TensorFlow Decision Forest Titanic

Rex Coleman

2024-11-13

Executive Summary

This project involves the comprehensive evaluation and optimization of TensorFlow Decision Forests for a classification task. The primary objective was to explore various hyperparameters to improve model performance and determine the most effective settings for the dataset. This report details the data exploration and preprocessing steps, the selection and training of the model, the hyperparameters evaluated, and the final results. The project demonstrates the advantages of using Gradient Boosted Trees and provides insights into the model's strengths and weaknesses compared to other machine learning models.

Table of Contents

1. Introduction

1.1 Overview of the Project

This project aims to evaluate and optimize TensorFlow Decision Forests for a classification task. By exploring various hyperparameters and their impact on model performance, the goal is to determine the most effective settings that enhance the accuracy and efficiency of the model. This evaluation involves thorough data exploration, preprocessing, and detailed model training processes.

1.2 Objectives and Goals

The primary objectives of this project are: - To understand the data and perform necessary preprocessing. - To train a baseline model using TensorFlow Decision Forests. - To improve the model through hyperparameter tuning and model ensembling. - To evaluate the impact of different hyperparameters on model performance. - To select the final model based on comprehensive performance metrics.

1.3 Importance of the Project

This project is significant as it showcases the power of Gradient Boosted Trees within TensorFlow Decision Forests for solving classification tasks. By systematically evaluating and tuning hyperparameters, the project highlights the importance of model optimization in achieving superior performance. The insights gained from this project are valuable for data scientists and machine learning practitioners aiming to leverage advanced techniques for enhanced model accuracy and reliability.

2. Data Exploration and Preprocessing

2.1 Description of the Dataset

The dataset used in this project is a comprehensive collection of data points relevant to the classification task at hand. It includes various features that contribute to the model's ability to make accurate predictions. The dataset is divided into training and testing sets to facilitate proper model evaluation and validation. The key characteristics of the dataset include the number of features, the target variable, and the distribution of classes.

2.2 Data Cleaning and Preprocessing Steps

Data cleaning and preprocessing are critical steps in preparing the dataset for model training. The steps undertaken in this project include: - **Handling Missing Values**: Identifying and imputing or removing missing values to ensure the dataset is complete. - **Feature Engineering**: Creating new features or transforming existing features to better represent the underlying data patterns. - **Normalization/Standardization**: Scaling numerical features to a common range to improve model performance. - **Encoding Categorical Variables**: Converting categorical variables into numerical formats using techniques such as one-hot encoding or label encoding. - **Splitting Data**: Dividing the dataset into training and testing sets to enable proper model validation.

2.3 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is conducted to gain insights into the dataset and identify any underlying patterns or anomalies. Key EDA steps include: - **Descriptive Statistics**: Summarizing the central tendency, dispersion, and shape of the dataset's distribution. - **Data Visualization**: Creating plots such as histograms, scatter plots, and box plots to visualize the distribution and relationships between features. - **Correlation Analysis**: Examining the correlation between features to identify potential multicollinearity issues and understand feature interactions. - **Class Distribution**: Analyzing the distribution of the target variable to address any class imbalance issues.

3. Model Selection and Training

3.1 Overview of TensorFlow Decision Forests

TensorFlow Decision Forests is a powerful library for training, evaluating, and serving decision forest models. It provides implementations of various decision tree algorithms, including Random Forests and Gradient Boosted Trees, which are highly effective for both classification and regression tasks. This section provides an overview of the TensorFlow Decision Forests library, its features, and its advantages.

3.2 Baseline Model Training

The baseline model serves as the initial benchmark for evaluating the performance of the decision forest model. The steps involved in training the baseline model include: - **Model Initialization**: Setting up the initial model using default hyperparameters. - **Training**: Fitting the model to the training data. - **Evaluation**: Assessing the model's performance on the testing set using various metrics such as accuracy, precision, recall, and F1 score.

3.3 Improved Model Training

Building on the baseline model, improved model training involves optimizing the model to enhance its performance. This includes: - **Feature Selection**: Identifying and selecting the most relevant features to reduce overfitting and improve model accuracy. - **Hyperparameter Tuning**: Systematically exploring different hyperparameter values to find the optimal settings for the model. - **Cross-Validation**: Implementing cross-validation techniques to ensure the model's robustness and generalizability.

3.4 Hyperparameter Tuning

Hyperparameter tuning is a crucial step in improving model performance. This involves: - **Grid Search**: Exhaustively searching through a specified subset of hyperparameters to identify the best combination. - **Random Search**: Randomly sampling hyperparameters from a defined space and selecting the best performing set. - **Bayesian Optimization**: Using probabilistic models to select hyperparameters that maximize the model's performance.

3.4.1 List of Hyperparameters Considered

- **min_examples**: Minimum number of examples required to make a split.
- **categorical_algorithm**: Strategies like "CART" and "RANDOM" for handling categorical features.
- **growing_strategy**: Methods like "LOCAL" and "BEST_FIRST_GLOBAL" for growing the decision tree.
- **max_depth**: Maximum depth of the tree.

- **max_num_nodes**: Maximum number of nodes in the tree.
- **shrinkage**: The learning rate for gradient boosting.
- **num_candidate_attributes_ratio**: Ratio of candidate features to consider for splits.
- **split_axis**: Strategies like "AXIS_ALIGNED" and "SPARSE_OBLIQUE" for splitting nodes.
- **sparse_oblique_normalization**: Normalization methods for sparse oblique splits.
- **sparse_oblique_weights**: Weight types for sparse oblique splits.
- **sparse_oblique_num_projections_exponent**: Exponent for the number of projections in sparse oblique splits.

3.4.2 Impact of Each Hyperparameter on Model Performance

- **min_examples**: Controls the minimum number of samples required to split an internal node. Higher values can prevent overfitting by ensuring each split is based on a sufficient number of samples.
- **categorical_algorithm**: Determines how categorical features are handled. "CART" splits based on the most frequent category, while "RANDOM" chooses a random category for splits. For instance, with the Titanic dataset, 'Embarked' can be split differently, influencing the model's understanding of passenger embarkation ports.
- **growing_strategy**: Defines how the decision tree is grown.
  - **LOCAL**: Focuses on controlling tree depth, preventing over-complexity.
  - **BEST_FIRST_GLOBAL**: Allows the tree to grow until a specified number of nodes is reached, which can capture more interactions between features like 'Age' and 'Fare'.
- **max_depth**: Limits the depth of the tree to control complexity. For example, a max depth of 8 might capture complex interactions but risk overfitting.
- **max_num_nodes**: Sets the maximum number of nodes, allowing for finer decision boundaries. More nodes might help the model distinguish between different groups of passengers.
- **shrinkage**: Controls the learning rate, influencing how much each tree contributes to the final model.
- **num_candidate_attributes_ratio**: Specifies the fraction of features to consider for each split, affecting how diverse the splits can be.
- **split_axis**: Determines the method of node splitting.
  - **AXIS_ALIGNED**: Splits based on a single feature.
  - **SPARSE_OBLIQUE**: Uses a combination of features, providing more flexibility but increasing complexity.
- **sparse_oblique_normalization**: Defines how to normalize feature weights in sparse oblique splits, impacting how the model handles feature scales.
- **sparse_oblique_weights**: Specifies the type of weights for sparse oblique splits.
- **sparse_oblique_num_projections_exponent**: Sets the exponent for the number of projections, affecting the complexity of splits.

3.4.3 Range of Values Explored for Each Hyperparameter

- **min_examples**: [2, 5, 7, 10]
- **categorical_algorithm**: ["CART", "RANDOM"]
- **growing_strategy**: ["LOCAL", "BEST_FIRST_GLOBAL"]
- **max_depth**: [3, 4, 5, 6, 8]
- **max_num_nodes**: [16, 32, 64, 128, 256]
- **shrinkage**: [0.02, 0.05, 0.10, 0.15]
- **num_candidate_attributes_ratio**: [0.2, 0.5, 0.9, 1.0]
- **split_axis**: ["AXIS_ALIGNED", "SPARSE_OBLIQUE"]
- **sparse_oblique_normalization**: ["NONE", "STANDARD_DEVIATION", "MIN_MAX"]
- **sparse_oblique_weights**: ["BINARY", "CONTINUOUS"]
- **sparse_oblique_num_projections_exponent**: [1.0, 1.5]

3.4.4 Final Hyperparameter Settings Used

- **min_examples**: 5
- **categorical_algorithm**: CART
- **growing_strategy**: BEST_FIRST_GLOBAL
- **max_depth**: 6
- **max_num_nodes**: 128
- **shrinkage**: 0.1
- **num_candidate_attributes_ratio**: 0.5
- **split_axis**: AXIS_ALIGNED
- **sparse_oblique_normalization**: STANDARD_DEVIATION
- **sparse_oblique_weights**: CONTINUOUS
- **sparse_oblique_num_projections_exponent**: 1.5

3.4.5 Additional Hyperparameters to Consider

- **max_features**: Number of features to consider when looking for the best split.
  - **Justification**: Limiting the number of features can reduce overfitting and improve generalization.
  - **Suggested range**: [0.5, 0.7, 0.9, 1.0]
- **subsample**: Fraction of samples to use for fitting individual base learners.
  - **Justification**: Introducing randomness by using only a fraction of samples can help prevent overfitting.
  - **Suggested range**: [0.5, 0.7, 0.9, 1.0]
- **learning_rate**: Weight of each individual tree.
  - **Justification**: Fine-tuning the learning rate can improve model performance and generalization.
  - **Suggested range**: [0.01, 0.05, 0.1, 0.2]
- **num_trees**: Number of trees in the forest.
  - **Justification**: More trees can improve performance but increase computation time. This project used 100.
  - **Suggested range**: [50, 100, 200, 500]
- **min_impurity_decrease**: Threshold for a split to be considered.
  - **Justification**: Controls the minimum decrease in impurity required to split a node, balancing model complexity and performance.
  - **Suggested range**: [0.0, 0.01, 0.05, 0.1]

3.4.6 Expanded Ranges of Values to Consider

- **max_depth**: [2, 4, 6, 8, 10, 12]
  - **Justification**: Including shallower and deeper trees can help find the optimal depth for capturing patterns without overfitting.
- **max_num_nodes**: [16, 32, 64, 128, 256, 512]
  - **Justification**: More nodes can allow for finer decision boundaries and better capture complex interactions.
- **shrinkage**: [0.01, 0.05, 0.10, 0.15, 0.2]
  - **Justification**: Testing lower and higher values can help find the optimal learning rate.

- **num_candidate_attributes_ratio**: [0.1, 0.3, 0.5, 0.7, 0.9, 1.0]
  - **Justification**: Adding intermediate values provides finer granularity in tuning, improving the model's ability to generalize.
- **sparse_oblique_num_projections_exponent**: [1.0, 1.2, 1.5, 1.8]
  - **Justification**: Including values between the existing range can provide better tuning and model performance.

3.5 Model Ensembling

Model ensembling involves combining multiple models to improve overall performance. Techniques used in this project include: - **Voting Ensemble**: Combining predictions from multiple models and using a majority vote or averaging to make the final prediction. - **Stacking**: Training a meta-model on the predictions of several base models to achieve better performance. - **Bagging and Boosting**: Implementing methods like Bagging (Bootstrap Aggregating) and Boosting to reduce variance and bias in the model.

3.5.1 Understanding TensorFlow Decision Forests TensorFlow Decision Forests (TF-DF) is an open-source library for training, evaluating, and serving decision forest models within the TensorFlow ecosystem. Decision forests, including Random Forests, Extremely Randomized Trees, and Gradient Boosted Trees, are ensemble learning methods that combine the predictions of multiple decision trees to produce a single output. TF-DF simplifies the use of these models within TensorFlow, leveraging its ecosystem for easy integration with other TensorFlow tools and models.

3.5.2 Types of Decision Forests There are several types of decision forests, each with unique characteristics and suitable applications:

1. **Random Forests**:
   - An ensemble method that trains multiple decision trees on different parts of the data and averages their predictions.
   - Strengths: Robust to overfitting, handles large datasets well, and provides good generalization performance.
   - Suitable for: Baseline models, quick prototyping, and tasks where interpretability and training speed are important.
2. **Extremely Randomized Trees (Extra Trees)**:
   - Similar to Random Forests but with more randomness in node splitting, which can lead to better generalization.
   - Strengths: Fast training, less prone to overfitting, often performs well with less parameter tuning.
   - Suitable for: Situations where training speed and robustness to hyperparameters are crucial.
3. **Gradient Boosted Trees (GBTs)**:
   - A sequential ensemble method that builds trees iteratively, where each tree tries to correct the errors of the previous ones.
   - Strengths: High accuracy, excellent for ranking and regression tasks, often outperforms other models on structured/tabular data.
   - Suitable for: Tasks requiring high predictive accuracy, handling complex interactions in data, and problems where fine-tuned performance is essential.

3.5.3 Choosing the Right Decision Forest Choosing the appropriate decision forest depends on several factors:

- **Data Size and Complexity**: Random Forests are generally faster and simpler to train on large datasets, while GBTs can handle complex interactions better.
- **Accuracy vs. Interpretability**: GBTs often provide higher accuracy but can be harder to interpret compared to Random Forests.
- **Training Time**: Random Forests and Extra Trees are parallelizable and thus faster to train, while GBTs require more time due to their sequential nature.
- **Overfitting Risk**: Extra Trees and Random Forests are robust to overfitting, whereas GBTs can overfit if not properly regularized.

3.5.4 Justification for Using Gradient Boosted Trees (GBTs) For this project, Gradient Boosted Trees (GBTs) were selected based on the following considerations:

- **High Accuracy Requirement**: Given the critical nature of the task, achieving high accuracy was paramount. GBTs are known for their superior performance in terms of accuracy, especially for structured/tabular data.
- **Handling Complex Interactions**: The project dataset likely contains complex interactions between features. GBTs are well-suited for capturing and modeling these interactions effectively.
- **Benchmark Performance**: In preliminary experiments, GBTs consistently outperformed other models, justifying their use for the final model.

By leveraging TensorFlow Decision Forests (TF-DF) and selecting Gradient Boosted Trees, the project aims to achieve the best possible performance, ensuring robust and accurate predictions. The choice of GBTs is grounded in their proven ability to handle complex data structures and deliver high accuracy, making them ideal for the project's objectives.

4. Results

- The initial model trained with default parameters achieved an accuracy of 0.8261 and a loss of 0.8609.
- After improving the default parameters, the model achieved an accuracy of 0.7826 and a loss of 1.0587.
- Hyperparameter tuning resulted in a model with an accuracy of 0.8630 and a loss of 0.6750.
- The ensemble of models provided a robust solution by averaging predictions from multiple models, resulting in a reliable final submission.

5. Conclusion

5.1 Summary

This project provided a thorough exploration and optimization of TensorFlow Decision Forests for a classification task, demonstrating the critical steps in data preprocessing, model training, and hyperparameter tuning. The systematic approach to model selection, particularly the focus on Gradient Boosted Trees, highlighted the importance of precise hyperparameter tuning in achieving superior model performance.

Key takeaways from this project include:

- **Data Exploration and Preprocessing**: Proper data cleaning, feature engineering, and EDA are foundational to building an effective model.
- **Baseline and Improved Models**: Establishing a baseline with default parameters and systematically improving it through targeted enhancements and hyperparameter tuning is crucial for achieving optimal performance.

- **Hyperparameter Tuning**: The exhaustive evaluation of hyperparameters such as `min_examples`, `categorical_algorithm`, `growing_strategy`, and others underscored their impact on model performance. The refined hyperparameters significantly improved the model's accuracy and robustness.
- **Model Ensembling**: Employing ensemble techniques like voting, stacking, bagging, and boosting further enhanced the model's generalizability and reliability.

The final model, leveraging Gradient Boosted Trees, showcased a notable improvement in accuracy and efficiency compared to the baseline model. This project not only underscores the power of TensorFlow Decision Forests but also demonstrates the meticulous process required to optimize machine learning models for real-world applications.

For aspiring data scientists, this project serves as a comprehensive guide on how to approach model selection, hyperparameter tuning, and performance optimization. The insights gained here are invaluable for tackling similar classification tasks in cybersecurity and other domains.

5.2 Future Work

Future enhancements to this project could include:

- **Exploring Additional Hyperparameters**: Investigating other hyperparameters and their interactions could uncover further performance gains.
- **Integrating Deep Learning Techniques**: Combining decision forests with deep learning models could potentially lead to even more powerful predictive models.
- **Applying to Diverse Cybersecurity Focused Datasets**: Testing the optimized model on different datasets to validate its robustness and generalizability.

By continuously refining the approach and exploring new techniques, the potential for creating highly accurate and efficient models remains vast.

This detailed evaluation and optimization of TensorFlow Decision Forests reflect my commitment to leveraging advanced machine learning techniques to solve complex problems. I am confident that my approach to model selection and performance optimization will contribute significantly to any data science team, particularly in the cybersecurity industry, where precise and reliable models are paramount.

Thank you for considering my portfolio. I look forward to the opportunity to discuss how my skills and experiences align with the needs of your team.

6. References

- **Original Kaggle Notebook**: Gusthema, achoum, 2023, Titanic competition w/ TensorFlow Decision Forests