Appendix

A1 Alternative Models

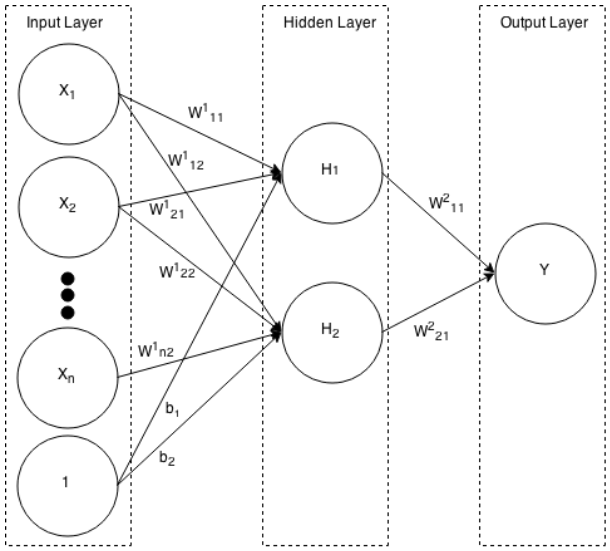
Autoregression X

We specify an autoregressive (AR) model with exogenous variables that accounts for endogeneity. We estimated the AR model of Equation 1 with two lags (the optimal lag length was selected by the Akaike information criterion):

$$Y_t = \beta X_t + \sum_{j=1}^{J} \phi_j Y_{t-j} + \epsilon_t$$

Because the VAR model parameters are not interpretable on their own (Sims 1980), effect sizes and significance are determined through the analysis of impulse response functions (IRFs) and the elasticities that are computed on the basis of the model (for details, see the Appendix).

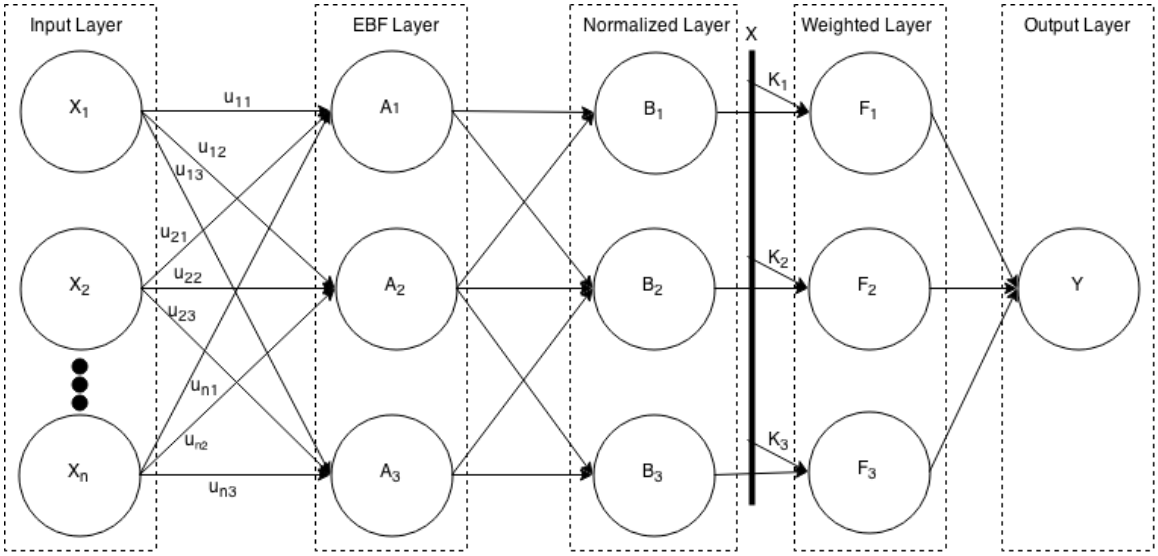
Figure A1 Multi-layered Feedforward Neural Networks



Feedforward neural networks is a widely used non-linear machine learning model. It is one type of artificial neural network (ANN) that was inspired by the structure of an actual brain. Each neuron in the network is able to receive input signals, process them and send an output signal.

As shown in Figure A1, this feedforward neural network has 3 layers of neurons, including inputs $X_1, X_2, \dots X_n$, hidden neurons H_1, H_2 and output Y. Information flows in only one direction, which is forward from the input neurons to the hidden neurons and to the output neuron. The connection between the i_{th} and the j_{th} neurons is characterized by the weight coefficient W_{ij} and the i_{th} neuron by the threshold coefficient b_i. Each neuron performs a weighted summation of the inputs, which then passes a nonlinear activation function. For example, $H_1 = f\left(b_1 + \sum_{i=1}^{n} W_{i1}^1 X_i\right)$, where f is the activation function. Specifically, we use the activation function $f(t) = \frac{1}{1+exp(-t)}$. The network was trained for 100 epochs with the back-propagation algorithm that is based on mean squared errors (Rumelhart et al., 1986). We use cross-validation to determine the number of hidden layers and the number of neurons in each hidden layer. The optimal structure of the network is two neurons in one hidden layer.

Figure A2 Self-Organizing Fuzzy Neural Network (SOFNN) Model



The self-organizing fuzzy neural network (SOFNN) is the five-layer fuzzy neural network that is shown in Figure A2. The five layers are the input layer, the ellipsoidal basis function (EBF) layer, the normalized layer, the weighted layer, and the output layer. The SOFNN has the ability to self-organize its own neurons in the learning process.

The input layer comprises the input vector $x = [x_1, x_2, \dots, x_n]$.

In the EBF layer, each neuron is a T-norm of Gaussian fuzzy membership functions that belong to the inputs of the network. Every neuron has both a center vector (c_{ij}) and a width vector (σ_{ij}), and the dimensions of these vectors are the same as the dimension of the input vector. Specifically, the membership function is

$$u_{ij} = exp\left[-\frac{\left(x_i - c_{ij}\right)^2}{2\sigma_{ij}^2}\right], i = 1,..,n; j = 1, \dots, u$$

and the output of the EBF layer is

$$A_j = exp\left[-\sum_{i=1}^{n}\frac{\left(x_i - c_{ij}\right)^2}{2\sigma_{ij}^2}\right], j = 1, \dots, u$$

Layer 3 is the normalized layer with output B_j as follows.

$$B_j = \frac{A_j}{\sum_{k=1}^{u} A_k} = \frac{exp\left[-\sum_{i=1}^{n}\frac{\left(x_i - c_{ij}\right)^2}{2\sigma_{ij}^2}\right]}{\sum_{k=1}^{u} exp\left[-\sum_{i=1}^{n}\frac{(x_i - c_{ik})^2}{2\sigma_{ik}^2}\right]}, j = 1, \dots, u$$

Layer 4 is the weighted layer. The neuron in this layer has two inputs, namely, the weighted bias w_{2j} and the output of layer 3, B_j. The output F_j is the product of these two inputs, i.e.,

$$F_j = w_{2j}B_j$$

The weighted bias is an inner product of a row vector k_j and the column vector $X = [1, x_1, \dots, x_n]$, as in Takagi & Sugeno 1985.

$$w_{2j} = k_j X = k_{j0} + k_{j1}x_1 + \cdots + k_{jn}x_n, j = 1, \dots, u$$

Layer 5 is the output layer. Each neuron is a summation of the incoming signals from layer 4. Thus,

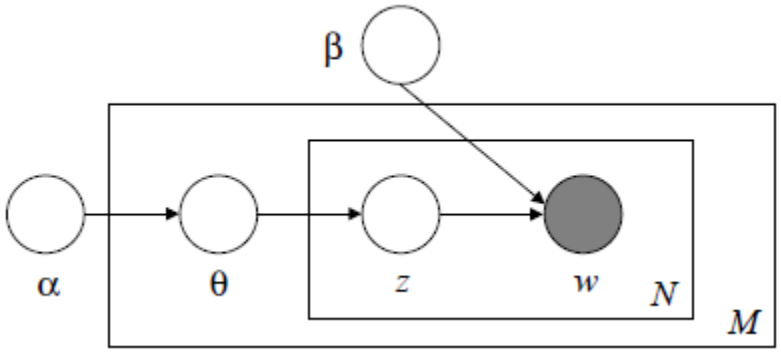
$$y(x) = \sum_{j=1}^{u} F_j = \frac{\sum_{j=1}^{u} w_{2j} exp\left[-\sum_{i=1}^{n}\frac{\left(x_i - c_{ij}\right)^2}{2\sigma_{ij}^2}\right]}{\sum_{k=1}^{u} exp\left[-\sum_{i=1}^{n}\frac{(x_i - c_{ik})^2}{2\sigma_{ik}^2}\right]}$$

The learning process of the SOFNN includes both structure learning (find an economical network size using the self-organizing approach) and parameter learning (using an on-line recursive least square algorithm that was developed by Leng et al. 2004).

LDA (Latent Dirichlet Allocation)

Latent Dirichlet Allocation (LDA, Blei, Ng and Jordan 2003) is a generative probabilistic model that uses an underlying set of "topics" to classify or summarize text documents. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words.

Figure A3 Latent Dirichlet Allocation Graphical Model



Specifically, LDA assumes that each document (which comprises N words) in a collection (of a total of M documents) is generated based on the following process (Figure A3):

1. Choose the joint distribution of a topic mixture $\theta \sim Dir\ (\alpha)$; and
2. For each word w_n, $n = 1,2, \dots, N$
    a. Choose a topic $z_n \sim Multinomial(\theta)$; and
    b. Choose a word w_n from $p(w_n|z_n, \beta)$, which is a multinomial probability that is conditioned on the topic z_n.

To estimate the model, we wish to find parameters a and b that maximize the (marginal) log likelihood of the data:

$$l(\alpha, \beta) = \sum_{d=1}^{M} log[p(w_d|\alpha, \beta)]$$

We use Collapsed Gibbs sampling (Steyvers and Griffiths 2004) methods to make inferences based on the model (as implemented in the R package "lda"). After estimation, we use the topic distribution to summarize the content of the documents. Specifically, we calculate

$$p(\theta|w, \alpha, \beta) = \frac{p(\theta, w|\alpha, \beta)}{p(w|\alpha, \beta)}$$

We are also interested in the distribution of words in each topic:

$$p(w|\theta, z, \beta)$$

For more detailed information concerning lda, please refer to Blei, Ng and Jordan 2003.

Griffiths, T.; Steyvers, M. (2004). "Finding scientific topics". *Proceedings of the National Academy of Sciences* **101** (Suppl 1): 5228–35.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of machine Learning research*, *3*, 993-1022.

Leng, G., Prasad, G., & McGinnity, T. M. (2004). An on-line algorithm for creating self-organizing fuzzy neural networks. *Neural Networks*, *17*(10), 1477-1493.

Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *Systems, Man and Cybernetics, IEEE Transactions on*, (1), 116-132.

Sims, Christopher A. (1980), "Macroeconomics and Reality," *Econometrica*, 48 (1), 1–48

A2 Regression and Prediction Results for the One-Week Window

We extended our analysis to a full-week window. We found that by doing so, the regression result or the prediction analysis results (as shown in Tables 23 and 24) become worse than the results with a 24-hour window for 50% of the models (Table 15) or a 48-hour window for 71% of the models (Table 16). This outcome means that adding information too far in advance of the start of the show dilutes the predictive power of the online platforms data. In addition, because there is a new episode every week, consumers may focus on discussing the previous episode in the first half of the one-week window and then shift to the new episode in the second half. It is intuitive that discussions of the previous episode are not directly helpful for predicting the demand of the new episode. A similar logic can explain why IMDB reviews have poor prediction performance because reviews, by their nature, evaluate an episode after consumers watch it[1].

[1] We thank the anonymous reviewer for this comment.

Table 23: Regression (with more omitted vars) 1-Week

Rating	1	2**	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	Rl	T	G	W	I	H	Rl + T	Rl + G	T + Sen	Rl+T+Sen	Topic	Rl+Topic	Rl+T+G+W+Topic	PC	Rl+PC	Rl+T+G+W+PC
Rating_lag	0.459 (<.001)						0.552 (<.001)	0.412 (<.001)		0.441 (<.001)		0.461 (<.001)	0.394 (<.001)		0.400 (<.001)	0.392 (<.001)
Tweets		0.003 (<.001)					0.001 (0.012)		5.38E-04 (0.027)	7.03E-04 (0.161)			4.85E-04 (0.026)			0.002 (0.027)
Google			1.87E-07 (0.026)					3.33E-06 (<.001)					1.60E-06 (<.001)			3.71E-07 (0.000)
Wiki				1.87E-05 (<.001)									7.13E-06 (0.102)			2.59E-05 (<0.001)
IMDB					-0.122 (0.389)											
Huffington						0.059 (0.194)										
Tweet_Pos									0.001 (0.269)	-0.001 (0.420)						
Tweet_Neg									0.008 (0.462)	0.002 (0.853)						
Tweet_PC1														0.358 (<.001)	0.543 (<.001)	0.440 (<.001)
Tweet_PC2														0.602 (<.001)	0.563 (<.001)	0.844 (<.001)
Tweet_PC3														0.968 (<.001)	0.404 (<.001)	0.629 (<.001)
Tweet_PC4														0.968 (<.001)	0.888 (<.001)	1.001 (<.001)
Tweet_T1											0.421 (<.001)	0.410 (<.001)	0.468 (<.001)			
Tweet_T2											1.220 (<.001)	0.767 (<.001)	0.514 (<.001)			
Tweet_T3											0.567 (<.001)	0.631 (<.001)	0.892 (<.001)			
Tweet_T4											1.472 (<.001)	0.473 (<.001)	1.460 (<.001)			
Tweet_T5											2.006 (<.001)	0.907 (<.001)	0.673 (<.001)			
Premier	0.369 (<.001)	0.345 (<.001)	0.356 (<.001)	0.271 (<.001)	0.431 (<.001)	0.375 (<.001)	0.323 (<.001)	0.232 (<.001)	0.269 (<.001)	0.285 (<.001)	0.368 (<.001)	0.365 (<.001)	0.184 (0.001)	0.336 (<.001)	0.363 (<.001)	0.385 (<.001)
Finale	-0.008 (0.743)	-0.037 (0.262)	-0.022 (0.618)	-0.077 (0.034)	-0.166 (0.004)	-0.098 (0.087)	-0.003 (0.988)	-0.087 (0.075)	-0.049 (0.321)	-0.056 (0.100)	-0.021 (0.528)	-0.005 (0.909)	-0.109 (0.005)	-0.019 (0.717)	-0.024 (0.608)	-0.043 (0.288)
Age	-0.119 (<.001)	-0.137 (<.001)	-0.186 (<.001)	-0.165 (<.001)	-0.236 (<.001)	-0.142 (<.001)	-0.052 (0.002)	-0.153 (<.001)	-0.183 (<.001)	-0.210 (0.001)	-0.153 (<.001)	-0.133 (0.019)	-0.130 (0.001)	-0.101 (0.006)	0.035 (<.001)	-0.149 (<.001)
R2**	0.756	0.067	0.051	0.121	0.034	0.029	0.761	0.780	0.068	0.779	0.814	0.833	0.836	0.856	0.860	0.864
Wald Chi2	244.914 (<.001)	151.014 (<.001)	113.397 (<.001)	203.413 (<.001)	108.363 (<.001)	79.748 (<.001)	274.045 (<.001)	274.531 (<.001)	152.769 (<.001)	279.413 (<.001)	284.262 (<.001)	300.824 (<.001)	521.527 (<.001)	404.451 (<.001)	422.980 (<.001)	442.243 (<.001)
AR(1)	-3.058 (0.001)						-2.883 (0.002)	-3.343 (0.000)		-3.086 (0.001)		-3.124 (0.001)	-3.312 (0.000)		-3.195 (0.001)	-3.147 (0.001)
AR(2)	0.047 (0.481)						1.536 (0.062)	0.277 (0.390)		0.451 (0.325)		0.209 (0.417)	-0.533 (0.298)		0.439 (0.331)	0.281 (0.388)
Sargan Chi2	24.762 (1.000)						20.623 (1.000)	20.659 (1.000)		23.233 (1.000)		17.739 (1.000)	7.200 (1.000)		25.313 (1.000)	21.292 (1.000)
MMSC-BIC	-11738						-3466	-11636		-11520		-11768	-11546		-11973	-11973

Table 24 Prediction: TV Series with Data from One-Week Window

Model		MAPE	MSE
	1	0.1269	0.0864
	2	0.4678	0.3362
	3	0.4834	0.3473
	4	0.4344	0.3106
	5	0.4736	0.3459
	6	0.4958	0.3857
	7	0.1180	0.0802
	8	0.1071	0.0717
	9	0.4657	0.3336
	10	0.1094	0.0733
	11	0.0947	0.0641
	12	0.0801	0.0551
	13	0.0827	0.0559
	14	0.0779	0.0516
	15	0.0719	0.0485
1 wk	16	0.0703	0.0484

A3 Details of Implementation of EMR

Machines: Amazon EC2 and EMR

Amazon Elastic Compute Cloud (EC2)[2] is one of the Amazon Web Services (AWS) that provides resizable computing capacity in the cloud. To use EC2, we simply run an Amazon Machine image using various configurations (which are called instance types) of memory, CPU, storage, and operating system that are optimal for our purposes. For example, we choose the m1.large instance type[3] with 7.5 GB of memory, 860 GB of local storage and 4 EC2 Compute Unit on a 64-bit Unix platform. Because it uses a pay-as you go pricing strategy, we pay for the computing capacity based on only the configuration and time of usage. EMR is Hadoop MapReduce running on EC2 instances. We use EMR because it is easily accessible and cheap compared with an onsite cluster. In addition, Mahout supports running on EMR, and all we need to do is to provide the customized java program files for our algorithms.

Cost: spot instance

To further reduce the cost of running programs on EMR, we choose the Spot Instances (aws.amazon.com/ec2/spot-instances) option of AWS. Instead of paying the high on-demand price, we bid on unused Amazon EC2 capacity. The bidding price (0.026/hour) is almost 1/10 of the on-demand price (0.24/hour).

Implementation

To select relevant Tweets, we obtained 1,874 files (daily Tweets) with a maximum size of 22 GB for a single file. Thus, we provisioned a cluster of 1 master node (m3.medium) and 160 m3.2 large (m3.2xlarge) core nodes, each of which allows for 12 mappers. Amazon has recommended using m3 instances, which perform better than the previous generation m1 instances. This task takes approximately 20 minutes.

To select relevant Wikipedia pages, we obtained 51,636 files (hourly page statistics) with a maximum size of 123 MB. We continued using the provisioned cluster of 1 master node (m3.medium) and 160 m3.2 large (m3.2xlarge) core nodes. We allow each mapper to take 27 files. This task takes approximately 30 minutes.

In this revision, we choose the m3.x2 large instance, which has 30 GB of memory, because the largest file size for the task of selecting relevant Tweets is 22 GB.

To perform the PCA job, we use the SSVD function in Mahout. We set the rank parameter at 100 and the number of reduce tasks at 4. The oversampling parameter is set at the default value of 15. The task finishes in 6 minutes for the TV show dataset.

[2] aws.amzon.com/ec2.

[3] For both the master node and the core nodes.