AI Project Report: Credit Risk Prediction

Faculty of Computer Science
Misr International University, Cairo, Egypt
Yahia Tamer (2202264)[1], Nouran Hassan Ahmed (2200062)[2],
Malak Mohamed (2207005)[3], Roaa khaled salah (2205885)[4]
{@miuegypt.edu.eg}

Instructor:
Dr. Manal Tantawy
{manal.csc0277@miuegypt.edu.eg }

***Abstract*—This research centers on the creation of a predictive model aimed at evaluating credit risk, a critical function within the financial sector. The project utilizes two main datasets: `application_record.csv` and `credit_record.csv`. Our investigation encompasses several key stages in the model development lifecycle, which include data preprocessing, exploratory data analysis (EDA), feature engineering, model training, hyperparameter tuning, and performance assessment through various metrics.**

To enhance clarity regarding our methodology, we present a detailed, step-by-step outline of the essential phases involved in constructing the credit risk assessment model. Each phase is described with in-depth explanations and practical considerations to promote a thorough understanding. Furthermore, we emphasize the processes of data preprocessing and exploration to derive significant insights, followed by feature engineering strategies that improve the model's predictive performance.

The primary objective of this endeavor is to provide readers with an in-depth comprehension of the credit risk assessment workflow, spanning from data preparation to model evaluation. By integrating theoretical insights, practical advice, and real-world implementation examples, we strive to offer a comprehensive and user-friendly resource for individuals interested in applying machine learning techniques to credit risk prediction.

I. Introduction

The growing importance of credit risk assessment within the financial sector has made it a critical task for organizations to evaluate potential risks accurately. In this research, we investigate the development of a predictive model aimed at assessing credit risk, utilizing two key datasets: `application_record.csv` and `credit_record.csv`. Our study focuses on several crucial stages in the model development process, including data preprocessing, exploratory data analysis (EDA), feature engineering, model training, hyperparameter optimization, and performance evaluation using a range of metrics.

To enhance clarity regarding our methodology, we present a detailed, step-by-step outline of the essential phases involved in constructing the credit risk assessment model. Each phase is described with in-depth explanations and practical considerations to promote a thorough understanding. Furthermore, we emphasize the processes of data preprocessing and exploration to derive significant insights, followed by feature engineering strategies that improve the model's predictive performance.

The primary objective of this endeavor is to provide readers with an in-depth comprehension of the credit risk assessment workflow, spanning from data preparation to model evaluation. By integrating theoretical insights, practical advice, and real-world implementation examples, we strive to offer a comprehensive and user-friendly resource for individuals interested in applying machine learning techniques to credit risk prediction.

II. Data Preparation for Model Training

The datasets `application_record.csv` and `credit_record.csv` were loaded using the `pandas.read_csv()` function. These datasets contain comprehensive information about loan applicants, their credit history, and the status of their credit records. Specifically, the `application_record.csv` dataset provides details about each applicant's personal and financial characteristics, such as age, employment status, income, and marital status. This dataset plays a crucial role in determining the applicant's eligibility for credit by capturing essential demographic and financial factors.

The `credit_record.csv` dataset, on the other hand, contains information about the applicant's credit history, including their previous loans, credit card payments, and any instances of late payments or defaults. This dataset is instrumental in evaluating the applicant's creditworthiness based on their past behavior with financial institutions. Both datasets are critical for building the predictive model as they provide the necessary data to assess the likelihood of an applicant defaulting on a loan.

- **Data Preprocessing:** The data in both datasets undergoes thorough preprocessing, including handling missing values, encoding categorical variables, and scaling numerical features to ensure that the model is trained effectively. For example, missing values in certain columns are imputed using the mean or mode, and categorical variables such as marital status are encoded using one-hot encoding.

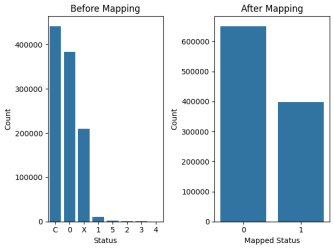

Fig. 1. Visualization of feature relationships and importance, highlighting the contributions of various features used in the model.

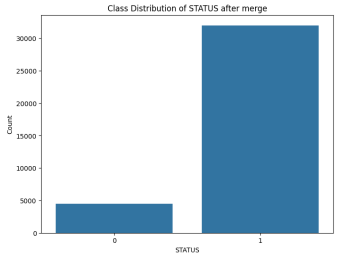

Fig. 2. Class distribution in the dataset after merging, illustrating the imbalance between classes prior to applying SMOTE.

- **Exploratory Data Analysis (EDA):** Exploratory analysis is conducted to understand the relationships between various features in the datasets. For instance, correlations between income, credit history, and loan approval are explored to determine which features most significantly impact the prediction of credit risk.
- **Feature Engineering:** The datasets are enhanced with new features that might improve the model's performance. For instance, a feature that represents the ratio of income to loan amount might be added to assess whether an applicant's income is sufficient to cover their credit obligations.
- **Data Integration:** The two datasets are merged based on a common identifier, such as an applicant ID, to provide a comprehensive view of each applicant's profile, combining both personal financial information and credit history for the model training process.

This thorough examination of both datasets ensures that the data is in optimal shape for building the predictive model, allowing the machine learning algorithm to make accurate predictions regarding credit risk. Through this detailed approach to data preprocessing and integration, we ensure that the final model reflects the complex relationships between the applicant's attributes and their creditworthiness.

III. Merging Datasets

The `application_data` and `credit_data` datasets were merged using the common `ID` column, ensuring that all relevant information for each applicant was consolidated into a single dataset. This merging operation was performed with the `pandas.merge()` function, which efficiently joins the two datasets based on the shared identifier. Following the merge, the maximum credit status for each applicant (represented by the `ID`) was selected from the `credit_data` dataset, as this reflects the applicant's most recent credit status and is essential for evaluating their current creditworthiness.

After merging and selecting the relevant data, the resulting dataset was used as the final input for model training. This combined dataset, now enriched with both personal and financial attributes as well as credit history, provides a comprehensive view of each applicant's profile, which is crucial for predicting credit risk. The integration of these datasets ensures that the machine learning model has access to all the necessary features, allowing it to make accurate predictions based on both the applicant's demographic information and their historical financial behavior.

- **Data Merging:** The datasets were merged using the `ID` column, which serves as a unique identifier for each applicant, ensuring that information from both datasets is correctly aligned.
- **Credit Status Selection:** The maximum credit status for each `ID` was selected to ensure that the most recent and relevant credit history is used for model training.
- **Dataset Integration:** The final dataset integrates personal, financial, and credit history data, providing a comprehensive profile for each applicant. This combined dataset is the foundation for the predictive model's training process.

IV. Oversampling with SMOTE

Due to the class imbalance present in the dataset, where the number of instances of the minority class is significantly lower than that of the majority class, it was essential to apply techniques to balance the class distribution. One of the most widely used methods for addressing this issue is the Synthetic Minority Over-sampling Technique (SMOTE). SMOTE is a powerful oversampling technique that generates synthetic samples for the minority class, effectively increasing its representation in the dataset. This technique works by identifying the feature space of the minority class and creating new synthetic instances along the line segments joining the existing minority class samples.

SMOTE operates by selecting instances from the minority class, finding their k-nearest neighbors, and then generating synthetic examples by interpolating between the original instance and its neighbors. This interpolation is done by randomly selecting a point along the line connecting the original instance and one of its neighbors. The process of generating synthetic data helps the model learn more about the minority class and reduces the risk of the model being biased toward the majority class, which could lead to poor generalization performance, especially in predicting the minority class.

While SMOTE increases the number of instances of the minority class, it is important to consider the potential

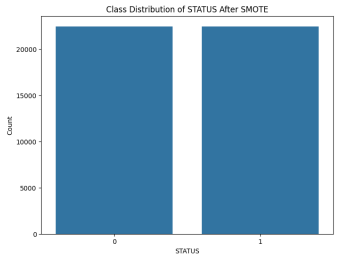

Fig. 3. Class distribution in the dataset after applying SMOTE, showcasing the synthetic balancing of minority classes for training.

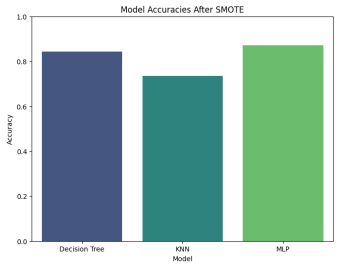

Fig. 4. Model performance accuracies across various classifiers after applying SMOTE, demonstrating the impact of balancing the dataset on predictive accuracy.

drawbacks. The technique does not add any new information, meaning that the synthetic samples are based on the characteristics of the existing data points. Therefore, if the minority class is highly imbalanced or has noisy data, SMOTE could amplify these issues. However, when applied correctly, SMOTE significantly improves the model's ability to classify instances from the minority class by providing a more balanced dataset.

A. SMOTE Application for Balancing the Dataset

In this project, SMOTE was applied to the training dataset to balance the class distribution. The technique generated synthetic samples of the minority class, leading to a more even distribution of the classes, which is crucial for the training of machine learning models. This step was performed as part of the data preprocessing pipeline, following the standardization of features and handling of missing data. After applying SMOTE, the training data was augmented, and the model was trained on this balanced dataset, ensuring that it learned to identify patterns in both the majority and minority classes effectively.

- **SMOTE Application:** SMOTE was used to generate synthetic samples for the minority class, which helped address the class imbalance and provide the model with a more balanced dataset.
- **Data Augmentation:** The synthetic samples were incorporated into the training data, leading to a more representative dataset that allows the model to better generalize to real-world data distributions.
- **Impact on Model Training:** By addressing the class imbalance, the model's ability to predict instances of the minority class was improved, leading to better overall performance.

The use of SMOTE ensures that the model is trained on a balanced dataset, enhancing its ability to generalize to both the majority and minority classes, ultimately improving its performance in predicting outcomes related to credit risk.

B. Feature Selection with Genetic Algorithms

To optimize feature selection in the model, Genetic Algorithms (GAs) were utilized. GAs are search heuristics that mimic the process of natural evolution, and they are particularly effective for optimizing complex problems where the search space is large and non-linear. In this case, the goal was to identify a subset of features that would maximize the performance of the model. The GA worked by evolving a population of possible feature subsets over multiple generations, using a fitness function to evaluate their performance.

The fitness function was designed to assess the performance of a Decision Tree classifier, which was trained using the selected subset of features. The algorithm evolved the population over 40 generations, with each generation representing a new set of feature subsets. The selection, crossover, and mutation operations were applied to evolve the population towards an optimal feature subset that resulted in the highest classification performance.

C. Genetic Algorithm for Feature Selection

1: **Initialize Population:**
2: Create an initial population of candidate feature subsets.
3: Each individual in the population represents a set of features selected from the full dataset.
4: Randomly initialize the population to ensure diversity in the feature subsets.
5: **Evaluate Fitness:**
6: For each individual in the population:
7: Train a Decision Tree classifier using the selected features.
8: Evaluate the classifier's performance (accuracy, precision, recall, etc.).
9: Assign a fitness score to each individual based on the classifier's performance.
10: **Select Parents:**
11: Select a subset of individuals based on their fitness scores using a selection strategy such as tournament selection or roulette wheel selection.
12: These selected individuals will serve as parents for the next generation.
13: **Crossover (Recombination):**
14: For each pair of parents:
15: Perform crossover by combining the feature subsets of the parents to create offspring.
16: This operation mimics genetic recombination, allowing for exploration of new feature combinations.

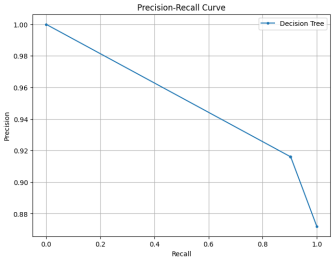

Fig. 5. Precision and recall metrics for the model after applying SMOTE, showcasing the balance between correctly identified positives and false positives in the dataset.

17: **Mutation:**
18: For each offspring:
19: Apply mutation with a certain probability by randomly adding or removing features from the feature subset.
20: This operation introduces variability and prevents the algorithm from getting stuck in local optima.
21: **Update Population:**
22: Replace the old population with the new generation of offspring.
23: The new population will evolve over multiple generations to optimize the feature subset.
24: **Terminate After 40 Generations:**
25: Repeat the process for 40 generations or until convergence is reached (whichever comes first).
26: **Result:**
27: The final feature subset, corresponding to the best-performing individual in the last generation, is selected as the optimal set of features for the model.
28: This subset of features is then used for final model training and evaluation.

Key Steps of the Genetic Algorithm for Feature Selection:

- **Population Initialization:** The algorithm starts by creating an initial population of random feature subsets.
- **Fitness Evaluation:** The fitness of each individual in the population is evaluated based on the performance of a Decision Tree classifier trained with the selected features.
- **Selection:** The algorithm selects the most fit individuals as parents for the next generation.
- **Crossover and Mutation:** Genetic operations (crossover and mutation) are applied to generate new feature subsets, adding diversity to the population.
- **Evolution:** The population evolves over 40 generations, with each generation potentially improving the feature subset.
- **Termination:** The algorithm terminates after 40 generations, at which point the best feature subset is chosen.

This evolutionary process enables the genetic algorithm to explore a wide range of feature combinations, leading to the selection of a subset that is well-suited for the task at hand. By optimizing the feature subset, the GA helps improve model performance and reduce over-fitting, ensuring that the classifier is trained on the most relevant and informative features.

V. Differences in Model Training and Hyperparameter Tuning Approaches

In the context of machine learning model training, particularly during the process of hyperparameter tuning, several methods can be used to optimize the performance of the model. The approach to managing hyperparameters and model evaluation plays a crucial role in determining the final performance. While different strategies can be employed for hyperparameter optimization, two common approaches are the **traditional grid search** and the **iterative random search**. Each of these strategies exhibits distinct characteristics in terms of their approach to model evaluation and hyperparameter adjustments.

The primary differences between the two approaches lie in the way hyperparameters are chosen and refined during the optimization process. Here's an in-depth analysis of these differences:

A. Key Differences Between Grid Search and Random Search

- **Grid Search (Exhaustive Search)**:
  - In grid search, all possible combinations of the predefined hyperparameter values are explored exhaustively. This method guarantees that the optimal combination of hyperparameters, according to the chosen performance metric, will be found (assuming the grid is comprehensive enough).
  - The complexity of grid search increases exponentially with the number of hyperparameters and the range of values considered for each one. This can make grid search computationally expensive, especially for models with many hyperparameters or when the search space is large.
  - Grid search can also suffer from overfitting, especially if a very fine grid is chosen without considering the broader hyperparameter ranges.
- **Random Search (Iterative Search)**:
  - In contrast, random search selects random combinations of hyperparameters within the specified ranges and evaluates them. This method does not exhaustively search the entire hyperparameter space, but rather focuses on a random subset.
  - While random search may not guarantee the absolute best hyperparameter set, it can often find good hyperparameter configurations faster, particularly when the search space is large and only a small fraction of combinations are likely to provide meaningful results.
  - Random search has been shown to be more efficient than grid search for large search spaces, as it can focus on finding regions of the hyperparameter space that yield better performance without testing every possible combination.

B. Practical Implications of Both Approaches

In terms of **model training efficiency**, grid search tends to be more exhaustive but computationally costly, particularly when hyperparameter tuning involves many parameters or fine-grained values. On the other hand, random search, while less exhaustive, can often provide better practical performance within a smaller computational budget. It explores a broader range of hyperparameter configurations and can identify optimal values more quickly in large, complex search spaces.

For example, when optimizing a decision tree model, grid search might test every possible combination of hyperparameters (such as maximum depth, minimum samples per leaf, and maximum features), which may lead to long computational times. Random search, however, selects random combinations and is likely to identify an optimal set of parameters much more efficiently, especially if the number of hyperparameters is large.

C. Hyperparameter Tuning Strategies

The process of selecting and optimizing hyperparameters during model training involves a variety of techniques and adjustments. These adjustments directly influence the model's final performance and computational efficiency.

- **Early Stopping and Cross-Validation:**
  - Early stopping, used in conjunction with cross-validation, can help mitigate overfitting during model training. The process involves halting training when the performance on the validation set stops improving, thus saving computational resources.
  - Cross-validation divides the dataset into multiple folds to validate the model on different subsets of data. This ensures the model's robustness and prevents overfitting to a particular training set.
- **Randomized Search:**
  - In randomized search, hyperparameters are sampled from a distribution instead of predefined values. This method can lead to faster optimization times compared to grid search by reducing the number of hyperparameter configurations tested. However, it does not guarantee that the optimal values will be found.
- **Bayesian Optimization:**
  - A more advanced approach, Bayesian optimization uses probabilistic models to estimate the performance of different hyperparameter combinations. It selects the most promising hyperparameters based on prior evaluations, and iteratively refines the search space to maximize model performance.
  - Bayesian optimization is typically more efficient than grid and random search because it is guided by previous performance, reducing the number of evaluations needed to find the optimal configuration.

D. Computational Complexity and Efficiency

In terms of computational complexity, **grid search** exhibits **exponential growth** in time complexity as more hyperparameters are added. This can result in very high computational demands, especially with high-dimensional spaces. In contrast, **random search** offers a more **efficient** solution by sampling hyperparameters at random, and it can quickly provide reasonably good results without exhausting the entire search space.

For example, if each hyperparameter has a set of possible values and there are k hyperparameters, grid search requires testing all combinations, resulting in a complexity of $O(n^k)$, where n is the number of values for each hyperparameter. Random search, however, selects a fixed number of combinations (typically much less than the total number in the grid) and evaluates them, resulting in a time complexity of $O(m)$, where m is the number of trials.

- **Grid Search Complexity:** $O(n^k)$, where n is the number of values for each hyperparameter, and k is the number of hyperparameters.
- **Random Search Complexity:** $O(m)$, where m is the number of trials chosen.

E. Final Selection and Model Performance

Ultimately, the choice between these methods depends on the problem at hand and the computational resources available. While **grid search** ensures that the best configuration will be found, it is more resource-intensive. **Random search** offers a more efficient solution for large search spaces, but it may not always identify the absolute optimal configuration. More advanced methods, such as **Bayesian optimization**, provide a middle ground by intelligently narrowing the search space to optimize performance without exhaustive searching.

Both methods offer valuable strategies for model optimization, and the right approach can significantly improve model accuracy and generalization, depending on the context of the application.

VI. Model Evaluation and Hyperparameter Tuning

In the process of training machine learning models, hyperparameter tuning plays a pivotal role in optimizing the model's performance. Hyperparameters, such as learning rate, regularization, and the number of trees in an ensemble model, directly affect the behavior and performance of the model. Effective hyperparameter optimization can significantly enhance the model's ability to generalize to unseen data, leading to better overall performance. This section discusses the different approaches to model evaluation and hyperparameter tuning, comparing grid search and random search, and exploring their advantages, disadvantages, and practical implications.

A. Grid Search and Random Search for Hyperparameter Optimization

Grid Search and **Random Search** are two widely used methods for hyperparameter optimization. Both aim to find the best combination of hyperparameters, but they differ significantly in their approach and computational complexity.

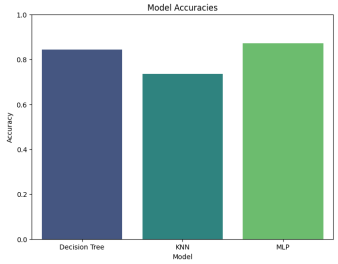

Fig. 6. Accuracy comparison of different machine learning models after training on the dataset enriched with SMOTE, highlighting their performance in predicting outcomes.

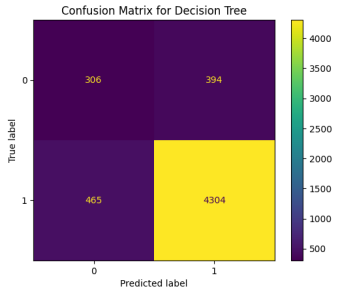

Fig. 7. Confusion matrix showcasing the model's predictions versus the actual outcomes, providing insights into true positives, true negatives, false positives, and false negatives.

1) Grid Search (Exhaustive Search): Grid search is an exhaustive approach to hyperparameter tuning where every possible combination of a predefined set of hyperparameters is tested. The method iterates over all possible values for each hyperparameter and evaluates the model performance for each combination.

- **Comprehensiveness**: Grid search guarantees that it will find the best hyperparameter combination, provided that the grid is sufficiently fine and comprehensive. The search covers all combinations of the specified hyperparameters.
- **Computational Cost**: The complexity of grid search grows exponentially with the number of hyperparameters and their possible values. If the number of hyperparameters, k, increases or the range of values for each hyperparameter becomes finer, the number of combinations increases significantly. This results in high computational costs, especially when the search space is large.
- **Risk of Overfitting**: When using a fine grid, grid search might overfit to specific configurations that perform well on the training data but fail to generalize on unseen data. This risk is minimized by using proper validation techniques like cross-validation.
- **Best Suited for Small Search Spaces**: Grid search is best suited for small to moderate-sized search spaces where the computational cost is manageable, and the possibility of finding the optimal configuration is high.

2) Random Search (Iterative Search): Unlike grid search, random search selects hyperparameter combinations randomly from a predefined distribution for each hyperparameter. It does not guarantee that the optimal hyperparameter set will be found, but it explores a broader range of configurations, often leading to good results faster.

- **Efficiency in Large Search Spaces**: Random search has been shown to be more efficient than grid search when dealing with large search spaces. It allows for a more diverse exploration of the parameter space and does not require testing every possible combination.
- **Performance Trade-Off**: While random search does not guarantee finding the best possible combination, it can still find good solutions with fewer evaluations. In many cases, random search can find hyperparameter configurations that result in competitive model performance, especially when the number of relevant hyperparameters is large.
- **Faster Optimization**: Because random search samples randomly, it is computationally less expensive than grid search. This approach can quickly yield a good set of hyperparameters, especially in high-dimensional spaces.
- **Risk of Missing Optimal Configurations**: The main drawback of random search is that it may not explore the most promising areas of the hyperparameter space, especially in cases where some hyperparameters have a greater impact on model performance than others.

B. Comparison of Grid Search and Random Search

While grid search ensures a systematic evaluation of all combinations, random search tends to be more efficient, especially in high-dimensional spaces. The key differences between these two approaches are summarized as follows:

- **Search Coverage**:
  - **Grid Search**: Exhaustive, testing all possible combinations within the predefined grid.
  - **Random Search**: Randomly samples a subset of the hyperparameter space, providing a broader search in less time.
- **Computational Cost**:
  - **Grid Search**: Can be computationally expensive, especially for large search spaces. Time complexity increases exponentially with the number of hyperparameters.
  - **Random Search**: More computationally efficient, with linear time complexity in terms of the number of trials. It may not find the absolute best combination but provides a good balance between speed and performance.
- **Performance Guarantee**:
  - **Grid Search**: Guarantees finding the optimal solution (if the grid is sufficiently fine).
  - **Random Search**: Does not guarantee finding the best solution but often performs well in practice, especially for large search spaces.

C. Model Evaluation during Hyperparameter Optimization

Model evaluation plays a crucial role in ensuring that the hyperparameter optimization process does not overfit the

model to the training data. To evaluate the model's performance during hyperparameter tuning, several techniques can be employed, including:

- **Cross-Validation**:
  - Cross-validation is a robust technique that divides the dataset into multiple folds. The model is trained on some folds and tested on others, and this process is repeated several times to assess its generalization performance. This technique helps reduce the risk of overfitting by ensuring that the model performs well on various subsets of data.
  - **k-Fold Cross-Validation**: In this variation of cross-validation, the data is divided into k subsets, or "folds". The model is trained on $k-1$ folds and validated on the remaining fold, and this is repeated k times. The average performance score across all folds is used to evaluate the model.
- **Early Stopping**:
  - Early stopping is a regularization technique where training is halted if the performance on a validation set stops improving. This prevents the model from overfitting to the training data and can save computational resources by terminating unnecessary training steps.
- **Hyperparameter Tuning with Validation Set**:
  - During hyperparameter tuning, a separate validation set is often used to evaluate different hyperparameter combinations. The performance on this set provides an estimate of how well the model will perform on unseen data, helping to guide the hyperparameter selection process.

D. Practical Implications and Recommendations

The choice between grid search and random search depends on several factors:

- **Grid Search** is recommended when the hyperparameter space is relatively small and the cost of exhaustive search is feasible. It is particularly useful when a precise search is required, and when the hyperparameter interactions are known to be important.
- **Random Search** is preferred when the hyperparameter space is large, and the goal is to find a good solution quickly. It is often the more efficient choice when the number of hyperparameters is high and computational resources are limited.

Ultimately, both techniques have their place in the hyperparameter optimization pipeline. In practice, a combination of both may be used, starting with random search to narrow down the search space and followed by a more fine-grained grid search in the regions identified by random search.

VII. Conclusion

In this report, we explored the pivotal role of hyperparameter tuning and model evaluation in enhancing the performance of machine learning models. We compared two widely-used methods for hyperparameter optimization: grid search and random search, examining their strengths, weaknesses, and computational implications. Grid search systematically evaluates all possible combinations of hyperparameters within a predefined grid, ensuring that the optimal configuration is identified. In contrast, random search samples hyperparameters randomly from a given distribution, offering a more computationally efficient approach that is particularly valuable when dealing with large search spaces.

When determining the best approach for hyperparameter tuning, it's crucial to consider not only the theoretical benefits—both methods have their advantages in different contexts—but also the practical performance aspects associated with them. Grid search may provide a comprehensive solution when the search space is small, but its computational cost increases exponentially as the number of parameters grows. On the other hand, random search offers a faster alternative, especially when the hyperparameter space is vast and resources are limited.

Understanding the trade-offs between grid search and random search empowers practitioners to make informed decisions about hyperparameter optimization based on the complexity of the problem and available computational resources. By leveraging robust evaluation techniques like cross-validation, one can ensure that the chosen hyperparameter set leads to a model that generalizes well to unseen data, ultimately improving the efficiency of the training process and the accuracy of the final model. This comprehensive approach to hyperparameter tuning and model evaluation is essential for achieving optimal machine learning performance in practical applications.