**List of Potential Projects Data and Text Mining Course 2022**

**LINK FOR PROJECT ALLOCATION -🡪** [**https://1drv.ms/w/s!AtcJs3OTsMZuiSApqWJaa2SDtz2Y**](https://1drv.ms/w/s!AtcJs3OTsMZuiSApqWJaa2SDtz2Y)

**Project 1: Zips Law distribution 1**

This project investigates various facets of the Zips law and how it can be empirically tested on a set of well-known corpuses.

1. First consider the Brown corpus that can be imported using NLTK library. You may consult Chapter 2 of NLTK online book [2. Accessing Text Corpora and Lexical Resources (nltk.org)](https://www.nltk.org/book/ch02.html) for coding examples. i) Write a program that displays in the log-scale, the frequency of words versus the ranking. ii) Display the plot together with its linear fit and the 95% confidence bounds and provide an estimation of the linear curve parameter, corresponding to the parameters of the Zipf’s law. For the 95% linear fit bounds estimation, You can inspire from many existing blogs on linear fitting and statistics. E.g., [[DS0001] — Linear Regression and Confidence Interval a Hands-On Tutorial | by Iago Henrique | The Startup | Medium](https://medium.com/swlh/ds001-linear-regression-and-confidence-interval-a-hands-on-tutorial-760658632d99). iii) Write a script and estimate the percentage of tokens that do not fall within the 95% bounds.
2. Repeat the tasks of 1) when we consider the bi-gram words of Brown Corpus.
3. Repeat tasks of 1) when we consider tri-gram words of Brown Corpus. Summarize the results of the fitting in a Table for the uni-gram, bi-gram and tri-gram.
4. We want to test how the behavior of Zipf’s law when combining corpuses. For this purpose, split randomly Brown corpus into 8 roughly equal samples (C1, C2, ..C8). Draw the zipfs’ law for corpus 1, then for corpus formed by concatenating {C1, C2}, then {C1,C2, C3, C4}, then {C1, C2, C3, C4, C5, C6, C7, C8} (each time we concatenate by (sub) corpus of same size. Draw all the illustrations on the same plot.
5. Consider the unigram Brown corpus of 1), Use part-of-speech tagging of brown corpus. Draw a Zipf’s law graph showing in the log-scale the frequency of the various tags versus their rankings. Show on the same plot the linear fit and its 95% confidence bound and estimate the percentage of tokens that fall outside the bound.
6. Repeat 4) when using the big-gram, highlighting the frequency of tag pairs and their rankings.
7. We want to investigate other stylistic properties of Brown Corpus. Consider the length of individual tokens in the corpus as the main attribute to explore. Draw the corresponding log-scale plot and test the suitability of the Zipf’s law distribution.
8. We want to repeat 6) by exploring the length of the sentences (in terms of number of tokens) in Brown corpus. Suggest a script that extracts sentences and calculate the length of sentences. Then draw the corresponding log-log plot and test the Zipf’s law distribution.
9. We want to provide a visual representation of short and long sentence of the corpus. For this purpose, write a script that retrieve all sentences whose length is less than or equal to four tokens and save the resulting sentences in a separate database. Repeat this process for sentences who length is more than or equal 10 tokens a. Use a wordcloud plot to visualize the content of each database where most frequent wording are highlighted. You may inspire from [How to create a word cloud in Python? (projectpro.io)](https://www.projectpro.io/recipes/create-word-cloud-python).
10. We want to evaluate the use of polysemy in Brown corpus. For this purpose, use a lexical database of your choice (e.g., WordNet) to identify tokens that have more one sense. Then consider a window of size 5 (2 tokens on left and 2 tokens on right hand side around the token with multiple senses. Report the number of other tokens within this interval that have also more than one sense. Write scripts that allow you to visualize histogram showing the percentage of homonyms that are associated with zero other homonyms in the 5 token interval, 1 homonym, 2 homonym, etc. Then write another script that draws the histogram of the tag (part-of-speech-tag) of the homonym.
11. Use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Also, comment on the limitations and structural weakness of the data processing pipeline.

**Project 2: Zips Law distribution 2.**

This project investigates various facets of the Zips law and how it can be empirically tested on a set of well-known corpuses. We want to compare the various corpus developed at different time period.

1. Consider the NPS Chat Corpus, available in NLTK online book, see [2. Accessing Text Corpora and Lexical Resources (nltk.org)](https://www.nltk.org/book/ch02.html). You can inspire from links provided in description of Project 1, to i) Write a program that displays in the log-scale, the frequency of words versus the ranking; ii) Display the plot together with its linear fit and the 90% confidence bounds and provide an estimation of the linear curve parameter, corresponding to the parameters of the Zipf’s law; iii) Write a script and estimate the percentage of tokens that do not fall within the 90% bounds.
2. Repeat 1) when considering the length of tokens used in the corpus.
3. Repeat 1) when considering the PoS tag
4. Repeat 1) when considering dialogue act
5. Now we want to explore how the corpus varies across various ages groups (teens, 20s, 30s, 40s and generic adult). For this purpose, suggest a script that displays the twenty most frequent token (excluding stopwords) in each age-group dataset. Draw worldcloud illustration for each age-group corpus. Write a script to identify tokens that appear only in one single age-group and not in other age-groups. List examples of such working, if any..
6. Run the topic modelling (using LDA) for each age group corpus with number of topic =10 and number of tokens by topic = 10. Provide in a table the results of each age-group corpus exploration.
7. We want to use the LDA classification to test the coherence and distinguishability of the each class. For the coherence metric calculus, we shall consider the LDA classification of a single age-group corpus is coherent if whenever a “word” is mentioned as part of a given topic, it will not be mentioned in any other topic. Therefore define the topic coherence as

Calculate the coherence for each age-group corpus.

Similarly, define the inter-group coherence by the amount of words (in the generated LDA) that co-occur in the topic definitions of other age-group corpus. Suggest an expression and a script that allows you to compute the inter-group coherence between each pair of age-group.

1. We want to investigate the content of each age category group using word embedding approach. For this purpose, use Empath category [GitHub - Ejhfast/empath-client: analyze text with empath](https://github.com/Ejhfast/empath-client). Suggest a graph to represent the main categories of each age-corpus.
2. Use the embedding vector generated by Empath category for each age-group corpus and calculate the similarity among the various pairs of age-group corpus using cosine similarities. Comment on the possible interpretations of the results.
3. We want to test the consistency in terms of dialogue act as well. Specifically, we want to test the extent to which the dialogue acts are consistent across the various age-group corpus. Therefore, for each age-group category, retrieve the subset of dataset corresponding to the same dialogue act, and compute its empath vector. Then compute the corresponding cosine similarity when aggregating the Empath vector of different age groups corresponding to the same dialogue act. The latter plays the role of the coherent measure. Repeat this process for each dialogue act. Similarly, use the Empath vector to test the similarity among the various dialogue acts. For this purpose, concatenate datasets of the same dialogue act together and use cosine similarity to test the similarity of each pair of dialogue acts. Then retrieve the pairs that yield relatively high values of similarity scores.
4. Repeat 10 when using Word2vec embedding instead of Empath category. You may use pre-trained word2vec.
5. Use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Also, comment on the limitations and structural weakness of the data processing pipeline.

**Project 3: Zips Law distribution 3**

This project investigates various facets of the Zips law and how it can be empirically tested on a set of well-known corpuses. We want to compare the various corpus developed at different time period.

1. Consider the corpus of old English poetry that you can download from [The York-Helsinki parsed corpus of Old English poetry (YCOEP)](http://hdl.handle.net/20.500.12024/2425). You can inspire from links provided in description of Project 1, to i) Write a program that displays in the log-scale, the frequency of words versus the ranking; ii) Display the plot together with its linear fit and the 90% confidence bounds and provide an estimation of the linear curve parameter, corresponding to the parameters of the Zipf’s law; iii) Write a script and estimate the percentage of tokens that do not fall within the 90% bounds.
2. Repeat 1) when considering the length of tokens used in the corpus.
3. We want to explore the syntactic / stylistic patterns of the poem. For this purpose, for each line of poem, write a script that outputs the most frequent tag and the less frequent tag. Save the result in an excel file. Then compute the tag-coherence of the poem. This is computed as the proportion of the most common frequent tag across all lines of the poem for frequent-tags, and the proportion of the most common less-frequent tag across all lines for less frequent tag. In other words, you should provide tag-coherence in two dimensions (less-frequent tag and most frequent tag). Suggest a plot where you display the various tags (less-frequent tag, and another one for most frequent tag). Note that in the case where most frequent or less frequent yields more than one output, this should be taken into account in the calculus of the intersection (common tags across all lines).
4. Repeat 3) when exploring the first and last line of each token of the line of the poem (after excluding any stopword).
5. Write a script that uses WordNet lexical database to count the number of homonyms in each line of poem, and draw histogram that shows the distribution of number of homonyms in poem lines. Draw another histogram illustrating the histogram of the tags associated to homonym word.
6. We would like to compare the phonetic of homonym and that of surrounding terms. For this purpose, use the “Fuzzy” library in python to generate the character string that identifies phonetically similar words (see fuzzy.Soundex (d) statement where d is the length of the generated string, and then use edit distance to compute the phonetic similarity between two words. See an example of implementation at <https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/>. Use the size of “d” open to account for more fine tuned phonetic evaluations. Therefore use the phonetic evaluation to compute the i) the edit distance between homonyms (should follow the order they appear in the document – first homonym and the second closest, then between the second one and the third one, etc..). Calculate the mean and standard deviation of the calculated distances. ii) Repeat the edit distance phonetic calculus when considering the first and last token of each line of poem (excluding stopword).
7. Explore the bigrams (words) generated by the corpus and consider the pair of tags associated to each bigram, write a script that identifies the most frequent pair of tags and the less frequent pair at each line of poem. Determine the variation of these two pairs across all lines of the poem using the same concept as tag coherence of 3).
8. Repeat 1-7, when using more modern poem corpus. You may consider list of ebook <http://www.gutenberg.org/> and select a poetry ebook of your choice.
9. Draw wordclouds for most similar phonetic lines in both corpus.
10. Use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Also, comment on the limitations and structural weakness of the data processing pipeline.

**Project 4. Poetry Analysis Using NLP**

This project investigates the poetry properties such as style, rhyme, emotion using Python NLTK, any other relevant packages and should show ability to handle big data using embeddings.

For this purpose, consider the Gutenberg collection of ebooks available at <http://www.gutenberg.org/>. You may notice that for each ebook, you can freely download the full text version of the book. Choose two poetry ebooks of your choice from two different topics (ideally contrasting topics) where each poetry book is made of several chapters, avoiding very short files and download their associated text files. Besides Gutenberg collection can be accessed in NLTK too (from nltk.corpus import gutenberg). The following specifications should be performed for each ebook.

1. Use appropriate NLTK coding (you can inspire from coding examples of the online NLTK book) in order to plot the histogram of the thirty most frequent words in each ebook, excluding stopwords. Save the result on an excel file.
2. We want to evaluate the extent to which these common wordings are linked to the title of the book and/or its chapter titles. Write a script that calculates i) the fuzzywuzzy (variation of edit distance, see lecture handout) ratio of each word (among the thirty frequent words) and title and each of chapter title, ii) Wu and Palmer semantic similarity measure of each word with title and semantic similarity of each word with each chapter title (use the semantic similarity between sentences as experienced in the laboratory session). Summarize the results in a table.
3. We want to comprehend the length of tokens in each chapter of the book. Write a script that calculates the length of each token in each chapter, excluding stopword and draws a histogram showing the distribution of token length within same chapter and another one illustrating the cross-chapters distribution.
4. Repeat 3) when considering the part-of-speech tag of tokens of each chapter.
5. We want to comprehend the distribution of named-entities across chapters. Use spicy named-entity tagger [Linguistic Features · spaCy Usage Documentation](https://spacy.io/usage/linguistic-features) to check the occurrence of named-entities in each chapter. Write a code to display the type of histogram of named-entities occurring at each chapter (organization, person, location types). After unifying the number of named-entity types, suggest a script that computes the mean and standard deviation of the histogram vector across the different chapter.
6. We want to investigate the emotion conveyed by the poetry. For this purpose, use SentiWordnet to compute the overall positive sentiment of each line of poem (calculated as the average of the positive sentiment of each token of the poem line) in each chapter. Then write a script to create a subdivision of 10 bins (by looking into minimal and maximal value of average positive sentiment per line), and draw the corresponding histogram showing the distribution of the positive sentiment of the first chapter. Repeat this process considering all poem (regardless of chapter) and draw the corresponding histogram.
7. Repeat 6) when considering the negative sentiment score.
8. We want to estimate the coherence of the lines poem using Empath categories [GitHub - Ejhfast/empath-client: analyze text with empath](https://github.com/Ejhfast/empath-client). Write a script that generates for input line of poem an embedding vector corresponding to the weight assigned to each category of Empath categories, and then calculate for each two consecutive lines their matching score as the value of their cosine similarity score. Draw a plot showing the variation of this matching score for every chapter. Ensure that the graph for the various chapters are drawn on the same plot to ease comparison (restrict to 6-7 chapter for readability purpose).
9. We would like to compare the phonetic of starting word and ending word of each line of the poem. For this purpose, use the “Fuzzy” library in python to generate the character string that identifies phonetically similar words, and then use edit distance to compute the phonetic similarity between two words. See an example of implementation at <https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/>. (leave “d” without any restriction in terms of number of characters of the generated phonetic code). The distance between the phonetic generated vectors corresponding to starting and ending word is generated in the following way: Assume L1 is the first word phonetic string, and L2 is the second word phonetic string generated by Fuzzy library. Then the phonetic association Sim(L1,L2)= 2\*S/ (length(L1)+length(L2))

where S is the length of the largest substring, which is common to both L1 and L2.

Calculate the value of Sim(L1,L2) for each line of the poem and size the result in excel file. Find out whether some curve fitting (polynomial, exponential or zipf can be fitted to the data). Motivate your answer and display appropriate plotting.

1. We now consider the lexical diversity LD in the poem. This can be expressed using the adjective/adverb-to-verb ratio (number of adjectives and adverbs to the number of verbs) in each line of the poem. Using part-of-speech tagging that identifies verb, adjective and adverb entities, suggest a program that calculates LD for each line of the code. Save the result in the database. Plot the graph of LD. Suggest a 10-equal subdivision of values of LD (take the highest value of LD and subtract the smallest value of LD and divide by 10 to find the bin value, and then take the smallest of LD and add bin, then 2\*bin, etc.. to find the next interval (You will end up with 10 intervals). Now calculate and plot the corresponding histogram (calculating the number of lines of poem whose LD value fall within a specific interval). Find out whether a parametric fitting (polynomial, logarithmic or exponential) can be achieved
2. Discuss your results with respect to existing literature regarding poem structures and artistic trends of your choice taking into account the status and type of poetry of the ebooks you used for this analysis.

**Project 5: Influence of translation on Poetry Style Assessment**

This project aims to investigate the structure of poetry in terms of original structure of the poems with respect to existing corpus. We shall concentrate on Chakspeare HAMLET, with a comparison between its original English version and French translation (See [Hamlet | AnyLang](https://anylang.net/en/books/fr/hamlet/read) for French translation and [Download Hamlet | The Folger SHAKESPEARE](https://shakespeare.folger.edu/shakespeares-works/hamlet/download/) for original version). We want to compare the preservation of inherent stylistic properties through this translation.

1. Write a script that allows you to retrieve the text corresponding to HAMLET (character) sayings in the manuscript, while discarding stopwords and numbering and any non-related text. Separate the sayings of HAMLET character at each act for both original text and translation.
2. Use appropriate Tokenizer to perform the standard preprocessing pipeline (eliminate stopwords, numbers, uncommon characters,..) and recover the root form of individual words using WordNet lemmatizer. Generate the corresponding vocabulary for both English and French corpus and save it a database. Compare the size of the vocabulary of English Corpus and French corpus.
3. Use NLTK tokenizer to distinguish various tokens in each text and suggest a script that calculates average length per line in terms of number of characters, and determine the distribution of the lengths after histogram illustration (you may consult NLTK online book for examples). Draw on the same plot the distribution of English and French corpus.
4. Suggest a script that draws and estimate Zipf’s law fitting using all data of HAMLET (character) corpus for both English and French (draw on the same plot English and French Zipf’s law fitting (you may consult Project 1 description and links to Zipf’s law fitting and confidence estimation).
5. Now we want to assess the coherence each line of English corpus with its counterpart in French corpus. Consider FastText embedding, see [Word vectors for 157 languages · fastText](https://fasttext.cc/docs/en/crawl-vectors.html), which is available for several languages. Write a script that calculates the embedding of the whole line as the average of the FastText embedding of individual words constituting the line. For French corpus, you should use the French embedding available from the above link. Now given the word embedding of a given line in English and its corresponding in French corpus, the consistency score is calculated as the cosine similarity between the two embedding vector. Then, display in a graph the variation of the consistency score for the whole corpus. Report in a table the mean, standard deviation, kurtosis and skewness values of the consistency scores.
6. We want to test the coherence in terms of sentiment analysis score. For this purpose, use Textblob sentiment analyzer as it supports several languages (see example at [French Sentiment Analysis Using TextBlob | Kaggle](https://www.kaggle.com/code/fedi1996/french-sentiment-analysis-using-textblob), which outputs for a given text input, positive, negative or neutral). For each line of English and French corpus, output the overall sentiment score. Then assign a Boolean output of 1 if the sentiment of both lines match and zero, otherwise. Draw the (0-1) histogram showing the proportion of matches and mismatches.
7. Now we want to test the distribution of the most common words in corpuses. For this purpose, consider the two most common adjectives in English corpus, and two most common adjectives in French corpus. Draw the plot of the lexical dispersion of each of these words. This correspond, for a given word, to a plot where you show positions (in terms of number of tokens) of that word from the starting token of the corpus. You may look at existing NLTK lexical dispersion plot available online.
8. Consider the 5 most common tokens (excluding the stopwords, numbering and uncommon characters) for each corpus. Draw the histogram showing the number of occurrences of each these tokens. Use a two-bar representation to the scores for English and French corpus. The equivalence of the histograms would indicate the one-to-one equivalence of these common words too. Now we want to evaluate the ordering of appearance of these common words. Write a script that would allow you to determine the proportion of permutations that co-occur in English and French corpus. This is somehow similar to kendal’s tau, which quantifies correlation between some reference ordering and automatic order (Check Python statistical library for further information).
9. We want to see how the common tokens vary in terms of number of syllables. You may use python library [Pyphen](http://pyphen.org/), to determine the number of syllables of each word. Write a script that calculates the number of syllables for the 100 common tokens in both corpus, and suggest a diagram how the two corpuses compare in terms of number of syllables of their most common tokens.
10. . Use appropriate literature to comment on the findings

**Project 6: Metaphor detection in poetry**

This project explores the detection of metaphors in poetry using natural language processing, aiming to distinguish figurative and non-figurative language.

We shall consider the common use of a phrase as literal use and its violation as an indicative of metaphorical use. The project initially attempts to imitate the approach of Neuman et al. (2013) published in PlusOne journal -Metaphor Identification in Large Texts Corpora- available online [Metaphor Identification in Large Texts Corpora (plos.org)](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0062343). So first consider a British national corpus. For testing, we shall consider the annotated corpus available at <https://www.eecs.uottawa.ca/~diana/resources/metaphor/type1_metaphor_annotated.txt>

In the above, the annotation at the end of the sentence i.e., @1@y indicates whether it is a metaphor (y) or not (n). Here the presence of ‘y’ indicates that it is a metaphor, whereas “1” indicates the first head word of the sentence, which is “poise”, in the part of speech tag sequence.

1. We consider the British national corpus, which contains around 100 millions of words from a wide range of genres (spoken, fiction,..) and tagged with C5 tag-set, see [NLTK :: nltk.corpus.reader.bnc module](https://www.nltk.org/api/nltk.corpus.reader.bnc.html) for examples and details, in our subsequent analysis. Using the examples in the link, write a script that upload the corpus and displays a histogram showing the proportion of tags for work “love” and “hate”; “like” and “dislike”. Display also the number of occurrences of each of these words in the corpus.
2. Consider the mutual information, see expression (2) in Neuman et al.’2003 paper, as a guideline to derive the metaphor-reasoning. You may help with other available implementations of mutual information, in [Collocations (nltk.org)](https://www.nltk.org/howto/collocations.html), [FNLP 2011: Tutorial 8: Working with corpora: mutual information (ed.ac.uk)](http://www.inf.ed.ac.uk/teaching/courses/fnlp/lectures/8/tutorial.html). Consider the words “man”, “sky”, “life”, “love”, “hate”. Write a program that identifies all adjectives, adverbs and verbs that occur within 3 lexical units (span = 3 in the formula of mutual information) in British national corpus and whose mutual information is equal or greater than 3, considered as the minimum statistical significance.
3. We would like to test this process in the previous metaphor annotated dataset. For this purpose, consider the following approach. Write a program that inputs each sentence of the annotated corpus, and then read the head word (given in the annotation), then calculate the mutual distance between the head-word and each of the first two words occurring either on the left hand side part or right hand side part of the head-word. If all mutual distances from head word with each of the two words situated at two lexical units are greater than 2.5, then we shall consider the sentence is not a metaphor, otherwise, it is a metaphor. Test this reasoning and report the result for each annotated sentence and save it in your database. Given the ground truth of the annotated dataset, calculate the corresponding accuracy, and comment on the efficiency of the proposed approach.
4. We consider the (adjective-noun) type of metaphor (referred to as Metaphor type III). A metaphor assumes to occur when the categories of noun and adjective are such that one is concrete and the other one is abstract. WordStat noun categorization based on WordNet, which classifies 69,817 nouns into 25 categories, of which 13 are concrete categories (e.g., artifact) provides a database for a such categorization. It is freely available in [Wordnet based categorization dictionary - Provalis Research](https://provalisresearch.com/products/content-analysis-software/wordstat-dictionary/wordnet-based-categorization-dictionary/). Write a program that allows you to retrieve the category of noun and adjective / adverb in a sentence according to WordStat.
5. Now we would like to imitate the procedure mentioned in Neuman’s paper for type III semaphore. Write a program that identifies the occurrence of Noun-Adjective/Adverb part-of-speech in a given sentence. Then, use WordNet lexical database to find out the number of senses of an adjective. If the adjective has one single sense, then return, no metaphor. If the Noun has no entry in wordnet, then return UNKNOWN. Otherwise (adjective has more than one sense and noun has an entry in WordNet), then identify the set S of nouns in the British national corpus who collocate with the given Noun of the given sentence (this corresponds to a set of nouns whose mutual information value is greater or equal than 3). Next, for each element (noun) of S, use the WordStat categorization to identify those who belong to concrete class. Let S1 be a subset of S, which contains these “concrete”-category nouns. If the number of elements in S1 is large, then restrict to the first three elements who have the highest mutual information values. Finally, to find out whether, whether the sentence containing adjective A and noun N is a metaphor, we need to test the compatibility of each elements of S1 with N. If there is no elements in S1 compatible with N, then we shall consider S as a metaphor, otherwise, it is not. To evaluate this compatibility, you can use the Wu and Palmer WordNet semantic similarity already implemented in NLTK. Therefore, assume that if the Wu and Palmer semantic similarity of at least of the nouns in S1 with N is greater than 0.3, then the compatibility between S1 and N is granted. (Note this is only a very rough approximation). Write a code that implements this reasoning and test it on two simple examples of your choice.
6. Test the above reasoning on the subset of <https://www.eecs.uottawa.ca/~diana/resources/metaphor/type1_metaphor_annotated.txt> where adjective-noun type of relationship occurs. Motivate your reasoning and answers. Estimate the accuracy accordingly, and report individual results in your database.
7. Instead of the calculus of the semantic similarity between N and each elements of S1 in step 4, we would like to use the wordnet domain of each individual words. For this purpose, you download the wordnet domain from [WordNet Domains (fbk.eu)](https://wndomains.fbk.eu/download.html). Therefore, the compatibility between N and an element N1 of S1 is granted if N and N1 belong to the same wordnet domain. Write a program that allows you to implement this reasoning and test it on simple sentences of your choice.
8. Test the reasoning of 6) on the same subset of annotated metaphor dataset used in 5) and compare the performance in terms of accuracy. Save individual results in your database as well.
9. Repeat 6) and 7) when using Reuter corpus (also accessible via NLTK) instead of British national corpus. Conclude on the impact of the corpus on the accuracy of metaphor identification.
10. Use appropriate literature to discuss your findings and their potential limitations.

**Project 7: Suomi24 Corpus Analysis**

This project aims to test the Zipf and Heap law on a large scale Finnish corpus Suomi24, which cover discussion held from 2001 till 2017.

#### Download the Suomi24 dataset from [The Suomi24 Corpus 2001-2017, VRT version 1.1 – META-SHARE (csc.fi)](https://metashare.csc.fi/repository/browse/the-suomi24-corpus-2001-2017-vrt-version-11/10d23b2a522911eaae85005056be118e1399c95f81c24248a0b11a6953398218/). You should click on the access location [**http://urn.fi/urn:nbn:fi:lb-2020021802**](http://urn.fi/urn:nbn:fi:lb-2020021802)

Select University of Oulu in Korp and then login with your university credential to have access to the data. The data is quite large 65GB, so you should make sure that either your local computer or cloud server account will be able to handle it and run basic search. Select keyword “hate” (with all its finish potential translations) that is likely actively occurring in Suomi24. Now we want to investigate the occurrence of this keyword and related vocabulary over years.

1. We want to track the occurrence of the keyword, in threads only of Suomi24 (not taking into account users’s replies and comments). Write down a program that allows you to retrieve the set of threads where the keyword occurs. Save the thread titles /contents together with their associated dates in your database.
2. We would like to investigate the timely progression of the vocabulary size of the collected threads. For this purpose, separate the original dataset into 16 or 17 sub-dataset dat1, dat2, …, dat17 corresponding to data for 2001, 2002, 2003, …, 2017. Use Vector counter of NLTK to extract the vocabulary of each year and its associated size.
3. Plot graphs showing the evolution of vocabulary size on a yearly basis and comment the results.
4. Plot graphs showing the evolution of the vocabulary size with respect to number of tokens for each year (one graph per year). Find out whether a parametric Heaps law can be fit. Utilize the confidence value of the prediction to draw upper bound and lower bound of the curve fitting and quantify to goodness of fit. You can inspire from examples in spacy.stats.linregress or examples in <https://towardsdatascience.com/how-to-perform-linear-regression-with-confidence-5ed8fc0bb9fe>. Test different values of the confidence value, e.g., 80%, 85%, 90% and 95% and report the number of points that fall outside the upper and lower bounds for each case.
5. Suggest a script, which identifies, for each year (from 2001 till 2017), the most frequent 10 words where your keyword co-occur most often at 3 units lexical distance (within a window of three words). You should provide 10 words for each year. Suggest a plot that represents the findings over all the years where the co-occurring words are represents as dots in the graph.
6. Suggest another graphical illustration, which show the yearly evolution of the ten most frequent co-occurring words, their overlapping and their intensities. Save the result of the yearly evolution in your database.
7. We want to investigate the threads that induces a larger discussion around the topic. Suggest a script, that for each year, evaluate the number of tokens in the discussion part of each thread title. Save the threats and their associated discussion score (in terms of number of tokens) in a separate database file. Manually read the top 5 ranked threads and write down your own perception and understanding of the content of the discussion in the database file.
8. Group all threads as well as the related discussion of each year in a separate file and conduct topic modelling using LDA (see implementation in Gensim library) with 2 topics and 5 keywords per topic. Write the result of the topic modelling in a database and reproduce it in a single excel table.
9. We would like to comprehend the content in terms of sentiment analysis. For this purpose, use AFINN sentiment analyzer, which supports several language, including Finnish language and lexicon where each word is assigned a polarity score. See, [afinn · PyPI](https://pypi.org/project/afinn/), for the python library and examples. Write a script for calculating, for each year, the sentiment score for each thread and related discussion. Save the result in a separate database. Separate in the database, the positive sentiment case from negative sentiment. Concatenate all statements yielding positive sentiment in a given year. This yields a text file for each year. Conduct topic modelling as before to identify the key topics associated to positive sentiment on yearly basis. Repeat this process for negative sentiment statements as well.
10. Use appropriate graphical illustrations to highlight the yearly variation of topics associated to each “hate”.
11. Use appropriate literature and state of the art work to comment on your results and methodology

**Project 8: English Corpus Analysis**

In this we shall consider the ICE Nigeria corpus, available in [ICE Nigeria download | SourceForge.net](https://sourceforge.net/projects/ice-nigeria/), and the brown corpus available from NLTK library.

We want to comprehend the occurrence of “hate” and “love” in both corpuses.

1. Write a script that outputs the various word inflections for words “hate” and “love” (e.g., hates, hated, hating, loves, loving, loved,..), and draw a histogram showing the number of occurrence of each word (with all its word inflection forms) in each corpus.
2. Write a script that outputs the 50 most co-occurring word (excluding stopwords) where the co-occurrence is calculated for a distance of 1, 2 and 3 token form the target word (one of inflecting words of “hate”, “love”) for each corpus. You may inspire from examples of NLTK online book.
3. Write a script that retrieves the sentences where the above co-occurrence occurs. Concatenate all sentences associated to the same target word, in a given corpus, into the same document that you save in the database file (a total for 4 separate files should be distinguished).
4. Draw wordcloud representation for each target word in each corpus and comment on the main trend of the discussions.
5. Use LDA implementation in genism library with number of topics =10 and number of keywords per topic = 10, to output the topics associated to each of the previously mentioned four documents.
6. Suggest a graphical illustration of the result of the topic modelling using wordcloud of the keywords or colored representation, see examples in [Topic modeling visualization - How to present results of LDA model? | ML+ (machinelearningplus.com)](https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/). Comment on the extent to which the occurrence of “hate” and “love” can be compared between the two corpuses.
7. Use Vader sentiment analyzer to calculate the overall sentiment of each document (within the four documents, previously established).
8. Using the tagging information of the corpus data, write a script that outputs the 50 most co-occurring adjectives or adverbs with the target words at a distance 1, 2, or 3 tokens. Draw histograms showing the proportion of each tag type for each target word in each corpus.
9. We would like to compare the phonetic of 50 most co-occurring words (at distance equal to one) between the two corpuses for each target word. For this purpose, use the “Fuzzy” library in python to generate the character string that identifies phonetically similar words, and then use edit distance to compute the phonetic similarity between two words. See an example of implementation at <https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/>. (leave “d” without any restriction in terms of number of characters of the generated phonetic code). Specifically, given a target word, e.g., “hate”, for each co-occuring word Di, generate the phonetic code (character), say, C1(Di). Then, similarly, given the set of phonetic code C2(Dj) of co-occurring word of “hate” in the other corpus, compute the distance between C1(Di) and their counterparts obtained using the other corpus, as the minimum edit distance min(C1(Di), C2(Dj)) over all j (all co-occurring words in the other corpus).
10. Calculate the value of Sim(L1,L2) for each line of the poem and size the result in excel file. Find out whether some curve fitting (polynomial, exponential or zipf can be fitted to the data). Motivate your answer and display appropriate plotting.
11. Comment on the findings using appropriate literature.

**Project 9: Sentiment Analysis and Hotel Reviews**

This project aims to investigate the sentiment and test various architecture for argumenting the sentiment polarity. One shall consider the hotel sentiment review available at [Sentiment Analysis from Hotel Reviews | Kaggle](https://www.kaggle.com/code/pacogiu/sentiment-analysis-from-hotel-reviews/data) which contains user’s review and rating. Download the dataset in an easy searchable database. You can also inspire from the available coding used to preprocess the dataset and calculate the sentiment.

1. Use the Vader sentiment analysis, available in NLTK, and write a script that you to input the review file and output the positive score, negative score, aggregate score and store the result in a database file. You may inspire from code, e.g., [SENTIMENTAL ANALYSIS USING VADER. interpretation and classification of… | by Aditya Beri | Towards Data Science](https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664).
2. Write a script to calculate the Pearson correlation between the user’s rating and aggregate sentiment score. Separate the sample dataset according to the nature of sentiment (positive, negative and neutral). For each class, calculate the correlation between user’s rating and the corresponding sentiment score. Conclude on the class that better ensures high correlation between sentiment and rating.
3. Write a script that calculates the length of the review in terms of number of tokens of each review.
4. Calculate the Pearson correlation score for both positive and negative sentiment cases between the sentiment score and the review length, and between review length and user’s rating. Conclude whether review length can be a valuable factor for comprehending the sentiment or the rating.
5. We want to test the hypothesis that negative review often contains ill-constructed sentence and uncommon wording. For this purpose, suggest a script that tests whether each word of the review text has an entry in WordNet and outputs the number of unmatched tokens. Save the result in a database file. Calculate the Pearson correlation between the number of unmatched tokens with the rating for each class of sentiment, and conclude whether the number of unmatched tokens can be relevant.
6. Now we want to comprehend inconsistent reviews. For this purpose, from the dataset, identify those reviews for which the aggregate sentiment is positive while the rating is low, and those where the sentiment is negative while the rating is high. Aggregate all these reviews into a single file.
7. Use wordcloud representation to plot the content of this file.
8. Use Empath categories to comprehend the main categories where conflict between rating and sentiment is high.
9. Use topic modelling with number of topics = 10 and number of keywords per topic = 10. See previous project description and draw keywords of the topic modelling result.
10. We would like to see to test the hypothesis that ambiguous reviews have bad readability. For this purpose, consider the Automated readability Index, which expresses the Readability of review as 4.71(#characters/word) + 0.25( #words/Sentence)-21.43. This is already implemented in python Textstat library <https://pypi.org/project/textstat/>. Calculate the automated readability index for each review and add it to database D. Then, test whether poor readability value entails high probability to belong to ambiguous class.
11. Repeat the steps 1-9 using two alternative sentiment analyzers tools: SentiStrength from <http://sentistrength.wlv.ac.uk/> and Textblob
12. Use appropriate literature to provide theoretical foundations for the results.

**Project 10: Sentiment Analysis and Machine learning**

Consider the Restaurant Review Dataset available at [Restaurant Customer Reviews | Kaggle](https://www.kaggle.com/datasets/vigneshwarsofficial/reviews). You may also scrutinize the various available implementations that used the dataset available in Kaggle. The dataset includes a user’s review and a binary indicator to indicate whether the user likes it or not.

1. Use initially textblob implementation of sentiment, which provides a three class output (positive, negative and neutral sentiment polarity). Assuming that both neutral and negative sentiment score are cast as part of “dislike” category or “0” and positive sentiment is cast in “1”, compute Pearson correlation between this constructed sentiment polarity and the annotation.
2. Repeat 1) when using the cosine similarity measure. Repeat this process when considering the correlation of the positive class alone and the correlation of the negative class alone.
3. Now we want to test the correlation with respect to some stylistic aspects of the review. Write a script that estimate the length of the review in terms of number of characters. Compute both the Pearson correlation and the cosine similarity between the Review length and the annotations.
4. We want to test the hypothesis that the opinion of about the restaurant is constructed according to Price, Quality of food, Quantity of the food, and location of restaurant. Suggest a script that allows you to identify Review that are more focused on price, quality of food, quantity of food and location of restaurant. You may consider a set of keywords that are most suitable to each category and then use simple string matching to match this effect. For each category, generate a binary vector indicating whether the given review focuses on the corresponding category.
5. Estimate the correlation using Pearson correlation and cosine similarity between each vector category and the data annotation.
6. We want to revisit the construction of the categories in 4). Instead of string matching, use the semantic similarity in the following way. Calculate the Wu and Palmer similarity between “price” and the Review (using the sentence-to-sentence similarity as in labs), repeat this process for the other three categories by suggestion a representative keyword (s) that will be used to calculate sentence-to-sentence similarity score.
7. We want to test another approach for computing the categories by using the empath categories embedding. For this purpose, re-visit the naming of the empath-categories in [GitHub - Ejhfast/empath-client: analyze text with empath](https://github.com/Ejhfast/empath-client) and select those that might be linked to Price, Quality, Quantity, Location. Write a code that allows you to determine appropriate categories from this embedding and then calculate the correlation score. Alternative to manual scrutinization of the Empath categories, you may also generate an empath category embedding for the keyword “price”, “food quality”, “food quantity”, “location” and then compute cosine similarity between the Review embedding vector and each of the above four embedding vectors, so that the one that yields the highest similarity score will be considered as the one that best represents the underlined category.
8. Repeat the 1-7) when using SentiStrength sentiment analyzer instead of textblob. The package is available from sentistrength.wlv.ac.uk.
9. We want to further emphasize on misclassified reviews. For this purpose concatenate all reviews for which the sentiment score is positive while the annotation is zero and those for which the sentiment is zero while the annotation is 1. Construct the Wordcloud of this dataset. Write a histogram showing the 10 most common wordings in this dataset. Comment on the findings.
10. Now we would like to build a machine learning model for sentiment analysis that takes into account the ambiguous cases identified in 9). For this purpose, write and script and review the preprocessing and stopword list to not discard relevant information in the context of sentiment analysis (e.g., avoid discarding negation cues, adjectives that subsumes polarity and apostrophes, lower-case as capitalization brings emotion,..), then use TfIdfVectorizer with a maximum feature set of 1000, minimum 2 repetition and no more than 60% of word repetition across sentences. Build this model for one dataset using randomly selected 70% training and 30% testing. Report the classification accuracy.
11. Use FastText encoding instead of TfidfVectorizer, see <https://github.com/facebookresearch/fastText/archive/v0.2.0.zip>

You should install Fasttext and consult one of the tutorial to find out how you will run it, see, e.g., [FastText Word Embeddings Python implementation - ThinkInfi](https://thinkinfi.com/fasttext-word-embeddings-python-implementation/).

Use the FastText embedding as feature vectors and test the performance in the original data (30% test data) and report the classification accuracy on the other two datasets. Comment on the limitations of the approach

1. Identify appropriate literature to comment on your findings and methodology.

**Project 10 Bis: Sentiment Analysis and Machine learning**

Consider the Restaurant Review Dataset available at [Restaurant Customer Reviews | Kaggle](https://www.kaggle.com/datasets/vigneshwarsofficial/reviews). You may also scrutinize the various available implementations that used the dataset available in Kaggle. The dataset includes a user’s review and a binary indicator to indicate whether the user likes it or not.

1. Use initially SentiStrength [SentiStrength - sentiment strength detection in short texts - sentiment analysis, opinion mining (wlv.ac.uk)](http://sentistrength.wlv.ac.uk/) implementation of sentiment, which provides negative and positive sentiment score, compute Pearson correlation between this constructed sentiment polarity and the annotation.
2. Repeat this process when considering the correlation of the positive class alone and the correlation of the negative class alone.
3. Now we want to test the correlation with respect to some stylistic aspects of the review. Write a script that estimate the number of personal pronouns and number of adjectives and number of adverbs using part-of-speech tagger of your choice. Compute both the cosine similarity between each of the above attributes (number of pronouns, number of adjectives, number of adverbs) and the annotation.
4. We want to test the hypothesis that the opinion of about the restaurant is constructed according to Price, Quality of food served in the restaurant, and friendly staff. Suggest a script that allows you to identify Review that are more focused on price, quality of food, friendly staff. You may consider a set of keywords that are most suitable to each category and then use simple string matching to match this effect. For each category, generate a binary vector indicating whether the given review focuses on the corresponding category.
5. Estimate the correlation using Pearson correlation between each vector category and the data annotation.
6. We want to revisit the construction of the categories in 4). Instead of string matching, use the semantic similarity in the following way. Calculate the Wu and Palmer similarity between “price” and the Review (using the sentence-to-sentence similarity as in labs), repeat this process for the other three categories by suggestion a representative keyword (s) that will be used to calculate sentence-to-sentence similarity score.
7. We want to test another approach for computing the categories by using the empath categories embedding. For this purpose, re-visit the naming of the empath-categories in [GitHub - Ejhfast/empath-client: analyze text with empath](https://github.com/Ejhfast/empath-client) and select those that might be linked to Price, Quality, Staff friendship. Write a code that allows you to determine appropriate categories from this embedding and then calculate the correlation score. Alternative to manual scrutinization of the Empath categories, you may also generate an empath category embedding for the keyword “price”, “food quality”, “friendly staff”, and then compute cosine similarity between the Review embedding vector and each of the above four embedding vectors, so that the one that yields the highest similarity score will be considered as the one that best represents the underlined category.
8. We want to further emphasize on misclassified reviews. For this purpose, concatenate all reviews for which the sentiment score is positive while the annotation is zero and those for which the sentiment is zero while the annotation is 1. Construct the Wordcloud of this dataset. Write a histogram showing the 10 most common wordings in this dataset. Comment on the findings.
9. Now we would like to build a machine learning model for sentiment analysis that takes into account the ambiguous cases identified in 9). For this purpose, write and script and review the preprocessing and stopword list to not discard relevant information in the context of sentiment analysis (e.g., avoid discarding negation cues, adjectives that subsumes polarity and apostrophes, lower-case as capitalization brings emotion,..), then use TfIdfVectorizer with a maximum feature set of 500, minimum 2 repetition and no more than 60% of word repetition across sentences. Build this model for one dataset using randomly selected 70% training and 30% testing. Report the classification accuracy.
10. Use Glove embedding instead of TfidfVectorizer, see [GloVe: Global Vectors for Word Representation (stanford.edu)](https://nlp.stanford.edu/projects/glove/). Use the Glove embedding as feature vectors and test the performance in the original data (30% test data) and report the classification accuracy on the other two datasets. Comment on the limitations of the approach
11. Identify appropriate literature to comment on your findings and methodology.

**Project 11. Analysis of political events using Twitter**

Consider The Twitter dataset of Ukraine conflict available at [Ukraine Conflict Twitter Dataset (53.33M tweets) | Kaggle](https://www.kaggle.com/datasets/bwandowando/ukraine-russian-crisis-twitter-dataset-1-2-m-rows). As the dataset is high, so you may consider taking only a sample of dataset up to 100MB dataset size. You may look at examples in NLTK online book of script handling tweet dataset.

1. Suggest a script using a sentiment analyzer of your choice and outcomes the sentiment polarity of each tweet in the collection. Draw the histogram of positive, negative and neutral polarity tweets.
2. We want group all posts of positive sentiment together as a single document, and those of negative sentiment together, and finally, neutral sentiment posts as a single document as well. For each of these documents, perform LDA operation with number of topics =10 and number of words per topics =10.
3. Draw the WordCloud representation of each of the three documents, both preprocessing is carried out and without preprocessing phase.
4. Use spacy to identify the named-entities in each of the above document and draw a histogram showing the 20 most frequent named-entities in each of the three documents.
5. Consider the term “Brexit”, and consider a window size of three on left and right hand side. Suggest a script that outputs the 20 most co-occurring words with Brexit at window size 7 (three on left and three on right) in each of the three documents. Represent the outcome as a histogram.
6. Run empath client <https://github.com/Ejhfast/empath-client> on each of the three documents and report categories which have no zero-value. Discuss the potential overlapping between LDA results and empath client results.
7. Now we would like to comprehend the public opinion in terms like, hate, trying to understand what user like / support and what they hate / dislike in each class. For this purpose, use wordnet to suggest a set of keywords semantically related to “hate” and a set of keywords semantically related to “like”. Write a script, which similarly to 5) identifies the list of word that co-occur with one of the keywords of hate (resp. like), at window size 7 and output the histogram of the 30 most co-occurring words for hate and 30 most co-occurring words for like, in each of the three documents. Suggest a manual categorization of these keywords in a way to ease explanation of the outcomes.
8. We want to focus on modal verbs (e.g., shall, must, need) to comprehend what need/must/shall be performed in each class. For this purpose, for each class of document, record the sentences that contain modal verbs altogether. Perform a simple count analysis to identify the most frequent words in each class. Then perform named-entity tagger to output the most frequent named-entities in each case, then empath client to record the most dominant categories in each case.
9. Repeat the above process when considering “wish”. Identify a set of semantically equivalent wording to “wish” and repeat the process 8) to identify frequent words, frequent named-entities and categories.
10. Identify relevant literature to comment on your finding and discuss limitations of the employed approach.
11. Design a simple GUI interface that allows you to demonstrate and exemplify your findings

**Project 12: Automatic Text Summarization 1**

This project aims to implement new approaches for automatic text summarization and evaluate their performances on small sample dataset. The Rouge-N metric is the standard in evaluating the

1. First, study the open text summarization available in <https://github.com/jaijuneja/PyTLDR> It uses an extraction based summarization where the sentences are scored and the highly scored sentences are included in the summarizer. Three scoring techniques have been implemented on this package. One is based on TextRank algorithm (it uses PageRank) and the second is based on Latent Semantic Analysis. (You can also check for another PageRank summarizer at <https://github.com/davidadamojr/TextRank>), while the third one uses relevance sentence scoring using cosine similarity, see details on the link. Check that the programs correctly when using either html documents or text documents as input. Demonstrate this finding through an example of your own original document and comment on the summarizer outputted by TextRank, Latent Semantic and Relevance sentence scoring algorithms.
2. Design a simple GUI where the user can input a link or source file of the document to be summarized and output the summarizer using each of the three above methods.
3. We would like to evaluate the performance of the three summarizers using a standard evaluation metric. ROUGE-2, ROUGE-3 are commonly employed to evaluate the extent of overlapping between an automatically generated abstract and a set of manually generated summaries. Consider the CNN/Dailymail dataset that you can download from <https://github.com/morningmoni/FAR>. You need a simple python script that allows you to quantify ROUGE-2 and ROUGE-3, you can inspire from numerous implementations available online of automatic summarizers. Your task is to assess the performance of each of three summarizers on CNN/Dailymail dataset using ROUGE-2 and ROUGE-3 metrics, You should . Comment on the performance and limitations of the tested algorithms.
4. We want to extend the above summarization by incorporating coherence of text with respect to named-entity. For this purpose, first use SpaCy named-entity tagger and identify person or organization named-entity. Suggest a simple heuristic that enables whenever a sentence outputted by a given algorithm contains a person or an organization named-entity, then other sentences in the original document that contain the same named-entity, if not outputted by the underlined algorithm, will also be included in the summarizer up to a certain threshold (that you can discuss and tune up). Run the newly designed algorithm on the same CNN/Dailymail dataset, and report the ROUGE-2 and ROUGE-3 performances.
5. Consider Wikisum dataset available at <https://registry.opendata.aws/wikisum/>. You may also consider a sample of the dataset if the processing takes too long to process.
6. Study an implementation of Edmundson summarization system, which uses basic features (word frequency, position, cue words, document structure) available in [Edmundson Heuristic Method for text summarization (opengenus.org)](https://iq.opengenus.org/edmundson-heuristic-method-for-text-summarization/). Test the program in terms of Rouge-1, Rouge-2 score for Wikisum dataset.
7. Review the readability measures implemented in <https://github.com/mmautner/readability>. Consider two readability measures of your choice and write a script that evaluates the readability score of the generated summary in each case.
8. Identify relevant literature that allows you to comment on the methodology and results of your implementation.
9. Suggest a GUI where the user can input his own text to be summarized (or a link / pointer to the location of the original document) and output the summary according to each of the aforementioned methodologies.

**Project 13: Automatic Summarization 2**

We shall consider structured document containing a title, abstract and a set of subsections. We would like to build a text summarizer such that it tracks important keywords in the document. For this purpose, the first step is identify these keywords.

1. Assume the initial input is given as html document (choose an example of your own), we hypothesize that important keywords are initially contained in the words of titles, abstract and possibly titles of subsections of the document. Suggest a simple python script that inputs an html document and outputs the lists of words in the title, abstract and title of section/subsections.
2. Write down a simple python script that allows you to output the histogram of word frequency in the document, excluding the stopwords (see examples in online NLTK book). Use SpaCy named-entity tagger to identify person-named entities and organization-named entities in the document.
3. We would like the summarizer to contain frequent wording (excluding stopwords) and as many named-entities as possible. For this purpose, use the following heuristic to construct the summarizer. First we shall assume each sentence of the document as individual sub-document. Use TfIdf vectorizer to output the individual tfidef score of each word of each sentence (after initial preprocessing and wordnet lemmatization stage). Then consider only sentences that contain person or organization named-entities and use similar approach to output the tfidf score of the named-entities in each sentence. Finally construct the sentence (S) weight as a weighted sum:

where NMTfiDF stands for the TfIdF of named-entity NM in sentence S. POSS corresponds to the sentence weight associated to the location of the sentence. So that the sentence location weight will be maximum (1) if located in the title of the document, 0.5 if located in the title of one of the subsection, 0.25 if located in the title one of the subsubsection, 0.1 if located in one representative object of the document, and 0 if located only in the main text. Make sure to normalize the term tfidf and Nm tfidf weights and suggest a script to implement the preceding accordingly, so that the summarizer will contain the 10 sentences with the highest Sweight scores.

1. Test the above approach with Opinosis dataset available at <https://kavita-ganesan.com/opinosis-opinion-dataset/#.YVw6J5ozY2x>, and record the corresponding Rouge-2 and Rouge-3 evaluation score.
2. We would like to improve the summarization by taking into account the diversity among the sentence in the sense that we would like to minimize redundancy among sentences. For this purpose, we shall use the sentence-to-sentence semantic similarity introduced in the NLP lab. Next, instead of recording only the 10 sentences with highest Sweight scores, we shall record the 20 top sentences in terms of Sweight scores. Then the selection of the top 10 sentences among the 20 sentences follows the following approach. First, order the 20 sentences in the decreasing order of their Sweight scores, say S1, S2, …, S20 (where S1 is the top ranked and S20 the 20th ranked sentence). Second, we shall assume that S1 is always included in the summarizer, we shall then attempt to find the other sentences among S2 till S20 to be included into the summarizer. Calculate the sentence-to-sentence similarity Sim(S1,Si) for i=1 to 20, the Sentence Sj that yields the minimum similarity with S1 will therefore be included in the summarizer. Next, for each of the remaining sentences Sk (with k different from 1 and j), we calculate the sentence similarity with Sj. Therefore the sentence Sp that yields minimum value of “Sim(Sp, S1)+Sim(Sp,Sj)” will be included in the summarizer (Note: the quantity Sim(Sp, S1) is already calculated in previous step). Similarly in the next phase, we should select a sentence Sl (l different from 1, j and k) so that “Sim(Sl, S1)+Sim(Sl,Sj)+Sim(Sl,Sp)”, Etc.. You then stop once you reached 10 sentences included in the summarizer. Suggest a script that includes this process.. and illustrate its functioning in the example you chosen in 1).
3. We would like to make the choice of keywords not based on histogram frequency but using the open source RAKE <https://www.airpair.com/nlp/keyword-extraction-tutorial>. Repeat the previous process of selecting the sentences that are associated to the ten first keywords generated by RAKE. Comment on the quality of this summarizer based on your observation
4. It is also suggested to explore alternative implementations with larger number of summarization approaches implemented- <https://github.com/miso-belica/sumy>. Show how each of the implemented summarizer behaves when inputted with the same document you used in previous case.
5. Now we would like to compare the above summarizers and those in 3), 5) and 7) on a new dataset constructed as follows. First select an Elsevier journal of your own and select 10 papers highly ranked in the journal according to citation index (The journal papers should be well structured to contain Abstract, Introduction and Conclusion). For each of the ten papers, consider the introduction as the main document to seek to apply summarizer, and consider the Abstract and Conclusion as two golden summary of the document that you can use for assessment using ROUGE-1 and ROUGE-2 evaluation. Report in a table the evaluation score of each summarizer.
6. Design a simple GUI that allows the user to input a text or a link to a document to be summarized and output the summarizer according to 3), algorithms implemented in 7)

**Project 14: Text Categorization –**

Study the PyMed API that provides access to PubMed through PubMed API, which is available at [pymed · PyPI](https://pypi.org/project/pymed/). and test that it works consistently in your platform.

Now we want to test the coherence of the outputs for some specific query. Consider the queries:

Q1: summarization of medical documents

Q2: clinical summarization of documents

The project aims to evaluates the consistency of the search outcome from both query and retrieved document perspectives.

1. Write a script that input the query Q1 /Q2 and outputs the 100 list of answers organized in terms of abstract, list of authors, title of document, publication venue, year and list of keywords, if any. Save the result in a database file.
2. We would like to evaluate the amount of dublication in the outputted answer for each query. Write a script that output the size of the abstract in terms of number of tokens. Then suggest an approach that determines if two titles of documents are identical. If so, then, the program checks the abstract count (in terms of number tokens), if this is found identical, then it is considered as a dublication.
3. We want to take into account the ordering information provided by the search API. For this purpose, assume that each abstract is considered as a separate document. So, Use information retrieval like approach, using tf-idf vectorizer in genism library to evaluate the similarity of each document to query (Q1 or Q2), then check whether the ordering of the search API is consistent with similarity scores for both Q1 and Q2. You may use the Kendal rank correlation coefficient (available in python statistics library) to evaluate the extent to which the ordering information is preserved between similarity score and search API result.
4. We would like to quantify the extent to which the title agrees with query. Repeat 3) when the documents are now constituted only of the title of the publication.
5. Now we would like to provide a graphical illustration of the results. For this purpose, consider the pretrained FastText embedding, [Word vectors for 157 languages · fastText](https://fasttext.cc/docs/en/crawl-vectors.html). Write a script that computes the FastText embedding vector of each abstract. Then compute the principal component analysis PCA with three components, of each embedding vector. This generates a vector of 3 dimension for each abstract. Now represent the constructed 3D vectors as a dot in 3D space. This illustrates the highly similar and highly dissimilar abstracts generated by the API.
6. Repeat 5) when using Glove embedding [GloVe: Global Vectors for Word Representation (stanford.edu)](https://nlp.stanford.edu/projects/glove/) instead of FastText embedding and the embedding of the document is seen as the average of the embedding vectors of individual words constituting the document.
7. Repeat 5) when considering only the titles of the documents instead of the whole abstract.
8. Use YAKE library github.com/LIAAD/yake to extract 5 most important keywords from each abstract. Then write a script to evaluate the amount of overlapping (e.g., number of common keywords) between any pair of abstract. Suggest a scatter graph that shows the interconnection between the various abstracts.
9. Design a simple interface that input a text query, retrieve document and visualize the result using one of the visualization tool we discussed earlier.
10. Suggest appropriate literature to comment on the findings.

**Project 15: Text Categorization –**

Consider the journal radiology by RSNA.

We want to test the coherence of the keywords and topics when entered by the authors in their published papers.

The journal includes a set of predefined categories (e.g., Breast imaging, Cardiac Radiology, Chest Radiology, Computed Tomography, Education, Emergency radiology, …)

1. Write a script to crawl the 100 abstracts of each category span at disperse period of time (preferably). The result would include title, authors, abstract and keyword, if available. Save the result in a database file.
2. We would like to evaluate the amount of overlapping between the topic, title and abstract. Initially use FuzzyWuzzy of edit distance to calculate the distance between each topic and title of document. Illustrate through graphical plot (1D plot), where each title is represented as a point in a one dimensional space, the coherence of each topic in this regard. Calculate the mean and standard deviation of this distance
3. Suggest a script that uses tf-idf vectorizer of genism to calculate the similarity between the topic title and the abstract. Calculate the mean and standard deviation over all abstracts associated to the same topic. Provide the result in a table.
4. Repeat 3) when considering the title of the document instead of the abstract.
5. Now we would like to provide a graphical illustration of the results. For this purpose, consider the pretrained FastText embedding, [Word vectors for 157 languages · fastText](https://fasttext.cc/docs/en/crawl-vectors.html). Write a script that computes the FastText embedding vector of each abstract as well as the Fasttext embedding of the topic description. Then use the principal component analysis PCA to reduce the dimension of the embedding to two. Then represent in a 2D space both the topic (as one dot) and all abstracts (as dots) to see how the abstracts are close to the reference point of the topic.
6. Repeat 5) when considering the titles of the document and not the abstract.
7. Repeat 5) when title are added to the abstract, yielding an extended abstract that will then be used for generating the embedding vector.
8. Repeat 5) when you consider the word2vec embedding instead of FastText.
9. Repeat 5) when using BERT embedding, [bert-embedding · PyPI](https://pypi.org/project/bert-embedding/), Note. You may use execution through Colab Google infrastructure to avoid memory issues with BERT implementation and execution.
10. Suggest appropriate literature to comment on the findings.

**Project 16: Climate Change News Analysis. Discovering Arguments**

1. Identify a hot topic in climate change of your choice which has been extensively commented in the media. You are free to phrase the topic as a set of keywords or single phrase or any combination of the above (e.g., pollution, emissions, water contamination…) and motivate your answer based on your observations and heuristic (literature, personal opinion,..,) (no need to do any programming task here)
2. Use news sources that offer APIs for conducting automatic search (e.g., guardian news paper, BBC news, etc.) and input your suggested topic as a search phrase. Design a program that retrieves the first 20 search outcomes (see examples in NLTK online book for crawling html documents, and also examples in <https://pypi.org/project/google-search-results-serpwow/#simple-example> for creating and using Google API for the purpose of retrieving query snippets). You can also manually check the retrieved search to include only those documents (search results) which contain high proportion of user’s generated content in terms of comments on the news or topic raised by another user. For each outputted document, generate a separate document that includes only comments related to that document.
3. We would like to test the extent of overlapping between the original document and the user-generated document. For this purpose, for each search output, use standard preprocessing including stopword removal and tokenization strategy and then draw the histogram of the most frequent words (outside stopword list) for both the original document and its corresponding user-comments. Calculate Jacquard index (ratio of number of common frequent words (among the top 20 most frequent terms) over the total number of distinct words in the top 20 frequent words) for each search result.
4. Similarly to Jacquard coefficient, run LDA model for identifying the topics of original search document (without user comment) and its associated user’s generated document. Use LDA with three topics and 5 words per topic. Create the list L1 of words generated by LDA for original document (without user’s comments) and the list L2 of words generated by LDA for the user’s generated document. Calculate the associated Jacquard index between L1 and L2.
5. Repeat reasoning 4) for sentiment analysis. For this purpose, use sentistrength, for calculating the vector positive and negative sentiment for both original and user’s generated document (each vector is two component vector corresponding to positive and negative sentiment value). Calculate Pearson correlation to calculate the statistical correlation between sentiment associated to original document (without comment) and that of user’s generated comment. Repeat this process for each search result.
6. We would like to evaluate the extent to which the users agree and/or disagree with policy-maker or public organization. For this purpose, identify the list of negative emotion wording using a corpus of your choice (e.g., Empath ..), then use parser tree to identify, in each user’s generated document, the entity the negative sentiment word is associated with. Generate the histogram of these entities in overall.
7. We now would like to investigate the behavior of users who make comments on the original document. For this purpose, elaborate a list of your own for agreement act (e.g., agree, OK, sure, right…) and another list of disagreement, and draw a histogram of agreement act and disagreement (just by counting number of agreement act related words and number of disagreement act related words).
8. Design and implement a simple GUI interface that would allow you to demonstrate and exemplify your reasoning.

**Project 17: Analysis of Online Reviews of Parking Mobile apps using NLP and Technology Acceptance Models (TAM).**

In this project, the students will collect Thousands of online reviews from Apple and Google Play stores from across Europe or at least Nordic area, to understand how users react to emergency apps that run in the Nordic region – **111 Suomi,** and **Emergency SOS, EmergencyPlus**. Feel free to add or amend if not enough reviews were collected. This project will provide insight into user behavior, sentiment toward emergency apps and highlight the most important topics regarding users' requests, demands and preferences in terms of emergency solutions or technology features.

1. Write a script to download data for the two apps and save them in CSV files separately. Data collection involves getting **user IDs, reviews, reviews’ ratings, reviews’ dates, and version history** of the apps from Google play and Apple store. Additionally, you can collect other information if you wish.
2. Ensure that all reviews are translated into English, clean the data, and process it. Concatenate the data from both stores but keep the two apps' data separate in two data-frames for further analysis. At the end you would have separate datasets, one for each App.

The following tasks are to be performed for each app separately, except for 6, 7, 8, and 12 which are common to all:

1. Perform the sentiment analysis of the reviews and try to classify each review as either positive (1), negative (-1), or neutral (0). You can use sentiment Vader (https://github.com/cjhutto/vaderSentiment) and use the compound results to make the classification by specifying thresholds. Add the sentiment results to a distinct column in your datasets (data-frames).

* **positive sentiment**: compound score >= 0.05
* **neutral sentiment**: (compound score > -0.05) and (compound score < 0.05)
* **negative sentiment**: compound score <= -0.05

1. Consider making two plots, one for ratings and another for sentiments over time. Use the date about new versions and new releases. Scatter these dates across each plot so you can observe the effect of the new version on the sentiment or rating.
2. We want to find the most discussed topics from users, and for that, perform LDA topic modeling and try to generate 10 topics.
3. Train a machine learning model (random forest or another one) with positive and negative sentiment reviews. Try to use different n-gram representations such as n-gram (2,3) or (3,4). Then perform feature selection to identify the most important words or n-gram elements that impacted the classification for positive and negative classes. Try to specify the class for each word or element you retrieve.
4. The next task is about the use of technology acceptance models (TAM) to assess how people respond to technology adoption decisions. For instance, we try to measure the level of satisfaction, perceived ease of use, perceived usefulness, and attitudes toward the technology.

Try to learn more about TAM from this review paper [1]. We are interested in understanding how the indicators: **perceived ease of use, perceived usefulness, satisfaction, attitude, and behavioral intentions** change over time. You can check this paper [2] as well for more information about some of the TAM indicators.

* 1. Generate keyword lists for each indicator based on your understanding of the topic. For instance: some keywords for **satisfaction** are {Satisfied, fulfill, gratify, meet, beneficial, content, happy, appeasement ……. etc.}.
  2. To augment the list of keywords, write a script that finds synonyms, hypernyms, and hyponyms for each word, by using WordNet database. Review the words manually and remove those which are not relevant.
  3. Perform necessary data processing for the list of keywords and reviews, for instance, Part of Speech tagging and lemmatization. Then classify each review and its data (rating, sentiment, and date) to a particular TAM indicator based on the common words. In the end, it is expected to have 5 data-frames, each one refers to data related to one TAM indicator.
  4. Repeat task 4 for each indicator.
  5. Calculate the Pearson correlation with a its associated p-value between different indicators and determine which are most strongly correlated. Highlight the correlations you find. Provide table to summarize your work.
  6. If the rating is provided, compute the correlation between the sentiment and rating. Repeat this process when you separate between positive and negative sentiment.

1. Identify relevant literature on parking behavior, TAMs, and other associated topics to support and discuss what you have found in previous sections.

Please contact the assistant, Nabil ([nabil.arhab@oulu.fi](mailto:nabil.arhab@oulu.fi)) for more details about the project.

[1]: Koul, S., Eydgahi, A., 2017. A systematic review of technology adoption frameworks and their applications. Journal of technology management& innovation 12, 106–113.

[2]: Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, 319-340.

[3]: Yoon, H. Y. (2016). User acceptance of mobile library applications in academic libraries: an application of the technology acceptance model. The Journal of Academic Librarianship, 42(6), 687-693.

**Project 18 Medical image captioning**

VQA-Med 2019 dataset introduced a set of radiology images and four main related categories questions and answers about Modality, Plane, Organ system and Abnormality. Each request considers one element only (e.g. what is the organ principally shown in this MRI? in what plane is this mammograph taken? is this a t1 weighted, t2 weighted, or flair image? what is most alarming about this ultrasound?). Then answers are made from the image content without requiring additional medical knowledge or domain-specific inference. These Q/A pairs could be explored to generate automatic captioning for their underlined images. Automatic image captioning aims at generating natural captions and meaningful textual description automatically for images, which is of great significance in scene understanding. The dataset can be downloaded from https://github.com/abachaa/VQA-Med-2019.

1. Download the dataset and visualize 15 Q/A pairs of your choice from different categories (Modality, Plane, Organ system and Abnormality).
2. Use appropriate function to create tokens (with and without removing stop-words, and lemmatization).
3. Plot for each case, the word occurrence frequency curve after ranking the tokens. Check whether a power-law distribution can be fitted or not by plotting the log-log curve. Explain the results.
4. Now we would like to build a model for the Q/A system. Start with a simple string based matching process, imitating the example in [Simple Question Answering (QA) Systems That Use Text Similarity Detection in Python - KDnuggets](https://www.kdnuggets.com/2020/04/simple-question-answering-systems-text-similarity-python.html) that uses string matching and naives Bayes’ classifier. You may notice that the system is quite limited but constitutes a good start. You may use one example in the validation test of the dataset to find out the type of outcome generated.
5. Instead of using string matching, modify the script in 4) to use Tfidf vectorizer and other types of classifiers (random forest, decision tree). Test the program on the same test query you used in 4). Discuss the limitations and how you may improve the results.
6. Use FastText embedding followed by a PCA (principal component analysis, for number components =5) and a random forest algorithm. Test the program on the same test query you used in 4). Discuss the limitations and how you may improve the results.

Consider the query: “In What plane is this mammograph taken? Which part of the body does this represent, which modality and plane was used to take it and what abnormality is it seen in this image?”

1. Create tokens after preprocessing the query (removing stop-words and lemmatization).
2. Use the image id to refer to each Q/A pair, combine all categories to construct one line description for each image. Then create a matrix representation using the Boolean model and find the closest image to the query.
3. Construct the tf-idf matrix representation of all Q/A pairs. Compute the similarity between the query and the images descriptions using different metrics.
4. Expand the query by replacing the words with their synonyms (then using its Pos tagging when extracting the synonyms).
5. Calculate the semantic similarity between new generated synonyms and the old tokens using one of the various available word-to-word semantic similarities.
6. Use the word2vec based similarity assuming that the vector associated to the whole new query corresponds to the average of the word2vec outputted vector associated to each token of the sentence, and then use the cosine similarity to compute the sentence-to-sentence similarity score.
7. Use countVector or tf-idf to represent the new query and calculate the closest image description to the query.
8. Identify appropriate literature in the field of medical image captioning to provide reasonable findings of the results in the previous steps.

**Project 19: Natural Language for Visual Reasoning** **NLVR**

The Natural Language for Visual Reasoning corpora are two language grounding datasets containing natural language sentences grounded in images. The task is to determine whether a sentence is true about a visual input. The data was collected through crowdsourcings, and solving the task requires reasoning about sets of objects, comparisons, and spatial relations. This includes two corpora: NLVR, with synthetically generated images, and NLVR2, which includes natural photographs. This project requires two special treats (Natural language processing and computer vision). The final model of this project will take two modalities as input to decide whether a given text reflects a given image (binary classification problem). Therefore, this project's primary objective is to make a visual reasoning model between given inputs (text, image) examine whether the text content is logically equivalent to the image content.

You can download the datasets from <https://lil.nlp.cornell.edu/nlvr/>

NLVR1: includes 92,244 pairs of English sentences grounded in synthetic generated images. This dataset can be used for semantic parsing.

1. Download NLVR dataset in your case NLVR1.
2. The datasets’ the textual data need some preprossessing, use the appropriate NLTK functions to do the following:

* Replace numbers to words '1' becomes 'one'.
* Correct spelling mistakes 'tleast' becomes 'at least'.
* Convert word from plural to singular form.
* Remove punctuation.
* Remove extra spaces text.strip().
* Make a final check to the entire dataset’s vocabulary using the BOW model (countvectorizer). Note: the vocabulary should not include words with similar roots such as {'blocks', 'block', 'items', 'item', …}. Its size should be around 150 different words.

All the dataset’s images are represented in the form of structured representation (SR). Structured representation corresponds to three regions of a synthetic image where each region have a maximum of eight objects, and each object is defined (x, y, color, size)

1. Encode both structured representation (SR) and the sentence associated with it using word2vec.
2. Build a deep learning model that takes as input encoded SR to generate a sequence that corresponds to the encoded sentence.
3. Build another deep model that can predict if the two inputs (SR’s generated sequence of the image and its sentence) are logically equivalent (True or False)/similary problem.
4. Merge the two models in one model.
5. Evaluate the final performance and report the accuracy (binary classification problem)

**More detailed will be provided once you start working on the project.**

Contact teaching assistant at [Yazid.bounab@oulu.fi](mailto:Yazid.bounab@oulu.fi)

**Project 20: Fake News Detection**

This project aims to design a system for Automatic Fake News Identification. The basis for fake news detection in stream of like news threads, where a news has a title and its associated body text consists in assessing the consistency between the title and message content.

1. Initially, start by studying the Fake News Challenge Initiative and first competition FNC-1, available at [Fake News Challenge](http://www.fakenewschallenge.org/). In the latter, four classes of association between title and message content of the news document have been set up: Agrees (body text agrees with headline), Disagrees (body text does not agree with headline), Discusses (body text discuss the same topic as the headline but does not take a position), Unrelated (body text discusses different topic than headline). The competition includes both a training and testing dataset. You may also notice that the top three participants used deep learning and neural network architectures and have maintained active GitHub account with all the source code made available, so that you may chose one to test and visualize the results.
2. Consider modifying the deep learning architecture (e.g., to include CNN and attention layer or any other architecture of your choice), write down the corresponding script and test it using the same dataset to record its performance in terms of accuracy levels on each of the four classes.
3. We shall consider building a simple rule-based approach that uses NLP components to assess the occurrence of any of the four classes. For this purpose, suggest a script that implements the following tasks. First, after initial preprocessing and stopword removal task, use a simple string matching to see the proportion of headline tokens which are present in the body text. We therefore assume that if this proportion is beyond certain threshold than the news can be Agrees, Disagrees or Discusses class. To find out what is the threshold value to use, you may run this reasoning on the testing dataset of the competition challenge. While, if the proportion is below the threshold, then the News is classified as “Unrelated”. Second, modify the list of stopword and preprocessing such that negation cannot be ignored (e.g., exclude words no, not, none, less, etc., from list of stop words). Suggest a script that implements this preprocessing and assumes the News to be in class Agrees if all tokens of headline are in body text; in Class Disagree if all tokens of headline are in body text and there is presence of negation in body text; Class Discusses if only part of headline token (beyond the established threshold) which are in body text. Test this heuristic on the testing dataset of Fake News Challenge and report the accuracy result for each class.
4. Now we want to modify slightly the heuristic in 3) by taking into account semantically similar wording. For this purpose, we shall consider the main part-of-speech tags in the headline. For this purpose, use part-of-speech tagger of your choice (e.g., Spacy parser) to identify the part-of-speech tags of both headline and body text document. Then write a script that uses wordnet lexical database to identify synonym of nouns (excluding named-entities), verb and adverb/adjective categories of the headline. Then modify the string matching so that a news is considered in class “Agrees” if i) all named-entities in the headline are present in the body text, ii) for each noun, verb, adjective/adverb category of the headline, either the same token occurs in the body or its synonym word occurs in the body text, iii) there is preservation of negation (if negation is present in headline, it should be present in body text, if headline has no negation, then body text should not contain any negation as well); It is class “Disagrees” if the last requirement (negation preservation) is violated; It is class “Discusses” if there the proportion of matching tokens using i) and ii) is beyond the defined threshold; Otherwise, it is class “Unrelated”. Write a script to test this new heuristic and report the accuracy result of Fake News Dataset Challenge on each of the four categories.
5. We want to test the above construction on Slovak News related to Corona virus and economic related discussions. For this purpose, consider 100 News in each case (Covid-19 and economic) and manually label the dataset in one of the four categories (Agrees, Disagrees, Discusses and Unrelated) according to your understanding of the content of the article. Then create an 80%, 20% split for training and testing, respectively.
6. First we want to test the transfer learning capability of the neural model developed in 1) and 2) to these new datasets. For this purpose run the same model on the testing data of Covid-19 and Economic and visualize the output and record the result in terms of accuracy levels of each of the four categories.
7. Retrain the models in 6) using the training data of Corona to test on Corona testing data and on Economic for testing on the corresponding testing dataset as well and record the performance for each case.
8. We also want to test the heuristic 3) and 4) on Corona and Economic dataset. Suggest a script that allows you to output the performance of class accuracy on each dataset accordingly.
9. Identify appropriate literature to comment on the technical soundness of the adopted approach, results and limitations.

**Project 21. Historical wordEmbedding 1**

This project aims to explore the distribution of selected keywords in historical corpuses to provide insights in terms of their temporal dynamic with respect to their contextual meaning.

* 1. Consider a book database of your choice, e.g., ISBNdb database [| ISBNdb](https://isbndb.com/apidocs/v2), use the API or any web crawling of your choice to collect book titles, spanning in the period 1900 till 2020. Use the ranges: 1900-1940; 1941-1960; 1961-1980; 1981-2000; 2001-2010; 2011-2020. Use the keyword “violence” as a guidelines for the search. Retrieve 100 titles (for each period of time), provided there are enough outputs. Save the result in a database file. We shall refer to the dataset corresponding to each time period as a corpus.
  2. Use wordcloud representation to illustrate the content of each corpus. See example in [Word Cloud: A Text Visualization tool | by Sawan Rai | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/word-cloud-a-text-visualization-tool-fb7348fbf502)
  3. Suggest a script that allows to check whether there are common results (book titles) among the various corpuses.
  4. Write a script that allows you to draw the histogram showing the 20 most common words of each time period, excluding stopwords. You may inspire from online NLTK book
  5. Now use topic modelling using Latent Dirichlet Allocation (LDA) as implemented in Gensim library for each corpus of titles corresponding to a given period of time. Use number of topics=5 and number of keywords per topic = 5. Use appropriate visualization of your choice to illustrate the overlapping among the topics at different time period.
  6. Use word2vec model as implemented in Genism (pretrained model) to illustrate the closeness of word “violence” with words of each corpus. For this purpose, we want to compare the embedding of “violence” and that of the main words in each corpus (excluding stopwords) and then perform dimension reduction to illustrate in 2D space. For this purpose, use the t-SNE representation. See an example in [Visualizing Word Vectors with t-SNE | Kaggle](https://www.kaggle.com/code/jeffd23/visualizing-word-vectors-with-t-sne/notebook).
  7. Repeat 4), when the word2vec model is retrained using each corpus. See again the example in [Visualizing Word Vectors with t-SNE | Kaggle](https://www.kaggle.com/code/jeffd23/visualizing-word-vectors-with-t-sne/notebook). Use word2vec model to visualize the five words that are more close to “violence” in each corpus.
  8. Repeat 4) when using Glove embedding instead of word2vec and then use PCA for dimension reduction. You may inspire from example of [Visualizing Word Embedding with PCA and t-SNE | by Ruben Winastwan | Towards Data Science](https://towardsdatascience.com/visualizing-word-embedding-with-pca-and-t-sne-961a692509f5).
  9. We want to assess the sentiment of each corpus. For this purpose, write a script that uses Vader sentiment analyzer to assess the sentiment of each book title in each corpus and then draw a graphical representation in a 2D graph where x-axis corresponds to the positive sentiment value and y-axis to the negative sentiment value, so that each book title’s sentiment is represented as a dot in graph according to the strength of positive and negative sentiment outputted by Vader sentiment analyzer. Provide graphical illustration for each corpus.
  10. We want how “violence” and “peace” are represented in book titles. For this purpose. Use the pretrained word2vec to calculate the embedding of “violence”, “peace” and each book title in each corpus. For each book’s title at each corpus, calculate the distance from “violence” to book title and from “piece” to book title using cosine similarity of the corresponding embedding vectors. In a 2D graph where the x-axis corresponds to violence intensity and y-axis corresponds to peace intensity, so that each book title is represented as a dot in this graph. Show graphs for each corpus.
  11. Use appropriate literature to show the motivate the results obtained in the preceding. Comment on the limitations of the approach employed in this project.

**Project 22. Historical wordEmbedding 2**

This project aims to explore the distribution of selected keywords in historical corpuses to provide insights in terms of their temporal dynamic with respect to their contextual meaning.

Consider the popular Harry potter movie whose dataset is available in [Harry Potter Dataset | Kaggle](https://www.kaggle.com/datasets/gulsahdemiryurek/harry-potter-dataset), indicating the content of the three movie distinguishing the script of each character of the movies. You may also find in Kaggle some code for data handling and visualization.

* + 1. Write a script that calculates the length of total script of each character and draws the histogram showing the length of script of each character.
    2. Consider the list of personal pronouns and Bing Liu opinion sentiment available at <http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Write a script that counts the total number of personal pronouns in each characters’s scripts, and draw histogram. Repeat this process for both positive and negative sentiment word according to Bing Liu lexicon, and draw the corresponding histogram.
    3. Consider the corpus of each character (concatenation of all scripts of each character). Perform the LDA for topic modelling with Number of Topics = 5 and Number of Keywords per Topic = 5. Visualize the outcomes of the topic modelling according to the method of your choice in such a way to seek the overlapping among the various characters.
    4. We want to compare the different characters in terms of the characteristics of their associated corpus. For this purpose, write a script that i) calculates the size of vocabulary used by each corpus (without any preprocessing); ii) lists the words are mentioned only once in the corpus (no repetitions); iii) identify the thirty most frequent words.
    5. We want to use the result of 4) to build a similarity measure among the different characters. For this purpose, write a script that calculates the similarity between two characters as the proportion of common vocabulary between the two. Draw the corresponding character-to-character matrix.
    6. Repeat 5) when the similarity is understood as the proportion of unique words in the corpus of both characters (words that have no repetition in the corpus).
    7. Construct a global vocabulary by integrating the vocabulary of each character. Write a script using tf-idf vectorizer of Gensim to provide a vector representation of each corpus and then use the cosine similarity to calculate the similarity between two characters. Write down the new similarity matrix.
    8. Use SentiWordnet (see link at previous projects), to calculate the sentiment score (positive and negative sentiment score) of each individual script of each character. Represent in a 2D space (x-axis corresponds to positive sentiment score and y-axis for negative sentiment score the sentiment of each individual script as a dot in this graph (you may use same color to represent the same character).
    9. Use pretrained word2vec model available in Gensim to represent each character’s script. Use the average operation to infer the word2vec embedding of every corpus. Then use cosine similarity to calculate the similarity of every pair of characters. Draw a matrix showing the similarity among the various characters.
    10. We want to use Word2vec model trained on historical corpus. Consider the histwords, available at [HistWords: Word Embeddings for Historical Text (stanford.edu)](https://nlp.stanford.edu/projects/histwords/). Repeat 5) when you use the historical word2vec embedding.
    11. Use appropriate literature to motivate the findings obtained at the manipulations

**Project 23. Personality trait**

This project aims to explore the distribution of selected keywords in historical corpuses to provide insights in terms of their temporal dynamic with respect to their contextual meaning.

Consider the popular Harry potter movie whose dataset is available in [Harry Potter Dataset | Kaggle](https://www.kaggle.com/datasets/gulsahdemiryurek/harry-potter-dataset), indicating the content of the three movie distinguishing the script of each character of the movies. You may also find in Kaggle some code for data handling and visualization.

* + 1. Write a script that gathers and concatenates the scripts corresponding to each character to make up a separate corpus for each character, and determine the characteristics of each corpus in terms of number of tokens, size of vocabulary, average length of individual script, standard deviation of script length, proportion of stopwords employed. Provide the results in a table to visualize the discrepancy among the various characters.
    2. We want to compare the various characters in terms of their personality traits, especially, the big-five personality traits (Openness, Consciousness, Extraversion, Agreeableness and Neuroticism). For this purpose, we rely on some existing implementations that infer from the text the corresponding personality trait. We may refer to some existing implementations in <https://github.com/topics/personality-traits>. For instance, you may use popular MELD or SenticNet. Study two of the provided implementations, and suggest a script that input the whole corpus of a given character and then outputs the scores assigned to each personality trait (a vector of five dimension). Use PCA to reduce the dimension of the outputted file into 2D dimension and then provide a graphical illustration (2D space) where each character is represented as a doc. Identify the characters that are close to each other in terms of personality traits.
    3. Repeat the 2) when using the cosine similarity of the five-personality vector scores of each pair of characters as metric to quantify how the two characters are close in terms of their personality. Draw the corresponding character matrix similarity.
    4. Repeat 2-3) when using the alternative implementation (e.g., SenticNet, MELD,..).
    5. Now we want to track the personality of trait of a given character at a local level, not after aggregating all its scripts. For this purpose, write a script that, for each script, it generates the corresponding five personality vector, and saves it in a database file.
    6. Now we want to evaluate the consistency of each character in terms of personality generated. For this purpose, assume that a given character is assigned the personality trait corresponding to the highest personality trait score. We evaluate the consistency of the character’s personality by the extent to which the dominating personality trait is invariant across the different scripts. Suggest an expression and a script that allows you to quantify this intuition, and compare the characters accordingly.
    7. Consider the sentiment of every script of each character according to Vader sentiment analyzer. We want to test whether a given personality trait is associated with a given sentiment polarity. Write a script that calculates the overall sentiment score (arithmetic sum of positive and negative sentiment score) of each script and associate this with the personality trait that corresponds to the highest personality trait score. For each character, generate a histogram that shows for each personality trait, the proportion of positive sentiment and proportion of negative sentiment associated to each personality trait that occurred in character’s script.
    8. Identify relevant literature to justify the findings and comment on the limitations of the approach.

**Project 24. Recommender Systems**

This project aims to test various recommendation systems strategies utilizing collaborative filtering, item-user collaborative filtering and content-based filtering. Consider the Deskdrop dataset, which contains a set of shared articles (shared\_articles.csv) and a set of interaction modes (users\_interactions.csv), which include View, Like, Comment, Follow and Bookmark. The file can be accessed at [Recommender Systems in Python 101 | Kaggle](https://www.kaggle.com/code/gspmoreira/recommender-systems-in-python-101), where some initial coding for data handling is also available. In this case, the items correspond to the articles and the user rating can be created by utilizing the information about View, Like, Comment, Follow and Bookmark.

* + - 1. Instead of using the interpretation provided in the source code of the provided link, we shall interpret the ranking as follows: Ranking = 5 if there is a Like regardless the presence of other attributes. Ranking =4 if there is a Follow and Bookmark. Ranking =3 if there is Follow or Bookmark but not both of them. Ranking =2 is there is Comment and not (Bookmark or Follow or Like). Ranking =1 if there is View only. Write down a script that generates the user-item rating according to the previous construction. We shall consider only those users who rated at least three articles and ignore other users. Similarly, we shall consider only articles, which have been rated by at least two users. Finally, in order to evaluate the performance of various algorithms, perform a random 80%-20% split of original dataset between training and testing, respectively.
      2. Study the metrics Recall@N, which evaluates whether the interacted item is among the top N items in the ranked list of 101 recommendations for a user. See, also the NDCG@N, which takes into account the rank of the relevant item in the ranked list. The provided link [Recommender Systems in Python 101 | Kaggle](https://www.kaggle.com/code/gspmoreira/recommender-systems-in-python-101) already include exemplifications of these. Inspire from the provided code to deliver an item related collaborative filtering implementation. Test the performance of the recommender system in terms of Recall@N and NDCG@N evaluations.
      3. Repeat 2) when using user-related collaborative filtering algorithm.
      4. Now we want to use the content of the comment section in the Deskdrop file as a guidelines for the recommendation. For this purpose, we first interpret the likeness and unlikeness using sentiment analysis. Use Vader sentiment analyzer to infer the aggregate sentiment score of each comment. Consider the vector carrying the ranting values and the vector carrying the corresponding sentiment score. Calculate Pearson correlation to find out whether the rankings are positively correlated with the sentiment scores.
      5. Consider the length of the comment text in terms of number of tokens it contains as an indicator variable. Write a script that generates this indicator variable and calculate Pearson correlation between rating vector and length indicator variable.
      6. Consider another indicator, which consists of the proportion of the stopwords and uncommon characters in each comment text. Write a script that calculates this indicator variable and calculates its Pearson correlation with rating scores.
      7. Study the implementation in [Recommender Systems in Python 101 | Kaggle](https://www.kaggle.com/code/gspmoreira/recommender-systems-in-python-101) where the content based filtering is built using tf-idf vector of the message content, we want to replace the tf-idf by a simple sentiment analyzer result using vader. Suggest a script that implements this paradigm and perform its evaluation using Recall@N and NDCG@N metrics.
      8. repeat 7) when the content of message is constituted by a vector corresponding to the concatenation of [Positive Sentiment score, Negative Sentiment Score, Length of the message, Ratio of stopwords /uncommon characters, Number of Personal Pronouns].
      9. identify appropriate literature that comments on the findings in previous steps and discuss the limitation of overall approach.

**Project 25: Sentence Paraphrasing**

This project aims to investigate the capability of various NLP techniques to identify paraphrasing.

Consider the popular Microsoft paraphrasing dataset, available at [Download Microsoft Research Paraphrase Corpus from Official Microsoft Download Center](https://www.microsoft.com/en-us/download/details.aspx?id=52398). This consists of 5800 pair of sentences along with human annotation indicating whether each pair corresponds to a paraphrase/semantic equivalence relation, or not.

* + - * 1. In order to test various approaches for paraphrasing interpretation, we utilize a random 80-20 split of the original dataset in term of training, testing ratio. Write a script that performs this operation.
        2. We shall consider that a pair of sentence is considered a paraphrase if they share a certain ratio of string matching. For this purpose, use the FuzzyWazzy implementation of edit distance (see lecture handout), so that if the FuzzyWazzy score is beyond some threshold, then the corresponding pair is considered paraphrase. Use the training sample to estimate the threshold (through some regression method or any other method of your choice), and perform the testing on the testing sample. Report the results in terms of accuracy score accordingly.
        3. Repeat 2) when various types of preprocessing were employed (stopwords removal, stemming, removing of abbreviations…) Feel free to capitalize according to your own thoughts and interpretation of the sentences.
        4. Use the sentence-to-sentence similarity measure using the Wu and Palma semantic similarity as exemplified in the laboratory session. Similarly to the preceding, we assume that paraphrasing occurs if the sentence-to-sentence similarity score is beyond certain threshold. Use the training dataset to estimate the threshold and perform the testing in the remaining sample.
        5. Use pretrained FastText embedding [Word vectors for 157 languages · fastText](https://fasttext.cc/docs/en/crawl-vectors.html) to generate the embedding vector of each sentence (as the average of the embedding vector of its individual wording), and then use cosine similarity to estimate the similarity between the two sentences. Again assume that if the similarity score is beyond some threshold then the pair is considered as paraphrase, otherwise, not. Use the training sample to estimate the value of this threshold, then estimate the accuracy score on the test sample.
        6. Repeat 5) when using BERT embedding [bert-embedding · PyPI](https://pypi.org/project/bert-embedding/).
        7. We want to build a classifier for performing this operation. Suggest an approach how you can use Tf-Idf for feature space, Support Vector Machine as a classifier in order to use the existing labelling and estimate the accuracy on the testing dataset.
        8. Identify appropriate literature to comment on the findings and discuss the viability of the overall approach for paraphrase detection.

**Project 26: Time series Analysis of Nepali Opinionated Text after Nepal Quake 2015**

This project aims to explore Nepali Quake 2015 from Nepali opinionated social media text. Identify at least two relevant blogs that accumulates data on Nepali language over a period of time. Special interest can focus on columns published on the different online Nepali news media portals after

the Nepal Quake highlighting different phases of event occurrence (damage caused, relief, frustration, mobilization, reconstruction) in a way to track peers, public and institutional trust. As Nepali is one of the low resource language, carefully select resources that contain subjective text related to Nepal earth quake 2015 (published within 6 months after the Nepal Quake) from news portals, and make sure to have two separate dataset types. One is more generic and contain any relevant social media data, while the other concerns only data collected from government agencies. Save the data in two separate database files of your choice. The dataset should include the timestamp to track the chronological order of the data.

1. Consider each dataset as a single corpus. Write a script that displays in log scale, the variation of the frequency of the tokens and their ranking. Tests the fitting of the power law distribution for each corpus. You may consult previous projects on power-law fitting. You may also consult, example of program in [Sentimental Analysis for the Nepali Language | by Akash Shrestha | Medium](https://medium.com/@aakashshrestha/sentimental-analysis-for-nepali-language-ed6dc5a09159) for Nepali tokenization.
2. Suggest a subdivision of the timestamp to highlight a certain number of time periods (should be the same for both dataset), and concatenate data that belong to the same period of time together, yielding corpus Ci (and Di) (i=1 to n, where n is the number time periods) for first and second dataset. Write a script that uses the translation API (see general notes) to translate the Nepali text into English. Then use Vader sentiment analyzer to calculate the overall sentiment of Ci and Di. Calculate Pearson correlation and the associated p-value to estimate the correlation between the sentiment issued from general public and that from public institutions.
3. We want to test the sentiment with respect to reconstruction effort only. For this purpose, use wordnet lexical database and any manual crafting to identify all keywords that are linked to reconstruction. Write a script that retrieves sentences associated to reconstruction effort. Then repeat 2) by calculating the sentiment associated to each time period and calculate the Pearson correlation accordingly.
4. We want to compare the various time periods in terms of word frequency. Write a scrip that identifies the 30 most frequent tokens in each time period, excluding the stopwords. Write a histogram showing the amount of overlapping between the two datasets for each time period.
5. Repeat 3), when focus on named-entities. For this purpose, you may use Spicy to identify the named-entities at each time period and draw the histogram accordingly.
6. With the help of manual checking, identify a set of named-entities that are associated to governmental agencies. Write a script that collects phrases where these named-entities are mentioned. Then for each time period, calculate the sentiment polarity using Vader-sentiment analyzer, to assess the public perception about governmental agencies.
7. Now we want to compare the various time periods in terms of topics contained in the dataset. For this purpose, use LDA implementation in Gensim of topic modelling with Number of topics =5 and Number of Keywords per topic =5. Suggest an expression that uses Jaccard index to estimate topic overlapping between the two datasets at each time period.
8. We want to determine the sentiment polarity using machine learning approach without recourse to API English translation. You may use Nepali sentiment lexicon from elsewhere and suggest use of typical binary SVM classifier on tf-idf features.
9. Suggest an approach to compare the sentiment polarity result when using the previously mentioned Translation API and Vader, and the newly build machine learning based approach.
10. Identify appropriate literature to comment on the results and discuss the limitations and strength of the previously approach for mapping public opinion on the current topic.

**--- + PhD suggested projects**

**Resources for Finnish text analysis**

**Turku Dependency Treebank**

Turku Dependency Treebank (TDT) was created by Turku BioNLP Group. It was rst released as a stand-alone version, but it has been integrated into Universal Dependencies project in 2015 - <http://universaldependencies.org/>-. Turku Dependency Treebank is shared with CC BY-SA 4.0 license.

**Finnish stopwords**

There are several collections of Finnish stopwords available on the internet. One extensive list is provided by University of Neuchâtel on their website (<http://members.unine.ch/jacques.savoy/clef/>) . Stopword lists are usually used for removing regularly used words.

**FinnPos**

FinnPos (<https://github.com/mpsilfve/FinnPos>) is an open-source morphological tagging and lemmatization toolkit for Finnish. FinnPos is released under Apache Software License v2.0.

**FinnWordNet**

FinnWordNet (<http://www.ling.helsinki.fi/en/lt/research/finnwordnet/>) is a lexical database for Finnish. It is a semantic network, linked through relations such as synonymy and antonymy. The first version was a direct translation of Princeton's English WordNet. Current FinnWord-Net version 2.0 was released in 2012 under CC-BY 3.0 license and is also a subject to original Princeton WordNet license.

**FinTWOL**

FinTWOL is a morphological parser for Finnish ([www.lingsoft.fi](http://www.lingsoft.fi)). It is a product developed by Lingsoft.

**OMorFi**

OMorFi is an open-source morphological analyzer for Finnish language. It is freely available for download (<https://github.com/flammie/omorfi>) and is released under GNU GPL v3 license.

**Word list of modern Finnish**

The institute for the languages of Finland, Kotus, has created a word list of modern Finnish words in 2007, see (<http://kaino.kotus.fi/sanat/nykysuomi/>). The list contains 94110 Finnish words in their basic forms along with the in inflection types encoded in UTF-8 and stored in XML-form. The list is released under GNU LGPL (Lesser General Public License), EUPL v.1.1 and CC BY 3.0 ND. Kotus has also published a wide variety of other Finnish resources (<https://www.kotus.fi/aineistot>), that may be helpful when doing text mining in other research areas.

**Translation API**

For short texts and low usage, Google translation API can be used. However for large dataset, since Google API becomes not costly, an alternative is to use free translation API like

<http://py-googletrans.readthedocs.io/en/latest/>