Modelo para la predicción de la valoración de las startups que han llegado a convertirse en unicornio

Pablo Hortal Llera

26 de julio de 2023

THE BRIDGE UNIVERSITY

Índice

Primer análisis de los datos

Importación de librerías y datos

En estas primeras líneas, estoy llevando a cabo dos pasos esenciales en cualquier proyecto de análisis de datos.

Importación de librerías

Estoy importando las librerías que utilizaré en mi análisis. Cada una de estas tiene un propósito específico:

- **numpy:** Me ofrece soporte para trabajar con arrays y matrices de gran tamaño, además de una gran colección de funciones matemáticas para operar con estos elementos.

- **pandas:** Me proporciona estructuras de datos y herramientas de análisis de datos flexibles y fáciles de usar. Es particularmente útil para manipular y analizar datos numéricos y series temporales.

- **scipy.stats:** Este es un submódulo de la librería SciPy que proporciona un gran número de distribuciones de probabilidad y una creciente biblioteca de funciones estadísticas.

- **seaborn:** Es una librería de visualización de datos en Python basada en matplotlib. Me ofrece una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

- **matplotlib.pyplot:** Me ofrece una forma de graficar datos y crear figuras de manera similar a la que ofrece MATLAB.

- **os:** Me proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo, como leer o escribir en el sistema de archivos.

Carga de datos

Cargo mi conjunto de datos 'World_Wide_Unicorn_Startups.csv' en un DataFrame de pandas llamado unicornios. Un DataFrame es una estructura de datos bidimensional de tamaño variable y potencialmente heterogénea. Es una de las estructuras de datos más comúnmente utilizadas en pandas para el análisis y la manipulación de datos. La ruta del archivo la paso como argumento a la función pd.read_csv(), que lee un archivo csv y lo convierte en DataFrame.

Vista preliminar de los datos

Después de cargar el conjunto de datos, es crucial obtener una visión preliminar de cómo se presentan los datos. Para hacer esto, estoy utilizando el método .head(20) del DataFrame unicornios, que devuelve las primeras 20 filas del DataFrame.

Aquí el resumen de lo que representa cada columna:

- **Company:** El nombre de la empresa.
- **Valuation:** La valoración de la empresa en miles de millones de dólares.
- **Date:** La fecha en que la empresa alcanzó su estatus de unicornio (valoración de más de 1 billón de dólares).
- **Country:** El país donde está ubicada la empresa.
- **City:** La ciudad en la que se encuentra la empresa.
- **Industry:** La industria a la que pertenece la empresa.
- **Investors:** Los inversores de la empresa.
- **Year:** El año en que la empresa alcanzó su estatus de unicornio.
- **Month:** El mes en que la empresa alcanzó su estatus de unicornio.
- **Day:** El día del mes en que la empresa alcanzó su estatus de unicornio.
- Este es un paso significativo en la exploración y análisis de datos, ya que me permite entender la estructura de mis datos. En particular, puedo ver qué tipo de información contiene cada columna y empezar a pensar en cómo podría utilizarla en mi análisis.

Información sobre los datos

El método .info() se usa para obtener un resumen conciso del DataFrame. Esto incluye la cantidad de entradas no nulas, el tipo de datos de cada columna y una indicación de cuánta memoria está utilizando el DataFrame.

En este caso, el DataFrame unicornios tiene 936 entradas, que van desde el índice 0 al 935. Cada una de las columnas tiene la siguiente información:

- **Company:** 936 entradas no nulas de tipo object (cadenas de texto).
- **Valuation:** 936 entradas no nulas de tipo float64 (números con decimales).
- **Date:** 936 entradas no nulas de tipo object (cadenas de texto).
- **Country:** 936 entradas no nulas de tipo object (cadenas de texto).
- **City:** 921 entradas no nulas de tipo object (cadenas de texto). Aquí hay algunos valores faltantes ya que debería haber 936 entradas.
- **Industry:** 936 entradas no nulas de tipo object (cadenas de texto).
- **Investors:** 936 entradas no nulas de tipo object (cadenas de texto).
- **Year:** 936 entradas no nulas de tipo int64 (números enteros).
- **Month:** 936 entradas no nulas de tipo int64 (números enteros).
- **Day:** 936 entradas no nulas de tipo int64 (números enteros).

Este paso es crucial para entender la calidad de los datos con los que estás trabajando. En particular, puedes ver si hay valores faltantes que debas abordar y si los tipos de datos son los esperados para cada columna. También te da una idea de si es necesario cambiar los tipos de datos de las columnas para su posterior procesamiento o análisis.

Limpieza de los datos

En este paso, estás eliminando las columnas 'Company' y 'Date' del DataFrame 'unicornios' utilizando el método drop(). Esto es una parte del proceso de limpieza de datos y preparación para el análisis.

La columna 'Company' se elimina porque el nombre de la empresa no es relevante para la predicción del valor de la empresa. En el contexto del aprendizaje automático, es un atributo no predictivo. Al eliminar esta columna, simplificas el modelo al no tener que tratar con variables categóricas de alta cardinalidad, que podrían requerir métodos avanzados como la codificación one-hot, la codificación de frecuencias, la incorporación de entidades o el tratamiento de categorías raras.

La columna 'Date' se elimina porque ya tienes la información de la fecha desglosada en las columnas 'year', 'month' y 'day'. Es un ejemplo de ingeniería de características, en la que se extraen características más útiles de los datos existentes. Mantener la columna 'Date' en su formato original no proporcionaría ningún beneficio adicional al modelo y solo haría que el conjunto de datos sea más difícil de manejar.

Este proceso simplifica el DataFrame y lo prepara para el análisis posterior. Es importante notar que las decisiones sobre qué columnas eliminar dependen del conocimiento específico del dominio, de los datos y de los objetivos de tu análisis o modelado.

Análisis univariante

Al llamar al método describe() en el DataFrame 'unicornios', estás generando estadísticas descriptivas que resumen la tendencia central, la dispersión y la forma de la distribución del conjunto de datos, excluyendo los valores NaN.

Estas estadísticas incluyen lo siguiente para cada columna:

- **count:** el número de filas no nulas, que puede ser útil para identificar columnas con valores faltantes.
- **mean:** la media de los valores, que es útil para obtener un sentido de la ubicación central de los datos.
- **std:** la desviación estándar de los valores, que es útil para entender la cantidad de variabilidad o dispersión en los datos.
- **min:** el valor mínimo.
- **25%:** el primer cuartil de los datos, donde el 25% de los puntos de datos son más bajos.
- **50%:** la mediana de los datos, el valor medio de la distribución.
- **75%:** el tercer cuartil de los datos, donde el 75% de los puntos de datos son más bajos.
- **max:** el valor máximo.

Este es un paso importante del Análisis Exploratorio de Datos (EDA) ya que te permite obtener un entendimiento rápido de la distribución de tus datos.

En este fragmento de código, estoy generando un histograma de la columna "Valuation", que representa la valoración en miles de millones de dólares de las empresas unicornio. Este gráfico muestra la distribución de los datos, es decir, cómo se distribuyen los valores de "Valuation" en todo el conjunto de datos. Al utilizar una escala logarítmica, es más fácil visualizar y entender la distribución de los datos cuando hay una gran variabilidad o diferencias extremas en los valores, como en este caso.

La función plt.hist() de la biblioteca Matplotlib se utiliza para generar el histograma, y los parámetros "bins=30" y "log=True" especifican que el histograma debe tener 30 barras y que se debe usar una escala logarítmica, respectivamente.

Es importante destacar que un histograma es una forma efectiva de visualizar la distribución de una variable numérica y puede ayudar a identificar tendencias, valores atípicos y la presencia de sesgos en los datos.

El título del gráfico ("Distribución de la Valoración (escala logarítmica)"), así como las etiquetas en los ejes x e y ("Valoración (miles de millones de dólares)" y "Cantidad de Startups", respectivamente) ayudan a identificar claramente lo que se está mostrando.

En este caso, observar la forma y los datos del histograma puede proporcionarme información sobre la concentración de las valoraciones de las empresas unicornio. Si, por ejemplo, hay un pico en el extremo inferior del histograma, eso indicaría que la mayoría de las empresas tienen valoraciones relativamente bajas, con pocas empresas alcanzando valoraciones extremadamente altas.

Este fragmento de código genera un gráfico de barras que muestra los 10 países con la mayor cantidad de startups unicornio.

La función value_counts() se utiliza en la columna 'Country' para contar cuántas veces aparece cada país en el conjunto de datos. La función nlargest(10) selecciona los 10 países con la mayor cantidad de startups unicornio. Finalmente, plot(kind='bar') genera un gráfico de barras a partir de estos datos.

El gráfico de barras es útil para comparar la cantidad de startups unicornio entre diferentes países. El eje y muestra la cantidad de startups unicornio y el eje x los nombres de los países.

Al visualizar esta información, podemos obtener información sobre qué países son los más predominantes en términos de producción de startups unicornio. Por ejemplo, si Estados Unidos aparece con la barra más alta, esto indicaría que tiene la mayor cantidad de estas empresas. Este tipo de análisis es útil para comprender dónde se concentra la actividad de las startups unicornio a nivel mundial.

Este fragmento de código genera un gráfico de caja (boxplot) de la valoración de las startups unicornio.

Un gráfico de caja es un método estandarizado para representar la distribución de los datos y es especialmente útil para comparar la distribución de los datos entre varios grupos. En este caso, nos muestra cómo se distribuyen los valores de las valoraciones de las startups unicornio.

En el gráfico, la caja central representa el rango intercuartil (es decir, desde el primer cuartil hasta el tercer cuartil), con la línea en el medio de la caja que representa la mediana (segundo cuartil). Los "bigotes" que se extienden desde la caja representan la dispersión de los datos, concretamente hasta 1.5 veces el rango intercuartil. Los puntos más allá de los bigotes representan valores atípicos.

Al analizar el gráfico, podemos observar cosas como:

- Si la mediana está centrada dentro de la caja, lo que sugiere una distribución simétrica de los datos.
- Si la caja es muy larga o muy corta, lo que indica una alta o baja variabilidad en los datos.
- Si hay muchos valores atípicos, lo que sugiere que hay algunas startups con valoraciones que se desvían significativamente del resto.

Este código genera un gráfico de barras de la cantidad de startups unicornio en función de la industria, considerando solamente las 10 industrias con la mayor cantidad de startups unicornio.

En este gráfico, cada barra representa una industria y su altura es proporcional a la cantidad de startups unicornio en esa industria. Las industrias están ordenadas de mayor a menor en función de la cantidad de startups unicornio. En este caso las más importantes son Fintech, Servicios de software y e-comerce.

Al analizar este gráfico, podemos observar cosas como:

- Cuáles son las industrias con mayor cantidad de startups unicornio. Esto puede darte una idea de en qué industrias se están generando las valoraciones más altas.

- Si hay alguna industria que destaca significativamente sobre las demás, lo que podría indicar una tendencia en la generación de startups unicornio.
- Si hay muchas industrias con una cantidad similar de startups unicornio o si hay algunas industrias con muchas más startups que las demás.

Preprocesamiento de Datos (Data Preprocessing):

Este código verifica si hay algún valor nulo en los datos. En Python, la función isna().sum() de pandas se utiliza para contar el número total de valores nulos o faltantes en cada columna de un DataFrame.

Los valores nulos o faltantes en los datos pueden ser un problema, ya que pueden afectar los resultados de los análisis y los modelos de Machine Learning. Por lo tanto, es importante identificar y manejar adecuadamente estos valores.

En este caso, podemos observar que hay 15 valores nulos en la columna 'City'. El resto de las columnas no tienen valores nulos.

Dependiendo de los requerimientos específicos de tu análisis o del modelo de Machine Learning que estés utilizando, podrías querer llenar estos valores nulos con un valor específico, eliminar las filas o las columnas que los contienen, o utilizar alguna otra estrategia para manejar estos valores.

En esta línea de código, estoy imputando los valores faltantes de la columna 'City' con la moda. La moda es el valor que aparece con mayor frecuencia en un conjunto de datos. En Python, la función mode() de pandas se utiliza para calcular la moda de una columna.

Estoy utilizando el método fillna() para rellenar los valores nulos en la columna 'City' con el valor de la moda. El argumento inplace=True se utiliza para realizar los cambios directamente en el DataFrame original.

Imputar los valores faltantes con la moda puede ser una buena opción cuando se está trabajando con datos categóricos y se desea mantener la distribución de los datos. Sin embargo, hay que tener en cuenta que esta estrategia puede introducir sesgos si la cantidad de valores nulos es alta en comparación con el tamaño total del conjunto de datos.

Estoy verificando si aún quedan valores nulos en la columna 'City' después de haber realizado la imputación. Para hacer esto, estoy utilizando la función isna().sum(). isna() devuelve una Serie de valores booleanos, donde True indica un valor nulo y False un valor no nulo. Al

aplicar sum() a esta Serie, Python automáticamente convierte los valores True en 1 y los False en 0, y luego suma todos los valores, proporcionando el número total de valores nulos en la columna. En este caso, espero que el número sea 0, indicando que todos los valores nulos se han llenado correctamente.

Estoy reemplazando los valores mal escritos o mal clasificados en la columna 'Industry'. En particular, estoy cambiando todas las entradas que dicen 'Finttech' por 'Fintech'. Esto es importante porque la limpieza y estandarización de los datos son pasos esenciales antes de realizar cualquier análisis más profundo. Al tener los datos limpios y estandarizados, me aseguro de que cualquier análisis o modelo que construya sea lo más preciso y relevante posible.

He corregido otro error de escritura en la columna 'Industry', cambiando 'Artificial Intelligence' por 'Artificial intelligence'. La consistencia en la nomenclatura de las categorías es vital para mantener la precisión en el análisis.

Luego, he creado una variable llamada 'startups_por_industria' que contiene el conteo de startups por cada industria. Esta información es valiosa para entender en qué industrias se concentran más estas empresas unicornio.

Resultados

Número de startups por industria:

- Fintech: 191
- Internet software & services: 167
- E-commerce & direct-to-consumer: 102
- Artificial intelligence: 73
- Health: 63
- Other: 51
- Supply chain, logistics, & delivery: 51
- Cybersecurity: 41

- Mobile & telecommunications: 37
- Data management & analytics: 36
- Hardware: 32
- Auto & transportation: 29
- Edtech: 27
- Consumer & retail: 23
- Travel: 13

Estos datos nos dan una visión general de las industrias en las que las startups unicornio tienden a concentrarse más. Por ejemplo, la industria Fintech parece tener el mayor número de estas empresas, seguida por el software y servicios de internet.

Primero, he identificado cuántos países únicos hay en la columna 'Country'. Resulta que hay un total de 47 países diferentes donde se encuentran las startups unicornio. Este es un indicador de la diversidad geográfica de estas empresas.

Después, he calculado el número de startups por país. De esta lista, se puede ver que las startups unicornio están principalmente concentradas en los Estados Unidos y China, seguidos por India y el Reino Unido. También es importante destacar que hay varias empresas unicornio en países que normalmente no se consideran los principales centros de startups, lo que muestra que la innovación puede surgir de cualquier parte del mundo.

Análisis Multivariante

El gráfico que he generado es un mapa de calor de la correlación entre las características numéricas en nuestro conjunto de datos. Este tipo de gráfico es útil para entender rápidamente las relaciones entre nuestras características.

En términos simples, una correlación es una medida de la relación entre dos variables. Los valores de correlación van de -1 a 1. Si el valor es cercano a 1, significa que hay una fuerte correlación positiva entre las dos variables. Cuando se acerca a -1, las variables tienen una fuerte correlación negativa.

Aquí, estamos utilizando el método .corr() que calcula la correlación de Pearson. Observando el mapa de calor de correlación, se puede ver que no hay una correlación significativa entre las variables numéricas de nuestra data. Esto significa que las características numéricas no están fuertemente relacionadas entre sí.

También es importante tener en cuenta que, aunque la correlación puede ayudar a identificar las relaciones existentes entre las variables, una correlación baja no necesariamente significa que las variables son irrelevantes para nuestro modelo. Cada característica todavía puede proporcionar información única que puede ayudar a nuestro modelo a hacer predicciones más precisas.

Análisis Bivariante

Este gráfico de línea que he generado representa la evolución de la valoración media de las startups unicornio a lo largo del tiempo. El eje X representa los años, mientras que el eje Y muestra la valoración media de las startups unicornio.

Observando el gráfico, podemos ver la tendencia de la valoración media de las startups a lo largo de los años. Esto puede dar una idea de cómo ha evolucionado la valoración de las startups unicornio en el pasado, lo que podría ser útil para entender la dinámica del mercado y prever las tendencias futuras.

Este tipo de análisis es útil porque nos permite entender mejor cómo se están comportando ciertas características en nuestro conjunto de datos a lo largo del tiempo. Al entender estas tendencias, podemos hacer predicciones más informadas y entender mejor la relación entre las características y la variable objetivo.

Este gráfico de barras que he creado representa la valoración media de las startups unicornio agrupadas por industria. El eje X muestra las distintas industrias y el eje Y muestra la valoración media de las startups unicornio.

Observando el gráfico, podemos ver cuál es la valoración media de las startups unicornio en cada industria. Esto puede ser útil para entender cuáles son las industrias más valiosas en términos de startups unicornio. Además, también nos permite ver la variabilidad entre las diferentes industrias.

Este tipo de análisis es útil para entender mejor las diferencias entre los grupos en nuestros datos. Al entender estas diferencias, podemos identificar las características que podrían tener un impacto significativo en la valoración de una startup unicornio.

Feature Engineering

Lo que acabo de hacer es una parte de la ingeniería de características en la columna 'Investors'. Primero, he obtenido una lista de todos los inversores únicos de la columna 'Investors' utilizando el método unique(). Luego, he recorrido cada fila de la columna 'Investors', he dividido cada cadena de inversores por comas para obtener una lista de inversores para cada startup, y luego he añadido cada inversor individual a la lista 'todos_inversores'. Finalmente, he creado una lista de todos los inversores únicos, 'inversores_unicos', utilizando la función set() para eliminar los duplicados y list() para convertir el conjunto resultante en una lista.

Ahora, con esta lista de 'inversores_unicos', tengo una mejor comprensión de los diferentes inversores involucrados en estas startups unicornio y puedo utilizar esta información para futuras tareas de análisis y modelado. Por ejemplo, podría usar esta información para crear nuevas características basadas en la presencia de ciertos inversores, o para analizar el impacto de diferentes inversores en la valoración de las startups.

He creado una nueva característica, 'num_investors', que representa el número de inversores de cada startup. Para hacerlo, he aplicado una función lambda a la columna 'Investors' que divide cada cadena de inversores por comas y luego toma la longitud de la lista resultante. Esto da el número de inversores para cada startup.

Como he enseñado antes, estoy trabajando con un conjunto de datos de "unicornios", que son empresas privadas valoradas en más de mil millones de dólares. Estos datos incluyen características categóricas como 'Country', 'City' y 'Industry'. Sin embargo, muchos algoritmos de aprendizaje automático funcionan mejor con entradas numéricas en lugar de categóricas. Para abordar esto, he decidido transformar estas variables categóricas en un formato que los modelos de aprendizaje automático puedan interpretar más eficientemente.

Para hacerlo, he utilizado la función get_dummies de pandas, la cual implementa una técnica llamada codificación One-Hot. Esta función transforma cada valor único en una columna

categórica en una nueva columna (o variable ficticia) en el DataFrame. Luego, asigna un valor de 1 o 0 a estas nuevas columnas para cada registro, dependiendo de si el registro tenía ese valor en la columna original.

Por lo tanto, cuando escribí el código unicornios = pd.get_dummies(unicornios, columns=['Country', 'City', 'Industry']), lo que hice fue transformar las columnas 'Country', 'City' e 'Industry' de mi DataFrame 'unicornios' en múltiples columnas numéricas nuevas. Por ejemplo, si la columna 'Country' originalmente contenía 'USA', 'UK' y 'China', ahora tengo tres nuevas columnas llamadas 'Country_USA', 'Country_UK' y 'Country_China', cada una indicando con un 1 si la empresa es de ese país y un 0 en caso contrario.

Al hacer esto, facilito que los algoritmos de aprendizaje automático puedan interpretar y utilizar estas características para aprender de los datos y hacer predicciones precisas.

Escalado de datos

Lo que acabo de hacer se llama preprocesamiento de datos, específicamente, la normalización de los datos. He utilizado la función StandardScaler de la biblioteca sklearn, que normaliza las características al restar la media y dividir por la desviación estándar.

Este proceso es importante porque algunos algoritmos de machine learning no funcionan bien si las características están en diferentes escalas. Por ejemplo, muchos algoritmos se basan en la distancia euclidiana entre dos puntos de datos, y si una característica está en una escala mucho mayor que otra, la característica con la escala mayor dominará. Al estandarizar las características, podemos evitar este problema y asegurarnos de que todas las características sean tratadas por igual en nuestro algoritmo.

En este caso, he normalizado las características 'Valuation' y 'num_investors'. Estas características ahora tendrán una media de 0 y una desviación estándar de 1, lo que nos permitirá utilizarlas en un algoritmo de machine learning que requiere que las características estén en la misma escala.

En este paso, estoy identificando aquellas características que tienen una correlación absoluta con la variable objetivo 'Valuation' mayor que un cierto umbral. En este caso, el umbral es 0.5. Este enfoque nos permite seleccionar aquellas características que tienen una fuerte relación con la variable objetivo, lo que puede mejorar el rendimiento de nuestro modelo de aprendizaje automático.

Después de obtener la correlación de las características con 'Valuation', estoy estableciendo un umbral de 0.2. Este umbral es un criterio arbitrario que he establecido para seleccionar las características que tienen una correlación significativa (tanto positiva como negativa) con 'Valuation'. El valor absoluto se usa aquí para considerar ambas correlaciones positivas y negativas.

Después, selecciono los índices de las características que tienen un valor absoluto de correlación mayor que el umbral. Al hacer esto, estoy creando una lista de las características que están más fuertemente correlacionadas con 'Valuation' y que, por lo tanto, son las más relevantes para predecir 'Valuation'.

Finalmente, imprimo estas características seleccionadas para revisar cuáles son. Esto me ayuda a comprender mejor cuáles son las características más relevantes en mi conjunto de datos.

Construcción y Evaluación de Modelos.

En este paso de mi proyecto, estoy construyendo y evaluando varios modelos de aprendizaje automático para predecir la valoración de los "unicornios". He importado una variedad de modelos, incluyendo Regresión Lineal, Regresión Polinomial, Árboles de Decisión, K-Vecinos más Cercanos (KNN), Máquinas de Vectores de Soporte (SVM) y Bosques Aleatorios.

Primero, selecciono las características que usaré para entrenar los modelos. Estas características incluyen el año, el número de inversores y las variables ficticias creadas a partir de la industria y el país de cada unicornio. Posteriormente, divido mi conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba, utilizando un 80% de los datos para entrenar y el 20% restante para probar.

A continuación, creo una lista de los modelos que quiero probar, cada uno con su nombre y la correspondiente función de la librería Scikit-Learn. Luego, entreno cada uno de estos modelos con mis datos. Para el modelo de Regresión Polinomial, añado un paso adicional en el que transformo mis características a términos polinomiales antes de entrenar el modelo. Este paso se hace porque la Regresión Polinomial puede capturar relaciones más complejas entre las características y la variable objetivo.

Una vez entrenado cada modelo, los utilizo para predecir las valoraciones de los unicornios en mi conjunto de pruebas. Luego, calculo varias métricas de error para cada modelo, incluyendo el Error Cuadrático Medio (MSE), el Error Absoluto Medio (MAE), la Raíz del Error Cuadrático Medio (RMSE), y el Coeficiente de Determinación, también conocido como R cuadrado. Estas métricas me ayudan a evaluar qué tan bien cada modelo está prediciendo las valoraciones y a comparar el rendimiento entre los modelos.

Finalmente, imprimo los resultados de cada modelo. Este paso me ayuda a entender el rendimiento de cada uno y a decidir cuál de ellos es el más prometedor para mi objetivo de

predecir las valoraciones de los unicornios. En base a estos resultados, puedo elegir el mejor modelo y, si es necesario, optimizarlo más para mejorar su rendimiento.

En este paso, decidí realizar validación cruzada en mis modelos para obtener una mejor idea de su rendimiento. La validación cruzada es un método que proporciona una evaluación más robusta de los modelos al dividir el conjunto de datos en 'k' subconjuntos y luego entrenar y probar el modelo 'k' veces, cada vez utilizando un subconjunto diferente como conjunto de prueba. He decidido usar 'k=5', lo que significa que estoy dividiendo mis datos en 5 subconjuntos.

Además, he incorporado la clase Pipeline de Scikit-Learn para garantizar que la transformación PolynomialFeatures se aplique durante cada iteración de la validación cruzada para la regresión polinomial. Antes, solo aplicaba la transformación a los datos de entrenamiento y prueba iniciales, pero con la validación cruzada, necesito que se aplique a cada subconjunto de datos de entrenamiento.

Primero, creo una lista de los modelos que quiero evaluar, al igual que antes, pero esta vez para la Regresión Polinomial, envuelvo la transformación PolynomialFeatures y la Regresión Lineal en un Pipeline. Un Pipeline es una forma de encadenar múltiples pasos de procesamiento y modelado en un solo estimador.

A continuación, para cada modelo, realizo la validación cruzada utilizando la función cross_val_score de Scikit-Learn. Esta función entrena y prueba el modelo 5 veces en diferentes subconjuntos de datos y devuelve 5 puntuaciones de error. Sin embargo, debido a que estoy utilizando el error cuadrático medio negativo como métrica de puntuación (es el valor negativo del error cuadrático medio), tomo el valor absoluto de estas puntuaciones y luego calculo su raíz cuadrada para obtener el error cuadrático medio raíz (RMSE) para cada iteración.

Finalmente, imprimo los RMSE de cada iteración, así como su media y desviación estándar, para cada modelo. Esto me da una idea de cómo se comporta cada modelo en promedio y cuánta variabilidad hay en sus puntuaciones de error. Estos resultados me permiten comparar los modelos de manera más efectiva y seleccionar el modelo que mejor se adapta a mis necesidades.

Llegados a este punto, decido realizar una optimización de los hiperparámetros de mi modelo de bosques aleatorios. Los hiperparámetros son parámetros del modelo que se establecen antes de entrenar el modelo y que no se aprenden a partir de los datos. Ajustar estos hiperparámetros puede mejorar la eficacia del modelo.

Para hacer esto, empleo la técnica de búsqueda de cuadrícula, que prueba todas las posibles combinaciones de los hiperparámetros que quiero ajustar. Los hiperparámetros que decido ajustar para mi modelo de bosques aleatorios son 'n_estimators' (el número de árboles en el bosque), 'max_depth' (la máxima profundidad de los árboles), 'min_samples_split' (el número mínimo de muestras requerido para dividir un nodo interno) y 'min_samples_leaf' (el número mínimo de muestras requerido para ser un nodo hoja).

Para realizar la búsqueda de cuadrícula, creo un diccionario 'param_grid' con los hiperparámetros y los valores que quiero probar. A continuación, instancio mi modelo de bosques aleatorios y creo un objeto de búsqueda de cuadrícula, especificando el modelo, la cuadrícula de parámetros, el número de pliegues para la validación cruzada (en este caso, 3), y configurando 'n_jobs' en -1 para utilizar todos los procesadores disponibles y acelerar la computación.

A continuación, ajusto la búsqueda de cuadrícula a mis datos de entrenamiento, lo que hace que pruebe todas las posibles combinaciones de los hiperparámetros especificados y encuentre la combinación que da los mejores resultados en la validación cruzada.

Finalmente, imprimo los mejores hiperparámetros encontrados y extraigo el mejor modelo resultante de la búsqueda de cuadrícula. Este modelo, que almaceno en la variable 'best_rf', es un modelo de bosques aleatorios con los hiperparámetros optimizados que utilizaré para las predicciones y evaluaciones futuras.

En esta parte y ya llegando al final, decido aplicar el modelo de Bosque Aleatorio (Random Forest) optimizado, que obtuve como resultado de mi búsqueda de cuadrícula, para hacer predicciones sobre mi conjunto de prueba. Las configuraciones de este modelo incluyen 50 estimadores, una profundidad máxima de 20, un mínimo de 5 muestras para dividir un nodo interno y un mínimo de 4 muestras requeridas para estar en un nodo hoja.

Primero, entreno este modelo optimizado con mi conjunto de datos de entrenamiento, que son las características y valores objetivo que reservé específicamente para este propósito. Esto es crucial para que el modelo pueda aprender patrones y relaciones dentro de los datos.

Una vez que el modelo está entrenado, lo utilizo para hacer predicciones en el conjunto de datos de prueba. Este conjunto de datos es diferente al de entrenamiento, y me permite evaluar cómo se comporta el modelo con datos que no ha visto antes, lo cual es un indicativo real de su capacidad de generalización.

A continuación, calculo varias métricas de rendimiento para evaluar la eficacia de mi modelo. El error cuadrático medio (MSE), el error absoluto medio (MAE) y la raíz del error cuadrático medio (RMSE) son medidas de la diferencia entre los valores que mi modelo predice y los valores reales. Cuanto más bajos son estos números, mejor es el modelo. También calculo el coeficiente de determinación (R2 score), que indica cuánta variabilidad en los datos puede explicar mi modelo. Cuanto más cerca esté este valor de 1, mejor es el rendimiento del modelo.

Finalmente, imprimo estas métricas para tener un registro claro de cómo se comporta mi modelo optimizado en los datos de prueba.

En esta última etapa de mi trabajo, decido guardar el modelo que entrené para su uso futuro. Esto es esencial porque no siempre querré pasar por el tiempo y los recursos computacionales para volver a entrenar el modelo cada vez que necesite hacer predicciones.

Para guardar mi modelo, utilizo la biblioteca joblib y su función dump. Joblib es una biblioteca en Python para la serialización de objetos Python, lo que significa que puede tomar casi cualquier objeto Python y convertirlo en un flujo de bytes. Esta característica la hace ideal para guardar modelos de machine learning.

El primer argumento para la función dump es el objeto que quiero guardar, en este caso, mi modelo optimizado de bosque aleatorio. El segundo argumento es el nombre del archivo en el que deseo guardar el objeto. Elijo el nombre 'new_model.joblib' para mi archivo.

Después de ejecutar este código, tengo un archivo llamado 'new_model.joblib' que contiene todo lo que necesito para cargar mi modelo y hacer predicciones en el futuro. Este archivo se puede cargar en otra sesión de Python o en otra máquina utilizando la función load de joblib.