API DOCUMENTATION

task1_2-lda_&_clustering.py:
This file has the implementation required to run LDA for task 1 and clustering for task 2.
For **TASK1**, the following description applies:
The main parameters of the model are defined in lines 157 to 172. These are:
load: boolean, describes if the source data has to be loaded (i.e. has been preprocessed already). This package already contains such data in the path: ./data/task1lda_data.json.
min_docs: int, minimum amount of documents a word needs to appear on in order to not be discarded.
max_docs: float, max ratio of documents a word is allowed to appear on in order to not be filtered out.
data_path: str, preprocessed data file (expected to be ./data/task1lda_data.json).
save_path: str, path to store a preprocessed model, in case *load* is true.
min_k, max_k: int, minimum and maximum amount of issues to use to train the LDA model.
k_step: int, step to take between min_k and max_k to train the model.
issue_words: int, number of words to consider to name a given issue.
use_best_k: bool, if true, the model does not consider any k-related parameter (i.e. issue number) and uses the best k values found beforehand.
passes: int, number of passes through the corpus a training model will perform.
workers*: int,* number of worker processes that will be used to train models. By default is the number of non-logical available CPUs minus one.
timeout*: int,* max time to wait for a set of models to be trained (is applied separately to each year, however, so the actual total training time would be 3xTimeout).
plot: bool, if true, the model accuracies (coherence and perplexity) plots would be saved to the current folder.
verbose: bool, if true, the program will print several messages during training.
store_vis_path: any, if of string type, is used as path to store model visualizations.

The main function is:
task1_lda(): Gets and prints the issues for the years: ‘2015’, ‘2016’ and ‘2017’ in the requested format.

The helper functions are:
preprocess_sentence(sent: list, lemma: WordNetLemmatizer, stop_words: set): preprocesses a given sentence SENT (list of word strings) by removing its stopwords and lemmatization. Uses the given LEMMA lemmatizer and STOP_WORDS set for such purpose.
tokenize_body(body: list): Tokenizes the given BODY (list of string sentences).
preprocess_data(path: str): Preprocesses the data from PATH (expected: dataset_korea_herald.zip)
get_vocabulary(data: pd.DataFrame, min_docs: int, max_docs: float): Gets the vocabulary from the given DATA dataframe (as returned by *preprocess_data*) and filters it given the *min_k* and *max_k* parameters.
get_data(data_path: str, load: bool, save_path): Gets the data from the given DATA_PATH (preprocessed file if LOAD is true, otherwise it should be the dataset_korea_herald.zip file).
If SAVE_PATH is a string, stores the preprocessed data on it.
train_lda(data: pd.DataFrame, vocab: Dictionary, min_k: int, max_k: int, k_step: int, passes:int, workers: int, timeout: int, plot: bool, verbose: bool, store_vis_path): Trains as many LDA models as required (defined by the *min_k, max_k and k_step* parameters explained before), given a DATA dataframe (preprocessed data), a vocabulary obtained from DATA and the parameters defined previously.
get_issue_name(issue_idx: int, model: LdaMulticore, issue_words: int): Gets an issue name given its index IDX, the model that obtained it MODEL and a number of ISSUE_WORDS.

issues_per_doc(data: pd.DataFrame, model: LdaMulticore): Obtains the probability distribution of the documents contained in DATA for the topics trained in MODEL.
rank_issues(data: pd.DataFrame, doc_to_issues: pd.Series)**:** Ranks a set of issues defined in DOC_TO_ISSUES (as returned by *issues_per_doc*) and returns a list of them, sorted by their score.
get_year_issues(year: str, data: pd.DataFrame, vocab: Dictionary, return_mdata: bool)**:** Trains a model for the given YEAR, given its DATA and VOCABulary. If RETURN_MDATA is true, will return a dictionary containing all the training information, otherwise a list of sorted issues for the given year.
get_issues(data: pd.DataFrame, return_mdata: bool): Gets all issues for the years in consideration given the entire DATA. *return_mdata* Is the argument defined in *get_year_issues.*

TASK2 is built on top of this implementation and has the following parameters (can be found in lines 238-246):
similarity_threshold: float, the minimum distance to consider two embeddings to be similar (i.e. belonging to the same cluster).
equality_threshold: float, minimum distance to consider two embeddings to be equal.
relevant_pos: set, relevant nltk POS tags (nouns, adverbs, verbs and adjectives).
grid_search_eps: bool, if True, the program will perform grid search with the epsilon parameter of the DBSCAN algorithm.

The main function for this task is:
task2_lda(): Tries to obtain all the events for the best 10 issues of each year (as obtained through the LDA model) and prints them in the required format.

And the helper functions:
print_issue_events(issue_name: str, events: list, related_events: list): Prints the issue with name ISSUE_NAME, its related EVENTS and RELATED_EVENTS in the desired format.
event_similarity(e1: dict, e2: dict): Calculates the cosine similarity between the embedding of two given events.
remove_equal_events(raw_events: list): Removes all seemingly equal events from a given events list.
get_related_events(events1: list, events2_lst: list): Given a list of events EVENTS1, and a list of lists of events EVENTS2_LST, this function finds and returns all the events in EVENTS2_LST that can be considered similar to any event in EVENTS1.
cluster_text(raw_text: list, emb_model, dbscan: DBSCAN): Clusters a given text given an pretrained embedding model EMB_MODEL and a clustering DBSCAN model.
find_events(text: list, time, emb_model, dbscan: DBSCAN, entity_recog): Tries to find the events contained in TEXT (list of tokenized, POS-tagged sentences), given its creation TIME, through the embedding model EMB_MODEL, a DBSCAN clustering class and an entity recognition helper class.
get_issue_events(issue_idx: int, model: LdaMulticore, doc_to_issues: pd.Series, data: pd.DataFrame, emb_model, dbscan: DBSCAN, entity_recog): Tries to get the events describing the issue identified by its ISSUE_IDX, given the LDA MODEL that generated it, a document-to-issues mapping, the DATA used to train the model, and other parameters which are the same ones of *find_events*.
test_epsilon(eps_min: float = 0.001, eps_max: float = 1.0, eps_step: float = 0.02): Performs grid search on the epsilon parameter of a DBSCAN model, in the range [eps_min, eps_max), with step eps_step.

task1_2-clustering.py:
This file provides the code necessary to execute clustering for Tasks 1 and 2.

1. sklearn.cluster.DBSCAN
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)
prameters:
- eps: float, default=0.5
- min_samplesint, default=5
- metricstr, or callable, default=’euclidean’
- metric_params : dict, default=None
Methods:
- fit(X[, y, sample_weight]) : Perform DBSCAN clustering from features, or distance matrix.
- fit_predict(X[, y, sample_weight]) : Compute clusters from a data or distance matrix and predict labels.
- get_params([deep]) : Get parameters for this estimator.
- set_params(**params) :Set the parameters of this estimator.

2. sklearn.feature_extraction.text.CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
class sklearn.feature_extraction.text.CountVectorizer(*, input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>)
parameters:
- input : {‘filename’, ‘file’, ‘content’}, default=’content’
- encodingstr, default=’utf-8’
- decode_error{‘strict’, ‘ignore’, ‘replace’}, default=’strict’
- strip_accents{‘ascii’, ‘unicode’}, default=None
- tokenizercallable, default=None
- stop_words{‘english’}, list, default=None
methods:
- fit(raw_documents[, y]) : Learn a vocabulary dictionary of all tokens in the raw documents.
- fit_transform(raw_documents[, y]): Learn the vocabulary dictionary and return document-term matrix.
- transform(raw_documents) : Transform documents to document-term matrix.s
returns: selfobject, Fitted estimator

3. sklearn.cluster.KMeans
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='auto')
parameters:
- n_clustersint, default=8
methods:
- fit(X[, y, sample_weight]) : Compute k-means clustering.
- fit_transform(X[, y, sample_weight]) : Compute clustering and transform X to cluster-distance space.
- fit_predict(X[, y, sample_weight]) : Compute cluster centers and predict cluster index for each sample.

return : selfobject, Fitted estimator

4 . en_core_web_sm

Components: tok2vec, tagger, parser, senter, ner, attribute_ruler, lemmatizer.
methods
- nlp(sentence)