CS410 Project Progress Report

Topic: Reproducing a Paper: Mining causal topics in text data: Iterative topic modeling with time series feedback.

Team: PYM

First	**Last**	**email**
Pallavi	Ravada	pravada2@illinois.edu
Yash	Skhwani	yashas2@illinois.edu
Michael	Bernardoni	mlb12@illinois.edu

Tasks completed:

- Data mining and cleaning of the text documents – completed.
- Data mining and cleaning of the betting probabilities – completed.
- Production of the word/time slice coverage for the time period – completed.
- Topic mining algorithm selection – selected Gensim LDA.
- Topic mining on the entire corpus – LDA algorithm implemented, baseline created
- Production of the topic coverage from one run of LDA – completed.
- Adding new topics back into the LDA algorithm after iteration – completed.
- Ability to iterate over the prior 3 steps for the entire time period - completed
- Understanding the ITMTF algorithm – completed.
- Completed the setup instructions for Gensim in anaconda – completed.

Tasks to do:

- Code the time sequence scoring function
- Code the word analysis scoring function
- Code the topic splitting
- Visualization of the final data

Challenges:

- The article has a section on μ. A μ of 0 means the prior is not considered. The higher the μ the stronger the prior. Gensim LDA has a concept of decay (between 0 and 1). Like a μ set to 0, if the decay is set to 1, the prior is not considered (similar to a μ of 0). The Gensim documentation states that a decay set between .5 and 1 will converge. We ran a test with a very small decay (.001), and the topic we added remained virtually intact. We are still discovering how the decay changes the ITMTF algorithm, hopefully similar to μ.
- We are still discussing the topics to be carried forward to the next iteration. The article proposes a “variable” topic approach and discusses keeping “buffer” topics. Not a challenge, but something we are looking at.

Detailed discussion of these steps follow:

Detailed discussion

Data mining and cleansing

Step 1: Data mining
First we mined the raw xml data and produced a .txt for each document that had a paragraph with the words “Gore” or “Bush”. We only included the paragraphs with the key words, but we kept the document intact, that is if a doc had 2 paragraphs with either the word “Bush” or “Gore” the output would be one document with those 2 paragraphs.

Note this is just prep work and is not included in the project for size considerations.

Step 2: Data cleansing - .\lda_data\LDAData.csv
For each file in the mined directory, we split the string into words. For each word we made each word lowercase, stripped out any character that was not alpha, and removed all stop words. We used stop words from: Onix Text Retrieval Toolkit Stop Word List 1:
https://www.lextek.com/manuals/onix/stopwords1.html .

We added the results for each document in a .csv file .\lda_data\LDAData.csv. Each document is a row: cell 1 contains the year; cell 2 contains the month; cell 3 contains the day; cell 4 contains the cleansed text string of the document

We also created a csv file .\lda_data\vocabulary.csv which contains unique vocabulary words in cell 1 and the count of the term in cell 2.

Step3: Data reduction - .\lda_data\LDAreduced.csv
Using the vocabulary csv .\lda_data\vocabulary.csv from step 2, we removed any word that only occurred once or twice (all words with counts over 2 were kept). We produced a csv file .\lda_data\vocabularyreduced.csv which contains the new list of unique vocabulary words.

Using the new vocabulary, we created a new csv .\lda_data\LDAreduced.csv in the same form as the un-reduced csv.

Step 3: Word coverage per time slice - .\lda_data\wordseries.csv
Using the vocabularyreduced.csv and the LDAreduced.csv we pre-processed a csv that contains the word coverage per time slice - .\lda_data\wordseries.csv. The first row is a header row that contains the unique words in the vocabulary, this row is not needed by the algorithm but was used during debugging. The first column in each row contains the time slice. All subsequent columns contain the word coverage during that time slice. This pre-processed file generates the word coverage over time used in the ITMTF algorithm.

Current data mining, cleansing, and pre-processed files in the project:

.\lda_data\LDAData.csv	cleaned data
.\lda_data\vocabulary.csv	cleaned data’s vocabulary
.\lda_data\LDAreduced.csv	removed words occurring 1 or 2
.\lda_data\vocabularyreduced.csv	removed data’s vocabulary list
.\lda_data\wordseries.csv	words counts per time slice

Topic Mining Algorithm Selection

The paper indicates that any topic mining algorithm can be used, but the author used the PLSA algorithm. As such, we attempted to use PLSA. First we discovered the PLSA algorithm pypi https://pypi.org/project/plsa/. The algorithm worked well in our test data sets, and had excellent data visualization techniques. The code did not have out of the box ways to add topics and topic priors, but the code was available. So we identified where to add new topics and where to set topic priors in the library's python code. However when we ran the full cleaned data, this library took over 12 hours to complete 1 model, and we would have to iterate 6-7 times.

One of our team members wrote a PLSA algorithm in C++. The C++ algorithm was significantly faster. However, running the entire corpus caused memory issues. Time does not permit adding sparse matrix processing or data swapping to disk.

Following the lead of other teams' discussion on Piazza, we then selected Gensim's LDA algorithm for topic mining https://radimrehurek.com/gensim/models/ldamodel.html#usage-examples. This algorithm does not have memory issues as it works on document chunks, and completes in a reasonable amount of time due to its use of sparse matrix processing (under 10 min on one of team member's home desktop). As an added bonus, the library had a way to add new topics and priors.

Instructions for adding this library into an Anaconda environment is in the appendix.

Algorithm Iteration Completed

The first step, loading the pre-processed files into arrays, is complete. The documents are loaded into memory. The pre-processed file had the date in the first cell. While the document is loaded into memory an array is created of the docs per time slice. This array will be used to create the topic coverage per iteration.

The word coverage preprocessed file is loaded into an array that can be used by the ITMFT algorithm.

The loaded docs are loaded into the Gensim's dictionary format. While loading the documents, a map is created that maps the index from the pre-processed vocabulary coverage to Gensim's token index for that word. This map is used to create the new topic matrix in the correct sequence, when new topics are returned from the ITMFT algorithm.

A Gensim corpus is created from the Gensim dict. As the documents do not change during the ITMFT algorithm (only the topic and topic priors), this corpus can be re-used throughout the entire ITMFT algorithm.

Running the Gensim LDA model is coded.

The ITMFT iteration is coded:

1. The preprocessed word coverage is passed into the algorithm
2. The document/topic probabilities are pulled from the LDA model
3. Using the document to time slice matrix, a topic coverage matrix is created from the document/topic probabilities.
4. These 2 coverage matrices are passed into the ITMFT scoring function
5. The ITMFT scoring function will create a new matrix of new topics and word probabilities prior
6. The coding of adding this matrix back into the LDA model is complete.

To do:

1. Code the ITMFT scoring function
2. Code the ITMFT topic creation function

Appendix

Adding Gensim LDA library to an Anaconda environment

Optional – create a new Anaconda environment to install the Gensim package:

1. Open Anaconda Navigator
2. Select Environments
3. Create an environment (i.e. "gensim")

Install genism in Anaconda

1. Open the Anaconda command prompt
2. If you created a new environment in the previous step:
   a. Activate the newly created environment if you created one ("Activate gensim")
   b. Run: conda install nb_conda_kernels (Proceed Y)
   c. Run: python -m ipykernel install --user --name myenv --display-name "Gensim"
      (you can use any display name you wish, this is what will show up on Jupyter Notebook)
   d. Run: pip install environment_kernels
3. Run: pip install --upgrade gensim

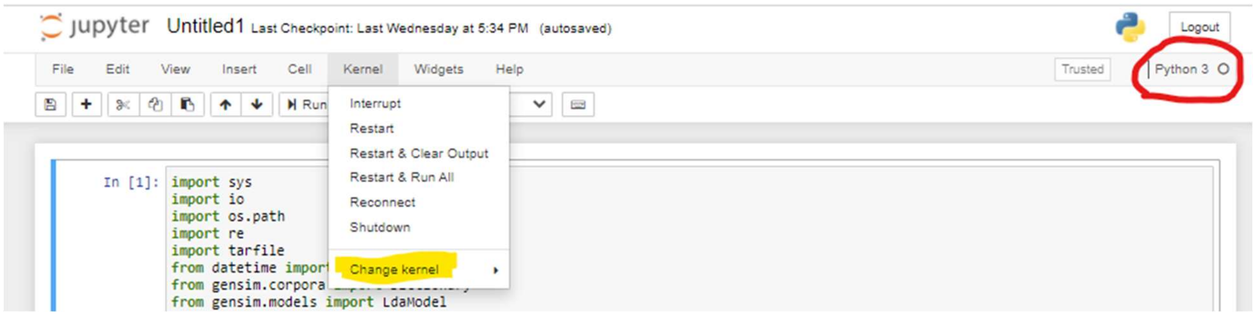
Start Jupyter Notebook in the directory you downloaded the project (if not your default)

1. Open the Anaconda command prompt
2. Start Jupyter Notebook in the directory you have downloaded this project
   (i.e., "jupyter notebook c:\projects")

TROUBLE SHOOTING NOTE:

When you open the project in Jupyter Notebook, look to the upper right and you can see what environment the project is running

If this is not the environment you just set up for Gensim, select Kernel from the notebook menu and select Change kernel, and change to the correct kernel.



```
In [1]: import sys
        import io
        import os.path
        import re
        import tarfile
        from datetime import
        from gensim.corpora
        from gensim.models import LdaModel
```