Write-Up Robot Judge

Huber Simon
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—**

I. Introduction

Automatic analysis of text written or said by politicians is useful in many ways. And they come in many forms - spoken on the senator floor, tweeted on twitter and printed in papers. Topic models are one method to analyse such text. It is designed to group documents with similar content. A lot of this techniques are unsupervised for multiple reasons:

1) There is just not enough data that is already labeled.
2) The target data set is very small or very specific.

By interpreting hashtags as labeled data for topics we can expand the labeled data immediately. There is already enough data labeled for a very wide range of topics on twitter. With 500 million tweets per day [1] and many of them tagged with different topics. From a sample of 1 million random twitter tweets 15.32% had at least 1 tag. The idea then was simple, learn a topic model through hashtags and apply it to other domains. In this work we were interested in applying it to speeches of politicians. In particular we thought about the following ideas: How does the topics of a speech for a senator change if he is up for reelection ? How are the topics distributed for a democrat or for a republican senator ? Can we use hash tags to predict the party ? And lastly how do the topics compare between speech and twitter for a specific politician and to the general trendy topics ?
The paper continuous in the following manor: In Section II I describe the used datasets. In Section III I discuss the two models we used to predict the hash tags and why one failed. Then in Section IV I describe the planned and executed experiments as well as their result. In Section VI I discuss the results and finally in Section VII I summarize and conclude the paper.

- General Idea - Hypothesis - Paper Structure

II. Data

In this section we present the used datasets, how they were obtained and the overall statistics.

A. Twitter Data

Twitter has a rule that one is not allowed to save tweets along with text and make it then publicly available as a dataset. This is because if one wants to make any changes to the tweets e.g. delete them etc. this is no longer globally possible if they are saved somewhere else online. So one is only allowed to save them via an unique ID that can be used to retrieve the tweets with the twitter api. A website that makes many important datasets available in this manner is found in [2]. I downloaded two datasets.

1) Congress Dataset: The 115th U.S. Congress Tweet Ids dataset contains 2,041,399 tweets. It covers a date distance from 2017-01-27 to 2019-01-02. It therefore covers 1 year before the senate elections and one year after the senate elections which makes it definitely very interesting for us.

2) News Dataset: The News Outlet Tweet Ids we also downloaded was a fallback dataset which in case the congress dataset was to ”fuzzy” could be used to train on more well written, structured and diverse topic distributed dataset. Which was of course just an hopeful assumption.

B. Speeches Data

The congressional records and therefore the speeches can all be publicly accessed on the congress page [3]. But this side makes it very hard to access specific information. So people developed software packages that parse this side for the information and presents it in an accessible format mostly json. I first tried a package from another student but due to quite a few bugs I had to switch to [4] which worked very nicely. I used it to parse all congress information from the years 2017 and 2018.

III. Plan

In the first part of this section I will tell about the failure and massive time loss that resulted in trying to adapt LDA to be used with the twitter data and in the second part a summary of the neural network based method that finally worked for predicting hash tags.

A. Latent Dirichlet Allocation (LDA)

1) Standard LDA: LDA was one of the algorithms for topic models that got presented in class. The generative model to generate a word makes it very simple to think about LDA. Given a document sample a topic and for this topic sample a word. And the word is the thing we observe and we can use to learn the corresponding distributions.
More formally the process depicted in Figure 1 is defined as:

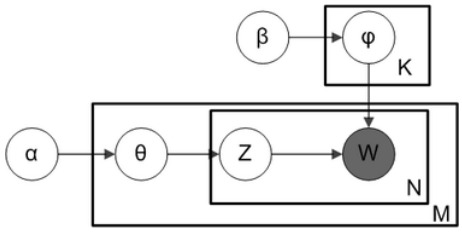

Figure 1. Plate notation for LDA with Dirichlet distributed topic-word distributions.

Given α, β
Sample topic-distribution: $\theta_m \sim Dir(\alpha)$
Sample word-in-topic-distribution: $\phi_k \sim Dir(\beta)$
Sample word-topic from topic-distribution: $z_{mn} \sim Cat(\theta_m)$
Sample word from word-in-topic-distribution: $w_{mn} \sim Cat(\phi[z_{mn}])$

Where $\alpha \in \mathbb{R}^K, \beta \in \mathbb{R}^{|V|}$ are the parameters for the dirichlet distribution.

2) Extended LDA: So naturally I thought it should be simple to extend LDA so that it can also predict hash tags in the following way. For some document and its document-topic distribution I sample 2 topics and then for the topic-word distribution I sample a word and additionally to the topic-word distribution I just have an topic-hashtag distribution of which I sample a hashtag. And the we can learn the correct distributions since we observe both word and hashtag. **More formally** we have only 3 additions (inserted in blue) to the process as defined in normal lda:

Given α, β, γ
Sample topic-distribution: $\theta_m \sim Dir(\alpha)$
Sample word-in-topic-distribution: $\phi_k \sim Dir(\beta)$
Sample hashtag-in-topic-distribution: $\kappa_k \sim Dir(\gamma)$
Sample word-topic from topic-distribution: $z_{mn}^{(w)} \sim Cat(\theta_m)$
Sample word from word-in-topic-distribution: $w_{mn} \sim Cat(\phi[z_{mn}])$
Hashtag-topic from topic-distribution: $z_{mt}^{(t)} \sim Cat(\theta_m)$
Hashtag from hashtag-in-topic-distribution: $t_{mn} \sim Cat(\kappa[z_{mn}])$

Where $\alpha \in \mathbb{R}^K, \beta \in \mathbb{R}^{|V|}, \gamma \in \mathbb{R}^K$ are the parameters for the dirichlet distribution. The plate in Figure 2 depicts the whole process graphically.

B. Training

Now the generative model is only the first step because we're not interested in generating words. We're interested in the so called posteriori distribution: Given words what are the topic-distribution of a document. Then from that generate hashtags by sampling a topic and sampling a hashtag for the document. This means that we have a so called inference

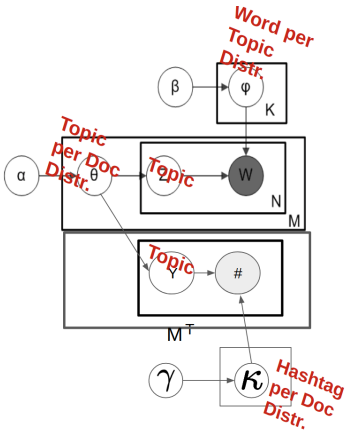

Figure 2. Plate diagram of extended version of LDA with additional distribution for hashtags for a given topic.

problem and normally one applies bayes formula to get the topic distribution:

$$p(\theta_m|w) = \frac{p(w|\theta_m)p(\theta_m)}{p(w)}$$

But the so called evidence term

$$p(w) = \int_{\theta_m} p(w|\theta_m)p(\theta_m)d\theta_m^K$$

is very hard to compute because every new possible topic increases computing time exponentially so this is nearly impossible to compute already for very little amount of topics. Researchers have proposed different methods to approximate this posteriori distribution with other methods and there are a lot of software frameworks that automatically calculate the posterior distirbution. I decided to use pyro [5] which allows to describe the generative model in a very pytonian way and uses pyTorch [6] as backend for matrix and optimization calculations which makes it ideal for big datasets. See the Code Section VIII for an overview of what is implemented in which files. The optimization process is either via a monte carlo method (MCMC) or via Variational Inference (VI). Bot methods are explained in the next two subsections very briefly.

1) Variational Inference: The big idea in Variational Inference (VI) is to propose a proposal distribution $q_\lambda(\theta_m)$ which gets adapted via parameter λ to be as close as possible to the real distribution we care about $p(\theta_m|w)$. Closeness is measured via kullback-leibler divergence but this can't be calculated directly most of the time so alternatively one maximizes the so called ELBO which minimizes the kullback-leibler divergence.

$$ELBO = \mathbb{E}_{q_\lambda(\theta_m)}[logp(\theta_m, w) - logq_\lambda(\theta_m)]$$

Then to maximize the ELBO in most software packages one calculates the gradient and applies one of the nowaways many optimization algorithms. (Adam, RMSprop, SGD,

etc.)
Since I used pyro to implement this, in pyro q_λ is described in the so called "guide" function and the joint distribution $p(\theta_m, w)$ is described in the so called "model".

2) Markov Chain Monte Carlo: Markov Chain Monte Carlo (MCMC) algorithms are a bit more involved in theory and there are a lot more flavors then in variational inference. But the underlying idea is the same for them aswell. MCMC algorithms propose a markov chain from which they draw correlated samples when moving from state to state in this chain but which promises to converge to the correct distribution. Now one may ask why we do not use this all the time when it converges to the correct distribution ? Sadly such a chain has a so called "warm up" stage where we have to throw away samples up to the point where the distribution is burned it and we do not know how long this warm up phase is. Therefore normally one uses empirically defined rules and tests to estimate the length of the warm-up phase. A second problem is that complicated distributions get broken up into multiple conditional distributions and it can get slow to sample such distributions sequentially.

3) Training Result: But it just didn't work. I tried a multitude of different proposal distributions and even neural networks. It just didn't really converge nicely. I initially thought that the problem was my extension on top of LDA. But it didn't work for LDA aswell.

C. Tweet2vec - Recurrent Neural Network

After the extended LDA somewhat failed I read multiple papers and tested a multitude of other models. The one that finally worked was a neural network based solution: Tweet2Vec [7]. The basic idea behind this model is that they view the character as smallest unit rather then the word as in LDA. This brings quite a few advantage. Most importantly misspelled words, slangs and slurs can be encoded with the same embedding meaning much easier. And we do not have to save all the words which due to the many abbreviations and slangs on twitter can blow up the vocabulary very much. In the congress dataset we have 11'498'020 number of different tokens. Furthermore they assume that a tweet has some latent representation and from this latent representation they predict the hashtag. To get a loss they use cross entropy loss based on predicted hashtag versus given hashtag. The neural network is mainly built out of Bi-directional Gated Recurrent Unit (Bi-GRU) which behave similarly as LSTM's (Long short-term memory) blocks. The size of the input is reasonable constraint on the number of characters a tweet can have - 145. The embedding part of the tweet2vec neural network can be seen in Figure 3. The decoder part is just a dense layer with the number of neurons equal to the number of hashtags. The output of this dense layer is then used as input for the softmax function to get the probability of each hashtag.

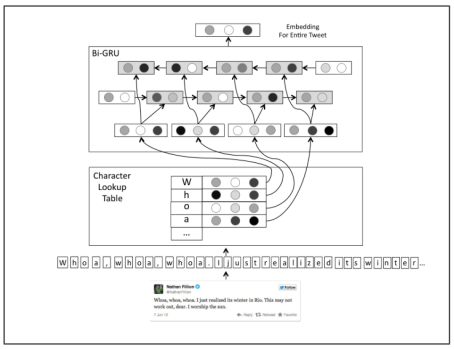

Figure 3. Tweet2vec embedding part of the neural network.

Name	Party	Num. Speeches
Feinstein	D	2
Stabenow	D	1
Klobuchar	D	1
Menendez	D	1
Cantwell	D	1
Warren	D	1
Gillibrand	D	1
Tester	D	1
Manchin	D	1
Murphy	D	5
Baldwin	D	1
Hirono	D	1
Carper	D	1
Sanders	I	1
Kaine	D	1
Brown	D	3
Heinrich	D	1
Whitehouse	D	1
Casey	D	1
Wicker	R	0
Barrasso	R	2
Fischer	R	1
Cruz	R	2
King	I	3

Table I
LIST OF SENATORS UP FOR REELECTION IN 2018. (D = DEMOCRAT, R = REPUBLICAN, I = INDEPENDENT)

IV. EXPERIMENTS & RESULTS

First lets reiterate more concrete about what we want to figure out and then subsequently how we design the experiments to test our hypothesis. We start with a simple one: Can we tag speeches of politicians with hashtags learned from twitter ? Then we go on to figure out how tags compare for someone giving a speech before and after campaigning for reelection.

A. Can we tag speeches of politicians with hashtags learned from twitter ?

1) Data: The politician set consisted of 449334 tweets which we splitted randomly into 90% training data and 10%

testing data. I counted all the hashtags and removed all that were not in the top 500 used hashtags (from 39671 total).

2) Results: As explained in previous sections I used tweet2vec. In Figure 9 one can see that we got nearly 25% recall for tag number 1 which is quite good considering how difficult the task is (Randomly predicting one of the top 500 would result in 0.2% recall). For the speeches I was only interested in the senators up for re-election. It is a set of 24 senators listed in Table I with their corresponding party and the number of speeches they gave in the years 2017 and 2018.

Sadly this dataset of speeches is very small. But it does not hinder us to tag the available speeches. To tweet2vec to be able to work on the speeches I had to cut it up into pieces of 145 and additionally I did some preprocessing like stemming, removing stop words and removing line breaks just to get a bit shorter speeches and get more content into those 145 characters.

In the boxes one can see the input for the trained tweet2vec in the top part and the tags tweet2vec predicted in the bottom part.

Senator Sanders:
support vote pension issue million american worker percent pension promise government allow happen stand worker meeting earlier today dealing
Tags: Kavanaugh WhatAreTheyHiding NM UK IIOA

Senator Cruz:
president authority refuse enforce president obama decree would enforce federal immigration exactly virtually every republican denounce executiv
Tags: ACA healthcare NM ClimateAction ProtectOurCare

From just checking the speeches vs the tags by hand it does not seem to be very effective in predicting tags for speeches. I really tried to also find a positive example but I just couldn't really find a satisfying one.

B. How does being before an election affect the tags a politican uses ?

After wasting a lot of time on the tagging speeches part I concentrated the little rest of my time on only analyzing the raw twitter data. And one can say that for some senators definitely tweets differently and is affected when having an election. I picked a democratic senator and a republican senator to showcase how tweets change. Local topic get very important for senators and most tweets will get tagged with the state their from. In figure 4 its the case for the 2012 election where Senator Hirono wants to get elected for Hawaii. In 5 one can see the figure of number of tweets at a specific

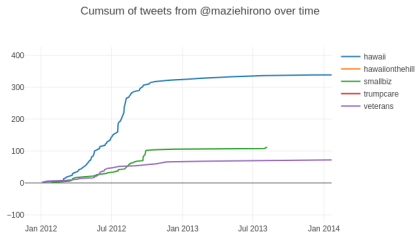

Figure 4. Cumulative sum of top 5 topics of senator Hirono from Hawaii.

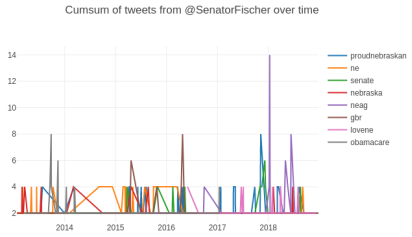

Figure 5. Senator Fischer from Nebraska number of tweets from 2012 - 2019 counted at specific points in time. Only tweets counted with tags of top 8 tags. #ne = nebraska, #neag = nebraska agrar, #lovene = love nebraska.

date from Senator Fisher of the republicans. Shortly before the election Nebraska specific tweets definitely spiked. But it is definitely not en if and only if indicator. There are a lot of Senators who don't care about tweeting for local problems e.g. Senator Warren and Senator Sanders. See Figure 7 for the distribution of tags for Senator Sanders and 8 for the distribution of tags over time for Senator Warren. In both it is obvious that they have more global priorities then their own states. The plots for all the other Senators can be checked in my github repository see Section VIII. But not only could it be that the tweet behaviour does not change but that a Senator could start tweeting about local events with no local election coming up. An example for this is Senator Cruz who had a hard election in Texas. But in the year of the election there was also a hurrican as strong as all 10 years and he tweeted a lot about that in that same time frame which is also the time of the election seen in Figure 6.

C. Hashtag analysis

D. Tweet2vec

E. Tweet2Vec

V. Negative Results

VI. Discussion

A. Data discussion

Todo: - just very few speeches data. Todo: especially discuss twitter data again.

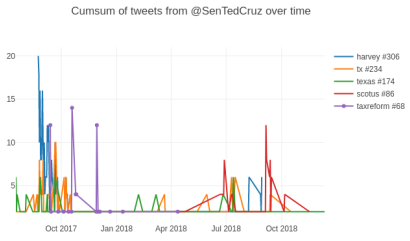

Figure 6. Senator Cruz from Texas top 5 used tags tweets distribution from mid 2017 to end 2018.

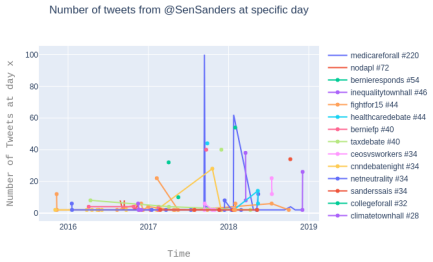

Figure 7. Number of tweets of senator Sanders at a specific day.

B. Model discussion

Todo: Neural network vs Other tools

C. Lessons learned

I want to write down a few words about the lessons I've learned and reflect a bit so that I may have learned something beside the technical.

1) The biggest point for me is apply the right tool to the right problem. Since I've read and heard the first time about probabilistic programming languages I searched for a way to apply it somewhere during my studies. I'm more on the graphics side in my computer science studies which made it difficult to apply it anywhere there. So along came this topic model problem and I tried everything to apply it to this problem. I think this gave me kind of a tunnel vision which lead to ignoring maybe more applicable paths.
2) Twitter data is hard. This is definitely something I had not considered before I took into consideration to work on this data. Due to the length restrictions on a tweet many properties that emerged from twitter are not simple to generalize to other texts.
3) When working with big data probably nothing comes close to deep learning currently. I always had the urge to not use the tools that everyone else is using which punished me hard on this task because all the state of the art in this kind of area (text / speech / vision) is based on neural networks.

So really in summary of this lessons: I should just be more conscious about my own biases I bring to a project and should probably ask earlier if I'm still on the right path.

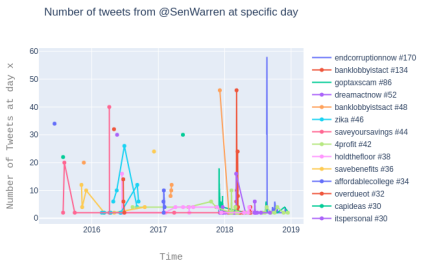

Figure 8. Number of tweets of senator Warren at a specific day.

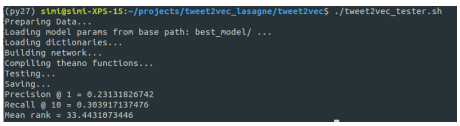

Figure 9. Results for testing on validation set. (10% of original dataset).

VII. Summary

VIII. Code

In this section I describe the code and structure of the github repository for reproducability as well as proof of my work. I will use: *base_url = https://github.com/SimiPro/robotjudge/blob/master/* to shorten the urls. If you don't want to download the whole repository but only want to look at the jupyter notebook I can recommend the website *https://nbviewer.jupyter.org/* where one can insert the link to the github notebook and it uses a much more powerful parser then github and especially can display a lot of plots that github can't.

A. Data parsing , downloading

I downloaded the datasets as mentioned in the data section. The data is unpacked in *base_url/congress_dataset*. The translator from tweet-id to csv is written in *base_url/trasnform_id_to_tweet_csv.ipynb*. For the news dataset there is equivalently the transformation code present but because the id dataset alone was already to big for github only the code and some meta data is there.

B. LDA

1) News Dataset and Standard Sklearn LDA: After some failed attempts on twitter data I checked if sklearn's LDA works on twitter data as well as on the sklean news dataset [8]. Later I also integrated some pyro tests on the sklearn news dataset in this file. Those experiments were conducted in *baseurl/lda_sklearn.ipynb*

2) Final Pyro Model on Twitter Data: The final results and experiments for LDA with pyro are safed in *baseurl/pyro_lda_final.ipynb*.

C. tweet2vec

For tweet2vec I created a new github repository to get a clean slate for this new model. The github url is: tweet2vec_baseurl = *https://github.com/SimiPro/tweet2vec_lasagne*. The readme is still correct to get it running.

1) Data:: The input datafiles indicate over their name how many tweets are in the dataset. E.g. *tweet2vec_baseurl/my_training_file_10000.txt* is the training file that contains 90% of 10000 tweets and *tweet2vec_baseurl/my_testing_file_10000.txt* contains the other 10% that make up the 10000 tweets. We have 3 datasets of size: 10000, 25000, 449334 number of tweets.

2) Tagging speeches:: *tweet2vec_baseurl/tweet2vec/tweet2vec_speeches.py* runs the embedding and tagging on each speech by using the best model which is safed in *tweet2vec_baseurl/tweet2vec/best_model*. The specific tagged speeches were gathered in *tweet2vec_baseurl/results* by politician name and date of the speech so that analysis over time get possible.

D. Data Analysis / Plots

base_url/analyze_predicted_tags.ipynb creates the plots based on the predicted tags whereas *base_url/TimeLineHashTags.ipynb* creates the plots based on real tags.

E. Statistics

base_url/statistics/count_hashtags.ipynb was used to count the number of hashtags in a random sample of over 1 million tweets. *base_url/statistics/count_tokens.ipynb* was used to count the number of different tokens in the congress dataset.

References

[1] S. Aslam, "Twitter by the Numbers: Stats, Demographics," https://www.omnicoreagency.com/twitter-statistics/, 01.2019, [Online; accessed 07-July-2019].

[2] "Tweet ID Datasets," https://www.docnow.io/catalog/, 01.2019, [Online; accessed 07-July-2019].

[3] "Congress Page," congress.gov/congressional-record, 01.2019, [Online; accessed 07-July-2019].

[4] D. D. J. C. Judd, Nicholas and L. Young, "congressional-record: A parser for the Congressional ," https://github.com/unitedstates/congressional-record, 2017, [Online; accessed 07-July-2019].

[5] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[7] B. Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen, "Tweet2vec: Character-based distributed representations for social media," *CoRR*, vol. abs/1605.03481, 2016. [Online]. Available: http://arxiv.org/abs/1605.03481

[8] "Sklearn News Datasets," https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html, 01.2019, [Online; accessed 07-July-2019].