ldayelpchallenge

```
This package was developed in order to run the Laten Dirichlet Allocation (LDA) on a set of reviews
provided by Yelp for the Yelp data challenge.

Reviews must be contained in a single file, with each review being a json object, one review per line:
{"votes": {"funny": 0, "useful": 0, "cool": 0}, "user_id": "---_j-GW5aCBtf62ihHwCw", "review_id": "tbF1KI-
PGpXnE34hfIvPgQ", "text": "My son, one of his friends and I went to L8 Night Bingo tonite...", "business_id": "v76uEBa0jkRl8AH28piX4w", "stars": 1, "date": "2013-
12-21", "type": "review"}

Reviews are first interpreted as json and filtered by language.
Reviews are then tokenized individually to create a list of unique tokens for each review (reviews are seen as a bag of words by LDA)
Next, a dictionary is build that assigns each unique token in the corpus a uniqure id for easier processing.
With that, a document-term-matrix is build that counts for each review the number of occurrences of each unique token.
Finally, the LDA algorith is run, returning a fitted LDA model that is saved in the LDAModelWrapper.
At the same time, the tokenized reviews are sorted and grouped by users.
Next, the LDAModelWrapper is used to return the topic representation of each review.
With this, we can build a mean vector for each user from all the reviews that user authored.
Now that the users are represented as vectors, we calculate all distances between all users.
Next, percentage (default 0.05) is taken as the cutoff for being a close distance between two users.
Finally, for each user we calculate his neighbors whose distance to the user is smaller than or equal to the cutoff.
These are then saved to the provided target directory.
```

Package Contents

LDAsetup	filter_lang	ldamodelwrapper	recommender
anticontract	get_threshhold	math_helper	sortbyusers

Functions

run(source, target, num_topics=100, passes=20, lang='en', distance_measure=<function euclidean>, percentage=0.05)

```
    Main entry point for this package. Contains and executes the whole data pipeline.

    Arguments:
    source -- The path string to the source file containing all reviews
    target -- The path string to the target directory where the neighbors for all users will be saved

    Keyword arguments:
    num_topics -- The number of topics LDA is supposed to discover (default 100)
    passes -- The number of iterations for the statistical inference algorithm (default 20)
    lang -- The language the reviews shall be sorted by (default 'en')
    distance_measure -- A python function that measures the distance between two vectors in a num_topics-dimensional vector space.
                        Must take two numpy arrays and return a float. (default euclidean)
    percentage -- The cutoff for being a close neighbor, i.e. two users are close if their distance is
                  within the closest percentage percent of all distances (default 0.05)
```

LDAsetup

This module holds the complete process that is necessary to execute Latent Dirichlet Allocation as implemented in the gensim package.

Modules

anticontract	json	pickle
gensim.corpora	gensim.models	time

Classes

DTMBuilder
DictionaryBuilder
LDAWrapper
ReviewTokenizer

class **DTMBuilder**

Builds a document-term-matrix from a given dictionary of id-token pairs

Methods defined here:

__init__(self, dictionary, srcTexts)
Initializes a new DTMBuilder.

Arguments:
dictionary -- The dictionary that assigns each token a unique id.
srcTexts -- A list of lists, where each list contains all unique tokens for a review.

build(self)
Builds the document-term-matrix for the given tokens using the ids from the dictionary.

save(self, fileName)
Save document-term-matrix to 'filename'.

Arguments:
fileName -- String path to where the dtm should be saved.

class **DictionaryBuilder**

Build dictionary that assigns an id to each unique token. Used for building the document-term-matrix.

Methods defined here:

__init__(self, tokDocs)
Initializes a new DictionaryBuilder.

Arguments:
tokDocs -
- Dictionary that holds all tokenized reviews. Keys are review ids, values the lists of tokens for each review.

build(self)
Build dictionary from the given tokenized reviews.

save(self, fileNameDict)
Save dictionary to 'fileNameDict'.

Arguments:
fileNameDict -- String path to where the dictionary should be saved.

class **LDAWrapper**

Wrapper class for easy use of LDA algorithm as given in gensim package

Methods defined here:

__init__(self, dtm, dictionary)
Initializes new LDAWrapper.

Arguments:
dtm -- document-term-matrix
dictionary -- Dictionary that assigns unique ids to tokens.

run(self, num_topics=100, passes=20)
Run the LDA algorithm as implemented in the gensim package.

```
        Arguments:
        num_topics -- The number of topics that LDA is supposed to discover. (default 100)
        passes -- The number of iterations for the statistical inference algorithm. (default 20)
```

save(self, fileName)

```
        Save document-term-matrix to 'filename'.

        Arguments:
        fileName -- String path to where the LDA Model should be saved.
```

class **ReviewTokenizer**

```
    Build a list of tokens for the given review(s)
```

Methods defined here:

__init__(self, reviews)

```
        Initializes a new ReviewTokenizer.

        Arguments:
        reviews -- A list containing all review objects.
```

save(self, fileName)

```
        Save the tokenized reviews to the file 'fileName'

        Arguments:
        fileName - String path for where to save the tokenized reviews.
```

tokenize(self)

```
        Tokenize (extract unique tokens) all reviews given in self.reviews
```

Data and other attributes defined here:

stop_en = [u'a', u'about', u'above', u'after', u'again', u'against', u'all', u'am', u'an', u'and', u'any', u'are', u"aren't", u'as', u'at', u'be', u'because', u'been', u'before', u'being', ...]

tokenizer = RegexpTokenizer(pattern='\\w+', gaps=False, discard_empty=True, flags=56)

filter_lang

Filter reviews by language using the langdetect module.

Modules

json

Functions

filter_by_language(reviews, lang='en')

```
Filter all reviews by language.

Arguments:
reviews -- List containing all reviews as strings

Keyword arguments:
lang -- The language we want the reviews to be filtered by (default 'en')

Returns:
List of filtered reviews containing only reviews written in 'lang' language
```

ldamodelwrapper

Holds the LDAModelWrapper, which wraps a precalculated LDAModel and provides methods for retrieving probability distributions.

Modules

gensim.corpora json gensim.models os

Classes

LDAModelWrapper

class **LDAModelWrapper**

Wrapper for loading and interacting with a precalculated LDAModel

Methods defined here:

__init__(self, LdaModel, dictionary, userTokens)

Initializes a new LDAModelWrapper.

Arguments:
LdaModel -- Either a file path string or an ldamodel returned by LDA.
dictionary -- A dictionary that was used to calculate the ldamodel. Assigns each unique token a unique id
userTokens -
- Dictionary with the user id as key and lists of lists of tokens from all reviews authored by that user as values

get_all_posteriors(self)

Retrieves the topic distributions for all users.

Returns:
Dictionary with user ids as keys and the lists returned by self.**get_user_posteriors** as values

get_user_posteriors(self, userTokens)

Retrieves the topic distributions for all reviews authored by a user.

Arguments:
userTokens -- A list of lists of all unique tokens that occur in all reviews authored by a specific user.

Returns:
List of lists which each list representing the topic distribution of a review authored by the given user.

recommender

Holds the recommender class which is used for different calculations that are needed to produce recommendations in the end.

Modules

json numpy

Classes

Recommender

class **Recommender**

Recommender class. Can find the closest neighbor for a given user as well as calculate all neighbors and all distances between all users.

Methods defined here:

__init__(self, means)
Initializes a new Recommender.

Arguments:
means -- A dictionary with the user ids as keys and their mean distribution vectors as values

calc_distances(self, distanceMeasure)
Calculates all distances between all users. Distance values are used as bins and their frequencies are then counted.

Arguments:
distanceMeasure -
- A python function that takes two numpy arrays and returns a float, the distance between the two distributions given by the arrays

Returns:
A dictionary where the keys are the rounded distances and the values are their frequencies.

calc_neighbors(self, userID, distanceMeasure, threshhold=None)
Calculates the distances to all other users for a given user by default, or only those who are closer than threshhold if given.

Arguments:
userID -- The user id as string for which we want all distances.
distanceMeasure -
- A python function that takes two numpy arrays and returns a float, the distance between the two distributions given by the arrays
threshhold -- Float that represents the cutoff for being a close neighbor (default None)

Returns:
List of lists, where each list is of the form [distance, id of neighbor].

findClosest(self, userID, distanceMeasure)
Find the neighbor with the shortest distance to the given user.

Arguments:
userID -- The user (string) whose neighbor we want to find.
distanceMeasure -
- A python function that takes two numpy arrays and returns a float, the distance between the two distributions given by the arrays

Returns:
List where first entry represents the distance and the second entry the id of the closest neighbor for the given user.

anticontract

```
Simple module to expand all english language contractions.
```

Modules

re

Functions

expand_contractions(s, contractions_dict={"'cause": 'because', "I'd": 'I had', "I'd've": 'I would have', "I'll": 'I shall', "I'll've": 'I shall have', "I'm": 'I am', "I've": 'I have', "ain't": 'am not', "aren't": 'are not', "can't": 'cannot', ...})

```
Takes a string and replaces all contractions with their long forms

Arguments:
s -- The string to be converted

Keyword arguments:
contractions_dict -
- Dictionary containg all contractions and their long forms

Returns:
String where have the contractions have been replaced.
```

Data

contractions_dict = {"'cause": 'because', "I'd": 'I had', "I'd've": 'I would have', "I'll": 'I shall', "I'll've": 'I shall have', "I'm": 'I am', "I've": 'I have', "ain't": 'am not', "aren't": 'are not', "can't": 'cannot', ...}
contractions_regex = <_sre.SRE_Pattern object>

get_threshhold

Holds the function that calculates the cutoff for close distances.

Functions

fivePercent(distances, percentage=0.05)

```
Calculates the cutoff by which a distance is considered close.

Arguments:
distances -- A dictionary with the distances as keys and their frequency as values

Keyword arguments:
percentage -
- Float where the cutoff should be (i.e. 0.05 for the closest 5 percent of all distances) (default 0.05)

Returns:
Float that markes the cutoff distance.
```

math_helper

Holds all needed mathematical functions.

Modules

numpy

Functions

euclidean(x, y)

Calculate the euclidean distance between two vectors.

Arguments:
x -- First vector.
y -- Second vector.

Returns:
Float that represents the euclidean distance.

mean(posteriors)

Calculates the mean distribution for the given distributions.

Arguments:
posteriors -
- List of lists, where each list represents a probability distribution over topics.

Returns:
List that represents the mean distribution.

sortbyusers

Holds the function to sort tokenized reviews by users.

Functions

sortByUsers(tokDocs)

Sorts the given tokenized reviews by users.

Arguments:
tokDocs -
- Dictionary where the review ids are keys and the values are lists representing the tokens in the review referenced by the key.

Returns:
Dictionary with user ids as keys and lists of lists as values, where each list represents the tokens of a review authored by that user.