Published in Towards Data Science

Tim Schopf Follow

Dec 23, 2021 · 7 min read · Listen

Unsupervised Text Classification with Lbl2Vec

An introduction to embedding-based classification of unlabeled text documents

Photo by Patrick Tomasso on Unsplash.

Therefore, text classifiers can be used to organize, structure, and categorize any kind of text.

Common approaches use supervised learning to classify texts. Especially BERT-based language models achieved very good text classification results in recent years. These conventional text classification approaches usually require a large amount of labeled training data. In practice, however, an annotated text dataset for training state-of-the-art classification algorithms is often unavailable. The annotation of data usually involves a lot of manual effort and high expenses. Therefore, unsupervised approaches offer the opportunity to run low-cost text classification for unlabeled data sets. Recently, unsupervised text classification is also often referred to as zero-shot text classification. **In this article, you will learn how to use Lbl2Vec to perform unsupervised text classification.**

How does Lbl2Vec work?

Lbl2Vec is an algorithm for unsupervised document classification and unsupervised document retrieval. It automatically generates jointly embedded label, document and word vectors and returns documents of categories modeled by manually predefined keywords. The key idea of the algorithm is that many semantically similar keywords can represent a category. In the first step, the algorithm creates a joint embedding of document, and word vectors. Once documents and words are embedded in a shared vector space, the goal of the algorithm is to learn label vectors from previously manually defined keywords representing a category. Finally, the algorithm can predict the affiliation of documents to categories based on the similarities of the document vectors with the label vectors. At a high level, the algorithm performs the following steps to classify unlabeled texts:

1. Use Manually Defined Keywords for Each Category of Interest

First, we have to define keywords to describe each classification category of interest. This process requires some degree of domain knowledge to define keywords that describe classification categories and are semantically similar to each other within the

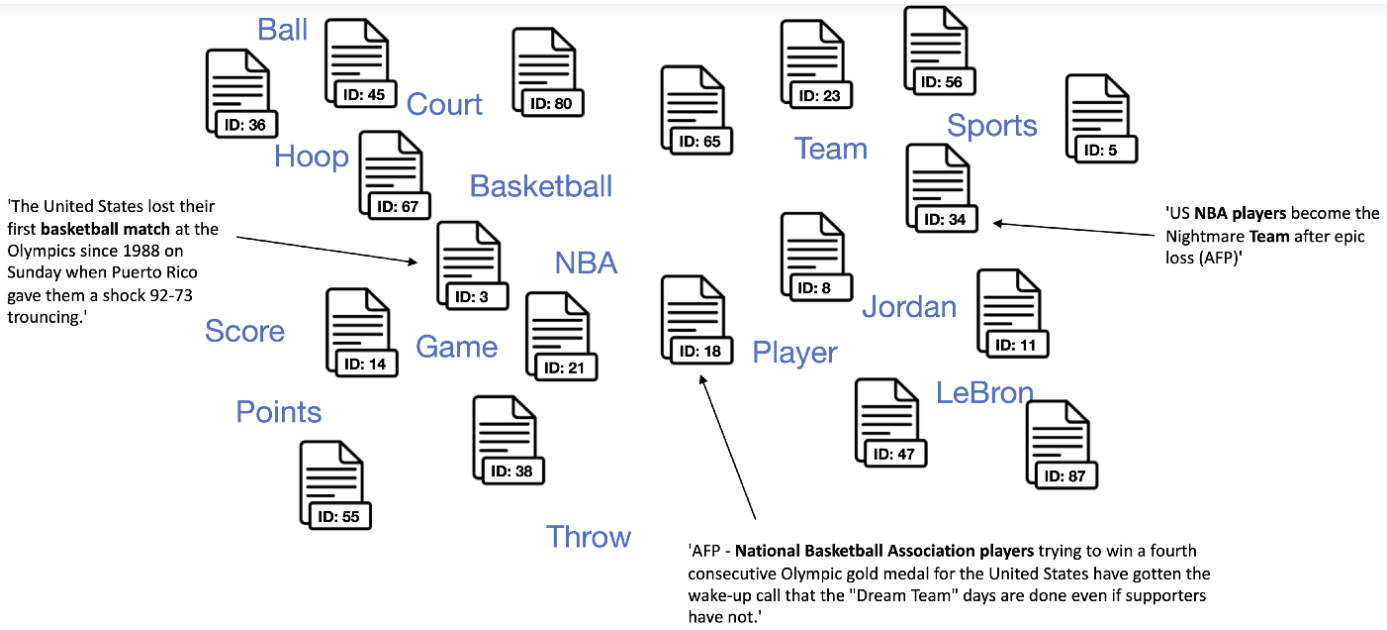
Basketball	Soccer	Baseball
NBA	FIFA	MLB
Basketball	Soccer	Baseball
LeBron	Messi	Ruth
...	...	...

Example keywords for different sports classification categories. Image by author.

2. Create Jointly Embedded Document and Word Vectors

> *An embedding vector is a vector that allows us to represent a word or text document in multi-dimensional space. The idea behind embedding vectors is that similar words or text documents will have similar vectors.* ***-Amol Mavuduru***

Therefore, after creating jointly embedded vectors, documents are located close to other similar documents and close to the most distinguishing words.

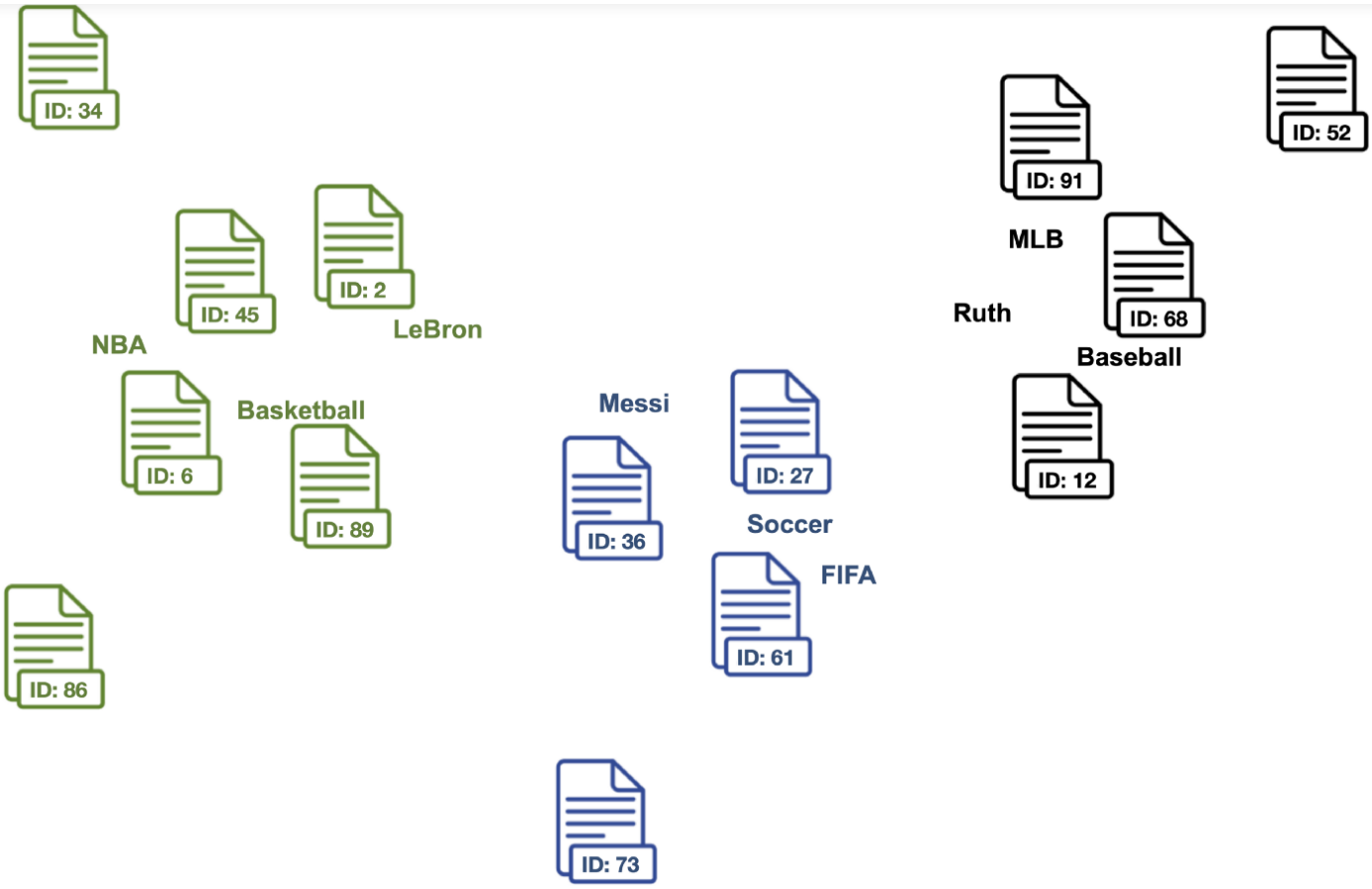


Jointly embedded word and document vectors. Image by author.

Once we have a set of word and document vectors, we can move on to the next step.

3. Find Document Vectors that are Similar to the Keyword Vectors of Each Classification Category

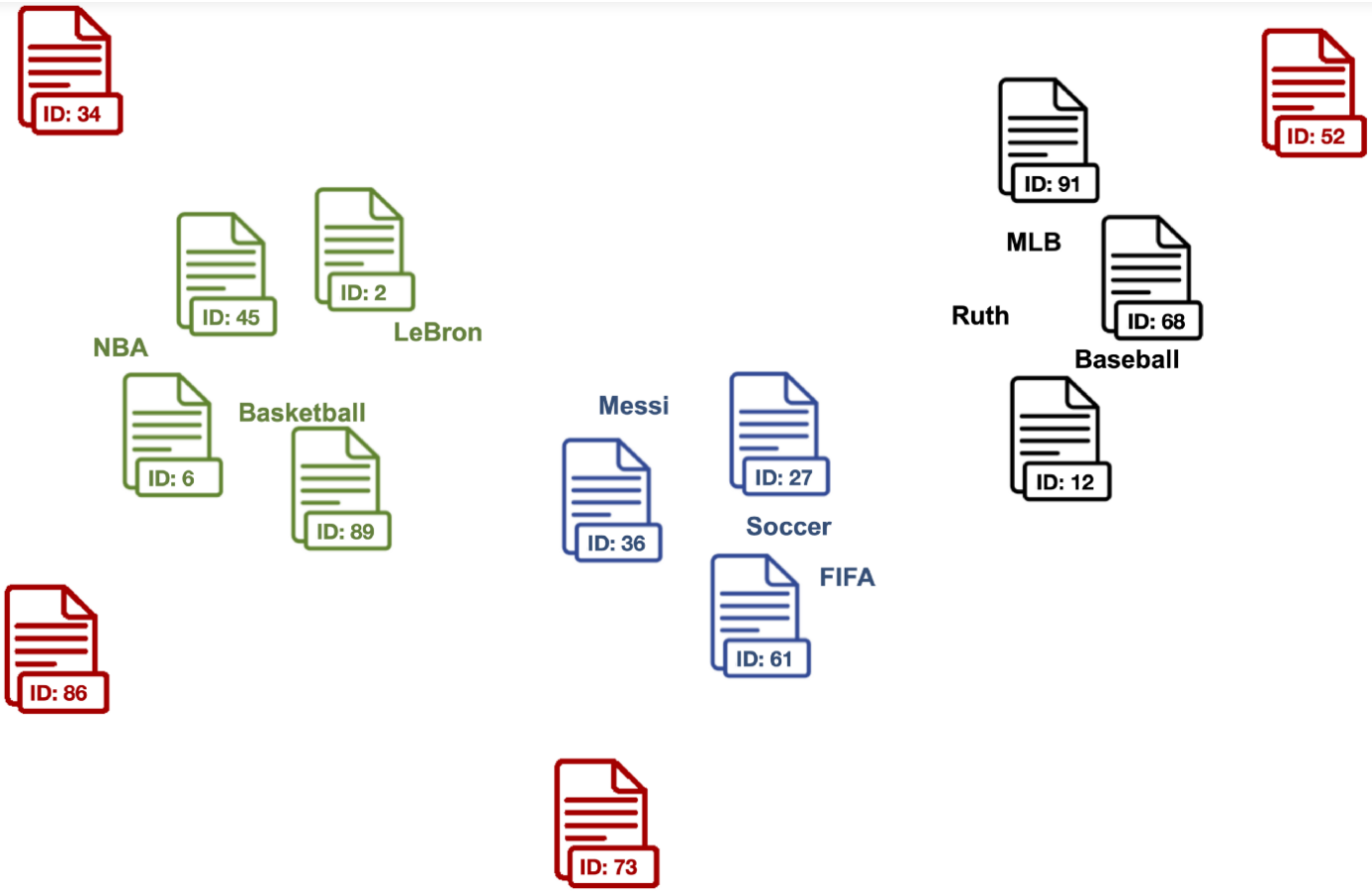
Now we can compute cosine similarities between documents and the manually defined keywords of each category. Documents that are similar to category keywords are assigned to a set of candidate documents of the respective category.



Classification category keywords with their respective set of candidate documents. Each color represents a different classification category. Image by author.

4. Clean Outlier Documents for Each Classification Category

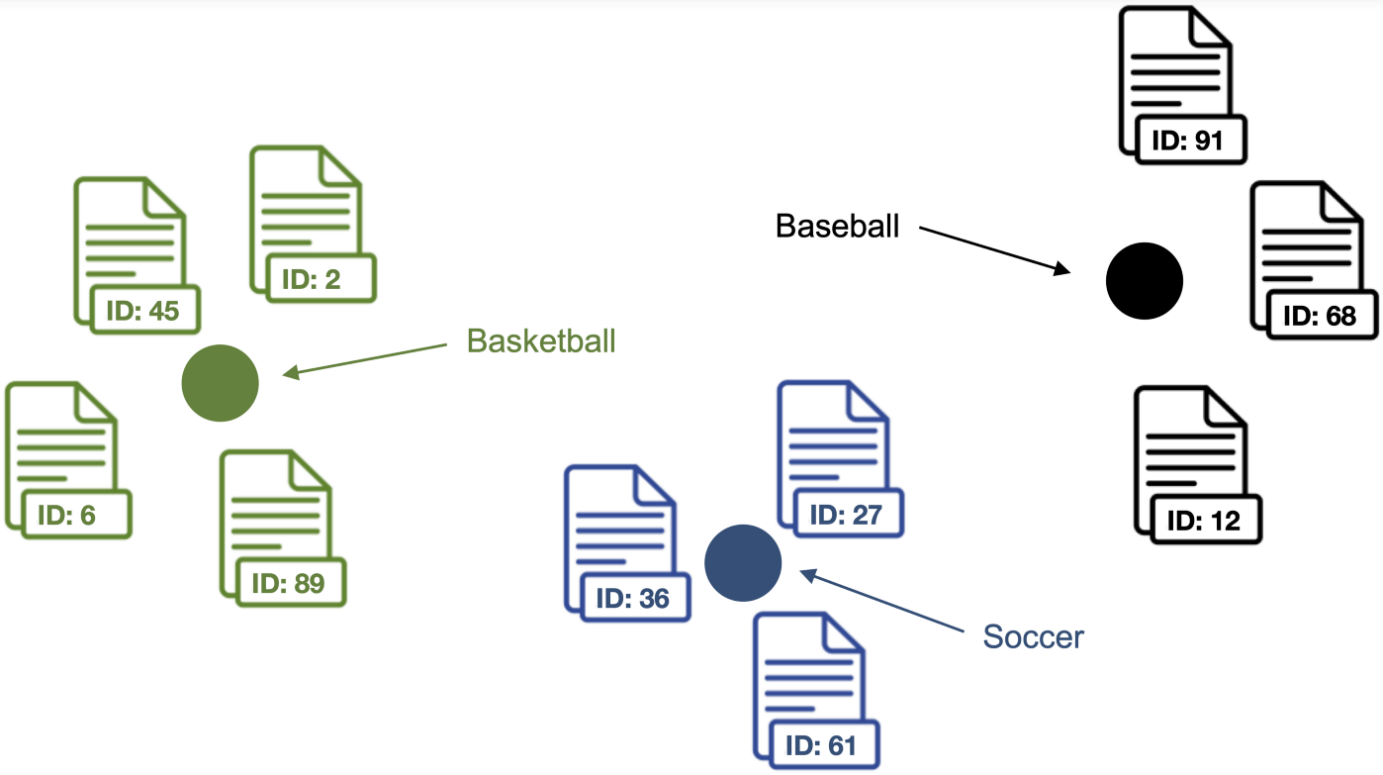
The algorithm uses LOF to clean outlier documents from each set of candidate documents that may be related to some of the descriptive keywords but do not properly match the intended classification category.



Red documents are outliers that are removed from the set of candidate documents. Image by author.

5. Compute the Centroid of the Outlier Cleaned Document Vectors as Label Vector for Each Classification Category

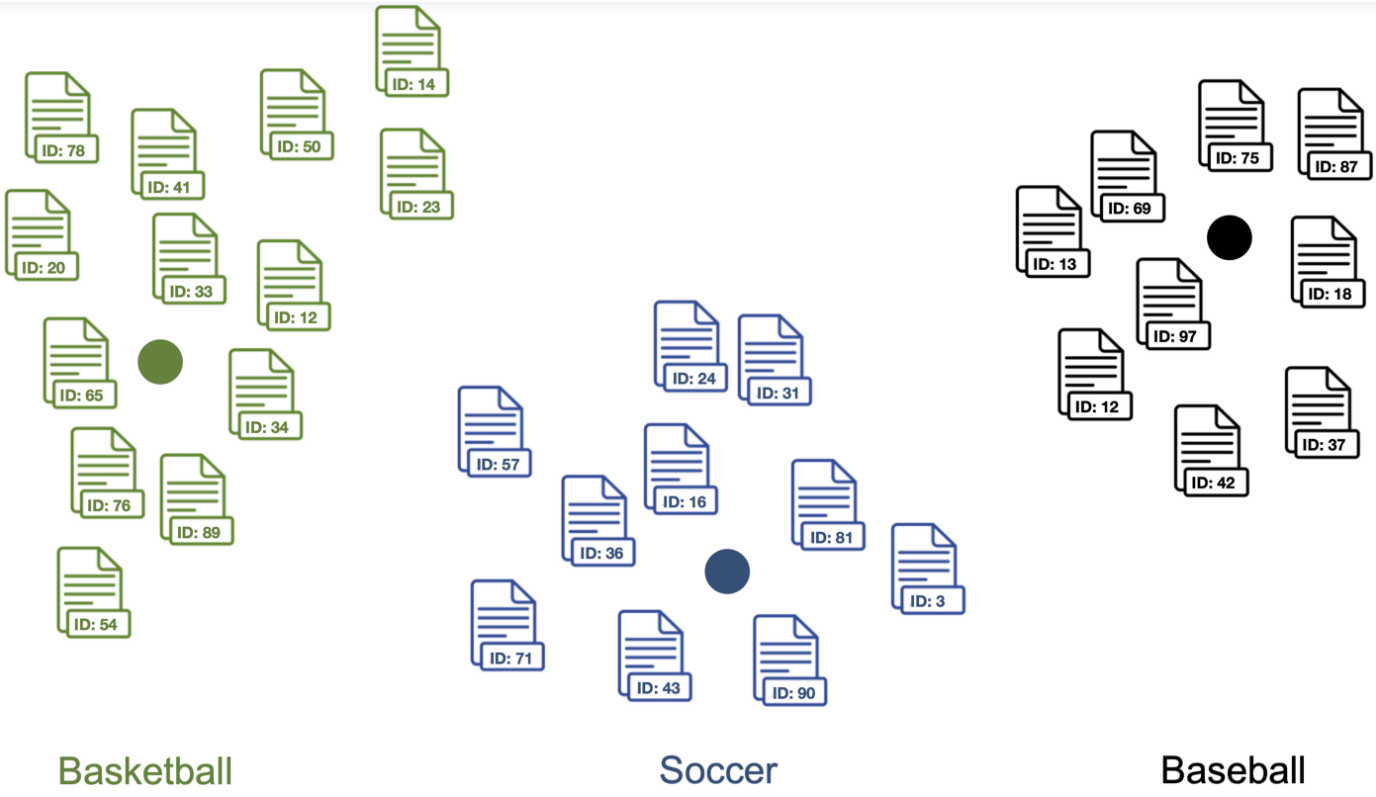
To get embedding representations of classification categories, we compute label vectors. Later, the similarity of documents to label vectors will be used to classify text documents. Each label vector consists of the centroid of the outlier cleaned document vectors for a category. The algorithm computes document rather than keyword centroids since experiments showed that it is more difficult to classify documents based on similarities to keywords only, even if they share the same vector space.



Label vectors, calculated as centroid of the respective cleaned candidate document vectors. Points represent the label vectors of the respective topics. Image by author.

6. Text Document Classification

The algorithm computes *label vector <-> document vector* similarities for each label vector and document vector in the dataset. Finally, text documents are classified as category with the highest *label vector <-> document vector* similarity.



Classification results for all documents in the dataset. Points represent label vectors of a classification category. Document colors represent their predicted classification category. Image by author.

Lbl2Vec Tutorial

In this tutorial we will use Lbl2Vec to classify text documents from the 20 Newsgroups dataset. It is a collection of approximately 20,000 text documents, partitioned evenly across 20 different newsgroups categoties. In this tutorial, we will focus on a subset of the 20 Newsgroups dataset consisting of the categories “rec.motorcycles”, “rec.sport.baseball”, “rec.sport.hockey” and “sci.crypt”. Furthermore, we will use already predefined keywords for each classification category. The predefined keywords can be downloaded here. You can also access more Lbl2Vec examples on GitHub.

Installing Lbl2Vec

We can install Lbl2Vec using pip with the following command:

```
pip install lbl2vec
```

```python
import pandas as pd
from sklearn.datasets import fetch_20newsgroups

# load data
train = fetch_20newsgroups(subset='train', shuffle=False)
test = fetch_20newsgroups(subset='test', shuffle=False)

# parse data to pandas DataFrames
newsgroup_test = pd.DataFrame({'article':test.data, 'class_index':test.target})
newsgroup_train = pd.DataFrame({'article':train.data, 'class_index':train.target})

# load labels with keywords
labels = pd.read_csv('20newsgroups_keywords.csv',sep=';')
```

lbl2vec_datareader.py hosted with ❤ by GitHub view raw

Preprocessing the Data

To train a Lbl2Vec model, we need to preprocess the data. First, we process the keywords to be used as input for Lbl2Vec.

```python
# split keywords by separator and save them as array
labels['keywords'] = labels['keywords'].apply(lambda x: x.split(' '))

# convert description keywords to lowercase
labels['keywords'] = labels['keywords'].apply(lambda description_keywords: [keyword.lower() for ke

# get number of keywords for each class
labels['number_of_keywords'] = labels['keywords'].apply(lambda row: len(row))

# lets check our keywords
print(labels)
```




	class_index	class_name	keywords	number_of_keywords
0	8	rec.motorcycles	[bikes, motorcycle]	2
1	9	rec.sport.baseball	[baseball]	1
2	10	rec.sport.hockey	[hockey]	1
3	11	sci.crypt	[encryption, privacy]	2

We see that the keywords describe each classification category and the number of keywords may vary.

Furthermore, we also need to preprocess the news articles. Therefore, we word tokenize each document and add gensim.models.doc2vec.TaggedDocument tags. Lbl2Vec needs the tokenized and tagged documents as training input format.

```
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import strip_tags
from gensim.models.doc2vec import TaggedDocument

# doc: document text string
# returns tokenized document
# strip_tags removes meta tags from the text
# simple preprocess converts a document into a list of lowercase tokens, ignoring tokens that are
# simple preprocess also removes numerical values as well as punktuation characters
def tokenize(doc):
    return simple_preprocess(strip_tags(doc), deacc=True, min_len=2, max_len=15)

# add data set type column
newsgroup_train['data_set_type'] = 'train'
newsgroup_test['data_set_type'] = 'test'

# concat train and test data
newsgroup_full_corpus = pd.concat([newsgroup_train,newsgroup_test]).reset_index(drop=True)

# reduce dataset to only articles that belong to classes where we defined our keywords
newsgroup_full_corpus = newsgroup_full_corpus[newsgroup_full_corpus['class_index'].isin(list(label
```

```
27  newsgroup_full_corpus['doc_key'] = newsgroup_full_corpus.index.astype(str)
28
29  # add class_name column
30  newsgroup_full_corpus = newsgroup_full_corpus.merge(labels, left_on='class_index', right_on='class
31
32  print(newsgroup_full_corpus.head())
```

	article	class_index	data_set_type	tagged_docs	doc_key	class_name	keywords	number_of_keywords
0	From: cubbie@garnet.berkeley.edu (...	9	train	([from, cubbie, garnet, berkeley, edu, subject...	0	rec.sport.baseball	[baseball]	1
1	From: crypt-comments@math.ncsu.edu\nSubject: C...	11	train	([from, crypt, comments, math, ncsu, edu, subj...	2	sci.crypt	[encryption, privacy]	2
2	From: george@minster.york.ac.uk\nSubject: Non-...	11	train	([from, george, minster, york, ac, uk, subject...	11	sci.crypt	[encryption, privacy]	2
3	From: williac@govonca.gov.on.ca (Chris William...	10	train	([from, williac, govonca, gov, on, ca, chris, ...	12	rec.sport.hockey	[hockey]	1
4	From: ayari@judikael.loria.fr (Ayari Iskander)...	10	train	([from, ayari, judikael, loria, fr, ayari, isk...	15	rec.sport.hockey	[hockey]	1

We can see the article texts and their classification categories in the dataframe. The “tagged_docs” column consists of the preprocessed documents that are needed as Lbl2Vec input. The classification categories in the “class_name” column are used for evaluation only but not for Lbl2Vec training.

Training Lbl2Vec

After preparing the data, we now can train a Lbl2Vec model on the train dataset. We initialize the model with the following parameters:

- **keywords_list** : iterable list of lists with descriptive keywords for each category.
- **tagged_documents** : iterable list of gensim.models.doc2vec.TaggedDocument elements. Each element consists of one document.
- **label_names** : iterable list of custom names for each label. Label names and keywords of the same topic must have the same index.
- **similarity_threshold** : only documents with a higher similarity to the respective

- **epochs** : number of iterations over the corpus.

```
from lbl2vec import Lbl2Vec

# init model with parameters
Lbl2Vec_model = Lbl2Vec(keywords_list=list(labels.keywords), tagged_documents=newsgroup_full_corpus

# train model
Lbl2Vec_model.fit()
```

Classification of Text Documents

After the model is trained, we can predict the categories of documents used to train the Lbl2Vec model.

```
from sklearn.metrics import f1_score

# predict similarity scores
model_docs_lbl_similarities = Lbl2Vec_model.predict_model_docs()

# merge DataFrames to compare the predicted and true category labels
evaluation_train = model_docs_lbl_similarities.merge(newsgroup_full_corpus[newsgroup_full_corpus['
y_true_train = evaluation_train['class_name']
y_pred_train = evaluation_train['most_similar_label']

print('F1 score:',f1_score(y_true_train, y_pred_train, average='micro'))
```

```
[Out]: F1 Score: 0.899581589958159
```

Our model can predict the correct document categories with a respectable score of $F1 \approx 0.90$. This is achieved without even seeing the document labels during training.

```
# predict similarity scores of new test documents (they were not used during Lbl2Vec training)
new_docs_lbl_similarities = Lbl2Vec_model.predict_new_docs(tagged_docs=newsgroup_full_corpus['tagge
# merge DataFrames to compare the predicted and true topic labels
evaluation_test = new_docs_lbl_similarities.merge(newsgroup_full_corpus[newsgroup_full_corpus['data
y_true_test = evaluation_test['class_name']
y_pred_test = evaluation_test['most_similar_label']
print('F1 score:',f1_score(y_true_test, y_pred_test, average='micro'))
```

```
[Out]: F1 Score: 0.889937106918239
```

Our trained Lbl2Vec model can even predict the classification categories of new documents with a score of $F1 \approx 0.89$. As mentioned before, this is achieved with a completely unsupervised approach where no label information was used during training.

For more details about the features available in Lbl2Vec, please check out the Lbl2Vec GitHub repository. I hope you found this tutorial to be useful.

Summary

Lbl2Vec is a recently developed approach that can be used for unsupervised text document classification. Unlike other state-of-the-art approaches it needs no label information during training and therefore offers the opportunity to run low-cost text classification for unlabeled datasets. The open-source Lbl2Vec library is also very easy to use and allows developers to train models in just a few lines of code.

Sources

1. Schopf, T.; Braun, D. and Matthes, F. (2021). Lbl2Vec: An Embedding-based Approach for Unsupervised Document Retrieval on Predefined Topics, (2021), Proceedings of the 17th International Conference on Web Information Systems and Technologies

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter

Emails will be sent to bkuropa@gmail.com.
Not you?