TP4

June 18, 2018

1 TP4 - Non-negative Matrix Factorization

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 scikit)

```
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
```

```
In [2]: def vectorizeFeatures(_vectorizer=None, _random_state=None):
            # Set default params
            if _vectorizer is None:
                vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=1000, stop_wo
            else:
                vectorizer = _vectorizer
            random_state = 1 if _random_state is None else _random_state
            # Fetch data and vectorize
            print("Loading dataset...")
            dataset = fetch_20newsgroups(shuffle=True, random_state=random_state,
                                         remove=('headers', 'footers', 'quotes'))
            data_samples = dataset.data[:2000]
            t0 = time()
            features = vectorizer.fit_transform(data_samples)
            feature_names = vectorizer.get_feature_names()
            print("done in %0.3fs." % (time() - t0))
            return features, feature_names
```

```
In [3]: def NMFModel(features, _vectorizerName=None, _random_state=None,
                  _beta_loss=None, _init=None, _W=None, _H=None, _K = None):

            n_samples = 2000
            n_features = 1000
            n_top_words = 20
            n_components = 10 if _K is None else _K
```

```
        vectorizerName = "tf_idf" if _vectorizerName is None else _vectorizerName
        random_state = 1 if _random_state is None else _random_state
        solver = 'mu'
        beta_loss = 'frobenius' if _beta_loss is None else _beta_loss
        init = 'random' if _init is None else _init

        print("Fitting the NMF model ("+beta_loss+" norm) with "+vectorizerName+" features
              "n_samples=%d and n_features=%d..." % (n_samples, n_features))

        t0 = time()
        if _init is None:
            nmf = NMF(n_components=n_components,
                      random_state=_random_state,
                      solver = solver,
                      beta_loss = beta_loss,
                      init = 'random',
                      alpha=.1, l1_ratio=.5).fit(features)
        else:
            nmf = NMF(n_components=n_components,
                      random_state=_random_state,
                      solver = solver,
                      beta_loss = beta_loss,
                      init = _init,
                      alpha=.1, l1_ratio=.5)
            nmf.fit_transform(features, W=_W, H=_H)
        print("done in %0.3fs." % (time() - t0))

        print("\nTopics in NMF model ("+beta_loss+" norm):")
        return nmf, n_top_words
```

```
In [4]: 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 [5]: def runExample(_vectorizer=None, _vectorizerName=None, _random_state=None, _beta_loss=N
                   _init=None, _W=None, _H=None, _K=None):
        features, feature_names = vectorizeFeatures(_vectorizer, _random_state)
        nmf, n_top_words = NMFModel(features, _vectorizerName, _random_state, _beta_loss, _
        print_top_words(nmf, feature_names, n_top_words)
```

1.0.1 Q1. 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).

The NMF Model in Scikit-Learn allows for different possible initialisations, among which three have been tested, all of them having by default a **TF-IDF Vectorizer**, the **same cost function** (*frobenius* l_2 normalization), and all of them using **multiplicative update (MU) rules**:

Init	Convergence Time (seconds)
Random	0.119
nndvsda	0.088
nndvsdar	0.173

From the scikit documentation it is possible to understand that the *nndvsda* (Nonnegative Double Singular Value Decomposition) performs a decomposition on the features and then fills in zero values with the average value of the features matrix; the *nndsvdar* initializiation performs the same operation, but fills zeros with very small random values.

The best model, in terms of convergence, was the *nndvsda* one, however the *random* one is very close.

```
In [6]: runExample(_init='random')
```

```
Loading dataset...
done in 0.429s.
Fitting the NMF model (frobenius norm) with tf_idf features, n_samples=2000 and n_features=1000
done in 0.119s.

Topics in NMF model (frobenius norm):
Topic #0: car cars tires miles insurance engine oil speed 000 price new condition power brake g
Topic #1: edu soon com send internet university mit mail cc article ftp hope information email
Topic #2: just thought wondering sure listen does wrong mean bad argument heard oh book driving
Topic #3: don know think like need look read mean pretty want really thinking does ve sure does
Topic #4: like time year good new ll game years way 10 got better team great didn space little
Topic #5: use window windows want using problem standard need hardware try good application wor
Topic #6: people government law rights israel think say true evidence did make person right cri
Topic #7: thanks windows file does mail know card advance hi help dos info files looking progra
Topic #8: god jesus bible faith does christ christian christians heaven sin believe life lord c
Topic #9: key chip clipper keys encryption government public secure enforcement phone nsa commu
```

```
In [7]: runExample(_init='nndsvda')
```

```
Loading dataset...
done in 0.301s.
Fitting the NMF model (frobenius norm) with tf_idf features, n_samples=2000 and n_features=1000
done in 0.088s.
```

```
Topics in NMF model (frobenius norm):
Topic #0: just people don think like know good time make way really say right ve want did ll ne
Topic #1: windows use dos using window program card help software pc drivers os application vi
Topic #2: god jesus bible faith christian christ does christians heaven sin believe lord life 
Topic #3: thanks know does advance mail info hi interested email anybody like list send informa
Topic #4: 00 sale car 10 condition price card new offer 250 asking 15 12 20 50 today cd 30 con
Topic #5: edu soon com send university internet mit ftp mail cc article information pub hope ma
Topic #6: file files problem format win sound read pub ftp save create running site self copy 
Topic #7: game team games year win play season players nhl runs goal hockey toronto division f
Topic #8: drive drives hard disk floppy mac software mb controller scsi computer rom apple pow
Topic #9: key chip clipper keys encryption government public use secure enforcement phone nsa 
```

```
In [8]: runExample(_init='nndsvdar')
```

```
Loading dataset...
done in 0.396s.
Fitting the NMF model (frobenius norm) with tf_idf features, n_samples=2000 and n_features=100
done in 0.173s.

Topics in NMF model (frobenius norm):
Topic #0: just people don think like good know time make way really say right ve want did ll ne
Topic #1: windows use dos using window program card help software pc drivers os application vi
Topic #2: god jesus bible faith christian christ christians does heaven sin believe lord life 
Topic #3: thanks know does advance mail info hi interested email anybody like list send lookin
Topic #4: 00 sale car 10 condition price card new offer 250 asking 15 today 12 50 cd 20 intere
Topic #5: edu soon com send university internet mit ftp mail cc article information pub hope ma
Topic #6: file files problem format win sound read pub ftp save site create running self image
Topic #7: game team games year win play season players nhl runs goal hockey toronto division f
Topic #8: drive drives hard disk floppy mac software mb controller scsi computer rom apple pow
Topic #9: key chip clipper keys encryption government public use secure enforcement phone nsa 
```

1.0.2 Q2. Compare and comment on the difference between the results obtained with l_2 cost compared to the generalised Kullback-Liebler cost.

The generalised Kullback-Lieber cost seems to be outperformed by the $ l_{2} $ cost, since it takes way longer to reach convergence (from ~0.15s to > 3s), and the topics extracted with Kullback-Leibler cost seems to be less precise. For example in topics #4 and #8, the words extracted with KL cost do not give very much information regarding the topics - while in previous tests it was possible to infer more details.

```
In [9]: runExample(_beta_loss='kullback-leibler')
```

```
Loading dataset...
done in 0.317s.
Fitting the NMF model (kullback-leibler norm) with tf_idf features, n_samples=2000 and n_featu
```

```
done in 3.332s.

Topics in NMF model (kullback-leibler norm):
Topic #0: thanks know need like want mail post send edu does list use time don information buy
Topic #1: using drive new used use version sale need machine card work video pc memory data so
Topic #2: think people wrong just god guess time believe person fact don really saying know ag
Topic #3: com won 20 team number haven short 10 st second news 30 media free 1993 12 let presi
Topic #4: say does read like thought know probably interested want come wondering point don tr
Topic #5: years got way old usually good edu just soon time ago couple maybe case ll low run t
Topic #6: yes true mean things work stuff doesn don know different good heard matter mind want
Topic #7: year sure look just trying good time better don using hear left car far said start u
Topic #8: right thing world make people like government law question use number given possible
Topic #9: windows email file ve program looking hi remember mail help work window sun programs
```

1.0.3 Q3. 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 the following test, the simpler CountVectorizer method was used for the term-frequency representation. Unlike TF-IDF (default in the above test), CountVectorizer reaches convergence in the default 200 max iterations, even though taking longer than previous tests with l_2 convergence.

```
In [10]: _vectorizer = CountVectorizer(max_df=0.95, min_df=2, max_features=1000, stop_words='e
         runExample(_beta_loss='kullback-leibler', _vectorizer=_vectorizer, _vectorizerName="C
```

```
Loading dataset...
done in 0.309s.
Fitting the NMF model (kullback-leibler norm) with CountVectorizer features, n_samples=2000 an
done in 2.209s.

Topics in NMF model (kullback-leibler norm):
Topic #0: use windows thanks using does know problem help card need new file scsi window work
Topic #1: said people didn went children came hiv did told time took home started women new ki
Topic #2: drive space disk hard drives israel controller rom earth bios data 16 floppy moon pr
Topic #3: government key public use law state chip encryption clipper keys gun president used
Topic #4: edu com mail send list news server faq information message david xfree86 thanks post
Topic #5: god does people jesus law believe bible church true person fact life point christian
Topic #6: don like think just know people ve good want way really time say make ll going sure
Topic #7: 10 55 11 game team play 12 15 20 18 period 25 17 13 19 14 22 year 24 23
Topic #8: car year just good power right bike better cars use new used speed point oil light e
Topic #9: graphics available edu ftp contact pub program version computer university mail file
```

1.1 Custom NFM Implementation

In [11]:
```
###### CUSTOM NMF IMPLEMENTATION ######
# Multiplicative Update Rules for NMF #
# estimation with beta divergences    #
import numpy

def custom_NMF(V, K, W=None, H=None, steps=50, beta=0, toll=0.1, show_div=False):

    F = len(V) #Number of V rows
    N = len(V[0]) #Number of V columns

    if W is None:
        W = numpy.random.rand(F,K)

    if H is None:
        H = numpy.random.rand(K,N)

    if N != len(H[0]):
        raise ValueError("Size for H[0] is different - found "+str(len(H[0]))+" in pla
    if F != len(W):
        raise ValueError("Size for F is different - found "+str(len(F))+" in place of

    #Setup n_iter
    n_iter = 1

    # Setup initial error
    init_error = _beta_div(V,W,H,beta,F,N,K)
    if show_div:
        print("Initial error: "+str(init_error))
    error = init_error

    for step in range(steps):

#         Tests with whole matrix : multiply = O | dot = *
        upd_UP = numpy.dot(W.T, numpy.multiply(pow(numpy.dot(W,H),beta-2), V))
        upd_DOWN = numpy.dot(W.T, pow(numpy.dot(W,H),beta-1))
        upd = upd_UP / upd_DOWN
        H = numpy.multiply(H, upd)

        upd_UP = numpy.dot(numpy.multiply(pow(numpy.dot(W,H),beta-2), V),H.T)
        upd_DOWN = numpy.dot(pow(numpy.dot(W,H),beta-1), H.T)
        upd = upd_UP / upd_DOWN
        W = numpy.multiply(W, upd)

        if toll > 0:
            new_error = _beta_div(V,W,H,beta,F,N,K)
            if show_div:
```

```
                print("Error on iteration "+str(n_iter)+": " +str(new_error))
            # Check if approximation error relative decrease is below the desired thr
            if ((error - new_error) / init_error) < toll:
                break
            error = new_error

        n_iter += 1

    return W, H

def _beta_div(V,W,H,beta,F,N,K):
    div = 0
    # Update beta_divergence
    WH = numpy.dot(W, H)
    for i in range(F):
        for j in range(N):
                x = V[i][j] if V[i][j] != 0 else numpy.finfo(numpy.double).tiny
                y = WH[i][j]
                if beta == 1: # generalized Kullback-Leibler divergence. x log(x/y) -
                    div += x*numpy.log(x/y) - x + y
                elif beta == 0: # Itakura-Saito divergence. (x/y) - log(x/y) -1
                    div += (x/y) * numpy.log(x/y) - 1
                else: # Euclidean distance. (1/beta(beta-1))(x^beta + (beta-1)y^beta
                    div += 1/(beta*(beta-1))*(pow(x,beta) + (beta-1)*pow(y,beta) - be
    return div

#######
```

In [16]:
```
features, feature_names = vectorizeFeatures()

V = numpy.random.rand(features.shape[0], features.shape[1])
V = numpy.array(V) # Data matrix F x N
K = 10

W, H = custom_NMF(V, K, beta = 1, toll = 0.0001, show_div = True)
```

```
Loading dataset...
done in 0.323s.
Initial error: 2598546.0529280994
Error on iteration 1: 198049.876475166
Error on iteration 2: 197385.47350395098
Error on iteration 3: 196824.9661585167
Error on iteration 4: 196345.73798660949
Error on iteration 5: 195931.30470955608
Error on iteration 6: 195569.33383094586
Error on iteration 7: 195250.4176342084
Error on iteration 8: 194967.25614096617
Error on iteration 9: 194714.09903612413
```

```
In [17]: nmf, n_top_words = NMFModel(features, _init='custom', _W=W, _H=H, _K=K)
         print_top_words(nmf, feature_names, n_top_words)
```

```
Fitting the NMF model (frobenius norm) with tf_idf features, n_samples=2000 and n_features=1000
done in 0.065s.

Topics in NMF model (frobenius norm):
Topic #0: god jesus bible faith does christian christ christians heaven sin believe lord life r
Topic #1: windows file dos files program problem using os help drivers running ftp ms version a
Topic #2: use drive new key car good software power chip used computer using need speed want 00
Topic #3: think don good need win extra book does did make case pretty course yes try means act
Topic #4: game team games year win play season players nhl runs goal hockey division toronto pl
Topic #5: thanks know does advance mail hi info interested anybody email looking help card appr
Topic #6: edu soon com send university internet mit mail ftp cc article pub hope information ho
Topic #7: just thought bike don sure wondering listen new bad ll really heard car driving wrong
Topic #8: like don sounds know look looks thing make way newsgroup got ve doing mean great say
Topic #9: people know don government say time right law really did let said going make way ve p
```