<u>CS410 Project Submission</u>

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

Hyun Duk Kim, Malu Castellanos, Meichun Hsu, ChengXiang Zhai, Thomas Rietz, and Daniel Diermeier. 2013. Mining causal topics in text data: Iterative topic modeling with time series feedback. In Proceedings of the 22nd ACM international conference on information & knowledge management (CIKM 2013). ACM, New York, NY, USA, 885-890. DOI=10.1145/2505515.2505612

<u>Team</u>: PYM

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

Link to Video:
https://illinois.zoom.us/rec/play/p1k2d8OXy9zlMU52qAKeYekW6uxafweLBO2aqz6R_DGSbHkY06jjyYQTueHspfeX-Fep54MHEkpEDyt1.ORGucOcp6XSypU7d?continueMode=true

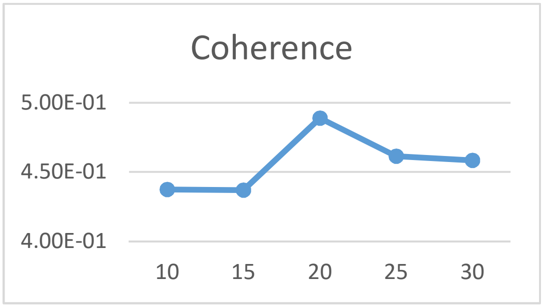
Initial Setup

A lot of preparation when into getting the environment 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 data curation steps and the library selection can be found in the Appendix.

Detailed instruction of the steps to setup the python environment and the libraries used can be found in the last sections of the Appendix (and in the readme file).

Creation of a Baseline



After the data was procured and cleaned, and the python environment created, we first set about creating a baseline. A tricky prospect in any topic mining algorithm is selecting the number of topics. The Gensim library has logging that allowed us to take a reasonable guess at a preliminary topic number. We created baselines with 10, 15, 20, 25, and 30 topics. Using the logging module, we captured the coherence of each model. While the paper suggested that 30 topics was an appropriate number (section 5.2.3), the results of the coherence logging gave us a hint that 20 topics might also be a good number to analyze. (If interested, the logging code is in notebook: *coherence_create_helper*. The logging was also used to tune the number of passes and the number of iterations to show that the model would converge with our extreme settings for decay.)

We then set about re-creating the algorithm in the paper. An analysis of the “classical” algorithm can be found in the section **Classical ITMTF Algorithm.** In addition to re-creating the algorithm, we set about creating an “improved” algorithm.

Instructions on running iterations with both the classical algorithm and the "improved" algorithm are in the comments of the main notebook: *ITMTF* (Note: the *ITMTF* notebook is the entry point to running the algorithm. Detailed comments on the parameter set up can be found at the top of the notebook)

"Improving" the algorithm

After recreating the paper's algorithm, we set about seeing if we could improve upon it.

While analyzing the paper, a sentence caught our eye. 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."*

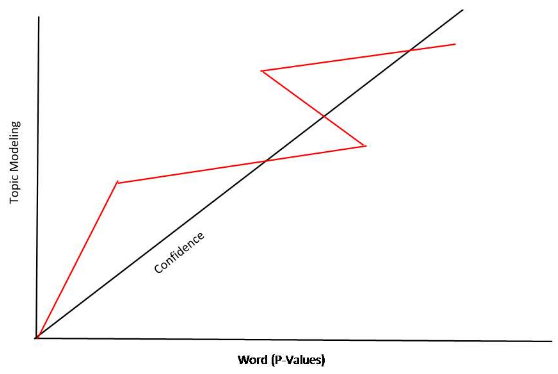
The documents are stationary, thus the word series would be static over time. The word streams, along with the Granger and Pearson statistics could be pre-processed. Our data mining had already collected the words per document, and the documents per time slice. It was not difficult to create word stream and pre-process all of the Pearson and Granger stats. (Please refer to the jupyter notebook *itmtf_prerun_stats* to see the python code used to pre-process the Granger and Pearson statistics.) (Please refer to the appendix for all of the libraries used in this project).

In our "classic" algorithm, after we run the granger test on the topic coverage stream, we added one step. We multiplied the topic/word probability from the model with the p-value that we had pre-processed for the word streams. We normalized this new number. The hope was that the algorithm would "nudge" the model into selecting words with higher statistical relevance to the betting time series and not just the words the model had selected as the top words.

This change did improve the algorithm, but we wished for something "bigger". (The final implementation of the classic algorithm in the project directory does contain this change.)

The goal of the algorithm is to iteratively improve our confidence in the topics selected by the algorithm. The topic modeling algorithm will refine topic coherence, and the ITMTF algorithm would then use word level analysis as a prior to nudge the topic modeling software toward more significant word choice for the topic.

A large part of the ITMTF algorithm is the splitting of significant topics with "positive" words placed in one topic, and "negative" words placed in another.



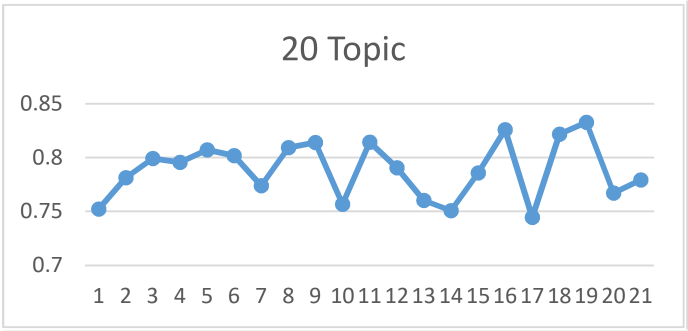
We decide to try a different (and simpler) approach. One of the benefits of topic modeling is words can have different impact in different contexts. For example, the word "rights" can have one impact on the betting sequence data if used in the context discussing "second amendment rights", and a different impact in another context discussing "civil rights". The classical algorithm would separate out the word "rights" in one of these topics.

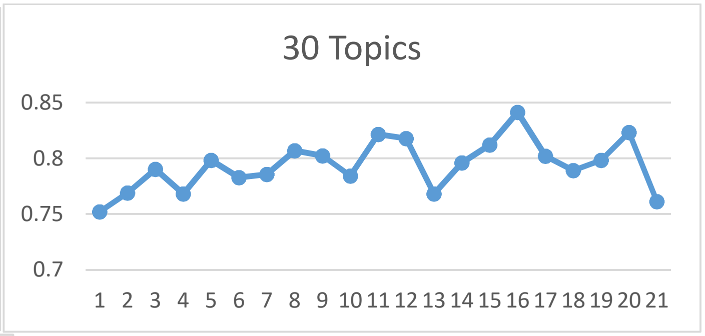
We created a simplified algorithm where the topics created by the model were left in place, only at each iteration we would analysis on every word (based upon the pre-processed word stream statistics).

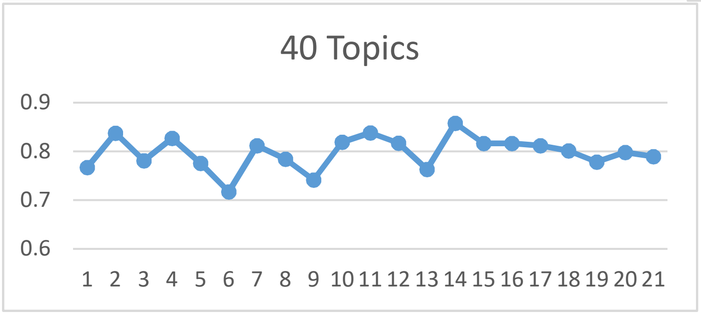
How the new algorithm works:

1. Run the model
2. Run a Granger analysis on the topic coverage across the time series (same as the classical algorithm)
    a. note this analysis is not used to refine the model, but is used to gather the confidence score for the run
    b. keep the current state if the confidence score of this new run is higher than the previous high confidence score
3. For each topic, multiply each word probability by the pre-process p-value of that word from the word stream analysis, and normalize
4. Use the new topic/word probabilities as a prior into the next round

We ran the algorithm using 20, 30, and 40 topics. Results are shown below:

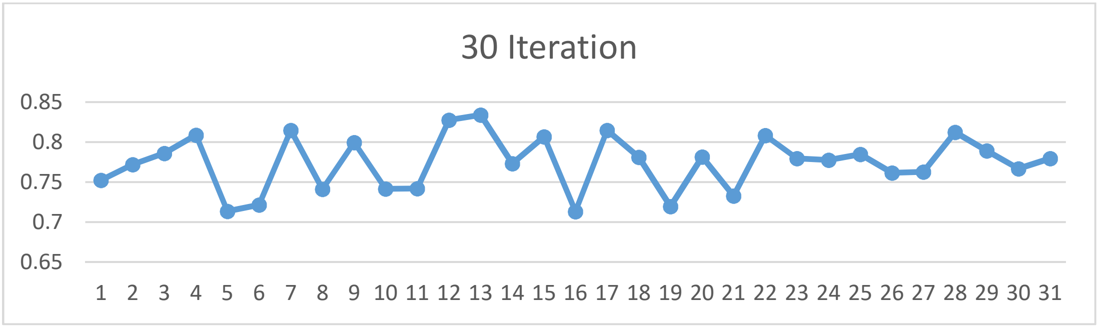






The runs resulted in remarkably similar scores. All 3 results seemed to hit a peak at the 11 or 12 iteration mark, and then all 3 models recovered to hit an ultimate peak at the14-16 iteration mark.

We also ran the 30 topic model over 30 iterations, to identify if there were further peaks – there were not. Results shown below:



The next step was to analyze the word output to see what was happening to the desired result. We captured the top words for significant topics at the 11-13 peak, and the peak that occurs after 16. The top words are shown below:

0 G+: gore	B+: bush	G+: court	G+: president	G+: campaign	G+: abortion	G+: george	G+: al	B+: gores	G+: justices
3 B+: bush	G+: george	B+: police	G+: school	G+: governor	G+: family	B+: glenn	B+: crime	G+: avenue	G+: black
5 B+: bush	G+: gore	G+: education	B+: tax	B+: bushs	G+: social	G+: security	G+: president	B+: federal	G+: george
8 B+: bush	G+: gore	G+: campaign	B+: bushs	G+: president	G+: texas	G+: george	G+: governor	G+: people	B+: vice
12 B+: vidal	G+: gore	B+: bushnell	B+: author	G+: love	G+: oct	B+: city	B+: theater	B+: sex	B+: page
21 G+: gore	G+: clinton	G+: president	B+: gores	G+: al	G+: convention	G+: campaign	G+: speech	B+: vice	B+: bush
22 B+: bush	G+: george	G+: republican	G+: gov	G+: national	G+: abortion	G+: president	G+: prochoice	G+: forces	B+: antiabortion

11 iteration peak

4 G+: gore	G+: health	B+: bush	G+: president	B+: vice	G+: governor	B+: care	G+: children	B+: lehrer	B+: companies
7 B+: industry	G+: aug	G+: entertainme	G+: re	B+: gores	B+: city	G+: oped	G+: brooklyn	G+: gorey	G+: bartlett
16 B+: bush	B+: bushs	G+: texas	G+: gore	G+: governor	G+: george	G+: president	B+: visited	G+: campaign	G+: hours
20 G+: president	G+: gore	G+: abortion	B+: bush	G+: court	G+: rights	B+: vice	G+: clinton	G+: al	G+: george
22 B+: article	B+: page	B+: bush	G+: front	G+: george	G+: oct	G+: al	G+: policy	B+: yesterday	G+: misstated

16 iteration peak

It became clear that after the first peak centered around 11th iterations, the model began to over fit the data with only words that were highly correlated to the betting data, such as “gore” and “bush”. The conclusion was for this set of data, the peak that occurs around 11 iterations produces the best data.

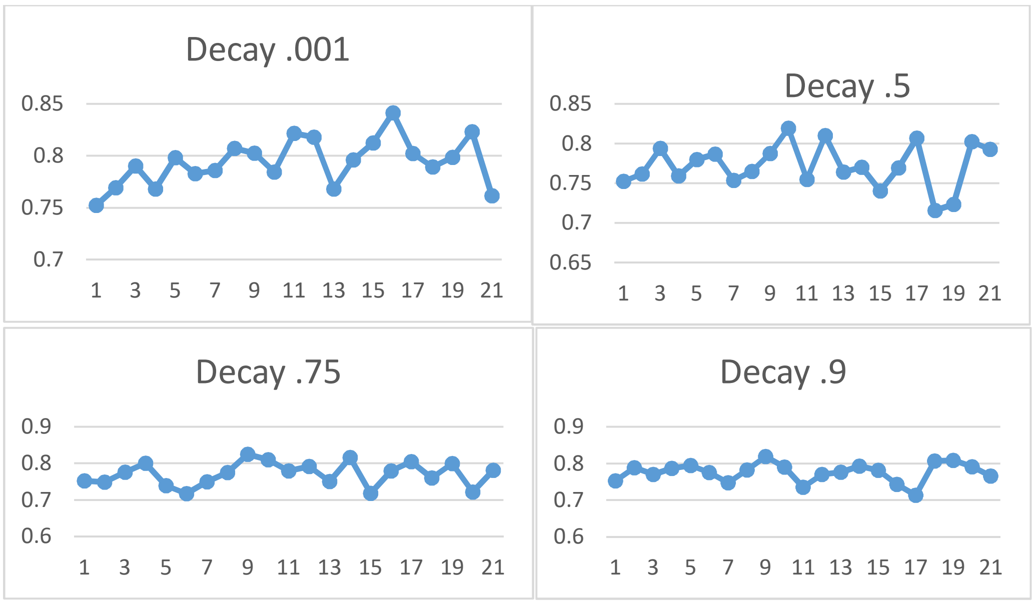
Next we turned our attention to the number of topics. We conducted 13 iteration runs with 40, 30, and 20 topics. The produced word data is shown below:

40										
0	G+: gore	G+: russia	B+: agreement	G+: arms	B+: chernomyrdin	B+: congress	B+: weapons	B+: russian	G+: minister	G+: law
3	G+: gore	B+: industry	B+: bush	G+: entertainment	B+: war	B+: gores	G+: day	B+: pine	G+: coffees	B+: vidal
7	G+: gun	B+: bush	B+: control	G+: gore	B+: association	G+: bill	G+: texas	G+: law	G+: rifle	G+: guns
8	G+: gore	B+: bush	G+: nader	G+: campaign	G+: president	G+: al	B+: vice	B+: gores	G+: george	G+: vote
10	B+: bush	B+: debates	G+: campaign	G+: gore	G+: debate	G+: commission	G+: presidential	B+: bushs	G+: president	B+: officials
19	B+: bush	G+: texas	G+: governor	B+: bushs	G+: governors	G+: george	G+: gore	G+: hispanic	G+: campaign	G+: president
22	G+: administration	B+: bush	G+: gore	G+: gorey	B+: companies	G+: issues	G+: top	B+: telecommunica	G+: al	B+: federal
24	G+: security	G+: social	G+: gore	B+: bush	B+: plan	G+: campaign	B+: gores	B+: bushs	G+: money	G+: president
26	G+: gore	G+: al	B+: gores	G+: convention	G+: president	G+: campaign	G+: vietnam	G+: democratic	G+: speech	G+: family
28	G+: debate	B+: bush	G+: gore	G+: president	G+: george	B+: vice	G+: al	B+: lehrer	G+: people	G+: texas
29	B+: bush	B+: bushs	G+: george	G+: friends	G+: texas	G+: campaign	B+: father	G+: family	G+: people	B+: time
30	B+: article	G+: bushwick	B+: yesterday	B+: am	G+: misstated	G+: street	G+: brooklyn	G+: york	G+: name	B+: copies
31	G+: abortion	G+: rights	G+: president	B+: bush	G+: george	G+: support	G+: republican	B+: decision	G+: platform	B+: nominee
33	B+: bush	G+: gore	G+: president	G+: campaign	G+: al	B+: vice	G+: george	G+: clinton	B+: gores	G+: house
36	B+: lazio	G+: clinton	G+: george	B+: donors	B+: senate	G+: republican	G+: political	G+: campaign	G+: presidential	G+: daley
30										
1	B+: lazio	B+: bush	G+: george	G+: york	G+: clinton	B+: lazios	G+: hillary	G+: gore	G+: al	G+: campaign
2	B+: bush	G+: court	G+: governor	G+: george	G+: gore	G+: death	G+: president	G+: texas	G+: troops	G+: penalty
4	B+: drug	B+: medicare	B+: plan	B+: prescription	G+: health	B+: bush	B+: coverage	B+: insurance	B+: elderly	B+: companies
10	G+: ms	B+: women	G+: im	G+: husband	G+: book	G+: life	G+: tipper	G+: schiff	G+: wife	B+: bushnell
13	B+: bush	B+: commercial	G+: campaign	G+: ad	B+: advertisement	G+: vietnam	B+: screen	B+: military	B+: word	B+: rats
20	G+: gore	B+: bush	B+: tax	B+: plan	B+: bushs	G+: security	G+: social	B+: gores	G+: president	B+: cut
25	B+: bush	G+: gore	G+: george	G+: al	B+: letterman	G+: debate	B+: jokes	B+: comedy	G+: president	G+: hes
26	G+: black	G+: gore	G+: al	G+: president	B+: article	B+: bush	G+: george	B+: play	B+: vice	B+: copies
29	B+: bush	B+: baseball	B+: rangers	G+: team	B+: owners	B+: war	G+: george	B+: owner	B+: arlington	G+: stadium
20										
6	G+: bushwick	G+: avenue	B+: police	G+: york	G+: street	B+: vidal	G+: brooklyn	B+: theater	B+: university	G+: gorey
12	G+: gore	B+: bush	G+: percent	B+: gores	G+: debate	G+: voters	G+: president	B+: poll	B+: vice	G+: people
14	G+: gore	G+: campaign	B+: bush	B+: fundraising	G+: house	G+: president	B+: million	G+: white	G+: democratic	B+: vice
16	B+: bush	G+: texas	G+: gore	**G+: abortion**	G+: george	B+: issue	G+: governor	G+: president	G+: gun	G+: gov

13 iteration runs with 20, 30, 40 topics.

For this set of data, 30 topics produced the best data. With 30 topics relevant data was quite dense.

Next we looked at the effect of decay. We ran 30 topic runs for 13 iterations with decay settings at .001 (strong effect from priors), .5 (the default for the Gensim library), .75 (the default for several other LDA libraries), and .9 (very low effect from priors)



Decay output

What we discovered was the new algorithm produced very similar data with differing decay settings. All runs had a peak at the 9-11 iterations, then varying fluctuations thereafter. At the lowest decay (highest impact from the prior) the fluctuations leading up to the 11th iteration peak were quite even and the can in confidence was steady. As one would expect, with the highest level of decay (lowest impact from the prior) the confidence remained quite flat. The intermediate levels of decay, .5 (the Gensim default) and .75 (the default for other LDA libraries) both showed improved confidence with a peak around the 11th iteration.

The new algorithm proved to be quite effective for this set of data. Confidence steadily improved and reached a peak around the 11th iteration. After that peak, the algorithm began to over fit the data and produce topics with few, but highly significant, words.

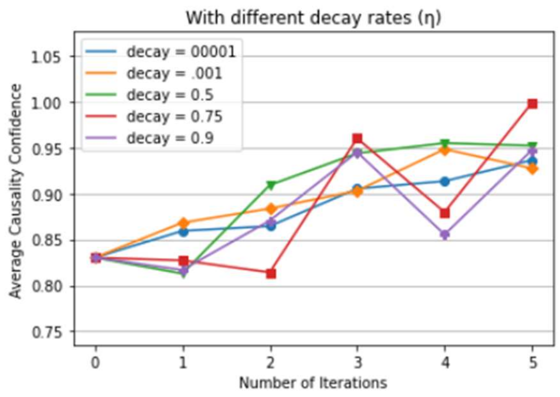
Furthermore, the new algorithm was successful without an overreliance on hyper parameter tuning. The number of topics had to be selected, but the default values for decay from the Gensim library produced results.

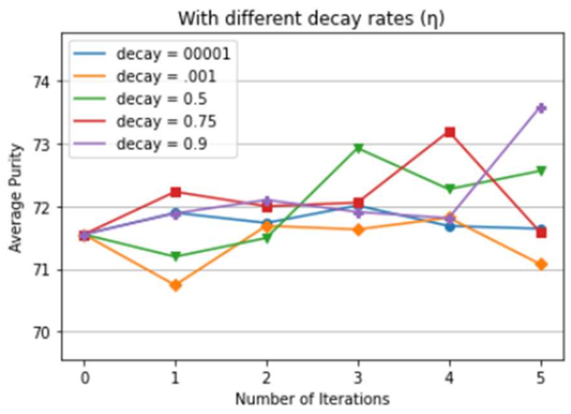
The new algorithm did require more iterations than the classic algorithm, and the word streams had to be pre-processed. However, once tuned, the Granger analysis would only need to be conducted during the peak window (9-12 iterations) looking for the maximum confidence.

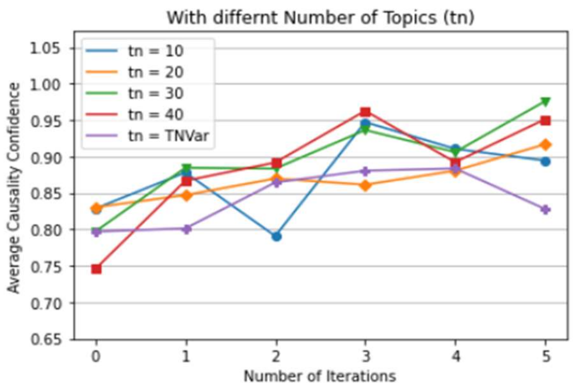
Classical ITMTF Algorithm

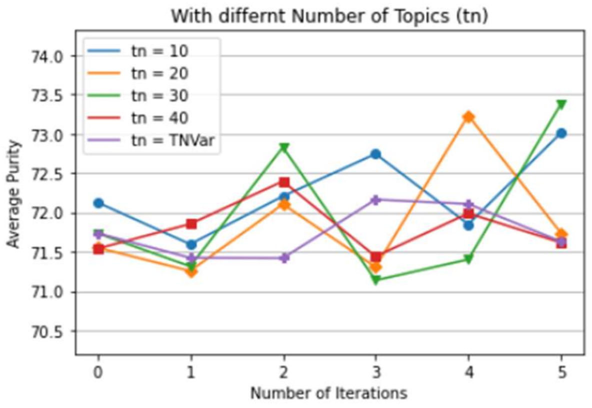
The team was able to re-crate the classical algorithm from the paper. The only real variation was the use of LDA vs PLSA. The paper used PLSA, but did state that any topic mining algorithm would work. In fact, that was one of the thrusts of the paper, a general framework. As discussed below, Gensim LDA was selected as it performed an iteration in a reasonable time (2 minutes vs over a ½ hour for the python PLSA algorithm).

Below are some of the plots produced by our implementation of the algorithm. (Please refer to the notebook Classic_Baseline_Plot)









When compared to the graphs produced in the paper, our implantation produced similar results. The average causality confidence steadily increased with both implementations, with higher prior impact showing the best results. Both implementations had somewhat flat purity.

We are unsure why the published results had such poor baseline numbers. The poor baseline numbers in the published word made the first iteration appear like a substantial jump. Our algorithm did show a large improvement for the first iteration (especially with the larger prior influence, as in the published paper), just not as dramatic.

Another variable would be the data cleaning. One of the topics published has “pres” “al” “vice” as its top 3 words. This topic clearly is about Vice President Al Gore. It seems the words “Gore” and “Bush” were removed as part of the data cleaning for the paper. This would have a large impact on the model, as the p-values for the words “gore” and “bush” were the largest of all words.

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

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 – Libraries used

Gensim Python LDA - https://radimrehurek.com/gensim/models/ldamodel.html

SciPy's pearson r - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html

statsmodels granger causality tests - https://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.grangercausalitytests.html

pyLDAvis - https://pyldavis.readthedocs.io/en/latest/index.html

glob - https://docs.python.org/3/library/glob.html

Matplotlib - https://matplotlib.org/

Library tested but not used:

pypi.orgs PLSA - https://pypi.org/project/plsa/

Appendix – Environment setup

For Windows:
Open an anaconda prompt, navigate to the project's directory and type:
conda env create -f ITMTF.yml

The created environment will be called "Gensim", when you open the notebook, you will have to change kernels to Gensim. See troubleshooting note below.

Adding Gensim LDA library to an Anaconda environment manually:
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

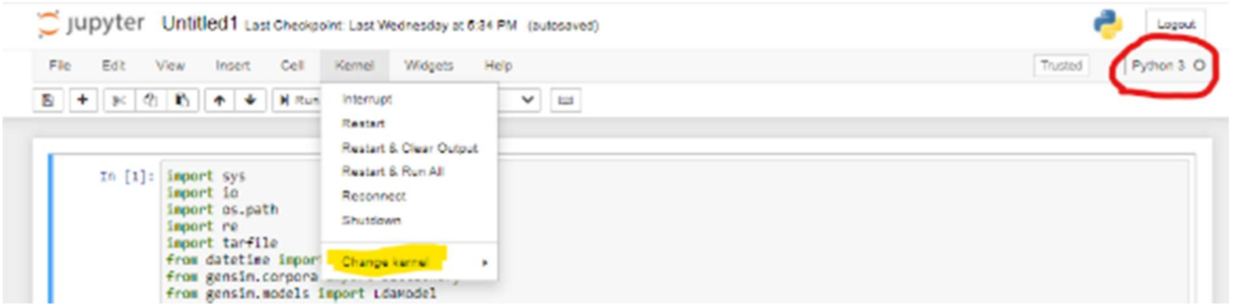
1. Open the Anaconda command prompt
2. If you created a new environment in the previous step:
    a. Activate the newly created environment if you created one ("Activate gensim")
    b. Run: conda install nb_conda_kernels (Proceed Y)
    c. Run: python -m ipykernel install --user --name myenv --display-name "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 genism
4. Run: pip install –upgrade pyldavis
5. Run: pip install –upgrade glob2
6. Run: pip install –upgrade matplotlib

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")

TROUBLESHOOTING 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 impor
        from gensim.corpora
        from gensim.models import LdaModel
```