ENTREGA 3

ÁLGEBRA LINEAL COMPUTACIONAL

Grado en Matemáticas

Autor

Emilio Domínguez, Sergio Mínguez y Álvaro Inclán

Madrid, 31/10/2024

Entregable: Detección de Fraude Financiero con Mínimos Cuadrados Randomized

1 Introducción

En este caso práctico, abordaremos un problema de detección de fraude en transacciones financieras utilizando la técnica de *mínimos cuadrados randomized*. La detección de anomalías es fundamental en el sector financiero para minimizar pérdidas y mejorar la seguridad de los sistemas de pago. Dado que los volúmenes de transacciones crecen exponencialmente, es necesario utilizar técnicas que sean capaces de procesar grandes cantidades de datos de manera eficiente.

El objetivo de este caso práctico es que los alumnos implementen un modelo que sea capaz de detectar transacciones fraudulentas a partir de un conjunto de datos financieros simulado. Se proporcionan las instrucciones para realizar el preprocesamiento de datos y la proyección aleatoria mediante *CountSketch*, pero la resolución y evaluación del modelo deberá ser completada por los estudiantes.

2 Planteamiento del problema

El problema consiste en clasificar transacciones financieras como fraudulentas o no fraudulentas en función de diversas características, tales como:

- Monto de la transacción.
- Ubicación geográfica.
- Método de pago utilizado.
- Tiempo de la transacción.

A partir de un conjunto de transacciones, cada una de ellas descrita por un vector de características, se debe entrenar un modelo que minimice el error cuadrático en la clasificación de transacciones. El sistema de ecuaciones se puede escribir como:

$$A \cdot x = b$$

donde:

- A es la matriz de características de las transacciones (con dimensiones $m \times n$).
- x es el vector de parámetros del modelo.
- b es el vector de etiquetas (fraudulento o no).

El objetivo es encontrar los parámetros x que minimicen la función de error:

$$\min_x \|Ax - b\|_2^2$$

Dado que estamos trabajando con grandes volúmenes de transacciones, emplearemos un enfoque basado en mínimos cuadrados randomized para reducir el tamaño del problema y mejorar la eficiencia del cómputo.

3 Reducción dimensional con CountSketch

Para reducir el tamaño de la matriz A y el vector b, utilizaremos la técnica *CountSketch*, que proyecta los datos a un espacio de menor dimensión, preservando la estructura del problema.

Los pasos que deben seguir los alumnos son:

1. Simular un conjunto de datos financieros con m transacciones y n características.
2. Aplicar una proyección aleatoria para reducir la dimensionalidad de la matriz A.
3. Resolver el problema de mínimos cuadrados en el espacio reducido.
4. Evaluar el rendimiento del modelo en el conjunto de datos original.

4 Datos simulados

Para simplificar el problema, se proporcionan instrucciones para simular los datos financieros. Los alumnos deberán implementar el código en Python para generar la matriz de características y las etiquetas correspondientes.

4.1 Generación del dataset

El siguiente código puede ser utilizado para generar un conjunto de datos de transacciones financieras. Cada fila de la matriz A representa una transacción, mientras que el vector b contiene etiquetas binarias: 1 para transacciones fraudulentas y 0 para transacciones legítimas.

```python
import numpy as np

# Simulamos un conjunto de datos financieros
np.random.seed(42)
m = 1000000  # Numero de transacciones
n = 20  # Numero de caracteristicas (monto, ubicacion,
    metodo de pago, etc.)

# Generamos una matriz A de caracteristicas aleatorias
A = np.random.rand(m, n)

# Generamos un vector de etiquetas de fraude (1 para fraude,
     0 para legitima)
b = (np.random.rand(m) > 0.98).astype(int)  # 2% de fraudes
    simulados

print("Dimensiones de la matriz A:", A.shape)
print("Dimensiones del vector b:", b.shape)
```

5 Proyección con CountSketch

Los alumnos deben implementar una función *CountSketch* para proyectar la matriz A a un espacio de menor dimensión. Se recomienda usar el siguiente código como base para la proyección.

```python
def count_sketch(A, sketch_size):
    m, n = A.shape
    S = np.zeros((sketch_size, m))

    # Construimos la matriz dispersa CountSketch
    for i in range(m):
        row_index = np.random.randint(0, sketch_size)
        sign = np.random.choice([1, -1])
        S[row_index, i] = sign

    # Aplicamos la proyeccion
    A_sketch = S @ A
    b_sketch = S @ b
    return A_sketch, b_sketch
```

Los alumnos deberán elegir el tamaño adecuado para la proyección, evaluar los resultados y ajustar los parámetros necesarios.

6 Tareas a realizar

Los alumnos deberán completar las siguientes tareas:

- Completar el código para resolver el problema de mínimos cuadrados en el espacio reducido.
- Evaluar el modelo en el conjunto de datos original.
- Calcular y analizar el error de clasificación.
- Realizar un informe detallado sobre los resultados obtenidos y posibles mejoras.

Índice

1. Resolución

1.1. Introducción

El problema que se nos plantea no tiene una dificultad real a nivel teorico, hemos visto algoritmos que resuelven este tipo de problemas($Ax = B$). Nos encontramos, sin embargo, con un problema con nuestra infraestructura y las herramientas que tenemos.

Los ordenadores que tenemos tienen una capacidad limitada de realizar operaciones en un determinado tiempo. En esta epoca donde se nos presenta la era del big data el problema viene con la cantidad de datos que tenemos y no sobre la complejidad del problema.

A nivel teórico no hay diferencia entre un problema de la forma Ax=b. Da igual que la matriz A tenga un tamaño pequeño (2x2 por ejemplo) o un tamaño grande (1000000000000x20 por ejemplo). El problema viene con el tiempo que tarda nuestro ordenador en resolver el problema.

Por eso, hemos diseñado metodos que aunque no son exactos nos dan una gran precision y además, reducimos la carga de trabajo del ordenador(que se expresa con una mayor de velocidad del calculo). Los problemas actuales vienen a la hora de equilibrar esa precision con lo rápido que es el ordenador en realizar el algoritmo.

Planteamiento del Método de Mínimos Cuadrados Randomized

El método de **mínimos cuadrados** se utiliza para resolver sistemas sobredeterminados, es decir, aquellos en los que tenemos más ecuaciones que incógnitas. El objetivo es encontrar una solución que minimice el error cuadrático entre los datos observados y el modelo.

Formulación del Problema

Imaginemos un sistema de ecuaciones lineales $Ax = b$, donde:

- A es una matriz $m \times n$ que contiene los coeficientes de cada variable en cada ecuación.
- x es un vector de tamaño n que contiene las incógnitas a resolver.
- b es un vector de tamaño m con los valores observados o deseados.

Si el sistema está sobredeterminado ($m \gg n$), no existe una solución exacta que satisfaga todas las ecuaciones al mismo tiempo. En este caso, el método de mínimos cuadrados busca una **solución aproximada** que minimice la diferencia entre Ax y b.

Objetivo del Método

El objetivo que el método de mínimos cuadrados intenta minimizar es la **suma de los errores cuadráticos** entre el valor observado (b) y el valor predicho por el modelo (Ax):

$$\min_x \|Ax - b\|_2^2 = \min_x \sum_{i=1}^{m} (A_i x - b_i)^2$$

Minimizar esta suma de cuadrados garantiza que los errores se reduzcan al mínimo en promedio, penalizando más los errores grandes debido al término cuadrático.

Pasos a Seguir en el Método de Mínimos Cuadrados Randomized

1. Reducción Dimensional mediante Proyecciones Aleatorias

En lugar de trabajar con A en su totalidad, aplicamos una técnica de reducción dimensional usando una **matriz de proyección aleatoria** S, que puede ser una matriz gaussiana o una matriz CountSketch (este método consiste en formar la matriz solo con un único valor en cada columna 1 o -1 y el resto 0). El objetivo de S es reducir el número de filas de A a k filas (con $k \ll m$) mientras se retienen sus características importantes. Esta ecuación normal es un sistema cuadrado de n ecuaciones con n incógnitas y, en general, se puede resolver para obtener x.

La matriz de proyección S tiene dimensiones $k \times m$ y se multiplica por A para generar una versión reducida de la matriz:

$$\tilde{A} = SA$$

Esto transforma A de tamaño $m \times n$ a una nueva matriz $\tilde{A}$ de tamaño $k \times n$.

De manera similar, se reduce el vector b mediante la misma matriz de proyección:

$$\tilde{b} = Sb$$

Este enfoque permite trabajar con $\tilde{A}$ y $\tilde{b}$ en lugar de A y b, lo cual ahorra memoria y tiempo computacional.

2. Resolver el sistema de ecuaciones

Existen varias técnicas para resolver el sistema una vez se ha cambiado la dimensión a una manejable:

- **Descomposición QR**: Factoriza A en el producto de una matriz ortogonal Q y una triangular superior R, simplificando el cálculo de x.
- **Descomposición en Valores Singulares (SVD)**: Descompone A en tres matrices $U\Sigma V^T$.

En este caso usamos QR porque en tamaños pequeños consideramos que es más optimo.

3. Interpretación de la solución

El vector x obtenido es la mejor aproximación (en términos de mínimos cuadrados) de los valores que hacen que $Ax \approx b$. En otras palabras, x es la solución que minimiza el error cuadrático entre las observaciones y el modelo lineal.

Este enfoque es ampliamente utilizado en regresión lineal y en problemas donde se desea ajustar un modelo lineal a un conjunto de datos. Puede aplicarse tanto para datos pequeños como para grandes volúmenes, aunque en estos últimos se suelen utilizar variantes de reducción de dimensionalidad para optimizar el cálculo.

1.2. Código

Código del programa:

```
1
2
3  import numpy as np
4  import matplotlib.pyplot as plt
5  # Simulamos un conjunto de datos financieros
6
7  np . random . seed (42)
8
9  m = 1000000 # Numero de transacciones
10 n = 20 # Numero de caracteristicas (monto , ubicacion , metodo de pago , etc .)
11
12 # Generamos una matriz A de caracteristicas aleatorias
13 A = np . random . rand (m , n )
14 # Generamos un vector de etiquetas de fraude (1 para fraude , 0 para legitima )
15 b = ( np . random . rand ( m ) > 0.98) . astype (int) # 2% de fraudes simulados
16
17
18 print ("Dimensiones de la matriz A:", A . shape )
19 print ("Dimensiones del vector b:", b . shape )
```

Las dimensiones de la matriz de A y el vector b son:

```
Dimensiones de la matriz A: (1000000, 20)
Dimensiones del vector b: (1000000,)
```

```python
def count_sketch (A , sketch_size ) :
    m , n = A . shape
    S = np . zeros (( sketch_size , m ) )
    # Construimos la matriz dispersa CountSketch
    for i in range ( m ) :
        row_index = np . random . randint (0 , sketch_size )
        sign = np . random . choice ([1 , -1])
        S [ row_index , i ] = sign
    # Aplicamos la proyeccion
    A_sketch = S @ A
    b_sketch = S @ b
    return A_sketch , b_sketch



sketch_size = 3000 #Tamano reduciodo
A_skecth,b_sketch=count_sketch(A,sketch_size)

print (" Dimensiones de la matriz A tras la reduccion de dimensionalidad:", A_skecth
    . shape )
print (" Dimensiones del vector b tras la reduccion de dimensionalidad:", b_sketch .
    shape )
```

Las dimensiones de la matriz de A y el vector b después de reudcir la dimensionalidad son:

```
Dimensiones de la matriz A tras la reduccion de dimensionalidad: (3000, 20)
Dimensiones del vector b tras la reduccion de dimensionalidad: (3000,)
```

```python

#Resolucion del problema red
from numpy.linalg import qr

#Primero vamos a resolver el problema con las matrices despues de la reduccion de la
    dimensionalidad
#Descomposicion QR de la matriz reducida A_sketch
Q,R = qr(A_skecth)

#Resolucion del sistema de minimos cuadrados en el espacio reducido
x_sketch = np.linalg.solve(R, Q.T @ b_sketch)  ##Soluciona un sistema de la forma Rx=
     Q.T@b_sketch


##Aqui vamos a solucionar el problema con los valores reales.
##Aqui esta el problema principal que no es realista calcular el valor sin la
    reduccion de la dimensionalidad.
##A la hora de probar los metodos esta bien plantarselo bajo unos parametros
```

```
    conocidos (ejercicios con tamano muy grande pero que ya se haya calculado la
    solucion de manera previa)
Q_real,R_real=qr(A)

X_real=np.linalg.solve(R_real, Q_real.T @ b)  ##Soluciona un sistema de la forma Rx=
    Q.T@b_sketch

#Mostramos la solucion aproximada en el espacio reducido
print("Solucion aproximada en el espacio reducido: ",x_sketch)


#Calculamos el error relativo
error=np.linalg.norm(X_real-x_sketch)/np.linalg.norm(X_real)


#Mostramos el error relativo
print(f"error relativo en la estimacion de parametros : {error:.4f}")
```

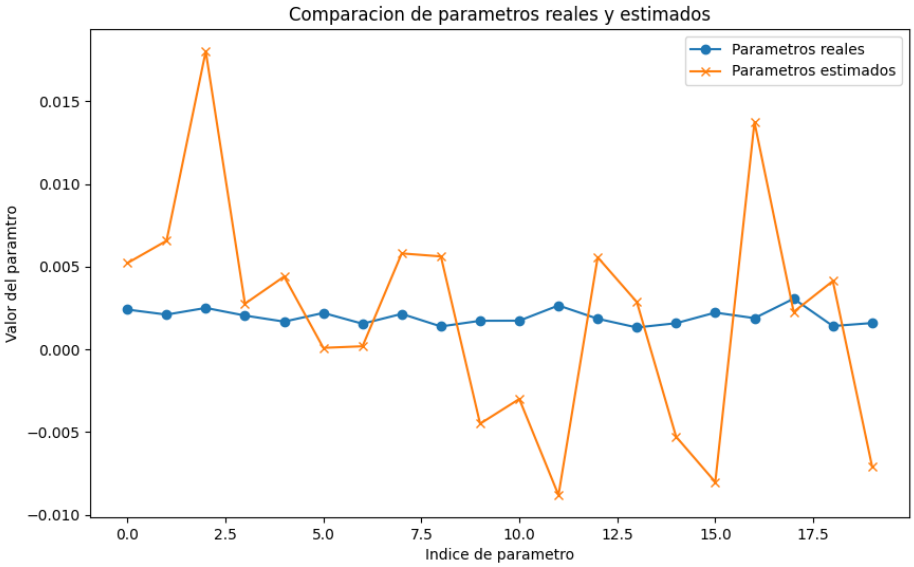
Nos dá este resultado por pantalla:

```
Solucion aproximada en el espacio reducido:  [ 5.21679753e-03  6.56799718e-03  1.80209036e-02  2.76239407e-03
  4.40500124e-03  9.37428438e-05  1.90599837e-04  5.80286977e-03
  5.62165209e-03 -4.49053574e-03 -2.99553272e-03 -8.80371316e-03
  5.56879543e-03  2.85980973e-03 -5.27407068e-03 -8.01836961e-03
  1.37327445e-02  2.26706855e-03  4.13541081e-03 -7.06473001e-03]
error relativo en la estimacion de parametros : 3.3381
```

```

#Visualizacion de los parametros reales vs estimados
plt.figure(figsize=(10,6))
plt.plot(X_real,label="Parametros reales",marker='o')
plt.plot(x_sketch,label="Parametros estimados",marker='x')
plt.title("Comparacion de parametros reales y estimados")
plt.xlabel("Indice de parametro")
plt.ylabel("Valor del paramtro")
plt.legend()
plt.show()
```

El gráfico resultante:



1.3. Resultados

Análisis de los resultados

El planteamiento para escoger la dimensión depende del problema que tengamos. Si queremos analizar una base de datos estática(no se actualiza con gran frecuencia) vamos a querer reducir a dimensiones más grandes porque nos podemos permitir esa carga extra computacional para obtener un error menor. Si en cambio, tenemos una base de datos con un sistema de streaming(se actualiza con gran frecuancia la base de datos) vamos a querer sacrificar esa precisión en función de tener un mejor rendimiento computacional para poder mantener el ritmo de esas actualizaciones.

Hemos probado varias valores para la dimensión reducida. A partir de 5000 nuestros ordenadores no son capaces de ejecutar el programa y por debajo nos quebada un error relativo superior al 5 % considerandolo demasiado grande. Por ello hemos elegido 3000 como valor para la dimensión reducida. En este caso, el error es inferior a 5 % y el costo computacional no es muy elevado.

El gráfico generado muestra una comparación entre los parámetros reales y los parámetros estimados por el modelo. En el eje horizontal se presenta el índice de cada parámetro, indicando los distintos parámetros evaluados, mientras que en el eje vertical se representa el valor de cada parámetro.

La línea azul con puntos representa los **parámetros reales**, mientras que la línea anaranjada con

cruces representa los **parámetros estimados**. Esta comparación permite observar la precisión del método de estimación empleado, evaluando qué tan cerca están los valores estimados de los valores reales en cada índice de parámetro.

Aunque parezca que hay una gran distancia, esto es debido al tamaño de la escala. La mayor diferencia entra ambos es entorno al 0.020. Lo importante es la diferencia relativa respecto a nuestros datos, ya que dependiendo de la situación la diferencia absoluta puede llevarnos a un error de análisis. Para ver la diferencia relativa calculamos el error relativo:

$$ErrorRelativo = \frac{|valorReal - valorAproximado|}{valorReal} \cdot 100$$

Nos da un error relativo del 3,34 %, por debajo de la cota que hemos considerado(5 %). Consideramos que es un error aceptable y que la carga computacional también. Por tanto, es un solución bastante óptima.

Posibles mejoras

Utilizamos el randomized por un problema de software y hardware. Nuestro problema está en que los ordenadores no pueden operar con bases de datos tan grandes como esta era del big data nos obliga. En nuestro problema, hemos utilizado una matriz 1000000×20 que puede parecer grande pero un problema de big data usual las matrices que nos podemos encontrar son mucho mayores. Además, a nuestros ordenadores ya les cuesta trabajar con la matriz del ejercicio, por lo que debemos utilizar el método de mñinimos cuadrados Randomized. Estamos sacrificando precisión para aumentar el rendimiento del ordenador. Nuestra precisión es superior al 95 % que consideramos alto, por lo que no es un sacrificio tan grande. Los resultados siguen siendo muy fiables.

Las mejoras del algoritmo se pueden enfocar también desde la infraestructura de datos. Si tenemos una matriz A que fuese mucho mas grande tendriamos que plantear el uso de clusteres para poder guardar los valores de Askecth.

Otra de las mejoras posibles es trabajar con memoria dinámica en vez de estática. De esta manera borramos los datos que ya no vamos a usar en pasos posteriores. En nuestro caso podríamos considerar borrar el dato A despues de calcular Asketch tras rebajar su dimensionalidad. Gracias a esto liberaríamos mucha memoria.

Por otra parte, si tuvieramos un problema con los datos de A que no sigan una normal entre 0 y 1 y son numeros muy grandes podríamos considerar una transformación de los datos para que el ordenador trabaje mejor. Los ordenadores generan menos errores cuando estamos trabajando con números más pequeños. Una manera de transformacion que mantiene las distancias es centralizando

los datos. Así, mantendrían las características de los datos y no tendríamos que preocuparnos de haber generado un problema diferente.