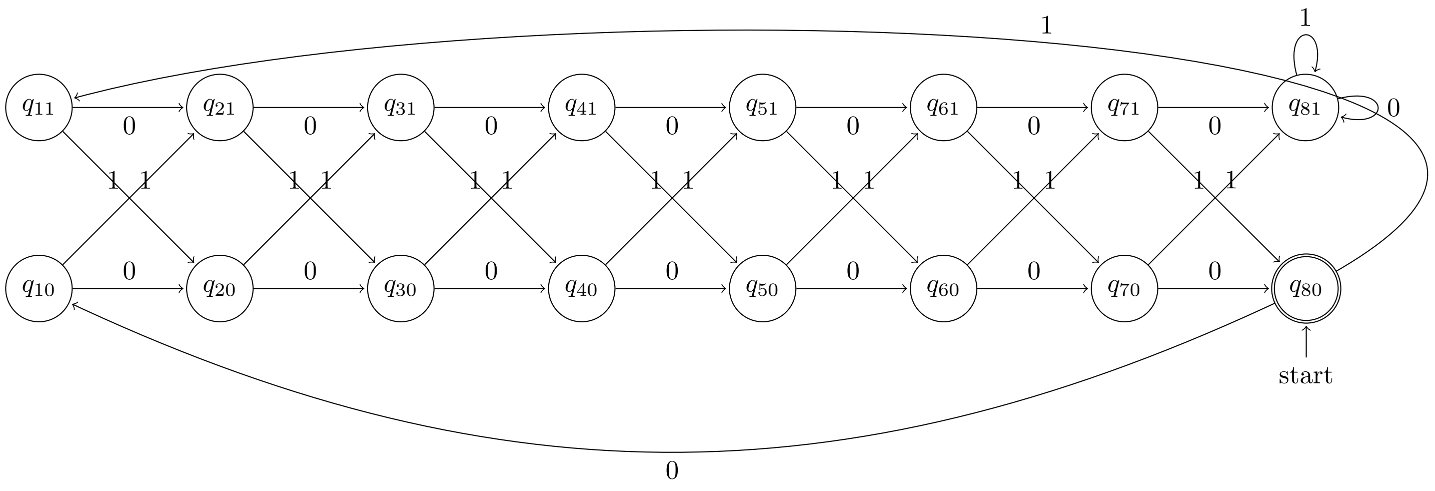
1 The model of Latent Dirichlet Allocation

The model of Latent Dirichlet Allocation is a generative probabilistic model for collections of discrete data [1]. Therefore it qualifies to be used for the explorative analysis of text corpora.

1.1 Structure



1.2 Variational Inference

2 Imlementation

2.1 Preparation

First we have to choose a text corpus to estimate the parameters of the LDA-model from. For this we use a dump from the simple english wikipedia [2], namely the version named 'All pages, current versions only'.

For the purpose of speeding up the development process we will perform our operations on an even smaller subset of only 5 articles, to avoid long loading times on every run. We split some articles of the downloaded file into a smaller file with the script in section 4.1.

2.2 Class : Dataset

We concentrate all operations regarding the preprocessing of the data in class named Dataset, which can be examined in section 4.2. Our goal here is to process all articles and end up with a matrix of wordcounts, each row representing an article and each column representing a word. Therefore we have to establish a common dictionary containing all occurring words from all

2.3 Class : LDA

3 Applying LDA

Literatur

[1] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, “Latent dirichlet allocation,” 2003.

[2] “simplewiki dump progress on 20181120,” https://dumps.wikimedia.org/simplewiki/20181120/, (Accessed on 12/15/2018).

4 Appendix

4.1 Extract subset

```bash
#!/usr/bin/env bash

cd dataset

#Choose how many pages to extract
n=1000
#Get line number of n-th occurence of the closing tag "</page>"
lineNum=$(
  grep -n  "</page>" simplewiki-20181120-pages-meta-current.xml | \
    head -n${n} | \
    tail -n1 | \
    cut -d: -f1 \
  )

#Copy large set upto lineNum into the file small.xml
head simplewiki-20181120-pages-meta-current.xml -n${lineNum} > small.xml

#Add closing tag to small.xml
echo "</mediawiki>" >> small.xml
```

4.2 Class : Dataset

```python
import xml.etree.ElementTree as ET
import multiprocessing as mp
import gensim as gsm
import scipy.sparse as sps
import scipy as sp
import numpy as np
from functools import partial
import time
from tqdm import tqdm


def preprocessText(page, onlyOverview=True):
    # Shortcut to get it running
    return gsm.parsing.preprocessing.preprocess_string(page)


def tuples2Matrix(tuples,
                  dictionarySize,
                  matrixFormat,
                  matrixDType):
    matrix = matrixFormat(
        (1, dictionarySize),
        dtype=matrixDType
    )

    for t in tuples:
        matrix[0, t[0]] = t[1]

    return matrix


class DataSet:
    def __init__(self,
                 path='../dataset/small.xml',
                 sparse=True,
                 verbose=False,
                 matrixFormat='sparse',
                 matrixDType=sp.int8):

        self.path = path
        if (matrixFormat == 'sparse'):
            self.matrixFormat = sps.lil_matrix
        else:
            self.matrixFormat = np.matrix
        self.matrixDType = matrixDType

        if verbose:
            print("Dataset => Loading File")

        start = time.perf_counter()
        self.loadXMLFile()
        end = time.perf_counter()
        self.loadTime = end - start

        if verbose:
            print("Dataset => Parsing " + str(self.numOfDocuments())
                  + " documents took: " + "{:10.4f}".format(self.loadTime) + "s")

        if verbose:
            print("Dataset => Building Matrix")
        start = time.perf_counter()
        self.buildMatrix()
        end = time.perf_counter()
        self.buildMatrixTime = end - start

        if verbose:
            print("Dataset => Building " + str(self.matrixFormat) + " " +
                  str(self.matrix.shape) + " took: " + "{:10.4f}".format(self.buildMatrixTime) + "s")

        if verbose:
            print("Dataset => Constructed")

    def numOfDocuments(self):
        return len(self.documents)

    def dictionarySize(self):
        return len(self.dictionary)

    def buildMatrix(self):
        partialTuples2Matrix = partial(tuples2Matrix,
                                       dictionarySize=self.dictionarySize(),
                                       matrixFormat=self.matrixFormat,
                                       matrixDType=self.matrixDType)

        with mp.Pool(mp.cpu_count() - 1) as p:
            counts = p.map(self.dictionary.doc2bow, tqdm(
                self.documents, desc='Counting words'))
            rows = p.map(partialTuples2Matrix, tqdm(
                counts, desc='Generating sparse document vectors'))

        self.matrix = sps.vstack(rows)

    def loadXMLFile(self):
        self.documents = list()
        root = ET.parse(self.path).getroot()
        xmlNamespaces = {'root': 'http://www.mediawiki.org/xml/export-0.10/'}

        # Extract text-attribute of pages in Wikipedia-namespace '0'
        texts = [
            page.find('root:revision', xmlNamespaces)
            .find('root:text', xmlNamespaces).text
            for page in root.findall('root:page', xmlNamespaces)
            if 0 == int(page.find('root:ns', xmlNamespaces).text)
        ]
        if self.verbose:
            print('Parse xml')

        # Parallel preprocessing of pages
        with mp.Pool(mp.cpu_count() - 1) as p:
            self.documents = p.map(preprocessText, tqdm(
                texts, desc='Preprocessing text'))

        # Build gensim dictionary
        self.dictionary = gsm.corpora.dictionary.Dictionary(self.documents)


if __name__ == '__main__':
    dataset = DataSet()
    print(dataset.numOfDocuments())
    print(dataset.dictionarySize())
    print(dataset.matrix.T.dot(dataset.matrix))
```

4.3 Class : LDA

```
from lda.dataset import DataSet
from tqdm import tqdm
import time


class LDA():
    def __init__(self,
                 maxit=1000,
                 verbose=False
                 ):
        self.verbose = verbose
        self.maxit = maxit

        self.alpha = None
        self.beta = None
        self.iterations = 0
        self.converged = False

    def expectation(self):
        """ For each document, find the optimizing values of the variational parameters """
        # print("Expect")
    def maximization(self):
        """ Maximize the resulting lower bound on the log likelihood """
        # print("Maximize")

    def numOfDocuments(self):
        return self.dataset.numOfDocuments()

    def numOf(self):
        return self.dataset

    def fit(self, dataset):
        self.dataset = dataset
        self.multinomials = 1
        if self.verbose:
            print("LDA => fitting to Dataset: " + str(dataset.matrix.shape))

        start = time.perf_counter()

        for iteration in tqdm(range(self.maxit), desc='EM: '):
            self.expectation()
            self.maximization()
            self.iterations += 1
            if self.verbose:
                print("LDA.fit() => iteration: " + str(self.iterations))
            if self.iterations > self.maxit:
                self.converged = True
                if self.verbose:
                    print("LDA.fit() => Maximum number of iterations reached!")

        end = time.perf_counter()

        self.inferenceTime = end - start

        if self.verbose:
            print("LDA => Fitting took: " +
                  "{:10.4f}".format(self.inferenceTime) + "s")


if __name__ == '__main__':
    dataset = DataSet()
    model = LDA()
    # model.fit(dataset)
```