Data Science Collective

Member-only story

Enhancing Time Series Forecasting with Dynamic Weighted Trees

Shenggang Li · Following

Published in Data Science Collective · 25 min read · 2 days ago

46 1

A Practical, High-Performance, and Interpretable Decision Tree Approach Incorporating Autoregressive Lags, Seasonal Dynamics, and Recency-Driven Weighting

Photo by Mantas Hesthaven on Unsplash

Introduction

In this paper, I proposed a practical, high-performance, and interpretable decision tree framework for time series forecasting that combines autoregressive lags, seasonal patterns, and recency-based weighting. The method builds on established *AR* practices by leveraging lagged values and cyclic features, then applying an exponential decay so that more recent data exerts greater influence.

I unify these techniques in a decision tree that learns clear, recursive rules. By incorporating weighted lags, seasonal indicators, and a moving average, the model rapidly adapts to changing data conditions. Additionally, the tree naturally captures nonlinear behaviors — eliminating the need for complex ensemble configurations.

This approach effectively integrates familiar linear *AR* strategies with the power of nonlinear modeling. It is easy to implement, delivers consistently strong forecasting results, and handles noise and data shifts more reliably than traditional models.

Moreover, this framework is flexible. You can adopt it as-is or extend it into more advanced paradigms, such as boosted tree time series forecasting. And because decision trees are inherently interpretable, the model provides transparent, straightforward rules that explain how predictions are made.

A Weighted Tree Approach for Time Series Forecasting

Algorithmic Motivation for a Tree-Based Time Series Method

Time series forecasting often relies on autoregressive (*AR*) principles, which assume linear relationships among lagged observations. For example, an *AR(1)* model might predict

$$y_{t+1} = \beta_0 + \beta_1 y_t + \epsilon$$

, focusing on a direct linear correlation with yesterday's value. However, real-world data can exhibit nonlinear swings and seasonal repetition that pure linear models miss. A decision tree tackles this by learning threshold-based splits — for instance, "*if y_{t−1} > 100*" then predict high, else predict low." Yet, a basic tree ignores time-based nuances, so we integrate:

1. Lagged Features (e.g., *y_{t-1}, y_{t-2}, ,...*) to capture *AR* effects,

2. Cyclical Patterns (e.g., *sin(2π × day/7)* to represent repeated weekly or monthly cycles, and

3. Exponential Decay Weights so that recent data points have stronger

change happens on Monday, data from two years ago is now less important. By assigning lower weights to older observations, the tree emphasizes recent events. Meanwhile, a sine or cosine representation of day-of-week helps the model split on seasonal patterns, and lagged values store immediate history. This combination offers a flexible, intuitive approach that outperforms conventional *AR* formulas in fast-changing environments, delivering both nonlinear learning and a user-friendly "*if–then*" structure for clarity.

Core Components: Decay, Moving Average, and Cycles

A major innovation of this framework is weaving time-relevant adjustments into the tree's standard regression process. We do this through three core components:

Decay Weighting

Instead of treating all historical data equally, each sample *(x_t, y_t)* gets a weight

$$w_t = \exp\big(-\alpha \times (T_{\max} - t)\big)$$

where *T_{max}* is the most recent training index, and α\alphaα is a small positive constant controlling how quickly old data fades. For instance, if *α* =

0.01 and our dataset extends from *t = 1* to *t = 1000*, the observation at *t = 990* might have a weight around 0.90, while the observation at *t = 500* could be below *0.20* — emphasizing that the more recent point is more relevant for training.

Example: Suppose we track daily sales for *10* days, from *t=1* to *t = 10*. If *α=0.05*, then *t=10* gets a weight of *e− 0.05×(10−10) = 1*, but *t=1* gets *e−0.05×(10−1)≈0.64*. This numeric gap highlights how older data becomes less dominant.

Moving Average (MA)

Beyond raw lagged values, we also compute a short-term average over the last *k* observations. For a window of 3, the *MA* at time *t* is

$$\mathrm{MA}_t = \frac{y_{t-1} + y_{t-2} + y_{t-3}}{3}$$

which helps smooth out single-day anomalies. A sudden spike on one day won't entirely throw off the model if the average remains stable. The tree sees this moving average as a "summary" feature — if *MA_t* is high, it might split on that threshold and predict accordingly.

Cyclical Inputs

Many time series have repeating patterns (daily, weekly, monthly). A common trick is to encode these cycles with sine and cosine transformations. For a weekly cycle, we define:

$$\text{sin\_dow} = \sin\left(\frac{2\pi \times \text{day\_of\_week}}{7}\right), \quad \text{cos\_dow} = \cos\left(\frac{2\pi \times \text{day\_of\_week}}{7}\right)$$

If *day_of_week* cycles from 0 (Monday) to 6 (Sunday), *sin_dow* and *cos_dow* trace out a smooth circle, avoiding awkward jumps from day 6 back to day 0. Imagine a scenario where usage is notably higher on weekends: the tree can pick up a threshold split like "if *cos_dow<−0.5*, usage is high," effectively isolating the weekend effect.

Collectively, these features allow the tree to make nuanced splits that reflect both immediate recency (through decay weighting), short-term level (moving average), and repetitive seasonality (*sine/cosine*). In simpler *AR* terms, one might attempt:

$$\beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \dots$$

, but in this approach, the tree is free to discover more complex boundary rules that can be far more effective when data trends fluctuate.

Construction to Forecasting: Theory and Steps

Once we've assembled these features, we train a regression tree using a weighted least squares loss. Formally, if *(x_i, y_i)* are training pairs — with *x_i* including lagged values, *MA_i*, and cyclical indicators — then each point has a weight *w_i*. My goal is to minimize:

$$\sum_{i=1}^{n} w_i \left(y_i - \hat{y}_i\right)^2$$

where *y^i* is the tree's prediction for *y_i*. At each branch, the tree splits on the feature threshold that best reduces this weighted sum of squared errors. Eventually, the data is partitioned into leaf nodes, each node taking the weighted average of its assigned *y*-values as the prediction.

Why Weighted Splits Help

If new data indicates a sudden shift — say, demand skyrockets this month — older points in the training set (with lower weights) won't pull the model away from noticing that change. Hence, the tree's splits become more reflective of current conditions, often boosting performance under fast-evolving time series.

Generating Forecasts

After the tree is trained, forecasting proceeds recursively:

Collect the last L observations (e.g., *y_{t}, y_{t-1}, … y_{t-L+1}*) and calculate any additional features (moving average over those points, plus the appropriate sine and cosine values for the forecast date).

Query the tree with this feature vector. The tree navigates from the root to a leaf, guided by splits like "if *sin_dow* > *0.5*, go left; else, go right".

Obtain the predicted value *y_{t+1}* from that leaf.

Append y_{t+1} to the series, so it becomes part of the lag set when predicting y_{t+2}.

Repeat until you forecast as many steps as needed, whether it's a week, a month, or beyond.

Interpretability and Better Performance

Unlike some "black-box" methods, regression trees are interpretable. We can print out a path of rules — like "if $MA \leq 120$ and $cos\_dow \leq -0.3$, predict 130" — to see exactly how the forecast was made. This helps decision makers trust the model and see the role that seasonality, recent lags, and weighting each play.

Performance often improves because the tree can handle nonlinearities (e.g., major spikes on certain weekdays) better than purely linear models. The exponential weighting also makes it agile in the face of shifting trends; older data gets progressively less emphasis, so the model adapts more readily.

Bringing it all together, this method merges well-established AR-like ideas (lags, moving averages) with cyclical encodings and a robust weighting mechanism, embedding them into a classic decision tree that yields strong, intuitive results. By minimizing a weighted squared-error loss, the approach focuses more on recent data without discarding the past entirely, and the final model is straightforward to apply: generate your features, feed them into the tree, and let the rules produce a forecast. The synergy of interpretability, performance, and simplicity makes this framework an attractive option for a wide range of time series forecasting tasks.

Code Test and Result Evaluation

In this section, I apply the time series framework to a daily electricity production dataset from *Kaggle*. The goal is to illustrate how decay weighting, moving averages, and cyclical features combine within a decision tree structure to capture both short-term fluctuations and seasonal cycles. I also compare performance against standard *AR* and *ARMA* models (fitted via *OLS*).

Key parts of the code include the *construct_lagged_dataset()* function, which creates input vectors by stacking lagged values, *sine/cosine* transformations of the day-of-week, and a short-term moving average. Note, this step also applies an exponential decay factor so that older observations carry less weight. Then, I train a *DecisionTreeRegressor* by specifying these weights in the *fit()* call (*sample_weight = weights_train*). This ensures the model emphasizes recent data when splitting and forming its final leaf predictions. After running the code, I collect *MAPE* values at various horizons and print out the tree's partition rules. These outputs reveal how the weighted tree outperforms simpler models and clarifies which features (like certain lags or days) drive the most impactful splits.

```
import pandas as pd
import numpy as np
import math
import datetime
import re
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import train_test_split

# Import statsmodels for OLS estimation
import statsmodels.api as sm
```

```
# ----------------------
# Suppress warnings (if any)
# ----------------------
import warnings
warnings.filterwarnings("ignore")

# ======== Helper Functions for Tree Model =========

def add_cycle_features(df, date_col="DATE"):
    """
    Adds cyclical features based on day-of-week.
    """
    df[date_col] = pd.to_datetime(df[date_col])
    df['dayofweek'] = df[date_col].dt.dayofweek  # Monday=0, Sunday=6
    df['sin_dow'] = np.sin(2 * np.pi * df['dayofweek'] / 7)
    df['cos_dow'] = np.cos(2 * np.pi * df['dayofweek'] / 7)
    return df

def add_ma_feature(series, window=3):
    """
    Computes a moving-average feature over a given window.
    """
    return series.rolling(window=window, min_periods=1).mean()

def construct_lagged_dataset(df, target_col, L, decay_rate, date_col="DATE"):
    """
    For each time t (>= L) build feature vector composed of:
      - L lagged values (most recent first)
      - Current cyclical features (sin_dow, cos_dow)
      - MA feature: average of previous 3 observations
    Also computes an exponential decay weight.
    """
    df = df.copy().reset_index(drop=True)
    n = len(df)
    X, y, weights, dates = [], [], [], []
    T_max = n - 1
    for t in range(L, n):
        lag_feats = df[target_col].iloc[t-L:t].values[::-1].tolist()
        sin_dow = df.loc[t, 'sin_dow']
        cos_dow = df.loc[t, 'cos_dow']
        ma_feat = df[target_col].iloc[max(t-3, 0):t].mean()
        feat = lag_feats + [sin_dow, cos_dow, ma_feat]
        X.append(feat)
        y.append(df.loc[t, target_col])
        dates.append(df.loc[t, date_col])
        weight = math.exp(-decay_rate * (T_max - t))
        weights.append(weight)
    return np.array(X), np.array(y), np.array(weights), pd.to_datetime(dates)

def forecast_recursive(model, seed_window, seed_date, forecast_horizon, L, exog_
```

```
        """
        Given a fitted regression model (OLS or tree-based) and a seed window (most
        forecast recursively for forecast_horizon days.
        If exogenous features for future dates are provided in exog_df, they are use
        otherwise, cycle features are computed from the forecast date.

        NOTE for tree-based models: the predicted feature vector must have the same
        """
        forecasts = []
        current_window = list(seed_window)
        current_date = seed_date
        for h in range(forecast_horizon):
            next_date = current_date + pd.Timedelta(days=1)
            lag_feats = current_window[-L:][::-1]
            if exog_df is not None:
                cyc_vals = exog_df.loc[next_date]
                cyc_array = np.array([cyc_vals['sin_dow'], cyc_vals['cos_dow']])
            else:
                dow = next_date.dayofweek
                cyc_array = np.array([np.sin(2 * np.pi * dow / 7), np.cos(2 * np.pi
            ma_feat = np.mean(current_window[-3:]) if len(current_window) >= 3 else
            # Concatenate lag features, cycle features, and MA feature. Total length
            feat = np.concatenate([np.array(lag_feats), cyc_array, [ma_feat]])
            # For tree models, do not add constant.
            X_pred = feat.reshape(1, -1)
            y_hat = model.predict(X_pred)[0]
            forecasts.append(y_hat)
            current_window.append(y_hat)
            current_date = next_date
        return forecasts

    def rolling_forecast_evaluation(model, df, target_col, L, forecast_horizon, deca
        """
        Rolling-origin evaluation for the tree-based forecasting model.
        Returns MAPE (%) computed over all rolling origins.
        """
        errors = []
        n = len(df)
        for i in range(start_idx, n - forecast_horizon):
            if i - L < 0:
                continue
            seed_window = df[target_col].iloc[i-L:i].values
            seed_date = df.iloc[i-1]['DATE']
            preds = forecast_recursive(model, seed_window, seed_date, forecast_horiz
            pred_avg = np.mean(preds)
            actual_avg = df[target_col].iloc[i:i+forecast_horizon].mean()
            if actual_avg == 0:
                continue
            errors.append(abs((actual_avg - pred_avg) / actual_avg))
        return np.mean(errors)*100 if errors else None
```

```
# ======== Additional Helper Functions: AR/ARMA with OLS =========

def rolling_forecast_evaluation_ar_ols(series, exog, p, forecast_horizon, start_
    """
    Rolling-origin forecast evaluation for an AR(p) model using OLS.
    Predictors: constant, p lagged values, and exogenous cycle features.
    'series' is a pandas Series of the target.
    'exog' is a DataFrame (with same index as series) containing cycle features
    Returns MAPE (%) over all rolling origins.
    """
    errors = []
    n = len(series)
    for i in range(start_idx, n - forecast_horizon):
        X_train, y_train = [], []
        for t in range(p, i):
            lags = series.iloc[t-p:t].values    # shape: (p,)
            cyc = exog.iloc[t].values             # shape: (2,)
            X_train.append(np.concatenate([lags, cyc]))  # total length = p + 2
            y_train.append(series.iloc[t])
        if len(X_train) == 0:
            continue
        X_train = np.array(X_train)
        y_train = np.array(y_train)
        # Manually add constant column
        X_train_const = np.column_stack((np.ones(len(X_train)), X_train))
        model = sm.OLS(y_train, X_train_const).fit()
        seed_window = series.iloc[i-p:i].values.copy().tolist()
        current_index = i
        preds = []
        for k in range(forecast_horizon):
            cyc_forecast = exog.iloc[current_index].values  # shape: (2,)
            x_row = np.concatenate([np.array(seed_window[-p:]), cyc_forecast])
            X_pred = np.column_stack((np.ones(1), x_row.reshape(1, -1)))  # shap
            y_hat = model.predict(X_pred)[0]
            preds.append(y_hat)
            seed_window.append(y_hat)
            current_index += 1
            if current_index >= n:
                break
        if len(preds) < forecast_horizon:
            continue
        pred_avg = np.mean(preds)
        actual_avg = series.iloc[i:i+forecast_horizon].mean()
        if actual_avg == 0:
            continue
        errors.append(abs((actual_avg - pred_avg) / actual_avg))
    return np.mean(errors)*100 if errors else None

def rolling_forecast_evaluation_arma_ols(series, exog, forecast_horizon, start_i
```

```
    """
    Rolling-origin forecast evaluation for a simplified ARMA(1,1) model estimate
    Predictors: constant, lag1 (y[t-1]), lagged error (approx. y[t-1] - y[t-2]),
                and exogenous cycle features.
    Returns MAPE (%) over all rolling origins.
    """
    errors = []
    n = len(series)
    for i in range(max(start_idx, 2), n - forecast_horizon):
        X_train, y_train = [], []
        for t in range(2, i):
            lag1 = series.iloc[t-1]
            lag_err = series.iloc[t-1] - series.iloc[t-2]
            cyc = exog.iloc[t].values
            X_train.append(np.concatenate([[lag1, lag_err], cyc]))  # length = 2
            y_train.append(series.iloc[t])
        if len(X_train) == 0:
            continue
        X_train = np.array(X_train)
        y_train = np.array(y_train)
        X_train_const = np.column_stack((np.ones(len(X_train)), X_train))
        model = sm.OLS(y_train, X_train_const).fit()
        seed_y = series.iloc[i]
        seed_err = series.iloc[i] - series.iloc[i-1]
        preds = []
        current_index = i+1
        for k in range(forecast_horizon):
            cyc_forecast = exog.iloc[current_index].values
            x_row = np.concatenate([[seed_y, seed_err], cyc_forecast])  # length
            X_pred = np.column_stack((np.ones(1), x_row.reshape(1, -1)))   # sha
            y_hat = model.predict(X_pred)[0]
            preds.append(y_hat)
            new_err = y_hat - seed_y
            seed_y = y_hat
            seed_err = new_err
            current_index += 1
            if current_index >= n:
                break
        if len(preds) < forecast_horizon:
            continue
        pred_avg = np.mean(preds)
        actual_avg = series.iloc[i+1:i+1+forecast_horizon].mean()
        if actual_avg == 0:
            continue
        errors.append(abs((actual_avg - pred_avg) / actual_avg))
    return np.mean(errors)*100 if errors else None

# ======== Main Code: Data Preparation =========

# Read Electric Production data (expects columns: ['DATE','IPG2211A2N'])
```

```
df = pd.read_csv(r"C:\backupcgi\final_bak\Electric_Production_tm.csv")
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.sort_values(by='DATE').reset_index(drop=True)
df = add_cycle_features(df, date_col="DATE")
df['MA_3'] = add_ma_feature(df['IPG2211A2N'], window=3)

target_col = "IPG2211A2N"
date_col = "DATE"
decay_rate = 0.01
candidate_lags = [3, 5, 7, 9]
forecast_horizons = [3, 5, 7, 9]

# Split data into training and test sets (80% train, 20% test)
split_ratio = 0.8
split_idx = int(len(df) * split_ratio)
train_df = df.iloc[:split_idx].reset_index(drop=True)
test_df = df.iloc[split_idx:].reset_index(drop=True)

print("Total observations:", len(df))
print("Training observations:", len(train_df))
print("Test observations:", len(test_df))

# --- Tree-Based Forecasting (Using the above functions) ---
results_tree = {}
for horizon in forecast_horizons:
    best_mape = float('inf')
    best_L = None
    best_model = None
    print(f"\n[Tree Model] Forecast Horizon: {horizon} days")
    val_start_idx = int(0.8 * len(train_df))
    for L in candidate_lags:
        X_train, y_train, weights_train, _ = construct_lagged_dataset(train_df,
        tree_model = DecisionTreeRegressor(max_depth=5, random_state=42)
        tree_model.fit(X_train, y_train, sample_weight=weights_train)
        mape_val = rolling_forecast_evaluation(tree_model, train_df, target_col,
        if mape_val is None:
            continue
        print(f"  Candidate Lag L={L}: Validation MAPE = {mape_val:.2f}%")
        if mape_val < best_mape:
            best_mape = mape_val
            best_L = L
            best_model = tree_model
    print(f"--> Best Lag for horizon {horizon} days: L = {best_L} with Validatio
    X_train_full, y_train_full, weights_train_full, _ = construct_lagged_dataset
    final_tree_model = DecisionTreeRegressor(max_depth=5, random_state=42)
    final_tree_model.fit(X_train_full, y_train_full, sample_weight=weights_train

    # --------------------------
    # Added Output: Print Decision Tree Rules
    # --------------------------
```

```
        from sklearn.tree import export_text
        feature_names = [f"lag_{i}" for i in range(1, best_L+1)] + ["sin_dow", "cos_
        tree_rules = export_text(final_tree_model, feature_names=feature_names)
        print("Decision Tree Rules:")
        print(tree_rules)
        # --------------------------

        test_mape_tree = rolling_forecast_evaluation(final_tree_model, test_df, targ
        print(f"[Tree Model] Test MAPE for horizon {horizon} days: {test_mape_tree:.
        results_tree[horizon] = {'Best_Lag': best_L, 'Test_MAPE': test_mape_tree}

print("\nFinal Results for Tree-Based Model (Forecast Horizon: Best Lag, Test MA
for h in forecast_horizons:
    res = results_tree[h]
    print(f"  {h} days -> Best Lag: {res['Best_Lag']}, Test MAPE: {res['Test_MAP

# ======== Additional: OLS-based AR and ARMA Models =========

# Prepare test series and cycle exogenous regressors (sin_dow and cos_dow)
series_ar = test_df[target_col].reset_index(drop=True)
exog_ar = test_df[['sin_dow', 'cos_dow']].reset_index(drop=True)

print("\nEvaluating OLS-based AR and ARMA Models on Test Set:")

for horizon in forecast_horizons:
    # AR(3) replaces the previous AR(1) candidate; note start_idx is now 3.
    mape_ar3 = rolling_forecast_evaluation_ar_ols(series_ar, exog_ar, p=3, forec
    # AR(2) remains.
    mape_ar2 = rolling_forecast_evaluation_ar_ols(series_ar, exog_ar, p=2, forec
    # ARMA(1,1) remains.
    mape_arma11 = rolling_forecast_evaluation_arma_ols(series_ar, exog_ar, forec

    print(f"\nForecast Horizon {horizon} days:")
    print(f"  AR(3) Test MAPE:    {mape_ar3:.2f}%")
    print(f"  AR(2) Test MAPE:    {mape_ar2:.2f}%")
    print(f"  ARMA(1,1) Test MAPE:{mape_arma11:.2f}%")
```

Here are the results:

```
Total observations: 397
Training observations: 317
Test observations: 80
```

```
[Tree Model] Forecast Horizon: 3 days
  Candidate Lag L=3: Validation MAPE = 1.91%
  Candidate Lag L=5: Validation MAPE = 2.23%
  Candidate Lag L=7: Validation MAPE = 1.86%
  Candidate Lag L=9: Validation MAPE = 2.08%
--> Best Lag for horizon 3 days: L = 7 with Validation MAPE = 1.86%
Decision Tree Rules:
|--- lag_6 <= 94.84
|   |--- lag_6 <= 81.44
|   |   |--- lag_6 <= 71.13
|   |   |   |--- lag_6 <= 62.54
|   |   |   |   |--- lag_6 <= 58.05
|   |   |   |   |   |--- value: [57.81]
|   |   |   |   |--- lag_6 >  58.05
|   |   |   |   |   |--- value: [63.11]
|   |   |   |--- lag_6 >  62.54
|   |   |   |   |--- lag_1 <= 67.83
|   |   |   |   |   |--- value: [66.69]
|   |   |   |   |--- lag_1 >  67.83
|   |   |   |   |   |--- value: [71.22]
|   |   |--- lag_6 >  71.13
|   |   |   |--- lag_1 <= 71.79
|   |   |   |   |--- lag_6 <= 71.99
|   |   |   |   |   |--- value: [77.68]
|   |   |   |   |--- lag_6 >  71.99
|   |   |   |   |   |--- value: [70.03]
|   |   |   |--- lag_1 >  71.79
|   |   |   |   |--- lag_5 <= 71.16
|   |   |   |   |   |--- value: [75.19]
|   |   |   |   |--- lag_5 >  71.16
|   |   |   |   |   |--- value: [80.45]
|   |--- lag_6 >  81.44
|   |   |--- lag_6 <= 90.78
|   |   |   |--- lag_1 <= 78.91
|   |   |   |   |--- lag_1 <= 68.53
|   |   |   |   |   |--- value: [73.15]
|   |   |   |   |--- lag_1 >  68.53
|   |   |   |   |   |--- value: [78.83]
|   |   |   |--- lag_1 >  78.91
|   |   |   |   |--- lag_6 <= 83.21
|   |   |   |   |   |--- value: [85.04]
|   |   |   |   |--- lag_6 >  83.21
|   |   |   |   |   |--- value: [89.09]
|   |   |--- lag_6 >  90.78
|   |   |   |--- lag_1 <= 81.92
|   |   |   |   |--- lag_1 <= 78.01
|   |   |   |   |   |--- value: [82.53]
|   |   |   |   |--- lag_1 >  78.01
|   |   |   |   |   |--- value: [86.89]
|   |   |   |--- lag_1 >  81.92
```

```
|   |   |   |   |--- ma_feat <= 91.12
|   |   |   |   |   |--- value: [97.69]
|   |   |   |   |--- ma_feat >  91.12
|   |   |   |   |   |--- value: [91.92]
|--- lag_6 >  94.84
|   |--- lag_5 <= 94.43
|   |   |--- lag_2 <= 78.04
|   |   |   |--- lag_2 <= 75.26
|   |   |   |   |--- value: [88.45]
|   |   |   |--- lag_2 >  75.26
|   |   |   |   |--- value: [85.96]
|   |   |--- lag_2 >  78.04
|   |   |   |--- lag_5 <= 92.87
|   |   |   |   |--- lag_7 <= 108.67
|   |   |   |   |   |--- value: [96.73]
|   |   |   |   |--- lag_7 >  108.67
|   |   |   |   |   |--- value: [99.68]
|   |   |   |--- lag_5 >  92.87
|   |   |   |   |--- lag_4 <= 87.81
|   |   |   |   |   |--- value: [103.36]
|   |   |   |   |--- lag_4 >  87.81
|   |   |   |   |   |--- value: [99.17]
|   |--- lag_5 >  94.43
|   |   |--- lag_2 <= 88.39
|   |   |   |--- lag_1 <= 89.32
|   |   |   |   |--- sin_dow <= 0.22
|   |   |   |   |   |--- value: [90.78]
|   |   |   |   |--- sin_dow >  0.22
|   |   |   |   |   |--- value: [97.45]
|   |   |   |--- lag_1 >  89.32
|   |   |   |   |--- lag_1 <= 98.37
|   |   |   |   |   |--- value: [102.77]
|   |   |   |   |--- lag_1 >  98.37
|   |   |   |   |   |--- value: [108.63]
|   |   |--- lag_2 >  88.39
|   |   |   |--- lag_1 <= 105.10
|   |   |   |   |--- cos_dow <= 0.20
|   |   |   |   |   |--- value: [110.00]
|   |   |   |   |--- cos_dow >  0.20
|   |   |   |   |   |--- value: [105.12]
|   |   |   |--- lag_1 >  105.10
|   |   |   |   |--- lag_5 <= 103.64
|   |   |   |   |   |--- value: [109.77]
|   |   |   |   |--- lag_5 >  103.64
|   |   |   |   |   |--- value: [115.92]

[Tree Model] Test MAPE for horizon 3 days: 3.32%

[Tree Model] Forecast Horizon: 5 days
  Candidate Lag L=3: Validation MAPE = 2.31%
```

```
  Candidate Lag L=5: Validation MAPE = 1.92%
  Candidate Lag L=7: Validation MAPE = 1.65%
  Candidate Lag L=9: Validation MAPE = 2.35%
--> Best Lag for horizon 5 days: L = 7 with Validation MAPE = 1.65%
Decision Tree Rules:
|--- lag_6 <= 94.84
|   |--- lag_6 <= 81.44
|   |   |--- lag_6 <= 71.13
|   |   |   |--- lag_6 <= 62.54
|   |   |   |   |--- lag_6 <= 58.05
|   |   |   |   |   |--- value: [57.81]
|   |   |   |   |--- lag_6 >  58.05
|   |   |   |   |   |--- value: [63.11]
|   |   |   |--- lag_6 >  62.54
|   |   |   |   |--- lag_1 <= 67.83
|   |   |   |   |   |--- value: [66.69]
|   |   |   |   |--- lag_1 >  67.83
|   |   |   |   |   |--- value: [71.22]
|   |   |--- lag_6 >  71.13
|   |   |   |--- lag_1 <= 71.79
|   |   |   |   |--- lag_6 <= 71.99
|   |   |   |   |   |--- value: [77.68]
|   |   |   |   |--- lag_6 >  71.99
|   |   |   |   |   |--- value: [70.03]
|   |   |   |--- lag_1 >  71.79
|   |   |   |   |--- lag_5 <= 71.16
|   |   |   |   |   |--- value: [75.19]
|   |   |   |   |--- lag_5 >  71.16
|   |   |   |   |   |--- value: [80.45]
|   |--- lag_6 >  81.44
|   |   |--- lag_6 <= 90.78
|   |   |   |--- lag_1 <= 78.91
|   |   |   |   |--- lag_1 <= 68.53
|   |   |   |   |   |--- value: [73.15]
|   |   |   |   |--- lag_1 >  68.53
|   |   |   |   |   |--- value: [78.83]
|   |   |   |--- lag_1 >  78.91
|   |   |   |   |--- lag_6 <= 83.21
|   |   |   |   |   |--- value: [85.04]
|   |   |   |   |--- lag_6 >  83.21
|   |   |   |   |   |--- value: [89.09]
|   |   |--- lag_6 >  90.78
|   |   |   |--- lag_1 <= 81.92
|   |   |   |   |--- lag_1 <= 78.01
|   |   |   |   |   |--- value: [82.53]
|   |   |   |   |--- lag_1 >  78.01
|   |   |   |   |   |--- value: [86.89]
|   |   |   |--- lag_1 >  81.92
|   |   |   |   |--- ma_feat <= 91.12
|   |   |   |   |   |--- value: [97.69]
```

```
|   |   |   |   |--- ma_feat >  91.12
|   |   |   |   |   |--- value: [91.92]
|--- lag_6 >  94.84
|   |--- lag_5 <= 94.43
|   |   |--- lag_2 <= 78.04
|   |   |   |--- lag_2 <= 75.26
|   |   |   |   |--- value: [88.45]
|   |   |   |--- lag_2 >  75.26
|   |   |   |   |--- value: [85.96]
|   |   |--- lag_2 >  78.04
|   |   |   |--- lag_5 <= 92.87
|   |   |   |   |--- lag_7 <= 108.67
|   |   |   |   |   |--- value: [96.73]
|   |   |   |   |--- lag_7 >  108.67
|   |   |   |   |   |--- value: [99.68]
|   |   |   |--- lag_5 >  92.87
|   |   |   |   |--- lag_4 <= 87.81
|   |   |   |   |   |--- value: [103.36]
|   |   |   |   |--- lag_4 >  87.81
|   |   |   |   |   |--- value: [99.17]
|   |--- lag_5 >  94.43
|   |   |--- lag_2 <= 88.39
|   |   |   |--- lag_1 <= 89.32
|   |   |   |   |--- sin_dow <= 0.22
|   |   |   |   |   |--- value: [90.78]
|   |   |   |   |--- sin_dow >  0.22
|   |   |   |   |   |--- value: [97.45]
|   |   |   |--- lag_1 >  89.32
|   |   |   |   |--- lag_1 <= 98.37
|   |   |   |   |   |--- value: [102.77]
|   |   |   |   |--- lag_1 >  98.37
|   |   |   |   |   |--- value: [108.63]
|   |   |--- lag_2 >  88.39
|   |   |   |--- lag_1 <= 105.10
|   |   |   |   |--- cos_dow <= 0.20
|   |   |   |   |   |--- value: [110.00]
|   |   |   |   |--- cos_dow >  0.20
|   |   |   |   |   |--- value: [105.12]
|   |   |   |--- lag_1 >  105.10
|   |   |   |   |--- lag_5 <= 103.64
|   |   |   |   |   |--- value: [109.77]
|   |   |   |   |--- lag_5 >  103.64
|   |   |   |   |   |--- value: [115.92]

[Tree Model] Test MAPE for horizon 5 days: 2.74%

[Tree Model] Forecast Horizon: 7 days
  Candidate Lag L=3: Validation MAPE = 2.20%
  Candidate Lag L=5: Validation MAPE = 1.48%
  Candidate Lag L=7: Validation MAPE = 1.48%
```

```
  Candidate Lag L=9: Validation MAPE = 2.44%
--> Best Lag for horizon 7 days: L = 7 with Validation MAPE = 1.48%
Decision Tree Rules:
|--- lag_6 <= 94.84
|   |--- lag_6 <= 81.44
|   |   |--- lag_6 <= 71.13
|   |   |   |--- lag_6 <= 62.54
|   |   |   |   |--- lag_6 <= 58.05
|   |   |   |   |   |--- value: [57.81]
|   |   |   |   |--- lag_6 >  58.05
|   |   |   |   |   |--- value: [63.11]
|   |   |   |--- lag_6 >  62.54
|   |   |   |   |--- lag_1 <= 67.83
|   |   |   |   |   |--- value: [66.69]
|   |   |   |   |--- lag_1 >  67.83
|   |   |   |   |   |--- value: [71.22]
|   |   |--- lag_6 >  71.13
|   |   |   |--- lag_1 <= 71.79
|   |   |   |   |--- lag_6 <= 71.99
|   |   |   |   |   |--- value: [77.68]
|   |   |   |   |--- lag_6 >  71.99
|   |   |   |   |   |--- value: [70.03]
|   |   |   |--- lag_1 >  71.79
|   |   |   |   |--- lag_5 <= 71.16
|   |   |   |   |   |--- value: [75.19]
|   |   |   |   |--- lag_5 >  71.16
|   |   |   |   |   |--- value: [80.45]
|   |--- lag_6 >  81.44
|   |   |--- lag_6 <= 90.78
|   |   |   |--- lag_1 <= 78.91
|   |   |   |   |--- lag_1 <= 68.53
|   |   |   |   |   |--- value: [73.15]
|   |   |   |   |--- lag_1 >  68.53
|   |   |   |   |   |--- value: [78.83]
|   |   |   |--- lag_1 >  78.91
|   |   |   |   |--- lag_6 <= 83.21
|   |   |   |   |   |--- value: [85.04]
|   |   |   |   |--- lag_6 >  83.21
|   |   |   |   |   |--- value: [89.09]
|   |   |--- lag_6 >  90.78
|   |   |   |--- lag_1 <= 81.92
|   |   |   |   |--- lag_1 <= 78.01
|   |   |   |   |   |--- value: [82.53]
|   |   |   |   |--- lag_1 >  78.01
|   |   |   |   |   |--- value: [86.89]
|   |   |   |--- lag_1 >  81.92
|   |   |   |   |--- ma_feat <= 91.12
|   |   |   |   |   |--- value: [97.69]
|   |   |   |   |--- ma_feat >  91.12
|   |   |   |   |   |--- value: [91.92]
```

```
|--- lag_6 >  94.84
|   |--- lag_5 <= 94.43
|   |   |--- lag_2 <= 78.04
|   |   |   |--- lag_2 <= 75.26
|   |   |   |   |--- value: [88.45]
|   |   |   |--- lag_2 >  75.26
|   |   |   |   |--- value: [85.96]
|   |   |--- lag_2 >  78.04
|   |   |   |--- lag_5 <= 92.87
|   |   |   |   |--- lag_7 <= 108.67
|   |   |   |   |   |--- value: [96.73]
|   |   |   |   |--- lag_7 >  108.67
|   |   |   |   |   |--- value: [99.68]
|   |   |   |--- lag_5 >  92.87
|   |   |   |   |--- lag_4 <= 87.81
|   |   |   |   |   |--- value: [103.36]
|   |   |   |   |--- lag_4 >  87.81
|   |   |   |   |   |--- value: [99.17]
|   |--- lag_5 >  94.43
|   |   |--- lag_2 <= 88.39
|   |   |   |--- lag_1 <= 89.32
|   |   |   |   |--- sin_dow <= 0.22
|   |   |   |   |   |--- value: [90.78]
|   |   |   |   |--- sin_dow >  0.22
|   |   |   |   |   |--- value: [97.45]
|   |   |   |--- lag_1 >  89.32
|   |   |   |   |--- lag_1 <= 98.37
|   |   |   |   |   |--- value: [102.77]
|   |   |   |   |--- lag_1 >  98.37
|   |   |   |   |   |--- value: [108.63]
|   |   |--- lag_2 >  88.39
|   |   |   |--- lag_1 <= 105.10
|   |   |   |   |--- cos_dow <= 0.20
|   |   |   |   |   |--- value: [110.00]
|   |   |   |   |--- cos_dow >  0.20
|   |   |   |   |   |--- value: [105.12]
|   |   |   |--- lag_1 >  105.10
|   |   |   |   |--- lag_5 <= 103.64
|   |   |   |   |   |--- value: [109.77]
|   |   |   |   |--- lag_5 >  103.64
|   |   |   |   |   |--- value: [115.92]

[Tree Model] Test MAPE for horizon 7 days: 2.38%

[Tree Model] Forecast Horizon: 9 days
  Candidate Lag L=3: Validation MAPE = 1.88%
  Candidate Lag L=5: Validation MAPE = 1.36%
  Candidate Lag L=7: Validation MAPE = 1.45%
  Candidate Lag L=9: Validation MAPE = 2.50%
--> Best Lag for horizon 9 days: L = 5 with Validation MAPE = 1.36%
```

```
Decision Tree Rules:
|--- lag_1 <= 88.51
|   |--- lag_1 <= 79.32
|   |   |--- lag_5 <= 73.95
|   |   |   |--- lag_1 <= 68.06
|   |   |   |   |--- lag_5 <= 60.65
|   |   |   |   |   |--- value: [59.30]
|   |   |   |   |--- lag_5 >  60.65
|   |   |   |   |   |--- value: [64.95]
|   |   |   |--- lag_1 >  68.06
|   |   |   |   |--- lag_1 <= 76.35
|   |   |   |   |   |--- value: [69.65]
|   |   |   |   |--- lag_1 >  76.35
|   |   |   |   |   |--- value: [73.93]
|   |   |--- lag_5 >  73.95
|   |   |   |--- lag_1 <= 71.13
|   |   |   |   |--- lag_5 <= 74.18
|   |   |   |   |   |--- value: [84.19]
|   |   |   |   |--- lag_5 >  74.18
|   |   |   |   |   |--- value: [71.73]
|   |   |   |--- lag_1 >  71.13
|   |   |   |   |--- lag_2 <= 80.23
|   |   |   |   |   |--- value: [81.38]
|   |   |   |   |--- lag_2 >  80.23
|   |   |   |   |   |--- value: [76.05]
|   |--- lag_1 >  79.32
|   |   |--- lag_5 <= 82.78
|   |   |   |--- lag_2 <= 78.90
|   |   |   |   |--- lag_4 <= 71.71
|   |   |   |   |   |--- value: [80.53]
|   |   |   |   |--- lag_4 >  71.71
|   |   |   |   |   |--- value: [90.16]
|   |   |   |--- lag_2 >  78.90
|   |   |   |   |--- lag_1 <= 87.19
|   |   |   |   |   |--- value: [76.66]
|   |   |   |   |--- lag_1 >  87.19
|   |   |   |   |   |--- value: [82.24]
|   |   |--- lag_5 >  82.78
|   |   |   |--- lag_5 <= 110.92
|   |   |   |   |--- lag_2 <= 86.51
|   |   |   |   |   |--- value: [92.01]
|   |   |   |   |--- lag_2 >  86.51
|   |   |   |   |   |--- value: [87.06]
|   |   |   |--- lag_5 >  110.92
|   |   |   |   |--- lag_3 <= 98.40
|   |   |   |   |   |--- value: [97.24]
|   |   |   |   |--- lag_3 >  98.40
|   |   |   |   |   |--- value: [99.51]
|--- lag_1 >  88.51
|   |--- lag_5 <= 93.97
```

```
|   |   |--- lag_1 <= 100.98
|   |   |   |--- lag_2 <= 91.11
|   |   |   |   |--- lag_4 <= 75.72
|   |   |   |   |   |--- value: [87.30]
|   |   |   |   |--- lag_4 >  75.72
|   |   |   |   |   |--- value: [98.02]
|   |   |   |--- lag_2 >  91.11
|   |   |   |   |--- lag_1 <= 94.67
|   |   |   |   |   |--- value: [85.84]
|   |   |   |   |--- lag_1 >  94.67
|   |   |   |   |   |--- value: [89.37]
|   |   |--- lag_1 >  100.98
|   |   |   |--- lag_1 <= 106.77
|   |   |   |   |--- sin_dow <= -0.88
|   |   |   |   |   |--- value: [101.63]
|   |   |   |   |--- sin_dow >  -0.88
|   |   |   |   |   |--- value: [95.37]
|   |   |   |--- lag_1 >  106.77
|   |   |   |   |--- lag_5 <= 91.95
|   |   |   |   |   |--- value: [98.83]
|   |   |   |   |--- lag_5 >  91.95
|   |   |   |   |   |--- value: [102.92]
|   |--- lag_5 >  93.97
|   |   |--- lag_3 <= 103.64
|   |   |   |--- lag_2 <= 87.79
|   |   |   |   |--- ma_feat <= 90.39
|   |   |   |   |   |--- value: [100.24]
|   |   |   |   |--- ma_feat >  90.39
|   |   |   |   |   |--- value: [104.40]
|   |   |   |--- lag_2 >  87.79
|   |   |   |   |--- lag_1 <= 102.34
|   |   |   |   |   |--- value: [107.18]
|   |   |   |   |--- lag_1 >  102.34
|   |   |   |   |   |--- value: [110.88]
|   |   |--- lag_3 >  103.64
|   |   |   |--- lag_3 <= 107.47
|   |   |   |   |--- lag_2 <= 96.99
|   |   |   |   |   |--- value: [89.36]
|   |   |   |   |--- lag_2 >  96.99
|   |   |   |   |   |--- value: [90.07]
|   |   |   |--- lag_3 >  107.47
|   |   |   |   |--- lag_1 <= 91.00
|   |   |   |   |   |--- value: [91.89]
|   |   |   |   |--- lag_1 >  91.00
|   |   |   |   |   |--- value: [93.86]

[Tree Model] Test MAPE for horizon 9 days: 2.25%

Final Results for Tree-Based Model (Forecast Horizon: Best Lag, Test MAPE %):
  3 days -> Best Lag: 7, Test MAPE: 3.32%
```

```
  5 days -> Best Lag: 7, Test MAPE: 2.74%
  7 days -> Best Lag: 7, Test MAPE: 2.38%
  9 days -> Best Lag: 5, Test MAPE: 2.25%

Evaluating OLS-based AR and ARMA Models on Test Set:

Forecast Horizon 3 days:
  AR(3) Test MAPE:    3.90%
  AR(2) Test MAPE:    5.02%
  ARMA(1,1) Test MAPE:4.35%

Forecast Horizon 5 days:
  AR(3) Test MAPE:    2.85%
  AR(2) Test MAPE:    4.56%
  ARMA(1,1) Test MAPE:3.92%

Forecast Horizon 7 days:
  AR(3) Test MAPE:    2.70%
  AR(2) Test MAPE:    4.85%
  ARMA(1,1) Test MAPE:4.23%

Forecast Horizon 9 days:
  AR(3) Test MAPE:    2.96%
  AR(2) Test MAPE:    5.42%
  ARMA(1,1) Test MAPE:4.76%
```

Summery of Output of Weighted Tree for Time Series Forecasting Model

The weighted decision tree model consistently delivers competitive accuracy across all forecast horizons. For instance, at a 3-day horizon, it achieves a Test *MAPE* of *3.32%*, and it improves further as the horizon extends to 9 days, culminating in a MAPE of *2.25%*. This outperforms simpler *AR(2)* and *ARMA(1,1)* approaches and often edges out *AR(3)* as well.

By incorporating a decay factor, moving averages, and cycle features (like day-of-week *sines/cosines*), the tree captures both immediate shifts and seasonal fluctuations. Its splits rely heavily on recent lagged values — reflected in rules that frequently compare "*lag_6*" or "*lag_1*" to thresholds — showing how fresh data steers the forecast. Unlike conventional linear

models, this tree pinpoints nuanced thresholds, letting it adapt quickly when patterns change.

This structure also helps decision makers. They can read each "if–then" branch to see exactly what triggers a higher or lower forecast, such as "*lag_6 > 90" or "cos_dow > 0.20*". By exposing how the forecast is formed, the model's recommendations become more transparent and easier to trust. Such clarity supports operational planning, where stakeholders can align staffing or resource allocation decisions with identifiable changes in the lag or seasonal signals.

Final Thoughts

I tackled the time series forecasting problem using a decision tree approach enhanced by decay weighting, ensuring recent data drives the model more strongly. Unlike standard *ARMA* frameworks, our weighted tree method directly incorporates recency, capturing short-term shifts without discarding older trends entirely.

Next, I tested and confirmed its predictive efficiency across different horizons. The ability to print out decision tree rules — or visualize them — gives practitioners immediate insight into which factors matter most (like specific lags or cyclical indicators). This transparency empowers decision makers to adapt resources or strategies to real-time changes.

Finally, this is only the foundation. Because it's a flexible framework, we can expand it in multiple ways: consider boosted trees for even stronger performance, or tackle multivariate forecasting by adding more variables

into the mix. There's real potential to keep refining and scaling this method, all while maintaining interpretability and agility for real-world applications.

The code and dataset can be accessed in the following GitHub repository: https://github.com/datalev001/tree_TSM/tree/main.

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

Time Series Analysis | Decision Tree | Forecasting | Python | Machine Learning

Responses (1)

Alex Mylnikov

What are your thoughts?

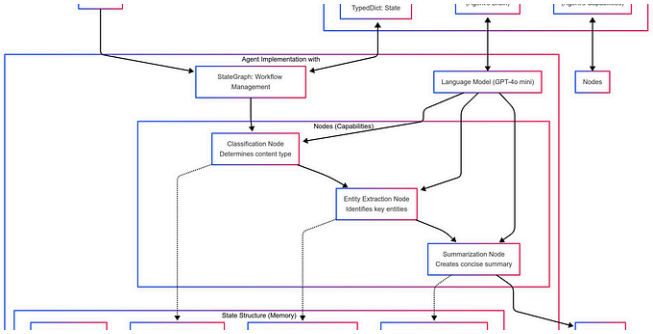
Idan Mandelbaum
19 hours ago

Awesome as always! Thank you!

Reply

More from Shenggang Li and Data Science Collective

In Towards AI by Shenggang Li



In Data Science Collective by Paolo Perrone

Reinforcement Learning-Enhanced Gradient Boosting Machines

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

Apr 1 287 3

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 58

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...

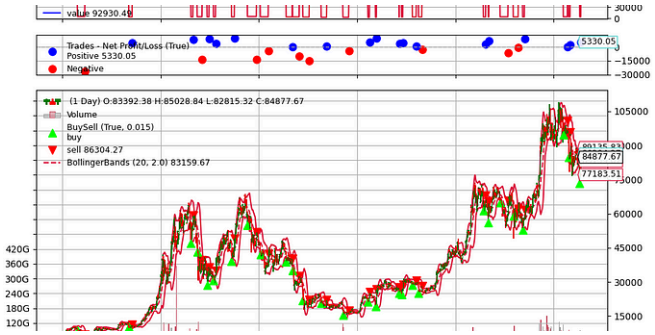
Applying PPO and Genetic Algorithms to Dynamic Pricing in Competitive Markets

Mar 17 188 3

See all from Shenggang Li

See all from Data Science Collective

Recommended from Medium



Ali AZARY

Bollinger Bands Mean-Reversion with ADX and RSI: Transforming...

(You can read this article and more on my website: https://www.aliazary.com/)

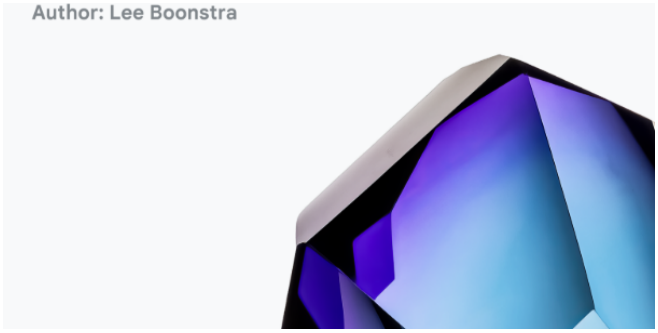
5d ago 33 1

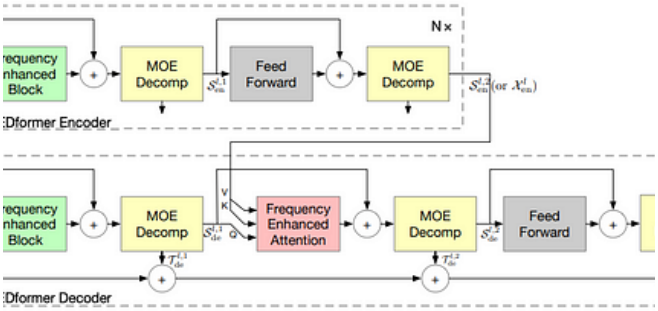
In TechToFreedom by Yang Zhou

11 Outdated Python Modules That You Should Never Use Again

And their modern alternatives that you should master

6d ago 728 18





Algo Insights

Google's 69-Page Prompt Engineering Masterclass: What's...

I've been writing about tech for three years, and let me tell you, nothing's grabbed me...

4d ago 22 1

Dong-Keon Kim

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

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

4d ago 18

Valeriy Manokhin, PhD, MBA, CQF

Predicting Full Probability Distributions with Conformal...

Introduction

Mar 28 293 1

Graham Giller

Can we Achieve Quant Fusion?

Is there a way to Introduce Technical Rules into a Rigorous Quantitative Framework for...

Mar 24 6 2

See more recommendations