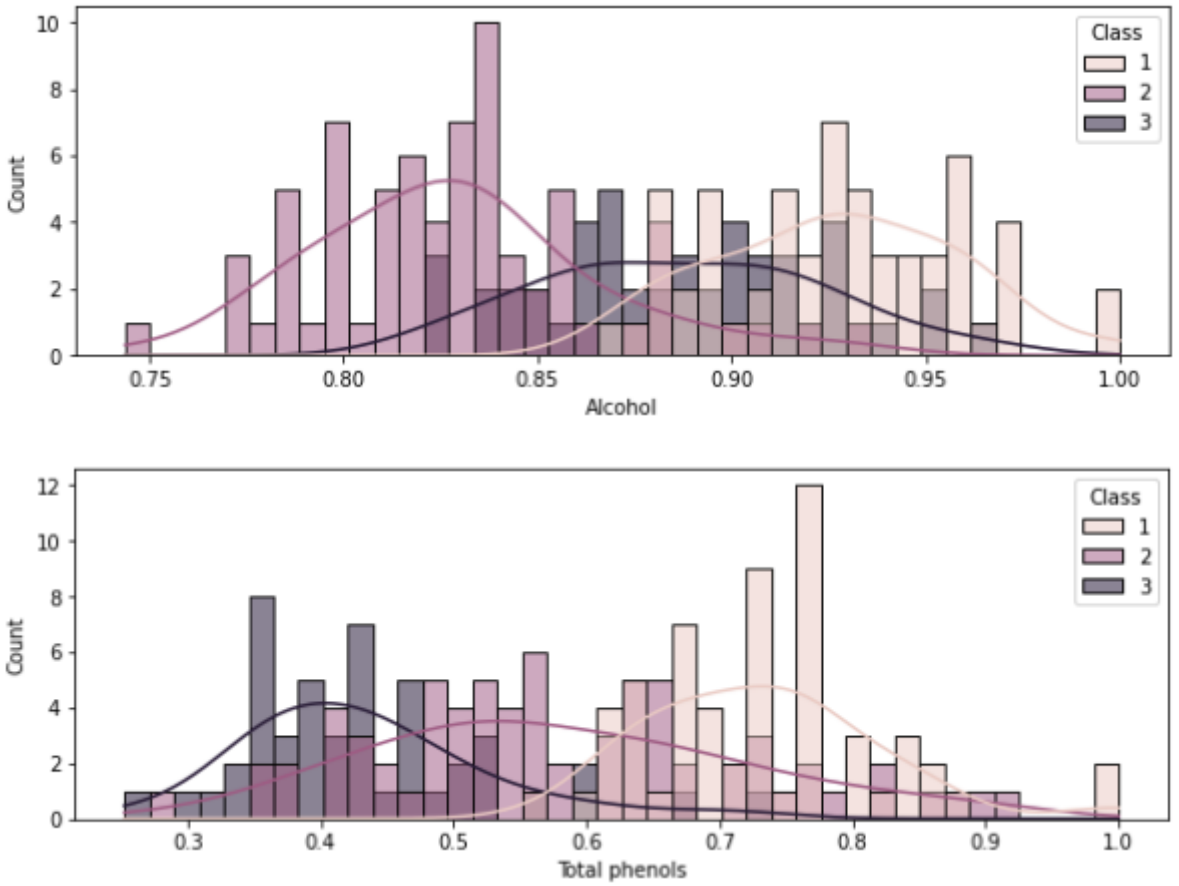
Lo que se busca con este modelos y este dataset es entrenar un grupo de datos para poder predecir la clase a la que el vino pertenece con base a los diferentes atributos. Dicho esto, antes de empezar es importante realizar una limpieza y configuración de datos para permitirnos correr el modelo de sklearn de manera correcta.

Visualización de datos *Mejora*

Es importante entender bien los datos, por lo que utilizar herramientas de visualización nos pueden ser de gran ayuda.



Si tomamos en cuenta la gráfica presentada aquí arriba, podemos identificar que los “Flavanoids” son un gran indicador que nos ayuda a clasificar la clase en la que pertenece un vino. Se puede visualizar que la mayor cantidad de vinos clasificados como clase 3 tienen un puntaje menor a 0.2. mientras que la mayor cantidad de vinos clasificados como clase 1 tienen un puntaje mayor a 0.5, dejando a los demás clasificados como clase 2 entremedio. Se puede ver en la gráfica la líneas de tendencias cuadráticas de las tres clases, el que estén separadas como en la gráfica indican que la variable tiene un gran impacto en la clasificación del vino.



Ahora bien, como mencionamos anteriormente, el la gráfica de “Flavanoids”, al simplemente ver estas otras dos gráficas, podemos también identificar que la variable de “Alcohol” y de “Total phenols” van a tener un gran impacto en la clasificación de clase. A simple vista, el utilizar únicamente estas tres variables dentro de nuestro modelo, no daría una buena predicción final. Sin embargo es importante considerar las otras variables que aunque puedan tener un efecto más ligero, nos ayuda a ajustar de mejor manera nuestro modelo y obtener una mayor precisión en nuestro modelo y generar una mejor predicción al final.

Modelo Parcial *Mejora*

Conjunto de prueba y un conjunto de validación

```
remove = ["Class","Malic acid", "Ash", "Alcalinity of ash", "Magnesium", "Nonflavanoid phenols", "Proanthocyanins",
"Color intensity", "Hue", "OD280/OD315 of diluted wines", "Proline"]
y = ['Class']
x = list(set(list(df.columns))-set(remove))
df[x] = df[x]/df[x].max()
```

Utilizamos este código de arriba para dos cosas. Primero separar la variable dependiente de "Class" la que buscamos predecir, y todos los otros campos en la variable independiente. En este caso buscamos correr un modelo parcial, por lo que estamos removiendo de las variables independientes todos los campos y dejando únicamente "Flavanoids", "Alcohol" y "Total phenols".Teniendo estos segmentos identificados corremos nuestro modelo.

Comenzamos, teniendo ya nuestros datos configurados el primer paso es separarlos en datos de entrenamiento y en datos de prueba. Para hacer esto utilizamos una función de Sklearn "*train_test_split*" la cual nos permite separar nuestros datos en dos grupos, de entrenamiento y de prueba.

```
X_train, X_test, y_train, y_test = train_test_split(df[x].values, df[y].values, test_size=0.2, random_state=100)
```

El beneficio de utilizar esta función es que podemos definir es que podemos definir cuantos datos queremos para entrenamiento y cuántos queremos para pruebas. Pero lo más importante aún es que la separación de los datos se realiza con un estado aleatorio que nos da las muestras aleatorias para ambos grupos, lo que es lo más correcto de hacer.

Grado de sesgo, grado de varianza y ajuste del modelo.

```
Resumen de datos de entrenamiento
[[49  2  0]
 [ 5 47  4]
 [ 0  1 34]]
```

	precision	recall	f1-score	support
1	0.91	0.96	0.93	51
2	0.94	0.84	0.89	56
3	0.89	0.97	0.93	35
accuracy			0.92	142
macro avg	0.91	0.92	0.92	142
weighted avg	0.92	0.92	0.91	142

```
Resumen de datos de prueba
[[ 8  0  0]
 [ 1 11  3]
 [ 0  5  8]]
```

	precision	recall	f1-score	support
1	0.89	1.00	0.94	8
2	0.69	0.73	0.71	15
3	0.73	0.62	0.67	13
accuracy			0.75	36
macro avg	0.77	0.78	0.77	36
weighted avg	0.75	0.75	0.75	36

Después de 999 iteraciones de entrenamiento con el modelo MLP, con los datos presentados arriba podemos concluir lo siguiente:

- Ambos datos (entrenamiento y prueba) tienen un nivel de sesgo medio dado f1-score, el que representa el porcentaje de precisión del modelo. Siendo un 0.91(91%) para los datos de entrenamiento, y un 0.75(75%) para los datos de prueba.
- Ambos datos (entrenamiento y prueba) tienen un nivel de varianza medio alto dado f1-score, el que representa el porcentaje de precisión del modelo. Siendo un 0.92(92%) para los datos de entrenamiento, y un 0.75(75%) para los datos de prueba.
- Teniendo esto en mente, y al considerar las matrices de confusión, podemos ver que el ajuste del modelo es underfit. Esto dado que en las matrices de confusión se representan múltiples valores fuera de lugar. Para los datos de entrenamiento se presentan 10/140 datos atípicos en la matriz, mientras que para los datos de prueba, se presentan 9/27 datos fuera de lugar, lo que nos da la precisión de 0.75

Ahora bien, mientras que este modelo no nos da un valor adecuado de precisión con el cual hacer predicciones, considerando que solo se implementaron 3 de las variables, es muy buena predicción. Dicho esto, es importante considerar por más mínimas que sean las demás variables ya que nos proporcionarán un mayor grado de confianza en nuestro modelo.

Modelo completo

Conjunto de prueba y un conjunto de validación

```
y = ['Class']
x = list(set(list(df.columns))-set(y))
df[x] = df[x]/df[x].max()
```

En este caso, en comparación con el modelo anterior, si consideramos el resto de las variables para nuestro modelo. Teniendo todas las variables independientes, las procesamos para que estén los datos en rango de 0 - 1, lo que nos permite establecer un estándar entre todas las variables y hacer las relaciones e interpretación entre estas más adecuadas.

De la misma manera que antes, utilizamos la función de Sklearn "*train_test_split*" la cual nos permite separar nuestros datos en dos grupos, de entrenamiento y de prueba.

```
X_train, X_test, y_train, y_test = train_test_split(df[x].values, df[y].values, test_size=0.2, random_state=100)
```

Grado de sesgo, grado de varianza y ajuste del modelo.

```
Resumen de datos de entrenamiento
[[51  0  0]
 [ 0 56  0]
 [ 0  0 35]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        51
           2       1.00      1.00      1.00        56
           3       1.00      1.00      1.00        35

    accuracy                           1.00       142
   macro avg       1.00      1.00      1.00       142
weighted avg       1.00      1.00      1.00       142


Resumen de datos de prueba
```

```
[[ 8  0  0]
 [ 0 15  0]
 [ 0  1 12]]
```

	precision	recall	f1-score	support
1	1.00	1.00	1.00	8
2	0.94	1.00	0.97	15
3	1.00	0.92	0.96	13
accuracy			0.97	36
macro avg	0.98	0.97	0.98	36
weighted avg	0.97	0.97	0.97	36

Después de 999 iteraciones de entrenamiento con el modelo MLP, con los datos presentados arriba podemos concluir lo siguiente:

- Ambos datos (entrenamiento y prueba) tienen un nivel de sesgo bajo dado f1-score, el que representa el porcentaje de precisión del modelo. Siendo un 1.00(100%) para los datos de entrenamiento, y un 0.97(97%) para los datos de prueba.
- Ambos datos (entrenamiento y prueba) tienen un nivel de varianza bajo dado f1-score, el que representa el porcentaje de precisión del modelo. Siendo un 1.00(100%) para los datos de entrenamiento, y un 0.97(97%) para los datos de prueba.
- Teniendo esto en mente, y al considerar las matrices de confusión, podemos ver que el ajuste del modelo es fitt. Esto dado que en las matrices de confusión se representan pocos valores fuera de lugar. Para los datos de entrenamiento no se presenta ningún dato atípico en la matriz, mientras que para los datos de prueba, solo se presenta un dato fuera de lugar proveniente de la agrupación para la clase 2, lo que nos da la precisión de 0.97.

Técnicas de regularización para mejorar el desempeño del modelo

Mientras que considero que se puede realizar una buena implementación del modelo con los datos de vinos, hay limitantes, o mejoras que se pudieran realizar para obtener una mayor presión dentro del modelo.

Un factor limitante considero es la cantidad de datos. 178 datos es buen número para lo que se realizó en esta práctica, sin embargo para un modelo realizado en el que se busca mayor precisión, sería ideal tener un dataset aún mayor que nos permita hacer el entrenamiento del modelo a cu mayor potencial.

Asimismo, sería importante considerar que se pueden presentar datos extremadamente atípicos, los que pueden afectar la precisión del modelo. Para mejorar el desempeño sería apropiado eliminar o reemplazar los datos que son extremadamente atípicos con el propósito de que no afecte de manera significativa nuestra predicción final.

Conclusión

En conclusión, el modelo implementado de "*Neural Network Multi-layer Perceptron classifier*" fue todo un éxito. Fuimos capaces de implementar el modelo con un 97% de precisión para la predicción de datos. Hay mejoras no tengo dudas, pero también fui capaz de aprender muchas cosas de las librerías y los modelos de predicción con base a esto.