NONNEGATIVE MATRIX FACTORISATION FOR TOPIC EXTRACTION

SLIM ESSID

TSIA214 PRACTICAL

- TOPIC EXTRACTION FROM DOCUMENTS -

The goal is to study the use of nonnegative matrix factorisation (NMF) for topic extraction from a dataset of text documents. The rationale is to interpret each extracted NMF component as being associated with a specific topic.

Study and test the following script (introduced on http://scikit-learn.org/stable/auto_examples/applications/plot_topics_extraction_with_nmf_lda.html):

```
# -*- coding: utf-8 -*-

# Author: Olivier Grisel <olivier.grisel@ensta.org>
#       Lars Buitinck
#       Chyi-Kwei Yau <chyikwei.yau@gmail.com>
# License: BSD 3 clause

from __future__ import print_function
from time import time

from sklearn.feature_extraction.text import TfidfVectorizer, \
    CountVectorizer
from sklearn.decomposition import NMF
from sklearn.datasets import fetch_20newsgroups

n_samples = 2000
n_features = 1000
n_components = 10
n_top_words = 20


def print_top_words(model, feature_names, n_top_words):
   for topic_idx, topic in enumerate(model.components_):
      message = "Topic #%d: " % topic_idx
      message += " ".join([feature_names[i]
                           for i in topic.argsort()[:-n_top_words - 1:-1]
                          ])
      print(message)
   print()


# Load the 20 newsgroups dataset and vectorize it. We use a few
```

```
# heuristics to filter out useless terms early on: the posts are
# stripped of headers, footers and quoted replies, and common
# English words, words occurring in only one document or in at
# least 95% of the documents are removed.

print("Loading dataset...")
t0 = time()
dataset = fetch_20newsgroups(shuffle=True, random_state=1,
                      remove=('headers', 'footers', 'quotes'))
data_samples = dataset.data[:n_samples]
print("done in %0.3fs." % (time() - t0))

# Use tf-idf features for NMF.
print("Extracting tf-idf features...")
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2,
                           max_features=n_features,
                           stop_words='english')
t0 = time()
tfidf = tfidf_vectorizer.fit_transform(data_samples)
print("done in %0.3fs." % (time() - t0))


# Fit the NMF model
print("Fitting the NMF model (Frobenius norm) with tf-idf features,
     "n_samples=%d and n_features=%d..."
     % (n_samples, n_features))
t0 = time()
nmf = NMF(n_components=n_components, random_state=1,
        alpha=.1, l1_ratio=.5).fit(tfidf)
print("done in %0.3fs." % (time() - t0))

print("\nTopics in NMF model (Frobenius norm):")
tfidf_feature_names = tfidf_vectorizer.get_feature_names()
print_top_words(nmf, tfidf_feature_names, n_top_words)
```

1. Test and comment on the effect of varying the initialisation, especially using random nonnegative values as initial guesses (for $\mathbf{W}$ and $\mathbf{H}$ coefficients, using the notations introduced during the lecture).
2. Compare and comment on the difference between the results obtained with ℓ_2 cost compared to the generalised Kullback-Liebler cost.
3. Test and comment on the results obtained using a simpler term-frequency representation as input (as opposed to the TF-IDF representation considered in the code above) when considering the Kullback-Liebler cost.

- CUSTOM NMF IMPLEMENTATION -

Implement the multiplicative update rules (derived from the majorisation-minimisation approach) for NMF estimation with β divergences, including the case $\beta = 1$ (generalised Kullback-Liebler divergence). Ensure that :

1. you can easily choose a custom initialisation for the **W** and **H** matrices ;
2. you can set a custom number of iteration ;
3. you can monitor the behaviour of the loss function across the iterations and that it is readily decreasing.

Compare your implementation with the one offered by scikit-learn.