# Weekly Feedback Slidedoc

Zhengqi Wang

June 26, 2024

# Table of Contents

# Introduction to the Pre-Fitted Voting Model

## Brief Intro

- The **Pre-Fitted Voting Model** combines multiple diverse models to enhance prediction accuracy.
- Unlike typical ensemble methods, each constituent model in a Voting Model is *pre-trained* on different aspects of the data.
- This approach leverages the *strengths* of individual models and mitigates their *weaknesses*.
- **Key Advantage**: Improves robustness and reliability of predictions by aggregating diverse perspectives.
- Application in our project: Used to optimize performance on complex datasets, addressing specific challenges previously highlighted.

# Libraries and Tools Used

- **Scikit-learn**: An essential library for model building, evaluation, and machine learning workflows. Used for its robust, efficient tools for statistical modeling and machine learning.
- **Pandas**: Employed for data manipulation and analysis, providing data structures and operations for manipulating numerical tables and time series.
- **NumPy**: Utilized for its powerful numerical arrays that offer comprehensive mathematical functions, crucial for handling large datasets and calculations.
- **CatBoost**: Chosen for its ability to handle categorical data automatically with best-in-class performance on many tasks.
- **Optuna**: Applied for optimizing machine learning algorithms, particularly effective in hyperparameter tuning to enhance model performance.
- **SHAP (SHapley Additive exPlanations)**: Used to interpret the output of machine learning models, providing insights into the contribution of each feature to the prediction.

## Random Forest Classifier

- **Function**:
  - Constructs multiple decision trees during training.
  - Outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.
- **Advantages**:
  - Handles large datasets with higher dimensionality.
  - Provides robust performance and reduces overfitting through averaging.
- **Usage in Voting Model**:
  - Adds robustness and accuracy by leveraging its ensemble nature.

## LightGBM Classifier

- **Function**:
  - Uses gradient boosting framework for tree-based learning algorithms.
- **Advantages**:
  - Optimized for efficiency and speed.
  - Excellent performance on large datasets with complex models.
- **Usage in Voting Model**:
  - Enhances the ensemble's ability to handle large-scale data efficiently.

## CatBoost Classifier

- **Function**:
  - Handles categorical features natively without extensive preprocessing.
- **Advantages**:
  - High performance with minimal parameter tuning.
  - Reduces the need for extensive preprocessing of categorical data.
- **Usage in Voting Model**:
  - Improves the ensemble by efficiently handling categorical variables.

## XGBoost Classifier

- **Function**:
  - Uses optimized distributed gradient boosting library.
- **Advantages**:
  - Highly efficient, flexible, and portable.
  - Widely used for its performance and speed.
- **Usage in Voting Model**:
  - Enhances the ensemble's prediction accuracy and speed.

# Introduction to Data Preprocessing

- **Purpose**: Set the stage for model accuracy and efficiency.
- Data preprocessing is crucial for cleaning, transforming, and making the data suitable for modeling.
- We apply several targeted preprocessing techniques to address specific challenges in our dataset.

# Data Cleaning

- **Purpose**: Remove inconsistencies and errors to improve model reliability.
- Techniques include handling missing values, removing duplicates, and correcting erroneous entries.
- Example:

```
df.drop_duplicates()
df.fillna(method='ffill')
```

# Data Transformation

- **Purpose**: Normalize data scales and encode categorical variables to enhance model performance.
- Key operations include scaling numerical data and encoding categorical variables for model compatibility.
- Code Snippet:

```
from sklearn.preprocessing import StandardScaler,
OneHotEncoder
scaler =
StandardScaler().fit_transform(df[['numerical_column']])
encoder =
OneHotEncoder().fit_transform(df[['categorical_column']])
```

# Feature Engineering

- **Purpose**: Create new features from existing data to provide deeper insights and improve prediction accuracy.
- Techniques involve interactions, polynomial features, and domain-specific aggregations.

# Model Building Functions

- **Purpose**: Outline the methods and techniques used for constructing and training our models.
- Emphasizes the strategic selection of models based on the dataset characteristics and project objectives.
- Overview of different models explored and rationale behind each choice.

# Model Initialization

- **Purpose**: Discuss how models are initialized and configured before training.
- Important to select the right model parameters to optimize performance.
- Example Initialization of a Random Forest Model:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100,
    random_state=42)
```

# Model Training

- **Purpose**: Detail the training process, including handling of overfitting, validation strategies, and iterative improvements.
- Discusses cross-validation techniques and performance metrics used to ensure robustness.
- Example Training Code:

```
model.fit(X_train, y_train)
```

# Model Evaluation and Selection

- **Purpose**: Explain how models are evaluated and the criteria for selecting the final model.
- Importance of using a variety of metrics to gauge performance accurately.
- Example of Evaluation:

```
from sklearn.metrics import accuracy_score
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
```

# Introduction to Optimization Techniques

## Optimization Techniques Overview

- **Purpose**: Explore the strategies employed to refine model parameters and improve predictive accuracy.
- Discusses the importance of hyperparameter tuning, feature selection, and algorithmic adjustments.
- Highlights the tools used, like Optuna, and the impact of these techniques on model performance.

# Hyperparameter Tuning with Optuna

- **Purpose**: Detail the use of Optuna for finding optimal model parameters.
- Optuna automates the process of choosing the best set of hyperparameters.
- Example of Optuna Setup and Execution:

```
import optuna
def objective(trial):
    param = {
        'n_estimators': trial.suggest_int('n_estimators',
            50, 300),
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'min_samples_split': trial.suggest_int('
            min_samples_split', 2, 15)
    }
    clf = RandomForestClassifier(**param)
    return cross_val_score(clf, X_train, y_train, n_jobs=-1)
        .mean()
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)
```

# Feature Selection Techniques

- **Purpose**: Discuss methods used to select the most impactful features, reducing model complexity and improving performance.
- Techniques such as recursive feature elimination, feature importance, and correlation matrices are employed.
- Example of Feature Selection:

```
from sklearn.feature_selection import RFE
selector = RFE(estimator=RandomForestClassifier(),
    n_features_to_select=5)
selector = selector.fit(X_train, y_train)
selected_features = X_train.columns[selector.support_]
```

# Algorithmic Adjustments

- **Purpose**: Explain adjustments made to the learning algorithms themselves to improve efficiency and accuracy.
- Discusses techniques like adjusting learning rates, ensemble methods, and regularization strategies.
- Example of Adjusting Learning Rate:

```
learning_rate = 0.01
for epoch in range(100):
    model.train(data, learning_rate)
    learning_rate *= 0.95  # Decaying learning rate
```

# Time Series Analysis of Target Variable Over Weeks

- The graph displays the target variable trends over 90 weeks.
- Notable is the peak around week 60 followed by a sharp decline.
- The shaded area represents the confidence interval, providing insight into the variability and certainty of the mean estimates.

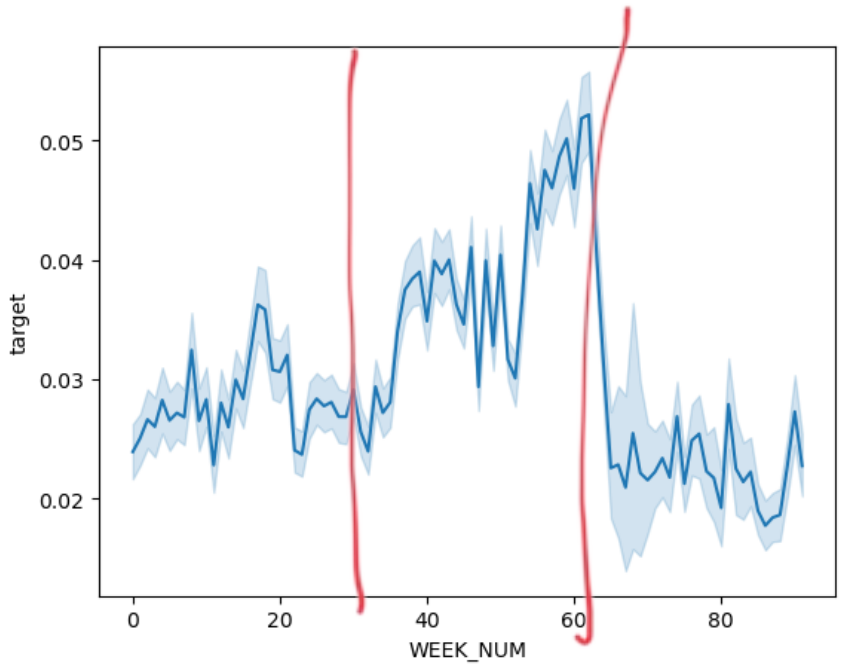


Figure: This plot showing the variation of the target variable over 90 weeks, including a confidence interval highlighting the variability in the data.

# Analysis of Target Variable Over Decision Weeks

- The image illustrates changes in the target variable as of week decisions.
- This plot help identify specific weeks where significant changes in the target variable, suggesting potential external impacts or seasonal effects.
- Observations of trends and anomalies in this plot can be crucial for optimizing decision-making processes related to the target variable.

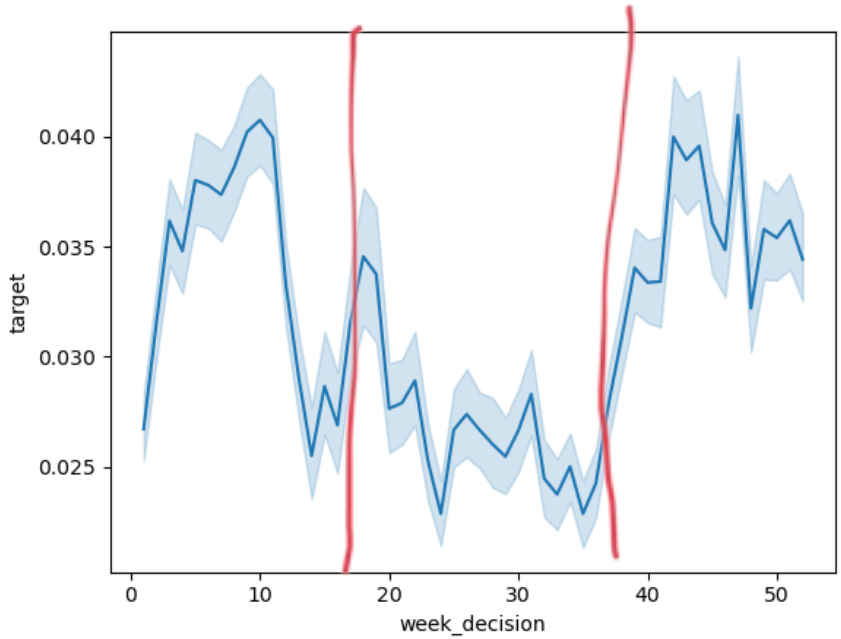


Figure: The plot displaying the fluctuation of the target variable across various decision weeks, showing a temporal trends and strong decision impacts on the target.

# Distribution of the Target Variable

- This plot highlights a significant imbalance between the two classes of the target variable: ’0’ (non-event) and ’1’ (event).
- Such imbalances can pose challenges in model training, potentially leading to biased predictions favoring the majority class.
- Consideration of techniques such as resampling or specialized algorithms might be necessary to address this imbalance for effective model training.

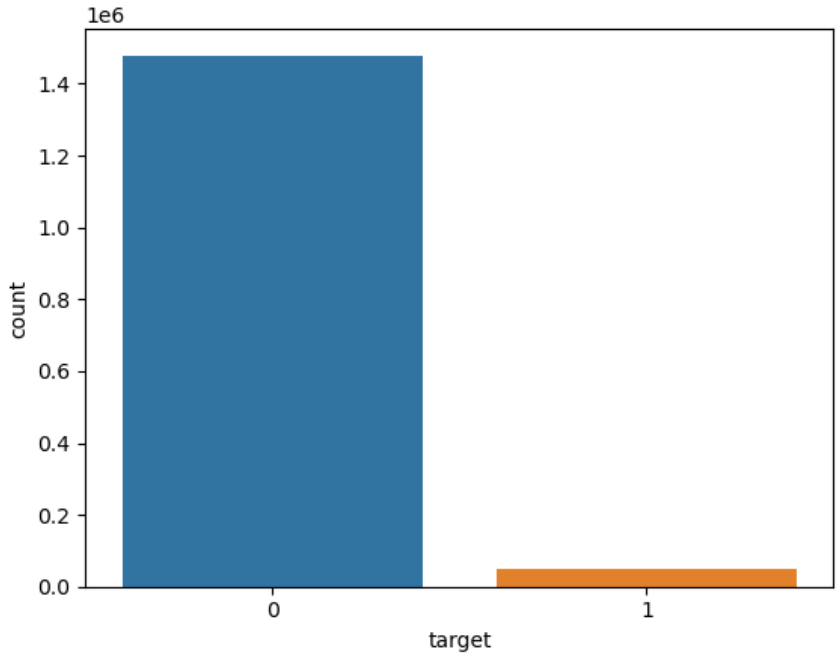


Figure: Bar plot showing the distribution of the target variable in the dataset. The count of class ’0’ significantly outweighs that of class ’1’, indicating an imbalanced dataset.

# Final Scores for Selected Cases

- Notice the variability in scores, suggesting differential impacts or outcomes across cases.
- These scores might be used to assess performance, prioritize actions, or understand model predictions.

| Case ID | Score |
|---|---|
| 57543 | 0.011331 |
| 57549 | 0.068231 |
| 57551 | 0.004470 |
| 57552 | 0.017425 |
| 57569 | 0.126229 |

Table: Table showing the final scores for a selection of cases. Scores reflect the outcome of the evaluated model or process.

# Summary of Key Findings

- The Pre-Fitted Voting Model significantly enhanced prediction accuracy by leveraging the strengths of multiple models.
- Hyperparameter tuning with Optuna resulted in measurable performance improvements across all models tested.
- Feature selection not only simplified the models but also improved their execution efficiency and interpretability.

# Lessons Learned

- The integration of diverse models and optimization techniques requires careful planning and rigorous testing.
- The importance of systematic data preprocessing was highlighted, demonstrating its impact on model performance.
- Collaborative tools and frameworks like Optuna provided essential insights into model behavior and performance enhancement.

# Future Directions

- Exploring additional ensemble techniques, such as stacked generalization, to further boost model robustness.
- Implementing deep learning models for areas of the dataset where traditional models underperformed.
- Integrating real-time data processing to enhance the model's applicability in dynamic environments.