Data Science Lab
Lab #2

Politecnico di Torino
October 16-17, 2019

Introduction

The purpose of this laboratory is to make you practice with the data preparation process. More specifically, you will tackle the task with tabular and textual input data, learning how to handle anomalies in the data, missing values and more.

1 Preliminary steps

1.1 Matplotlib availability

In order to complete some of the optional exercises you will need the `matplotlib` plotting library. You can read more about it on its official documentation. You can check if it is already installed by running `import matplotlib` either in Jupyter or iPython.

Among its numerous functionalities, you will use the histogram plotting function in this laboratories. An histogram is a simple representation of the distribution of numerical data. It presents the binned range of values on x-axis (i.e. a series of intervals in which data is divided) and the frequency of each bin on the y-axis. You can read more about it on Wikipedia.

Here it is a short python script on how to use `matplotlib` to plot the histogram of values contained in a list. Figure 1 shows an example of the resulting chart.

```
from random import gauss
import matplotlib.pyplot as plt

l = [gauss(0, 1) for _ in range(500)]
plt.hist(l)
plt.title('Gaussian distribution (mu=0, sigma=1)')
plt.show()
```

Note that, if you are using Jupyter Notebook, you should add the following line, after importing matplotlib:

```
%matplotlib inline
```

1.2 Datasets download

For this lab, two different datasets will be used. Here, you will learn more about them and how to retrieve them.

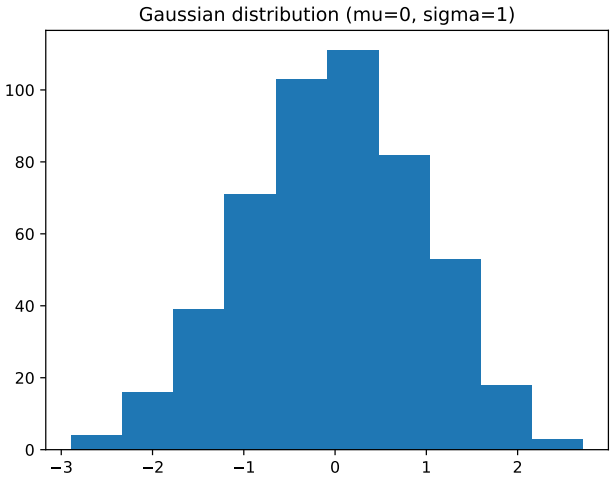

Figure 1: A gaussian distribution plotted with Matplotlib

Global Land Temperature

The Global Land Temperature (GLT) dataset is a large collection of measurements actively maintained by Berkeley Earth. It contains the raw source data measured with stations all around the globe, plus an intermediate format and several formatted output files. Data span from ∼1750 up to recent days with monthly and daily availability. Measurements are provided by hemispheres, states, countries, cities and more. You can read more about the dataset at the Berkeley Earth website.

For the purpose of this laboratory you will work on a modified, smaller but dirtier, version of the original GLT dataset, to stress the importance of data preprocessing. More specifically, this didactic version contains the formatted output files of the major cities of the globe with monthly granularity. For the sake of simplicity, the analysis will range between almost two centuries (i.e. between the years 1817 and 2012).

The dataset is composed of ∼200k rows corresponding to the measurements taken the first day of the month in a given city. Each measurement is then described by 7 values:

- Date, when the measurement was taken
- AverageTemperature
- AverageTemperatureUncertainty
- City, from which the measurement was taken
- Country
- Latitude
- Longitude

The dataset is available in CSV format. You can find it at the following URL

`https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/GLT_filtered.csv`

Info: even though laboratories do not have a strict order, we consider what you have accomplished during the previous ones to be part of your experience. At this point you should be able to download the file and parse it with the `csv` python module.

IMDb reviews

Internet Movie Database (IMDb) is a popular online platform that gathers many information related to multimed like movies, tv shows, video games and many more. You can navigate to its website to explore the huge amount of updated content it offers. The only difference with the previously used CSV datasets is that, in this one, the first row of the file contains a "header" (i.e. the name of each column). You should skip the first row and start reading from the second one on.

IMDb has soon become a rich data source for the scientific community. Among the others, people's reviews are one of the most important types of data that can be retrieved from the platform. The collection of reviews on a specific movie, for example, contains intrinsic information about its approval rating. Therefore, a considerable amount of recent science works has addressed the identification of the sentiment within textual reviews and surveys. In the context of movie reviews, the sentiment analysis would seek to discover if the reviewer liked the movie or not based on the content of the text, with the sentiment being represented as numerical value (e.g. a score between 1 and 10) or a binary one (e.g *Positive* or *Negative*). Wikipedia provides a general overview on the topic of sentiment analysis.

During this laboratory you will work on the dataset collected and used by Maas et al. 2011. We will focus on a smaller portion of the whole dataset, known as the training data (you will learn more on training and test datasets). It contains $25,000$ user reviews collected from IMDb for different movies. Since each review has also a numerical score between 1 and 10, the authors considered the ones with a score lower than 4 to have a negative sentiment, while the ones with a score higher than 6 to have a positive sentiment. The dataset includes $12,500$ positive and $12,500$ negative reviews. For the purpose of this laboratory, reviews, which originally came in different files, have been organized in a single CSV file. Each line of the file refers to a single review and has two fields:

1. the textual comment;

   Info: you do not have to handle commas inside the text. Each comment comes surrounded by the ¨ character which helps the `csv` module to properly split the fields for you.

2. a binary value indicating either the positive or negative sentiment, represented respectively by a '1' and a '0'.

Here it is a pseudo-representation of the first lines of your dataset:

```
review,label
<review_text>,1
<review_text>,1
<review_text>,0
```

Warning: the first line of the file contains the header, which describes the schema of your dataset. It is commonplace to store data in this format.

You can download the dataset from the following link

`https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/aclimdb_reviews_train.txt`

2 Exercises

Note that exercises marked with a (*) are optional, you should focus on completing the other ones first.

2.1 Data preprocessing on Global Land Temperature

The main goal of this exercise is to learn how to clean a real-world dataset searching for anomalies, such as missing values or outliers, in its data.

Prerequisites. There are many ways to handle missing values. One can decide to delete a row of the dataset based on whether a missing value is present or not. This strategy can be adopted when the dataset is large and the information loss does not affect the overall distribution. Another common solution is to fill every missing value. If data has not a specific order, they can be replaced with the mean (or the median) of the involved attribute. Temporal data, instead, allow to replace missing values with values of adjacent rows, e.g. by averaging them. Clearly, this technique is possible if the data type allows to compute the mean.

1. Load the Global Land Temperature dataset as a list of lists. Before starting, take a moment to better inspect the attributes you are going to work on. How many of them are nominal, how many continuous or discrete?

2. Analyze the attribute `AverageTemperature`, which contains missing values. Fill any gap with the arithmetic mean among the closest antecedent and the closest successive measurements in time, taken in the same city. Assume the following rules for edge cases:

   (a) it can happen that a missing value does not have a preceding (or successive) measurement. This happens when the missing value is the first (or last) value of the dataset. If this is the case, consider the missing value to be preceded (or followed) by a 0, then compute the mean accordingly.

   ```
   original_list = [  '', 5, 6, '' ]
   step_1        = [ 2.5, 5, 6, '' ] # (0 + 5) / 2
   step_2        = [ 2.5, 5, 6,  3 ] # (6 + 0) / 2
   ```

   (b) if there are consecutive missing values, just compute them in temporal order and use the newly inserted values to evaluate the following ones. Here it is an example with a simple list where both (a) and (b) rules have been applied:

   ```
   original_list   = [ '', '', 24, 28.9 ]
   step_1          = [ 12, '', 24, 28.9 ] # (0 + 24) / 2
   step_2          = [ 12, 18, 24, 28.9 ] # (12 + 24) / 2
   ```

3. Define a function that, given the name of a city and an integer $N > 0$, prints:

   (a) the top N hottest measurements;

   (b) the top N coldest measurements.

4. (*) Let's search for other anomalies in data distribution with the help of `matplotlib`. Plot the distribution of the average land temperatures for Rome and Bangkok using the aforementioned histogram plotting function.

   Info: calling the `plt.hist()` method twice will draw the second histogram onto the canvas generated by the first call.

   As you can see, Rome and Bangkok have very different temperature distributions, but this seems plausible. What it looks strange is the large difference in their temperatures' magnitude. Is it possible that *all* sensors from Bangkok stations, along the *entire* time-span, were faulty? Could they were configured to use another representation of the temperature measurement? Can you figure out a data cleaning step to solve it?

   Before continuing, try to answer to these questions.

5. (*) One might think that Bangkok sensor provide temperature samples in degrees Fahrenheit while the ones located in Rome use the Celsius notation, which is the common representation in the whole dataset. Write a function to transform Fahrenheit measurements back to Celsius, apply it to your data and plot the two distribution again.

Info: remember that the mapping function from Celsius to Fahrenheit is the following

$$T_F = 1.8 \cdot T_C + 32$$

2.2 Textual data preparation on IMDB reviews

This exercise is meant to get you acquainted with the preprocessing of textual data. You can find useful information about Information Retrieval theory in the book "Introduction to Information Retrieval" (Manning, Raghavan, and Schütze 2008) also available at Introduction to Information Retrieval Stanford web page.

Prerequisites. Textual data (like that from IMDb reviews) comes without a specific structure. Even when the text itself is correctly formatted, with no syntax error, misspelled words, misplaced punctuation or any other possible flaw generated by a human typing on a keyboard (which is, in practice, never the case), many data analytics and machine learning algorithms cannot operate on the bare textual representation. The metrics you have already encountered (e.g. arithmetic mean and standard deviation) do not make sense in this context. Nonetheless, many practical tasks would benefit from mathematical computations on textual documents. In all these cases, the data preprocessing step becomes crucial.

Along this exercise you will focus on one of them. Let's imagine the IMDb platform wants an automatic procedure to assign a numerical value to a user review, based of the content of the text, to identify the rate he or she would have given to the movie. Simplifying the task, you might want to assign a label, Positive or Negative, to identify the overall sentiment, i.e. whether the user liked the movie or not.
To do so, you will implement the *TF-IDF* weighting scheme on IMDb reviews. Then, you will compute the distance between a review whose sentiment is unknown and the two sets of positive and negative comments, assigning the sentiment of the closest one among the two.

1. Load the IMDb dataset as a list of lists.
2. Apply the tokenization function listed below to your reviews. Please refer to the function's docstring [1] for the input and output parameters. The tokenization procedure splits each comment in tokens (i.e. separate words).

```
import string

def tokenize(docs):
    """Compute the tokens for each document.

    Input: a list of strings. Each item is a document to tokenize.
    Output: a list of lists. Each item is a list containing the tokens of the
    relative document.
    """
    tokens = []

    for doc in docs:
        for punct in string.punctuation:
            doc = doc.replace(punct, " ")
        split_doc = [ token.lower() for token in doc.split(" ") if token ]
        tokens.append(split_doc)

    return tokens
```

3. The next step requires the computation of the *term frequency* (TF) of each token within its respective document. Although there exist different techniques to evaluate the frequency, we will now assume that the TF of a token t in a document d is equal to the number of occurrences of t in d. Compute the TF for all your reviews.

[1] Python docstrings are string literals that are not processed by the interpreter. Their main purpose is to describe how a piece of the project works (e.g. a module, a function, a class, etc.) in a concise way. Like many other Python stuff, you can find a further description on the official documentation.

Info: if you decide to count the frequency of every term in the collection of reviews for each document (i.e. not only the terms contained in that document), you will build what is known in literature as *Bag-of-Words*. However, since many terms are not present in all documents, you will end up with many values equal to 0. This representation is a sparse matrix. Storing many 0-valued cells like in sparse matrices is quite inefficient. As a consequence, you should pay attention to the choice of your data structure.

Here it is an example of TF on a toy sentence:

```
tokens = ['with', 'great', 'power', 'comes', 'great', 'responsibility']
TF(tokens) = {'with': 1, 'great': 2, 'power': 1, 'comes': 1, 'responsibility': 1}
```

4. We will now compute the *inverse document frequency* (IDF). While the TF gives an idea of the weight of a token within a document, the IDF is used to find its significance among the entire collection of documents (i.e. your reviews). One possible way of computing it is:

$$IDF_t = \log \frac{N}{DF_t}$$

where N is the number of documents and DF_t is the *document frequency* of a token, i.e. the number of documents in which t appears at least once. As you can see, $IDF_t \in [0, \log N]$. Furthermore, a low value means that the token appears in the majority of the documents, hence its presence is not relevant to characterize any subset of them, whereas an high value indicates relevance for a few documents.

   (a) Compute the DF for all of your tokens;

   (b) Compute the IDF for all of your tokens;

   (c) Try to sort the IDF values in ascending order. Which tokens (i.e. words) came to the top? Can you figure out why?

5. Compute the $TF\text{-}IDF$. Combine the definitions of term frequency (TF) and inverse document frequency (IDF), to produce a composite weight for each term in each document. The $TF\text{-}IDF$ weighting scheme assigns to a term t a weight in the document d given by:

$$TF\text{-}IDF_{t,d} = TF_{t,d} \times IDF_t \tag{1}$$

   In other words, $TF\text{-}IDF_{t,d}$ assigns to term t a weight in document d that is

   - high when t occurs many times within a small number of documents;
   - low when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
   - lowest when the term occurs in virtually all documents.

   For dictionary terms that do not occur in a document, the weight is zero.

   The suggested output structure is a list of dictionaries. Each dictionary represents a document and contains its tokens as key and weights as values:

```
TF_IDF = [{ "token_1": weight_1, ... "token_N": weight_N}, { ... }, ... ]
```

6. (*) Sentiment analysis. Given a document, identify if it belongs to positive or negative comments calculating the similarity between the comments in the two groups.

   How do we quantify the similarity between two documents exploiting $TF\text{-}IDF$ vector representation?

   A first attempt might consider the magnitude of the vector difference between two document vectors. This measure suffers from a drawback: two documents with very similar content can have a

significant vector difference simply because one is much longer than the other. Thus the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger. To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d1 and d2 is to compute the cosine similarity of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$:

$$cos\_sim(d_1, d_2) = \frac{\vec{V}(d_1) * \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} \tag{2}$$

To solve this simple task of sentiment analysis you have to:

(a) Take the vector representation of the first document in the collection. We will call it the *test document*.

(b) Identify the positive and negative comments and separate them into two groups.

(c) Compute the cosine similarity between the test document and all the other positive and negative comments separately.
Given that the vector representation for a document $\vec{V}(d_n)$ is the dictionary of tokens with their weights, use the following functions to compute the cosine similarity:

```
def norm(d):
    """Compute the L2-norm of a vector representation."""
    return sum([ tf_idf**2 for t, tf_idf in d.items() ])**.5

def dot_product(d1, d2):
    """Compute the dot product between two vector representations."""
    word_set = set(list(d1.keys()) + list(d2.keys()))
    return sum([( d1.get(d, 0.0) * d2.get(d, 0.0)) for d in word_set ])

def cosine_similarity(d1, d2):
    """
    Compute the cosine similarity between documents d1 and d2.

    Input: two dictionaries representing the TF-IDF vectors for documents
    d1 and d2.
    Output: the cosine similarity.
    """
    return dot_product(d1, d2) / (norm(d1) * norm(d2))
```

(d) Assign the label (positive or negative) to the selected document analysing the mean similarity with respect to the group of positive comments and the mean similarity with respect to the group of negative comments.

(e) Are you able to identify the correct group? Why?

(f) Repeat the previous steps with other documents. Do you think the $TF\text{-}IDF$ pre-processing is enough to perform a simple task of sentiment analysis?

References

[1] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.