**Big Data Analysis**

**Understanding Wikipedia with Latent Semantic Analysis**

## Ji-Hyeong Han

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

Dept. of Computer Science and Engineering



**Contents**

1. Latent Semantic Analysis (LSA)
2. Document-Term Matrix
3. Term Frequency \* Inverse Document Frequency (TF-IDF)
4. Singular Value Decomposition (SVD)
5. Practice

2

# Latent Semantic Analysis (LSA)

3

**Unstructured Text Data**

### Most of the work in data engineering

* + Assembling data into some sort of queryable format

### Structure data

* + When the structured data is tabular, we can use SQL for query

### Unstructured text data

* + A whole different set of challenges
  + Using LSA (Latent Semantic Analysis)

4

**Latent Semantic Analysis (LSA)**

### A technique in natural language processing

* + Seek to better understand a corpus of documents and the relationships between the words in those documents

### Attempt to distill the corpus into a set of relevant concepts

* Three attributes of each concept
  + A level of affinity for each document in the corpus
  + A level of affinity for each term in the corpus
  + An importance score reflecting how useful the concept is in describing variance in the data set

5

**Latent Semantic Analysis (LSA)**

### For example

* + LSA might discover a concept with high affinity for the terms “Asimov” and “robot”
  + And high affinity for the documents “foundation series” and “science fiction”

### By selecting only the most important concepts, LSA can throw away some irrelevant noise and merge co-occurring strands to come up with a simpler representation of the data

* LSA can provide scores of similarity between terms and other terms, between documents and other documents, and between terms and documents
  + A deeper understanding than simply on counting occurrences and co- occurrences of words
  + These similarity measures are ideal for tasks such as finding the set of documents relevant to query terms, grouping documents into topics, and finding related words 6

**Document-Term Matrix**

### Before performing any analysis, LSA requires transforming the raw text of the corpus into a document-term matrix

* + Each column represents a term that occurs in the corpus
  + Each row represents a document
  + Each element represents the importance of the column’s term to the row’s document

7

**TF-IDF**

### TF-IDF (Term Frequency \* Inverse Document Frequency)

* + To calculate each element of document-term matrix

- 𝑇𝑇𝑇𝑇𝑇𝑇𝑇𝑇 𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑓𝑓𝑇𝑇𝑓𝑓𝑓𝑓𝑓𝑓 = 𝑇𝑇𝑇𝑇𝑇𝑇𝑇𝑇 𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑓𝑓𝑇𝑇𝑓𝑓𝑓𝑓𝑓𝑓 𝑖𝑖𝑓𝑓 𝑑𝑑𝑑𝑑𝑓𝑓𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑑𝑑

𝑇𝑇𝑑𝑑𝑑𝑑𝑇𝑇𝑇𝑇 𝑑𝑑𝑇𝑇𝑇𝑇𝑇𝑇 𝑖𝑖𝑓𝑓 𝑑𝑑𝑑𝑑𝑓𝑓𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑑𝑑

- 𝐼𝐼𝑓𝑓𝐼𝐼𝑇𝑇𝑇𝑇𝐼𝐼𝑇𝑇 𝑑𝑑𝑑𝑑𝑓𝑓𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑑𝑑 𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑓𝑓𝑇𝑇𝑓𝑓𝑓𝑓𝑓𝑓 = log

𝑇𝑇𝑑𝑑𝑑𝑑𝑇𝑇𝑇𝑇 𝑓𝑓𝑓𝑓𝑇𝑇𝑛𝑛𝑇𝑇𝑇𝑇 𝑑𝑑𝑓𝑓 𝑑𝑑𝑑𝑑𝑓𝑓𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑑𝑑 𝑖𝑖𝑓𝑓 𝑓𝑓𝑑𝑑𝑇𝑇𝑐𝑐𝑓𝑓𝑐𝑐

𝑓𝑓𝑓𝑓𝑇𝑇𝑛𝑛𝑇𝑇𝑇𝑇 𝑑𝑑𝑓𝑓 𝑑𝑑𝑑𝑑𝑓𝑓𝑓𝑓𝑇𝑇𝑇𝑇𝑓𝑓𝑑𝑑𝑐𝑐 𝑤𝑤𝑤𝑇𝑇𝑇𝑇𝑇𝑇 𝑑𝑑𝑤𝑇𝑇 𝑑𝑑𝑇𝑇𝑇𝑇𝑇𝑇 𝑇𝑇𝑐𝑐𝑐𝑐𝑇𝑇𝑇𝑇𝑇𝑇𝑐𝑐

### Two intuitions of TF-IDF

* 1. We would expect that the more often a term occurs in a document, the more important it is to that document
  2. Not all terms are equal in a global sense

It is more meaningful to encounter a word that occurs rarely in the entire corpus than a word that appears in most of the document

Thus the metric uses the *inverse* of the word’s appearance in documents in the full corpus

8

**TF-IDF**

### log of IDF

* + The frequency of words in a corpus tends to be distributed exponentially
  + A common word will often appear ten times as often as a middly common word, which in turn might appear ten or a hundred times as often as a rare word
  + Basing a metric on the raw IDF would give rare words too huge weight and practically ignore the impact of all other words
  + To capture this distribution, the scheme uses the log of IDF
  + This mellows the differences in document frequencies by transforming the multiplicative gaps between them into additive gaps

9

**Assumptions in LSA**

### The model relies on a few assumptions

1. It treats each document as a “bag of words”, meaning that it pays no attention to the ordering of words, sentence structure, or negations
2. The model has difficulty dealing with polysemy, the use of the same word for multiple meanings
   * For example, the model can’t distinguish between the use of “band” in “Radiohead is the best band ever” and “I broke a rubber band”
   * If both sentences appear often in the corpus, it may come to associate “Radiohead” with “rubber”

10

**Singular Value Decomposition (SVD)**

### Now we have document-term matrix

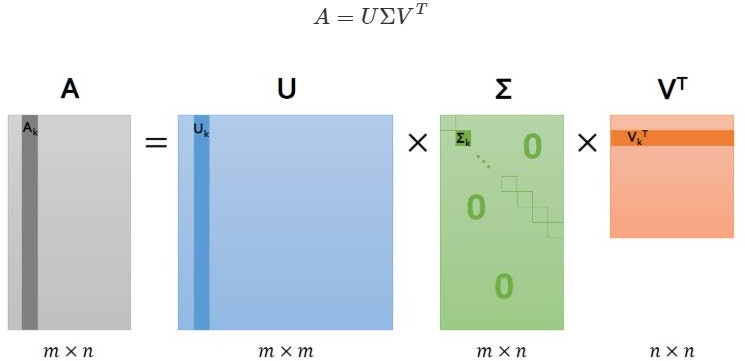
* The LSA can proceed to the factorization and dimensionality reduction, by SVD
* SVD
  + SVD takes 𝑇𝑇 × 𝑓𝑓 matrix and returns three matrices that approximately equal it when multiplied together

𝑀𝑀 ≈ 𝑈𝑈𝑈𝑈𝑉𝑉𝑇𝑇

* + 𝑈𝑈: 𝑇𝑇 × 𝑘𝑘 matrix whose rows form an orthonormal basis for the document space
  + 𝑈𝑈: 𝑘𝑘 × 𝑘𝑘 diagonal matrix, each of whose entries correspond to the strength of one of the concepts
  + 𝑉𝑉𝑇𝑇: 𝑘𝑘 × 𝑓𝑓 matrix whose columns form an orthonormal basis for the term space

11

**Singular Value Decomposition (SVD)**

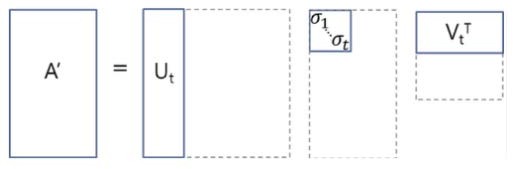


12

**Singular Value Decomposition (SVD)**

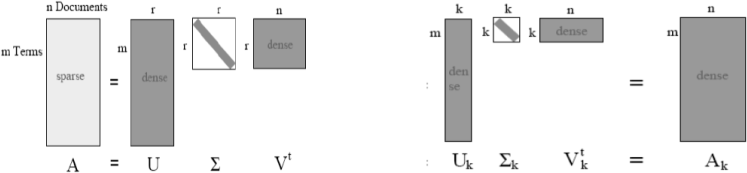
### Truncated SVD

* + Select top t singular values



13

**LSA**



14

**Singular Value Decomposition (SVD)**

### Dimensionality reduction by choosing the value of 𝑘𝑘

* + 𝑇𝑇: the number of documents
  + 𝑓𝑓: the number of terms
  + The SVD is parameterized with a number 𝑘𝑘, less than or equal to 𝑓𝑓, which indicates how many concepts to keep around
  + When 𝑘𝑘 = 𝑓𝑓, the product of the factor matrices reconstitutes the original matrix exactly
  + When 𝑘𝑘 < 𝑓𝑓, the multiplication results in a low-rank approximation of the original matrix
  + 𝑘𝑘 is typically chosen to be much smaller than 𝑓𝑓

15

**Singular Value Decomposition (SVD)**

### Each vector of term, document, concept spaces

* + A vector in term space
    - A vector with a weight on every term
  + A vector in document space
    - A vector with a weight on every document
  + A vector in concept space
    - A vector with a weight on every concept
  + Each vector of term, document, or concept defines an axis in its respective space (because it is orthonormal basis), and the weight ascribe to the term, document, or concept means a length along that axis

16

**Singular Value Decomposition (SVD)**

* 𝑉𝑉 is an 𝑓𝑓 × 𝑘𝑘 matrix
  + Each row corresponds to a term and each column corresponds to a concept
  + It defines a mapping between term space (the space where each point is an *n*- dimensional vector holding a weight for each term) and concept space (the space where each point is *k*-dimensional vector holding a weight for each concept)
* 𝑈𝑈 is an 𝑇𝑇 × 𝑘𝑘 matrix
  + Each row corresponds to a document and each column corresponds to a concept
  + It defines a mapping between document space and concept space
* 𝑈𝑈 is a 𝑘𝑘 × 𝑘𝑘 diagonal matrix
  + Holds the singular values
  + Each diagonal element in 𝑈𝑈 corresponds to a single concept
  + The magnitude of each of these singular values corresponds to the importance of that concept

17

**Singular Value Decomposition (SVD)**

### So SVD outputs a bunch of numbers

How can we inspect these to verify they actually relate to anything useful?

* The 𝑉𝑉 matrix contains a column for every concept and a row for every term
  + The value at each position can be interpreted as the relevance of that term to that concept
  + So, we can find the most relevant terms to each of the top concepts from

𝑉𝑉 matrix

18

# Practice

19

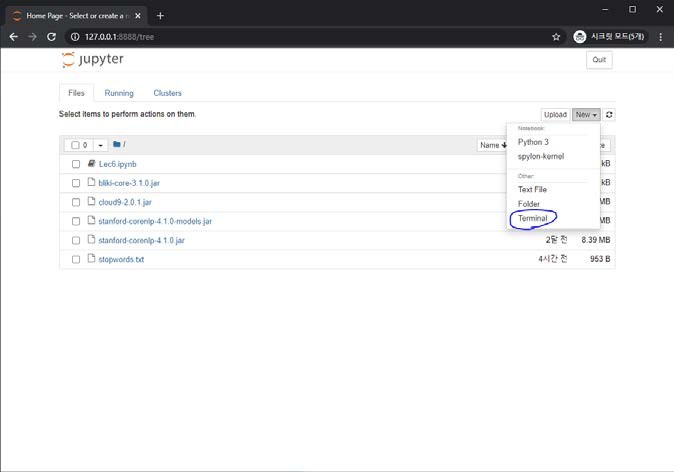
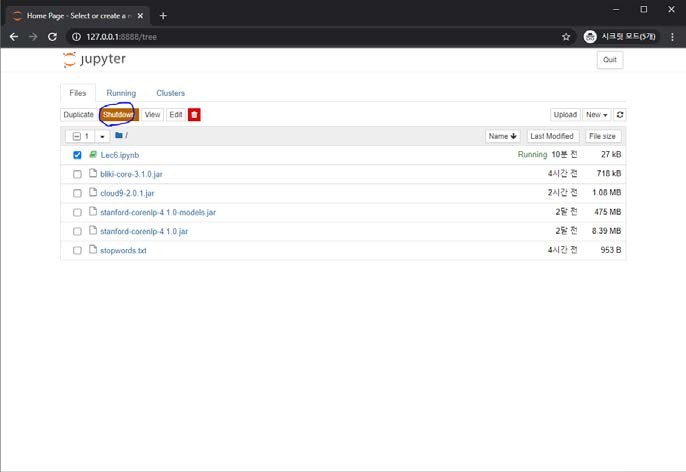
**Import Dependencies**

### In this practice, we rely on several libraries to make our lives easier

* Download prebuild libraries from the following URL
  + <https://www.dropbox.com/s/iqwguxrndz4yn53/Lec6.zip?dl=0>

### Unzip and Upload to Jupyter Notebook

* Shutdown all running notebook files and open the terminal



20

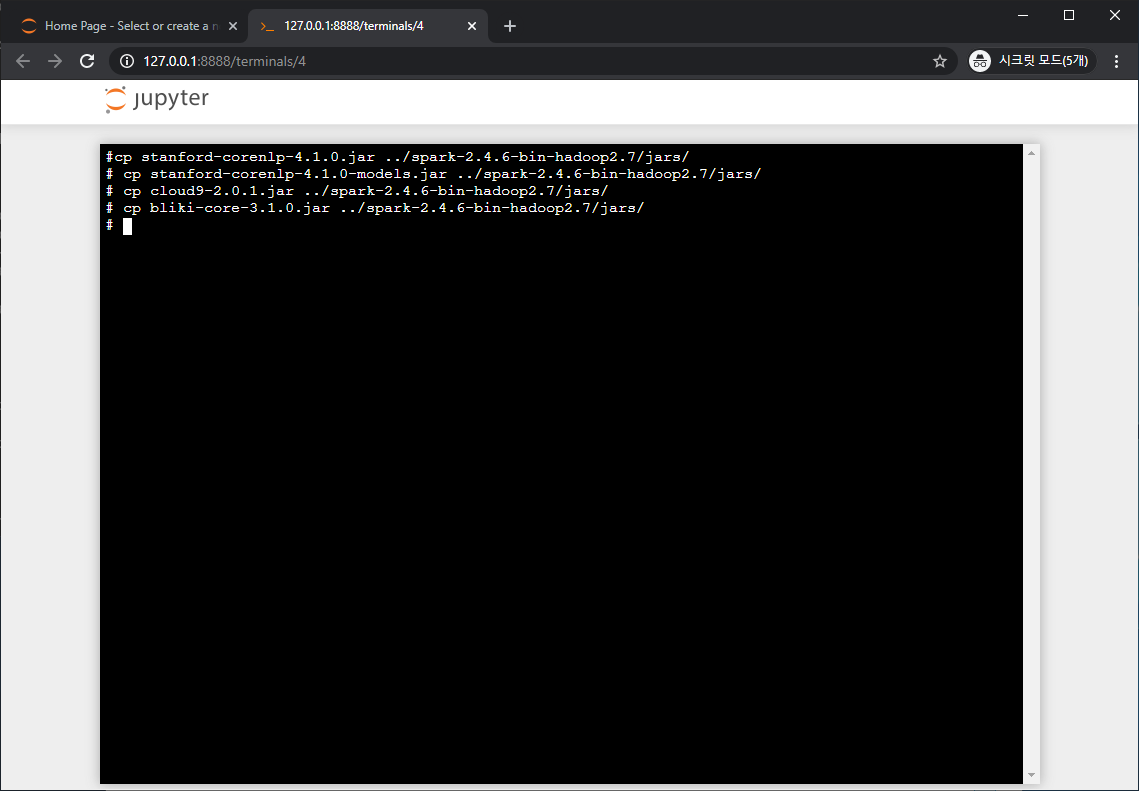
**Import Dependencies**

### Add prebuild libraries to Spark

# cp stanford-corenlp-4.1.0.jar ../spark-2.4.6-bin-hadoop2.7/jars/

# cp stanford-corenlp-4.1.0-models.jar ../spark-2.4.6-bin-hadoop2.7/jars/ # cp cloud9-2.0.1.jar ../spark-2.4.6-bin-hadoop2.7/jars/

# cp bliki-core-3.1.0.jar ../spark-2.4.6-bin-hadoop2.7/jars/

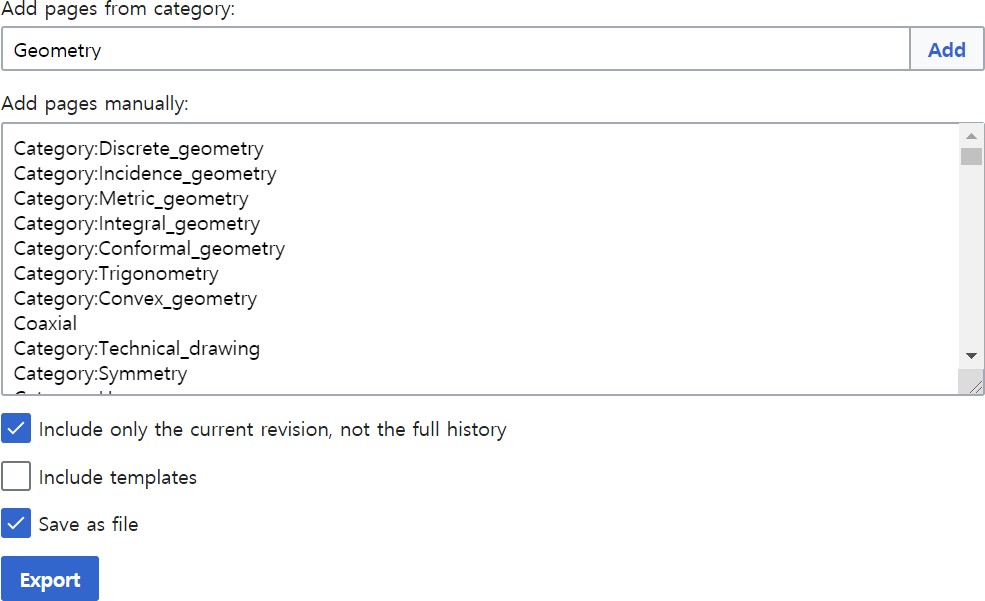


21

**Download Data**

* https://en.wikipedia.org/wiki/Special:Export
* Add pages from category
  + Geometry

### Export and rename “Wikipedia-Geometry.xml”

22

**Possible Errors**

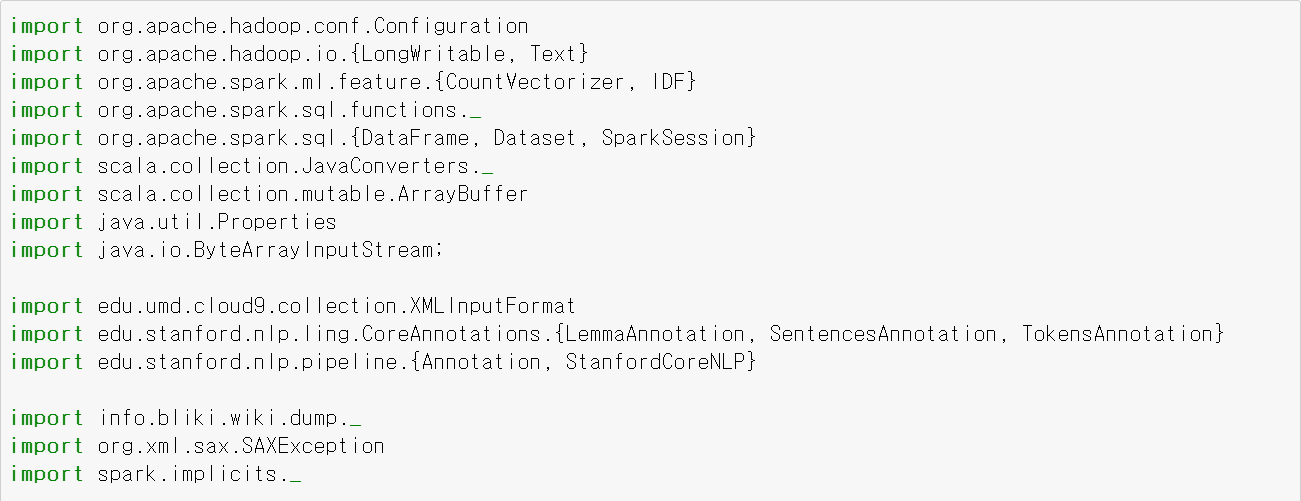
### If you have a following error in the following steps, please re-run that particular cell

* + error: missing or invalid dependency detected while loading class file 'QualifiedTableName.class'.Could not access type AnyRef in ackage scala, because it (or its dependencies) are missing. Check your build definition for missing or conflicting dependencies.

23

**Start Practice**

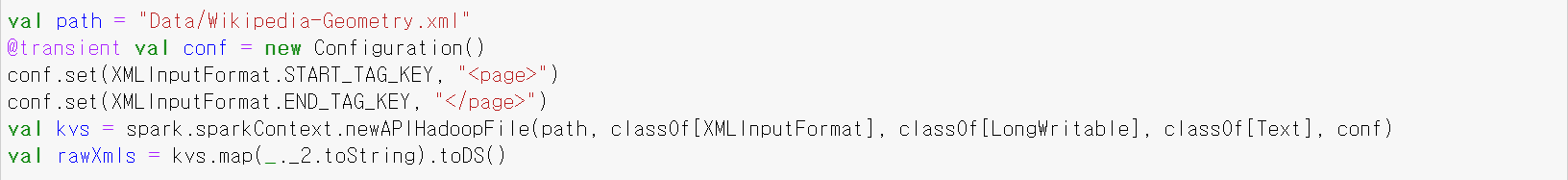
### Import libraries



24

**Preprocessing of Wikipedia dump**

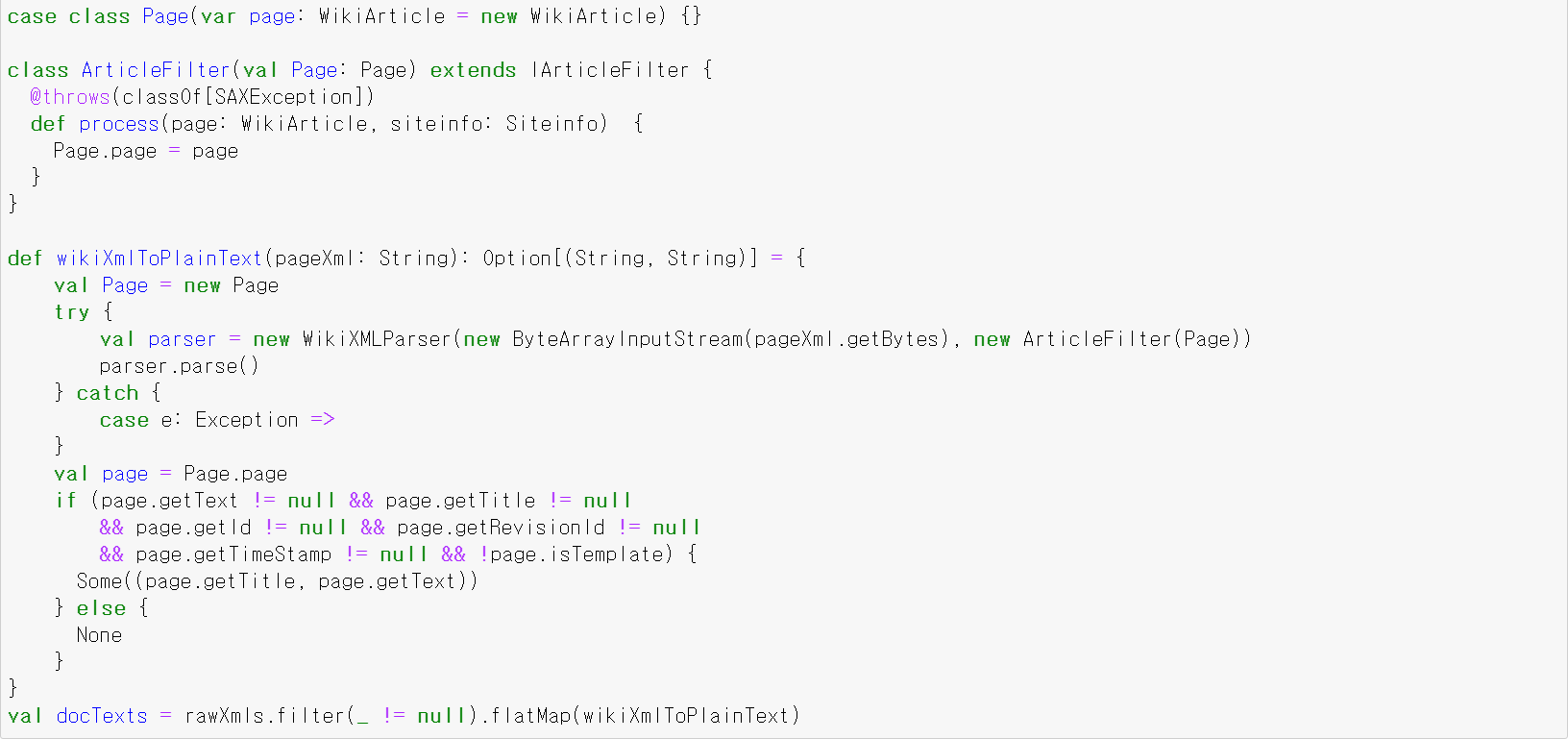
### Split up the Wikipedia dump into documents using Cloud9 library we use (Cloud9 is derived from the Apache Mahout project)



25

**Preprocessing of Wikipedia dump**

### Turning the Wiki XML into the plain text by using Cloud9 library



26

**Preprocessing of Wikipedia dump**

### Lemmatization

* + With the plain text in hand, next we need to turn it into a bag of terms
  + This step requires care for a couple of reasons
    - Common words like “the” and “is” take up space but at best offer no useful information to the model.

So, filtering out a list of “stopwords” can both save space and improve fidelity

* + - Terms with the same meaning can often take slightly different forms.

For example, “monkey” and “monkeys” do not deserve to be separate terms. Nor do “nationalize” and “nationalization”.

Combining these different inflectional forms into single terms is called lemmatization

* + - The Stanford Core NLP project provides an excellent lemmatizer and we are going to use it

27

**Preprocessing of Wikipedia dump**

### Lemmatization (cont’d)

1. First, you should make sure there is “stopwords.txt” in your working directory. So, please move “stopwords.txt” file to your working directory (You can find “stopwords.txt” is at /aas/ch06-lsa/src/main/resources)
2. Doing lemmatization

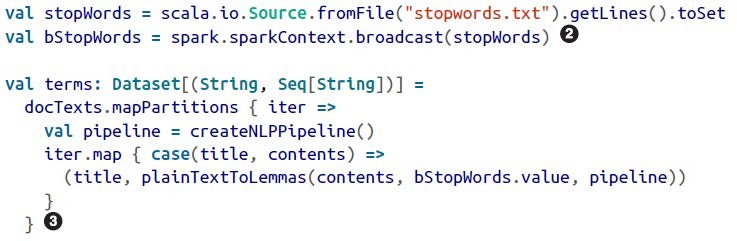
28

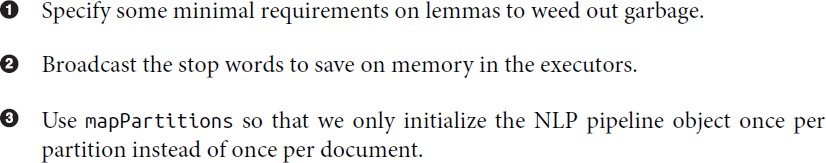
**Preprocessing of Wikipedia dump**



29

**Preprocessing of Wikipedia dump**

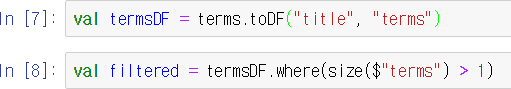




30

**Computing TF-IDF to Make Document-Term Matrix**

### Convert our data set into a data frame and filter out all documents that have zero or one term

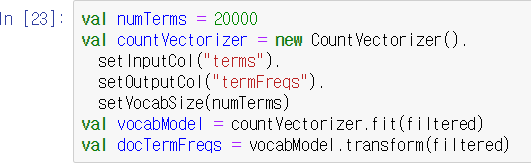


31

**Computing TF-IDF to Make Document-Term Matrix**

### Computing TF-IDF

* + CountVectorizer is an estimator that can help compute the term frequencies
  + CountVectorizer scans the data to build up a vocabulary, a mapping of integers to terms, encapsulated in the CountVectorizerModel, a Transformer
  + The CountVectorizerModel can then be used to generate a term frequency Vector for each document
  + The vector has a component for each term in the vocabulary, and the value for each component is the number of times the term appears in the document



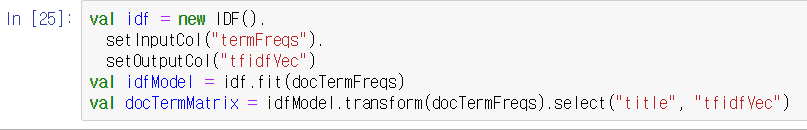
32

**Computing TF-IDF to Make Document-Term Matrix**

### Caching the resulting DataFrame



* With the document frequencies in hand, we can compute the inverse document frequencies
  + For this, we use IDF, another Estimator, which counts the number of documents in which each term in the corpus appears and then uses these counts to compute the IDF scaling factor
  + The IDFModel that it yields can then apply these scaling factors to each term in each vector in the data set

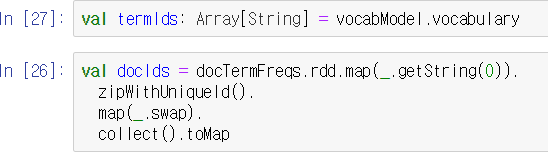


33

**Computing TF-IDF to Make Document-Term Matrix**

### During TF-IDF computation, we lose the ability to key by strings

* Thus, if we want to trace what we learn back to recognizable entities, it’s important for us to save a mapping of positions in the matrix to the terms and document titles in our original corpus

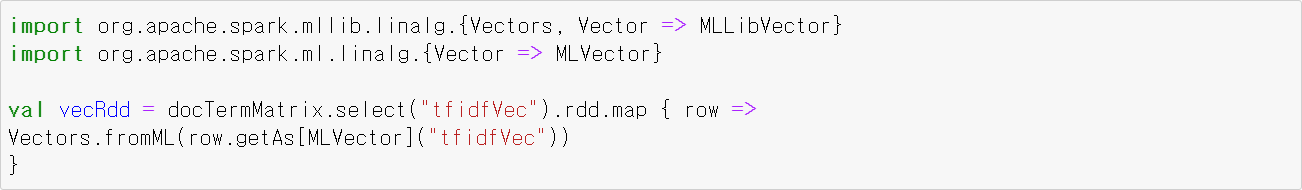


34

**Singular Value Decomposition (SVD)**

### spark.ml (which operates on DataFrames) does not include an implementation of SVD.

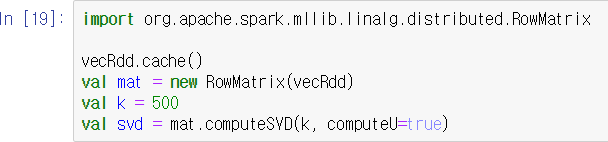
* However, the older spark.mllib (which operates on RDD) does
* So, convert our document-term matrix from DataFrame to RDD



35

**Singular Value Decomposition (SVD)**

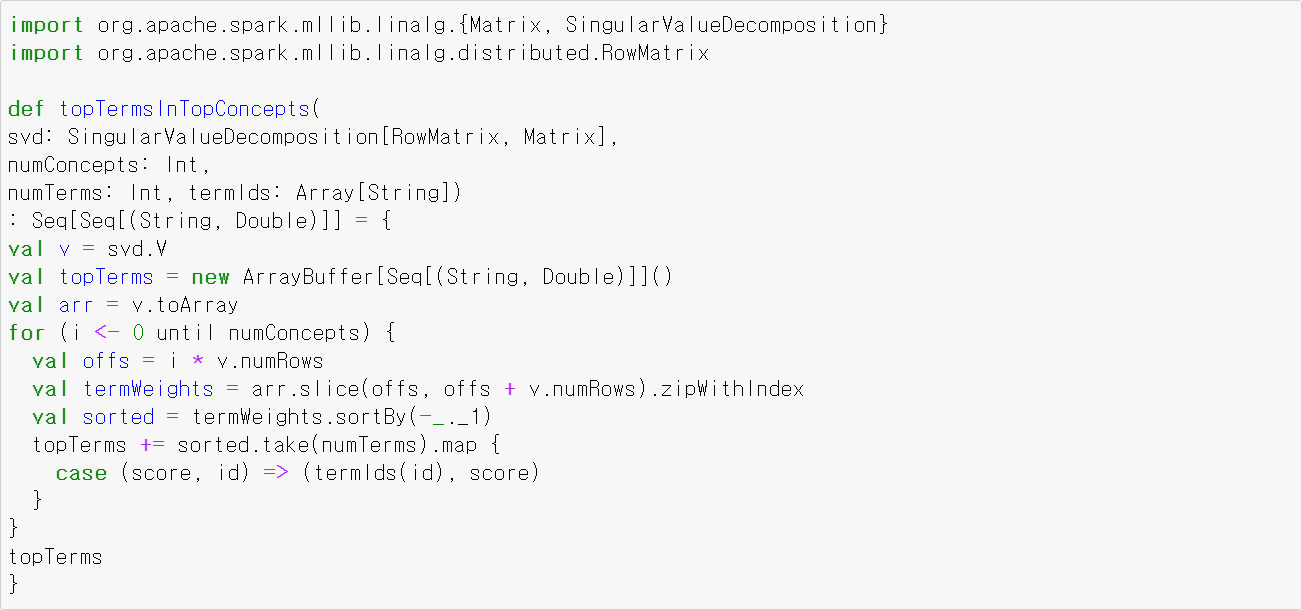
### To find the SVD, we simply warp an RDD of row vectors in a RowMatrix and call computeSVD



36

**Finding Important Concepts**

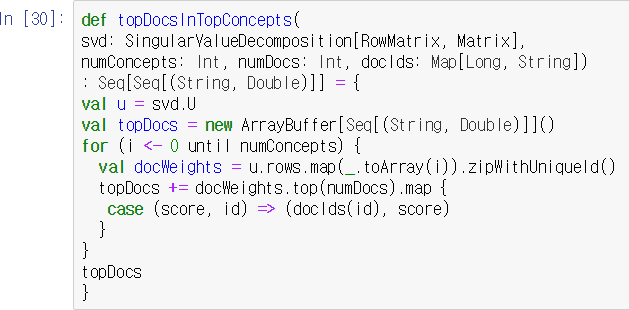
### Find the actual terms that correspond to the positions in the term vectors



37

**Finding Important Concepts**

### Find the document relevant to each of the top concepts



38

**Finding Important Concepts**

### Let’s inspect the first few concepts

39