Big Data Science

—

Descriptive Modeling and Unsupervised Learning of Textual Data

Part One – analytics on long text

Part Two – analytics on **short text** from big data will be assigned separately

Learning outcomes

- Learning the fundamentals of cleaning textual data to prepare it for predictive analytics.
- Converting text documents (unstructured) to a compact document-term matrix (structured).
- Building descriptive models to identify themes, concepts, and topics in textual data.
- Developing a "sliding window"-based algorithm to identify frequently co-occurring phrases and compound words.
- Applying LDA algorithm to learn topics from texts (optional but highly recommended)
- Implementing and applying the k-means clustering algorithm to cluster textual data using cosine similarity as the distance metric.
- Converting document features into a lower dimension using dimensionality reduction techniques.
- Visualizing the results of an unsupervised learning model.

Part 1: Descriptive modeling of textual data

Introduction

What is textual data?

Documents, emails, tweets, blogs, publications, search queries, news, metadata, text messages, online reviews - among others. We live in a cyberspace of documents. Textual data is everywhere around us. As data scientists, it is important to learn how to analyze text documents. Before applying predictive analytics to textual data, one must convert raw, unstructured textual data into *structured* data. Core predictive models take as input a document-term matrix, where every document is represented as a vector in a *n*-dimensional space of *n* terms (or keywords) that are most representative within the corpus of documents.

Converting unstructured text into a structured format remains a difficult data mining task. In text mining, you can analyze a text either though descriptive modeling, predictive modeling, or both. A *descriptive model* extracts the overall patterns, word frequencies, word similarities and derives themes and concepts. In descriptive modeling, you tend to find the best representation of the document in question, with the optimal set of words that best represents the target document in the collection of documents (the corpus). In *predictive modeling* of textual data on the other hand, we try to make a prediction about or classify the documents, for example in sentiment analysis.

Descriptive modeling provides the **foundation** for performing predictive modeling. In this homework, you will develop a descriptive model in Java that takes a dataset of textual data and converts it into structured data. In addition, the descriptive model that you will develop will derive the underlying ideas and themes of each document in the corpus. You will then cluster your document representations and visualize your results.

Dataset

The dataset consists of three folders, each containing different documents. The documents are real news articles, all in English and of varying lengths.

Each document folder contains documents from a different topic. The goal of this first part of the assignment will be to auto-generate keywords, key phrases or topics, for each document folder. This task is also known as features extraction.

Tasks

Preprocessing

The first step in this assignment is to preprocess each document in the dataset.
You are free to use any **Java** library you find suitable for this task and you have to write an object-oriented programming-based code in Java.

We recommend the use of the Stanford CoreNLP library. Or the simple API that might be sufficient to perform some of the tasks for the purpose of this assignment. In particular, your code should perform the following steps during preprocessing:

Again, you must use the object-oriented programming paradigm with Java, do not have all the method in one class. Make sure you structure your code following these tasks:

1. **Filter and remove stop words**. Stop words are words such as "the", "of", "and", etc. and usually do not contain any meaningful information for identifying document topics or similarities. The CoreNLP library contains a good list of stop words, but there are many others available online that you can use. Google is known to have an up to date stop words and it is available on the web , you just need to search for it, you may have to use Google's stop words for better results.
2. **Apply tokenization, stemming and/OR lemmatization.** Tokens are the words taken from a block of text once it has been split into its individual words (called tokens). It is common to also remove punctuation at the same time. Lemmatization refers to regularizing the resulting list of words based on their part-of-speech (POS) tags. For more information on stemming and lemmatization see: reference

Tasks 3, 4 should be used to generate the terms (concepts) which are the topics of the corpus.

3. **Apply named-entity extraction (NER).** NER aims to overcome a common problem in separating words by only using whitespace characters between the words. For example, "the Microsoft Corporation" has three tokens. "The" is a stop word and should be removed, and "Microsoft Corporation" should really be treated as one token. By using NER we can identify a set or a group of words that have a single meaning and combine them to a single token (for example by merging all the tokens with an underscore). This technique most commonly applies to the names of people or organizations.

4. **Use a sliding window approach to merge phrases that belong together.** While NER usually relies on built-in word lists or capitalization of entity tokens, there are other words that consist of one or more-word forms (called compounds). For example, "computer science", "beauty pageant", or "student athlete compensation" are all phrases that frequently occur together and have a single meaning. Just like for the NER, you should identify these **n-grams** and merge them into a single token (key phrase) according to your judgement on the ones that should be grouped together. To do this, you might want to iterate over the entire dataset at least once to collect the word frequencies for all the possible n-grams, and merge the ones that co-occur above a certain minimum frequency threshold. You should experiment with the size of your n-grams and your minimum frequency counts to see which gives you the best results, but 2-grams and 3-grams are likely to be most common.

   ----- at the end of this stage you will need to generate the topics (the terms, 1-gram, 2-grams or 3-grams …) that **best** represent the whole corpus. The list of topics should be displayed to the console and also written to a text file. The topics should be sorted by an order of importance that you need to define (e.g. frequency, or frequency percentage: Occurrences/total numbers of words in the corpus)

5. (Optional +20pts) **Generating topics using LDA**

LDA (Latent-Dirichlet-Allocation) is a topic modeling algorithm that extracts a given number of topics from an input set of raw documents. The algorithm models each document as combination of topics that exist in the corpus. In other words, the algorithm, assumes that the topics to be discovered in the whole corpus is present to some degree within each document. In this section you will take have to apply LDA to extract the topics and compare the topics you got with LDA with the topics extracted using the previous methods in (3 and 4). You are allowed to use Java libraries that takes the corpus (documents) and generates topics: (you might have to clean the documents before using LDA depending of which library's implementation you are using)

http://mallet.cs.umass.edu/

or

http://jgibblda.sourceforge.net/

or any other Java implementation of LDA.

or

you can find use any other library for LDA.

Generating a document-term matrix

After preprocessing, you should construct your **document-term matrix.** Each row in your matrix should correspond to one document in the input dataset. Each column should represent one term or key-phrase or key-word (concept) of your final set of terms or keyphrases across all documents (after tokenization, lemmatization, and merging NER, n-gram tokens, and LDA). Each cell should contain the number of times that each term occurred in each document. This will result in a relatively sparse vector representation of each document, since only a few of the complete list of terms will occur in each document and most of the values in the matrix will be zero.

Once your matrix is constructed, you should transform it using term frequency–inverse document frequency (TF-IDF). TF-IDF is a very useful measure in text mining that helps with down-weighting terms that are frequent across all documents while promoting terms that occur frequently in the current document, but are generally rare. A short introduction, implementation details, and additional references can be found here: http://www.tfidf.com/ **Please note that other than for the preprocessing step, you should not be using any additional Java libraries.**

Part 2: Clustering textual data

Now that you converted unstructured collections of documents to a document-term matrix, you will develop a clustering algorithm that should group similar documents vectors together (hoping to automatically generate three folders). Of course, in a real-world setting, you wouldn't usually know which documents belonged to a similar domain in advance, and it would be the job of your clustering algorithm to model the topics correctly. In this assignment, we are giving you the document folders in order to be able to evaluate the performance of your TF-IDF vector representations and the implementation of your clustering algorithm.

Tasks

K-means clustering and document similarity

You should implement your own version of the popular k-means clustering algorithm that takes as input the TF-IDF matrix you generated in part 1, and an integer *k* that specifies the number of output clusters. Since there are 3 document folders in the dataset, setting *k=3* is likely to result in best results. In addition, your algorithm should accept a similarity measure. By default k-means clustering utilizes the Euclidean distance between two data points. However, for textual data it often makes sense to use the cosine similarity between document vectors instead. Your implementation should support both, and you should compare your results for both measures.

Why cosine similarity over Euclidean distance?

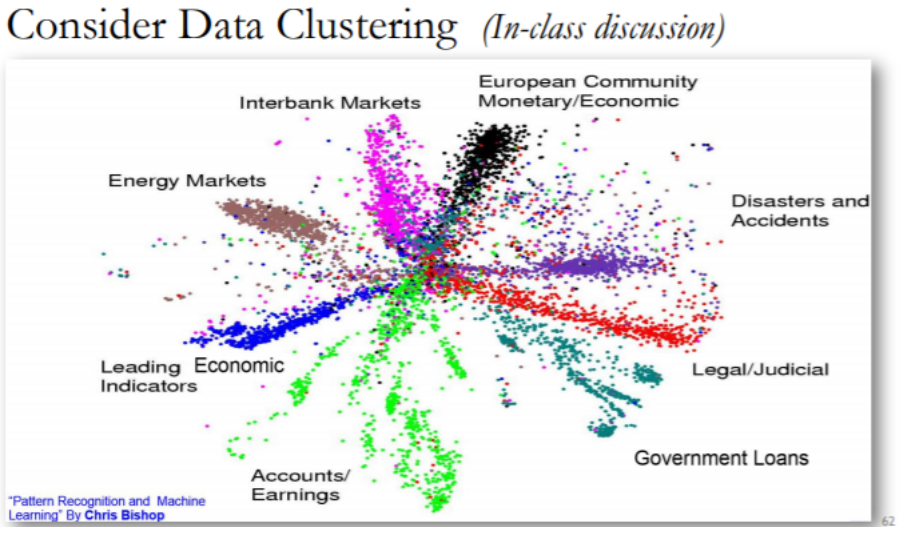
The major difference between Euclidean distance and cosine similarity is that the former is purely a distance measure, whereas the latter measures the angle between vectors without taking their magnitude into account. In other words, by using cosine similarity instead of Euclidean distance we are able to remove the effect of mere word count/frequency, which is usually desirable when dealing with documents of different lengths (since two documents of unequal length might still be about the same topic and thus semantically similar). Mathematically, computing the cosine similarity is equivalent to computing the Euclidean distance between normalized unit vectors. Please note that you are not allowed to use any existing Java libraries for this step.

Visualization and model performance

Most classification algorithms are evaluated by generating a confusion matrix, where each row represents the instances of a *predicted* class, and each column represents the instances of an *actual* class. Hint: Depending on how well your clustering algorithm works, you could try to guess the actual class label for each cluster based on the majority of predicted class labels.

You should also use Precision and Recall to measure the F-measure performance of your implementation.

In addition, you should *visualize* your output clusters, as well as the original documents clusters given in the dataset. Give a different color to each cluster and assess how well your clustering algorithm works from there. **Based on the keywords extracted on part one, try to get guess the main domain (topic) of each folder, mention those as labels on the documents (similar to the in-class discussion we had on data clustering of text data).** See the image below for reference.



Since your TF-IDF matrix contains high-dimensional vector representations, you will need to use a dimensionality reduction technique from chapter two (such as principal component analysis (PCA) or singular-value decomposition (SVD)) to convert the data into a 2-dimensional space for plotting.

For plotting the clusters you should use a Java library that allows to do the plotting, there are many (e.g Jfreechart…)

Submission

Deliverables

Your submission should consist of the following:

- All source code of your program, in Java and Object Oriented Programming. You must use Eclipse IDE. and you must export your project as a Java Eclipse project (use the export option)

Your code should be structured using the Object-Oriented Programing paradigm. One possible. design could be as follows.

- You can have the names of the files in text file ("data.txt") each name in one line or the name of the folders where the text files reside (paths or just the names of the files since you will have them in the same folder as your project).
- You can use file I/O java libraries to read from the "data.txt" file.
- Then you can have a Java class that will perform the pre-processing of the textual data.
- Then a class that will take care of generating the term-document matrix and apply the TF-IDF transformation.
- Next, a class that implements the similarity methods.
- Then another class that will be responsible for data clustering.
- A separate class for LDA
- Another class for visualization.
- Finally, a class responsible for evaluating your clustering results and printing the confusion matrix.

- Any dependency files that your Java program relies on.
- A detailed README file describing how to execute your program from the command line. Any assignments that we are unable to run it on Eclipse IDE will be returned ungraded.
- A plain text file "topics.txt" that is automatically generated and that contains the topics you extracted for each of the three document folders in part 1. You should also display the topics to the console.
- The plots you generated in part 2, both for the original dataset and the clusters you generated.
- The confusion matrix for your output clusters, as well as precision/recall/F measure score.

- **In addition to the LDA (+20pts) here another bonus: implementing the k-means++ algorithm (5pts).** As you know, the choice of the initial (random) centroids will have a major effect on the performance of k-means. Research and implement an algorithm that will enhance your k-means algorithm, e.g. k-means++.

Grading

Deliverables	Points
Preprocessing of documents	10
Generating the document-term matrix	10
TF-IDF transformation	15
Generating topics	10
Implementing k-means clustering	20
Implementing Euclidean distance	5
Implementing cosine similarity	5
Visualizing the clusters	10
Generating the confusion matrix and computing precision/recall/F1-score	15
(optional) Implementing k-means++ (optional) LDA	5 20
Total	**100 (+25 optional)**