Towards AI

Member-only story

Adaptive Multi-Teacher Distillation for Enhanced Supervised Learning

A Novel Approach for Dynamically Combining Multiple Predictive Models into a Lightweight High-Performance Student Model

Shenggang Li · Subscribed

Published in Towards AI · 10 min read · Mar 24, 2025

Photo by Damon Hall on Unsplash

Introduction

In practical supervised learning, using a single predictive model like *XGBoost, LightGBM*, or *Random Forest* is standard. But often, combining these models boosts performance significantly. Traditional methods blend predictions from multiple models with fixed weights or logistic regression, treating each model equally across all predictions. This is easy but misses the chance to leverage each model's specific strengths based on different situations or inputs.

Our New Idea: Adaptive Multi-Teacher Distillation

To solve this limitation, I propose a novel approach: instead of static blending, I employ a lightweight neural network "student" that dynamically

learns from multiple sophisticated “teacher” models simultaneously. Each teacher — like *XGBoost* or *Random Forest* — predicts probabilities, which the student uses during training. Critically, the student network also learns attention weights, which dynamically determine how much each teacher influences each specific prediction. For example, when predicting if a customer will respond to a promotion, the student might rely more on *XGBoost* for younger customers but prefer Random Forest predictions for older customers.

Why This Method is Promising?

Unlike simple logistic blending, the adaptive distillation method offers two distinct advantages: Firstly, by allowing dynamic weighting for each prediction, the student model can tailor itself flexibly, capturing complex patterns no single model or static ensemble can. Secondly, distillation enables the student model to internalize valuable knowledge from all teachers into one compact neural network, delivering higher accuracy and better generalization performance. This approach results in a highly efficient and compact final model — perfect for deployment scenarios where speed, interpretability, and accuracy matter equally.

Multiple-Teacher Adaptive Distillation: A Novel Approach to Improve Predictive Modeling

Motivation: Combining Wisdom from Multiple Models

In machine learning, it’s common practice to rely on a single model — such as *XGBoost, LightGBM*, or *Random Forest* — to make predictions. However, just as diverse experts often make better decisions collectively, multiple predictive models (“teachers”) can guide a simpler, lightweight model

("student") to learn better overall patterns. This method is called knowledge distillation, and specifically, I explore a scenario called adaptive multiple-teacher distillation.

How it Works in Simple Terms?

Think of knowledge distillation as teachers (complex models) teaching a simpler student model to learn from their collective knowledge. Traditionally, the student learns from just one teacher or a fixed average of several teachers. Here, we propose a dynamic way for the student to intelligently select or weigh teachers based on each specific situation (input data). This adaptive method allows the student model to absorb the strengths of different teacher models selectively and dynamically.

Mechanism

Given a dataset with inputs X and labels y, suppose we train several different teacher models:

- *XGBoost*: $f_{XGB}(X)$
- *LightGBM*: $f_{LGB}(X)$
- *RandomForest*: $f_{RF}(X)$

Each of these models generates a prediction probability, representing the confidence that an input belongs to a particular class:

$$f_{teacher}(X) = [f_{XGB}(X), f_{LGB}(X), f_{RF}(X)]$$

Now, our student model *f_student(X,θ)* not only makes its own prediction but also simultaneously learns to dynamically allocate trust to each teacher. This trust or weighting mechanism is achieved through an attention-like approach:

Attention *weights(X) = [a_XGB, a_LGB, a_RF]*, where

$$a_{XGB} + a_{LGB} + a_{RF} = 1$$

The student model's prediction is trained to align closely with a weighted combination of teacher predictions:

$$f_{student}(X, \theta) \approx a_{XGB} \cdot f_{XGB}(X) + a_{LGB} \cdot f_{LGB}(X) + a_{RF} \cdot f_{RF}(X)$$

Adaptive Learning via Distillation Loss

To measure how closely the student aligns with the teachers, we define a special distillation loss, typically the mean squared error (*MSE*):

$$\text{Loss}_{teacher}(X, \theta) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t \in \{XGB, LGB, RF\}} a_{t,i} \left(f_t(X_i) - f_{student}(X_i, \theta) \right)^2$$

where *n* is the number of data points, and *a_{t, i}* is the dynamically computed attention weight for teacher *t* at data point *X_i*.

Additionally, the student also learns directly from the true labels *y* through a standard supervised binary cross-entropy loss:

$$\text{Loss}_{supervised}(X, y, \theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(f_{student}(X_i, \theta)) + (1 - y_i)\log(1 - f_{student}(X_i, \theta))]$$

The final loss used for training the student model is a combination of these two:

$$\text{Total Loss} = \text{Loss}_{supervised} + \text{Loss}_{teacher}$$

What Happens During Training?

Initially, the student has no preference among the teachers. As training progresses:

- The student's internal attention mechanism begins to discover which teacher models perform better for different types of inputs.
- Gradually, for each individual prediction, the student assigns higher weights to teachers that consistently produce accurate predictions for that input or similar inputs.
- Over epochs, the student's internal representation becomes optimized not just by mimicking teachers, but by selectively learning from the most relevant teachers at each step.

Interpreting Attention Weights

By examining the learned attention weights, we understand how the student balances teachers' advice:

- If attention weights are roughly equal, it suggests each teacher provides valuable complementary insights.

- If one or two teachers dominate, it indicates specialized strengths in certain feature spaces or sample subsets.

For example, in the provided results, we observed something like:

```
Mean Attention Weights (XGB,LGB,RF): [0.24, 0.38, 0.38]
```

This reveals that the student found more reliability in *LightGBM* and RandomForest for the tested dataset but still significantly utilized *XGBoost*'s predictions. Such an adaptive mechanism is more powerful than a fixed average because it naturally learns complex decision boundaries from diverse sources of information.

Why Does this Work Better?

- Robustness: Different teachers might specialize in different regions of input space or handle uncertainty differently.
- Adaptivity: Instead of learning from a fixed ensemble, the student flexibly learns from each teacher dynamically per input instance.
- Efficiency: The final student model remains computationally lightweight, combining diverse expertise into a single streamlined model.

Case Study: Adaptive Multi-Teacher Distillation in Marketing Analytics

I tested a new idea, Adaptive Multi-Teacher *Distillation*, where one lightweight "student" model dynamically learns from three advanced "teacher" models (*XGBoost, LightGBM, Random Forest*). Each teacher was trained individually to predict if customers would respond to a promotional campaign.

To protect privacy, I generated synthetic but realistic marketing data. It included key customer attributes like age, income, recent purchase activity, loyalty score, holiday indicators, and shopping channel preferences. I trained the models on customers who received no promotions (control group) and evaluated them on customers who did receive promotions (treatment group).

Instead of simply averaging teacher predictions, our student model learned dynamically how much to trust each teacher's predictions, depending on individual customer profiles. This approach ensured more personalized and accurate predictions by capturing the unique strengths of each teacher model.

```
# Updated version of the Bayesian distillation code with all 5 improvements appl

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score, accuracy_score
from scipy.stats import ks_2samp
import torch
import torch.nn as nn
import torch.optim as optim

df = pd.read_csv("marketdata.csv")
```

```
df = df[df["Promo"] == 1]  # Only use treatment data
train_df, test_df = train_test_split(df, test_size=0.4, random_state=42)

# Preprocessing
ohe = OneHotEncoder()
ch_train = ohe.fit_transform(train_df[["Channel"]]).toarray()
ch_test = ohe.transform(test_df[["Channel"]]).toarray()

X_train = pd.concat([
    train_df[["Age", "Income", "Days", "Holiday", "Loyalty"]].reset_index(drop=T
    pd.DataFrame(ch_train).reset_index(drop=True)
], axis=1)
y_train = train_df["Purchase"].values

X_test = pd.concat([
    test_df[["Age", "Income", "Days", "Holiday", "Loyalty"]].reset_index(drop=Tr
    pd.DataFrame(ch_test).reset_index(drop=True)
], axis=1)
y_test = test_df["Purchase"].values

# Convert column names to strings to avoid sklearn error
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.3, ran

# Train teacher models
xgb = XGBClassifier().fit(X_tr, y_tr)
lgb = LGBMClassifier().fit(X_tr, y_tr)
rf = RandomForestClassifier().fit(X_tr, y_tr)

teachers = {"XGBoost": xgb, "LightGBM": lgb, "RandomForest": rf}

def teacher_preds(X):
    return np.vstack([
        xgb.predict_proba(X)[:, 1],
        lgb.predict_proba(X)[:, 1],
        rf.predict_proba(X)[:, 1]
    ]).T

tp_train = teacher_preds(X_tr)
tp_val = teacher_preds(X_val)
tp_test = teacher_preds(X_test)
```

```
# Prepare PyTorch tensors
X_tr_t = torch.tensor(X_tr, dtype=torch.float32)
y_tr_t = torch.tensor(y_tr, dtype=torch.float32).unsqueeze(1)
tp_train_t = torch.tensor(tp_train, dtype=torch.float32)

X_val_t = torch.tensor(X_val, dtype=torch.float32)
y_val_t = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)
tp_val_t = torch.tensor(tp_val, dtype=torch.float32)

X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_test_t = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)
tp_test_t = torch.tensor(tp_test, dtype=torch.float32)

# Define student model
class AdaptiveDistillModel(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.shared = nn.Sequential(
            nn.Linear(input_dim, 128), nn.ReLU(),
            nn.Linear(128, 64), nn.ReLU(),
            nn.Linear(64, 32), nn.ReLU()
        )
        self.student_head = nn.Sequential(nn.Linear(32, 1), nn.Sigmoid())
        self.attn_layer = nn.Linear(32, 3)
        self.temperature = nn.Parameter(torch.tensor(1.0))

    def forward(self, x):
        h = self.shared(x)
        attn_logits = self.attn_layer(h) / self.temperature
        attn_weights = nn.Softmax(dim=1)(attn_logits)
        student_output = self.student_head(h)
        return student_output, attn_weights

model = AdaptiveDistillModel(X_tr.shape[1])
optimizer = optim.Adam(model.parameters(), lr=0.0005)
bce_loss_fn = nn.BCELoss()
kl_loss_fn = nn.KLDivLoss(reduction='batchmean')

best_auc, patience = 0, 10

for epoch in range(100):
    model.train()
    optimizer.zero_grad()
    student_pred, att_w = model(X_tr_t)
    blended_teacher = torch.sum(att_w * tp_train_t, dim=1, keepdim=True)
    blended_teacher_log = torch.log(blended_teacher + 1e-7)
    kl_loss = kl_loss_fn(blended_teacher_log, student_pred)
    y_smooth = y_tr_t * 0.9 + 0.05
    bce_loss = bce_loss_fn(student_pred, y_smooth)
    loss = bce_loss + 0.5 * kl_loss
```

```
        loss.backward()
        optimizer.step()

        model.eval()
        with torch.no_grad():
            val_pred, _ = model(X_val_t)
            val_auc = roc_auc_score(y_val, val_pred.numpy())
            if val_auc > best_auc:
                best_auc = val_auc
                torch.save(model.state_dict(), 'best_student.pt')
                patience = 10
            else:
                patience -= 1
                if patience == 0:
                    break
        print(f"Epoch {epoch+1}: Loss {loss.item():.4f}, Val AUC {val_auc:.4f}")

    # Evaluation
    # Evaluation
    model.load_state_dict(torch.load('best_student.pt'))
    model.eval()
    with torch.no_grad():
        test_pred, att = model(X_test_t)
        test_np = test_pred.numpy()
        auc = roc_auc_score(y_test, test_np)
        acc = accuracy_score(y_test, test_np > 0.5)
        ks_stat = ks_2samp(test_np[y_test == 1], test_np[y_test == 0])[0]
        att_mean = att.numpy().mean(axis=0)

    import matplotlib.pyplot as plt
    print("\n--- Final Student Model Performance ---")
    print(f"AUC: {float(auc):.4f}, Accuracy: {float(acc):.4f}, KS: {float(ks_stat):.
    print(f"Mean Attention Weights (XGB, LGB, RF): {att_mean}")

    # Evaluate teacher models
    print("\n--- Individual Teacher Model Performance ---")
    for name, model_t in teachers.items():
        prob = model_t.predict_proba(X_test)[:, 1]
        auc_t = roc_auc_score(y_test, prob)
        acc_t = accuracy_score(y_test, prob > 0.5)
        ks_t = ks_2samp(prob[y_test == 1], prob[y_test == 0])[0]
        print(f"{name} - AUC: {float(auc_t):.4f}, Accuracy: {float(acc_t):.4f}, KS:
```

The outputs from the code are presented below:

```
Epoch 1: Loss 0.6962, Val AUC 0.4801
Epoch 2: Loss 0.6946, Val AUC 0.5257
Epoch 3: Loss 0.6931, Val AUC 0.5747
Epoch 4: Loss 0.6916, Val AUC 0.6190
Epoch 5: Loss 0.6901, Val AUC 0.6522
Epoch 6: Loss 0.6887, Val AUC 0.6743
Epoch 7: Loss 0.6872, Val AUC 0.6894
Epoch 8: Loss 0.6857, Val AUC 0.7001
........
Epoch 27: Loss 0.6439, Val AUC 0.7448
Epoch 28: Loss 0.6407, Val AUC 0.7454
Epoch 29: Loss 0.6375, Val AUC 0.7460
Epoch 30: Loss 0.6342, Val AUC 0.7465
Epoch 31: Loss 0.6308, Val AUC 0.7470
Epoch 32: Loss 0.6274, Val AUC 0.7475
Epoch 33: Loss 0.6240, Val AUC 0.7479
Epoch 34: Loss 0.6207, Val AUC 0.7483
Epoch 35: Loss 0.6173, Val AUC 0.7487
Epoch 36: Loss 0.6140, Val AUC 0.7490
........
Epoch 43: Loss 0.5941, Val AUC 0.7512
Epoch 44: Loss 0.5920, Val AUC 0.7514
Epoch 45: Loss 0.5901, Val AUC 0.7517
Epoch 46: Loss 0.5885, Val AUC 0.7519
Epoch 47: Loss 0.5872, Val AUC 0.7521
Epoch 48: Loss 0.5861, Val AUC 0.7523
Epoch 49: Loss 0.5852, Val AUC 0.7524
Epoch 50: Loss 0.5846, Val AUC 0.7526
Epoch 51: Loss 0.5842, Val AUC 0.7527
Epoch 52: Loss 0.5839, Val AUC 0.7528
Epoch 53: Loss 0.5837, Val AUC 0.7529
Epoch 54: Loss 0.5836, Val AUC 0.7530
Epoch 55: Loss 0.5836, Val AUC 0.7530
.....
Epoch 97: Loss 0.5790, Val AUC 0.7539
Epoch 98: Loss 0.5790, Val AUC 0.7539
Epoch 99: Loss 0.5790, Val AUC 0.7539
Epoch 100: Loss 0.5789, Val AUC 0.7538

--- Final Student Model Performance ---
AUC: 0.7574, Accuracy: 0.6759, KS: 0.3803

Mean Attention Weights (XGB, LGB, RF):
[0.20416522 0.7506458  0.04519264]

--- Individual Teacher Model Performance ---
XGBoost - AUC: 0.7352, Accuracy: 0.6787, KS: 0.3471
```

```
LightGBM - AUC: 0.7513, Accuracy: 0.6913, KS: 0.3719
RandomForest - AUC: 0.7225, Accuracy: 0.6683, KS: 0.3280
```

Interpreting the Results: Smarter Model Blending with Attention

I tested three teacher models — *XGBoost, LightGBM*, and *Random Forest* — on treatment data. *LightGBM* came out on top with an *AUC* of *0.7513*, followed by *XGBoost* (*AUC 0.7352*) and Random Forest (*AUC 0.7225*).

Then I trained a student model using a weighted attention distillation method. Instead of just averaging predictions, the student learned how much to trust each teacher based on the input. Early on, the model wasn't great (*AUC ~0.48*), but as training progressed, it steadily improved.

student also achieved accuracy of *0.6759* and *KS* of *0.3803*.

What's cool is the attention weights: the student gave about *75%* weight to *LightGBM*, *20%* to *XGBoost*, and only *5%* to *Random Forest*. It automatically figured out which teacher to listen to more.

This approach is more flexible than fixed averaging and lets the student model learn smarter, more personalized predictions.

The dataset and related code are publicly available at: https://github.com/datalev001/dist_blend_mult/tree.

Conclusion

In this study, I proposed an adaptive distillation method inspired by Large Language Models (*LLMs*) and traditional model blending. Instead of using simple averaging, I created a lightweight student model that dynamically learned how to blend predictions from stronger teachers like *XGBoost*, *LightGBM*, and *Random Forest*. By combining these teachers into one adaptive loss function and using early stopping, the approach achieved better overall results compared to individual models.

The results confirmed that dynamic distillation gives practical advantages, improving prediction accuracy and flexibility for real-world scenarios. Looking forward, we could further boost performance by experimenting with different neural network architectures, refining attention mechanisms, or testing on more varied real-world datasets to confirm robustness across multiple business situations.

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

Distillation Supervised Learning Llm AI Python

Published in Towards AI

79K Followers · Last published 1 hour ago

Follow

The leading AI community and content platform focused on making AI accessible to all. Check out our new course platform: https://academy.towardsai.net/courses/beginner-to-advanced-llm-dev

Written by Shenggang Li

2.3K Followers · 77 Following

Subscribed

Responses (4)

Alex Mylnikov

What are your thoughts?

Willisjames

Apr 1

Becoming better hosted

Reply

Lee Herman
Mar 29

To add to Mainak's question, how is this method different from Stacking?

Reply

Mainak Sen
Mar 28

what is the difference between knodledge distilation and your method?

Reply

See all responses

More from Shenggang Li and Towards AI

In Towards AI by Shenggang Li

Reinforcement Learning for Business Optimization: A Genetic...

Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets

In Towards AI by Gao Dalie (高達烈)

Manus AI + Ollama: Build & Scrape ANYTHING (First-Ever General AI...

Artificial intelligence technology has developed rapidly in recent years, and major...

Mar 17 187 3 Mar 11 2K 19

In Towards AI by Gao Dalie (高達烈)

Gemma 3 + MistralOCR + RAG Just Revolutionized Agent OCR Forever

Not a Month Ago, I made a video about Ollama-OCR. Many of you like this video

Mar 24 552 6

In Towards AI by Shenggang Li

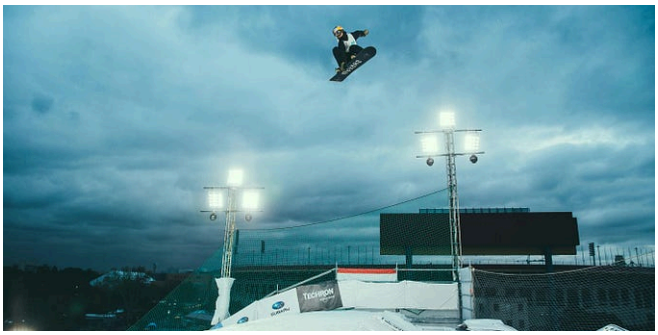
Reinforcement Learning-Enhanced Gradient Boosting Machines

A Novel Approach to Integrating Reinforcement Learning within Gradient...

Apr 1 276 3

See all from Shenggang Li See all from Towards AI

Recommended from Medium



In Towards AI by Shenggang Li

Reinforcement Learning-Enhanced Gradient Boosting Machines

A Novel Approach to Integrating Reinforcement Learning within Gradient...

Apr 1 276 3

In Data Science Collective by Buse Şenol

Model Context Protocol (MCP): An End-To-End Tutorial With Hands-...

What is MCP? How to create an MPC Server that brings news from a web site with Claude...

Mar 18 1.1K 12

The Data Beast

20 Cutting-Edge Statistical Techniques Every Data Scientist...

In today's fast-paced data world, traditional methods are evolving rapidly. In 2025, the...

Mar 7 212 2

In Learn AI for Profit by Nipuna Maduranga

You Can Make Money With AI Without Quitting Your Job

I'm doing it, 2 hours a day

Mar 24 3.2K 155

In Everyday AI by Manpreet Singh

Craziest MCP Servers You Must Try

I remember when I first heard about MCP (Model Context Protocol). I thought

Mar 9 1.4K 15

In Psyc Digest by Alessia Fransisca

The 1-Minute Introduction That Makes People Remember You...

A Behavioral Scientist's Trick to Hack the "Halo Effect"

Mar 19 13.7K 294

See more recommendations