Lucidworks

LAB: Spark and Data Prep

Prerequisites

- Register for Kaggle
  - Instructions here: Registering for Kaggle

Machine Setup

Lucidworks

This lab assumes that you are using an AWS virtual machine provided by Lucidworks Training. If this is not the case, your filepaths and IP addresses will vary significantly from those shown.

- If you have not already done so, install Fusion
  `unzip fusion-4.0.1.zip`

- Using **vi**, open the **fusion.properties** file
  `vi ./fusion/4.0.1/conf/fusion.properties`

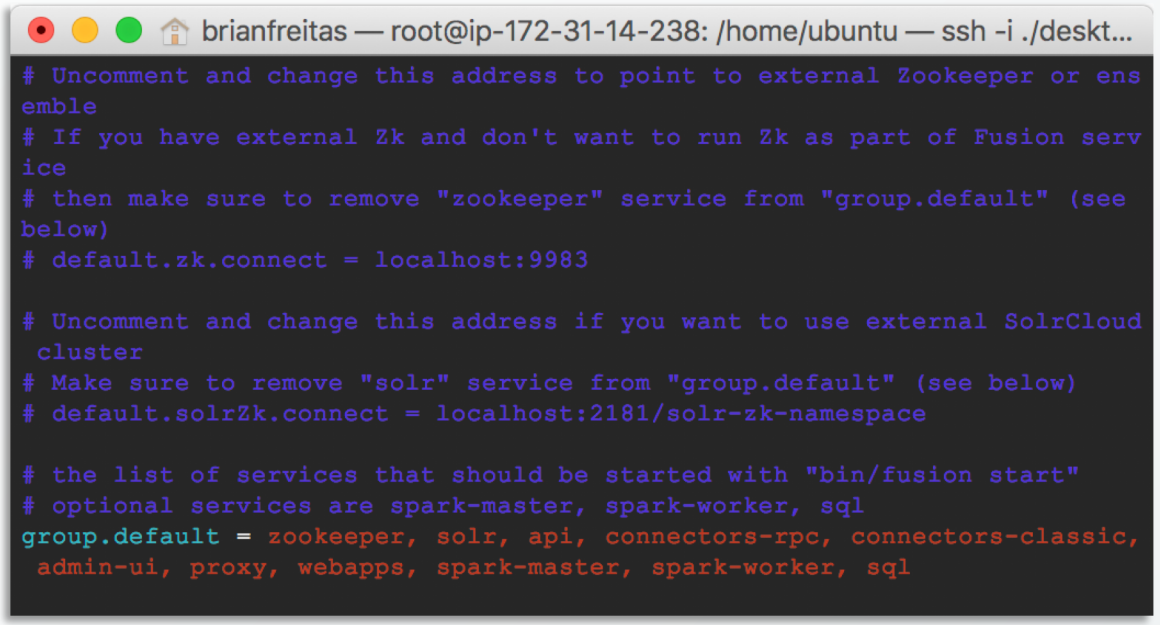
Machine Setup—Editing fusion.properties

Lucidworks

Among other things, this file controls which services will run on the local machine and how much memory they should be allocated.

We will add Spark and SQL to these services.

- Add the following services to **group.default**:
  `spark-master, spark-worker, sql`



Machine Setup—Editing fusion.properties

Lucidworks

- Locate the parameters listed below, and edit the memory values as shown

Parameter	Value
connectors-rpc.jvmOptions	-Xmx4g –Xms4g
connectors-classic.jvmOptions	-Xmx4g –Xms4g
solr.jvmOptions	-Xmx8g –Xms8g
spark-worker.jvmOptions	-Xmx8g -Xms8g

The default memory usage parameters are kept deliberately small, so that a Fusion demo will run on a standard laptop with no configuration changes. This virtual machine is far more powerful and the tasks we will do far more complex. Hence the increases.

```
# API service
api.port = 8765
api.stopPort = 7765
api.jvmOptions = -Xmx1g -Xss256k -Dhttp.maxConnections=1000

# Connectors RPC service
connectors-rpc.port = 8771
connectors-rpc.pluginPortRangeStart = 8871
connectors-rpc.pluginPortRangeEnd = 8971
connectors-rpc.jvmOptions = -Xmx4g -Xms4g

# Connectors service
connectors-classic.port = 8984
connectors-classic.stopPort = 7984
connectors-classic.jvmOptions = -Xmx4g -Xms4g -Dcom.lucidworks.connectors.pip
elines.embedded=false

# Zookeeper
zookeeper.port = 9983
zookeeper.jvmOptions = -Xmx256m

# Solr
solr.port = 8983
solr.stopPort = 7983
solr.jvmOptions = -Xmx8g -Xms8g

# Spark master
spark-master.port = 8766
spark-master.uiPort = 8767
spark-master.jvmOptions = -Xmx512m
spark-master.envVars=SPARK_SCALA_VERSION=2.11,SPARK_PUBLIC_DNS=${default.addr
ess},SPARK_LOCAL_IP=${default.address}

# Spark worker
spark-worker.port = 8769
spark-worker.uiPort = 8770
spark-worker.jvmOptions = -Xmx8g -Xms8g
spark-worker.envVars=SPARK_SCALA_VERSION=2.11,SPARK_PUBLIC_DNS=${default.addr
ess},SPARK_LOCAL_IP=${default.address}
```

Machine Setup—Editing fusion.properties

Lucidworks

- Exit INSERT mode
  `<escape>`

- Save changes and quit **vi**
  `:wq`

Machine Setup—Editing config.properties

Lucidworks

- Using vi, create a new file **config.properties**

```
vi ./fusion/4.0.1/conf/config.properties
```

This file will govern Spark behavior

- Paste the following parameters into **config.properties**:

```
spark.executor.memory = 8g
spark.worker.memory = 8g
spark.default.parallelism = 36
```

- Exit INSERT mode, save changes, and quit **vi**

```
<escape>  :wq
```

Machine Setup—Preparing a Fusion App

Lucidworks

- In your bash/shell terminal, start Fusion
  `./fusion/4.0.1/bin/fusion start`

Initial startup may require ~3 minutes

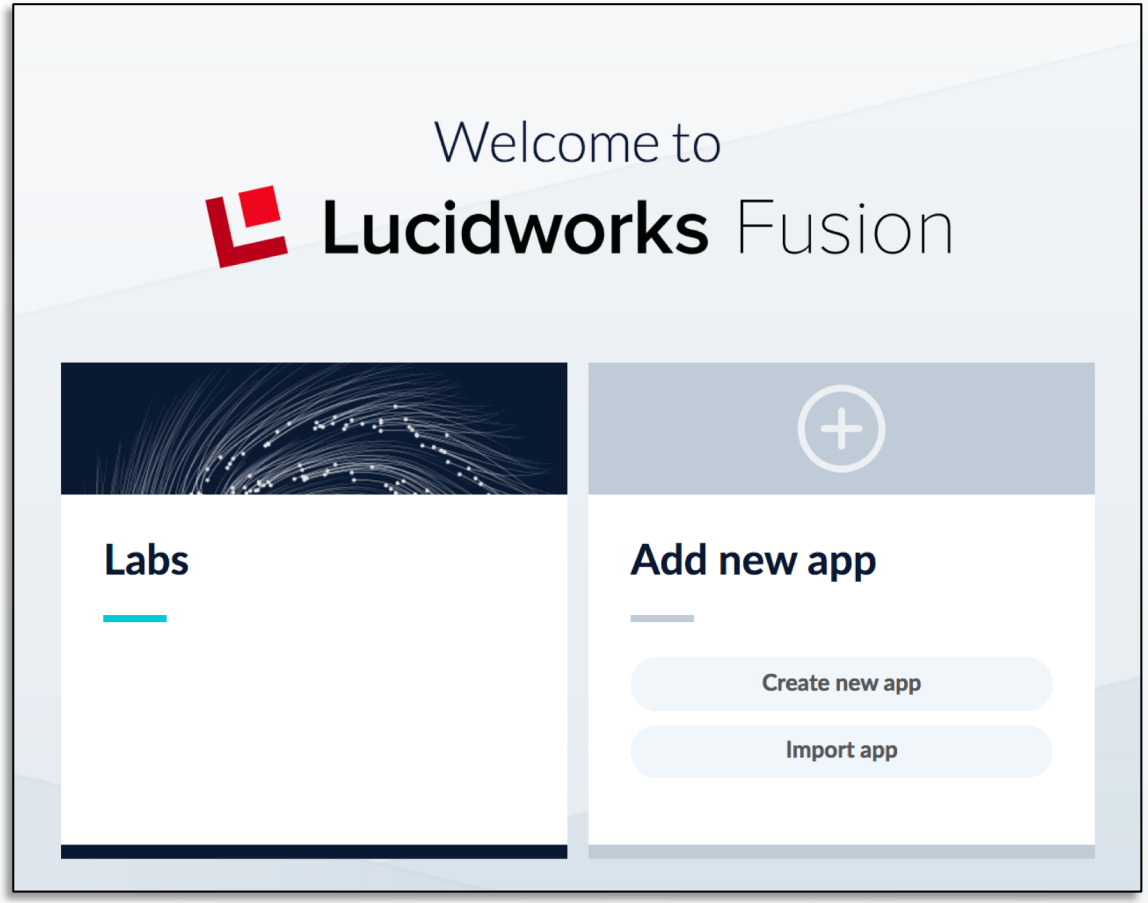
- In a web browser, open Fusion Admin
  `<your-vm-ip>:8764`

- If prompted, use credential **admin** and password **Lucidworks1**

Machine Setup—Preparing a Fusion App

Lucidworks

- Create a new Fusion App named **Labs**



Preparing Sample Data with Spark Shell

Starting Spark Shell

Lucidworks

- In your bash/shell terminal, open a new tab

*If using an AWS instance, be sure to connect to it via **ssh** or **PuTTY***

- Start up **Spark Shell**

```
./fusion/4.0.1/bin/spark-shell -m 6g -c 6
```

Importing Data into Spark

Lucidworks

Spark is a distributed-computing framework packaged as part of Fusion. It runs on Scala.
https://en.wikipedia.org/wiki/Scala_(programming_language)

- Import the BestBuy catalog into a DataFrame named **ecommerceFullCatalog**

```
val ecommerceFullCatalog =
spark.read.parquet("/opt/fusion/training/ecommerce/")
```

```
[scala> val ecommerceFullCatalog = spark.read.parquet("/opt/fusion/training/ecommerce/")
18/04/30 19:48:55 ERROR : Failed to resolve default logging config file: config/java.util.logging.
properties
Console logging handler is not configured.
18/04/30 19:48:58 WARN ObjectStore: Version information not found in metastore. hive.metastore.sch
ema.verification is not enabled so recording the schema version 1.2.0
18/04/30 19:48:58 WARN ObjectStore: Failed to get database default, returning NoSuchObjectExceptio
n
18/04/30 19:48:58 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectExce
ption
ecommerceFullCatalog: org.apache.spark.sql.DataFrame = [accessories: string, albumLabel: string ..
. 45 more fields]

scala>
```

Importing Data into Spark

Lucidworks

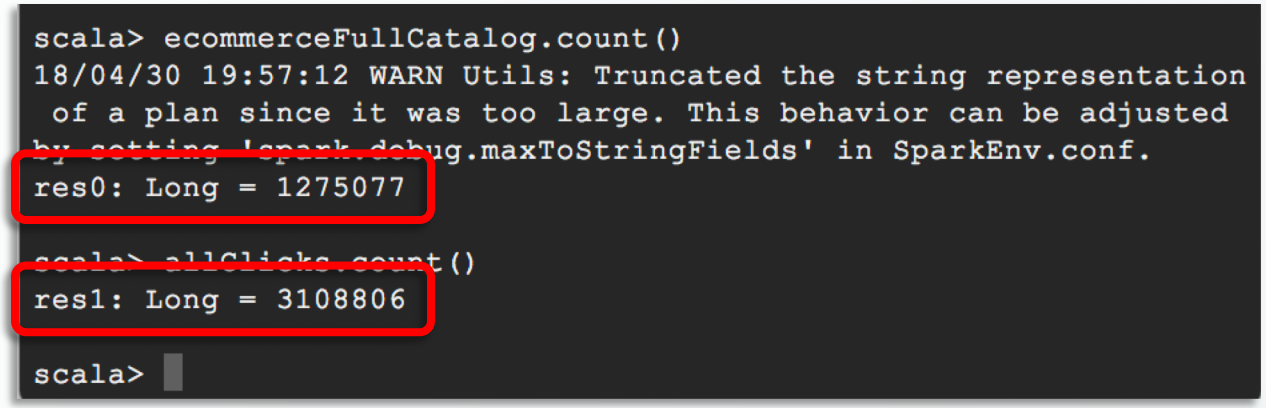
- Import the BestBuy click signals into a DataFrame named **allClicks**

```
val allClicks = spark.read.parquet("/opt/fusion/training/bestbuy-signals-parquet")
```

```
[scala> val ecommerceFullCatalog = spark.read.parquet("/opt/fusion/training/ecommerce/")
18/04/30 19:48:55 ERROR : Failed to resolve default logging config file: config/java.util.logging.properties
Console logging handler is not configured.
18/04/30 19:48:58 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
18/04/30 19:48:58 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
18/04/30 19:48:58 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
ecommerceFullCatalog: org.apache.spark.sql.DataFrame = [accessories: string, albumLabel: string ... 45 more fields]

[scala> val allClicks = spark.read.parquet("/opt/fusion/training/bestbuy-signals-parquet")
allClicks: org.apache.spark.sql.DataFrame = [_lw_parser_absolute_resource_name: array<string>, _lw_parser_content_type: array<string> ... 56 more fields]

scala>
```

- Verify that the imports were successful by checking the number of objects in each DataFrame:

```
ecommerceFullCatalog.count

allClicks.count
```



~1.2M products and ~3.1M clicks. This is a very large number of documents! We will not need all of these to build a useful dataset, and the massive volume would slow down processing times. We will trim the DataFrame to only use a subset of the total data

Downsampling the Catalog Data

Lucidworks

```
scala> ecommerceFullCatalog.printSchema
root
 |-- accessories: string (nullable = true)
 |-- albumLabel: string (nullable = true)
 |-- albumTitle: string (nullable = true)
 |-- artistName: string (nullable = true)
 |-- bundledIn: string (nullable = true)
 |-- cast: string (nullable = true)
 |-- categoryIds: string (nullable = true)
 |-- categoryNames: string (nullable = true)
 |-- class: string (nullable = true)
 |-- color: string (nullable = true)
 |-- condition: string (nullable = true)
 |-- crew: string (nullable = true)
 |-- customerReviewAverage: double (nullable = true)
 |-- customerReviewCount: long (nullable = true)
 |-- department: string (nullable = true)
 |-- depthCategoryIds: string (nullable = true)
 |-- depthCategoryNames: string (nullable = true)
 |-- description: string (nullable = true)
 |-- details: string (nullable = true)
 |-- features: string (nullable = true)
 |-- format: string (nullable = true)
 |-- frequentlyPurchasedWith: string (nullable = true)
 |-- genre: string (nullable = true)
 |-- hardGoodType: string (nullable = true)
 |-- id: string (nullable = true)
 |-- image: string (nullable = true)
 |-- lengthInMinutes: long (nullable = true)
 |-- longDescription: string (nullable = true)
 |-- manufacturer: string (nullable = true)
 |-- mpaaRating: string (nullable = true)
 |-- name: string (nullable = true)
 |-- plot: string (nullable = true)
 |-- regularPrice: double (nullable = true)
 |-- relatedProducts: string (nullable = true)
 |-- releaseDate: timestamp (nullable = true)
```

In order to intelligently trim the dataset, we first must learn more about the data structure.

- Request a list of all fields used by the **ecommerceFullCatalog** DataFrame: **`ecommerceFullCatalog.printSchema`**

*The catalog data is quite rich, giving us many possible ways to limit our sample. In this case, we will filter by **department**.*

```
 |-- salePrice: double (nullable = true)
 |-- salesRankLongTerm: long (nullable = true)
 |-- salesRankMediumTerm: long (nullable = true)
 |-- salesRankShortTerm: long (nullable = true)
 |-- shortDescription: string (nullable = true)
 |-- softwareGrade: string (nullable = true)
 |-- startDate: timestamp (nullable = true)
 |-- studio: string (nullable = true)
 |-- subclass: string (nullable = true)
 |-- suggest: string (nullable = true)
 |-- type: string (nullable = true)
 |-- weight: long (nullable = true)
```

Downsampling the Catalog Data

Lucidworks

- Create a new DataFrame called **someHardGoods**, containing only products from the **ACCESSORIES**, **APPLIANCES**, and **COMPUTERS** departments:

```
val someHardGoods = ecommerceFullCatalog.filter("department IN
('ACCESSORIES', 'APPLIANCE', 'COMPUTERS')")
```

- Verify success by checking the number of items in the new DataFrame:

```
someHardGoods.count
```

```
scala> val someHardGoods = ecommerceFullCatalog.filter("department IN ('ACCESSOR
IES', 'APPLIANCE', 'COMPUTERS')")
someHardGoods: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [accesso
ries: string, albumLabel: string ... 45 more fields]

scala> someHardGoods.count()
res3: Long = 42012
```

Downsampling the Clickstream Data

Lucidworks

As with the catalog documents, in order to intelligently work with this data, we first must learn more about the data structure.

- Request a list of all fields used by the **allClicks** DataFrame:
  `allClicks.printSchema`

As you can see, there is a lot of extra stuff here…

```
|-- _lw_parser_absolute_resource_name: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- _lw_parser_content_type: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- _lw_parser_diagnostic_id: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- _lw_parser_id: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- _lw_parser_max_parsing_depth: array (nullable = true)
|    |-- element: long (containsNull = true)
|-- _lw_parser_parsing_depth: array (nullable = true)
|    |-- element: long (containsNull = true)
|-- _lw_parser_record_number: array (nullable = true)
|    |-- element: long (containsNull = true)
|-- _lw_parser_resource_name: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- _lw_parser_type: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- attr_params.boost_s_: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- collection: string (nullable = true)
```

```
|-- count_i: long (nullable = true)
|-- date: timestamp (nullable = true)
|-- date_day: string (nullable = true)
|-- date_month: string (nullable = true)
|-- date_year: string (nullable = true)
|-- day_of_week: string (nullable = true)
|-- doc_id: string (nullable = true)
|-- doc_id_s: string (nullable = true)
|-- doc_ids_s: string (nullable = true)
|-- fields: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- filter: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- filters_orig_ss: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- filters_s: string (nullable = true)
|-- flag_s: string (nullable = true)
|-- fusion_query_id: string (nullable = true)
|-- hits: long (nullable = true)
|-- hour_of_day: long (nullable = true)
|-- http_method: string (nullable = true)
```

```
|-- id: string (nullable = true)
|-- is_fusion_query: boolean (nullable = true)
|-- params.boost_s: string (nullable = true)
|-- params.debug_s: string (nullable = true)
|-- params.defType_s: string (nullable = true)
|-- params.echoParams_s: string (nullable = true)
|-- params.facet_s: string (nullable = true)
|-- params.indicator_s: string (nullable = true)
|-- params.json.nl_s: string (nullable = true)
|-- params.query_time_dt: timestamp (nullable = true)
|-- params.wt_s: string (nullable = true)
|-- pipeline_id: string (nullable = true)
|-- platform: string (nullable = true)
|-- qtime: long (nullable = true)
|-- query: string (nullable = true)
|-- query_orig_s: string (nullable = true)
|-- query_s: string (nullable = true)
|-- query_t: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- res_offset: long (nullable = true)
|-- response_type: string (nullable = true)
|-- rows: long (nullable = true)
|-- time: long (nullable = true)
|-- timestamp_tdt: timestamp (nullable = true)
|-- totaltime: long (nullable = true)
|-- type: string (nullable = true)
|-- type_s: string (nullable = true)
|-- tz_timestamp_txt: array (nullable = true)
|    |-- element: string (containsNull = true)
|-- user_id: string (nullable = true)
|-- user_id_s: string (nullable = true)
```

Downsampling the Clickstream Data

Lucidworks

*Recall that the minimal information required in a signal is the **user**, the **query**, the **action taken**, and the **timestamp**. We will trim the clicks down to this, in order to make them easier to work with.*

- Create a new DataFrame called **trimmedClicks**, containing only **query**, **doc_id**, **fusion_query_id**, **filters_s**, **type**, **count_i**, **timestamp_tdt**, **user_id**, and **id**

```
val trimmedClicks =
allClicks.select("query","doc_id","fusion_query_id","filters_s","type",
"count_i","timestamp_tdt","user_id","id")
```

- Verify success by checking the number of items in the new DataFrame:

```
trimmedClicks.count
```

```
scala> val trimmedClicks = allClicks.select("query","doc_id","fusion_query_id
","filters_s","type","count_i","timestamp_tdt","user_id","id")
trimmedClicks: org.apache.spark.sql.DataFrame = [query: string, doc_id: strin
g ... 7 more fields]

[scala> trimmedClicks.count                                                   ]
res1: Long = 3108806
```

Downsampling the Clickstream Data

Lucidworks

Since we are only keeping three of the departments from the catalog, we only need clicks for items from those departments. We can do this by using a JOIN statement on someHardGoods.

- Create a new DataFrame called **hardGoodClicks**, containing only those clicks associated with items in the **somehardGoods** DataFrame:

```
val hardGoodClicks =
trimmedClicks.alias("TC").join(someHardGoods.withColumnRenamed("id",
"doc_id"), Seq("doc_id")).select("TC.*", "name", "longDescription",
"department")
```

*The final **select** statement specifies that we should keep all of the columns from **trimmedClicks**, but only **name**, **longDescription**, and **department** from **someHardGoods**. This will prevent our click documents from becoming bloated with ALL of the catalog metadata.*

Downsampling the Clickstream Data

Lucidworks

- Verify success by checking the number of items in the new DataFrame: **`hardGoodClicks.count`**

```
scala> val hardGoodClicks = trimmedClicks.alias("TC").join(someHardGoods.withColumnRenamed("id",
"doc_id_s"), Seq("doc_id_s")).select("TC.*", "name", "longDescription", "department")
hardGoodClicks: org.apache.spark.sql.DataFrame = [doc_id_s: string, id: string ... 8 more fields]

scala> hardGoodClicks.count
res10: Long = 994176
```

~1M is still more clicks than we need for a compelling demonstration, and represents more processing time and power than we'd care to spend in upcoming labs We will continue to downsample.

Downsampling the Clickstream Data

Lucidworks

Clicks are most impactful when they are part of a trend; e.g., a popular item gathering many clicks, or a user clicking on multiple related items. We will filter our clicks down to these.

- Create a new DataFrame called **userClickCounts** that records the number of clicks that each user has generated:

```
val userClickCounts =
hardGoodClicks.groupBy("user_id").count.withColumnRenamed("count",
"user_count")
```

- Create a new DataFrame called **itemClickCounts** that records the number of times each item has been clicked:

```
val itemClickCounts =
hardGoodClicks.groupBy("doc_id").count.withColumnRenamed("count",
"item_count")
```

Downsampling the Clickstream Data

Lucidworks

- JOIN the user click counts and item click counts onto the **hardGoodClicks,** in a new DataFrame called **clicksWithCounts**:

```
val clicksWithCounts = hardGoodClicks.join(userClickCounts,
Seq("user_id")).join(itemClickCounts, Seq("doc_id"))
```

- Verify success by checking the new **item_count** and **user_count** fields:

```
clicksWithCounts.select("query",
"user_id","doc_id","user_count",
"item_count").show
```

```
+----------------+--------------------+------+----------+----------+
|           query|             user_id|doc_id|user_count|item_count|
+----------------+--------------------+------+----------+----------+
|usb graphics card|eb7510025c8d2bc5c...|1092716|         1|        36|
|   interface card|93c5e2444eb53e89f...|1092716|         1|        36|
|          usb 3.0|5e9bca543105acb33...|1092716|         1|        36|
|          usb 3.0|c900fae60ee79bfff...|1092716|         2|        36|
| pci usb 3.0 card|786b92f3ead7de802...|1092716|         3|        36|
|       usb 3 card|86c641677fe83ec25...|1092716|         1|        36|
|            usb3|b28e30ef7790bd509...|1092716|         1|        36|
|     usb 3.0 card|2d808b34979f08501...|1092716|         1|        36|
| usb express card|43385dede4b380f42...|1092716|         3|        36|
|     usb 3.0 port|7a93e2c26e315cde6...|1092716|         1|        36|
|     usb 3.0 card|5a9aeff9dc5ac403b...|1092716|         1|        36|
|      usb 3.0 pci|fdaca1a249893de82...|1092716|         1|        36|
|      usb 3.0 pci|d77b474803481d3d9...|1092716|         1|        36|
|          usb 3.0|566ee5c50c4944c46...|1092716|         1|        36|
|          usb 3.0|b4b341a9e696a14a9...|1092716|         2|        36|
|          usb 3.0|09432c58133ec1103...|1092716|         2|        36|
|            usb 3|0bf35cab4bc7d563e...|1092716|         1|        36|
|          usb 3.0|93a8e3deeb58a50e1...|1092716|         1|        36|
|          usb 3.0|7a67c0acd32563357...|1092716|         1|        36|
|            usb 3|78d7b728de9c5531e...|1092716|         6|        36|
+----------------+--------------------+------+----------+----------+
only showing top 20 rows
```

Downsampling the Clickstream Data

Lucidworks

Finally, let's limit our click data to the following:

- *Clicks generated by users that have clicked on at least 2 items*
- *Clicks for items that have been clicked on at least 4 times*

- Create a new DataFrame called **usefulClicks** that contains only those clicks that are part of a useful pattern, as described above:

```
val usefulClicks = clicksWithCounts.filter("user_count > 2 AND
item_count > 4").drop("user_count","item_count")
```

- Verify success by checking the number of items in the new DataFrame:

```
usefulClicks.count
```

```
scala> usefulClicks.count
res8: Long = 285666
```

Downsampling the Clickstream Data

Lucidworks

- Check the number of unique users represented in **usefulClicks**:
  `usefulClicks.select("user_id").distinct.count`

- Check the number of unique items represented in **usefulClicks**:
  `usefulClicks.select("doc_id").distinct.count`

```
scala> usefulClicks.count
res8: Long = 285666

scala> usefulClicks.select("user_id_s").distinct.count
res9: Long = 64803

scala> usefulClicks.select("doc_id_s").distinct.count
res10: Long = 6936
```

*Now we have ~285k clicks, from ~65k users and for ~7k items (out of the ~42k total items in **someHardGoods**). This is a useful and manageable dataset.*

Updating Click Dates

Lucidworks

The data in this set is from 2011. That is not a problem per se, but recall that signal aggregation and clickstream relevance boosting are subject to a time decay, so that newer clicks will outweigh older ones. This means that data from 2011 will have almost no weight. We will modify the dates on these clicks so that they are more current, and thus will have a larger effect on relevance and aggregation behaviors.

We will do this by adding a date-updating class to Spark, which we can then apply on the clicks.

- Open the code block paste mode in Spark Shell:
  `:paste`

Updating Click Dates

Lucidworks

- Paste this code block into Spark Shell

The SignalsTimestampUpdater class shifts all timestamps in the input DataFrame equally, such that the latest input timestamp becomes NOW

- Exit paste mode:
  <ctrl> d

```
object SignalsTimestampUpdater extends Serializable {
 import spark.implicits._
 import org.apache.spark.sql.functions._
 import org.apache.spark.sql.DataFrame
 import java.sql.Timestamp

 def updateTimestamps(signalsDF: DataFrame): DataFrame = {
   val now = System.currentTimeMillis()
   val maxDate =
signalsDF.agg(max("timestamp_tdt")).take(1)(0).getAs[Timestamp](
0).getTime
   val diff = now - maxDate
   val addTime = udf((t: Timestamp, diff : Long) => new
Timestamp(t.getTime + diff))

   //Remap some columns to bring the timestamps current
   val newDF = signalsDF
     .withColumnRenamed("timestamp_tdt", "orig_timestamp_tdt")
     .withColumn("timestamp_tdt", addTime($"orig_timestamp_tdt",
lit(diff)))
   newDF
 }
}
```

Updating Click Dates

Lucidworks

- Apply the timestamp update to **usefulClicks**, saving to a new DataFrame called **updatedClicks**:

```
val updatedClicks =
SignalsTimestampUpdater.updateTimestamps(usefulClicks)
```

- Verify success by ckecking the number of items in **updatedClicks**:

```
updatedClicks.count
```

```
scala> val updatedClicks = SignalsTimestampUpdater.updateTimestamps(usefulClicks
)
updatedClicks: org.apache.spark.sql.DataFrame = [doc_id_s: string, user_id_s: st
ring ... 106 more fields]

scala>

scala> updatedClicks.count
res9: Long = 285666
```

Loading Data into Solr

Lucidworks

It is possible, though not normally useful, to load data directly into Solr, bypassing the Fusion layer. It makes sense in this case because we are using data whose structure and cleanliness have been verified, and which are already located on the Fusion server

- Load the **someHardGoods** data into the **Labs** collection you created earlier:
```
someHardGoods.write.format("solr").option("collection",
"Labs").option("soft_commit_secs", "10").save
```

- Load the **updatedClicks** data into the **Labs-signals** collection:
```
updatedClicks.write.format("solr").option("collection",
"Labs_signals").option("soft_commit_secs", "10").save
```

Verifying Successful Data Load

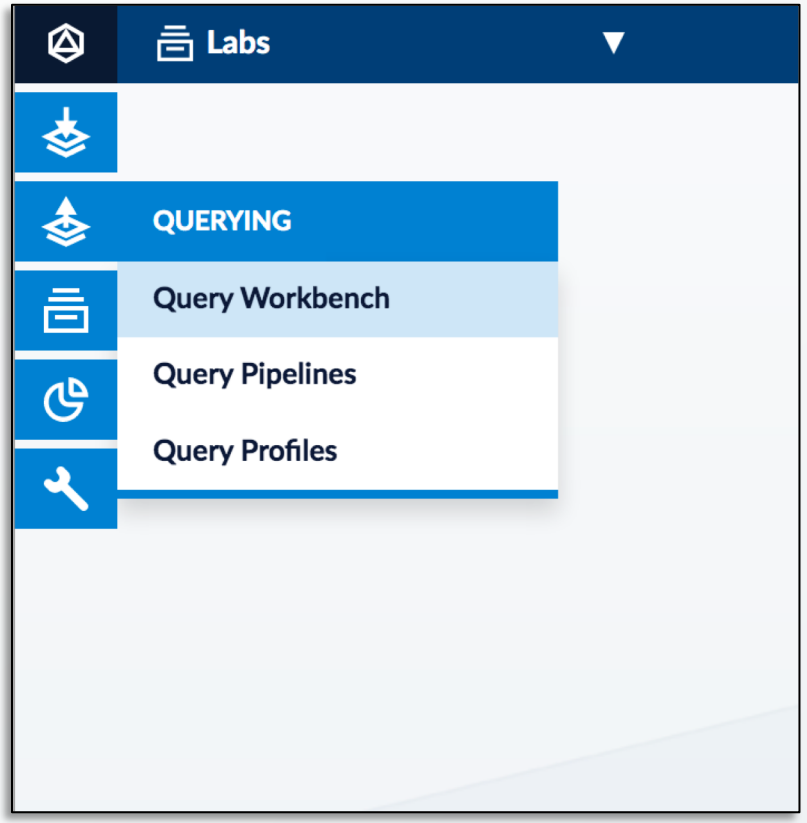
Verifying Successful Data Load

Lucidworks

We will verify successful data loading by using the Query Workbench in Fusion

- In a web browser, open Fusion Admin:
  `<your_vm_ip>:8764`

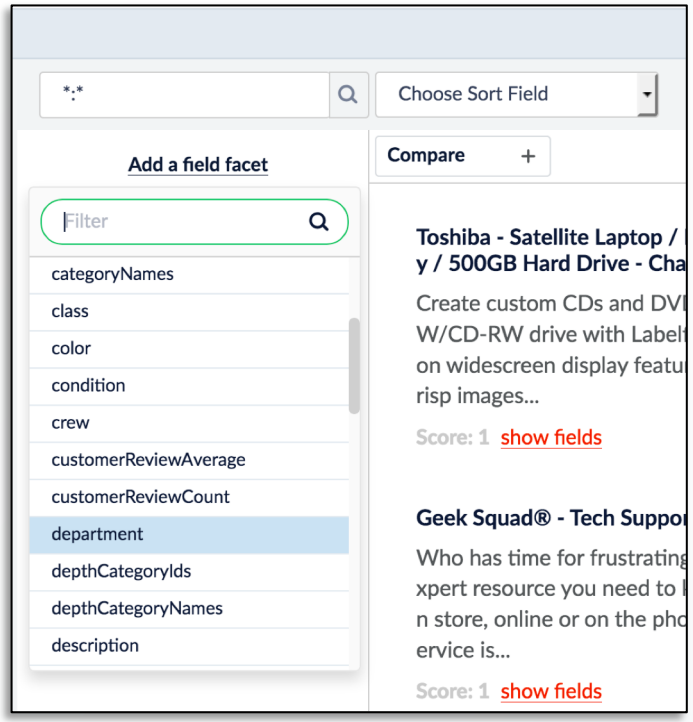
- In the left side menu, go to
  QUERYING > Query Workbench

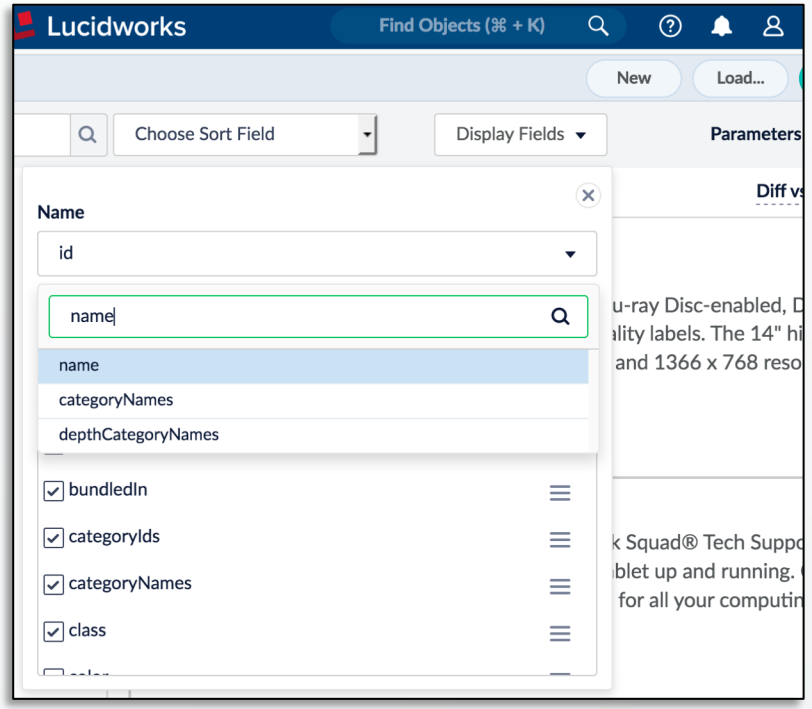


Verifying Successful Data Load

Lucidworks

- In the **Display Fields** dropdown, change the **Name** field from **id** to **name**
- **Add a field facet** on **department**

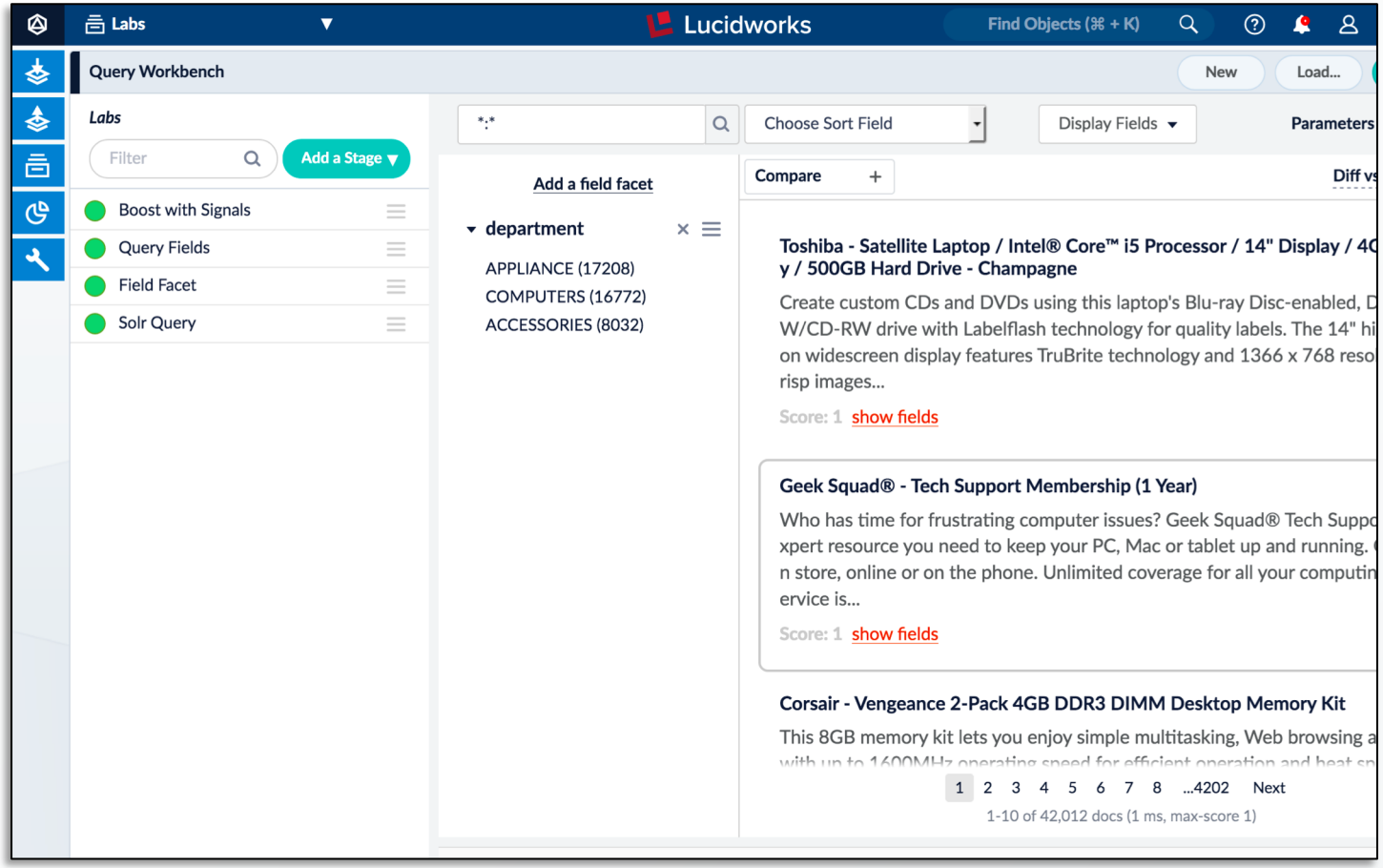




Verifying Successful Data Load

Lucidworks

- The **Labs** Query Workbench should look similar to this:



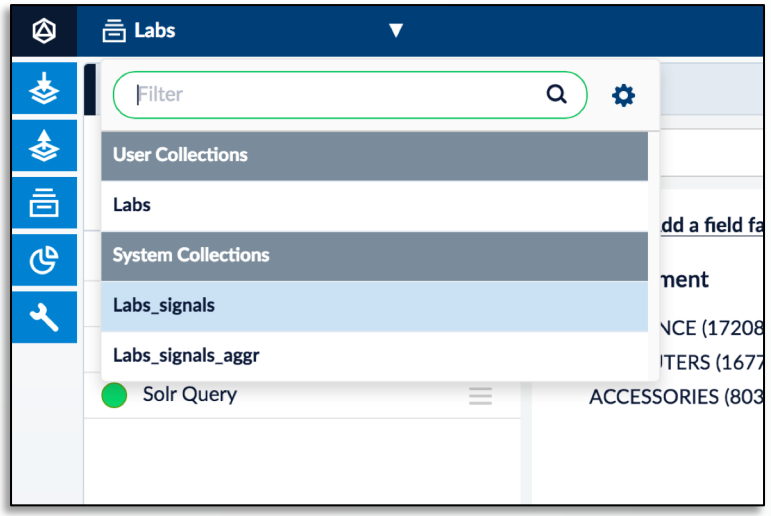
- **Save** these changes.

*If prompted, specify that you are overwriting the existing **Labs** pipeline*

Verifying Successful Data Load

Lucidworks

- In the top left dropdown, change to the **Labs_signals** collection

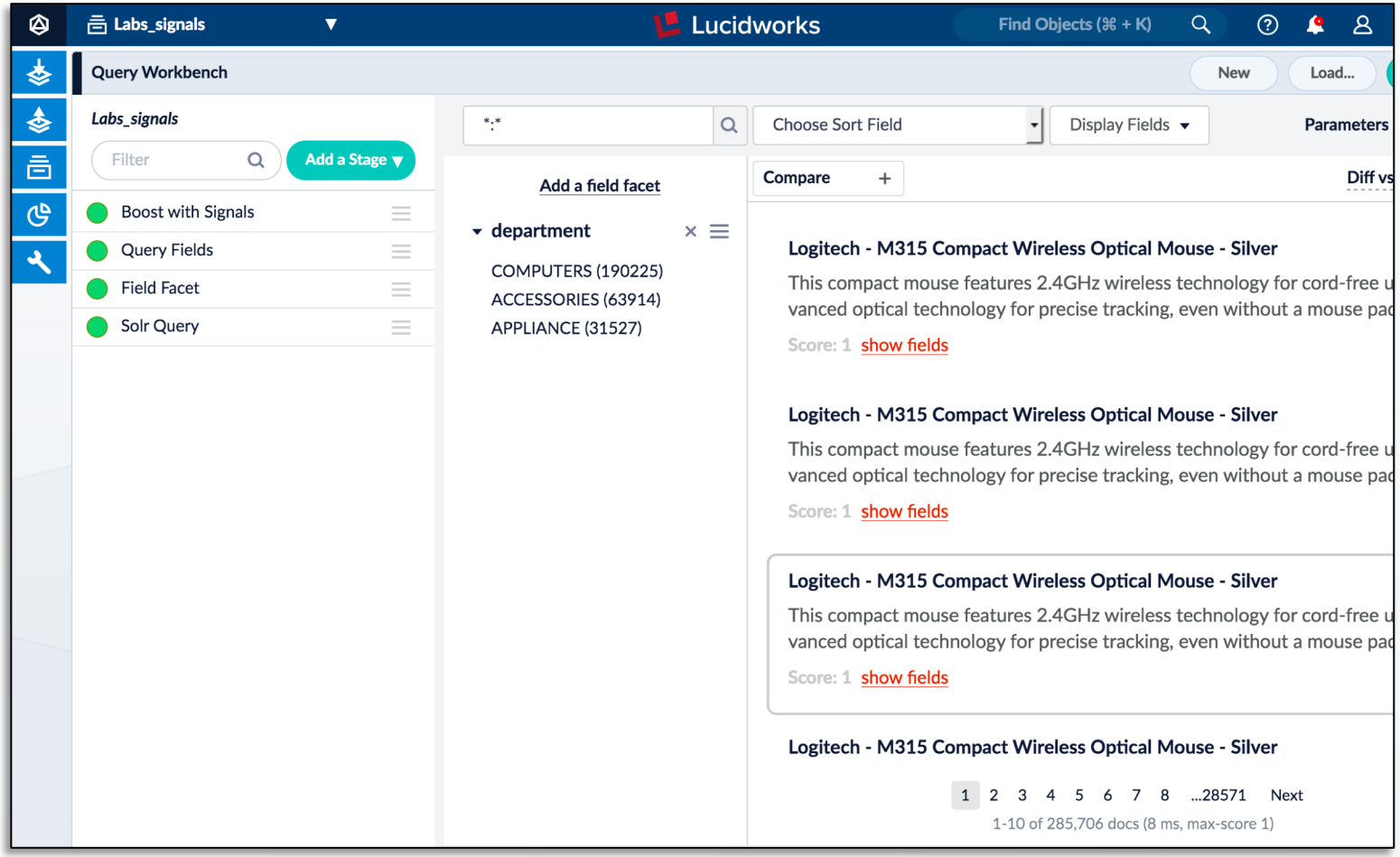


- As before, go to **QUERYING > Query Workbench**
- In the **Display Fields** dropdown, change the **Name** field from **id** to **name**
- **Add a field facet** on **department**

Verifying Successful Data Load

Lucidworks

- The **Labs_Signals** Query Workbench should look similar to this:



- **Save** these changes.

Note that these signals contain all of the data from the associated catalog item. This is NOT recommended design for production, as the signals index would be gigantic. However, this can be very useful for our examples, as the click data is easier for us to read and comprehend.

Closing Spark Shell

Closing Spark Shell

Lucidworks

Recall that when we started Spark Shell, we assigned 6 cores and 6GB of memory to it. These resources will be unavailable to Fusion as long as Spark Shell is running

- In Spark Shell, enter the **quit** command:

```
:quit
```