Exercise sheet 1

Natural Language Processing

Task 1

The file magic.txt. Load the data inside into your console.

It contains the description of 1419 *Magic: The Gathering* cards. This description contains crucial information about each card, such as its name, its mana cost, its type and its effect. Your task will be to extract all crucial information from this data set of unstructured text and turn it into a well-structured data format.

Task 2

Each line in the document represents the information about one card. Split the lines (separator "\n") to be able to look at each card individually. The result should be a list of strings/a character vector.

Task 3

The information about each card is given in the following format:
CardName: **[...]** CardCost: **[...]** CardType: **[...]** CardEffect: **[...]**
Exploit this format to extract and save each bit of information separately. Turn the information you collected into a coherent data frame with the columns "Name", "Cost", "Type" and "Effect".

Task 4

Which of the words "Creature", "Sorcery", "Instant", "Enchantment" and "Artifact" appears most often in all the texts within the "Type" column?

Recommended packages & functions

R: strsplit(), table(), readLines(file(...)), data.frame(), grepl()
Python: str.split(), collections.Counter(), open(...).readlines(), pandas.DataFrame()

Exercise sheet 2

Natural Language Processing

Task 1

In Moodle you will find three files, each containing 2 movie reviews: reviews1.txt, reviews2.txt and reviews3.txt. One of the files has a UTF-16 encoding, while the other two are UTF-8 encoded. Load the texts within them into your console. If you have used the correct encoding, the texts in your console should be readable for a human. Each review should be one element in a list of six total elements.

Task 2

Apply elementary text handling (”preprocessing“) steps. That is, within each review

- Remove punctuation, numbers and special characters
- Turn all letters into lower case
- Split the text into individual words

The result should be a list of lists of Strings (Python) or a list of character vectors (R). Each inner list/character vector represents a review as separated words.
Count how often each word occurs in this text corpus and display the 10 most common words.

Task 3

Use each one automated word stemming- and lemmatization method for your programming language. Apply them to the corpus resulting from task 2 and compare the resulting texts when applying each. Which of the two approaches would you prefer?

Task 4

Use your preferred corpus from task 3 and apply stop word removal. That is, remove every word from a stop word list from your text. Beware that you have to apply the same pre-processing of your text to your stop words, such as removing the apostrophe from ”don’t”.
Compare the most common words with the results from task 2. What do you notice?

Recommended packages & functions

R: gsub(), stringi::stri replace all(), tm::removePunctuation(), tm::removeNumbers(), tolower(), tm::stemDocument(), tm::stopwords(), textstem:lemmatize words()
Python: str.isalpha(), str.isspace(), re.sub, str.lower(), nltk.stem.PorterStemmer, nltk.stem.WordNetLemmatizer, nltk.corpus.stopwords

Exercise sheet 3

Natural Language Processing

Task 1

The file Potter.zip. Unpack it. It contains 7 txt-files, each containing the text one of the ”Harry Potter”-books. Load those txt-files into your console.

Task 2

To compare the books, we must know, which book it is we are looking at. Each file contains one particular line for every page in the book:

Page | page number book name - J.K. Rowling

Use regular expressions to automatically detect the name of the book from the texts.

Task 3

The texts in the the txt-files are not ”clean” yet. To analyze them properly, we need to do additional preprocessing steps.

- Remove the page indicator from the texts. That is, remove all lines that have the form mentioned in task 2
- Trim the start of the document until the first chapter starts.
- Remove the headers of all chapters. These are written in CAPS (all letters are capitalized). Detect this using regular expressions.
- Replace all line breaks (”\n”) with a whitespace (” ”).

The result should be a list of 7 large stings, one for each book.

Task 4

Apply elementary preprocessing steps in any order you prefer: punctuation- and number removal, lower casing, lemmatization/stemming and stop word removal. The result should be a list of lists. Each inner list represents a book as a list of words.

Task 5

Calculate the tfidf for your corpus. Return the words with the highest tfidf for each of the 7 books. Does the result give you an idea of what the books are about? If not, why?

Recommended packages & functions

R: tidytext::bind tf idf()
Python: sklearn.feature extraction.text.TfidfVectorizer

Exercise sheet 4

Natural Language Processing

Task 1

The file songs.csv, containing the lyrics of songs found on Spotify ranging since 1980.
Load the data set into your console.
We will now perform an elementary sentiment analysis. We will analyze, how positive or negative the mood in these song texts is and how it changes across time.

Task 2

Apply preprocessing to the given texts. Keep in mind, that we intend to use sentiment dictionaries to analyze the text later. How does this knowledge change your approach to preprocessing?

Task 3

In dictionary.csv you will find a sentiment dictionary. “Positive words” will have positive values while “Negative words” will have negative values.
Use this dictionary to calculate the sentiment score of each text, that is sum up all sentiment values to the corresponding words in said text. A negative score will thus indicate a negative text, while a positive value will indicate a positive text.

Task 4

Plot the average sentiment value for each year in the data set in one plot to compare how the sentiment changed over the years. Interpret the resulting graph.

Recommended packages & functions

R: as.POSIXct(), tidyverse::group by(), tidyverse::summarise()
Python: pandas.to datetime(), pandas.groupby(), datetime.datetime.strptime(), matplotlib

Exercise sheet 5

Natural Language Processing

Task 1

The file NewsCategorizer.xlsx. Load the file into your console. We are interested in the columns “category”, and “short description” and want to see whether the short descriptions match their respective category and can be detected using text clustering.

Task 2

Preprocess the texts so that they are fit for an analysis. Argue the preprocessing steps that you are using.

Task 3

Train an LDA model on this data with $K = 10$ and 200 iterations (if this takes too long on your hardware, you can also use 50 iterations).

Task 4

Calculate the tfidf-score for each word in each text and perform k-means clustering using the tfidf-score with 10 clusters.

Task 5

Compare the clusters of the k-means clustering with the true news category labels. Do the clusters represent the categories well? How about the LDA soft-clusters – does the content of the topics match the categories?

Recommended packages & functions

R: readxl::read xlsx, kmeans, tosca::LDAgen, tosca::LDAPrep
Python: pandas.read excel, sklearn.cluster.KMeans, gensim.corpora.dictionary, gensim.models.ldamodel

Exercise sheet 6

Natural Language Processing

Task 1

The file ASoIaF.zip. It contains the five books of the “A song of ice and fire” series in a plain txt-format. Load all files into your console.

We are interested in how the story and its themes develop over time. For this, we will train a topic model on each book and compare them.

Task 2

Remove unwanted fragments that are not part of the narrative. Then split the texts into individual chapters, resulting in one large (chronologically ordered) list of chapters for each book.

Task 3

Preprocess the texts so that they are fit for an analysis. Argue the

Task 4

Train five LDAs with K = 10 and 50 iterations on the very first book. Compare the resulting topics from these four and the model that was originally trained on the first book in task 4. What do you notice? Is “topic 1” the same topic in all models?

Task 5

Train an LDA with K = 10 and 50 iterations on each book separately. Compare the models by keeping your findings of task 4 in mind.

Additional information for Python users

The top words function of the gensim function outputs unweighted top words (which will be dominated by stop words). To get more meaningful top words, you can use the function we provide in the file utils.py, which takes a gensim LDA object as an input.

Recommended packages & functions

R: tosca::LDAgen, tosca::LDAPrep, tosca::topWords()

Python: gensim.corpora.dictionary, gensim.models.ldamodel, gensim.models.ldamodel.topwords()

Exercise sheet 7

Natural Language Processing

Task 1

The files emotion dataset.csv and seed words.json.

They contain a corpus of Tweets that contain labels for a emotion classification task (anger, joy, sadness and optimism) and a list of possible seed words for each emotion to initialize an LDA with. In a seeded LDA, we force the model to not find topics itself, but the topics we plant into its training using the seed words. Using words referring to certain emotions, we force our LDA to model emotions rather than the content/topics of the Tweets. We will analyze, how we can utilize an unsupervised or a seeded LDA for text classification.
As this is not a default task for topic modeling, we cannot use the popular implementation but have to refer back to niche implementations, which is common for more niche NLP-tasks. For R, you can use the "seededlda" package, which contains decent documentation in its help-pages. For Python, you will find the file lda_model.py in moodle, which is a condensed version of this GitHub repository. See the recommended functions below for more help.

Task 2

Preprocess the texts so that they are fit for an analysis. Argue the use the preprocessing steps you take for the given analysis.

Task 3

Train five LDAs with $K = 4$ topics on your texts. For Python, set n_features=10000 and set a high n_iter-value. For R, you will need the quanteda::dfm function (see the documentation of seededlda::textmodel lda).

Task 4

Train a seeded LDA with the seeds from Task 1 on your texts using the same parameters as in Task 3. Look at the model's top words. Did the seeding work?

Task 5

Calculate the most dominant topic for each document for the models from task 3 and 4. Compare the results with the true emotion labels for eacht text by using a confusion matrix. Which model does better? Is it good enough or do you think, we need to find a more method more suited to a classifcation task?

Recommended packages & functions

R: jsonlite::fromJSON(), data.table::fread(), quanteda::dfm, quanteda::dfm trim(), seededlda::textmodel lda(), seededlda::textmodel seededlda(), table()
Python: json.load(), pandas.read csv(), lda model.LDA Model, lda model.LDA Model.documents to topic model(), lda model.LDA Model.display topics(), sklearn.metrics.confusion matrix()

Exercise sheet 8

Natural Language Processing

Task 1

The file biden.csv. It contains every tweet in the month prior to the
U.S. presidential election in 2020 containing the hash tag #joebiden. Load the file into your console. It contains each tweet in the “tweet” column and the date of the tweet’s creation in the “created at” column.

We are interested in how the topics of the tweets develop over time. For this, we will train a dynamic topic model called RollingLDA on the speeches and compare the resulting topical changes in the following tasks.

Task 2

Remove unwanted fragments that are not relevant for our analysis. Preprocess the texts so that they are fit for an analysis. Argue the use the preprocessing steps you take for the given analysis.

Task 3

Train a normal LDA on the entire corpus with $K = 30$.

Task 4

Train a RollingLDA on the corpus. Set the time chunk length to three days and choose $K = 30$. If this takes a lot of time, chose prototype=1 and lower the epoch count.

Task 5

Compare the evolution of topics within the model: does the content of any topic change in particular between the time chunks? Would you prefer this model or the model you used in task 3?

Recommended packages & functions

R: RollingLDA, topWords(getTopics(RollingLDA))
Python: https://github.com/K-RLange/ttta, ttta.methods.rolling lda

Exercise sheet 9

Natural Language Processing

Task 1

The file trek.json and characters.csv. The first file contains tran- scripts of 5 Star Trek tv shows, separated into the individual episodes. The second file contains the name of characters, the tv show they appear in and their respective rank or role in the show.
In this exercise, we will investigate, how well Word2Vec models the relationships between characters in the Star Trek franchise and how different window sizes can change the relationships that are being mapped by the model.
Please note: The names “obrien” and “tpol” originally contained an apostrophe. For Word2Vec to recognize the characters correctly, you have to remove each apostrophe with an **empty** string!

Task 2

Preprocess the texts so that they are fit for an analysis. Argue the use the preprocessing steps you take for the given analysis.

Task 3

Train a Word2Vec model on all transcripts with a window size of two (i.e. two words in each direction) and a vector dimension of 300. Train another model with the same parameters and only change the window size to ten.

Task 4

We will now use the characters from characters.csv and see, how well Word2Vec differentiates the different tv shows. Calculate the cosine similarities of all possible character pairs for both models. Then, calculate the average similarity between all character pairs within each tv show and the average pairwise similarity to all characters of a different tv show. In the end you should have a 5x5 matrix, containing average pairwise similarities between and within all 5 tv shows.
What do you notice? Which model does differentiate the characters of a tv show better from other tv shows?

Task 5

Repeat task for for the role-column, which contains information of the role the characters represent in the tv show. Again, compare the inner vs. outer similarities within these groups. Which model works better for this task?

Recommended packages & functions

R: word2vec::word2vec(), word2vec::as.matrix.word2vec(), proxy::dist()
Python: gensim.corpora.Dictionary, gensim.corpora.Dictionary.doc2bow(), gensim.models.Word2Vec, scipy.spatial.distance.cdist(

Exercise sheet 10

Natural Language Processing

Task 1

The files fake train.csv and fake test.csv. They each contain a set of news articles that either contain factual news or fake news.
In the upcoming tasks, we will try to differentiate fake news from real news by comparing their document embeddings. For this, we will train a document embedding model on the whole corpus. Then, we will use the embeddings of our train-corpus to train a logistic regression model that tries to predict the labels of the test-corpus given their embeddings.

Task 2

Preprocess the texts so that they are fit for an analysis. Argue the use the preprocessing steps you take for the given analysis.

Task 3

Train a Doc2Vec model on all documents from both the training and test corpus with a window size of four and a vector dimension of 300.

Task 4

Create a data frame of all document embeddings of the documents within the training corpus and the label of the respective document. Use this data frame to train a logistic regression that uses the embeddings to predict the label of the document.
Use it to predict the labels of all documents in the test-corpus using their embeddings. Compare the resulting labels to the true labels and return the classification rate. How well does the model perform?

Task 5

Repeat tasks 3 and 4 with one adjustment: Train your initial Doc2Vec model only on the train-corpus. This way, the test-corpus is entirely unobserved for our model.

Task 6

Repeat tasks 3 and 4 with a different adjustment: Train a Word2Vec model and create a document embedding for each document by averaging all word vector that document contains. Which of the three models from task 3, 5 and 6 performs best in classifying the documents?

Recommended packages & functions

R: doc2vec
Python: gensim.models.doc2vec.Doc2Vec, gensim.models.doc2vec.Doc2Vec.infer_vector(), gensim.models.doc2vec.TaggedDocument

Exercise sheet 11

Natural Language Processing

Task 1

The file Language Detection.csv. In it, you will find a number of texts and the respective language they stem from. Your task is to determine the language of the texts as efficiently as possible in an unsupervised analysis.

To do this, please visit Huggingface, a website on which open source transformer models are available for the public to download. Look for appropriate models that are able to solve the given task without additional training and compare their performances.

Some models on Hu gingface also provide the option for an inference API directly on the website, enabling you to test out even large language models that usually would not run on local hardware. Please keep in mind, that you are not meant to train the model yourselves, as transformer models take a lot of time to train on weak GPUs or even CPUs (like most Laptops have).

Recommended packages & functions

R: huggingfaceR
Python: transformer

Exercise sheet 12

Natural Language Processing

Task 1

The files train.csv and test.csv. Load the data set into your console. The data sets contain questions and their associated coarse- and fine-grained labels from the TREC data set. The dataset consists of 6 coarse-grained categories and 50 fine-grained cate- gories. In the coming tasks, we will compare different the classification rates of different models for this multi-label classification task. Preprocess the data so that it is suitable for the different pipelines in the upcoming tasks.

Task 2

Train a Doc2Vec model on the data set and train a classifier based on the document embeddings of the train data set and the coarse labels.

Task 3

Train a lora adapter on top of BERT base uncased as a supervised model using the coarse labels of the training data set.

Task 4

Fine tune a BERT base uncased model as a supervised model using the coarse labels of the training data set.

Task 5

Evaluate your three models on the test data set using a macro f1 score and compare their performances as well as the time it took to train them. Which model would you chose in which situation?

Task 6

Repeat tasks 2-5 with the fine grained labels instead. Has anything changed?

Recommended packages & functions

Python: adapters, sklearn.metrics.f1 score

Exercise sheet 13

Natural Language Processing

The file movies.txt. In it, you will find summaries of different movies. We do however not know the genres of these books. Your task is to perform an unsupervised analysis to make an educated guess, which genres might be part of the data set. This is an open exercise. You will not be tasked with a specific model to use. Instead, you can use any model we have talked about in the exercises and the lecture so far to solve this task, except for Transformer models.

Task 1

Argue the usefulness of the following models for the task at hand in 2-3 sentences:

- Dictionary-based analysis
- Latent Dirichlet Allocation
- Word2Vec

Which method do you deem to be the best for this task?

Task 2

Preprocess the data so that it is fit for the pipeline you intend to solve the task with. Shortly explain every step of preprocessing and why it is useful for this analysis.

Task 3

Solve the task by creating a pipeline with the model you deem to be optimal for this task

Exercise sheet 14

Natural Language Processing

Task 1

The file bitcoin.csv, containing Reddit comments of the bitcoin- Subreddit from 2022. Read the file into your console.
From this data set, we only need the columns “created”, determining the date at which the post was created, “title” containing the title of the post and ”selftext” containing additional text from the post, if any. For our analysis, we want to analyze “title” and “selftext” as one combined entitiy for each text. So for each post, join the two respective strings if there is a selftext.

We will now perform a simple sentiment analysis and compare the resulting time series with the actual Bitcoin price, which you can find in bitcoin prices.csv.

Task 2

Apply preprocessing to the given texts. Keep in mind, that we intend to use sentiment dictionar- ies to analyze the text later. How does this knowledge change your approach to preprocessing?

Task 3

In dictionary.csv you will find a sentiment dictionary. “Positive words” will have positive values while “Negative words” will have negative values.
Use this dictionary to calculate the sentiment score of each text, that is sum up all sentiment values to the corresponding words in said text. A negative score will thus indicate a negative text, while a positive value will indicate a positive text.

Task 4

Compare the daily difference in market values in the file bitcoin price.csv and your sentiment scores with a correlation coefficient of your choice. Do the comments explain the behaviour of the bitcoin price evolution well?

Recommended packages & functions

R: corr(), as.POSIXct(), tidyverse::group by(), tidyverse::summarise()
Python: pandas.to datetime(), pandas.groupby(), pandas.corr()

Exercise sheet 15

Natural Language Processing

Task 1

The file 110723EUParl.docx. Read the text inside into your console.

It contains a plenary protocol of the European Parliament from the 11th of July 2023. In this exercise, you will use regular expressions to extract meta information from the file and separate the text into smaller parts. Write functions that can be generalized to other protocols of a similar structure. That is: when you are for instance trying to remove the table of contents, do not just remove the first 7 pages, but find a way to automatically detect when the main part starts (for instance using Regex). When you want to find a certain set of tokens using Regex you might want to proceed as follows:

1. Look at the original document and identify structural features that could be used to identify the tokens you are looking for.
2. Translate these loose strucutral features into Regex.
3. Use Regex to see whether you detect your desired tokens and only your desired tokens.
4. Modify your Regex to solve the exercise

Task 2

Identify the date and the weekday of the plenary discussion. Transform the date into a date-object that can be used to create timelines in your programming language (e.g. pack age datetime in Python and as.Date() in R).

Task 3

From your large text, filter out the Attendance Register (here: page 35) and the cover sheet as well as the table of contents (here: everything until page 8) and remove them so that only the main part of the document is left.

Task 4

Split the text into the individual chapters provided in the original document.

Recommended packages & functions

R: gsub(), stringi::stri officer::docx extract first, stringi::stri summary(), officer::read detect regex, doccx() Python: re.search(), datetime.datetime.strptime(), docx.Document()

Exercise sheet 17

Natural Language Processing

The file arXiv train.csv and arXiv test.csv. It contains the titles, abstracts and categories of 107,055 papers uploaded to the preprint-platform arXiv. You are tasked to create a pipeline, which predicts the category (given in the terms column) of a paper. The train corpus contains 100 labeled examples for the 4 classes. This is an open exercise. You will not be tasked with a specific model to use. Instead, you can use any model

Task 1

Argue the usefulness of the following models for the task at hand in 2-3 sentences:

• Dictionary-based analysis

• Latent Dirichlet Allocation

• Word2Vec

• Doc2Vec

• BERT

Are there other analyses that might help to solve this task? Which analyses do you deem to be the best for this task?

Task 2

Preprocess the data so that it is fit for the pipeline you intend to solve the task with. Shortly explain every step of preprocessing and why it is useful for this analysis.

Task 3

Solve the task by creating a pipeline with the model you deem to be optimal for this task

Exercise sheet 17

Natural Language Processing

Task 1

The file trump.xml, containing Speeches of Donald Tump's speeches during the campaign rallies of his 2016 presidential election. The file also contains meta information about the place and date of the speech. We are however only interested in the speeches themselves. Read the xml file into your console as if it were a simple text-file and then use Regex to filter out the speeches.

Task 2

Apply elementary tokenization steps. That is, within each speech

• Remove punctuation, numbers and special characters
• Turn all letters into lower case
• Tokenize the text into individual words

The result should be a list of lists (list of vectors for R). Each inner list represents a speech as a list of words. Count how often each word occurs in this text corpus and display the 10 most common words.

Task 3

Use each one automated word stemming- and lemmatization method for your programming language. Apply them to the corpus resulting from task 2 and compare the resulting texts when applying each. Which of the two approaches would you prefer?

Task 4

Use your ”best” corpus from task 3 and apply stop word removal. That is, remove every word from a stop word list from your text. Beware that you have to apply the same pre-processing of your text to your stop words, such as removing the apostrophe from ”don't”. Compare the most common words with the results from task 2. What do you notice?

Recommended packages & functions

R: tm::stemDocument(), tm::stopwords(), textstem:lemmatize words()
Python: nltk.stem.PorterStemmer, nltk.stem.WordNetLemmatizer, nltk.corpus.stopwords