POLS 4724: Practicum Pre-Analysis Plan

Aman Choudhri

November 4, 2024

Motivation

Comparing the performance of different machine learning (ML) algorithms in practice is made difficult by the cost of evaluating a given model. A single training run, even on a GPU in an academic compute cluster, may take upwards of 2 days for common image classification benchmark datasets like `ImageNet` . These compute requirements naturally limit the extent to which budget-constrained researchers can systematically compare proposed novel techniques to state-of-the-art baselines. Rather than running a proposed algorithm head-to-head with a baseline on a variety of benchmark datasets, researchers may be forced to limit the scope of their comparisons to just a few hand-picked datasets—which they hope will be illustrative of algorithm performance as a whole.

This may be viewed as a problem of "superpopulation inference." Formally, researchers observe some performance difference between two algorithms on some subset $d \ll D$ of some available set of D benchmarks, and hope that their estimates satisfy $\hat{\tau}_d \approx \hat{\tau}_D$. Of course, this poses some problems. The subset d may not be "representative" of the broader set of benchmarks D, by which I mean that observed performance improvement from algorithms on d may fail to hold true on D in general. This issue makes it hard to declare categorically that any one algorithm is better than another.

In this practicum, I propose a somewhat silly idea to alleviate the computational burden of systematic performance comparisons: randomized experimentation. I describe this idea as silly, because in this case we may reasonably be able to observe both potential outcomes. We might run two different ML algorithms on the same dataset, obtain performance measurements $Y_i^{(0)}, Y_i^{(1)}$, and compare them as if they were potential outcomes to observe a true "individual treatment effect." But I maintain that it is nonetheless an interesting question to study how we should best allocate a fixed evaluation budget across models and benchmark datasets, assuming we cannot run all our models of interest on all our datasets.

Specifically, my question of interest is the following: can a randomized experiment in which D benchmark datasets are randomly assigned to two different ML algorithms, $z \in \{0, 1\}$, recover the true performance difference between $z = 0$ and $z = 1$ better than an exact performance difference observation on a handpicked finite subset d of the benchmarks?

Plan

The subjects of this experiment will be $D = 30$ common supervised image classification datasets. Our outcome of interest is the test-set classification accuracy of an ML model trained on the dataset, and the choice of model will represent our treatment/control conditions.

Subjects

I selected these datasets from a collection of open-access datasets made available through the ML Python package `torchvision`. I restricted to datasets whose task was image classification, meaning a ML model should input an image and output an integer representing the "class" to which the image should belong. I also removed from consideration datasets that would take too long to train on, like the aforementioned `ImageNet` dataset.

For each dataset, we have the following covariates: number of images, number of classes, and average image size. All are measured pre-treatment. Since images are often of different shapes, we'll define the computed covariate S_i as average number of pixels in the image, which we compute by multiplying the width and height.

Treatment

The control condition in this experiment is the convolutional neural network (CNN) architecture, which was introduced in 1998 [4] and popularized in 2012 [3]. It remains one of the most popular neural network setups for image processing to this day. We'll denote it by $z = 0$. The treatment condition, written as $z = 1$, is the more novel vision transformer (ViT) architecture [1], introduced in 2020.

Outcome

For each dataset i, our outcomes of interest are $Y_i(0)$ and $Y_i(1)$, the classification accuracies of the assigned model on an unseen test set. Specifically, a given model will be trained only on a *subset* of a dataset i, with the rest of the data set aside and dubbed the "test set." After training, then, the model will produce predicted class assignments for each unseen image in the test set. The classification accuracy is simply the proportion of correct class assignments by the model. Since the outcome is a proportion, we note that $0 \leq Y_i(z) \leq 1$.

Randomization

Since we have access to covariates from the datasets, we will use blocked random assignment. One covariate in particular is relevant here, what I'll call the *richness*: the average number of images per class. Roughly speaking, the more images there are available for each class, the easier the task since there is more information available for the model to pull from. I'll block on richness thresholding at 100 examples per class, since there are comparatively few datasets in that regime and I want to make sure we try both models on those kinds of datasets. This results in the following group sizes:

Table 1: Number of Datasets by Richness

R_i	Richness Range	Count
0	$<$ 100 examples/class	7
1	$>$ 100 examples/class	23

We'll use complete random assignment within each block. For $R_i = 0$, we'll assign 3 datasets to treatment and 4 to control. For $R_i = 1$, we'll assign 11 to treatment and 12 to control.

The silly nature of the randomization here means we won't face noncompliance and that we are safe to assume excludability and non-interference. However, we may face an issue of attrition, with models failing to train in time or failing to converge at all. This may correlate with potential outcomes, as datasets with smaller potential classification accuracies $Y_i(z)$ may be harder to train on, meaning we have a higher chance of attrition among those datasets. We discuss how to respond to this in the following section.

Analysis

For this experiment, I'm primarily interested in the average treatment effect

$$\mathbb{E}[Y_i(1) - Y_i(0)],$$

which I'll estimate using the block-adjusted difference-in-means estimator. However, we also have access to the *log image size* covariate, S_i. Generally the larger the image, the harder the classification task—so I'd like to adjust for S_i as well.

To compute an ATE estimate using both a blocking indicator R_i and a covariate S_i , we will use weighted regression of the form

$$Y_i \sim \alpha + \beta z_i + \gamma \log(S_i).$$

We transform the image size S_i into logarithm scale before the regression, as a one-unit difference on the log scale is more meaningful and consistent across a variety of scales.

We will need to use a weighted regression since the treatment assignment probabilities per block are not exactly the same. The weights for this regression are given by equation 4.12 in [2]: the weight for subject i in block j is given by

$$w_{ij} = \frac{1}{p_{ij}} z_i + (\frac{1}{1 - p_{ij}})(1 - z_i),$$

where p_{ij} is the probability that subject i within block j was assigned to treatment.

I hypothesize no treatment effect. I'll test this hypothesis using the linear regression analysis, using a two-tailed test on the coefficient representing our estimate $\hat{\beta}$.

As discussed previously, we face a potential challenge of attrition here, with missing data likely being correlated with potential outcomes $Y_i(z)$. If attrition occurs, we will conduct the above analysis among the results that we do have, the Always-Reporters. We won't attempt any of the correction methods, since the introduction of randomness may already add too much noise to discern any true performance differences. If attrition does occur, we leave it to a follow-up study to retry the outcome measurement on the subjects, relying on the nature of our problem to assume that these will return the same potential outcomes.

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021.

[2] Alan S. Gerber and Donald P. Green. *Field Experiments: Design, Analysis, and Interpretation*. W. W. Norton, New York, 1st ed edition, 2012.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov./1998.

Appendix

Datasets

Table 2: Overview of Image Classification Datasets

	Task	Size	Num Classes	Average Image Size
CIFAR10	General objects	60,000	10	32x32
CIFAR100	General objects	60,000	100	32x32
Caltech101	General objects	9,146	101	300x200
Caltech256	General objects	30,607	256	300x200
CelebA	Face	202,599	10177	178x218
Country211	Countries	75,000	249	468x382
DTD	Textures	5,640	47	494x450
EMNIST	Handwriting	814,255	47	28x28
EuroSAT	Land	27,000	10	64x64
FER2013	Faces	35,887	7	48x48
FGVCAircraft	Aircraft	10,000	100	1101x749
FashionMNIST	Clothing	70,000	10	28x28
Flowers102	Flowers	7,169	102	631x534
Food101	Food	101,000	101	512x512
GTSRB	Traffic Signs	51,840	43	50x50
Imagenette	General objects	13,394	10	471x408
KMNIST	Handwriting	70,000	10	28x28
LFWPeople	Faces	13,233	5749	250x250
MNIST	Handwriting	70,000	10	28x28
Omniglot	Handwriting	32,460	964	105x105
OxfordIIITPet	Animals	7,400	37	437x391
PCAM	Medical scans	327,680	2	96x96
QMNIST	Handwriting	120,000	10	28x28
RenderedSST2	Sentiment from text images	8,741	2	448x448
SEMEION	Handwriting	1,593	10	16x16
STL10	General objects	13,000	10	96x96
SUN397	Scenes	108,753	397	959x775
SVHN	Digits	630,420	10	32x32
StanfordCars	Cars	16,185	196	700x483
USPS	Handwriting	9,298	10	16x16

Code

First, here is the code to generate random assignments.

Code 1: Randomization code, from `randomization.R`

```r
datasets <- read.csv('../datasets.csv')

# Check that we have 30 subjects
N <- 30
if (nrow(datasets) != N) {
  print(datasets)
}

# Perform randomization

# for R_i = 0, we assign 3 to treatment
m_block_0 <- 3
# for R_i = 1, we assign 11 to treatment
m_block_1 <- 11

set.seed(0527)

block_0_treated <- sample(which(datasets$richness_block == 0), m_block_0)
block_1_treated <- sample(which(datasets$richness_block == 1), m_block_1)

Z <- rep(0, N)
Z[block_0_treated] <- 1
Z[block_1_treated] <- 1

assignments <- data.frame(dataset=datasets$Dataset, Z=Z)
write.csv(assignments, '../assignments.csv')
```

Here is the code for the regression analysis.

Code 2: Analysis code, from `main.R`

```r
# Required packages
library(tidyverse)  # for data wrangling
library(sandwich)   # for robust standard errors
library(kableExtra) # for nice tables
library(ggplot2)    # for plotting
library(texreg)     # for LaTeX regression tables
library(ri2)        # for randomization inference

# Read data
datasets <- read.csv("../datasets.csv")
assignments <- read.csv("../assignments.csv")

# Merge datasets
data <- datasets %>%
  inner_join(assignments, by = c("Dataset"="dataset")) %>%
  mutate(
    block = richness_block,
    # Log transform image size
    log_img_size = log(img_size),
  )

# Calculate assignment probabilities by block
probs <- data %>%
  group_by(block) %>%
  summarize(
    n = n(),
    n_treated = sum(Z),
    p = n_treated / n
  )

# Join probabilities back and calculate weights
data <- data %>%
  left_join(probs, by = "block") %>%
  mutate(weight = Z/p + (1-Z)/(1-p))

# Function to create regression tables
create_reg_table <- function(results, filename) {
  texreg(
    results,
    file = paste0("doc/generated/", filename, ".tex"),
    booktabs = TRUE,
    caption = "Treatment Effect Estimates",
    label = paste0("tab:", filename),
    include.ci = FALSE,
    custom.model.names = c("Block-Adjusted", "Covariate-Adjusted"),
    custom.coef.names = c(
      "(Intercept)" = "Intercept",
      "Z" = "Treatment (ViT)",
      "log_img_size" = "Log Image Size"
    )
  )
}

# After we observe outcomes Y, we'll run:

# Function to analyze outcomes
analyze_outcomes <- function(data) {
  # Fit models
  m1 <- lm(Y ~ Z, data = data, weights = weight)
  m2 <- lm(Y ~ Z + log_img_size, data = data, weights = weight)

  # Create results table
  create_reg_table(list(m1, m2), "treatment_effects")

  # Randomization inference
  ri_out <- conduct_ri(
    Y ~ Z + log_img_size,
    blocks = ~block,
    declaration = declare_ra(blocks = data$block,
                             block_prob = probs$p)
  )

  # Create diagnostic plots
  p1 <- ggplot(data, aes(x = log_img_size, y = Y, color = factor(Z))) +
    geom_point() +
    geom_smooth(method = "lm") +
    labs(x = "Log Image Size", y = "Accuracy", color = "Treatment") +
    theme_minimal()

  ggsave("doc/generated/outcome_by_size.pdf", p1)

  # Balance plot
  p2 <- ggplot(data, aes(x = log_img_size, fill = factor(Z))) +
    geom_density(alpha = 0.5) +
    labs(x = "Log Image Size", y = "Density", fill = "Treatment") +
    theme_minimal()

  ggsave("doc/generated/covariate_balance.pdf", p2)

  return(list(
    models = list(m1, m2),
    ri = ri_out,
    plots = list(p1, p2)
  ))
}
```