**Title**
Applications of Statistics in Machine Learning Problems



**Author**
Wei, Lifeng

**Publication Date**
2021

Peer reviewed|Thesis/dissertation

**Applications of Statistics in Machine Learning Problems**

By

LIFENG WEI

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

STATISTICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

---

James Sharpnach, Co-Chair

---

Cho-Jui Hsieh, Co-Chair

---

Alexander Aue

Committee in Charge

2021

To my beloved wife, for how unique you are and how lucky you make me.

# Contents

Applications of Statistics in Machine Learning Problems

**Abstract**

Recent improvements in machine learning methods have significantly advanced many fields including computer vision and natural language processing. While the models have become increasingly complex, they have also become harder to interpret and be trusted.

In my thesis, the focus is to improve or solve machine learning problems with clear guidance from statistics and design algorithms which can be supported or explained by statistics. We show statistics can be applied from various aspects and play different roles.

In the first project, I use a heuristic search method to design a simply but effective neural network compression method, with a theoretical understanding of how the search works. In the second project, optimization based on Kullback-Leibler divergence is applied to fine-tune the output distribution of an image segmentation module to improve quality of segmentation. In the third project you will see how hypothesis testing can help measure the similarity between datasets, which is a crucial question for a better understanding for maching learning.

Besides these applications on machine learning algorithms, there are also engineering challenges in real-life projects. For the fourth project, I summarized my contribution to Healthy Davis Together project, including the systems I built with full details, the logic behind those systems and a mobility variable I designed.

**Acknowledgments**

First, I would like to say thank you to my family. Without my parents' guidance and support from the first day of school, there is no chance for me to walk out of a small town and pursue my Ph.D. in a new country. They've made themselves the excellent examples for me to follow. Without my wife's company and love, there would be much less happiness but more anxious along that path I've taken. Her encouragements and unique suggestions helped me overcome several challenging tasks.

Besides my family members, I would like to express my sincere gratitude to my advisors, Prof. Sharpnack and Prof. Hsieh, for continuous support of my Ph.D. studies and researches. Their trust and patience allowed me to explore and dive into different areas and study questions that are intriguing as well as exciting. I could not have imagined having better advisors and mentorships.

I would like to thank the rest of my thesis committee: Prof. Aue, Prof. Ethan and Prof. Li, for their insightful comments on the thesis, and also the encouragement on my research path.

I have also received so many help from staffs in the statistics department, Pete, Nehad, Sarah and Andi. Their capable hands saved me from technique issues and administrative problems through the entire five years, allowing me to focus on my studies and researches.

My sincere thanks also go to Dr. Yi-min Jiang, Dr. Aitzaz Ahmad and Mr. Stephen Lau. The internship opportunities and mentorships you provided to me are great lessons in my career path. The hand on experiences allow me to feel the difference between academia and industry and inspired me to focus on my researches, dive deeper and think harder.

Last but not least, I feel also extremely lucky to meet other PhDs and masters at Davis. I wish our friendship would last forever. The card games we played, the festivals we celebrated and the helps I received from you are an important part of these five years.

CHAPTER 1

# Introduction

In recent years, machine learning is rapidly developing and applying to many scenarios. In computer vision, new structures are being brought up as backbones and downstream tasks like object detection and image segmentation [**4**, **6**, **27**, **28**, **36**, **38**, **69**, **82**]. For natural language processing, the appearance of Transformer structure [**85**] and large-scale pretraining tasks totally changed how people process text information [**21**, **68**]. Reinforcement learning has also made progress in both theory and applications. New learning algorithms enable stable and diverse learning [**23**, **24**, **26**, **33**, **50**, **59**], and applications like OpenAI Five [**64**], AlphaStar[1] show promising applications of control with reinforcement learning.

However, concerns rise with improvements. Most developments, especially in computer vision and natural language processing, are excellent implementations of simple but important ideas. For example, Focal Loss [**53**] in object detection, attention mechanism in Transformer [**85**], pretraining tasks for BERT [**21**] and all its successors [**17**, **47**, **49**, **75**, **90**], and the hidden but important engineering details in reinforcement learning [**22**]. They address important issues for different tasks and proposed efficient solutions but we still only have empirical but not theoretical understanding of them. This makes machine learning sometimes hard to be explained, trusted, or fixed when a problem appears.

In this dissertation, we focus on solving machine learning problems with clear guidance from statistics. You can see different roles played by statistics. From an intuitive modification, to the very basic of the design of the whole algorithm. Our results will show that with a clear understanding of the changes we make, simple ideas can also lead to stable and significant improvements in machine learning problems.

In Chapter 2, we applied a simple heuristic search algorithm, Monte Carlo Tree Search, to improve network compression quality, i.e. compress more weights while maintaining performance.

[1] `https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii`

We analyze how it works in a one-layer condition to understand the relationships for hyper-parameter tuning.

In Chapter 3, we use unsupervised fashion to train a multi-stage image segmentation module, which tries to segment out one object for each stage. The output from each stage is fine-tuned to optimize some Kullback-Leibler divergence. We achieved a good balance between extra time consumption and improvement for segmentation.

In Chapter 4, we address an important question in machine learning: how similar two datasets are. The question is important for better understanding of machine learning problems such as domain adaptation and evaluating generative models. We designed a novel and flexible framework to calculate similarity between datasets with graph structure and hypothesis testing.

Last but not least, we come back to real-life problems in Chapter 5. Full details of my contribution in Healthy Davis Together project are described, as well as the logic and challenges behind. We built a back-testing framework for fast model evaluation and iteration. We designed a mobility variable to better reflect the spread speed of virus locally. And there is also a system to connect two sources of data, which are collected on different geo resolutions.

CHAPTER 2

# Search to Compress: Layer-wise Compression of Deep Neural Networks by Monte Carlo Tree Search

## 2.1. Abstract

Deep neural networks typically have millions of parameters in order to achieve good performance. The huge model size prevents a wider use of deep neural networks, especially on mobile devices. It is thus important to study how to compress the models without sacrificing efficiency. Previous methods like parameter pruning and quantization have been successfully used to reduce the memory cost, but they were usually applied uniformly to all the layers. To achieve a better compression rate, one has to select different compression strategies for different layers, which requires manual parameter setup (e.g., choosing different compression strategies for convolution versus fully connected layers). In this paper, we pose model compression as a search problem, and apply Monte Carlo Tree Search to obtain the best configuration. Our method can be applied to all kinds of models without human interference, and can be used to boost the performance of most existing compression algorithms. We tested our method to VGG and ResNet networks on ImageNet dataset, and observed significant improvements over uniform pruning and quantization approaches.

## 2.2. Introduction

In this paper we study model compression for deep neural networks. Although this topic has been studied extensively in the past few years and many methods including quantization [**39**, **52**, **84**], pruning [**34**,**35**,**56**,**58**,**71**], and low-rank approximation [**20**,**40**,**74**,**93**] have been developed, previous approaches either uniformly applied to each layer or require human knowledge to decide different strategies for different layers. For example, it is common to treat convolutional layers and fully connected layers differently in order to get the best compression rate. This tuning process is quite

slow and often requires domain knowledge, which may not be available when it comes to networks that we have less knowledge about.

To resolve this difficulty, we propose an algorithm to find the best configuration for compressing a given network without human interference. Assume each layer has a few "actions" that can be selected, such as pruning 80% of the parameters, pruning 90% of the parameters, or quantizing every weight to 4 bits. Model compression can then be posed as a search problem—finding the best overall configuration to maximize the "compression objective", defined as a weighted average of validation accuracy and compression rate.

Unfortunately, when there are $T$ layers and each layer has $n$ actions to try, there will be $n^T$ configurations in total and a brute-force search will not be feasible. To make this process faster, we rethink the compression procedure as a game, where at each layer the player wants to find the best action to maximize the final reward, which is defined to be the compression objective in our case. This motivates a Monte Carlo Tree Search approach (used in AlphaGo [**77**]) to find the best configuration. We modify this algorithm to fit the model compression task, and prove some interesting properties about the decisions for the simplified case.

The experimental results on ImageNet demonstrate two interesting findings. First, our method can be used to find layer-wise pruning and quantization parameters that outperform uniform pruning/quantization. This verified that our method can sucessfully find a better per-layer configuration and boost the performance of existing methods. Second, if we give the options for both quantization and pruning for each layer, our method can find even better compression configuration outperforming pure pruning or pure quantization. This indicates it is important to mix several strategies for compression, and our method can successfully find a good configuration for that.

### 2.3. Related Work

Here we briefly review previous work on neural network compression. The strategies can be roughly categorized into pruning, quantization, low-rank approximation, and some other mixed strategies to combine two or more structures.

Compression by Pruning. It has been observed that deep neural networks usually have many unimportant weights with small magnitude, and often removing those weights won't hurt the

performance. Based on this idea, [**35**] used pruning method with $L_1$ and $L_2$ norm and retrain to maintain accuracy. These two steps are done iteratively. [**58**] designed an elegant function so that the pruning process can be performed during backpropagation. [**34**] showed that a simple thresholding approach surprisingly works well in practice, and demonstrate superior performance on several state-of-the-art networks. Besides pruning weights, some recent approaches including [**56**, **71**] aim to remove neurons in the trained network.

Compression by Quantization. Quantization is another effective way to reduce the model size of deep networks. [**84**] first explored an implementation using 8-bit quantization and reduced the computation cost affordable for CPUs. [**39**] proposed to quantize the whole network in the training phase, and show little accuracy lost when presenting each weight with 2–4 bits. [**52**] further proposed to solve an optimization problem to identify optimal fixed point bit-width allocation. In addition to element-wise quantization, some recent approaches also consider quantizing vectors in the given weight matrix. [**88**] applied vector quantization on both convolution layers and fully connected layers to accelerate computation. [**14**] combined the idea of quantization and codes to improve the performance of quantization. Both scalar quantization and vector quantization were discussed.

Compression by Low-rank approximation. Low-rank tensor approximation has also been used to reduce the size of weight matrices. [**74**] approximated the fully-connected layer using top-k singular value decomposition. Later on [**20**, **40**] proposed to apply tensor decomposition methods to compress the convolution kernels in CNN. [**93**] recently showed that low-rank approximation is useful for both reducing memory cost and improving testing time.

Mixed strategy. To achieve the best compression of state-of-the-art networks, usually multiple compression strategies have to be used. The deep compression paper in [**34**] combined all three methods and improved them with special techniques like retraining, retrain code book or Huffman coding and got state-of-the-art results. [**91**] reconstructed the weight matrices by matrix factorization and a sparse matrix to reduce the size of deep neural networks. A comprehensive survey of deep neural network compression can be found in [**13**].

### 2.4. Our approach

In this paper, we propose a new method for compression of neural networks based mainly on Monte Carlo Tree Search(MCTS) and a loss based methods for pruning and quantization. In this section we will first review MCTS and then propose our loss based compression method. Then the application of MCTS will be explained.

**2.4.1. MCTS.** MCTS is a searching method in tree-structured search space. Assume that there is a multi-layer tree, with several son nodes for each node and only the leaves stores a value, which further might be random variables. The number of nodes is so large that it is almost impossible to find the optimal value by visiting all leaves.

Instead of directly doing search in the whole search space, MCTS creates its own smaller search space, here we just call it 'search tree'. The search process follows this loop:

(1) Starts from current root

(2) As long as we are not in a leaf of search tree, we choose son:

$$j = \text{argmax}_i v_i + c_{puct} * p_i * \frac{\sqrt{\sum_k N_k}}{N_i + 1}$$

Here $i$ is the index of sons. $v_i$ is the evaluation result, usually an average of past evaluations. $N_i$ is the number that this node has been evaluated. $p_i$ is a prior belief which can be ignored when no belief exists. This is similar to UCB [**3**]

(3) After having reached a leaf in the search tree by some given method, evaluate this leaf. The value will be stored as an evaluation result for all nodes on the path between root to this leaf.

(4) Do necessary expansion and cut off redundant nodes if there is any. See details in later discussion.

If a leaf in search tree is visited for a number of times, it's sons in the original search space will be included to the search tree. This is called Expansion. The process of cutting off redundant nodes is also called pruning but to avoid confusions with the pruning for neural networks we call it cutting off here. In the original setting once an action from the current root is made, only the subtree under the chosen action will be kept.

**2.4.2. Compression method.** In the past people have proposed different ideas for pruning/quantization. Here we implement a different but quite simple idea: loss approximation.

Assume $w$ is the parameter, which can be divided into $p$ different groups $\{w_1, w_2, \ldots, w_p\}$. $f$ is the model, $\hat{w}$ is the pruned/quantized parameter. $C$ is the set of values that can be chosen for each weight. Usually people will choose $\hat{w}$ purely based on some distance measurement:

$$\min_{\hat{w}} \sum_{j=1}^{p} \|w_j - \hat{w}_j\|^2 \quad \text{where} \quad \hat{w}_j \in C \ \forall j = 1, \ldots, p$$

However, this does not take loss function into account. It's possible that there is a solution that is closer in distance but leads to much larger loss function. [**43**] tried to minimize loss function directly:

$$\min_{\hat{w}} \frac{1}{n} \sum_{i=1}^{n} \text{loss}(f(\hat{w}, x_i), y_i) \quad \text{where} \quad \hat{w}_j \in C \ \forall j = 1, \ldots, p \tag{2.1}$$

However, solving (2.1) is time consuming (even slower than training). Instead, we plan to optimize the first or second order approximation of the loss function. For example, we can approximate the loss function by

$$\begin{aligned}
\text{loss}(f(\hat{w}, x_i), y_i) &\approx \text{loss}(f(w, x_i), y_i) + (\hat{w} - w)^T \nabla_w \text{loss}(f(w, x_i), y_i) + \frac{\lambda}{2}\|w - \hat{w}\|^2 \\
&= \sum_{j=1}^{p} \big(\frac{\lambda}{2}(\hat{w}_j - w_j)^2 + (\hat{w}_j - w_j)\nabla_{w_j} \text{loss}(f(w, x_i), y_i)\big) + \text{constant} \\
\frac{1}{n}\text{loss}(f(\hat{w}, x_i), y_i) &\approx \sum_{j=1}^{d} \big(\frac{\lambda}{2}(\hat{w}_j - w_j)^2 + (\hat{w}_j - w_j)(\frac{1}{n}\sum_{i=1}^{n} \nabla_{w_j} \text{loss}(f(w, x_i), y_i))\big) + \text{constant}
\end{aligned}$$

Plugging in this approximation into (2.1), the problem can be decomposed into 1 dimensional problems for each variable $\hat{w}_j$, and each of them will be

$$\min_{\hat{w}_j} \frac{\lambda}{2}\big((\hat{w}_j - w_j) + \frac{1}{\lambda}\nabla_{w_j} L(w)\big)^2 \quad \text{where} \quad \hat{w}_j \in C \tag{2.2}$$

In practice, $w$ is the network weight and we can divide $w$ into $\{w_1, w_2, \ldots, w_p\}$ by different layers. For pruning, we can simply choose the positions that minimizer the sum of element-wise loss. For

quantization, we can first solve this equation in continuous domain, then optimizer must lie between two values in $C$. We only need to compare these two values.

One thing to mention, however, is that the suitable hyperparameter $\lambda$ changes in scale for pruning and quantization since the loss structures are quite different. To make this approximation work, $\lambda$ has to be in a reasonable range. Within this range it's not sensitive. Details are in the following section.

**2.4.3. Compression of Neural Networks.** We formulate the compression task as simultaneously finding best compression options for all layers in the network. But we know even if the options are limited for each layer, the total number of choices still grows exponentially with the depth of network, which makes the task impossible to solve. We apply MCTS and combine it with the compression scheme proposed in Section 3.2 to overcome this difficulty. In later experiments we can see that the overall complexity becomes almost linear to the depth of the network.

To apply MCTS we need to define the following: the meaning of each node, action space, evaluation method and criterion to make actions as well as cutting off branches. Clearly each layer in MCTS can correspond to a layer in the network and the action space will be the set of possible compression options. Each node will correspond to a compression configuration. The design of evaluation and cutting of branches are nontrivial, as we will discuss below.

2.4.3.1. *Different Evaluation.* Since we are applying MCTS, a method to evaluate each possible action is necessary. One direct metric is the accuracy of the compressed network with a penalty for the remaining size of network:

$$v_i = acc_i - c \times r_i$$

Here $i$ is the index for a node, $acc_i$ is the accuracy and $r_i$ is the relative size of compressed network compared with original network. $c$ is a coefficient of penalty. This evaluation is directly connected to our goal: compress the network while maintaining its performance. The parameter $c$ balances the trade-off between compression and performance.

Although we cannot obtain the exact value of $acc_i$, it's not hard to estimate it with a small batch of sample. The variance of the estimator would be small as well since it's an average of some Bernoulli random variables. The variances cannot be larger than $\frac{1}{4k}$, where $k$ is the batch size.

We also consider another metric based on the loss function:

$$v_i = -loss_i - c \times r_i$$

The meaning of $i$, $c$ and $r_i$ are the same as above. $loss_i$ is the cross-entropy loss between the true label and predicted distribution. Namely

$$loss_i = \mathbb{E}[H(label_x, dist_x)] = \mathbb{E}[-\log(dist_x[label_x])]$$

where $x$ is the index for data, $label_x$ is the true label for sample $x$ and $dist_x$ is the predicted distribution of labels on sample $j$, which is the output of a softmax layer. Again this cannot be obtained accurately so that we have to use estimators from samples. The variance of this estimator is hard to know since the loss is not bounded. But if we assume that the worst case is uniform distribution on all classes, the variance is no larger than $\frac{\log^2(N)}{4k}$, where $k$ is the batch size and $N$ is the total number of classes.

2.4.3.2. *Cutting off branches.* Originally, once an action from the root of search tree is evaluated over a certain number of times, the actions will be chosen and only the subtree under this action will be kept. This method is also applicable in our task but it gives us less idea about how many layers we are searching.

To have a better control of the layers we search at the same time, we choose an alternative cutting method. In our experiments, we cut off branches when the current deepest node in search tree is $m$ layers deeper than the current root. For all sons of current root, only the one leading to the current deepest node will be kept. In this way we can guarantee that there are always $m$ layers of network being searched. And we stop searching once a node in the final layer is visited for a number of times. From results in Figure 2.3 we can see a larger $m$ does have positive influence on results.

**2.4.4. Full Algorithm.** Algorithm 1 describes the full steps to compress a pretrained network with our method.

**2.4.5. Analysis.** Now we analyse the performance of MCTS within one layer. We assume each leaf node is a random variable, and each time we test a node for $b$ times to get the average score. This corresponds to our compression setting, where we test the performance on a mini-batch with $b$

**Algorithm 1:** Monte Carlo Tree Search for Compressing DNN

Create the first Node, name it *start*, the state is empty
**while** Have not reached the last layer of DNN **do**
  $Search(start)$ {See Algorithm 2 for Search function}
  **if** Currect tree depth>Chosen tree depth **then**
    Cut off branches and change root node
  **end if**
**end while**

**Algorithm 2:** Search function in Algorithm 1

$i \leftarrow \text{argmax}_a Q(s,a) + c_{puct}\frac{\sqrt{\sum_a N(s,a)}}{N(s,a)+1}$ {This is Selection}
**if** Son $i$ is the last layer of DNN **then**
  $v \leftarrow Score(s+i)$ {s+i means add i to the end of state s}
  {Score is the function used to evaluate}
**else if** Son $i$ corresponds to a node **then**
  $v \leftarrow Search(s+i)$
**else**
  $v \leftarrow Score(s+i)$
  **if** $N(s,a) >$ threshold **then**
    Expand: create a node for $i^{th}$ son. {This node corresponds to s+i}
  **end if**
**end if**
$N(s,a) \leftarrow N(s,a)+1$
$V(s,a) \leftarrow V(s,a)+v$
$Q(s,a) \leftarrow \frac{V(s,a)}{N(s,a)}$
return $v$

validation samples when we reach a leaf node. We have claimed that the next move can be decided by simply choosing the action which is first tested multiple times. This is supported by the following theorem.

THEOREM 2.1. *Consider a MCTS with only one layer. If we now have k choices with expected values* $v_1, v_2, \cdots, v_k$ *and variances* $\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2$, $\sigma_i^2 \leq M\ \forall i$. *W.l.o.g we can assume* $v_1 \geq v_2 \geq \cdots \geq v_k$. *Set* $d = v_1 - v_2$. *Each time the selection rule is following* (**??**) *with constant* $c_{puct}$, *and the selected choice is tested by a mini-batch with size b. We make the decision if the test time of a choice is more than t. Then the probability that we can choose the best choice is approximately lower*

*bounded by*

$$\begin{cases} 1-(k-1)t\cdot exp(-\frac{bd^2t}{4M}) & if \quad \sqrt{t} < \frac{c_{puct}}{d} \\ 1-(k-1)t\cdot exp(-\frac{bd^2t}{16M}) & if \quad \frac{c_{puct}}{d} \leq \sqrt{t} \leq \frac{2c_{puct}}{d} \\ 1-(k-1)t\cdot exp(-\frac{bc_{puct}d\sqrt{t}}{4M}) & if \quad \sqrt{t} > \frac{2c_{puct}}{d} \end{cases}$$

The full proof is in Appendix A.1. We can see that larger $k$ (more choices) leads to a smaller probability. But actually $(k-1)t$ can be replaced by $(k-1)(t-m_{c_{puct},k,t})$, where $m_{c_{puct},k,t}$ is the smallest possible visit time with given $c_{puct}, k, t$. Details about this are in the proof. Clearly $m_{c_{puct},k,t}$ will increase with $k$ so the probability will not always decrease with increasing $k$.

The above theorem proved that the naive approach to pick the next step is efficient. This is for the single layer search. In multi-layer search, we can regard each node's value as a weighted average of all its children.

Besides, this theorem also provides a guideline for choosing these parameters without knowing much about the DNN to be compressed. $M$ can be roughly estimated and $k$ is usually pre-determined. $b$ can be the largest possible number that can be handled by the hardware. So now a suitable $t$ can be chosen if we want this MCTS method being able to distinguish a difference $d$. None of these steps requires knowledge about the neural network.

### 2.5. Experiments

In this section we present some empirical results to show the influence of some hyperparameters and the effect of our method. The experiments are conducted on ImageNet2012 [**73**] dataset. Most comparisons are done one VGG16 [**78**] and the compression result are also available for VGG19 [**78**], Resnet18 and Resnet34. [**36**]. All pretrained networks are shared by Pytorch.

**2.5.1. Choice of $\lambda$.** The choice of $\lambda$ is tricky: If it is too small, the estimated optimal weights will go too far on gradient's direction. But if it is too large, any deviance from original weights will be punished too much and original weights will be the only choice.

We also realized that within a large range the result is not sensitive to the choice of $\lambda$. In this paper we choose $\lambda$ by quantizing the network to 8-bit or prune out 25% weights and use the smallest $\lambda$ whose loss in accuracy before retrain is less than 2%. We searched for $\lambda$ in a low precision. This does not guarantee the performance would be the best but our results shows that this is already

better than directly prune/quantize the original weights. In Figure 2.1 we can see the clear gap between using/not using a $\lambda$.

One more thing to mention is that for pruning and quantization the choice of $\lambda$ are quite different. This comes from the different loss structure. For pruning method some weights directly goes to 0 while others remain unchanged. For quantization method all weights changed in its own small neighborhood. It turns out pruning method would result in a larger change w.r.t norm and thus a smaller $\lambda$. Below is our choice of $\lambda$:

| Network | VGG16 | VGG19 | Resnet18 | Resnet34 |
|---|---|---|---|---|
| Pruning | 15 | 15 | 5 | 5 |
| Quantization | 40 | 45 | 20 | 20 |

TABLE 2.1. Choice of $\lambda$

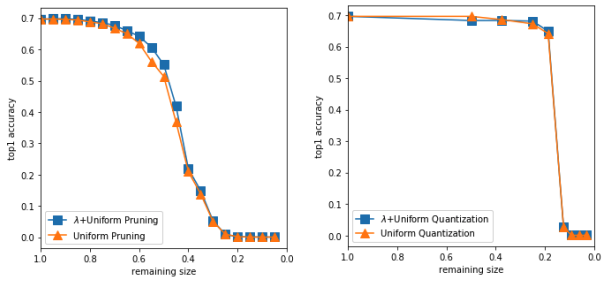



FIGURE 2.1. The above pictures are the compression results of different methods tested on resnet18. The left one uses pruning and the right one uses quantizetion. Three settings are tested: Uniform pruning of original weights, uniform pruning of adjusted weights and MCTS+pruning for adjusted weights. We can see a clear improvement for using $\lambda$ in pruning. For quantization the performance drops so fast and the gap is hard to find.

**2.5.2. Choice of Compression Method.** Our experiments show that the compression results of two methods are almost the same while using loss as evaluation is much faster, which means takes less steps in MCTS. Details can be found in Figure 2.2. We can see in remainsize size vs top1 accuracy plot there is no significant difference between two criterions but the steps used are quite distinguishable.

Although the variance of the later metric seems much larger, the actual performance of the later metric is better with similar results but less exploration steps in MCTS. We think the possible two reasons are:

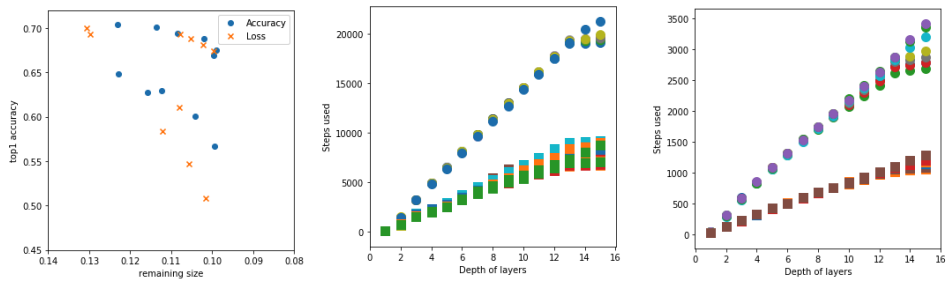



FIGURE 2.2. Left: compression result of VGG16 with two methods. Upper curve comes from quantization and the lower curve comes from pruning. Middle: Steps used in MCTS for VGG16, pruning only. Right: Steps used in MCTS for VGG16, quantization only. Dots are from accuracy and squares are from loss. Each color corresponds to one experiment.

- The actual variance is much smaller than the worse case, since we start from a well trained network.
- The cross-entropy loss is less delusive. The prediction can still be 'accurate' as long as the true label has an advantage against the rest, but cross-entropy loss will favor those whose advantages are larger.

**2.5.3. Different Search Space.** Here the search space means the search tree created with MCTS. There are some hyperparameters to choose for MCTS as well. They are:

- Prior belief $p_i$
- Exploration coefficient $c_{puct}$
- Maximum depth $m$ allowed for MCTS
- Number of explorations $t$ before expansion

For now we only discuss the later two since $m$ alone determines the search tree and $t$ has a more direct impact on the computation resources required. From Figure 2.3 we can see with a larger $m$ the result improves but the same $m$ with a larger $t$ might not.

**2.5.4. Comparison of Results.** In this section we study the performance we can obtain from pruning, quantization, and their combination. For now we do not take retraining into consideration.

To have a comprehensive understanding of the performance, we start from small punishment for remaining size so that the there is almost no loss in accuracy. Then we increase the punishment until the compressed network can do only random guess (accuracy dropping to 0.001).

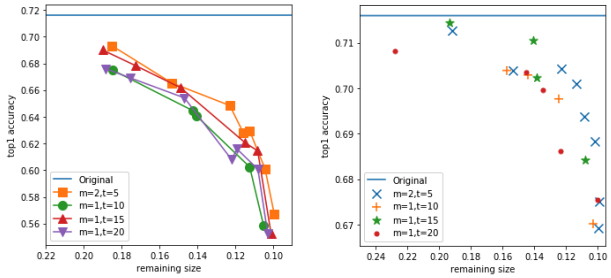



FIGURE 2.3. Left: results for pruning VGG16 with different search space. Right: results for quantizing VGG16 with different search space. We can see that there is a small improvement when $m$ increases from 1 to 2 in pruning but the change of $t$ does not have a clear impact. For quantization again there is no clear difference since the result is already good enough.

Basically we are interested in two questions: To what extend can our method improve the results of unifrom pruning/quantization. And what can we get by combine pruning with quantization. In Figure 2.4 we can see after applying MCTS the accuracy is much higher that the uniform method and In Figure 2.5, we can see using quantization alone is better than using pruning alone, which is also true in Figure 2.4. We also notice that they almost share the same compression limit and by combining two methods together, we are able to bypass that original limit without much further loss in accuracy.

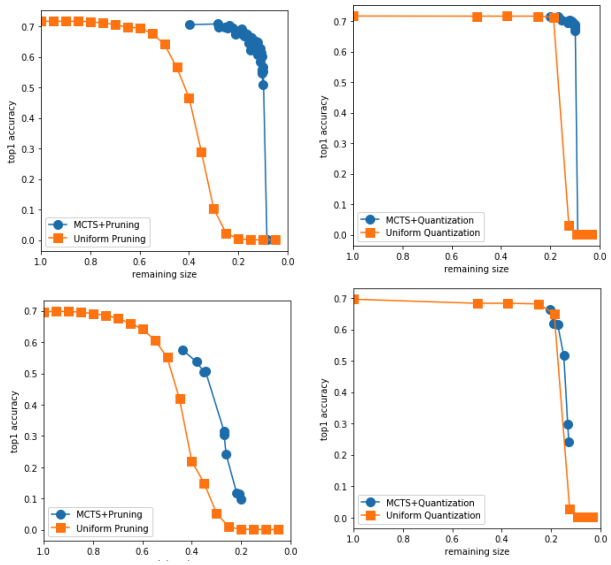



FIGURE 2.4. Accuracy with/without MCTS. Upper Left: Pruning results of VGG16. Upper Right: Quantization results of VGG16. Bottom Left: Pruning results of Resnet18. Bottom Right: Quuntization results of Resnet18.

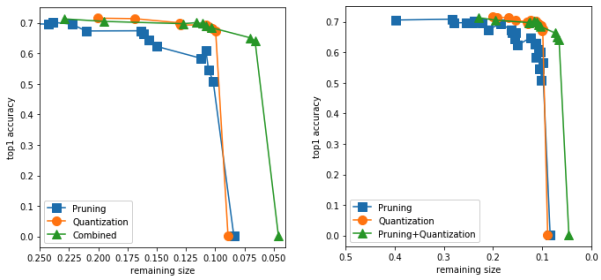



FIGURE 2.5. Accuracy vs Remainning size for different methods. Left: Results of three methods of VGG16. Right: Results of three methods of Resnet18.

**2.5.5. Retraining.** After compression we always retrain our models to reduce the loss in accuracy. In the previous sections we have shown that our methods can bring a huge improvement before retraining process. In this session we will show that the result we can get after retraining is still better than the uniform methods.

Our retrain schedule is to train the compressed network on training set for 3 epoches. The initial learning rate is $10^{-5}$ and after each epoch the learning rate will be divided by 10. We apply this learning schedule to all retraining process. Retraining for pruned layers is to reset the pruned weights to be 0 again after each step. Retraining for quantized layers is to quantize back every 10 steps.

In Table 2.2 we compare the accuracy after retraining for our method and uniform method. We can see in most cases our method outperforms the uniform method instead of the smallest pruning case. We also noticed that 20% is also the compression limit of our method, which means any more compression will make the compressed network give random guess. So we guess that to achieve more compression, some layers are pruned out too much that although it does not affect it's performance during compression, it harms the capability of the network.

| Network | | Pruning | | Quantization | | Original |
|---|---|---|---|---|---|---|
| Size Remain | | 20% | 40% | 5 bits | 7 bits | Accuracy |
| Resnet18 | Uniform | 65.67% | 68.58% | 68.69% | 69.57% | 69.76% |
| | MCTS | 64.10% | 69.06% | 69.27% | 69.67% | |
| Resnet34 | Uniform | 69.95% | 72.33% | 72.26% | 72.87% | 73.30% |
| | MCTS | 69.14% | 72.94% | 72.39% | 72.94% | |

TABLE 2.2. Retraining results for Resnet

One last question of our interest is whether a compressed network would have better performance than another after retraining if the performance before retraining is better. To study this we retrained

several compressed network of Resnet18 and the result is in Figure 2.6. We can see that for each method there is a positive correlation, almost linear relationship, between accuracy before/after retraining.

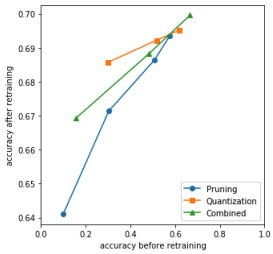



FIGURE 2.6. Accuracy before/after retraining of Resnet18 of three compression methods

In the end we list the performance of most compressed networks using our methods. Again we can see the loss in pruning is much larger than the other two methods. We think the reason is the same as we have discussed, some layers are pruned out too much for the extreme compression.

| | Pruning | | Quantization | | Combined | | Original |
|---|---|---|---|---|---|---|---|
| Network | Size Remain | Accuracy | Size Remain | Accuracy | Size Remain | Accuracy | Accuracy |
| VGG16 | 10.17% | 69.10% | 10.23% | 70.67% | 6.59% | 70.06% | 71.59% |
| VGG19 | 9.40% | 68.67% | 7.58% | 70.55% | 6.93% | 70.63% | 72.38% |
| Resnet18 | 20.04% | 64.10% | 13.30% | 68.57% | 11.76% | 66.93% | 69.76% |
| Resnet34 | 19.98% | 69.14% | 14.00% | 72.27% | 13.03% | 72.25% | 73.30% |

TABLE 2.3. Retraining results for maximum compression

## 2.6. Conclusion

In this paper we proposed two novel ideas for network compression. The approximation of loss help us quickly find a better pruning/quantization than the naive methods and MCTS shows a way to find a good compression configuration without searching an exponentially large space. Besides, this methods can still make improvements like the choice of $\lambda$ and avoiding too much pruning. These are also future questions to study.

CHAPTER 3

# Unsupervised Object Segmentation with Explicit Localization Module

## 3.1. Abstract

In this paper, we propose a novel architecture that iteratively discovers and segments out the objects of a scene based on the image reconstruction quality. Different from other approaches, our model uses an explicit localization module that localizes objects of the scene based on the pixel-level reconstruction qualities at each iteration, where simpler objects tend to be reconstructed better at earlier iterations and thus are segmented out first. We show that our localization module improves the quality of the segmentation, especially on a challenging background.

## 3.2. Introduction

A crucial part of human intelligence is scene understanding, which means decomposing the scene into objects and discovering their relationships. Here we mainly focus on object segmentation, an important method for scene decomposition. In a supervised learning scheme, recent methods [**55**, **63**] mainly rely on convolution neural network (CNN) or its variant to minimize the deviance between generated object masks and the ground truth. In an unsupervised learning scheme, traditional pixel clustering methods [**19**] lead to more sophisticated image clustering methods and loss with CNN [**42**]. Other models achieve state-of-the-art performances by learning object masks and imagine reconstruction [**9**] through one network [**29**] or two separate networks. Inspired by the observation that typical unsupervised models learn to reconstruct relatively simple backgrounds in early epochs and then more complicated details in later epochs, we believe that different parts of images have a different level of reconstruction difficulty. We argue that those higher-level details are the objects, which usually should not share characters with background and thus harder to be reconstructed during the first few epochs.

We approach this problem from pixel-level clustering and iterative object segmentation, similar to MONet [**9**]. We propose a network that removes MONet's attention network and segments objects one-by-one through the reconstruction quality mask. Since MONet's attention network does not rely on a reconstruction image before updating the network parameters, which may lead to inconsistencies between the reconstruction image and attention mask, our model utilizes a reconstruction image that decides which area to focus, which instead leads to a consistent approach that "where the network reconstructs is where it focuses." In our method, the first step segments the background and then later steps "fill in" the image details missing from the "first impression." Objects are considered higher-level details of the scenes that cannot be easily reconstructed by the "first impression" of the scene , i.e. the background. Moreover, a group of pixels should be considered as an object only if its parts move as a whole (except for deformative objects). Thus, objects should be considered through a clear and explicit localization mask.

Our contributions are as follows. We propose a new algorithm that localizes the areas needed for object segmentation by directly measuring reconstruction quality. This coarse-grained estimate of the focused area is then fine-tuned by a Gaussian Mixture Model with very few components that cluster the pixels in that area to obtain a detailed boundary for the object. More importantly, compared with models that rely on network output attention masks, our model has an explicit localization module that guarantees masks of objects are localized. We show that this explicit localization module is necessary for more complicated datasets, such as Montezuma's Revenge, where some objects share similar color but have different modeling difficulties.

### 3.3. Models

Review of MONet is in Section 3.3.1 and our motivation of model design is in Section 3.3.2. Section 3.3.3 discusses how we measure the quality of reconstruction and Section 3.3.4 explains how we conduct the local clustering through GMM. We observe that the reconstruction quality between the background and the objects are adversarial with each other, so we propose a method that mitigates this effect in Section 3.3.5. The overview of our model and algorithm is Section 3.3.6.

**3.3.1. MONet Overview.** As a recent work of unsupervised scene decomposition, MONet is an important framework on which our model is based. The main idea of MONet is to learn to

reconstruct the input image by identifying one object at a time. To achieve this goal, an attention model is trained and outputs an attention mask, which claims to focus on one single object in a given image. A VAE is trained to reconstruct the object covered by this attention mask. Furthermore, VAE also tries to recover the attention mask obtained by the attention model to stabilize the training. The network is tuned so that the "background" is always given attention in the first step.

To be specific, if we use $\mathbf{s}^{(k)}$ to represent the "unexplained ratio" of each pixel after the $k$-th step, the "explained ratio" at $k$-th step as $\mathbf{m}^{(k)}$, and the original image input as $\mathbf{x}$, the idea of MONet can be written as:

$$
\begin{aligned}
\mathbf{s}^{(0)} &= \mathbf{1} \\
\mathbf{s}^{(k)} &= \mathbf{s}^{(k-1)}(1 - \alpha_\phi(\mathbf{x}; \mathbf{s}^{(k-1)})) && 1 \leq k < K \\
\mathbf{m}^{(k)} &= \mathbf{s}^{(k-1)}\alpha_\phi(\mathbf{x}; \mathbf{s}^{(k-1)}) && 1 \leq k < K \\
\mathbf{m}^{(K)} &= 1 - \sum_{k=1}^{K-1} \mathbf{m}^{(k)}
\end{aligned}
$$

where the $\alpha_\phi$ is a trainable attention network parameterized by $\phi$ and $K$ is the total number of segmentation steps. We adopt this framework but use a different method, which we detail in later sub-sections, to find the mask at each step.

**3.3.2. Model Motivation.** The idea of MONet, which is basically to use attention as masks and cover the image step by step, is natural. But in practice, we found that it is very hard to tune the hyperparameters to learn a good attention model. Specifically, it's hard to make sure that the CNN-based attention model finds masks for objects. This motivates us to propose a new method with an explicit nonparametric localization module that helps find objects. To produce better object boundaries, we considered simpler clustering methods like GMM or KNN and used traditional features like color or location. Besides, instead of finding an accurate attention mask for the objects directly, we choose to find a larger area that contains the object inside and split the object from the local area in the second step. The smaller area makes it possible to use a simple clustering algorithm to detect the object in different remaining not-yet-covered parts.

**3.3.3. Reconstruction Quality Estimates and Coarse Grained Attention.** We first measure the quality of the reconstruction by the pixel-wise square error between the reconstruction and the input images. The pixel-wise reconstruction quality, $Q_{i,j}$, is measured by

$$Q_{i,j} = \exp(-\frac{\mathbf{s}||\mathbf{x}_{i,j} - \mathbf{x}^{re}_{i,j}||^2}{2\sigma^2}),$$

where $\mathbf{x}^{re}$ is the reconstruction image and $\mathbf{x}$ is the original image. $\mathbf{x}_{i,j}$ and $\mathbf{x}^{re}_{i,j}$ are RGB vectors at the corresponding pixels determined by $\{i, j\}$. $\sigma$ is a hyperparameter for variance. In order to locate an object roughly by reconstruction quality, both the pixel reconstruction and its neighbor pixels' reconstruction matter. To evaluate that, we used a non-trainable convolution kernel with *weights* $= 1$, *stride* $= 1$, and SAME padding on $\mathbf{Q}$. Relying on non-trainable kernels, we can avoid trivial solutions found by neural networks. The output of this convolution indicates how well the network reconstructs at each local region; partially overlapping regions are allowed. We then find the location with the highest output value, denoted by $(i_c, j_c)$. This pair of coordinates indicates the center of the region with the best reconstruction quality, and our model generates a rough attention $\mathbf{G}$ based on $(i_c, j_c)$. Given our rough assumption that attention has peak 1 in the center and decays gradually, we choose a Butterworth filter **[10]** to model attention in a region. Mathematically, a Butterworth filter is

$$G(r, n, f) = \frac{1}{\sqrt{1 + (\frac{r}{f})^{2n}}}.$$

Here $r$ is the distance from a point to the center and $n, f$ are hyperparameters. With the center point determined by $\mathbf{Q}$, we can easily obtain the local mask by this filter.

In practice we treated the horizontal and veritcal coordinates independently and the Butterworth filter attention $\mathbf{G}$ on pixel $(i, j)$ is

$$\mathbf{G} = G(|i - i_c|, n, f) \cdot G(|j - j_c|, n, f).$$

**3.3.4. GMM.** A Gaussian mixture model (GMM) is capable of finding components when we have samples from a mixture of separate Gaussian distributions. In this segmentation task, it is possible to approximate the distribution of pixels if we treat them as 5-d Gaussian random variables.

The 5 dimensions are RGB channels and two coordinates, denoted as

$$\mathbf{y}_{i,j} = (\mathbf{x}_{i,j}, i, j).$$

For a general GMM, if we know there are $k$ groups and we somehow initialized their means as $\mu^{(1)}, \mu^{(2)}, \cdots, \mu^{(k)}$ and let

$$z_{i,j,k} = P(\mathbf{y}_{i,j} \text{ in class } k), \quad \mathbf{z}_i \in \mathbb{R}^k,$$

we can easily obtain the update rule for GMM:

$$\mu^{(k)} = \frac{\sum_{i,j} z_{i,j,k} \mathbf{y}_{i,j}}{\sum_{i,j} z_{i,j,k}} \quad k \in \{1, 2, \dots\}$$
$$z_{i,j,k} = \frac{\mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}{\sum_k \mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}$$

Here $\mathcal{K}(\cdot, \cdot)$ is a kernel function. In GMM, since we assume each group is a Gaussian distribution, we choose the multi-variate Gaussian density function as our kernel.

For our problem, the pixels in which we are interested might have been partially/totally explained by previous steps. So they should have less/no impact on the clustering process in later steps. Thus we give each pixel a "weight" that indicates its importance. The weight is easy to find: simply use the Butterworth filter weight $\mathbf{G}^{(k)}$:

$$\mu^{(k)} = \frac{\sum_{i,j} z_{i,j,k} w_{i,j} \mathbf{y}_{i,j}}{\sum_{i,j} z_{i,j,k}} w_{i,j} \quad k \in \{1, 2, \dots\}$$
$$z_{i,j,k} = \frac{\mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}{\sum_k \mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}$$

And in practice we found that giving a hard threshold gives clearer segmentation, so we use

$$\mathbf{w} = \mathbf{1}_{\mathbf{G}^{(k)} > 0.5}$$

**3.3.5. Adversary Between background and object reconstruction.** A problem we observe while using reconstruction quality as masks is that the more details the background could obtain through training, the worse the segmentation result will be, because it leaves less room for perfect reconstruction of objects in later iterations. In order to mitigate this problem, we generate a

**Algorithm 3:** $L_{GMM}$

**Result:** Image reconstruction and segmentation masks

Initialize $\mathbf{z}_{i,j,k} = 0.5$ for $k = 1, 2$;

Initialize $\mu_{\mathbf{1}} = \mathbf{0}, \mu_{\mathbf{2}} \sim \mathit{Unif}(0, 1)$;

Input vectors $\mathbf{y}_{i,j}$;

Calculate $\mathbf{G}^{(k)}$ based on the Butterworth filter;

Initialize weights by $\mathbf{w} = \mathbf{1}$ if $\mathbf{G}^{(k)} > 0.5$;

**for** *L iterations* **do**

  Update $\mu^{(k)} = \frac{\sum_{i,j} z_{i,j,k} w_{i,j} \mathbf{y}_{i,j}}{\sum_{i,j} z_{i,k} w_{i,j}}, k = 1, 2$;

  Update variance for each coordinate for each group;

  Update $z_{i,j,k} = \frac{\mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}{\sum_{k=1}^{2} \mathcal{K}(\mathbf{y}_{i,j}, \mu^{(k)})}$

**end**

$I_{i,j} = z_{i,j,2}$;

return $\mathbf{I}$

mask for the background by subtracting all the intermediate objects' attention masks. The portion of the image that should be captured by the background at the first iteration is derived from the input image masked by this background mask. In other words, $\mathbf{m}^{(1)}$ for background is computed by

$$\mathbf{m}^{(1)} = 1 - \sum_{k=2}^{K} \mathbf{m}^{(k)}.$$

**3.3.6. Model Overview.** Our model utilizes VAE [**44**] for simple datasets or an auto-encoder with skip-connection for complicated datasets. We assume there are $K$ objects, including background and our model keeps track of the not-yet-explained areas by a remaining-mask $\mathbf{s} \in [0, 1]^{h \times w}$. As an alternative, we can use stick-breaking process to find just the right number of K. In every iteration, our model tries to reconstruct the unexplained part. We evaluate the reconstruction quality and denote the quality by $\mathbf{Q} \in [0, 1]^{h \times w}$. A basic observation is that VAEs and auto-encoders will learn the representation of often observed items so the reconstruction quality of these objects will get better faster than other areas. Thus, except for the first iteration where the background is found, we can locate the object roughly by finding the area where reconstruction quality is high. Then we look into the area and tell whether each pixel belongs to the object. As for the first iteration, we directly use the quality of reconstruction of each pixel as its explained ratio by the background.

For the following iterations, the image- and remaining-mask are input to the network again for reconstruction. Different from the first iteration, a location-sensitive mask, denoted by $\mathbf{L} \in [0, 1]$, is

derived from the location where the reconstruction quality is the best. Then the remaining-mask is updated again and the next iteration starts. The remaining-mask $\mathbf{s}$ and object component mask $\mathbf{m} \in [0, 1]$ are updated by the formula:

$$\mathbf{s}^{(k)} = \mathbf{s}^{(k-1)}(1 - \mathbf{Q}^{(k)}\mathbf{L}^{(k)})$$
$$\mathbf{m}^{(k)} = \mathbf{s}^{(k-1)}\mathbf{Q}^{(k)}\mathbf{L}^{(k)},$$

where $k \in \{1, 2, \cdots, K\}$ denotes the $k$th iteration. Moreover, when $k$ is 0, $\mathbf{L}_1$ is $\mathbf{1}$ to indicate no location-sensitive mask is applied to the background.

Denote $c$ as input image's channel, $\theta$ as the trainable weights for encoder-decoder, $\phi$ as the constant weights for a convolution layer that evaluates the area reconstruction quality from pixel-wise reconstruction quality mask $\mathbf{Q}$, $\sigma_1$ as the constant variance for pixel-wise reconstruction quality mask, and $\sigma_2$ as the constant variance for location mask $\mathbf{L}$. The algorithm we use is summarized in Algorithm 4 and its flowchart is in Figure 3.1.

**Algorithm 4:** Background and Object Segmentation

**Result:** Image reconstruction and segmentation masks

Initialize a remaining-mask $\mathbf{s}^{(0)}$ as $\mathbf{1}$.;

**for** *K iterations* **do**

  $\mathbf{x}^{re,(k)} = g_\theta(\mathbf{x}, \mathbf{s}^{(k-1)})$ ; `// decode`

  $\mathbf{Q}^{(k)} = \exp(-\frac{\mathbf{s}^{(k-1)} \sum_c (\mathbf{x}_c^{re,(k)} - \mathbf{x_c})^2}{2\sigma_1})$ $c \in \{0, 1, 2\}$; `// decode quality mask`

  with no-gradient:

    $\mathbf{Q}'^{(k)} = Q_{\phi=\mathbf{1}}(\mathbf{s}^{(k-1)}\mathbf{Q}^{(k)})$; `// area decode quality`

    $i, j = \arg\max(\mathbf{Q}^{(k)})$;

    $\mathbf{L}^{(k)} = L_{GMM}(\mathbf{x}, i, j; \sigma_2)$; `// location mask 3`

    $\mathbf{s}^{(k)} = \mathbf{s}^{(k-1)}(1 - \mathbf{Q}^{(k)}\mathbf{L}^{(k)})$;

    $\mathbf{m}^{(k)} = \mathbf{s}^{(k-1)}\mathbf{Q}^{(k)}\mathbf{L}^{(k)}$;

**end**

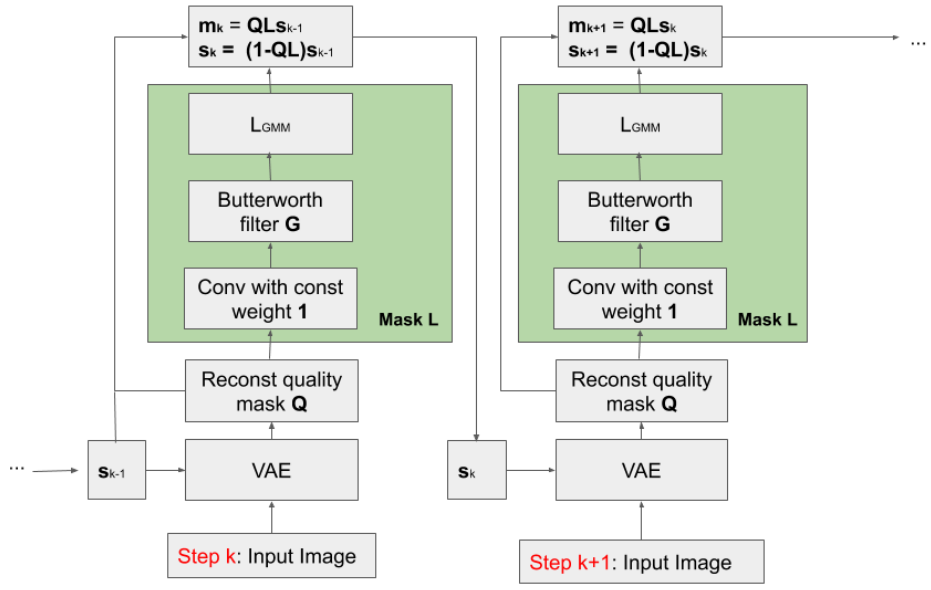



FIGURE 3.1. Our model structure and how model $\mathbf{m}$ and $\mathbf{s}$ are computed.

The only trainable parameter $\theta$ is updated through the following equation:

$$\begin{aligned}\mathbf{Loss} = &\sum_{k=1}^{K} \mathbf{m}^{(k)}(\mathbf{x}^{re,(k)} - \mathbf{x}^{gt,(k)})^2 \\ &+ \beta \sum_{k=1}^{K} (\mathbf{1} - \mathbf{m}^{(k)})(\mathbf{x}^{re,(k)} - \zeta)^2 \\ &+ \gamma \sum_{k=1}^{K} D_{KL}(p(\tilde{\mathbf{z}}^{(k)}|\mathbf{x}, \mathbf{s}^{(k-1)})||p(\tilde{\mathbf{z}}))\end{aligned} \tag{3.1}$$

where $\zeta$ is a constant prior for pixels that are masked out by $\mathbf{m}$, $\beta$ is a hyperparameter that controls the weight of prior loss, and $\gamma$ is the hyperparameter that controls the weight of KL-div for the VAE prior. For auto-encoder $\gamma = 0$, $\tilde{\mathbf{z}}$ is the prior for enbedding of VAE and $\tilde{\mathbf{z}}^{(k)}$ is the embedding in the $k$th step. Lastly, if we use $\mathbf{m}^{(k)}$ as the object mask, then redefining $\mathbf{x}^{re,(k)} := \mathbf{m}^{(k)}\mathbf{x}^{re,(k)}$ for Equ. 3.1 helps, since the reconstruction at this time focuses more on the region found by $\mathbf{m}^{(k)}$. This allows mask $\mathbf{Q}$ to contribute to the loss, but $\mathbf{L}$ is strictly constant.

### 3.4. Related Work

A lot of the recent progress has been made as a result of convolutional neural networks (CNN) and their variants. Many of the progress on object segmentation has been based on supervised learning, where people label the images given their prior knowledge and train the network accordingly. Fully Convolutional Networks [**55**] is a paradigm network architecture for semantic segmentation and more advanced results are achieved by recently with a semi-convolutional operator [**63**].

People also work on unsupervised object segmentation through neural network. Unsupervised object discovery [**37**] with a pre-trained model, such as VGG, is proposed, but it is highly based on the performance of the pre-trained model; the model is not end-to-end unsupervised. An object segmentation method characterizes pixel similarities based on CNN. Recently, a generative adversarial network (GAN) for object segmentation [**12**] is successfully applied to real world dataset. However, this model is based on the network itself to find attention masks, which are good in datasets where the objects are salient and big enough.

The most related work is MONet [**9**], which segments objects iteratively through a scope mask. Its performance relies on the interactions between the attention mask and VAE for reconstruction. A similar idea is also used in IODINE [**29**], where each independent embedding tries to recover an object and its mask for the object. Then all recovered objects are combined with a normalized mask to reconstruct the original image. It uses special techniques to learn the joint posterior embedding to overcome the shortage of VAE, which is only able to learn independent posterior embedding given the input. In practice, tuning the parameters such that the CNN-based attention model masks exactly over objects is very difficult: there is no feedback loop between the reconstruction image and the attention module for every scene-decomposing step in MONet. Therefore, what to decomposed in every step is purely determined by the attention model; it is difficult to guarantee that the attention model masks over a localized region and that region happens to contain an object. Compared to MONet, our model directly computes an attention mask from reconstructed images, and we have an explicit localized module that makes sure our model focuses on local regions.

Lastly, in terms of object discovery through a sequence of frames, a network [**67**] that is based on optical flow can learn moving objects. Neural Expectation Maximization [**30**] proposes to learn embeddings of objects through a sequence of frames through EM and learn the transformation

from frames to embeddings through training. Based on NEM, Relational NEM [**83**] where object relationships are extracted through the embedding and R-NEM achieves better performance than NEM. Object discovery through a sequence of frames provides more information than independent frames. Since our model currently only focuses on object segmentation on images, this direction is interesting future work.

### 3.5. Experiments

We test our model on three different datasets. We mainly focus on the performance on object segmentation.

**Multi-dSprites** [**41**] This dataset has a colored background and a random number of objects. We use 60 000 training samples and 10 000 testing samples. We use the ARI score provided. We conduct an ablation study on this dataset.

**The Category Flower Dataset** [**62**] This dataset is a real-world flower dataset. We use the same data split as provided. We use the sumScoreIoU [**12**].

**Montezuma's Revenge** Montezuma's Revenge OpenAI Gym [**8**] is a game where object discovery plays an important role in hierarchical deep reinforcement learning proposed [**45**] and goal-driven/symbolic planning for reinforcement learning [**57**]. We use 10 000 training samples with a random policy. For testing sets, we use a pre-trained policy where the agent successfully solve the first stage. We manually label 100 samples of nine objects, including the agent, the skull, the rope, the key (may be missing as the agent obtains it), two doors, and three ladders. Since current GAN-based segmentation [**12**] supports segmenting simple scene (one foreground object only), and official implementations of MONet or IODINE are not available online, we only report our AMI score on Montezuma's Revenge as a benchmark result. The AMI score is calculated as in NEM [**30**].

**3.5.1. Results and Discussion.** We summarize our results in Table 3.1. We also analyze the results by different datasets in the following paragraphs.

**Multi-dSprites** Our model so far is not able to achieve as good results as MONet, because this dataset has samples in which objects are partially covered by another object, a situation that GMM cannot handle easily. For ablation study, where the whole location mask **L** is removed and only the reconstruction quality **Q** remains, we found that the segmentation metric provided [**41**]

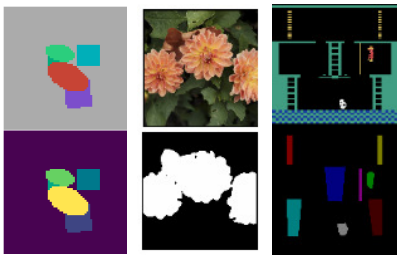


FIGURE 3.2. Sample ground truth images (top) and segmentation masks (bottom) for Multi-dSprites (left), Category Flower Dataset (middle), and Montezuma's Revenge (right).

TABLE 3.1. Benchmark results for different dataset and models. The higher the better for all measurements.

| **Dataset** | **Multi-dSprites** | **Category Flower Dataset** | **Montezuma's Revenge** |
|---|---|---|---|
| MONet | 0.904±0.008 | — | — |
| IODINE | 0.767±0.056 | — | — |
| ReDO | — | 0.764±0.012 | — |
| Ours | 0.621±0.004 | 0.632±0.001 | 0.375±0.002 |

does not apply to our case, because images with one object leads to NAN as our model incorrectly decomposes that object to different object slots. We see similar poor performance when we leave the GMM module and **L** only without training the network.

**Category Flower Dataset** This is a real world dataset where the background is more complicated and thus requires more sophisticated background removal techniques. Similar to the results achieved when IODONE [**29**] applies their model to a real-world dataset, we observe a noticeable gap between our model and the benchmark, because the assumption of 5D vector in GMM module, RGB and x-y coordinates, is too simple for real world dataset.

**Montezuma's Revenge** Our model can achieve an AMI score of 0.375. Figure 3.3 is a reconstruction of the objects in a typical frame. Figure A.1 in Appendix A.2 provides more reconstruction results. Most of the important objects that can serve as goals are found by our model, including the three ladders, skull, the doors, and even the keys. More strikingly, our model successfully finds objects, such as the ladders at the bottom, that share the same color as the walls. Since the wall is easier to reconstruct, it is captured by the background, whereas the ladders, with their more complicated details, are left to be captured by later object slots. Through this experiment,

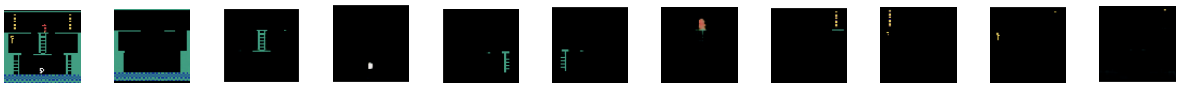


FIGURE 3.3. Randomly picked sample of reconstructed image by objects for Montezuma's Revenge, where objects locations are provided. The first figure is input image. Each important object has been found by our model, and its location visually matches with our provided, showing that our GMM module does find the objects.

we confirm our argument that objects can be extracted iteratively with different reconstruction difficulties, which we believe can be a new method for object discovery.

TABLE 3.2. Location found by objects and their corresponding location.Matching between objects visual position with the x-y coordinates provided shows that our model successfully find the objects.

| **Objects** | **y-axis** | **x-axis** |
|---|---|---|
| back ground | — | — |
| middle Ladder | 0.365 | 0.497 |
| skull | 0.798 | 0.489 |
| bottom right ladder | 0.595 | 0.827 |
| bottom left ladder | 0.595 | 0.180 |
| agent | 0.231 | 0.531 |
| right door | 0.282 | 0.880 |
| left door | 0.090 | 0.137 |
| key | 0.378 | 0.067 |
| — | 0.026 | 0.887 |

**3.5.2. Object Location Extractor.** One of the benefits of our model is that the objects' location is automatically extracted. We train our model in only 9 epochs and extract the locations of the objects through the coordinate means calculated with the GMM. A random reconstructed frame is provided as Figure 3.3 and the corresponding objects and their location is provided in Table 3.2. The location is shown as scaled from 0–1 in $y$ (vertical axis) and $x$ (horizontal axis). $(0, 0)$ is the top left of the image. Most of the objects are found by our model, visually near the locations provided.

## 3.6. Further Smoothing on Multinomial Segmentation

**3.6.1. Optimization Problem Setup.** TestSuppose that $Q$ is a $K$-channel image that is a multinomial distribution, $\sum_k Q_{ijk} = 1$, $Q_{ijk} \geq 0$. Then we would like to search for another

multinomial distribution, $X$ that is similar to $Q$ but is segmented in the sense that it is piece-wise constant over different regions. The similarity measure we will use is the KL divergence,

$$D(Q||X) = -\sum_{i,j,k} Q_{ijk} \log \frac{X_{ijk}}{Q_{ijk}}.$$

and then we would like to force regions of the image to have the same X values, so we introduce the following grouped total variation penalty,

$$\rho(X) = \lambda \left( \sum_{i,j} \sqrt{\sum_k (X_{ijk} - X_{i+1,jk})^2} + \sqrt{\sum_k (X_{ijk} - X_{i,j+1,k})^2} \right)$$

Define the following operators,

$$(\nabla_1 X)_{ijk} = X_{ijk} - X_{i+1,jk},$$

$$(\nabla_2 X)_{ijk} = X_{ijk} - X_{i,j+1,k}.$$

Define the following mixed norm,

$$\|Z\|_\rho = \sum_{ij} \|Z_{ij.}\|_2$$

so that

$$\rho(X) = \lambda(\|\nabla_1 X\|_\rho + \|\nabla_2 X\|_\rho),$$

where $X^\top$ indicates the transpose in the $j$ coordinate. Introduce the auxiliary parameters $Z_1 = \nabla X, Z_2 = \nabla X^\top$. Then we introduce the augmented Lagrangian with dual parameters $U_1, U_2$,

$$L(X, Z, U) = D(Q||X) + \rho(Z) + \frac{\rho}{2} \left( \|\nabla_1 X - Z_1 - \rho^{-1} U_1\|^2 + \|\nabla_2 X - Z_2 - \rho^{-1} U_2\|^2 \right),$$

where $\rho(Z) = \lambda(\|Z_1\|_\rho + \|Z_2\|_\rho)$ is an abuse of notation.

$U$ and $X$ are updated with gradient steps, with $X$ being projected onto the simplex by I-projection, which is simply

$$X_{ijk} \leftarrow \frac{(X_{ijk})_+}{\sum_k (X_{ijk})_+},$$

where $x_+ = \max\{x, 0\}$.

The solution to $Z$ is a grouped soft thresholding, but we can also replace it with a hard thresholding (this is the flexibility of the ADMM approach),

$$Z_{1ij.} \leftarrow A_{ij.} \cdot 1\{\|A_{ij.}\|_2 > \lambda/\rho\},$$

where $A = \nabla_1 X - \rho^{-1} U_1$, and similarly for $Z_2$.

**3.6.2. Application on Image Segmentation.** In our scenario, $Q$ is the direct output from our algorithm and $X$ would be a further smoothed segmentation. As a segmentation, we would expect that a single pixel should mainly belong to one class instead of evenly distributed over several classes. In order to force $X$ away from uniform distribution, we shall add

$$\lambda_2 \sum_{i,j,k} \frac{1}{K} \log(K \cdot X_{ijk})$$

into the loss. This is the negetive KL-Divergence between X and uniform distribution. By minimizing loss we are somehow make this KL-Divergence large. This only changes how the gradient of X is calculated. Now the new Lagrangian is

$$\begin{aligned} L(X, Z, U) = D(Q||X) + \rho(Z) + \lambda_2 \sum_{i,j,k} \frac{1}{K} \log(K \cdot X_{ijk}) \\ + \frac{\rho}{2} \left( \|\nabla_1 X - Z_1 - \rho^{-1} U_1\|^2 + \|\nabla_2 X - Z_2 - \rho^{-1} U_2\|^2 \right), \end{aligned}$$

However, with this optimization objective, the previous mentioned ADMM approach gets unstable in the update of $X$. A more stable and basic method is to optimize another set of variables $L$ which satisfies

$$softmax(L_{ij.}) = X_{ij.}$$

This would require one more layer of backpropobation. But There is no interaction between $L_{i_1 j_1 .}$ and $L_{i_2 j_2 .}$ unless $i_1 = i_2$ and $j_1 = j_2$. So we can backpropobate the gradient only channel-wise.

For clarity we ignore the old notations and only consider $X = softmax(L), L = (L_1, L_2, \cdots, L_K)$ and assume we know $\frac{\partial \mathcal{L}}{\partial X_j}$ for all $j \in \{1, 2, \cdots K\}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial L_i} &= \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial X_j} \cdot \frac{\partial X_j}{\partial L_i} \\
&= \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial X_j} \cdot \frac{\partial \frac{\exp(L_j)}{\sum \exp(L)}}{\partial L_i} \\
&= \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial X_j} \cdot \frac{\exp(L_j) \cdot 1(i=j) \cdot \sum(\exp(L)) - \exp(L_i) \cdot \exp(L_j)}{(\sum \exp(L))^2} \\
&= \sum_{j=1}^{K} \frac{\partial \mathcal{L}}{\partial X_j} \cdot (1(i=j) \cdot X_i - \frac{\exp(L_i) \cdot \exp(L_j)}{(\sum \exp(L))^2}) \\
&= X_i(\frac{\partial \mathcal{L}}{\partial X_i} - \sum_{j=1}^{K} X_j \cdot \frac{\partial \mathcal{L}}{\partial X_j})
\end{aligned}
$$

In practice within 10 iterations $X$ will be smooth enough. We found that the segmentation metrics improved by $2 \sim 5\%$ and time comsumption increased by around $8 \sim 10\%$

### 3.7. Discussion and future work

In this paper, we propose a new unsupervised object segmentation algorithm with an explicit localization module. The localization module serves as an attention mask derived from the reconstruction quality. By iteratively segmenting the objects, our method finds objects one-by-one, filling in the details of the image missing from previous iterations. We empirically confirm our beliefs that those details correspond to objects.

As for future work, it is promising to extend this work in a sequence of frames, a context where objects are mostly consistent between frames. We also notice that GMM does not always lead to good results and more sophisticated (local) segmentation algorithms could possibly lead to better results. Lastly, our model still has a decent amount of prior knowledge injected through hyperparameters. Making the model simpler should be helpful in future work.

CHAPTER 4

# A Graph-Based Dataset Similarity Metric

## 4.1. abstract

A number of metrics are available for computing similarity and distance between a pair of samples from same or different datasets. However, finding a notion of how similar two datasets are in distribution remains an open question. This paper presents a novel framework to evaluate the similarity between datasets based on a graph structure and statistical hypothesis testing. We also provide theoretical guarantees to explain the behavior of our proposed similarity metric. The proposed framework provides great flexibility for modeling similarity between datasets as it is agnostic to data type, model and task. Application to a specific data type, model, or task only requires an appropriate sample-level similarity metric provided by domain experts. Through various experiments, we show that our framework is sensitive to perturbations in data, and is able to tell datasets apart based on their semantic similarity. Our results also suggest a strong correlation between our metric and domain adaptation hardness across different datasets. Further improvements to the framework are also discussed and left as future directions.

## 4.2. Introduction

Finding a notion of similarity or distance between datasets is important for better understanding of domain adaptation, model generalization, evaluating the quality of generative models, and other aspects of machine learning. This work focuses on similarity in the context of Natural Language Processing (NLP). Various sample-level similarity metrics have been developed in the NLP space. These include basic $n$-gram overlap scores like ROUGE [**51**] and BLEU [**66**], as well as distance metrics based on word embeddings, like Word Mover's Distance [**46**] and Sentence Mover's Similarity [**16**]. More recently, with the introduction of powerful BERT models [**21**], the quality of sentence embeddings has considerably improved. These embeddings have been used to calculate

similarity scores, such as BERTScore [**92**], Sentence BERT [**70**] and BLEURT [**76**]. However, these similarity scores have not been generalized to establish similarity between datasets.

It is important to distinguish between sample-level and dataset-level similarity measures. Similar datasets do not necessarily have highly similar samples. For similarity between datasets, a more reasonable criterion is to check whether elements of the datasets have similar support and density in the space of samples. The similarity between samples would naturally be correlated to the metric in this space. This criterion would sometimes find similar datasets without having similar samples, for example, two datasets with uniform but sparse distribution over a large support. Establishing a notion of similarity between two datasets are in distribution remains an open problem.

In this paper, we propose a novel similarity framework for datasets. It combines a graph structure and statistical hypothesis testing, and makes no assumption on the feature space, like Euclidean or isotropic. We provide theoretical justifications for why our metric provides a similarity measure between datasets. We perform multiple experiments on various tasks and several standard NLP datasets to show the sensitivity of our similarity metric to different modifications of datasets and their usage. We also discuss further improvements that can be made to our proposed framework.

### 4.3. Related Work

Prior work in establishing similarity between datasets has addressed this problem from different aspects. Traditional methods have relied on first or second order statistics for similarity measures. Samples are first represented as embeddings and these embeddings are then used to calculate various statistics to obtain their distance in Euclidean space. The main drawback of these methods is the heavy reliance on assumptions of the underlying distributions. We study some of these methods in the context of domain adaptation and provide those results in A.3.2 in Supplementary Material.

Another metric for similarity is **Maximum Mean Discrepancy(MMD)**, which maximizes the difference in expectation of a function over two distributions, where the function is also maximized over a certain class of functions. More details of this metric can be found in [**79**, **80**]. **Optimal transport (OT)** between two datasets has also been proposed as a similarity measure. This method estimates the OT distance of the distribution of embedded datasets. Various approximation methods

of distributions, embedding algorithms and Wasserstein-type distances have been studied in this space. More details of this line of work can be found in [**2**, **18**, **25**, **61**].

Recently, [**1**] proposed to use two separate metrics, $P_\alpha$ and $R_\beta$, to understand how two distributions, especially their central areas, overlap with each other. The method trains an encoder to fit one dataset so that its encoding, $\mathcal{D}_1$, falls into a sphere $\mathcal{S}_1$ with a pre-selected center $c_1$. The same encoder encodes the other dataset for its encoding $\mathcal{D}_2$ with calculated center $c_2 = \frac{\sum_{i=1}^{n_2} x_{2i}}{n_2}$. Then, it defines

$$P_\alpha = \mathbb{P}(x \in \mathcal{S}_1^\alpha | x \in \mathcal{D}_2), R_\beta = \mathbb{P}(x \in \mathcal{S}_2^\beta | x \in \mathcal{D}_1), IP_\alpha = \int_{\alpha=0}^{1} P_\alpha d\alpha, IR_\beta = \int_{\beta=0}^{1} R_\beta d\beta$$

Here $\mathcal{S}_1^\alpha$ means the smallest sphere centered at $c_1$ that covers at least $\alpha$ of $\mathcal{D}_1$, and $\mathcal{S}_2^\beta$ is the smallest sphere centered at $c_2$ that covers at least $\beta$ of $\mathcal{D}_2$. In their paper, $\mathcal{D}_1$ is a real dataset and $\mathcal{D}_2$ is a generated dataset with a generator. These two metrics are used to evaluate the quality of generation. That paper also developed a simple rule to distinguish between authentic and unauthentic generated samples. For any generation $x \in \mathcal{D}_2$, find

$$x_{i^*} = \text{argmin}_{x_{1i} \in \mathcal{D}_1} ||x_{1i} - x||$$

$$x_{j^*} = \text{argmin}_{x_{ij} \in \mathcal{D}_1 / \{x_{1i^*}\}} ||x_{1j} - x_{1i^*}||$$

Generation $x$ will be classified as authentic if and only if $||x_{1i^*} - x_{1j^*}|| < ||x - x_{1i^*}||$. However, some implicit assumptions required to justify the methods may be violated in practice. The simple rule used to examine the quality of generation is based on two contradicting assumptions. More details of these contradicting assumptions are provided in Supplementary material A.3.3 and A.3.4.

## 4.4. Methodology

**4.4.1. Graphbased Framework.** We start with some basic notations. Define $\mathcal{X}$ as the space of all possible samples of interest, or the feature space, which is not necessarily parametric. The first and second dataset are denoted by $\mathcal{D}_1 = \{x_{11}, x_{12}, \cdots, x_{1N_1}\} \subseteq \mathcal{X}$ and $\mathcal{D}_2 = \{x_{21}, x_{22}, \cdots, x_{2N_2}\} \subseteq \mathcal{X}$, respectively. Here $N_1$ and $N_2$ are the size of two datasets. In order to measure similarity, we need a tractable similarity function $s : (\mathcal{X} \times \mathcal{X}) \mapsto [0, 1]$ to measure the probability that two samples are found *similar*. The function can also be an indicator function: $s : (\mathcal{X} \times \mathcal{X}) \mapsto \{0, 1\}$.

Next, define a graph $\mathcal{G} = (\mathcal{E}, \mathcal{V})$ where each vertex corresponds to a sample $\mathcal{V} = \mathcal{D}_1 \cup \mathcal{D}_2$. Note that this is not a strict union of sets. There is one unique vertex for each of the samples that are identical in $\mathcal{D}_1$ and $\mathcal{D}_2$. For any two vertices $x_i,\ x_j$, there will be an edge connecting them with probability $s(x_i, x_j)$. Therefore, $\mathcal{E} = \{(x_i, x_j) \mid p_{ij} \leq s(x_i, x_j), 1 \leq i, j \leq N_1 + N_2, p_{ij} \overset{i.i.d.}{\sim} \mathcal{U}(0,1)\}$.

We can now calculate the similarity score for two datasets based on the neighborhood for all vertices with the following steps.

(1) For vertex $x_i$, define the neighborhoods as vertices that are directly connected with it, $\mathcal{N}_{x_i} = \{x | x \in \mathcal{V}, (x_i, x) \in \mathcal{E}\}$, and further define $n_1 = |\{x \in \mathcal{N}_{x_i} | x \in \mathcal{D}_1\}|$ and $n_2 = |\{x \in \mathcal{N}_{x_i} | x \in \mathcal{D}_2\}|$.

(2) Define $p = \mathbb{P}(x \in \mathcal{D}_1 | x \in \mathcal{N}_{x_i})$ and let $\hat{p} = \frac{n_1}{n_1+n_2}$ be its estimate. Calculate the $p$-value for the following hypothesis test for every vertex $x_i$ This will give us two sets of $p$-values for two datasets: $\{p_{11}, p_{12}, \cdots, p_{1N_1}\}$ for $\mathcal{D}_1$ and $\{p_{21}, p_{22}, \cdots, p_{2N_2}\}$ for $\mathcal{D}_2$.

$$H_0 : p = \frac{N_1}{N_1 + N_2} \longleftrightarrow H_1 : p \neq \frac{N_1}{N_1 + N_2} \tag{4.1}$$

The $p$-values can be calculated as follows

$$p\text{-value} = \mathbb{P}\left( |\mathcal{N}(0,1)| > \frac{\sqrt{n_1 + n_2}|\hat{p} - p_0|}{\sqrt{p_0(1-p_0)}} \right), p_0 = \frac{N_1}{N_1 + N_2} \tag{4.2}$$

However, when $n_1 + n_2$ is very large, $p$-value will be almost 0 with a small difference between $\hat{p}$ and $p_0$. This property is harmful as it will give a zero similarity score to any two large enough datasets. To prevent our method from being too sensitive, we set an upper bound $c$ for $n_1 + n_2$ and set $c = 100$ in simulation studies and experiments on real datasets. Now the $p$-values are calculated as

$$p\text{-value} = \mathbb{P}\left( |\mathcal{N}(0,1)| > \frac{\sqrt{\min(c, n_1 + n_2)}|\hat{p} - p_0|}{\sqrt{p_0(1-p_0)}} \right), p_0 = \frac{N_1}{N_1 + N_2} \tag{4.3}$$

Note that with Equation 4.3, it's no longer a '$p$-value', but we keep calling it $p$-value for convenience. Clearly we cannot do any inference with these $p$-values. But it does not hurt as we are not interested in the real $p$-values, or the conclusions of the hypothesis tests in

Equation 4.1. Our goal is to find a measure of the similarity of datasets and Equation 4.3 serves as a non-linear transformation that maps $\hat{p}$ into $[0, 1]$.

(3) Calculate average $p$-value, $\bar{p}_1$ and $\bar{p}_2$, for both datasets and calculate their harmonic average as the final similarity score for these two datasets. We use harmonic average because two datasets can be considered 'similar' only if both $\bar{p}_1$ and $\bar{p}_2$ are high.

$$\bar{p}_1 = \frac{\sum_{i=1}^{N_1} p_{1i}}{N_1},\ \bar{p}_2 = \frac{\sum_{i=1}^{N_2} p_{2i}}{N_2},\ \text{similarity score } \bar{p} = \frac{2}{\frac{1}{\bar{p}_1} + \frac{1}{\bar{p}_2}} \tag{4.4}$$

**4.4.2. Theoretical Guarantees.**

LEMMA 4.1. *For hypothesis test in equation 4.1, under $H_0$, the p-value distribution converges to $\mathcal{U}(0, 1)$.*

LEMMA 4.2. *For random variables $X \sim Binom(N, p)$ and $Y \sim Binom(M, p)$, $X$ and $Y$ are independent. Further we define $n = X + Y$. Then for any fixed value of $n$ we have*

$$lim_{N\to+\infty, M\to+\infty} \frac{P(X = k|X + Y = n)}{b(k, n, \frac{N}{M+N})} \rightrightarrows 1$$

THEOREM 4.1. *For any specific sample $x \in \mathcal{X}$, build its neighborhood $\mathcal{N}_x$ as described in 4.4.1. If for any sample $x' \in \mathcal{D}_1 \cup \mathcal{D}_2$, it has probability $p$ of being in $\mathcal{N}_x$, then for any fixed neighborhood size, the expectation of p-value will converge to* 0.5.

COROLLARY 4.1. *With the same settings as in Theorem 4.1 and the probability no longer being a constant but following a prior distribution $F$, i.e., $\mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_1 \cup \mathcal{D}_2) \sim F$, the conclusion of Theorem 4.1 still holds.*

Theorem 4.1 states that for samples with a small neighborhood, its $p$-value will converge to $\mathcal{U}(0, 1)$ and an expected value of 0.5 if the distributions of two datasets are the same. Besides, for different samples in $\mathcal{D}_1 \cup \mathcal{D}_2$, they do not share a common probability $p$ of becoming $x$'s neighborhood but we can still apply Theorem 4.1. To better understand this, we can treat samples in $\mathcal{D}_1$ or $\mathcal{D}_2$ as i.i.d. samples from two priors. Corollary 4.1 then explains why they still converge. Theorem 4.1 and Corollary 4.1 together show convergence for $n_1 + n_2 \leq c$. For large neighborhoods, we have

THEOREM 4.2. *For any specific sample $x \in \mathcal{X}$, build its neighborhood $\mathcal{N}_x$ as described in 4.4.1, if there exists $\epsilon_0 \in (0, 1]$ s.t. $\frac{N_2}{N_1} \in [\epsilon_0, \frac{1}{\epsilon_0}]$, we have $\frac{n_1}{n_1+n_2} \cdot \frac{N_1p_1+N_2p_2}{N_1p_1} \xrightarrow{a.s.} 1$.*

Like Theorem 4.1, Theorem 4.2 can also be generalized to priors.

COROLLARY 4.2. *With the same settings as in Theorem 4.2, and $p_1$ and $p_2$ now distributed as priors $F_1$ and $F_2$, respectively, the conclusion of Theorem 4.2 still holds. All we have to do is to replace $p_1$ and $p_2$ with $\mathbb{E}[F_1]$ and $\mathbb{E}[F_2]$.*

THEOREM 4.3. *For any sample $x \in \mathcal{X}$, build its neighborhoods with the steps described in 4.4.1, and calculate their p-values for the hypothesis test described in 4.4.1. If the two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ have a bounded size ratio and have the same distribution over $\mathcal{X}$, then with large enough datasets and a pre-selected similarity function $s(x_1, x_2)$, all p-values will converge to 1 in probability.*

All proofs are provided in A.3.1 in Supplementary Material.

Theorems 4.2 and 4.3 show that for large neighbors, $\hat{p} = \frac{n_1}{n_1+n_2}$ will be arbitrarily close to $\frac{N_1p_1}{N_1p_1+N_2p_2}$. This value will further degenerate to the probability in our null hypothesis $\frac{N_1}{N_1+N_2}$ when $\mathcal{D}_1$ and $\mathcal{D}_2$ share the same distribution over $\mathcal{X}$. Therefore, similar distributions of datasets will lead to higher $p$-values, and thus a higher similarity score.

However, there is some randomness in our metric, as $n_1$ and $n_2$ are themselves random. Although as our later experiments show that the variance is small, it does make two identical datasets have a similarity score smaller than 1. This is not a limitation as it also reflects how *confident* we are in saying that the two datasets are similar. While identical datasets may come from identical distribution, they may also be produced by pure chance. When the number of samples is small, two different distributions may have similar or even identical samples. This scenario is less likely as the number of samples increases. This is the main difference between Theorem 4.1, and Theorems 4.2 and 4.3. Theorem 4.1 states that the $p$-value follows a uniform distribution when neighborhood size is small. Theorems 4.2 and 4.3 establish that the $p$-value would converge to 1 for large neighbors. It is also straightforward to circumvent this issue by replacing $n_1$ and $n_2$ with $\mathbb{E}[n_1]$ and $\mathbb{E}[n_2]$ in equation 4.3.

The most important benefit is that our metric does not require any properties of feature space $\mathcal{X}$. Existing methods like **[2]** and **[1]** require some notion of 'distance' on the sample space, and

treat it as Euclidean space. We have found that this assumption may be unreliable (see A.3.3 in Supplementary Material for more details). Our method, however, does not make any assumptions of the sample space. We only require different distributions on $\mathcal{X}$ remain different after they are mapped by $s(x_1, x_2)$.

## 4.5. Detailed Explanation for Equation 4.3

**4.5.1. What our adjustment does.** The upper bound of $n_1 + n_2$ in equation 4.3 might look unnatural and here we would explain what problems it can solve and how it is beneficial.

From Theorem 4.2 we know that $\hat{p} = \frac{n_1}{n_1+n_2}$ can be arbitrarily close to $\frac{N_1 p_1}{N_1 p_1 + N_2 p_2}$, where $p_1 = \mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_1)$ and $p_2 = \mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_2)$. When two datasets do come from the same distribution, we should have $p_1 = p_2$ whatever our choice of $x$ is and the null hypothesis would also be true as we always have

$$
\begin{aligned}
p &= \mathbb{P}(x' \in \mathcal{D}_1 | x' \in \mathcal{N}_x) \\
&= \frac{\mathbb{P}(x' \in \mathcal{D}_1, x' \in \mathcal{N}_x)}{\mathbb{P}(x' \in \mathcal{D}_1, x' \in \mathcal{N}_x) + \mathbb{P}(x' \in \mathcal{D}_2, x' \in \mathcal{N}_x)} \\
&= \frac{\mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_1)\mathbb{P}(x' \in \mathcal{D}_1)}{\mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_1)\mathbb{P}(x' \in \mathcal{D}_1) + \mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_2)\mathbb{P}(x' \in \mathcal{D}_2)} \\
&= \frac{p_1 \cdot \frac{N_1}{N_1+N_2}}{p_1 \cdot \frac{N_1}{N_1+N_2} + p_2 \cdot \frac{N_1}{N_2+N_2}} \\
&= \frac{N_1 p_1}{N_1 p_1 + N_2 p_2}
\end{aligned}
$$

This value equals to $\frac{N_1}{N_1+N_2}$ when $p_1 = p_2$

When two datasets come from the same distribution, $p$-value from equation 4.2 will approximately follow $\mathcal{N}(0, 1)$ with large $n_1$ and $n_2$, and the variance of $\hat{p}$, $\frac{p_0(1-p_0)}{\sqrt{n_1+n_2}}$, decreases to almost 0.

But when two datasets come from different distributions, for at least some samples $x \in \mathcal{D}_1 \cup \mathcal{D}_2$, we will have $p_1 \neq p_2$ and $\frac{N_1 p_1}{N_1 p_1 + N_2 p_2} \neq \frac{N_1}{N_1+N_2}$. Now with large $n_1$ and $n_2$, $p$-value from equation 4.2 will quickly degenerate to a point mass at 0 and become meaningless when comparing different datasets. Furthermore, the similarity score will decrease as dataset size increases, which is not a good property for a similarity metric as you can make two datasets less similar by simply increasing their size.

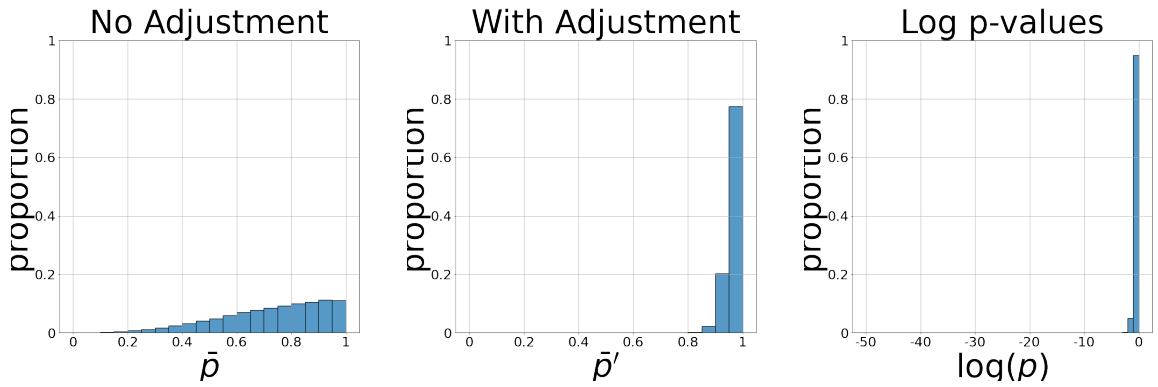
With our modification in equation 4.3, once $n_1 + n_2 > c$, the similarity score only depends on $p_1$ and $p_2$, which is determined by the distribution of datasets. This modification makes the similarity score change only when the distribution of datasets change and be robust to other irrelevant factors such as number of samples in datasets. From simulation results, we can have a better understanding of the effect.

**4.5.2. Simulation.** Another method to deal with vanishing $p$-values is to take log of $p$-values. The log of the $p$-value was used to construct a grouped p-value for false discovery rate control in sequential selection procedures in [**31**]. In simulation, we found the problem of log $p$-value is underflow: when $p$-value lower than some threshold, we cannot find its exact value computational and it will become 0. So we can still change the similarity value by only changing the dataset sizes. In the following simulations, we will show how three methods behave under different situations.

4.5.2.1. *Distribution of p-values.* We first calculate similarity scores between several normal distributions and the standard normal distribution. For each normal distribution $\mathcal{N}(\mu, \sigma^2)$, we sample $N$ samples from it, and then another $N$ samples from $\mathcal{N}(0, 1)$ and calculate the similarity score for these two datasets.

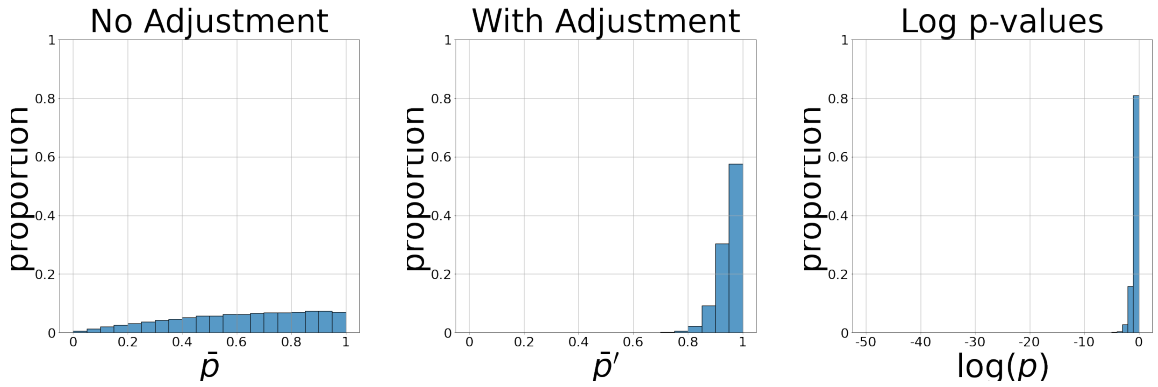
Results can be found from Figure 4.1 to 4.28. **Three columns means different processing of $p$-values**. From left to right, **'No Adjustment'** means the vanilla $p$-value from Equation 4.2, **'With Adjustment'** means the $p$-values are from Equation 4.3, and **Log-$p$-values** if they are $\log(p - \text{value})$. You can also find them in the subcaptions as well. **We calculated 7 pairs similarity for each method**. **The distributions and size of datasets can be found in the main captions**. We can see that only with the adjustments in Equation 4.3, the distributions do not degenerate to a point mass at 0. When use log $p$-values or without any adjustment, the bar on 0 is dominant for when two distributions are not close w.r.t KL divergence. . We use $\bar{p}$ to represent the similarity scores with no adjustment, $\bar{p}'$ for similarity scores with adjustment, and $\log(p)$ for log $p$-values. To avoid calculating $\log(0)$, we replace 0 $p$-values with 1e-20 in simulation.

4.5.2.2. *Dataset Size's Effect.* Next, we visualize how similarity score changes for different KL divergence as well as different dataset sizes. We still use the simulation process above and calculate the similarity. In Figure 4.29 we can see that without the adjustment, similarity can change with



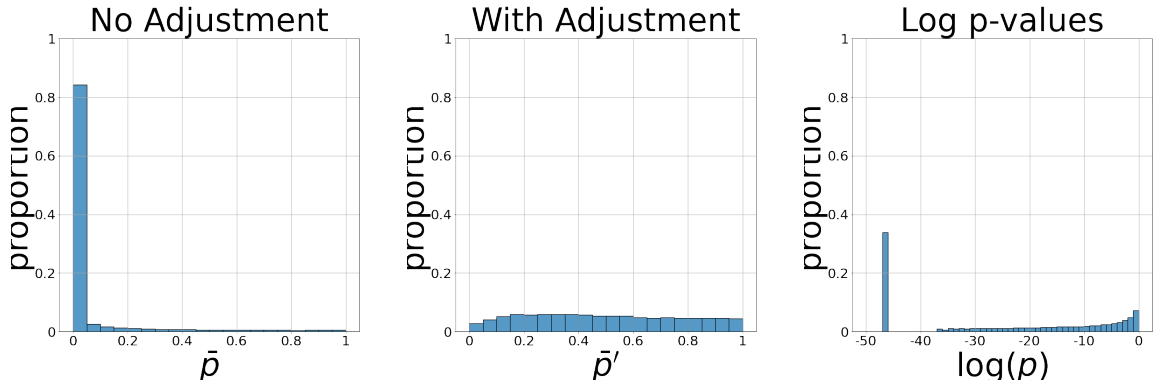

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.1. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



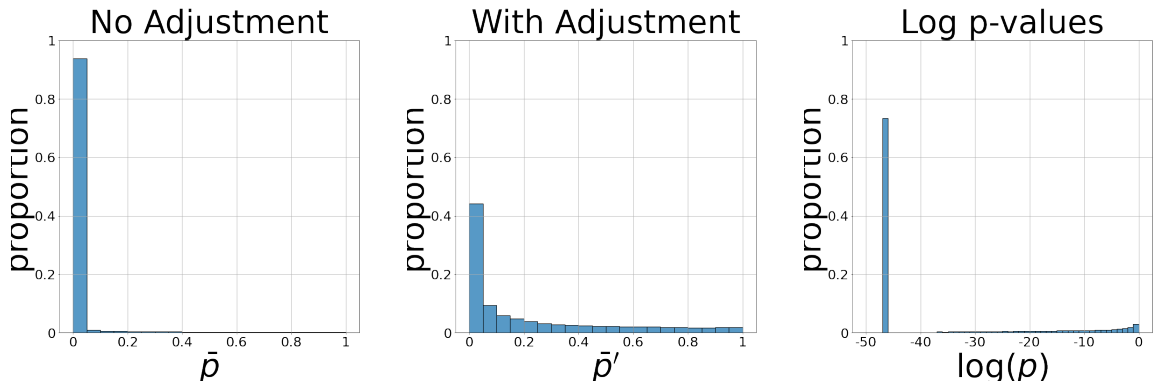

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.2. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



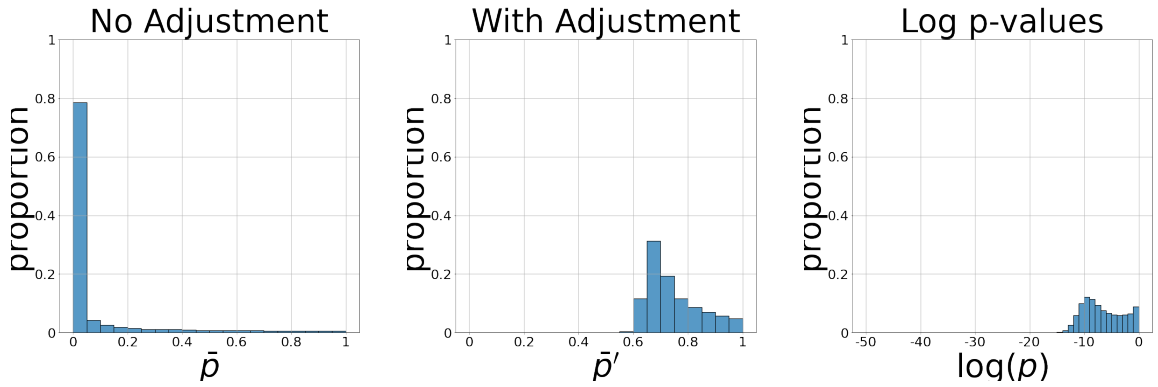

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.3. $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



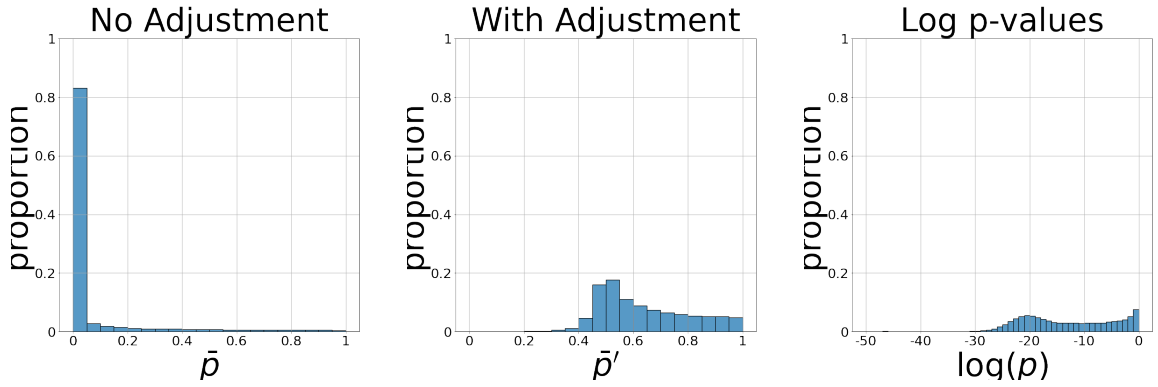

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.4. $\mathcal{N}(0,1)$ and $\mathcal{N}(2,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



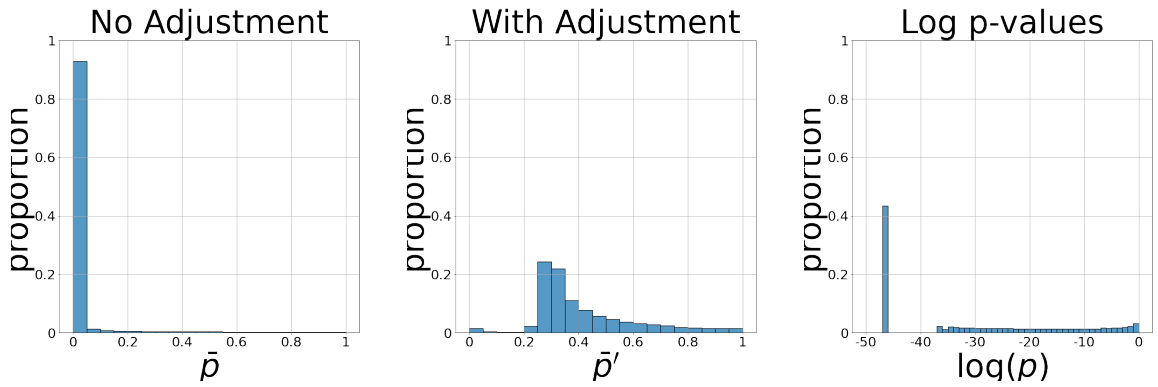

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.5. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



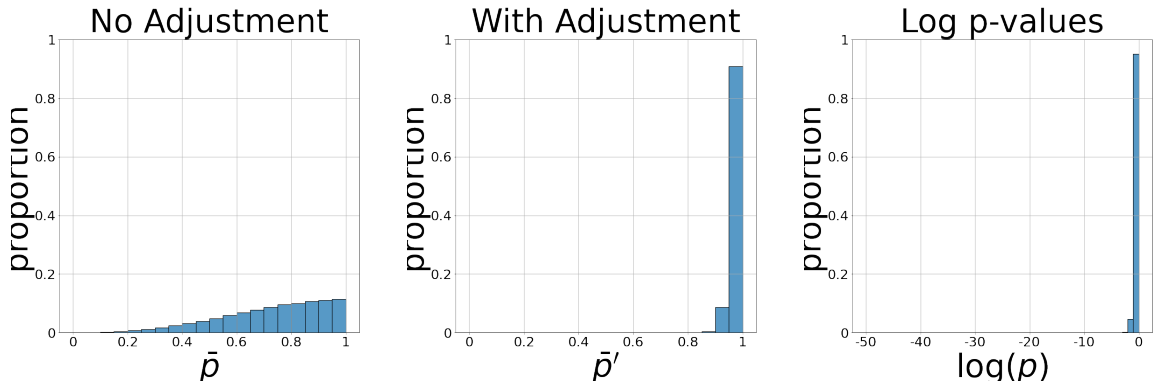

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.6. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



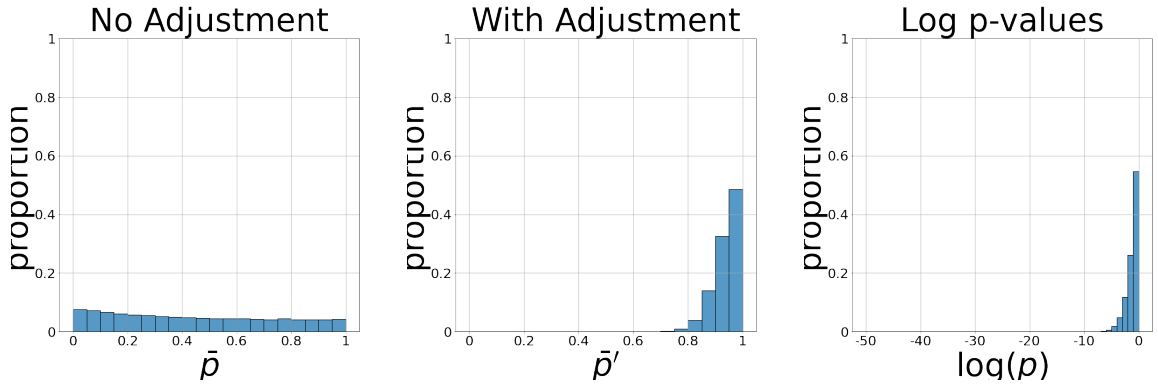

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.7. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,2)$, $\mathcal{N}_1 = \mathcal{N}_2 = 5000$



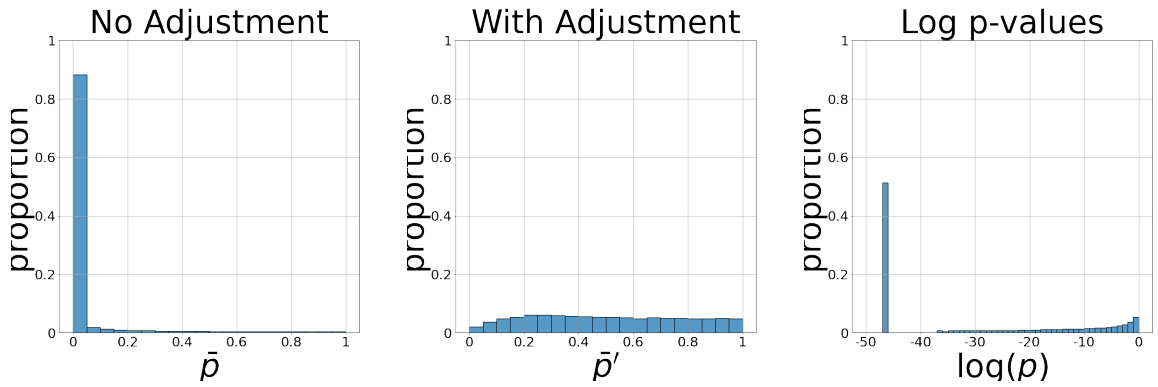

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.8. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



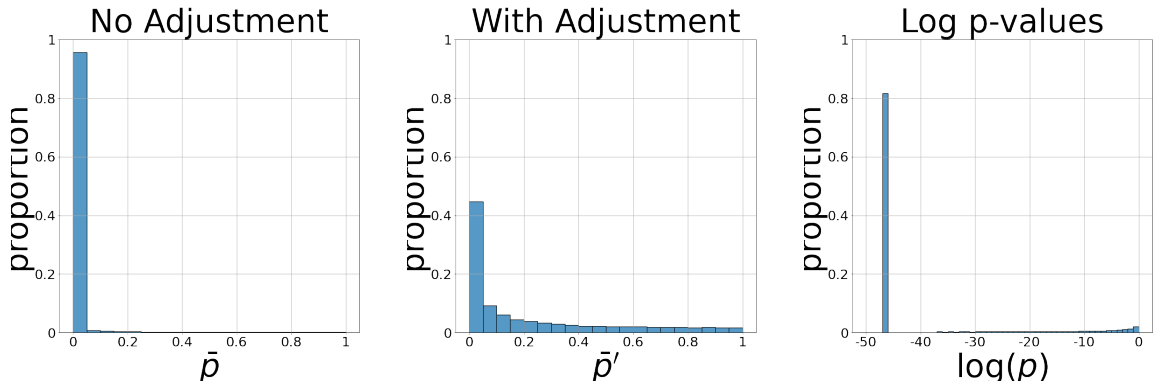

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.9. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



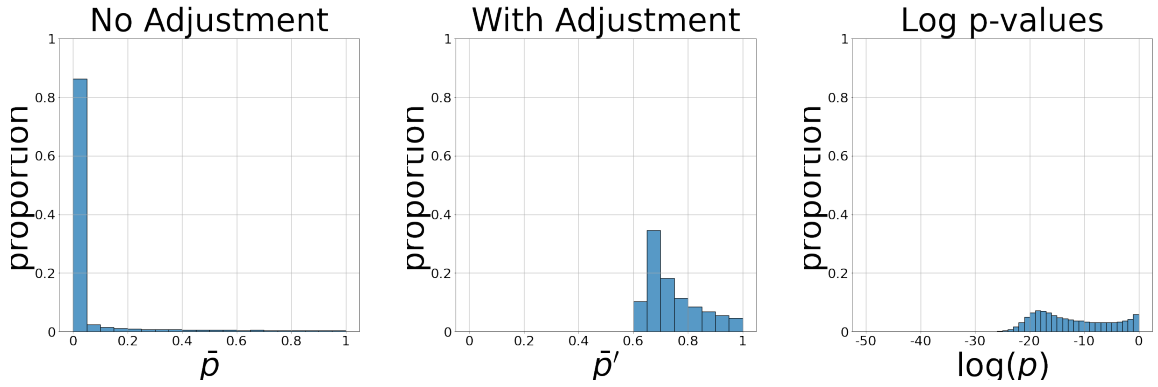

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.10. $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



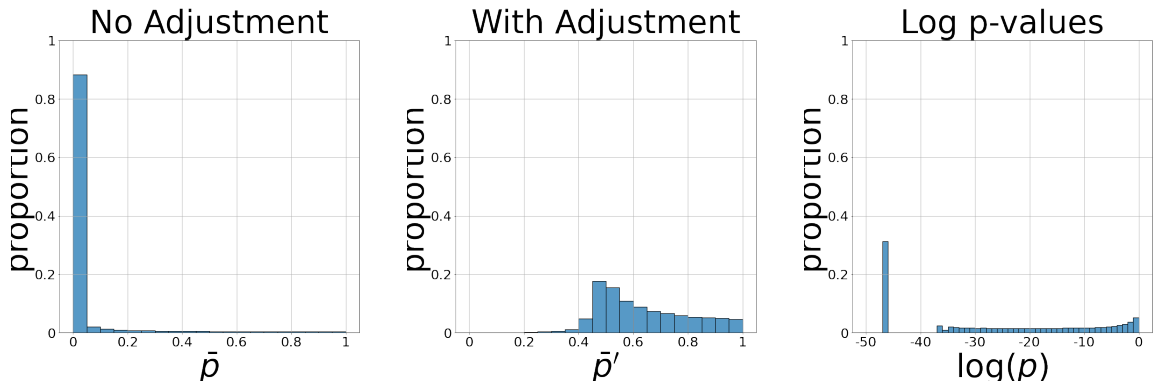

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.11. $\mathcal{N}(0,1)$ and $\mathcal{N}(2,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



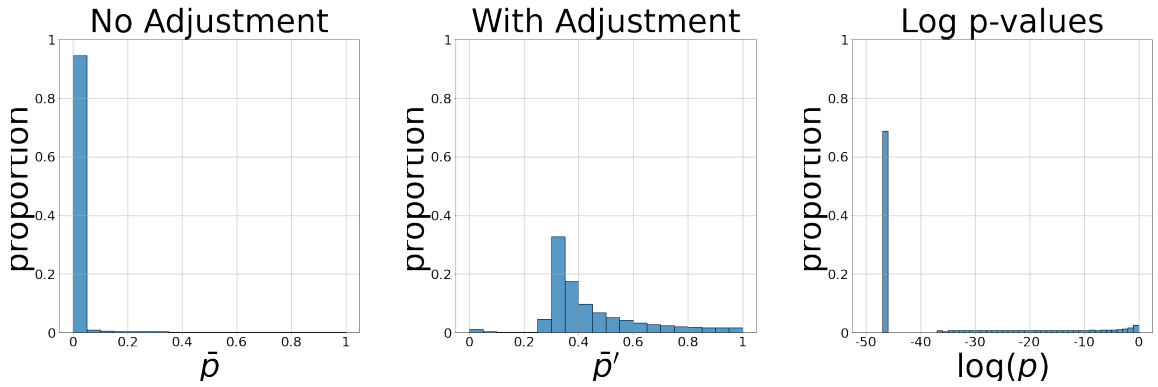

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.12. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



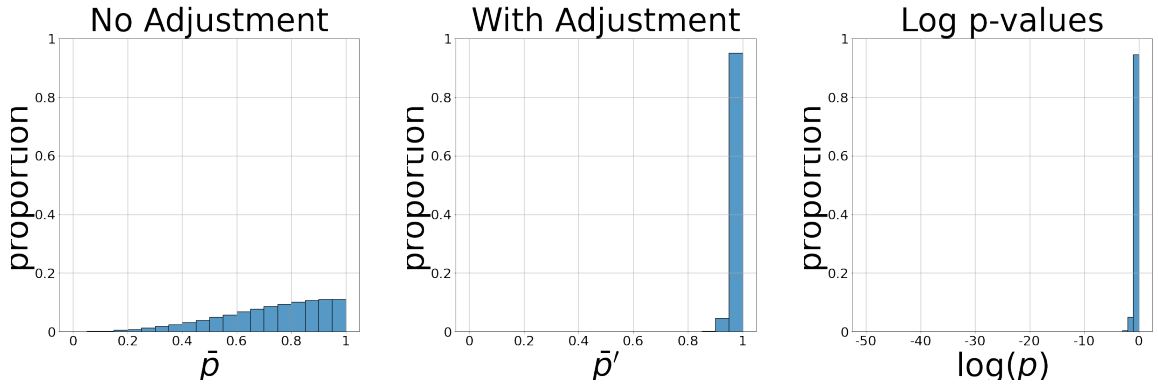

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.13. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



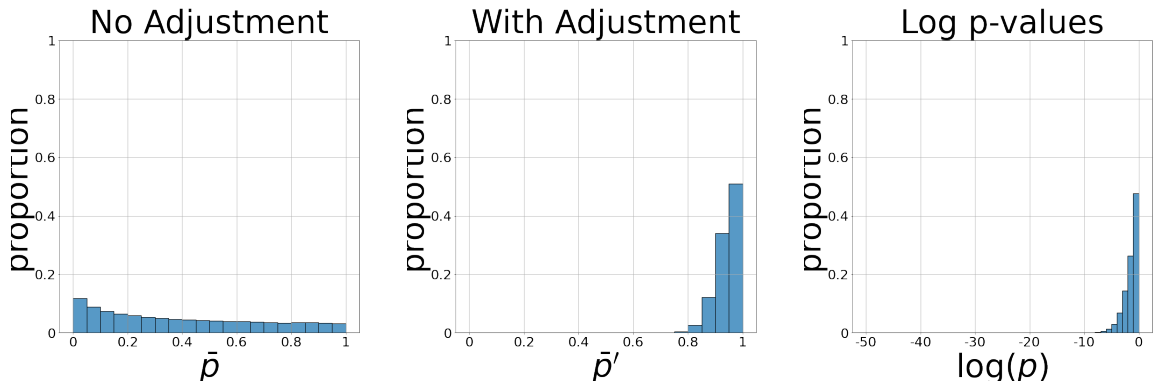

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.14. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,2)$, $\mathcal{N}_1 = \mathcal{N}_2 = 10000$



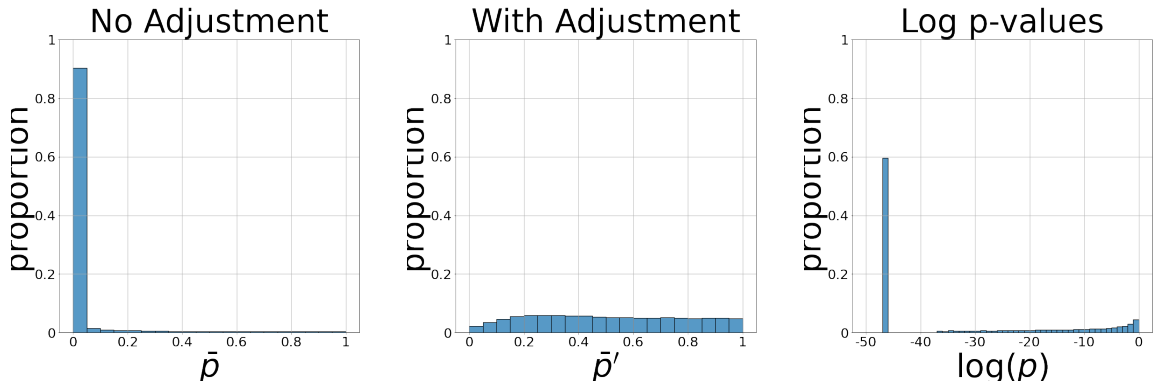

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.15. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



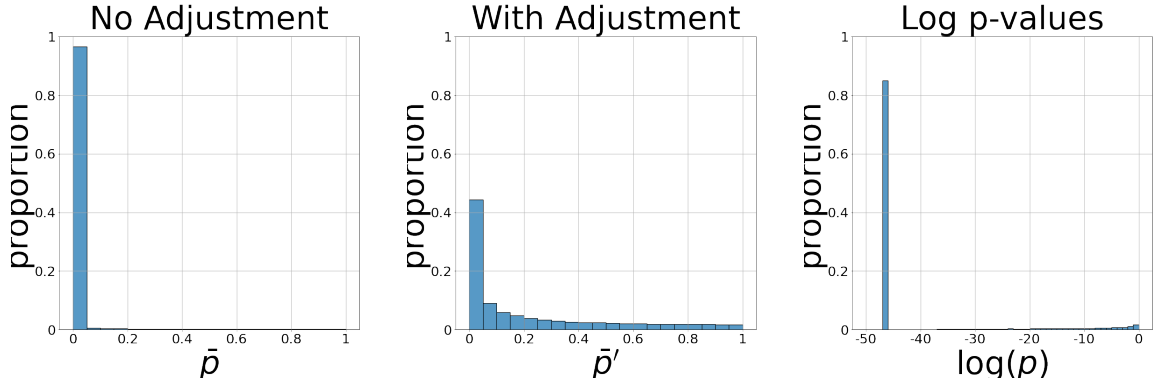

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.16. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



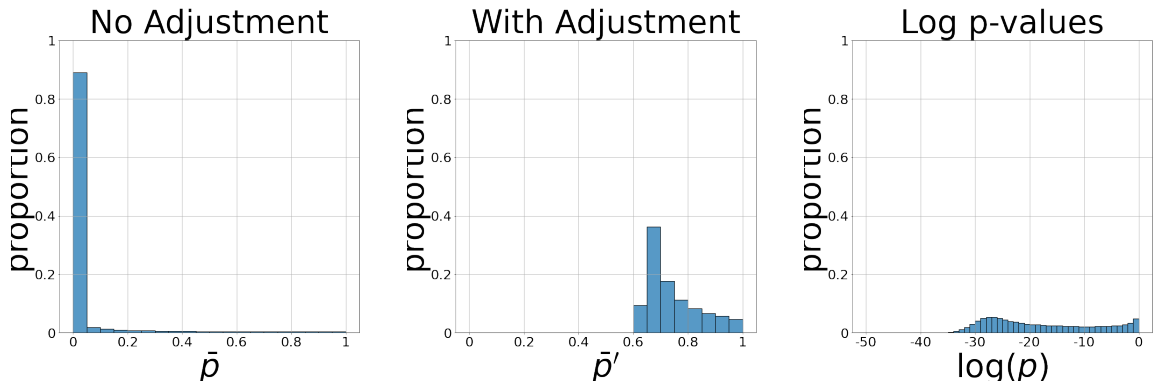

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.17. $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



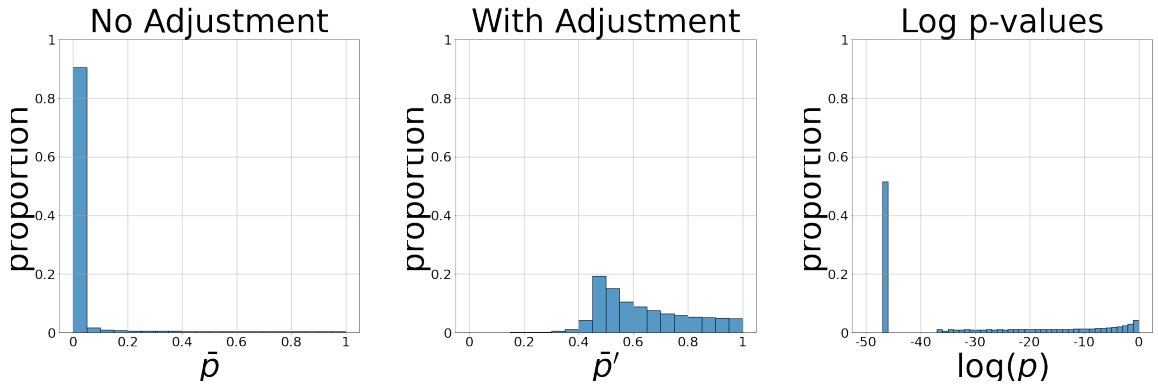

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.18. $\mathcal{N}(0,1)$ and $\mathcal{N}(2,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



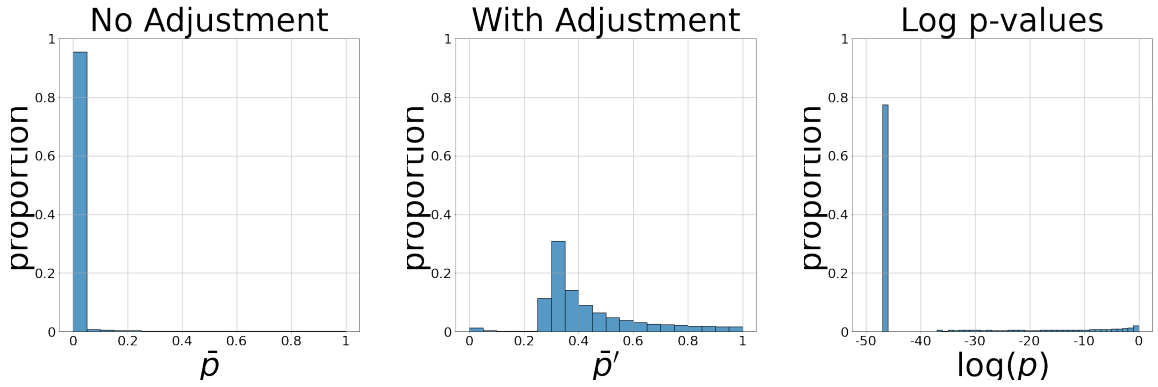

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.19. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



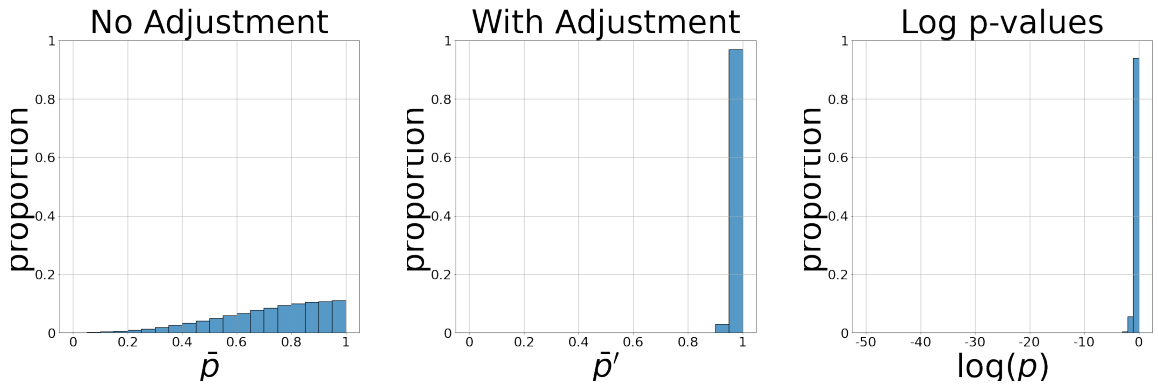

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.20. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



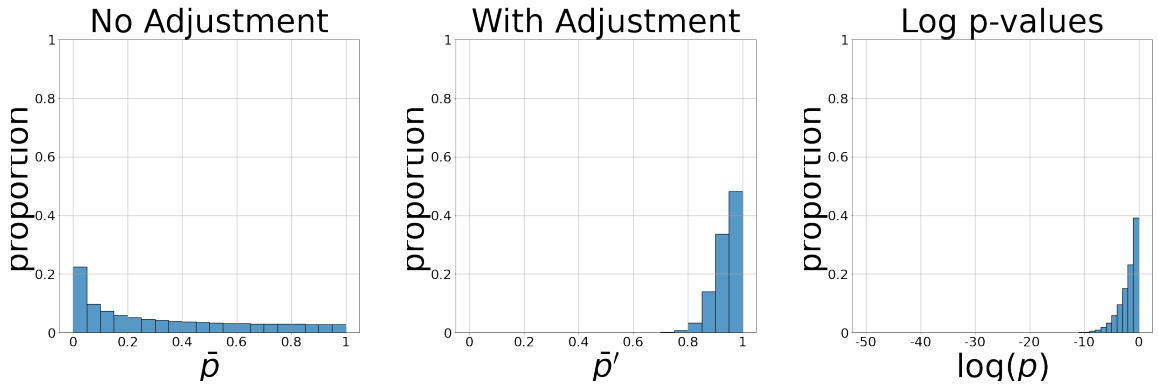

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.21. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,2)$, $\mathcal{N}_1 = \mathcal{N}_2 = 15000$



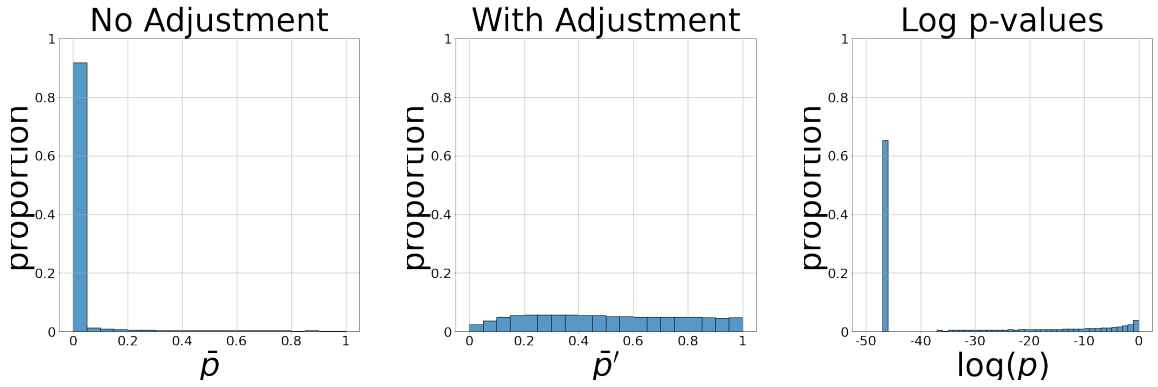

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.22. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$



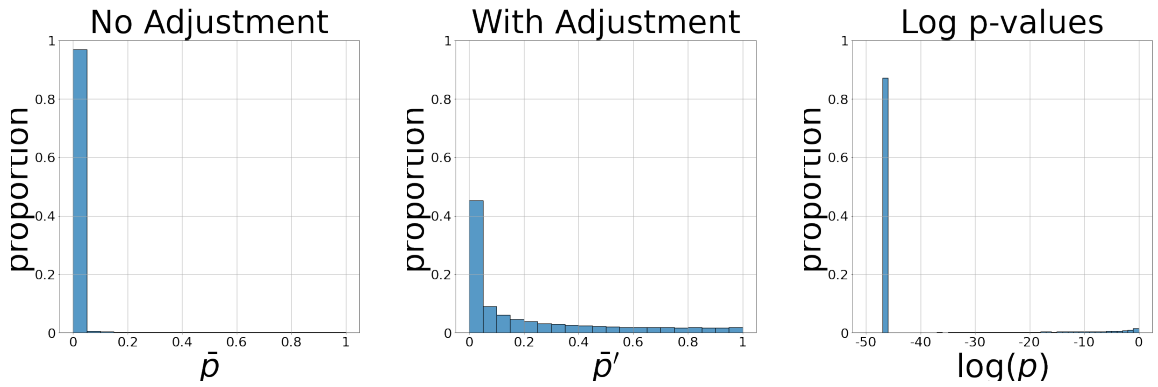

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.23. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$



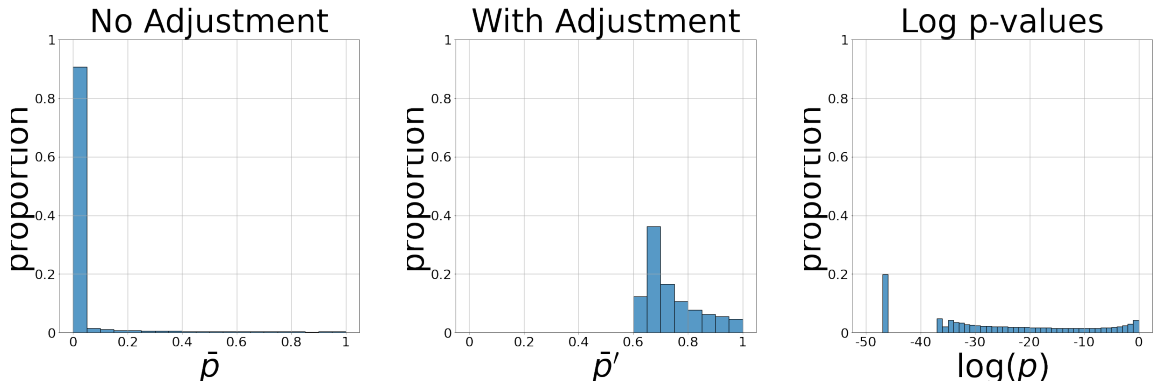

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.24. $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$



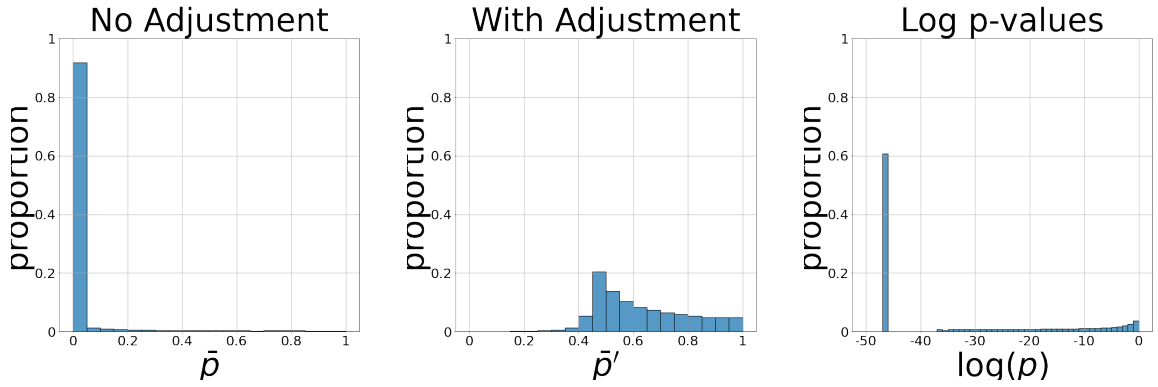

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.25. $\mathcal{N}(0,1)$ and $\mathcal{N}(2,1)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$



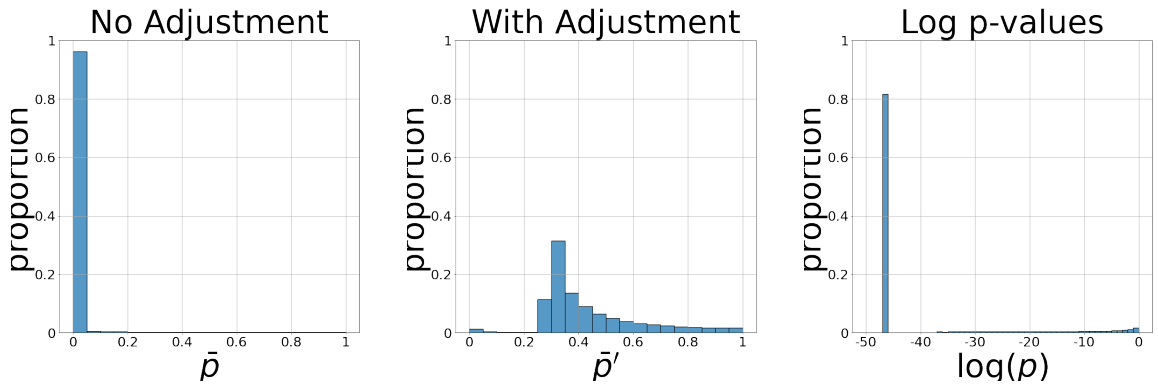

(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.26. $\mathcal{N}(0,1)$ and $\mathcal{N}(0,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$




(A) No Adjustment (B) With Adjustment (C) Log $p$-value

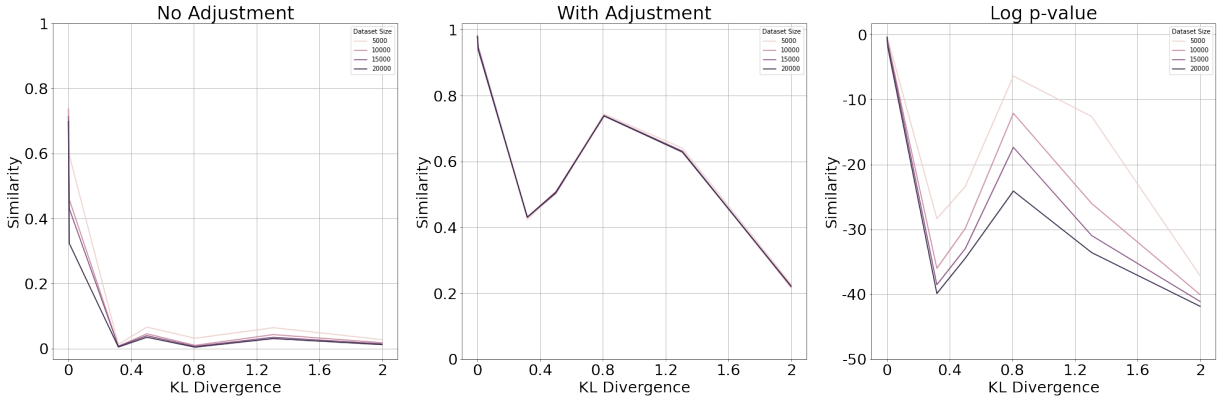
FIGURE 4.27. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,0.5)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$




(A) No Adjustment (B) With Adjustment (C) Log $p$-value

FIGURE 4.28. $\mathcal{N}(0,1)$ and $\mathcal{N}(0.5,2)$, $\mathcal{N}_1 = \mathcal{N}_2 = 20000$

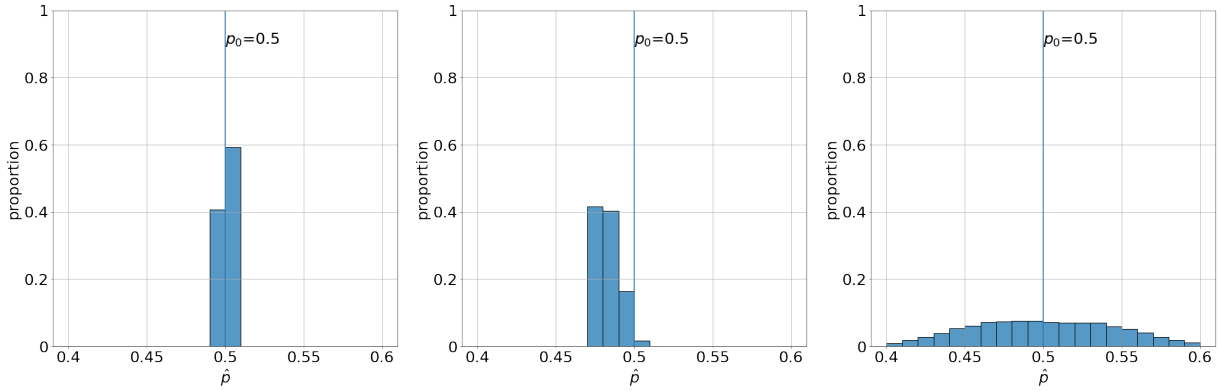
simply change the dataset sizes. And with the adjustment in Equation 4.3, it only changes when KL Divergence changes, i.e. when the dataset's distribution changes.



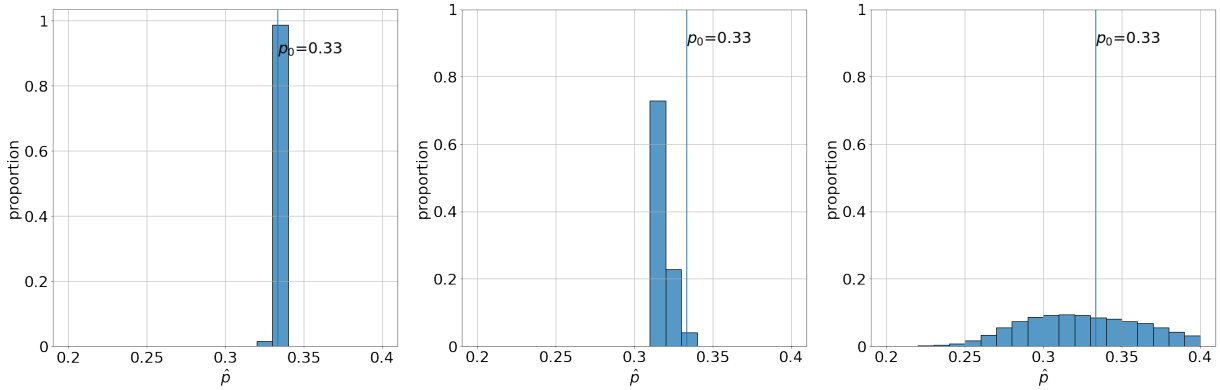

(A) No Adjustment (B) With Adjustment (C) log $p$-vlaues

FIGURE 4.29. Change of Similarity Scores w.r.t KL Divergence and Dataset sizes

**4.5.3. Distribution of $\hat{p}$.** We are also interested in understanding empirically how the distribution of $\hat{p}$ changes. Clearly the distribution are affected by both the distributions of two datasets being compared, and the sizes of these two datasets. To study this, we randomly generate $N_1 = 10000$ samples from $\mathcal{N}(0,1)$ as $\mathcal{D}_1$ and generate $\mathcal{D}_2$ with three distributions: $\mathcal{N}(0,1), \mathcal{N}(0,0.5)$ and $\mathcal{N}(1,1)$. We also test $N_2 = 10000$ and $N_2 = 20000$ for each of them. The results are in Figure 4.30.




(A) $\mathcal{D}_2 \sim \mathcal{N}(0,1), N_2 = 10000$ (B) $\mathcal{D}_2 \sim \mathcal{N}(0,0.5), N_2 = 10000$ (C) $\mathcal{D}_2 \sim \mathcal{N}(1,1), N_2 = 10000$




(D) $\mathcal{D}_2 \sim \mathcal{N}(0,1), N_2 = 20000$ (E) $\mathcal{D}_2 \sim \mathcal{N}(0,0.5), N_2 = 20000$ (F) $\mathcal{D}_2 \sim \mathcal{N}(1,1), N_2 = 20000$

FIGURE 4.30. Distribution of $\hat{p}$

**4.5.4. Other Possible Methods.** As we have shown in Lemma 4.2 the conditional distribution of $n_1$ given $n_1+n_2$ is a hyper-geometric distribution. Actually, the data can fit into a $2\times2$ contingency table like Table 4.1

TABLE 4.1. $2 \times 2$ Contingency Table of $\mathcal{N}_{x_i}$

| | from $\mathcal{D}_1$ | from $\mathcal{D}_2$ |
|---|---|---|
| in $\mathcal{N}_{x_i}$ | $n_1$ | $n_2$ |
| not in $\mathcal{N}_{x_i}$ | $N_1 - n_1$ | $N_2 - n_2$ |

And we should test if two columns are independent with fisher's exact test, or we can use the $\chi^2$ test

If we use fisher's exact test, there would be two problems: **First**, how to properly define the other tail at the opposite direction and calculate the probability of it. **Second**, fisher's exact test will lead to an overall time complexity of $\mathcal{O}(n^3)$, where $\mathcal{O}(n^2)$ is required for all pairwise connection

probabilities and for each pair, the CDF of hypergeometric distribution takes $\mathcal{O}(n)$ in the worse case (an extremely dense graph). On the other hand if we use $\chi^2$ test, we cannot guarantee it's assumptions are met, like a large sample size or a somehow balanced distribution over the four blocks. So we choose not to take these routes.

### 4.6. Experiments

We conduct several experiments to show how metric captures similarity, its sensitivity to detect small differences in datasets, and its applications in downstream tasks.

**4.6.1. Datasets.** We use several publicly available text datasets to study the performance of our similarity framework. The datasets are:

**Imdb Review** dataset comprises 50000 reviews of movies from users of Imdb. Reviews are labeled as positive and negative. The dataset is split into a training set and a test set, each containing 12500 positive reviews and 12500 negative reviews.

**OpinRank** dataset comprises reviews for cars and hotels. There are 41748 car reviews for 30 models and 254749 hotels reviews for 10 different cities.

**SST5** dataset consists of 11855 movie reviews of five classes: very negative, negative, neutral, positive, and very positive. The dataset also provides a train/dev/test split. A simplified version of SST5 is **SST2**, which discards the neutral class and combines the other 4 classes into 2 classes: positive and negative. SST2 has a total of 9613 reviews.

**Airline Comments** dataset comprises comments on airlines collected from Twitter, which is available on Kaggle[1]. We discard its neutral comments, which leaves 2363 positive comments and 9178 negative ones.

**Amazon Product Review** dataset is collected by **[54]** with around 2000 reviews for each of 16 classes of products. This dataset is a subset of **[5]**.

**Noisy Imdb** dataset is another version of **Imdb Review** dataset available on Kaggle [2]. This dataset is cleaned and tokenized by Kaggle and is therefore, slightly different from the original dataset.

[1] `https://www.kaggle.com/crowdflower/twitter-airline-sentiment`
[2] `https://www.kaggle.com/c/word2vec-nlp-tutorial`

**4.6.2. Our Choices of Sample-Level Similarity.** As mentioned earlier, our framework works for any similarity function $s$. We have used two different similarity functions to study the effectiveness and robustness of our proposed framework.

- **Embeddings**: For each sample we get its BERT encoding on [CLS] token, with pretrained BERT weights provided by huggingface [**87**]. We then define

$$s(x_i, x_j) = \frac{u - \max(\min(d, u), l)}{u - l}, \quad d = ||x_i - x_j||$$

  Here $u > l$ are pre-selected hyperparameters to make sure that different pairs have different probabilities of similarity. In all our experiments, we use $l = 5,\ u = 14$.

- **Local Sensitive Hashing (LSH)**: LSH first finds all $n$-grams from the dataset, and randomly permutes them $r \cdot b$ times to generate integer codes with length $r \cdot b$ for each sample. Then every $r$ digits of code are used as hashing keys to create buckets. Samples that share common buckets will be neighborhoods [**48**]. For LSH, we have

$$s_n(x_i, x_j) = 1 - (1 - (\text{jaccard}_n(x_i, x_j))^r)^b$$

  Here $n$ denotes the corresponding $n$-gram and $\text{jaccard}_n(x_i, x_j)$ denotes the Jaccard similarity between $x_i$ and $x_j$. In the following experiments, unless specified, we use $n = 2, r = 1, b = 150$ for Imdb, use $n = 2, r = 1, b = 50$ for OpinRank and SST5.

- The hyper-parameters are chosen such that most sample pairs have a moderate (0.1 to 0.9) probability of being neighbors. This would help the final ratio and $p$-value to reflect the difference. We manually verified the Jaccard similarity or $L_2$ distance distribution for several random samples against two datasets. It is observed in our experiments that the distributions are always stable and it is straightforward to find suitable hyper-parameters.

Note that embedding-based method requires $\mathcal{O}(n^2)$ time complexity to compute all pair-wise distances. In contrast, LSH-based method only requires $\mathcal{O}(n)$ by completing a single round of mapping from samples to buckets. We have only used LSH-based similarity score in some experiments due to this complexity. Besides, similarity scores based on different sample-level similarity functions are not comparable.

### 4.6.3. Sensitivity.

4.6.3.1. *Impact of Dataset Composition.* We focus on hotel reviews of four cities: London, New York, San Francisco and Las Vegas. We split the data into training and test set by randomly selecting 20000 reviews out of each city to create a test set with 80000 reviews. Table 4.2 shows the size of the training set for the four cities.

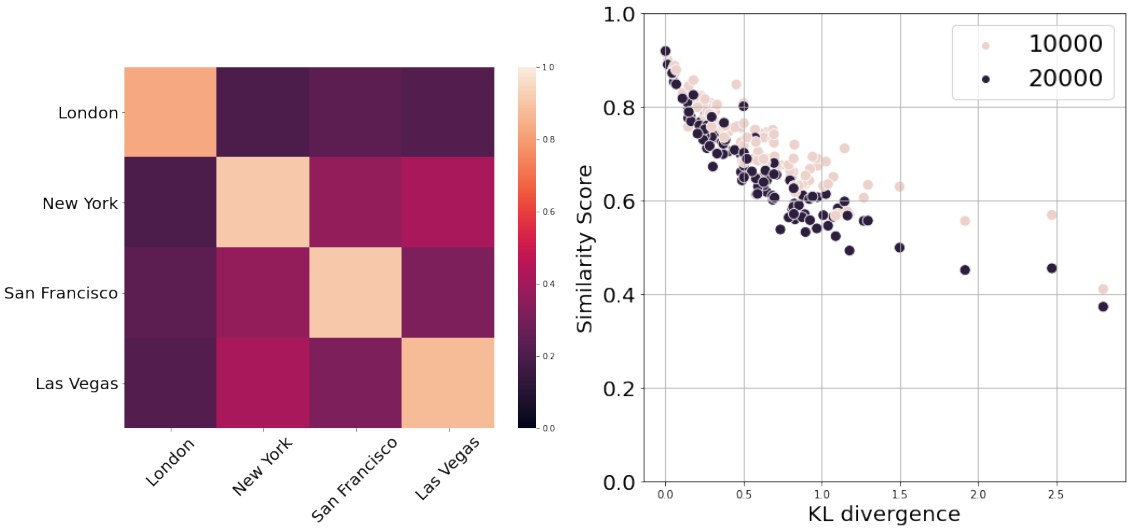
TABLE 4.2. Training Set Composition

| City | London | New York | San Francisco | Las Vegas |
|---|---|---|---|---|
| # of Hotel Reviews | 59349 | 35143 | 10401 | 6682 |

We randomly sample 10000 or 20000 reviews from the test set with specific proportions for each city, and calculate its LSH similarity with the entire training set. Since different cities have dissimilar distributions of reviews (See Figure 4.31a), the similarity between the sampled reviews and reviews in the training set is mainly determined by the difference of their distributions over cities. Similar distributions over cities will lead to higher similarity, and vice-versa. Figure 4.31b shows the relationship between our similarity metric and KL divergence between distributions over cities. It can be observed that our metric can capture changes in the dataset's distribution. Similarity scores are higher when the sampled set contains 10000 samples instead of 20000. As described earlier, a smaller dataset leads to smaller neighborhoods, which, in turn, results in larger $p$-values, and thus a larger similarity score.

4.6.3.2. *Semantic Clustering.* We study if our metric can find semantically similar datasets by using the Imdb dataset. For notational convenience, we call the four subsets in Imdb as *pos1*, *pos2*, *neg1* and *neg2*, where 'pos' and 'neg' denote 'positive' and 'negative', and 1 and 2 denote training and test set, respectively. We calculate all pair-wise similarity scores for four subsets with different sample-level similarity scores. Results in Figure 4.32 show that our metric consistently gives a higher similarity score to subsets with similar semantic meaning. The increased similarity between positive subsets and negative subsets are due to the small size of neighborhoods as the graph gets sparse.

4.6.3.3. *Metric Behavior under Noisy Datasets.* Next, we check if our metric is sensitive to small noise in datasets. In order to do so, we create a noisy dataset and check its similarity with the original dataset. We randomly drop words in Imdb dataset, randomly mask out tokens and replace them with BERT or Condition-BERT [**89**] (CBERT) in SST5. For the Imdb dataset, we




(A) Heatmap of Similarity among Cities (B) KL Divergence and Similarity Score

FIGURE 4.31. Metric behavior as dataset composition changes. Left: Reviews for different cities are not similar. Right: Relationship between similarity score and KL divergence.

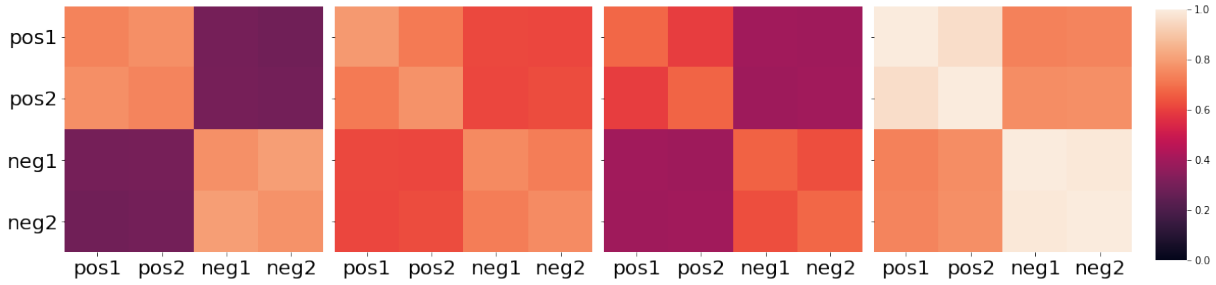
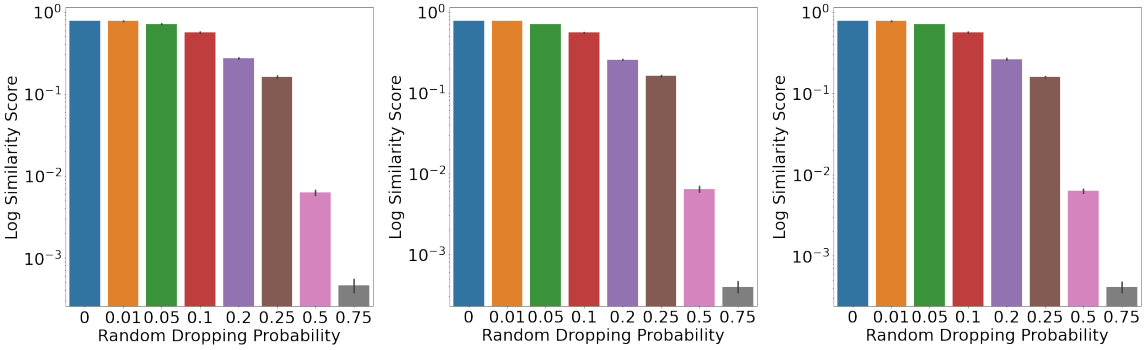



FIGURE 4.32. Heatmaps for subsets pair-wise similarity scores. From left to right: LSH-based, $(n, r, b) = (2, 1, 150)$, LSH-based, $(n, r, b) = (2, 2, 150)$, LSH-based, $(n, r, b) = (3, 1, 150)$, embedding-based

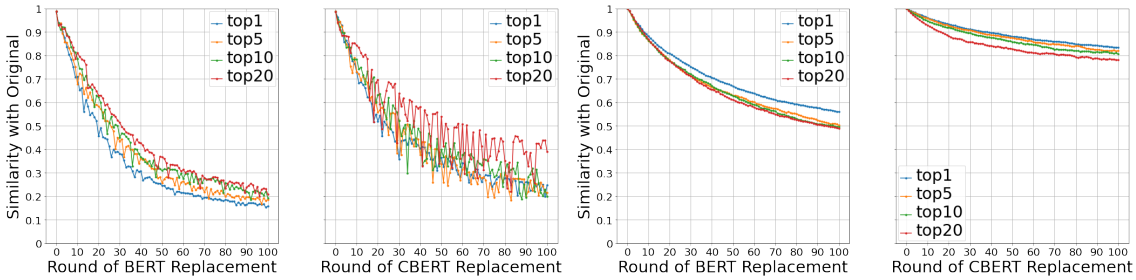
test with various values of $p$, the probability with which words are dropped. We randomly drop words from sentences in the test set's positive subset and calculate its similarity with the training set's positive subset. For each probability, we conduct 10 independent runs and in each run, every word will independently be dropped with probability $p$. For SST5 dataset, we start with the original dataset, tokenize it, randomly mask out one token from each sentence and predict the token with the prediction model. This tokenize, random mask, predict loop is repeated 100 times for each model. The dataset becomes more and more noisy with every iteration. We tested top-1, top-5, top-10 and

top-20 sampling strategies to create a total of 800 noisy SST5 datasets. CBERT model is fine-tuned for 20 epochs with default hyper-parameters from the author's code[3].




(A) LSH $n = 2,\ r = 1,\ b = 150$ (B) LSH $n = 3,\ r = 1,\ b = 150$ (C) LSH $n = 4,\ r = 1,\ b = 150$

FIGURE 4.33. Log LSH similarity for Noisy Imdb Datasets. Similarity is calculated between the subsets of noisy test set positives and original training set positives.



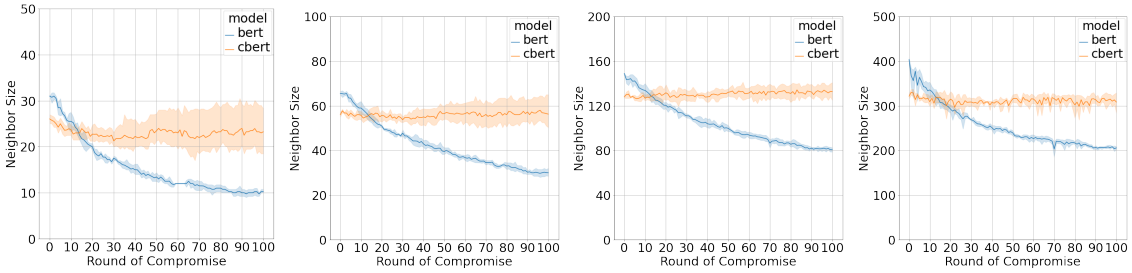

(A) LSH $n = 2,\ r = 1,\ b = 50$ (B) Embedding-based

FIGURE 4.34. Similarity for Noisy SST5 Datasets. (a) LSH-based similarity score. (b) Embedding-based similarity score. Similarity is calculated between the noisy dataset and the original dataset.

Figures 4.33 and 4.34 show that our metric is sensitive to small noise in data. When $p$ reaches 0.1, or ten rounds after BERT or CBERT replacement, the similarity scores have a clear decrease as the dataset becomes more and more noisy. Our metric can also detect the difference between BERT and CBERT by finding CBERT more similar with the original dataset. Besides, different sampling methods matter in case of CBERT only. This makes sense as CBERT is fine-tuned with both context and label to predict masked tokens more accurately.

[3] `https://github.com/1024er/cbert_aug`

One interesting observation from Figure 4.34 is that embedding-based similarity score detects a significant difference between BERT and CBERT. However, LSH-based similarity score does not detect such a difference. This is because of decreasing neighborhood sizes in the process of LSH. In Figure 4.35, we can see that the 2.5%, 5%, 10% and 20% percentiles of neighborhood sizes decreases by 2 to 3 times for in case of BERT but does not happen for CBERT. Smaller neighborhood sizes lead to larger $p$-values and ultimately makes BERT and CBERT performance similar.




(A) 2.5% percentile (B) 5% percentile (C) 10% percentile (D) 20% percentile

FIGURE 4.35. Change of Different Percentiles of Neighborhood Size: How the 2.5%, 5%, 10% and 20% quantile of neighborhood size changes as round of compromise increases. All have a huge decrease, suggesting they are likely to create large $p$-values. Y-axis values are not shared among figures.

**4.6.4. Applications : Domain Adaptation Performance.** Absence of rich labeled data is a common challenge faced by many downstream machine learning applications. A popular solution is domain adaptation where a model pre-trained on a much larger dataset in one domain is then adapted to a desired domain. One application of our similarity metric is in domain adaptation. We argue that similarity between two datasets could work as a good indicator of how suitable the pre-trained model in source domain is for the target domain.

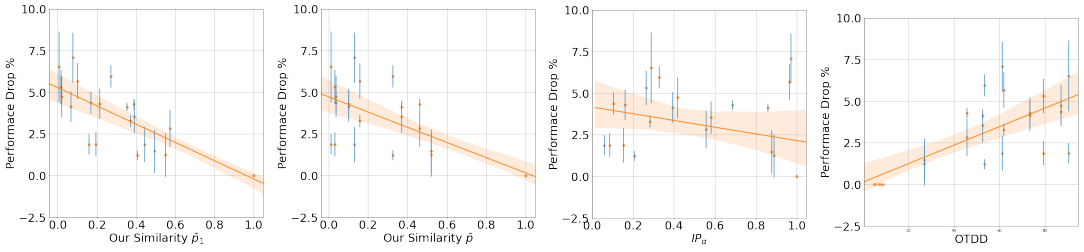
To justify this idea, we test all pairs of domain adaptation for dataset **Imdb Review**, **SST2**, **Ariline Comments**, **Amazon Product Review** and **Noisy Imdb**. All these datasets have two labels: positive and negative. For each dataset, we obtain their embeddings on [CLS] token with a pre-trained BERT model, and then train a two-layer fully connected network on the top of it with ReLU activation. We train the classifier for 5 epochs (except for Airline Comments, where we only trained 1 epoch) with a learning rate 0.001 and directly test the pre-trained model on all datasets for evaluation, with and without fine-tuning. When fine-tuned, we randomly sample 1024 reviews from

the target domain and fine-tune with batch size 32, learning rate 0.001 for 3 epochs. All experiments are done with 10 different seeds.

To evaluate how the performance of domain adaptation, we analyze the loss in accuracy when compared with directly training on target domain.

$$\mathcal{T}(\mathcal{D}_S \to \mathcal{D}_T) = 100 \times \frac{\text{accucary}(\mathcal{D}_T) - \text{accuracy}(\mathcal{D}_S \to \mathcal{D}_T)}{\text{accuracy}(\mathcal{D}_T)} \tag{4.5}$$

where $\mathcal{D}_S$ is the source domain on which the model is trained, $\mathcal{D}_T$ is the target domain for which the model is to be tested, and accuracy $(\mathcal{D}_T)$ is calculated as accuracy$(\mathcal{D}_T \to \mathcal{D}_T)$. The benefit of this metric is that it can separate the difficulty of learning each task.




(A) Ours $\bar{p}_1$, $r = -0.784, \rho = -0.805$ (B) Ours $\bar{p}$, $r = -0.685, \rho = -0.615$ (C) $IP_\alpha$ in **[1]**, $r = -0.301, \rho = -0.359$ (D) OTDD in **[2]**, $r = 0.666, \rho = 0.634$

FIGURE 4.36. Drop in accuracy with finetuning. Orange dots are mean values and error bars are obtained through 10 runs. Solid line is the fitted regression line with 95% confidence interval.

We compare our methods with the recent methods in **[1, 2]**, and check if methods have strong rank correlations (i.e. Spearman's correlation) with the model performance. In Figure 4.36, it can be observed that while all methods except for $IP_\alpha$ exhibit a strong correlation between the adaptation performance and the respective metric, our metric is clearly more closely correlated.

We also compared with more distance metrics and full results are available in Supplementary Material A.3.2. We have three interesting findings: **(1)** Without fine-tuning, the separation defined in Linear Fisher Discriminant is the best and our metrics $\bar{p}$ closely follows. **(2)** With fine-tuning, another metric of ours, $\bar{p}_1$, the average $p$-value for the source domain, outperforms all others by a large margin. **(3)** Traditional methods work well practically.

However, there is a subtle clarification here. Although our experiment results show a strong correlation between similarity and adaptation performance, effective domain adaptation *does not*

*only require* very similar datasets over sample space $\mathcal{X}$. It also requires that the relationship between samples and labels, $p(y|x)$ remains similar. If machine learning training methodology could deal with long-tail distribution and edge cases well, we would only need the support of $\mathcal{D}_S$ to cover the support of $\mathcal{D}_T$ for effective domain adaptation between source and target domains.

## 4.7. Discussion

The aforementioned experiments help to establish that our metric provides a similarity measure to compare datasets from different aspects. In the discussion below, we briefly point out the future directions to further improve this framework.

**Independence Analysis**. For all theoretical properties, we assume that the edges are totally independent. However, this might not be the case. LSH is one situation where the edges are correlated. Measuring the quantitative influence of this correlation is essential for more reliable similarity scores.

**Efficiency**. In our experiments, embedding based similarity between two 10K-ish datasets with 768-dimension embedding takes around 1 hour and similarity between two 25K datasets takes over 4 hours for each run. LSH methods include more randomness than necessary to increase speed. A faster processing method without introducing too much randomness will be desirable.

**Difference between $\bar{p}_1$ and $\bar{p}_2$**. In our experiments, $\bar{p}_1$ and $\bar{p}_2$ can be quite different. We also found $\bar{p}_1$ is closely related to domain transfer performance but $\bar{p}_2$ does not. A deeper understanding of these one-side average $p$-values should provide us a better understanding of complex datasets.

**Sparse Connections**. As shown in Section 4.6.3.3, a relatively small neighborhood size leads to a larger $p$-value. This will make the framework less sensitive when datasets are sparse or dealing with outliers. Handling this situation remains an open question.

**Label Similarity**. In this paper, we only focus on the similarity in feature space but not on the labels. Extending this framework to include the labels is a promising direction of future research. [**2**] directly takes the distance between labels as an additional composition of the total distance but this cannot be simply applied here if we don't assume that the space is Euclidean.

**Sample Level Inference**. [**1**] proposed a method to check authentic generations versus unauthentic ones, but we found it neither theoretically sound nor accurate in practice (see Supplementary Material A.3.4). This direction is also crucial for a deeper understanding of generative models.

## 4.8. Conclusion

In this paper, we have proposed a novel dataset-level similarity score framework. This framework is based on a graph structure and statistical hypothesis testing, and can work with any sample-level similarity score. We only require a sample-level similarity score that is appropriately defined for the data, model or task at hand. This makes our framework very flexible to establish similarity between datasets. We provide theoretical guarantees to explain why the framework would work in practice. Empirical experiments on several publicly available datasets and common NLP tasks show the effectiveness and sensitivity of our metric to changes in data. Since this is the first step toward establishing a dataset-level similarity framework, there remain some open questions and potential improvements. We point out those questions as directions for future research.

CHAPTER 5

# COVID-19 Prediction Efforts at Health Davis Together

## 5.1. Abstract

COVID-19 has changed people's lives from almost every aspect. To stop the spread of COVID-19 and save people's lives, governments have put tremendous effect into fighting against the pandemic. In order to most efficiently stop the spread of this virus, we must have a deeper understanding of its pattern. For this purpose, researches in Healthy Davis Together project are collaborating on several studies to make sure the restrictions are effective and people are getting protected. In this paper, we describe the engineering efforts, including model evaluation framework, data preprocessing details, mobility variable design, and also connecting different systems for comprehensive data analysis.

## 5.2. Introduction

Since the outbreak of COVID-19, or SARS-CoV-2, in Jan 2020, the way people live has been changed. In order to stop the further spread of the virus and get people's live back to normal, governments around the word are taking actions, such as face mask orders, shutting down of business and schools and vaccines. To better react to this pandemic and save more people's lives, a deeper understanding of COVID's pattern and effect of different restrictions is needed.

For this purpose, we have Healthy Davis Together (HDT), A joint project between the City of Davis and UC Davis with a goal to prevent the spread of COVID-19 and facilitate a coordinated and gradual return to regular city activities and reintegration of UC Davis students back into the Davis community. As a member of the forecasting team, I worked with other excellent people in the group and tried to understand more of COVID-19 through various methods. My involvement in HDT project can be grouped into mainly three parts. A back-testing framework to fast iterate and evaluate predictive models, constructed a mobility variable to facilitate other aspects of research, and also connected wastewater data with HDT testing data.

### 5.3. Back-testing Framework

A predictive model of daily increase in infection or death is most likely to provide us with a better understanding of how the virus develops, spreads and how we should stop it. It's natural to model the infection or death count as a time series. To find the most suitable model, or understand how different models are performing in a given period of time, a framework that can easily choose model, time range, geographical range, data source and evaluation metric is needed.

**5.3.1. Overview.** To evaluate a predictive model within a given time period, the best practice is to choose multiple train/test split and conduct the train test loop multiple times to get an estimate of prediction accuracy with less noise and randomness. Besides, there are also some noise and periodicity in the reported data, some smoothing and pooling may also be helpful. To build a framework that support light code testing and the required flexibility, the Algorithm 5 is designed:

**Algorithm 5:** Initial Back-testing Framework

**Result:** Back-testing Framework for Predictive Models

Input model $\mathcal{M}$, start date $t_1$, end date $t_2$, smoothing interval $\Delta t$, minimum required training data $t_{\min}$, maximum prediction length $L$, data $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ between $t_1$ and $t_2$, evaluation metric $s(y, \hat{y})$

Calculate total available number splits $N = \lfloor \frac{t_2 - t_1}{\Delta t} \rfloor - 1$

**for** *i from 0 to N-1* **do**

  Smooth data from date $t_{\text{start}} = t_2 - (i+1) \cdot \Delta t$ to date $t_{\text{end}} = t_2 - i \cdot \Delta t$, and obtain smoothed training signals $X_{N-i}$ and prediction target $y_{N-i}$

**end**

**for** *i from* $t_{min}$ *to N* **do**

  Train model $\mathcal{M}$ with $X_{1:i}$, $y_{1:i}$;

  Make predictions for up to next $L$ steps, $\hat{y}_{i+1:\min(i+L,N)}$;

  Calculate evaluation scores $v_{i,1:\min(L,N-i)} = s(y_{i+1:\min(i+L,N)}, \hat{y}_{i+1:\min(i+L,N)})$;

**end**

**for** *i from 1 to L* **do**

  Calculate average evaluation score $\bar{v}_i = \bar{v}_{\cdot,i}$

**end**

**5.3.2. Data Source.** The framework mainly uses the data available from Delphi's covidcast API [1]. Supported sources includes jhu-csse, usa-facts and indicator-combination. Both confirmed cases and death cases are supported as prediction target. Note that these data sources usually have

[1] `https://cmu-delphi.github.io/delphi-epidata/api/covidcast.html`

3 to 21 days behind actual date, depending on the choice of state. The framework will automatically adjust $t_2$ according to the detected delay.

Besides, each date's data will also be updated after the first time it's released as there are corrections. Here we call this factor 'as-of date', meaning which date's version of data we choose to use. One feature of COVID-19 indicators is that there will be back corrections and records in the past may change after its first release. To best simulate what would happen when we apply a trained predictive model, we adjust the way to obtain training and testing data in evaluation. For both and training and testing data, we choose the earliest as-of dates when the data is available to us. For example, if the training data is from April 1st to May 1st, and testing data is from May 2nd to May 11th and the delay is 3 days, then training data's as-of date will be May 4th (3 days after May 1st) and testing data's as-of date will be May 14th (3 days after May 11th). This inevitably will change the back-testing framework slightly: the training data $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ will no longer be provided in the beginning, but queried for each train test split separately. The modified back-testing framework is in Algorithm 6

**Algorithm 6:** Modified Back-testing Framework

**Result:** Back-testing Framework for Predictive Models

Input model $\mathcal{M}$, start date $t_1$, end date $t_2$, smoothing interval $\Delta t$, minimum required training data $t_{\min}$, maximum prediction length $L$, evaluation metric $s(y, \hat{y})$

Calculate total available number splits $N = \lfloor \frac{t_2 - t_1}{\Delta t} \rfloor - 1$

**for** *i from 0 to N-1* **do**

  Find training start time $t_{\text{start}} = t_2 - (i+1) \cdot \Delta t$, training end time $t_{\text{end}} = t_2 - i \cdot \Delta t$ and prediction end time $t_0 = \min(t_2, t_2 + (L-i) \cdot \Delta)$ Get correct as-of dates for $t_{\text{end}}$ and $t_0$, note as $t'_{\text{end}}$ and $t'_0$;

  Query data from $t_1$ to $t_{\text{end}}$ with as-of date $t'_{\text{end}}$, smooth date between $t_{\text{start}}$ and $t_{\text{end}}$ for training data $X_{i,\text{train}}$, $y_{i,\text{train}}$;

  Query data from $t_{\text{end}} + 1$ to $t_0$ with as of date $t'_0$, smooth them for testing data $y_{i,\text{test}}$;

  Train model $\mathcal{M}$ with $X_{i,\text{train}}$, $y_{i,\text{train}}$;

  Make predictions for up to next $L$ steps, $\hat{y}_{i,\text{pred}}$;

  Calculate evaluation scores $v_{i,1:\min(L,N-i)} = s(y_{i,\text{test}}, \hat{y}_{i,\text{pred}})$;

**end**

**for** *i from 1 to L* **do**

  Calculate average evaluation score $\bar{v}_i = \bar{v}_{\cdot,i}$

**end**

**5.3.3. Data Imputation.** We also realized that data from all sources suffers from the same problem: missing data for many dates without a clear pattern. Our understanding is that for some states or counties, the data is not reported or updated for those dates. If the missing appears in training data, we impute them with B-spline **[65]** implemented by Scipy **[86]**.

**5.3.4. Predictive Models.** We applied two models to predict the daily death counts. The first model is called Armadillo, proposed in **[7]**, which relies on a mobility variable and historical death counts. The Armadillo model predicts death counts as a proportion of the infection rates with a known delay distribution, while the infection rates follow a discretized epidemic renewal equation with the effective reproduction number depending on mobility variables. The second model is called ARLIC model, which is developed by another member, Stephen Sheng. The model relies on a leading indicator $L$, a delay distribution $D$ and the historical death counts $C$. The model is trained with the following steps:

(1) Deconvolve leading indicator $L$ with delay distribution $D$, obtain $\tilde{L}$.
(2) Fit an AR kernel $f_L$ on $\tilde{L}$, fitted values for $\tilde{L}$ is $\hat{L}$
(3) Fit another AR kernel $f_C$, which first calculates $I_t$ with $\hat{L}$, then convolve $I_t$ with $D$ for the estimate of death counts $\hat{C}$. This kernel is trained to minimize the difference between $C$ and $\hat{C}$
(4) When trying to predict future daily death counts, the model first predict future $\hat{L}$, then obtain $I_t$ and calculate $\hat{C}$ with convolution between $I_t$ and $D$

**5.3.5. Evaluation Metrics.** We apply two evaluation metrics for understanding the performance of trained models. Mean Absolute Error (MAE) and Wasserstein-1 distance ($W_1$). Implementation of MAE is straightforward so we skip the details here. Since $W_1$ is a distance between distribution CDFs, it takes relative weights instead of absolute counts. A naive implementation of $W_1$ will ignore the prediction error in total death counts but only capture the relative death counts. Our implementation adds a 'fake' last day into both real death counts and prediction to balance the total death counts. Denote $y_{1:t}$ as real death counts and $\hat{y}_{1:t}$ is our prediction, define $\delta = \sum y_{1:t} - \sum \hat{y}_{1:t}$. We define

$$y' = \text{concatenate}(y_{1:t}, \max(0, -\delta)), \quad \hat{y}' = \text{concatenate}(y_{1:t}, \max(0, \delta))$$

This would balance the sum of $y'$ and $\hat{y}'$

## 5.4. Mobility Variable

While testing our models with the back-testing framework, we found Armadillo model performs consistently worse than ARLIC model. We realized Armadillo model heavily relies on the quality of mobility variable and the mobility variables we can obtain from Delphi's API are too noisy with not enough information. To discover the full potential of Armadillo model, as well as to fuel other aspects of studies, such as the change in people's mobility pattern, the influence of different restriction orders, we decided to construct our own mobility variable with SafeGraph data [2].

**5.4.1. Overview.** Our goal is to construct an interpretable Census Block Group (CBG) level mobility variable. Ideal mobility variable should be proportional to the potential infection risk at that CBG for a given time period. One existing work **[11]** checks the visits from different CBGs to Points Of Interests (POI). It assumes when infected people visits POIs, it will interact with and potentially infect susceptible people in the CBG where that POI is located. And the new infection rate would be proportional to the expected number of infected people visiting a certain CBG at the same time. They model the new infection as a Poisson process whose rate is calculated based on the assumption.

Our mobility variable made an extra assumption and a simplified fitting procedure. We assume that new infection happens not only because infected people visits other CBGs, but they meet other susceptible people in the POIs they are visiting. So the new infections can happen not only in the CBGs where that POIs are located but also other CBGs whose susceptible people are visiting the same POIs as the infected people.

With this basic assumption, we analyse and construct our own mobility variable.

**5.4.2. Notations.** Table 5.1 summarizes the symbols we use to construct the mobility variable.

**5.4.3. Sources of Infection.** Now we can formularize the sources of new infections. The basic idea is when infected people visit a POI, healthy, susceptible people in the same POI have a

[2] `https://www.safegraph.com/`

TABLE 5.1. Symbols

| Symbol | Meaning |
|---|---|
| $c_i$ | CBG with index $i$ |
| $p_j$ | POI number $j$ |
| $I_{ij}$ | indicator of POI $p_j$ being in CBG $c_i$ |
| $N_i$ | Population size of $c_i$ |
| $r_i$ | Proportion of population infected in $c_i$ |
| $W_{ijt}$ | number of visitors who live in $c_i$ and visit POI $p_j$ at time $t$ |
| $\tau_{jt}$ | infection transfer coefficient for POI $p_j$ at time $t$ |
| $\lambda_{it}$ | infection transfer coefficient for CBG $c_i$ at time $t$ |

probability of being infected, depending on how many infected people are there in the same POI at the same time. The expectation of new infections should be proportional to three values: susceptible population size, infected population size and the infection transfer coefficient. So the new infection at time $t$ for each CBG $c_i$ should be:

$$\lambda_{it} \sum_{i'} \sum_{j} I_{ij} \cdot W_{i'jt} \cdot r_{i'}(1 - r_i) \cdot N_i \tag{5.1}$$

and

$$\sum_{j} \tau_{jt} \sum_{i'} W_{ijt} W_{i'jt} r_{i'}(1 - r_i) \tag{5.2}$$

Equation 5.1 means the contribution of infected people visited CBG $c_i$ and Equation 5.2 means the contribution of healthy people of CBG $c_i$ get infected while visiting other POIs.

**5.4.4. Time Resolution.** The Poisson process in **[11]** is estimated hourly. Visits data from SafeGraph is on a weekly level. To estimate the mobility variable on hourly level, they fit the required unknown parameters through a procedure called iterative proportional fitting procedure (IPFP). This procedure, as they mentioned in their paper, may not guarantee convergence as there are two marginal distributions to fit.

To avoid the complexity and the potential failure of the fitting procedure, we choose to use a lower resolution, which is day, instead of hour. We do this not only to avoid the potential failure of IPFP but also because we found that the number of devices tracked by SafeGraph is significantly

changing from day to day, suggesting that the visit data is noisy by nature. So a simpler but robust model should be more reliable.

**5.4.5. Estimate of $W_{ijt}$.** We only have $W_{ijt}$ on a weekly basis and we would like to estimate the values on a daily basis. Since there is noise in the SafeGraph data, we deployed a Bayesian model and use that to smooth the effect of observation noise. Note the distribution of visitors to a POI in a week is a joint distribution $\mathcal{F}$ over CBGs and 7 days of a week. We further assume

$$\mathcal{F} = \mathcal{F}_{day} \cdot \mathcal{F}_{CBG}$$

that is the distribution over CBGs and weekdays are independent. Now we only need to estimate the prior of $\mathcal{F}_{day}$ and $\mathcal{F}_{CBG}$. For convenience, as the observation follows a multinomial distribution, we choose the prior as its conjugate prior, Dirichlet distribution. Here we explain how we estimate the prior for $\mathcal{F}_{day}$ in details.

Assume the prior is

$$Dir(\alpha_1, \alpha_2, \cdots, \alpha_7)$$

with observation $n_1, n_2, \cdots, n_7$, the marginal distribution is

$$\begin{aligned}
L(n_1, n_2, \cdots, n_7) &= \int_{\sum_{i=1}^7 p_i=1, p_i \geq 0} \frac{\Gamma(\sum_{i=1}^7 \alpha_i)}{\prod_{i=1}^7 \Gamma(\alpha_i)} \prod_{i=1}^7 p_i^{\alpha_i - 1} \cdot \frac{n!}{n_1! n_2! \cdots n_7!} \prod_{i=1}^7 p_i^{n_i} dp_1 dp_2 \cdots dp_7 \\
&= \frac{n!}{n_1! n_2! \cdots n_7!} \frac{\Gamma(\sum_{i=1}^7 \alpha_i)}{\prod_{i=1}^7 \Gamma(\alpha_i)} \int_{\sum_{i=1}^7 p_i=1, p_i \geq 0} \prod_{i=1}^7 p_i^{\alpha_i + n_i - 1} \\
&= \frac{n!}{n_1! n_2! \cdots n_7!} \frac{\Gamma(\sum_{i=1}^7 \alpha_i)}{\prod_{i=1}^7 \Gamma(\alpha_i)} \cdot \frac{\prod_{i=1}^7 \Gamma(\alpha_i + n_i)}{\Gamma(n + \sum_{i=1}^7 \alpha_i)} \\
&= \frac{n! \Gamma(\sum_{i=1}^7 \alpha_i)}{\Gamma(n + \sum_{i=1}^7 \alpha_i)} \cdot \prod_{i=1}^7 \frac{\Gamma(\alpha_i + n_i)}{n_i! \Gamma(\alpha_i)} \\
&= \prod_{k=1}^n \frac{k}{k - 1 + \sum_{i=1}^7 \alpha_i} \cdot \prod_{i=1}^7 \prod_{k=1}^{n_i} \frac{\alpha_i + k - 1}{k} \\
l(n_1, n_2, \cdots, n_7) &= \log L(n_1, n_2, \cdots, n_7) \\
&= \sum_{k=1}^n \log(k) - \log(k - 1 + \sum_{i=1}^7 \alpha_i) + \sum_{i=1}^7 \sum_{k=1}^{n_i} \log(\alpha_i + k - 1) - log(k)
\end{aligned}$$

All we have to do is to maximize the log-likelihood for all observations. This is done in the gradient ascent fashion. We also choose only to estimate the priors for a POI only if it has appeared at least 3 times in the dataset to avoid naive overfitting. For all POIs that have no more than 2 records, they share a common prior distribution.

When doing inference, we count the total number of visitors to each POI for a given week, and re-distribute them to all CBGs and 7 days of that week. This procedure will create non-integer number of visitors but we don't think this is a problem as long as the values get closer to the true distribution of visitors. Our ultimate goal is only to create a mobility variable proportional to the increase of infection.

**5.4.6. Estimate Other Parameters.** For all the parameters required in equation 5.1 and 5.2, $N_i$ is directly available from SafeGraph data, $r_i$ can be estimated with a given model, like SEIR, or other available sources. In our case, as we have the saliva test results from HDT[3] so we estimate $r_i$ for all CBGs in Yolo County. As for $\tau_{jt}$ and $\lambda_{it}$, we optimize them to maximize the cosine correlation of our mobility variable and the daily new infected cases.

Another factor that would clear affect the value of $\lambda_{it}$ and $\tau_{jt}$ is the real-time condition of those POIs, such as the density of visitors in that POI and how long they stay for average. These factors are also considered in [**11**]. The density can be obtained by the number of visitors divided by the area of POI and we choose the median time people stay in the POI in that week.

**5.4.7. Empirical Comparison.** For comparison, we choose the rate of Poisson distribution in [**11**] as baseline mobility variable. We tune the order of area of POI, the order of people's median stay time to see how two mobility variables behave. Further more, as the number of POIs is too large, we simplify $\tau_{jt} = \tau_t$ for all POIs.

As for $\lambda_{it}$, we use experiments to check if they can also be simplified to a common value among different CBGs, $\lambda_t$. We use cosine similarity of mobility variable and new infection counts in a given range of days to evaluate the quality of mobility variable. The cosine similarity is calculated for each CBG separately. Testing data is from Nov 18th, 2020 to March 6th, 2021 within Davis (CBGs start with '06113'). Empirical results are available in Figure 5.1. We call the results 'overall cosine' when we do not distinguish $\lambda_{it}$s and 'row wise cosine' when we do. Results show the impact of order

[3] `https://healthydavistogether.org/testing/`

of POI area and time of people's stay, as well as the necessity of distinguishing transfer coefficient for different CBGs.

## 5.5. Wastewater Processing

One concern of using HDT's saliva test result is that people who suspect of being infected will be more likely to get tested, especially for CBGs that are distant to the main campus area. To have a better understanding of this potential positive bias, and also a more accurate estimate of infection severity of different areas, we decide to combine wastewater data. People try to estimate the density of two specific genes of COVID-19 virus from wastewater. However, we only know the connection map and the collection point locations, so a convenient tool for each analysis is needed. In this section, the process of data cleaning, analysing and processing is described.

There is another potential benefit from wastewater analysis. We know that people react differently after infected by the virus: some has no symptoms at all, some behaves like catching usually cold, while some quickly find it hard to breath. It's also possible that different people will spread different amount of the virus. So only the number of infected patients may not be enough to describe the potential risk in an area. Wastewater can help by viral shedding in fecal matter.

**5.5.1. Build Connection Graph.** Raw data mainly consists of two parts:

- A table of maintenance hole information, including names and coordinates. And
- A table of pipes connecting maintenance holes, including names of starting and ending maintenance hole, with coordinates for all points along the road.

Looking into the two tables, we realized that the naming system in two tables is not exactly the same, but maintenance holes that share the same name will always share the same coordinates. So we decide to use coordinates as identifier for different maintenance holes. With coordinates as identifier, we build a directed graph with pipes' information in the second table. After investigation, only the initial and last pair of coordinates, which correspond to the starting and ending maintenance holes, are meaningful. All coordinates in the middle are only used to guide the direction but does not correspond to any existing maintenance holes and pipes sharing same middle points do not intersect at all. With all these clarified, the directed graph that comprises pipes connecting maintenance holes becomes acyclic.

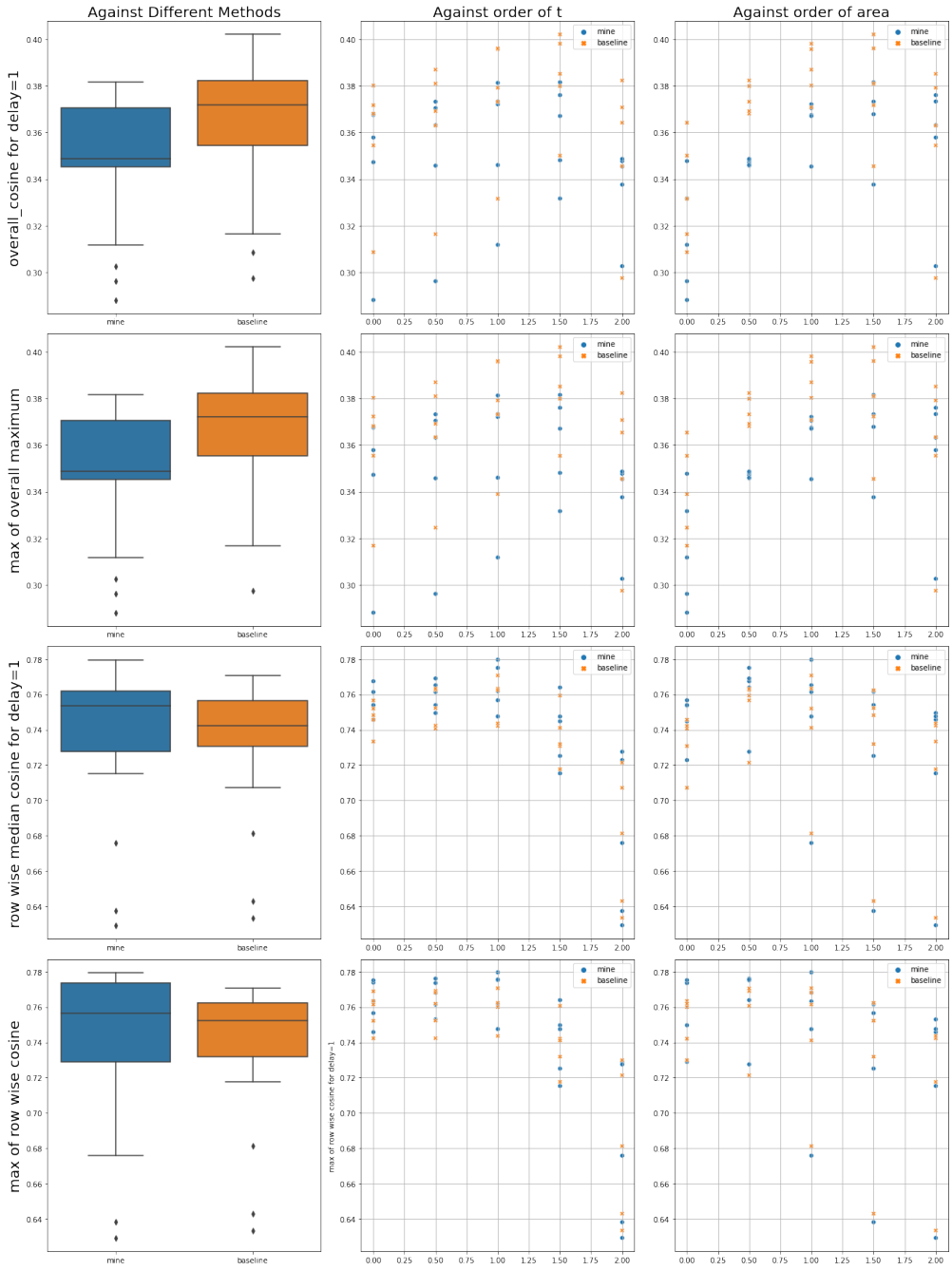



FIGURE 5.1. Comparsion of different mobility variables. Left column is the boxplot for two mobility models, middle column compares different choices of the order of t, right column compares different choices of the order of area of POIs. The first two rows simplify $\lambda_{it} = \lambda$ for all CBGs and the last two rows do not. Clearly, we can see that after optimizing the infection transfer coefficient for each CBG independently, our mobility variable improved significantly and gets better than the baseline method.

**5.5.2. Find Upstream/Downstream.** The upstream of a maintenance hole is the sources of wastewater collected of the maintenance hole, and the downstream of a maintenance hole is where you can collect it. Finding upstream/downstream is not difficult given the graph we have built. To help people better understand the system, we created a visualization tool, which takes in the coordinate of a point, and visualize the upstream or downstream of the maintenance hole that is closest to the given coordinate. See Figure 5.2 and 5.3 as examples. We also checked the upstream for all collection points. Collection points are maintenance holes from where wastewater samples are collected. See Figure A.5a to A.7j in Appendix A.4

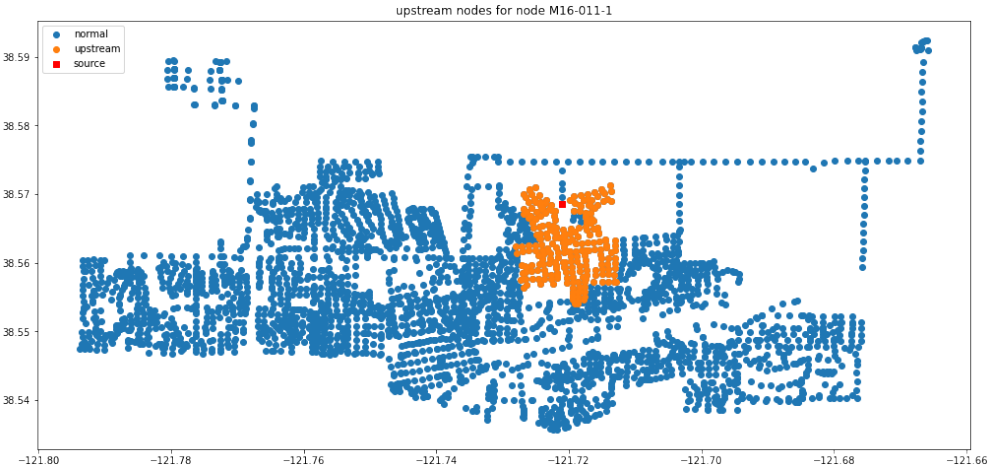



FIGURE 5.2. Upstream for a maintenance hole

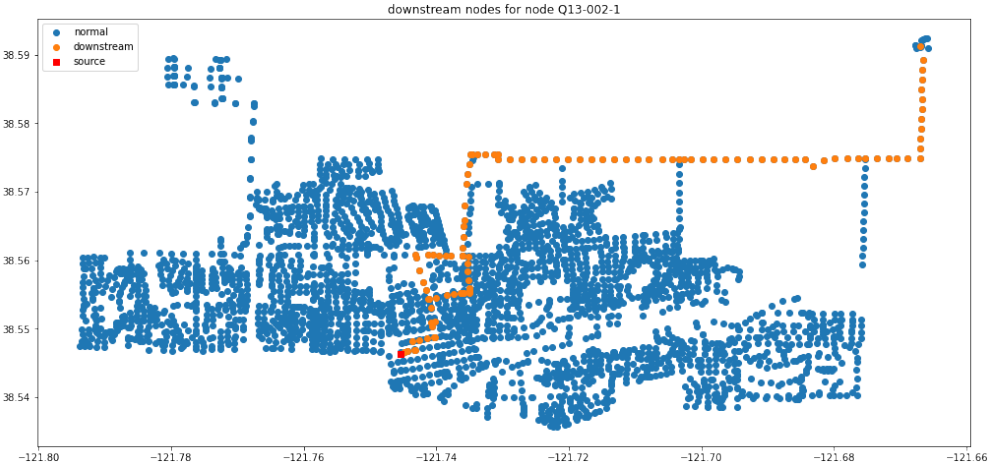



FIGURE 5.3. Downstream for a maintenance hole

**5.5.3. Analysis of Wastewater Composition.** From the visualization of collection points, it's clear that different collection points cover different sizes of areas and these areas are overlapping. So it is important to understand the composition of wastewater at each collection point quantitatively. To do this analysis, we need to know

- How wastewater from confirmed cases are collected
- How wastewater from healtht people are collected

Now we assume that each person, healthy or not, produces the same among of wastewater per day and we call this amount a 'unit'. The same procedure can answer the two questions above: first determine how many units of wastewater go into the system through each maintenance hole, then pass the number of units of wastewater from upstream maintenance holes to downstream ones. If a certain maintenance hole can go to downstream maintenance holes through multiple pipes, the wastewater is evenly split to each of the pipes. Further more, we track not only the total amount of wastewater but also each census block's contribution. We choose to monitor census blocks because for test results from HDT saliva test, each testee's living census block is the best geo resolution we have.

We also detected some potential problems with the current collecting system. For most census blocks, a single collection point can fully collect their wastewater, or two separate collection points. But there are also census blocks that are only partially collected by the current collection system.

**5.5.4. Applications.** With this system connecting the saliva test results and wastewater test results on different geo resolutions, we can now use regression methods to analysis the relationship between infection rates and potential infection dangers and perhaps other factors for a clearer picture of the infection in different neighbors at Davis.

## 5.6. Summary

COVID-19 hit us unprepared. In the journey to understand its spread pattern, to find effective prevention strategies, and to evaluate how ready we are prepared, we explored multiple resources for comprehensive and detailed data, employed various models to understand what's happening around us, and built systems from scratch to facilitate different area of researches. Now we are almost in the fully reopen stage but the researches should still continue. A better understanding is in need to

make sure we have taken enough actions to protect people from the virus, and also prepare us for potentially another hit from other kinds of virus. We hope this analysis will show people possible directions for further studies, and the tools we build make it easier for accessing necessary data.

APPENDIX A

# Appendix

## A.1. Appendix for Chapter 2

### A.1.1. Proofs.

THEOREM A.1. *Consider a MCTS with only one layer. If we now have k choices with expected values* $v_1, v_2, \cdots, v_k$. *W.l.o.g we can assume* $v_1 \geq v_2 \geq \cdots \geq v_k$. *their variances are*$\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2$. $\sigma_i^2 \leq M \ \forall i$. *Set* $d = v_1 - v_2$. *Probability is estimated by a mini-batch with size b. The constant we use is* $c_{puct}$. *We make the decision if the test time of a choice is more than t. Then the probability that we can choose the best choice is approximately lower bounded by*

$$\begin{cases} 1-(k-1)texp(-\frac{bd^2t}{4M}) & if \quad \sqrt{t} < \frac{c_{puct}}{d} \\ 1-(k-1)texp(-\frac{bd^2t}{16M}) & if \quad \frac{c_{puct}}{d} \leq \sqrt{t} \leq \frac{2c_{puct}}{d} \\ 1-(k-1)texp(-\frac{bc_{puct}d\sqrt{t}}{4M}) & if \quad \sqrt{t} > \frac{2c_{puct}}{d} \end{cases}$$

To prove this theorem we need to use the following lemma.

LEMMA A.1. *If* $X \sim N(\mu, \sigma^2)$, *then we have*

$$P(X - \mu \geq t) \leq e^{-\frac{t^2}{2\sigma^2}} \ \forall \ t \geq 0$$

PROOF. For any random variable $X$, $\mathbb{E}[X] = \mu$, we can define another random variable $Y = e^{\lambda(X-\mu)}$. By Markov's inequality we have

$$P(X - \mu \geq t) = P(e^{\lambda(X-\mu)} \geq e^{\lambda t}) \leq \frac{\mathbb{E}(e^{\lambda(X-\mu)})}{e^{\lambda t}}$$

When $X \sim N(\mu, \sigma^2)$, the moment generating function is

$$\mathbb{E}[e^{\lambda X}] = e^{\mu\lambda + \frac{\sigma^2\lambda^2}{2}}, \quad \forall \ \lambda$$

And we have

$$P(X-\mu \geq t) \leq \frac{\mathbb{E}(e^{\lambda(X-\mu)})}{e^{\lambda t}} = \frac{e^{\frac{\sigma^2\lambda^2}{2}}}{e^{\lambda t}} = e^{\frac{\sigma^2\lambda^2}{2}-\lambda t}$$

When $\lambda = \frac{t}{\sigma^2}$ the right side get its minimum. We have

$$P(X-\mu \geq t) \leq e^{-\frac{t^2}{2\sigma^2}} \ \forall\ t \geq 0$$

□

Now we can prove the theorem

PROOF. We use $N_1, N_2, ...N_k$ to represent the tested time of each choice when the decision is going to be made. $\hat{v_1}, \hat{v_2}...\hat{v_k}$ are the estimated values. Consider the probability that the $i^{th}$ choice is the one we finally pick. If we want to pick the $i^{th}$ choice we need

$$N_i = t,\ i = \text{argmax}_a \hat{v_a} + c_{puct}\frac{\sqrt{\sum N_j}}{N_a+1}$$

So

$$\begin{aligned}
& P(\text{Choose the } i^{th} \text{ choice}) \\
\leq\ & P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 \leq t) \\
=\ & \sum_{n=1}^{t} P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 = n) P(N_1 = n | N_i = t, N_1 \leq t) \\
\leq\ & t \times \max_n P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 = n)
\end{aligned}$$

Notice that the $t$ comes from $\sum_n = 1^t$. If we know the lower bound of $N_n$, which is $m_{c_{puct},k,t}$ mentioned before, $t$ can be replaced by $t - m_{c_{puct},k,t}$.

And for any $n \leq t$ we have

$$\begin{aligned} & P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1}|N_i = t, N_1 = n) \\ = \quad & P(\hat{v_i} - \hat{v_1} > c_{puct}\sqrt{\sum_j N_j}\frac{t-n}{(t+1)(n+1)}|N_i = t, N_1 = n) \\ = \quad & P(\hat{v_i} - \hat{v_1} + v_1 - v_i > c_{puct}\sqrt{\sum_j N_j}\frac{t-n}{(t+1)(n+1)} + v_1 - v_i|N_i = t, N_1 = n) \end{aligned}$$

Remember that

$$\hat{v_i} - \hat{v_1} + v_1 - v_i|N_i = t, N_1 = n \sim N(0, \frac{\sigma_1^2}{nb} + \frac{\sigma_i^2}{tb})$$

Apply the lemma we have

$$\begin{aligned} & P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1}|N_i = t, N_1 = n) \\ \leq \quad & \exp(-\frac{1}{2}\frac{c_{puct}^2(\sum_j N_j)\frac{(t-n)^2}{(n+1)^2(t+1)^2} + (v_1 - v_i)^2 + 2c_{puct}\sqrt{\sum_j N_j}\frac{t-n}{(n+1)(t+1)}(v_1 - v_i)}{\frac{\sigma_1^2}{nb} + \frac{\sigma_i^2}{tb}}) \end{aligned}$$

Consider the exponent and remember $\sigma_1^2 \leq M, \sigma_i^2 \leq M$

$$\begin{aligned} & -\frac{1}{2}\frac{c_{puct}^2(\sum_j N_j)\frac{(t-n)^2}{(n+1)^2(t+1)^2} + (v_1 - v_i)^2 + 2c_{puct}\sqrt{\sum_j N_j}\frac{t-n}{(n+1)(t+1)}(v_1 - v_i)}{\frac{\sigma_1^2}{nb} + \frac{\sigma_i^2}{tb}} \\ \leq \quad & -\frac{1}{2}\frac{c_{puct}^2(\sum_j N_j)\frac{(t-n)^2}{(n+1)^2(t+1)^2} + (v_1 - v_i)^2 + 2c_{puct}\sqrt{\sum_j N_j}\frac{t-n}{(n+1)(t+1)}(v_1 - v_i)}{\frac{M}{b}\frac{n+t}{nt}} \\ = \quad & -\frac{b}{2M}\frac{1}{n+t}[c_{puct}^2(\sum_j N_j)\frac{(t-n)^2nt}{(n+1)^2(t+1)^2} + nt(v_1 - v_i)^2 + 2c_{puct}\sqrt{\sum_j N_j}\frac{(t-n)nt}{(n+1)(t+1)}(v_1 - v_i)] \end{aligned}$$

We have $\frac{nt}{(n+1)(t+1)} \geq \frac{t}{2(t+1)}, \frac{nt}{(n+1)^2(t+1)^2} \geq \frac{t^2}{(t+1)^4}, \frac{1}{t+n} \geq \frac{1}{2t}\ \sum_j N_j \geq t$

$$\begin{aligned} \leq \quad & -\frac{b}{2M}\frac{1}{2t}[c_{puct}^2 t\frac{t^2}{(t+1)^4}(t-n)^2 + nt(v_1 - v_i)^2 + 2c_{puct}\sqrt{t}\frac{v_1 - v_i}{2}\frac{t}{t+1}(t-n)] \end{aligned}$$

Also remember that $v_1 - v_i \geq d$, we get

$$\begin{aligned} &\leq -\frac{b}{4M}[c_{puct}^2 \frac{t^2}{(t+1)^4}(t-n)^2 + nd^2 + c_{puct} d \frac{\sqrt{t}}{t+1}(t-n)] \\ &= -\frac{b}{4M}[\frac{c_{puct}^2 t^2}{(t+1)^4} n^2 + (d^2 - c_{puct} d \frac{\sqrt{t}}{t+1} - \frac{2c_{puct}^2 t^3}{(t+1)^4})n + \frac{c_{puct}^2 t^4}{(t+1)^4} + \frac{c_{puct} dt\sqrt{t}}{t+1}] \end{aligned}$$

Now we want to find the largest upper bound w.r.t $n$. So we need to minimize the quartic form in []

If $n$ can be any real number, it is minimized when

$$n = n^* = \frac{d^2 - c_{puct} d \frac{\sqrt{t}}{t+1} - \frac{2c_{puct}^2 t^3}{(t+1)^4}}{-\frac{2c_{puct}^2 t^2}{(t+1)^4}} = t + \frac{d(t+1)^3}{2c_{puct} t^{3/2}} - \frac{d^2}{2c_{puct}^2}\frac{(t+1)^4}{t^2}$$

① If $n^* > t$, which is $\frac{d(t+1)^3}{2c_{puct} t^{3/2}} - \frac{d^2}{2c_{puct}^2}\frac{(t+1)^4}{t^2} > 0$

This could happen if and only if $c_{puct}^2 - 4d^2 > 0$. And we get

$$\frac{c_{puct} - \sqrt{c_{puct}^2 - 4d^2}}{2d} < \sqrt{t} < \frac{c_{puct} + \sqrt{c_{puct}^2 - 4d^2}}{2d}.$$

When $d$ is relative small compared to $c_{puct}$, it is approximately $0 < \sqrt{t} < \frac{c_{puct}}{d}$

$$\begin{aligned} &P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i + 1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1 + 1} | N_i = t, N_1 = n) \\ \leq\ &\exp(-\frac{btd^2}{4M}) \end{aligned}$$

② If $n^* < 0$, which is $t + \frac{d(t+1)^3}{2c_{puct} t^{3/2}} - \frac{d^2}{2c_{puct}^2}\frac{(t+1)^4}{t^2} < 0$

This requires t to be large and for convenience we can use $\frac{t}{t+1} \approx 1$. This gives us.

$$
\begin{aligned}
& \sqrt{t} > \frac{2c_{puct}}{d} \text{ and } n^* = 1 \\
& P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 = n) \\
\leq \quad & \exp(-\frac{b}{4M}(c_{puct}^2 + c_{puct}d\sqrt{t})) \\
\text{③} \quad & \text{Otherwise } \frac{c_{puct}}{d} < \sqrt{t} < \frac{2c_{puct}}{d} \text{ and } n^* = t + \frac{d(t+1)^3}{2c_{puct}t^{3/2}} - \frac{d^2}{2c_{puct}^2}\frac{(t+1)^4}{t^2} \\
& P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 = n) \\
\leq \quad & \exp(-\frac{b}{4M}(-\frac{d^4(t+1)^4}{4c_{puct}^2t^2} - \frac{d^2(t+1)^2}{4t} + \frac{d^3(t+1)^3}{2c_{puct}t^{3/2}} + d^2t)) \\
& \text{Use } \frac{d}{2c} < \frac{1}{\sqrt{t}} < \frac{d}{c} \\
\leq \quad & \exp(-\frac{b}{4m}(-d^2\frac{(t+1)^4}{t^3} - \frac{d^2}{4}\frac{(t+1)^2}{t} + \frac{d}{2}\frac{(t+1)^3}{t^2}) + d^2t) \\
= \quad & \exp(-\frac{bd^2}{16Mt^3}(t^4 - 12t^3 - 19t^2 - 14t - 4)) \\
\approx \quad & \exp(-\frac{bd^2t}{16M})
\end{aligned}
$$

Finally we just have to notice that

$$
\begin{aligned}
& P(\text{Choose the } 1^{st} \text{ choice}) \\
= \quad & 1 - \sum_{i=2}^{k} P(\text{Choose the } i^{st} \text{ choice}) \\
\geq \quad & 1 - (k-1)t \times \max_n P(\hat{v_i} + c_{puct}\frac{\sqrt{\sum N_j}}{N_i+1} > \hat{v_1} + c_{puct}\frac{\sqrt{\sum N_j}}{N_1+1} | N_i = t, N_1 = n)
\end{aligned}
$$

The last probability is constrained by the above three conditions. So combining all above we finished the proof for the theorem. □

## A.2. Appendix for Chapter 3

Here we have more detalied results for our stage-wise segmentation and reconstruction.

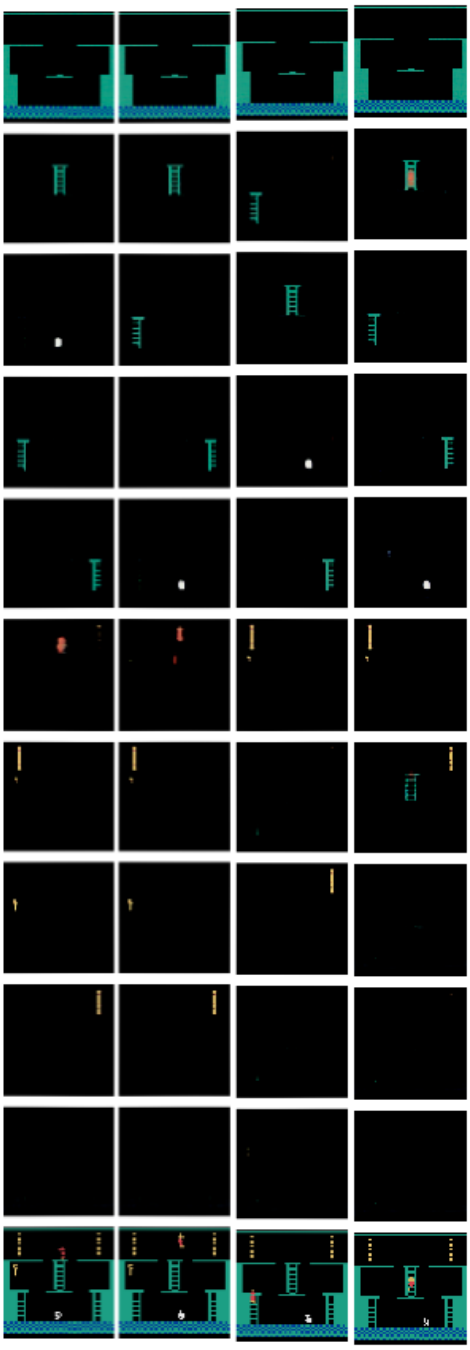


FIGURE A.1. Randomly picked samples of reconstructed image by objects for Montezuma's Revenge. Last row is input/ground truth image. These sample reconstruction images demonstrate that our model almost decomposes the scene perfectly into different objects.

## A.3. Appendix for Chapter 4

### A.3.1. Theorems and Proofs.

LEMMA A.1. *For hypothesis test in equation 4.1, under $H_0$, the distribution p-value converges to* $\mathcal{U}(0,1)$

PROOF. See **[60]** □

LEMMA A.2. *For random variables $X \sim Binom(N,p)$ and $Y \sim Binom(M,p)$, $X$ and $Y$ are independent. Further we define $n = X + Y$. Then for any fixed value of $n$ we have*

$$\lim_{N\to+\infty, M\to+\infty} \frac{P(X=k|X+Y=n)}{b(k,n,\frac{N}{M+N})} \rightrightarrows 1$$

PROOF.

$$\begin{aligned}
\mathbb{P}(X=k|X+Y=n) &= \frac{\mathbb{P}(X=k)\mathbb{P}(Y=n-k)}{\mathbb{P}(X+Y=n)} \\
&= \frac{\binom{N}{k}\binom{M}{n-k}}{\binom{M+N}{n}} \\
&= \frac{N!}{k!\,(N-k)!} \cdot \frac{M!}{(n-k)!\,(M-n+k)!} \cdot \frac{n!\,(M+N-n)!}{(M+N)!} \\
&= \frac{N!}{(N-k)!} \cdot \frac{M!}{(M-n+k)!} \cdot \frac{(M+N-n)!}{(M+N)!} \cdot \binom{n}{k} \\
b(k,n,\frac{N}{M+N}) &= \binom{n}{k}(\frac{N}{M+N})^k(\frac{M}{N+M})^{n-k} \\
\frac{\mathbb{P}(X=k|X+Y=n)}{b(k,n,\frac{N}{M+N})} &= \frac{N!}{N^k(N-k)!} \cdot \frac{M!}{M^{n-k}(M-n+k)!} \cdot \frac{(M+N)^n(M+N-n)!}{(M+N)!}
\end{aligned}$$

All three fractions on the right side will uniformly converges to 1. Here we only prove the last one. Clear we have

$$\frac{(M+N)^n(M+N-n)!}{(M+N)!} \geq 1$$

Then we have

$$\begin{aligned}
\lim_{M,N\to+\infty} \frac{(M+N)^n(M+N-n)!}{(M+N)!} &\leq \lim_{M,N\to+\infty} \frac{(M+N)^n}{(M+N-n+1)^n} \\
&= \lim_{M,N\to+\infty} (1+\frac{n-1}{M+N-n+1})^n \\
&= \lim_{M,N\to+\infty} (1+\frac{1}{\frac{M+N}{n-1}+1})^n \\
&= \lim_{M,N\to+\infty} (1+\frac{1}{\frac{M+N}{n-1}})^{\frac{M+N}{n-1}\cdot\frac{n(n-1)}{M+N}} \\
&= e^0 = 1
\end{aligned}$$

□

THEOREM A.1. *For any specific sample $x \in \mathcal{X}$, build its neighborhood $\mathcal{N}_x$ as described in 4.4.1. If for any sample $x' \in \mathcal{D}_1 \cup \mathcal{D}_2$, it has probability $p$ of being in $\mathcal{N}_x$, then for any fixed size of neighbor, the expectation of p-value will converge to* 0.5.

PROOF. Define

$$n_1 = |\{x' \in \mathcal{N}_x | x' \in \mathcal{D}_1\}|, N_1 = |\mathcal{D}_1|, N_2 = |\mathcal{D}_2|$$

we can apply Lemma A.2 and conclude that the distribution of, $n_1$ will converge to binomial distribution with probability $\frac{N_1}{N_1+N_2}$. So we know the test statistics will also converge to the distribution when $H_0$ is ture, which is $\mathcal{U}(0,1)$ according to Lemma A.1. So the expectation of p-value will converge to 0.5 □

COROLLARY A.1. *Still use the settings in Theorem A.1, but now the probability is no longer a constant, but follows a prior distribution $F$, i.e.*

$$\mathbb{P}(x' \in \mathcal{N}_x | x' \in \mathcal{D}_1 \cup \mathcal{D}_2) \sim F$$

*the conclusion still stands.*

PROOF. Have a prior over $p$ does not change the fact that all samples have the same probability of being in the neighborhood of $x$. Now the probability is just $\mathbb{E}[F]$. So we can apply Theorem A.1 and finish the proof. □

THEOREM A.2. *For any specific sample $x \in \mathcal{X}$, build its neighborhood $\mathcal{N}_x$ as described in 4.4.1, keep the notation of $n_1, n_2, N_1$ and $N_2$. If $\exists \epsilon_0 \in (0,1] s.t. \frac{N_2}{N_1} \in [\epsilon_0, \frac{1}{\epsilon_0}]$, We have $\frac{n_1}{n_1+n_2} \cdot \frac{N_1p_1+N_2p_2}{N_1p_1} \overset{a.s.}{\to} 1$*

PROOF. If $p_1 = 0$ or $p_2 = 0$, the conclusion is straight froward. We only proof for $p_1 \neq 0$ and $p_2 \neq 0$

$$\frac{n_1}{n_1+n_2} \cdot \frac{N_1p_1+N_2p_2}{N_1p_1} = \frac{n_1}{N_1p_1} \cdot \frac{N_1p_1+N_2p_2}{n_1+n_2} = \frac{n_1}{N_1p_1} \cdot \frac{p_1+p_2 \cdot \frac{N_2}{N_1}}{\frac{n_1}{N_1} + \frac{n_2}{N_2} \cdot \frac{N_2}{N_1}}$$

According to Law of Large Numbers **[15]**, we have

$$\mathbb{P}(\lim_{N_1 \to +\infty} \frac{n_1}{N_1} = p_1) = 1 \quad \mathbb{P}(\lim_{N_2 \to +\infty} \frac{n_2}{N_2} = p_2) = 1$$

So with probability 1, $\forall \eta > 0$, when both $N_1$ and $N_2$ are sufficiently large, with probability 1 we have

$$|\frac{n_1}{N_1} - p_1| < \eta, |\frac{n_2}{N_2} - p_2| < \eta$$

So we have

$$|(\frac{n_1}{N_1} + \frac{n_2}{N_2} \cdot \frac{N_2}{N_1}) - (p_1 + p_2 \cdot \frac{N_2}{N_1})| = |(\frac{n_1}{N_1} - p_1) + \frac{N_2}{N_1}(\frac{n_2}{N_2} - p_2)| \leq (1 + \frac{1}{\epsilon_0})\eta$$

We also know that $0 < p_1 + p_2\epsilon_0 \leq p_1 + \frac{N_2}{N_1}p_2 \leq p_1 + \frac{p_2}{\epsilon_0}$. So with probability equal to 1 we have

$$\lim_{N_1, N_2 \to +\infty} \frac{p_1 + p_2 \cdot \frac{N_2}{N_1}}{\frac{n_1}{N_1} + \frac{n_2}{N_2} \cdot \frac{N_2}{N_1}} = 1$$

$$\lim_{N_1 \to +\infty} \frac{n_1}{N_1p_1} = 1$$

$$\lim_{N_1, N_2 \to +\infty} \frac{n_1}{n_1+n_2} \cdot \frac{N_1p_1+N_2p_2}{N_1p_1} = 1$$

□

COROLLARY A.2. *Still use the settings of Theorem A.2, but now $p_1$ and $p_2$ follows prior distribution $F_1$ and $F_2$ separately, the conclusion still stands. All we have to do is to replace $p_1$ and $p_2$ with $\mathbb{E}[F_1]$ and $\mathbb{E}[F_2]$*

PROOF. Have a prior over $p_1$ and $p_2$ does not change the fact that each sample will be selected randomly and independently. Now in $\mathcal{D}_1$ the samples will be selected with probability $\mathbb{E}[F_1]$ and in $\mathcal{D}_2$ it's $\mathbb{E}[F_2]$. So we can apply Theorem A.2 and finish the proof. □

THEOREM A.3. *For any sample $x \in \mathcal{X}$, build its neighborhoods with the steps described in 4.4.1 and calculate their p-values for the hypothesis test described in 4.4.1. If the two datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ have a bounded size ratio and have the same distribution over $\mathcal{X}$, then with large enough datasets and a pre-selected similarity function $s(x_1, x_2)$, all these p-values will converge to 1 in probability.*

PROOF. Consider we have one specific sample $x$. For another sample $x'$ that is randomly selected from $\mathcal{D}_1$, it has a probability $p = s(x, x')$ of being in $\mathcal{N}_x$, the neighborhood of $x$. Clear this probability is also random and its distribution relies the choice of $s$ and the distribution of $\mathcal{D}_1$ over $\mathcal{X}$. With both specified, it's a fixed prior. Now if the random sample comes from $\mathcal{D}_2$, the distribution for $p = s(x, x')$ is still the same because $\mathcal{D}_1$ and $\mathcal{D}_2$ has the same distribution over $\mathcal{X}$. So we can apply Corollary A.2 with $F_1$ and $F_2$ being identical. So we have

$$\frac{n_1}{n_1 + n_2} \cdot \frac{N_1 p_1 + N_2 p_2}{N_1 p_1} \xrightarrow{a.s.} 1$$

This means $\forall \delta > 0$ when $N_1$ and $N_2$ are sufficiently large, with probability 1

$$\left|\frac{n_1}{n_1 + n_2} - \frac{N_1 p_1}{N_1 p_1 + N_2 p_2}\right| \leq \delta \left|\frac{N_1 p_1}{N_1 p_1 + N_2 p_2}\right|$$

Since $\frac{N_1}{N_2}$ and $\frac{N_2}{N_1}$ are bounded, we further know that this different can be arbitrarily small. So $\forall \delta > 0$ when $N_1$ and $N_2$ are sufficiently large, with probability 1

$$\left|\frac{n_1}{n_1 + n_2} - \frac{N_1 p_1}{N_1 p_1 + N_2 p_2}\right| \leq \delta$$

Now for any $\epsilon > 0, \exists \delta > 0$ such that when the test statistics $|T| < \delta$, we will have $|\text{p-value} - 1| < \epsilon$. And now $|T| < \delta$ only needs $\frac{n_1}{n_1+n_2}$ be very close to $\frac{N_1}{N_1+N_2}$, since all other components in equation 4.3 are bounded. So $\forall \epsilon > 0$, with probability 1, we have

$$\lim_{N_1, N_2 \to +\infty} \mathbb{P}(|\text{p-value} - 1| < \epsilon) = 1$$

So the p-value of sample $x$ converges to 1 in probability. Thus this is also correct for any finite number of samples. □

### A.3.2. Full Results for Domain Adaptation.

A.3.2.1. *Other Distance Metrics.* We compared various distance and similarity measures to analyze if they are good indicators of the domain adaptation transfer effect. In addition to our framework of similarity, $IP_\alpha$ in [**1**] and OTDD in [**2**], we also compared several traditional metrics as listed in Section 3 in [**32**]. These metrics are defined as follows.

- Mean Based Metrics: These methods are based on mean of sample embeddings. $\mathcal{L}_2$ **Distance** finds the mean vector for both datasets, $\hat{\mu}_1 = \frac{\sum_{i=1}^{n1} d_{1i}}{n_1}$ and $\hat{\mu}_2 = \frac{\sum_{i=1}^{n2} d_{2i}}{n_2}$ and calculates $||\hat{\mu}_1 - \hat{\mu}_2||^2$ as the distance. **Cosine Similarity** defines similarity as $\frac{\hat{\mu_1} \cdot \hat{\mu_2}}{||\hat{\mu_1}|| \cdot ||\hat{\mu_2}||}$. These metrics are unreliable for two reasons: average calculation will ignore all shape information of the distribution, and also assumes the embedding space to be Euclidean and isotropic.
- Covariance Based Metric: Another method uses covariance information, **CORAL, correlation alignment** [**81**]. It defines the distance based on the distance between covariance matrices.

  $$d(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{4d}||S_1 - S_2||_F^2$$

  Here, $S1$ and $S2$ are covariance matrices for two datasets and $d$ is the dimension of embedding. $||\cdot||_F^2$ is Frobenius norm for matrix. Like the previous two metrics, CORAL also relies on some geometric assumptions of the embedding space.
- Fisher Linear Discriminant: **Fisher Linear Discriminant (FLD)** attempts to find a projection that will maximize the distance between groups while preventing within group distance from becoming too large. Mathematically, it maximizes $\frac{w^\top S_B w}{w^\top S_W w}$, where $S_B$ is the between-dataset covariance matrix $S_B = (\hat{\mu}_1 - \hat{\mu}_2)^\top (\hat{\mu}_1 - \hat{\mu}_2)$ and $S_W$ is within-dataset covariance matrix $S_W = \sum_{i=1}^{n_1} (x_{1i} - \hat{\mu}_1)^\top (x_{1i} - \hat{\mu}_1) + \sum_{i=1}^{n_2} (x_{2i} - \hat{\mu}_2)^\top (x_{2i} - \hat{\mu}_2)$. This method has good statistical properties but the maximization might be sensitive to noise when two datasets are similar.
- Maximum Mean Discrepancy: **Maximum Mean Discrepency(MMD)** is a common solution. For any class of function $\mathcal{F}$, MMD is defined as

  $$MMD_{\mathcal{F}}[\mathcal{D}_1, \mathcal{D}_2] = \sup_{f \in \mathcal{F}} \mathbb{E}_x[f(x)] - \mathbb{E}_x[f(x)]$$

The optimization is not trivial but a closed form solution exists when $\mathcal{F}$ is the unit ball in a reproducing kernel Hilbert space, introduced in the 1907 work of Stanislaw Zaremba. The closed form solution has an unbiased estimation with its characteristic kernel $k$.

$$MMD^2_{\mathcal{F}}[\mathcal{D}_1, \mathcal{D}_2] = \frac{1}{n_1^2}\sum_{i=1}^{n_1}\sum_{i'=1}^{n_1} k(x_{1i}, x_{1i'}) - \frac{2}{n_1 n_2}\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} k(x_{1i}, x_{2j}) + \frac{1}{n_2^2}\sum_{j=1}^{n_2}\sum_{j'=1}^{n_2} k(x_{2j}, x_{2j'})$$

This method is sound in theory but would have to calculate all pairwise output through the kernel function, which is not desirable. Besides, the distance requires proper choice of the right class of function, which cannot be guaranteed.

A.3.2.2. *Experiment Results.* Full results are available in Table A.3, Table A.4 and Figure A.3. We highlight the two best correlations. We can see that FLD is consistently the best under all settings. Our method is the second best w.r.t. Spearman's correlation. We prefer Spearman's correlation as it is more robust and we have no specific reason to believe a linear relationship exists.

**Without Fine-tuning**

TABLE A.1. Absolute values of Correlation with Drop in Accuracy without fine-tuning

| Metric | Ours $\bar{p}_1$ | Ours $\bar{p}$ | $IP_\alpha$ | OTDD | $\mathcal{L}_2$ | cosine | MMD | FLD | CORAL |
|---|---|---|---|---|---|---|---|---|---|
| Pearson $r$ | 0.678 | 0.646 | 0.539 | 0.680 | 0.667 | **0.697** | 0.691 | **0.789** | 0.521 |
| Spearman $\rho$ | **0.867** | 0.862 | 0.671 | 0.827 | 0.830 | 0.830 | 0.830 | **0.886** | 0.685 |

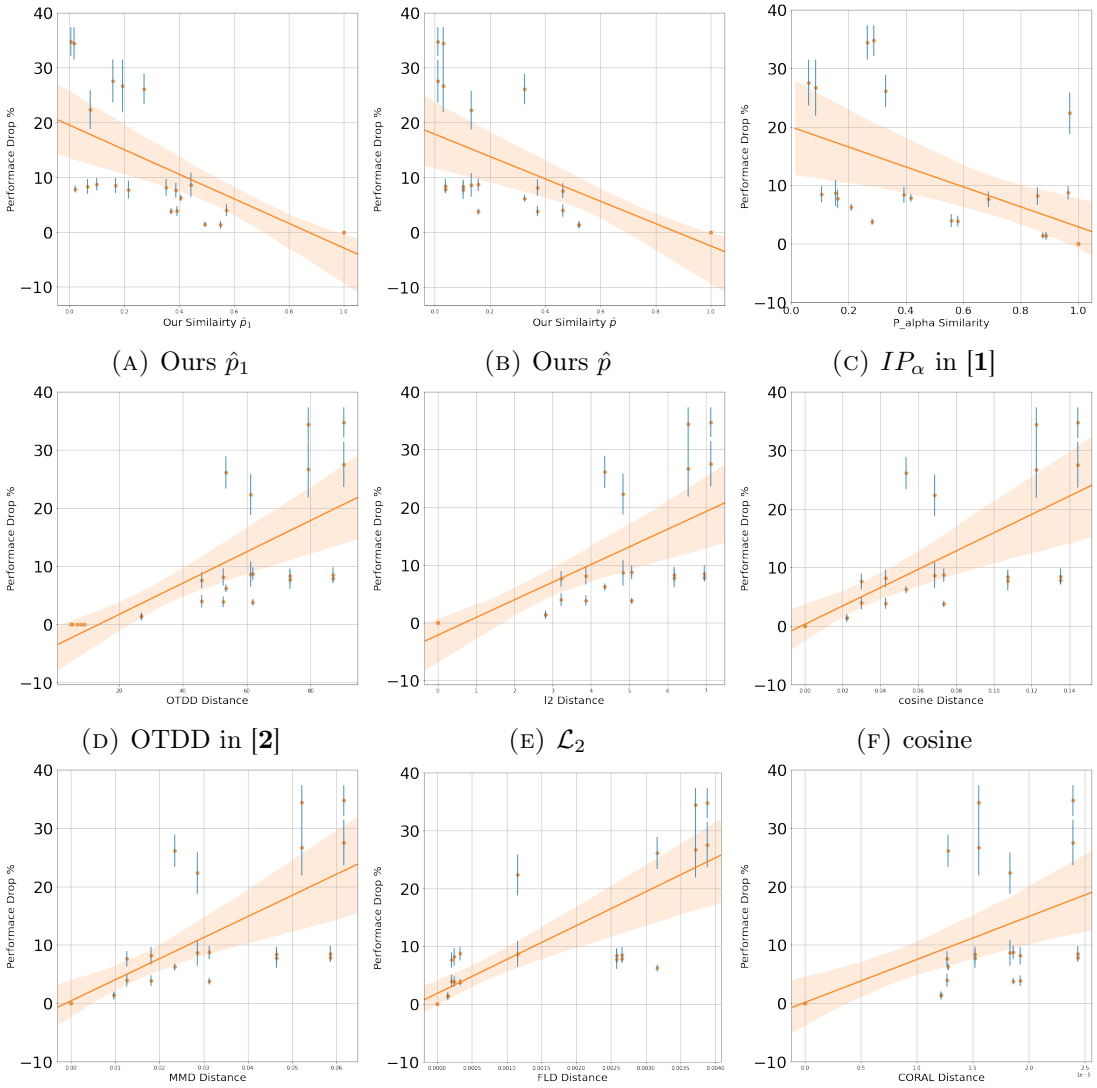
TABLE A.2. Absolute values of Correlation with Increase in Loss without fine-tuning

| Metric | Ours $\bar{p}_1$ | Ours $\bar{p}$ | $IP_\alpha$ | OTDD | $\mathcal{L}_2$ | cosine | MMD | FLD | CORAL |
|---|---|---|---|---|---|---|---|---|---|
| Pearson $r$ | 0.596 | 0.572 | 0.504 | 0.610 | 0.598 | **0.646** | 0.640 | **0.757** | 0.447 |
| Spearman $\rho$ | 0.817 | **0.862** | 0.727 | 0.816 | 0.819 | 0.819 | 0.819 | **0.920** | 0.659 |

**With Fine-tuning**

TABLE A.3. Absolute values of Correlation with Drop in Accuracy with finetuning

| Metric | Ours $\bar{p}_1$ | Ours $\bar{p}$ | $IP_\alpha$ | OTDD | $\mathcal{L}_2$ | cosine | MMD | FLD | CORAL |
|---|---|---|---|---|---|---|---|---|---|
| Pearson $r$ | **0.784** | **0.685** | 0.301 | 0.666 | 0.658 | 0.565 | 0.563 | 0.404 | 0669 |
| Spearman $\rho$ | **0.805** | 0.615 | 0.359 | 0.634 | **0.637** | **0.637** | **0.637** | 0.600 | 0.634 |




(A) Ours $\hat{p}_1$ (B) Ours $\hat{p}$ (C) $IP_\alpha$ in **[1]**

(D) OTDD in **[2]** (E) $\mathcal{L}_2$ (F) cosine

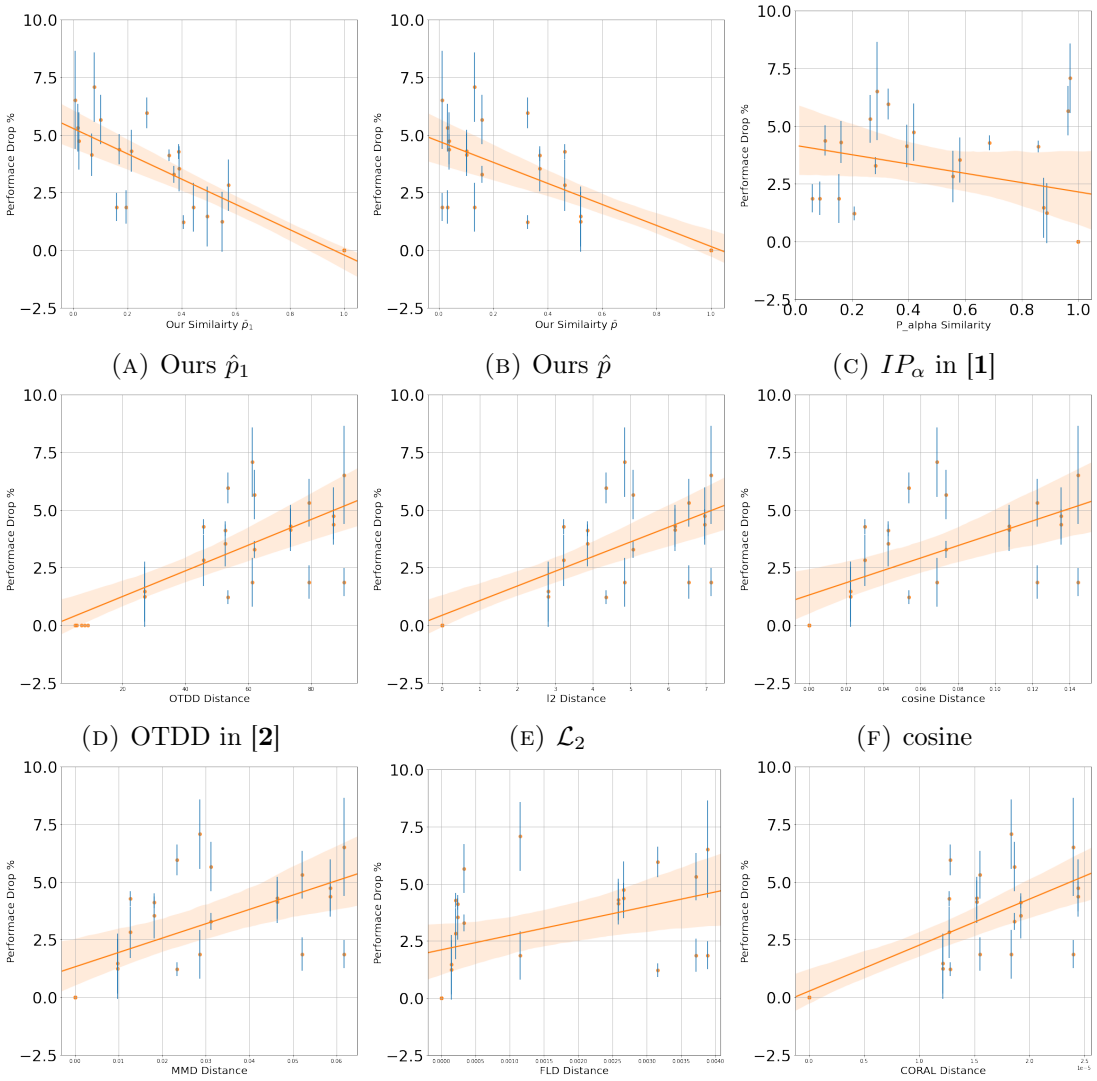
(G) MMD (H) FLD (I) CORAL

FIGURE A.2. Full results for Domain Adaptation without fine-tuning

TABLE A.4. Absolute values of Correlation with Increase in Loss with finetuning

| Metric | Ours $\bar{p}_1$ | Ours $\bar{p}$ | $IP_\alpha$ | OTDD | $\mathcal{L}_2$ | cosine | MMD | FLD | CORAL |
|---|---|---|---|---|---|---|---|---|---|
| Pearson $r$ | **0.825** | **0.766** | 0.435 | 0.741 | 0.734 | 0.631 | 0.629 | 0.481 | 0.732658 |
| Spearman $\rho$ | **0.794** | 0.649 | 0.425 | 0.661 | **0.663** | **0.663** | **0.663** | 0.635 | 0.659 |

**A.3.3. Problems with $P_\alpha$ and $R_\beta$.** The work in **[1]**, that proposed $P_\alpha$ and $R_\beta$, ignores the distributions for samples' angles in the sphere. To examine if angles can be ignored, we trained a Deep One Class SVM **[72]** encoder and used hyper-parameters in **[1]**: $\mu = 0.01, c = \mathbf{1}, d_z = 32$. The




(A) Ours $\hat{p}_1$ (B) Ours $\hat{p}$ (C) $IP_\alpha$ in **[1]**

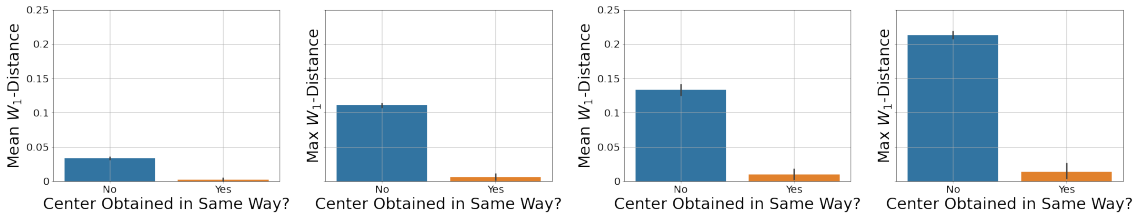
(D) OTDD in **[2]** (E) $\mathcal{L}_2$ (F) cosine

(G) MMD (H) FLD (I) CORAL

FIGURE A.3. Full results for Domain Adaptation with fine-tuning

network structure is a frozen BERT model followed by a two-layer feed forward network without bias terms. The hidden layer has 128 neurons and BERT's output embedding on [CLS] is the input to the feed forward network.

The encoder is trained on a training set with 69349 hotel reviews of London from OpinRank dataset. Two test sets contains the remaining10000 reviews of London, and 11834 reviews of Dubai, respectively. We obtained their embeddings and then calculate their polar system coordinates to get their 31-dimensional angles. For each dataset, we calculate two sets of angles. One is obtained with




(A) normalized w.r.t. length of domain (B) normalized w.r.t. effective range

FIGURE A.4. Mean and max of $W_1$ distance over 31 angle dimensions with different normalization, for all 15 possible pairs of set of angles

**1** used as center of sphere, and the other with a fitted center from embeddings. So we have 6 sets of angles in total.

Then, we calculate the average and max Wasserstein-1 distance ($W_1$) over 31 dimensions. We chose $W_1$ because it has a straight forward geometric meaning: the smallest average distance each sample has to be moved in order to make two distributions the same. Since different angle dimensions have different ranges, we normalize $W_1$ distance fo each dimension w.r.t their length of domain ($\pi$ or $2\pi$), or the effective range, which is defined as the length of their 95% confidence intervals as many dimensions take only a small proportion of their domain.

The results are available in Figure A.4. One disturbing finding is that it is not that the pair of datasets has a major impact on $W_1$ distance, but that how the centers are chosen for them. We can see the distance is almost 0 when two datasets use the same way to obtain their center, i.e. both use **1** or both fitted with the embeddings. But when only one of them uses **1** and the other is fitted, the distance is multiple times larger and cannot be ignored. In the paper that proposed $P_\alpha$ and $R_\beta$, $\mathcal{D}_1$ uses **1** while $\mathcal{D}_2$ uses fitted center, which means there is likely to be a significant shift in angle distribution, which is ignored in their metrics.

**A.3.4. Problem with Authentic Check.** In order to check authenticity of a generated sample, the authors made an assumption that generated unauthentic sample's embedding should lie within a small neighborhood of the original real sample because it is only a slight variation of the original sample.

$$x^{\text{unauthentic}} \sim \mathcal{N}(x^{\text{original}}, \epsilon)$$

Ideally $\epsilon$ could be small enough that we will find

$$||x_{1i^*} - x_{1j^*}|| \geq ||x^{\text{unauthentic}} - x_{1i^*}||$$

for unauthentic samples. While for authentic samples, if the original dataset have enough samples and becomes 'dense' enough, $||x_{1i^*} - x_{1j^*}||$ will be small enough such that

$$||x_{1i^*} - x_{1j^*}|| < ||x^{\text{authentic}} - x_{1i^*}||$$

The contradiction here is, both 'small $\epsilon$' assumption and 'dense' dataset assumption are to make one of the two distances arbitrary small. But they cannot be both arbitrary small. So this method can at most be good at one thing: high recall for unauthentic samples, or high precision for authentic samples.

To further validate their idea, we tested this method on **Imdb Review** dataset and **Noisy Imdb** dataset. We only compare the training set, which contains 25000 samples. We already know that **Noisy Imdb** differs from **Imdb Review** only by some data cleaning process, so no sample should be authentic if we take it as generated samples.

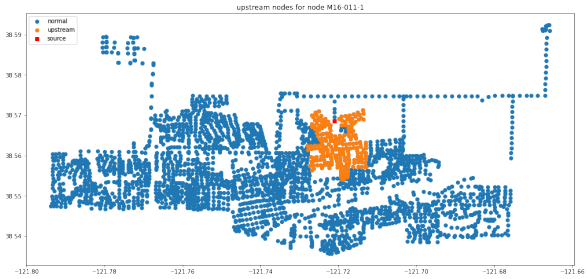
We train a encoder with the same structure as described above with the same hyper-parameters. As the order of two datasets are not the same, we use edit distance. For each sample in **Noisy Imdb**, we take its first 30 characters and find the sample in **Imdb Review** whose first 30 characters has the shortest edit distance. Then for this selected sample in **Imdb Review**, we check the edit distance between the whole string with the sample in **Noisy Imdb**. We consider it a success pair if the edit distance between whole strings are less than 10. Using this strategy, we find the original sample for 21497 samples out of 25000. And for the 25000 samples in **Noisy Imdb**, we encode them and find the nearest neighborhood in embedded space **Imdb Review** using Frobenius norm and also check if they are authentic using method in [**1**].

It turns out that 17105, about 68.4% samples in **Noisy Imdb** are considered as authentic. For samples that we have matched an original sample with high confidence, only 6700 are the same as their nearest neighborhood found in embedding space. We also have more results for different threshold of edit distance (see Table A.5). This simply shows that their method needs to be used with great caution.

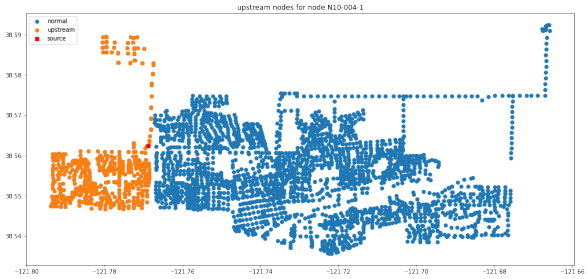
TABLE A.5. Number of matched Nearest neighborhood for different values of threshold for edit distance

| Threshold for Edit Distance | 5 | 10 | 20 | 30 | 50 | 100 |
|---|---|---|---|---|---|---|
| Number of matched Nearest neighborhood | 6005 | 6700 | 6843 | 6857 | 6857 | 6857 |

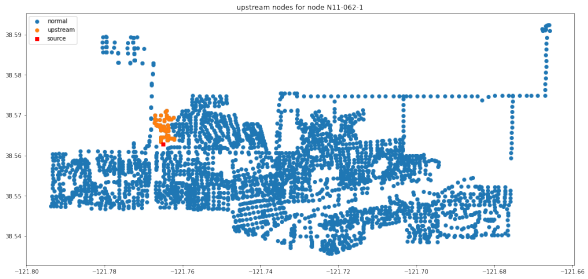
## A.4. Appendix for Chapter 5



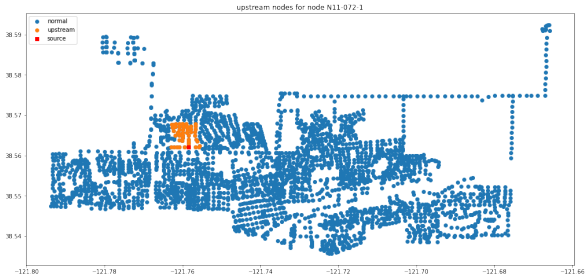

(A) Collection Point 1



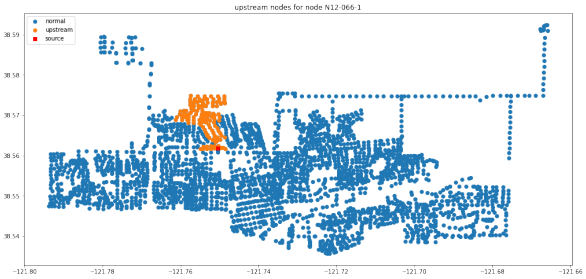

(B) Collection Point 2



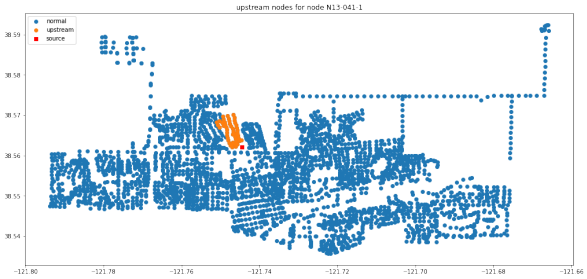

(C) Collection Point 3



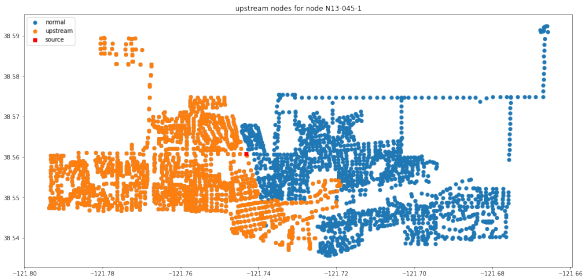

(D) Collection Point 4



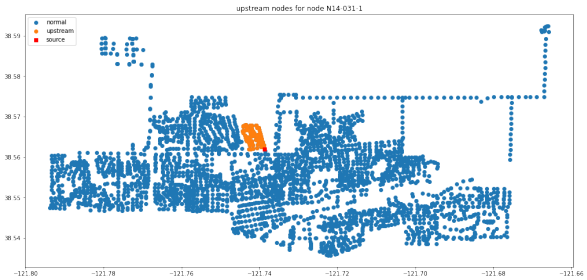

(A) Collection Point 5



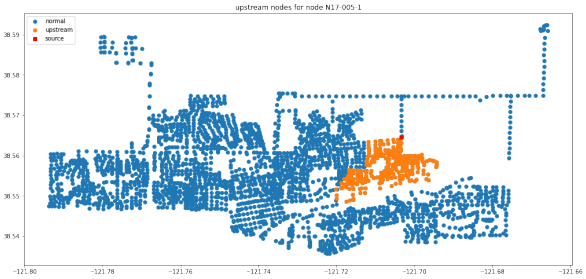

(B) Collection Point 6



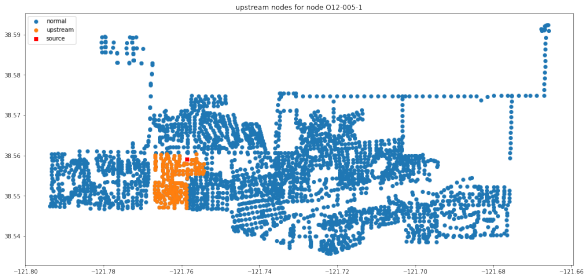

(C) Collection Point 7



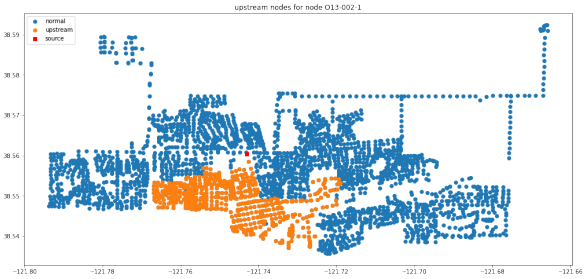

(D) Collection Point 8



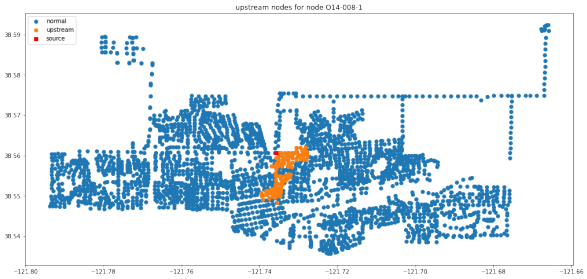

(E) Collection Point 9



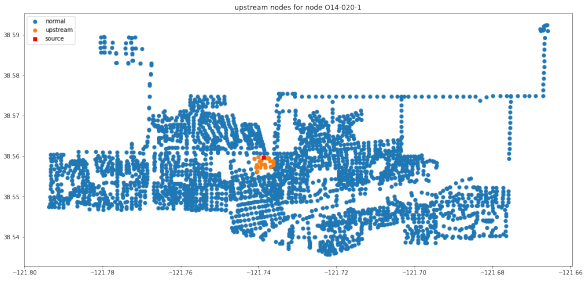

(F) Collection Point 10



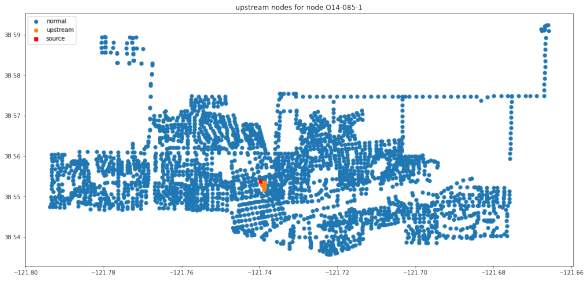

(G) Collection Point 11



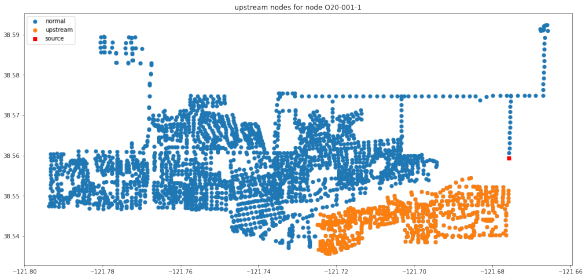

(H) Collection Point 12



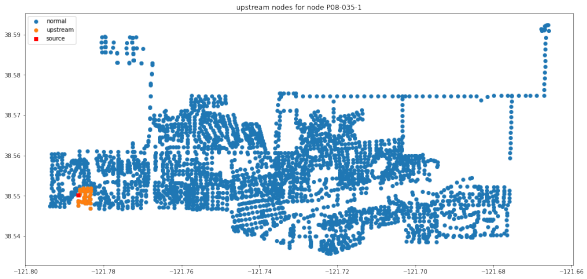

(I) Collection Point 13



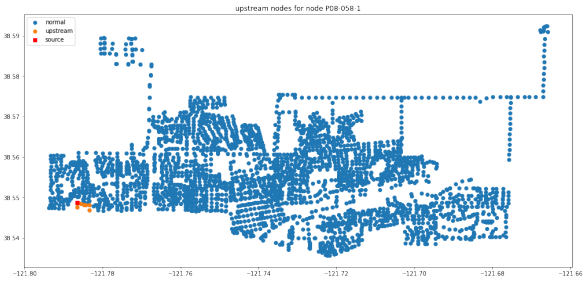

(J) Collection Point 14



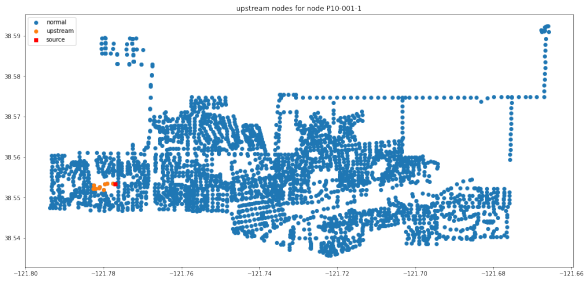

(A) Collection Point 15



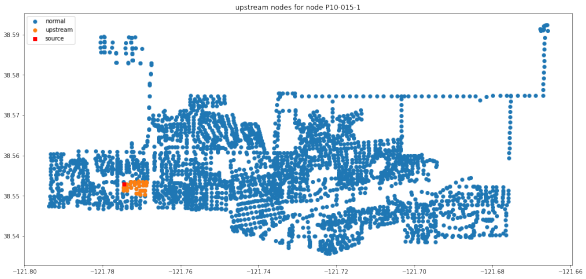

(B) Collection Point 16



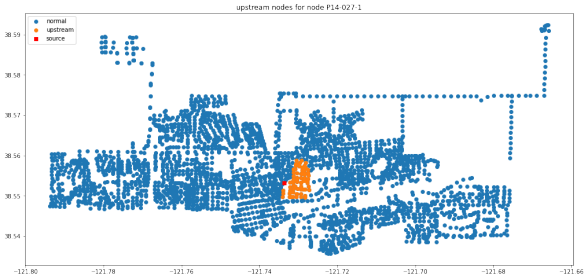

(C) Collection Point 17



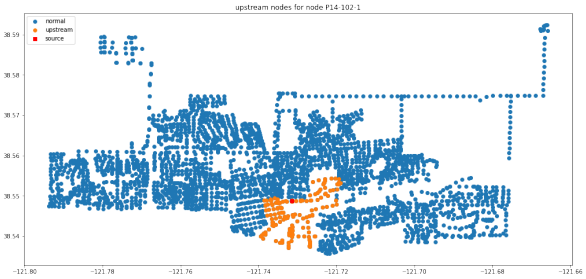

(D) Collection Point 18



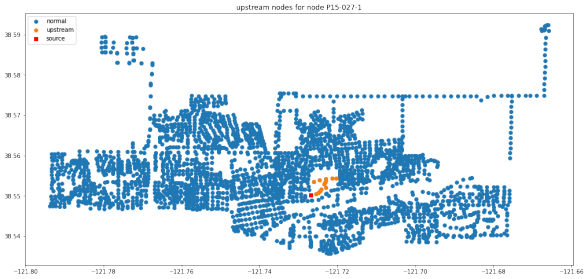

(E) Collection Point 19



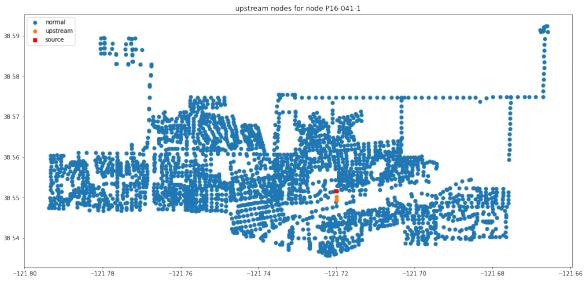

(F) Collection Point 20



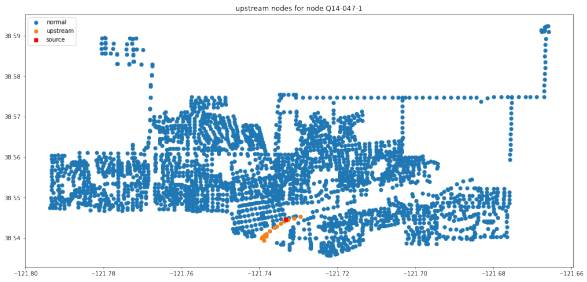

(G) Collection Point 21




(H) Collection Point 22




(I) Collection Point 23




(J) Collection Point 24

# Bibliography


[1] A. M. Alaa, B. van Breugel, E. Saveliev, and M. van der Schaar, *How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models*, 2021.

[2] D. Alvarez-Melis and N. Fusi, *Geometric dataset distances via optimal transport*, in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds., vol. 33, Curran Associates, Inc., 2020, pp. 21428–21439.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, *Finite-time analysis of the multiarmed bandit problem*, Machine Learning, 47 (2002), pp. 235–256.

[4] G. Avarikioti, R. Brunner, A. Kiayias, R. Wattenhofer, and D. Zindros, *Structure and content of the visible darknet*, 2018.

[5] J. Blitzer, M. Dredze, and F. Pereira, *Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification*, in Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Prague, Czech Republic, June 2007, Association for Computational Linguistics, pp. 440–447.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020.

[7] M. Bonvini, E. Kennedy, V. Ventura, and L. Wasserman, *Causal inference in the time of covid-19*, 2021.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016.

[9] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner, *Monet: Unsupervised scene decomposition and representation*, arXiv preprint arXiv:1901.11390, (2019).

[10] S. Butterworth, *On the theory of filter amplifiers*, Experimental Wireless and the Wireless Engineer, 7 (1930), pp. 536,541.

[11] S. Chang, E. Pierson, P. Koh, J. Gerardin, B. Redbird, D. Grusky, and J. Leskovec, *Mobility network models of covid-19 explain inequities and inform reopening*, Nature, 589 (2021), pp. 82–87. Funding Information: Acknowledgements We thank Y.-Y. Ahn, R. Appel, C. Chen, J. Feng, N. Fishman, S. Fullerton, T. Hashimoto, M. Kraemer, P. Liang, M. Lipsitch, K. Loh, D. Ouyang, R. Rosenfeld, S. Sagawa, J. Steinhardt, R. Tibshirani, J. Ugander, D. Vrabac, seminar participants and Stanford's Computer Science and Civil Society for support and comments; and N. Singh, R. F. Squire, J. Williams-Holt, J. Wolf, R. Yang and others at SafeGraph for mobile phone mobility data and feedback. This research was supported by US National Science Foundation under

OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), IIS-2030477 (RAPID), Stanford Data Science Initiative, Wu Tsai Neurosciences Institute and Chan Zuckerberg Biohub. S.C. was supported by an NSF Fellowship. E.P. was supported by a Hertz Fellowship. P.W.K. was supported by the Facebook Fellowship Program. J.L. is a Chan Zuckerberg Biohub investigator. 

[12] M. Chen, T. Artières, and L. Denoyer, *Unsupervised object segmentation by redrawing*, arXiv preprint arXiv:1905.13539, (2019).

[13] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, *A survey of model compression and acceleration for deep neural networks*, CoRR, abs/1710.09282 (2017).

[14] Y. Choi, M. El-Khamy, and J. Lee, *Universal deep neural network compression*, CoRR, abs/1802.02271 (2018).

[15] K. L. Chung, *Elementary probability theory with stochastic processes*, Springer Science & Business Media, 2012.

[16] E. Clark, A. Celikyilmaz, and N. A. Smith, *Sentence mover's similarity: Automatic evaluation for multi-sentence texts*, in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, July 2019, Association for Computational Linguistics, pp. 2748–2760.

[17] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, *Electra: Pre-training text encoders as discriminators rather than generators*, 2020.

[18] J. Delon and A. Desolneux, *A wasserstein-type distance in the space of gaussian mixture models*, 2020.

[19] A. Delong and Y. Boykov, *A scalable graph-cut algorithm for nd grids*, in 2008 IEEE Conference on Computer Vision and Pattern Recognition, 2008.

[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, *Exploiting linear structure within convolutional networks for efficient evaluation*, in Advances in neural information processing systems, 2014, pp. 1269–1277.

[21] J. Devlin, M. Chang, K. Lee, and K. Toutanova, *BERT: pre-training of deep bidirectional transformers for language understanding*, CoRR, abs/1810.04805 (2018).

[22] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, *Implementation matters in deep policy gradients: A case study on ppo and trpo*, 2020.

[23] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, *Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures*, 2018.

[24] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, *Diversity is all you need: Learning skills without a reward function*, 2018.

[25] C. Frogner, F. Mirzazadeh, and J. Solomon, *Learning entropic wasserstein embeddings*, in International Conference on Learning Representations, 2019.

[26] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018.

[27] R. Girshick, *Fast r-cnn*, 2015.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014.

[29] K. Greff, R. L. Kaufman, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner, *Multi-object representation learning with iterative variational inference*, in Proceedings of the 36th International Conference on Machine Learning, K. Chaudhuri and R. Salakhutdinov, eds., vol. 97 of Proceedings of Machine Learning Research, Long Beach, California, USA, 09–15 Jun 2019, PMLR, pp. 2424–2433.

[30] K. Greff, S. van Steenkiste, and J. Schmidhuber, *Neural expectation maximization*, in Advances in Neural Information Processing Systems, 2017, pp. 6691–6701.

[31] M. G. G'Sell, S. Wager, A. Chouldechova, and R. Tibshirani, *Sequential selection procedures and false discovery rate control*, Journal of the Royal Statistical Society: Series B: Statistical Methodology, (2016), pp. 423–444.

[32] H. Guo, R. Pasunuru, and M. Bansal, *Multi-source domain adaptation for text classification via distancenet-bandits*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 7830–7838.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018.

[34] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*, CoRR, abs/1510.00149 (2015).

[35] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, CoRR, abs/1506.02626 (2015).

[36] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, CoRR, abs/1512.03385 (2015).

[37] K.-J. Hsu, Y.-Y. Lin, and Y.-Y. Chuang, *Co-attention cnns for unsupervised object co-segmentation*, in Proceedings of the 27th International Joint Conference on Artificial Intelligence, AAAI Press, 2018, pp. 748–756.

[38] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2018.

[39] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, arXiv preprint arXiv:1609.07061, (2016).

[40] M. Jaderberg, A. Vedaldi, and A. Zisserman, *Speeding up convolutional neural networks with low rank expansions*, arXiv preprint arXiv:1405.3866, (2014).

[41] R. Kabra, C. Burgess, L. Matthey, R. L. Kaufman, K. Greff, M. Reynolds, and A. Lerchner, *Multi-object datasets*. https://github.com/deepmind/multi-object-datasets/, 2019.

[42] A. Kanezaki, *Unsupervised image segmentation by backpropagation*, in Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2018.

[43] S. Khoram and J. Li, *Adaptive quantization of neural networks*, in ICLR, 2018.

[44] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv preprint arXiv:1312.6114, (2013).

[45] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, *Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation*, in Advances in neural information processing systems, 2016, pp. 3675–3683.

[46] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, *From word embeddings to document distances*, in Proceedings of the 32nd International Conference on Machine Learning, F. Bach and D. Blei, eds., vol. 37 of Proceedings of Machine Learning Research, Lille, France, 07–09 Jul 2015, PMLR, pp. 957–966.

[47] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, *Albert: A lite bert for self-supervised learning of language representations*, 2020.

[48] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, Cambridge University Press, USA, 2nd ed., 2014.

[49] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*, 2019.

[50] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, 2019.

[51] C.-Y. Lin, *ROUGE: A package for automatic evaluation of summaries*, in Text Summarization Branches Out, Barcelona, Spain, July 2004, Association for Computational Linguistics, pp. 74–81.

[52] D. Lin, S. Talathi, and S. Annapureddy, *Fixed point quantization of deep convolutional networks*, in International Conference on Machine Learning, 2016, pp. 2849–2858.

[53] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, *Focal loss for dense object detection*, 2018.

[54] P. Liu, X. Qiu, and X. Huang, *Adversarial multi-task learning for text classification*, CoRR, abs/1704.05742 (2017).

[55] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[56] J. Luo, J. Wu, and W. Lin, *Thinet: A filter level pruning method for deep neural network compression*, CoRR, abs/1707.06342 (2017).

[57] D. Lyu, F. Yang, B. Liu, and S. Gustafson, *Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 2970–2977.

[58] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, and R. Schettini, *Automated pruning for deep neural network compression*, CoRR, abs/1712.01721 (2017).

[59] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning*, Nature, 518 (2015), pp. 529–533.

[60] D. Murdoch, Y.-L. Tsai, and J. Adcock, *P-values are random variables*, The American Statistician, 62 (2008), pp. 242 – 245.

[61] B. Muzellec and M. Cuturi, *Generalizing point embeddings using the wasserstein space of elliptical distributions*, in Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds., vol. 31, Curran Associates, Inc., 2018.

[62] M.-E. Nilsback and A. Zisserman, *Automated flower classification over a large number of classes*, in Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Dec 2008.

[63] D. Novotny, S. Albanie, D. Larlus, and A. Vedaldi, *Semi-convolutional operators for instance segmentation*, in The European Conference on Computer Vision (ECCV), September 2018.

[64] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. D?biak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Jozefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, *Dota 2 with large scale deep reinforcement learning*, 2019.

[65] G. Opfer and G. D. Knott, *Interpolating cubic splines*, J. Approx. Theory, 112 (2001), pp. 319–321.

[66] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, *Bleu: a method for automatic evaluation of machine translation*, in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, July 2002, Association for Computational Linguistics, pp. 311–318.

[67] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, *Learning features by watching objects move*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2701–2710.

[68] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language models are unsupervised multitask learners*, (2019).

[69] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016.

[70] N. Reimers and I. Gurevych, *Sentence-bert: Sentence embeddings using siamese bert-networks*, CoRR, abs/1908.10084 (2019).

[71] F. M. Rueda, R. Grzeszick, and G. A. Fink, *Neuron pruning for compressing deep networks using maxout architectures*, CoRR, abs/1707.06838 (2017).

[72] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, *Deep one-class classification*, in Proceedings of the 35th International Conference on Machine Learning, J. Dy and A. Krause, eds., vol. 80 of Proceedings of Machine Learning Research, PMLR, 10–15 Jul 2018, pp. 4393–4402.

[73] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), 115 (2015), pp. 211–252.

[74] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 6655–6659.

[75] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter*, 2020.

[76] T. Sellam, D. Das, and A. P. Parikh, *Bleurt: Learning robust metrics for text generation*, arXiv preprint arXiv:2004.04696, (2020).

[77] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484 EP –. Article.

[78] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[79] B. K. Sriperumbudur, K. Fukumizu, and G. R. G. Lanckriet, *Universality, characteristic kernels and rkhs embedding of measures*, 2010.

[80] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Scholkopf, and G. R. G. Lanckriet, *Hilbert space embeddings and metrics on probability measures*, 2010.

[81] B. Sun, J. Feng, and K. Saenko, *Correlation alignment for unsupervised domain adaptation*, CoRR, abs/1612.01939 (2016).

[82] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015.

[83] S. van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber, *Relational neural expectation maximization: Unsupervised discovery of objects and their interactions*, in International Conference on Learning Representations, 2018.

[84] V. Vanhoucke, A. Senior, and M. Z. Mao, *Improving the speed of neural networks on cpus*, in Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011, 2011.

[85] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2017.

[86] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272.

[87] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, *Transformers: State-of-the-art natural language processing*, in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, Oct. 2020, Association for Computational Linguistics, pp. 38–45.

[88] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, *Quantized convolutional neural networks for mobile devices*, CoRR, abs/1512.06473 (2015).

[89] X. Wu, S. Lv, L. Zang, J. Han, and S. Hu, *Conditional bert contextual augmentation*, 2018.

[90] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, *Xlnet: Generalized autoregressive pretraining for language understanding*, 2020.

[91] X. Yu, T. Liu, X. Wang, and D. Tao, *On compressing deep models by low rank and sparse decomposition*, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2017), pp. 67–76.

[92] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, *Bertscore: Evaluating text generation with BERT*, CoRR, abs/1904.09675 (2019).

[93] X. Zhang, J. Zou, K. He, and J. Sun, *Accelerating very deep convolutional networks for classification and detection*, IEEE transactions on pattern analysis and machine intelligence, 38 (2016), pp. 1943–1955.