ingenieria-de-caracteristicas

November 6, 2024

1 Introducción a la Ingeniería de Características

La **Ingeniería de Características** es uno de los pilares de la ciencia de datos y el aprendizaje automático. Este proceso implica transformar los datos en variables o características útiles que maximizan el rendimiento predictivo de los modelos. Desde convertir variables categóricas en números hasta crear nuevas características a partir de las existentes, la ingeniería de características busca aprovechar toda la información disponible en los datos de manera efectiva.

A continuación, exploraremos su origen histórico, la importancia del proceso y algunas de las técnicas clave que se aplican en la ingeniería de características en la actualidad.

1.1 Historia de la Ingeniería de Características

La ingeniería de características tiene sus raíces en los primeros estudios estadísticos y matemáticos de la década de 1960, cuando el análisis de datos era principalmente manual y centrado en modelos lineales simples. Durante estos años, los científicos reconocieron que el valor predictivo de un modelo dependía en gran medida de las variables utilizadas y de cómo se transformaban. Estos primeros pasos en selección y transformación de variables sentaron las bases de lo que hoy conocemos como ingeniería de características.

En las décadas de 1980 y 1990, con el crecimiento de la inteligencia artificial y el aprendizaje automático, los modelos comenzaron a volverse más complejos y el volumen de datos aumentó considerablemente. Esto dio lugar a una práctica más estructurada de la ingeniería de características, en la que los científicos de datos aplicaban técnicas de transformación de variables para mejorar la precisión y la capacidad predictiva de los modelos.

A medida que avanzaba el siglo XXI y se desarrollaban algoritmos más sofisticados, como las redes neuronales profundas, la ingeniería de características se consolidó como una disciplina esencial. Incluso con estos modelos avanzados, capaces de aprender automáticamente características complejas de los datos, sigue siendo crucial preparar y transformar las características de manera adecuada para lograr modelos efectivos y eficientes.

1.2 ¿Qué es la Ingeniería de Características?

La ingeniería de características es el proceso de seleccionar, transformar y crear variables que maximizan el rendimiento de un modelo. Su objetivo es proporcionar una representación de los datos que mejore la precisión del modelo y reduzca el ruido o la redundancia. Esta preparación adecuada permite que el modelo detecte patrones de manera más efectiva y haga predicciones más precisas.

1.2.1 Principales Etapas de la Ingeniería de Características

1. **Transformación de Variables Categóricas**: La mayoría de los algoritmos de aprendizaje automático requieren datos numéricos, por lo que es necesario convertir las variables categóricas en valores numéricos a través de técnicas como el *label encoding* (codificación de etiquetas), el *one-hot encoding* (codificación binaria) o métodos avanzados como el *target encoding*.
2. **Generación de Nuevas Características**: Crear características adicionales a partir de las variables existentes puede enriquecer el modelo y mejorar su rendimiento. Por ejemplo, se pueden combinar variables de fecha en una sola variable de "estación del año" o derivar métricas de ventas promedio a partir de datos transaccionales.
3. **Normalización y Estandarización**: Ajustar la escala de las variables numéricas es crucial para ciertos modelos, especialmente aquellos que utilizan métricas de distancia, como *K-Nearest Neighbors* o regresión logística. La normalización y la estandarización también ayudan a mejorar la convergencia de modelos complejos como las redes neuronales.
4. **Selección de Características**: Eliminar características irrelevantes o redundantes reduce el ruido y la complejidad del modelo, conservando solo las variables que aportan información valiosa. Métodos como el análisis de correlación o la importancia de características ayudan a identificar las variables más significativas.
5. **Codificación Avanzada de Características**: Cuando existen muchas categorías únicas en una variable (por ejemplo, códigos de país o tipos de productos), se pueden usar métodos avanzados de codificación como el *hash encoding* o *binary encoding*, que permiten una representación más compacta sin perder información relevante.

1.3 Importancia de la Ingeniería de Características

La ingeniería de características es fundamental para crear modelos predictivos efectivos. La calidad y relevancia de las características no solo mejoran la precisión del modelo, sino que también permiten interpretarlo de manera más efectiva. Además, cuando las características están bien diseñadas y seleccionadas, se reduce el riesgo de sobreajuste, mejorando así la capacidad del modelo para generalizar a nuevos datos.

En la práctica, se ha demostrado que la ingeniería de características puede tener un impacto mucho mayor en el rendimiento del modelo que el propio algoritmo. Por ello, se considera una de las habilidades más importantes en el trabajo de un científico de datos.

Además, la ingeniería de características ayuda a reducir la complejidad computacional al disminuir la cantidad de datos redundantes o irrelevantes y mejorar la eficiencia del modelo, un aspecto clave en aplicaciones en tiempo real o de gran escala.

1.4 Técnicas Comunes de Ingeniería de Características

Existen diversas técnicas para llevar a cabo la ingeniería de características, cada una con ventajas y desventajas según el tipo de datos y el modelo:

1. **Label Encoding y One-Hot Encoding**: Transformación de variables categóricas en valores numéricos mediante etiquetas o vectores binarios.
2. **Target Encoding**: Reemplaza cada categoría por el promedio de la variable objetivo, una técnica especialmente útil en problemas de clasificación.

3. **Hash Encoding**: Transforma categorías en una representación numérica compacta utilizando una función hash.
4. **Embeddings**: Representaciones densas de variables categóricas, especialmente útiles en redes neuronales profundas para variables con muchas categorías.
5. **Generación de Características Derivadas**: Crear nuevas variables a partir de transformaciones matemáticas (sumas, restas, promedios) de variables originales.
6. **Normalización y Estandarización**: Ajuste de escala de las variables numéricas para mejorar la convergencia y precisión de los modelos.
7. **Selección de Características Basada en Importancia**: Identificación de las características más relevantes usando métodos como bosques aleatorios o pruebas estadísticas.

1.5 Ejemplo de la Ingeniería de Características en la Práctica

Supongamos que tenemos un conjunto de datos de ventas que incluye variables como "categoría de producto", "precio", "tiempo en inventario" y "ubicación de la tienda". Un científico de datos podría aplicar los siguientes pasos de ingeniería de características:

- **Codificación de Variables Categóricas**: Transformar "categoría de producto" y "ubicación de la tienda" en variables numéricas mediante técnicas como *one-hot encoding* o *target encoding*.
- **Generación de Nuevas Características**: Crear variables como "nivel de descuento" y "mes del año" basadas en la fecha y el precio, enriqueciendo el modelo con información adicional.
- **Normalización de Variables Numéricas**: Escalar "precio" y "tiempo en inventario" para que tengan una escala comparable y evitar que una de estas variables tenga una influencia excesiva en el modelo.
- **Selección de Características**: Evaluar la importancia de cada variable y eliminar aquellas que no contribuyan significativamente al rendimiento del modelo, optimizando así su eficiencia.

Cada uno de estos pasos ayuda al modelo a aprender patrones más significativos y a generalizar mejor en nuevos datos, incrementando su precisión y robustez.

La ingeniería de características es una etapa fundamental en el flujo de trabajo de la ciencia de datos y el aprendizaje automático. Desde sus primeros días en los modelos estadísticos lineales hasta su papel en los modelos avanzados actuales, esta práctica ha demostrado ser esencial para mejorar la precisión y eficiencia de los modelos. En un mundo de datos complejos y masivos, la ingeniería de características es una habilidad clave para los científicos de datos, capaz de marcar la diferencia entre un modelo promedio y uno de alta calidad.

2 Conversión de Variables Categóricas a Numéricas en Ciencia de Datos

En ciencia de datos, la conversión de variables categóricas a numéricas es fundamental para poder utilizar estos datos en modelos de machine learning. Además del label encoding y el one-hot encoding, existen otros métodos para transformar variables categóricas en representaciones numéricas. Aquí te presento algunas alternativas:

2.1 1. Label Encoding

Convierte cada categoría en un número entero único. Este método es comúnmente utilizado en modelos de machine learning donde es necesario transformar variables categóricas en un formato que pueda ser entendido por algoritmos de aprendizaje automático.

2.1.1 Ejemplo:

Considera una columna "Color" con valores ["Rojo", "Verde", "Azul"]. La codificación se realizaría de la siguiente manera: - Rojo: 0 - Verde: 1 - Azul: 2

Si tu conjunto de datos es: | Color | |:———:| | Rojo | | Verde | | Azul | | Verde |

Tras aplicar el Label Encoding, obtendrías: | Color | Color_Encoded | |:———:|:—————:| | Rojo | 0 | | Verde | 1 | | Azul | 2 | | Verde | 1 |

2.1.2 Ventajas:

- **Simplicidad**: Fácil de implementar y entender; no requiere transformaciones complicadas.
- **No Aumenta Dimensionalidad**: Mantiene el número de características constantes, lo cual es importante en datasets grandes.
- **Rápido**: La codificación es rápida y eficiente en términos de tiempo de ejecución.
- **Compatible con Modelos Lineales**: Funciona bien con algoritmos lineales como regresión logística y SVM, ya que no introduce características adicionales.
- **Uso Mínimo de Memoria**: Consume poca memoria, ya que cada categoría es representada por un único número entero.

2.1.3 Desventajas:

- **Orden Artificial**: Introduce un orden que no existe entre las categorías (por ejemplo, Rojo no es "menor" que Verde).
- **Mal Rendimiento en Algunos Modelos**: Puede inducir a error en modelos que asumen que hay una relación ordinal entre las categorías, lo que puede llevar a predicciones incorrectas.
- **No Captura Relaciones**: No refleja similitudes o relaciones entre categorías; simplemente asigna números.
- **Problemas con Nuevas Categorías**: Las nuevas categorías que no estaban en el conjunto de entrenamiento deben ser codificadas manualmente, lo que puede causar inconsistencias.
- **Puede Generar Overfitting**: Podría sobreajustarse en modelos que son muy sensibles a los valores de entrada, especialmente si las categorías son limitadas.

2.1.4 Recomendaciones:

- **Uso en Modelos Lineales**: Ideal para algoritmos como regresión o SVM, donde se requiere un número fijo de características.
- **Explorar Correlaciones**: Antes de aplicar, verifica si hay algún orden natural en las categorías, lo cual podría justificar el uso de Label Encoding.
- **Validación Cruzada**: Emplear técnicas de validación cruzada para evaluar el rendimiento y evitar sobreajuste en modelos.
- **Combinación con Otros Métodos**: Puede ser útil en combinación con one-hot encoding en datasets donde algunas variables son ordinales y otras nominales.

- **Análisis de Frecuencia**: Estudiar la distribución de categorías y su frecuencia antes de la codificación para entender su posible impacto en el modelo.

2.2 2. One-Hot Encoding

One-Hot Encoding es un método que transforma cada categoría de una variable categórica en una nueva columna binaria (0 o 1). Cada columna representa la presencia o ausencia de una categoría particular en los datos.

2.2.1 Ejemplo:

Consideremos una columna "Fruta" con valores ["Manzana", "Naranja", "Banana"]. La codificación se realizaría de la siguiente manera:

Si tu conjunto de datos original es: | Fruta | |:———:| | Manzana | | Naranja | | Banana | | Naranja |

Tras aplicar el One-Hot Encoding, obtendrías: | Fruta | Manzana | Naranja | Banana | |:———:|:———:|:———:|:———:| | Manzana | 1 | 0 | 0 | | Naranja | 0 | 1 | 0 | | Banana | 0 | 0 | 1 | | Naranja | 0 | 1 | 0 |

2.2.2 Ventajas:

- **No Introduce Orden Artificial**: Las categorías son independientes entre sí, evitando la creación de un orden no deseado.
- **Mejor Rendimiento en Modelos No Lineales**: Muchos algoritmos, como árboles de decisión, se benefician al tener información binaria clara.
- **Facilita la Interpretación**: Cada columna representa claramente una categoría, facilitando la interpretación de los resultados.
- **Captura Interacciones**: Permite que el modelo aprenda interacciones entre categorías a través de columnas separadas.
- **Compatibilidad con Modelos Basados en Distancia**: Funciona bien con algoritmos que utilizan métricas de distancia, como KNN.

2.2.3 Desventajas:

- **Aumento de Dimensionalidad**: Puede incrementar significativamente el número de características, especialmente con categorías con alta cardinalidad.
- **Escalabilidad**: Dificulta el manejo de grandes conjuntos de datos con muchas categorías únicas, ya que puede generar matrices dispersas grandes.
- **Problemas de Multicolinealidad**: Si se utilizan múltiples columnas para la misma variable, puede introducir colinealidad, lo que podría afectar ciertos modelos.
- **Complejidad en la Interpretación**: En algunos casos, puede dificultar la comprensión general de las relaciones, especialmente en conjuntos de datos grandes.
- **Inconsistencia con Nuevas Categorías**: Nuevas categorías no vistas durante el entrenamiento no se pueden representar y deben manejarse de forma especial.

2.2.4 Recomendaciones:

- **Uso en Modelos No Lineales**: Ideal para algoritmos como árboles de decisión, random forests o redes neuronales donde las interacciones entre categorías son importantes.

- **Manejo de Alta Cardinalidad**: Evitar su uso en variables con muchas categorías únicas, ya que puede llevar a un espacio de características demasiado grande.
- **Considerar Técnicas de Reducción**: Aplicar técnicas como PCA (Análisis de Componentes Principales) o técnicas de agrupamiento para reducir la dimensionalidad después de aplicar One-Hot Encoding.
- **Combinación con Otros Métodos**: Puede combinarse con Label Encoding en casos donde algunas variables sean ordinales y otras nominales.
- **Análisis de Impacto en el Modelo**: Realizar pruebas A/B para evaluar el impacto de One-Hot Encoding en el rendimiento del modelo antes de su implementación final.

2.3 3. Ordinal Encoding

Ordinal Encoding convierte cada categoría en un número entero basado en un orden específico. Este método es útil cuando las categorías tienen una relación jerárquica o un orden natural.

2.3.1 Ejemplo:

Imaginemos una columna "Tamaño" con valores ["Pequeño", "Mediano", "Grande"]. La codificación podría ser: - Pequeño: 1 - Mediano: 2 - Grande: 3

El conjunto de datos original sería: | Tamaño | |:———-:| | Pequeño | | Mediano | | Grande | | Mediano |

Tras aplicar Ordinal Encoding, obtendrías: | Tamaño | Tamaño Encoded | |:———-:|:————:| | Pequeño | 1 | | Mediano | 2 | | Grande | 3 | | Mediano | 2 |

2.3.2 Ventajas:

- **Representación Natural**: Captura el orden de las categorías de manera efectiva, lo que puede ser útil para modelos que se benefician de esta información.
- **Menor Dimensionalidad**: A diferencia de One-Hot Encoding, no aumenta la dimensionalidad del conjunto de datos.
- **Simplicidad en Implementación**: Es fácil de implementar y entender.
- **Compatible con Modelos Lineales**: Funciona bien con modelos que suponen un orden, como la regresión lineal.
- **Uso Mínimo de Memoria**: Consume menos espacio en comparación con métodos que generan múltiples columnas.

2.3.3 Desventajas:

- **Orden Artificial en Categorías No Ordenadas**: Puede inducir a error si las categorías no tienen un orden inherente, lo que puede llevar a resultados engañosos.
- **Riesgo de Interpretación Incorrecta**: Los modelos pueden interpretar las diferencias entre los valores codificados como significativas, lo cual no siempre es correcto.
- **Problemas con Nuevas Categorías**: Nuevas categorías que no están en el conjunto de entrenamiento pueden no ser manejadas adecuadamente.
- **Limitaciones en Modelos No Lineales**: No siempre es adecuado para modelos no lineales, donde el orden puede no ser relevante.
- **Puede Generar Overfitting**: Si se tiene un número limitado de categorías, puede resultar en sobreajuste en modelos complejos.

2.3.4 Recomendaciones:

- **Uso en Datos con Orden Natural**: Ideal para variables que tienen una jerarquía clara, como "Bajo", "Medio", "Alto".
- **Validación Cruzada**: Realiza validación cruzada para evaluar el impacto de esta codificación en el modelo y evitar sobreajuste.
- **Combinación con Otras Técnicas**: Considera combinarlo con métodos como One-Hot Encoding para variables donde el orden no es relevante.
- **Análisis de Correlaciones**: Realiza un análisis previo para verificar la existencia de un orden significativo entre las categorías antes de aplicar este método.
- **Evaluación de Impacto en el Modelo**: Prueba el rendimiento del modelo con y sin Ordinal Encoding para medir su efecto en la precisión.

2.4 4. Target Encoding (o Mean Encoding)

Target Encoding consiste en reemplazar cada categoría por el valor promedio de la variable objetivo (target) para esa categoría. Este método es especialmente útil en problemas de clasificación binaria y puede capturar mejor la relación entre la categoría y la variable objetivo.

2.4.1 Ejemplo:

Supongamos que tenemos un conjunto de datos de ventas con una columna "Ciudad" y una columna "Ventas" que es nuestra variable objetivo:

Ciudad	Ventas
Ciudad A	100
Ciudad B	200
Ciudad A	150
Ciudad C	300
Ciudad B	250

Al aplicar Target Encoding, calculamos el promedio de ventas por ciudad:

- Ciudad A: (100 + 150) / 2 = 125
- Ciudad B: (200 + 250) / 2 = 225
- Ciudad C: 300 / 1 = 300

La tabla resultante después del encoding sería:

Ciudad	Ventas	Ciudad Encoded
Ciudad A	100	125
Ciudad B	200	225
Ciudad A	150	125
Ciudad C	300	300
Ciudad B	250	225

2.4.2 Ventajas:

- **Captura de Relaciones**: Puede capturar relaciones importantes entre las categorías y la variable objetivo.
- **No Aumenta Dimensionalidad**: A diferencia de métodos como One-Hot Encoding, no aumenta el número de columnas en el conjunto de datos.
- **Mejora del Rendimiento del Modelo**: Puede mejorar significativamente la precisión del modelo al reflejar el impacto de cada categoría en el target.
- **Facilidad de Implementación**: Relativamente fácil de implementar y entender.
- **Flexibilidad**: Funciona bien en modelos tanto lineales como no lineales.

2.4.3 Desventajas:

- **Riesgo de Sobreajuste**: Puede provocar sobreajuste si no se maneja adecuadamente, especialmente en conjuntos de datos pequeños.
- **Dependencia del Conjunto de Datos**: El rendimiento puede variar significativamente según la distribución del conjunto de datos.
- **No Se Puede Usar con Nuevas Categorías**: Nuevas categorías en el conjunto de prueba que no están presentes en el entrenamiento no se pueden codificar adecuadamente.
- **Problemas con Multicolinealidad**: Puede introducir multicolinealidad si se utilizan varias variables categóricas de este tipo en el mismo modelo.
- **Complejidad Adicional**: Puede agregar complejidad a la preparación de datos, ya que se requiere calcular promedios.

2.4.4 Recomendaciones:

- **Uso en Clasificación Binaria**: Especialmente útil en problemas de clasificación binaria donde se busca la relación directa con la variable objetivo.
- **Validación Cruzada**: Implementar técnicas de validación cruzada para minimizar el riesgo de sobreajuste, usando técnicas como K-fold.
- **Manejo de Nuevas Categorías**: Considerar cómo manejar nuevas categorías, como asignarles un valor medio global o utilizar una técnica de suavizado.
- **Combinar con Otras Técnicas**: Puede ser beneficioso combinar Target Encoding con otros métodos de codificación para enriquecer la representación de las características.
- **Evaluación de Impacto en el Modelo**: Siempre evaluar el impacto en el rendimiento del modelo comparando con otros métodos de codificación.

2.5 5. Binary Encoding

Binary Encoding combina aspectos de label encoding y one-hot encoding, pero de una manera más compacta. Este método primero convierte cada categoría en un número entero (label encoding), y luego transforma este número a una representación binaria. Finalmente, se utiliza una columna por cada dígito binario.

2.5.1 Ejemplo:

Supongamos que tenemos tres categorías: “Perro”, “Gato” y “Pájaro”. Primero, aplicamos el label encoding:

- Perro: 0

- Gato: 1
- Pájaro: 2

Luego, convertimos estos números a su representación binaria de la siguiente manera:

- 0 en binario: 00
- 1 en binario: 01
- 2 en binario: 10

Finalmente, la tabla resultante sería:

Animal	Encoded	Binary Encoded
Perro	0	00
Gato	1	01
Pájaro	2	10

En términos de columnas, esto se representaría como dos columnas, ya que la representación binaria de 2 necesita dos bits.

2.5.2 Ventajas:

- **Menor Dimensionalidad**: Reduce la cantidad de columnas en comparación con one-hot encoding, lo que es útil para conjuntos de datos con muchas categorías.
- **Captura de Relaciones**: Puede capturar relaciones entre categorías que pueden ser importantes para el modelo.
- **Eficiente en Memoria**: Usa menos memoria que el one-hot encoding en variables con muchas categorías.
- **Manejo de Categorías**: Permite incluir nuevas categorías sin tener que rediseñar el conjunto de datos.
- **Compatible con Varios Modelos**: Funciona bien con varios algoritmos de aprendizaje automático.

2.5.3 Desventajas:

- **Complejidad Adicional**: Puede ser más complicado de implementar y entender que los métodos más simples.
- **Interpretación de Resultados**: La interpretación de los resultados puede ser menos intuitiva debido a la naturaleza binaria.
- **Riesgo de Overfitting**: Puede llevar a un modelo que se ajuste demasiado a los datos si se usa sin cuidado.
- **No Captura Orden**: Similar al label encoding, no captura un orden inherente entre las categorías.
- **Dificultades con Nuevas Categorías**: Nuevas categorías deben ser convertidas y pueden no tener una representación binaria adecuada si no se manejan adecuadamente.

2.5.4 Recomendaciones:

- **Uso en Variables con Muchas Categorías**: Ideal para variables que tienen muchas categorías únicas y que necesitan reducir la dimensionalidad.

- **Evaluar Desempeño**: Probar el rendimiento del modelo en comparación con otras técnicas de codificación como one-hot encoding.
- **Considerar el Contexto del Modelo**: Usar en contextos donde se sabe que la relación binaria puede tener significado (ejemplo: sistemas de recomendación).
- **Monitorear el Riesgo de Sobreajuste**: Implementar validación cruzada para evitar el sobreajuste del modelo.
- **Combinación con Otras Técnicas**: Considerar su uso junto con otras técnicas de codificación para obtener un mejor rendimiento en el modelo.

2.6 6. Frequency Encoding

Frequency Encoding asigna a cada categoría el número de veces que aparece en el conjunto de datos. Este método es útil cuando la frecuencia de una categoría puede tener una relación directa con la variable objetivo, ya que introduce un aspecto cuantitativo en lugar de solo ordinal o categórico.

2.6.1 Ejemplo:

Supongamos que tenemos una columna "Ciudad" con los siguientes datos: - Ciudad X - Ciudad Y - Ciudad X - Ciudad Z - Ciudad Y - Ciudad Y

Primero, contamos cuántas veces aparece cada ciudad:

- Ciudad X: 2
- Ciudad Y: 3
- Ciudad Z: 1

Luego, sustituimos cada ciudad en la columna original con su frecuencia:

Ciudad	Frecuencia Encoded
Ciudad X	2
Ciudad Y	3
Ciudad Z	1

Así, la columna "Ciudad" se transforma en "Frecuencia Encoded".

2.6.2 Ventajas:

- **Simplicidad**: Fácil de implementar y entender.
- **Captura Información Importante**: Permite que el modelo considere la popularidad de una categoría.
- **No Aumenta Dimensionalidad**: Mantiene la estructura del conjunto de datos sin agregar columnas adicionales.
- **Ideal para Modelos de Regresión**: Puede mejorar la precisión en problemas de regresión donde la frecuencia está correlacionada con el objetivo.
- **Ahorro de Espacio**: Consume menos memoria que el one-hot encoding en variables con muchas categorías.

2.6.3 Desventajas:

- **Orden No Intencional**: Puede dar la impresión de que hay un orden cuando solo hay frecuencias.
- **Falta de Captura de Relaciones Semánticas**: No refleja similitudes o relaciones inherentes entre las categorías.
- **Problemas con Datos Desbalanceados**: Si algunas categorías son muy raras, su representación puede no ser representativa.
- **No Útil en Todas las Situaciones**: Puede no ser eficaz en problemas donde la frecuencia no tiene relación con el target.
- **Riesgo de Overfitting**: Puede inducir a sobreajuste si se utiliza en variables con pocas observaciones.

2.6.4 Recomendaciones:

- **Útil en Datos Categóricos Desbalanceados**: Ideal cuando algunas categorías son significativamente más comunes.
- **Comprobar Correlación**: Evaluar la relación entre la frecuencia de las categorías y la variable objetivo antes de aplicar la codificación.
- **Combinar con Otros Métodos**: Considerar usarlo junto con one-hot encoding o target encoding para mejorar la robustez del modelo.
- **Implementar Validación Cruzada**: Asegurarse de que la frecuencia no introduce sobreajuste al modelo.
- **Usar en Modelos de Regresión y Árboles**: Funciona bien con modelos que pueden interpretar relaciones numéricas, como la regresión y los árboles de decisión.

2.7 7. Hash Encoding (o Hashing Trick)

Hash Encoding convierte cada categoría en un hash numérico dentro de un rango específico. Este método es especialmente útil cuando hay muchas categorías y se busca reducir la dimensionalidad. A través de una función hash, se puede especificar el número de columnas en la representación final, permitiendo así una representación más compacta sin perder información crítica.

2.7.1 Ejemplo:

Supongamos que tenemos una columna “Fruta” con las siguientes categorías: - Manzana - Plátano - Naranja - Fresa - Kiwi

Utilizando Hash Encoding, aplicamos una función hash que convierte cada nombre de fruta en un número entero. Si decidimos que queremos una representación de 3 columnas, la asignación de hash podría ser algo así:

Fruta	Hash	Representación
Manzana	12	110 (binary)
Plátano	25	101 (binary)
Naranja	18	011 (binary)
Fresa	21	100 (binary)
Kiwi	14	001 (binary)

Así, cada fruta se convierte en una representación binaria de 3 bits, permitiendo que el modelo trabaje con menos dimensiones.

2.7.2 Ventajas:

- **Reducción de Dimensionalidad**: Permite trabajar con una representación más compacta, útil en conjuntos de datos grandes.
- **Simplicidad**: Fácil de implementar, ya que se puede aplicar una función hash estándar.
- **Flexible**: Se puede especificar el número de columnas en la representación, ajustando el modelo según las necesidades.
- **No requiere Aprendizaje de Categorías**: No se necesita conocer todas las categorías por adelantado, ya que la función hash genera valores en tiempo de ejecución.
- **Manejo de Nuevas Categorías**: Las nuevas categorías se pueden agregar sin necesidad de reentrenar el modelo, simplemente se les aplica el hash.

2.7.3 Desventajas:

- **Colisiones**: Dos o más categorías diferentes pueden ser asignadas al mismo hash, lo que puede causar pérdida de información.
- **Interpretabilidad Limitada**: La representación resultante no es fácil de interpretar y puede dificultar el análisis posterior.
- **No Captura Relaciones**: Las similitudes o diferencias entre categorías pueden no ser representadas adecuadamente.
- **Rendimiento Dependiente del Hash**: La calidad del modelo puede depender de la función hash utilizada y de cómo se distribuyen los hashes.
- **Dificultad en el Manejo de Desbalance**: En categorías desbalanceadas, algunas podrían no ser representadas adecuadamente en el espacio reducido.

2.7.4 Recomendaciones:

- **Ideal para Variables con Muchas Categorías**: Úsalo cuando se trabaja con variables categóricas que tienen un número elevado de categorías únicas.
- **Análisis Previo de Colisiones**: Antes de implementar, se recomienda analizar cómo la función hash podría generar colisiones.
- **Combinar con Otros Métodos**: Considerar usarlo junto con técnicas como one-hot encoding para capturar mejor la información categórica.
- **Validación y Pruebas**: Probar la efectividad del hash encoding mediante validación cruzada para observar el impacto en el rendimiento del modelo.
- **Uso en Aplicaciones en Tiempo Real**: Ideal para entornos donde la rapidez de codificación es esencial, como aplicaciones en tiempo real que manejan datos en constante cambio.

2.8 8. Embeddings (para aprendizaje profundo)

El método de **embeddings** es ampliamente utilizado en el contexto del aprendizaje profundo para representar variables categóricas. En lugar de usar codificaciones tradicionales, las categorías se mapean a vectores densos en un espacio de menor dimensión. Estos vectores son aprendidos durante el proceso de entrenamiento del modelo, permitiendo que el sistema capture relaciones semánticas entre las categorías.

2.8.1 Ejemplo:

Imagina que estamos trabajando con una base de datos de películas que incluye una columna "Género" con categorías como "Acción", "Comedia", "Drama", "Terror" y "Ciencia Ficción". En lugar de aplicar one-hot encoding o label encoding, se pueden usar embeddings para representar cada género como un vector de 3 dimensiones, así:

Género	Embedding
Acción	[0.8, 0.2, 0.1]
Comedia	[0.3, 0.6, 0.1]
Drama	[0.5, 0.4, 0.1]
Terror	[0.1, 0.3, 0.6]
Ciencia Ficción	[0.6, 0.2, 0.2]

En este caso, el embedding de "Acción" es un vector en un espacio de 3 dimensiones, donde cada dimensión puede capturar aspectos diferentes de la relación del género con otros géneros o características de las películas.

2.8.2 Ventajas:

- **Captura de Relaciones Semánticas**: Los embeddings pueden aprender y representar relaciones complejas entre categorías, lo que puede mejorar la calidad de las predicciones.
- **Reducción de Dimensionalidad**: A diferencia de técnicas como one-hot encoding, los embeddings representan categorías en un espacio de menor dimensión, lo que ahorra memoria y recursos computacionales.
- **Flexibilidad**: Se pueden aprender representaciones personalizadas a medida que el modelo se entrena, adaptándose a los patrones en los datos.
- **Generalización Mejorada**: Facilita la generalización a nuevos datos, ya que el modelo puede entender relaciones implícitas.
- **Compatibilidad con Redes Neuronales**: Los embeddings se integran fácilmente en arquitecturas de redes neuronales, potenciando su rendimiento en tareas complejas.

2.8.3 Desventajas:

- **Requiere Más Datos**: Para entrenar embeddings de manera efectiva, se necesitan grandes volúmenes de datos, lo que puede ser un desafío en conjuntos de datos pequeños.
- **Complejidad**: Implementar embeddings puede ser más complicado que otras técnicas de codificación, requiriendo más recursos computacionales y tiempo de entrenamiento.
- **Dificultad en la Interpretación**: Los vectores de embeddings no son fáciles de interpretar, lo que puede complicar el análisis posterior de los resultados.
- **Sobreajuste**: Existe el riesgo de que el modelo se ajuste demasiado a los datos de entrenamiento si no se maneja adecuadamente, especialmente en conjuntos de datos pequeños.
- **Dependencia de la Arquitectura**: La efectividad de los embeddings puede variar significativamente según la arquitectura de la red neuronal utilizada.

2.8.4 Recomendaciones:

- **Uso en Problemas Complejos**: Ideal para problemas que requieren entender relaciones complejas entre categorías, como el procesamiento de lenguaje natural o la recomendación de productos.
- **Tamaños de Datos Suficientes**: Asegúrate de contar con un conjunto de datos lo suficientemente grande para entrenar los embeddings de manera efectiva.
- **Exploración de Diferentes Dimensiones**: Experimentar con diferentes dimensiones para los vectores de embeddings para encontrar el equilibrio óptimo entre rendimiento y complejidad.
- **Monitoreo del Rendimiento**: Evaluar constantemente el rendimiento del modelo durante el entrenamiento para evitar el sobreajuste y garantizar que los embeddings están aprendiendo patrones útiles.
- **Combinación con Otras Técnicas**: Considerar usar embeddings junto con otras técnicas de codificación o características derivadas para enriquecer la representación de los datos.

2.9 9. Weight of Evidence (WoE)

El método de **Weight of Evidence** (WoE) es comúnmente utilizado en el análisis de riesgo de crédito y en modelos de clasificación binaria. Este enfoque mide la relación entre una categoría y la variable objetivo calculando la diferencia entre las proporciones de la clase objetivo para cada categoría. Se utiliza principalmente para transformar variables categóricas en variables numéricas que reflejen la fuerza de la relación con la variable de interés.

2.9.1 Ejemplo:

Supongamos que estamos analizando datos de clientes para predecir si un cliente es probable que incumpla un préstamo. Tenemos una columna "Estado Civil" con categorías como "Soltero", "Casado" y "Divorciado". Podemos calcular las proporciones de incumplimiento y no incumplimiento de la siguiente manera:

Estado Civil	Proporción de Incumplidores	Proporción de No Incumplidores
Soltero	40% (40 de 100)	60% (60 de 100)
Casado	20% (20 de 100)	80% (80 de 100)
Divorciado	30% (30 de 100)	70% (70 de 100)

A continuación, calculamos el WoE para cada estado civil utilizando la fórmula:

$$WoE = \log\left(\frac{P(\text{Incumplimiento} \mid \text{Estado Civil})}{P(\text{No Incumplimiento} \mid \text{Estado Civil})}\right)$$

Los resultados son:

- $\text{WoE(Soltero)} = \log\left(\frac{0.4}{0.6}\right) \approx -0.41$
- $\text{WoE(Casado)} = \log\left(\frac{0.2}{0.8}\right) \approx -1.39$
- $\text{WoE(Divorciado)} = \log\left(\frac{0.3}{0.7}\right) \approx -0.58$

Estado Civil	WoE
Soltero	-0.41
Casado	-1.39
Divorciado	-0.58

Esto significa que "Casado" tiene una relación más débil con el incumplimiento en comparación con "Soltero" y "Divorciado".

2.9.2 Ventajas:

- **Interpretabilidad**: El WoE proporciona una medida fácil de interpretar sobre la relación de cada categoría con el objetivo, lo que es útil para análisis posteriores.
- **Eficiente para Clasificación Binaria**: Se adapta bien a modelos de clasificación que tienen dos clases, como la regresión logística.
- **Manejo de Desbalance**: Ayuda a manejar conjuntos de datos desbalanceados, donde algunas clases son mucho más frecuentes que otras.
- **Reducción de Sobreajuste**: Puede reducir el riesgo de sobreajuste al transformar categorías en un formato que es menos propenso a capturar ruido.
- **Adaptable**: Puede aplicarse a diferentes tipos de variables categóricas y es útil en diversos contextos, no solo en finanzas.

2.9.3 Desventajas:

- **Requiere Datos Históricos**: Necesita datos históricos significativos para calcular proporciones y realizar análisis efectivos.
- **Potencial de Malinterpretación**: Si no se comprende adecuadamente, puede llevar a malas interpretaciones de los resultados.
- **Dependencia del Conjunto de Datos**: Los resultados pueden ser muy sensibles a los cambios en los datos, lo que puede afectar la estabilidad de los WoE.
- **No Funciona Bien con Niveles Altos de Categorización**: Con un número muy alto de categorías, la interpretación puede volverse complicada y poco práctica.
- **Limitado a Modelos de Clasificación Binaria**: Su aplicabilidad es más restringida en comparación con otros métodos de codificación que pueden funcionar en múltiples contextos.

2.9.4 Recomendaciones:

- **Uso en Modelos de Riesgo de Crédito**: Ideal para análisis de crédito, fraudes y cualquier aplicación donde la variable objetivo sea binaria.
- **Asegúrate de Contar con Suficientes Datos**: Verifica que tengas datos históricos suficientes para que el cálculo de proporciones sea significativo.
- **Combinar con Análisis Exploratorio**: Realiza un análisis exploratorio previo para entender la relación entre las categorías y la variable objetivo.
- **Monitorea el Rendimiento del Modelo**: Observa cómo el uso de WoE impacta el rendimiento del modelo durante la validación cruzada.
- **Considerar el Análisis de Sensibilidad**: Dado que los resultados pueden ser sensibles a cambios en los datos, un análisis de sensibilidad puede ser útil para evaluar la robustez de los resultados.

2.10 10. Polynomial Encoding

La **Polynomial Encoding** es una extensión del one-hot encoding que transforma variables categóricas en un espacio de características polinomiales. A diferencia de la codificación tradicional, donde cada categoría se representa como un vector binario, la codificación polinómica permite que las características resultantes contengan valores intermedios en lugar de solo 1s y 0s. Esto puede ser especialmente útil en modelos que pueden beneficiarse de interacciones más complejas entre categorías.

2.10.1 Ejemplo:

Imaginemos que tenemos una variable categórica "Tamaño" con las categorías ["Pequeño", "Mediano", "Grande"]. Con el one-hot encoding, esto se representaría como:

Tamaño	One-Hot Encoding
Pequeño	[1, 0, 0]
Mediano	[0, 1, 0]
Grande	[0, 0, 1]

Con la codificación polinómica, podríamos mapear estas categorías a características que representen no solo su presencia, sino también interacciones. Si aplicamos un polinomio de grado 2, podríamos generar características como:

Tamaño	Tamaño_Pequeño	Tamaño_Mediano	Tamaño_Grande	Interacción: Tamaño_Pequeño * Tamaño_Mediano	Interacción: Tamaño_Mediano * Tamaño_Grande
Pequeño	1	0	0	0	0
Mediano	0	1	0	0	0
Grande	0	0	1	0	0

Esto permite que el modelo capture relaciones más complejas y no lineales entre las categorías.

2.10.2 Ventajas:

- **Captura Interacciones**: Permite al modelo aprender relaciones no lineales entre las características categóricas.
- **Mayor Flexibilidad**: Aumenta la capacidad del modelo para ajustarse a patrones complejos en los datos.
- **Sencillez en Modelos Avanzados**: Es especialmente útil en algoritmos que pueden beneficiarse de la no linealidad, como árboles de decisión y redes neuronales.
- **Evita la Dimensionalidad Alta de One-Hot**: A diferencia del one-hot encoding, que puede crear muchas columnas, la codificación polinómica puede resultar en un conjunto más manejable.
- **Mejora el Rendimiento**: Puede aumentar la precisión del modelo al permitir que este capture relaciones más sutiles entre categorías.

2.10.3 Desventajas:

- **Complejidad Computacional**: Puede aumentar significativamente la complejidad del modelo y el tiempo de entrenamiento, especialmente con grandes conjuntos de datos.
- **Dificultad de Interpretación**: Las características polinómicas pueden ser más difíciles de interpretar en comparación con las representaciones tradicionales.
- **Riesgo de Sobreajuste**: Al agregar más características derivadas, hay un mayor riesgo de sobreajuste, especialmente en conjuntos de datos pequeños.
- **No Siempre Necesario**: Para datos donde no se esperan interacciones complejas, la codificación polinómica puede ser innecesaria y no aportar beneficios.
- **Requiere Preprocesamiento**: Necesita un proceso de selección de características para identificar cuáles son las interacciones más útiles.

2.10.4 Recomendaciones:

- **Uso en Modelos Avanzados**: Ideal para modelos que pueden capturar relaciones no lineales, como árboles de decisión, bosques aleatorios y redes neuronales.
- **Realizar Análisis de Necesidad**: Antes de implementar, analiza si tus datos requieren la complejidad que ofrece esta codificación.
- **Combinación con Otros Métodos**: Puede ser útil combinarla con otras técnicas de codificación para maximizar el rendimiento del modelo.
- **Cuidado con el Sobrecoste Computacional**: Ten en cuenta los recursos computacionales disponibles, ya que la codificación polinómica puede ser intensiva en cálculos.
- **Evaluar el Rendimiento en Validación Cruzada**: Monitorea el rendimiento del modelo con validación cruzada para asegurar que la complejidad añadida se traduzca en una mejora real.

[1]:
```
# Importar las librerías necesarias
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el conjunto de datos de California Housing
# Este conjunto de datos incluye información sobre casas en California, como␣
 ↪edad, número de habitaciones, etc.
california = fetch_california_housing(as_frame=True)
data = california.frame

data.describe()
```

[1]:
```
              MedInc      HouseAge      AveRooms     AveBedrms    Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
```

```
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

            AveOccup      Latitude     Longitude   MedHouseVal
count   20640.000000  20640.000000  20640.000000  20640.000000
mean        3.070655     35.631861   -119.569704      2.068558
std        10.386050      2.135952      2.003532      1.153956
min         0.692308     32.540000   -124.350000      0.149990
25%         2.429741     33.930000   -121.800000      1.196000
50%         2.818116     34.260000   -118.490000      1.797000
75%         3.282261     37.710000   -118.010000      2.647250
max      1243.333333     41.950000   -114.310000      5.000010
```

3 Ejemplo Usando Label Encoding

[2]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el conjunto de datos de California Housing desde scikit-learn
california = fetch_california_housing(as_frame=True)
data = california.frame
data.head()
```

[2]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	\
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	

	Longitude	MedHouseVal
0	-122.23	4.526
1	-122.22	3.585
2	-122.24	3.521

```
3    -122.25        3.413
4    -122.25        3.422
```

[3]:
```
# Definir los cuantiles del 20% para 'MedHouseVal' para crear quintiles
quantiles = [data['MedHouseVal'].quantile(i / 5) for i in range(1, 5)]  #␣
 ↪Cuantiles del 20%

# Definir etiquetas para las 5 categorías resultantes
labels = ['Muy Bajo', 'Bajo', 'Medio', 'Alto', 'Muy Alto']

# Definir los límites de los bins para clasificar usando los quintiles
num_bins = [data['MedHouseVal'].min()] + quantiles + [data['MedHouseVal'].max()]

# Crear una nueva columna 'MedHouseVal_Category' que categoriza los valores de␣
 ↪MedHouseVal
data['MedHouseVal_Category'] = pd.cut(data['MedHouseVal'],
                                      bins=num_bins,
                                      labels=labels,
                                      include_lowest=True)

data.head()
```

[3]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	\
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	

	Longitude	MedHouseVal	MedHouseVal_Category
0	-122.23	4.526	Muy Alto
1	-122.22	3.585	Muy Alto
2	-122.24	3.521	Muy Alto
3	-122.25	3.413	Muy Alto
4	-122.25	3.422	Muy Alto

[4]:
```
# Aplicar Label Encoding a 'MedHouseVal_Category' para convertir las categorías␣
 ↪en números
label_encoder = LabelEncoder()
data['MedHouseVal_Label_Encoding'] = label_encoder.
 ↪fit_transform(data['MedHouseVal_Category'])
data.head()
```

[4]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	\
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	

```
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal MedHouseVal_Category  MedHouseVal_Label_Encoding
0    -122.23        4.526             Muy Alto                           3
1    -122.22        3.585             Muy Alto                           3
2    -122.24        3.521             Muy Alto                           3
3    -122.25        3.413             Muy Alto                           3
4    -122.25        3.422             Muy Alto                           3
```

[5]:
```
# Separar las características (X) y la variable objetivo (y)
X = data.drop(['MedHouseVal', 'MedHouseVal_Category',␣
 ↪'MedHouseVal_Label_Encoding'], axis=1)
y = data['MedHouseVal_Label_Encoding']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Entrenamiento del modelo de regresión lineal
model = LinearRegression()
model.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = model.predict(X_test)

# Evaluar el rendimiento del modelo usando R^2 y RMSE
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'R^2: {r2:.4f}')  # Coeficiente de determinación
print(f'RMSE: {rmse:.4f}')  # Raíz del error cuadrático medio
```

```
R^2: 0.0989
RMSE: 1.3414
```

[ ]:
```
# Ajustar las predicciones para que estén dentro del rango de los quintiles
y_pred_clipped = np.clip(y_pred, num_bins[0], num_bins[-1])

# Clasificar las predicciones ajustadas en categorías de quintiles
y_pred_categories = pd.cut(y_pred_clipped, bins=num_bins, labels=label_encoder.
 ↪classes_)

# Asegurarse de que no haya NaN en las predicciones
valid_indices = ~y_pred_categories.isna()  # Obtener los índices de␣
 ↪predicciones válidas
```

```
# Filtrar las predicciones y las etiquetas reales según los índices válidos
y_pred_valid = y_pred_categories[valid_indices]
y_test_valid = y_test.iloc[valid_indices]  # Asegurarse de que las predicciones
 ↪y y_test coincidan

# Usar LabelEncoder para convertir las categorías de predicciones en números
y_pred_encoded = label_encoder.transform(y_pred_valid)

# Matriz de confusión basada en las categorías de las predicciones y los
 ↪valores reales
conf_matrix = confusion_matrix(y_test_valid, y_pred_encoded)

# Visualización de la matriz de confusión
plt.figure(figsize=(10, 7))  # Ajustar el tamaño de la figura
sns.set(font_scale=1.4)  # Aumentar el tamaño de la fuente para mejor
 ↪legibilidad
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_,
            cbar_kws={'label': 'Número de Predicciones'})  # Etiqueta para la
 ↪barra de color

# Añadir etiquetas y título a la gráfica
plt.xlabel('Predicción (Categorías de Valor de Casa)', fontsize=12)
plt.ylabel('Realidad (Categorías de Valor de Casa)', fontsize=12)
plt.title('Matriz de Confusión: Predicciones vs Realidad', fontsize=16)
plt.xticks(rotation=45)  # Rotar etiquetas del eje X para mejor visibilidad
plt.yticks(rotation=0)   # Mantener etiquetas del eje Y en horizontal
plt.show()
```