Artículo
Alberto Fuentes · 15 mayo, 2020 Lectura de 12 min

Un contenedor para probar Machine Learning ML con IRIS, Spark y Zeppelin

¡Muy buenas! Repasemos hoy un genial artículo de David E. Nelson sobre cómo montar un entorno de pruebas utilizando contenedores Docker para experimentar con IRIS, Spark y Zeppelin.

Gracias a la creciente disponibilidad de contenedores y el cada vez más útil Docker para Windows / MacOs, tengo mi propia selección de entornos preconfigurados para aprendizaje automático y *data science*. Por ejemplo, Jupyter Docker Stacks y Zeppelin en Docker Hub. Gracias también a la edición IRIS Community en un contenedor, tenemos un práctico acceso a una plataforma de datos que admite tanto el aprendizaje automático y análisis junto con otras muchas funciones. Al usar contenedores, no necesito preocuparme por actualizaciones automáticas que puedan arruinar mi área de pruebas. Si mi oficina se inundara y el portátil quedara inservible, podría recrear fácilmente el área de pruebas con un único archivo de texto, que por supuesto subí a un sistema de control de versiones ;-)

A continuación compartiré el archivo de Docker Compose que usé para crear un área de pruebas basada en un contenedor para Machine Learning. El área de pruebas comprende dos contenedores: uno con un entorno Zeppelin y Spark, el otro con la plataforma de datos InterSystems IRIS community. Ambos usan imágenes disponibles en Docker Hub. Luego mostraré cómo configurar el InterSystems Spark Connector para conectarlos entre sí. Para terminar, cargaré algunos datos en InterSystems IRIS y usaré Spark para hacer exploración y visualización de datos y algún aprendizaje automático muy básico. Por supuesto, mi ejemplo apenas tocará la superficie de las capacidades tanto de Spark como de InterSystems IRIS. Sin embargo, espero que el artículo sea útil para que otros comiencen a realizar tareas más complejas y útiles.

Archivo Docker Compose

- Dos contenedores: uno contiene InterSystems IRIS Community Edition y el otro contiene tanto el entorno de bloc de notas Zeppelin y Apache Spark. Ambos contenedores se basan en imágenes tomadas de la tienda de Docker.
- Una conexión de red entre ambos contenedores. Con esta técnica, podemos usar los nombres de los contenedores como nombres de host al configurar la comunicación entre los contenedores.
- Directorios locales montados en cada contenedor. Podemos usar estos directorios para hacer que los archivos jar queden disponibles para el entorno Spark y para que algunos archivos de datos queden disponibles para el entorno IRIS.
- Un volumen con nombre para la funcionalidad duradera %SYS que requiere InterSystems IRIS. InterSystems IRIS requiere volúmenes con nombres cuando se ejecuta dentro de contenedores en Docker para Windows. Por más información sobre este tema, consulte abajo los enlaces a otros artículos de la comunidad.
- Mapear algunos puertos de red dentro de los contenedores a puertos disponibles fuera de los contenedores para brindar un fácil acceso.

```
version: '3.2'

services:
  # contenedor con InterSystems IRIS
  iris:
    # imagen de iris community edition a extraer de la tienda Docker.
    image: store/intersystems/iris:2019.1.0.510.0-community
    container_name: iris-community
```

```
    ports:
    # 51773 es el puerto predeterminado del superserver
    - "51773:51773"
    # 52773 es el puerto predeterminado del webserver/portal de gestión
    - "52773:52773"

    volumes:
    # Crea un volumen con nombre durable_data que guardará los datos duraderos %SYS
    - durable:/durable
    # Mapea un directorio /local para permitir el fácil traspaso de archivos y script
s de prueba
    - ./local/samples:/samples/local

    environment:
    # Configurar la variable ISC_DATA_DIRECTORY al volumen durable_data que definimos
 antes para usar %SYS duradero
    - ISC_DATA_DIRECTORY=/durable/irissys

    # Agrega el contenedor IRIS a la red definida a continuación.
    networks:
      - mynet

  # contenedor con Zeppelin y Spark
  zeppelin:
    # bloc de notas zeppelin con imagen de spark para extraer desde docker store.
    image: apache/zeppelin:0.8.1

    container_name: spark-zeppelin

    # Puertos para acceder al entorno Zeppelin
    ports:
    # Puerto para bloc de notas Zeppelin
    - "8080:8080"
    # Puerto para página de trabajos Spark
    - "4040:4040"

    # Mapea directorios /local para guardar blocs de notas y acceder a archivos jar.
    volumes:
    - ./local/notebooks:/zeppelin/notebook
    - ./local/jars:/home/zeppelin/jars

    # Agrega el contenedor Spark y Zeppelin a la red definida a continuación.
    networks:
    - mynet


   #Declara el volumen con nombre para el %SYS duradero de IRIS
volumes:
  durable:

  # Define una conexión de red entre ambos contenedores.
networks:
  mynet:
    ipam:
      config:
        - subnet: 172.179.0.0/16
```

Puesta en funcionamiento

Coloca el archivo compose en un directorio. El nombre del directorio se convierte en el nombre del proyecto Docker. Deberá crear subdirectorios que coincidan con los mencionados en el archivo docker-compose.yml. Por lo tanto, la estructura de directorio se ve así:

```
|_local
    |_ jars
    |_notebooks
    |_samples
|_ docker-compose.yml
```

Para iniciar los contenedores, ejecuta esta comando dentro del directorio del proyecto:

```
$ docker-compose up –d
```

El *flag*-d inicia los contenedores en modo *detached* (estarán ejecutando en *background*).
Puedes usar el comando logs de docker para los registros de log:

```
$ docker logs iris-community
```

Para inspeccionar el estado de los contenedores, podemos ejecutar lo siguiente:

```
$ docker container ls
```

Accederemos al Portal de gestión de IRIS con la URL: http://localhost:52773/csp/sys/UtilHome.csp
La primera vez que iniciemos sesión en IRIS, entraremos con superUser / SYS. A continuación se nos pedirá modificar la contraseña por defecto.

Al bloc de notas Zeppelin accedemos a través de la URL: http://localhost:8080

Copiar algunos archivos jar

Para usar el InterSystems Spark Connector, el entorno Spark debe acceder a dos archivos jar:
* intersystems-jdbc-3.0.0.jar
* intersystems-spark-1.0.0.jar

Estos archivos jar están dentro de IRIS en el contenedor iris-community. Debemos copiarlos al directorio localmente mapeado, para que el contenedor spark-zeppelin pueda acceder a ellos.

Para esto, podemos usar el comando cp de Docker para copiar todos los archivos desde dentro del contenedor iris-community hacia uno de los directorios locales visibles para el contenedor spark-zeppelin:

```
$ docker cp iris-community:/usr/irissys/dev/java/lib/JDK18 local/jars
```

Agregar datos

Sin datos no hay Machine Learning. Podemos usar los directorios locales montados por el contenedor iris-

community para agregar datos a IRIS. Utilizaremos el clásico conjunto de datos sobre la flor Iris. Desde hace tiempo este conjunto de datos cumple la función de ejemplo "Hola Mundo" para Machine Learning.

Puedes descargar o extraer una definición de clase de InterSystems para generar los datos, junto con código para varios ejemplos relacionados, de GitHub (Samples-Data-Mining). Nos interesa únicamente un archivo de este conjunto: DataMining.IrisDataset.cls.

Copia DataMining.IrisDataset.cls al directorio <project-directory>/local/samples. A continuación, para abrir un intérprete bash dentro del contenedor iris-community, ejecuta:

```
$ docker exec –it iris-community bash
```

Desde el intérprete bash, inicia una sesión de terminal IRIS con el usuario *superuser* y la contraseña que cambiaste anteriormente en el portal de gestión.

```
/# iris session iris
```

Cargamos la clase para generar los datos en IRIS:

```
USER>Do $System.OBJ.Load(“/samples/local/IrisDataset.cls”,”ck”)
```

A continuación, generamos los datos:

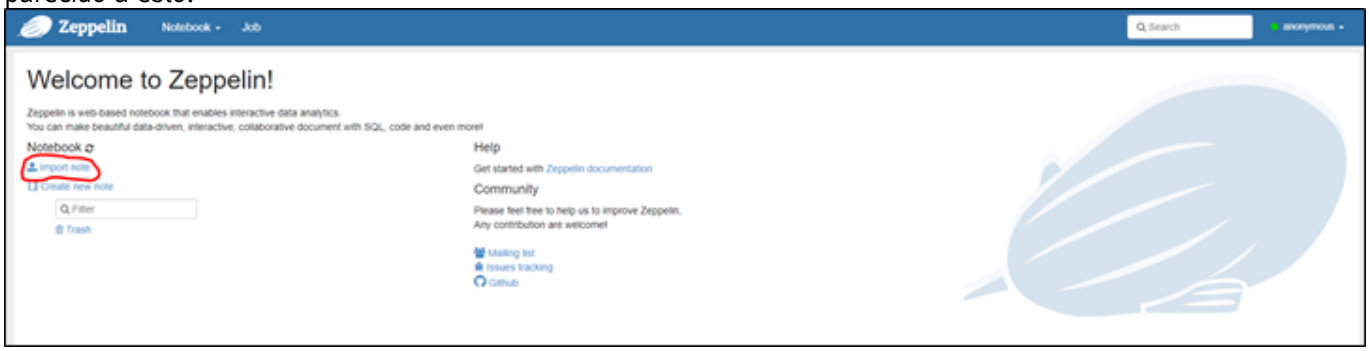
```
USER>Set status = ##class(DataMining.IrisDataset).load()
USER>Write status
```

La base de datos ahora debería contener ahora datos de 150 ejemplos de flores Iris.

Lanzar Zeppelin y Configurar nuestro bloc de notas

En primer lugar, descarga la nota "Machine Learning Hello World" de Zeppelin disponibles https://github.com/denelson/DevCommunity.

Podrás abrir el bloc de notas Zeppelin en su navegador web en la URL http://localhost:8080. Debería verse algo parecido a esto:



Hacemos click en *Import note* e importamos la nota “ Machine Learning Hello World.json ” que habíamos descargado antes.

En el primer párrafo de código cargaremos el controlador de InterSystems JDBC y Spark Connector. Por defecto, los blocs de notas Zeppelin ofrecen la variable z para acceder al contexto Zeppelin. Tienes más información disponible sobre contexto Zeppelin en la documentación.

```
%spark.dep
//z supplies Zeppelin context
z.reset()
z.load("/home/zeppelin/jars/JDK18/intersystems-jdbc-3.0.0.jar")
z.load("/home/zeppelin/jars/JDK18/intersystems-spark-1.0.0.jar")
```

Antes de ejecutar el párrafo de código, hacemos click en la flecha abajo que se encuentra al lado de la palabra *anonymous* y escogemos *Interpreter*.

En la página *Intepreters*, buscamos *spark*, hacemos click en el botón *restart* (reiniciar) del lado derecho y luego en OK en el cuadro emergente que aparecerá.

Regresamos al bloc de notas "Machine Learning Hello World" y hacemos clic en la pequeña flecha de la derecha para ejecutar el párrafo. Debería resultar algo parecido a esto:

Conectar con IRIS y explorar los datos

Ya está todo configurado. Ahora podemos conectar el código que se ejecuta en el contenedor *spark-zeppelin* con InterSystems IRIS, que se ejecuta en nuestro contenedor iris-community, y comenzar a explorar los datos que agregamos antes. El siguiente código Python se conecta con InterSystems IRIS y lee la tabla de datos que cargamos en un paso anterior (DataMining.IrisDataset), para luego mostrar las primeras diez filas.

A continuación dejo algunas notas sobre el siguiente código:

- Necesitamos pasar el usuario y la contraseña en la conexión a IRIS. Utiliza la contraseña que hayas establecido en pasos anteriores. En el ejemplo usamos SuperUser/SYS1.
- El iris dentro del fragmento de código spark.read.format(“ iris ”) es un alias para la clase com.intersystems.spark, el conector *spark*.
- La URL de conexión, incluyendo "IRIS" al inicio, especifica la ubicación del servidor maestro Spark predeterminado de InterSystems IRIS.
- La variable spark apunta a la sesión Spark brindada por el intérprete Spark de Zeppelin.

```
%pyspark
uname = "SuperUser"
pwd = "SYS1"
# sesión spark disponible de forma predeterminada a través de la variable spark.
# La URL usa el nombre del contenedor, iris-community, como nombre de host.
iris = spark.read.format("iris").option("url","IRIS://iris-community:51773/USER").opt
ion("dbtable","DataMining.IrisDataset").option("user",uname).option("password",pwd).l
oad()
iris.show(10)
```

Nota: tienes más información dispone en Uso de InterSystems Spark Connector y SparkContext, SQLContext, SparkSession, ZeppelinContext.

Al ejecutar el párrafo anterior se obtiene la siguiente salida:

Cada fila representa una flor individual y registra el largo y ancho de sus pétalos, su largo y ancho de sépalo, y la especie de Iris a la que pertenece.

```
%pyspark
iris.groupBy("Species").count().show()
```

Así, hay tres especies de Iris distintas representadas en los datos. Los datos representan a cada especie por igual. Incluso podemos usar la biblioteca *matplotlib* de Python para dibujar algunas gráficas. Este es el código para representar Largo de pétalo (*Petal length*) frente a Ancho de pétalo (*Petal width*):

```
%pyspark
%matplotlib inline
import matplotlib.pyplot as plt
#Recuperar una matriz de objetos de fila desde el DataFrame
items = iris.collect()
petal_length = []
petal_width = []
for item in items:
    petal_length.append(item['PetalLength'])
    petal_width.append(item['PetalWidth'])

plt.scatter(petal_width,petal_length)
plt.xlabel("Petal Width")
plt.ylabel("Petal Length")
plt.show()
```

Incluso para el ojo no entrenado, parece haber una fuerte correlación entre largo y ancho de pétalo. Deberíamos poder predecir de forma confiable el largo de pétalo en base al ancho de pétalo.

Un poco de aprendizaje automático

Para predecir el largo de pétalo en base al ancho de pétalo, necesitamos un modelo de la relación entre ambos. Podemos usar Spark para crear dicho modelo muy fácilmente. A continuación utilizamos un código que usa la API de regresión lineal de Spark para entrenar un modelo de regresión.

El código hace lo siguiente:
1. Crea un nuevo *DataFrame* de Spark que contiene las columnas de largo y ancho de pétalo. La columna de ancho de pétalo representa la "característica" y la columna de largo de pétalo representa las "etiquetas". Usamos las características para predecir las etiquetas.
2. Divide los datos aleatoriamente en un conjunto de entrenamiento (70%) y uno de pruebas (30%).
3. Usa los datos de entrenamiento para ajustar el modelo de regresión lineal.
4. Aplica el modelo a los datos de prueba y luego muestra el largo de pétalo, ancho de pétalo, características y predicciones.

```
%pyspark
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

# Transformar la(s) columna(s) de "Características" (Features) al formato vectorial c
orrecto
df = iris.select('PetalLength','PetalWidth')
vectorAssembler = VectorAssembler(inputCols=["PetalWidth"],
```

```
                                    outputCol="features")
data=vectorAssembler.transform(df)

# Dividir los datos entre conjuntos de entrenamiento y prueba.
trainingData,testData = data.randomSplit([0.7, 0.3], 0.0)

# Configurar el modelo.
lr = LinearRegression().setFeaturesCol("features").setLabelCol("PetalLength").setMaxI
ter(10)

# Entrenar el modelo con los datos de entrenamiento.
lrm = lr.fit(trainingData)

# Aplicar el modelo a los datos de prueba y mostrar sus predicciones de largo de péta
lo (PetalLength).
predictions = lrm.transform(testData)
predictions.show(10)
```

La línea de regresión

El "modelo" no es más que una línea de regresión a través de los datos. Sería interesante conocer la pendiente y punto de intersección con el eje vertical de esa línea. También sería interesante poder visualizar la línea superpuesta a nuestro diagrama de dispersión. El siguiente código recupera la pendiente e intersección con el eje vertical del modelo entrenado y luego las usa para agregar una línea de regresión al diagrama de dispersión de los datos de largo y ancho de pétalo.

```
%pyspark
%matplotlib inline
import matplotlib.pyplot as plt

# recuperar la pendiente e intersección con el eje vertical de la línea de regresión
del modelo.
slope = lrm.coefficients[0]
intercept = lrm.intercept

print("slope of regression line: %s" % str(slope))
print("y-intercept of regression line: %s" % str(intercept))

items = iris.collect()
petal_length = []
petal_width = []
petal_features = []
for item in items:
    petal_length.append(item['PetalLength'])
    petal_width.append(item['PetalWidth'])
fig, ax = plt.subplots()
ax.scatter(petal_width,petal_length)
plt.xlabel("Petal Width")
plt.ylabel("Petal Length")
y = [slope*x+intercept for x in petal_width]
ax.plot(petal_width, y, color='red')

plt.show()
```

¿Qué más podemos hacer?

Hay mucho más que podemos hacer. Obviamente, podemos cargar conjuntos de datos mucho más grandes e interesantes en IRIS. Vea, por ejemplo, los conjuntos de datos Kaggle (https://www.kaggle.com/datasets). Con una licencia completa de IRIS podríamos configurar la fragmentación y ver cómo Spark aprovecha el paralelismo que permite la fragmentación al ejecutarse a través del InterSystems Spark Connector. Spark, por supuesto, brinda muchos otros algoritmos de aprendizaje automático y análisis de datos. Admite varios otros lenguajes, incluyendo Scala y R.

#Contenedorización #Machine Learning (ML) #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/un-contenedor-para-probar-machine-learning-ml-con-iris-spark-y-zeppelin