# Homework 2

刘佳迎
2021 秋 - 大数据分析

2021 年 11 月 15 日

## 1 LSI 算法

已知词项文档矩阵

$$C=\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

利用 LSI 算法，求词项与文档各自的 3 维表示。

文本挖掘中，主题模型、聚类算法关注于从样本特征的相似度方面将数据聚类。比如通过数据样本之间的欧式距离，曼哈顿距离的大小聚类等。而主题模型，是对文字中隐含主题的一种建模方法。比如从“人民的名义”和“达康书记”这两个词我们很容易发现对应的文本有很大的主题相关度，但是如果通过词特征来聚类的话则很难找出，因为聚类方法不能考虑到到隐含的主题这一块。找到隐含的主题，一般都是基于统计学的生成方法。即假设以一定的概率选择了一个主题，然后以一定的概率选择当前主题的词。最后这些词即为当前的文本的表达。所有词的统计概率分布可以从语料库获得。

潜在语义索引 (LSI)，又称为潜在语义分析 (LSA)，可以解决两类问题，一类是一词多义，如“bank”一词, 可以指银行, 也可以指河岸; 另一类是一义多词, 即同义词问题, 如“car”和“automobile”具有相同的含义，如果在检索的过程中，在计算这两类问题的相似性时，依靠余弦相似性的方法将不能很好的处理这样的问题。所以提出了潜在语义索引的方法，利用 SVD 降维的方法将词项和文本映射到一个新的空间，本题目中，利用 SVD 算法将文本和词项映射为三维表示，下文为代码实现部分。

In [ ]:

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from scipy.sparse.linalg import svds
from scipy import sparse
import seaborn as sns
from matplotlib.axis import Axis

#基于矩阵分解的结果，复原矩阵
def rebuildMatrix(U, sigma, V):
    a = np.dot(U, sigma)
    a = np.dot(a, np.transpose(V))
    return a

def GetMatrixImage(fig,fignum,vt,vt_name):
    for i in range(len(vt)-1):
        ax = fig.add_subplot(9,2,i*2+fignum)
        # Set tick font size
        for label in (ax.get_xticklabels() + ax.get_yticklabels()):
            label.set_fontsize(16)
        # Show ticks in the left and lower axes only
        Axis.set_label_coords(ax.yaxis,0, 0.5)
        Axis.set_label_coords(ax.xaxis,0.5, 0)
        ax.spines['bottom'].set_position(('data', 0))
        ax.spines['left'].set_position(('data', 0))
        g = sns.scatterplot( x="v{}".format(i+1), y="v{}".format(i+2),data=pd
            "v{}".format(i+1):vt[i],
            "v{}".format(i+2):vt[i+1]
        }),palette="Set2")
        img_title = "Spectral Subspace Plots of {0}{1} and {0}{2} ".format(vt_
        g.set_title(img_title,fontsize = 20)
        ax.set_xlabel(vt_name+str(i+1),fontsize = 20)
        ax.set_ylabel(vt_name+str(i+2),fontsize = 20)
        for p in ax.patches:
            height = p.get_height()
            ax.text(p.get_x()+p.get_width()/2.,
                    height + 3,
                    '{:1.2f}%'.format(100*height/len(vt)),
                    ha="center")
```

# 1.LSI

In [ ]:

```
data = [
    [1,0,1,0,0,0],
    [0,1,0,0,0,0],
    [1,1,0,0,0,0],
    [1,0,0,1,1,0],
    [0,0,0,1,0,1]]
data
```

Out[ ]:

```
[[1, 0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0, 0],
 [1, 1, 0, 0, 0, 0],
 [1, 0, 0, 1, 1, 0],
 [0, 0, 0, 1, 0, 1]]
```

In [ ]:

```
yelp_matrix = sparse.coo_matrix(data, dtype=float)
print(yelp_matrix)
```

```
u, s, vt = svds(yelp_matrix,k=3,which = 'LM')
```

```
  (0, 0)        1.0
  (0, 2)        1.0
  (1, 1)        1.0
  (2, 0)        1.0
  (2, 1)        1.0
  (3, 0)        1.0
  (3, 3)        1.0
  (3, 4)        1.0
  (4, 3)        1.0
  (4, 5)        1.0
```

### 三维表示的特征值

In [ ]:
```
s_matric = np.diag(s)
print(s_matric)
```

```
[[1.27529025 0.         0.        ]
 [0.         1.59438237 0.        ]
 [0.         0.         2.16250096]]
```

### 词项的三维表示

In [ ]:
```
u
```

Out[ ]:
```
array([[-0.56949758,  0.29617436,  0.44034748],
       [ 0.5870217 ,  0.33145069,  0.12934635],
       [ 0.36768998,  0.51111524,  0.47553026],
       [-0.15490588, -0.35057241,  0.70302032],
       [ 0.4145917 , -0.64674677,  0.26267284]])
```

### 文档的三维表示

In [ ]:
```
vt.T
```

Out[ ]:
```
array([[-0.2797116 ,  0.28645399,  0.74862305],
       [ 0.74862305,  0.52845914,  0.2797116 ],
       [-0.44656311,  0.18576119,  0.2036288 ],
       [ 0.2036288 , -0.6255207 ,  0.44656311],
       [-0.12146715, -0.21987976,  0.32509596],
       [ 0.32509596, -0.40564094,  0.12146715]])
```

### 利用SVD表达的三维表示复原词项文档矩阵

In [ ]:
```
data_lsi = np.absolute(rebuildMatrix(u, s_matric, vt.T))
data_lsi
```

Out[ ]:
```
array([[1.05129307, 0.02780367, 0.60595265, 0.01803027, 0.29396119,
        0.31199146],
       [0.15137923, 0.91794411, 0.1791829 , 0.05321203, 0.11619747,
        0.06298543],
       [0.87211017, 1.06932334, 0.15137923, 0.04495516, 0.0981672 ,
        0.05321203],
       [1.0332628 , 0.01803027, 0.29396119, 0.98830764, 0.64113441,
        0.34717322],
       [0.01803027, 0.0097734 , 0.31199146, 1.00633791, 0.34717322,
        0.65916468]])
```

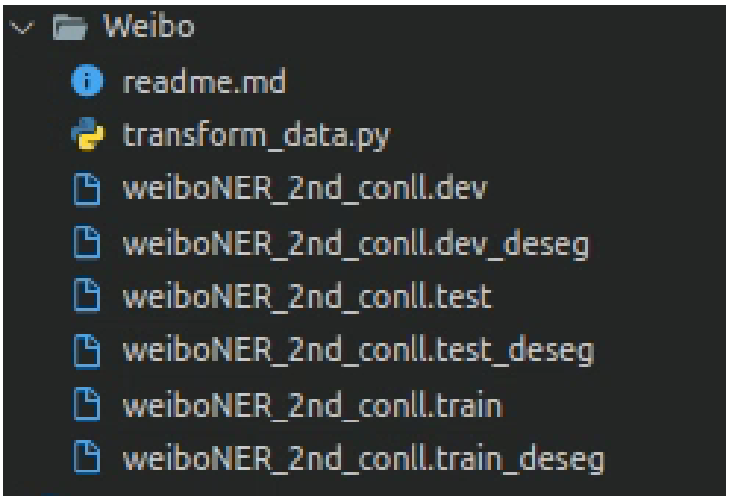
# 2 中文命名实体识别任务

## 2.1 题目背景

由于读研期间研究中文命名实体识别任务较多，且需要针对特定领域的小样本词汇进行 NER，需额外考虑引入专有词汇信息，因此本题目中采用了来自复旦大学邱锡鹏老师团队的 FLAT: Chinese NER Using Flat-Lattice Transformer 模型。复现论文在 weiboNER_2nd_conll 测试集识别 F1 精度 67.71，paperswithcode 官方代码测试精度为 68.55。

| Rank | Model | F1 | Precision | Recall | Accuracy-NE | Accuracy-NM | Overall | Paper | Code | Result | Year | Tags |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | FGN | 71.25 | | | | | | FGN: Fusion Glyph Network for Chinese Named Entity Recognition | | | 2020 | |
| 2 | SA-NER | 69.8 | | | | | | Named Entity Recognition for Social Media Texts with Semantic Augmentation | | | 2020 | |
| 3 | AESINER | 69.78 | | | | | | Improving Named Entity Recognition with Attentive Ensemble of Syntactic Information | | | 2020 | |
| 4 | Locate and Label | 69.16 | | | | | | Locate and Label: A Two-stage Identifier for Nested Named Entity Recognition | | | 2021 | |
| 5 | FLAT+BERT | 68.55 | | | | | | FLAT: Chinese NER Using Flat-Lattice Transformer | | | 2020 | |

## 2.2 数据源




本次用到的数据为中国社交媒体（微博）命名实体识别数据集（Weibo-NER-2015）（Weibo: https://github.com/OYE93/Chinese-NLP-Corpus/tree/master/NER/weibo）：包含地名、人名、机构名、行政区名四类。该语料库包含 2013 年 11 月至 2014 年 12 月期间从微博上采集的 1890 条信息，有两个版本 (weiboNER.conll 和 weiboNER_2nd_conll)，共 1890 样例，本次实验使用了 weiboNER_2nd_conll。并将其 BIO 格式转化为 BMES 格式文件。

## 2.3 实验环境及相关参数

| 参数 | 解释 | 参数值 |
|---|---|---|
| ff | feed-forward 中间层的节点个数 | 3 |
| layer | Transformer 中 Encoder_Layer 的数量 | 1 |
| head | multi-head-attn 中 head 的个数 | 8 |
| head_dim | multi-head-attn 中每个 head 的编码维度 | 20 |
| ff_activate | feed-forward 中的激活函数 | ‘relu’ |
| use_bigram | 是否使用双字符编码 | 1 |
| use_rel_pos | 是否使用相对位置编码 | True |
| pos_norm | 是否对位置编码进行 norm(pe/pe_sum) | False |
| k_proj | attn 中是否将 key 经过 linear 层 | False |
| q_proj | attn 中是否将 query 经过 linear 层 | True |
| v_proj | attn 中是否将 value 经过 linear 层 | True |
| r_proj | attn 中是否将相对位置编码经过 linear 层 | True |
| embed_dropout | embedding 中的 dropout | 0.5 |
| ff_dropout | ff 层中的 dropout | 0.15 |
| ff_dropout_2 | 第二个 ff 层中的 dropout | 0.15 |

## 2.4 运行结果

flat_main.py 5 | best_metric.log | fastNLP_module.py | paths.py | prepose.py


```
{"metric": {"SpanFPreRecMetric": {"f": 0.0, "pre": 0.0, "rec": 0.0}, "label_acc": {"acc": 0.93278}, "step": 135, "epoch": 1}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.0, "pre": 0.0, "rec": 0.0}, "label_acc": {"acc": 0.927275}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.029703, "pre": 0.4, "rec": 0.015424}, "label_acc": {"acc": 0.934303}, "step": 405, "epoch": 3}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.009368, "pre": 0.222222, "rec": 0.004785}, "label_acc": {"acc": 0.927747}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.071942, "pre": 0.535714, "rec": 0.03856}, "label_acc": {"acc": 0.935618}, "step": 540, "epoch": 4}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.05977, "pre": 0.764706, "rec": 0.0311}, "label_acc": {"acc": 0.929232}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.186253, "pre": 0.677419, "rec": 0.107969}, "label_acc": {"acc": 0.940048}, "step": 810, "epoch": 6}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.163793, "pre": 0.826087, "rec": 0.090909}, "label_acc": {"acc": 0.934224}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.306397, "pre": 0.443902, "rec": 0.233933}, "label_acc": {"acc": 0.940948}, "step": 1080, "epoch": 8}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.31145, "pre": 0.43038, "rec": 0.244019}, "label_acc": {"acc": 0.937462}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.46443, "pre": 0.485955, "rec": 0.44473}, "label_acc": {"acc": 0.944825}, "step": 1350, "epoch": 10}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.36, "pre": 0.406627, "rec": 0.322967}, "label_acc": {"acc": 0.917763}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.565909, "pre": 0.507128, "rec": 0.640103}, "label_acc": {"acc": 0.950433}, "step": 1485, "epoch": 11}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.467391, "pre": 0.428287, "rec": 0.514354}, "label_acc": {"acc": 0.92768}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.588095, "pre": 0.547672, "rec": 0.634961}, "label_acc": {"acc": 0.953894}, "step": 2835, "epoch": 21}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.567929, "pre": 0.53125, "rec": 0.610048}, "label_acc": {"acc": 0.947851}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.621336, "pre": 0.571121, "rec": 0.681234}, "label_acc": {"acc": 0.960125}, "step": 2970, "epoch": 22}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.569231, "pre": 0.526423, "rec": 0.619617}, "label_acc": {"acc": 0.951494}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.632111, "pre": 0.621891, "rec": 0.642674}, "label_acc": {"acc": 0.960817}, "step": 3105, "epoch": 23}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.607143, "pre": 0.604265, "rec": 0.610048}, "label_acc": {"acc": 0.953451}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.663172, "pre": 0.676471, "rec": 0.650386}, "label_acc": {"acc": 0.966424}, "step": 3240, "epoch": 24}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.623441, "pre": 0.651042, "rec": 0.598086}, "label_acc": {"acc": 0.960197}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.678804, "pre": 0.686842, "rec": 0.670951}, "label_acc": {"acc": 0.967809}, "step": 3915, "epoch": 29}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.655901, "pre": 0.682171, "rec": 0.631579}, "label_acc": {"acc": 0.963435}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.690998, "pre": 0.655889, "rec": 0.730077}, "label_acc": {"acc": 0.968432}, "step": 4050, "epoch": 30}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.65063, "pre": 0.624176, "rec": 0.679426}, "label_acc": {"acc": 0.963368}}}}
{"metric": {"SpanFPreRecMetric": {"f": 0.713407, "pre": 0.683962, "rec": 0.745501}, "label_acc": {"acc": 0.967809}, "step": 4455, "epoch": 33}}
{"metric": {"data_test": {"SpanFPreRecMetric": {"f": 0.677108, "pre": 0.682039, "rec": 0.672249}, "label_acc": {"acc": 0.964852}}}}
```


在 weiboNER_2nd_conll 训练得出的最优模型，其测试集识别 F1 精度 67.71，paperswithcode 官方代码测试精度为 68.55。