Supplementary Materials

1. ADDITIONAL EXPERIMENTAL DETAILS

We provide additional details on the conducted experiments below.

1.1. Method, evaluation task and baselines

We tested our estimation method with three different setups: an image-based simulation, an experimental text dataset, and a naturally-simulated experimental genomics dataset. We considered the same evaluation task for all setups:

1. Using a train set, with various "seen" W values, we train a model for phi prediction, and use $\mathbb{E}[\Phi \mid W, Z]$ to fit a lasso regression to predict Y, giving rise to $\mathbb{E}[Y \mid \hat{\Phi}]$ predictions.
2. Next, we see a train split from a test set, corresponding to an "unseen" intervention w', on which we relearn $\mathbb{E}[\Phi \mid w', Z]$
3. Finally, we test Y predictions on a test set of the unseen w' test set, using our relearned $\mathbb{E}[\Phi \mid w', Z]$ model, and our previously trained $\mathbb{E}[Y \mid \hat{\Phi}]$ Lasso regression models. Thus, for a test split of the test set, we predict for Y for previously unseen w', z pairs, i.e. $\mathbb{E}[Y \mid do(w'), Z']$.

For further clarity, we provide a visual description of datasets used in different stages of the method above in Figure 1. We compared our estimation accuracy, via Mean Squared Error, as Y labels become available in the unseen w' regime (10-100% of labels). We record our loss against four baselines estimating the same quantity: 1. linear model, as a Lasso regression with cross validation to pick the coefficient λ on the regularization term in the range [0.05,1], 2. SVM predictor with default parameters from (Pedregosa et al., 2011), 3. Random Forest regression with default parameters from (Pedregosa et al., 2011), aside for specifying 5 minimal samples in split, and 4. Gradient Boosting with default parameters from (Pedregosa et al., 2011), aside for maximum depth which was set to 5. We provide code to reproduce our results alongside this document. We also tested a Multi Layered Perceptron baseline, but found it to perform much worse than alternatives without dedicated tuning, and subsequently did not include it in the results.

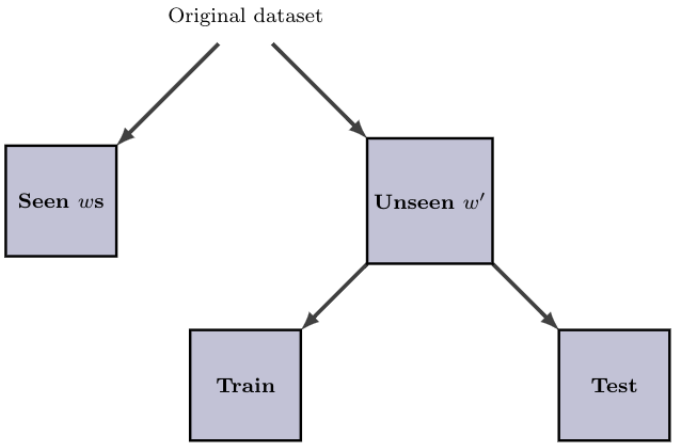

Figure 1. Description of dataset splits used in different parts of our experiment, corresponding to items 1-3 in 1.1.

For the prediction task we include two settings: one for a single configuration of the target Y, and one where we present average performance over 100 different settings of the parameters θ in the structural equation giving rise to Y. Since our

test set on which we report result is rather small, for the first setting we average results of the baselines over 10 different shuffles of the dataset on which we train and report result. We do this for the baselines and not for our own method as the baselines have access to 10 different proportions of the target Y, which involved small number of samples and largely varying performance based on the ordering of the dataset. Our method makes no use of labels, and thus is not vulnerable to this variability in performance. However, when we conduct the experiment over 100 different Y configurations, the existence of ample different examples of data better accounts for this variability in performance based on sample ordering. Thus, we simply report results for a single data ordering, and report the mean and standard error over 100 different Y parameters settings instead.

1.2. Dataset construction and models' training

1.2.1. Image Perturbation

The image perturbation is a simulation dataset that was put together to demonstrate the key ideas of the work. We create a dataset of 10,000 examples, images of size 10X10, equally made up of 5 pixel patterns: cross, square, crossing diagonal, pyramid and diamond. Patterns were allocated for each index by sampling one of 5 pattern indicators from a multinomial with equal probability for each shape. Next, the pattern indicator Z also served to select one of 5 set of probabilities that were used to seed a multinomial from which a perturbation pattern W was picked.

The perturbation was put together via a procedure that used the indicator W as a location to be used in a Multivariate Normal distribution with a two dimensional identity covariance matrix, $I(2)$. For each example in the dataset, we sampled 1000 tuples (x,y) from the Multivariate Normal described, checked whether they fell within the image size (i.e. $\geq (0, 0)$ and $< (10, 10)$). Each time such tuple fell within the borders of the image, a perturbation of size 0.1 was added to the location (x,y) in the image.

This perturbation regime was added as a mask to the original image reflected by Z, to create the post-perturbation X. For the construction of ϕs, we created five different 1-d convolution transformations, with randomly initialized weights, which will be indexed by Z and applied to X. There will be 4 resulting ϕs for each image, each corresponding to the 4 quadrants of the image. ϕ_1 most clearly varies with W, with potential little effect on ϕ_2 and ϕ_3, but ϕ_4 should see close to no effect in response to W, based on perturbation locations. Finally, Y was constructed as a linear combination of ϕs, with the weights [0.7, 0, 0, -0.5] applied to them, and added Gaussian noise $N(0, 0.1)$.

Finally, for the test set with an unseen perturbation pattern w', 2,000 examples that were featured in the training set of size 10,000, were used with a different perturbation, with location $w' = 5$. Previous perturbations ranged from 0-3. For specific structural equations corresponding to this description, see Figure 5 in the manuscript.

For the g model, estimating $\mathbb{E}[\Phi \mid w', Z]$, we train a Multi-task MLP with 3 hidden layers, 512 hidden dimensions, and the ReLU activation function. We use 100 epochs and 50 epochs to train the first-stage model g for seen w, and for the unseen w' train split respectively. We use the Adam optimizer, with learning rates of 0.002 and 0.001 used for optimization of each stage respectively. We use a train batch size of 400, and test batch size of 100. We set the seed at 42. We also include the code to reproduce the results, see code files for any additional hyperparameter setting.

Finally, for the robustness to settings of θ experiment (Figure 8), we had to sample the weights from a distribution such that we can repeat the process 1500 times. We chose a uniform distribution $\theta \sim Unif(-0.3, 0.3)$, and added noise to Y from a $N(0, 0.01)$. We chose those such that similar stats of Y can be achieved, compared to the single Y setting experiment. See generation code in Python_Img_Humor/data_generation_notebooks/ImgPertSim.ipynb.

1.2.2. Humor Micro Edits

The construction of the humor micro edits dataset closely followed the analysis and description in (Hossain et al., 2019). The original datasets provided there already contained original headlines (which we used as Z for our purposes), edit words (W for us) and humorous post-edit headlines (X)[1]. Following the analysis in the paper, we chose to represent each of these sentences and words using their pre-trained 6 Billion-token GloVe word-embedding vector representations, trained originally on 2014 English Wikipedia and Gigaword 5[2] (Pennington et al., 2014). We only included examples in which all words were correctly identified in the pre-trained word embedding, following a standard cleaning procedure (see code for exact details).

[1] Access to original dataset at https://www.cs.rochester.edu/u/nhossain/humicroedit.html.

[2] Available for download at https://nlp.stanford.edu/projects/glove/.

Z, W and X where does based on vector representation of the original dataset. There are three additional steps we have taken to compose the final dataset. First, we clustered the edit words using K-means clustering with K=20 (implemented via Scikit-learn), following the same procedure carried out in (Hossain et al., 2019). We used the labels of these clusters to create the training and test split, such that the test set included edit words from one cluster we left out to function as an unseen intervention W'. The unseen intervention was chosen to be one of the larger 5 clusters, to ensure enough training examples for estimation exist. The cluster that was randomly chosen for the results shown in the paper is cluster 11.

Next, we constructed high-level descriptions of the intervention and its implications on X as ϕ. $\Phi = \{\phi\}_{i=1}^{30}$, and each one was inspired by analysis and hypotheses from (Hossain et al., 2019):

1. ϕ_1: Length of resulting edited sentence *(does not vary with w)*
2. ϕ_2: Mean cosine distance between GloVe vector of edit word and the rest of words in sentence *(varies with w)*
3. ϕ_3: Location index of replaced word *(does not vary with w)*
4. ϕ_4: Sentiment polarity of edit word, using the pre-trained sentiment processor from (Qi et al., 2020)[3] *(varies with w)*
5. ϕ_5: Sentiment polarity of resulting sentence, using the pre-trained sentiment processor from (Qi et al., 2020) (does not vary with w)
6. ϕ_6: Cosine distance between GloVe vector of edited word and GloVe vector of original word *(varies with w)*
7. ϕ_7-ϕ_{10}: Cosine distance of GloVe vector of edit word from neighboring words (2 preceding, 2 succeeding) *(does not vary with w)*
8. ϕ_{11}-ϕ_{30}: Distance of mean GloVe embedding of final sentence from clusters' centroids *(does not vary with w)*

The set of ϕs, which all correspond to different data types, were all scaled to have 0 mean and unit variance, to make them more comparable. Finally, following Φ as defined above we constructed an outcome variable Y as a linear combination of Φ, with added noise sampled from $N(0, .5)$. We sampled weights for each $\phi, \theta \sim U(-1, 1)$, while keeping a random third of the weights at 0. Additionally, we ensured at least one of the ϕs varying with W is also zeroed out, to have diversity of all possible cases present.

For the g model, estimating $\mathbb{E}[\Phi \mid w', Z]$, we train a Multi-task MLP with 3 hidden layers, 512 hidden dimensions, and the ReLU activation function. We use 700 epochs and 100 epochs to train the first-stage model g for seen w, and for the unseen w' train split respectively. We use the Adam optimizer, with learning rates of 0.002 and 0.001 used for optimization of each stage respectively. We use a train batch size of 400, and test batch size of 100. We set the seed at 42. We also include the code to reproduce the results, see code files for any additional hyperparameter setting.

1.2.3. Gene Knockouts

Our GENIE3 model follows the instructions of Huynh-Thu et al. (2010), who scale all genes prior to analysis and fit a series of random forest regressions predicting the expression of each "downstream" gene as a function of the 334 candidate transcription factors (TFs). Each forest contains 1000 trees, with $mtry = \sqrt{334}$. The adjacency matrix is computed using the impurity importance measure originally proposed by Breiman (2001).

We simulate TF data from a multivariate Gaussian distribution with parameters estimated via maximum likelihood. This matrix is then propagated through our GENIE3 model to simulate expression values for downstream genes, with random Gaussian noise $\mathcal{N}(0, \sigma^2)$, where σ is the RMSE of the corresponding random forest on out-of-bag data. This data – TFs and downstream genes – together comprise the matrix Z of simulated baseline *E. Coli* gene expression.

We sort TFs by outdegree and simulate a knockout experiment on the top ten by replacing their values with a scalar 1 unit less than the observed minimum for each (how much less is irrelevant, as random forests are invariant to monotone transformations). We record the outdegreee of these TFs as W and the resulting expression matrix as X.

To compute Φ, we filter out all TFs with outdegree less than 100 and treat each of the remaining 168 TFs as the hub of a module. For downstream genes, module membership is determined by whether the given TF was assigned importance

[3]Full usage details available at https://stanfordnlp.github.io/stanza/sentiment.html.

of at least 10 in the GENIE3 adjacency matrix. For each module, we compute the first kernel principal component on a subsample of $n = 1000$ using a radial basis function with default bandwidth given by the median Euclidean distance. These weights are then used to project the remaining data Z and X into the latent space. We define Φ as the difference between pre- and post-intervention expression values for the kernel eigengene. We proceed to estimate a series of $\mathbb{E}[\phi_j \mid Z, W]$ regressions on a training set comprising 8 of 10 w values using random forests with 500 trees and $mtry = p/3$, where p is the number of genes in a given module.

Each ϕ_j is sorted by its association with W using Spearman correlation. Y is then simulated as a linear function of the top and bottom 25 ϕ's, with nonzero weights drawn from $\mathcal{N}(\pm 4, 1)$, where ± 4 denotes that the amplitude is multiplied by -1 with probability 0.5. A lasso regression $\mathbb{E}[Y \mid \hat{\Phi}]$ is fit to the aforementioned training set, with L_1 penalty λ selected via 10-fold cross-validation. Since, by construction $Z \perp\!\!\!\perp W$ in this experiment, the conditional independence tests can be replaced by a marginal independence test. We use the Spearman correlation to measure the association between W and each ϕ_j using the training set.

References

Breiman, L. Random Forests. *Machine Learning*, 45(1):1–33, 2001. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004.

Hossain, N., Krumm, J., and Gamon, M. "president vows to cut <taxes> hair": Dataset and analysis of creative text editing for humorous headlines. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 133–142, 2019.

Huynh-Thu, V. A., Irrthum, A., Wehenkel, L., and Geurts, P. Inferring regulatory networks from expression data using tree-based methods. *PLoS ONE*, 5(9):1–10, 2010.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.