Universidad Alfonso X el Sabio

cuantización llm de 4 bits

Germán Llorente y Carlos Puigserver

November 4, 2023

November 4, 2023

1 Experiencia con Aprendizaje Automático y Pandas en Proyectos

Como estudiante de ingeniería matemática, hemos tenido la oportunidad de trabajar con aprendizaje automático y Pandas en varios proyectos. En el pasado, hemos creado funciones combinando columnas en conjuntos de datos utilizando Pandas. Esto es especialmente útil cuando necesitas crear nuevas características para entrenar modelos de aprendizaje automático. Pandas ofrece una amplia gama de funciones para manipular y transformar datos, lo que facilita la creación de nuevas columnas derivadas de las existentes.

En cuanto a los problemas de rendimiento, sí, hemos enfrentado desafíos al procesar grandes volúmenes de datos. Cuando trabajas con conjuntos de datos extensos, es fundamental optimizar el código para evitar problemas de memoria y tiempo de ejecución. Una gran parte de nuestro aprendizaje en este campos ha residido en la limpieza de datos, teniendo una asignatura en el primer sño acerca de este tema y todo un curso (Ironhack) para profundizar en segundo año. En algunos casos, hemos tenido que considerar técnicas como el muestreo aleatorio para reducir el tamaño del conjunto de datos y hacer que sea más manejable para el análisis y el entrenamiento de modelos. Además, he aprendido sobre técnicas de optimización, como el uso adecuado de índices en Pandas y el empleo de operaciones vectorizadas para mejorar la eficiencia en el manejo de datos grandes.

2 Iteración a través de Filas en un DataFrame de Pandas

Pandas es una poderosa biblioteca de Python para el análisis de datos que proporciona una estructura de datos llamada DataFrame para manejar datos

tabulares. En el contexto de Pandas, hay varias técnicas para iterar a través de las filas de un DataFrame, cada una con sus propias ventajas y desventajas. A continuación, se presentan algunas de las técnicas más comunes para iterar filas en un DataFrame de Pandas:

2.1 Iteración con `iterrows()`

```
for index, row in df.iterrows():
    # Acceder a los valores de cada fila
    print(row['Columna1'], row['Columna2'])
```

Ventajas:

- Fácil de entender y usar.

Desventajas:

- Relativamente lento para DataFrames grandes debido a su naturaleza basada en Python y la necesidad de convertir cada fila en un objeto `Series`.
- No es adecuado para la modificación en su lugar de los datos del DataFrame original.

2.2 Usando `apply()`

```
def procesar_fila(row):
    # Acceder a los valores de cada fila
    return row['Columna1'] * 2, row['Columna2'] + 10

df[['Columna1', 'Columna2']] = df.apply(procesar_fila, axis=1)
```

Ventajas:

- Permite aplicar funciones personalizadas a las filas.
- Puede ser más eficiente que `iterrows()` para ciertas operaciones.

Desventajas:

- Puede ser más lento que otras técnicas vectorizadas, especialmente para grandes DataFrames.

2.3 Usando `iteritems()`

```
for column, values in df.iteritems():
    # Acceder a los valores de cada columna
    print(column)
    for value in values:
        print(value)
```

Ventajas:

- Útil para iterar a través de columnas y acceder a valores específicos.

Desventajas:

- No es adecuado para iterar filas.

2.4 Iteración Vectorizada

```
# Operaciones vectorizadas en columnas
df['Columna1'] = df['Columna1'] * 2
df['Columna2'] = df['Columna2'] + 10
```

Ventajas:

- Más eficiente y rápida que las técnicas basadas en bucles para operaciones elementales en columnas.
- Aprovecha la optimización de operaciones vectorizadas de NumPy.

Desventajas:

- Limitado a operaciones que se pueden vectorizar.

2.5 Usando `numpy.nditer()`

```
import numpy as np

for index, row in np.ndenumerate(df.values):
    # Acceder a los valores de cada fila
    print(row)
```

Ventajas:

- Utiliza la eficiente biblioteca NumPy para la iteración, que es más rápida que las técnicas puramente basadas en Python.

Desventajas:

- Similar a `iterrows()`, puede ser más lento para DataFrames grandes debido a su naturaleza basada en Python.

2.6 Usando `df.itertuples()`

```
for row in df.itertuples():
    # Acceder a los valores de cada fila
    print(row.Index, row.Columna1, row.Columna2)
```

Ventajas:

- Más rápido que `iterrows()` debido a que devuelve namedtuples que son más ligeros y más rápidos que las Series.

Desventajas:

- Aún puede ser más lento para DataFrames grandes comparado con operaciones vectorizadas.

2.7 Usando `df.eval()`

```
for index in range(len(df)):
    # Acceder a los valores de cada fila
    print(df.eval('Columna1.iloc[{}] * 2'.format(index)),
          df.eval('Columna2.iloc[{}] + 10'.format(index)))
```

Ventajas:

- Permite evaluar expresiones de forma dinámica para cada fila.

Desventajas:

- La sintaxis puede volverse complicada para expresiones complejas.
- Puede ser menos eficiente que las operaciones vectorizadas para grandes DataFrames.

2.8 Vectorización de Pandas: Acelerando las Operaciones con Datos

El enfoque de vectorización de Pandas implica el uso de operaciones vectorizadas para realizar cálculos en grandes conjuntos de datos. En lugar de agregar valores únicos uno por uno, los datos se agrupan en vectores para realizar operaciones de manera más eficiente. A continuación, se detallan las ventajas y desventajas de este enfoque:

2.8.1 Ventajas:

- **Velocidad Aumentada:** La vectorización de Pandas puede ser significativamente más rápida que enfoques iterativos como 'iterrows()'. En el ejemplo proporcionado, la vectorización es 1500 veces más rápida, lo que se traduce en una mejora significativa del rendimiento.
- **Aprovechamiento de la CPU:** Pandas puede procesar objetos de Series en paralelo, utilizando todos los núcleos de CPU disponibles. Esto permite una distribución eficiente del trabajo, aprovechando al máximo la capacidad de procesamiento de la computadora.
- **Sintaxis Intuitiva:** La sintaxis para operaciones vectorizadas es sencilla y fácil de entender. Al usar operaciones directas en columnas o Series de Pandas, se pueden realizar cálculos complejos de manera clara y concisa.
- **Optimización Interna:** Pandas se encarga de la vectorización internamente, utilizando código C optimizado y bloques de memoria contiguos para acelerar las operaciones. Esto garantiza una ejecución rápida y eficiente de las operaciones vectorizadas.

2.8.2 Desventajas:

- **Limitaciones en la Complejidad:** Aunque las operaciones vectorizadas son poderosas, algunas operaciones complejas pueden ser difíciles de expresar de manera vectorizada. En tales casos, es posible que se necesite recurrir a enfoques más complejos o iterativos para lograr el resultado deseado.
- **Requiere Familiaridad:** Para aprovechar al máximo la vectorización de Pandas, los usuarios deben estar familiarizados con las operaciones y

métodos proporcionados por la biblioteca. Esto puede requerir un tiempo de aprendizaje para aquellos que son nuevos en el uso de Pandas.

A continuación se presenta un ejemplo que ilustra cómo utilizar la vectorización en Pandas para realizar operaciones de suma en columnas de un DataFrame.

```
import pandas as pd
import timeit

# Crear un DataFrame de ejemplo
data = {
    'src_bytes': [100, 200, 300, 400],
    'dst_bytes': [50, 150, 250, 350]
}
df = pd.DataFrame(data)

# Enfoque de iterrows() para sumar las columnas
def iterrows_sum():
    result = []
    for index, row in df.iterrows():
        result.append(row['src_bytes'] + row['dst_bytes'])
    return result

# Enfoque de vectorización para sumar las columnas
def vectorized_sum():
    return df['src_bytes'] + df['dst_bytes']

# Medir el tiempo de ejecución para ambas técnicas
iterrows_time = timeit.timeit(iterrows_sum, number=1000)
vectorized_time = timeit.timeit(vectorized_sum, number=1000)

# Imprimir resultados
print(f'Tiempo usando iterrows(): {iterrows_time:.6f} segundos')
print(f'Tiempo usando vectorización: {vectorized_time:.6f} segundos')
```

Los resultados del tiempo de ejecución del código son los siguientes:

```
Tiempo usando iterrows(): 0.030567 segundos
```

```
Tiempo usando vectorización: 0.002424 segundos
```

Como se puede observar, el enfoque de vectorización es considerablemente más rápido que el enfoque de `iterrows()`, lo que demuestra la eficacia de la vectorización en operaciones sobre DataFrames en Pandas. En el repositorio se ha adjuntado un cuaderno de Google Colab en el que se emplean todas estas técnicas con un dataset de Pandas. Los resultados se muestran en el mismo documento, yendo estos acorde a lo expuesto previamente.

En resumen, la vectorización de Pandas ofrece una forma poderosa y eficiente de realizar operaciones en grandes conjuntos de datos. Al aprovechar las operaciones vectorizadas, los usuarios pueden lograr mejoras significativas en el rendimiento y la velocidad de ejecución, especialmente en comparación con enfoques iterativos más lentos como 'iterrows()'.

3 Reflexiones sobre la Iteración en Pandas

Reflexionar sobre los hallazgos relacionados con la iteración en Pandas me ha proporcionado una comprensión más profunda sobre la importancia de elegir las técnicas de iteración adecuadas para manipular y analizar datos tabulares. Antes de esta investigación, solía depender principalmente de técnicas de iteración como `iterrows()` y `apply()`, sin cuestionar su eficiencia en términos de rendimiento. Sin embargo, ahora comprendo que estas técnicas pueden ser subóptimas para conjuntos de datos grandes debido a su naturaleza basada en bucles y su lentitud en comparación con las operaciones vectorizadas.

El aprendizaje más valioso que he obtenido es la importancia de la vectorización en Pandas y NumPy. La vectorización permite realizar operaciones en columnas o filas enteras de un DataFrame de manera eficiente y rápida, eliminando la necesidad de bucles explícitos. Esta técnica no solo mejora el rendimiento de los cálculos, sino que también hace que el código sea más claro y conciso. Al comprender y aplicar la vectorización, he adquirido una herramienta poderosa para el análisis de datos y la manipulación eficiente de grandes conjuntos de datos.

Mirando hacia el futuro como científico de datos, este conocimiento me permitirá abordar problemas más complejos y manejar conjuntos de datos más grandes y variados. La capacidad de elegir y aplicar las técnicas de iteración y vectorización adecuadas será crucial para optimizar mis análisis y modelos, lo que, a su vez, mejorará mi productividad y eficacia en el campo de la ciencia de

datos.

Además, esta experiencia también me ha enseñado la importancia de cuestionar las prácticas habituales y buscar constantemente formas más eficientes y efectivas de abordar problemas de análisis de datos. La tecnología y las herramientas están en constante evolución, y como científico de datos, es esencial mantenerse al tanto de las mejores prácticas y técnicas actuales para ser un profesional competente y efectivo en este campo dinámico y en constante cambio.