MAKING SENSE OF UNSTRUCTURED DATA

Code update by Juan David Correa astropema@google.com Feb 2025

- Added Graphical Output and Modified Code to use updated libraries.

Case Study - Data Analysis with Human-Generated Text

Instructor: Tamara Broderick

In this document, we walk through some tips to help you with doing your own analysis on MIT EECS faculty data using stochastic variational inference on LDA.

1. Scraping your own dataset
2. Pre-processing the dataset
3. Implementing your own LDA code

Implementing your own SVI-LDA code

Latent Dirichlet allocation (LDA) is a generative statistical model in natural language processing, and can be used to discover ‘topics’ in a large set of documents. This is first presented by David Blei, Andrew Ng, and Michael Jordan.

The key idea is that if we see a ‘topic’ as a collection of certain words, we can look at each document as a collection of topics, the proportion of each topic depends on the proportion of words in the document that are associated with that topic. For example, the ‘sports’ topic may consist of the words: tennis, football, gymnastics. When given a set of documents, we can calculate the posterior distribution for the topics. In the original LDA paper, this is done using a coordinate descent algorithm for mean-field variational inference, and later on researchers also used Gibbs Sampling and expectation propagation. In this tutorial we will be looking only at Stochastic Variational Inference for LDA. SVI was first published in 2013 by Matt Hoffman, David Blei, Chong Wang, and John Paisley.

Traditional coordinate-descent variational inference requires each update to be carried out with all of the data, and these updates become inefficient when the dataset gets large as each update scales linearly with the size of the data. The key idea with SVI is to update global variational parameters more frequently. Using local and global parameters, and given the dataset with a known number of datapoints, we could randomly take 1 data point at a time, update the local parameter, and project the change into the global parameters. Like traditional coordinate-descent variational inference, this is done until the result converges, i.e., the change in the global parameters is smaller than a certain value. The implementation we will be talking about is a naive implementation of the algorithm described in the original paper

. **Variable Notation**

Here we provide a brief overview of the input variables for LDA and SVI. Variables that can be set are the following:

• λ: what we want in the end (the posterior distribution for the topics for each word

• vocab: this is the overall vocabulary we will have in the docs

• K: this is the number of topics we want to get in the end

• D: this is the total number of documents

• α: parameter for per-document topic distribution

• η: parameter for per-topic vocab distribution

• τ: delay that down weights early iterations

• κ: forgetting rate, controls how quickly old information is forgotten; the larger the value, the slower it is.

• max:iterations: the number of maximum iterations the updates should go on for. We usually set a check such that if the difference in two consecutive values of λ is smaller than a certain value, we say the algorithm has converged. However, sometimes we could set this certain value too small, so we set a maximum iteration value to avoid updates running forever.

LDA Generative Model

We review the LDA generative model here. LDA assumes each document has K topics with different proportions. It models a corpus w of size D as follows:

• Draw distribution over vocabulary βk ~ Dirichlet(η) for topics k $\in$ {1...K}

• For each document d $\in$ {1...D} :

– Draw topic proportions θd ~ Dirichlet(α);

– For each word $Wd\ n$ in the document:

- Draw topic indicator $Zd\ n$~ Multinomial (θd)
- Draw word $Wd\ n$ ~ Multinomial (βZdn)

Note that this model follows the ‘bag of words’ assumption, such that given the topic proportions, each word drawn is independent of any other words in the document.

Libaries

In [17]:
```
from bs4 import BeautifulSoup
from requests import get
import nltk
from nltk import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
import collections
import pandas as pd
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/obaozai/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /Users/obaozai/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
The history saving thread hit an unexpected error (OperationalError('attempt
to write a readonly database')).History will not be written to the database.
```

LDA Generative Model

Priors:

- Distribution over vocabulary for topic k in {1..K}: beta[k] ~ Dirichlet(V, eta)
- Distribution over topics (latent variables): theta ~ Dirichlet(K, alpha)

For each document:

- Choose number of words: N ~ Poisson(ξ)

For each of the N words w:

- Choose a topic: z ~ Cat(K, theta)
- Choose a word: w ~ Cat(V, beta[z])

Note: This model follows the ‘bag of words’ assumption, such that given the topic proportions, each word drawn is independent of any other words in the document.

Graphical representation

Variational Inference

To use variational inference, the edges between θ (theta), z and w are removed to make inference on LDA model tractable.

Variational Inference

Global Variables

In [18]:
```
faculty_url = 'https://www.eecs.mit.edu/role/faculty/?fwp_role=faculty'
arXiv_format = 'arxiv.org/find/{}/1/au:+{}_{}/0/1/0/all/0/1' # arxiv.org/fir
search_url_format = 'https://arxiv.org/search/?query="{}"&searchtype=author'
subjects = {'Computer Science': 'Computer Science',
            'Electrical Engineering': 'Electrical Engineering and Systems Sc
            'Physics': 'Physics'}
all_papers_columns = ['Name', 'Abstract']
```

Web Sraping

1. Get Facultys

Using BeautifulSoup (https://www.crummy.com/software/BeautifulSoup/), and by analyzing the structure of the source code of arXiv, we could scrape the name list of MIT EECS faculty members. Using this information, we could list the query we send to arXiv. A possible format for the arXiv search for papers by authors is the following:

arxiv.org/find/(subject)/1/au:+(lastname)_(initial)/0/1/0/all/0/1

You could therefore adapt the names you scraped, and query through all the relevant arXiv search pages.

Within the arXiv source code, look for < class span=list-identifier >, which will give the identifier for the papers listed in your query results. Similarly look for the tag for the “Abstract” within each paper and scrape the abstract for each paper you find.

Note that you might want to scrape more information than you need and then do some local processing with the text you have instead.

In [19]:
```
from urllib.request import Request, urlopen
faculty_url = "https://www.eecs.mit.edu/role/faculty/?fwp_role=faculty"
hdr = {'User-Agent': 'Mozilla/5.0'}
req = Request(faculty_url,headers=hdr)
page = urlopen(req)
faculty_page_content = BeautifulSoup(page,'html.parser')
#print(faculty_page_content)
```

In [20]:
```
names=[]
names = [x.text for x in faculty_page_content.find_all("h5")]
```

2. Scrape Papers

In [21]:
```python
#scrapping abstracts
def scrapeArXiV(names):
    papers = list()
    for name in names:
        search_url = search_url_format.format(name.replace(' ', '+'))
        papers_author = get(search_url)
        papers_author_content = BeautifulSoup(papers_author.content, 'html.p
        papers_author_body = papers_author_content.body
        results = papers_author_body.find_all("li", class_="arxiv-result")
        abstracts = [result.find("span", class_="abstract-full") for result

        abstracts_content = [abstract.a.unwrap() for abstract in abstracts]
        abstracts_content = [abstract.contents[0] for abstract in abstracts]

        if abstracts_content:
            papers = papers + abstracts_content

    return papers
```

In [22]:
```python
papers = scrapeArXiV(names)
```

Text Preprocessing

Pre-processing the dataset

In the original work we have processed the data as raw documents as the dataset size was small. However if you want to use Matthew Hoffman's original SVI code instead, that code takes a text file with a specific format. Once you have each abstract in a separate text file, you may find the 2017 © Massachusetts Institute of Technology following Python packages useful: io, collections, nltk. It is good practice to keep your dataset in its own folder, so io can be used to access that folder using a constant (relative) path. Read each file and use nltk.tokenize to tokenize each chunk of text. Use collections to process each abstract using a Counter/Dictionary, before writing the counts of words of each individual abstract as a line in the text file.

In [23]:
```python
def word_cleaning_and_count(s):
    s_lower = s.lower()

    cleaning_set = set(stopwords.words('english'))
    tokens = word_tokenize(s_lower)
    tokens = [token for token in tokens if token.isalpha()]
    word_dict = dict(collections.Counter(tokens))
    for key in cleaning_set:
        word_dict.pop(key, None)
    return word_dict
```

In [24]:
```python
papers_word_dict = [word_cleaning_and_count(paper) for paper in papers]
dup_keys = []
for i in range(len(papers_word_dict)):
```

```
    dup_keys = dup_keys + list(papers_word_dict[i].keys())

vocab = list(collections.Counter(dup_keys).keys())
lookup_table = dict(zip(vocab, range(len(vocab))))
```

Save data

In [25]:
```
import json
with open('data/names', 'w') as fout:
    json.dump(names, fout)
with open('data/papers', 'w') as fout:
    json.dump(papers, fout)
with open('data/papers_word_dict', 'w') as fout:
    json.dump(papers_word_dict, fout)
with open('data/vocab', 'w') as fout:
    json.dump(vocab, fout)
with open('data/lookup_table', 'w') as fout:
    json.dump(lookup_table, fout)
```

LDA

In [26]:
```
#similar to k in K-means clustering. We want to divide abstracts into 5 topi
no_topics = 5
```

Load data

Please make an empty folder named data in your working directory

In [27]:
```
import json
with open('data/names', 'r') as json_file:
    names = json.load(json_file)
with open('data/papers', 'r') as json_file:
    papers = json.load(json_file)
with open('data/papers_word_dict', 'r') as json_file:
    papers_word_dict = json.load(json_file)
with open('data/vocab', 'r') as json_file:
    vocab = json.load(json_file)
with open('data/lookup_table', 'r') as json_file:
    lookup_table = json.load(json_file)

vocab_size = len(vocab)
```

Using sklearn

In [28]:
```
doc_vecs = []
for paper in papers_word_dict:
    doc_vec = [0 for _ in range(vocab_size)]
    for token, occurs in paper.items():
```

```
        doc_vec[lookup_table[token]] = occurs
    doc_vecs.append(doc_vec)
```

In [29]:
```
from sklearn.decomposition import LatentDirichletAllocation

# Run the LDA
lda = LatentDirichletAllocation(n_components=no_topics, learning_method='onl

def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print('Topic %d:' % (topic_idx))
        print(' '.join([vocab[i] for i in topic.argsort()[:-no_top_words - 1

#using top 10 words present in each topic
no_top_words = 10
display_topics(lda, doc_vecs, no_top_words)
```

```
Topic 0:
channel quantum network error communication memory performance codes capacit
y decoding
Topic 1:
quantum optical materials energy using devices systems magnetic applications
material
Topic 2:
system design data users robot agents planning user task agent
Topic 3:
models data model learning training performance tasks methods language neura
l
Topic 4:
algorithm n algorithms problem show time graph model problems data
```

- Topic 0: It may contain abstracts from traditional computer science
- Topic 1: It may contain abstracts from modern computer science and Graph algorithms
- Topic 2: It may contain abstracts from machine learning and deep learning.
- Topic 3: It may contain abstracts belonging to modern computer science and machine learning.
- Topic 4: It may contain topics from quantum theory and material science.

End-to-end Code (SVILDA algorithm)

In [30]:
```
doc_vecs = []
for paper in papers_word_dict:
    wordslist = []
    countslist = []
    for token, occurs in paper.items():
        wordslist.append(lookup_table[token])
        countslist.append(occurs)
    doc_vecs.append((wordslist, countslist))
```

In [31]:
```
from svilda import SVILDA
iterations = 10000
```

```
lda = SVILDA(vocab, no_topics, len(doc_vecs), 0.1, 0.01, 1, 0.75, iterations
lda.runSVI(doc_vecs)
```

```
ITERATION 0  running document number  1650
ITERATION 100  running document number  1333
ITERATION 200  running document number  117
ITERATION 300  running document number  1590
ITERATION 400  running document number  318
ITERATION 500  running document number  2679
ITERATION 600  running document number  3552
ITERATION 700  running document number  3316
ITERATION 800  running document number  3819
ITERATION 900  running document number  3498
ITERATION 1000  running document number  1935
ITERATION 1100  running document number  3091
ITERATION 1200  running document number  1757
ITERATION 1300  running document number  3634
ITERATION 1400  running document number  2802
ITERATION 1500  running document number  3444
ITERATION 1600  running document number  3159
ITERATION 1700  running document number  974
ITERATION 1800  running document number  2620
ITERATION 1900  running document number  2994
ITERATION 2000  running document number  2883
ITERATION 2100  running document number  2993
ITERATION 2200  running document number  23
ITERATION 2300  running document number  3137
ITERATION 2400  running document number  1037
ITERATION 2500  running document number  1734
ITERATION 2600  running document number  2169
ITERATION 2700  running document number  1217
ITERATION 2800  running document number  758
ITERATION 2900  running document number  3438
ITERATION 3000  running document number  3324
ITERATION 3100  running document number  863
ITERATION 3200  running document number  929
ITERATION 3300  running document number  424
ITERATION 3400  running document number  3475
ITERATION 3500  running document number  54
ITERATION 3600  running document number  2432
ITERATION 3700  running document number  2149
ITERATION 3800  running document number  2728
ITERATION 3900  running document number  1721
ITERATION 4000  running document number  1985
ITERATION 4100  running document number  2782
ITERATION 4200  running document number  1104
ITERATION 4300  running document number  2340
ITERATION 4400  running document number  877
ITERATION 4500  running document number  1975
ITERATION 4600  running document number  3107
ITERATION 4700  running document number  2016
ITERATION 4800  running document number  3432
ITERATION 4900  running document number  2899
ITERATION 5000  running document number  354
ITERATION 5100  running document number  3380
ITERATION 5200  running document number  3598
ITERATION 5300  running document number  553
ITERATION 5400  running document number  2083
ITERATION 5500  running document number  1027
```

```
ITERATION 5600  running document number  2117
ITERATION 5700  running document number  2691
ITERATION 5800  running document number  2025
ITERATION 5900  running document number  495
ITERATION 6000  running document number  1101
ITERATION 6100  running document number  242
ITERATION 6200  running document number  3872
ITERATION 6300  running document number  3391
ITERATION 6400  running document number  3528
ITERATION 6500  running document number  494
ITERATION 6600  running document number  2198
ITERATION 6700  running document number  150
ITERATION 6800  running document number  1717
ITERATION 6900  running document number  1927
ITERATION 7000  running document number  807
ITERATION 7100  running document number  3657
ITERATION 7200  running document number  2235
ITERATION 7300  running document number  3548
ITERATION 7400  running document number  545
ITERATION 7500  running document number  1251
ITERATION 7600  running document number  2420
ITERATION 7700  running document number  1895
ITERATION 7800  running document number  985
ITERATION 7900  running document number  2504
ITERATION 8000  running document number  1078
ITERATION 8100  running document number  2436
ITERATION 8200  running document number  1594
ITERATION 8300  running document number  1596
ITERATION 8400  running document number  1364
ITERATION 8500  running document number  1840
ITERATION 8600  running document number  3666
ITERATION 8700  running document number  190
ITERATION 8800  running document number  396
ITERATION 8900  running document number  1256
ITERATION 9000  running document number  2624
ITERATION 9100  running document number  1671
ITERATION 9200  running document number  1690
ITERATION 9300  running document number  1783
ITERATION 9400  running document number  2160
ITERATION 9500  running document number  1245
ITERATION 9600  running document number  4
ITERATION 9700  running document number  1909
ITERATION 9800  running document number  2081
ITERATION 9900  running document number  3755
```

In [32]:

```python
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model._lambda):
        print('Topic %d:' % (topic_idx))
        print(' '.join([vocab[i] for i in topic.argsort()[:-no_top_words - 1

no_top_words = 10
display_topics(lda, doc_vecs, no_top_words)
```

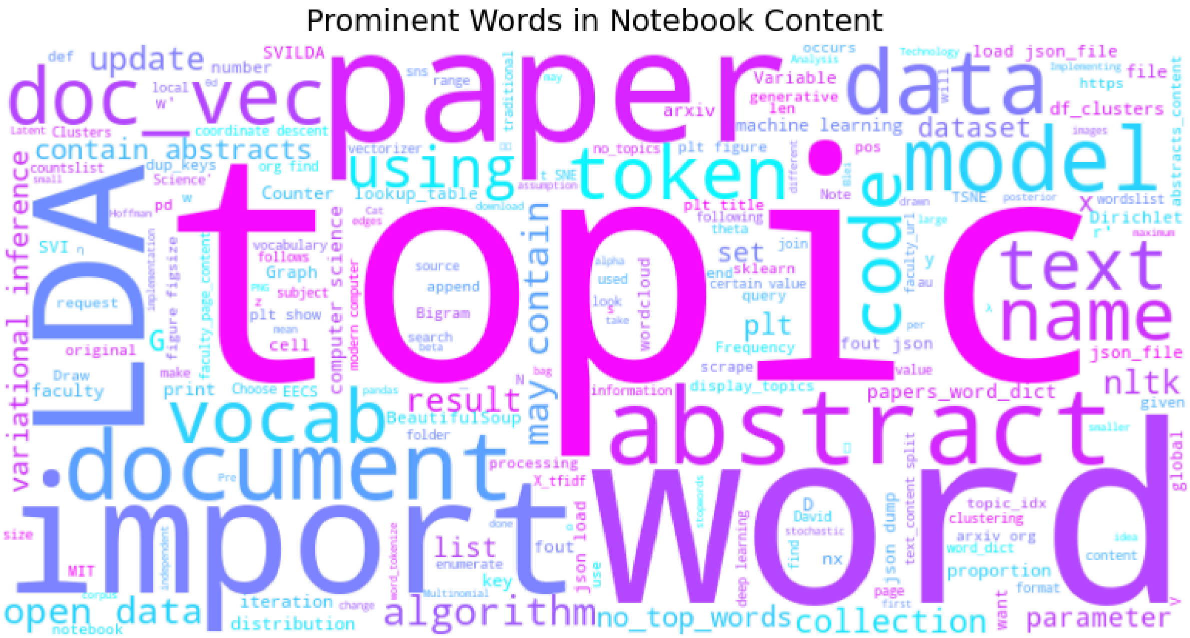
```
Topic 0:
algorithm n work propose demonstrate language set design quantum provide
Topic 1:
models present tasks training systems given space image linear often
Topic 2:
model using problem results new approach graph networks use large
Topic 3:
learning show also two number information network neural k different
Topic 4:
data algorithms performance time method paper methods problems framework fir
st
```

- Topic 0: It may contain abstracts from machine learning and deep learning.
- Topic 1: It may contain abstracts from machine learning algorithms.
- Topic 2: IIt may contain abstracts from modern computer science and machine learning algorithms
- Topic 3: It may contain abstracts from machine learning algorithms and Graph algorithms
- Topic 4: It may contain topics from computer science systems.

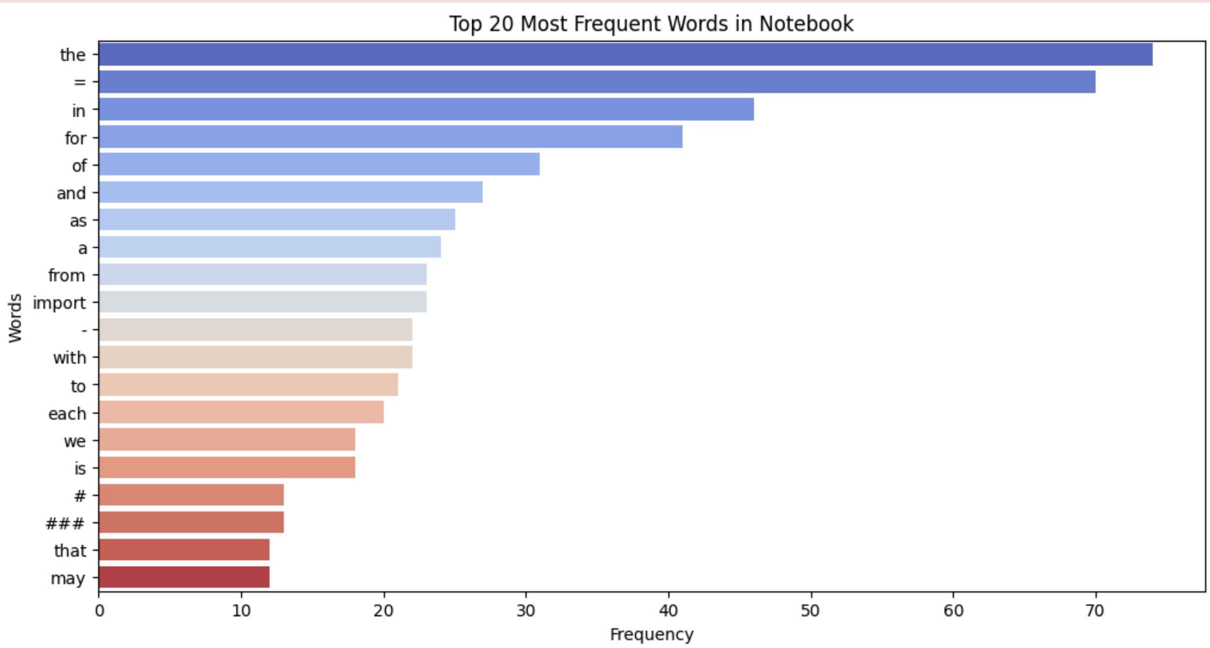
In [35]:

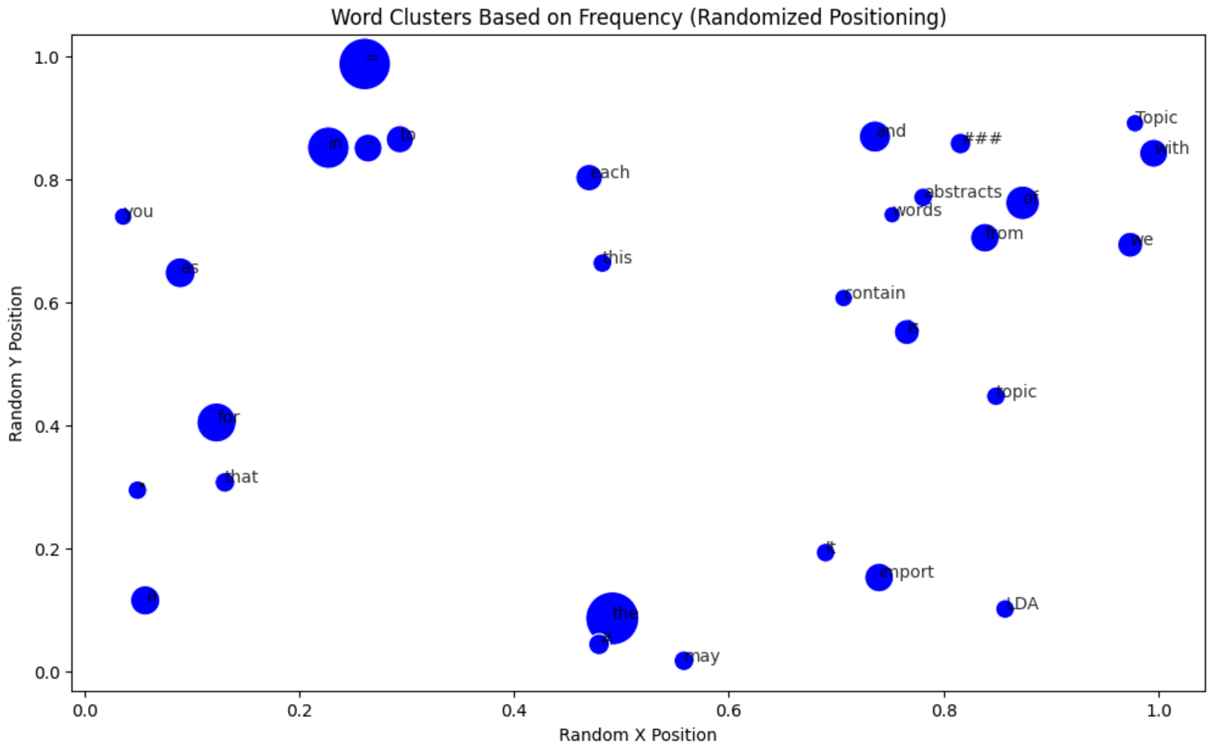
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
import random
from wordcloud import WordCloud
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.util import bigrams

# Load text from notebook (Make sure to replace 'notebook_data' with your ac
text_content = "\n".join([cell["source"] for cell in notebook_data["cells"]

### Word Cloud: Most Prominent Words ###
wordcloud = WordCloud(width=800, height=400, background_color="white", color
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("Prominent Words in Notebook Content", fontsize=14)
plt.show()

### Word Frequency Bar Chart ###
words = text_content.split()
word_counts = Counter(words)
common_words = word_counts.most_common(20)
word_freq_df = pd.DataFrame(common_words, columns=["Word", "Frequency"])

plt.figure(figsize=(12, 6))
sns.barplot(x="Frequency", y="Word", data=word_freq_df, palette="coolwarm")
plt.title("Top 20 Most Frequent Words in Notebook")
plt.xlabel("Frequency")
plt.ylabel("Words")
```

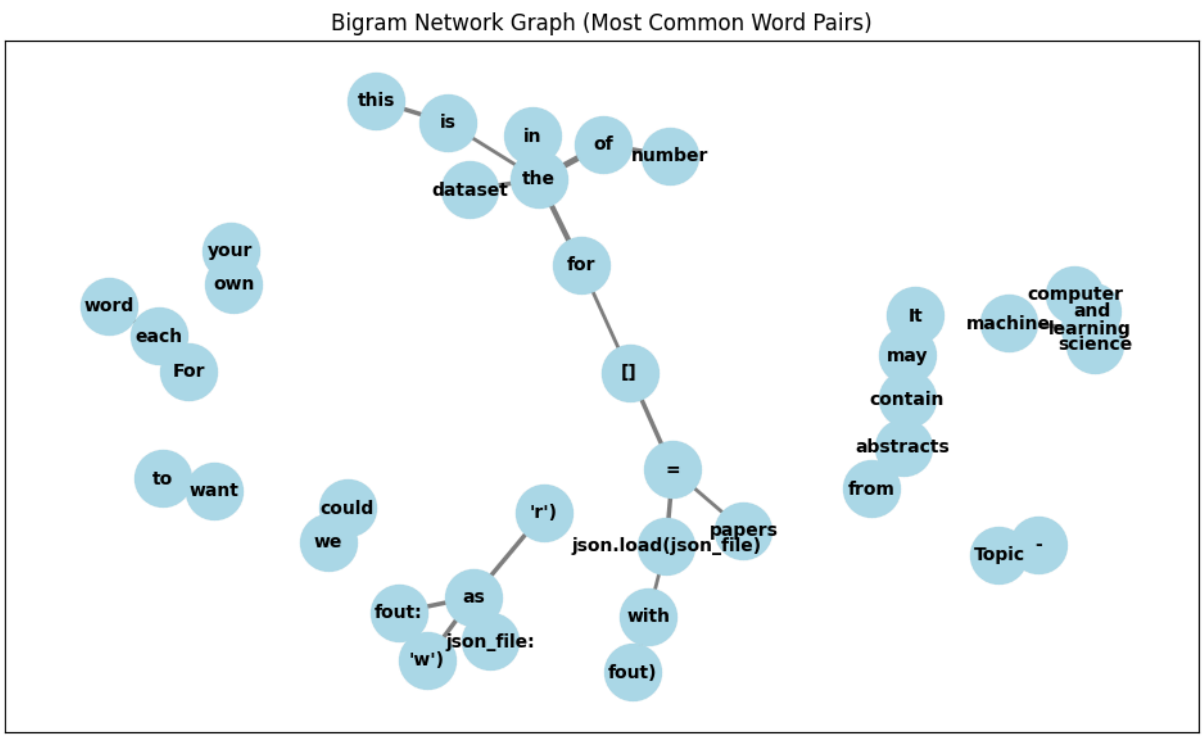
```
plt.show()

### Word Clusters Based on Frequency (Randomized Positioning) ###
top_words = word_counts.most_common(30)  # Get top 30 words
df_clusters = pd.DataFrame(top_words, columns=["word", "frequency"])

# Assign random x, y positions for visualization
df_clusters["x"] = [random.uniform(0, 1) for _ in range(len(df_clusters))]
df_clusters["y"] = [random.uniform(0, 1) for _ in range(len(df_clusters))]

plt.figure(figsize=(12, 7))
sns.scatterplot(x="x", y="y", size="frequency", data=df_clusters, sizes=(100

# Add word labels
for i, txt in enumerate(df_clusters["word"]):
    plt.annotate(txt, (df_clusters["x"][i], df_clusters["y"][i]), fontsize=1

plt.title("Word Clusters Based on Frequency (Randomized Positioning)")
plt.xlabel("Random X Position")
plt.ylabel("Random Y Position")
plt.show()

### Bigram Network Graph (Common Word Pairs) ###
word_tokens = text_content.split()
bigram_list = list(bigrams(word_tokens))
bigram_counts = Counter(bigram_list).most_common(30)

G = nx.Graph()
for (word1, word2), freq in bigram_counts:
    G.add_edge(word1, word2, weight=freq)

plt.figure(figsize=(12, 7))
pos = nx.spring_layout(G, k=0.5)
nx.draw_networkx_nodes(G, pos, node_color="lightblue", node_size=1000)
nx.draw_networkx_edges(G, pos, width=[G[u][v]['weight'] / 2 for u, v in G.ec
nx.draw_networkx_labels(G, pos, font_size=10, font_weight="bold")
plt.title("Bigram Network Graph (Most Common Word Pairs)")
plt.show()
```




```
/var/folders/q0/xfs5xjxx50xdjh4tzn1psdnw0000gn/T/ipykernel_61032/3045815497.
py:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed
in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the
same effect.

  sns.barplot(x="Frequency", y="Word", data=word_freq_df, palette="coolwar
m")
```







Conclusion

- LDA gives better result as compared to SVILDA.
- LDA is able to group topics more precisely into different and meaningful clusters.
- The research papers are mostly related to algorithms, core computer science and machine/deep learning.
- Other than computer science the topics are related to Quantum theory and material science.