Ranking Causal Effects - Causal Models @Uber

Ben Draves

About Me

My name is Benjamin Draves and I'm a Staff Applied Scientist at Uber in NYC. I work on building ML recommendation and targeting systems to grow Uber's consumer and earner businesses.

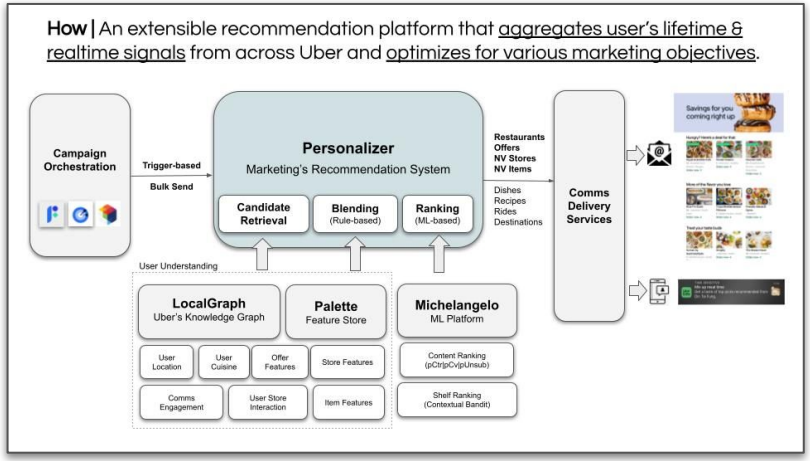
Data Science Topics that Excite Me

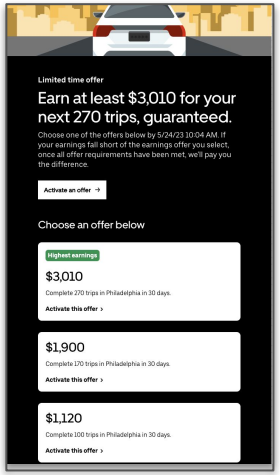
- Recommendation Systems
- Causal Machine Learning
- Graph Embeddings and Inference

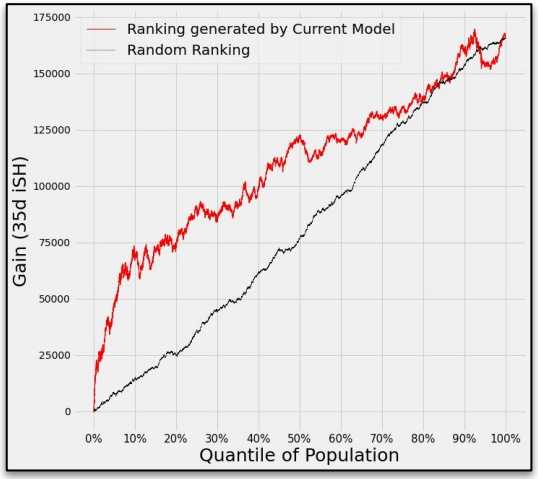
Experiences & Education

- [2022-Present] Applied Scientist @ Uber
- [2017-2022] Ph.D. | M.A. in Stat @ Boston University
- [2014-2017] B.S. in Math @ Lafayette College

Growing Uber's Core Businesses







Recommendation & Personalization

I work on personalizing customer's experiences in out-of-app channels to drive product awareness, adoption, and growth. This oftentimes manifests in **recommending restaurants, merchants, and offers** to customers to encourage product exploration.

Targeting & Orchestration

I develop communication targeting and orchestration systems to provide relevant and timely notifications to customers which service core business goals. This oftentimes manifests in marrying **causal machine learning models** and techniques from **ads pacing systems.**.

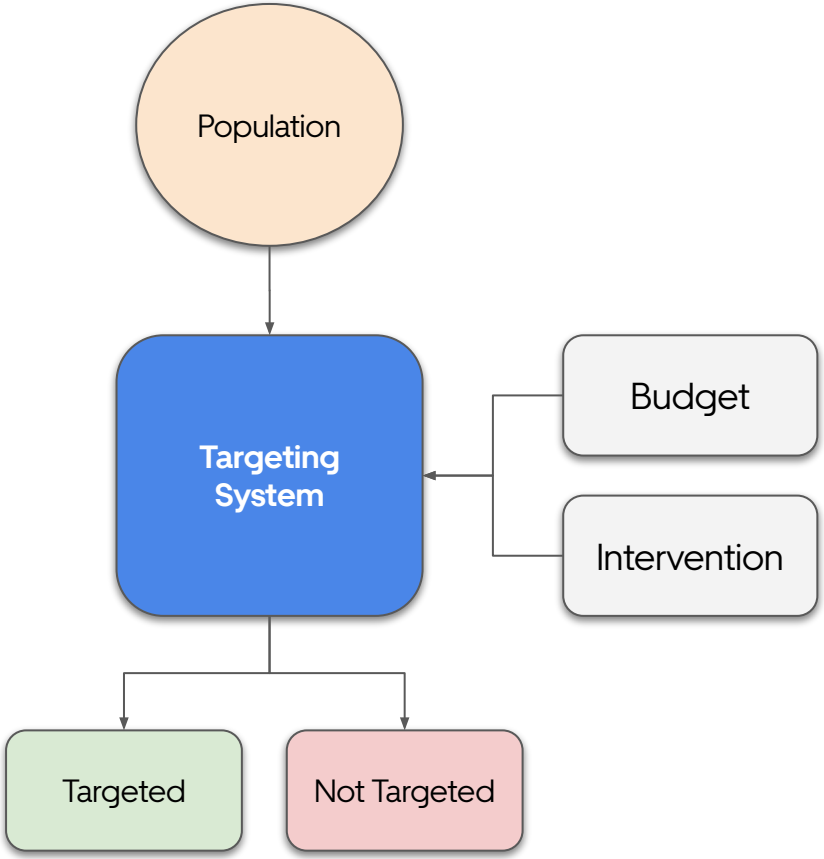
Motivating Problem | Marketing Intervention with Budget Constraint

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Examples

- Free Uber One membership for 1M
- Bonus to drivers for completing 5 trips
- Weekly newsletter with restaurant promos



Agenda

01 Metalearners
02 DL & Robustness Extensions
03 Ranking Causal Effects
04 Conclusion & Discussion

Metalearners

Notation | Neyman-Rubin Potential Outcomes Framework

Data Generation Process

Assume we have the DGP $W = (X, T, Y) \sim \mathbb{P}$ where

- $X \in \mathcal{X} \subseteq \mathbb{R}^d$ represent covariates
- $T \in \{0, 1\}$ is a binary treatment
- $Y \in \mathbb{R}$ is a continuous outcome

We collect an i.i.d. sample $\{(x_i, t_i, y_i)\}_{i=1}^n \overset{i.i.d.}{\sim} \mathbb{P}$

CATE Definition

Each unit has a two potential outcomes $(Y(0), Y(1))$ where $Y(t)$ represents the outcome given $T = t$. The CATE is given by

$$\tau(x) = \mathbb{E}[Y(1) - Y(0)|X = x]$$

Identification Assumptions

We take the standard assumptions (i) Consistency (ii) Positivity (iii) Unconfoundedness so we can write CATE as

$$\tau(x) = \mu_1(x) - \mu_0(x)$$

where $\mu_t(x) = \mathbb{E}[Y(t)|X = x]$ are the conditional outcome regressions.

Supplements

Let the propensity score be given by

$$e(x) = \mathbb{P}(T = 1|X = x)$$

Let the nuisance functions be given by

$$\eta = (\mu_1, \mu_0, e)$$

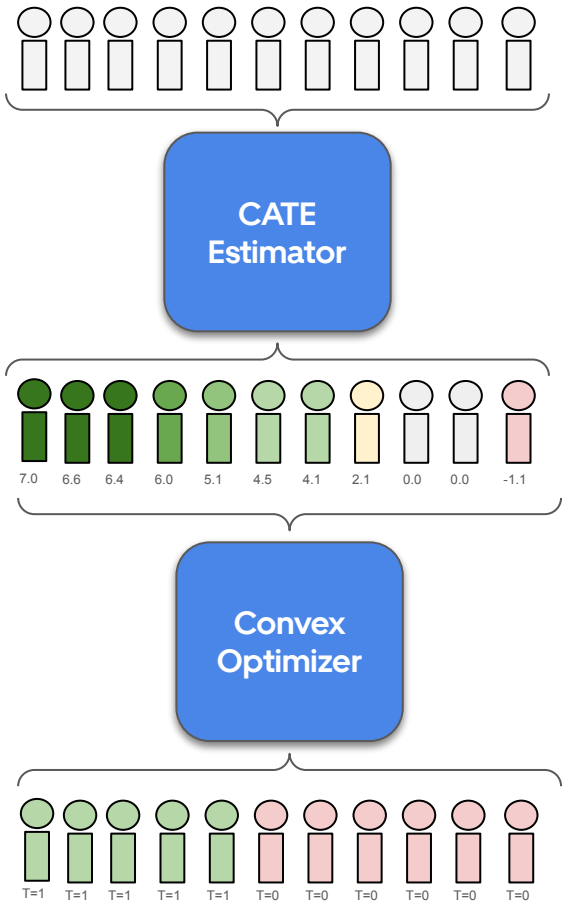
Motivating Problem | Marketing Intervention with Budget Constraint

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Use convex optimization to assign treatment



Motivating Problem | Marketing Intervention with Budget Constraint

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Use convex optimization to assign treatment

$$\underset{z_i \in \{0,1\}}{\text{maximize}} \quad \sum_{i=1}^{N} z_i \hat{\tau}(x_i)$$

$$\text{subject to} \quad \sum_{i=1}^{N} z_i \hat{c}(x_i) \leq B$$

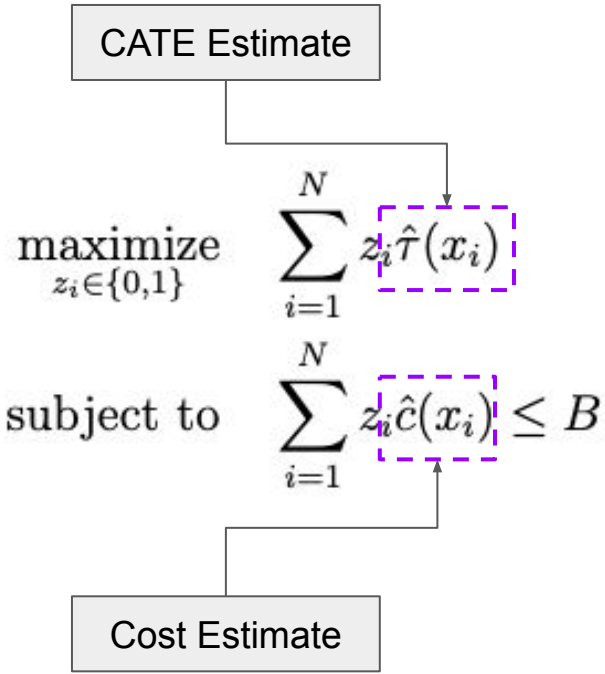
Motivating Problem | Marketing Intervention with Budget Constraint

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Use convex optimization to assign treatment



Metalearners | S/T/X Learners

S Learner

- Fit **one** ML model with the treatment indicator as a covariates

$$\mu(x,t) = \mathbb{E}[Y^{\text{obs}}|X = w, T = t]$$

- Estimate CATE via two prediction calls

$$\hat{\tau}_S(x) = \hat{\mu}(x,1) - \hat{\mu}(x,0)$$

metalearners > s_learner.py

```python
"""S-Learner: estimates CATE using a single model with treatment as a feature."""

import numpy as np
from BaseLearner import BaseLearner

class SLearner:
    def __init__(self, base_learner: BaseLearner = None):
        self.model = base_learner

    def fit(self, X, T, Y):
        #augment X with treatment
        XT = np.column_stack([X, T])
        self.model.fit(XT, Y)
        return self

    def predict_cate(self, X):
        n = len(X)
        X1 = np.column_stack([X, np.ones(n)])
        X0 = np.column_stack([X, np.zeros(n)])
        return self.model.predict(X1) - self.model.predict(X0)

```

Metalearners | S/T/X Learners

T Learner

- Fit **two** ML models with the treatment / control data respectively

$$\mu_1(x) = \mathbb{E}[Y(1)|X = x] \qquad \mu_0(x) = \mathbb{E}[Y(0)|X = x]$$

- Estimate CATE via differencing

$$\hat{\tau}_T(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$$

metalearners > t_learner.py

```python
"""T-Learner: estimates CATE using separate models for T/C groups."""

import numpy as np
from BaseLearner import BaseLearner


class TLearner:
    def __init__(self, base_learner: BaseLearner = None):
        factory = lambda: base_learner
        self.model_t = factory()
        self.model_c = factory()

    def fit(self, X, T, Y):
        self.model_t.fit(X[T == 1], Y[T == 1])
        self.model_c.fit(X[T == 0], Y[T == 0])
        return self

    def predict_cate(self, X):
        return self.model_t.predict(X) - self.model_c.predict(X)

```

Metalearners | S/T/X Learners

X Learner

- Fit **two** ML models with the treatment / control data respectively

$$\mu_1(x) = \mathbb{E}[Y(1)|X = x] \qquad \mu_0(x) = \mathbb{E}[Y(0)|X = x]$$

- Impute ITE for each unit

$$\tilde{D}(1) = Y(1) - \hat{\mu}_0(X) \qquad \tilde{D}^{(0)} = \hat{\mu}_1(X) - Y(0)$$

- Fit **two** models to estimate impute ITEs

$$\hat{\tau}_1(x) = \mathbb{E}[\tilde{D}^{(1)}|X = x] \qquad \hat{\tau}_0(x) = \mathbb{E}[\tilde{D}^{(0)}|X = x]$$

- Estimates CATE via

$$\hat{\tau}_X(x) = \hat{e}(x)\hat{\tau}_0(x) + (1 - \hat{e}(x))\hat{\tau}_1(x)$$

metalearners > x_learner.py

```python
"""X-Learner: uses pseudo-outcomes and propensity weighting to handle imbalanced treatment."""

import numpy as np
from BaseLearner import BaseLearner

class XLearner:
    def __init__(self,
                 outcome_learner: BaseLearner = None,
                 effect_learner: BaseLearner = None,
                 propensity_learner: BaseLearner = None
                 ):
        self.mu0 = outcome_learner
        self.mu1 = outcome_learner
        self.tau0 = effect_learner
        self.tau1 = effect_learner
        self.propensity = propensity_learner

    def fit(self, X, T, Y):
        X0, Y0 = X[T == 0], Y[T == 0]
        X1, Y1 = X[T == 1], Y[T == 1]

        # Stage 1: fit outcome models on each group
        self.mu0.fit(X0, Y0)
        self.mu1.fit(X1, Y1)

        # Stage 2: pseudo-outcomes cross-impute
        D1 = Y1 - self.mu0.predict(X1)   # treated: actual - counterfactual(control)
        D0 = self.mu1.predict(X0) - Y0   # control: counterfactual(treated) - actual

        self.tau1.fit(X1, D1)
        self.tau0.fit(X0, D0)

        # Stage 3: propensity model for weighting
        self.propensity.fit(X, T)
        return self

    def predict_cate(self, X):
        e = self.propensity.predict_proba(X)[:, 1]   # P(T=1 | X)
        return e * self.tau0.predict(X) + (1 - e) * self.tau1.predict(X)
```

Metalearners | Statistical Guarantees & Empirical Evidence

Meta Learner	Statistical Guarantees	Strengths	Weaknesses
S	Converges if base learners do: $\|\hat{\mu}(x,t) - \mu(x,t)\| = O(r_n)$ implies $\|\hat{\tau}_S(x) - \tau(x)\| = O(r_n)$	• Simple to implement • Lower variance due to treatment group pooling	• Model can regularize away treatment covariate
T	Converges if base learners do: $\|\hat{\mu}_0(x) - \mu_0(x)\| = O(r_{0,n})$ $\|\hat{\mu}_1(x) - \mu_1(x)\| = O(r_{1,n})$ implies $\|\hat{\tau}_T(x) - \tau(x)\| = O(r_{1,n} + r_{0,n})$	• Simple to implement • Lower bias due to no treatment group pooling	• Model adapts poorly to class imbalances
X	Converges if base learners do: $\|\hat{\tau}_X(x) - \tau(x)\| = O(r_{\mu,n} + r_{\tau,n} + n^{-1/2})$	• Handles class imbalance • If the CATE curve is simple, then the rate improves to the larger group	• Complex model management

Metalearners | Uber Example - Earner Incentives

T-Learner

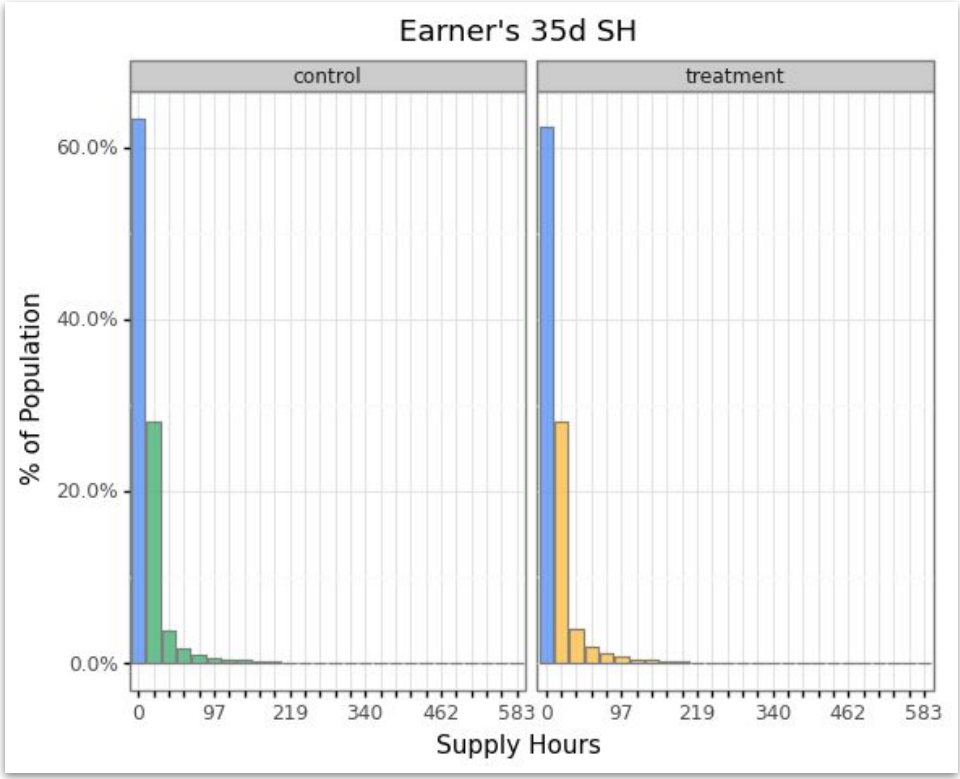
- Fit two SH regression models
- Incremental SH prediction is delta between the two model predictions

$$\hat{\tau}(\mathbf{x}) = \hat{\mu_1}(\mathbf{x}) - \hat{\mu_0}(\mathbf{x})$$

trained with treatment data

trained with control data

- Base learners are XGBoost Regressor fit with a [tweedie loss]



Metalearners | Uber Example - Earner Incentives

T-Learner

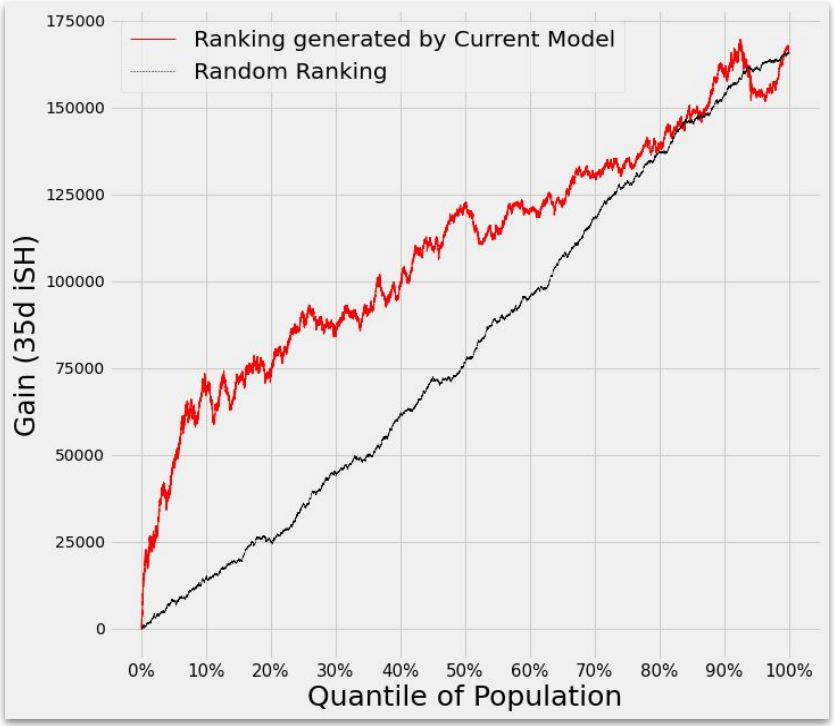
- Fit two SH regression models
- Incremental SH prediction is delta between the two model predictions

$$\hat{\tau}(\mathbf{x}) = \hat{\mu_1}(\mathbf{x}) - \hat{\mu_0}(\mathbf{x})$$

trained with treatment data

trained with control data

- Base learners are XGBoost Regressor fit with a tweedie loss



DL & Robustness Extensions

Modern Improvements | Areas for Improvement

Model Complexity

Maintaining multiple models in practice can be complex. Can we simplify the engineering design?

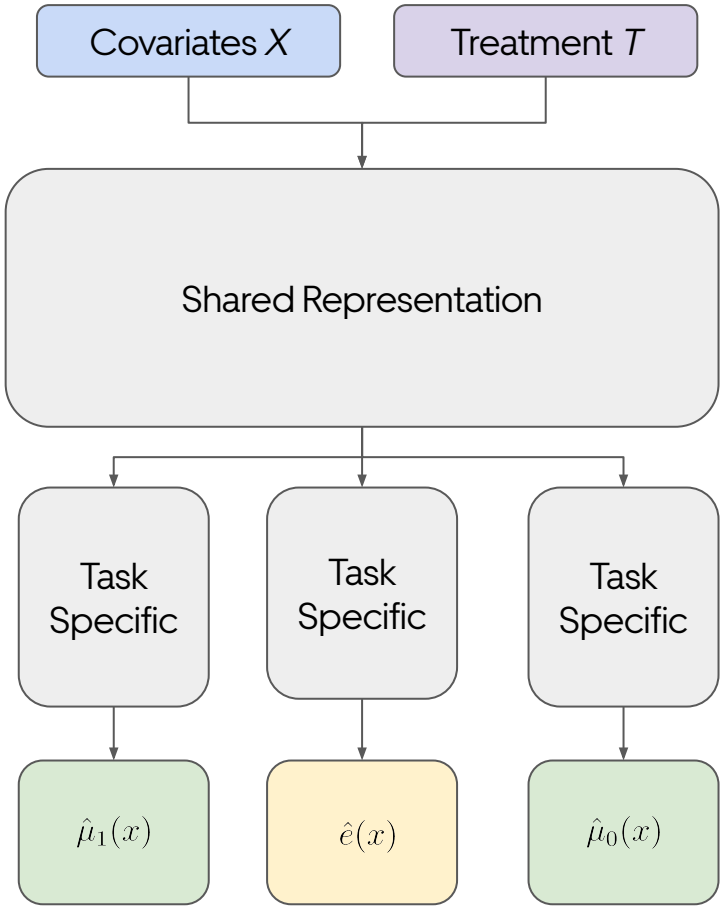
Statistical Robustness

S/T/X learners suffer from “plugin” bias. Can make our CATE estimators more robust to upstream estimation errors?

Modern Improvements | Tarnet, CFR, DragoNet, ...

DL Architectures

- For each S/T/X learner the first stage needs to estimate the nuisance functions $\eta = (\mu_1, \mu_0, e)$ from the covariates
- Can we use an multi-output DL model to make this prediction simpler to manage?
- Many DL methods available (survey) produce these estimates. We discuss DragonNet
  - DragonNet: $\hat{\eta}_{DN}(x) = (\hat{\mu}_1(x), \hat{\mu}_0(x), \hat{e}(x))$



Modern Improvements | Tarnet, CFR, DragonNet, ...

DL Architectures

- For each S/T/X learner the first stage needs to estimate the nuisance functions $\eta = (\mu_1, \mu_0, e)$ from the covariates
- Can we use an multi-output DL model to make this prediction simpler to manage?
- Many DL methods available (survey) produce these estimates. We discuss DragonNet
  - DragonNet: $\hat{\eta}_{DN}(x) = (\hat{\mu}_1(x), \hat{\mu}_0(x), \hat{e}(x))$

metalearners › dragonnet.py

```python
"""
DragonNet: shared-representation network with joint outcome + propensity heads.

Reference: Shi et al. 2019 — "Adapting Neural Networks for the Estimation
of Treatment Effects"  (https://arxiv.org/abs/1906.02120)
"""

import numpy as np
import torch
import torch.nn as nn

class _DragonNet(nn.Module):
    def __init__(self, input_dim, hidden_dim, dropout):
        super().__init__()
        self.trunk = nn.Sequential(
            nn.Linear(input_dim, hidden_dim), nn.ELU(),
            nn.Linear(hidden_dim, hidden_dim), nn.ELU(),
            nn.Linear(hidden_dim, hidden_dim), nn.ELU(),
            nn.Dropout(dropout),
        )
        self.y0 = nn.Sequential(nn.Linear(hidden_dim, hidden_dim // 2), nn.ELU(),
                                nn.Linear(hidden_dim // 2, 1))
        self.y1 = nn.Sequential(nn.Linear(hidden_dim, hidden_dim // 2), nn.ELU(),
                                nn.Linear(hidden_dim // 2, 1))
        self.propensity = nn.Sequential(nn.Linear(hidden_dim, 1), nn.Sigmoid())

    def forward(self, x):
        z = self.trunk(x)
        return self.y0(z).squeeze(-1), self.y1(z).squeeze(-1), self.propensity(z).squeeze(-1)

```

Modern Improvements | Training

Training Details

- Multiobjective loss
  - Propensity estimation
  - Outcome loss

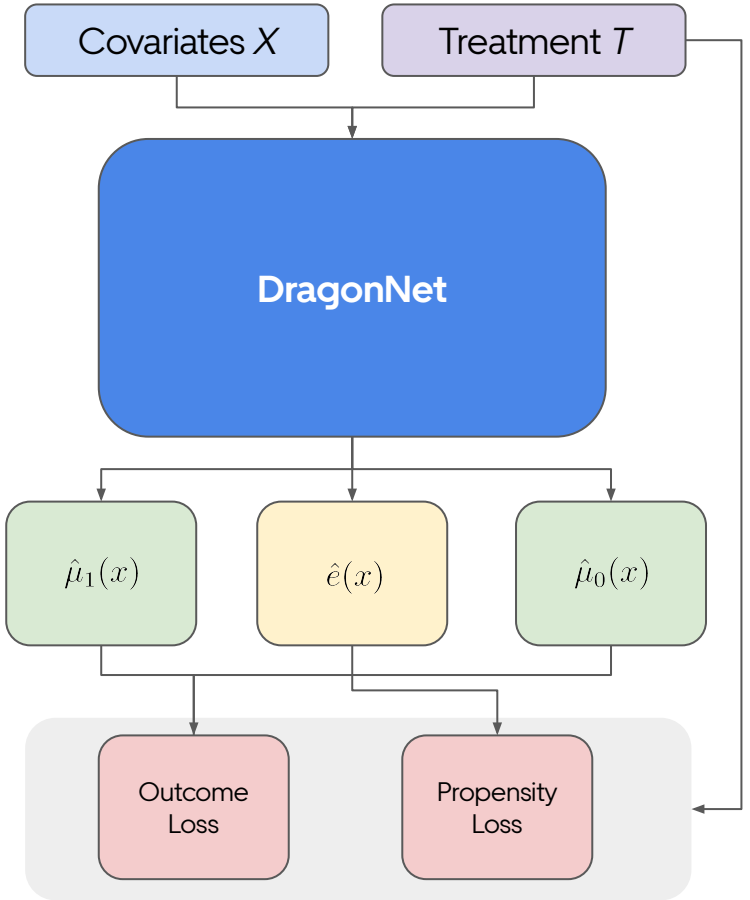
$$\mathcal{L} = \mathcal{L}_{out} + \mathcal{L}_{prop}$$

- Only compute loss based on labeled data for outcomes

$$\mathcal{L}_{out} = -\frac{1}{N}\sum_{i=1}^{N}\left[I(t_i = 1)(y_i - \hat{\mu}_1(x_i))^2 + I(t_i = 0)(y_i - \hat{\mu}_0(x_i))^2\right]$$

- Propensity loss computed for all labels

$$\mathcal{L}_{prop} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{e}(x_i)) + (1 - y_i)\log(1 - \hat{e}(x_i))$$



Modern Improvements | Training

Training Details

- Multiobjective loss
  - Propensity estimation
  - Outcome loss

$$\mathcal{L} = \mathcal{L}_{out} + \mathcal{L}_{prop}$$

- Only compute loss based on labeled data for outcomes

$$\mathcal{L}_{out} = -\frac{1}{N}\sum_{i=1}^{N}\left[I(t_i = 1)(y_i - \hat{\mu}_1(x_i))^2 + I(t_i = 0)(y_i - \hat{\mu}_0(x_i))^2\right]$$

- Propensity loss computed for all labels

$$\mathcal{L}_{prop} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{e}(x_i)) + (1 - y_i)\log(1 - \hat{e}(x_i))$$

metalearners > dragonnet.py

```
32  class DragonNet:
50      def fit(self, X, T, Y):
51          X_t = torch.tensor(X, dtype=torch.float32)
52          T_t = torch.tensor(T, dtype=torch.float32)
53          Y_t = torch.tensor(Y, dtype=torch.float32)
54
55          self.net = _DragonNet(X.shape[1], self.hidden_dim, self.dropout)
56          opt = torch.optim.Adam(self.net.parameters(), lr=self.lr)
57          mse = nn.MSELoss()
58          bce = nn.BCELoss()
59
60          n = len(X_t)
61          self.net.train()
62          for _ in range(self.epochs):
63              idx = torch.randperm(n)
64              for start in range(0, n, self.batch_size):
65                  b = idx[start : start + self.batch_size]
66                  xb, tb, yb = X_t[b], T_t[b], Y_t[b]
67
68                  y0, y1, prop = self.net(xb)
69
70                  # Outcome loss: use y1 for treated, y0 for controls
71                  y_hat = torch.where(tb == 1, y1, y0)
72                  outcome_loss = mse(y_hat, yb)
73                  prop_loss    = bce(prop, tb)
74
75                  loss = outcome_loss + self.alpha * prop_loss
76                  opt.zero_grad()
77                  loss.backward()
78                  opt.step()
79
80          self.net.eval()
81          return self
82
83      def predict_cate(self, X):
84          self.net.eval()
85          with torch.no_grad():
86              X_t = torch.tensor(X, dtype=torch.float32)
87              y0, y1, _ = self.net(X_t)
88          return (y1 - y0).numpy()
89
```

Modern Improvements | Identifying Robustness

What is "Robustness"

- We are interested in estimate the CATE

  $$\tau(x) = \mathbb{E}[Y(1)|X = x] - \mathbb{E}[Y(0)|X = x]$$

  but oftentimes suffer "plug-in" bias

- Can we devise "second stage" estimation procedures witch estimate $\tau(x)$ that are robust to certain errors
- What type of errors do we want to be robust to?

Neyman Orthogonality

If the nuisance estimates are biased, overfit, or converging slowly, we want $\hat{\tau}(x_i)$ to still be consistent and converge quickly.

Double Robustness

If only one of our nuisance estimators (propensity scores or surface function) are correctly specified, then our estimator $\hat{\tau}(x_i)$ is still unbiased and consistent.

Modern Improvements | Takeaways

Most of the best available methods today follow a two stage approach:

- **Stage 1**: Estimate nuisance parameters & necessary parameters for inference with best methods from ML.
- **Stage 2**: Use nuisances estimates as an input to the second optimization problem which should be robust to first stage noise.

Ranking Causal Effects

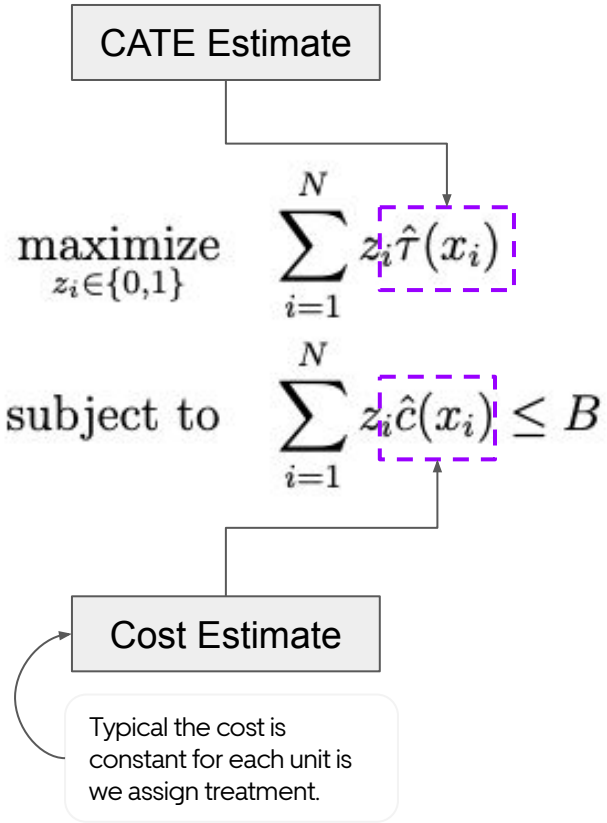
Ranking Causal Effects | Revisiting Our Motivating Example

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Use convex optimization to assign treatment



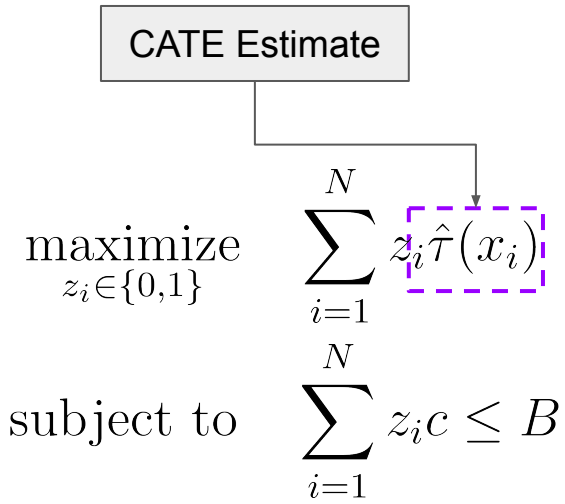
Ranking Causal Effects | Revisiting Our Motivating Example

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Use convex optimization to assign treatment



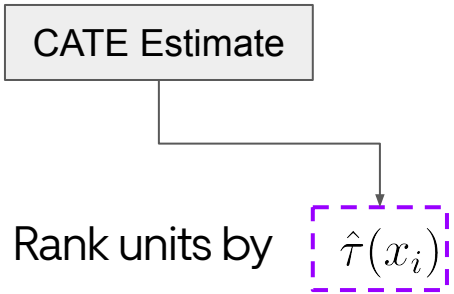
Ranking Causal Effects | Revisiting Our Motivating Example

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Select top $k = \lfloor B/c \rfloor$ ranked units



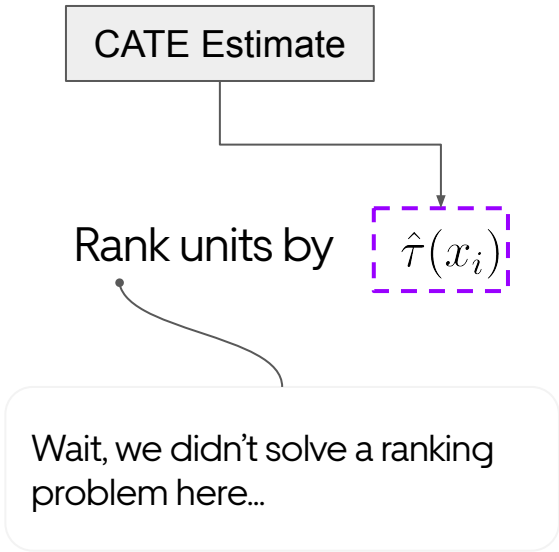
Ranking Causal Effects | Revisiting Our Motivating Example

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Select top $k = \lfloor B/c \rfloor$ ranked units



Ranking Causal Effects | Revisiting Our Motivating Example

Problem Statement

Given an a population ***P***, intervention ***T***, and budget ***B***, how we do we optimally allocation treatment across the population?

Solution Formulation

1. Identify growth metric of interest Y
2. Estimate inc. increase in growth metric $\hat{\tau}(x_i)$
3. Select top $k = \lfloor B/c \rfloor$ ranked units

Stage 1: Estimate nuisance parameters & necessary parameters for inference with best methods from ML.

$$\eta = (\mu_1, \mu_0, e)$$

Stage 2: Use nuisances estimates as a second optimization problem which should be robust to first stage noise.

Ranking units by $\hat{\tau}(x_i)$

Is this true?

Ranking Causal Effects | Rank-learner: Arno et al. 2026.

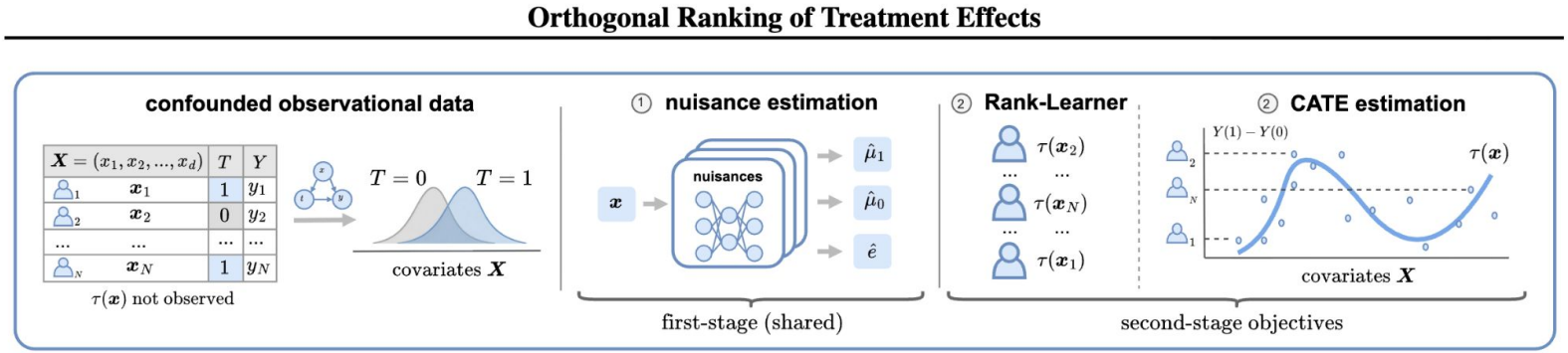


Figure 1. **Two-stage learners for ranking treatment effects.** *Left:* Confounded observational data with unobserved treatment effects. *Center:* First-stage estimation of nuisance functions (i.e., response surfaces and propensity score). *Right:* Second-stage objectives: our *Rank-Learner* vs. standard CATE estimation. Here, our proposed *Rank-Learner* directly optimizes a pairwise, Neyman-orthogonal ranking objective that targets the relative ordering of treatment effects. In contrast, standard CATE estimation optimizes a pointwise regression objective to recover treatment effect magnitudes, which is harder than necessary for ranking.

Ranking Causal Effects | Rank-learner: Arno et al. 2026.

Rank Learner Overview

- CATE estimation is generally a much more difficult problem than ranking.
- Authors propose **optimizing a pairwise ranking loss** in the second stage that is **Neyman Orthogonal**
- This allows for
  - Stage 1 - use the best from ML
  - Stage 2 - optimizing ranking directly

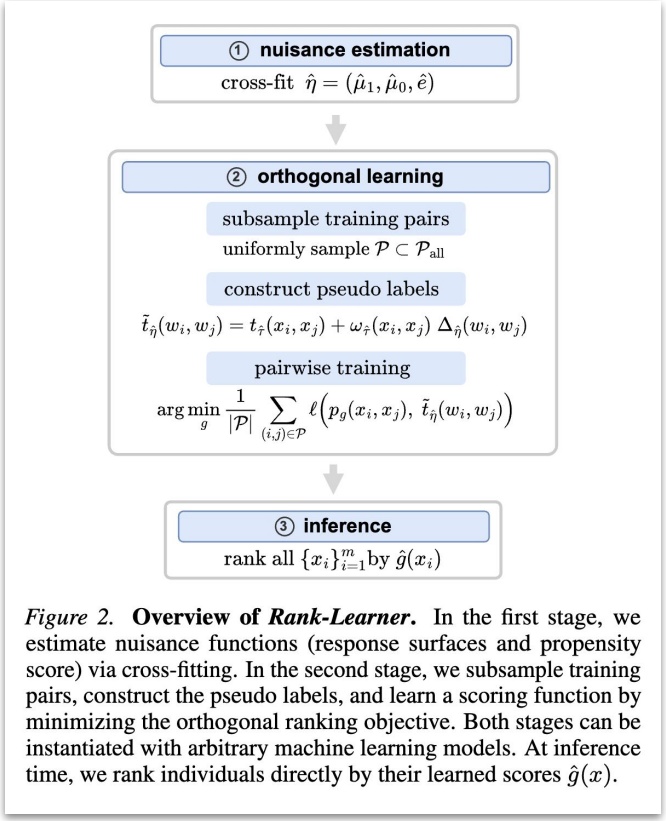


Figure 2. **Overview of *Rank-Learner*.** In the first stage, we estimate nuisance functions (response surfaces and propensity score) via cross-fitting. In the second stage, we subsample training pairs, construct the pseudo labels, and learn a scoring function by minimizing the orthogonal ranking objective. Both stages can be instantiated with arbitrary machine learning models. At inference time, we rank individuals directly by their learned scores $\hat{g}(x)$.

Ranking Causal Effects | Rank-learner: Arno et al. 2026.

Rank Learner Overview

- CATE estimation is generally a much more difficult problem than ranking.
- Authors propose **optimizing a pairwise ranking loss** in the second stage that is **Neyman Orthogonal**
- This allows for
  - Stage 1 - use the best from ML
  - Stage 2 - **optimizing ranking directly**

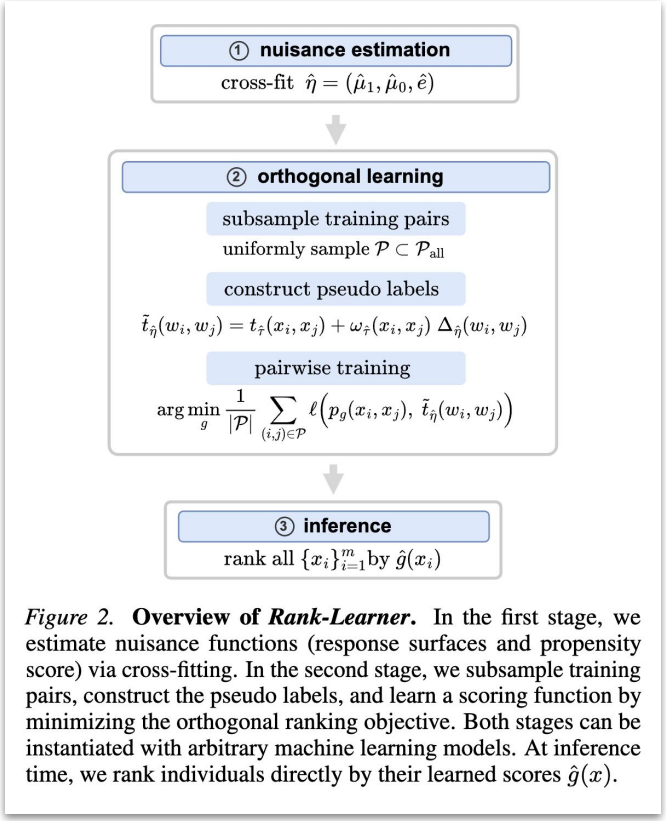


Figure 2. **Overview of *Rank-Learner*.** In the first stage, we estimate nuisance functions (response surfaces and propensity score) via cross-fitting. In the second stage, we subsample training pairs, construct the pseudo labels, and learn a scoring function by minimizing the orthogonal ranking objective. Both stages can be instantiated with arbitrary machine learning models. At inference time, we rank individuals directly by their learned scores $\hat{g}(x)$.

Ranking Causal Effects | Learning the Ranker

Ranking Loss Construciton

- **Goal**: Learn a ranking function $g(\cdot) : \mathcal{X} \to \mathbb{R}$ such that relative ordering is maintained

  $$\tau(x) > \tau(x') \implies g(x) > g(x')$$

- **Loss**: Authors take a pairwise ranking approach
  - Use estimated $\hat{\tau}(x_i)$ to derive labels
  - Learn $g(\cdot)$ by estimating those labels

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), b_\tau(X, X'))\right]$$

Ranking Causal Effects | Learning the Ranker

Ranking Loss Construciton

- **Goal**: Learn a ranking function $g(\cdot) : \mathcal{X} \to \mathbb{R}$ such that relative ordering is maintained

  $$\tau(x) > \tau(x') \implies g(x) > g(x')$$

- **Loss**: Authors take a pairwise ranking approach
  - Use estimated $\hat{\tau}(x_i)$ to derive labels
  - Learn $g(\cdot)$ by estimating those labels

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}[\ell(p_g(X, X'), b_\tau(X, X'))]$$

Binary cross entropy loss

$$l(p, t) = -t \log(p) + (1 - t) \log(1 - p)$$

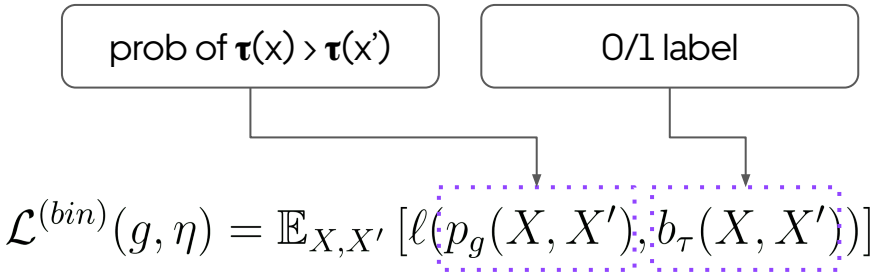
Ranking Causal Effects | Learning the Ranker

Ranking Loss Construciton

- **Goal**: Learn a ranking function $g(\cdot) : \mathcal{X} \to \mathbb{R}$ such that relative ordering is maintained

  $$\tau(x) > \tau(x') \implies g(x) > g(x')$$

- **Loss**: Authors take a pairwise ranking approach
  - Use estimated $\hat{\tau}(x_i)$ to derive labels
  - Learn $g(\cdot)$ by estimating those labels



Ranking Causal Effects | Learning the Ranker

Ranking Loss Construciton

- **Goal**: Learn a ranking function $g(\cdot) : \mathcal{X} \to \mathbb{R}$ such that relative ordering is maintained

  $\tau(x) > \tau(x') \implies g(x) > g(x')$

- **Loss**: Authors take a pairwise ranking approach
  - Use estimated $\hat{\tau}(x_i)$ to derive labels
  - Learn $g(\cdot)$ by estimating those labels

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), b_\tau(X, X'))\right]$$

$$p_g(X, X') = \sigma(g(X) - g(X'))$$

$$I(\tau(x) > \tau(x'))$$

Ranking Causal Effects | Learning the Ranker

Ranking Loss Construciton

- **Goal**: Learn a ranking function $g(\cdot) : \mathcal{X} \to \mathbb{R}$ such that relative ordering is maintained

  $$\tau(x) > \tau(x') \implies g(x) > g(x')$$

- **Loss**: Authors take a pairwise ranking approach
  - Use estimated $\hat{\tau}(x_i)$ to derive labels
  - Learn $g(\cdot)$ by estimating those labels
- This loss is minimized for any $g(\cdot)$ that preserves ordering of CATE - Much easier than CATE estimation!

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), b_\tau(X, X'))\right]$$

$p_g(X, X') = \sigma(g(X) - g(X'))$

$I(\tau(x) > \tau(x'))$

Ranking Causal Effects | Learning the Orthogonal Ranker

Orthogalizing the Loss

- **Goal**: Can we update the loss so the function we learn $g(\cdot)$ is robust to Stage 1 noise?

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), b_\tau(X, X'))\right]$$

Ranking Causal Effects | Learning the Orthogonal Ranker

Orthogalizing the Loss

- **Goal**: Can we update the loss so the function we learn $g(\cdot)$ is robust to Stage 1 noise?
- Authors change the loss in two ways
    1. Smoothing - ensure the loss is differentiable with respect to nuisance

$$\mathcal{L}^{(bin)} \rightarrow \mathcal{L}^{(soft)}$$

$$\mathcal{L}^{(bin)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), b_\tau(X, X'))\right]$$

$$\mathcal{L}^{(soft)}(g, \eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X, X'), t_\tau(X, X'))\right]$$

"Smooth" Labels

$$t_\tau(X, X') = \sigma\left(\frac{\tau(X) - \tau(X')}{\kappa}\right)$$

Ranking Causal Effects | Learning the Orthogonal Ranker

Orthogalizing the Loss

- **Goal**: Can we update the loss so the function we learn $g(\cdot)$ is robust to Stage 1 noise?
- Authors change the loss in two ways
  1. Smoothing - ensure the loss is differentiable with respect to nuisance
     $$\mathcal{L}^{(bin)} \rightarrow \mathcal{L}^{(soft)}$$
  2. Orthogonalization - augment the labels with an orthogonal correction
     $$\mathcal{L}^{(soft)} \rightarrow \mathcal{L}^{(orth)}$$

$$\mathcal{L}^{(bin)}(g,\eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X,X'), b_\tau(X,X'))\right]$$

$$\mathcal{L}^{(soft)}(g,\eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X,X'), t_\tau(X,X'))\right]$$

$$\mathcal{L}^{(orth)}(g,\eta) = \mathbb{E}_{X,X'}\left[\ell(p_g(X,X'), \tilde{t}_\eta(X,X'))\right]$$

"Orthogonalized" Labels

$$\tilde{t}_\eta(X,X') = t_\tau(X,X') + \omega_\tau(X,X')\Delta_\eta(X,X')$$

Ranking Causal Effects | Training the Orthogonal Ranker

Training

Phase 1

1. Estimate nuisance functions with cross fitting

$$\hat{\eta} = (\hat{\mu}_1, \hat{\mu}_0, \hat{e})$$

Phase 2

1. Sample training pairs

$$\mathcal{P} = \{(i,j) : i \in [n], j \in [n], i \neq j\} \subseteq \mathcal{P}_{all}$$

2. Construct pseudo labels

$$\tilde{t}_\eta(x_i, x_j) = t_\tau(x_i, x_j) + \omega_\tau(x_i, x_j')\Delta_\eta(x_i, x_j')$$

3. Supervised training loop over loss

$$\mathcal{L}^{(orth)}(g, \hat{\eta}) = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \ell\left(p_g(x_i, x_j), \tilde{t}_{\hat{\eta}}(x_i, x_j)\right)$$

Ranking Causal Effects | Training the Orthogonal Ranker

Training

Phase 1

1. Estimate nuisance functions with cross fitting

$$\hat{\eta} = (\hat{\mu}_1, \hat{\mu}_0, \hat{e})$$

Phase 2

1. Sample training pairs

$$\mathcal{P} = \{(i, j) : i \in [n], j \in [n], i \neq j\} \subseteq \mathcal{P}_{all}$$

2. Construct pseudo labels

$$\tilde{t}_\eta(x_i, x_j) = t_\tau(x_i, x_j) + \omega_\tau(x_i, x_j')\Delta_\eta(x_i, x_j')$$

3. Supervised training loop over loss

$$\mathcal{L}^{(orth)}(g, \hat{\eta}) = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \ell\left(p_g(x_i, x_j), \tilde{t}_{\hat{\eta}}(x_i, x_j)\right)$$

metalearners › mlp_ranker.py

```python
"""
MLPRanker: default ranker for RankLearner.
Reference: https://arxiv.org/abs/2602.03517
"""

import torch
import torch.nn as nn
import torch.nn.functional as F


class _MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, dropout):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim), nn.ELU(),
            nn.Linear(hidden_dim, hidden_dim), nn.ELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, 1),
        )

    def forward(self, x):
        return self.net(x).squeeze(-1)

```

Ranking Causal Effects | Training the Orthogonal Ranker

Training

Phase 1

1. Estimate nuisance functions with cross fitting

$$\hat{\eta} = (\hat{\mu}_1, \hat{\mu}_0, \hat{e})$$

Phase 2

1. Sample training pairs

$$\mathcal{P} = \{(i,j) : i \in [n], j \in [n], i \neq j\} \subseteq \mathcal{P}_{all}$$

2. Construct pseudo labels

$$\tilde{t}_\eta(x_i, x_j) = t_\tau(x_i, x_j) + \omega_\tau(x_i, x_j')\Delta_\eta(x_i, x_j')$$

3. Supervised training loop over loss

$$\mathcal{L}^{(orth)}(g, \hat{\eta}) = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \ell\left(p_g(x_i, x_j), \tilde{t}_{\hat{\eta}}(x_i, x_j)\right)$$

```python
class MLPRanker:
    def fit(self, X, phi, tau):
        """
        X   : covariates
        phi : doubly robust scores ϕ_η(W)
        tau : CATE estimates τ̂(X) = μ₁(X) − μ₀(X)
        """
        X_t   = torch.tensor(X,   dtype=torch.float32)
        phi_t = torch.tensor(phi, dtype=torch.float32)
        tau_t = torch.tensor(tau, dtype=torch.float32)
        n     = len(X_t)

        self.mlp = _MLP(X.shape[1], self.hidden_dim, self.dropout)
        opt = torch.optim.Adam(self.mlp.parameters(), lr=self.lr)

        self.mlp.train()
        for _ in range(self.epochs):
            idx   = torch.randint(n, (self.batch_pairs, 2))
            i, j  = idx[:, 0], idx[:, 1]

            # pseudo-label t̃_η = t_τ + ω_τ · Δ_η  (clamped for BCE)
            t_tau = torch.sigmoid((tau_t[i] - tau_t[j]) / self.kappa)
            omega = t_tau * (1 - t_tau) / self.kappa
            Delta = (phi_t[i] - tau_t[i]) - (phi_t[j] - tau_t[j])
            t_tilde = torch.clamp(t_tau + omega * Delta, 0.0, 1.0)

            # Predicted pairwise probability p_g(X, X′) = σ(g(X) − g(X′))
            p_g = torch.sigmoid(self.mlp(X_t[i]) - self.mlp(X_t[j]))

            # Back propagation loss
            loss = F.binary_cross_entropy(p_g, t_tilde)
            opt.zero_grad()
            loss.backward()
            opt.step()

        self.mlp.eval()
        return self
```

Ranking Causal Effects | Training the Orthogonal Ranker

Inference

1. Collect features $\{x_i : i = 1, 2, \ldots, m\}$
2. Call model $\hat{r}_i = \hat{g}(x_i) \quad i = 1, 2, \ldots m$
3. Rank based on $\{\hat{r}_i\}_{i=1}^m$

Table 4. **Semi-synthetic benchmarks**: Test AUTOC (mean ± std dev over five seeds) with training size $n = 1{,}000$. *Higher is better.* The oracle row reports AUTOC obtained by ranking the test set using the true treatment effects. Best mean is shown in **bold**.

Method	MovieLens	MIMIC-III	CPS
oracle	1.39	1.22	1.01
T-learner	1.31 ± 0.03	1.12 ± 0.05	0.87 ± 0.08
DR-learner	1.34 ± 0.02	1.16 ± 0.02	0.92 ± 0.02
Plug-in ranker	1.30 ± 0.03	1.11 ± 0.05	0.87 ± 0.08
Rank-learner (*ours*)	**1.35 ± 0.01**	**1.18 ± 0.02**	**0.95 ± 0.01**

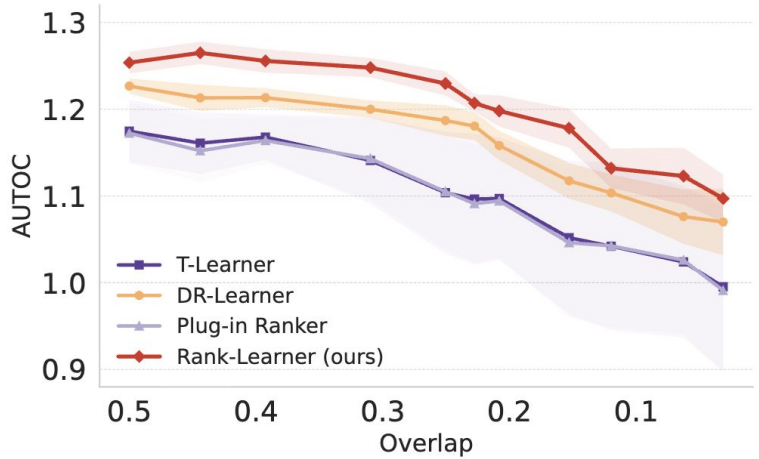


Figure 4. **Synthetic benchmark (overlap sensitivity).** Test AUTOC (mean ± s.e. over five seeds) as a function of overlap (decreasing left to right) for $n = 500$. *Higher is better.* Overlap is varied by changing treatment assignment, remaining components of the synthetic data-generating process are kept fixed.

Conclusion

Conclusion | Discussion

Takeaways

- CausalML in industry is a mixture of engineering feasibility & causal inference best practices
- Metalearners allow us to leverage the best ideas from the ML community flexibly
- Second stage inference approaches that are (i) robust and (ii) problem specific perform the best

Questions?

Contact Information

- Linkedin
- Email
  - benjamin.draves@gmail.com
  - ben.draves@uber.com

Modern Improvements | Identifying Robustness

What is "Robustness"

- We are interested in estimate the CATE

  $$\tau(x) = \mathbb{E}[Y(1)|X = x] - \mathbb{E}[Y(0)|X = x]$$

  but oftentimes suffer "plug-in" bias

- Can we devise "second stage" estimation procedures witch estimate $\tau(x)$ that are robust to certain errors
- What type of errors do we want to be robust to?

Neyman Orthogonality

If the nuisance estimates are biased, overfit, or converging slowly, we want $\hat{\tau}(x_i)$ to still be consistent and converge quickly.

Double Robustness

If only one of our nuisance estimators (propensity scores or surface function) are correctly specified, then our estimator $\hat{\tau}(x_i)$ is still unbiased and consistent.

Modern Improvements | R Learner

R Learner

- Fit **two** ML models to predict nuisance

$$m(x) = \mathbb{E}[Y|X = x] \qquad e(x) = \mathbb{P}(Y = 1|X = x)$$

- Residualize outcome and treatment

$$\tilde{Y} = Y - \hat{m}(X) \qquad \tilde{T} = T - \hat{m}(X)$$

- Estimate $\hat{\tau}(x_i)$ by minimizing

$$\hat{\tau}(\cdot) = \arg\min_{\tau} \sum_{i=1}^{N} \left\{(\tilde{y}_i - \tilde{t}_i \tau(x_i))^2\right\}$$

metalearners › r_learner.py

```python
"""
R-Learner: residualises both outcome and treatment before fitting CATE.
Reference: Nie & Wager 2017 — https://arxiv.org/abs/1712.04912
"""

import numpy as np
from BaseLearner import BaseLearner

class RLearner:
    def __init__(self,
                 outcome_learner: BaseLearner = None,
                 propensity_learner: BaseLearner = None,
                 effect_learner: BaseLearner = None):
        self.outcome_model    = outcome_learner
        self.propensity_model = propensity_learner
        self.effect_model     = effect_learner

    def fit(self, X, T, Y):
        # Stage 1: cross-fit m(X) and e(X) to avoid overfitting
        m_hat = self._cross_fit_outcome(X, T, Y)
        e_hat = self._cross_fit_propensity(X, T)

        # Stage 2: compute residuals
        Y_res = Y - m_hat          # outcome residual
        T_res = T - e_hat          # treatment residual

        # Stage 3: weighted regression  τ(X) on pseudo-targets
        # target  = Y_res / T_res,  weight = T_res²
        weights = T_res ** 2
        targets = np.where(np.abs(T_res) > 1e-6, Y_res / T_res, 0.0)

        self.effect_model.fit(X, targets, sample_weight=weights)
        return self

    def predict_cate(self, X):
        return self.effect_model.predict(X)

```