3rd International Conference on Advanced Research Methods and Analytics (CARMA2020)
Universitat Politècnica de València, València, 2020
DOI: http://dx.doi.org/10.4995/CARMA2020.2020.11557

Pruned Wasserstein Index Generation Model and wigpy Package

Fangzhou Xie
Department of Economics, New York University, USA, Department of Economics, Rutgers University, USA.

Abstract

Recent proposal of Wasserstein Index Generation model (WIG) has shown a new direction for automatically generating indices. However, it is challenging in practice to fit large datasets for two reasons. First, the Sinkhorn distance is notoriously expensive to compute and suffers from dimensionality severely. Second, it requires to compute a full $N \times N$ matrix to be fit into memory, where N is the dimension of vocabulary. When the dimensionality is too large, it is even impossible to compute at all. I hereby propose a Lasso-based shrinkage method to reduce dimensionality for the vocabulary as a pre-processing step prior to fitting the WIG model. After we get the word embedding from Word2Vec model, we could cluster these high-dimensional vectors by k-means clustering and pick most frequent tokens within each cluster to form the "base vocabulary". Non-base tokens are then regressed on the vectors of base token to get a transformation weight and we could thus represent the whole vocabulary by only the "base tokens". This variant, called pruned WIG (pWIG), will enable us to shrink vocabulary dimension at will but could still achieve high accuracy. I also provide a wigpy[1] *module in Python to carry out computation in both flavors. Application to Economic Policy Uncertainty (EPU) index is showcased as comparison with existing methods of generating time-series indices.*

Keywords: *Wasserstein Index Generation Model (WIG); Lasso Regression; Pruned Wassersteinn Index Generation (pWIG); Economic Policy Uncertainty Index (EPU).*

[1] https://github.com/mark-fangzhou-xie/wigpy

1. Introduction

Recently, the Wasserstein Index Generation model (Xie, 2020) was proposed to generate time-series sentiment indices automatically. There have been several methods (Azqueta-Gavaldón, 2017; Baker, Bloom, & Davis, 2016; Castelnuovo & Tran, 2017; Ghirelli, Pérez, & Urtasun, 2019) proposed to generate time series sentiment indices, but, to the best of my knowledge, WIG is the first automatic method to produce sentiment indices completely free of manual work.

The WIG model runs as follows. Given a set of documents, each of which is associated with a timestamp, it will first cluster them into several topics, shrink each topic to a sentiment score, then multiply weights for each document to get document sentiment, and then aggregate over each time period. However, its computation on large dataset come with two challenges: (1) the calculation for Sinkhorn algorithm suffers from its notoriously computational complexity and the computation will soon become prohibitive; (2) this Optimal Transport-based method requires to compute a full $N \times N$ matrix, where N is the size of vocabulary, and it will become impossible to fit this distance matrix into memory after some threshold. Therefore, I propose a pruned Wasserstein Index Generation model (pWIG) to reduce dimensionality of vocabulary prior to fitting into the WIG model. This variant could represent the whole corpus in a much smaller vocabulary and then be fit in any memory-limited machine for the generation of time-series index. What is more, I also provide the *wigpy*[2] package for Python that could perform both version of WIG computation.

This paper first contributes to the EPU literature and tries to provide better estimations of that seminal time-series indices automatically. This article also relates itself to the new area of Narrative Economics (Shiller, 2017), where we could extract time-series sentiment indices from textual data, and thus provide a better understanding of how do narratives and sentiments relate to our economy.

2. Pruned Wasserstein Index Generation Model

We first review the original WIG model.

2.1. Review of Wassserstein Index Generation model

A major component of WIG model is the Wasserstein Dictionary Learning (Schmitz et al., 2018). Given a set of document $Y = [y_m] \in \mathbb{R}^{N \times M}$, each doc $y_m \in \Sigma^N$ is associated with a timestamp and N, M are length of dictionary and number of documents in corpus, respectively. Our first step is to cluster documents into topics $T = [t_k] \in \mathbb{R}^{N \times K}$, where

[2] https://github.com/mark-fangzhou-xie/wigpy

$K \ll M$, and associated weights $\Lambda = [\lambda_m] \in \mathbb{R}^{K \times M}$. Thus, for a single document y_m, we could represent it as $y_m \approx t_k \lambda_m$. Documents and topics lie in N-dimensional simplex and are word distributions. Another important quantity for computing WIG, is the cost matrix $C^{N \times N}$ and $C_{ij} = d^2(x_i, x_j)$, where each $x_i \in \mathbb{R}^{1 \times D}$ is the D-dimensional word embedding vector for the i-th word in the vocabulary. In other words, matrix C measures the "cost" of moving masses of words, and now we can proceed and define the Sinkhorn Distance.

Definition 1 (Sinkhorn Distance).

Given discrete distributions $\mu, \nu \in \mathbb{R}_+^N$, and C as cost matrix,

$$S_\varepsilon(\mu, \nu; C) \coloneqq \min_{\pi \in \Pi(\mu,\nu)} \langle \pi, C \rangle + \varepsilon \mathcal{H}(\pi)$$

$$s.t. \;\; \Pi(\mu, \nu) \coloneqq \{\pi \in \mathbb{R}_+^{N \times N}, \pi 1_N = \mu, \pi^T 1_N = \nu\},$$

where $\mathcal{H}(\pi) \coloneqq \sum_{ij} \pi_{ij} \log(\pi_{ij} - 1)$, negative entropy, and ε is the Sinkhorn regularization weight.

We could then set up the loss function and minimization problem as follows:

$$\min \sum_{m=1}^{M} \mathcal{L}\left(y_m, y_{S_\varepsilon}(T(R), \lambda_m(A); C, \varepsilon)\right),$$

$$s.t. \;\; t_{nk}(R) \coloneqq \frac{e^{r_{nk}}}{\sum_{n'} e^{r_{n'k}}}, \lambda_{nk}(A) \coloneqq \frac{e^{a_{km}}}{\sum_{k'} e^{a_{k'm}}}.$$

By this formula, we wish to minimize the divergence between original document y_m and the predicted (reconstructed) $y_{S_\varepsilon}(\cdot)$ given by Sinkhorn distance. Moreover, the constaints of this minimization problem considers *Softmax* operation on each of the columns of the matrices R and A, so that T and Λ will be (column-wise) discrete densities, as is required by the Sinkhorn distance.

For computation, we first initialize matrices R and A by drawing from Standard Normal distribution and then perform Softmax to obtain T and Λ. During training process, we keep track of computational graph and obtain the gradient $\nabla_T \mathcal{L}(\cdot\,; \varepsilon)$ and $\nabla_\Lambda \mathcal{L}(\cdot\,; \varepsilon)$ with respect to T and Λ. R and A are then optimized by Adam optimizer (Kingma & Ba, 2015) after each batch, and the automatic differentiation is done by PyTorch framework (Paszke et al., 2017).

After conducting Wasserstein Dictionary Learning on documents for clustering, the next step of WIG would be to generate time-series indices based on the topics. The model first reduces each topic vector t_k to a scalar by Singular Value Decomposition and then multiply the weight matrix to get document-wise sentiment score for the whole corpus. We then add up the scores for each month and then produce the final monthly index.

2.2. Pruned WIG (pWIG) Model

Although enjoying many nice theoretical properties (Villani, 2003), the computation for Optimal Transport has been known for its complexity. This burden has been eased by Cuturi (2013) and it has attracted much attention in machine learning community since then.

However, there are still two aspects that hindering our application to textual analysis. First of all, vocabulary will easily go to a very large one, and the computation for Sinkhorn loss will soon become prohibitive. Moreover, after passing a certain point, it not even possible to fit the distance matrix C into the memory, especially when considering the limited VRAM for GPU acceleration.[3]

I therefore propose the following procedure to reduce the vocabulary dimension and could avoid feeding the full vocabulary matrix into WIG model. It first clusters all word vectors by k-means clustering, and then selects a subset of tokens from each of the cluster to form "base tokens."[4] We could then use Lasso[5] to regress word vectors of all other tokens on the vectors of these "base tokens" to ensure sparse weight vector, which will have zero component on non-import features.

Formally speaking, we set up the following minimization problem for the k-means clustering:

$$argmin_{\mathcal{K}_1,\mathcal{K}_2,\cdots,\mathcal{K}_n} \sum_{k=1}^{K} \sum_{x\in\mathcal{K}_k} ||x-\mu_k||,$$

where μ_k is the mean of points in cluster $\mathcal{K}_k$ and $k \in \{1,\cdots,K\}$. We can certainly choose some most frequent tokens from each cluster to form a final subset whose length matches our desire.[6] By doing so, we also represent the whole vocabulary by the most representative tokens. The indices for these "base tokens" are collected in the index set,

---

[3] My configuration is Nvidia 1070Ti (8G). Under single precision, each digit will occupy 4 bytes, and, in my case, I can only fit, theoretically at most, a square matrix of dimension 44,721. I have a relatively small dataset from The New York Times and my vocabulary is of length 9437, but many NLP applications will have much more tokens than I do. In such a case, the WIG model will become infeasible.

[4] The number of tokens to be considered as "base tokens" is arbitrary, meaning that the compression ratio could potentially be made arbitrarily small. In other words, the researcher could choose such a number that the model can be fitted into the memory of her machine, regardless of the number of tokens she had for the corpus. And that is exactly the way why we need to compress the dictionary by "pruning" some non-important tokens.

[5] A similar approach (Mallapragada, Jin, & Jain, 2010) was proposed using group-Lasso to prune visual vocabulary, but in the area of image processing.

[6] A very simple choice would be $Word\ per\ Cluster = \frac{Maximum\ Vocabulary\ Length}{Number\ of\ Clusters}$.

$$\mathfrak{B} = \{b \in \{1, \cdots, N\} \mid x_b = 1\}.$$

Obviously, $\mathfrak{B}^C$ is also defined by excluding "base tokens" from the whole vocabulary. N is the size of vocabulary and x_b is the b-th token in the vocabulary.

Denote word vector for "base tokens" as v_b and others as v_o, we have

$$v_o = \sum_{b=1}^{B} \alpha_{o,b} v_b + \lambda \sum_{b=1}^{B} |\alpha_{o,b}|.$$

For each o, we will have a vector $\alpha_{o,b}$ of length B, where B is the dimension of "base vocabulary."

Previously in the WIG model, we obtain the word distribution for each single document y_m by calculating its word frequency, and that will give us a N-dimensional distribution vector. Here, in the pWIG variant, we replace the non-base tokens by weighted base-tokens and could thus represent the word simplex of documents in only B-dimensional spaces.

Now that we have successfully represent our dataset in s smaller vocabulary, we could proceed to define our distance matrix $C_{ij} = d^2(x_i, x_j)$, where $i, j \in \mathfrak{B}$. Here we have everything we need for the regular WIG model and we fit it using the shrinkage-transformed word distributions and distance matrix.

3. Numerical Experiments

*3.1.*wigpy *Package for Python*

To carry out the computation of WIG and pWIG model, I also provide the *wigpy* package under MIT license. Notice that the original WIG model is a new implementation, though part of the codes is modified from the codes of original WIG paper.

The main model is wrapped in the "WIG" class, where it contains a set of hyperparameters[7] to tune the model, and some parameters to control the behavior of preprocessing and Word2Vec training process.

Notice that the previous implementation of WIG model only supports hand-written Adam optimizer, and the optimization for document weights were optimized column-wise. In other words, each document will only be used to update the column of weight in matrix Λ for that given document. The new implementation wraps the whole model in PyTorch,

[7] For example, embedding depth ($emsize$), batch size ($batch\_size$), number of topics (num_topics), Sinkhorn regularization weight (reg), optimizer learning rate (lr), $L2$ penalty for optimizer ($wdecay$), $L1$/LASSO weight ($l1\_reg$), maximum number of tokens allowed by pWIG algorithm ($prune\_topk$).

providing many optimizers to choose by PyTorch optimizer class. What is more, each document will accumulate gradient and the whole Λ matrix will be updated all together.

3.2. Application to Generating Economic Policy Uncertainty Index (EPU)

To test for the pWIG model's performance, I run the model on the same dataset from the WIG paper. It consists of news headlines collected from The New York Times from 1980 to 2018. As I am implementing a new version of WIG, as provided by the *wigpy* module, I run the original WIG model and report its result as well.

I run both variants of WIG model separately, by calling wigpy package, to set for hyper-parameters by splitting training, evaluation, and testing data as 60%, 10%, and 30%, respectively.

For the original WIG, hyper-parameters are chosen as follows: depth of embedding $D = 50$, batch size $s = 32$, number of topics $K = 4$, learning rate for Adam $\rho = 0.001$, Sinkhorn regularization weight $\varepsilon = 0.1$; for the pWIG, depth of embedding $D = 50$, batch size $s = 64$, number of topics $K = 4$, learning rate for Adam $\rho = 0.001$, Sinkhorn regularization weight $\varepsilon = 0.08$.

I also report Pearson's and Spearman's correlation test on four set of automatically generated EPU indices (one LDA-based EPU (Azqueta-Gavaldón, 2017), one WIG-based EPU (Xie, 2020), and two flavor of WIG given by *wigpy* package in this paper), against the original EPU[8] (Baker et al., 2016).

Table 1. Pearson's and Spearman's correlation statistics[9]

EPU Flavor	Pearson's	Spearman's
LDA	77.48%	75.42%
WIG	80.24%	77.49%
WIG-wigpy	**80.53%**	**77.71%**
pWIG-wigpy	80.50%	77.64%

Apparently, as is shown in Table 1, all three WIG methods outperform LDA-based method by 3% in Pearson's test and more than 2% in Spearman's test. This fact has been established by the previous WIG paper. Moreover, if we compare results within three WIG-

[8] https://www.policyuncertainty.com/

[9] Since the LDA-based EPU was only available from 1989-2016, the test is performed using time-series indices within the same range.

related methods, this new implementation of original WIG in *wigpy* package shows better result than the previous implementation. The pruning method does not differ much from the new implemented WIG algorithm and is even better than the previous implementation of original WIG!

Table 2. Correlation statistics with other indices[10]

	VIX Pearson's	VIX Spearman's	Michigan Pearson's	Michigan Spearman's
WIG-wigpy	34.20%	19.56%	-56.40%	-49.38%
pWIG-wigpy	34.27%	19.82%	-56.45%	-49.62%

In Table 2, the correlation statistics between EPU generated by WIGs and two other indices: VIX and Michigan Consumer Confidence Sentiment index. As reported (Baker et al., 2016), EPU has a correlation of 0.58 between VIX and -0.742 between Michigan index. Since our objective is to produce a similar index of EPU, but using an automatic approach, we should expect our WIG-based EPU to have a similar relationship with these other two indices. This is indeed the case here, and we can certainly observe the positive and negative relationship when comparing the VIX and Michigan indices[11].

[10] Here I am comparing both flavors of WIG indices with VIX index and Michigan Consumer Sentiment index, using both Pearson's and Spearman's test. As VIX is only available up to 1986, and the WIG indices was generated up to 2018, I therefore take all the indices from 1986 to 2018 to perform the test. As usual, all indices are scaled to have mean 100 and unit standard deviation. Moreover, the correlation between two WIG indices is 99.86%.

[11] It may be confusing why the "sentiment index" generated by WIG models has a negative relationship with "Michigan Consumer Sentiment index," since both names contain "sentiment." However, there is a clear distinction of the usage of the same word in two different contexts. The famous Michigan index is expressed as the consumer confidence levels, and the higher the index, the more confident the consumers are. The word "sentiment", as used by WIG, is to capture the subjective information expressed in the texts. In the application of EPU, it is used to capture the intensity of opinions towards the uncertainty of policy, as conveyed by newspaper articles. It is very obvious that what it captures is negative feelings, and the higher the index, the more uncertain that people feel. In other words, although bearing the same word "sentiment" in their names, the underlying element is strikingly different and thus show a negative relationship between each other. Moreover, the WIG model does not limit its use in EPU. As soon as we apply the WIG models to other (textual) datasets, the meaning of "sentiment" will be changed accordingly. In total, the word "sentiment" used in WIG models is more versatile and should be distinguished from the usage as in the Michigan index.

3. Conclusion

This paper further extends the Wasserstein Index Generation (WIG) model, by selecting a subset of tokens to represent the whole vocabulary to shrink the dimension. The showcase of generating EPU has shown that the performance is retained while dimension being reduced. Moreover, a package, *wigpy*, is provided to carry out the computation of two variants of WIG.

References

Azqueta-Gavaldón, A. (2017). Developing news-based Economic Policy Uncertainty index with unsupervised machine learning. *Economics Letters*, *158*, 47–50.

Baker, S. R., Bloom, N., & Davis, S. J. (2016). Measuring Economic Policy Uncertainty. *The Quarterly Journal of Economics*, *131*, 1593–1636.

Castelnuovo, E., & Tran, T. D. (2017). Google It Up! A Google Trends-based Uncertainty index for the United States and Australia. *Economics Letters*, *161*, 149–153.

Cuturi, M. (2013). Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26* (pp. 2292–2300). Curran Associates, Inc.

Ghirelli, C., Pérez, J. J., & Urtasun, A. (2019). A new economic policy uncertainty index for Spain. *Economics Letters*, *182*, 64–67.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*.

Mallapragada, P. K., Jin, R., & Jain, A. K. (2010). Online visual vocabulary pruning using pairwise constraints. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3073–3080.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., … Lerer, A. (2017). Automatic differentiation in PyTorch. *NIPS-W*.

Schmitz, M. A., Heitz, M., Bonneel, N., Ngolè, F., Coeurjolly, D., Cuturi, M., … Starck, J.-L. (2018). Wasserstein Dictionary Learning: Optimal Transport-Based Unsupervised Nonlinear Dictionary Learning. *SIAM Journal on Imaging Sciences*, *11*, 643–678.

Shiller, R. J. (2017). Narrative Economics. *American Economic Review*, *107*, 967–1004.

Villani, C. (2003). *Topics in Optimal Transportation*. American Mathematical Society.

Xie, F. (2020). Wasserstein Index Generation Model: Automatic generation of time-series index with application to Economic Policy Uncertainty. *Economics Letters*, *186*, 108874.