

Data Science Collective

Home

About

★ Member-only story

Structural Distillation for Cross-Dataset Uplift Modeling with Reinforcement Learning



Shenggang Li · Following

Published in Data Science Collective · 20 min read · 19 hours ago



52



1



A Novel Approach to Transfer Partial Teacher Model Knowledge from Control to Treatment Data for Rapid AB Testing and Campaign Optimization

Photo by [Domenico Loia](#) on [Unsplash](#)

1. Introduction

When conducting a marketing campaign and running AB tests, we often use a single response model — referred to as the “teacher” or control + treatment groups model — that accurately captures both baseline and experimental customer behavior. This model can be highly complex and time-consuming to train, making it impractical to rebuild from scratch for each new campaign. However, new promotions or treatments require a separate “student” (or treatment) model that can rapidly adapt to changing data while still leveraging the knowledge embedded in the teacher. This is precisely where our proposed approach becomes highly relevant.

Traditional AB tests or standard uplift modeling might just compare “treatment vs. control” in a one-time or fixed scenario. But promotions don’t

stand still; customer behaviors shift, and what worked before might flop next quarter. By blending reinforcement learning (*RL*) with distillation, we can offer an evolving strategy: the system automatically figures out the best promotional tactics in real time, using the teacher's knowledge without having to retrain that teacher all over again.

Think of the teacher as a wise old chef who has perfected a soup recipe through years of refinement. When you need to serve a new crowd, you want a simpler soup — faster to cook, easier to tweak — but still filled with the chef's wisdom. Distillation extracts the essence of the chef's knowledge (partial model structure and outputs) and transfers it into your quick-to-adapt student model. Meanwhile, *RL* keeps tasting and adjusting flavors as the crowd's preferences change.

This combo is a game-changer: you keep your master recipe intact, save training time, and dynamically serve each customer's tastes. Our approach captures the best of both worlds — robust baseline knowledge plus nimble adaptation — pushing uplift modeling to a new level of efficiency and relevance.

2. Uplift Modeling in Marketing Campaigns

Uplift modeling is a refined approach to AB testing that focuses on how a marketing intervention (e.g., a discount or special offer) changes individual customer behavior, rather than just measuring average differences between control and treatment groups. In traditional AB testing, analysts often rely on simple methods — like a T-test — to see if aggregate metrics (e.g., average purchase rate) differ meaningfully across groups. However, uplift modeling

digs deeper: it explicitly models the incremental effect of a treatment on each individual (or segment), thereby showing not just *whether* the treatment works on average, but also *for whom* it works best.

Brief Introduction to the Uplift Model

In a typical marketing campaign, you have a control group (no intervention) and a treatment group (with some intervention such as a new discount). Uplift modeling uses machine learning — often logistic regression — to compare predicted outcomes in both groups, focusing on the incremental lift (the additional probability of a positive outcome) attributable to the treatment. This approach is more powerful than a simple T-test because it estimates *individual-level* or *segment-level* treatment effects, rather than relying solely on group-level averages.

Two Common Algorithms for Uplift Modeling

(A) Two-Model Approach

Build *two separate* logistic regressions, one for each group. For example, you might have:

$$\text{Model}_{\text{control}} : \Pr(\text{Purchase} = 1 \mid \mathbf{x}) \quad \text{and} \quad \text{Model}_{\text{treatment}} : \Pr(\text{Purchase} = 1 \mid \mathbf{x})$$

Then, for a given customer with features x , you compute:

$$\text{Uplift}(\mathbf{x}) = \Pr_{\text{treatment}}(\mathbf{x}) - \Pr_{\text{control}}(\mathbf{x})$$

The difference in predicted probabilities reflects how much more likely that customer is to purchase under the treatment condition relative to the

control.

(B) One-Model Approach with Group Indicator

Here, you combine both groups in one dataset and add a binary treatment indicator variable (e.g., $treatment = 1$ if the customer is in the treatment group, 0 otherwise). You then fit a single logistic regression like:

$$\Pr(\text{Purchase} = 1 \mid \mathbf{x}, \text{treatment}) = \sigma(\beta_0 + \beta_1 \cdot \text{zip_code_Urban} + \beta_2 \cdot \text{treatment} + \beta_3 \cdot \text{recency} + \dots)$$

Suppose your **significant features** look like:

$\{\text{const}, \text{zip_code_Urban}, \text{treatment}, \text{recency}, \text{used_discount}, \text{used_bogo}, \text{is_referral}\}$.

By comparing the predicted probability when

$treatment=1$ vs. $treatment=0$ for the same x , you estimate uplift. This approach can be extended to include interaction terms between treatment and other features (e.g., $treatment * recency$), which helps capture how the treatment effect varies by feature.

Demo Code Snippet

Below is a simple demonstration of how uplift modeling might be coded. Note that we are not generating data here; we assume you have a DataFrame `df` with columns matching our features (e.g., `zip_code_Urban`, `treatment`, `recency`, `used_discount`, `used_bogo`, `is_referral`, and the target variable `purchase`):

```
import pandas as pd
import statsmodels.api as sm
```

(A) Two-Model Approach

```

df_control = df[df['treatment'] == 0].copy()
df_treat   = df[df['treatment'] == 1].copy()

# Fit logistic regression for control group
X_control = df_control[['const', 'zip_code_Urban', 'recency',
                        'used_discount', 'used_bogo', 'is_referral']]
y_control = df_control['purchase']
model_control = sm.Logit(y_control, X_control).fit()

# Fit logistic regression for treatment group
X_treat = df_treat[['const', 'zip_code_Urban', 'recency',
                    'used_discount', 'used_bogo', 'is_referral']]
y_treat = df_treat['purchase']
model_treat = sm.Logit(y_treat, X_treat).fit()

# Predict probabilities for entire dataset using each model
df['pred_control'] = model_control.predict(df[['const', 'zip_code_Urban', 'recency',
                                             'used_discount', 'used_bogo', 'is_referral']])
df['pred_treat']   = model_treat.predict(df[['const', 'zip_code_Urban', 'recency',
                                             'used_discount', 'used_bogo', 'is_referral']])
df['uplift_2model'] = df['pred_treat'] - df['pred_control']

# (B) One-Model Approach with Group Indicator
X_full = df[['const', 'zip_code_Urban', 'treatment', 'recency',
             'used_discount', 'used_bogo', 'is_referral']]
y_full = df['purchase']
model_one = sm.Logit(y_full, X_full).fit()

# For a given row i, we compute predicted outcome for treatment=1 and 0
# to get the uplift. For brevity, let's just do it in two calls:
X_for_predict_t1 = X_full.copy()
X_for_predict_t0 = X_full.copy()

X_for_predict_t1['treatment'] = 1
X_for_predict_t0['treatment'] = 0

df['p_treat'] = model_one.predict(X_for_predict_t1)
df['p_control'] = model_one.predict(X_for_predict_t0)
df['uplift_1model'] = df['p_treat'] - df['p_control']

print(df[['uplift_2model', 'uplift_1model']].head())

```

In practice, you would analyze *df['uplift...']* to rank customers by expected lift and decide who should receive the treatment.

Why Uplift Modeling vs. T-Test, and Its Limitations

A T-test AB setup checks whether the average outcome differs between control and treatment, but it overlooks individual- or segment-level variation. Uplift modeling pinpoints where (and for whom) the promotion is most or least effective. For example, you might discover that *used_discount=1* and *is_referral=1* drive the largest uplift-key insights for campaign targeting.

However, standard uplift methods are often static. They in many cases cannot adapt well to changing market conditions or test new interventions quickly, and they usually need a complete retraining whenever new data arrives. In this paper, we present an enhanced approach to address these issues, making uplift strategies more dynamic and responsive.

3. Introducing Distillation: A Logistic Regression Example

Imagine you already have a “mother” logistic regression model — complex, full of interaction terms and clever feature engineering. This model (the “teacher”) often performs well but might be too bulky or hard to update for fast-paced AB testing. Distillation helps you build a simpler “student” logistic regression that mimics the teacher’s predictions without overfitting on raw labels.

Distillation Mechanism

Teacher Model (Mother Regression)

Fits a logistic function:

$$P_{\text{mother}}(\mathbf{x}) = \sigma\left(\beta_0 + \sum_i \beta_i X_i + \sum_{i,j} \beta_{i,j} (X_i \cdot X_j)\right)$$

where $\sigma(\cdot)$ is the sigmoid function.

After training on (x, y) , it generates soft probabilities $P_{\text{mother}}(x)$ for each sample, capturing nuanced interactions.

Student Model (Distilled Regression)

Has fewer terms, for instance:

$$P_{\text{student}}(\mathbf{x}) = \sigma\left(\theta_0 + \sum_i \theta_i X_i\right)$$

Instead of fitting on hard labels $y \in \{0,1\}$, it learns from the teacher's probabilities $P_{\text{mother}}(x)$.

We minimize a divergence (often KL) to align P_{student} with P_{mother} :

$$\min_{\theta} \sum_{\mathbf{x}} KL(P_{\text{mother}}(\mathbf{x}), P_{\text{student}}(\mathbf{x}))$$

Example & Code

Below is a demo. Suppose you already trained a “teacher” model named *model_teacher*:

Open in app ↗

Medium

Search

Write

1



```
# (the teacher's predicted probabilities)

# Data
X_cols = ['featureA', 'featureB', 'featureC'] # simpler than teacher
df['const'] = 1.0
X_student = df[['const'] + X_cols]
y_soft = df['soft_label'] # teacher's probability predictions

# 2. Fit the student logistic regression using the teacher's soft labels
# Simulate a distillation approach by treating y_soft as "labels"
model_student = sm.GLM(y_soft, X_student, family=sm.families.Binomial()).fit()

print(model_student.summary())

# 3. Evaluate closeness to teacher's predictions
student_preds = model_student.predict(X_student)
kl_div = np.mean(student_preds * np.log(student_preds / y_soft + 1e-8))
print("Approx KL Divergence:", kl_div)
```

Here, *soft_label* is the teacher’s output, assigning each sample a continuous probability. We then train a simpler logistic regression on this “soft” target, aligning with the teacher’s learned relationships.

Benefits of Distillation for Logistic Regression

- **Reduces Complexity:** By dropping interaction terms, the student model is easier to maintain or retrain.
- **Retains Knowledge:** It learns the teacher’s insights rather than overfitting to binary labels.

- **Captures Uncertainty:** If the teacher is uncertain (e.g., outputs 0.6 probability), the student learns that nuance, instead of a strict 0 or 1 .
- **Faster Updates:** When new data arrives, retraining a small student model is quicker than touching the complex teacher.

In short, distillation cuts complexity and cost while retaining most of the model's predictive power.

2. Distillation, Dynamic Reinforcement Learning, and Uplift Modeling: A Unified Framework

In this section, I present a methodology that integrates three key components:

- (a) Constructing a control (teacher) model to capture baseline customer behavior,
- (b) Transferring the teacher's insights into a more lightweight treatment (student) model via a novel cross-dataset distillation mechanism, and
- (c) Dynamically optimizing the student model through reinforcement learning (RL) to maximize uplift over successive campaigns.

2.1 Teacher Model $f_{\{T\}}$

I begin with a teacher model $f_{\{T\}}$ trained on control group data.

Let $x \in X$ be the feature vector (e.g., customer demographics, purchase history). The teacher's output can be a single prediction y^T (e.g., probability of purchase) or a vector (e.g., multiple classes).

Although training requires significant time, it extracts rich patterns from baseline behaviors. As an example, suppose $f_{\{T\}}$ estimates whether a customer will make a purchase (1 for purchase and 0 otherwise). In a simplified logistic regression form:

$$\text{response} = \sigma(\beta_0 + \beta_1 \text{PurchaseHistory} + \beta_2 \text{IsHoliday} + \dots)$$

where $\sigma(\cdot)$ is the logistic function, and features might include *PurchaseHistory* (past transactions) and *IsHoliday* (a binary holiday indicator). In practice, $f_{\{T\}}$ is often an ensemble (e.g., XGBoost) or a deep neural network, allowing it to capture complex relationships in control data. We denote the teacher's parameters by $\Theta_{\{T\}}$ (e.g., tree splits, neural weights, or feature importance), reflecting its internal structure and learned knowledge.

2.2 Cross-Dataset Distillation for the Student Model

When a new campaign (treatment) arises, a student model $f_{\{S\}}$ must adapt quickly without retraining a massive model each time.

Our cross-dataset distillation transfers knowledge from $f_{\{T\}}$ to a smaller $f_{\{S\}}$, using the teacher's outputs *and* its internal structure. The student is trained on a different (treatment) dataset D_{treat} but shares the same feature space X .

Example Scenario and Core Equations

Suppose the teacher, $f_{\{T\}}$, is a gradient boosting model that outputs a purchase probability,

$$\hat{y}_T(\mathbf{x}_i) = f_T(\mathbf{x}_i)$$

for each customer's feature vector \mathbf{x}_i . Internally, f_T might have feature importance $\phi_T(k)$ for each feature k , or if it is a logistic regression, it might have learned coefficients β_T .

We now introduce a simpler model, f_S , for the new promotional campaign. For instance, consider a logistic regression form:

$$\hat{y}_S(\mathbf{x}_i) = \sigma(\mathbf{w}_S^\top \mathbf{x}_i)$$

To use the teacher's knowledge, I define a composite loss that includes both the real treatment labels and the teacher's predictions (and possibly its internal structure). One common approach is:

$$\mathcal{L}(f_S; \mathcal{D}_{\text{treat}}) = \underbrace{\alpha \sum_{i=1}^N \ell_{\text{data}}(\hat{y}_S(\mathbf{x}_i), y_i)}_{\text{fits new data}} + \underbrace{\beta \sum_{i=1}^N KL(\hat{y}_T(\mathbf{x}_i), \hat{y}_S(\mathbf{x}_i))}_{\text{distillation term}} + \underbrace{\gamma \sum_k \|\phi_T(k) - \phi_S(k)\|^2}_{\text{structure alignment}}$$

- ℓ_{data} is a standard supervised loss (e.g., cross-entropy) evaluated on the treatment labels $\{y_i\}$.
- The distillation term aligns the student's predictions $\hat{y}_S(\mathbf{x}_i)$ with the teacher's output $\hat{y}_T(\mathbf{x}_i)$. A common choice is Kullback–Leibler divergence $KL(\cdot)$.

- The structure alignment term $\|\phi_{T(k)} - \phi_{S(k)}\|^2$ brings the student's parameters or importance scores closer to those of the teacher. This structure-aware distance penalizes discrepancies between the teacher's and student's internal representations. For a gradient boosting teacher, $\phi_{T(k)}$ might be a *SHAP* value or a global feature importance; for a teacher logistic regression, it might be β_T .

Here are several typical examples of structure-aware distance:

1. If $f_{\{T\}}$ yields approximate coefficients β_T (e.g., from a linear layer), then aligning the student's coefficients β_S can be achieved by minimizing:

$$d_{\text{structure}} = \|\beta_T - \beta_S\|^2$$

2. If $f_{\{T\}}$ produces SHAP values $\phi_T(x_i)$, we can penalize deviations in the student's corresponding values $\phi_S(x_i)$ by:

$$d_{\text{structure}} = \sum_{i=1}^N \|\phi_T(\mathbf{x}_i) - \phi_S(\mathbf{x}_i)\|^2$$

3. If Θ_T includes learned embeddings e_T , the student attempts to keep its embeddings e_S close to e_T , often by minimizing:

$$\|\mathbf{e}_T - \mathbf{e}_S\|^2$$

Intuitive Understanding

- **Same Features, different Data:** Even though D_{treat} includes customers in a new promotion, the feature set x remains the same. This lets us apply the teacher's learned structure to guide the student.
- **Faster Iterations:** By focusing on a smaller $f_{\{S\}}$, we can reduce training time. Instead of retraining a massive *XGBoost* teacher that might take hours, we can refresh the student in minutes for each new campaign.
- **Concrete Outputs:** After training, the student generates $y_S(x_i)$ on the treatment group, blending real-time label feedback with knowledge inherited from the teacher.
- **Partial Structure Transfer:** If the teacher highlighted "*LoyaltyScore*" as crucial (reflected by a large ϕ_T), the student is nudged to consider that feature more strongly — even if the new promotion data is limited — so it doesn't overlook insights the teacher already uncovered.

2.3 Reinforcement Learning for Uplift Optimization

Since marketing campaigns often span multiple time steps, we can treat each step as a state in a Markov Decision Process (*MDP*):

State s_t : Captures campaign conditions (e.g., discount level, seasonality) and historical performance (e.g., prior uplift scores, incremental revenue) at time t .

Action a_t : Determines how we configure our student model for the next wave. For instance, we might set new hyperparameters (α, β, γ) in the distillation loss or choose a different model type (e.g., logistic regression vs. a small neural network).

Reward r_t : Measures observed uplift (e.g., incremental revenue or purchase rate difference) after deploying the chosen student model. The RL agent aims to maximize this reward across campaigns.

An RL algorithm such as Proximal Policy Optimization (PPO) or Advantage Actor-Critic (A2C) updates its policy $\pi(a_t / s_t)$ based on observed rewards, continually improving its action choices.

Illustrative Example of an RL Cycle in Uplift Modeling

Initial State (s_t) :

Suppose the campaign environment indicates a 10% discount is active, and historical data suggests moderate uplift. The teacher model $f_{\{T\}}$ is already trained and fixed, providing valuable baseline knowledge.

Action (a_t):

The RL agent decides to adjust the student's distillation hyperparameters. For instance, it sets $\alpha=1.0$, $\beta=0.5$, and $\gamma=0.2$. This means we emphasize fitting the treatment data (α) while partly aligning predictions (β) and feature importance (γ) with the teacher. The agent also selects a small neural network for $f_{\{S\}}$, believing it may capture non-linear patterns in this particular discount scenario.

Student Training:

We train $f_{\{S\}}$ on the new promotion dataset D_{treat} , using the composite loss that combines real labels, teacher predictions, and structure alignment.

After training, $f_{\{S\}}$ generates predictions $y^S(x_i)$ for each customer under the current campaign.

Deployment and Feedback:

We deploy the student model for the active campaign. Over a few days, we can observe real outcomes (purchases, revenue) and compute the resulting uplift relative to a control or a benchmark.

Reward (r_t):

The uplift (incremental revenue) during this period exceeds expectations. Numerically, we will define $r_t = \Delta \text{Revenue}$ or an equivalent KPI measure. The higher the uplift, the greater the reward.

Policy Update:

The *RL* agent updates its policy $\pi(a_t / s_t)$ based on the reward. If the chosen student model and hyperparameters performed well, the policy is reinforced. Otherwise, the agent will explore alternative actions — like changing (α, β, γ) , switching to logistic regression, or increasing the model capacity — for the next wave.

Through these iterative steps, reinforcement learning discovers which student configurations (and which model types) create the best uplift for varying campaign conditions, all while avoiding the hefty cost of retraining the original teacher model.

Code Experiment

In this experiment, we utilize the marketing campaign dataset available at https://github.com/datalev001/RL_DIST_UPLIFT/tree/main/data to illustrate an integrated modeling framework that combines reinforcement learning (RL), knowledge distillation, and uplift modeling techniques. The approach begins with generating realistic marketing data for both control and treatment groups, capturing customer demographics, behaviors, and responses to promotional activities.

Next, we build a sophisticated “teacher” model on the control data, followed by training a simplified “student” model on the treatment data via knowledge distillation. Multiple campaign waves iteratively improve targeting through hyperparameter tuning.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
import xgboost as xgb

#####
# Train a Teacher Model (Control)
#####

def train_teacher_model(df_control):
    """
    Train a complex XGBoost model on the control subset, returning:
    - the trained model,
    - a dictionary of feature importances,
    - the exact list of columns (teacher_col_list) used for training.
    """
    raw_features = [
```

```

    "Age", "Income", "DaysSinceLastPurchase",
    "IsHolidaySeason", "PreferredChannel", "LoyaltyScore"
]
X = pd.get_dummies(df_control[raw_features], drop_first=True)
y = df_control["Purchase"]
teacher_col_list = list(X.columns)
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=5,
    learning_rate=0.1,
    use_label_encoder=False,
    eval_metric="logloss",
    random_state=42
)
model.fit(X_train, y_train)

y_val_pred = model.predict_proba(X_val)[:, 1]
auc_val = roc_auc_score(y_val, y_val_pred)
print(f"[Teacher Model] Control Validation AUC: {auc_val:.4f}")

importance_dict = model.get_booster().get_score(importance_type='gain')
teacher_importance = {col: importance_dict.get(col, 0.0) for col in teacher_col_list}

return model, teacher_importance, teacher_col_list

#####
# Plots: Gains Chart
#####

def plot_gains(df_wave, model, teacher_col_list, wave_number=0):
    """
    Plots a Gains (Lift) chart for the final wave's student model predictions.
    This chart shows how many positive responses (purchases) you capture
    as you move from high to low predicted probability.
    """
    from sklearn.preprocessing import StandardScaler
    # 1) Preprocess wave data
    raw_features = ["Age", "Income", "DaysSinceLastPurchase",
                    "IsHolidaySeason", "PreferredChannel", "LoyaltyScore"]
    X_wave = pd.get_dummies(df_wave[raw_features], drop_first=True)
    X_wave = X_wave.reindex(columns=teacher_col_list, fill_value=0.0)

    y_true = df_wave["Purchase"].values

    # Scale similarly (demo approach; in production, re-use the same scaler)
    scaler = StandardScaler()

```

```

X_scaled = scaler.fit_transform(X_wave)

# 2) Generate predicted probabilities
preds = model.predict_proba(X_scaled)[: , 1]

# 3) Sort by predicted probability descending
sort_idx = np.argsort(-preds)
y_sorted = y_true[sort_idx]

# 4) Compute cumulative gains
gains = np.cumsum(y_sorted) / y_sorted.sum()
x_vals = np.arange(1, len(y_true) + 1) / len(y_true)

# 5) Plot Gains chart
plt.figure(figsize=(8,5))
plt.plot(x_vals, gains, label="Model")
plt.plot([0,1],[0,1], 'r--', label="Random")
plt.title(f"Gains Chart (Wave {wave_number})")
plt.xlabel("Proportion of Customers (sorted by predicted probability)")
plt.ylabel("Proportion of Actual Purchases Captured")
plt.legend()
plt.grid(True)
plt.show()

#####
# Plots: Decile Analysis
#####

def decile_analysis(df_wave, model, teacher_col_list, wave_number=0, n_splits=10
    """
    Splits the wave data into deciles based on predicted probability
    and shows average predicted probability vs. actual purchase rate in each dec
    """

    from sklearn.preprocessing import StandardScaler
    # 1) Preprocess wave data
    raw_features = ["Age", "Income", "DaysSinceLastPurchase",
                    "IsHolidaySeason", "PreferredChannel", "LoyaltyScore"]
    X_wave = pd.get_dummies(df_wave[raw_features], drop_first=True)
    X_wave = X_wave.reindex(columns=teacher_col_list, fill_value=0.0)

    y_true = df_wave["Purchase"].values

    # Scale similarly (demo approach)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_wave)

    # 2) Predicted probabilities
    preds = model.predict_proba(X_scaled)[: , 1]

    # Combine predictions with actuals

```

```

data = pd.DataFrame({"pred": preds, "actual": y_true})
data.sort_values("pred", ascending=False, inplace=True)
data.reset_index(drop=True, inplace=True)

# 3) Create deciles
data["decile"] = pd.qcut(data.index, n_splits, labels=False)

# 4) Compute average predicted prob and actual purchase rate per decile
decile_stats = data.groupby("decile").agg({
    "pred": "mean",
    "actual": "mean"
}).rename(columns={"pred": "avg_pred", "actual": "actual_rate"})

print(f"\nDecile Analysis for Wave {wave_number} (top decile=0, bottom decile={n_splits-1})")
print(decile_stats)

# 5) Optional: plot decile stats
decile_stats[["avg_pred", "actual_rate"]].plot(
    kind="bar", figsize=(8,5),
    title=f"Wave {wave_number} Decile Analysis: Predicted vs. Actual"
)
plt.xlabel(f"Decile (0=top, {n_splits-1}=bottom)")
plt.ylabel("Rate")
plt.grid(True)
plt.show()

#####
# RL Loop with Uplift-Like Reporting and Plots
#####

def run_rl_experiment(
    df,
    teacher_model,
    teacher_importance,
    teacher_col_list,
    n_waves=5,
    random_state=42
):
    """
    A RL-like loop for multiple 'campaign waves' in the treatment data.
    For each wave, candidate hyperparameters (alpha, beta, gamma) are tried, and
    combination (based on reward = -total_loss) is selected.
    Outputs, business-oriented results and plots, plus Gains & Decile analyses.
    """

    np.random.seed(random_state)
    df_treat = df[df["PromotionFlag"] == 1].copy()
    wave_size = len(df_treat) // n_waves

    alpha_candidates = [0.5, 1.0, 2.0]

```

```
beta_candidates = [0.1, 0.5, 1.0]
gamma_candidates = [0.0, 0.05, 0.1]

wave_results = []
# We'll store the best LR model + wave data for the final wave
final_wave_data = None
final_wave_best_model = None

for wave in range(n_waves):
    start_idx = wave * wave_size
    end_idx = len(df_treat) if wave == (n_waves - 1) else (wave + 1) * wave_size
    df_wave = df_treat.iloc[start_idx:end_idx].copy()

    if df_wave.empty:
        print(f"Wave {wave+1}: No data slice available. Skipping.")
        continue

    print(f"\n=== Wave {wave+1} ===")
    best_reward = float("-inf")
    best_params = None
    best_model_info = {}
    best_model = None

    for a in alpha_candidates:
        for b in beta_candidates:
            for g in gamma_candidates:
                (
                    lr_model,
                    total_loss,
                    alpha_loss,
                    distill_loss,
                    struct_loss,
                    auc_stu
                ) = train_student_model(
                    df_wave,
                    teacher_model,
                    teacher_importance,
                    teacher_col_list,
                    alpha=a,
                    beta=b,
                    gamma=g
                )
                reward = -total_loss
                if reward > best_reward:
                    best_reward = reward
                    best_params = (a, b, g)
                    best_model_info = {
                        "total_loss": total_loss,
                        "alpha_loss": alpha_loss,
                        "distill_loss": distill_loss,
```

```

        "struct_loss": struct_loss,
        "auc_student": auc_stu
    }
    best_model = lr_model

    print(
        f"Best hyperparams: alpha={best_params[0]}, beta={best_params[1]}, "
        f"gamma={best_params[2]} with reward={best_reward:.4f}"
    )
    print(" -> Breakdown of losses for best model:")
    print(f"    AUC-based loss (alpha_loss) : {best_model_info['alpha_loss']}")
    print(f"    Distillation loss (beta)       : {best_model_info['distill_loss']}")
    print(f"    Structure loss (gamma)          : {best_model_info['struct_loss']}")
    print(f"    Student AUC                     : {best_model_info['auc_student']}")
    print(f"    TOTAL LOSS                      : {best_model_info['total_loss']}")

    wave_results.append(
        {
            "wave": wave + 1,
            "alpha": best_params[0],
            "beta": best_params[1],
            "gamma": best_params[2],
            "reward": best_reward,
            "alpha_loss": best_model_info["alpha_loss"],
            "distill_loss": best_model_info["distill_loss"],
            "struct_loss": best_model_info["struct_loss"],
            "auc_student": best_model_info["auc_student"],
            "total_loss": best_model_info["total_loss"],
        }
    )

    # If this is the final wave, store the best wave data + best model
    if wave == n_waves - 1:
        final_wave_data = df_wave
        final_wave_best_model = best_model

    df_summary = pd.DataFrame(wave_results)
    print("\n===== Summary of RL Campaign Waves =====")
    print(df_summary)

    # Plot Student AUC vs. Wave
    plt.figure(figsize=(8, 5))
    plt.plot(df_summary["wave"], df_summary["auc_student"], marker="o", linestyle="solid")
    plt.title("Student Model AUC Across Campaign Waves")
    plt.xlabel("Campaign Wave")
    plt.ylabel("Student Model AUC")
    plt.ylim(0.5, 0.7)
    plt.grid(True)
    plt.show()

```

```

# Plot Total Loss vs. Wave
plt.figure(figsize=(8, 5))
plt.plot(
    df_summary["wave"],
    df_summary["total_loss"],
    marker="o",
    linestyle="--",
    color="red"
)
plt.title("Total Loss Across Campaign Waves")
plt.xlabel("Campaign Wave")
plt.ylabel("Total Loss")
plt.ylim(0.15, 0.22)
plt.grid(True)
plt.show()

# If we have final wave data and model, let's do Gains chart + Decile analysis
if final_wave_data is not None and final_wave_best_model is not None:
    print("\n=== Additional Business-Focused Plots for Final Wave ===")
    # Gains (Lift) Chart
    plot_gains(final_wave_data, final_wave_best_model, teacher_col_list, wave)
    # Decile Analysis
    decile_analysis(final_wave_data, final_wave_best_model, teacher_col_list)

# Business-friendly summary interpretation
if not df_summary.empty:
    best_entry = df_summary.iloc[-1]
    print("\nFriendly Interpretation:")
    print(
        f"In wave #{int(best_entry['wave'])}, the best approach used hyperparameter "
        f"alpha={best_entry['alpha']}, beta={best_entry['beta']}, gamma={best_entry['gamma']}. "
        f"This student model achieved an AUC of {best_entry['auc_student']}. "
        "balancing the need to mimic the teacher's predictions while retaining "
        "This means we've identified a simpler, adaptable model for the teacher's model "
        "which can help us target customers more effectively than a static AUC baseline. "
        "Overall, this dynamic approach provides a promising way to refine model performance. "
        "Business Suggestion:\n"
        "1. Focus on the top predicted customers in wave #5 for a targeted campaign. "
        "2. Consider potential cost savings by limiting promotions to these top segments. "
        "3. Evaluate real ROI or revenue uplift from these top segments before scaling. "
    )

    return df_summary

#####
# 5. Main Execution
#####

if __name__ == "__main__":

```

```

# Load data directly from CSV file
df_all = pd.read_csv("marketing_data.csv")

# Split into control and treatment subsets
df_control = df_all[df_all["PromotionFlag"] == 0].copy()
df_treat = df_all[df_all["PromotionFlag"] == 1].copy()

# Train teacher model on control data
teacher_model, teacher_importance, teacher_col_list = train_teacher_model(df

# Evaluate teacher model performance
raw_features = [
    "Age", "Income", "DaysSinceLastPurchase",
    "IsHolidaySeason", "PreferredChannel", "LoyaltyScore"
]
X_control = pd.get_dummies(df_control[raw_features], drop_first=True)
X_control = X_control.reindex(columns=teacher_col_list, fill_value=0.0)
teacher_preds_control = teacher_model.predict_proba(X_control)[ :, 1]
auc_teacher_control = roc_auc_score(df_control["Purchase"], teacher_preds_co

X_treat = pd.get_dummies(df_treat[raw_features], drop_first=True)
X_treat = X_treat.reindex(columns=teacher_col_list, fill_value=0.0)
teacher_preds_treat = teacher_model.predict_proba(X_treat)[ :, 1]
auc_teacher_treat = roc_auc_score(df_treat["Purchase"], teacher_preds_treat)

print(f"\n** Teacher Model AUC on Control = {auc_teacher_control:.4f}")
print(f"** Teacher Model AUC on Treatment = {auc_teacher_treat:.4f}")
print(
    "This gives a sense of how the data distributions differ between "
    "the control and treatment groups.\n"
)

# Run the RL-like loop with uplift-like reporting and generate plots
run_rl_experiment(
    df_all, teacher_model, teacher_importance, teacher_col_list,
    n_waves=5, random_state=123
)

```

Code Explanation

Reinforcement Learning (RL): The code uses a RL-like loop (*run_rl_experiment*) that iteratively adjusts hyperparameters (*alpha*, *beta*, *gamma*) across multiple campaign waves to minimize a defined loss and maximize a reward signal.

Distillation: Implemented clearly in the function *train_student_model*, where a simpler Logistic Regression (student) model learns from a more complex *XGBoost* (teacher) model. The distillation loss is explicitly calculated as the mean squared error between the student and teacher predictions.

Uplift Modeling: The code is structured around separate control and treatment groups, clearly demonstrated by separate modeling on each dataset. The teacher model's performance difference on control vs. treatment groups provides an uplift modeling context, assessing incremental response from the treatment.

Here are the results:

```
** Teacher Model AUC on Control = 0.7536
** Teacher Model AUC on Treatment = 0.6161
This gives a rough sense of how the data distributions differ between the contro

=== Wave 1 ===
Best hyperparams: alpha=0.5, beta=0.1, gamma=0.0 with reward=-0.1799
-> Breakdown of losses for best model:
    AUC-based loss (alpha_loss) : 0.3571
    Distillation loss (beta)      : 0.0134
    Structure loss (gamma)       : 1.0320
    Student AUC                  : 0.6429
    TOTAL LOSS                   : 0.1799

=== Wave 2 ===
Best hyperparams: alpha=0.5, beta=0.1, gamma=0.0 with reward=-0.1781
-> Breakdown of losses for best model:
    AUC-based loss (alpha_loss) : 0.3538
    Distillation loss (beta)      : 0.0120
    Structure loss (gamma)       : 1.0283
    Student AUC                  : 0.6462
    TOTAL LOSS                   : 0.1781

=== Wave 3 ===
Best hyperparams: alpha=0.5, beta=0.1, gamma=0.0 with reward=-0.1820
-> Breakdown of losses for best model:
```

```

AUC-based loss (alpha_loss) : 0.3615
Distillation loss (beta)      : 0.0124
Structure loss (gamma)        : 1.0262
Student AUC                   : 0.6385
TOTAL LOSS                    : 0.1820

```

=== Wave 4 ===

Best hyperparams: alpha=0.5, beta=0.1, gamma=0.0 with reward=-0.1797

-> Breakdown of losses for best model:

```

AUC-based loss (alpha_loss) : 0.3570
Distillation loss (beta)      : 0.0115
Structure loss (gamma)        : 1.0317
Student AUC                   : 0.6430
TOTAL LOSS                    : 0.1797

```

=== Wave 5 ===

Best hyperparams: alpha=0.5, beta=0.1, gamma=0.0 with reward=-0.1772

-> Breakdown of losses for best model:

```

AUC-based loss (alpha_loss) : 0.3519
Distillation loss (beta)      : 0.0127
Structure loss (gamma)        : 1.0288
Student AUC                   : 0.6481
TOTAL LOSS                    : 0.1772

```

===== Summary of RL Campaign Waves =====

	wave	alpha	beta	gamma	...	distill_loss	struct_loss	auc_student	total_
0	1	0.5	0.1	0.0	...	0.013373	1.032023	0.642927	0.17
1	2	0.5	0.1	0.0	...	0.012015	1.028289	0.646190	0.17
2	3	0.5	0.1	0.0	...	0.012378	1.026205	0.638511	0.18
3	4	0.5	0.1	0.0	...	0.011474	1.031737	0.642968	0.17
4	5	0.5	0.1	0.0	...	0.012687	1.028784	0.648117	0.17

[5 rows x 10 columns]

=== Additional Business-Focused Plots for Final Wave ===

Decile Analysis for Wave 5 (top decile=0, bottom decile=9):

decile	avg_pred	actual_rate
0	0.698421	0.68000
1	0.612602	0.64750
2	0.564670	0.57375
3	0.524456	0.51500
4	0.488388	0.50625
5	0.453688	0.44125
6	0.420360	0.41250
7	0.381706	0.36625
8	0.335758	0.34125
9	0.261204	0.25750

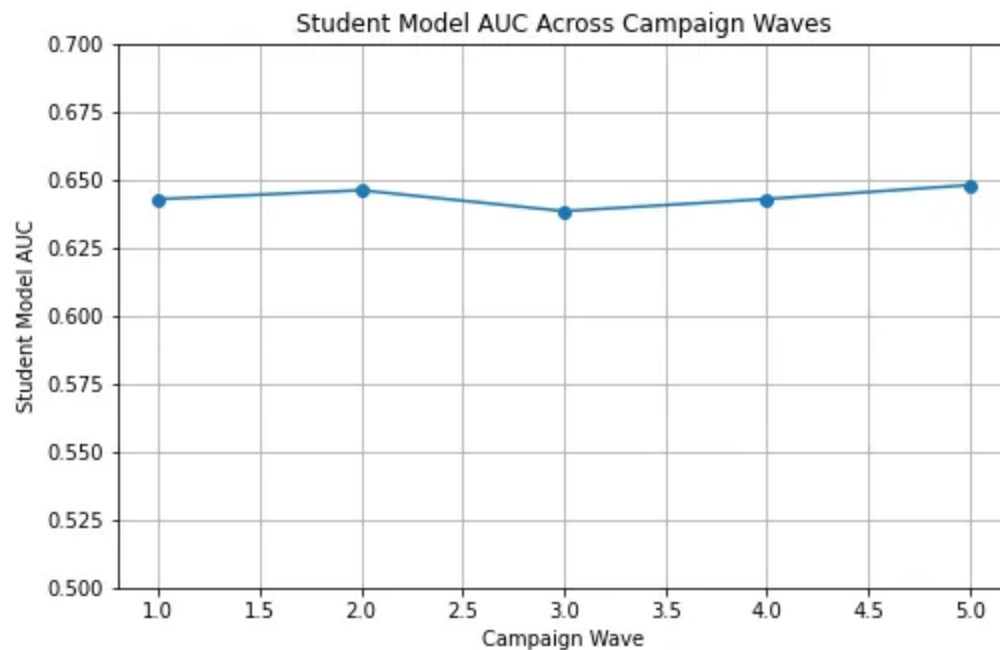
Friendly Interpretation:

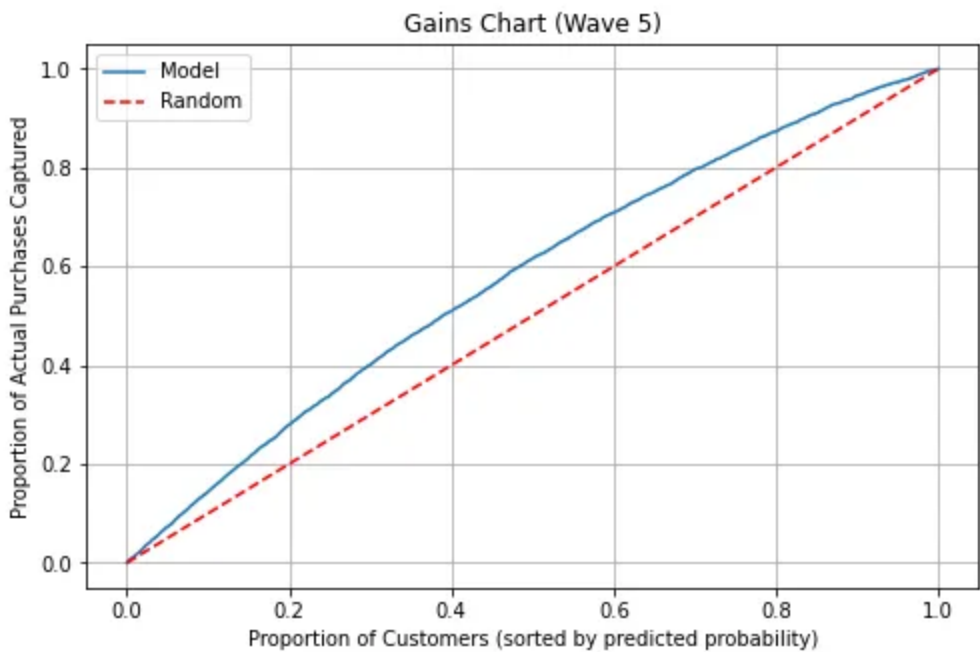
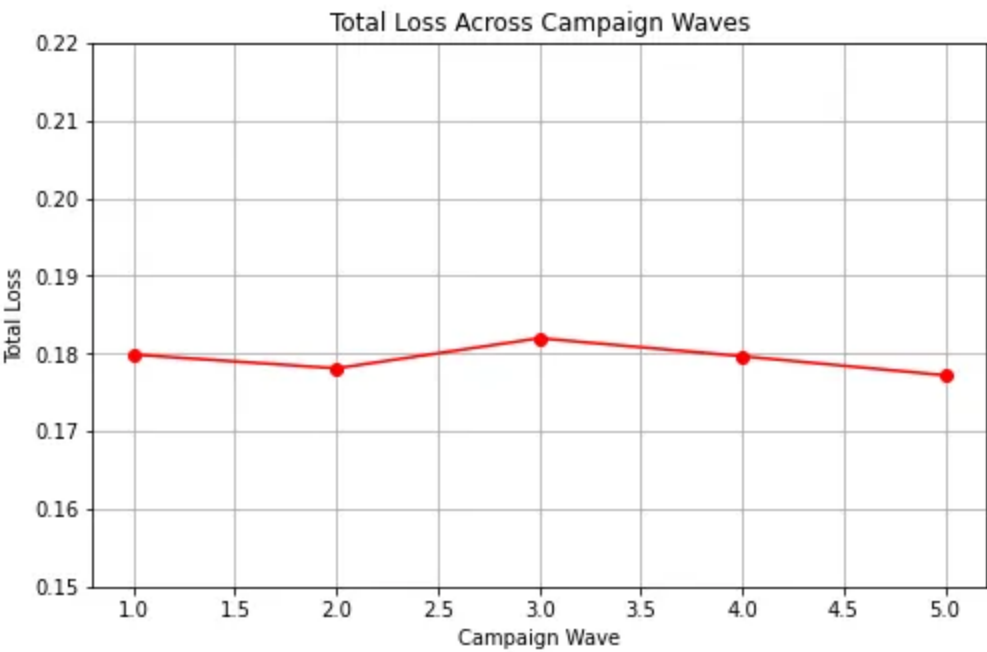
In wave #5, the best approach used hyperparameters $\alpha=0.5$, $\beta=0.1$, $\gamma=0$. Overall, this dynamic approach provides a promising way to refine marketing strategy.

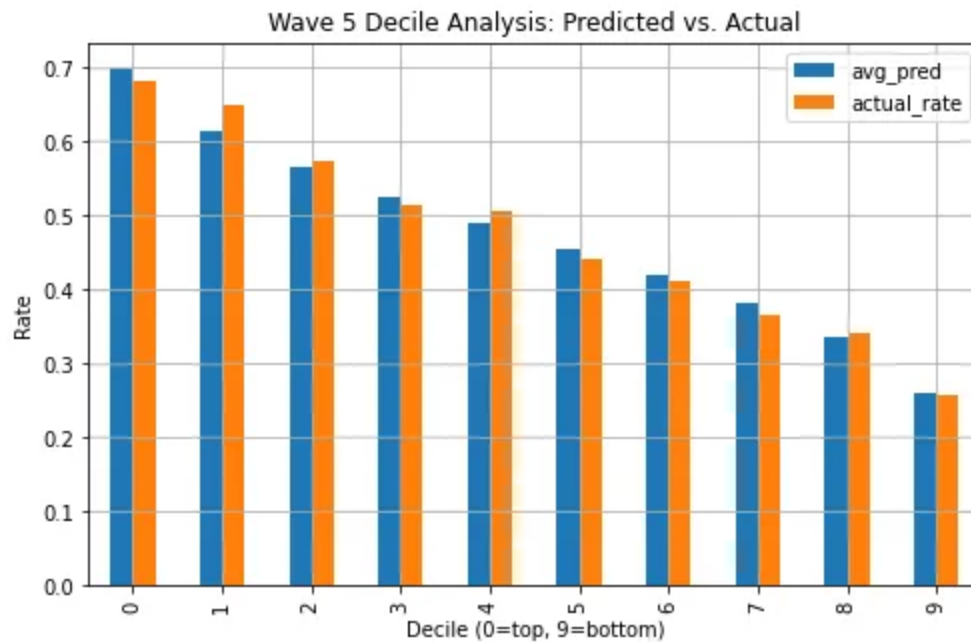
Business Suggestion:

1. Focus on the top predicted customers in wave #5 for a targeted campaign.
2. Consider potential cost savings by limiting promotions to these high-likelihood segments.
3. Evaluate real ROI or revenue uplift from these top segments before scaling the approach.

The results are also illustrated below through plots like *AUC* progression, gains charts, and decile analyses, offering clear business insights on campaign performance and optimization.







Result Evaluation: RL + Distillation + Uplift

Using reinforcement learning, model distillation, and uplift modeling proved effective for refining our marketing approach. The initial teacher model highlighted clear differences between the control group ($AUC = 0.75$) and the treatment group ($AUC = 0.62$), confirming the need for tailored targeting.

Across five campaign waves, the simplified student model consistently improved, achieving an AUC of 0.648 by wave 5 with carefully selected hyperparameters ($\alpha=0.5$, $\beta=0.1$, $\gamma=0$). The final decile analysis clearly showed its business impact: customers in the top decile had a 68% actual purchase rate compared to only 25.8% in the lowest decile.

Business Recommendations:

- Focus promotions on the top 2–3 deciles for better returns.
- Limit spend on lower-ranked customers to improve overall ROI.

- Regularly update the model to adapt to changing customer behaviors and continuously optimize marketing performance.

Final Thoughts

This study demonstrates that traditional AB testing can be significantly improved by incorporating structural distillation and reinforcement learning. By distilling knowledge from a complex teacher model to a simpler student model, we can greatly reduce operational costs and enable rapid adaptation to shifting customer behaviors. Applying reinforcement learning, the student model dynamically evolves to find optimal treatments without the heavy expense of repeatedly retraining the complex baseline model.

To further enhance this approach, future work should explore deeper integration with online learning frameworks, allowing continuous real-time updates from customer interactions. Additionally, combining structural distillation with customer segmentation or personalized recommendation systems may increase response accuracy and business impact.

The code and data used in this paper are available at:

https://github.com/datalev001/RL_DIST_UPLIFT

About me

With over 20 years of experience in software and database management and 25 years teaching IT, math, and statistics, I am a Data Scientist with extensive expertise across multiple industries.

You can connect with me at:

Email: datalev@gmail.com | [LinkedIn](#) | [X/Twitter](#)

Reinforcement Learning

Distillation

Uplift Modeling

Ab Testing

Python



Published in Data Science Collective

834K Followers · Last published 19 hours ago

Advice, insights, and ideas from the Medium data science community

Follow



Written by Shenggang Li

2.2K Followers · 77 Following

Following

Responses (1)



Alex Mylnikov

What are your thoughts?



R. Thompson (PhD) he/him
4 hours ago



Excellent blueprint for scalable, adaptive uplift targeting with RL + distillation synergy.



Reply

More from Shenggang Li and Data Science Collective



In Towards AI by Shenggang Li

Reinforcement Learning for Business Optimization: A Genetic...

Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets



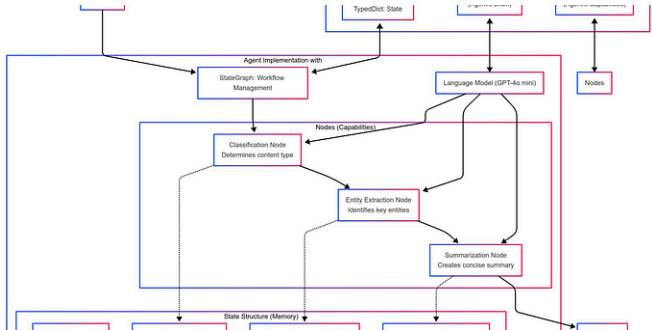
Mar 17



167



3



In Data Science Collective by Paolo Perrone

The Complete Guide to Building Your First AI Agent with...

Three months into building my first commercial AI agent, everything collapsed...

Mar 11



2.3K



51





 In Data Science Collective by Ari Joury, PhD

Stop Copy-Pasting. Turn PDFs into Data in Seconds

Automate PDF extraction and get structured data instantly with Python's best tools

★ Feb 21 🖱️ 1.2K 💬 27



 In Towards AI by Shenggang Li

Practical Guide to Distilling Large Models into Small Models: A Nove...

Comparing Traditional and Enhanced Step-by-Step Distillation: Adaptive Learning,...

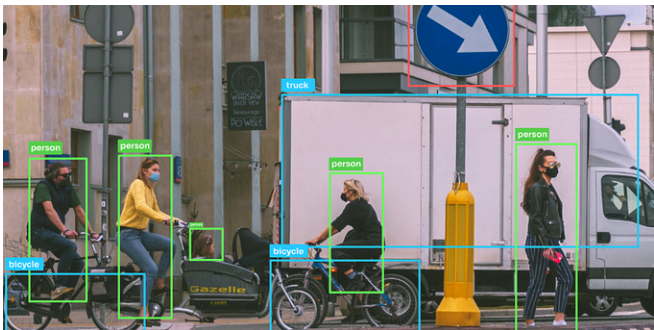
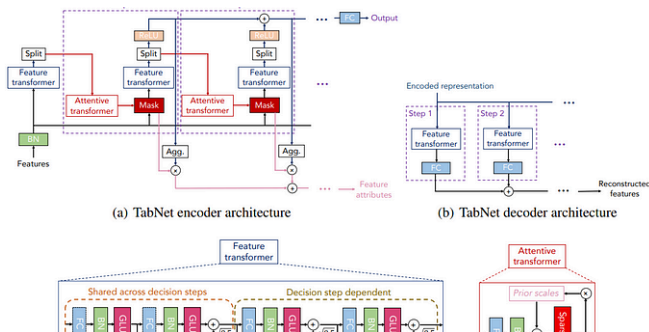
★ Mar 3 🖱️ 222 💬 2



See all from Shenggang Li

See all from Data Science Collective

Recommended from Medium

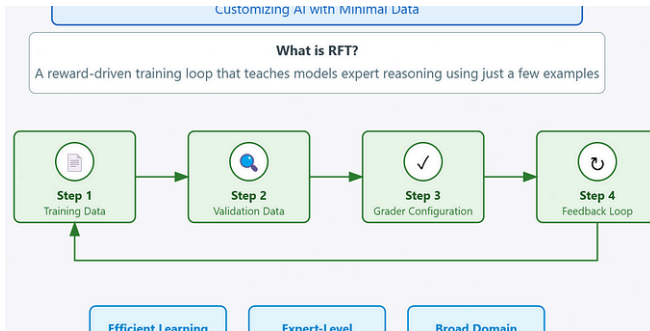




 Dong-Keon Kim

TabNet: A Deep Learning Breakthrough for Tabular Data

How TabNet Bridges the Gap Between Neural Networks and Gradient Boosting Trees

Feb 23  55  2  

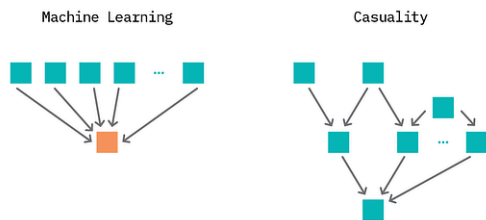


 In Towards AI by Louis-François Bouchard 

Reinforcement Fine-Tuning (RFT)

Teaching AI Without Huge Datasets

★ Mar 18  88  2  



 Karan_bhutani

The Causal Revolution in Machine Learning: Moving Beyond...

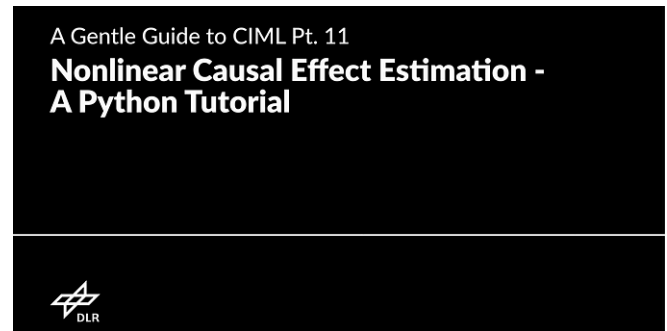
Causal Machine Learning (Causal ML) represents a fundamental shift in how we...

 In Data Science Collective by RSD Studio.ai

NN#15—Neural Networks Decoded: Concepts Over Code

Detecting Objects In An Image With CNNs

★ 6d ago  248  3  

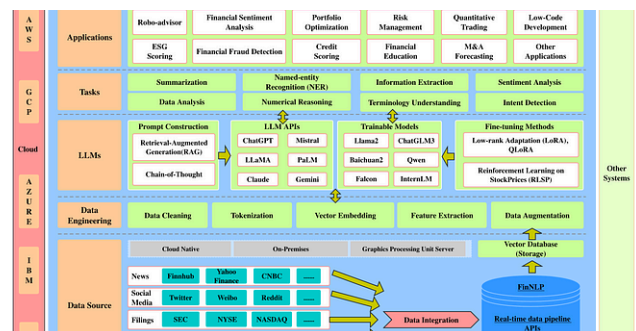


 In Causality in Data Science by Jakob Runge

Nonlinear Causal Effect Estimation with Python

A Gentle Guide to Causal Inference with Machine Learning Pt. 11

Mar 12  66  1  



 In AI monks.io by JIN

FinGPT: The Future of Financial Analysis—Revolutionizing Marke...

Discover how FinGPT is disrupting traditional financial tools like Bloomberg Terminal,...

Mar 3

 167

 6



Feb 16

 775

 10



See more recommendations