**List of Potential Projects Data and Text Mining Course 2021**

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

**Project 1: Poetry Analysis Using NLP**

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

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

1. Use appropriate NLTK coding (you can inspire from coding examples of the online NLTK book) in order to plot the histogram of the thirty most frequent words in each ebook. Save the result on excel file. Comment on the nature of these words in terms of their relevance to the content of the ebook. Suggest your own approach to evaluate such relevancy (think of title of book, title of chapters and how this match with most frequent words).
2. Use the frequency of the words to fit the Zipf distribution for each ebook. You can imitate the example of <https://finnaarupnielsen.wordpress.com/2013/10/22/zipf-plot-for-word-counts-in-brown-corpus/>. Draw the zipf fitting cure for each ebook and comment on the goodness of fit.
3. Identify Part-of-speech tagging of all words of the two ebooks using NLTK part-of-speech tagger and save the result on an Excel file. Trace the frequency of the various of part-of-speech tags. Find out whether a Zipf law can be fitted for each book. Explain your reasoning using statistical evidence.
4. We would like to find out the structure of the poems in terms of length of each line of the poem. For this purpose, write down a program that calculates the length of each line of the poem in terms of number characters. Save the result in an excel file. Next, plot the frequency of the various length values. Find out whether a polynomial, exponential, Zipfs law fitting can be achieved? Motivate your answer. You can inspire from examples in [Curve Fitting With Python (machinelearningmastery.com)](https://machinelearningmastery.com/curve-fitting-with-python/), or library spacy.stats.linregress or examples in <https://towardsdatascience.com/how-to-perform-linear-regression-with-confidence-5ed8fc0bb9fe>.
5. Now we want to investigate the structure of poem in terms of category of starting word and ending word in each line of the poem. Investigate the variation of the part-of-speech tag of starting and ending word (excluding punctuation characters) across all lines of the poem in both ebooks. How this variation takes place across different chapters. Use corresponding illustrations to justify your answers. Comment on the phonetic compatibility of poem in each ebook.
6. We want to investigate the variation of the sentiment across lines of poem and chapters. For this purpose, use NLTK sentiment analyzer of your choice or other sentiment analyzers defined in other python libraries to calculate the overall sentiment of each line of the poem as well as sentiment of starting word and ending word of each line. Save your response in an excel file. Use illustrations of your choice to visualize the sentiment across lines and chapters, and another illustration to visualize the possible compatibility of starting and ending word in each line of the poem for both ebooks.
7. We would like to compare the usage of negation for each ebook. For this purpose, write a program that allow you to identify the occurrence of negation in the text where the negation is identified through a set of preselected terms (i.e., single word like not, none, or as affix as in can(n‘t), (im)perfect)) using string matching method. Write a program that identifies the occurrence of negation in every set of five consecutive lines of the poem. Compare the frequency of occurrence of negation in each ebook. Use the code to identify the type of preposition tagset (use Part-of-speech tag) that occur in the line where the negation is detected. Plot the frequency of each preposition tag in each ebook.
8. We would like to compare the phonetic of starting word and ending word of each line of the poem. For this purpose, use the “Fuzzy” library in python to generate the character string that identifies phonetically similar words, and then use edit distance to compute the phonetic similarity between two words. See an example of implementation at <https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/>. However, leave the size of the generated string open. The distance between the phonetic generated vectors corresponding to starting and ending word is generated in the following way: Assume L1 is the first word phonetic string, and L2 is the second word phonetic string generated by Fuzzy library. Then the phonetic association Sim(L1,L2)= 2\*S/ (length(L1)+length(L2))

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

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

1. 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 2: 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, John Keats [Keats: Poems Published in 1820 by John Keats - Free Ebook (gutenberg.org)](https://www.gutenberg.org/ebooks/23684), but feel free if you want to use another one, provided the author of eighteen centuries to enable comparison with recent trend.

1. Consider the brown corpus that you can access using NLTK (see example in Online NLTK book), which is already annotated using part-of-speech tagger. Plot the graph showing the frequency of words in the corpus. Save the result in a database. Next, consider the part-of-speech tag of each word in the corpus as annotated, similarly plot the frequency of the different tags in the corpus.
2. Now repeat this reasoning with the poem. Use appropriate NLTK modules to draw the word frequency after a stemming operation using WordNet lemmatizer (available at nltk.stem.wordnet), see examples of codes in Online NLTK book for this purpose. Use Penn Treebank tagset to identify part-of-speech tag of each word of the poem using (nltk.ois\_tag), and then report the frequency of each tagset. Save your results of word frequency and tagset frequency in your database. Draw the plot of word frequency and tagset frequency.
3. Now we would like to assess how much the poem structure deviates from the corpus. First, normalize the word frequency in both poem data and brown corpus. Compute the histogram of matching using the following approach. First consider the set of the five most frequent items in corpus and the set of five most frequent items in poem data, and compute the percentage V(1) of common items between the two sets. Then repeat this process for the next 5 most common items and calculate V(2), and so on, till V(30).. Draw the histograms V of item-matching accordingly. Comment on the type of common items between poem data and brown corpus.
4. Repeat the previous task where the items correspond to the part-of-speech tags.
5. Now we want assess the compatibility in terms of bi-gram occurrence. For this purpose, use appropriate NLTK package to tokenize all bi-gram in the brown corpus, calculate the frequency of occurrence of each bi-gram and save the result in your database. Repeat this process for poem data. Next, calculate the histogram of the matching bi-gram using the approach pointed out in 3).
6. We would like to test the monosense usage in the structure of poem. For this purpose, write a program that uses WordNet to retrieve the number of senses for every inputted individual word. So, after performing stopword removal and Wordnet Lemmatizer for each word of the poem, calculate the average sense per line (summing up the number of senses each individual word has and dividing by the total number of words used), and save the result in your database. Report also in the database the number of senses for each individual word in the database.
7. Plot a graph showing the sense frequency of individual words (e.g., frequency of words with unique sense, two senses, three senses and so on). Find out whether a parametric curve (polynomial, Zipf) can be fit. You can inspire from examples in [Curve Fitting With Python (machinelearningmastery.com)](https://machinelearningmastery.com/curve-fitting-with-python/), or library spacy.stats.linregress or examples in <https://towardsdatascience.com/how-to-perform-linear-regression-with-confidence-5ed8fc0bb9fe>.
8. We now consider the lexical diversity LD in the poem. This can be expressed using the adjective/adverb-to-verb ratio (number of adjectives and adverbs to the number of verbs) in each line of the poem. Using part-of-speech tagging that identifies verb, adjective and adverb entities, suggest a program that calculates LD for each line of the code. Save the result in the database. Plot the graph of LD. Suggest a 10-equal subdivision of values of LD (take the highest value of LD and subtract the smallest value of LD and divide by 10 to find the bin value, and then take the smallest of LD and add bin, then 2\*bin, etc.. to find the next interval (You will end up with 10 intervals). Now calculate and plot the corresponding histogram (calculating the number of lines of poem whose LD value fall within a specific interval). Find out whether a parametric fitting (polynomial, logarithmic or exponential) can be achieved.
9. Now we want to test the distribution of the most common words in the poem. For this purpose, consider the three most common verbs, 3 most common adverbs and 3 most common adjectives in the poem. Draw the plot of the lexical dispersion of each of these words. This correspond, for a given word, to a plot where you show positions of that word from the starting token of the poem. You may look at existing NLTK lexical dispersion plot available online.
10. We would like to study the emotion of the poem in terms of valence and arousal. See the implementation in [GitHub - bagustris/text-vad: VAD analysis of text using some affective lexicon (ANEW, SENTIWORDNET, and VADER)](https://github.com/bagustris/text-vad), or in the worst case scenario, you may restrict to ANEW manually annotated dataset for valence and arousal, and calculate the value of valence and arousal in each line of the poem and repeat the previous process of plotting the graph showing variations of valence and arousal across all lines of poems. Next perform a 10 fold subdivision taking into account the smallest and highest values and draw the histograms, then try to fit the corresponding parametric curve.
11. Run the FuzzyWuzzy program [fuzzywuzzy · PyPI](https://pypi.org/project/fuzzywuzzy/) for string matching to find out the ratio of string overlap (O) between two consecutive lines of the poem. Plot a graph showing the variation of O’s values. Create a subdivision of 10 equal bins taking into account the smallest and largest value of O’s values. Then calculate and draw the corresponding histogram, and see whether a parametric (polynomial, exponential, logarithmic, Zipf) can be fitted.
12. Use appropriate literature to discuss your findings.

**Project 3: Metaphor detection in poetry**

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

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

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

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

**Project 4: Suomi24 Corpus Analysis**

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

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

Select University of Oulu in Korp and then login with your university credential to have access to the data. The data is quite large 65GB, so you should make sure that either your local computer or cloud server account will be able to handle it and run basic search. Select two antagonist keywords of your choice that are likely to actively occur in Suomi24, e.g., hate and love; buy and sell; …. Now we want to investigate the occurrence of this antagonistic relation over years.

1. We want to track the occurrence of your chosen antagonistic words, in threads only of Suomi24 (not taking into account users’s replies and comments). Write down a program that allows you to retrieve the set of threads where either of the two antagonistic keywords occur. Save the thread titles /contents together with their associated dates in your database.
2. We would like to investigate the timely progression of the vocabulary size of the collected threads. For this purpose, separate the original dataset into 16 or 17 sub-dataset dat1, dat2, …, dat17 corresponding to data for 2001, 2002, 2003, …, 2017. Use Vector counter of NLTK to extract the vocabulary of each year and its associated size.
3. Plot graphs showing the evolution of vocabulary size on a yearly basis and comment the results.
4. Plot graphs showing the evolution of the vocabulary size with respect to number of tokens for each year (one graph per year). Find out whether a the parametric Heaps law can be fit. Utilize the confidence value of the prediction to draw upper bound and lower bound of the curve fitting and quantify to goodness of fit. You can inspire from examples in spacy.stats.linregress or examples in <https://towardsdatascience.com/how-to-perform-linear-regression-with-confidence-5ed8fc0bb9fe>. Test different values of the confidence value, e.g., 80%, 85%, 90% and 95% and report the number of points that fall outside the upper and lower bounds for each case.
5. Suggest a script, which identifies, for each year (from 2001 till 2017), the most frequent 10 words where your keyword co-occur most often at 2 units lexical distance (within a window of two words). You should provide 10 words for each keywords of your pair of antagonist words and for each year. Restrict the search to only thread titles/contents.
6. Suggest graphical illustrations, which show the yearly evolution of the ten most frequent co-occurring words, their overlapping and their intensities. Save the result of the yearly evolution in your database.
7. We would like to compare the occurrence of each antagonistic keyword in terms of sentiment. For this purpose, use AFINN sentiment analyzer, which supports several language, including Finnish language and lexicon where each word is assigned a polarity score. See, [afinn · PyPI](https://pypi.org/project/afinn/), for the python library and examples. Write a script for calculating the overall sentiment for each of the two keywords, by individual year. Plot on a graph showing the yearly evolution of the sentiment of individual words.
8. Now we want to use the topic modelling (LDA) for each of the two keywords. Use the LDA implementation in genism library and input, for each year, the collected set of threads associated with a given keyword, with a number of 5 topics and 5 words per topic. Save the results in your database.
9. Use appropriate graphical illustrations to highlight the yearly variation of topics associated to each keyword, and the overlapping among the two antagonist keywords.
10. Use appropriate literature and state of the art work to comment on your results and methodology

**Project 5: Reuter Dataset Classification**

This project focuses on testing various approaches for classification of benchmark Reuters-21578 dataset, which contains 10788 documents, partitioned into a training set of 7769 and testing set of 3019 documents, categorized into 90 categories. The dataset is available from NLTK.

1. Use appropriate NLTK coding to visualize the number of documents per category in the training set. Is the dataset class balanced? Motivate your answer.
2. Construct the feature set using CountVectorizer of the training dataset so that the maximum size of the feature set is 100 and each word feature should be contained in at least 3 documents and in less than 70% of total documents to be considered. Use standard preprocessing and stopword list, with WordNet lemmatizer.
3. Use NaivesBayes classifier to generate the model and test it on the testing dataset. Output the accuracy and F1 measure of this classifier. Compare the result with Random Forest and linear SVM classifiers.
4. Show the results of this classifier when the total number of features varies as 1000, 1500, 2000, 2500, 3000, 3500, 4500, 5000.
5. Repeat 3) when a PCA (of 95% information preservation) is used to reduce the dimension of the feature space.
6. Instead of CountVectorizer, use TfidfVectorizer to create the feature set as in 2) and test the classification accuracy and F1 measure of Naives’ Bayes classifier.
7. Use Word2vec to create a feature vector to each document by averaging over all word2vec representations of individual words constituting the document. Use this feature set to train the Naives’ Bayes classifier and output the accuracy, F1 measure and confusion matrix on testing data.
8. Now we would like to tackle the discrepancy of the number of instances in training data among the various categories. For this purpose, create a new partition, where you remove the categories that contains very large number of instances and very low number of instances as compared to the rest of the categories. Repeat 3) and provide the results on the table.
9. Similarly, use word2vec model to calculate the new performance on the testing data for the new set of categories in 8).
10. Now we would like to handle the data discrepancy in the class, we would like to perform data augmentation on the small size classes. For this purpose, for the classes, which contain small number of instances, write a script that allows you to duplicate the number of small samples at an order close to the average of other classes. Then repeat the process 3) and test whether an enhancement of the results on the testing data can be noticed.
11. Now we would like to test the influence of the multi-category documents on the classification. For this purpose, write a script, which scrutinizes the training set, so that every document, which is found to belong to more than one category in the training set is removed. Repeat 3) and word2vec model to evaluate the performance of the model on the test data, and discuss the contribution of this removal action.
12. Use appropriate literature in order to comment on your findings.

**Project 6: Sentiment Analysis and Machine learning**

Consider the three sentiment annotated dataset available in [mpad/datasets at master · giannisnik/mpad · GitHub](https://github.com/giannisnik/mpad/tree/master/datasets), labelled as positive and negative. We aim to test the transferability of sentiment analysis model from one dataset to another one.

1. Write a script that distinguishes the text of each sentence of the dataset and the label category in each dataset. Save the result in a database of your choice.
2. Write a script that calls SentiStrength package available from sentistrength.wlv.ac.uk to calculate the sentiment polarity of each sentence of each dataset. You can then compare the obtained sentiment with the annotated one. Report the accuracy of the SentiStrength sentiment for each dataset.
3. Perform a small manual check on the misclassified sentences to point to some generic statements about the potential clues in the sentences that rendered the sentiment polarity unfaithful.
4. Repeat steps 2 and 3 when using textblob sentiment analyzer. Compare and comment on the performance of Texblob and SentiStrength.
5. Now we would like to build a machine learning model for sentiment analysis. For this purpose, write and script and review the preprocessing and stopword list to not discard relevant information in the context of sentiment analysis (e.g., avoid discarding negation cues, adjectives that subsumes polarity and apostrophes, lower-case as capitalization brings emotion,..), then use TfIdfVectorizer with a maximum feature set of 1000, minimum 3 repetition and no more than 70% of word repetition across sentences. Build this model for one dataset using randomly selected 70% training and 30% testing. Report the classification accuracy.
6. Test the model generated in 5) to test its performance on the other two datasets. Report and comment on the performance of a such transfer learning approach.
7. Use FastText encoding instead of TfidfVectorizer, see <https://github.com/facebookresearch/fastText/archive/v0.2.0.zip>

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

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

1. Repeat 7) where FastText are re-trained on your dataset used for building the model in 5). You may use Google colab in order to speed the execution of your code as large amount of memory will be required. See [Google Colab - Using Free GPU (tutorialspoint.com)](https://www.tutorialspoint.com/google_colab/google_colab_using_free_gpu.htm) for a tutorial
2. Identify appropriate literature to comment on your findings and methodology.
3. Suggest a GUI of your choice that enables the assessor to comprehensively the achievement of each of the previous project specifications.

**Project 7: Environment keyword mapping**

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

1. Consider the wordings: “nature”, “pollution”, “sustainability”, “environmentally friendly”. 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)](https://stackabuse.com/python-for-nlp-working-with-facebook-fasttext-library/). 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 8: 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 “carbon footprint”. 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 ith 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](https://spacy.io/usage/linguistic-features) 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](https://www.section.io/engineering-education/word-cloud/).
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 9. Semantic similarity and Synonymy relation**

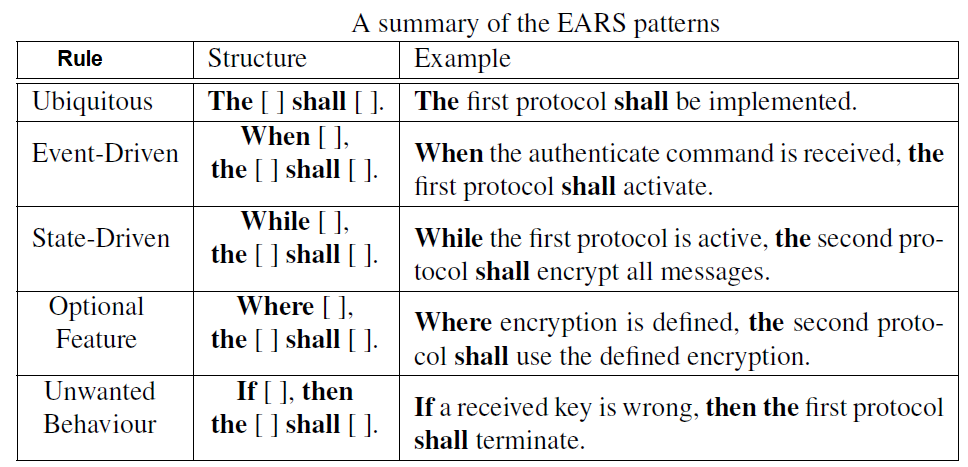
This project aims to investigate the agreement between synonym / antonymy relations and their associated semantic or embedding relationship.

Consider the list of most common synonyms in English, that you can download from [Synonyms for the 96 most commonly used words in English | Just English](https://justenglish.me/2014/04/18/synonyms-for-the-96-most-commonly-used-words-in-english/).

1. First we want to evaluate the agreement of each list of synonym words with the WordNet semantic similarity. Intuitively, one expects two fully synonym words to be semantic similar as well so that their semantic similarity value is one, if normalized within unit interval. However, this intuition is often violated. To test this hypothesis, first manually distinguish verb synonyms and noun synonyms only. This is because adjective/adverbs do not have a hierarchical representation in WordNet lexical database. Then write a program that calculates for each list of synonym nouns or verbs, the Wu-Palmer semantic similarity measure of every pair of words in the list. Write the results on a table. For each list of synonyms, determine the average semantic similarity value and its standard deviation value.
2. Repeat the preceding when you use Word2vec for calculating semantic similarity. In other words, for each word in the synonym list, assign the correspond word2vec vector encoding and then use cosine similarity for calculating the similarity between two pair of words. Save in your database both individual similarity of each pair as well as the average and standard deviation for each list.
3. Now in order to explore the adverb / adjective synonymy list, write a script that allows you, for each adverb or adjective, to retrieve the derivationally related form in WordNet and then select the noun or verb forms. Next, calculate the Wu-Palmer sematic similarity value of the underlined nouns/or verbs (whatever entity is available from derivationally related form). If both verb and noun exist in the derivationally related form of the adjective/adverb from both words of the pair, select the category (verb or noun word) that yields maximum Wu-Palmer similarity. Use this approach to derive for each adjective in the synonymy list, the corresponding noun/noun, if available, (if not available just ignore the pair or feel free to come up with another solution to turn adjective/adverb into their corresponding noun or verb) and use the Wu-Palmer semantic similarity to test the validity of the synonymy relation. Calculate the average and standard deviation of the semantic similarity of each synonym list. Comment on the observation your made. Repeat the preceding when using Lin similarity, which uses corpus based statistics instead of WordNet hierarchical structure only. This is also implemented in NLTK package. Repeat also this reasoning when you use word2vec cosine similarity metric as a basis to calculate the similarity.
4. We want to test the hypothesis whether synonyms who have close popularity are likely to yield close semantic similarity more than synonym which are unequally popular. For this purpose, write a script, which uses brown corpus to identify the popularity of individual word in the synonymy list. Next, write a script that re-order the pairs of synonyms according to the closeness of their frequency in the corpus. Check whether the similarity values (obtained using Wu-Palmer measure, Lin measure or cosine similarity in case of word2vec representations) positively correlate with the closeness in their popularity (the latter can be defined for instance as the inverse of difference in the popularity of the two synonym words). Use Pearson correlation coefficient to evaluate the correlation value and its p-value indicating the statistical significance.
5. Consider the list of antonym words available at [Common Opposites - Antonyms Vocabulary Word List - Enchanted Learning](https://www.enchantedlearning.com/wordlist/opposites.shtml). You many manually copy and past the list and save it as a separate file. Write a script, which calculates the Wu-Palmer semantic similarity of each pair of antonym words. Repeat this process when the similarity is calculated according to word2vec cosine similarity metric. Calculate the average, standard deviation, skewness and kurtosis for both Wu-Palmer semantic similarity and word2vec cosine similarity metrics. Comment on whether there is some trend in terms of similarity value and presence of antonyms.
6. Write a script that allows you to re-order the antonym word pairs according to the closeness between the popularity of word pairs in brown corpus. Then calculate Person correlation between this closeness and both Wu-Palmer semantic similarity and word2vec cosine similarity.
7. Repeat 6) when using glove embedding instead of word2vec, see [Basics of Using Pre-trained GloVe Vectors in Python | by Sebastian Theiler | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db) for an example of application.
8. Repeat the steps 5) and 6) when considering the list of common Duos vocabulary list available at [Famous and Common Duos Vocabulary Word List - Enchanted Learning](https://www.enchantedlearning.com/wordlist/duos.shtml).
9. Identify relevant literature to comment on the results and limitations of the approach
10. Suggest GUI that allows you to demonstrate the achievement of the above specifications.

**Project 10. EARS Rule detection**

This project aims to advance in implementing and assessing EARS (Easy Approach to Requirement Syntax) rules employed for software requirement specification, which consists of 6 types of rules, depending on the occurrence of modal verbs, specific prepositions and articles, see example in Table below.



The project aims to use natural language processing-based analysis to identify and monitor the occurrence of such rules in corpora. You may consult short tutorial available in [Microsoft PowerPoint - ICCGI 2013 EARS.pptx [Read-Only] (iaria.org)](https://www.iaria.org/conferences2013/filesICCGI13/ICCGI_2013_Tutorial_Terzakis.pdf).. You may notice for instance that the presence of “shall” alone without presence of When, While, Where, If, does not necessarily entail that the underlined phrase is not Ubiquitous. Especially, Ubiquitous rule exhibits i) a fundamental system property, ii) do not require a stimulus to be executed, iii) universal property.

1. Use a parser-tree of your choice (e.g., Stanford Parser, NLTK Parser, Spacy parser) to write a script that allows the identification of the basic structure of an ubiquitous rule. In essence this involves identifying part-of-speech-tag set, with the focus on presence of preposition “the” preceeding a noun tag, the presence of verb tag “shall” in some context and absence conditional statements, etc.. Motivate your reasoning using selected examples of your choice from the tutorial or other sources of your choice.
2. Consider the FIDE Laws of Chess, which indicates rules of Chess game. This can be downloaded from [chess\_olympiad\_regulations.pdf (fide.com)](https://handbook.fide.com/files/handbook/chess_olympiad_regulations.pdf), that you can turn to text format. Perform a manual labelling of the rules mentioned in the dataset, focusing on Ubiquitous rules. Now use the script of 1) in order to identify ubiquitous rules from the FIDE dataset, and estimate the accuracy of the system. Comment on the efficiency of the rules implemented for detecting Ubiquitous rule in 1).
3. Explore the implementation available in [GitHub - armsp/template-conformance: Quick implementation of conformance to requirements template like EARS, RUPP & AGILE user story](https://github.com/armsp/template-conformance) where a python script for identifying all EARS rules is available. Test the script for identifying Ubiquitous rules from FIDE dataset, and comment on the accuracy performance.
4. We would like to find out how the accuracy of the identification varies with syntactic and surface-level. For this purpose, write a python script that outputs, for each statement rule of FIDE dataset, i) the number of words of the statement, ii) the proportion of number of distinct tokens over the total number of words, iii) the proportion of preposition words over the total number of words, iv) the number of nouns, v) the total number of adjectives, vi) the readability assessment, which is quantified as 4.71(#characters/word) + 0.25( #words/Sentence)-21.43. This is already implemented in python Textstat library <https://pypi.org/project/textstat/>. Summarize in a table the variation of Ubiquitous rule identification with respect to the above criteria.
5. Repeat the process 1-4) for the event driven rule and for state driven rules.
6. Suggest another dataset of your choice where Ubiquitous rules are present to test your method. The dataset should be annotated to distinguish ubiquitous and non-ubiquitous rules.
7. Identify relevant literature to comment on the findings and the difficulty raised by the process of EARS rule identification.

**Project 11: Sentiment Analysis and Hotel Reviews**

This project aims to investigate the sentiment and test various architecture for argumenting the sentiment polarity. One shall consider the hotel sentiment review available at [Trip Advisor Hotel Reviews | Kaggle](https://www.kaggle.com/andrewmvd/trip-advisor-hotel-reviews) which contains user’s review and rating. Download the dataset in an easy searchable database. We would like to compare various sentiment analyzers and contrast evidence from individual raters.

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 D, which contains these information for each review.
2. Repeat 1) using another sentiment analyzer of your choice (i.e., Textblob sentiment, NLTK Vader sentiment), and save the result in the same database D (you may only provide the overall sentiment score if the sentiment software does not provide separate value for positive and negative scores).
3. Draw on the same plot, the sentiment analyzer 1 (SentiStrength), sentiment analyzer 2 (Your alternative choice) and the user’s rating (You can suggest a normalization of the user’s rating to make the display more interesting). Assume that rating 1-2 correspond to negative polarity, 3 to neutral and 4-5 to a positive polarity. Determine for each sentiment analyzer, the correlation between the vector carrying the overall sentiment score and that of the overall rating (Use the formula of Pearson correlation coefficient to calculate the correlation, which is implemented in several Python libraries) and calculate the correlation between vector carrying sentiment score of a given sentiment analyzer (SentiStrength and the alternative one) and the vector carrying the sentiment according to User’s rating.
4. We want to find out whether there is difference between positive and negative polarity set of reviews in terms of correlation. Separate the set of review found to be assigned to positive polarity and negative polarity according to user’s rating, yield Set 1 and Set 2. Re-Calculate the correlation score for both sentiment analyzer 1 and sentiment analyzer 2 in Set 1 and Set 2.
5. We want to further explore the content to see the presence of potential cues in the text review that could distinguish positive and negative review. Write a script that outputs three variables i) the number of tokens in the review, ii) the number of pronouns and iii) the number of named-entities. Calculate the correlation of each of these variables with the sentiment score and comment on the result.
6. Consider the list of wording that induce explanation (e.g., because, in view of, for the reason, for the purpose, since, as…)- identified from a source of your choice. We would like to test the hypothesis whether the negative sentiment entail more argumentation.. and.. whether positive sentiment entails argumentation as well.. Use the same reasoning as above by construction a binary vector showing the presence of absence of explanatory related wording in the reviews.. Then calculate the ratio of the presence of such explanatory wording in Positive reviews.. and the ratio in the negative reviews.
7. We shall consider another illustration of the content of the text as provided by categories generated by Empath Client <https://github.com/Ejhfast/empath-client>. Apply Empath Client to each review data and record the categories for which the associated score is non-zero. Report these categories in the database D.
8. Construct histogram of Empath categories showing the frequency of each category for positive sentiment dataset and that pertaining to negative sentiment. Comment on the use of Empath categories to track the sentiment polarity. Scrutinize these categories provided by Empath, and discriminate categories that are intuitively assigned to positive sentiment and those intuitively assigned to negative sentiment. Evaluate the validity of the statement that “the presence of positive sentiment associated Empath category in the review entails a positive sentiment” and “the presence of negative sentiment associated Empath category in the review entails a negative sentiment”. Propose how you will validate such statements using a simple correlation analysis (Pearson Correlation score with p-value).
9. Concatenate all text review corresponding to positive review together (rating 1-2) as Text 1 and all reviews pertaining to negative polarity (rating 4-5) as Text 2. We want to explore the content of Text 1 and Text 2. Use WordCloud illustrations to represent the content of Text 1 and Text 2. Comment whether some dissimilarity can be highlighted.
10. Run Topic modelling using LDA library, choosing 5 topics and 6 words per topic. Comment on possible overlapping between LDA outputs and Empath outcomes for both positive and negative posts.
11. We would like to investigate the discrepancy (contradiction) between the Sentiment polarity induced by either the Sentiment Analyzer 1 or 2 and the User’s rating. Use a thresholding technique (setting up a threshold to distinguish positive and negative rating based sentiment and another threshold to decide whether the difference between the sentiment analyzer score and that of user’s rating is significant). Add an attribute in the database D to indicate whether a given review belongs to this Ambiguous Class. We would like to test the hypothesis that “badly written reviews are likely to be included in this ambiguous class”. We would like to use a simple quantification of badly written review by counting the number of words of the review who have not an entry in standard dictionary. Use an example dictionary of your choice (e.g., WordNet, Merrium dictionary,..) to test this hypothesis as before using simple ratio based reasoning.
12. Repeat 11) when we would to test the hypothesis that ambiguous reviews are likely to be short. So, construct a vector containing the cardinal of each review report and see whether the above claim is sustainable.
13. We would like to see to test the hypothesis that ambiguous reviews have bad readability. For this purpose, consider the Automated readability Index, which expresses the Readability of review as 4.71(#characters/word) + 0.25( #words/Sentence)-21.43. This is already implemented in python Textstat library <https://pypi.org/project/textstat/>. Calculate the automated readability index for each review and add it to database D. Then, test whether poor readability value entails high probability to belong to ambiguous class.

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 12: Alphabet Ring Analysis**

This project aims to explore how the alphabet letters compare when using Google search engine.

Consider again the process of extracting and identifying Google snippets corresponding to a given user’s query as detailed in Project 8. Now we want to the query will be each of alphabet letter (A, B, C, …, Z) and we output the first 10 snippets.

1. Write a script that allows you to retrieve the 10 snippets of each alphabet letters and the search count number corresponding to the total number of search outcomes for the given alphabet query.
2. Suggest a script that would allow you to cluster the 26 alphabet letters according to their popularity in terms of search count. This should only rely on the Search count without extra user’s manipulation in terms of additional queries to be requested. Therefore, simple thresholding techniques, mean and dispersion cues should be used. You may inspire from …..
3. We want to evaluate the similarity between the letters according to the similarity between the generated snippets. Write down a script that uses the FuzzyWuzzy to compute the string matching score between two alphabet letters as the maximum FuzzyWuzzy similarity score of all snippet pairs (pi,pj) where pi corresponds to a snippet of one letter, and pj corresponds to snippet of the other alphabet letter. Generate the similarity matrix, showing for each pair of alphabet, the corresponding similarity according to the above construction. Save the result in a database.
4. Now we want to calculate the similarity between the snippet pairs using TfIdf instead of FuzzyWuzzy. For this purpose, concatenate all snippets from all search outcomes to form all documents corpus. Use the TfIdf Vectorizer to generate vector representation of each snippet and use cosine similarity to calculate the similarity of each pair snippet. Assume as in 2) that the semantic similarity score of two alphabet letter is determined as the maximum score over all pairs (one snippet belongs to one alphabet letter and the other snippet to another one). Generate the corresponding alphabet similarity matrix. Save the result your database.
5. We want to repeat the calculus of the snippet similarity using word2vec cosine similarity. For this purpose, input each snippet to the pretrained word2vec module and output the encoding vector of size 100. Therefore, use the cosine similarity score to compute the similarity score between two snippets. Use this reasoning to build the alphabet similarity and save it in the same database.
6. Repeat 5) when FastText embedding is used instead of word2vec. (See link of use of Fasttext in previous projects).
7. Now we want to cluster the outcomes of these findings into 5 clusters. Write a script that uses k-means clustering algorithm in order to cluster the alphabet letters into 5 clusters, according to FuzzyWuzzy, TfiDF, Word2vec and FastText based similarity. Especially, list the alphabet letters assigned to each cluster according to each similarity based approach.
8. Next, we want to compare the similarity based analysis with phonetic based evaluation. For this purpose, use the phonetic evaluation using the Fuzzy library described in Project 1, for instance, to calculate the phonetic similarity between each pair of alphabet letter. Save the result in your database.
9. Now suggest an approach that computes the correlation score between phonetic evaluation and similarity using FuzzyWUzzy, TfIdf and word2vec and FastTest.
10. Use appropriate child learning literature to comment on the findings
11. Suggest a GUI that enables fast assessment of your deliverables.

**Project 13: WordSense Disambiguation 1**

The project consists in implementing a new scheme of wordsense disambiguation using Python NTLK, wordnet and supervised classification.

1. Consider a simple example sentence “Next month, this plant will reach its maximum production capacity after months of repair”. Follow the example in the lecture handout to write a script that uses wordnet lexical database and lesk algorithm to disambiguate the sense of target word “plant” in the above sense.
2. Now we want to use semantic similarity instead of Lesk algorithm, write down a script that uses a part-of-speech tagger of your choice (e.g., NLTK tagger) that uses the following reasoning. First, if the target word is either a verb or a noun, then use Wu and Palmer wordnet semantic similarity to calculate, for each sense of the target word, the average semantic similarity between of that sense with every other word of the same part of speech in the sentence. Second, if the target word is not a noun or a verb, then use the derivationally related form in WordNet to identify the corresponding noun or verb, and repeat the above reasoning. Illustrate the functioning of this algorithm on the provided example and on another example on your own where the target word is rather an adjective.
3. Now instead of using wordnet database, we want to use merriam webster’s dictionary that you can access using the free API in [Merriam-Webster Dictionary API](https://dictionaryapi.com/). For this purpose, first uses Spacy parser to tokenize and identify part of speech of each word in the above sentence. Then retrieve each gloss defining each sense of the target word. Next, use the following interpretation of Lesk algorithm to disambiguate the target word “plant”. Write a script that retrieves the various senses of the target word with the same part of speech tag as that identified by Spacy parser. Next, calculate the total number of common words between each sense gloss and the words of the original sentence containing the target word to be disambiguated. The sense that yields the largest number of common words will then be considered as the correct sense of the target word. If there is no intersection between any of the sense gloss of the target and the word of the original sentence, expand the words of the sentence (except the target word) by the synonym list of each word according to merriam webster’s dictionary API and repeat the process again. If there is still no intersection then pick up a sense at random. Illustrate the above reasoning on the original sentence in 1) and other example of your own.
4. Now we want word embedding word2vec as a basis for wordsense disambiguation. Write a script that output for each target word, with a given tag, the gloss of every sense (also referred as signature of the sense in the lecture handout) according to wordnet. Assume the word2vec of a given sense calculated as the average word2vec of the words included in the signature of that sense. Next consider a phrase T constituted of all words of the original sentence except of target words. Use word2vec to output the encoding vector of T. Next calculate the cosine similarity of the word2vec vector of each signature and the encoding vector of T, and deduce the correct sense as the one that maximizes the cosine similarity. Illustrate the above reasoning on the original sentence of 1) and another sentence of your choice.
5. Now we want to extrapolate the above reasoning on Semeval2 dataset that you can import in NLTK package, see also see, e.g., [Ted Pedersen - Sense Tagged Text (umn.edu)](https://www.d.umn.edu/~tpederse/data.html)) for a general description of the data. First use NLTK to retrain word2vec with training set of semeval.
6. Choose a testing example of your choice in Semeval2 corpus, use the pretrained word2vec as in 4) and the signature of each sense of the target word to disambiguate the correct sense of the target word as in 4). Repeat this process when the word2vec are re-trained on the Semeval2 training dataset.
7. Repeat 6) for all target words in the testing set of Semeval2 and estimate the accuracy level.
8. Suggest a GUI that would allow you to test the achievements of the above specifications.
9. Identify appropriate literature to comment on your findings and pipeline.
10. Suggest a simple GUI that allows you to demonstrate the achievement of the above specification

**Project 14: WordSense Disambiguation in Medical Context 2**

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 15: Semantic Similarity 1**

This project aims to investigate the similarity between two phrases using internet search results only.

1. We want to study a new WebJaccard similarity –see paper “WebSim: A web based semantic similarity measure” in Proc. of the 21st Annual Conference of the Japanese Society of Artificial Intelligence, 2007. More specifically, for two words P and Q, the similarity Sim(P,Q) is calculated as follows

Sim(P, Q) = N11 / (N11 + N10 + N01)

where N11 corresponds to the page count for the query “P AND Q), or equivalently, the number of results when the “P AND Q” is inputted to the search engine (i.e., Google search, yahoo search). N10 corresponds to the page count of the query “P AND NOT Q”. N01 corresponds to the total page count of the query “B AND NOT A”. Suggest a script that implements the above Web-based semantic similarity using Google Search API or yahoo search API.

1. Now we would like to see how the above webSimilarity behaves when dealing with special case wording. For this purpose, suggest 30 words P in environment science, and another set of words Q such that pairs (P,Q) are such that i) P is synonym to Q; ii) P and Q exhibit antonym relation; iii) P is a direct hyponym of Q. (words Q maybe of course different in each category). For each of the three scenarios, report in a table the result of the WebSimilarity values for each of the three cases, the associated mean value and standard deviation.
2. Use wordnet library to calculate the Wu & Palmer, path length and Leacock Chodorow semantic similarity of the above pairs. Report the results in the table. Compare the result with that of WebSimilarity and comment on the agreement with the intuition. You can add extra examples to justify your choice.
3. Now we want to use Snippet instead of page count. For this purpose, use Google Search API (should register to get a key), and collect snippet of the search outcome, see an example in <https://www.thetopsites.net/article/52274742.shtml>. Provide a small pseudo code showing your implementation steps and an example of an answer.
4. Consider a selection from the pairs suggested in 2). Retrieve the first five snippets for query P and the first five snippets for query Q. Then one assumes that Sim(P,Q) is calculated as the ratio of the common words between the five snippets of P and the five snippets of Q over the total number of distinct words of all the ten snippets, after appropriate tokenization, stemming and noise removal in the original text. Compare the results with that of wordnet similarities and comment on the appropriate. Assume such similarity is called Sim\_Snippet1.
5. Consider three queries of your choice which carry opposite or contrasting effects, e.g., group activity versus individual activity; hate speech versus harmonious talk; democratic regime versus authoritarian regime. For each query (of the six queries), retrieve the first 10 snippets. For each contrasting query, report i) the number of snippets which are shared between the two queries; ii) the amount of overlapping between the two set of snippets. For the purpose of ii), you may concatenate all snippets of the first query into a single document D1 and that of the second query into another document D2, then use fuzzy-string matching <https://www.datacamp.com/community/tutorials/fuzzy-string-python> to generate the percentage of the overlapping. The latter uses edit distance and takes into account all possible alignments of D1 and D2. Summarize the result of the three double queries in a table. Comment on whether Google search tends to discriminate or re-assemble contrasting queries. The newly constructed similarity is called Sim\_Snippet2.
6. Apply Sim\_Snippet2 to the same examples used in 5) and compare the results. Comment on the soundness of each approach with respect to commonesense.
7. Consider the datasets of word pairs whose similarity is manually annotated, especially MC-28, Word-Sim, RG, available at <https://github.com/alexanderpanchenko/sim-eval>. Similarly to the work on this repository, we would like to test the usefulness of any new similarity measure by computing its correlation with human judgment (using Pearson coefficient).
8. 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 16: Semantic Similarity 2**

We aim in this project to study a new semantic similarity for two short text document

1. Consider two sentences S1 and S2 which are tokenized for instance, say S1=(w1, w2,..,wn) and S2= (p1, p2, …, pm). Consider the approach implemented in Lab2 for calculating the semantic similarity between sentences using WordNet semantic similarity. We want to test this strategy on publicly available sentence database. For this purpose, use STSS-131 dataset, available in “A new benchmark dataset with production methodology for short text semantic similarity algorithms” by O’Shea, Bandar and Crockett (ACM Trans. on Speech and Language Processing, 10, 2013). Use Pearson correlation coefficient to test the correlation of your result with the provided human judgment.
2. We want to introduce other modification on this semantic similarity. For this purpose, suggest a script that proceeds in the following. For sentence S1 and S2, first, use Spacy to identify part of speech tag of each token of the sentences and all named-entities in S1 and S2. We want to discard those named-entities from the calculus of sentence to sentence semantic similarity. Next, use WordNet derivationally related form to turn each verb, adjective /adverb token into its corresponding noun token. Then calculate the sentence-to-sentence semantic similarity as the average of Wu-Palmer semantic similarity calculated over all pairs (N1i, N2j) where N1i and N2j stand for ith and jth noun-token of S1 and S2, respectively. Report the results of this semantic similarity on STSS-131 dataset and the Pearson correlation with human judgment. Comment on the overall process whether it yields improvement and its limitations.
3. Now we want to implement a new semantic similarity-based measure. The idea is to use some hierarchical reasoning and explore the WordNet Hierarchy. For this purpose, proceed in the following. For each sentence, use the parser tree to identify various part-of-speech of individual token of the sentence. Generate the list of hypernyms H1 and hyponyms H2 of each noun of the sentence. Repeat this process for each verb. Compute the semantic similarity between the two sentences as

Implement the above similarity expression in your python code

1. Test the above similarity on STSS-131 database and report the Pearson correlation with human judgments.
2. Study another text similarity using both wordnet semantic similarity and string similarity provided in <https://github.com/pritishyuvraj/SOC-PMI-Short-Text-Similarity->. Check the behavior of program for some intuitive sentences (very similar sentences, ambiguous and very dissimilar ones)
3. Report the result of the above similarity on STSS-131 and report the corresponding Pearson correlation with human judgments
4. Suggest an interface of your choice that would allow the user to input a textual query in the form of a pair of sentences and output the similarity score according to the various methods described above.

**Project 17: Semantic Similarity 3**

This project will demonstrate various methodologies for computing sentence-to-sentence similarity scores.

1. Consider the datasets of word pairs whose similarity is manually annotated, especially MC-28, Word-Sim, RG, available at <https://github.com/alexanderpanchenko/sim-eval>. Similarly to the work on this repository, we would like to test the usefulness of any new similarity measure by computing its correlation with human judgment (using Pearson coefficient). Review how Pearson Coefficient works and identify python script to achieve this. Study Datamuse API, which outputs a set of words that are available to a query word. This API is available at <http://www.datamuse.com/api/>.
2. We would like to test the similarity between the pair (X,Y) by using the output of the Datamuse API for both X and Y. Set the number of outcome words in the API to be large, e.g., 100. Use Jaccard similarity to compute the similarity between X and Y (Counting the ratio of common words among the outputs of X and Y Datamuse API over the total number of distinct words in the two outputs).
3. Repeat this process of calculating the similarity between each pair in MC-28 dataset, and then calculate the correlation coefficient with the human judgment using Pearson coefficient. Try to optimize the parameters of Datamuse API call by testing distinct number of outputs and monitor the value of the correlation until you reach the highest correlation value. Use this configuration to calculate the correlation value for other datasets, and compare the result with other state-of-art results as reported in relevant literature (e.g., previous sim-eval repository) .
4. We would like to test the above strategy at sentence level as well. For this purpose, given sentence S1 and S2, which are tokenized for instance as S1=(w1, w2,..,wn) and S2= (p1, p2, …, pm). Then, the representation of S1, will consist of the overlap of the Datamuse output of each individual token w1, w2,…wm (It is important to set the number of outputted words per API call high in order to increase the chance of overlapping), add to this list the tokens of S1 as well. Repeat the same process for S2 and then compute the similarity between S1 and S2 as jaccard similarity of the representation of S1 and representation of S2. Write a simple python code that allows you to achieve this.
5. We want to test this strategy on publicly available sentence database. For this purpose, use STSS-131 dataset, available in “A new benchmark dataset with production methodology for short text semantic similarity algorithms” by O’Shea, Bandar and Crockett (ACM Trans. on Speech and Language Processing, 10, 2013). Use Pearson correlation coefficient to test the correlation with the provided human judgment.
6. Repeat the process of sentence similarity of STSS-131 but using word2vec embedding as in the labs assuming the vector representation of a sentence is the average of the word embedding vectors of all tokens in the sentence. Calculate the corresponding Pearson correlation as well.
7. Repeat 6) when using Glove and FastText embedding. Comment on the overall performance of Datamuse API based approach for text similarity.
8. Study the programs in <https://github.com/gsi-upm/sematch> they attempt to provide other individual similarity but using other lexical database, not necessarily WordNet. Accommodate the provided files into your compare and repeat the calculus of Sim(S1,S2) for each pair of sentence where the individual similarity Sim(s,t) is calculated using YAGO concepts or DBPedia concepts freely available from the program. Compare the behavior of the result.
9. Suggest GUI of your own that facilitate the demonstration of your finding above

**Project 18: Climate Change News Analysis. Discovering Arguments**

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

**Project 19. Analysis of Brexit landscape in UK political events using Twitter**

Consider The Twitter dataset of UK Brexit available at Brexit Data Project BDD in Kaggle, [Brexit Data project BDD | Kaggle](https://www.kaggle.com/natmonkey/brexit-data-project-bdd). The dataset includes both raw text data and cleaned text data. You may look at examples in NLTK online book of script handling tweet dataset.

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

**Project 20: Automatic Text Summarization 1**

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

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

**Project 21: Automatic Summarization 2**

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

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

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

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

**Project 21. Hate Speech detection**

This project aims to shed the light on identifying hate speech from free text and test few implementations.

1. Consult the database of hate speech database maintained at <http://hatespeechdata.com/> . Consider the dataset CONAN in the database, which has 1% abusive content among 1,288 posts. We would like to investigate the structure of the dataset in terms of categories present. Consider the subclass S1 of abusive content and subclass S2 of non-abusive content. Draw a wordcloud representation of S1 and S2.
2. Use Empath categorization and output for each message in S1 and S2, the list of categories who have non-zero weight. Report the result in a database that you will provide as a project deliverable. Conclude whether Empath categorization can be considered as a discriminative feature to recognize hate content from non-hate content.
3. Repeat 2) by using Harvard General Inquirer available in <http://www.wjh.harvard.edu/~inquirer/inquirerbasic.xls>
4. Suggest a simple heuristic that uses Empath categorization and General Inquirer that would allow you to identify the presence of hate content.
5. Evaluate the performance of this heuristic using ground truth of CONAN dataset. You can also test this heuristic in another dataset of your choice of hatespeechdata.com
6. Study the implementation available at <https://github.com/pinkeshbadjatiya/twitter-hatespeech> of the paper “Deep learning for hate speech detection Tweets” by Pinkesh Badjatiya (www’17 proceedings, 2017) and demonstrate that you closely reproduce the performance claimed by the authors in their paper. If not, comment on your findings accordingly.
7. Study the implementation highlighted in <https://github.com/younggns/comparative-abusive-lang> but when using dataset of 5) or 4), report the corresponding accuracy, F1 and precision/recall.
8. Study the paper “Improving Hate Speech Detection with Deep Learning Ensembles” by Zimermann et al. LREC 2018), which uses the same dataset (Racisim, Sexism and none) and its implementation available at <https://github.com/stevenzim/lrec-2018>. Demonstrate that the program is working appropriately, and discuss its performances with respect to the two other previous implementations.
9. We would like to test the above four implementations (three above papers and heuristic) using different datasets. For this purpose select another dataset from other hatespeechdata.com repositories and check that your program can run successfully.
10. Suggest a GUI of your own that allows us to exemplify the different steps above.

**Project 22 Medical image captioning**

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

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

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

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

**Project 23: Text Categorization – ACM Classification**

Study the ACM classification available in <https://www.acm.org/publications/class-2012>. Create hierarchical classes that fit the description of the ACM classification.

You can also study the good tutorial at <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/> on text classification..

The project aims to design a system that enables us to classify a technical document to its best ACM category match.

1. First, implement a program that input words used in the ACM classification document and output list of synonyms, inflected words and potential technical abbreviations. For instance, for word “classification”, the possible output is “class, classify, classifications, classes, classified, category, categorization, categorize, taxonomy, …” You may use the PyThesaurus API (available at pypi.org/project/py-thsaurus/) for the outputting the list of synonyms.
2. We would like to test the classification scheme on a set of ACM documents. Typically, in standard ACM documents, the paper already includes the classification as part of the meta data associated to the underlined document. We would like to test how this association is reflected when we take into account other meta-data of the document and its content. This provides us with efficient tools to quantify the extent to which authors are consistent with their labeled classification. First, select a topic of your choice in ACM library and download hundred of ACM documents (you can restrict to title, author, abstract, keywords and classification) using any available API, or even copy and past.
3. We would like to quantify the extent to which the title agrees with ACM classification. First, design a program that calculate the Jaccard distance between the title words and each of the classes of the ACM classification document and infer the classification that minimizes the Jaccard distance. Provide different variations of the Jaccard distance depending on the extent of word inflections, synonyms employed. Test the accuracy of the result given the ground truth provided by the classification metadata
4. Repeat 3) when you take into account, list of keywords, keywords+title, abstract, abstract+keywords+title.
5. Now we would like to build a classifier that learns the classification type using a subset of collected documents as a training dataset. Use Df-idf as feature set and test various classifiers (Bayes, SVM, ..). Suggest alternative set of features and evaluate the feature relevancy using appropriate toolbox from Gensim and elsewhere. Provide a comparison analysis of the various feature set, classifiers.
6. Repeat 5) using word-embedding features; namely, each word is substituted by its word2vec vector representation (300 dimension vector) using pre-trained model. For a phrase constituted of a set of words, the vector representation corresponds to the mean vector of the word2vec representation vector of individual words constituting the phrase.
7. Extend the previous approach on a selected articles when the whole article is downloaded. Design your own metric to support the comparative analysis.
8. Design a simple interface that allows the user to test the input a text query, retrieve document from ACM library and test the results of the

**Project 23Bis: Text Categorization – ACM Classification**

Study the ACM classification available in <https://www.acm.org/publications/class-2012>. Create hierarchical classes that fit the description of the ACM classification.

You can also study the good tutorial at <https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/> on text classification..

The project aims to design a system that enables us to classify a technical document to its best ACM category match and evaluates the extent to which the user’s inputted keywords match the system classification.

1. For this purpose, we want to track the evolution of COVID-19 publication trend since its first occurrence in ACM library. For this purpose, perform a check on ACM digital library with keyword search “COVID-19” on monthly basis (check the record for 1st till 31 December 2019, then from 1st January to 31 January, 2020, then from 1st February till 28th February 2020, .. until 1st September 2021 till 30th September 2021). For each month, store i) the number of search outcomes, and, the following attributes of the first 30 articles: ii) the title of article; iii) the list of keywords inputted by the authors; iv) the ACM classification (CCS Concepts), v) the abstract of the paper. Save the above attributes in your database. Feel free to perform the preceding using ACM library API or in the worst-case scenario through manual copy & past.
2. Write a script that displays a graph showing the evolution of the number of search outcomes per month since Dec 2019. Display the histogram showing the number of search outcomes in each quarter since December 2019.
3. We want to rely only on the keywords inputted by the users. Suggest a script that allows you to determine the 10 most common keywords among all the collected articles. Draw on the same plot, the evolution of the number of articles associated to each keyword over time. Discuss the results, and comment on the keywords that appear only rarely.
4. Similarly, write a script to displays the histogram of the CCS Concepts (ACM classification keywords), and plot that shows the timely evolution of the five most comment CCS concepts.
5. We would like to evaluate the extent to which user’s inputted keywords match the document content. For this purpose, first, group the articles according to the CCS concept (ACM classification), so that we would like to find out whether some ACM classification shows a better fitting than others. Next, write a script that allows you to determine the proportion of string matching between the title of the article and the user’s inputted keywords. Draw a histogram showing the proportion of match for the main CCS concept categories.
6. Now we would like to enlarge the matching by taking into account the semantically related words. For this purpose, expand the titles of the document by including synonyms words - You may use the PyThesaurus API (available at pypi.org/project/py-thsaurus/) or WordNet lexical database for outputting the list of synonyms. Next, repeat 5) when using the extended title.
7. Now we view the matching in terms of overlapping between text abstract and user’s inputted keywords. For this purpose, we want to take into account the quality of this overlapping. Proceed in the following way. First, generate the tf-idf matrix of the abstract text of the document. Then take the average of the sum of the tf-idf value of each keyword (if the keyword is not in the abstract, its tf-idf is automatically equal to zero). Assume the value of this average as the amount of overlapping between user’s inputted keywords and the abstract. Draw the histogram showing the amount of this overlap for each CCS category.
8. Now we want to quantify this overlapping using word2vec representation. For this purpose, use word2vec available in Gensim library to generate a 300 vector for the whole abstract. Similarity generate the word2vec vector embedding for each user’s inputted keyword, and calculate the cosine similarity between the abstract embedding vector and each keyword embedding, then take the average to yield a single value indicating the similarity between the abstract and the user’s inputted keyword list. Plot the histogram indicating the amount of overlapping for each CCS category.
9. Repeat 8) when you replace the abstract by “abstract and title”.
10. Design a simple interface that allows the user to test the input a text query, retrieve document from ACM library and test the results of the

**Project 24: Analysis of Online Reviews of Parking Mobile apps using NLP and Technology Acceptance Models (TAM).**

In this project, the students will collect Thousands of online reviews from Apple and Google Play stores from across Europe or at least Nordic area, to understand how users react to the two parking apps that run in the Nordic region - **EasyPark** and **ParkMan**. This project will provide insight into user behavior, sentiment toward parking apps and highlight the most important topics regarding users' requests, demands and preferences in terms of parking solutions or technology features.

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

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

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

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

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

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

* 1. Examine the items of surveys and questionnaires found in the following related papers [3][4][5][6][7]; additionally, you may look for more relevant papers and surveys in Google Scholar. Then generate keyword lists for each indicator based on the elements of surveys and questionnaires. For instance: some keywords for **satisfaction** are {Satisfied, fulfill, gratify, meet, beneficial, content, happy, appeasement ……. etc.}.
  2. To augment the list of keywords, write a script that finds synonyms, hypernyms, and hyponyms for each word, by using WordNet database. Review the words manually and remove those which are not relevant.
  3. Perform necessary data processing for the list of keywords and reviews, for instance, Part of Speech tagging and lemmatization. Then classify each review and its data (rating, sentiment, and date) to a particular TAM indicator based on the common words. In the end, it is expected to have 5 data-frames, each one refers to data related to one TAM indicator.
  4. Repeat task 4 for each indicator.
  5. Calculate the Pearson correlation with a P value between different indicators and determine which are most strongly correlated. Highlight the correlations you find. Provide table to summarize your work.

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

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

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

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

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

[4]: Liao, P. W., & Hsieh, J. Y. (2010, August). Using the Technology Acceptance Model to explore online shopping behavior: Online experiences as a moderator. In 2010 International Conference on Management and Service Science (pp. 1-4). IEEE.

[5]: Ashraf, A. R., Thongpapanl, N., & Auh, S. (2014). The application of the technology acceptance model under different cultural contexts: The case of online shopping adoption. Journal of International Marketing, 22(3), 68-93.

[6]: Huang, Y. C., Chang, L. L., Yu, C. P., & Chen, J. (2019). Examining an extended technology acceptance model with experience construct on hotel consumers’ adoption of mobile applications. Journal of Hospitality Marketing & Management, 28(8), 957-980.

[7]: Shukla, A., & Sharma, S. K. (2018). Evaluating consumers’ adoption of mobile technology for grocery shopping: an application of technology acceptance model. Vision, 22(2), 185-198.

**Project 25. Hate Speech/ Cyberbullying detection and negation scope.**

In this project, the student will perform hate speech/cyberbullying detection from free text and experiment with the negation scope and negated dataset generation. For cyberbullying context, "I hate you," and "I don’t hate you" sentences are different and alter the meaning in terms of cyberbullying. This motivates the current work, aiming to contribute to the lack of scalability and significant bias observed in non-hate speech detection.

The overall methodology of the experiment includes a the four-stage process as: datasets collection & pre-processing; negated datasets creation, feature engineering, and classification; finally, testing and validation with original datasets and negated dataset. For experiment setup, use 20-30% as test data and the rest as training data. For both experiments, with the original dataset and negated dataset, use the same test sample. The student will only perform negation for training samples; test samples must be untouched.

1. Use any of the hate speech/cyberbullying databases maintained at <http://hatespeechdata.com/>. In addition, there is two additional cyberbullying datasets available at <https://github.com/saroarjahan/Negation_project/>, AskFm and FormSpring, which can be used as well. We would like to investigate the structure of the dataset in terms of the categories present. Consider the subclass S1 of hate/cyberbullying content and subclass S2 of non-abusive content. Draw a word-cloud representation of S1 and S2 and explain the word features.
2. Perform dataset pre-processing examples stop-word removal, stemming, special character removal, emoji removal, number, hashtag, mention tag, etc. Report different preprocessing outcome and try to explain the results. Use simple LR classifier and tf-idf as feature representation for different pre-processing performance comparisons. Use best pre-processing for rest of the experiments.
3. Perform Sentiment analysis of each of the posts and get positive, negative, or neutral scores. You can use sentiment Vader (<https://github.com/cjhutto/vaderSentiment>). Use this sentiment score as a feature and report classification Accuracy and F1 score by using simple Logistic Regression as classifier and sentiment score as a feature. Dou you think sentiment analysis can be used as feature?
4. Run classification with LR and tf-itdf as a baseline model so that you can compare the result with other state-of-the-art practices. Study the implementation available at <https://github.com/pinkeshbadjatiya/twitter-hatespeech> of the paper "Deep learning for hate speech detection Tweets" by Pinkesh Badjatiya (www'17 proceedings, 2017) report result of the Accuracy and F1 score. Try to implement CNN+fastText architecture.
5. Run sate-of-the-art practices BERT model, example BERT-base-uncased and BERT-multilingual, report both model Accuracy and F1 score. Scripts available on GitHub page <https://github.com/saroarjahan/Negation_project/> . Student are open to experimenting with other BERT models, for example, BERT related specifically to hate. Different BERT models can be found here <https://huggingface.co/models> .
6. We would like to test the above 4,5 implementation at least for two different datasets. Students are welcome to experiment with more datasets. Try to explain your result of Accuracy and F1 score for the above experiment for 4, 5, which are mainly LR+tf-idf, CNN+fastText, BERT-base-uncased and BERT-multilingual.
7. Negation dataset generation: In this part, students perform negation dataset generation of the previous training dataset. A possible negation detection and augmentation algorithm could be as: Load a sentence. First, perform negation findings by using NegEx. If there is negation contained, then perform PoS Tagging on the word. Every time check if the word belongs to one of the verb forms or adjective forms, then perform either adding antonym instead of it, or add negation before it (with different forms of negations and stemming if it is a verb), or pass the word doing nothing, or remove the negation from the word.

For example, the sentence, "Alex does not like Steve Jobs". The algorithm will first be analyzed by NegEx and will find the negation part "not Like" in the sentence. It will then perform the PoS tagging and check if the "like" word has any antonym by using python library WordNet with NLTK. Since the "like" word has antonyms, it will be replaced by "hate," and the result would be, "Alex does not hate Steve job." If NLTK library fails to produce antonym, then the negation part will be removed. In that case, the output result would be "Alex does like Steve Jobs." Remember, if negation is performed, there might need to change the annotation of the posts. Students are very welcome to propose a new algorithm for negation detection and negated dataset generation.

1. After negation dataset generation, merge the negated dataset and previous training dataset together and apply 4,5 and report the Accuracy and F1 scores. Compare the score before and after negation.
2. Suggest a GUI of your own that allows us to exemplify the different steps above.

Please contact the assistant, Saroar ([mjahan18@edu.oulu.fi](mailto:mjahan18@edu.oulu.fi)) for more details about the project.

# **Project 26: Development of a software system for processing text documents. [Reserved]**

This project is aimed at solving the problem of abstracting works of fiction, which includes the creation of algorithms for automatic identification of actors and their characteristics. So we want to focus on a specific character of the fiction work.

1) Identify an artistic Russian fiction piece of work that you want to investigate (online book or set of chapters of the book containing a set of characters). Write a script in NLTK package to import the book dataset and any additional resources. Identify the list of preprocessing tasks that you think are relevant to your analysis, including preprocessing, tokenization and morphological analysis. Use the tokenize module from nltk library to tokenize the text.

2) Prior to select the parser to use, we want to evaluate two parsers on manually labelled dataset. Suggest a small manually labelled dataset of 30 sentences with different levels of complexity. You may use National Corpus of the Russian language ([Национальный корпус русского языка (ruscorpora.ru)](https://processing.ruscorpora.ru/search.xml?env=alpha&api=1.0&mycorp=&mysent=&mysize=&mysentsize=&dpp=&spp=&spd=&mydocsize=&mode=main&lang=ru&sort=i_grtagging&nodia=1&text=lexform&req=%D0%98+%D0%BE%D0%BD%D0%B8+%D1%85%D0%BE%D1%82%D0%B5%D0%BB%D0%B8%2C+%D1%87%D1%82%D0%BE%D0%B1%D1%8B+%D0%BC%D0%B0%D0%BC%D0%B0+%D0%BD%D0%B5%D0%BF%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE+%D0%BC%D1%8B%D0%BB%D0%B0+%D0%B2+%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B5+%D0%BF%D0%BE%D0%BB%D1%8B)). And define a set of performance metrics (part of speech tags identification, segmentation, composed word identification). Study the libraries of two analyzers MyStem — <https://github.com/nlpub/pymystem3> and Pymorphy2 — [pymorphy2 · PyPI](https://pypi.org/project/pymorphy2/). Compare the performance of these two parsers according to the set of metrics you defined earlier.

3) Use the Russian person name database to identify the person names from the fiction dataset. Use NLTK library to display the frequency of the occurrence of these names in the fiction dataset. You may use the description below for name tagging. Write two programs: based on your own rules using pymorphy2 (mystem) and using Natasha library — <https://github.com/natasha/natasha>. It is recommended to take at least two literary works to compare the correctness of the two programs. The texts can be input to the program from a txt file. Make comparison tables.

5) Evaluate the quality of the program using criteria such as completeness *P = B / A* and accuracy *E = B* / (quality metrics), where *A* — the number of names in the text, *B* — correctly found names in the text and — the number of found.

6) For each found character (person name), we want to identify his fiction characteristics such as direct speech (both formal conversation and informal conversations), direct actions and, possibly, their appearance (search for appearance to develop, if project time allows). Searching for actions and appearance will require syntactic analysis of sentences. Examine the work and use the modules from the Natasha library designed for this purpose. Use regular expressions to search for direct speech. Evaluate the quality of the algorithms' work with quality metrics (completeness and accuracy) as well. Summarize the overall pipeline of the approaches and tools employed and suggest simple evaluation metrics to test the quality of the outcome. Provide the outcome if this evaluation on your online book.

7) Use appropriate literature on fiction analysis to comment on your findings and ways forward.

8) Use GUI of your choice to demonstrate the execution of each specification of your project.

**Project 27: Natural Language for Visual Reasoning** **NLVR**

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

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

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

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

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

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

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

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

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

**Detailed specifications of projects 28-31 will be provided after signed data sharing agreement**

**Project 28: Radiologist Report Summarization 1**

**Idea**: Radiologist report is a written statement about of radiologist’s interpretation about the imaging result, highlighting any observed abnormality, potential presence of a given disease and judgment confidence.

The project will therefore adopts an abstractive summarization approach where he attempts to identify the disease (s), arguments that support the presence of the disease and arguments that refute the existence of the disease together with any uncertainty discourse in a way to ease potential interactions among clinicians. Medical taxonomy will be employed to monitor medical terminology.

**Project 29: Literature Review of Radiologist report Analysis**

**Idea**: We want to build a concise literature review of the topics on automatic radiologist report analysis. The starting point is to devise a set of relevant words that will be employed as keywords to scopus database using scopus API. We retrieve the titles of the search, and draw histogram showing yearly evolution of the research field. We should also scrutinize the abstract of the articles for specific diseases, treatment (using medical ontology) and draw histogram showing the proportion of each disease / treatment types that the papers focused on. In addition, the retrieved papers will be investigated using bibliometric information of the papers (i.e., citations, location, type of analysis …).

**Project 30: Mining University of Oulu Hospital ESKO database**

**Idea**: The project consists in analyzing a sample of ESKO database which contains patient records and various updates made by clinician. Given the nature of the data and the missing information. The project will focus on lexical analysis of the content using medical ontology for medical vocabulary matching, edit distance (to account for the various abbreviations employed), evolution of the various abbreviations, and trying to find out whether the recorded follow up sessions can be predicted using the history of the patient communication and record updates. The project will also try to shed light on extra services (police, social services, specialized care) raised by the clinician and how this is related to current communication records.

**Project 31: Fake News Detection [Reserved]**

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

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

**--- + PhD suggested projects**

**Resources for Finnish text analysis**

**Turku Dependency Treebank**

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

**Finnish stopwords**

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

**FinnPos**

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

**FinnWordNet**

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

**FinTWOL**

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

**OMorFi**

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

**Word list of modern Finnish**

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

**Translation API**

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

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