Procesamiento de Grandes Volúmenes de Datos

Katherine Rodríguez Rodríguez Reinaldo Cánovas Gamón

Proyecto 2: Análisis Histórico de Datos Financieros

Contents

1 Introducción

El análisis de datos es una disciplina que ha experimentado un notable crecimiento en todos los campos, así como en organizaciones y empresas, debido a la necesidad de contar con herramientas que analicen datos y que estos sirvan para la toma de decisiones eficaces y eficientes.
Los datos han crecido de manera exponencial en las últimas décadas, lo que ha generado una serie de problemas, como el acceso a esa cantidad de datos, su extracción y almacenamiento. Por lo tanto, se introduce aquí la definición de *Big Data*.
Big data se refiere a conjuntos de datos extremadamente grandes y complejos que no pueden gestionarse ni analizarse fácilmente con las herramientas tradicionales de procesamiento de datos.

Este trabajo no solo pretende abordar los desafíos asociados con la gestión de grandes volúmenes de datos, sino también demostrar su aplicación práctica a través de un análisis histórico de datos financieros, ofreciendo a analistas e inversores una comprensión profunda de los mercados financieros y respaldando estrategias de inversión informadas y efectivas. Una de las grandes empresas tecnológicas de los Estados Unidos la cual es una de las que alberga más patentes es la IBM (International Business Machines Corporation). El análisis financiero se centrará alrededor de esta empresa.

2 IBM

International Business Machines Corporation (IBM) es una empresa tecnológica multinacional estadounidense. IBM fabrica y comercializa hardware y software para computadoras, y ofrece servicios de infraestructura, alojamiento de Internet, y consultoría en una amplia gama de áreas relacionadas con la informática, desde computadoras centrales hasta nanotecnología.

3 Ingesta de Datos

3.1 API Alphavantage

Para realizar la extracción de datos históricos sobre finanzas, se utiliza la API Alpha Vantage. Esta API devuelve la cadena de opciones histórica completa para un símbolo específico en una fecha determinada, abarcando más de 15 años de historia. También proporciona la volatilidad implícita (IV) y los griegos comunes (por ejemplo, delta, gamma, theta, vega y rho). Las cadenas de opciones se ordenan por fechas de vencimiento en orden cronológico. Dentro de la misma fecha de vencimiento, los contratos se organizan por precios de ejercicio de menor a mayor.

.La volatilidad implícita es una medida de la expectativa del mercado derivada de su precio. Se muestra el número total de contratos de opciones que están actualmente abiertos y no han sido ejercidos ni cerrados, el número total de contratos negociados durante un período específico. Un volumen más alto puede indicar mayor interés en la opción, el último precio al que se negoció el contrato de opción, el precio marcado o estimado del contrato, que puede ser utilizado como referencia para evaluar su valor actual y el precio más alto que un comprador está dispuesto a pagar por el contrato..
.Los "griegos" son medidas que describen cómo se espera que los precios de las opciones respondan a diferentes factores del mercado:

Delta: Mide la sensibilidad del precio de la opción a cambios en el precio del activo subyacente.

Gamma: Mide la tasa de cambio de Delta sobre el precio del activo subyacente.

Theta: Representa la tasa de cambio del precio de la opción con respecto al tiempo.

Vega: Mide la sensibilidad del precio de la opción a cambios en la volatilidad del activo subyacente.

Rho: Mide la sensibilidad del precio de la opción a cambios en la tasa de interés.

3.2 Apache Kafka

Kafka es un sistema distribuido que consta de servidores y clientes que se comunican a través de un protocolo de red TCP de alto rendimiento. Puede implementarse en hardware sin sistema operativo, máquinas virtuales y contenedores, tanto en entornos locales como en la nube.

Servidores: Kafka se ejecuta como un clúster de uno o más servidores que pueden abarcar varios centros de datos o regiones en la nube. Algunos de estos servidores forman la capa de almacenamiento, conocidos como brokers. Otros servidores ejecutan Kafka Connect para importar y exportar continuamente datos como flujos de eventos, integrando Kafka con sistemas existentes, como bases de datos relacionales y otros clústeres de Kafka. Para permitir la implementación de casos de uso críticos, un clúster de Kafka es altamente escalable y tolerante a fallos; si alguno de sus servidores falla, los demás asumirán su carga de trabajo para garantizar operaciones continuas sin pérdida de datos.

Clientes: Los clientes permiten escribir aplicaciones distribuidas y microservicios que leen, escriben y procesan flujos de eventos en paralelo, a gran escala y de manera tolerante a fallos, incluso ante problemas de red o fallos en las máquinas.

Los **productores** son las aplicaciones cliente que publican (escriben) eventos en Kafka, mientras que los **consumidores** son aquellos que se suscriben (leen y procesan) estos eventos. En Kafka, los productores y los consumidores están totalmente desacoplados y son agnósticos entre sí, lo cual es un elemento clave en el diseño para lograr la alta escalabilidad por la que Kafka es conocido. Por ejemplo, los productores nunca tienen que esperar a los consumidores. Kafka ofrece varias garantías, como la capacidad de procesar eventos exactamente una vez.[1]

3.3 Captura de datos

Se implementa el entorno de Kafka utilizando Docker y Docker Compose, lo que nos permite configurar y gestionar de manera rápida y escalable los con-

tenedores.

Para la extracción de datos históricos, se optó por el método de procesamiento en lotes. Este enfoque resultó más conveniente, ya que los datos históricos suelen ser grandes y estáticos.
Se configura la consola y el diseño de Rich para una visualización mejorada. También se configuran el servidor Kafka y el productor de Kafka, que enviará mensajes a un tema (topic) específico. Los mensajes serán serializados en formato JSON.

```
        from kafka import KafkaProducer
                import json
                from rich.console import Console
                from rich.layout import Layout

                console = Console()
                layout = Layout()
                KAFKA_TOPIC = 'ibm_options'
                KAFKA_SERVER = 'localhost:9092'
                producer = KafkaProducer(
                bootstrap_servers=[KAFKA_SERVER],
                value_serializer=lambda x: json.dumps(x).encode('
                    utf-8')
                )
```

El consumer se configura para escuchar en el tema "ibm-options" en el servidor de Kafka especificado. A medida que recibe mensajes, los deserializa de JSON a un diccionario de Python.

```
        consumer = KafkaConsumer(
        KAFKA_TOPIC,
        bootstrap_servers=KAFKA_SERVER,
        value_deserializer=lambda x: json.loads(x.decode('utf-8')),
        auto_offset_reset='earliest',
        enable_auto_commit=True
        )
```

4 Almacenamiento de Datos

4.1 The Hadoop Distributed File Systemp (HDFS)

El sistema de archivos distribuidos de Hadoop (HDFS) es un sistema de archivos diseñado para gestionar grandes conjuntos de datos que pueden ejecutarse en hardware básico. HDFS es el sistema de almacenamiento de datos más popular para Hadoop y permite escalar un único clúster de Apache Hadoop a cientos e incluso miles de nodos. Dado que gestiona de forma eficiente grandes volúmenes de datos con un alto rendimiento, HDFS se puede utilizar como canalización de datos y es ideal para soportar análisis de datos complejos.[2]
En este trabajo, hay una función llamada save-data que se encarga de guardar

los mensajes recibidos del Kafka Consumer en un sistema de archivos distribuido HDFS o localmente. Si el cliente HDFS (hdfs-client) está configurado, intenta guardar el archivo en el directorio HDFS especificado (HDFS-DIR). Se utiliza with hdfs-client.write para escribir los datos en un archivo en HDFS. Si el archivo se guarda correctamente, se imprime un mensaje de éxito en la consola y se devuelve el tipo de almacenamiento (HDFS) junto con el nombre del archivo (ver Figura 1).

```
def save_data(data, timestamp):
        """Guarda los datos en HDFS o localmente"""
        filename = f"ibm_options_{timestamp}.json"

        if hdfs_client:
                try:
                        file_path = f"{HDFS_DIR}{filename}"
                        with hdfs_client.write(file_path, encoding=
                            'utf-8') as writer:
                                json.dump(data, writer)
                                console.print(f"[green]Archivo
                                    guardado en HDFS: {filename}[/
                                    green]")
                        return "HDFS", filename
                except Exception as e:
                        console.print(f"[red]Error al escribir en
                            HDFS: {str(e)}[/red]")

        local_path = os.path.join(LOCAL_DIR, filename)
        with open(local_path, 'w', encoding='utf-8') as f:
                json.dump(data, f)
        console.print(f"[yellow]Archivo guardado localmente: {
            filename}[/yellow]")
        return "Local", filename
```

5 Procesamiento de los Datos

Para acceder a los datos después de ser almacenados y realizar un procesamiento, se extraen utilizando InsecureClient(hdfs-url), que establece la conexión con el servidor HDFS. El análisis se visualizará apoyándonos en Streamlit, una biblioteca de Python. Después de extraer los archivos JSON del servidor, se unifican todos en un DataFrame de Dask, que admite la computación paralela. Cuando el conjunto de datos no "cabe en la memoria", Dask extiende el conjunto de datos para que "quepa en el disco". Dask nos permite escalar fácilmente a clústeres o reducir a una sola máquina en función del tamaño del conjunto de datos.

Para uno de los análisis, fue necesario estandarizar los datos y eliminar de la base de datos todas las filas con valores faltantes. Además, se convierten las columnas numéricas al formato adecuado.

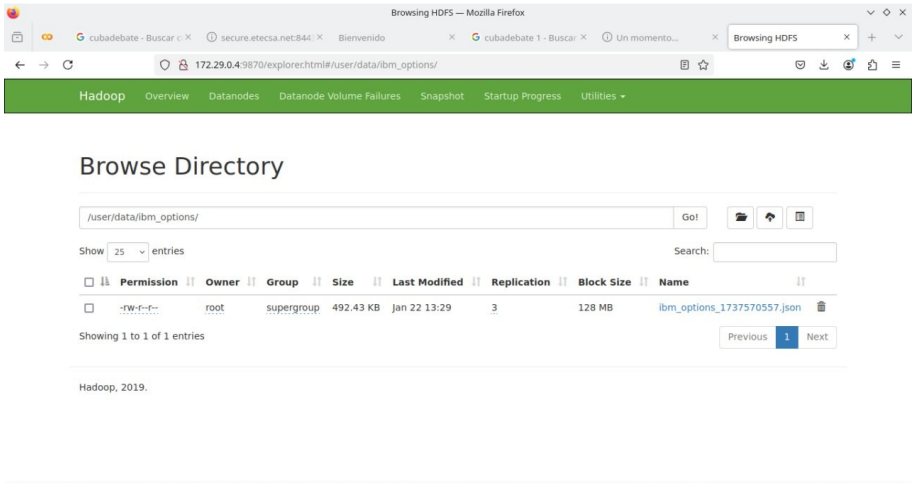

Figure 1: Esta imagen muestra toda la data almacenada en HDFS.

Figure 2: Esta imagen muestra en consola que la data se guardó correctamente.

6 Análisis de los Datos

En el análisis financiero de IBM, el usuario puede seleccionar el tipo de opción (*put* o *call*). Para ayudar a identificar tendencias y momentos de compra o venta, se puede utilizar un gráfico de líneas donde se selecciona la métrica a visualizar. Una vez seleccionada esta métrica, se muestra su distribución mediante un histograma, lo que puede revelar sesgos u outliers. Además, un gráfico de barras compara el volumen y el interés abierto (*open interest*) entre opciones *call* y *put*. El volumen indica la actividad y las expectativas del mercado, mientras que el interés abierto muestra la liquidez de las opciones. La aplicación incluye un modelo de Random Forest Regressor para predecir el precio de una opción (*last*) basado en factores clave (como el strike, la implied volatility, etc.). Para evaluar la efectividad del modelo, se utiliza un gráfico de dispersión que visualiza si el modelo se ajusta a los datos reales. A continuación, se presenta un gráfico de barras que muestra qué características tienen el mayor impacto en las predicciones del modelo, lo que permite determinar cuáles son los factores

clave.

7 Referencias

[1]https://kafka.apache.org/documentation/#gettingStarted
[2]https://www.ibm.com/es-es/topics/hdfs