Pei Lun Hung
CS 410
Professor ChengXiang Zhai
December 13th, 2020

Documentation

1. Code functionality overview

This code parses a large data set[1] and extracts a set of documents consisting of paragraphs related to the 2000 United States presidential election, contested between Republican George W. Bush and Democrat Al Gore. It then goes on to apply latent Dirichlet allocation (LDA), a generative statistical model, to this document set, thus identifying candidate topics in the set. Normalized prices based on the Democratic candidate, Al Gore, are then extracted from the Iowa Electronic Markets (IEM) 2000 Presidential Winner-Takes-All Market[2], and Granger testing is performed to determine possible causality of the candidate topics, indicating which topical key words are most likely to impact the candidate's IEM price positively or negatively—that is, their likelihood of being elected. This can be taken as a prior for future iterations of causal topic mining.

In terms of future applications, the code can be modified to identify target paragraphs of any nature from *The New York Times* corpus. Various external time series data can also be used in conjunction—physical data comes to mind as a useful application. Or, public opinion polling on issues like climate change or social problems could be coupled with data from *The New York Times* corpus in order to find any sort of causal relationship between media coverage and public opinion surrounding those issues.

[1] Namely, *The New York Times* corpus, from May to October 2000

[2] A prediction market allowing traders to buy and sell contracts based on that year's presidential election.

2. Software implementation

Note: *mining_causal_topics.ipynb* combines code and results from *identify_candidate_topics.py*, *iowa_electronic_markets.py*, and *granger_testing.py* into one coherent file; this notebook is the easiest place to see code and results past the election paragraph parsing stage, though documentation has been organized based on the subfiles for ease.

find_election_paragraphs.py: filters May through October 2000 *New York Times* articles for election-related paragraphs

- *find_election_paragraphs(directory)*: given a directory containing XML article data, walks through all subdirectories and parses all files. Writes all election-related paragraphs (those containing “Gore” or “Bush”) to a file called “election_paragraphs.csv”
    - Helper functions:
        - *parse_xml(xml, election_paragraphs)*: checks XML article for person tag, then parses relevant articles’ full text for relevant paragraphs
        - *contains_election_words(text)*: returns true if and only if “Bush” or “Gore” is a substring of the input text
        - *format_date(file_path)*: returns formatted date from directory path

identify_candidate_topics.py: applies LDA to the document set to identify candidate topics

- Data wrangling
    - Wrangle and clean paragraph data by removing punctuation and lowercasing text
    - Remove stop words from the paragraph data and vectorize the text using scikit-learn, then learn the vocabulary dictionary, deriving a document-term matrix over the paragraph set
- Topic modeling with LDA
    - Using LDA, learn the documents as bags of words and identify candidate topics in the paragraph set
    - Transform the document-term matrix according to the fitted LDA model, and index on date (dates range from May 1st, 2000 to October 31st, 2000)

iowa_electronic_markets.py: wrangles, cleans, and normalizes IEM 2000 Presidential Winner-Takes-All Market data

- Data wrangling
    - *format_date(date)*: given IEM-style date (mm/dd/yy), returns *NYT*-style date (yyyy-mm-dd)
    - Read from “iem_2000.txt”—candidate price data from May to October 2000—and drop irrelevant columns (namely: units, volume, low price, high price, and average price), keeping only date, contract, and last price data; index on date
    - Derive lists of Democratic and Republican candidate prices based on the “contract” field (“Dem” indicates Democrat; “Rep” indicates Republican)
    - Compute normalized prices based on the Democratic candidate
        - For each date index, normalized price = Dem price / (Dem price + Rep price)
- Election coverage vs. election forecast
    - Concatenate the normalized price data to the topic coverage data from *identify_candidate_topics.py*; this brings together topic words and price data, indexed by date

granger_testing.py: perform Granger tests to determine significant causal words and their impact

- Granger testing
    - Perform Granger tests[3] to test causality with max lags up to 3
- Significant causal words
    - Compute average p-values across time lags (from 1 up to 3) for each candidate topic term, based on parameter F tests

[3] Granger tests compute lagged correlation measures to find potential causal relationships between time series—here, they look for potential causal relationships between *New York Times* coverage and candidate electability

  - Sort candidate topic terms by causality probability, where a smaller p-value corresponds to a larger probability that there exists some type of causal relationship between the two time-indexed data
- Prior influence
  - Determine positive impact terms and negative impact terms, where positive impact terms are the ones with p-values in the lower half, and negative impact terms are the ones with p-values in the upper half

3. Usage

- Install Jupyter using the documentation provided here
- Open terminal and run the following command:
  - git clone https://github.com/kplhung/CourseProject.git
- Launch Jupyter Notebook and navigate to the CourseProject directory
- Open and run *find_election_paragraphs.ipynb*
  - [Kernel] → [Restart & Run All]
- Open and run *mining_causal_topics.ipynb*
  - [Kernel] → [Restart and Run All]

4. Team member contributions

This was an individual project.