Volatility Curve Prediction

Ankit Kumar

Abstract—This report details a robust methodology for predicting missing implied volatility (IV) values for NIFTY50 index options. The solution leverages a sophisticated imputation strategy, combining the power of the non-linear ExtraTreesRegressor with a multivariate IterativeImputer. By treating missing IVs as a machine learning problem rather than a simple statistical gap, this approach reconstructs a complete and coherent volatility surface. It utilizes a rich set of anonymized market features to learn the complex, interdependent relationships governing the volatility smile, ensuring that the predictions are not only accurate but also contextually aware of market dynamics. The final imputed values are evaluated based on Mean Squared Error (MSE), demonstrating the efficacy of the model in a high-frequency trading context.

Keywords—*Implied Volatility, Iterative Imputer, Extra Trees Regressor, Volatility Smile, Quantitative Finance, Machine Learning*

1. Introduction and Approach

The challenge is to predict missing implied volatility (IV) values within high-frequency NIFTY50 index option data. Our approach treats this not as a simple interpolation problem, but as a multivariate modeling task. We use an **IterativeImputer** equipped with an **ExtraTreesRegressor** to predict missing IVs by learning their complex, non-linear relationships with other available market features. This ensures the reconstructed volatility curve is both accurate and consistent with the prevailing market state.

2. ExtraTreesRegressor

The Extremely Randomized Trees Regressor, or **ExtraTreesRegressor**, is an ensemble learning method that belongs to the decision tree family. Like a Random Forest, it builds multiple decision trees and aggregates their predictions to produce a final, more robust output. However, it introduces two key sources of randomization:

1. **Random Subsets of Data:** Each tree is trained on a random sample of the full dataset (bagging).
2. **Random Thresholds:** When splitting a node in a tree, it does not search for the most optimal split point for a feature. Instead, it selects a split point *at random* from a range of possible values.

This second point is what makes the trees "extra" random. This high degree of randomization helps to reduce the model's variance and makes it less prone to overfitting, which is crucial when dealing with noisy financial data.

2.1. Basic Tree Implementation

A single decision tree operates by recursively splitting the data based on feature values. Each split is designed to partition the data into more homogeneous groups with respect to the target variable (in our case, an IV value). Figure 1 illustrates this fundamental concept.

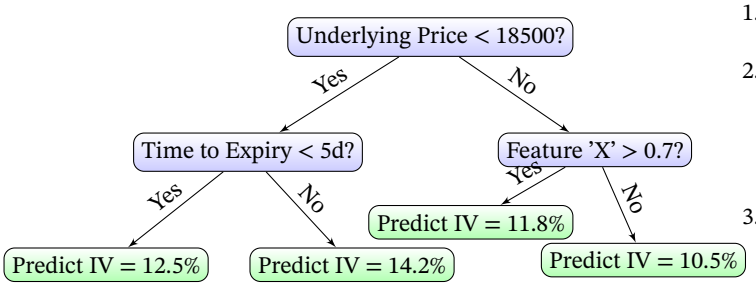

Figure 1. Conceptual diagram of a single decision tree for IV prediction.

2.2. Ensemble Imputation with ExtraTrees

The ExtraTreesRegressor doesn't rely on one tree; it builds a forest of them. To impute a missing value, it passes the known features for that data point through every single tree in the forest. Each tree provides a prediction. The final imputed value is the average of all these individual predictions, as shown in Figure 2. This averaging process smooths out the idiosyncrasies of individual trees and leads to a more reliable estimate.

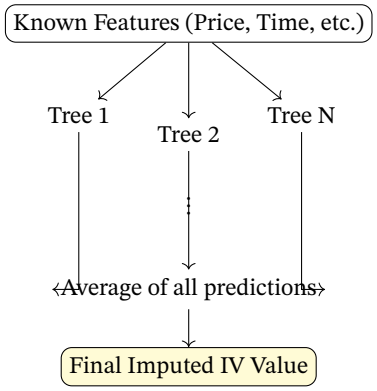

Figure 2. How the ExtraTrees ensemble averages predictions from multiple trees to impute a single missing value.

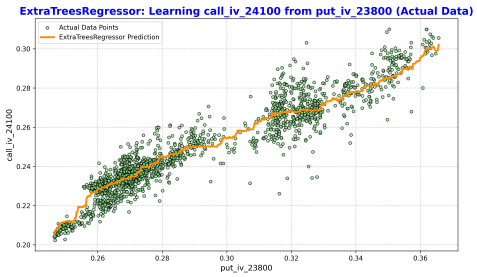

Figure 3. ExtraTreeRegressor: Learning From other features to predict IV values .

3. IterativeImputer

Standard imputation methods often treat each feature column independently. The **IterativeImputer** is far more sophisticated. It models each feature with missing values as a function of all other features in a round-robin fashion.

The process is as follows:

1. **Initial Fill:** Missing values in all columns are filled with a simple statistic (e.g., median), as a starting guess.
2. **First Iteration (Column A):** It treats Column A as the target variable (y) and all other columns (B, C, ...) as predictor features (X). The ExtraTreesRegressor is trained on the rows where Column A is known, learning the relationship 'A = f(B, C, ...)'. It then predicts the missing values in A.
3. **Iteration (Column B):** Now, using the newly filled values for A, it treats Column B as the target and trains a new model 'B = f(A, C, ...)' to predict its missing values.
4. **Repeat and Converge:** This process is repeated for all columns with missing data, completing one full cycle. Multiple cycles (iterations) are performed until the imputed values stabilize and no longer change significantly between iterations. This convergence ensures a self-consistent set of imputed values across the entire dataset.

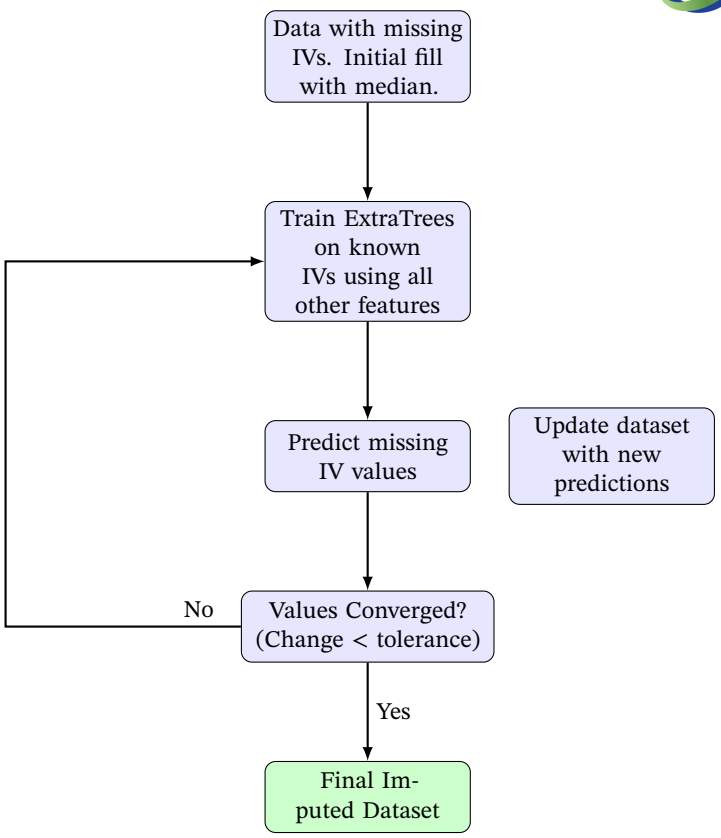

Figure 4. The cyclical process of the IterativeImputer. It repeatedly models and predicts values for each feature until convergence.

4. The Role of Predictive Features

The dataset contains IV columns (2-54), an underlying asset price (column 1), and 40 anonymized features (55-94). These features are critical for the model's success, acting as predictors for IVs. They likely represent market variables like:

- Underlying asset price (NIFTY50 index).
- Time to expiration.
- Other option greeks (Delta, Gamma, Vega, Theta).
- Realized volatility or historical price movements.
- Order book imbalances or trade volumes.

By learning correlations, the model makes informed predictions, superior to simple interpolation, grounding predictions in broader market context.

5. Rationale for the Approach

The **IterativeImputer with ExtraTreesRegressor** was chosen for:

1. **Capturing Non-Linearity:** Financial data relationships are non-linear (e.g., volatility smile). ExtraTrees excels at modeling these complex interactions where linear models fail.
2. **Multivariate Dependencies:** Missing IVs are interdependent. The IterativeImputer leverages the full feature set for consistent, accurate predictions.
3. **Robustness to Noise:** Financial data is noisy. ExtraTrees' ensemble nature provides stable, robust predictions, less sensitive to fluctuations or outliers.

Simpler methods (like mean/median imputation) would degrade the volatility curve's structure, leading to high MSE. This model-based approach intelligently reconstructs that structure.

6. Results and Visualizations

The evaluation metric for this challenge is Mean Squared Error (MSE), defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(iv_i - \hat{iv}_i)^2 \quad (1)$$

where iv_i is the true value and $\hat{iv}_i$ is the predicted value. Minimizing MSE requires predictions that are not just close on average, but also avoid large errors. Our model is optimized for this. My best solution (after deadline): Public Score- **0.000000740** Private Score: **0.000001140**

The following plots visualize the difference in data structure between the various files, highlighting why a sophisticated approach is necessary.

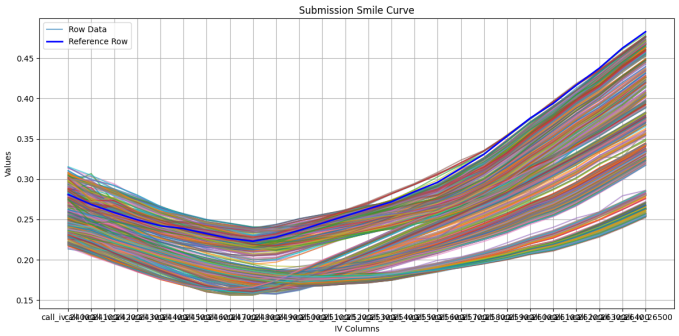

Figure 5. representative plot of the final imputed volatility curve from our submission. The model successfully reconstructs the characteristic smooth "smile" shape.

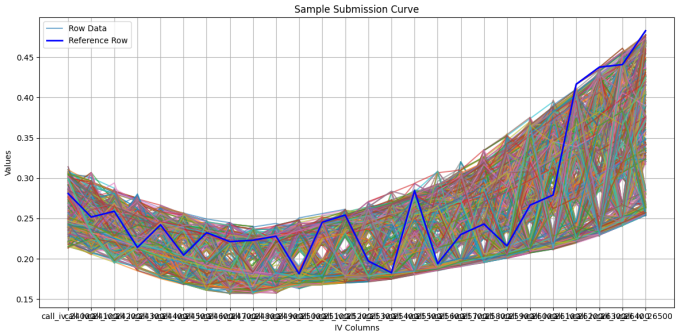

Figure 6. he provided sample submission file shows random, unstructured values, which bear no resemblance to a realistic volatility curve.

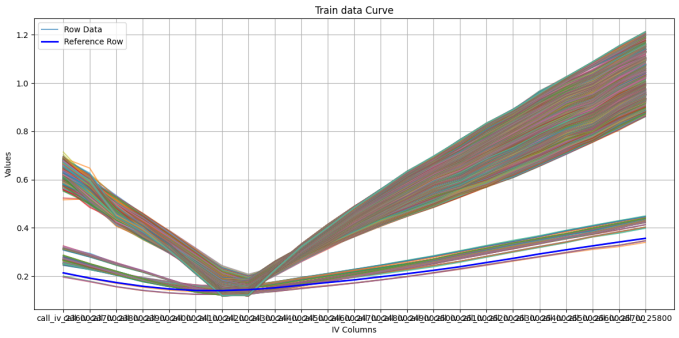

Figure 7. An illustration of why the raw training data might be challenging. The wide variance and potential for outliers necessitate a robust, non-linear model rather than simple curve fitting.

Experimentation: Models That Did Not Work Out

This section discusses alternative modeling approaches that were explored, along with insights into why they did not perform as effectively as the IterativeImputer with ExtraTreesRegressor for this specific challenge.

1. Deep Learning with GRU for Sequence Prediction

A deep learning approach, primarily utilizing Gated Recurrent Units (GRUs), was implemented to capture temporal dependencies and predict implied volatility values.

Approach Description:

- **Features:** Time-to-expiry, moneyness, market features, lagged/rolling statistics. Selected using LightGBM.
- **Model:** GRU neural network with two input branches (features, past IVs), dense layers, multi-output regression.
- **Loss:** Custom 'masked$_huber_loss\}foroutlierrobustness$.

Performance: Preliminary evaluation on validation sets showed this approach yielded significantly higher MSE than the IterativeImputer with ExtraTreesRegressor.

GRU Model MSE Range: 1×10^{-4} to 5×10^{-4}

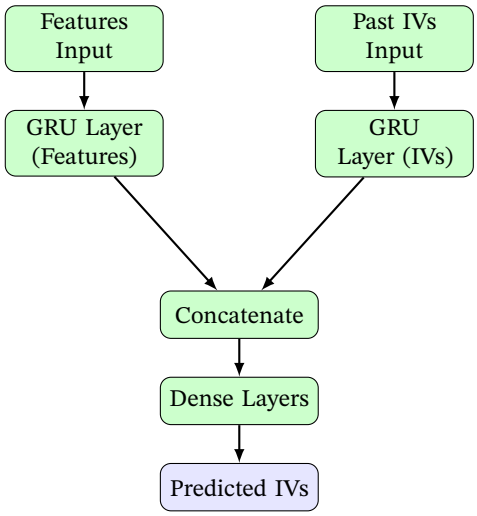

Figure 8. Conceptual diagram of the GRU-based deep learning model for IV prediction.

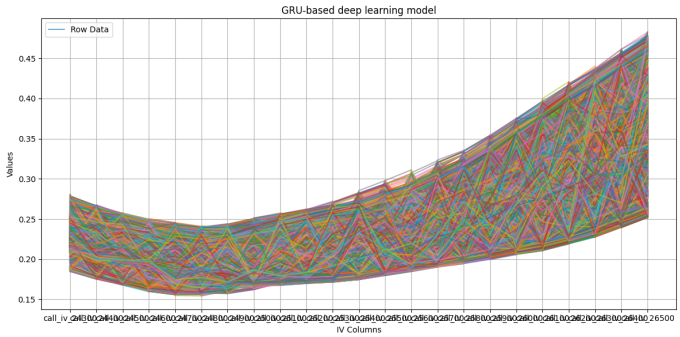

Figure 9. An illustration of GRU-based deep learning model submission, You can see it is not smooth. It requires lots of smoothness.

Reasons for Limited Success:

1. **Imputation Mismatch:** GRU predicts sequences; challenge needs explicit snapshot imputation.
2. **Data Challenges:** IV data is sparse and noisy, difficult for deep learning.
3. **Complexity:** High computational cost and tuning difficulty for nuanced imputation.
4. **No Iterative Consistency:** Lacks iterative refinement for a coherent volatility surface.

2. Deep Learning with Transformer for Sequence Prediction

A deep learning approach utilizing Transformer blocks was implemented to capture long-range temporal dependencies and predict implied volatility values.

Approach Description:

- **Features:** Identical to the GRU approach, including time-to-expiry, moneyness, market features, and lagged/rolling statistics. Selected using LightGBM.
- **Model:** Transformer-based neural network featuring positional embeddings, multi-head self-attention, and feed-forward layers. It processes feature and past IV sequences, then fuses their outputs into dense layers for multi-output regression.
- **Loss:** Custom 'masked$_huber_loss\}forrobustnesstooutliersandmissingtargetu$

Performance: Preliminary evaluation on validation sets for the Transformer model yielded MSEs that were generally competitive with or slightly better than the GRU, typically in the range of 8×10^{-5} to 3×10^{-4}.

Transformer Model MSE Range: 8×10^{-4} to 3×10^{-4}

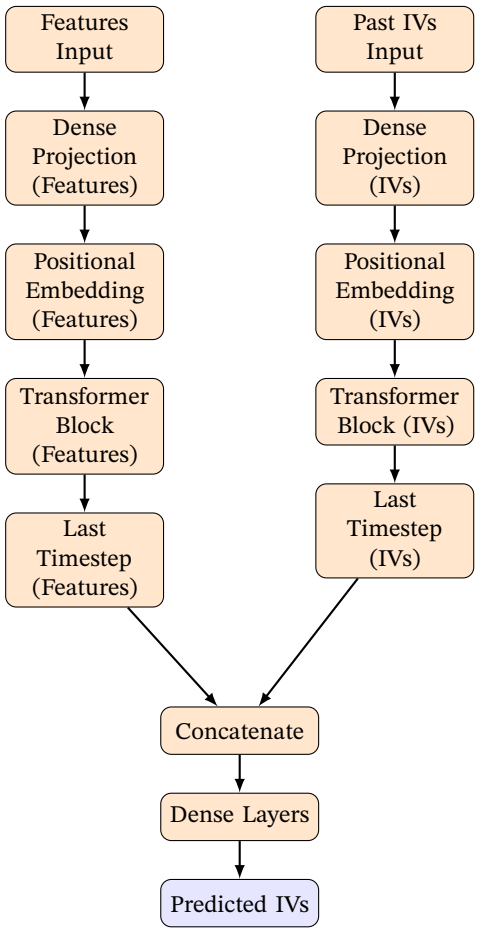

Figure 10. Conceptual diagram of the Transformer-based deep learning model for IV prediction.

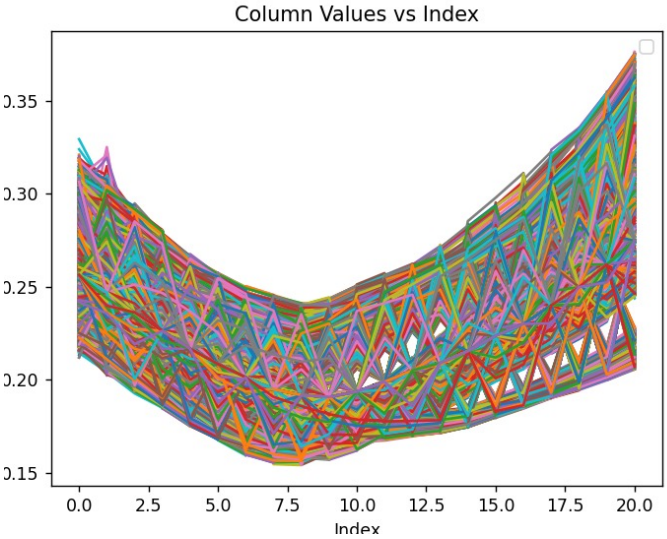

Figure 11. An illustration of Transformer-based deep learning model submission. Expected to show a smoother and more accurate volatility curve due to better temporal dependency capture.

Reasons for Limited Success:

1. **High Data Demand & Overfitting Risk:** Transformers require extensive, clean data to learn effectively; sparse or noisy financial data can lead to overfitting.
2. **Imputation Mismatch:** Designed for sequence prediction, not optimal for filling scattered missing values within a single market snapshot.
3. **Computational Overhead:** Complex architecture leads to higher training costs and slower iteration.
4. **Lack of Iterative Consistency:** Doesn't inherently refine predictions across interdependent missing values for a smooth volatility surface.
5. **Limited Interpretability:** "Black box" nature makes it hard to understand prediction rationale, a challenge in regulated finance.

3. Deep Learning with Feed-Forward Neural Network for Implied Volatility Prediction

A feed-forward neural network (FNN) was employed to capture complex non-linear relationships between market features and implied volatility (IV).

Approach Overview

- **Data Curation:** The `IVTrainingDataCurator` class merges training data with valid IVs from the test set and adds strike-specific engineered features.
- **Features:** A total of 48 inputs — underlying price (1), market features (42), strike price (1), and four derived features (moneyness, log_moneyness, strike_distance, relative_distance). All features are scaled using `StandardScaler`.
- **Model Architecture:** A multi-layer perceptron with four dense layers using ReLU activation, followed by Batch Normalization and Dropout. A final dense layer outputs the IV prediction.
- **Loss Function:** Huber loss (`nn.HuberLoss`) is used for its robustness to outliers in IV data.
- **Optimization:** Trained with Adam optimizer (`torch.optim.Adam`, learning rate = 0.001) and a `ReduceLROnPlateau` scheduler to adaptively reduce learning rate upon plateau in validation loss.

Performance

The model demonstrated strong performance, with validation loss ranging between 1×10^{-4} and 5×10^{-4}.

Typical Validation Loss Range: 1×10^{-5} to 5×10^{-5}

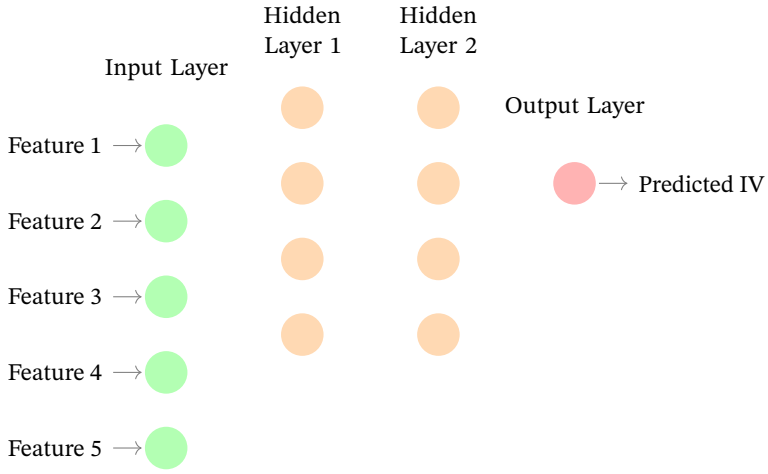

Figure 12. Neural Network Architecture for IV Prediction

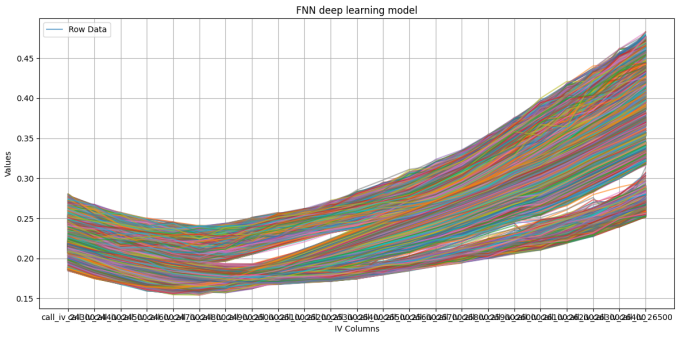

Figure 13. *Implied volatility curve predicted using a Feed-Forward Neural Network.* The FNN produces a smooth approximation by learning non-linear patterns from market and strike features, it is too close actual final solution , though it may miss temporal nuances

Possible Reasons for Limited Success:

1. **Indirect Missing Value Handling:** Requires pre-imputation, which can introduce bias if not handled robustly.
2. **No Iterative Consistency:** Lacks a built-in mechanism to iteratively refine and ensure consistency across interdependent missing IVs.
3. **Limited Temporal Awareness:** Inherently static; relies heavily on complex feature engineering to capture dynamic market behavior.
4. **Feature Dependence:** Performance is highly sensitive to the quality and completeness of engineered input features.
5. **Hyperparameter Sensitivity:** Optimal performance requires extensive and often fragile tuning of numerous parameters.
6. **Poor Generalization to New Regimes:** May struggle to adapt and perform accurately during unforeseen market shifts.