TP 4 NONNEGATIVE MATRIX FACTORISATION FOR TOPIC EXTRACTION

June 19, 2018

ZHU Fangda & ZHANG Bolong

```
In [1]: from time import time
        from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
        from sklearn.decomposition import NMF, LatentDirichletAllocation
        from sklearn.datasets import fetch_20newsgroups
        import numpy as np
        import scipy.sparse as sp
```

1 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) :

```
In [2]: n_samples = 2000
        n_features = 1000
        n_components = 10
        n_top_words = 20
```

```
In [3]: 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()
```

```
In [4]: def preprocess(vectorizer='tf_idf', verbose=False):
            print("Loading dataset...")
            t0 = time()
            dataset = fetch_20newsgroups(shuffle=True, random_state=1,
                                         remove=('headers', 'footers', 'quotes'))
```

```
    data_samples = dataset.data[:n_samples]
    if(verbose):
        print("Loading dataset done in %0.3fs." % (time() - t0))

    if vectorizer == 'tf_idf':
        # Use tf-idf features for NMF.
        _vectorizer = TfidfVectorizer(input = "content", max_df=0.95,
                                              stop_words='english')
    elif vectorizer == 'tf':
        # Use tf features for NMF.
        _vectorizer = CountVectorizer(input = "content", max_df=0.95,
                                  max_features=n_features,
                                  stop_words='english')
    else:
        raise ValueError("Excepted value of vectorizer is tf_idf or tf.")

    t0 = time()
    features = _vectorizer.fit_transform(data_samples)
    feature_names = _vectorizer.get_feature_names()
    if(verbose):
        print(" for LDA...")
        print("Extracting" + vectorizer +  "features done in %0.3fs." % (time() - t0))
    return features, feature_names
```

```
In [5]: def NMF_SK(features, _vectorizerName=None, W=None, H=None, K = None
                  ,random_state=None
                  ,solver= 'cd', beta_loss = 'frobenius', init='random',verbose = False ):

    t0 = time()
    nmf = NMF(n_components, init, solver, beta_loss,
                  random_state=random_state,
                  alpha=.1, l1_ratio=.5, verbose = verbose).fit(features)
    if init =='random':
        nmf = nmf.fit(features)
    else:
        nmf = nmf.fit_transform(features, W=_W, H=_H)

    if(verbose):
         print("NMF done in %0.3fs." % (time() - t0))
    return nmf, n_top_words
```

1.1 Test and comment on the effect of varying the initialisation, especially using random

nonnegative values as initial guesses (for W and H coefficients, using the notations introduced during the lecture)

```
In [6]: features, feature_names = preprocess()
        nmf, n_top_words = NMF_SK(features)
```

```
        print_top_words(nmf, feature_names, n_top_words)
```

```
Loading dataset...
Topic #0: just people don think like know time good make way really say ve right want did use
Topic #1: god jesus bible faith christian christ christians does heaven sin believe lord life
Topic #2: drive drives hard disk software floppy card mac 00 power computer scsi controller ap
Topic #3: car cars tires miles new engine insurance 00 price condition oil speed power 000 goo
Topic #4: game team games year win play season players nhl runs goal toronto hockey division f
Topic #5: edu soon send com university internet mit ftp mail cc article pub information hope e
Topic #6: thanks know does mail advance hi info interested email anybody looking card help lik
Topic #7: windows file dos files program use using window problem help os running drivers appl
Topic #8: key chip clipper keys encryption government public use secure enforcement phone nsa
Topic #9: bike insurance recommend live good course contact 250 dog open 500 org turn ground b
```

```
In [7]: features, feature_names = preprocess()
        nmf, n_top_words = NMF_SK(features, random_state = 26)
        print_top_words(nmf, feature_names, n_top_words)
```

```
Loading dataset...
Topic #0: just people don think like know time good way make really say ve right did ll new wa
Topic #1: god jesus bible faith christian christ christians does heaven sin believe lord life
Topic #2: car cars tires miles new engine insurance price 00 oil condition power speed good 00
Topic #3: windows file dos files program using problem window os help running drivers ftp ms v
Topic #4: key chip clipper keys encryption government public enforcement secure phone law nsa
Topic #5: thanks know does mail advance hi info interested email anybody looking card help lik
Topic #6: drive drives disk hard software card floppy 00 mac computer power scsi controller ap
Topic #7: game team games year win play season players nhl runs goal hockey toronto division f
Topic #8: use want window hardware need standard windows using good encryption available doing
Topic #9: edu soon send com university internet mit ftp mail cc article pub hope information e
```

According to the result, we can find the initial value of W and H have a influence to the final results. So we can find the alogo is not stable. The result depends on the initialisation, we may say that the results are similar, the order of topic is different with different initial value.

1.2 Compare and comment on the difference between the results obtained with l_2 cost compared to the generalised Kullback-Liebler cost

```
In [8]: features, feature_names = preprocess()
        nmf, n_top_words = NMF_SK(features,solver = 'mu', beta_loss='kullback-leibler')
        print_top_words(nmf, feature_names, n_top_words)
```

```
Loading dataset...
```

```
D:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\nmf.py:1035: ConvergenceWarni
  " improve convergence." % max_iter, ConvergenceWarning)
```

```
Topic #0: thanks using windows need use know help hi does file software problem work advance ve
Topic #1: work people heard state small different write going able news tell unless gets idea c
Topic #2: want time make sure things let got good hard stuff real like way look need nice long
Topic #3: used use guess public general wouldn years key using light government course rest cur
Topic #4: wrong support way believe usually people says did matter reason set word far com time
Topic #5: year post won mail send working thanks said posting check number don reply runs lot n
Topic #6: years team new 20 ago states play women 11 possible 40 13 second started jewish 1993
Topic #7: looking interested new world price sale university good sell couple buy offer cost we
Topic #8: think does say read thought just don trying yes know like true people question mean g
Topic #9: just don really like right thing think know maybe little probably remember way edu ti
```

The topics found seem similar, but not exactly, for example, there is no topic about religion for Kullback-Liebler cost which may not be very precise. Also, l2 cost may be more efficient, it extracts more information regarding the topics. And l2 cost converge more fast, the results of kullback-leibler is very larger. So with kullback-leibler, we can get WH which are more close to V for the same number of steps.

1.3 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.

```
In [9]: features, feature_names = preprocess('tf')
        nmf, n_top_words = NMF_SK(features,solver = 'mu', beta_loss='kullback-leibler')
        print_top_words(nmf, feature_names, n_top_words)
```

```
Loading dataset...
Topic #0: don just like think people know good make way ve want really say going sure ll doesn
Topic #1: didn car people said just know went like did time don came going home old got come ri
Topic #2: edu com mail graphics send pub file ftp server files code faq list message image cs i
Topic #3: government key use law state public israel encryption clipper chip keys section gun u
Topic #4: 10 drive 55 disk 16 11 hard drives 25 15 17 controller 18 12 rom 21 card 20 23 13
Topic #5: space year game team play years earth points moon surface probe season games flyers l
Topic #6: god does people jesus bible law believe true church point fact life christian time di
Topic #7: people 000 new hiv health children research president 1993 said aids april national s
Topic #8: use using thanks time problem does windows used need work know bit scsi speed help wi
Topic #9: software version pc contact price computer thanks dos 00 type available new machines
```

Neither the simple Term Frequency representation and the simple Count of tokens has a better result. For example, topic 4, there is no effective or much useful information/word. With tf_idf, it is more easy to distinguish the similar topic.

2 Custom NMF Implementation

```
In [10]: def _special_sparse_dot(W, H, X):
             """Computes np.dot(W, H), only where X is non zero."""
```

```
        if sp.issparse(X):
            ii, jj = X.nonzero()
            dot_vals = np.multiply(W[ii, :], H.T[jj, :]).sum(axis=1)
            WH = sp.coo_matrix((dot_vals, (ii, jj)), shape=X.shape)
            return WH.tocsr()
        else:
            return np.dot(W, H)
```

In [11]:
```
def _beta_divergence(X, Y, beta):
    if beta == 0:
        return np.sum(X/Y - np.ma.log(X/Y) - 1)
    elif beta == 1:
        item = np.ma.log( np.ma.divide(X,Y))
        item = item.filled(0)
        return np.sum(np.multiply(X,item) - X + Y)
    else:
        term1 = X**beta
        term2 = (beta - 1) * Y**beta
        term3 = beta * np.multiply(X,Y**(beta-1))
        term = (term1 + term2 - term3) / (beta*(beta-1))
        return np.sum(term)

def custom_NMF(V, K, W=None, H=None, beta = 1, steps=50, show_loss=False):
    if (V.ndim != 2):
        raise ValueError('The dim of V should be 2 but found ' + str(V.ndim))
    if (K < 2):
        raise ValueError('The K should a integer bigger then 2 but found ' +
                         str(K))

    F, N = V.shape
    if (W == None):
        W = np.random.rand(F, K)
    if (H == None):
        H = np.random.rand(K, N)

    pre_error = 0
    error = 0
    for step in range(steps):
        WH = W.dot(H)
        H_num = W.T.dot(np.multiply( np.power(WH,beta-2),V))
        H_den =  W.T.dot(np.power(W.dot(H), beta-1))
        H = np.multiply(H, np.ma.divide(H_num, H_den))
        WH = W.dot(H)
        W_num = np.multiply(WH**(beta-2),V).dot(H.T)
        W_den =  np.dot(WH**(beta-1), H.T)
        W = np.multiply(W, np.ma.divide(W_num, W_den))

        H = np.clip(H, 10**-150, None)
```

```
                W = np.clip(W, 10**-150, None)

                if(show_loss and (step+1) %25 == 0):
                    pre_error = error
                    WH = _special_sparse_dot(W, H, V)
                    error = _beta_divergence(V, W.dot(H), beta)

                    print("Iteration %d Error: %.3f" % (step + 1,error) )
                    print("Iteration %d Relative Error: %.3f" % (step,pre_error - error) )

            return np.asarray(W), np.asarray(H)
```

```
In [12]: features, feature_names = preprocess()
```

```
Loading dataset...
```

```
In [13]: W, H = custom_NMF(features.toarray(), 10, beta = 10, show_loss=True, steps = 100)
```

```
Iteration 25 Error: 0.430
Iteration 24 Relative Error: -0.430
Iteration 50 Error: 0.403
Iteration 49 Relative Error: 0.027
Iteration 75 Error: 0.402
Iteration 74 Relative Error: 0.001
Iteration 100 Error: 0.402
Iteration 99 Relative Error: 0.000
```

```
In [14]: W, H = custom_NMF(features.toarray(), 10, beta = 2, show_loss=True, steps = 1000)
```

```
Iteration 25 Error: 887.589
Iteration 24 Relative Error: -887.589
Iteration 50 Error: 885.227
Iteration 49 Relative Error: 2.362
Iteration 75 Error: 884.746
Iteration 74 Relative Error: 0.481
Iteration 100 Error: 884.470
Iteration 99 Relative Error: 0.276
Iteration 125 Error: 884.352
Iteration 124 Relative Error: 0.117
Iteration 150 Error: 884.294
Iteration 149 Relative Error: 0.058
Iteration 175 Error: 884.276
Iteration 174 Relative Error: 0.018
Iteration 200 Error: 884.257
Iteration 199 Relative Error: 0.019
Iteration 225 Error: 884.245
Iteration 224 Relative Error: 0.012
```

```
Iteration 250 Error: 884.239
Iteration 249 Relative Error: 0.006
Iteration 275 Error: 884.236
Iteration 274 Relative Error: 0.003
Iteration 300 Error: 884.232
Iteration 299 Relative Error: 0.004
Iteration 325 Error: 884.230
Iteration 324 Relative Error: 0.002
Iteration 350 Error: 884.227
Iteration 349 Relative Error: 0.002
Iteration 375 Error: 884.225
Iteration 374 Relative Error: 0.002
Iteration 400 Error: 884.221
Iteration 399 Relative Error: 0.004
Iteration 425 Error: 884.220
Iteration 424 Relative Error: 0.002
Iteration 450 Error: 884.218
Iteration 449 Relative Error: 0.002
Iteration 475 Error: 884.217
Iteration 474 Relative Error: 0.001
Iteration 500 Error: 884.216
Iteration 499 Relative Error: 0.001
Iteration 525 Error: 884.215
Iteration 524 Relative Error: 0.001
Iteration 550 Error: 884.214
Iteration 549 Relative Error: 0.001
Iteration 575 Error: 884.213
Iteration 574 Relative Error: 0.001
Iteration 600 Error: 884.212
Iteration 599 Relative Error: 0.001
Iteration 625 Error: 884.211
Iteration 624 Relative Error: 0.001
Iteration 650 Error: 884.209
Iteration 649 Relative Error: 0.001
Iteration 675 Error: 884.207
Iteration 674 Relative Error: 0.002
Iteration 700 Error: 884.205
Iteration 699 Relative Error: 0.002
Iteration 725 Error: 884.201
Iteration 724 Relative Error: 0.005
Iteration 750 Error: 884.193
Iteration 749 Relative Error: 0.007
Iteration 775 Error: 884.183
Iteration 774 Relative Error: 0.010
Iteration 800 Error: 884.166
Iteration 799 Relative Error: 0.017
Iteration 825 Error: 884.140
Iteration 824 Relative Error: 0.027
```

```
Iteration 850 Error: 884.109
Iteration 849 Relative Error: 0.031
Iteration 875 Error: 884.084
Iteration 874 Relative Error: 0.024
Iteration 900 Error: 884.065
Iteration 899 Relative Error: 0.019
Iteration 925 Error: 884.047
Iteration 924 Relative Error: 0.018
Iteration 950 Error: 884.023
Iteration 949 Relative Error: 0.024
Iteration 975 Error: 883.995
Iteration 974 Relative Error: 0.028
Iteration 1000 Error: 883.968
Iteration 999 Relative Error: 0.027
```

```
In [15]: print("Custome MNF:")
         for topic_idx in range(n_components):
                 message = "Topic #%d: " % topic_idx
                 message += " ".join([feature_names[i]
                                      for i in H[topic_idx,:].argsort()[:-n_top_words - 1:-1]])
                 print(message)
         print()
```

```
Custome MNF:
Topic #0: windows file dos using program use window files problem help os application running c
Topic #1: god jesus bible faith does christian christians christ believe heaven life sin lord c
Topic #2: people just don know like say time right did make ve really said law government thing
Topic #3: think don just use good like need pretty extra make yes sure bible early try reading
Topic #4: key chip clipper keys encryption government use public phone secure enforcement data
Topic #5: drive drives hard disk card software floppy pc mac apple power scsi computer controll
Topic #6: car new 00 10 bike price good year sale cars space power engine years cost miles cond
Topic #7: thanks know does mail advance hi info interested anybody like email looking help appr
Topic #8: game team year games play win season ll players nhl just runs toronto flyers division
Topic #9: edu soon com send university internet ftp mail mit information article pub cc email a
```

```
In [16]: print("sklearn MNF:")
         features, feature_names = preprocess()
```

```
sklearn MNF:
Loading dataset...
```

```
In [17]: nmf, n_top_words = NMF_SK(features, solver='mu',  beta_loss='kullback-leibler', verbos
         print_top_words(nmf, feature_names, n_top_words)
```

```
Epoch 10 reached after 0.421 seconds, error: 218.052710
Epoch 20 reached after 0.846 seconds, error: 214.712050
```

```
Epoch 30 reached after 1.272 seconds, error: 213.776898
Epoch 40 reached after 1.701 seconds, error: 213.329420
Epoch 50 reached after 2.145 seconds, error: 213.059091
Epoch 60 reached after 2.588 seconds, error: 212.872114
Epoch 70 reached after 3.033 seconds, error: 212.729480
Epoch 80 reached after 3.469 seconds, error: 212.608326
Epoch 90 reached after 3.903 seconds, error: 212.508895
Epoch 100 reached after 4.334 seconds, error: 212.448747
Epoch 110 reached after 4.778 seconds, error: 212.385838
Epoch 120 reached after 5.227 seconds, error: 212.313436
Epoch 130 reached after 5.642 seconds, error: 212.263410
Epoch 140 reached after 6.078 seconds, error: 212.226382
Epoch 150 reached after 6.508 seconds, error: 212.190192
Epoch 160 reached after 6.937 seconds, error: 212.148670
Epoch 170 reached after 7.384 seconds, error: 212.115660
Epoch 180 reached after 7.824 seconds, error: 212.084777
Epoch 190 reached after 8.257 seconds, error: 212.058514
Epoch 200 reached after 8.689 seconds, error: 212.033088


D:\ProgramData\Anaconda3\lib\site-packages\sklearn\decomposition\nmf.py:1035: ConvergenceWarnin
  " improve convergence." % max_iter, ConvergenceWarning)


Epoch 10 reached after 0.455 seconds, error: 218.389820
Epoch 20 reached after 0.890 seconds, error: 214.904369
Epoch 30 reached after 1.309 seconds, error: 213.869140
Epoch 40 reached after 1.740 seconds, error: 213.353410
Epoch 50 reached after 2.184 seconds, error: 213.074355
Epoch 60 reached after 2.613 seconds, error: 212.886821
Epoch 70 reached after 3.048 seconds, error: 212.752822
Epoch 80 reached after 3.486 seconds, error: 212.647430
Epoch 90 reached after 3.929 seconds, error: 212.544264
Epoch 100 reached after 4.355 seconds, error: 212.481188
Epoch 110 reached after 4.793 seconds, error: 212.412379
Epoch 120 reached after 5.209 seconds, error: 212.360674
Epoch 130 reached after 5.649 seconds, error: 212.296555
Epoch 140 reached after 6.068 seconds, error: 212.246282
Epoch 150 reached after 6.501 seconds, error: 212.206117
Epoch 160 reached after 6.931 seconds, error: 212.165737
Epoch 170 reached after 7.369 seconds, error: 212.129729
Epoch 180 reached after 7.815 seconds, error: 212.107796
NMF done in 16.559s.
Topic #0: year probably people don tell going given really job money general weeks lot free pa
Topic #1: like years just stuff ago doing things working good run runs really sounds new speed
Topic #2: used time using new wouldn simple mind course use second way light low yes uses curr
Topic #3: thanks windows mail using hi software need help file advance does pc email card prog
Topic #4: said people heard think come thing right person simply making seen law hear ve start
```

```
Topic #5: know want wrong work does need try looking use don think similar help questions thoug
Topic #6: think problem make use just remember work problems support number sure set little dri
Topic #7: say world read god true people usually don yes really makes agree way says actually t
Topic #8: look maybe times right kind way guess got news ve small check game just point good li
Topic #9: edu interested subject write sale source good university 20 following posting 11 1993
```

The result of our custome MNF is pretty good. Comparing the implementation with the one offered by scikit-learn, the sklearn MNF seems better and fastern the beta_loss is smaller and converge fast. It extracts more effective topics and information.