

The creation of the JSON file essential for the D3 Visualisations is created by the functions in the ‘parsingEnron’ python executable. The class starts off by creating an object of type ‘Email’ in order to store the parsed data from the given dataset. The object accepts the email sender, recipients and body as parameters.

Parsing

The root directory “maildirtest” which holds the dataset provided is iterated over with three iterators ‘directory’, ‘subdirectory’ and ‘filenames’, where ‘filenames’ is iterated over with iterator ‘filename’ in a nested for loop. The file is opened using prebuilt os functions, where the parsed data is stored in a variable ‘data’.

The sender of the emails is found with the use of the *email.parser* library. An instance of the library was created where, in this case, ‘from’ was passed as an argument to return the ‘from’ section of the email. A similar operation is carried out when reading the ‘to’ section of the email where the result of the *emailParser* instance is saved into the recipients variable. The same operation is carried out for the ‘bcc’ and the ‘cc’ fields. Since all of the ‘to’, ‘bcc’ and ‘cc’ fields indicate a recipient, the recipient variable is set to be a list for a set of recipients. Similarly, the body of the email is parsed.

An object of type Email is instantiated and appended to the list of all emails in the dataset. Similarly, a list of all the email addresses is created. Afterwards, a dictionary is created of all the combined documents passed between two distinct people by iterating through the emails and recipients lists.

Preprocessing

Preprocessing entails the carrying out of arranging the data to be valid for TF-IDF weighting and Cosine Similarity. Preprocessing consists of tokenisation, casefolding, stopword removal, stemming and symbol removal. Tokenisation is the process of turning sensitive data into nonsensitive data or tokens, this happens with the information being swapped into an algorithmically generated number with the same length and format of the initial data. Casefolding converts all the data passed into lowercased symbols. A pre set value of stop words, such as “and”, “then” and “is” are removed from the dataset through the execution of stop word removal, this selection selects data which is more important or relevant to the document. Stemming is the process of setting a particular word to its most basic form, for example: “Stemming” or “Stemmed” are outputted to be “Stem”. Symbols usually carry negligible importance in a document, therefore a variable symbol is iterated over and checks whether a variable in data contains a symbol, if a symbol is found, the data is appended and placed in a temporary array.

Calculating TF-IDF

Term Frequency – Inverse Document Frequency is a technique to quantify a word in documents. A weight is computed for each word which signifies the importance of the word in the document and ergo the dataset. This weight is then stored in a term by document matrix.

Firstly, the Document Frequency is calculated. The DF calculation function returns a dictionary of the number documents in which each word can be found. The calculation happens by iterating through the length of the dataset and the tokens of the dataset. The tokens are added to a temporary dictionary set in the document frequency function. Afterwards, the dictionary is iterated over, where each value in the dictionary is set to be the length of the same element.

The document values, keys and unique words are now set to be stored in a list, and a DF dictionary is calculated on the list of values. The dictionary *tfidf* is being filled up with an iteration over the length of the dataset and the number of unique tokens as a nested for loop. The TF is calculated by dividing the counter of the current token by the length of the tokens. The DF is calculated by value to be the element in the dictionary of the current token. Meanwhile, the IDF can be described as the log of the length of the dataset divided by the DF. Therefore, the TFIDF is the TF multiplied by the IDF.

Finally, the algorithm gets all the documents in which a user X participates in, participation can be either sending or receiving. All of the TF-IDF values of the documents have their averages stored and the average value is linked with user X. This process is repeated for all the users. The last command executed is that the dictionary is converted to a vector.

Calculating K- Means Clustering Coefficient

The clustering algorithm was tackled by firstly converting the dataset into a *numpy* array. Three points are initialized randomly from the dataset. These three numbers were used as indices and the data points of the indices were returned. The three numbers are our initial centroids, which are then converted to a two-dimensional array. Iterating through all the data points in the dataset, we calculate the distance from the current point to all of the three centroids. And then the point is assigned to the centroid with the smallest distance to it. Finally, the centroids are moved based on the mean of the data points.

Cosine Similarity

Instead of building our own cosine similarity functions, we opted to use the predefined cosine function in the *numpy* library. This is because, upon testing the dot product and normal multiplication were outputting the same values, which is clearly a runtime error, hence resulting in all of the cosine similarity values to be 1. So instead of settling and working with incorrect values, we used the predefined function to continue with the tackling of the task at hand.

Division of Work

Paolo tackled most of Task 1 and Owen tackled most of Task 2 but when either met any problems, we consulted each other, hence participating in both tasks equally. It is also important to note that we checked in frequently on each other to ensure that both students are working and are grasping knowledge of the task at hand.