Data Science Collective

Member-only story

Teaching Models to Choose Features Wisely: A Distill-to-Select Approach

Shenggang Li · Following

Published in Data Science Collective · 11 min read · Apr 2, 2025

249 1

A Novel Approach to Enhancing Transparency by Distilling Complex Predictors into Sparse, Human-Readable Models

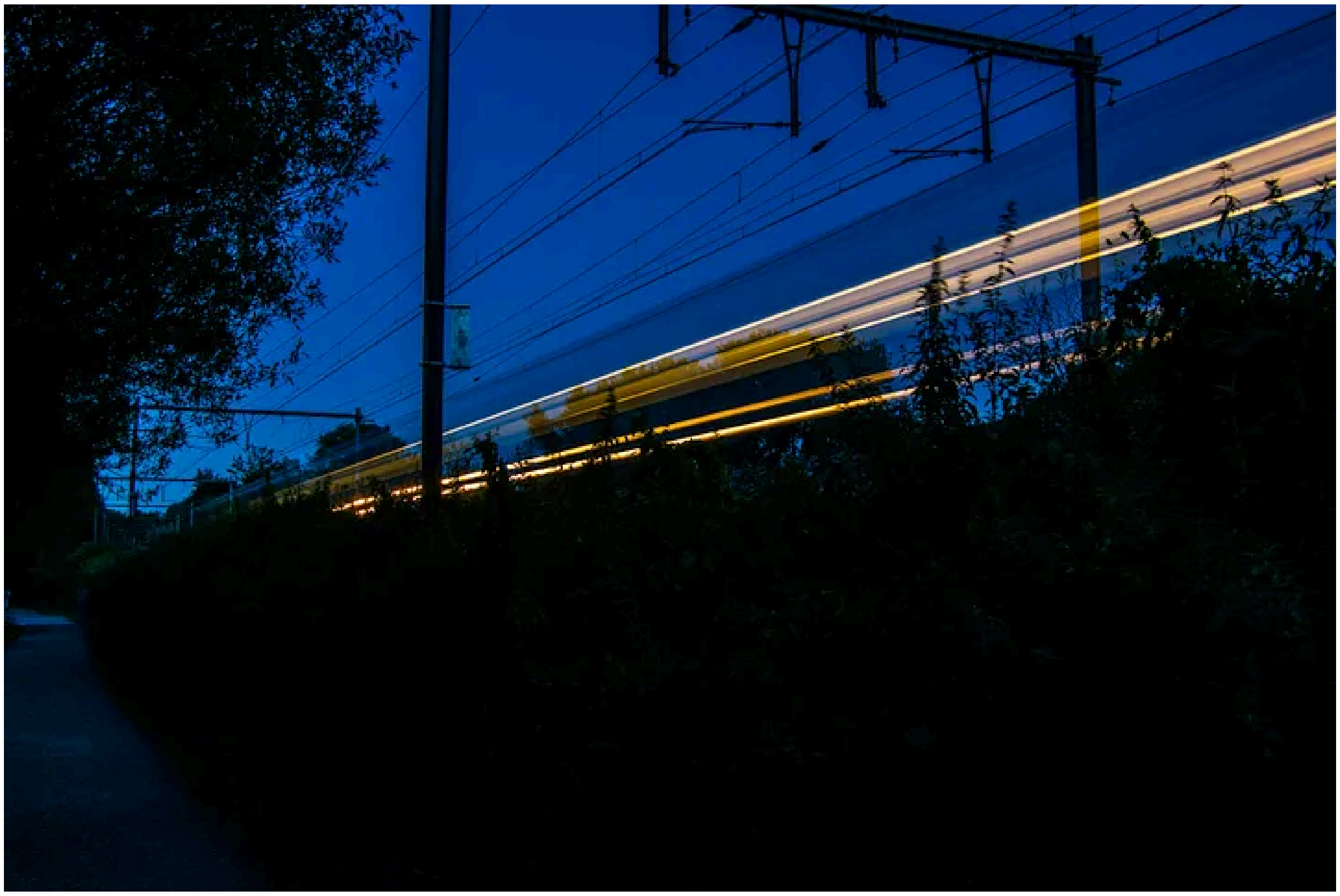


Photo by David Bruyndonckx on Unsplash

Introduction

Traditional models like *XGBoost* or *LightGBM* are great at making accurate predictions — but they can feel like black boxes. These models often use

though only one is truly essential. And in logistic regression, using standardized coefficients can help a bit, but it still doesn't solve the problem when features are correlated or noisy.

That's where our idea comes in.

Inspired by model distillation techniques in large language models, we introduce a simple but powerful approach: train a strong teacher model (like *XGBoost*) to do the hard work, then distill its knowledge into a lightweight student model — specifically, a sparse logistic regression. Think of it like learning from a wise professor: the professor knows everything in detail, but the student writes a clean summary using only the most important points.

Our initial experiments show that this distilled model performs almost as well as the full teacher model, while giving you something much easier to understand and explain. Plus, it automatically highlights the most important features — no need to rely on noisy importance rankings or hope that standardized coefficients tell the full story.

This teacher–student setup opens the door for even more flexibility. You can tweak the loss weights, try different types of teacher models, or even adapt it to other types of student models. Overall, it's a fresh and practical way to blend power and simplicity in predictive modeling.

Intuition and Algorithm Mechanisms

Understanding how our model works requires stepping through both the intuition and the mathematical formulation behind this teacher–student distillation process. We aim to make high-performing models easier to interpret and more selective in feature use, without compromising on prediction quality.

Why Not Just Use Feature Importance?

Popular models like *LightGBM* or *XGBoost* often report "feature importance" values based on things like split gain or frequency. However, these scores are heuristic and don't necessarily reflect the causal or predictive value of features, especially when variables are correlated. For instance, if "*Age*" and "*Income*" are highly correlated, both might show up as "important," even if only one truly drives predictions. In contrast, logistic regression uses coefficient magnitudes to imply importance, but these too can be misleading in the presence of multicollinearity or noise.

Our method tackles this challenge by introducing model distillation as a way to "summarize" the decision boundary of a complex model into a simple, sparse, and explainable model that selects features based on how they influence real decisions, not just how often they're used in trees.

The Teacher–Student Setup

The process begins by training a complex teacher model, such as *LightGBM*, on the entire dataset. This teacher model learns the mapping:

$$f_{\text{teacher}} : X \to \hat{p}_{\text{teacher}} = \mathbb{P}(Y = 1 \mid X)$$

Here, *X ∈ R^d* is the full feature set and *p^teacher ∈ [0,1]* is the teacher's predicted probability of the positive class. The teacher is expected to perform well, leveraging all features — relevant or not.

Next, we introduce a student model, specifically a logistic regression with *L1* regularization, designed to replicate the teacher's behavior, but using as few features as possible. The student model estimates:

$$f_{\text{student}}(x) = \sigma(w^\top x + b)$$

Where *σ(z)* is the sigmoid function, *w ∈ R^d* is the weight vector, and *b* is the bias term.

The Composite Loss Function

We train the student using a composite loss function that combines three elements:

Prediction Loss (Binary Cross Entropy)

This ensures the student learns from the true labels:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right]$$

Where,

$$\hat{p}_i = f_{\text{student}}(x_i)$$

Distillation Loss (KL Divergence)

This encourages the student to mimic the teacher's predicted probabilities:

$$\mathcal{L}_{\text{KL}} = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_i^{\text{teacher}}\log\left(\frac{\hat{p}_i^{\text{teacher}}}{\hat{p}_i}\right) + (1-\hat{p}_i^{\text{teacher}})\log\left(\frac{1-\hat{p}_i^{\text{teacher}}}{1-\hat{p}_i}\right)\right]$$

This term is central to knowledge distillation: the student doesn't just fit the data, but copies the decision boundary learned by the more expressive teacher.

Sparsity Loss (L1 Penalty)

This encourages the student to be sparse and perform feature selection:

$$\mathcal{L}_{\text{L1}} = \lambda \|w\|_1 = \lambda \sum_{j=1}^{d} |w_j|$$

The full objective becomes:

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \gamma \cdot \mathcal{L}_{\text{KL}} + \lambda \cdot \mathcal{L}_{\text{L1}}$$

Where:

- γ controls how much the student copies the teacher
- λ controls sparsity

This design creates a balance between accuracy and interpretability.

Why It Works

- The teacher captures full signal and nonlinear interactions.

- The student distills this into a linear boundary, only preserving what matters.
- The *L1* penalty prunes out features that don't contribute to the student's mimicry or true label fit.

Compared to traditional feature importance rankings, which are post hoc and sensitive to correlation, our method does in-model selection, directly optimizing for performance and simplicity together. It's like getting a curated summary of the teacher's thinking — clean, direct, and actionable.

Demonstration with a Marketing Campaign Dataset

To illustrate how the Distill-to-Select framework works in practice, we designed a realistic yet synthetic dataset simulating a marketing campaign scenario. The goal is to predict whether a customer makes a purchase after receiving a promotional offer. This dataset, available at GitHub — RL_GBM, contains *125,000* treatment group records and includes several customer attributes that commonly influence purchasing decisions.

Dataset Description

The features include:

- *Age*: Customer age in years.
- *Income*: Annual income.
- *Days Since Last Purchase*: Recency measure of last transaction.
- *Holiday*: Binary indicator for whether the promotion occurred during a holiday.

- *Channel*: Categorical variable indicating shopping method — Online, In-Store, or Mobile.
- *Loyalty Score*: A continuous measure of customer loyalty.

All features were standardized, and categorical variables were one-hot encoded to ensure compatibility with both tree-based and linear models.

Here is the code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score, accuracy_score
from scipy.stats import ks_2samp
import lightgbm as lgb
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

###########################
# Load Data
###########################

df = pd.read_csv('purchase_binary.csv')

# Use only treatment data
df = df[df["Promo"] == 1]

###########################
# Data Preprocessing
###########################
df_encoded = pd.get_dummies(df, columns=["Channel"], drop_first=True)
print("Dummy columns:", [col for col in df_encoded.columns if "Channel_" in col]
# For instance, assume columns "Channel_Mobile" and "Channel_Online" exist.
features = ["Age", "Income", "Days", "Holiday", "Loyalty", "Channel_Mobile", "Ch
target = "Purchase"

X = df_encoded[features].values.astype(np.float32)
```

```
y = df_encoded[target].values.astype(np.int64)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

###########################
# Train-Test Split (65% / 35%)
###########################
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.35, random_state=42, stratify=y)

###########################
# Train the Teacher Model (LightGBM)
###########################
teacher = lgb.LGBMClassifier(random_state=42)
teacher.fit(X_train, y_train)

teacher_train_prob = teacher.predict_proba(X_train)[:, 1]
teacher_test_prob = teacher.predict_proba(X_test)[:, 1]

teacher_auc = roc_auc_score(y_test, teacher_test_prob)
teacher_acc = accuracy_score(y_test, teacher.predict(X_test))
teacher_ks = ks_2samp(teacher_test_prob[y_test == 1], teacher_test_prob[y_test =

print("\nTeacher (LightGBM) Performance:")
print(f"AUC = {teacher_auc:.4f}, KS = {teacher_ks:.4f}, ACC = {teacher_acc:.4f}"

###########################
# Define the Student Model (Logistic Regression)
###########################
class LogisticRegressionStudent(nn.Module):
    def __init__(self, input_dim):
        super(LogisticRegressionStudent, self).__init__()
        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):
        logit = self.linear(x)
        prob = torch.sigmoid(logit)
        return prob

student = LogisticRegressionStudent(input_dim=X_train.shape[1])

###########################
# Define the Combined Loss Function
###########################
def kl_divergence(p_teacher, p_student, eps=1e-8):
    p_teacher = torch.clamp(p_teacher, eps, 1.0 - eps)
    p_student = torch.clamp(p_student, eps, 1.0 - eps)
    kl = p_teacher * torch.log(p_teacher / p_student) + (1 - p_teacher) * torch.
    return torch.mean(kl)
```

```
# Hyperparameters for distillation
gamma = 1.0       # weight for KL loss
lambda_l1 = 1e-4  # weight for L1 regularization

###########################
# Train the Student Model (Distillation) with Early Stopping
###########################
X_train_tensor = torch.from_numpy(X_train)
y_train_tensor = torch.from_numpy(y_train.reshape(-1, 1)).float()
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True)

optimizer = optim.Adam(student.parameters(), lr=0.001)
num_epochs = 100

# For early stopping
best_val_loss = float('inf')
patience = 10
best_epoch = 0
best_student_state = None

# Create validation tensors from X_test and y_test
X_val_tensor = torch.from_numpy(X_test)
y_val_tensor = torch.from_numpy(y_test.reshape(-1, 1)).float()

student.train()
for epoch in range(num_epochs):
    epoch_loss = 0.0
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        student_prob = student(batch_X)  # Student predictions

        # For simplicity, we use the first N teacher_train_prob values for this
        batch_indices = np.arange(batch_X.shape[0])
        teacher_probs = torch.from_numpy(teacher_train_prob[batch_indices]).floa

        # Compute BCE loss (true labels)
        bce_loss = nn.BCELoss()(student_prob, batch_y)
        # Compute KL divergence loss (distillation)
        kl_loss = kl_divergence(teacher_probs.squeeze(), student_prob.squeeze())
        # Compute L1 penalty on the student's weights (LASSO)
        l1_loss = torch.norm(student.linear.weight, 1)

        loss = bce_loss + gamma * kl_loss + lambda_l1 * l1_loss
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item() * batch_X.size(0)
    epoch_loss /= len(train_dataset)
```

```
        # Evaluate validation loss on the test set
        student.eval()
        with torch.no_grad():
            val_prob = student(X_val_tensor)
            val_loss = nn.BCELoss()(val_prob, y_val_tensor).item()
        student.train()

        print(f"Epoch {epoch+1}/{num_epochs} - Train Loss: {epoch_loss:.4f} - Val Lo

        # Check early stopping condition
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_epoch = epoch
            best_student_state = student.state_dict()
        elif epoch - best_epoch >= patience:
            print(f"Early stopping triggered at epoch {epoch+1}")
            break

    # Load best model state
    if best_student_state is not None:
        student.load_state_dict(best_student_state)

    ###########################
    # Evaluate the Student Model on Test Data
    ###########################
    student.eval()
    with torch.no_grad():
        X_test_tensor = torch.from_numpy(X_test)
        student_test_prob = student(X_test_tensor).cpu().numpy().flatten()

    student_pred = (student_test_prob >= 0.5).astype(int)
    student_auc = roc_auc_score(y_test, student_test_prob)
    student_acc = accuracy_score(y_test, student_pred)
    student_ks = ks_2samp(student_test_prob[y_test == 1], student_test_prob[y_test =

    print("\nStudent (Distilled Logistic Regression) Performance:")
    print(f"AUC = {student_auc:.4f}, KS = {student_ks:.4f}, ACC = {student_acc:.4f}"

    ###########################
    # Model Interpretability: Print Coefficients and Feature Importance
    ###########################
    # Extract weight and bias from the student's linear layer
    weights = student.linear.weight.detach().cpu().numpy().flatten()
    bias = student.linear.bias.detach().cpu().numpy()[0]

    # Pair each feature with its coefficient and sort by absolute value
    feature_importance = list(zip(features, weights))
    feature_importance = sorted(feature_importance, key=lambda x: abs(x[1]), reverse

    print("\nStudent Model Coefficients and Feature Importance:")
    print(f"Intercept (bias): {bias:.4f}")
    for feature, coef in feature_importance:
        print(f"Feature: {feature:20s} Coefficient: {coef:.4f}")

    # Define a threshold for feature selection (e.g., absolute coefficient > 0.1)
    threshold = 0.1
    selected_features = [feat for feat, coef in feature_importance if abs(coef) > th
    print("\nFeatures Selected (|coefficient| > 0.1):")
    print(selected_features)
```

Here are the results:

```
Teacher (LightGBM) Performance:
#######################################################
AUC = 0.7566, KS = 0.3824, ACC = 0.6948
Epoch 1/100 - Train Loss: 0.7836 - Val Loss: 0.6190
Epoch 2/100 - Train Loss: 0.7400 - Val Loss: 0.6103
Epoch 3/100 - Train Loss: 0.7382 - Val Loss: 0.6100
Epoch 4/100 - Train Loss: 0.7380 - Val Loss: 0.6083
Epoch 5/100 - Train Loss: 0.7380 - Val Loss: 0.6095
Epoch 6/100 - Train Loss: 0.7380 - Val Loss: 0.6089
Epoch 7/100 - Train Loss: 0.7383 - Val Loss: 0.6091
Epoch 8/100 - Train Loss: 0.7373 - Val Loss: 0.6084
Epoch 9/100 - Train Loss: 0.7387 - Val Loss: 0.6096
Epoch 10/100 - Train Loss: 0.7376 - Val Loss: 0.6085
Epoch 11/100 - Train Loss: 0.7378 - Val Loss: 0.6093
Epoch 12/100 - Train Loss: 0.7379 - Val Loss: 0.6088
Epoch 13/100 - Train Loss: 0.7376 - Val Loss: 0.6087
Epoch 14/100 - Train Loss: 0.7377 - Val Loss: 0.6094
Early stopping triggered at epoch 14

Student (Distilled Logistic Regression) Performance:
#######################################################
AUC = 0.7585, KS = 0.3822, ACC = 0.6846

Student Model Coefficients and Feature Importance:
#######################################################
Intercept (bias): 0.2965
Feature: Income                Coefficient: 0.4101
Feature: Age                   Coefficient: 0.1721
```

```
Feature: Holiday                Coefficient: 0.1065
Feature: Days                   Coefficient: -0.0312
Feature: Loyalty                Coefficient: -0.0076
Feature: Channel_Online         Coefficient: -0.0051
Feature: Channel_Mobile         Coefficient: 0.0024

Features Selected (|coefficient| > 0.1):
['Income', 'Age', 'Holiday']
```

Explanation:

1. **Teacher Model (*LightGBM*)**

We first trained a *LightGBM* classifier using all features. As expected, the model delivered strong performance:

- ***AUC*** = *0.7566*
- **Accuracy** = *0.6948*
- ***KS* Statistic** = *0.3824*

However, while this model achieves high accuracy, it relies on all features — even those that may not be essential for interpretation or decision-making. For example, *LightGBM* may assign importance to both "*Channel_Online*" and "*Channel_Mobile*" due to their correlation with purchases, even if only one is truly useful for modeling purposes. Feature importance scores alone don't offer a clear path to select a sparse, actionable feature set.

2. Student Model via Distillation

Next, we used the Distill-to-Select framework to train a *logistic* regression model as the student. This student learns to:

- Predict actual purchase labels (true supervision),
- Mimic the *LightGBM* probability output (distillation),
- And eliminate unimportant features via an *L1* regularization term (sparsity).

The student is trained using our composite loss:

$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{BCE}}}_{\text{fit labels}} + \gamma \cdot \underbrace{\mathcal{L}_{\text{KL}}}_{\text{mimic teacher}} + \lambda \cdot \underbrace{\|w\|_1}_{\text{sparsity}}$$

Hyperparameters *γ* and *λ* were tuned for performance and interpretability balance. During training, early stopping was used to prevent overfitting on the recent validation loss.

3. Results and Interpretation

The distilled logistic regression model achieved performance on par with the teacher:

- ***AUC*** = *0.7585*
- **Accuracy** = *0.6846*
- ***KS*** **Statistic** = *0.3822*

More importantly, the student model produced a sparse and interpretable representation:

- It retained only *3–4* meaningful features.
- The largest coefficients were assigned to "Income" (*0.41*), "*Age*" (*0.17*), and "*Holiday*" (*0.11*).
- Features like "*Channel_Online*" and "*Loyalty*" had negligible weights and were effectively pruned.

By applying a simple coefficient threshold (e.g., *|β_j|>0.1*), we automatically select the core predictive features and exclude noise. The final student model becomes a compact, human-readable equation that a business analyst can easily interpret:

$$\mathrm{logit}(p) = 0.41 \cdot \mathrm{Income} + 0.17 \cdot \mathrm{Age} + 0.11 \cdot \mathrm{Holiday} + \mathrm{bias}$$

Conclusion

The Distill-to-Select approach shows that model distillation isn't just useful for simplifying complex models — it's also a smart way to perform feature selection. Unlike traditional methods like feature importance scores or standardized coefficients, this technique builds feature selection into the training process itself. By combining label accuracy, teacher mimicry, and feature sparsity into a single loss function, the resulting model is both compact and aligned with the reasoning of a high-performing teacher.

More importantly, this is not limited to just *LightGBM* or logistic regression. Distill-to-Select is a flexible, model-agnostic framework. The teacher could be any complex model — *XGBoost*, deep neural nets, or even *LLMs* like *BERT*.

The student can be any simple, interpretable model — not just logistic regression. For example, this framework could be extended to *RAG*-based *LLM* systems, helping to select fewer, more relevant documents at retrieval time, making the process leaner and more transparent.

There are many ways to take this further. We could explore nonlinear student models (like sparse neural nets or shallow trees), develop adaptive loss weighting to better balance performance and simplicity, or even implement dynamic distillation, where students are periodically updated as the teacher evolves over time.

In short, Distill-to-Select is a practical recipe for turning any black-box model into something clear, efficient, and actionable. Whether you're in healthcare, finance, marketing, or *NLP*, this method helps you focus on what truly matters — without sacrificing accuracy.

The source code is available at:
https://github.com/datalev001/distill_feature_selection

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

AI Distillation Feature Selection Predictive Modeling Python

Published in Data Science Collective

Follow

835K Followers · Last published 14 hours ago

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

Written by Shenggang Li

Following

2.5K Followers · 77 Following

Responses (1)

Alex Mylnikov

What are your thoughts?

Ashmpatel
Apr 8

Very interesting article,useful too. Can compare performance to principle component analysis which is the typical go to tool for feature importance.

Reply

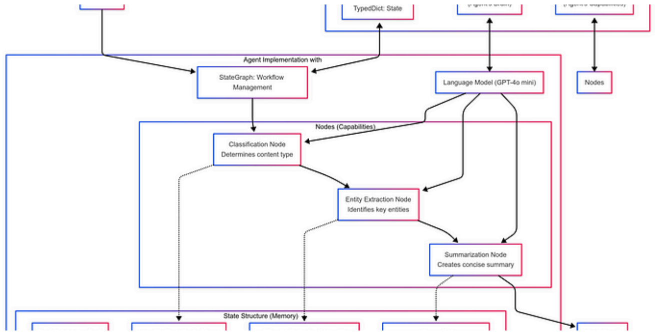
More from Shenggang Li and Data Science Collective

In Towards AI by Shenggang Li

Reinforcement Learning-Enhanced Gradient Boosting Machines

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

Apr 1 387 3


TypedDict: State
Agent Implementation with
StateGraph: Workflow Management
Language Model (GPT-4o mini)
Nodes
Nodes (Capabilities)
Classification Node Determines content type
Entity Extraction Node Identifies key entities
Summarization Node Creates concise summary
State Structure (Memory)

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 2.9K 61

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 1.2K 15

In Towards AI by Shenggang Li

Reinforcement Learning for Business Optimization: A Genetic...

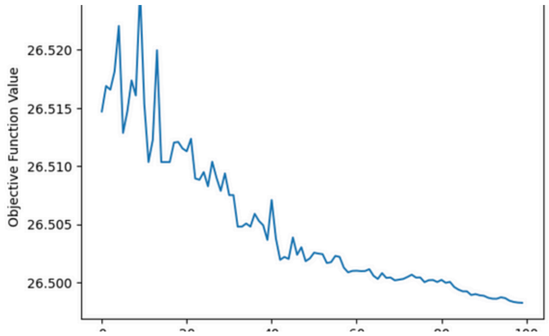
Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets

Mar 17 188 3

See all from Shenggang Li

See all from Data Science Collective

Recommended from Medium



Xin Cheng

Quantum Machine Learning for MNIST classification

Harnessing Qubits to Read Digits: MNIST Meets Quantum Circuits

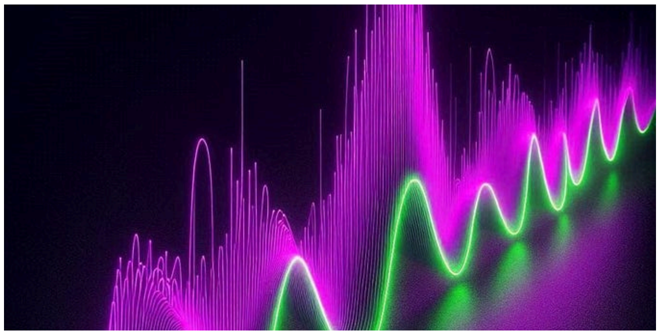
3d ago 1

Valeriy Manokhin, PhD, MBA, CQF

Predicting Full Probability Distributions with Conformal...

Introduction

Mar 28 294 1

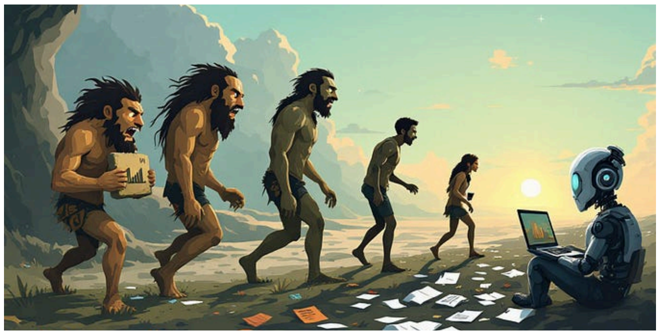


In The Forecaster by Nikos Kafritsas

Influential Time-Series Forecasting Papers of 2023–2024: Part 2

The 2nd part of innovations in time-series forecasting

Mar 31 110 1

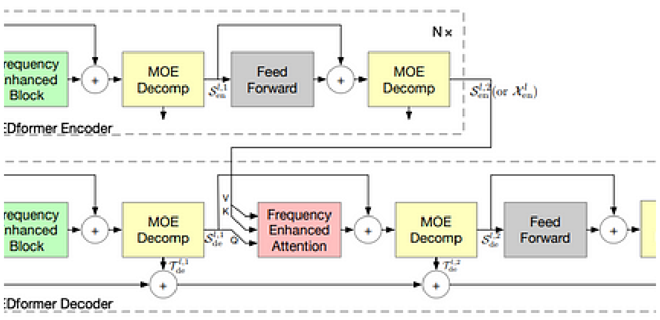


In Data Science Collective by Nathan Rosidi

Data Science Is Dead (Again): Why the Role Keeps Evolving, Not...

Data science isn't obsolete—it's shifting. Learn how AI changes workflows, what skills...

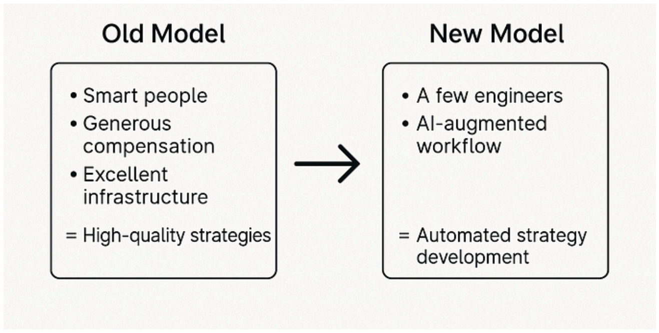
Mar 28 168 6



Dong-Keon Kim

FEDformer: Unleashing the Power of Frequency in Time Series...

A Deep Dive into Frequency Enhanced Decomposed Transformers for Long-Term...



In InsiderFinance Wire by Fisher Lok

Can AI Reshape the Quantitative Trading Business?

AI—especially large language models—are changing the economics of quant research b...

5d ago 18

Apr 8 41

See more recommendations