Universität Freiburg
Institut für Informatik
Dr. Fang Wei-Kleiner
Victor Anthony Arrascue Ayala

Georges-Köhler Allee, Geb. 51
D-79110 Freiburg
fwei@informatik.uni-freiburg.de
arrascue@informatik.uni-freiburg.de

Lab Course: Distributed Computing Using Spark
Summer Term 2021

3. Experiment: Text processing with Spark

Deadline: 30.06.2021

Submission Guidelines:
For this experiment you need to submit a Jupyter Notebook (.ipynb).
Please follow the conventions for this kind of file(s).
You will find them in ILIAS (*Submission_Conventions.pdf*).
Remember to use comments to explain your solutions.

Prolog. The goal of this experiment is to explore and learn text processing techniques provided by spark and apply it on the citeulike dataset[1]. Dealing with textual data (processing, presenting and modeling) belongs to the field of Natural Language Processing (NLP). In this experiment, you will learn how to extract useful features from textual content and how to employ them in machine learning algorithms. A simple and standard approach for text-feature extraction is known as the bag-of-words representation. The bag-of-words approach treats a piece of text content as a set of the *terms* that appear in the text. This set is the result of a process composed of a sequence of operations. Here are three common and widely used operations: Tokenization: splitting the text into a set of tokens (generally words, numbers, and so on); Stop word removal: removing very common words that don't hold meaningful information such as: *the*, *or* and *and*; and Stemming: reduces the words to their stem or root. For example: {cars, car's} → car; {am, are, is} → be.
You might refer to the following sources for this experiment:

- The book: Machine Learning with Spark.
- The DAQL lecture SS2018
- The Spark's documentation for the Spark ML library:
  `https://spark.apache.org/docs/latest/ml-guide.html`

[1]In this experiment we will use the same dataset as in Experiment 1 (*citeulike_dbis_experiment_1.zip*)

Exercise 3. 1 (Vector representation for the papers, 10)
Write a python program that uses spark to generate the bag-of-words representation for each paper as following:

- First, extract the set of "important" terms out of all textual content available in the dataset. Let's call this set $\mathcal{T}$. We will limit the papers representations to only those terms that appear in $\mathcal{T}$. Follow the following steps to find $\mathcal{T}$:
  1- Consider the the text formed by concatenating the title and the abstract of each paper as the input.
  2- Apply tokenization: we want to keep only words that contain English letters in addition to the two characters: (-) and (_). Split on nonword character(s): all characters except the characters we want to keep.
  3- Remove (-) and (_) from the words, this allows equal treatment of the following two words as an example: meta-data and metadata.
  4- Remove words with less than 3 letters
  5- Remove stop words: note that Spark ML has a Stopword remover.
  6- Apply stemming: stemming is provided by some external libraries, for example: NLTK.
  7- Words that appear in a lot of papers don't hold significant information. On the other hand, words that appear rarely are not useful. Therefore, remove words appear in more than 10% of the papers. Afterwards, keep only the words that appear in at least 20 papers. To do this, you need to sort the words descendingly by the number of papers in which they appear.
  8- The output of the last step is a relatively large list. We will reduce it so that you can run the following exercises on your machines. Therefore, keep only top 1000 words. This will be the set of important words $\mathcal{T}$ and we will call the members of this set from now on "terms".
- Second, implement the bag-of-word representation for each paper:
  1- Let's first replace each term $t \in \mathcal{T}$ with a unique index i_t that takes a value from the range between 0 and $|\mathcal{T}| - 1$. This allows dealing with terms as integer values instead of strings.
  2- We want to model each paper p as a term vector. It is a vector of size $|\mathcal{T}|$, where the entry at index i_t is the the number of times the term t appears in p. As a single paper contains only a small subset of $\mathcal{T}$, this vector is very sparse. Therefore, you can use a sparse representation for this vector. Check the Spark SparseVector.
  3- The final output should be a dataframe with the structure: paper_id, TermFrequencyVector

Exercise 3. 2 (TF-IDF representation for the papers, 10)
In this exercise, you will model each paper using the Term Frequency- Inverse Document Frequency (TF-IDF) model. Starting from the dataframe you calculated in the last exercise, transform each TF vector into a TF-IDF vector by calculating the TF-IDF scores.
The TF-IDF score for the term t in paper p is calculated as the multiplication of the Term Frequency (TF) by the Inverse Document Frequency (IDF):

$$\text{TF-IDF}(t, p) = TF(t, p) * IDF(t)$$

$TF(t, p)$ is the term frequency of t in p.
$IDF(t)$ is the Inverse Document Frequency computed as follows:

$$IDF(t) = log\frac{m+1}{DF(t)+1}$$

Where m is the number of all papers. $DF(t)$ is the number of papers which contain t.

- Use the *pyspark.ml.feature.IDF* estimator to calculate the TF-IDF vectors.

This allows to model each paper as a vector defined over the vocabulary (the set of terms of all documents), whose values are the TF-IDF scores of the corresponding terms with respect to that paper. Let this vector be $\vec{p}_{tf-idf}$.

Exercise 3. 3 (Latent Direchlet Allocation (LDA), 10 points)
Latent Direchlet Allocation (LDA) is an algorithm that enables extracting a set of topics that describe the content of a set of documents, and representing the documents using the extracted topics. The number of extracted topics k is usually much smaller than the number of terms: $k \ll |\mathcal{T}|$. Therefore, LDA helps in achieving a compact representation of the documents by representing them in terms of the LDA topics vectors (with size k) instead of the TF-IDF vectors (with size $|\mathcal{T}|$).
The resulting topics are vectors of probability distributions over the set of terms $\mathcal{T}$ (topic-terms distribution):

$$\vec{T} =< p(t_1), p(t_2), ..., p(t_{|\mathcal{T}|}) >$$

Where T is a topic, $t_i \in \mathcal{T}$ is a term, and $p(t_i)$ is the probability of the term t_i appearing in the topic T. We call these topics "latent topics" because we can't always give them representative names, we just know which terms form a certain topic. For example, a latent topic can be represented in the terms as follows: {RNA: 20%, DNA: 30%, chromosome: 40% and Gene:10%}.
Additionally, LDA finds the paper-topics distribution for each paper, a probability distribution over the set of extracted topics:

$$\vec{p}_{LDA} =< p(T_1), p(T_2), ..., p(T_k)$$

Where $\vec{p}_{LDA}$ is the paper-topic distribution of the paper p and $p(T_i)$ is the probability of the topic T_i appearing in the paper p.
Spark ML, provides an implementation for LDA. First, familiarize your self with the LDA algorithm, understand the expected input and how to fit an LDA model. Learn how to get the paper-topic distribution for a paper given its **TermFrequency** vector. Then, work on the following task:

- Using the **TermFrequency** vectors of all papers, run the LDA algorithm with $k = 40$ latent topics and show the top 5 **terms** for each extracted latent topic.

Exercise 3. 4 (User Profiling, 10)
Having the TF-IDF and LDA representations for textual documents allows us to model the users in the terms space.

a) Create a function that produces a user profile $\vec{u}$ for each user as the summation of the TF-IDF vectors ($\vec{p}_{tf-idf}$) of the papers that appear in the user's library P_u^+:

$$\vec{u}_{tf-idf} = \sum_{\vec{p}_{tf-idf} \in P_u^+} \vec{p}_{tf-idf}$$

b) In the same way create a function for LDA-based profiles $\vec{u}_{LDA}$ for each user as the summation of the paper-topics vectors of the papers from the user's library.

Exercise 3. 5 (Sampling and data preparation, 10 points)
As a preparation for the off-line evaluation task to be carried out in the next experiment sheet, implement a sampler (as a function) which does the following:

a) Randomly selects n users. Let U_s be the set of sampled users.
b) Splits the library of each sampled user $u \in U_s$ into training set (TR_u) and test set (TS_u), 80% and 20%, respectively. Store the training sets of all sampled users into a single dedicated dataframe. Do the same for all test sets.
c) Computes the profiles u_{tf-idf}^{TR} and u_{LDA}^{TR} for those sampled users aggregating her papers which appear **only** in TR_u using your functions created in task 3.4. Store those profiles for later use.

Note: While sampling users you don't have to propagate the sampling to the papers catalog. Paper profiles $\vec{p}_{tf-idf}$ and $\vec{p}_{LDA}$ have to be kept as originally computed in this experiment, i.e. using the whole dataset.