My Update - Smart Applications

DAVIDE MARCHI

January 2025

1 Introduction

During the group project, I was part of Topic 7 and was responsible for the explainability aspects of the system. My contributions included developing explainability solutions for both the retrieval-augmented generation (RAG) system and the forecasting models. Building on this foundation, during my updates to the **forecasting system**, I focused on two key aspects: **improving explainability** and **enhancing performance**. The original implementation, while functional, left room for improvement in both areas.

The system required users to wait several seconds to obtain predictions, which significantly impacted user experience, especially in the web-based environment where additional overheads further slowed the process. Additionally, the system lacked global explanations for the model's behavior, providing only local explanations for individual predictions. This meant users could only understand the factors influencing a specific prediction without gaining insights into the overall importance of features across the dataset.

I aimed to address these limitations. First, I wanted to introduce global explainability to provide users with a clear understanding of what factors were most influential across all predictions, not just a specific one. Second, I sought to reduce the response time for generating predictions and their explanations to make the system more responsive.

The updates detailed in this report showcase the work undertaken to achieve these goals. I added a global explainability feature using SHAP values, visualized in a new graph integrated into the GUI. I also transitioned from LIME to SHAP for local explainability, leveraging its efficiency and suitability for tree-based models like XGBoost. These improvements significantly enhanced the system's usability, interpretability, and reduced computation times. The updated codebase, including all implemented changes, is available in my GitHub repository[1].

2 Before the Updates

Before implementing my updates, the forecasting system relied on a combination of LIME for local explainability and bootstrapping for calculating uncertainty bounds. While these techniques were effective, they introduced performance challenges that hindered the user experience.

2.1 Local Explainability with LIME

Local explainability was implemented using LIME[2] (Local Interpretable Model-Agnostic Explanations), a widely-used tool for interpreting individual predictions. LIME works by perturbing the input data, making numerous predictions with slight variations, and analyzing the resulting changes in the output. This approach provides insights into the contribution of specific input features to a given prediction.

LIME was chosen during the initial development because of its model-agnostic nature and ease of use. However, this method's reliance on repeated perturbations and predictions significantly impacted performance. For example, generating explanations for 30 predictions required many perturbation-based computations, leading to noticeable delays.

2.2 Bounds with Bootstrapping

Uncertainty bounds were calculated using bootstrapping, another model-agnostic method. This technique perturbs the input data based on the standard deviation of the dataset, simulating variations that may occur in real-world scenarios. The model then generates predictions for each perturbed input, and the resulting values are used to calculate the bounds with a specified confidence level (e.g., 95%).

2.3 Performance Measurements

The combined use of bootstrapping and LIME led to significant delays, especially in the web-based environment where additional overheads (e.g., database queries, API calls, GUI processing) further slowed the system. The table below summarizes the performance measurements before my updates:

Environment	**Sequence Length**	**Predictions**	**Time (seconds)**
Local Machine	15	30	2.93
Website (with GUI)	15	30	4.60

Table 1: Comparison of prediction times in different environments.

Performance measurements were conducted 10 times, and the averages were calculated to ensure robustness and reliability of the results. The tests were run on an AMD Ryzen 7 4800H CPU with a clock speed of 2.90 GHz.

2.4 Limitations

The performance issues stemming from LIME and bootstrapping were a significant drawback, particularly in time-sensitive applications. Additionally, the system lacked a global explainability feature, which meant that users could only understand the factors influencing a specific prediction. This limited their ability to gain insights into the overall behavior of the forecasting model across the entire dataset.

3 Update: Global Explainability

One of the major enhancements in my updates was the introduction of global explainability to the forecasting system. This feature provides insights into the overall importance of features across the entire dataset, addressing the limitation of only having local explanations for individual predictions. By visualizing the global behavior of the model, users can better understand the factors that consistently influence predictions.

3.1 Transition to SHAP for Global Explainability

The new implementation leverages SHAP[3] (SHapley Additive exPlanations) values, which are specifically designed for global and local interpretability. Unlike LIME, which focuses on local explanations through perturbation, SHAP provides consistent, model-agnostic measures of feature importance while also supporting model-specific optimizations.

For the forecasting model, which uses XGBoost for regression, I utilized the `shap.TreeExplainer` [4]. This model-specific explainer is optimized for tree-based models like XGBoost and enables faster and more detailed analysis compared to model-agnostic alternatives. Using this tool, I computed SHAP values for the entire dataset, allowing for the generation of a global feature importance graph.

The graph, visualized as a bee swarm plot in the GUI, highlights how different features (such as specific days in the sequence) contribute to the model's predictions. This plot provides users with a comprehensive view of the model's behavior, making it easier to identify trends and influential features.

3.2 Codebase Changes Across Topics

Introducing global explainability required modifications to ensure SHAP values were generated, stored, and communicated effectively across the system. These updates were seamlessly integrated into existing workflows without introducing additional API calls or computation steps.

3.2.1 Topic 7 Codebase Changes

In the Topic 7 codebase, I updated the `ForecastExplainer` to compute SHAP values for both individual predictions and the entire dataset. These global SHAP values are now returned alongside the predictions and local explanations, eliminating the need for additional methods or separate calls.

Key changes included:

- Integrating `shap.TreeExplainer` to calculate SHAP values for the dataset.
- Ensuring global SHAP values are included in the output structure alongside predictions and local explanations.
- Maintaining compatibility with model-agnostic approaches to ensure flexibility for future models.

3.2.2 Topic 3 Codebase Changes

The Topic 3 codebase, responsible for data processing, did not require additional API calls or logic changes. The global SHAP values were included in the existing response structure from the `ForecastExplainer`, making integration straightforward.

Key changes:

- Updating the handling of returned data to process predictions, local explanations, and global SHAP values as part of a unified response.

3.2.3 Topic 4 Codebase Changes

In the Topic 4 codebase, which manages API interactions, I updated the classes used to interface between the GUI and Topic 3. This ensured that the global SHAP values could be transmitted to the GUI without altering the existing API structure.

Key updates:

- Modifying the interface classes to accommodate global SHAP values in the response.
- Ensuring the updated classes remained compatible with the existing API endpoints and workflows.

3.2.4 Topic 6 Codebase Changes

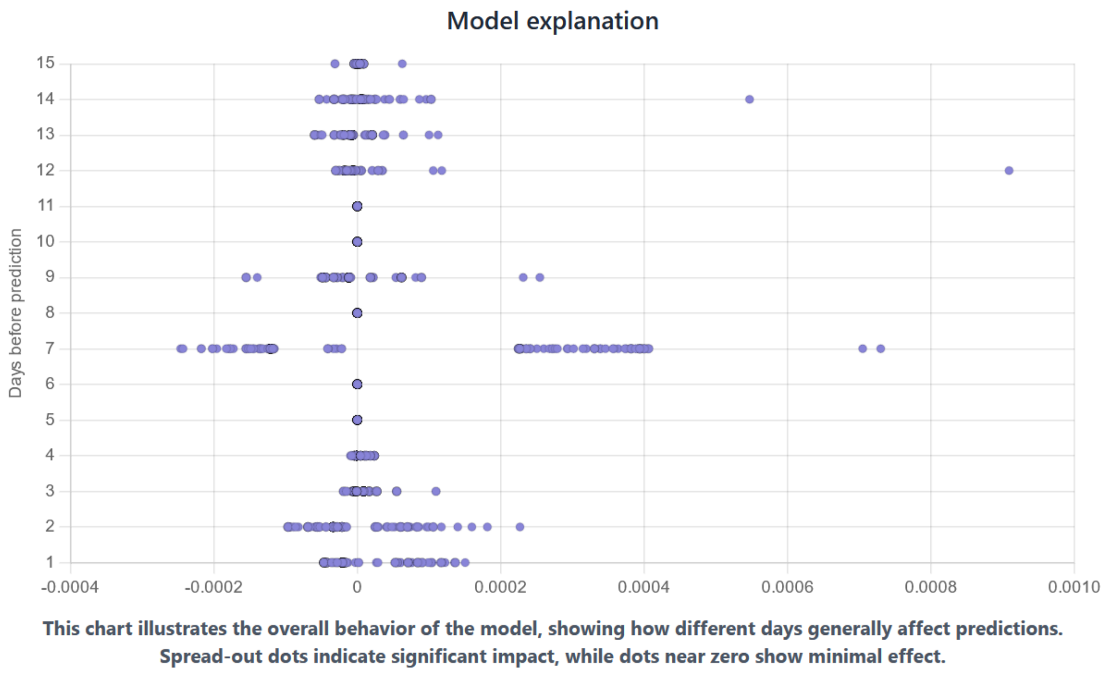
In the Topic 6 codebase, which manages the GUI, I introduced a new section to display global explainability data. This included creating a bee swarm plot to visualize SHAP values and updating the user interface to provide clear instructions for interpreting the graph.

Key changes:

- Adding functionality to process API responses and extract global SHAP data.
- Implementing the bee swarm plot visualization using a suitable plotting library.
- Ensuring the GUI provided users with an intuitive way to access and understand global explainability.

3.3 Final Output

The addition of global explainability has transformed the forecasting system, providing users with a deeper understanding of the model's behavior. The bee swarm plot, now available in the GUI, highlights the relative importance of features, making the system more transparent and user-friendly.



This chart illustrates the overall behavior of the model, showing how different days generally affect predictions. Spread-out dots indicate significant impact, while dots near zero show minimal effect.

Figure 1: Screenshot of a global explainability bee swarm plot made using shap values.

4 Update: Performance

Another enhancement in my updates was optimizing the performance of the forecasting system. The primary focus was to significantly reduce the time required for predictions and explanations, making the system more responsive and improving the user experience. By transitioning from LIME to SHAP for local explainability, I was able to achieve substantial performance gains while maintaining interpretability.

4.1 Transition from LIME to SHAP for Local Explainability

In the original implementation, local explainability was generated using LIME, a method that perturbs the input data and observes changes in predictions to determine feature importance. While effective, LIME required numerous perturbations and repeated model predictions, which increased computational overhead.

In my updates, I replaced LIME with SHAP for local explainability. Specifically, I used `shap.TreeExplainer`, which is optimized for tree-based models like XGBoost. This change allowed me to eliminate the need for perturbations and significantly reduced the number of predictions required to generate explanations. SHAP directly calculates feature importance, making it both faster and more accurate in this context.

4.2 Impact on Performance

The switch to SHAP resulted in a dramatic reduction in execution time for predictions and explanations. Below are the performance improvements observed before and after the updates:

Performance Measurements

Environment	Sequence Length	Predictions	Method	Time (seconds)	Reduction (%)
Local Machine	15	30	LIME + Bootstrapping	2.93	-
Local Machine	15	30	SHAP + Bootstrapping	0.16	94.54
Website (with GUI)	15	30	LIME + Bootstrapping	4.60	-
Website (with GUI)	15	30	SHAP + Bootstrapping	0.82	82.17

Table 2: Comparison of prediction times and reductions across methods and environments.

The reduction percentages highlight the significant performance gains:

- **Local case**: Execution time reduced by 94.54%.
- **Website case**: Execution time reduced by 82.17%.

These measurements were conducted 10 times to ensure robustness and averaged to obtain reliable results. Testing was performed on an AMD Ryzen 7 4800H CPU with a clock speed of 2.90 GHz.

4.3 Preserving Compatibility

An essential consideration during these updates was maintaining compatibility with the original system. Despite transitioning to a model-specific explainer for XGBoost, I ensured that the `ForecastExplainer` retained its model-agnostic functionality. This means that the explainer can still work with other types of models if required in future implementations. The changes were carefully designed to enhance performance without sacrificing flexibility.

4.4 Final Output

The optimized performance has significantly improved the usability of the forecasting system. Predictions and their explanations are now generated almost in real-time, with execution times consistently under 1 second, even in the web-based environment. This improvement greatly enhances the responsiveness of the system, making it more user-friendly and practical for real-world applications.

5 Summary of Updates

The updates to the forecasting system significantly enhanced its transparency and performance, addressing key limitations of the original implementation.

5.1 Key Achievements

1. Global Explainability: Introduced SHAP-based global explainability, visualized through a bee swarm plot in the GUI, providing insights into overall feature importance.
2. Improved Local Explainability: Replaced LIME with SHAP TreeExplainer for local explanations, enabling faster and more detailed insights.
3. Performance Optimization: Achieved substantial reductions in execution time:
   (a) Local case: 94.54% reduction.
   (b) Website case: 82.17% reduction.
4. Seamless Integration: Integrated changes without adding new API calls or breaking compatibility, ensuring the system remains model-agnostic for future updates.

6 Notes and Future Considerations

6.1 Alternative Approaches

During the updates, I also implemented an alternative method for calculating uncertainty bounds using residuals instead of bootstrapping. This approach calculates the residuals (differences between model predictions and true values) to estimate confidence intervals. While promising, this method has limitations. The primary challenge is that the models are trained on all available data, making it difficult to estimate residuals for unseen data without introducing bias. Additionally, residual-based bounds rely on residuals being normally distributed, which cannot be guaranteed for all combinations of KPIs and machines. While this method is implemented and available for use, it is actually not used, since it may be better suited for scenarios where overfitting is minimized, or the data distribution can be closely monitored.

6.2 Maintaining Flexibility

A key consideration during the updates was preserving the compatibility of the `ForecastExplainer` with models beyond XGBoost. Despite transitioning to SHAP TreeExplainer for XGBoost-specific optimizations, the explainer retains its ability to operate in a model-agnostic way. This ensures that the forecasting system remains adaptable to future changes, such as the introduction of new models or alternative architectures.

6.3 Future Improvements

Looking ahead, the system could benefit from exploring additional optimizations and explainability features. For example, integrating alternative global explainability techniques could provide complementary insights. Further refinement of residual-based bounds could also be pursued by testing on datasets specifically designed to minimize overfitting or by combining bootstrapping and residuals for hybrid confidence intervals.

References

[1] Davide Marchi. *Smart Factory Repository.* GitHub repository, Accessed: January 5, 2025. 2025. URL: `https://github.com/davide-marchi/smartfactory`.

[2] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 2016, pp. 1135–1144.

[3] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30.* Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[4] Scott M. Lundberg et al. “From local explanations to global understanding with explainable AI for trees”. In: *Nature Machine Intelligence* 2.1 (2020), pp. 2522–5839.