Disclaimer:

This document and the attached dataset are for the student use only. The student is **NOT** allowed to post this document or the dataset in websites like coursehero.com, studynotes.com or any similar website. Posting the homework document or the dataset on those sites will result into a legal copyright violation.

Big Data Science

—

Descriptive Modeling and Clustering of Textual Data

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.
- 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, ads, scholarly 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. 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 or topic modeling.

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 document folders, each containing eight different documents. The documents are relevant global news articles, all in English and of varying lengths.

Each document folder contains documents from a different domain. The goal of this first part of the assignment will be to auto-generate keywords, 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 (or you can write your own code), we recommend the use of the Stanford CoreNLP library. The simple API 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: You must use the object oriented programming paradigm with Java. Make sure you structure your code following the following tasks.

1. **Filter 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.

2. **Apply tokenization, stemming and 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

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 remaining 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 NER, you should identify these **n-grams** and merge them into a single token. 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.

Generate 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/key-phrase/key word (concept) of your final set of terms across all documents (after tokenization, lemmatization, and merging NER and n-gram tokens). Each cell should contain the number of times that each term occurred in the 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 downweighting 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/

Finally, you should use your transformed matrix to generate keywords, or topics, for each document folder, for example by combining all the document vectors together and then sorting the terms for each folder based on their TF-IDF scores.

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 back into their original folder structure. 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.

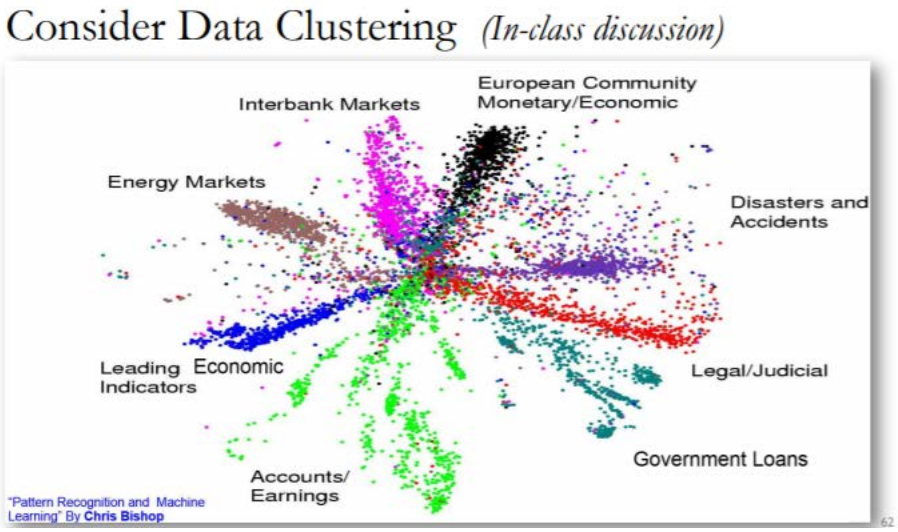
In addition, your algorithm should accept a similarity measure. By default k-means clustering utilizes the Euclidean distance between two data points. However, for textual data it often makes sense to use the cosine similarity between document vectors instead. Your implementation should support both, 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. You should also use Precision and Recall to measure the F-measure performance of your implementation.
(standards by NIST.gov)

Since there are no class labels in clustering, you should instead *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 of each folder, mentions those as labels on the documents (similar to the in-class discussion we had on data clustering of text data – see slide bellow (where dots of different colors represent clusters of documents)**



Consider Data Clustering *(In-class discussion)*

"Pattern Recognition and Machine Learning" By **Chris Bishop**

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 are free to use any programming language, software or tool you like. You are also allowed to use a Java library to apply the dimensionality reduction.

Submission

Deliverables

Your submission should consist of the following:

- All source code of your program, in Java.
- 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 on a standard Unix terminal will be returned ungraded.
- A plain text file "topics.txt" that contains the topics you extracted for each of the three document folders in part 1.

- The plots you generated in part 2, both for the original dataset and the clusters you generated.

Bonus points

You are welcome to implement the following for extra credit:

- **Generate a confusion matrix for your output clusters**. 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.
- **Implement the k-means++ algorithm.** 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	20
Generating topics per folder	10
Implementing k-means clustering	20
Implementing Euclidean distance	5
Implementing cosine similarity	5
Visualizing the clusters	20
(optional) Generating the confusion matrix	10
(optional) Implementing k-means++	10
(total)	**100 (+20 optional)**