jange

Sep 21, 2021

Contents:

1 Why? 3

2 Example 5

3 Installation 7

3.1 Getting started . . . 7
3.1.1 Classification . . . 7
3.1.2 Clustering . . . 8
3.2 API Reference . . . 9
3.2.1 Data Stream API . . . 9
3.2.2 Clustering Operation API . . . 11
3.2.3 Dimension Reduction Operation API . . . 12
3.2.4 Nearest Neighbors Operation API . . . 12
3.2.5 Text Operations API . . . 13
3.2.6 Operations Utilities API . . . 20

Python Module Index 21

Index 23

Welcome to **jange**, a hasslefree NLP library built with two things in mind:

- easy experiments with less boilerplate
- easy deployment for production use case

jange is powered by "industrial strength" NLP capabilities of *spacy* and battle-tested *scikit-learn* for different machine learning algorithms for clustering, classification, topic modeling and more.

CHAPTER 1

Why?

Most of the NLP projects I've done have very similar kind of setup:

- load data
- clean
- vectorize
- train clustering or classification model
- evaluate results
- if ok then export the models/associated files otherwise repeat from *clean*
- load exported binaries in a "production package" and serve the requests

As you know while experimenting, most of the time will be spent cleaning, vectorizing and training. But once you are satisfied with the results it can take quite a lot of effort to deploy it to production.

jange aims to represent the steps involved from cleaning to evaluating and using it in the production to be as simple as possible.

Not convinced? Take a look at an example below

CHAPTER 2

Example

```
from sklearn.datasets import fetch_20newsgroups
from jange import stream, ops, vis

texts, labels = fetch_20newsgroups(shuffle=False, return_X_y=True)

# create a data stream
ds = DataStream(items=texts)

# apply operations to the stream to get clusters
clusters_ds = ds.apply(
   ops.text.remove_emails(),
   ops.text.remove_links(),
   ops.text.tfidf(max_features=5000),
   ops.cluster.minibatch_kmeans(n_clusters=4)
)

for text_idx, cluster in zip(clusters_ds.context, clusters_ds):
   print(texts[text_idx][200:500]) # discard email header and limit to 300 chars
   print(f"Cluster = {cluster}")
   print("\n\n")
```

Check out tutorial *Getting started*

CHAPTER 3

Installation

Using pip is recommended. All necessary dependencies will be installed.

```
pip install jange
```

spacy will be installed however you will need to download the language models seperately. For example to download a small English spacy model you need to run

```
python -m spacy download en_core_web_sm
```

Visit https://spacy.io/models for available models in various sizes and languages.

3.1 Getting started

This tutorial introduces you to basic concepts of working with **jange**. This guide should be enough for you get started with experimenting and deploying.

We'll cover three common NLP applications

- Classification
- Clustering
- Topic Modeling

3.1.1 Classification

A classification is a problem where we want to assign one or more labels to an input. Let's load the classic newsgroup data-set and train a model to assign categories.

```
# %% imports
import logging

import pandas as pd
```

(continues on next page)

(continued from previous page)

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

from jange import ops, stream

logging.basicConfig(level=logging.INFO, format="%(name)s %(levelname)s %(message)s")

# %% load data, shuffle and create a stream
df = pd.read_csv(
    "https://raw.githubusercontent.com/jangedoo/jange/master/dataset/bbc.csv"
)
df = df.sample(frac=1.0)  # shuffles the data
train_df, test_df = train_test_split(df)
ds = stream.from_df(train_df, columns="news", context_column="type")
test_ds = stream.from_df(test_df, columns="news", context_column="type")


# %% train model
train_preds_ds = ds.apply(
    ops.classify.spacy_classifier(name="classifier"),
    op_kwargs={
        "classifier": {
            "fit_params": {
                "y": ds.context,
                "classes": list(set(ds.context)),
                "n_iter": 10,
            }
        }
    },
)


# %% evaluate model
with ops.utils.disable_training(train_preds_ds.applied_ops) as new_ops:
    preds_ds = test_ds.apply(*new_ops)

preds = list(preds_ds)
print(classification_report(test_ds.context, [p.label for p in preds]))


# %%
```

Well that was easy. What just happened? jange abstracts many steps necessary for building common NLP applications in `jange.apps` module. We use `apps.classifier()` function to train a model to categorize the news articles into different categories. Under the hood, the input texts are cleaned by removing emails, numbers, hyperlinks etc and then lemmatized. After this basic pre-processing step, a TF-IDF model is trained to extract feature vectors for each input text which is then passed to a `sklearn.linear.SGDClassifier` to predict the labels. Take a look at the evaluation results. The accuracy of the model is pretty good. If you don't know what all the stuff in the results mean then don't worry, we'll cover that in another section.

3.1.2 Clustering

Clustering is a process that finds items that are similar to each other and groups them together. For clustering, you don't need to provide labels like you do in classification problems. jange provides a function *cluster* to quickly get started. You need to pass some texts and how many clusters you want to create and everything is done for you automatically.

```
# %% Load data
from jange import ops, stream, vis

ds = stream.from_csv(
    "https://raw.githubusercontent.com/jangedoo/jange/master/dataset/bbc.csv",
    columns="news",
    context_column="type",
)
print(ds)

# %% Extract clusters
# Extract clusters
result_collector = {}
clusters_ds = ds.apply(
    ops.text.clean.pos_filter("NOUN", keep_matching_tokens=True),
    ops.text.encode.tfidf(max_features=5000, name="tfidf"),
    ops.cluster.minibatch_kmeans(n_clusters=5),
    result_collector=result_collector,
)

# %% Get features extracted by tfidf
features_ds = result_collector[clusters_ds.applied_ops.find_by_name("tfidf")]

# %% Visualization
reduced_features = features_ds.apply(ops.dim.tsne(n_dim=2))
vis.cluster.visualize(reduced_features, clusters_ds)
```

3.2 API Reference

3.2.1 Data Stream API

class jange.stream.**DataStream**(*items: Iterable[Any], applied_ops: Optional[List[T]] = None, context: Optional[Iterable[Any]] = None*)

A class representing a stream of data. A data stream is created as a result of some operation. DataStream object can be iterated which basically iterates through the underlying data. The underlying data is stored in *items* attribute which can be any iterable object.

Parameters

- **items** (*iterable*) – an iterable that contains the raw data
- **applied_ops** (*Optional[List[Operation]]*) – a list of operations that were applied to create this stream of data

Example

```
>>> ds = DataStream(items=[1, 2, 3])
>>> print(list(ds))
>>> [1, 2, 3]
```

Variables

- **applied_ops** (*List[Operation]*) – a list of operations that were applied to create this stream of data

- **items** (*iterable*) – an iterable that contains the raw data

class jange.stream.**CSVDataStream**(*path: str, columns: Union[str, List[str]], context_column: Optional[str] = None*)

Represents a stream of data by reading the contents from a csv file. pandas library is used to read the csv.

Parameters

- **path** (*str*) – path to the csv file to read. This parameter is passed directly to *pandas.read_csv* method
- **columns** (*Union[str, List[str]]*) – a column name or a list of column names in the csv file. The values from the given column(s) are used to create a stream. If a list is passed then each item in the stream will be a list of values for the given columns in that order.

Example

```
>>> ds = CSVDataStream(path="news_articles.csv", columns=["body", "title"])
```

Variables

- **df** (*pd.DataFrame*) – a pandas DataFrame object created after reading the csv file
- **columns** (*Union[str, list]*) – a list of column names or a single column name. This value is used to select data from those columns only
- **path** (*str*) – path to the csv file

class jange.stream.**DataFrameStream**(*df, columns: Union[str, List[str]], context_column: Optional[str] = None*)

Represents a stream of data by iterating over the rows in a pandas DataFrame object.

Parameters

- **df** (*pd.DataFrame*) – pandas DataFrame object
- **columns** (*Union[str, List[str]]*) – a column name or a list of column names in the dataframe. The values from the given column(s) are used to create a stream. If a list is passed then each item in the stream will be a list of values for the given columns in that order.

Example

```
>>> df = pd.DataFrame([{"text": "text 1", "id": "1"}, {"text": "text 2", "id": "2
↪"}])
>>> ds = DataFrameStream(df=df, columns="text")
>>> print(list(ds))
>>> ["text 1", "text 2"]
>>> ds = DataFrameStream(df=df, columns=["id", "text"])
>>> print(list(ds))
>>> [["1", "text 1"], ["2", "text 2"]]
```

Variables

- **df** (*pd.DataFrame*) – a pandas DataFrame object

- **columns** (*Union[str, list]*) – a list of column names or a single column name. This value is used to select data from those columns only

3.2.2 Clustering Operation API

class jange.ops.cluster.**ClusterOperation**(*model: sklearn.base.ClusterMixin*, *name: str = 'cluster'*)

Operation for clustering. This class uses scikit-learn clustering models.

Models under sklearn.cluster can be used as the underlying model to perform clustering.

Parameters

- **model** (*sklearn.base.ClusterMixin*) – See this module's SUPPORTED_CLASSES attribute to check what models are supported
- **name** (*str*) – name of this operation, default *cluster*

Variables

- **model** (*sklearn.base.ClusterMixin*) – underlying clustering model
- **name** (*str*) – name of this operation

Example

```
>>> ds = DataStream(...)
>>> ds.apply(ClusterOperation(model=sklearn.cluster.KMeans(3)))
```

jange.ops.cluster.**kmeans**(*n_clusters: int*, *name: str = 'kmeans'*, ***kwargs*) → jange.ops.cluster.ClusterOperation

Returns ClusterOperation with kmeans algorithm

Parameters

- **n_clusters** (*int*) – number of clusters to create
- **name** (*str*) – name of this operation, default *kmeans*
- **kwargs** – keyword arguments to pass to sklearn.cluster.KMeans class

Returns Operation with KMeans algorithm

Return type *ClusterOperation*

Example

```
>>> op = kmeans(n_clusters=10)
```

jange.ops.cluster.**minibatch_kmeans**(*n_clusters: int*, *name: str = 'minibatch_kmeans'*, ***kwargs*) → jange.ops.cluster.ClusterOperation

Returns ClusterOperation with mini-batchkmeans algorithm

Parameters

- **n_clusters** (*int*) – number of clusters to create
- **name** (*str*) – name of this operation, default *minibatch_kmeans*
- **kwargs** – keyword arguments to pass to sklearn.cluster.MiniBatchKMeans class

Returns Operation with MiniBatchKMeans algorithm

Return type *ClusterOperation*

Example

```
>>> op = minibatch_kmeans(n_clusters=10)
```

3.2.3 Dimension Reduction Operation API

This module contains commonly used dimension reduction algorithms

class jange.ops.dim.**DimensionReductionOperation**(*model: sklearn.base.TransformerMixin*, *name: str = 'dim_reduction'*)

Operation for reducing dimension of a multi-dimensional array. This operation is primarily used for reducing large feature space to 2D or 3D for easy visualization.

Parameters

- **model** (*TransformerMixin*) – a scikit-learn model that reduces the dimensions. Usually it will be PCA or TSNE. See *SUPPORTED_CLASSES* for all scikit-learn models that are supported
- **name** (*str*) – name of this operation

jange.ops.dim.**pca**(*n_dim: int = 2*) → jange.ops.dim.DimensionReductionOperation

DimensionReductionOperation with PCA

Parameters **n_dim** (*int, optional*) – reduce the original n-dimensional array to *n_dim* array, by default 2

Returns

Return type *DimensionReductionOperation*

jange.ops.dim.**tsne**(*n_dim: int = 2*) → jange.ops.dim.DimensionReductionOperation

DimensionReductionOperation with TSNE

Parameters **n_dim** (*int, optional*) – reduce the original n-dimensional array to *n_dim* array, by default 2

Returns

Return type *DimensionReductionOperation*

3.2.4 Nearest Neighbors Operation API

class jange.ops.neighbors.**GroupingOperation**(*name: str = 'grouping'*)

Operation to group a list of pairs.

This operation is similar to clustering but instead requires a list of pairs. It then uses the pairs data to create a graph and find connected components to group the items.

e.g. is there are pairs [("a", "b"), ("b", "c"), ("e", "f")] then the groups formed will be [{'a', 'b', 'c'}, {'e', 'f'}]

The items in the DataStream should be a tuple where each tuple indicates a pair as follows: *<item1, item2, *other_properties>*. All other entries in the tuple except *item1* and *item2* will not be used by the operation and is discarded. Typically, the output of *ops.neighbors.SimilarPairOperation* is passed to this operation.

Parameters **name** (*str*) – name of this operation, default *grouping*

Variables **name** (*str*) – name of this operation

class jange.ops.neighbors.**NearestNeighborsOperation**(*n_neighbors: int = 10, metric='cosine', name: str = 'nearest_neighbors'*)

class jange.ops.neighbors.**SimilarPairOperation**(*sim_threshold=0.8, metric='cosine', n_neighbors=10, name: str = 'similar_pair'*)

Finds similar pairs

This operation uses nearest neighbors algorithms from sklearn.neighbors package to find similar items in a dataset and convert them into pairs. Unlike nearest neighbors, where you get *n_neighbor* items for each item in the input, similar pairs will only return distinct occurence of any two items. The input data stream should contain a numpy array or a scipy sparse matrix.

Variables

- **sim_threshold** (*float*) – minimun similarity threshold that each should pair have to be considered as being similar
- **model** – any model from sklearn.neighbors package. default *sklearn.neighbors.NearestNeighbors*
- **name** (*str*) – name of this operation. default *similar_pair*

Example

```
>>> features_ds = stream.DataStream(np.random.uniform(size=(20, 100)))
>>> op = SimilarPairOperation(sim_threshold=0.9)
>>> similar_pairs = features_ds.apply(features_ds)
```

3.2.5 Text Operations API

Cleaning Operations

This module contains several text cleaning operations

class jange.ops.text.clean.**CaseChangeOperation**(*mode: str = 'lower', name: str = 'case_change'*)

Operation for changing case of the texts.

Parameters

- **mode** (*str*) – one of *lower*, *upper* or *capitalize*
- **name** (*str*) – name of this operation

Example

```
>>> ds = DataStream(["AAA", "Bbb"])
>>> list(ds.apply(CaseChangeOperation(mode="lower")))
["aaa", "bbb"]
```

Variables

- **mode** (*str*) – one of [‘lower’, ‘capitalize’, ‘upper’]
- **name** (*str*) – name of this operation

exception jange.ops.text.clean.**EmptyTextError**

class jange.ops.text.clean.**TokenFilterOperation**(*patterns: List[List[Dict[KT, VT]]], nlp: Optional[spacy.language.Language] = None, keep_matching_tokens=False, name: Optional[str] = 'token_filter'*)

Operation for filtering individual tokens.

Spacy’s token pattern matching is used for matching various tokens in the document. Any tokens matching the filter can either be discarded or kept while discarding the non matching ones.

Parameters

- **patterns** (*List[List[Dict]]*) – a list of patterns where each pattern is a List[Dict]. The patterns are passed to spacy’s Token Matcher. see https://spacy.io/usage/rule-based-matching for more details on how to define patterns.
- **nlp** (*Optional[spacy.language.Language]*) – spacy’s language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **keep_matching_tokens** (*bool*) – if true then any non-matching tokens are discarded from the document (e.g. extracting only nouns) if false then any matching tokens are discarded (e.g. stopword removal)
- **name** (*Optional[str]*) – name of this operation

Example

```
>>> nlp = spacy.load("en_core_web_sm")
>>> # define patterns to match [a, an, the] tokens
>>> patterns = [
    [{"LOWER": "a"}],
    [{"LOWER": "an"}],
    [{"LOWER": "the"}]
]
>>> # define the token filter operation to match the patterns and discard them
>>> op = TokenFilterOperation(patterns=patterns, nlp=nlp, keep_matching_
↪tokens=False)
>>> ds = stream.DataStream(["that is an orange"])
>>> print(list(ds.apply(op))
["that is orange"]
```

See https://spacy.io/usage/rule-based-matching#adding-patterns-attributes for more details on what token patterns can be used.

Variables

- **nlp** (*spacy.language.Language*) – spacy’s language model
- **keep_matching_tokens** (*bool*) – whether to discard the tokens matched by the filter from the document or to keep them
- **patterns** (*List[List[Dict]]*) – patterns to pass to spacy’s Matcher
- **name** (*str*) – name of this operation

jange.ops.text.clean.**lemmatize**(*nlp: Optional[spacy.language.Language] = None, name='lemmatize'*) → jange.ops.base.SpacyBasedOperation

Helper function to return SpacyBasedOperation for lemmatizing. This operation returns a stream.DataStream where each item is a string after being lemmatized.

Parameters

- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns **out**

Return type SpacyBasedOperation

jange.ops.text.clean.**lowercase**(*name='lowercase'*) → jange.ops.text.clean.CaseChangeOperation

Helper function to create CaseChangeOperation with mode="lower"

jange.ops.text.clean.**pos_filter**(*pos_tags: Union[str, List[str]], keep_matching_tokens: bool = False, nlp: Optional[spacy.language.Language] = None, name: Optional[str] = 'filter_pos'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to filter tokens based on Part of Speech

Parameters

- **pos_tags** (*Union[str, List[str]]*) – a single POS tag or a list of POS tags to search for. See https://spacy.io/api/annotation#pos-tagging for more details on what tags can be used. These depend on the language model used.
- **keep_matching_tokens** (*bool*) – if true then tokens having the given part of speech are kept and others are discarded from the text. Otherwise, tokens not having the given part of speech tags are kept
- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

Example

```
>>> ds = stream.DataStream(["Python is a programming language"])
>>> print(list(ds.apply(ops.text.filter_pos("NOUN", keep_matching_tokens=True))))
[programming language]
```

jange.ops.text.clean.**remove_emails**(*nlp: Optional[spacy.language.Language] = None, name: Optional[str] = 'remove_emails'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to remove emails

Parameters

- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

jange.ops.text.clean.**remove_links**(*nlp: Optional[spacy.language.Language] = None*, *name: Optional[str] = 'remove_links'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to remove hyperlinks

Parameters

- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

jange.ops.text.clean.**remove_numbers**(*nlp: Optional[spacy.language.Language] = None*, *name: Optional[str] = 'remove_numbers'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to remove numbers

Parameters

- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

jange.ops.text.clean.**remove_short_words**(*length: int*, *nlp: Optional[spacy.language.Language] = None*, *name: Optional[str] = 'remove_short_words'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to remove tokens that have fewer characters than specified

Parameters

- **length** (*int*) – atleast this many characters should be in the token, otherwise it is discarded
- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

jange.ops.text.clean.**remove_stopwords**(*words: List[str] = None*, *nlp: Optional[spacy.language.Language] = None*, *name: Optional[str] = 'remove_stopwords'*) → jange.ops.text.clean.TokenFilterOperation

TokenFilterOperation to remove stopwords

Parameters

- **words** (*List[str]*) – a list of words to remove from the text
- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded

- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

Example

```
>>> ds = stream.DataStream(["Python is a programming language"])
>>> print(list(ds.apply(ops.text.remove_stopwords())))
[Python programming language]
>>> print(list(ds.apply(ops.text.remove_stopwords(words=["programming"]))))
[Python is a language]
```

jange.ops.text.clean.**token_filter**(*patterns: List[List[Dict[KT, VT]]], keep_matching_tokens, nlp: Optional[spacy.language.Language] = None, name: Optional[str] = 'token_filter'*) → jange.ops.text.clean.TokenFilterOperation

Helper function to create TokenFilterOperation

Parameters

- **patterns** (*List[List[Dict]]*) – a list of patterns where each pattern is a List[Dict]. The patterns are passed to spacy's Token Matcher. see https://spacy.io/usage/rule-based-matching for more details on how to define patterns.
- **nlp** (*Optional[spacy.language.Language]*) – spacy's language model or None. If None then by default *en_core_web_sm* spacy model is loaded
- **keep_matching_tokens** (*bool*) – if true then any non-matching tokens are discarded from the document (e.g. extracting only nouns) if false then any matching tokens are discarded (e.g. stopword removal)
- **name** (*Optional[str]*) – name of this operation

Returns

Return type *TokenFilterOperation*

jange.ops.text.clean.**uppercase**(*name='uppercase'*) → jange.ops.text.clean.CaseChangeOperation

Helper function to create CaseChangeOperation with mode="upper"

Encoding Operations

This module contains several text encoding algorithms including binary or one-hot encoding, count based and tf-idf

jange.ops.text.encode.**count**(*max_features: Optional[int] = None, max_df: Union[int, float] = 1.0, min_df: Union[int, float] = 1, ngram_range: Tuple[int, int] = (1, 1), name: Optional[str] = 'count', **kwargs*) → jange.ops.base.ScikitBasedOperation

Returns count based feature vector extraction. Uses sklearn's CountVectorizer as underlying model.

Parameters

- **max_features** (*Optional[int]*) – If some value is provided then only top *max_features* words order by their count frequency are considered in the vocabulary
- **max_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency higher than the given value. If the value is float, then it is considered as a ratio.

- **min_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency less than the given valu. If the value is float, then it is considered as a ratio.
- **ngram_range** (*Tuple[int, int]*) – The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used. For example an `ngram_range` of `(1, 1)` means only unigrams, `(1, 2)` means unigrams and bigrams, and `(2, 2)` means only bigrams.
- **name** (*str*) – name of this operation
- ****kwargs** – Keyword parameters that will be passed to the initializer of CountVectorizer

Returns

- *SklearnBasedEncodeOperation*
- **See https** (*//scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html*)
- *for details on the paramters and more examples.*

jange.ops.text.encode.**one_hot**(*max_features: Optional[int] = None*, *max_df: Union[int*, *float] = 1.0*, *min_df: Union[int*, *float] = 1*, *ngram_range: Tuple[int*, *int] = (1*, *1)*, *name: Optional[str] = 'one_hot'*, ***kwargs*) → jange.ops.base.ScikitBasedOperation

Returns operation for performing one hot encoding of texts.

Uses sklearn.feature_extraction.text.CountVectorizer class with binary=True mode

Parameters

- **max_features** (*Optional[int]*) – If some value is provided then only top *max_features* words order by their count frequency are considered in the vocabulary
- **max_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency higher than the given value. If the value is float, then it is considered as a ratio.
- **min_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency less than the given valu. If the value is float, then it is considered as a ratio.
- **ngram_range** (*Tuple[int, int]*) – The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used. For example an `ngram_range` of `(1, 1)` means only unigrams, `(1, 2)` means unigrams and bigrams, and `(2, 2)` means only bigrams.
- **name** (*str*) – name of this operation
- ****kwargs** – Keyword parameters that will be passed to the initializer of CountVectorizer

Returns

- *SklearnBasedEncodeOperation*
- **See https** (*//scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html*)
- *for details on the paramters and more examples.*

jange.ops.text.encode.**tfidf**(*max_features: Optional[int] = None*, *max_df: Union[int*, *float] = 1.0*, *min_df: Union[int*, *float] = 1*, *ngram_range: Tuple[int*, *int] = (1*, *1)*, *norm: str = 'l2'*, *use_idf: bool = True*, *name: str = 'tfidf'*, ***kwargs*) → jange.ops.base.ScikitBasedOperation

Returns tfidf based feature vector extraction. Uses sklearn's TfidfVectorizer as underlying model.

Parameters

- **max_features** (*Optional[int]*) – If some value is provided then only top *max_features* words order by their count frequency are considered in the vocabulary
- **max_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency higher than the given value. If the value is float, then it is considered as a ratio.
- **min_df** (*Union[int, float]*) – When building vocabulary, ignore terms that have document frequency less than the given valu. If the value is float, then it is considered as a ratio.
- **ngram_range** (*Tuple[int, int]*) – The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that min_n <= n <= max_n will be used. For example an `ngram_range` of `(1, 1)` means only unigrams, `(1, 2)` means unigrams and bigrams, and `(2, 2)` means only bigrams.
- **norm** (*str*) – Each output row will have unit norm, either: * ‘l2’: Sum of squares of vector elements is 1. The cosine similarity between two vectors is their dot product when l2 norm has been applied. * ‘l1’: Sum of absolute values of vector elements is 1.
- **use_idf** (*bool*) – Enable inverse-document-frequency reweighting.
- **name** (*str*) – name of this operation
- ****kwargs** – Keyword parameters that will be passed to the initializer of CountVectorizer

Returns

- *SklearnBasedEncodeOperation*
- **See https** *(//scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html)*
- *for details on the paramters and more examples.*

Embedding Operations

This module contains operations for extracting word/document embeddings using a language model.

class jange.ops.text.embedding.**DocumentEmbeddingOperation**(*nlp: Optional[spacy.language.Language] = None*, *name: str = 'doc_embedding'*)

Operation to calculate document's vector using word-embeddings. Word embedding of each token are collected and averaged.

Parameters

- **nlp** (*Optional[Language]*) – a spacy model
- **name** (*str*) – name of this operation

Example

```
>>> ds = DataStream(["this is text 1", "this is text 2"])
>>> vector_ds = ds.apply(DocumentEmbeddingOperation())
>>> print(vector_ds.items)
```

Variables

- **nlp** (*Language*) – spacy model
- **name** (*str*) – name of this operation

jange.ops.text.embedding.**doc_embedding**(*nlp: Optional[spacy.language.Language] = None, name: str = 'doc_embedding'*) → jange.ops.text.embedding.DocumentEmbeddingOperation

Helper function to return DocumentEmbeddingOperation

Parameters

- **nlp** (*Optional[Language]*) – a spacy model
- **name** (*str*) – name of this operation

Returns

Return type *DocumentEmbeddingOperation*

3.2.6 Operations Utilities API

jange.ops.utils.**disable_training**(*ops: List[jange.base.Operation]*) → List[jange.base.Operation]

Disables the "training mode" of operations so that these operations can be used for inference.

Some operations like tfidf, kmeans, sgd etc. need to learn from the data and by default they are in "training mode" which will learn from the stream that is passed to them. Once these have been trained we want to use it in production so we need to disable the training.

Example

```
>>> with ops.utils.disable_training(stream.applied_ops) as new_ops:
>>>     ops.utils.save(new_ops, path="./operations")
```

Parameters **ops** (*List[Operation]*) – a list of operations

Python Module Index

j

jange.ops.cluster, 11
jange.ops.dim, 12
jange.ops.neighbors, 12
jange.ops.text.clean, 13
jange.ops.text.embedding, 19
jange.ops.text.encode, 17
jange.ops.utils, 20
jange.stream, 9

Index

C

CaseChangeOperation (*class in jange.ops.text.clean*), 13
ClusterOperation (*class in jange.ops.cluster*), 11
count() (*in module jange.ops.text.encode*), 17
CSVDataStream (*class in jange.stream*), 10

D

DataFrameStream (*class in jange.stream*), 10
DataStream (*class in jange.stream*), 9
DimensionReductionOperation (*class in jange.ops.dim*), 12
disable_training() (*in module jange.ops.utils*), 20
doc_embedding() (*in module jange.ops.text.embedding*), 20
DocumentEmbeddingOperation (*class in jange.ops.text.embedding*), 19

E

EmptyTextError, 14

G

GroupingOperation (*class in jange.ops.neighbors*), 12

J

jange.ops.cluster (*module*), 11
jange.ops.dim (*module*), 12
jange.ops.neighbors (*module*), 12
jange.ops.text.clean (*module*), 13
jange.ops.text.embedding (*module*), 19
jange.ops.text.encode (*module*), 17
jange.ops.utils (*module*), 20
jange.stream (*module*), 9

K

kmeans() (*in module jange.ops.cluster*), 11

L

lemmatize() (*in module jange.ops.text.clean*), 14
lowercase() (*in module jange.ops.text.clean*), 15

M

minibatch_kmeans() (*in module jange.ops.cluster*), 11

N

NearestNeighborsOperation (*class in jange.ops.neighbors*), 13

O

one_hot() (*in module jange.ops.text.encode*), 18

P

pca() (*in module jange.ops.dim*), 12
pos_filter() (*in module jange.ops.text.clean*), 15

R

remove_emails() (*in module jange.ops.text.clean*), 15
remove_links() (*in module jange.ops.text.clean*), 16
remove_numbers() (*in module jange.ops.text.clean*), 16
remove_short_words() (*in module jange.ops.text.clean*), 16
remove_stopwords() (*in module jange.ops.text.clean*), 16

S

SimilarPairOperation (*class in jange.ops.neighbors*), 13

T

tfidf() (*in module jange.ops.text.encode*), 18
token_filter() (*in module jange.ops.text.clean*), 17
TokenFilterOperation (*class in jange.ops.text.clean*), 14

tsne() (*in module jange.ops.dim*), 12

U

uppercase() (*in module jange.ops.text.clean*), 17