Data Mining

Homework 3

Due: 26/12/2017, 23:59.

Instructions—Read carefully!

You must hand in the homeworks electronically and before the due date and time.
Handing in: You must hand in the homeworks by the due date and time by an email to Mara (`sorella@dis.uniroma1.it`) that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework #`
where `#` is the homework number. In the text you must also mention the your name and student id (matricola). After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 2 days after you submit then contact Mara.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups (but the writeup should be yours, as per the collaboration policy), and to ask Mara (first) or Aris in case of questions.**

Problem 1. In this problem we will study two sampling techniques for estimating the center of a set of points and their 1-means cost.

We are given a set P of n points in $\mathbb{R}^d$. Assume that the centroid of P is the origin, that is, $c(P) := \frac{1}{n}\sum_{x_i \in P} x_i = 0$. The 1-means optimal cost is $\text{Cost} := \sum_{i=1}^{n} \|x_i - c(P)\|_2^2 = \sum_{i=1}^{n} \|x_i\|_2^2$. We would like to investigate properties of two sampling procedures. Both are used by k-means++.

Uniform Sampling We sample a set S of m points uniformly at random, that is, the point x_i is sampled with probability $p_i = \frac{1}{n}$.

D^2 Sampling We sample a set S of m points ta random where the probability of sampling x_i is equal to $p_i = \frac{\|x_i\|_2^2}{\text{Cost}}$.

We will show that uniform sampling is able to estimate the center well, but is likely to fail estimating the 1-means cost. Instead, D^2 sampling is likely to fail to estimate well the center, but is able to estimate the 1-means cost. Consider the following problems.

1. Show that $\hat{E}_p = \frac{1}{m}\sum_{x_i \in S} \|x_i\|_2^2 \cdot \frac{1}{p_i}$ is an unbiased estimator for cost for both sampling distributions. In other words, show that $\mathbf{E}\left[\hat{E}_p\right] = \text{Cost}$. (Hint: Let $S = \{S_1, \ldots, S_m\}$, where $S_i \in \mathbb{R}^d$ is a random variable equal to the poiint $x_i \in P$ was selected.)

2. Let $\mu(S) := \frac{1}{m}\sum_{x_i \in S} x_i$ be the expected mean for uniform sampling. Show that $\mathbf{E}\left[\|\mu(S)\|_2^2\right] = \frac{1}{m} \cdot \frac{\text{Cost}}{n}$.

3. Give an instance where for any set of $m \ll n$ sampled points, $\hat{E}_p$ has high probability to be far off from cost for uniform sampling.

4. Let $\mu(S) := \sum_{x_i \in S} \frac{\frac{1}{p_i}}{\sum_{x_i \in S} \frac{1}{p_i}} x_i$ be the average mean for D^2 sampling. Give an instance, where $\mu(P)$ is very likely to be be far off from the $\mu(S)$.

5. Show that $\hat{E}_p = \text{Cost}$, for D^2 sampling.

Problem 2. Topic modeling using LDA and Spark.

In this exercise we will make use of the Spark machine learning library (spark.mllib) to identify topics from a collection of newgroups' posts. Intuitively, given that a document is about a particular topic, one would expect certain words to appear in the document more or less frequently: 'algorithms', 'compiler', and 'array' will appear more often in documents about computer science, 'democracy', 'politician', and 'policy' in documents about politics, and 'word', 'the', and 'is' may appear equally in both. Furthermore, a document typically concerns multiple topics in different proportions, especially so in cross-disciplinary documents (e.g., 60% about biology, 25% about statistics, and 15% computer science in a bioinformatics article). A topic model captures this intuition in a mathematical framework to examine and discover what the topics might be and what each document's balance of topics is. In particular, we will use a topic modeling technique called Latent Dirichlet Allocation (LDA).

LDA can be thought of as a clustering algorithm as follows:

- Topics and documents both exist in a feature space, where feature vectors are vectors of word counts (bags of words).
- Topics correspond to cluster centers, and documents correspond to examples (rows) in a dataset.
- Rather than estimating a clustering using a traditional distance, LDA uses a function based on a statistical model of how text documents are generated.

To understand how the LDA topic model works, you are encouraged to read this paper.

For this exercise you will need **Apache Spark**. **Databricks** provides a community version of its cloud service, which grants you cluster resources to run your code using Python or Scala notebooks in an integrated and easy way. If you wish to use Databricks, set up your account here, then go to *Clusters* > *Create Cluster*[1]. Then fire up a new Notebook using your preferred language. Alternatively, if you use Java or prefer to use Spark on your own computer, you will need a local installation of Spark, and to run it in local mode. You are free to use any language you prefer, among **Python**, **Scala** and **Java**.
If you decide to use a notebook, export it to HTML and attach it to the solutions.

Dataset and data cleaning Our dataset will be the *20_Newgroup* Usenet articles dataset, that can be found here. Read the description to have a sense of it. Indeed, this is a very small dataset, that we use for the purpose of making everyone able to process it. Nevertheless we are going to use code that enables to perform the same task on a big distributed dataset.

[1]Note that your resource allocation grant will expire after two hours of inactivity. In case this happens, you will have to recreate the cluster manually following the same procedure.

If you are using Databricks, the dataset will be readily available as a Spark parquet file from the Databricks repository.
To load it into a Spark DataFrame, do:

```
dataset = sqlContext.read.parquet("/databricks-datasets/news20.binary/data-001/training")
```

The format is: `(id,topic,text,label)` where we can ignore the label field.
If you are not using Databricks, you will have to download the dataset on your local machine. Note that in this case, since each newsgroup is stored in a subdirectory, and each article as a separate file, to recover the topic field, you will have to make use of the `sc.wholeTextFiles()` command, which will read in the entire directory of text files, and return a key-value pair of (filePath, fileContent), that you can manipulate to obtain id, topic and text fields.
Remember to cache the dataframe (`dataset.cache()`), since we will need it for all the following steps.

Each document text begins with a header containing some email header metadata that we don't need, as we are only interested in the body of the document. We can do a bit of simple data cleaning here by removing the metadata of each document, which reduces the noise in our dataset. To simplify this task, we will be content to simply trim all text before the "*writes:* " text.

To accomplish this step you can write a simple User defined function (UDF). Use the same function to also lowercase the text field and produce a final dataframe `cleaned_data` like top of Figure 1.

Spark ML Pipeline In order to train an LDA model, we will need to set up a Spark ML Pipeline. Basically, a pipeline has the same working principles of a Spark jobs DAG: it is made of a set of stages (*Transformers*), that in our case will be arranged as a simple directed line (Fig. 1). Such stages will be described by a series of Spark `mllib` objects that perform subsequent transformations of the input.
Next we describe these stages:

1. **Tokenization**
   First, we will split each document into tokens (terms). With Spark DataFrames, we can use RegexTokenizer. It will need an input column name (from our original dataframe), and an output column name. The minimum token length should be 4.

2. **Stop words**
   We attempt to remove common and useless words. With DataFrames, we can use StopWordsRemover. A list of words can be specified by the `stopWords` parameter. Default stop words are accessible with
   `StopWordsRemover.loadDefaultStopWords(language)`. It is a good idea to also include single letters, the empty string, and other common words to reduce the noise.

3. **Vector of word counts**
   LDA takes in a vector of term counts as input. We will use CountVectorizer to produce a sparse representations for the documents over the vocabulary. You can use the `vocabSize` parameter, e.g., `vocabSize=2048` to limit the size of the vocabulary to the top 2048 occurring words by corpus frequency, as well as other parameters from the documentation.

4. **Configuring the LDA Estimator**
   Now it comes the most important step, defining the LDA Model estimator. The mllib.LDA object requires specifying a number of parameters.

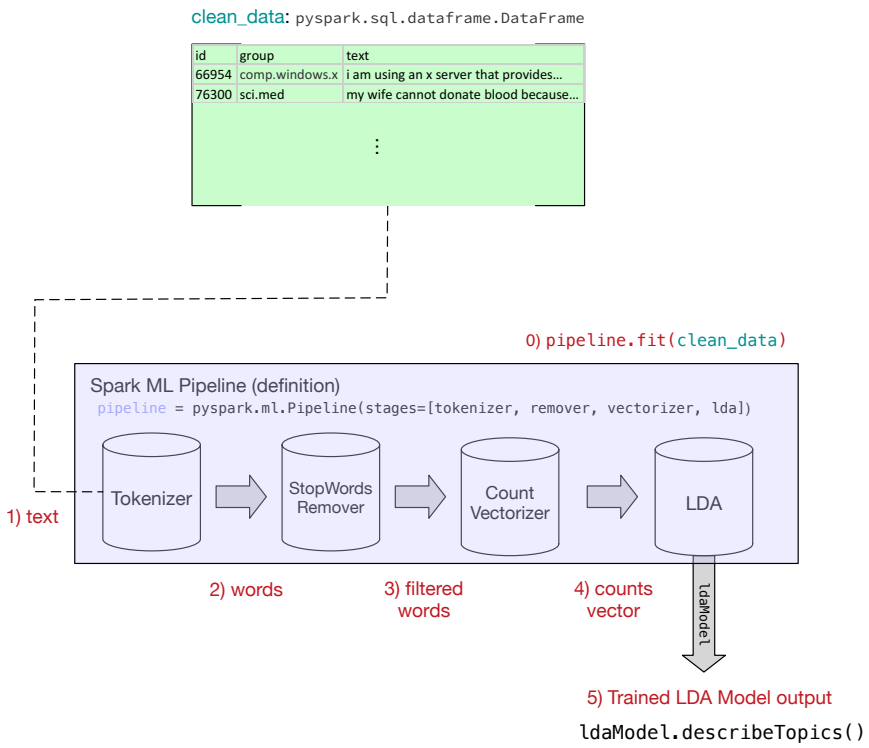

Figure 1: Overview of an LDA pipeline using Spark mllib (pyspark).

The first parameter will be the number of topics k. Choosing k requires a bit of domain knowledge. As we know that there are 20 unique newsgroups in our dataset, we will set `numTopics` to 20. However, some newsgroup topics may overlap, so it not said that this is the most effective choice for k. The second parameter is the inference algorithm. In this case we will use `optimizer="em"` an optimized based on the *expectation-maximization (EM)* technique, that we have seen in class. Last, we have to specify the maximum number of iterations. Start with `maxIter=20` and later see whether using higher values improves your results.

5. **Set up the pipeline**
   As anticipated, we are using the `mllib.Pipeline` abstraction, a Pipeline object. An ordered list with the objects that we have created so far (see Fig. 1) must be passed using the `stages=` parameter. Then we fire up the estimation using `pipeline.fit(clean_data)` that will return a PipelineModel object.

6. **Examine Results**
   From each stage, we can obtain the output: e.g., to extract the learned model, we can use: `ldaModel = pipelineModel.stages[3]`.

   The data log-likelihood gives us a statistic for evaluation. This statistics is always negative, and closer to 0 is better.
   Show its value: `ldaModel.trainingLogLikelihood()`.

   We can also examine the topics we learned.
   Use `found_topics = ldaModel.describeTopics(maxTermsPerTopic = 5)` to get back a dataframe where each topic is represented as a list of token indices, and a corresponding list of token

topicId	term	probability
1	apr	0.025767721936293543
1	post	0.021474524966372692
1	news	0.020732527964708643
2	god	0.05623996643879446
2	jesus	0.021825949973176603
2	believe	0.021227243605627312
2	christian	0.016160326500834567
2	does	0.01592470898171277
3	people	0.035599641239684836

Figure 2: Topical terms probabilities.

weights for each topic.

We now want to map tach token index to text terms. To do this mapping, we need a correspondence of indexes to terms in the corpus `vocabulary`, from the output of stage number 2. **Print out all 20 collections of words with their corresponding term probabilities**. Note that since `found_topics` will contain just 20 rows (one per topic), you can convert it to a Pandas dataframe for easier manipulation. Alternatively, you can keep using Spark SQL Dataframe, and UDF functions.

7. **Visualization**
   Next we will try visualizing our topics with a bubble chart. In this chart, each bubble corresponds to a topical term, bubble color represents a topic, and size the corresponding term probability. To prepare the input for this chart, we need to manipulate our (pandas) dataframe to achieve a representation like the one in Figure 2, where we have only one row per each term, along with the corresponding topic and term probability.

   We are going to use a D3.js library that works on JSON objects, so we need to convert our dataframe to JSON format. if you have a pandas dataframe this can be accomplished using the `.to_json(orient="records")`. The d3.js code needed for the visualization is available here. If you are using Databricks, to visualize the output you can use `displayHTML("""code""")` to embed it into a notebook cell. Notice that the code contains the embedded JSON object (`rawJson`), that is concatenated in the Javascript code. If you called some fields in a different way, than you'll have to modify the script accordingly for it to function properly. The final visualization should look like Fig. 3.

You are encouraged to play around with all the parameters we used[2], and others from the documentation to see if you find more meaningful topics. Especially, try increasing the number of stopwords, also using the feedback you get from the visualization, to try to remove some noise. Note that even if you keep all paramers fixed, you will get slightly different results every time you run the LDA pipeline since obviously the process includes some randomization.

[2]If you are interested to know more about LDA and the em-based inference process you can also read http://obphio.us/pdfs/lda_tutorial.pdf

Figure 3: Bubble chart visualization of inferred topics from the 20 Newgroup dataset.