**CS410 Project Submission**

**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](mailto:pravada2@illinois.edu)

Yash Skhwani [yashas2@illinois.edu](mailto:yashas2@illinois.edu)

Michael Bernardoni [mlb12@illinois.edu](mailto:mlb12@illinois.edu)

A lot of prep work when into getting the project ready even before the ITMTF algorithm was analyzed in detail. First, data had to be collected, mined, prepped, and reduced into a form that could easily be loaded before each run. Furthermore topic mining and stats libraries had to be selected. Detailed analysis of these steps can be found in the Appendix.

Section 4.2.3 “While we observe correlations between non-textual series and both word streams and topic streams, we do not compute correlations for all word streams. Word level analysis would give us finer grain signals. However, generating all the word frequency time series and testing correlations would be very inefficient.”

Please refer to the jupyter notebook itmtf\_prerun\_stats to see the python code used to pre-process the granger and pearson statistics.

**de**

**d**



**D**

**d**

**Classical ITMTF Algorithm**

**D**

**C**



**Appendix - Topic Mining Algorithm Selection**

The paper indicates that the PSLA algorithm was used. As such, we attempted to us PSLA. First we discovered the PSLA algorithm pypi <https://pypi.org/project/plsa/>. The algorithm worked well in our test data sets, and had excellent data visualization techniques. We identified where to add new topics in the library’s python code with the iteration feedback. However when we ran the full cleaned data, this library took over 12 hours to complete 1 model.

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

Following the lead of other teams discussed 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, and completes in a reasonable amount of time (under 10 min on one of team member’s home desktop).

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

**Appendix - Data mining and cleansing**

The python code used to clean the data can be viewed in the itmtf\_cleaning jyputer notebook.

**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 used in the algorithm, but makes the file human readable. 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 will be used in the ITMTF algorithm.

**Current data mining and cleansing 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  
.\ LDA\_data\LDAwordseries.csv words counts per time slice

**Step 4: Betting information**The betting data is publicly available at the following site: <https://iemweb.biz.uiowa.edu/closed/pres00_WTA.html>

Python was used to clean the data, and smooth the data into both 3 day and 5 day averages. The python code can be viewed at the following site: [Bush Vs Gore Betting Data - Google Drive](https://drive.google.com/drive/folders/1d86_hgBWId2jAfqFlRt88psT5jGHl-CH)

**Topic Mining Algorithm Selection**

The paper indicates that the LDA algorithm was used. As such, we attempted to us LDA. First we discovered the LDA algorithm pypi <https://pypi.org/project/plsa/>. The algorithm worked well in our test data sets, and had excellent data visualization techniques. We identified where to add new topics in the library’s python code with the iteration feedback. However when we ran the full cleaned data, this library took over 12 hours to complete 1 model.

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

Following the lead of other teams discussed 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, and completes in a reasonable amount of time (under 10 min on one of team member’s home desktop).

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

**Appendix – Environment setup**

**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:
   1. Activate the newly created environment if you created one (“Activate gensim”)
   2. Run: conda install nb\_conda\_kernels (Proceed Y)
   3. 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)
   4. 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.

