**Big Data Analysis**

**Analyzing Co-Occurrence Networks**

**with GraphX**

Ji-Hyeong Han

## ( [jhhan@seoultech.ac.kr](mailto:jhhan@seoultech.ac.kr) )

Dept. of Computer Science and Engineering



**Contents**

1. Network Science & GraphX of Spark
2. Data & Process of Network Analysis
3. Practice

2

# Network Science & GraphX of Spark

3

**Network Science**

* Data scientist come in all shapes and size from a remarkably diverse set of academic backgrounds (such as computer science, mathematics, physics, neuroscience, sociology, political science, etc.)

## Although these fields study different things, they all share two important characteristics

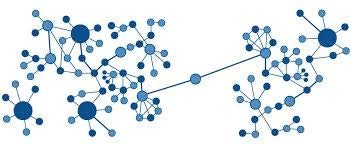
* + All of these fields are interested in understanding relationships between entities, whether between neurons, individuals, or countries
  + The explosion of digital data over the past decade has given researchers access to vast quantities of information about these relationships and required that they develop new skills in order to acquire and manage these data sets

4

**Network Science**

## So, the field of network science was born

* + Network science applies tools from graph theory
  + The mathematical discipline that studies the properties of pairwise relationships (called edges) between a set of entities (called vertices)

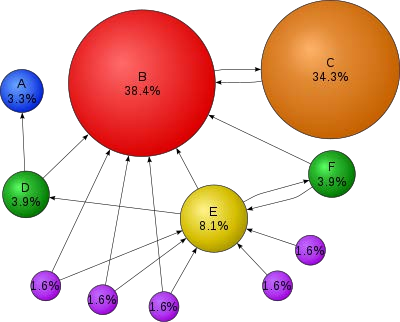


5

**Network Science**

## Network science have had a significant impact in the business world as well

* + Almost every major internet company derives a significant fraction of its value from its ability to build and analyze an important network of relationships better than any of its competitors
  + The recommendation algorithms used at Amazon and Netflix rely on the networks of consumer-item purchases (Amazon) and user-movie ratings (Netflix)
  + Facebook and LinkedIn have built graphs of relationships between people that they analyze in order to organize content feeds, promote advertisements, and broker new connections
  + Google used the PageRank algorithm that the founders developed to create a fundamentally better way to search the web

[PageRank: a way of measuring the importance of website pages] 6

**GraphX**

## Over time, as the graphs became larger and data scientists needed to analyze them faster, new graph-parallel processing frameworks, like Pregel at Google, Giraph at Yahoo, and GraphLab at CMU

* + These frameworks supported fault-tolerant, in-memory, iterative, and graph-centric processing

## We’re going to use a Spark library called GraphX, which extends Spark to support many of the graph-parallel processing tasks

* + Although GraphX cannot handle every graph computation as quickly as the custom graph frameworks do
  + But, it is a Spark library, so it is relatively easy to bring GraphX into your normal data analysis workflow whenever you want to analyze a network- centric data set

7

**GraphX**

## GraphX was created prior to the introduction of DataFrames in Spark 1.3, and its APIs are all designed to work with RDDs

* + More recently, effort is being made to port GraphX to the new APIs developed around DataFrames; it is called GraphFrames
  + GraphFrames promises a number of benefits over the legacy GraphX API, including richer support for reading and writing graphs to serialized data formats via the DataFrame API and expressive graph queries

## So, at this time, we’re going to use the fully functional GraphX API

8

# Data & Process of Network Analysis

9

**Data**

## MEDLINE (Medical Literature Analysis and Retrieval System Online)

* + A database of academic papers that have been published in journals covering the life sciences and medicine
  + The main database contains more than 20 million articles and is updated 5 days a week

## Due to the volume of citations and the frequency of updates, the research community developed an extensive set of semantic tags, called MeSH (Medical Subject Headings)

* + Provide a meaningful framework that can be used to explore relationships between documents to facilitate literature reviews
  + PubGene (now incorporated in Coremine medical) demonstrated applications of biomedical text mining by launching a search engine that allowed users to explore the graph of MeSH terms that connect related documents together

<http://www.coremine.com/medical/#search>

10

**Process of Network Analysis**

## We are going to use Scala, Spark, and GraphX to acquire, transform, and then analyze the network of MeSH terms

* Our goal will be to get a feel for the shape and properties of the citation graph
  + We will attach this from a few different angles to get a full view of the data set
  1. First, we’ll get our feet wet by looking at the major topics and their

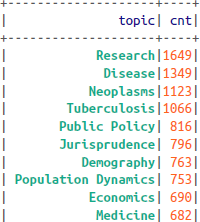
co-occurrences – A simpler analysis that doesn’t require using GraphX

* 1. Then, we’ll look for connected components – Can one follow a path of citations from any topic to any other topic, or is the data actually a set of separate smaller graphs?
  2. We’ll move on to look at the degree distribution of the graph, which gives a sense of how the relevance of topics can vary, and find the topics that are connected to the most other topics

11

**1. Analyzing the MeSH Major Topics and Co-occurrences**

## We can see the overall distribution of tags in our data set by calculating some basic summary statistics, such as the number of records and a histogram of the frequencies of various major MeSH topics



* + The most frequently occurring major topics are, unsurprisingly, some of the most general ones, like Research, Disease, or the slightly less

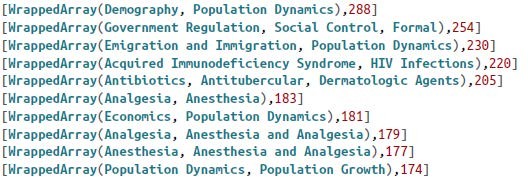
generic Neoplasms (신생물, 종양) and Tuberculosis (결핵)

12

**1. Analyzing the MeSH Major Topics and Co-occurrences**

* Our primary interest is in co-occurring MeSH topics
  + There are 14,548 topics in our data, there are potentially

14,548 \* 14,547 / 2 = 105,814,878 unordered co-occurrence pairs

* + However, the count of co-occurrences reveals that only 213,745 pairs actually appear in the data set, a tiny faction of the possible pairs
  + As we might have suspected from the counts of the most frequently occurring major topics, the most frequently occurring co-occurrence pairs are also relatively uninteresting
  + Most of the top pairs such as Demography (인구통계학) and Population Dynamics (개체군역학), are either the product of two of the most frequently occurring individual topics, or terms that occur together so frequently that they are nearly

synonyms (동의어) 13

**2. Construing Co-occurrence Network**

## As we saw previous step, our standard tools for summarizing data don’t provide us much insight

* + The overall summary statistics we can calculate, like raw counts, don’t give us a feel for the overall structure of the relationships in the network, and the co-occurrence pairs that we can see are usually the ones that we care about least

## What we really want to do is analyze the co-occurrence network

* + By thinking of the topics as vertices in a graph
  + And the existence of a citation record that features both topics as an edge between those two vertices

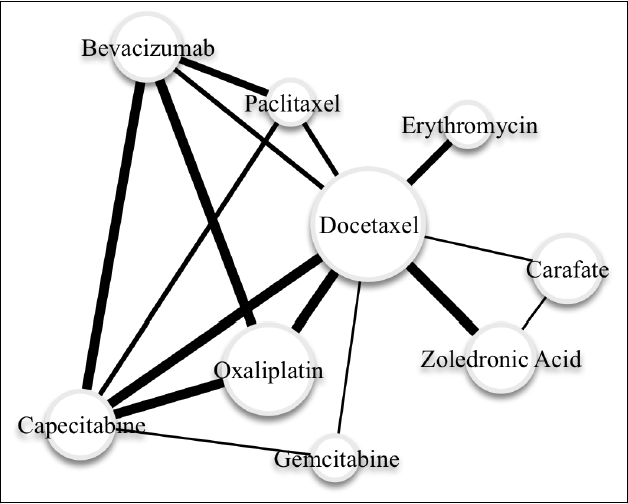
## Then, we can compute network-centric statistics that would help us understand the overall structure of the network and identify interesting local outlier vertices that are worthy of further investigation

14

**2. Construing Co-occurrence Network**

## We can also use co-occurrence networks to identify meaningful interactions between entities that are worthy of further investigation

* + The under graph is a part of co-occurrence graph for combinations of cancer drugs that were associated with adverse events (부작용) in the patients who were taking them



증식당뇨망막병증

15

**2. Construing Co-occurrence Network**

* GraphX is a Spark library that is designed to help us analyze various kinds of networks using the language and tools of graph theory
  + Because GraphX builds on top of Spark, it inherits all of Spark’s scalability properties, which means that it is capable of carrying out analyses on extremely large graphs that are distributed across multiple machines
  + GraphX also integrates well with the rest of the Spark platform
* GraphX is based on two custom RDD implementations that are optimized for working with graphs
  + VertexRDD[ VD ]: a specialized implementation of RDD[ (VertexId, VD)], in which the VertexId type is an instance of Long and is required for every vertex, while VD can be any other type of data associated with the vertex, and is called the vertex attribute
  + EdgeRDD[ ED ]: a specialized implementation of RDD[ Edge[ED]], where Edge is a case class that contains two VertexId values and an edge attribute of type ED.
  + Given both a VertexRDD and an associated EdgeRDD, we can create an instance of the Graph class, which contains a number of methods for efficiently performing graph computations

16

**3. Understanding the Structure of Networks**

## Connected components

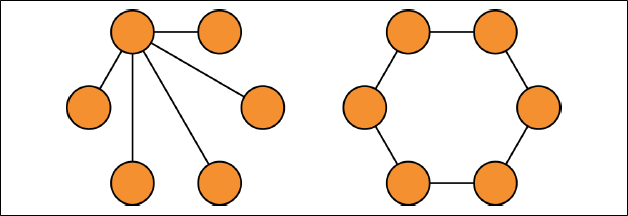
* + One of the most basic things we want to know about a graph is whether or not it is connected
  + In a connected graph, it is possible for any vertex to reach any other vertex by following a path, which is simply a sequence of edges that lead from one vertex to another
  + If the graph isn’t connected, it may be divided into a smaller set of connected subgraphs that we can investigate individually
  + Connectedness is a fundamental graph property, so GraphX includes a built-in method for identifying the connected components in a graph

17

**3. Understanding the Structure of Networks**

## Degree distribution

* + A connected graph can be structured in many different ways
    - For example, there might be a single vertex that is connected to all of the other vertices, but none of those other vertices connect to each other
    - If we eliminated that single central vertex, the graph would shatter into individual vertices
    - We might also have a situation in which every vertex in the graph was connected to exactly two other vertices, so that the entire connected component formed a giant loop



* + To gain additional insight into how the graph is structured, it’s helpful to look at the degree of each vertex, which is simply the number of edges that a particular vertex belongs to

18

**4. Filtering Out Noisy Edges**

## In the current co-occurrence graph of MeSH, the edges are weighted based on the count of how often a pair of concepts appears in the same paper

* + The problem with this simple weighting scheme is that it doesn’t distinguish concept pairs that occur together whether because they have a meaningful semantic relationship or because they happen to both occur frequently for any type of document

## We need to use a new edge-weighting scheme that takes into account how “interesting” or “surprising” a particular pair of concepts

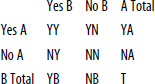
* We will use Pearson’s chi-squared test to calculate this “interestingness” in a principled way – that is, to test whether the occurrence of a particular concept is independent from the occurrence of another concept

19

**4. Filtering Out Noisy Edges**

## Chi-Squared test

* + For any pair of concepts A and B, we can create a 2x2 contingency table



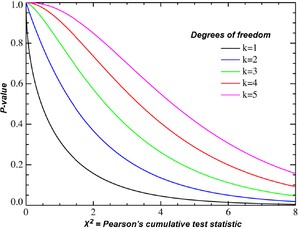


* + A large chi-squared statistic indicates that the variables are less likely to be independent, and thus we find the pair of concepts more interesting
  + Now, we can map the calculated chi-squared value to the edge as a new edge weight

20

**4. Filtering Out Noisy Edges**

* We want use the chi-squared statistic value to filter out edges that don’t appear to have any meaningful relationship between the co-occurring concepts
* For a 2x2 contingency table in which there is no relationship between the variables, we expect that the value of the chi-square metric will follow the chi- squared distribution with one degree of freedom
  + The 99.999th percentile of the chi-squared distribution with one degree of freedom is approximately 19.5, so let’s try this value as a cutoff to eliminate edges from the graph, leaving us with only those edges where we are extremely confident that they represent an interesting co-occurrence relationship

21

# Practice

22

**Download Data**

## We are going to use a part of MEDLINE data

* + Open the terminal from Jupyter Notebook and run the following commands

# mkdir medline\_data # cd medline\_data

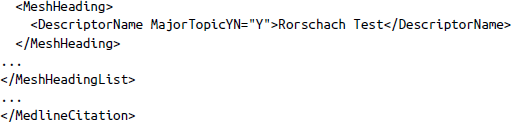
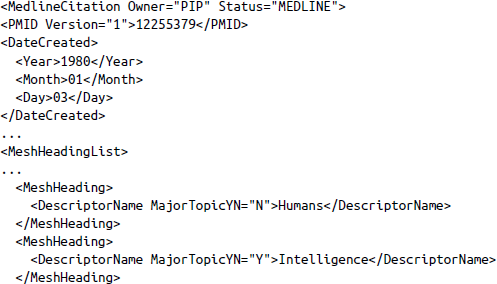
# wget ftp://ftp.nlm.nih.gov/nlmdata/sample/medline/medsamp2016a.xml.gz # gunzip \*.gz

# ls -ltr

23

**Download Data**

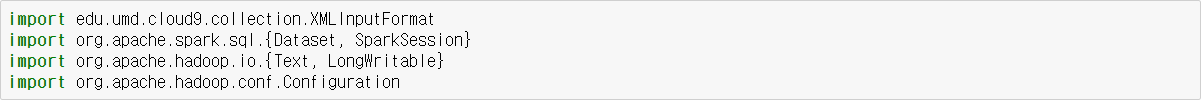
## Example of data



24

**Data Preprocessing**

## Import libraries

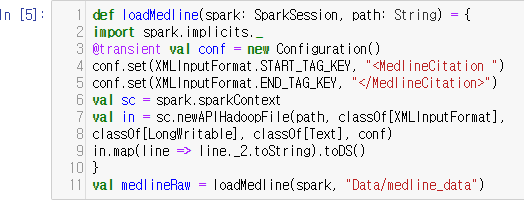


25

**Data Preprocessing**

## Write a function to read the XML-formatted MEDLINE data into the shell

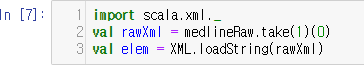
* + You should carefully modify the data path according to yours

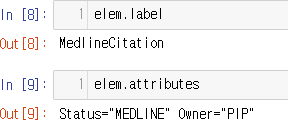


26

**Data Preprocessing**

## Parsing XML documents with Scala’s XML library

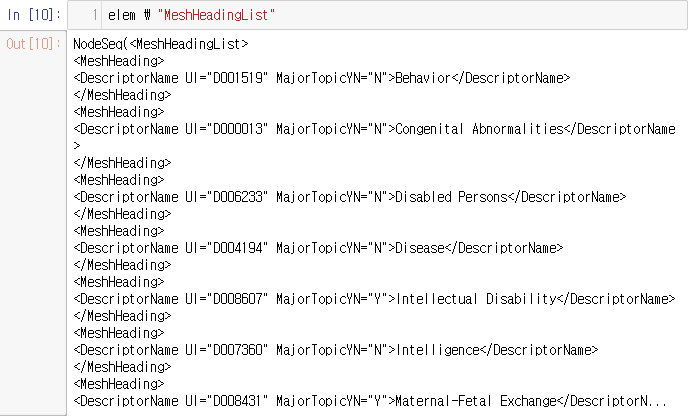
* + Scala’s support for parsing and querying XML document is truly excellent, and we will be availing ourselves of it to help extract the information we want from the MEDLINE citations
    - To see how it works, let’s get started by pulling the unparsed first citation record into our Spark shell
    - The elem variable is an instance of the scala.xml.Elem class, which is how Scala represents an individual node in an XML document
    - The class contains a number of built-in functions for retrieving information about the node and its contents



27

**Data Preprocessing**

## Parsing XML documents with Scala’s XML library (cont’d)

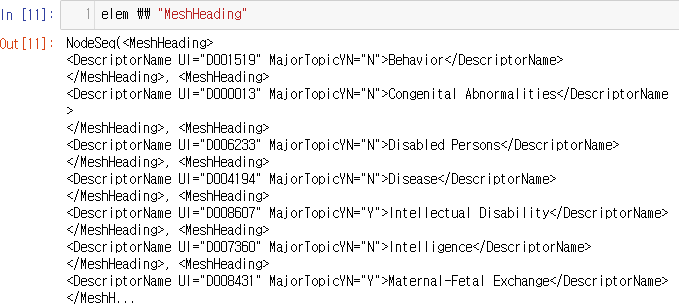
* + elem variable also contains a small set of operators for finding the children of a given XML node
    - For retrieving a node’s direct children by name, is called \

28

**Data Preprocessing**

## Parsing XML documents with Scala’s XML library (cont’d)

* + elem variable also contains a small set of operators for finding the children of a given XML node
    - To extract non-direct children of a given node, we need to use \\

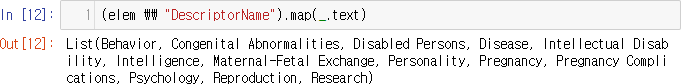


29

**Data Preprocessing**

## Parsing XML documents with Scala’s XML library (cont’d)

* + elem variable also contains a small set of operators for finding the children of a given XML node
    - We can also use \\ operator to get at the DescriptorName entries directly

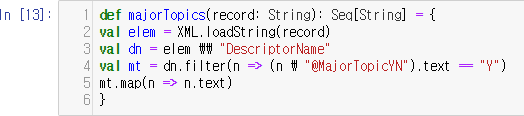


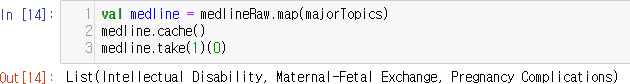
30

**Data Preprocessing**

## You can see that each of the DescriptorName entries has an attribute called MajorTopicYN that indicates whether or not this MeSH tag was a major topic

* + We can use this to create a filter that only returns the names of the major MeSH tags for each article

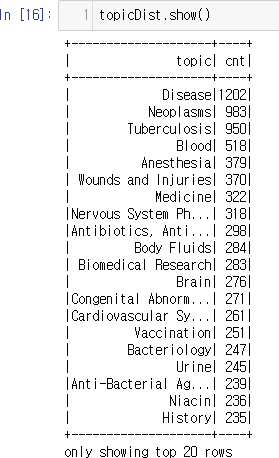
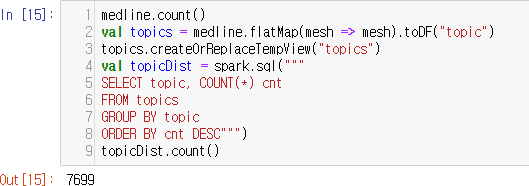




31

**1. Analyzing the MeSH Major Topics and Co-occurrences**

## Calculate the number of records



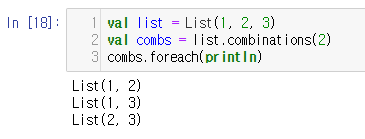
32

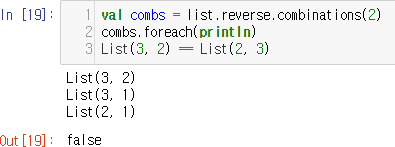
**1. Analyzing the MeSH Major Topics and Co-occurrences**

## To get the co-occurrences, we need to generate all of the two- element subsets of list of strings

* + We can use Scala’s Collections library which has a built-in method called

combinations to make generating these sublists extremely easy

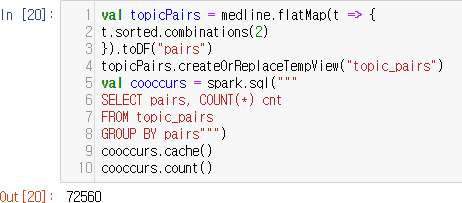
* + - This is the simple example
  + We need to be careful that all of the lists are sorted in the same way
    - Because the lists returned from the combinations function depend on the order of the input elements, and lists with the same elements in a different order are not equal to one another

 33

**1. Analyzing the MeSH Major Topics and Co-occurrences**

## To get the co-occurrences, we need to generate all of the two- element subsets of list of strings (cont’d)

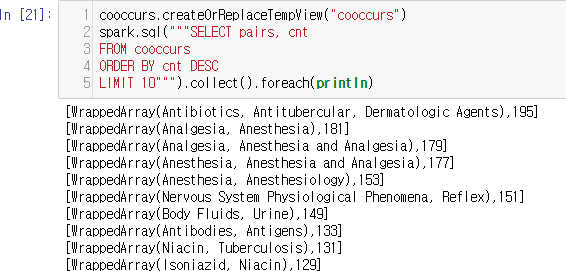
* + Now, we generate two-element subsets for our data



34

**1. Analyzing the MeSH Major Topics and Co-occurrences**

## Look at the most frequently appearing co-occurrence pairs in the data

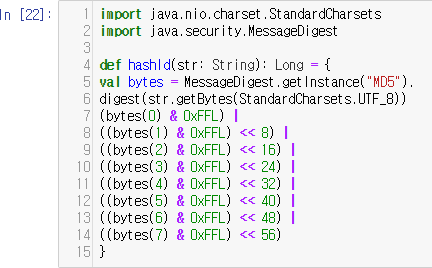


35

**2. Construing Co-occurrence Network**

## The first requirement in creating a graph is to have a Long value that can be used as an identifier for each vertex in the graph

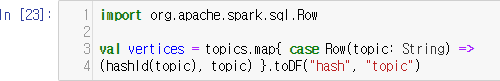
* + So, we need a way to come up with a unique 64-bit value that can be associated with each topic string
  + We are going to copy a hashing implementation from Google’s Guava Library to create a unique 64-bit identifier for each topic using the MD5 hashing algorithm



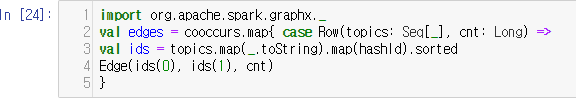
36

**2. Construing Co-occurrence Network**

* We can apply this hashing function to our MEDLINE data to generate a data frame that will be the basis for the set of vertices in our co-occurrence graph



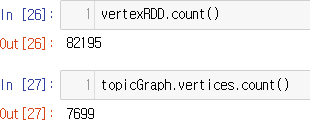
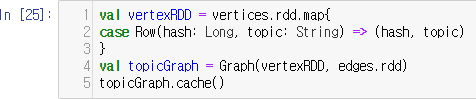
* We will generate the edges for the graph from the co-occurrence counts that we created in the previous section



37

**2. Construing Co-occurrence Network**

* Now we have both the vertices and the edges
* We can create our Graph instance and mark it as cached

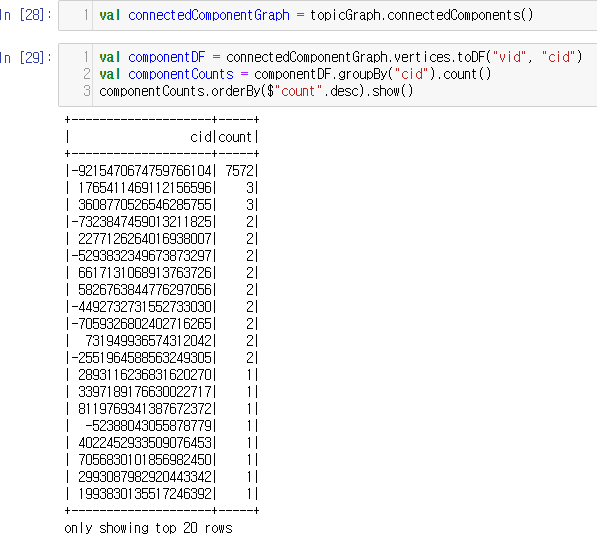


38

**3. Understanding the Structure of Networks**

* Call the connectedComponents method on the graph to see the graph structure

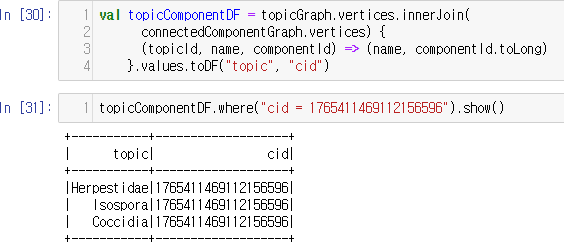
The largest component includes more than 90% of the vertices, while the second largest contains only 3 vertices



39

**3. Understanding the Structure of Networks**

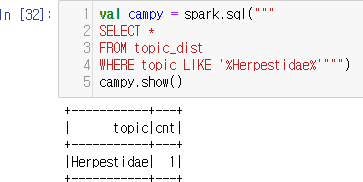
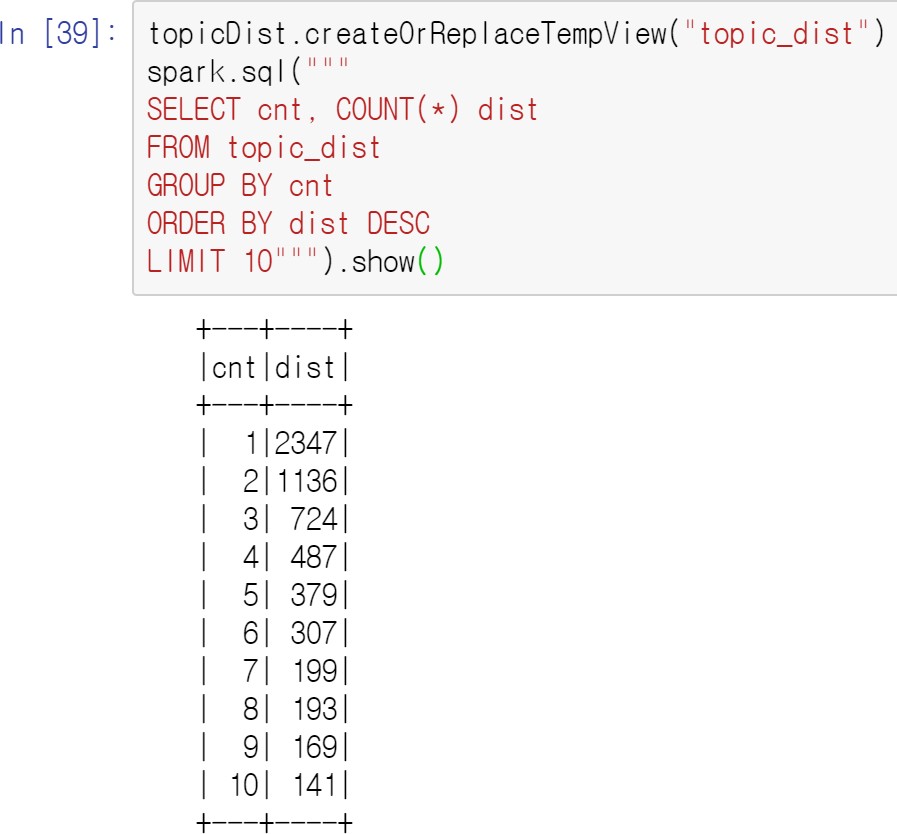
* It’s worthwhile to take a look at the topics for some of these smaller components, if only to understand why they were not connected to the largest component
  + To see the names of the topics, we’ll need to joint the VertexRDD for the connected components graph with the vertices from our original concept graph



40

**3. Understanding the Structure of Networks**

## Let’s take a look at the original topic distribution to see if there are any similarly named topics of smaller connected graph

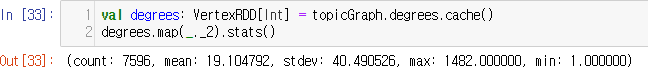


41

**3. Understanding the Structure of Networks**

* Calculate the degree of each vertex by calling the degrees method on the

Graph object



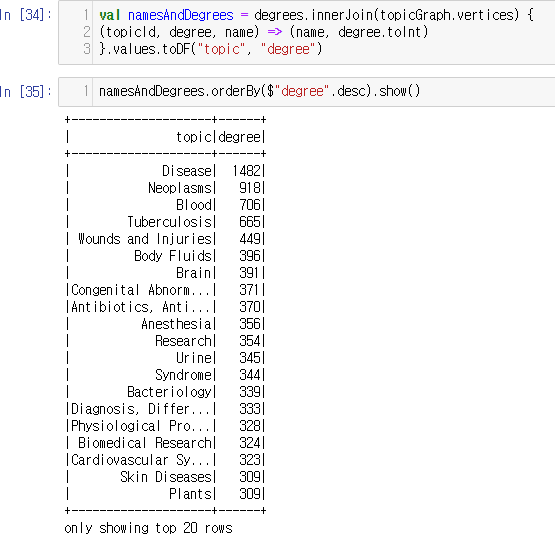
* + Note that the mean is relatively small, indicating that the average vertex in the graph is only connected to a small fraction of the other nodes, the maximum value indicates that there is at least one highly connected node in the graph

42

**3. Understanding the Structure of Networks**

* Let’s take a closer look at the concepts for these high-degree vertices by joining the degrees VertexRDD to the vertices in the concept graph

gh-degree vertices



Unsurprisingly, most of the hi refer to the generic concepts

43

**4. Filtering Out Noisy Edges**

* Calculate chi-squared statistic
  + Get total number of entries



* + Get the counts of how many documents feature each concept



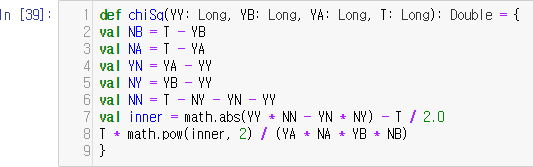
* + Create a new graph using the above VertexRDD of counts as the vertex along with the existing edges RDD



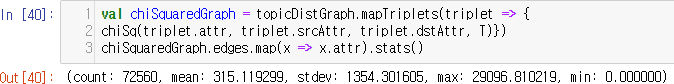
44

**4. Filtering Out Noisy Edges**

* Calculate chi-squared statistic (cont’d)

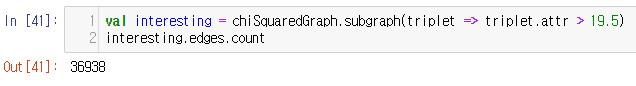


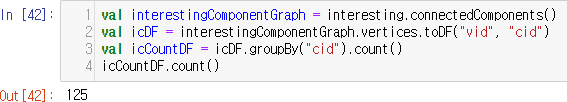
* Apply this method to transform the value of the graph edges



45

**4. Filtering Out Noisy Edges**

* Apply cutoff (=19.5) to eliminate edges from the graph
* Analyzing the filtered graph

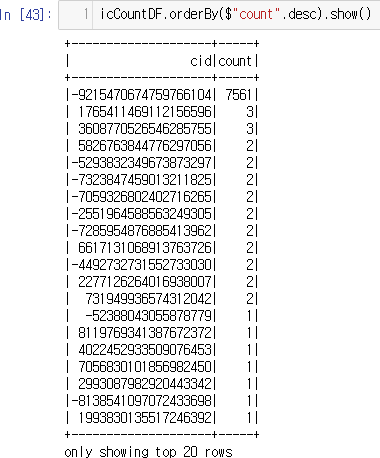


46

**4. Filtering Out Noisy Edges**

## Analyzing the filtered graph (cont’d)

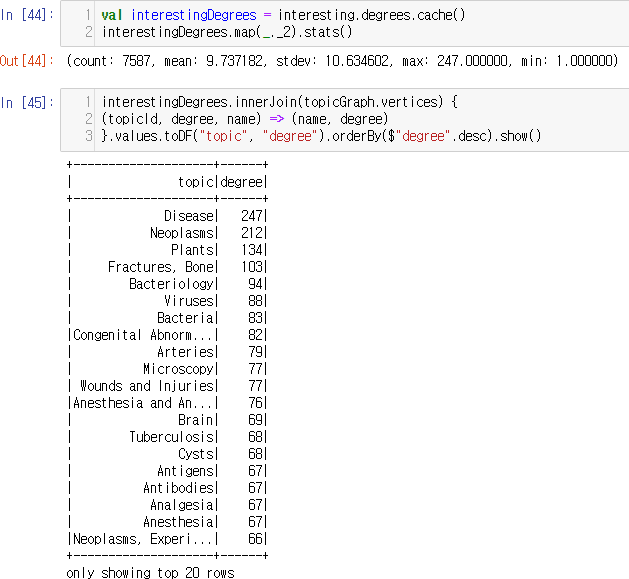
There is some decrease of connected component (from 7572 to 7561)



47

**4. Filtering Out Noisy Edges**

* Analyzing the filtered graph (cont’d)

Our chi-squared filtering criterion appears to have the desired effect

It’s eliminating edges in our graph related to generic concepts, while preserving the edges in the rest of the graph that represent meaningful and interesting semantic relationships between concepts

48