Final Project

Semester August-December 2021

1. Details

- The project will be developed in teams of 3 persons.
- Project is due by Sunday December 12, 2021.
- For each problem, generate a .py file as **pf_01.py**, **pf_02.py** and **pf_03.py**
- Each person uploads to Teams four files. 1-3) the code files. 4) a .TXT file with the name of the team members, sorted in alphabetical order starting with the given name (nombre).
- Each team will present and explain their code personally online (shared screen).
- I will ask questions to each team member about the code implementation and the solution for the problem. Only the person that is being questioned could answer.
- The teams will present in a random order.
- The presentation will be on Monday December 13, 2021, at 10.00h
- A team could present before the deadline if there is a prior agreement with the professor.
- **Do the code implementations by yourselves. Do not copy code from the internet or any other source. The purpose of the project is for you to learn how the methods work and to apply the knowledge from the course.**

2. Description

For the problems, you will use the file with your friends posts we have collected fb_posts.csv (attached here). The file contains three fields: user, gender, publication.

1. Implement the multinomial version of the Naïve Bayes classifier for training and testing.

The posts in the dataset are organized by gender (f or m). You will use the dataset to build a Naïve Bayes classifier that will be able to determine if a new publication (one that is not in the dataset) comes from a man (m) or a woman (f).

The classifier has two phases (algorithms): a training (to build the classifier) and a testing (to test the classifier with new posts).

The algorithm for the training phase is as follows:

TRAINMULTINOMIALNB($\mathbb{C}, \mathbb{D}$)
1 $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
2 $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}, c$)
5 $prior[c] \leftarrow N_c / N$
6 $text_c \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS($\mathbb{D}, c$)
7 **for each** $t \in V$
8 **do** $T_{ct} \leftarrow$ COUNTTOKENSOFTERM($text_c, t$)
9 **for each** $t \in V$
10 **do** $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11 **return** $V, prior, condprob$

Where C is the set of categories, for our dataset C = {f, m}; and D is a collection of posts, where each post is labelled with a category from C (f or m) and represented as a bag of words (collection of words).

The testing algorithm is as follow:

APPLYMULTINOMIALNB($\mathbb{C}, V, prior, condprob, d$)
1 $W \leftarrow$ EXTRACTTOKENSFROMDOC(V, d)
2 **for each** $c \in \mathbb{C}$
3 **do** $score[c] \leftarrow \log prior[c]$
4 **for each** $t \in W$
5 **do** $score[c] \mathrel{+}= \log condprob[t][c]$
6 **return** $\arg\max_{c \in \mathbb{C}} score[c]$

Where C is the set of categories, for our dataset C = {f, m}; V, prior and condprob are the vocabulary, the priors, and the conditional probabilities respectively (these are computed in the training phase); and d is a new post to be classified represented as a bag of words (collection of words).

For the training phase, the training set consists of all the FB posts and their categories. The test phase will take as input any string formed by words (i.e., any FB post) and the program will output the predicted class (f or m) for such string (post). For testing use FB posts that you did not use during training.

2. Compute the information gain of each word in the vocabulary

Information gain measures the amount of information a word carries regarding the categories in the datasets. The highest the information gain, the more important the word.

Extract the vocabulary as the set of unique words appearing in the posts, and for each one compute its information gain.

To compute information gain for a word (token) t we use the following equation:

$$IG(t) = -\sum_{i=1}^{m} P(c_i)\log\big(P(c_i)\big) + P(t)\sum_{i=1}^{m} P(c_i|t)\log\big(P(c_i|t)\big) + P(\hat{t})\sum_{i=1}^{m} P(c_i|\hat{t})\log\big(P(c_i|\hat{t})\big)$$

In this equation:

- c_i is the i-th category. In our dataset we have two categories {f, m}.
- $P(c_i)$ is the probability of the i-th category.
- $P(t)$ is the probability of word t appearing in the posts.
- $P(\hat{t})$ is the probability of word t NOT appearing the posts.
- $P(c_i|t)$ is the conditional probability of the i-th category given that the word t appeared.
- $P(c_i|\hat{t})$ is the conditional probability of the i-th category given that the word t DOES NOT appeared.

Remember that for computing probabilities you only need to count. For the conditional probabilities use the equations seen in class.

Sort the words in the vocabulary by their information gain and return the 50 with the highest value.

3. Implement the single-linkage hierarchical clustering per user

The dataset is organized in posts, first you will need to concatenate all the posts for each user. In this way, each user is represented as a long sequence of words. Transform this representation using tf-idf and normalize, so that you will have a vector representing each user.

The algorithm for single-linkage hierarchical clustering is as follow:

1. Create a proximity matrix using the Euclidean distance. This is a square matrix as shown below. The number of columns and rows is the same as the number of users. The number in each cell is the Euclidean distance between each pair of users.
2. Set sequence number $m = 0$. This number will control de level of the hierarchy.

$$\begin{bmatrix} 0 & d_{12}^2 & d_{13}^2 & \cdots & d_{1n}^2 \\ d_{21}^2 & 0 & d_{23}^2 & \cdots & d_{2n}^2 \\ d_{31}^2 & d_{32}^2 & 0 & \cdots & d_{3n}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1}^2 & d_{n2}^2 & d_{n3}^2 & \cdots & 0 \end{bmatrix}$$

3. Find the minimum distance (minimum value) in the proximity matrix and find the indexes in row/column associated with such distance. If there is a draw in the minimum value, select the first one. The indexes of row/column represent the most similar users.
4. Merge the indexes in a single cluster (use a list to cluster the indexes together).
5. Increase the sequence number $m = m + 1$.
6. Update the proximity matrix by deleting the rows and columns corresponding to the merged indexes (two rows and two columns).
7. Update the proximity matrix by adding a new row and a new column that represent the new cluster. To fill the cells with the distance between the new cluster and all the other users, use the minimum distance between the users in the cluster and any other user.
8. If all users are in one cluster, stop, else, go to step 3.

When a merge occurs, e.g., indexes [5, 9], this must be treated as a new single element when merged with another index or cluster, e.g., [[5, 9], [3, 4]].

The output must be a list of nested lists, such as: [[[5, 9], [3, 4]], 8]. This will represent the hierarchy of the clustering, with the most inside lists as the basis of the hierarchy.