Navaja de Ockham en Machine Learning

Alumno: Eddy Kennedy Mamani Hallasi

18/02/2025

Contents

1 Repositorio

Para aquellos interesados en explorar el código y el dataset utilizado en este informe, he creado un repositorio donde pueden acceder a todo el material. A continuación se encuentran los enlaces relevantes:

Código Python y Dataset Local: El código utilizado para aplicar la **Navaja de Ockham** en machine learning, junto con el dataset local, están disponibles en el siguiente repositorio de GitHub:
`https://github.com/MetodosDeOptimizacion/Navaja_de_Ockham.git`

Video Explicativo: Para una mejor comprensión del principio de la **Navaja de Ockham** y su aplicación, pueden ver este video en YouTube:
https://www.youtube.com/watch?v=Jcw5KBZ5Y28

Estos recursos les permitirán profundizar en los detalles y entender mejor la aplicación práctica del principio de la **Navaja de Ockham**.

2 Introducción

La **Navaja de Ockham** es un principio filosófico que se puede aplicar a muchos campos, incluyendo el análisis de datos y el machine learning. Básicamente, nos dice que, cuando existen varias explicaciones posibles para un problema, la más sencilla es la mejor. En el contexto de machine learning, esto quiere decir que, cuando tenemos varios modelos que explican los datos de forma similar, el más simple debería ser el elegido. Este principio ayuda a evitar que el modelo se vuelva demasiado complejo y pierda su capacidad para generalizar, lo que nos lleva a problemas como el **sobreajuste**.

Este informe está enfocado en explicar cómo la **Navaja de Ockham** se aplica en la selección de modelos, y cómo evaluar los modelos usando conceptos como **sobreajuste**, **subajuste**, **validación** y **regularización**.

3 Sobreajuste y Subajuste

Antes de ver cómo aplicamos la Navaja de Ockham, es importante entender dos conceptos clave: **sobreajuste** (overfitting) y **subajuste** (underfitting).

3.1 Sobreajuste (Overfitting)

El sobreajuste ocurre cuando un modelo es demasiado complejo. Si un modelo tiene demasiados parámetros o es demasiado flexible, es posible que se ajuste demasiado bien a los datos de entrenamiento, capturando hasta el ruido o las fluctuaciones aleatorias que no tienen importancia. Esto hace que el modelo funcione muy bien en los datos de entrenamiento, pero tenga un rendimiento pésimo cuando se enfrenta a nuevos datos (de prueba).

Es como un estudiante que memoriza las respuestas sin entender el contenido. Cuando le cambian las preguntas, no sabe cómo resolverlas.

3.2 Subajuste (Underfitting)

El subajuste, por otro lado, ocurre cuando el modelo es demasiado simple. Si el modelo no tiene suficiente capacidad para capturar las relaciones entre las variables, no podrá explicar correctamente los datos, y por lo tanto, tendrá un mal rendimiento tanto en los datos de entrenamiento como en los datos de prueba.

Es como un estudiante que solo estudia lo más básico y no es capaz de resolver ni las preguntas fáciles.

4 Validación en Machine Learning

La **validación** es un paso importante para ver si nuestro modelo realmente sabe generalizar, es decir, si no solo ha aprendido los datos de entrenamiento sino que también puede hacer predicciones correctas sobre datos que no ha visto antes. Existen diferentes técnicas de validación, pero las más comunes son:

- **Hold-Out**: Los datos se dividen en dos partes: una para entrenamiento y otra para prueba. El modelo se entrena con los datos de entrenamiento y se evalúa con los datos de prueba.

- **K-Fold Cross Validation**: Los datos se dividen en k partes (o "folds"). El modelo se entrena $k-1$ veces con diferentes subconjuntos de los datos y se evalúa sobre el subconjunto restante. Esta técnica es más robusta que Hold-Out, ya que asegura que cada punto de datos se use tanto para entrenamiento como para evaluación.

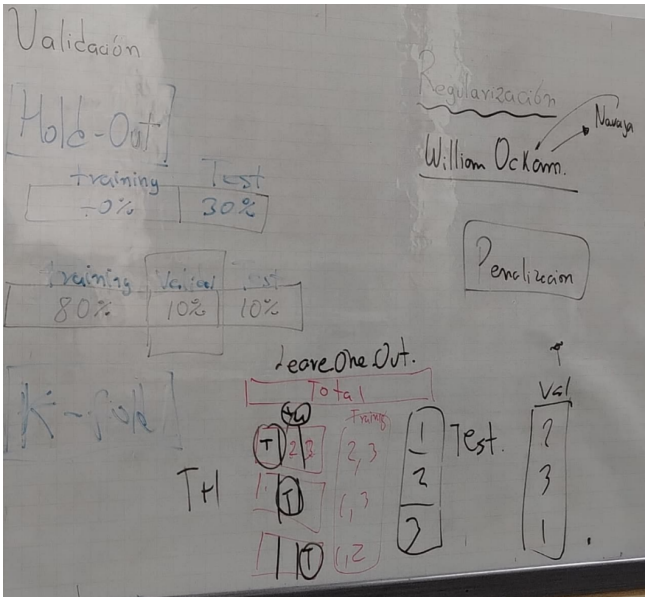

Figure 1: Ejemplo hecho en clases

5 Regularización

La **regularización** es una técnica que se usa para evitar el sobreajuste. Se agrega un término extra a la función de error del modelo para penalizar modelos demasiado complejos. Existen dos tipos comunes de regularización:

5.1 L1 Regularización (Lasso)

La **L1 regularización** añade una penalización proporcional al valor absoluto de los coeficientes del modelo. Esto puede hacer que algunos coeficientes sean exactamente cero, lo que lleva a la eliminación de algunas variables. Este tipo de regularización es útil cuando se tiene un gran número de variables y se desea realizar una *selección de variables*, eliminando las menos importantes.

Ejemplo: Imagina que estás construyendo un modelo para predecir el precio de una casa y tienes muchas características, como el tamaño, el número de habitaciones, el número de baños, la antigüedad, la ubicación, etc. Al aplicar la regularización L1, el modelo podría eliminar algunas de estas variables (es decir, hacer que sus coeficientes sean cero) si considera que no aportan mucho a la predicción. Esto ayuda a simplificar el modelo.

Un ejemplo simple de Lasso sería el siguiente:

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \lambda \sum_{i=1}^{n} |\beta_i|$$

Aquí, λ es el parámetro de regularización. A medida que λ aumenta, más coeficientes de β se hacen cero, lo que simplifica el modelo.

5.2 L2 Regularización (Ridge)

La **L2 regularización** añade una penalización proporcional al cuadrado de los coeficientes del modelo. A diferencia de Lasso, L2 no hace que los coeficientes sean exactamente cero, pero los reduce. Esto ayuda a evitar que los coeficientes crezcan demasiado, lo que puede causar que el modelo se ajuste demasiado a los datos de entrenamiento (sobreajuste).

Ejemplo: Imagina que estás construyendo el mismo modelo para predecir el precio de una casa. En lugar de eliminar algunas variables como lo hace Lasso, la regularización L2 reducirá el impacto de algunas características, especialmente aquellas que no son tan relevantes. Así, en lugar de hacer que los coeficientes sean cero, simplemente los hace más pequeños.

Un ejemplo simple de Ridge sería el siguiente:

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \lambda \sum_{i=1}^{n} \beta_i^2$$

En este caso, λ controla la penalización. A medida que λ aumenta, los coeficientes de β se hacen más pequeños, lo que ayuda a evitar el sobreajuste.

5.3 Diferencia entre L1 y L2

La principal diferencia entre L1 y L2 es cómo manejan los coeficientes del modelo. L1 (Lasso) puede eliminar completamente algunas variables, mientras que L2 (Ridge) solo reduce la magnitud de los coeficientes. Dependiendo de la situación, uno u otro puede ser más adecuado:

- Si tenemos un gran número de variables y creemos que algunas no son útiles, la regularización **L1 (Lasso)** es útil porque realiza una selección de características eliminando las variables menos importantes. - Si preferimos mantener todas las variables y solo queremos reducir su impacto, la regularización **L2 (Ridge)** es más adecuada.

En algunos casos, también se puede usar una combinación de ambas, conocida como **ElasticNet**, que combina las ventajas de ambos métodos.

Ambas técnicas ayudan a que el modelo sea más simple y, por lo tanto, menos propenso a sobreajustarse.

6 Aplicando la Navaja de Ockham

Para entender cómo funciona la **Navaja de Ockham** en machine learning, vamos a ver un ejemplo práctico. Supongamos que tenemos un dataset con características de casas y queremos predecir el **Precio** de las casas. El dataset tiene las siguientes características:

Tamaño (m^2)	**Habitaciones**	**Baños**	**Antigüedad (años)**	**Precio (S/.)**
100	3	2	10	200,000
150	4	3	5	300,000
80	2	1	20	150,000
120	3	2	8	250,000
200	5	4	2	450,000
90	2	1	15	180,000

Table 1: Dataset de casas con sus características y precios.

El objetivo es predecir el **Precio** de la casa basándonos en sus características.

6.1 Modelos

Tenemos dos modelos para predecir el precio:

1. **Modelo A (Simple)**: Este modelo utiliza solo el **Tamaño** de la casa para predecir el **Precio**.
2. **Modelo B (Complejo)**: Este modelo utiliza todas las características disponibles (**Tamaño**, **Habitaciones**, **Baños**, **Antigüedad**).

6.2 Evaluación de los Modelos

Vamos a evaluar ambos modelos utilizando el **Error Cuadrático Medio (MSE)**. Este indicador nos ayuda a ver qué tan bien predice el modelo comparado con los valores reales. La fórmula del MSE es la siguiente:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Donde:

- y_i es el precio real de la casa.
- $\hat{y}_i$ es el precio predicho por el modelo.
- n es el número total de casas.

6.3 Paso a Paso

6.3.1 1. Modelo A (Simple)

Supongamos que entrenamos el **Modelo A** con solo la característica **Tamaño** de las casas. Las predicciones de este modelo podrían ser:

- Predicciones del Modelo A:

$$\hat{y}_1 = 210,000, \quad \hat{y}_2 = 280,000, \quad \hat{y}_3 = 160,000, \quad \hat{y}_4 = 240,000, \quad \hat{y}_5 = 440,000, \quad \hat{y}_6 = 175,000$$

Ahora, calculamos el **MSE**:

$$MSE = \frac{1}{6}\left[(200,000 - 210,000)^2 + (300,000 - 280,000)^2 + (150,000 - 160,000)^2 + (250,000 - 240,000)^2 + (4\right.$$

Calculamos cada término y sumamos los errores:

$$MSE = \frac{1}{6}\left[100,000,000 + 400,000,000 + 100,000,000 + 100,000,000 + 100,000,000 + 25,000,000\right]$$

$$MSE = \frac{825,000,000}{6} = 137,500,000$$

6.3.2 2. Modelo B (Complejo)

Ahora, entrenamos el **Modelo B**, que utiliza todas las características disponibles (**Tamaño**, **Habitaciones**, **Baños**, **Antigüedad**). Las predicciones para este modelo podrían ser:

- Predicciones del Modelo B:

$$\hat{y}_1 = 200,500,000, \quad \hat{y}_2 = 290,000,000, \quad \hat{y}_3 = 145,000,000, \quad \hat{y}_4 = 245,000,000, \quad \hat{y}_5 = 445,000,000, \quad \hat{y}_6 = 1$$

Calculamos el **MSE** para el **Modelo B**:

$$MSE = \frac{1}{6}\left[(200,000,000 - 200,500,000)^2 + (300,000,000 - 290,000,000)^2 + (150,000,000 - 145,000,000)^2 +\right.$$

Realizamos los cálculos:

$$MSE = \frac{1}{6}\left[250,000,000 + 100,000,000 + 25,000,000 + 25,000,000 + 25,000,000 + 4,000,000\right]$$

$$MSE = \frac{429,000,000}{6} = 71,500,000$$

6.3.3 3. Comparación de los Modelos

- **Modelo A (Simple)**: MSE = 137,500,000
- **Modelo B (Complejo)**: MSE = 71,500,000

El **Modelo B** tiene un MSE más bajo, lo que indica que en términos de precisión en los datos de entrenamiento, es mejor. Sin embargo, el **Modelo B** es más complejo, ya que utiliza más características. Aquí es donde entra el principio de la **Navaja de Ockham**.

6.4 Usando la Navaja de Ockham

El principio de la **Navaja de Ockham** nos dice que, cuando dos modelos tienen un rendimiento similar, debemos preferir el modelo más simple. Aunque el **Modelo B** tiene un MSE más bajo, el **Modelo A** es más simple y tiene menos posibilidades de sobreajustarse a los datos. Además, la diferencia en el MSE no es tan grande como para justificar la complejidad adicional del **Modelo B**.

6.5 Decisión

Aunque el **Modelo B** tiene un MSE más bajo, el **Modelo A** es preferido por su simplicidad. La **Navaja de Ockham** nos dice que debemos elegir el modelo más simple que aún sea capaz de hacer buenas predicciones. Así, evitamos el sobreajuste y mantenemos un modelo más interpretable y generalizable.

7 Conclusión

La **Navaja de Ockham** nos enseña que, cuando tenemos varias opciones para resolver un problema, siempre es mejor elegir la más simple. En machine learning, esto significa que debemos preferir modelos sencillos, ya que tienden a generalizar mejor y a evitar el sobreajuste. Aunque un modelo más complejo

pueda dar mejores resultados en los datos de entrenamiento, el modelo más simple suele ser más efectivo a largo plazo, ya que es más fácil de interpretar y se adapta mejor a nuevos datos.

En resumen, aplicar la **Navaja de Ockham** nos ayuda a escoger modelos eficientes, que no solo funcionan bien en los datos de entrenamiento, sino también en datos nuevos, asegurando un buen rendimiento sin complicar de más el modelo.

8 Anexo: Manual de Uso del Código Python

En esta sección se detalla cómo utilizar el código Python para ejecutar el análisis y aplicar el principio de la **Navaja de Ockham** con un dataset.

8.1 Requisitos previos

8.1.1 Instalación de librerías necesarias

Para ejecutar este código, debes tener instaladas las siguientes librerías de Python:

- **pandas**: Para manejar y analizar los datos.
- **scikit-learn**: Para crear modelos de regresión y dividir los datos.
- **matplotlib** (opcional): Para visualizar los resultados, si lo deseas.

Puedes instalar estas librerías utilizando `pip`. Abre una terminal y ejecuta el siguiente comando:

Instalación de librerías

```
pip install pandas scikit-learn matplotlib
```

8.1.2 Descarga del dataset

El código requiere un archivo de datos en formato **.xlsx**. Este archivo debe contener las siguientes columnas:

- `site_latitude`: La latitud del sitio.
- `site_longitude`: La longitud del sitio.
- `site_elevation`: La elevación del sitio (esta columna se usará como la variable objetivo en el análisis).

8.1.3 Preparación del archivo

Asegúrate de que tu archivo de datos esté en formato **.xlsx** y sea accesible desde el directorio donde ejecutarás el código.

8.2 Uso del Código Python

8.2.1 Carga del archivo de datos

El código lee el archivo de datos utilizando la librería `pandas`. Asegúrate de que el archivo de datos esté en el formato adecuado con las columnas mencionadas anteriormente. Además, el archivo debe estar ubicado en la misma ruta que se especifica en el código o debes modificar la ruta para que coincida con la ubicación real del archivo en tu sistema.

Carga del dataset

```
import pandas as pd

# Ruta del archivo, asegurate de que el archivo este en la ubicacion correcta
tfile_path = 'C:/Users/VICTUS/Documents/Ockham/systems.xlsx'

df = pd.read_excel(file_path)
print(df.head())  # Muestra las primeras filas del dataset para verificar la
```

Si el archivo se carga correctamente, deberías ver las primeras filas del dataset impresas en la consola. En caso de error, verifica que la ruta del archivo sea correcta y que el archivo esté en formato `.xlsx` válido.

8.2.2 Ejemplo de ejecucion del codigo

Aplicacion > ockham.py > ...

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Leer el archivo Excel (.xlsx)
file_path = 'C:/Users/VICTUS/Documents/Ockham/systems.xlsx'  # Asegúrate de poner la ruta correcta
df = pd.read_excel(file_path)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
4        39.7420       -105.1727         1994.7

Información del dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   site_latitude   158 non-null    float64
 1   site_longitude  158 non-null    float64
 2   site_elevation  158 non-null    float64
dtypes: float64(3)
memory usage: 3.8 KB
None

Estadísticas descriptivas:
       site_latitude  site_longitude  site_elevation
count     158.000000      158.000000      158.000000
mean       32.891884      -90.033817      307.187342
std         5.179995       11.798025      600.791345
min        27.714000     -121.774200        2.000000
25%        29.928800      -90.127300       10.000000
50%        29.980900      -90.002300       10.000000
75%        39.413150      -82.650200      137.750000
max        46.670400      -68.017800     1994.700000

Correlación entre las variables numéricas:
                site_latitude  site_longitude  site_elevation
site_latitude        1.000000       -0.179350        0.593934
site_longitude      -0.179350        1.000000       -0.667613
site_elevation       0.593934       -0.667613        1.000000

Error Cuadrático Medio (MSE) - Modelo Simple: 132663.46694487857

Coeficientes del modelo simple:
[ 59.10549736 -29.44815712]

Error Cuadrático Medio (MSE) - Modelo Complejo: 1.5726600967696246e-25

Coeficientes del modelo complejo:
[-4.95247418e-14 -9.99200722e-16  1.00000000e+00]
```

Figure 2: Ejemplo del codigo ejecutado