Polytechnic University of Catalonia

Search and Analysis of Information

Bachelor's Degree in Data Science and Engineering

Mid-term Exam Answers

J.P. Zaldivar

Supervisor
Marta Arias

12th November, 2023

Preliminary Clarifications

- All the answers to the questions of the exam are in the present `pdf` file. In the python notebook, only relevant comments regarding the code.
- The Laboratory sessions we done jointly with Carlos Arbones. The code for computing the *tf-idf* weights was extracted from one of those Laboratory sessions and modified to serve the purpose of the exam.
- The docid's were assumed to start at 0.
- The `altair` package was to plot the graphs and thus the `pandas` package was used to create the adequate Dataframes for `altair`. The `colorama` package was used only for aesthetic purpouses of the notebook.

Question 1 (7 points)

The accompanying file `matrix.txt` contains the document-term frequency matrix of a small fictitious corpus of 105 documents using a vocabulary of 2785 terms. That is, `matrix[i, j]` contains the number of times that the j-th term appears in the i-th document. We can use the indices to this matrix as document and term identifiers. In addition, you can find the actual terms sorted according to their appearance in this matrix in the file `vocab.txt`.

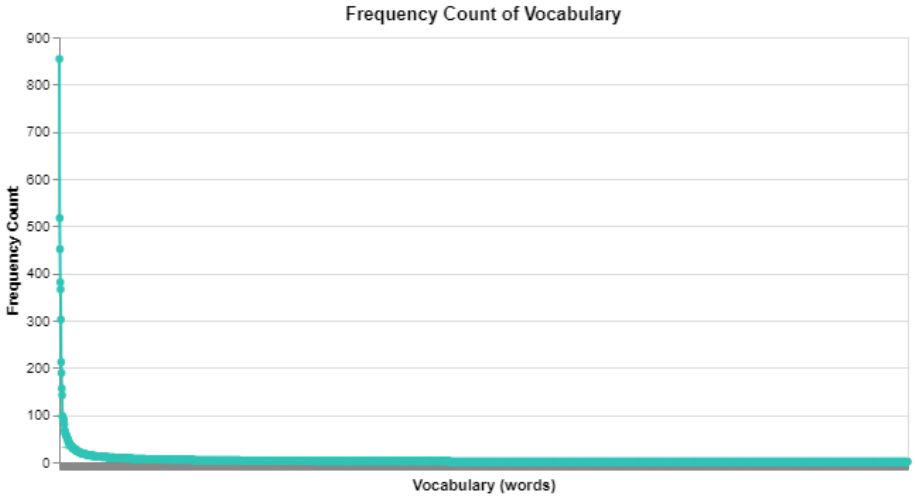
a.

Simple manipulations of this matrix yield interesting statistics. For example, if we sum all the rows of this matrix, we end up with a row-vector containing the frequencies of each term in the whole corpus (i.e., the word frequency counts). As a first step, let us analyze whether the frequency counts in this corpus follow the typical power law $f(r) = cr^{-\alpha}$. In some cases, it is better to use two distinct power law regimes so that:

$$f(r) = \begin{cases} cr^{-\alpha_1} & r \leq r* \\ cr^{-\alpha_2} & r > r* \end{cases}$$

where r^* is the rank at the frontier between the two regimes. In the case of our corpus, do you think this produces a better fit for the frequency counts? It is enough to show this graphically. In any case, show data together with estimated laws and describe the α coefficients that you have found.

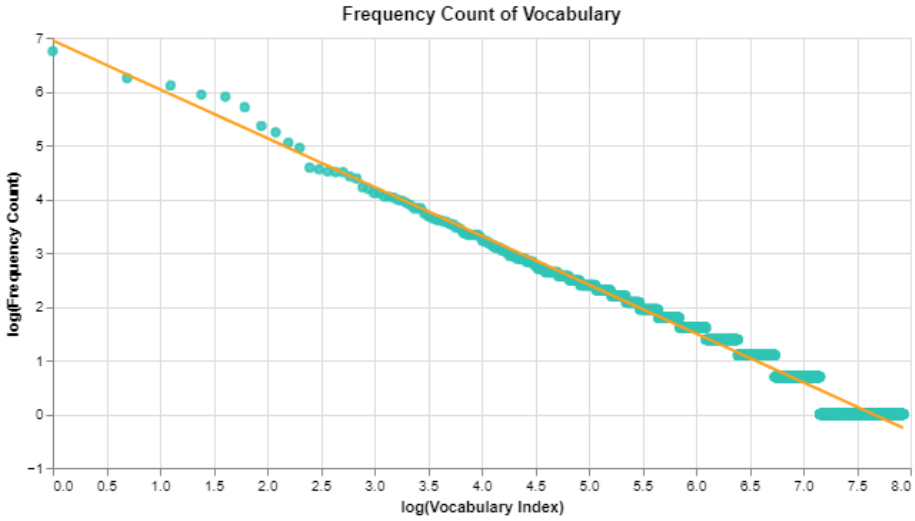
At first glance it appears that the frequency counts within this corpus exhibit characteristics consistent with a power law distribution. This observation is derived from our graphical representation of the data, which is illustrated in the following graph.



The log-log transformation can simplify the relationship, making the power-law equation a linear equation $\log(f(r)) = \log(c) - \alpha \cdot \log(r)$. We need to convert the data into a form where a linear relationship might become apparent if it indeed follows a power-law distribution and thus extract the coefficients.

To compute α and c we use the `np.polyfit` function of `numpy`. The first coefficient of the result is the slope of the linear model, which corresponds to α in the power-law equation. The second coefficient is the intercept, which corresponds to $\log(c)$. The values (rounded up to 4 decimals) are as follow:

$$\alpha = -0.9078, \quad \log(c) = 6.948, \quad c = 1041.0396$$



We can see an appropriate fit of the points. Although there is a distinctive separation of the data points in the upper left zone of the graph as well as in the bottom right, which could indicate

the presence of multiple regimes.

One possible approach would be to adjust the power laws depending on the words frequency. For high frequencies, a power law with exponent α_1 could be defined and α_2 for low frequency-words.

The designed method to find the best value of r* was done by dividing the data into two groups: one group (group 1) for ranks less than or equal to $r*$ and another group (group 2) for ranks greater than $r*$. This would optimally separate the corpus's word frequency distribution into two distinct power law regimes. The selected $r*$ is the position of the rank that minimizes the mean squared error (MSE) between the fitted linear models for each group.

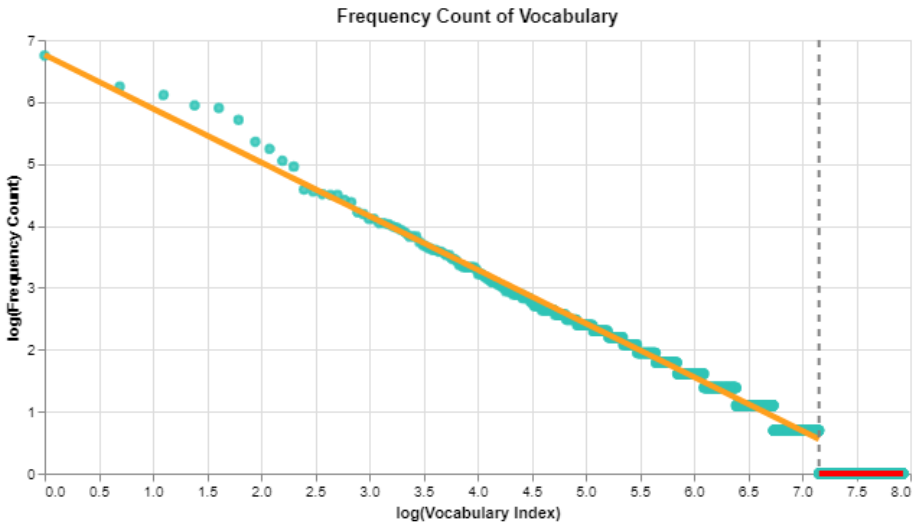
The results suggest that the best separation point $r*$ to divide the word frequency distribution into two distinct power-law regimes is at $r* = 1281$. Which suggests that the corpus can be effectively separated into two distinct regimes at around the middle point of the rank distribution.

$$\alpha_1 = -0.8667, \quad \log(c) = 6.7592, \quad c = 861.9274$$

This indicates a power-law distribution with a relatively steep decay, where the word frequencies decrease quickly as the rank increases for high frequency words.

$$\alpha_2 = 0.0, \quad \log(c) = 0.0, \quad c = 0.0$$

For the second regime (ranks greater than 1281), α_2 is approximately 0.0. This means that there is no power-law behavior in this regime. Instead, it suggests that the word frequencies in this portion of the distribution are likely to be relatively constant distributed.



The result of $r*$ separates the words with low frequencies ($f(r) \approx 1$) that are the infrequent ones, which correspond to approximately half of the words in the corpus. This presence of a substantial number of words with low frequencies suggests that the corpus contains many rare or less frequently used terms.

b.

Describe what basic matrix operations you would perform on our input corpus matrix to compute the following:

1. The lengths of the documents in our corpus.

   To calculate the lengths of the documents (i.e., the number of terms in each document including repetitions), we need to sum the values in each row (i-th row/document) of the matrix. This can be done by computing the row-wise sums. For document i, the length of (d_i) can be calculated as:

   $$\text{len}(d_i) = \sum_{j=0}^{N-1} matrix[i, j]$$

   where N is the total number of terms in the vocabulary.

2. The number of documents in the posting list of the j-th term.

   To find the number of documents containing the j-th term, we need to count the non-zero values in the j-th column of the matrix. This can be done by computing the column-wise count. For term j, the number of documents containing the term $\text{len}(L_j)$ can be obtained from:

   $$\text{len}(L_j) = \sum_{i=0}^{D-1} \mathbf{1}_{[i,j]}, \quad \mathbf{1}_{[i,j]}, = \begin{cases} 1 \text{ if } matrix[i, j] \neq 0 \\ 0 \text{ if otherwise} \end{cases}$$

   where D is the total number of documents in the corpus.

3. The average length of posting lists in an inverted index that keeps only presence/absence of terms.

   In an inverted index that keeps only the docid's that contain each term, we can calculate the average length of posting lists by finding the average number of documents containing each term. The average length ($\bar{L}$) can be calculated as:

   $$\bar{L} = \frac{1}{N} \sum_{j=0}^{N-1} \text{len}(L_j)$$

   where N is the total number of terms in the vocabulary, and $\text{len}(L_j)$ is the length of the posting list for the j-th term from question 2.

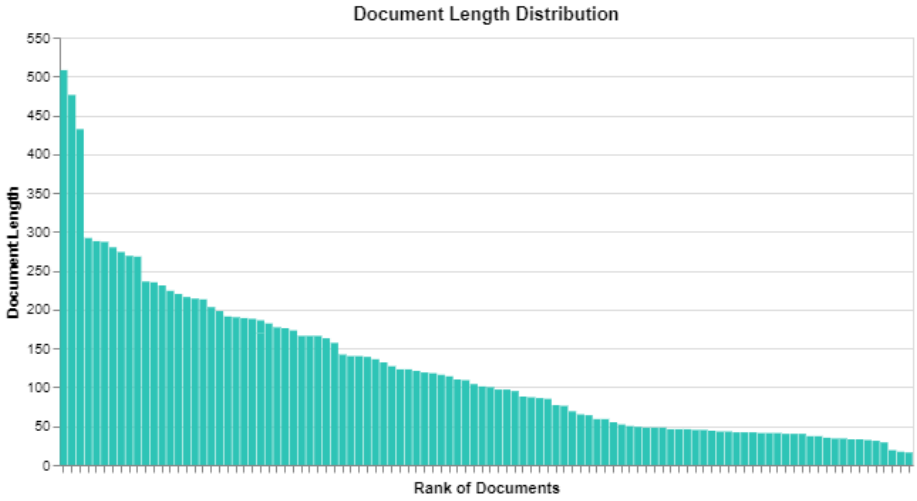
c.

Compute and show in a graphically meaningful way the following distributions:

1. Documents' lengths

   By calculating the length (number of terms including repetitions) of each document in the corpus. We use the given document-term frequency `matrix` and sum the values in each row to find the length of each document.

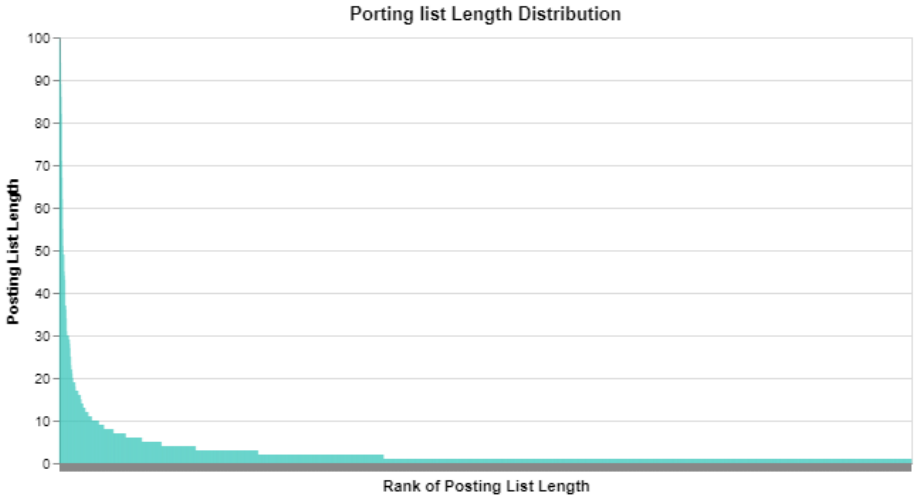
   ```
   doc_len = np.sum(matrix, axis=1)
   ```



2. Posting lists' lengths in the inverted index representing our corpus

   For each term (i.e. column) in the document-term frequency `matrix` we count the non-zero terms to compute the length of the posting list. There is no need to explicitly compute the inverted index given that we only want to compute the posting lists' length.

   $$\text{pst\_list\_len} = \text{np.count\_nonzero(matrix, axis=0)}$$



Do you think any of these follow a power law distribution? Qualitative answers suffice.

The first distribution does not follow a power law distribution from a qualitative point of view. This can be justified as it does not present the Zipf's Law behavior. Multiple documents have large lengths for low ranks and the distribution has rather a linear decrease in the rank of documents.

On the other hand, the distribution of posting list lengths is heavily skewed, with a few elements (posting lists) having very large lengths (low ranks), and the majority of elements having lower

frequencies (higher ranks). This suffices as a qualitative justification that the posting list length distribution behaves similar to the Zipf's Law and thus would follow a power law distribution.

d.

Now suppose that someone performed a search against our corpus with the single query term 'data' and obtained the following result of document IDs (in this order): $4, 91, 60, 3, 46, 88, 7, 36, 16, 77$. How would Rocchio update the original query after one round? Use $\alpha = 0.5$, $\beta = 0.3$, $\gamma = 0.0$, and $k = 3$. Use pseudo-relevance feedback for this, and make sure all involved vectors are normalized. Write the top-10 non-stop words in the new query that you find, together with their corresponding weights.

Pseudo-relevance feedback aims to enhance search query performance by assimilating insights from the highest-ranked search results. It treats the first k top-ranked documents as relevant, without user labeling and uses their content to refine and expand the original query.

The Rocchio relevance feedback algorithm adjusts the query vector based on feedback from relevant and non-relevant documents.

$$q^{'} = \alpha \cdot q + \beta \cdot \frac{1}{|R|} \sum_{d \in R} d - \gamma \cdot \frac{1}{|NR|} \sum_{d \in NR} d$$

The components are as follow:

$$q = [4, 91, 60, 3, 46, 88, 7, 36, 16, 77],$$
$$R = [4, 91, 60], \quad NR = [3, 46, 88, 7, 36, 16, 77]$$

Since γ (that controls the weight of the negative feedback) is 0, the vector NR representing the aggregated terms from the top-ranked non-relevant documents is of no use.

Assuming that the weights for the documents correspond to the tf-idf schema, the resultant query after one round of Rochhino update is:

Word	**Weight**
data	0.5529
web	0.0363
medical	0.0319
topic	0.0294
companies	0.0294
health	0.0278
wiscentd	0.0272
motivated	0.0266
visually	0.0266
entities	0.0266

Table 1: Top 10 Words and Weights after a Rochhio update from original query q

As seen, there are no stop-words in the resultant top-10 results. No filter was done in the code due to limitations of the exam in the available packages. Instead, a visual search of the stop-words was made based on this list of stop-words in this github-web. As seen there was no need to do remove any word from the table since no stop-words were selected in the top-10.

Question 2 (3 points)

Using the corpus from the previous question, a user makes the following (Boolean) query: "analysis AND performance AND new". Based on the worst-case methodology for finding best execution plans discussed in class, we should execute the query intersecting those terms with shortest posting lists first.

Indicate whether in this case the methodology produces the cheapest execution plan. Show the real cost of solving the query in all possible execution plans for this query, assuming that posting lists are implemented as sorted lists and ANDs are computed using the linear merge-like intersection algorithm.

After extracting the posting list for each one of the three elements in the query, these are their respective posting lists' length:

Query word	**Posting list length**	**Posting list**
analysis	11	[1 4 8 24 25 30 38 42 81 82 85]
performance	13	[10 30 33 34 39 42 49 56 71 76 79 80 101]
new	17	[5 6 11 13 14 16 17 34 36 44 49 53 57 76 77 80 94]

Table 2: Posting lists' length for the query words.

The methodology of executing Boolean queries by intersecting terms with the shortest posting lists first is based on the assumption that shorter posting lists should be processed before longer ones to reduce the number of comparisons. The execution plan would state as follows:

(analysis AND performance) AND new

Given that we 'analysis' and 'performance' are the terms with shortest posting lists respectively, we should intersect 'analysis' with 'performance' first, and then the result with 'new.' The resulting worst-case cost would be:

Instruction	**Comparisons**	**Result**
(analysis AND performance)	11 + 13 = **24**	11
(analysis AND performance) AND new	11 + 17 = **28**	11
Total Cost	23 + 28 = **52**	-

Table 3: Worst-case methodology cost

However, it does not guarantee the cheapest execution plan in all cases as seen shortly. Assuming that posting lists are implemented as sorted lists and AND are computed using the linear merge-like intersection algorithm, the real cost would be the number of iterations of the algorithm (i.e. number of comparisons). We have 6 possible permutations of the query terms, which each one of them result in the next costs table.

Query Execution Plan	Real Cost
(analysis AND performance) AND new	32
(analysis AND new) AND performance	27
(performance AND analysis) AND new	32
(performance AND new) AND analysis	37
(new AND analysis) AND performance	27
(new AND performance) AND analysis	37

Table 4: Real Costs for Different Query Execution Plans

We can observe that executing the query 'analysis AND new' first (or 'new AND analysis') results in the lowest real cost, which is 27. Which makes it clear that the methodology of intersecting terms with the shortest posting lists first does not always produce the cheapest execution plan for this specific query.

Repeat the analysis for the query "project AND system AND data".

After extracting the posting list for each one of the three elements in the query, these are their respective posting lists' length:

Query word	Posting list length	Posting list
project	55	[0 1 2 4 5 6 7 8 9 10 11 12 14 16 17 ... 97 98 101 104]
system	23	[2 4 6 10 11 12 13 17 28 34 35 38 ... 71 74 78 94 97]
data	27	[1 2 3 4 7 8 10 ... 71 76 77 83 85 88 91 93]

Table 5: Posting lists' length for the query words.

The execution plan by intersecting terms with the shortest posting lists first would state as follows:

(system AND data) AND project

Given that we 'system' and 'data' are the terms with shortest posting lists respectively, we should intersect 'system' with 'data' first, and then the result with 'project.' The resulting worst-case cost would be:

Instruction	Comparisons	Result
(system AND data)	$23 + 27 = \mathbf{50}$	23
(system AND data) AND project	$23 + 55 = \mathbf{78}$	23
Total Cost	$50 + 78 = \mathbf{128}$	-

Table 6: Worst-case methodology cost

For the real cost cases: Assuming that posting lists are implemented as sorted lists and AND are computed using the linear merge-like intersection algorithm, the real cost would be the number of iterations of the algorithm (i.e. number of comparisons). We have 6 possible permutations of the query terms, which each one of them result in the next costs table.

Query Execution Plan	Real Cost
(project AND system) AND data	95
(project AND data) AND system	93
(system AND project) AND data	95
(system AND data) AND project	79
(data AND project) AND system	93
(data AND system) AND project	79

Table 7: Real Costs for Different Query Execution Plans

One can notice that when the query 'system' AND 'data' is executed first, or 'data' AND 'system,' it yields the lowest real cost of 79. This highlights that the approach of intersecting terms with the shortest posting lists first has, in this instance, led to the most cost-effective execution plan for this particular query.