List of Potential Projects Data and Text Mining Course 2025

LINK FOR PROJECT ALLOCATION -→
https://1drv.ms/w/s!AtcJs3OTsMZuiSApqWJaa2SDtz2Y

StopWord Behavior Analysis

Project 1: Stopword, wordnet and Corpus exploration 1

This project aims to explore the occurrence of stopword list in popular corpuses. First consider the default NLTK English stopword list, called, say, *L*. Stopwords are by default the most frequent words in corpus, and therefore, contributing less to the distinction among various documents. We want to test this hypothesis, by considering several corpuses.

1. Write a program that estimates the proportion of stopwords of L in the |L| most frequent words of the corpus (|L| is the number of words of L). Perform this operation for Brown Corpus, British National Corpus, Gutenburg, WebText and NPS_Chat, and Reuters corpora, which are mostly accessible using NLTK package. Illustrate the finding using a bar plot diagram and present the result in a table. Suggest a potentially new stopword list that shows consensus among most of corpora (most frequent words that appear almost in all corpora).
2. We want to explore the variation of PoS tags of L. Suggest a script that calculates the proportion of each PoS tag in L (you may restrict this operation to only 2 to 3 corpora), and calculates the proportion of these stopwords that have entry in WordNet. Next, for those words that have entry in WordNet, calculate the number of synsets. Present the result in a table, indicating the average, standard deviation, maximum and minimum number of synsets.
3. We want to quantify the semantic similarity of those words in L who have entry in WordNet. Suggest a script that infer WordNet derivationally related terms of non-noun category to calculate the Wu and Palmer semantic similarity of every pair of words. Then generate the average, standard deviation, maximum and minimum value of the semantic similarity values. Present the result in a table.
4. We want to reproduce specifications 2) and 3) for a 2 non-English languages. By noticing that NLTK already supports several other languages and where the default stopword lists are freely available. Select 2 other languages of your choice, and use of a part-of-speech of your choice to generate the average, standard deviation, maximum and minimum of pairs of words that have WordNet link.
5. Repeat 4) where instead of using local PoS tagger, we use open translation API (similar to Google translation) to translate foreign stopword list to English and then use English PoS

tagging to identify words that have WodNet entry, and then compute average semantic similarity measure accordingly. Present the result in a table.

6. Instead of WordNet semantic similarity, use word-embedding of each stopword word in L, then calculate the similarity between each pair of words using vector cosine similarity. Then calculate the overall similarity as the average of all pair cosine similarity measures, together with the associated standard deviation, minimum and maximum cosine similarity values. Perform this operation when the embedding is supported by word2vec, FastText and BERT-base embedding.
7. Repeat 6) for each of the other foreign language analyzed previously. You may use one of the multilingual models or any other model of your choice for word embedding.
8. Repeat 7) when using the English translation of the foreign stopword list and then similar word embedding pairwise cosine similarity is performed.
9. We would like to revisit the specification 1) and test other alternatives to frequency calculus. You may inspire from the paper in Sarica S, Luo J (2021) Stopwords in technical language processing. PLoS ONE 16(8): e0254937. Instead of frequency, test measures such as inverse-document frequency (IDF), term-frequency-inverse-document-frequency (TFIDF), entropy, information con tent, information gain and Kullback-Leibler divergence, where sentence of the corpus is treated as a single document. For each of these measures, test extent to which the stopword list is a correct representative of the |L| most highest scores according the corresponding measure. Use appropriate table to summarize the result accordingly. You may also look at implementation available in A universal information theoretic approach to the identification of stopwords.
10. Suggest an approach to implement quantification (1) in Sarica S, Luo J (2021)'s paper as basis for calculating the strength of the connection between the stopwords, where the count(wi wj) computes the count of wi appearing within a distance 2 (two tokens) of wi. Generate the score(wi wj) for every pair of stopword words in Brown Corpus, Gutenburg, WebText, NPS_Chat and Reuters corpora. Summarize the interconnection of these stopwords in an appropriate table, and use appropriate graphical illustration.
11. Use Vader sentiment analysis to analyze the sentiment of each sentence of Gutenburg, WebTxt and NPS_Chat and identify which the proportion of sentiment polarity for each stopword in each corpus. Summarize the findings using appropriate graphical illustration, and tables.
12. Identify appropriate literature to discuss your findings and comment on the limitation of the theoretical approach stated in these specifications.

Project 2: Stopword behavior and Corpus exploration 2

This project aims to explore the occurrence of stopword list in popular corpuses. We first consider the default English stopword list in NLTK, called, say, *L*.

1. Stopwords are by default the most frequent words in corpus, and therefore, contributing less to the distinction among various documents. We want to test this hypothesis, by considering several corpuses. Consider the default NLTK English list of stopwrds, say, L. We consider corpora where documents are categorized, for instance, Reuter and NPS_Chat corpora. Write a program that estimates the proportion of stopwords in each category of Reuter and NPS_Chat and present the result in a table, and relevant graphical illustration.
2. We want to test how the stopwords are important in information retrieval task. For this purpose, by assuming single post as an individual document, and by considering a separate database for each category (all posts assigned to the same category are considered as a single database), write a program that computes the TF-IDF matrix for each category: construct the vocabulary of each database by performing lemmatization but without stopword removal, and then computes TF-IDF of each stopword word. Repeat this process for each category, and compute the average, standard deviation, maximum and minimum TF-IDF value for each stopword across categories of the corpus. Repeat this process for both corpora and comment on the suitability of each stopword in retrieval task where an ideal scenario would be where the corresponding TF-IDF value is equal to zero.
3. Assuming that TF-IDF scores are used as a basis for identifying relevant stopword list, write a program that identifies the most appropriate stopword list, considering that words that achieve lowest score across the largest number of categories. Identify the 100 most suitable stopword list for each corpus (Reuter and NPS_Chat), and then estimate the proportion of overlapping with the default NLTK English.
4. Assume that any stopword word “S” is such that its associated TF-IDF score is zero in at least 50% of categories in the above two corpora, write a program that identifies the corresponding list of stopwords in each corpus, and the extent of overlapping with the default NLTK stopword list for English language.
5. Consider a foreign language of your choice whose NLTK parser and default stopword list are available, and then identify a classification dataset of your choice in that language (similar to Reuter dataset where data is split into a set of categories). Repeat the tasks in 3) and 4) for this dataset.
6. Consider the metrics δ and φ defined in expressions (5) and (6) in paper of Armano et al. “Stopwords identification by means characteristics and discriminants analysis.”, available at

8221ee7a0ce4ee8c8ddfe9d1b6b5fb29dd2c.pdf, where the class "C" corresponds to category and term "t" to any word of the corpus. Stopwords are those who score high positive values in φ and very small values in $|\delta|$ (absolute value of δ). Use this reasoning to identify the 5 most appropriate stopwords and compute their associated precision with respect to NTLK default list (percentage of these five words that occur in NLTK default list). Repeat this reasoning for the 10 most appropriate stopwords; for the 20 most appropriate stopwords; then for 30, 40, 50, 60, 70, 80, 90 and 100 most appropriate stopwords. Use appropriate illustration to show the nested evaluation of the δ-φ -based stopword construction, for each corpus.

7. Provide a two-dimension representation of the 50 most appropriate stopword identified in 6), in the scale δ versus $|\varphi|$, for each corpus.
8. Use k-means algorithm, e.g., implementation available in scikit-learn, either with a fixed number of clusters or adaptive number of clusters, to cluster these 30 stopwords into appropriate disjoint clusters. Illustrate the findings in a graphical plot for each corpus.
9. Repeat 8) when using the default NLTK stopword.
10. We want to compare the result of 8) when using alternative representations. For this purpose, consider the Word2vec representation of each stopword identified in 7). Then use t-SNE visualization for high dimensional vector (available in scikit-learn) to represent each stopword in a 2D representation, then use the same k-mean algorithm used in 8) to identify appropriate categorization of these stopwords. Comment on the comparison with clustering result of 8). Repeat this process for the default NLTK stopword list.
11. Repeat 10) when using FasText and Glove embedding.

Identify appropriate literature to discuss your findings and comment on the limitation of the theoretical approach stated in these specifications.

Project 3: Stopword behavior and Corpus exploration 3

This project aims to explore the occurrence of stopword list in Old English corpus and more recent English corpus. We consider the default English stopword list in NLTK, called, say, *L*. The project utilizes two datasets: Old English corpus, available in Listing folder '/download/ORACC/2017-09/vrt/' at www.kielipankki.fi, which is manually PoS tagged, and includes Corpus of Ancient Mesopotamian Scholarship; Digital Corpus of Cuneiform Lexical Texts; Royal Inscriptions of Babylonia online; Royal Inscriptions of the Neo-Assyrian Period and State Archives of Assyria Online. In addition, we also consider the BBC news RSS feeds dataset available in Kaggle BBC News. The dataset contains attributes (title, date, guild, link, description).

1. Write a program that extracts each of the Old English corpus: Corpus of Ancient Mesopotamian Scholarship; Digital Corpus of Cuneiform Lexical Texts; Royal Inscriptions of Babylonia online; Royal Inscriptions of the Neo-Assyrian Period and State Archives of Assyria Online and save it in an easy format for subsequent reasoning.
2. For each of the above corpora, write a script that calculates the total number of tokens, the proportion of tokens that entry in WordNet, and vocabulary size, using a stemming approach of your own. Summarize the result in a table.
3. Similarly, write a program that extracts the title and description attribute content of BBC News dataset and put them in a single document (without date, guild, and link attributes). Then summarize in a table the overall statistics of this newly constructed dataset in terms of total number of tokens, proportion of tokens that have WordNet entry and size of vocabulary.
4. We want to evaluate how the vocabulary size evolves with the number of tokens employed. For this purpose, write a script that evaluates the vocabulary size when only the first 5% of tokens were employed, then 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%, 75%, 80%, 85%, 90%, for each corpus of Old English dataset and for the newly constructed BBC news dataset. Illustrate this evolution in the graph and test whether Heaps' law can be fit to each corpus. Use appropriate statistical analysis to evaluate the significance of the fit. You may inspire from some 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.
5. We want to explore the power-of-law distribution of the word frequency and test whether the Zipf's law distribution can be accommodated to each corpus. For this purpose, use the log-log scale of the frequency of tokens and their ranking, and examine whether a linear fit can be accommodated, up to a given statistical significance level (e.g., 90% confidence bound). Draw the curve fitting for each corpus together with the interval bound levels, and comment on the Zipf's law distribution for each corpus.

6. Now we want to study the extent to which most frequent words can be considered as stopwords. For this purpose, for each corpus, write a program that identifies the 100 most frequent tokens in each corpus. Then use the Jaccard coefficient in python to quantify the amount of overlapping between the 100-most frequent tokens across corpora. This assumes
7. Repeat 6) when you we employ Jaccard coefficient to evaluate the amount of overlapping between the 100-frequent words and NLTK stopword list. Present the result of 6) and 7) in a 6x6 matrix.
8. Repeat 6) and 7) when the 100-most frequent words is replaced by the 100-smallest TF-IDF values (where each sentence in the corpus plays the role of a document in TF-IDF calculus).
9. Repeat 6) and 7) when the 100-most frequent tokens are replaced by the 100-tokens according to their information content values where the information of a token T is $I(T) = -\log_{10}(P(T))$.
10. Consider the implementation of an information-theoretic for stopwords' identification from a corpus, available in Gerlach et al.'s paper A universal information theoretic approach to the identification of stopwords | Nature Machine Intelligence. Use the available implementation to determine the stopwords list for each corpus, and then calculate the extent of their overlapping using Jaccard coefficient across corpus, and with respect to default NLTK list. Also calculate the TF-IDF and information content of each stopword identified by the Gerlach et al.'s paper. Suggest relevant metrics to evaluate the usefulness of the identified stopwords.
11. Take your own initiative and identify more state-of-the-art approaches for automatic stopwords identification from corpus, and discuss potential comparison with the methods implemented above.
12. Identify relevant literature to comment on the findings and discuss the results in relevant theoretical lenses.

Project 4: Corpus exploration 4

This project aims to explore the occurrence of specific entities in large corpus. We shall consider Brown corpus, Reuter and NPS_Chat corpora.

1. We want to track the occurrence of named-entities in the corpus. Use Spacy named-entity recognition package to identify named entities in Brown corpus. Draw a wordCloud representation of the set of named-entities identified by Spacy package. Next, in order to evaluate power-law fitting, write a program that extracts the named entities in the first 5% of tokens from the beginning, then in the 10% tokens from the beginning, then 15%, 20%, 25%, …, till 90% (at step of 5%), then plot a graph showing the number of named-entities versus the number of tokens employed. Use log-scale to find out whether Heaps law can be fitted with reasonable statistical significance (e.g., 90% confidence bounds). You may inspire from some 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.
2. Repeat 1) when considering the size of number named-entities versus vocabulary size for 5%, 10%, 15%, …, 90% of original tokens.
3. We want to comprehend the occurrence of named-entities throughout the corpus. For this purpose, suggest a script that computes, for each named-entity, the distance (in terms of number of tokens) from the current named-entity and the next one. Then suggest a discretization (e.g., 20 bins, by counting the shortest and longest distance), and plot a histogram showing the number of named-entities occurring in each bin.
4. Suggest a script that generates a curve that best passes through the various histograms. You may use any package in python that fits a curve to a histogram, available in statistics package or elsewhere, e.g., **scipy.optimize.curve_fit**`
5. We want to explore the length of the identified named-entities. Suggest a script that calculates the length of each named-entity in terms of number of characters. Provide the average, standard deviation, maximum and minimum value. Then the reasoning of 3-4) to construct a histogram plot on the lengths of named-entities, and then the best curve that fits the histogram.
6. Consider the modal verbs, which include, ‘must’, ‘shall’, ‘will’, ‘should’, ‘can’, ‘could’, ‘may’, ‘might’. Modal verbs translate the presence of concepts such as permission, necessity and possibility in the corpus. Write a script that calculates the frequency of each modal verb in the corpus. Next, we would like to study the occurrence of modal verbs throughout the corpus. For this purpose, for each occurrence of a modal verb, calculate the distance (in terms of number of tokens) to the next modal verb. Then similarly to 3), suggest a discretization of the distance range into bins and provide a histogram plot indicating the number of modal verbs in each bin. Then as in 4), suggest a curve that best fits the histogram.

7. Repeat 1-6) when using Reuter corpus
8. Repeat 1-6) when using NPS_Chat corpus.
9. Take your own initiative and identify one state-of-the-art approach for automatic analysis of corpus behavior that can easily be imitated for the three corpora under consideration and provide some qualitative and quantitative comparison with previous specifications.
10. Identify relevant literature to comment on the findings and discuss the results in relevant theoretical lenses.

Project 5: Zips Law distribution of Finnish Wikipedia Corpus.

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

1. Consider the 2018 Finnish Wikipedia dump articles available in https://www.kielipankki.fi/download/wikipedia-fi/wikipedia-fi-2017-src/, containing all Finnish Wikipedia articles available in January 2018. The dataset is provided with a set of linked entities, where each entity links to a given page. Write appropriate commands to download the dataset and create a dataframe of the resource in your NLP project.
2. Suggest a script that counts the number of linked entities and the number of tokens (including linked entities) in each page.
3. Now we want to study the asymptotic behavior of the total number of tokens per page. First, determine the minimum number of tokens (*min*) and the maximum number of tokens (*max*) per page observed over all pages, then construct a 20-bin subdivision from *min* to *max*. Suggest a script that draws the frequency of each bin with respect to their ranking and fit Zipf's law distribution with 90% confidence bounds, then calculate the R-square and adjusted R-square to evaluate the statistical significance of the fitting. See, e.g., r2_score — scikit-learn 1.7.2 documentation
4. Use appropriate function for fitting a curve to histogram to draw the best fitting curve to histogram generated in 3), using, e.g., scipy.optimize library.
5. Repeat 2) and 3) when the total number of tokens is substituted by the total number of linked tokens (clickable tokens).
6. We want to explore the evolution of number of linked entities with respect to total number of tokens. For this purpose, consider the first page (whole document pointed out by one entity link) of your choice and suggest a script that counts the total number of tokens I(1) and total number of linked entities C(1), then for page 1 and page 2 together, estimate I(2) and C(2), …, for page 1, page 2, .., page k (all together), estimate I(k) and C(k),…, until all (or most of) pages are covered. Then draw the evolution of C as function of I, and fit Heap's law to the data when the value of parameter beta and K of Heap's law are determined by the interpolation algorithm. Estimate also the R-square and adjusted R-square statistics, and draw the 90% confidence interval fit.
7. From the structure of Wikipedia articles, suggest a script that identifies linked entities which are associated to Location (*L*) and those associated to Organization (*P*). Write a script that counts the number of location entities and Organization entities in each page.
8. Write a script that counts the number of *L* in page 1, page 1 & 2, page 1, 2 &3, …, as a function of pages employed, and then study the fitting of Leaps' law as in previous case.

9. Write a script that counts the number of *P* in page 1, page 1 & 2, page 1, 2 &3, …, as a function of pages employed, and then study the fitting of Leaps' law as in previous case.
10. We want to study the behavior of *L* and *P* in terms of number of characters. Suggest a script that calculates the length of Location entities and Organization entities. Then plot a 10bins representation histogram (by taking into account the smallest and largest length) to count the frequency of each bin in case of *L* and *P*.
11. Suggest any additional input that would allow you to further elucidate any of the preceding, and 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 6: Zips Law distribution and Suomi24 Corpus.

This project investigates various facets of the Zips law and how it can be empirically tested on Finnish Suomi24 corpus.

Download the Suomi24 dataset from *The Suomi24 Corpus 2001-2017, VRT version 1.1 – META-SHARE (csc.fi).*
You should click on the access location ***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 (s) or cloud server account will be able to handle it and run basic search.

1. Select keyword "hate speech" (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. Write a script that query the whole dataset for the query terms associated to hate-of-speech (Query 1). Save the thread titles /contents together with their associated dates in a database file of your choice.
2. Repeat the process 1) for antonym word of hate-of-speech, e.g., friendly speech, civility, freedom of speech with their Finnish translations and related vocabulary (Query 2).
3. Suggest a script that draws on the same plot the yearly evolution of the number of threats associated to Query 1, the yearly evolution of the number of threats associated to Query 2, and the yearly evolution of the total number of threats that are associated to both Query 1 and Query 2.
4. We want to elaborate more in terms of the scope of outcome to Query 1 and Query 2. For this purpose, write a script that calculates the yearly evolution of the average size (in terms of number of tokens) of threats associated to Query 1 and that associated to Query 2. Display the result in plot.
5. Repeat specification 4) when considering the size of the vocabulary in the outcomes to Query 1 and Query 2.
6. Suggest a script that would allow you to test Heap's law for Query 1 outcome, and for Query 2 outcome data. Plot the corresponding graph and estimate the statistical significance in terms of R-squared and adjusted R-squared statistics (See previous projects for some pointers).
7. Suggest a script, which identifies, for each year (from 2001 till 2017), the most frequent 10 words, excluding stopwords, in outcome of Query 1 and outcome of Query 2 data and suggest appropriate graphical illustration that also shows their intensities (number of occurrences of each of these words).
8. Write a script that compiles only the titles of the threads in each year for both Query 1 outcome and Query 2 outcome, and draw a wordcloud representation for each year (use intelligent and compacted representations of the figures to illustrate the findings in minimal plots). You may inspire from implementation in https://github.com/amueller/word_cloud, among others.

9. We would like to test the evolution of sentiment in each Query result. For this purpose, you may restrict to the top ten threats in terms of number of tokens per each year in the output result of each Query and perform the following sentiment analysis. Utilise the lexicon available in https://github.com/Helsinki-NLP/SELF-FEIL with a focus on positive and negative categories and compute the overall positive sentiment and overall negative sentiment according to the sentiment score of lexicon words available in each thread, then average the result within [0,1] interval. Suggest a script that draws the yearly evolution of positive and negative sentiment of each query result.
10. Identify from your reported literature state of the art method that you can accommodate to achieve any of specification above. Motivate your answers and manipulations by appropriate literature citation.

Project 7: Zips Law distribution in food domain.

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

Download the Suomi24 dataset from *The Suomi24 Corpus 2001-2017, VRT version 1.1 – META-SHARE (csc.fi).* You should click on the access location ***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 (s) or cloud server account will be able to handle it and run basic search.

1. Consider the Fineli database of Finnish food maintained by THL, which includes over 4000 foods and their nutrition intake, see Frontpage - Fineli, for database access and detail. Write a script that search for food entries (Fineli) in Suomi24 and retrieve the threads and messages where these foods occurred together with the associated date of the threads. Now we want to investigate the occurrence of these foods over time. Write a script that gathers the foods occurred monthly and calculate the total nutrition intake by summing up the corresponding nutrition intake of each food according to Fineli database, and draw the graph of timely evolution (monthly) of the nutrition intake according to the mentioned food.
2. Write a script that gathers the thread titles associated with the identified foods in 1) in each month to form a single dataframe (for each month). Now we want to determine the sentiment polarity of each dataframe. For this purpose, use the AFINN model for sentiment analysis that employs simple lexicon-based approach, see implementation in GitHub - fnielsen/afinn: AFINN sentiment analysis in Python, to determine the overall monthly when inputting each dataframe. Draw the evolution of sentiment score and nutrition intake on the same plot, and calculate the Pearson correlation to evaluate the correlation between the two entities.
3. Repeat 2) when considering yearly evolution of sentiment and nutrition intake instead of monthly evolution. Comment on whether further refinement of the timely evolution can illustrate correlation between the two entities.
4. Now we want to explore the food diversity according to the records in 1). For this purpose, write a script that calculates the number of distinct foods in each dataframe (monthly), and then draw on the same plot the timely evolution of the food diversity and nutrition intake, and then calculates the Pearson correlation between the two entities. Repeat this process when yearly evolution is considered instead of monthly evolution.
5. The use of Fineli for quantifying food diversity may not be ideal, propose your own manual categorization based on your understand of the various types of foods mentioned in Fineli, and then re-process specification 4) accordingly.

6. Now we want to explore the asymptotic behavior of the vocabulary employed in 1). For this purpose, suggest a script that calculates the vocabulary size of thread titles gathered in month 1, in month 1 and 2, in month 1, 2 & 3, and so on. Next write a scripts that shows the evolution of the vocabulary size with respect to number of tokens in Month 1, Month 1 &2, Month 1, 2 & 3, and so on, and study the fitting of the Heap's law. Evaluate the goodness of fit using R-squared and adjusted R-squared statistics.
7. Write a script that computes the length of thread titles identified in 1) and suggest a 20 bin subdivision of the length size. Then construct the graph of the frequency of each bin with respect to the ranking, and study the fitting of Zipfs' law distribution. Provide the corresponding R-squared and adjusted R-squared statistics.
8. Identify from your reported literature state of the art method that you can accommodate to achieve any of specification above. Motivate your answers and manipulations by appropriate literature citation.

Project 8. Zipfs law distribution and hate-speech

This project investigates various facets of the Zips law and other corpus based analysis of hate-speech.

Download the Kaggle hate-speech dataset Hate Speech and Offensive Language Dataset. The dataset includes 2.5Mbytes of Twitter dataset, which is manually annotated by CrowdFlower into offensive_language, hate_speech, and None.

1. By constructing a dataframe of Twitter dataset assigned to the same category, suggest a script that outputs the vocabulary set of each category, the size of the vocabulary, the total number of tokens, the average number of tokens per post and its standard deviation, the average number of pronouns per post and the associated standard deviation, the ten most frequent tokens in each category, excluding the stopword list. Represent the statistical result in a clear table, presenting global statistical trend of each category in terms of number of tokens, size of vocabulary, average number and standard deviation of number of annotators involved, and the 30 most frequent terms outside the stopword list. Comment on the balance of the different categories, and the extent of overlapping among the 30 most frequent tokens (outside stopwords) in each category. Use wordcloud illustration to represent the content of each category using the above 30 most frequent words.
2. Next, use the tf-idf vectorizer of genism library to calculate the tf-idf score of each token of each category dataset, assuming the overall vocabulary that includes all categories. Then identify the 30-most relevant tokens for each category. Discuss how this list overlaps with the above 30-most frequent tokens.
3. We want to study the power-law distribution of Zipf's law of each category. Suggest a script to draw and evaluate the zipf's law fitting for the dataframe of each category, and compute the corresponding R^2 and adjusted R^2 statistics. Draw the zipf's law fitting in log-scale and the 90% confidence bounds, then comment on the significance of the fitting.
4. Suggest a script that determine the Penn-Treebank part-of-speech tag of each token in each dataframe, and then test whether a Zipfs law distribution can be fitted, when looking into the frequency of the corresponding PoS, using R^2 and adjusted R^2 statistics.
5. Now we want to test the evolution of vocabulary of each category. For this purpose, suggest a script to draw and test whether Heap's law can be fitted in each category dataframe by drawing the vocabulary size as a function of total number of tokens employed (You may use some incremental increase of the number tokens). Provide the corresponding R^2 and adjusted R^2 statistics, and draw the fitting curve and 90% confidence bounds.
6. Now we want to comprehend the similarity and differences among the categories. Using the vocabulary of each dataframe, calculate the Jaccard similarity between every pair of categories as the ratio of

common vocabulary size over overall vocabulary size of both categories. Summarize in a table the similarity among each pair in view of Jaccard similarity metric.

7. Use Vader sentiment analysis package https://github.com/cjhutto/vaderSentiment to computer the overall sentiment of each post, then draw a 10-bin histogram showing the proportion of posts whose overall sentiment score falls in a given bin. This yields four histogram distributions (one histogram distribution for each dataframe). Next, use the standard Euclidean histogram distance to compute the similarity between every pair of categories with respect to sentiment scores.
8. Use the empath categorization https://github.com/Ejhfast/empath-client to generate an embedding vector for each dataframe's category. Then the cosine similarity to compute the similarity of each pair of category, and summarize the result in a table to comment on discrimination power of the empath categories.
9. Repeat 8) when pretrained doc2vec embedding (https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e) was used instead of empath category. Similarly, repeat this process when using DistilBERT (distilbert/distilbert-base-uncased · Hugging Face) embedding (although there is a limit in terms of size of textual input, so you may consider embedding of posts only and then average over all posts of the same category. Draw a table that summarizes the key findings and discuss the extent to which embedding can be used to discriminate between the categories.
10. We want to evaluate the various categories in terms of linguistic quality of the posts. For this purpose, suggest a script to identify the number of unmatched tokens in each dataframe, for instance by seeking whether the token has an entry in Wordnet. You may also inspire by some existing implementation such as Spelling Correction Of The Text Data In Natural Language Processing | by Nutan | Medium. Estimate the number of spelling errors in each category and its proportion with respect to the total number of posts in the category.
11. We further want to explore the type of error found in 10) by identifying the cost of the correction. For this purpose, suggest a script that inspires from the link provided in 10) to utilize the pyspellchecker to output the closes correction, and then calculate the edit distance between the suggested correction and original faulty token. We therefore the proportion of errors at Edit distance 1, edit distance 2, edit distance 3, etc.. at each category. Suggest a bar box to provide a graphical illustration, Then summarize the result in a table and comment in terms of discrimination power.
12. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 9: Historical Finnish Word Embeddings from News

This project aims to investigate how Finnish words change their meaning over time by training and comparing word embeddings on news datasets. From Kielipankki (https://www.kielipankki.fi/aineistot/ylenews/) collect four news datasets: 2022–2024, 2021, 2019–2020 and 2011–2018.

1. Suggest a script that extracts the tokens and vocabulary of each dataset after stopword removal using the default NLTK list for Finnish language. Then use the Gensim TF-IDF vectorizer to generate the dictionary and TF-IDF score of each token, considering all datasets) as well as the 50 most frequent tokens (excluding Stopwords) and 50 tokens with highest TF-IDF scores.
2. Calculate the Jaccard similarity score to quantify the similarity between the datasets from the 50 most-frequent word perspectives and 50 most highest TF-IDF scores. Present the result in 4 x 4 matrix.
3. Present an illustration of the content of the most frequent words and highest TF-IDF scores using wordCloud illustrations for each dataset.
4. Now we want to track the yearly occurrent of the 5 most frequent terms and the 5 most highest TF-IDF scores. For this purpose, suggest a script that draws on the same plot the yearly evolution of the frequency of each of these five terms in each dataset.
5. Now we want to review the preceding when appropriate preprocessing is performed in order to take into account the various transformations of the same word. For this purpose use appropriate parser of your choice, e.g., Turku's Finnish parser, Multilangual parser, prosodic parser, among others, then repeat the vocabulary extraction and tasks of specifications 1, 2, 3 and 4.
6. Use the Gensim library to train separate Word2Vec models for each dataset.
7. Choose at least five target words (e.g., *tekoäly*, *ilmasto*, *politiikka*) and calculate the cosine similarity of their embedding vectors between different time periods. Normalize the similarity scores using a set of stable function words such as "*minä"*, "*ja"*.
8. Perform a neighborhood change analysis for the target words by comparing the nearest neighbors across models.
9. Use t-SNE or a similar method to display the neighborhoods of the target words in a 2D space.
10. Repeat task 5 for the list of frequent words obtained in task 2.
11. Rank most frequent and target words by the magnitude of their semantic shift across time periods and identify which words changed the most.
12. Provide a discussion of the results with concrete examples of target words and frequent words, including how their associations evolved in the Finnish news context. Comment on possible social, political or cultural reasons behind the observed changes.

Project 10: 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.

First you may start by a poem in one of available ebooks at http://www.gutenberg.org/. For instance, Beowulf. An Anglo-Saxon Epic Poem Beowulf: An Anglo-Saxon Epic Poem by J. Lesslie Hall | Project Gutenberg.

1. We want to study the differences and similarities among the various chapters of the book. For this purpose, suggest a script that uses NLTK library to identify the vocabulary of the whole book and then identify the vocabulary size of each chapter, the total number of tokens of each chapter, and then draw a bar plot of the ratio (vocabulary size)/ (total number of tokens), per chapter
2. We want to estimate the extent of overlapping between the different chapters. For this purpose, write a script that calculates, for any pair of chapters, the ratio between the size of the common vocabulary between the two chapters and the size of the overall vocabulary of the two chapters. Write the result in n x n matrix where n is the total number of chapters.
3. We want to evaluate the chapters' similarity in terms of topics. For this purpose, write a program that applies LDA with three topics and 8 words per topic for each chapter, then, for any pair of chapters, calculate the amount of overlapping between the corresponding LDA results, using existing approaches for LDA topic comparison. You may utilize some implementations available in How to Compare LDA Models — gensim (radimrehurek.com), and then compute an overall score. You may also raise your own initiative in comparing the result of the LDA models. Summarize the result in a n x n matrix as well.
4. Write a script that uses Empath categorization https://github.com/Ejhfast/empath-client for each chapter to generate an embedding vector, then, for any pair of chapters, calculate the similarity between the underlined empath as the cosine similarity between the corresponding embedding vectors. Summarize the result in a nxn matrix.
5. Write a script that uses the EmoTag1200 lexicon EmoTag/data/EmoTag1200-scores.csv at master · abushoeb/EmoTag · GitHub, where each lexicon term is assigned score for the eight motion state (anger, anticipation, disgust, fear, joy, sadness, surprise, trust.), to calculate the emotion vector of each chapter by taking the arithmetic average of the identified lexicon terms for each emotion state. Then, for any pair of chapters, calculate the emotion similarity by taking the cosine similarity of their associated emotion vectors. Summarize the result for all chapter pairs in a nxn matrix.
6. We want to investigate readability of the various chapters. For this purpose, consider four readability index: Gunning fog formula, Fry readability graph, Forecast formula and Flesch score, see definition and implementation available at Readability Index in Python(NLP) - GeeksforGeeks, among others. For each chapter generate a four dimension vector corresponding to the scores of the four readability indices. Then compute the readability similarity between the chapters and the cosine similarity between the corresponding vectors.
7. We want to compare the result of the previous similarity-based approaches for potential correlation. For this purpose, write a script that turns the matrix into vector representation, e.g., if the matrix has 3 components, say, A, B, C, then we use a vector representation containing the pairs [(A,B), (B,C), (A,C)], then calculate the Pearson correlation coefficient between all pairs of vectors generated in 2), 3), 4), 5) and 6). Summarize the result in a matrix and comment on the potential correlation among the various indicators used to compute the similarity.
8. We want to investigate how the most frequent terms occur in different chapters. For this purpose, identify the three frequent terms in the book, excluding stopwords and uncommon characters. Then for each of these three words, write a script that identifies the tokens occurring within 2-token interval around the target word, their part of speech-tag and their emotion vector (in case it matches one of EmoTag1200 lexicon. Suggest a plot that represents each of these words as multi-bar (for key part-of-

speech tags and emotion state over a fixed number of lines of poem) and where the x-axis stands for line, highlighting the different chapters.)

9. Now we want to explore further poesy-like attributes including rhyme, syllabus. For this purpose, study the open source Standford's Transhistorical Poetry Project available at poesy/README.md at master · quadrismegistus/poesy · GitHub. Install the package, and then write a script that outputs the rhyme of lines of the poem as a long sequence. Next, suggest a script that identifies the most frequent subsequences and their associated length (for ababcdcdefefgg, we can see that subsequence ab, cd, ef have frequency 2 for each and length 2)

10. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 11 Hate and Love in Poetry

This project aims to explore the Hate and Love poetry in Gutenberg. For this purpose, focus on the poetry books in http://www.gutenberg.org. The use your own methodology to identify list of poem books related to Hate and another list of poem books related to Love. Construct a dataframe for Hate's ebooks and another one for Love's ebook

1. Use appropriate NLTK coding (you can inspire from coding examples of the online NLTK book) to generate high level statistical description of each dataframe in terms of number of tokens, vocabulary size, number of ebooks. Present the result in a table.
2. Use the information about the date of publication to construct a histogram showing the number of publication of each category in each time period.
3. Now we want to comprehend the occurrence of hate and love wording in these poems. For this purpose, consider the list of terms, which are strongly connected to Love, and another list of words strongly connected to Hate. Use synonymy relations and other external dictionary of your choice to generate this list. Next, for each dataframe, suggest a script that quantifies the proportion of hate related words and love related words in each category and present the result in a histogram.
4. Next, we want to explore how the context of these words were employed in each dataframe. For this purpose, suggest a script that scans for each occurrence of hate or love related words appearance to retrieve the words that appear within 2-token distance on the left and right hand side of the target word. Finally identify the frequency of these words in each dataframe. Present the results as wordcloud illustration showing the context of hate occurrence and another one for Love occurrence in each dataframe. Comment on the interpretation of these results.
5. Next, we want to comprehend the sequential occurrence of Love related words and Hate related words in each dataframe. For this purpose, suggest a script that calculates the distance (in terms of number of

tokens) between one hate-related word to the immediate next one in the dataframe, and another one for computing the distance between love-related word and the next one. Use appropriate discretization (taking into account the difference between the maximum and minimum distance) to construct a 10-bin histogram showing the frequency of each distance range for both Hate and Love related terms.

6. Repeat the specification 5), when we consider the distance between a hate-related word and the next love-related term. Comment on the result regarding the possible asymmetry of the results, and how this can be linked to the context words identified in specification 4.
7. Use appropriate script to identify the 100-most frequent tokens of each dataframe outside the stopword list. We want to evaluate the extent to which hate and love related terms are related the most frequent terms. Suggest an approach to evaluate this proximity using both string matching and semantic similarity.
8. Repeat the specification 7) when working for each poem in dataframe Love and each poem of dataframe hate.
9. We want to test the diversity of each dataframe. For this purpose, we consider the titles of the poems in each dataframe. Suggest a script that uses FastText embedding to calculate the similarity between any pairs of titles in each dataframe using cosine similarity and assuming the embedding of the title is the average of the embedding of its constituted words. Calculate the average semantic similarity measures of all pairs and their associated standard deviation, maximum and minimum value.
10. 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. For this, you may select one poem at random from the first dataframe and another one from the second dataframe (You may also select the poem whose title brings the highest similarity with the hate/love related terms for instance, or any other reasoning). 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. Perform this process for both poems in each dataframe.
11. 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 12. Kaggle Text to Emotion Dataset

Consider the dataset in Emotion Dataset for Emotion Recognition Tasks where around 4MB Twitter datasets are classified into 6 categories: anger, fear, joy, love, sadness, surprise,.

1. Write a program to construct the dataframe of each emotion category as labelled in this dataset. Then write a script that identifies the vocabulary set of each dataframe, the proportion of unique tokens (that are not shared by any other category's vocabulary), the average number of tokens per post (with standard deviation), the average number of pronouns per post (with standard deviation), the average number of out-of-vocabulary tokens (no entry in wordnet lexical database). Summarize these findings in a table and comment on the key characteristic of each dataframe data.
2. Write a script to i) draw the wordcloud illustration of each dataframe; ii) identify the top 20 most frequent tokens, excluding stopwords, of each category; iii) construct a vocabulary of these top 20 frequent tokens over all categories, iv) estimate the Pearson correlation coefficient of every pair of categories (assuming the vector of each category is represented in the dimension of the 20 most frequent tokens vocabulary and whose value indicates the frequency of the corresponding token). Present the result in a matrix and comment on the relevance of token frequency to discriminate the categories.
3. Now consider the empath category https://github.com/Ejhfast/empath-client of the dataframe of each category. Write a script that computes the Pearson correlation of the vector embedding generated by the empath category between every pair of dataframe. Show the result in an 8x8 matrix representation, and comment on the ability of empath category to discriminate various emotion states.
4. Now we want to use word2vec as a basis to generate new vocabulary and evaluate the overlapping between the classes. For this purpose, write a script that generates the word2vec embedding of each of the emotion states (anger, fear, joy, sadness, surprise, love.), then applies the principal component analysis (PCA — scikit-learn 1.6.dev0 documentation) with a number of components equal to 2 to project each embedding vector onto a 2D space, then plot on 2D space all the 8 emotions states (where each emotion state is represented as a point in the 2D space). Comment on the proximity between the various motion states. You may discuss the suitability of the representation with respect to existing psychological models (see previous project specification for an example of a pointed on this issue).
5. Now we want to compare the above results with the generated vocabularies. For this purpose, suggest a script that uses word2vec to generate the thirty most close words to each of the six emotion names. Then use the Jaccard distance utilizing the size of the common wording (among the thirty words generated for each emotion name and the size of the all wording associated to either of the two categories) as a tool to quantify this proximity. Use your own approach to show the extent to which the ordering among the various pairs in 4) matches the ordering generated by the above constricted Jaccard distance.
6. Repeat 4) when using FastText (gensim: models.fasttext – FastText model (radimrehurek.com)), Glove (Gensim word vector visualization (stanford.edu)) and BERT Roberta (RoBERTa (huggingface.co)) Embedding, and comment on the findings.
7. We want to see if fine-tuning the model on local examples can increase the effectiveness. For this purpose, suggest a way to utilize the dataframe for each category to retrain the word2vec, FastText, Glove and Roberta model and reconstruct the 2D plot for each of the emotion state.
8. We want to assess the emotion prediction capability of machine learning algorithms. For this purpose, assume that 80% of data were used for training and 20% for testing, and using tf-idf features, and SVM classifier model, provide the precision, recall and F1-score. Repeat this process when stopword removal is applied. Compare the performances when count-vectorizer is employed instead of tf-idf.
9. Study the state-of-the-art DeepMoji emotion recognition implementation in https://github.com/huggingface/torchMoji And use the package to calculate the F1-score on the testing dataset. If the time permit, you may seek to retrain DeepMoji with the training dataset available in this project.
10. In Kaggle, there are many state-of-the art implementations on the same dataset where enhanced performances have been employed. Use your own exploration and suggest a new deep learning model

(can be ensemble of some already implemented state-of-the-art models) that have the prospect to improve the prediction performance and exhibit the results in terms of precision, recall and F1-score.

11. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 13. Emoticons generation Analysis

This project further explores the connection between emoticons and emotion text using some existing lexicons and embedding-based analysis. We particularly focus on the eight emotions: anger, anticipation, disgust, fear, joy, sadness, surprise, trust.

1. We consider emoTag1200 dataset EmoTag/data/EmoTag1200-scores.csv at master · abushoeb/EmoTag · GitHub where for each emoticon is assigned scores to each of the above eight emotions based on some manual annotation task. Identify a resource where each emoticon is assigned the corresponding text.
2. We want to test the compatibility between the emoticons' definitions and Tag1200 labeling. For this purpose, suggest a script that generates an Embedding vector (i.e., word2vec embedding) of the textual definition of a given emoticon. Then, use the same embedding for each for the eight emoticons, and calculate the cosine similarity between the embedding of the emoticon's textual definition and the embedding vector of each of the 8 emotion names. This yields an eight dimension vector quantifying the similarity to each of the emotions. Finally compute the Pearson correlation coefficient as a measure that evaluates the mapping between emoticon definition and Tag1200's labeling.
3. Repeat the process in 2) when different embedding strategies were employed, e.g., doc2vec, FastText, Glove, BERT. Summarize the result in appropriate table and Comment on the findings.
4. Consider the Twitter dataset available in Kaggle Tweets With Emoji (kaggle.com) where for each emoji there are around 20K tweets. Choose 6 emoji of your choice, containing antagonist emoji, e.g., happiness and sadness, with their associated 20K tweets. Suggest a script that determines for each emoji the 30 most frequent terms in their associated 20K tweet messages. Write down a script that generates a matrix showing the number of common wording among the thirty most frequent words identified.
5. Write a script that inputs the 20K tweet messages for each emoji and outputs the embedding vector using DistilBERT. Then use the available TSNE and its scikit-learn implementation TSNE — scikit-learn 1.5.2 documentation to project each embedding vector into a 2D space. Finally, draw in a 2D space the corresponding the result of each of the six emojis. Comment on the proximity between these emojis based on the expected intuition behind each emoiji.
6. Repeat 5) when the embedding is constructed using the Empath categorization https://github.com/Ejhfast/empath-client for each emoji, and draw the corresponding 2D plot where each emoji is represented as a point in the graph.

7. Instead of using the Empath categorization, we want to use topic modelling to handle the category-record matching. For this purpose, write a script that uses LDA topic modelling with one topic and 8 words per topic. For each emoiji labelled datata (20K tweet messages), we would like to investigate this matching by studying the mapping between the six keywords generated by the LDA and the corresponding labelled emoji. Suggest an approach that would allow you to match between the eight generated keywords and the emoji label in view of emoTag construction. For instance, assume the output of LDA is keywords K1, K2,.., K8, and assume the emoji G has weight vector [x_1, x_2, x_3, …, x_8]

   Emoji_weight = $\sum_{i \in I} x_i$ where I stands for the set of keywords among K1-K8 that coincide with keywords generated by LDA scheme.

8. Instead of using LDA, we use another state-of-the-art topic modelling technique based on transformer: BerTopic, see https://github.com/MaartenGr/BERTopic. Study the example provided in github and suggest how you will proceed to imitate the reasoning of 7) to test how BERTopic generated topics can be matched with emoTag labelling. Use the illustration provided in BERTopic to draw the result of this topic modelling. Summarize the result in a table.
9. We want to study the reverse process from text to emoji. For this purpose, study the existing MIT DeepMoji project https://github.com/huggingface/torchMoji where for a given textual post, an automatically generated emoji is outputted. Suggest a script that output for each tweet the corresponding emoji. Next, we want to use to this approach to evaluate the result with respect to the emoji label. For this purpose, for the 20K tweets associated with a given emoji label, say, *G*, calculate the proportion of emoji *G* generated by DeepMoiji. Summarize the result in a table showing the proportion of matching for each class.
10. We want to explore the antagonism relationship to perform another evaluation using NLP modules, building on the assumption that such a relation should yield disparate evidence. For this purpose, consider dataframe corresponding to two antagonist emoji (e.g., sadness and happiness), and consider again the set of 20 most frequent terms of each dataframe, excluding stopwords and uncommon characters, say H1 and H2 (sadness and happiness), and then we want to evaluate the proportion of words between the two sets H1 and H2 that are linked through antonymy relation. Suggest a script that enables you to implement this reasoning using wordnet or any lexical database of your choice.
11. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

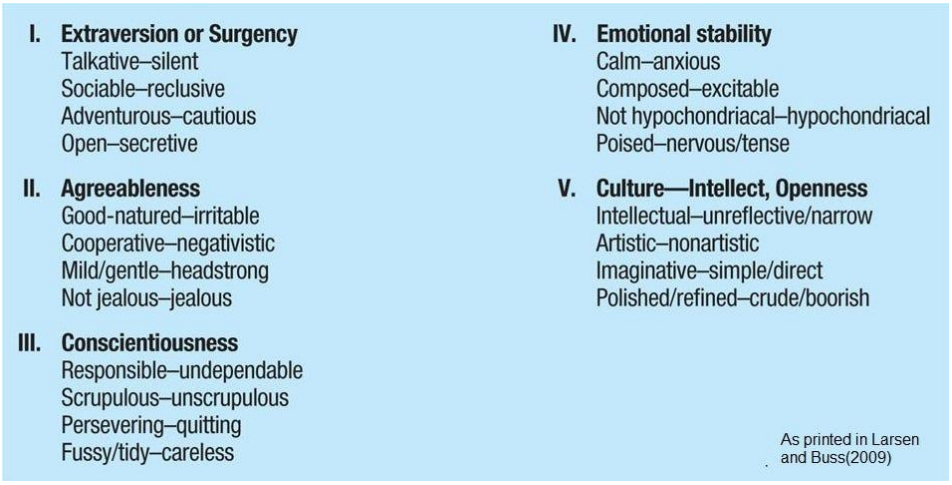
Project 14. Personality Analysis

This project explores the link between emotion and personality in dialogue systems. We especially focus on five trait personality model (extraversion, agreeableness, openness, conscientiousness, neuroticism). For this purpose, we explore the dialogue Personality EmotionLine Dataset (PEML), where from conversation A to B and B back to A, the five trait personality scores of A are recorded, together with emotion and sentiment of each of the utterances. The dataset is available at PELD/data/Dyadic_PELD.tsv at main · preke/PELD · GitHub.

1. We want to explore the general trend of the conversation data. For this purpose, create a dataframe for each speaker by gathering all his utterances in a single file. Then write a script that determines the

vocabulary set, vocabulary size, total number of tokens, total number of repetitions of words in the same post, total number of confirmation words (e.g., yes, OK, sure), total number of negation tokens, associated to each speaker. Summarize the result in a table, and then draw a subgraph that shows on the same plot the evolution of number of repetitions, number of negation, number of confirmation-like tokens, with respect to the number of tokens employed for each speaker (You may create some subdivision from the total number of tokens to ensure enough datum are used to represent the graphical illustration). Calculate the overall personality for each speaker by averaging over all instances of the original dataset, and comment on possible similarities and differences between speakers and whether some attributes are more associated with some personality patterns.

2. We want to evaluate the extent to which a speaker changes topic during conversation. In an attempt to quantify this generic trend, we hypothesize this is the case if the speaker quickly introduces new vocabulary and yields higher number of topics. To test this hypothesis, suggest a script that plots the evolution of the vocabulary size with respect to the number of tokens employed (draw on the same plot the distribution of each speaker), then comment on the suitability of the approach to discriminate key personality traits. To test the second aspect, suggest a script that uses the LDA to determine the optimal number of topics using global measures such as coherence and perplexity. You may inspire from tutoring such as Demystifying Topic Modeling Techniques in NLP | by Vijay Choubey | Medium. Similarly, comment on the suitability of the approach to highlight some personality traits.
3. We want to test whether some emotion patterns are occurring with some particular personality trait. For this purpose, we use the NRC emotion dataset where around 20K words have been assigned values for valence, arousal and dominance (VAD) dimension, see EmotionDynamics/lexicons at master · Priya22/EmotionDynamics · GitHub. Write a script that identifies NRC lexicon terms from dataframe of speaker and calculates the overage valence, arousal and dominance score normalized with respect to the total number of lexicon terms in the dataframe, as well the corresponding standard deviations, minimum and maximum value. Comment whether some valence, arousal, dominance values provide insights to discriminate some personality traits.
4. Now instead of considering overall speaker's utterances, we want to investigate individual personality trait cases. For this purpose, from PMEL dataset, in the personality attribute records, write a script that reports the dominant personality trait from the scores of the five personality traits, and then constructs a new dataframe for each personality trait by gathering all utterances that fall on this category. Then repeat the process of estimating the vocabulary set, vocabulary size, number of repetitions on same post, number of confirmation-like tokens, number of negation-like tokens. Then, for each dataframe, draw a plot showing the evolution of the size of vocabulary, number of negation-like token, number of confirmation-like tokens, with respect to the total number of tokens employed (you may create a subdivision from the total number of tokens of each dataframe). Comment whether some specific discrepancy can be spotted among the various dataframes.
5. We want to consider the definitions of each personality trait given in the figure below



I. Extraversion or Surgency
Talkative–silent
Sociable–reclusive
Adventurous–cautious
Open–secretive

II. Agreeableness
Good-natured–irritable
Cooperative–negativistic
Mild/gentle–headstrong
Not jealous–jealous

III. Conscientiousness
Responsible–undependable
Scrupulous–unscrupulous
Persevering–quitting
Fussy/tidy–careless

IV. Emotional stability
Calm–anxious
Composed–excitable
Not hypochondriacal–hypochondriacal
Poised–nervous/tense

V. Culture—Intellect, Openness
Intellectual–unreflective/narrow
Artistic–nonartistic
Imaginative–simple/direct
Polished/refined–crude/boorish

As printed in Larsen and Buss(2009)

Suggest a script that uses an approach for generating embedding of all dataframe and another embedding for each definition term, then it uses the cosine similarity to evaluate the similarity between the dataframe and personality definition. Especially, if personality trait A has multiple definition, say, for instance T1, T2, T3. Then for dataframe embedding T, the score assigned to trait A is computed as

Score (A) = max (cosine (embedding(T1), T), cosine(embedding(T2), T), cosine(embedding(T3), T)

You may use doc2vec embedding, DistilBERT embedding

Comment on the compatibility between the embedding-based personality and the labelled dataset.

6. We want to use the original PEML dataset to build a machine learning / deep learning model that predicts the score of the five personality trait of a speaker A, for each dialogue A-B-A (when B is another speaker), given the three utterances and/or associated emotion scores. Suggest a machine learning / deep learning of your choice that allows you to perform this prediction. You may consider regression-like machine learning model, with inputs as tf-idf vector (possibly by limiting the size of vocabulary using tfif-vectorizer parameter function, augmented by the emotion scores. Evaluate various configuration of features such as with and without emotion states, limited vocabulary, bigram, among others.
7. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

--

Project 15: Novels's book Analysis

This project aims to investigate the structure and stylistic properties of a Novel book Goerge Elliot Middlemarch by George Eliot | Project Gutenberg available in Gutenberg resource, focusing on the dialogue part between the different characters. The text contains character's name followed by a short description of the state of character (text under square bracket) followed by the text stated by the given character. Sometimes, in the text stated by the given character, there is a mention of another character, to point to the fact that the discussion is mainly targeting the mentioned character.

1. First, by utilizing all textbook as input, we want to elucidate the importance of individual characters. For this purpose, suggest a script that records the number of occurrences of each character in the whole textbook. Next, we assume the importance of the characters is indicated by the number of citations each character is receiving in the text statement. Summarize the result in a table indicating the importance of the characters according to i) frequency of occurrence of characters, ii) citations count of the characters.
2. Suggest a script that gathers the text associated with each character, excluding the short text under square bracket concerning the state of the target, as a single dataframe, and then, elucidates each dataframe by identifying its vocabulary set, vocabulary size, total number of tokens, total number of adjectives, total number of adverbs, total number of verbs, total number of nouns. Summarize in a table the above main statistical characteristics.
3. Write a script that draws the evolution of the vocabulary size with respect to the number of tokens for each dataframe, and then study the fitting of the Heap's law and its statistical significance in terms of R-squared and adjusted R-squared statistics (see Project 1 for possible pointers). Comment on the *talkativity* of each character.
4. Consider the modal verbs (can, may, must, shall, will, could, might, should, would), write a script that tracks the occurrence of modal verbs in each dataframe, then plot a graph showing the evolution of the number of modal verbs employed with respect to the number of tokens, and then study the possible fitting of a polynomial interpolation cure, indicating the 90% confidence curve as well (see pointers in Project 1). Comment on the difference between the usage of modal verbs by each character.
5. Suggest a script that identifies the five most frequent tokens, excluding the stopwords and uncommon characters in the whole textbook, then, estimate the frequency of occurrence each of these words in each dataframe of an individual character, yielding a five dimension vector for each character. Calculate the similarity of each pair of characters using cosine similarity of the corresponding vectors. Summarize the result in a table.
6. Suggest a script that generates embedding vector for each dataframe, using for instance, doc2vec embedding, and then compare the similarity between the characters in terms of the corresponding cosine similarity score between the corresponding embedding vectors. Summarize the result in a table.
7. Repeat 6) when empath category-based embedding https://github.com/Ejhfast/empath-client is employed. Summarize the result in a matrix and comment on the results.
8. Use the NLTK parser to identify the PoS tags of each token in each dataframe and displays a histogram showing the frequency of each tag in the dataframe. Next, use the dispersion plot, see examples in https://www.scikit-yb.org/en/latest/api/text/dispersion.html, to show the dispersion of dominant PoS tag with respect to word offset.
9. Now we want to track individual messages by each character. For this purpose, suggest a script that uses sentiWordNet lexicon GitHub - aesuli/SentiWordNet: The SentiWordNet sentiment lexicon, where each token is assigned a positive and negative score, if no lexicon is found, it is neutral, to determine the sentiment of each character message by summing up and averaging overall sentiment scores across all the lexicon terms. Then represents each character's message as a point in a triangle-based illustration

with nodes corresponding to positive, negative and neutral. Trace the plot corresponding to each character.

10. Now we want to focus on those messages where the square bracket text contains an emoiji like text, e.g., smiling, after a manual scrutinizing of the text, draw a list of messages together with their associated emojis. We want to quantify the extent of which the text matches the corresponding emoji. For this purpose, study the MIT DeepMoji project https://github.com/huggingface/torchMoji where for a given textual post, an automatically generated emoji is outputted, and generate emoji for each of the conversation message in the list and study the matching between the two corresponding emojis.
11. Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 16: NLP for Disaster Message Classification and Analysis By FuzeI

Description:

During and immediately after natural disaster there are millions of communications to disaster response organizations either direct or through social media. Disaster response organizations must filter and pull out the most important messages from this huge number of communications and redirect specific requests or indications to the proper organization that takes care of medical aid, water, logistics, etc. Every second is vital in this kind of situations, so handling the message correctly is the key.

This project focuses on building an end-to-end NLP pipeline to classify disaster-related messages and extract actionable insights for emergency response. The dataset includes over 30,000 messages from multiple disaster events such as earthquakes, floods, and hurricanes. The messages has been classified in 36 different categories related to disaster response and they have been stripped of sensitive informations in their entirety. A translation from the original language to english has also been provided.

Task 1: Data loading and exploratory data analysis

- Load the dataset using pandas and inspect key columns (message, genre, and category labels)
- Check for missing values, null entries, and duplicates.
- Count the number of messages in each category and visualize the top 10 most frequent ones using a bar plot.
- Print the number of unique categories per message (some messages belong to multiple classes).
- Discuss any data imbalance you observe (for example, many messages might be related to “aid-related”).

Deliverables: A short summary of dataset structure, one bar plot of top 10 labels, and comments on imbalance.

Task 2: Text Cleaning and Preprocessing

- Focus on the message column.

- Convert all text to lowercase. Remove URLs, numbers, punctuation, hashtags, user mentions, and extra whitespace using regular expressions.
- Remove stop words. Apply lemmatization. Store the cleaned text in a new column called *clean_text*.

Deliverables: Cleaned dataset with *clean_text* column and code snippet showing each cleaning step.

Task 3: Text Representation Using Classical Methods

- Use Bag-of-words and TF-IDF to convert cleaned text into numerical features.
- Limit the vocabulary size to control computation time. Split the data into train (80%) and test (20%) sets.
- Visualize the most frequent words using word cloud.
- Train a small Word2Vec model and visualize word embeddings using PCA.

Deliverables: Vectorized text data ready for modelling, with shapes of feature matrices printed and one visualization (bar chart or word cloud)

Task 4: Multi-label Classification Using Traditional ML models

- Train Logistic Regression, Random Forest, and SVM models using TF-IDF features to classify the messages into different categories.
- Evaluate models using accuracy, precision, recall, and F1-score. Plot confusion matrix for each model. Compare which algorithm performs best.

Deliverables: Classification results, confusion matrix, and short analysis of which model performs better and why.

Task 5: Deep Learning-Based Classification

- Convert the TF-IDF matrix or Word2Vec embeddings into tensors.
- Build a LSTM model, train for 10 epochs and monitor accuracy/loss using validation split.
- Compare deep learning results with your best traditional model from Task 4. Discuss the advantages and limitations of deep models with small datasets.

Deliverables: Training and validation accuracy/loss plots, model evaluation metrics, and brief comparison summary.

Task 6: Transformer-Based Model Fine-tuning

- Use a pre-trained model such as DistilBERT from hugging face transformers.
- Fine tune for 10 epochs using your training and validation sets.
- Evaluate on the test set using accuracy, precision, recall, and macro F1.
- Compare with the LSTM results from Task 5 and discuss performance differences.

Deliverables: Fine-tuned DistilBERT model evaluation report and one comparison table (traditional ML vs. deep learning vs. transformer)

Task 7: Named Entity Recognition (NER) for Critical Information Extraction

- Use spaCy's pre-trained model *(en_core_web_sm)* to detect named entities (like locations, organizations, people) from the *clean_text*.

- Print a few sample outputs showing highlighted entities. Count and plot the top 10 most frequent entity types.
- Discuss how entity extraction could help responders (e.g., detecting city names for routing aid).

Deliverables: Examples of extracted entities, one frequency plot of entity types, and short interpretation.

Task 8: Topic Modelling for Thematic Analysis

- Use Latent Dirichlet Allocation (LDA) and BERTopic to find key themes. Define the topics and list the top 15 words for each.
- Visualize the topic-word distribution using a bar chart or pyLDAvis. Interpret each topic briefly (e.g., "medical needs", "infrastructure damage").

Deliverables: List of discovered topics with top words, visualization of topics, and short interpretation of main themes.

Task 9: Sentiment and Emotion Analysis

- Apply VADER and TextBlob to compute sentiment polarity scores for each *clean_text*. Assign sentiment labels (positive, neutral, or negative).
- Create a bar plot showing sentiment distribution across all messages. Merge sentiment scores with categories and visualize how sentiment varies by disaster type (heatmap or grouped bar chart).

Deliverables: Sentiment labelled dataset, sentiment distribution plot, and a brief comment on overall tone of communications.

Task 10: Report and Literature Review

Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 17: Post-Event Damage Assessment Using News Article Analysis By Fuzel

Description:

This project aims to develop an NLP pipeline that automatically evaluates post-disaster damage based on textual information extracted from online news articles. Scrape real-time disaster-related reports from public sources and use NLP techniques to analyse the event's severity, affected areas, and emotional tone. The ultimate goal is

to classify and summarize the level of damage described in the news coverage, providing a data-driven perspective for situational awareness and early impact assessment.

Dataset:

News articles related to recent disaster such as floods, earthquakes, or storms will be collected using web scraping tools. Suggested source Google News RSS. The data will contain metadata (headline, date, source) and full article text for analysis.

Task 1: Event Selection and Data Collection

Select a recent disaster event such as a flood, storm, or earthquake. Use Google News RSS or the Newspaper3k library to scrape 200 articles related to that event. Extract the article title, publication date, source, and full text. Save the scraped data in a structured CSV file for further analysis.

Deliverable: CSV file with columns title, data, source, and article_text.

Task 2: Data Cleaning and Preprocessing

Clean the scraped article texts by removing HTML tags, punctuation, numbers, URLs, and irrelevant symbols using regular expressions. Convert text to lowercase and remove stopwords using NLTK or spaCy. Apply lemmatization to normalize words and store the cleaned version in a new column clean_text.

Deliverable: Cleaned dataset with a clean_text column and example before-after text samples.

Task 3: Language Filtering and Quality Check

Ensure only English articles are retained by using the langdetect package to filter out non-English content. Check for missing or duplicated entries and remove them. Print summary statistics such as article count, average text length, and number of valid entries after cleaning.

Deliverable: Filtered dataset with descriptive statistics confirming data quality.

Task 4: Keyword Extraction and Frequency Analysis

Extract key disaster-related terms using TF-IDF and CountVectorizer. Identify the top 20 keywords that frequently appear in the articles. Visualize these terms using a bar chart and a word cloud to capture dominant themes such as infrastructure damage or emergency response.

Deliverable: Frequency plots and word cloud of top damage-related terms.

Task 5: Corpus Statistical Analysis and Zipf's Law

Compute token frequency distribution and plot rank vs. frequency on a log-log scale to test Zipf's Law. Discuss whether the slope indicates a Zipfian pattern.

Deliverable: Zipf plot and short explanation.

Task 6: Lexical and Readability Analysis

Perform lexical richness and readability assessments on the cleaned corpus. Compute metrics such as Type-Token Ratio (TTR), Lexical Density, and readability indices like Flesch Reading Ease and Gunning Fog Index. Discuss what these scores reveal about the linguistic complexity of disaster-related news reporting.

Deliverable: A summary table of lexical and readability metrics with a short interpretation of the results.

Task 7: Sentiment and Emotion Detection

Perform a detailed sentiment and emotion analysis using tools such as VADER, TextBlob, and a pre-trained emotion model from Hugging Face. For each article, compute sentiment polarity (positive, neutral, negative) and detect emotion scores across categories like fear, anger, sadness, joy, and surprise. Visualize emotion intensities using radar charts or bar plots and discuss emotional tone trends across the corpus.

Deliverable: Dataset with sentiment and emotion columns, and visualizations showing emotional patterns in post-disaster coverage.

Task 8: Named Entity and Quantitative Impact Analysis

Use spaCy's Named Entity Recognition (NER) to extract relevant entities such as locations, organizations, and quantities (e.g., number of casualties, financial loss). Focus on identifying and aggregating quantitative indicators (e.g., "200 homes destroyed," "€5 million damage") to approximate impact levels. Create frequency plots showing top mentioned places or agencies and summarize key damage statistics.

Deliverable: Entity frequency plots and a short summary highlighting top affected regions and key quantitative expressions of damage.

Task 9: Event Impact Scoring Model

Develop an automatic event-impact scoring system that integrates multiple signals such as sentiment polarity, emotion intensity, frequency of damage-related keywords, and number of quantitative losses mentions. Design a scoring metric to measure the normalized Impact Score. Rank the articles by the designed metric to identify the most severe reports.

Deliverable: Impact-score computation script, ranked article list, and brief explanation of chosen metric.

Task 10: Statistical Summary and Visualization Dashboard

Combine Zipf analysis, lexical metrics, emotion trends, and impact scores into a unified analytical summary. Visualize correlations (e.g., between emotion intensity and impact score) using heatmaps and scatter plots. Create an interactive dashboard presenting all results.

Deliverable: Interactive dashboard showcasing integrated insights.

Task 11: Report and Literature Review

Use appropriate literature to comment on the findings. Also, identify any additional input that would allow you to further elucidate any of the preceding, and use appropriate literature of corpus linguistic literature to justify your findings and comment on the obtained results. Finally, comment on the limitations and structural weakness of the data processing pipeline.

Project 18: Hybrid Food Recommender with NER and Graph Analysis By Mehrdad

Goal: Build a **hybrid food recommender system** that integrates text mining (NER + sentiment), graph analysis, and collaborative filtering.

Dataset: Consider the Dataset of Food Recommendation Systems-(RAW_recipes.csv + RAW_interactions.csv).
Specifications:

1. **Exploratory Data Analysis (EDA):**
   - Summarize recipes based on preparation time, nutrition, and number of steps.
   - Visualize popular tags (e.g., "vegan", "vegetarian") using bar charts or word clouds.
   - Compare differences between top-rated and low-rated recipes.
2. **NER for Ingredient Extraction:**
   - Apply SpaCy or BERT-based NER to extract detailed food entities from recipe descriptions and reviews (e.g., “low-fat cheddar cheese”).
   - Compare extracted entities with the ingredients column to evaluate consistency.
3. **User Profiling:**
   - Construct profiles for each user including:
     - Set of rated recipes
     - Aggregated ingredients
     - Cuisine preference (from tags)
   - Visualize differences (e.g., users with >50 recipes rated vs. beginners).
4. **Aspect-Based Sentiment Analysis:**
   - Use models like LCF-BERT or lexicons to analyze sentiment toward **specific aspects** (e.g., taste, health, ease of preparation).
   - Compare aspect-level sentiment with star ratings.
5. **Ingredient-Tag Graph Construction:**
   - Build a bipartite graph between recipes and ingredients/tags.
   - Identify central nodes (most connected ingredients) using degree centrality.
6. **Graph-Based Similarity:**
   - Use graph embeddings (Node2Vec, DeepWalk) to measure similarity between recipes.
   - Visualize clusters of similar recipes with network plots.
7. **Hybrid Recommender System:**

The aim is to **design and evaluate a hybrid recommendation system** that combines **three different information sources**:

- **Ingredient/Tag Similarity (Graph-Based)**
- **Review Sentiment Embeddings**
- **Collaborative Filtering (CF) on Ratings**

Step 1: Ingredient/Tag Similarity (Graph-Based)

- Use the recipe-ingredient graph you built in **Task 5 & 6**.
- Compute recipe similarity scores using graph embeddings (Node2Vec, DeepWalk) or tag-based similarity (Jaccard/TF-IDF).

Step 2: Review Sentiment Embeddings

- Take user reviews, convert them into vector embeddings using BERT/Doc2Vec.
- Aggregate embeddings per recipe (e.g., average embedding of all reviews for that recipe).
- Optionally weight embeddings by sentiment polarity (e.g., positive reviews have more weight).
- Compute similarity between recipes based on these review embeddings.

Step 3: Collaborative Filtering on Ratings

- Implement **User-Based CF** (Pearson correlation between users) or Item-Based CF (cosine similarity between items).

- Predict user ratings for unseen recipes using weighted averages of neighbors.

Step 4: Fusion into a Hybrid Model

- Combine the three components:
  - **Graph Similarity Score** (ingredients/tags)
  - **Review Similarity Score** (sentiment embeddings)
  - **CF Predicted Rating**

- Possible fusion strategy is:
  - **Weighted Sum:**

$$FinalScore(u,i) = \alpha \times GraphSim(i) + \beta \times ReviewSim(i) + \gamma \times CF(u,i)$$

where α, β, γ are weights parameters.

8. **Explainable Recommendation:**
   - Add explanations such as:
     - "We recommend this recipe because you liked other pasta dishes with tomatoes."
   - Use attention weights or SHAP to highlight which features drive recommendations.
9. **Evaluation:**
   - Split user ratings into 80% train / 20% test.
   - Evaluate with **RMSE/MAE** for rating prediction.
   - Use **Precision@N, Recall@N, NDCG** for ranking quality.
10. **Extensions (Optional):**

- Integrate knowledge graph embeddings (e.g., TransE, DistMult).
- Explore cross-domain recommendation (recipes + drinks).

Project 19: Food Review NLP with Emotion and Explainability By Mehrdad

Goal: Use **emotion detection, NER, and explainable NLP** to analyze reviews and enhance food recommendations.
Dataset: Consider the Dataset of Food Recommendation Systems
Specifications:

1. **Exploration of Review Dataset:**
    - Analyze review lengths (avg. words per review).
    - Visualize frequent adjectives (e.g., "delicious", "bland").
2. **Emotion Classification:**
    - Classify reviews into emotion categories using NRC Emotion Lexicon or a fine-tuned transformer.
    - Compare distributions of emotions across rating groups.
3. **Emotion vs. Rating Correlation:**
    - Test if "joy" reviews map to higher ratings and "anger" to lower ratings.
    - Compute correlation coefficients between ratings and emotion scores.
4. **Cuisine-Based Emotion Analysis:**
    - Group recipes by cuisine (Italian, Indian, etc.).
    - Compare which cuisines attract more joyful vs. angry reviews.
5. **NER for Ingredient Mentions in Reviews:**
    - Extract specific ingredients from user reviews.
    - Analyze if certain ingredients (e.g., "garlic", "tofu") are linked with specific emotions.
6. **Aspect Sentiment Analysis:**
    - Extract opinions about preparation time, difficulty, healthiness.
    - Build aspect-sentiment word clouds.
7. **Explainable Classification Models:**
    - Train rating/emotion classifiers.
    - Use **LIME/SHAP** to show which words/phrases drive predictions.
8. **Semantic Similarity of Reviews:**
    - Compute BERT embeddings for reviews.
    - Cluster reviews by semantic similarity.
    - See if similar reviews also share emotions/ratings.
9. **Review-Enhanced Recommendation:**
    - Combine rating data + review embeddings into a hybrid recommendation model.
    - Compare with pure collaborative filtering baseline.
10. **Evaluation:**
    - Classification metrics (Accuracy, F1, Confusion Matrix).
    - Recommendation metrics (RMSE, Precision@N).
11. **Extensions (Optional):**
    - Compare embedding methods (Word2Vec vs. BERT vs. DistilBERT).
    - Visualize embedding spaces using t-SNE or PCA.

Project 20: IMDB Sentiment with Sarcasm, Style, and Genre Trends By Mehrdad

Goal: Improve IMDB sentiment classification by incorporating **sarcasm detection, stylistic features, and genre-specific analysis**.
Dataset: dataset of IMDB movie reviews .
Specifications
Specifications

1. **Exploratory Data Analysis (EDA):**
    - Analyze dataset statistics: number of movies, distribution of IMDB ratings, runtime, gross earnings.
    - Visualize trends over time (e.g., average IMDB rating by year, box office gross by year).
2. **Genre Analysis:**
    - Tokenize the Genre column (multi-label).
    - Find most common genres, genre combinations, and their average IMDB ratings.
    - Visualize genre popularity over decades.
3. **Overview Text Preprocessing:**
    - Clean and preprocess Overview (lowercase, remove stopwords, lemmatization).
    - Compute average overview length (in words/chars) and analyze its correlation with IMDB rating or Meta_score.
4. **Keyword Extraction from Overviews:**
    - Use TF-IDF or RAKE to extract top keywords from movie overviews.
    - Compare keywords between high-rated (>8.0) and low-rated (<6.0) movies.
5. **Sentiment/Emotion Analysis on Overviews:**
    - Apply VADER/TextBlob/BERT to measure sentiment polarity of each overview.
    - Explore whether sentiment correlates with IMDB rating or box office gross.
6. **Named Entity Recognition (NER):**
    - Apply NER on Overview to extract references to locations, organizations, or people.
    - Analyze which entities appear most frequently in high-grossing vs. low-grossing films.
7. **Predicting IMDB Rating (Regression Task):**
    - Use text features (overview embeddings), genre, runtime, certificate, and votes to predict IMDB_Rating.
    - Compare regression models: Linear Regression, Random Forest, XGBoost, Deep Learning.
    - Evaluate with MAE, RMSE.
8. **Classification: Predicting Movie Success:**
    - Define success label (e.g., IMDB rating ≥ 7.5 = "hit").
    - Train classifiers (Logistic Regression, SVM, Random Forest, BERT-based text classifier).
    - Evaluate with Accuracy, Precision, Recall, F1.
9. **Director and Actor Impact Analysis:**
    - Rank directors and actors by average IMDB rating, votes, or gross.
    - Build a network graph of collaborations (e.g., directors linked to frequent actors).
10. **Topic Modeling on Overviews:**
    - Use LDA or BERTopic to discover common movie themes (e.g., “romance”, “sci-fi adventure”).
    - Track how topics change across decades.
11. **Meta_score vs. IMDB Rating Discrepancy:**
    - Compare critic (Meta_score) vs. audience (IMDB_Rating).
    - Identify movies where critics and audiences strongly disagree.
12. **Extensions (Optional):**

- Try multimodal prediction: use both Overview text embeddings + numerical features for rating prediction.
- Apply explainability tools (SHAP, LIME) to interpret which words in overviews most influence predictions.
- Explore fairness/bias: do certain genres consistently get lower/higher critic scores than audience scores

Project 21: Tweet Sentiment with Multimodal Fusion and Temporal Trends By Mehrad

Goal: Use **transformers + metadata fusion** to predict tweet sentiment while analyzing temporal and demographic trends.
Dataset: Sentiment Analysis Dataset with text, selected_text, sentiment, user age, country, etc.
Specifications

1. **Exploratory Data Analysis (EDA):**
    - Compute sentiment class balance (positive, neutral, negative).
    - Visualize tweet frequency by time of day (morning, afternoon, evening).
    - Compare average tweet length (chars, words) across sentiments.
2. **Preprocessing and Normalization:**
    - Clean tweets (remove @mentions, URLs, hashtags, emojis).
    - Handle slang and contractions using a normalization dictionary (e.g., "u → you").
    - Compare cleaned vs. raw text samples.
3. **NER for Entity Mentions:**
    - Apply NER to extract brands, places, or public figures from tweets.
    - Analyze sentiment polarity associated with each entity (e.g., "Apple" → mostly positive).
4. **Aspect-Based Sentiment Analysis:**
    - Identify aspects in tweets (e.g., product, service, politics, health).
    - Perform sentiment analysis at the aspect level instead of whole-text polarity.
    - Compare aspect-based vs. overall sentiment predictions.
5. **Temporal Sentiment Trends:**
    - Group tweets by time (hour/day).
    - Plot sentiment evolution (are negative tweets more common at night?).
    - Compare weekdays vs. weekends.
6. **Demographic Influence on Sentiment:**
    - Group by Age_of_User and Country.
    - Compare sentiment distribution across age groups (do younger users tweet more negatively?).
    - Normalize counts by population density.
7. **Topic Modeling on Tweets:**
    - Apply LDA or BERTopic to discover main topics (e.g., politics, entertainment, sports).
    - Track how topic prevalence differs across sentiment categories.
    - Visualize with word clouds or topic maps.
8. **Transformer-Based Sentiment Classification:**
    - Fine-tune BERT, RoBERTa, or DistilBERT for sentiment classification.
    - Compare performance with classical baselines (TF-IDF + Logistic Regression).
    - Use evaluation metrics: Accuracy, F1, Precision, Recall.
9. **Hybrid Fusion Model (Text + Metadata):**
    - Create a multi-input model where:
        - Branch 1: processes tweet embeddings (BERT).
        - Branch 2: processes metadata (Age, Country, Density).
    - Fuse both branches for final sentiment prediction.
    - Compare hybrid vs. text-only model.
10. **Explainability and Error Analysis:**
    - Use SHAP/LIME to identify which words or metadata features drive sentiment predictions.
    - Manually inspect misclassified tweets (sarcasm, slang, mixed sentiment).
    - Highlight common error cases (short tweets, negations like "not good").

11. **Bias and Fairness Analysis:**
    - Evaluate if sentiment predictions are biased toward certain countries or age groups.
    - Report fairness metrics (demographic parity, equal opportunity).
12. **Extensions (Optional):**
    - Try zero-shot or few-shot sentiment classification with GPT models using prompt engineering.
    - Experiment with multimodal embeddings (text + metadata + emojis/hashtags).
    - Compare transformer embeddings with simpler embeddings (Word2Vec, TF-IDF).

Project 22: Retrieval-Augmented Recipe Search and Information Retrieval By Mehrdad

Goal: Build a **recipe retrieval and question-answering system** using information retrieval methods and Retrieval-Augmented Generation (RAG). The system should allow users to type a query (e.g., *"low-fat chicken pasta under 500 calories"*) and retrieve matching recipes based on structured nutritional attributes and textual fields.

Consider HUMMUS dataset

Dataset Columns:

- recipe_id, title, duration, directions, recipe_url, tags, servingsPerRecipe, servingSize [g], calories [cal], caloriesFromFat [cal], totalFat [g], saturatedFat [g], cholesterol [mg], sodium [mg], totalCarbohydrate [g], dietaryFiber [g], sugars [g], protein [g], direction_size, ingredients_sizes, ingredients, image_url, who_score_normalized, health_category

Specifications

1. **Exploratory Data Analysis (EDA):**
    - Analyze distributions: calories, protein, fat, sodium, cooking duration.
    - Visualize recipe health categories (e.g., healthy, very_healthy, unhealthy).
    - Compute correlations (e.g., protein vs. calories).
2. **Text Preprocessing of Queries and Recipes:**
    - Normalize recipe titles, direction, tags, and ingredients (lowercasing, tokenization, stopword removal).
    - Preprocess user queries in the same way for consistency.
    - Compare statistics: average ingredient length, number of tags per recipe.
3. **Indexing for Search:**
    - Build an inverted index using recipe title, tags, ingredients, and directions.
    - Implement basic keyword search with TF-IDF or BM25.
    - Example query: *"gluten-free pasta with tomato"* → retrieve recipes with relevant tags + title.
4. **Semantic Embeddings for Retrieval:**
    - Use BERT/Doc2Vec embeddings for semantic similarity between queries and recipes.
    - Compare keyword-based (BM25) vs. semantic search.
    - Example: Query *"low-carb chicken meal"* should match recipes tagged as "keto" or "low-carb" even if not exact matches.
5. **Hybrid Search:**
    - Combine keyword-based (BM25) and embedding-based retrieval with a weighted score.
    - Evaluate whether hybrid search improves retrieval over single methods.
6. **Structured Attribute Filtering:**
    - Allow structured filters based on numeric fields.
    - Example queries:
        - *"recipes under 400 calories"*
        - *"high-protein (>20g) vegetarian dish"*
    - Implement SQL-style filtering combined with IR.
7. **Query Understanding & Expansion:**
    - Implement query expansion with synonyms (e.g., *"aubergine"* → *"eggplant"*).
    - Handle vague queries like *"healthy dessert"* by expanding to tags + nutrition thresholds.
8. **RAG (Retrieval-Augmented Generation):**
    - Integrate a pre-trained LLM (e.g., GPT, BERT-QA, LLaMA) with the retrieval pipeline.
    - System pipeline:

1. Retrieve top-k candidate recipes.
2. Pass them as context to the LLM.
3. Generate a natural language answer explaining the recommendation.

- Example Query: *"Suggest a quick vegetarian dinner under 20 minutes."*
  - Retrieval: recipes with duration < 20 and tag vegetarian.
  - LLM: *"Here are some options: Quick Veggie Stir Fry (15 min, 250 calories), Tomato Basil Pasta (18 min, 400 calories). Both are vegetarian and under 20 minutes."*

9. **Evaluation of Retrieval Models:**
   - Create a set of test queries with known relevant recipes.
   - Metrics: Precision@k, Recall@k, MRR (Mean Reciprocal Rank), NDCG.
   - Compare keyword search vs. embeddings vs. hybrid vs. RAG-based answers.
10. **Explainability and User Experience:**
    - Generate explanations for retrieved results (e.g., "This recipe matches because it has <400 calories and contains chicken.").
    - Include highlighted snippets from directions/ingredients in the results page.
11. **Visualization & Interface (Optional but Encouraged):**
    - Build a simple web interface (Streamlit/Flask/React).
    - User enters query → system returns ranked list with:
      - Recipe title
      - Image (image_url)
      - Calories/protein/fat summary
      - Explanation (from RAG or rule-based).
12. **Extensions (Optional Advanced Task):**
    - Personalization: incorporate user preferences (e.g., low-salt diet, disliked ingredients).
    - Multi-turn retrieval: allow conversational queries like "Show me vegan desserts. Now only under 300 calories."
    - Compare RAG with pure retrieval-based QA (BM25 + extractive QA model).

Project 23: Fine-Tuning LLMs for Recipe Recommendation and Instruction Following By Mehrdad

Goal:
The aim of this project is to **fine-tune a Large Language Model (LLM)** on recipe data to improve its ability to follow food-related instructions. You will generate instruction-response training data from the recipe dataset, fine-tune an LLM, and compare its performance with a non-fine-tuned baseline. A key part of the project is **detecting and analyzing hallucinations** (when the model generates incorrect or non-existent information).

Consider HUMMUS dataset

Specifications
1. Dataset Exploration
- Summarize and visualize key nutritional features: calories, protein, and sodium.
- Explore the distribution of duration to compare quick vs. long recipes.
- Identify the most common health_category values.
- List the top tags and frequent ingredients.
- Plot a simple correlation (e.g., protein vs. calories).

2. Instruction Dataset Generation
- Convert recipe data into **instruction-response pairs** that simulate user queries.
- Example formats:
    - **Instruction:** *"Suggest a healthy pasta recipe under 400 calories."* **Response:** *"Vegetarian Tomato Basil Pasta - 350 calories, 15g protein, ready in 20 minutes."*
    - **Instruction:** *"List three low-sodium chicken dishes."* **Response:** *"1. Lemon Garlic Chicken (180mg sodium)… 2. Grilled Herb Chicken (200mg sodium)… etc."*
- Include a variety of queries: ingredient filters, nutrition limits, preparation time, health category, or combinations.

3. Data Formatting for Fine-Tuning
- Structure the dataset in JSONL with fields:
- {
- "instruction": "Suggest a vegan dessert under 300 calories",
- "input": "",
- "output": "Fruit Salad with Citrus Dressing - 250 calories, 5g fiber, 15 min preparation."
- }
- Split into **training (80%)**, **validation (10%)**, and **test (10%)** sets.

4. Baseline (Non-Fine-Tuned LLM)
- Use a pre-trained model (e.g., GPT-2, LLaMA) without fine-tuning.
- Prompt it with test queries and record outputs.
- Example: Query *"Give me a high-protein vegetarian recipe under 20 minutes"*.
    - Baseline might return vague or incorrect answers.

5. Fine-Tuning the LLM
- Fine-tune a small/medium LLM (e.g., GPT-2, Mistral, LLaMA-2 with LoRA).

- Train on the generated instruction-response dataset.
- Track training/validation loss and document hyperparameters.

6. Evaluation Framework

- Use test queries to compare fine-tuned vs. baseline models.
- Evaluate outputs on:
    - **Relevance:** Does the answer match the query constraints?
    - **Factual Accuracy:** Do calories, protein, or nutrition values match dataset entries?
    - **Fluency:** Is the answer coherent and natural?
- Metrics: BLEU/ROUGE (text similarity), Precision/Recall (constraint satisfaction), human evaluation for fluency.

7. Hallucination Detection

- Define hallucinations as:
    - **Factual:** Model invents values not in dataset (e.g., wrong calorie counts).
    - **Recipe:** Model suggests a recipe not present in dataset.
- Automatically check outputs against dataset fields.
- Calculate hallucination rate = % of outputs with incorrect/made-up facts.

8. Hallucination Mitigation

- Strategies to reduce hallucinations:
    - Ground outputs in retrieved recipe data before generation.
    - Post-check outputs against structured fields and correct mismatches.
    - Add system-level constraints in prompts (e.g., *"Only use facts from dataset."*).

9. Error Analysis

- Compare error cases between baseline and fine-tuned models.
- Example:
    - Baseline → *"Vegan Chicken Curry under 200 calories"* (hallucination).
    - Fine-tuned → *"Tofu Stir-Fry - 190 calories, vegan, 18 min prep."* (valid).
- Identify frequent errors such as unrealistic calorie estimates or invented recipes.

10. Comparison Study

- Summarize differences between **baseline** and **fine-tuned**:
    - Baseline: more flexible, but prone to hallucinations.
    - Fine-tuned: more accurate, dataset-grounded, better at constraint-following.
- Provide tables/graphs of evaluation metrics.

12. Extensions (Optional)

- Try **parameter-efficient fine-tuning** (LoRA, adapters) to save resources.
- Explore **data augmentation** (paraphrased queries, synthetic instructions).
- Compare **RAG + fine-tuning** vs. **fine-tuning only** for hallucination reduction.

Project 24: PDF-to-Knowledge Graph → Instruction Generation → LLM Fine-Tuning

By Mehrdad

Goal

Extract structured knowledge from a domain PDF, construct a **knowledge graph**, generate **fact triples** and **instruction-response pairs** grounded in KG, then **fine-tune LLM** and evaluate it against baselines

Source Data

- "SUSTAINABLE HEALTH FROM FOOD" **PDF** document in food domain
- You will **not** rely on external web sources; all supervision must be **grounded in PDF** (and its KG).

Specifications

1) PDF Ingestion & Parsing

- Extract text with page/section anchors (page number, heading hierarchy).
- Preserve structure: titles, subsections, lists, tables, figures' captions.
- For tables: parse into machine-readable frames (CSV/JSON) when possible.
- Deliverables: raw_text.jsonl (chunks with metadata), tables/*.csv.

2) Domain Schema & Ontology Draft

- Define a minimal ontology for the domain (entities, attributes, relations):
  - Recipe, Ingredient, Nutrient, Technique, relations like hasIngredient, hasNutrient, requiresTechnique.
- Document data types, cardinalities, and URI/prefix conventions.
- Deliverable: ontology.yaml (or TTL if using RDF).

3) NER & Keyphrase Extraction

- Run a domain-adapted NER pipeline (spaCy/transformer) + keyphrase extraction (YAKE/KeyBERT) over text chunks.
- Map mentions to ontology classes (entity typing).
- Resolve coreference (merge aliases, e.g., "vitamin C" ↔ "ascorbic acid").
- Deliverable: entities.jsonl with canonical IDs and mention spans.

4) Relation Extraction & Triple Building

- Use rule-based patterns and/or relation extraction models to detect relations (e.g., *Ingredient X contains Nutrient Y*, *Technique Z requires Temperature T*).
- Convert to triples (RDF or property graph). Include provenance (page, line span).
- Deduplicate and validate (schema consistency).
- Deliverables: triples.ttl (RDF) or graph.json (property graph).

5) Knowledge Graph Construction & Storage

- Load triples into a graph store (RDF triplestore like GraphDB/Fuseki, or Neo4j).
- Create indices; verify counts by class/relation.
- Run sample SPARQL or Cypher queries to sanity-check coverage (e.g., list top ingredients by frequency; nutrients per ingredient).
- Deliverable: query notebook with 5-10 example queries.

6) Fact Set & Contradiction Checks

- Materialize **atomic facts** from the KG (subject-predicate-object) with natural-language paraphrases.
- Detect intra-document contradictions or overlaps (e.g., numeric conflicts across pages); flag for curation.
- Deliverables: facts.jsonl with fields: {triple, text_support, page, paraphrases[]}.

7) Instruction-Response Dataset Creation (Grounded)

- Generate **instruction-response pairs** grounded strictly in facts/KG.
- Types of instructions:

    - **Factoid QA:** “What nutrient is high in X?” → grounded answer + citation (page/section).
    - **List/Compare:** “List ingredients rich in fiber under 100 kcal/100g.”
    - **Reasoning:** “If a recipe requires Y technique, which safety temperature applies?”
    - **Constraint queries:** “Give two vegan recipes under 400 kcal.”
- Include **input grounding**: attach the top-k supporting facts/triples to each pair during generation (kept for training or for eval only).
- Format (JSONL):
- {
- "instruction": "List two ingredients rich in vitamin C.",
- "input": "",
- "output": "1) Bell pepper 2) Orange. (Source: p.12, p.47)",
- "grounding": [{"s":"Bell_pepper","p":"hasNutrient","o":"Vitamin_C","page":12}, ...]
- }
- Train/val/test split (80/10/10). Ensure **no leakage** (e.g., split by section/page blocks).

8) Baselines: RAG over KG vs. Zero-Shot LLM

- **RAG-KG**: Retrieve from KG (SPARQL/Cypher) + chunk text; feed to an LLM to generate answers with citations.
- **Zero-shot LLM**: Prompt a non-fine-tuned model using only the user instruction (no KG).
- Record answers and metadata (latency, context length).

9) Fine-Tuning Setup

- Choose model and method (e.g., **Mistral-7B + LoRA** or **LLaMA-2-7B + adapters**).
- Train on the instruction dataset. Log training/validation loss; track early stopping.
- Keep **prompt template** consistent (system guardrails: “Only answer using facts from the provided corpus; include sources.”).

10) Evaluation (Quality, Factuality, Faithfulness)

- Automatic metrics:
    - **Exact match / F1** for factoid.
    - **ROUGE/BLEU** for generations (limited value; use cautiously).
    - **Constraint satisfaction rate** (for structured queries).
    - **Faithfulness**: string-match (or fuzzy match) of outputs to KG facts.
- Human or rubric-based:
    - **Relevance**, **Completeness**, **Citation correctness**.
- Compare **Fine-tuned** vs. **RAG-KG** vs. **Zero-shot** across a shared test set.
- Summarize in tables/plots.

11) Hallucination Detection & Mitigation

- **Detection:** For each generated answer:
    - Align claimed entities/values to KG; mark unsupported spans.
    - **Hallucination rate** = % answers with unsupported claims..

12) Delivery & Demo (Optional but Encouraged)

    - Simple UI (Streamlit/Gradio): Input: user question, Output: answer + inline citations (page/section) + “View sources” and Toggle: **Zero-shot**, **RAG-KG**, **Fine-tuned**.

Project 25: RAW PDF → Facts & Instruction-Response Dataset By Mehrdad

Goal

Build a robust pipeline that ingests a **raw PDF**, extracts and normalizes content (text, tables, figures/captions), and produces two artifacts:

1. **Facts dataset**: canonical, deduplicated, and **provenance-linked** atomic facts.
2. **Instruction-response dataset**: grounded Q/A and task instructions **derived only from the extracted facts**.

Consider "Nordic nutrition recommendations" PDF which **c**ontains nutrition guidelines, culinary techniques, food safety manuals, product catalogs.

Specifications

1) PDF Ingestion & Structure Recovery

- Parse text with section headings and page numbers.
- Example (from NNR2023):
- Section: Food Processing Recommendations
- Page: 102-103
- Text: "Breastfeeding should be preferred compared to infant formulas...
- SSB and energy drinks should be limited...
- Whole grain cereal products should preferentially be used instead of refined cereal products..."
- Output: JSON with {section, page, text_chunk}.

2) Cleaning & Normalization

- Remove numbering, fix line breaks.
- Example cleaned text:
- Breastfeeding should be preferred compared to infant formulas.
- Consumption of sugar-sweetened beverages (SSB) and energy drinks should be limited.
- Whole grain cereals should be preferred over refined cereals.
- ...

3) Entity Extraction (NER)

- Detect **key entities**: foods, nutrients, food groups, actions.
- Example entities:
  - Breastfeeding, Infant formulas
  - Sugar-sweetened beverages (SSB), Energy drinks
  - Whole grain cereal products, Refined cereal products
  - Fruit juices, Deep-fried potatoes, Vegetable oils, Butter, Trans fatty acids, Saturated fat, Salt, Added sugar

4) Fact Candidate Extraction

- Convert recommendations into **atomic fact candidates**.
- Example facts:
  - (Breastfeeding) - shouldBePreferredOver → (Infant formula)
  - (Sugar-sweetened beverages, energy drinks) - shouldBeLimited → (Intake)
  - (Whole grain cereals) - shouldBePreferredOver → (Refined cereals)
  - (Deep-fried potatoes) - shouldBeLimited → (Consumption)
  - (Vegetable oils) - shouldBePreferredOver → (Butter, Hard margarine, Tropical oils)

- (Dietary pattern) - shouldLimit → (Saturated fat, Salt, Sugar)

5) Canonicalization & Deduplication

- Normalize synonyms:
  - SSB → Sugar-sweetened beverages
  - Butter-mixes → Butter and margarine
- Deduplicate repeated recommendations (e.g., “limit added sugar” in both DRVs and FBDGs).

6) Fact Quality & Provenance

- Each fact linked to source:
- Fact: "Whole grain cereals should be preferred over refined cereals"
- Source: NNR2023, Section 'Food Processing Recommendations', Page 102
- Confidence: High (explicit recommendation statement)

7) Instruction Template Library

- Templates for converting facts into user queries:
  - **Factoid:** "What is recommended instead of infant formula?"
  - **Comparison:** "Which cereal products are recommended over refined ones?"
  - **Constraint:** "Which drinks should be limited according to the guidelines?"
  - **List:** "List three food products that should be limited in NNR2023."
  - **Why/Reasoning:** "Why should vegetable oils be preferred over butter and tropical oils?"

8) Instruction-Response Generation (Grounded)

- Example instruction-response pairs (grounded in facts):

1.

```
{
 "instruction": "What is recommended instead of infant formula?",
 "input": "",
 "output": "Breastfeeding is recommended over infant formulas. (NNR2023, p.102)",
 "supports": ["fact:Breastfeeding_preferredOver_InfantFormula"],
 "citations": [{"pdf":"NNR2023","page":102}],
 "type": "factoid"
}
```

2.

```
{
 "instruction": "Which drinks should be limited?",
 "input": "",
 "output": "Consumption of sugar-sweetened beverages (SSB) and energy drinks should be limited. (NNR2023, p.102)",
 "supports": ["fact:SSB_limited","fact:EnergyDrinks_limited"],
 "citations": [{"pdf":"NNR2023","page":102}],
 "type": "constraint"
}
```

3.

```
{
 "instruction": "List three foods that should be limited.",
 "input": "",
```

"output": "According to NNR2023, the following foods should be limited: (1) deep-fried potatoes and potato products with added fat and salt, (2) sweets and confectioneries, (3) milk and dairy products high in saturated fat. (NNR2023, p.103)",
"supports": ["fact:DeepFriedPotatoes_limited","fact:Sweets_limited","fact:DairyHighFat_limited"],
"citations": [{"pdf":"NNR2023","page":103}],
"type": "list"
}

9) Hallucination Prevention

- All outputs must come **only from extracted facts**.
- If query cannot be answered: *"Not found in NNR2023 recommendations."*

10) Hallucination Detection

- Check each response against fact IDs.
- Flag outputs that mention entities not in the fact set.
- Example hallucination: *"Limit fried rice"* → flagged, since it's not in extracted facts.

11) Quality Assurance

- Validate JSON schema.
- Ensure each instruction has at least one supporting fact + citation.
- Human spot-check: sample 50 instructions, rate for correctness & clarity.

12) Packaging & Data Card

- Deliver facts.jsonl, instructions_train/val/test.jsonl.
- Write a **Data Card** describing:
    - Source PDF(s)
    - Fact extraction methodology
    - Instruction generation rules
    - Known limitations (coverage, ambiguity in guidelines)

Project 26: Literature Mining & Fact Analysis for "Explainable Recommender Systems" By Mehrdad

Goal

Design and implement a pipeline that automatically searches, collects, and analyzes research papers for a specific topic. The project should include automatic download of papers, metadata extraction, exploratory analysis (publication year, countries, universities, …), and fact-based analysis of trends and methods.

Specifications

1. Define Research Scope

- Define a keyword query: "Explainable Recommender Systems" (and possible variations like "XAI Recommender", "Interpretable Recommendation").

2. Automatic Paper Retrieval

- Use APIs like **Semantic Scholar, Arxiv, or CrossRef,** to fetch metadata and full-text PDFs.
- Collect at least **100 papers (open access)** (titles, abstracts, DOIs, authors, year, venue).
- Store results in papers_metadata.json.

3. Downloading PDFs

- For each paper with open-access PDF available, download automatically.
- Save with structured naming: papers/<year>_<firstAuthor>_<title>.pdf.

4. Metadata Extraction

- Extract structured metadata:
  - **Year of publication**
  - **Authors**
  - **Affiliations (University, Institution, Country)**
  - **Venue (conference/journal)**
  - **Citation count (if available)**
- Store in papers_metadata_enriched.json.

5. Exploratory Data Analysis (EDA)

Perform descriptive analytics:

- **Publication Trends:** Number of papers per year.
- **Top Venues:** Conferences/journals with the most publications.
- **Top Authors & Universities:** Count of papers per author/university.
- **Geographic Distribution:** Country-level publication counts (visualization).
- **Collaboration Networks:** Build a co-authorship graph.

Deliverables: Visualizations (bar plots, line plots, collaboration graph).

6. Text Preprocessing

- Extract abstracts and full text. Clean text: lowercasing, tokenization, stopwords removal, lemmatization. And prepare corpus for text mining.

7. Topic Modeling & Clustering

- Apply topic modeling (LDA, BERTopic) on abstracts.
- Identify major research themes ("neural explainability," "user studies," "evaluation metrics").

- Visualize topic evolution over time.

8. Advanced Fact Extraction (from Abstracts, Full Texts, and Tables)

- **From Abstracts & Full Texts:**
  - **Methods Used:** Detect whether a paper uses *Matrix Factorization, Attention Models, Graph Neural Networks, Transformers, LLM-based methods*.
  - **Application Domains:** Extract mentions of *movies, tourism, e-commerce, music, healthcare, social media, etc.*
  - **Explanation Types:** Identify whether the system provides *feature-based, natural language, visual, counterfactual, or interactive explanations*.
  - **Evaluation Metrics:** Extract standard metrics (*RMSE, MAE, NDCG, Recall@K*) as well as *qualitative user studies or A/B testing*.
- **From Tables (Table Parsing + NLP):**
  - Parse experimental result tables (e.g., "NDCG@10 = 0.423" for Model X).
  - Extract structured facts like:
    - (PaperID) - achievesMetric → (NDCG@10: 0.423 on MovieLens)
    - (PaperID) - baselineComparison → (GNN outperforms MF by +12%)
  - Capture *dataset names* (MovieLens, Amazon, Yelp, LastFM).
- **Represent as Fact Triples (with Provenance):**
  - (PaperID) - usesMethod → (Graph Neural Network)
  - (PaperID) - applicationDomain → (Healthcare)
  - (PaperID) - evaluatesOn → (MovieLens-1M, NDCG@10=0.423)
  - (PaperID) - achievesImprovementOver → (Matrix Factorization, +12%)

Deliverable: facts.jsonl where each fact has {paper_id, subject, predicate, object, evidence (text/table snippet), page/section}.

9. Advanced Fact-Based Analysis Tasks

Students must use the extracted facts to perform **deeper analyses**:

1. **Method Trends & Evolution:**
   - Which algorithms dominate in each time period (pre-2018 vs. 2019–2025)?
   - Are GNNs and Transformers replacing Matrix Factorization?
2. **Application Domain Leadership:**
   - Which domains (movies, e-commerce, healthcare, tourism) are most studied?
   - Which domains are emerging recently (e.g., health, LLM-based recommendation)?
3. **Dataset Usage Patterns:**
   - Which datasets are most frequently used? (MovieLens, Amazon, Yelp, LastFM, proprietary industry datasets).
   - Which domains have less standardized benchmarks?
4. **Performance Trends:**
   - Extract **metric values from tables** and compare across methods.
   - Example analysis: *"On MovieLens-1M, GNNs achieve average NDCG@10=0.42, while MF achieves 0.31."*
5. **Geographic & Institutional Leadership:**
   - Which countries and universities publish most on *explainable recommenders*?
   - Which institutions dominate specific sub-themes (e.g., visual explanations in Asia vs. counterfactuals in Europe)?
6. **Collaboration & Network Analysis:**

- Build a co-authorship or institution-collaboration graph.
- Identify key hubs of research activity.

10. Knowledge Graph Creation (Optional Advanced Task)

- Build a small **Knowledge Graph** linking papers, authors, methods, domains, and explanation types.
- Allow SPARQL/Cypher queries like:
  - *"Find all papers on healthcare recommenders from 2018–2023 that use LLM."*

11. Comparative Study of Subfields

- Compare early vs. recent research:
  - Early (pre-2018) → mainly rule-based, matrix factorization.
  - Recent (2019–2025) → deep learning, GNNs, transformers, LLM explainability.
- Highlight shifts in methods and evaluation.

Project 27: Environment keyword mapping

This project aims to review selected environmental topics according to Wikipedia-based description.

1. Consider the wordings: “nature”, “resilience”, “sustainability”, “climate change”. Suggest a script where your input each of these wordings and output the corresponding Wikipedia page. You may look at the examples shown in Python for NLP: Working with Facebook FastText Library (stackabuse.com). Reorganize the each webpage so that the titles of subsections and the list of entities (clickable keywords from the webpage, expect for the reference list) are stored separately.
2. Assume the content of each webpage is a single document. Use relevant NLTK script to create a corpus constituted of the four document, and appropriate proprocessing and lemmatization, to construct the TfIdfVectorizer of each document and then calculate the cosine similarity of each pair of these documents. Provide the result in a table and comment on the findings.
3. Repeat 2) when the documents are restricted only to the titles of subsections of each document.
4. Repeat 3) when entity-categories are used as a basis to represent each document. Comment on the findings and discuss how Wikipedia entity category can be used to improve similarity calculus among each pair.
5. Use a script to calculate Wu and Palmer WordNet semantic similarity between each pair in 1). Write a vector reproducing the similarity of each pair and then calculate the correlation between the semantic similarity result and each of the above Wikipedia based similarity.
6. Now we want to further extend the entity-based similarity in the following way. Write a script that scraps the content of each entity (using beautiful soup or any other scrapper of your choice) and retrieves all the entity-category (clickable keywords) identified during this first exploration stage. The process uses only a first pass exploration.
7. Repeat the TfIdfVectorizer representation and recalculate the cosine similarity between individuals words of the four keywords in 1).
8. Use the pretrained word2vec to represent each words and calculate the corresponding similarity.
9. Now we want to consider the news around each of these keywords. Select a news forum of your choice and write a script that retrieves documents for each keyword. Ensure that you collect sufficiently enough data (few hundreds) various time periods. Create a dabaset where you store documents for each keyword for each time period according to your time subdivision of your own. Use WordCloud library to display the world cloud representation of news textual data associated to each keyword.
10. Repeat the calculus of the TfIdfVectorizer based similarity among each pair at each time interval.
11. Comment on the results using appropriate literature
12. Suggest a GUI that would enable you to illustrate the functioning of the various specification of your project.

Project 28: Mining nature from the web

This project aims to assess how the search engine can teach community by promoting documents that can deliver positive message about the target word. We shall consider the target word “carbon footprint”.

1. We want to initially analyze the outcome of Google search for a query “climate change”. Write a script that would enable you to retrieve the 100 snippets corresponding to the result of the search. You can inspire from example in https://www.thetopsites.net/article/52274742.shtml. You may notice that using free Google developer account would not allow you to reach 100, so it is fine to manually copy and past the snippets and save the result in an excel sheet. You are also free to explore snippets from other search engine that put less limitations on number of snippets gathered using free account.
2. We shall consider each snippet as a single document, say Qi for the i^{th} snippet. Use the WuzzyFuzzy string matching (see example in the course handout) to calculate the score of string matching of each pair of the set of the first 10 snippets of the search outcome. Especially, calculate the value of three variables: V1 corresponding to the total number of similar snippets (string matching score equals 100%); V2 for the average string matching of all pairs; V3 for the standard deviation corresponding to this average value. Now repeat this process for the first 20 snippets, then first 30 snippets, .. set of 100 snippets. Draw the plots showing the variation of V1, V2 and V3 in this consonant representation (embedded set) of the search outcome.
3. Now we would like how to see how the search outcomes differ in terms of sentiment polarity. For this purpose, use a sentiment analyzer of your choice (e.g., textblob, SentiWordNet, SentiStrength) to calculate the sentiment polarity of each snippet. Using previous consonant subdivision, and let V4 be the vector carrying the average sentiment for each subset (first 10 snippets, first 20, first 30, .., first 100) and let V5 be the vector carrying the associated standard deviation values. Plot the variation of V4 and V5.
4. Now we would like to restrict the matching between the snippets to the common named-entities occurred in both snippets of the pair. For this purpose, write a script that uses spacy named-entity tagger Linguistic Features · spaCy Usage Documentation for identifying named-entities in each snippet, and perform a simple matching to identify proportion of named-entities shared between two snippets of each pair. Use this reasoning to calculate and plot the corresponding V1, V2 and V3, in the same spirit as in 2).
5. Concatenate all snippets into a single document, and then draw a WordCloud representation of the document. You may see examples of WordCloud in Creating Word Clouds in Python | Engineering Education (EngEd) Program | Section.
6. Repeat reasoning 2) and 3) when seeking the outcomes of the search from bbc news website. Use beatifulSoup scrapper or any other tool to scrap the titles of each search outcome only. Retrieve the first 100 search outcome, and compute, draw the variables V1, V2 and V3, V4 and V5.
7. Concatenate all search outcomes in a single document as in 4) and draw the associated wordcloud.
8. Discuss the results in the light of relevant literature.
9. Suggest a GUI that allows you to illustrate the functioning of the various specification of your project

Project 29: Sentiment Analysis of Hotel Review

This project aims to investigate the sentiment and test various architectures for comprehending sentiment polarity in London hotel dataset. We shall consider a manually labelled dataset available at 515K Hotel Reviews Data in Europe. The dataset contains review of luxurious 1493 hotels in Europe with hotel locations, reviews and user's rating. We shall focus on the ambiguity that may raise among the users.

1. Use the SentiStrength from http://sentistrength.wlv.ac.uk/ to determine the positive, negative and overall (sum of positive and negative) sentiment score for each review. Provide a database D1, which contains these information for each review.
2. Use Pearson correlation coefficient to evaluate the correlation of the overall sentiment score of each review with the user's rating.
3. Gather all reviews pertaining to the same hotel altogether. Calculate the average, standard deviation, kurtosis of user's ratings. Identify the hotels with low standard deviation and cases with relatively high standard deviations. Comment whether the high variation of standard deviation occurs in expensive hotel or cheap hotels (using for instance information about the hotel stars). Construct a histogram showing for each star category, the proportion of hotels for which the standard deviation is greater than some threshold ε (that you select yourself, depending on the observed range of variation of the standard deviation). We shall refer to this new class of hotels whose standard deviation is beyond the threshold as Ambiguous Class.
4. Assume that for a given hotel whose users' rating standard deviation is higher than the pre-selected threshold, if the majority of the reviews are assigned negative sentiment according to user's rating, then the hotel is assumed to belong to negative class. Likewise for positive sentiment. Provide therefore the two subclasses of this ambiguous class review. Provide an attribute in the database D1 indicating whether a given hotel belongs to Ambiguous class and whether it is positive or negative subclass. Draw a histogram highlighting the proportion of positive and negative subclass in the original Ambiguous Class.
5. We would like to explore the content of each of the above subclass. Concatenate all reviews of each subclass and use wordCloud (see an example at https://www.geeksforgeeks.org/generating-word-cloud-python/) that highlights the most frequent wording used. Plot WordCloud for both positive and negative subclass. Comment on the findings
6. Use LDA with five topics and five words per topic to determine the topic distribution of the positive and negative subclass. Report the result in the database 1. Compare the result of the LDA output with the WordCloud finding (check the extent of the overlapping with high frequent words for instance or whether some LDA wording are directly related to WordCloud via other semantic network.
7. Consider categories generated by Empath Client https://github.com/Ejhfast/empath-client. Apply Empath Client to positive and negative subclasses reporting only those categories who held non-zero values. Record these findings in database D1.
8. Calculate the ratio of overlapping over the number of Empath categories as an indication of agreement between empaths and common corpus.
9. Repeat 8) by calculating the agreement between Empath categories and LDA.
10. Consider the standard five class categories often employed in hotel review: Price, Service, Parking, Room, Location, Food. Suggest your own ontology construction that gathers words associated to each category (for instance using list of synonym, thesaurus, hypernyms,…).
11. Use your developed ontology to seek the histogram of the frequency of the occurring five categories for both positive subclass and negative subclass.
12. Use the Harvard General Inquirer available in http://www.wjh.harvard.edu/~inquirer/inquirerbasic.xls and identify the list of adjective /adverbs pertaining to positive sentiment and that pertaining to negative sentiment. Now we would like to track the occurrence of these words in positive and negative subclass dataset. For this purpose, for each adjective/adverb who occurs in the subclass, we would like to identify nouns that are associated to it. For this purpose, use NLTK parser-tree to identify the first noun that is associated to it. For

instance “Service is bad” the adjective “bad” should be linked to Service. Alternative to parser-tree would be to take a window of two words around the underlined adjective/adverb and identify the one that corresponds to a name category (you can use standard part-of-speech tagger to identify the noun word category). Report in the database, for each of these adjective/adverb, the list of nouns for which they have been found it is associated to, for both positive and negative subclass dataset. Comment on the potential misconception this may cause and potential agreement with previous findings.

13. Design a simple GUI of your choice that show the execution of each of the above tasks in a way to ease the task of the assessor or external end-user.

Project 30: Wordsense Disambiguation Using Wikipedia 2

The project consists in implementing a new scheme of word disambiguation using Python NTLK and Wikipedia. Typically, a target word like “plant” will have different entries (clickable) that correspond to various senses associated to “plant”. You can also search “plant (disambiguation)” and this should guide you to different senses associated to “plant”. We shall consider initially the sentence S: “This plant requires watering every morning”

1) We would like to disambiguate the target word “plant” in sentence S using Wikipedia entries and compare this with WordNet. First, use a simple Lesk implementation to find and display the sense associated to the target word in S.
2) We would like to use Wikipedia as the main lexical database, use either Wikipedia python API or Wikipedia dump (large scale download) and search & display the various entries associated to query “plant” (or you can also directly query the outcome of the query “plant (disambiguation)”. Output the various senses. Next, calculate the amount of overlapping between each sense gloss and the context word of the sentence similarly to simplified Lesk’s approach. Display the sense that yields the highest overlapping.
3) Consider AQUAINT dataset, restrict only to first 50 news documents). The database is available at https://catalog.ldc.upenn.edu/LDC2002T31. Identify a target of your choice and use your script developed in 2) to disambiguate the target word
4) Consider WikiSim framework available from https://github.com/asajadi/wikisim. Install and Test the above package on your computer and show that you can run simple similarity between pairs, and wordsense disambiguation. Report the evaluation results for Senseeval-2 dataset. To comprehend the approach you can study their paper “Vector Space Representation of Concepts Using Wikipedia Graph Structure” – Alternative strategies should be discussed with your teacher if failed to execute Wikisim as the package is not stable
5) We would like to use the vector representation generated by WikiSim to evaluate the similarity with human judgment on selected dataset. For this purpose, refer to the above mentioned paper, and taking into the general dataset (MC, RG, WS353) for which the ground truth is known, and compute the correlation of the words of each pair of the dataset with that of human judgment. Compare this with word2vec, Glove and FastText correlation value.

6) We would like to generate a new vector given as a convex combination of WikiSim generated vector and FasText vector. Start by a convex combination of factor 0.5 and computer the new correlation value for the aforementioned dataset. Then start to see various combination of the convex combination to see if any improvement on the results occurs.
7) Design a GUI of your choice to demonstrate your findings

Project 31: WordSense Disambiguation in Medical Context

This project aims to contribute to Word Sense Disambiguation in the domains of biomedical and clinical text.

1) Initially study again Lesk implementation in NLTK. Consider the sentence "I have been prescribed two important drugs today during my visit to clinic" and assume that the target word is "drug". Construct a simple script that outputs the correct sense of "drug" in the above sentence using standard Lesk algorithm. And, comment on the outcome the efficiency of the result.
2) Download and study MSH-WSD dataset from https://github.com/clips/yarn. The database contains 203 ambiguous terms that include regular terms, acronyms. Make a random selection of 10 acronyms and 10 regular terms. Use Lesk algorithm to perform the wordsense disambiguation of each of these 10 acronym and 10 regular terms and use the ground truth to compute the average accuracy on regular terms and average accuracy on acronyms
3) Study the various disambiguation algorithms implemented in https://github.com/alvations/pywsd. Show that you can run the program on a simple sentence containing a target word to disambiguate and output for various disambiguation algorithms (Original Lesk, adapted/extended Lesk, simple Lesk extended with hypernym and hyponyms, cosine Lesk, path-semantic similarity, Information content-semantic similarity, highest Lemma count.
4) Run the above various algorithms in 3) on MSH-WSD dataset and report the average accuracy on all acronyms and average accuracy on all regular terms.
5) We would like to extend similarity-based Lesk algorithm to include phonetic matching, which evaluates phonetically similar wording. 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/. Demonstrate this reasoning on the example sentence pointed out in 1) and generate a matrix that indicates the phonetic distance between each pair of words of the sentence.
6) We would like to extend the Wu and Palmer-semantic similarity extension of Lesk algorithm implemented in 3) by incorporating the phonetic similarity as well so that the overall similarity is computed as a convex combination of Wu and Palmer similarity and phonetic similarity (after being normalized) with coefficient of 0.5. Suggest and an implementation and run your new code on selected MSH-WSD dataset.

7) Suggest a GUI of your choice that allows the user to visualize the various outputs of the disambiguation task. Ideally the use can input a sentence and a target word to be disambiguated.

Project 32: 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 extractive based summarization method 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. You may inspire from the Summarizer toolkit provided in the Moodle section of the course.
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. Use ChatGPT API (you may inspire from implementation of Summarizer toolkit provided in Moodle section of the course) as a basis to generate 10 different summarizers of the same original text that you used previously. Then use these summaries as golden summarizes to quantify the summarizer generated in 4) using ROUGE-2 and ROUGE-3.
6. Consider the Opinosis dataset available at https://kavita-ganesan.com/opinosis-opinion-dataset/#.YVw6J5ozY2x, which contains sentences extracted from user's reviews on a approximately 51 topics, each having around 100 sentences on average, and includes gold standard summaries. Test

the performance of TextRank, Latent Semantic and Relevance sentence scoring on this dataset in terms of Rouge-1 and Rouge-2.

7. 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). Test the program in terms of Rouge-1, Rouge-2 score for Opinosis dataset.
8. Now we want to modify the implementation in 6) to account for topic of document in light of the structure of Opinosis. Suggest an approach how to achieve this goal and script that implements your approach. Test the result of the summarizer on Opinosis dataset and CNN/dailymail dataset.
9. Identify relevant literature that allows you to comment on the methodology and results of your implementation.
10. 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.

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). 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/