Material del módulo de Spark Streaming

2022-01-24

El material del módulo de Spark Streaming está repartido en tres ficheros ZIP:

- `Spark-Streaming-I.zip`: material para la sesión 1 (*Streaming clásico*, i.e. *RDD Streaming*)
- `Spark-Streaming-II.zip`: material para la sesión 2 (*Structured Streaming*)
- `Spark-Streaming-III.zip`: material para la sesión 3 (*ML sobre Streaming*)

Guía de ficheros

Una vez descomprimido, los ZIPs producen los siguientes ficheros y directorios (suponiendo que se han descomprimido en el mismo sitio):

Carpeta	Fichero	Contenido
`DATA`	***Datos de prueba para streaming vía puerto TCP***	
	`PublishData`	Script Bash para publicar un fichero de texto en un puerto TCP (i.e. crea una fuente local de datos)
`01.base`	***Streaming clásico (RDD).*** *Los scripts Python de esta carpeta se conectan todos a un puerto TCP y leen la fuente de datos publicada por* `PublishData`	
	`streaming0_simple.oy`	Lee texto y lo escribe en pantalla
	`streaming1_stateless.py`	Lee texto y cuenta la frecuencia de palabras en cada lote
	`streaming2a_stateful.py`	Lee texto, cuenta frecuencia de palabras, conserva el estado de un lote al siguiente, y saca la lista de las 20 palabras más frecuentes
	`streaming2b_stateful.py`	Igual que el anterior, pero además incorpora un filtro de stopwords
	`streaming3_recovery.py`	Como el anterior, pero además persistir el estado en disco y lo recupera al arrancar
	`streaming4_exercise.py`	Ejercicio de Streaming clásico planteado
	`streaming5_window.py`	Lee texto y procesa los lotes que caen en una ventana de tiempo deslizante
	`spark-streaming-basic.ipynb`	Ejemplo básico de streaming sobre un notebook[1]

Carpeta	Fichero	Contenido
`02.structured`	***Ejemplos de Streaming Estructurado*** *Los ficheros de esta carpeta usan Spark Structured Streaming*	
	`ss_base1.py`	Fuente de datos local (puerto TCP), cuenta palabras y escribe a consola las 40 más frecuentes en cada lote.
	`ss_base2a.py`	Escribe las 40 más frecuentes, usando un *sink* de tipo `foreach` con una función
	`ss_base2b.py`	Graba a disco la lista de las 40 más frecuentes, usando un *sink* de tipo `foreach` con una clase
	`ss_base2c_ejercicio.py`	Ejercicio de Streaming Estructurado planteado
	`ss_base3.py`	Crea una ventana deslizante y cuenta palabras dentro de las ventanas
	`ss_base4.py`	Crea una ventana deslizante, cuenta palabras dentro de las ventanas, y usa un *sink* de tipo `foreachBatch`
`03.kafka`	***Spark Streaming con fuente de datos Kafka*** *Los ficheros de esta carpeta se conectan a una fuente de datos Kafka (por defecto* `cluster1bigata.ii.uam.es`*)*	
	`ss-kafka1.py`	Ejemplo de lectura de datos desde una fuente Kafka[2]
	`ss-kafka2.py`	Lectura de datos desde una fuente Kafka, descomposición del mensaje
	`ss-kafka3.py`	Lectura de datos desde una fuente Kafka y cálculo de agrupaciones, activando checkpointing para recuperación automática
	`ss-kafka.ipynb`	Ejemplo de notebook para leer datos en streaming de una fuente Kafka.
`04.ml`	***Machine Learning sobre Spark Streaming*** *Contiene dos ejercicios de Machine Learning sobre Streaming.* *Se conectan a una fuente de datos externa (por defecto* `cluster1bigata.ii.uam.es`*)*	
	`spark-submit-kafka`	Script de ayuda: permite lanzar un trabajo de streaming con fuente de datos Kafka[3]
	`streaming-kafka.py`	Script para capturar y presentar por consola un *topic* de Kafka
`04.ml/data`	*Datasets para los ejercicios de ML sobre Streaming (descargados previamente de la fuente)*	
`04.ml/e1`	*Ejercicio 1: aprendizaje no supervisado online para streaming*	
	`streaming-kmeans-base.py`	online k-means para datos meteorológicos, plantilla base para mostrar los datos

1 Exige que el kernel del Notebook haya arrancado con al menos dos hebras de ejecución (ver documentación)
2 Exige configurar Spark para que acceda a fuentes de datos Kafka con Spark Streaming RDDs (ver documentaciñon)
3 No será necesario si ya hemos configurado Spark en `spark-defaults.conf` para que incluya el paquete de Kafka en todos sus jobs.

Carpeta	Fichero	Contenido
	`streaming-kmeans-save.py`	Script de captura de datos y grabación a un fichero CSV
	`ml-kmeans.ipynb`	notebook de entrenamiento de k-means a partir de un dataset ya descargado
	`steps/ks*.py`	online k-means (7 scripts con implementación progresiva, paso a paso)
`04.ml/e2`	*Ejercicio 2: aprendizaje supervisado usando entrenamiento offline*	
	`streaming-class-base.py`	clasificador del idioma de tweets, plantilla base para mostrar datos
	`streaming-class-save.py`	Script de captura datos y grabación a un fichero CSV
	`ml-clas.ipynb`	notebook de entrenamiento offline para crear los modelos que luego se cargarán en la parte de streaming
	`steps/scb*.py`	clasificación de datos en streaming (5 scripts con implementación progresiva, paso a paso)
	`steps`	dos scripts con el proceso completo (uno carga el pipeline de ML, el otro los modelos individuales)

Observaciones:

- Para que los scripts que usan fuentes de datos Kafka funcionen es necesario que las clases de Kafka estén disponibles. Esto se puede hacer de varias formas (tal como se describe en la documentación):
  - cargar el paquete correspondiente en la ejecución (el script `spark-submit-kafka` automatiza este proceso)
  - o definirlo en la configuración,
- En algunos casos es posible que `spark-submit` falle si el nombre completo del fichero que se ejecuta (es decir, nombre de fichero + directorio completo) contiene espacios
- Si se le quieren pasar parámetros al programa que `spark-submit` ejecuta, hay que hacerlo después de un doble guión:

  ```
  spark-submit <fichero python> -- <parámetros para el fichero python>
  ```

  esto es así porque lo que viene entre el fichero Python y el doble guión se interpreta como parámetros para el intérprete Python que ejecuta el *driver* de Spark, no como parámetros para el script de Python que se ejecuta.
- Los scripts de Bash se ejecutan dentro de la máquina virtual como

  ```
  bash <script> <parámetros>
  ```

Ejercicios de Machine Learning

El material incluye dos ejercicios relacionados con Machine Learning en Spark Streaming:

1. uno con aprendizaje no supervisado y Streaming clásico (*Streaming k-means*), usando online learning + pre-training
2. otro con aprendizaje supervisado y Streaming estructurado (clasificador binario con *offline training*)

El primero lee los datos desde un socket (ya que a partir de Spark 3 no hay componente para leer datos de Kafka a un Dstream), el segundo utiliza como fuente de datos Kafka. Los ficheros de material para los dos ejemplos están en el subdirectorio `04.ml`

e1: Streaming k-means

El objetivo de este ejercicio es realizar un algoritmo de *clustering* en tiempo real, agrupando datos provenientes de una fuente meteorológica

- los datos contienen mediciones de sensores meteorológicos que proporcionan, entre otros, valores de temperatura, humedad, presión y viento para un conjunto de estaciones meteorológicas distribuidas por toda España
- la fuente de datos **`meteo`** disponible en el servidor de `cluster1bigdata.ii.uam.es` proporciona datos en tiempo real de esas estaciones
- se trata de agrupar las estaciones usando *online k-means*, de forma que
  - a cada estación entrante (via streaming) se le asigne un clustering
  - los clusters vayan evolucionando, adaptándose a los datos

El proceso es *online machine learning*, con un añadido posterior de modelo preentrenado. El flujo de funcionamiento sigue el orden de los ficheros en la tabla de ficheros mostrada anteriormente:

1. captura de datos de la fuente a ficheros
2. creación de modelo (*k-means offline*) con los datos capturados
3. aplicación de un modelo online, cuyo punto de arranque (centroides iniciales) es el calculado offline, con los datos de la fuente según van llegando

El tercer proceso, la creación y aplicación del modelo online, se ha descompuesto en 7 ficheros, que van aplicando poco a poco los pasos de proceso necesarios (cada fichero contiene el anterior más un paso adicional).

e2: clasificador binario

- La fuente de datos **`tweet`** disponible en el servidor Kafka de `cluster1bigdata.ii.uam.es` proporciona una muestra de tweets en dos idiomas: español e inglés
- Nota: el idioma que figura en el mensaje es el declarado en la cuenta del usuario productor de cada tweet. Es posible que un tweet concreto esté redactado en un idioma distinto de la cuenta; en ese caso ese dato estaría incorrectamente etiquetado

El proceso es *machine learning* con *offline training*. El flujo de funcionamiento sigue el orden de los ficheros en la tabla de ficheros mostrada anteriormente:

1. captura de datos de la fuente a ficheros
2. creación de modelo (clasificador binario de texto) con los datos capturados, y grabación a disco del pipeline que contiene el modelo
3. carga del modelo y aplicación a los datos de la fuente según van llegando

El tercer proceso, la creación y aplicación del modelo, se ha descompuesto en 5 ficheros, que van aplicando poco a poco los pasos de proceso necesarios (cada fichero contiene el anterior más un paso adicional).