Data Science and Machine Learning

Supervised Learning Project

Group 16:

Bruna Luisa do Amaral da Silva – 20231102
Miguel António Oliveira Figueiredo – 20231436
Paulo Manuel Gonçalves Oliveira Valente da Cruz – 20231980
Pedro Miguel Albuquerque Gomes – 20231126
Simona Costache - 20231983

June 2024

Index

Abstract

"QuestDice" is a leader in the board game industry, celebrated for innovative games that captivate players worldwide. With the success of our initial product reaching nearly 200,000 customers, we are preparing to launch a second game. The following report on QuestDice describes an in-depth analysis of data exploration, preprocessing, feature selection, and predictive modeling using supervised learning methods. Drawing from a dataset of historical customer interactions and their responses to a previous marketing campaign, a predictive model employing sophisticated algorithms was developed. This model aims to predict the likelihood of customer purchases in the upcoming second product launch of QuestDice. The strategic use of these data-driven techniques is intended to enhance the targeting of marketing efforts, thereby optimizing resource allocation and increasing the effectiveness of future campaigns.

To accomplish this, we will employ a variety of predictive modeling techniques. By leveraging these models, we plan to analyze a comprehensive dataset containing customer demographics, preferences, and feedback from our initial product launch. This approach will enable us to uncover patterns and trends that predict customer purchasing decisions, enhancing the accuracy of our forecasts and allowing us to better tailor our marketing efforts.

The objective is to create a binary variable indicating whether a customer will purchase our second product (1 for 'will buy', 0 for 'will not buy'). This approach will enable us to target our marketing resources more effectively, reducing costs and maximizing the return on investment for our new product launch.

1. Data Exploration

The dataset utilized for our analysis, “train.csv”, consists of **15,589 entries** spanning across 21 distinct columns, following the elimination of duplicate records. The dataset features a blend of different data types including one float, sixteen integer, and four object variables. Our primary focus, the **“Buy_product “** variable, serves as the target variable, identifying customer decisions to purchase (1) or not purchase (0) the board game.

Initial exploration involved a detailed assessment of data types, missing values, and descriptive statistics of the variables. Notably, the **“Year_Birth”** variable presented some missing entries and an error in data type as it should be represented as an integer rather than a float. Moreover, discrepancies in categorical data were found, such as 167 misclassified entries under **“Newsletter_Subscription”** where "y" should be corrected to "yes." Additionally, the variable **“S5_MaterialQuality”** reported a maximum value of 6, which is outside the expected range of 0 to 5, suggesting data entry errors. Analysis of the distribution of variables indicated high skewness in **“S11_RuleClarity”**(-0.75068) and **“S13_ReplayValue”**(-0.74513), so potential outliers or extreme values which may impact the model’s performance if not addressed.

The exploration also covered categorical features, revealing insights such as the predominant game genre preference for **'Strategy'** among 47.8%, followed closely by **'Party'** games. The Membership

attribute showed that a majority, approximately 69%, of the customers have a **Premium membership**, which could correlate with purchasing behavior. In our data exploration, we have identified that there are 195 missing values in the "**Year_Birth"** variable. This issue necessitates careful handling during the preprocessing phase to ensure the integrity and robustness of our predictive model.

Further analysis of the skewness across numerical variables, as seen in Figure 1., confirmed that most distributions were moderately skewed. This observation, particularly for variables like **"S1_GameplayExperience", "S2_GameMechanics",** and **"S6_ValueForPrice",** indicates that transformations or normalization may be required to align these variables more closely with the assumptions of the predictive models we plan to use.

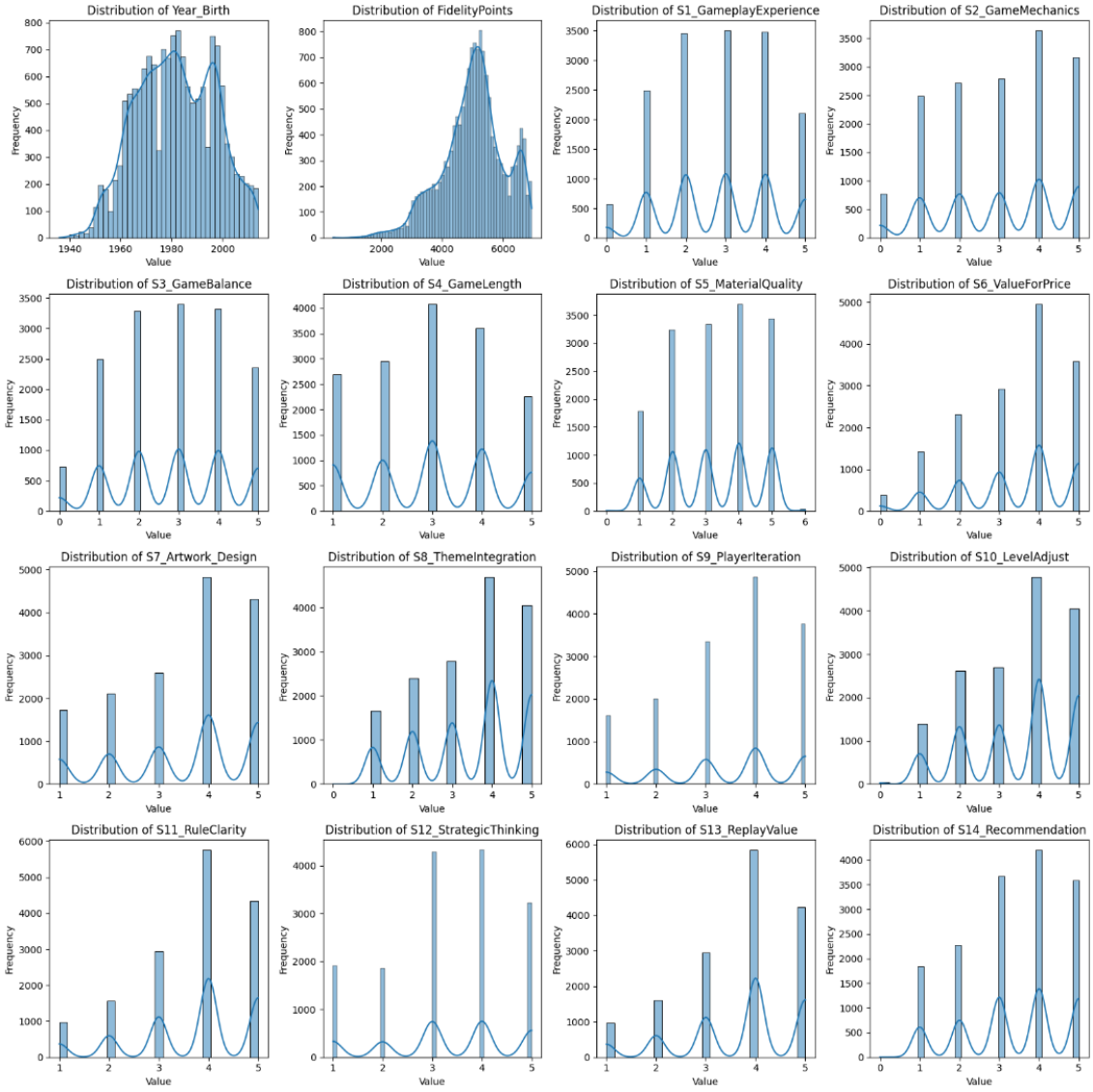

Figure 1. Histogram plots of the dataset variables

The survey responses highlighted through the histogram and boxplot visualizations indicated that customers highly value **"S11_RuleClarity", "S13_ReplayValue"**, and **"S9_PlayerIteration"**.

Conversely, aspects like **“S1_GameplayExperience”** and **“S3_GameBalance”** received lower ratings, suggesting areas for potential improvement in game development.

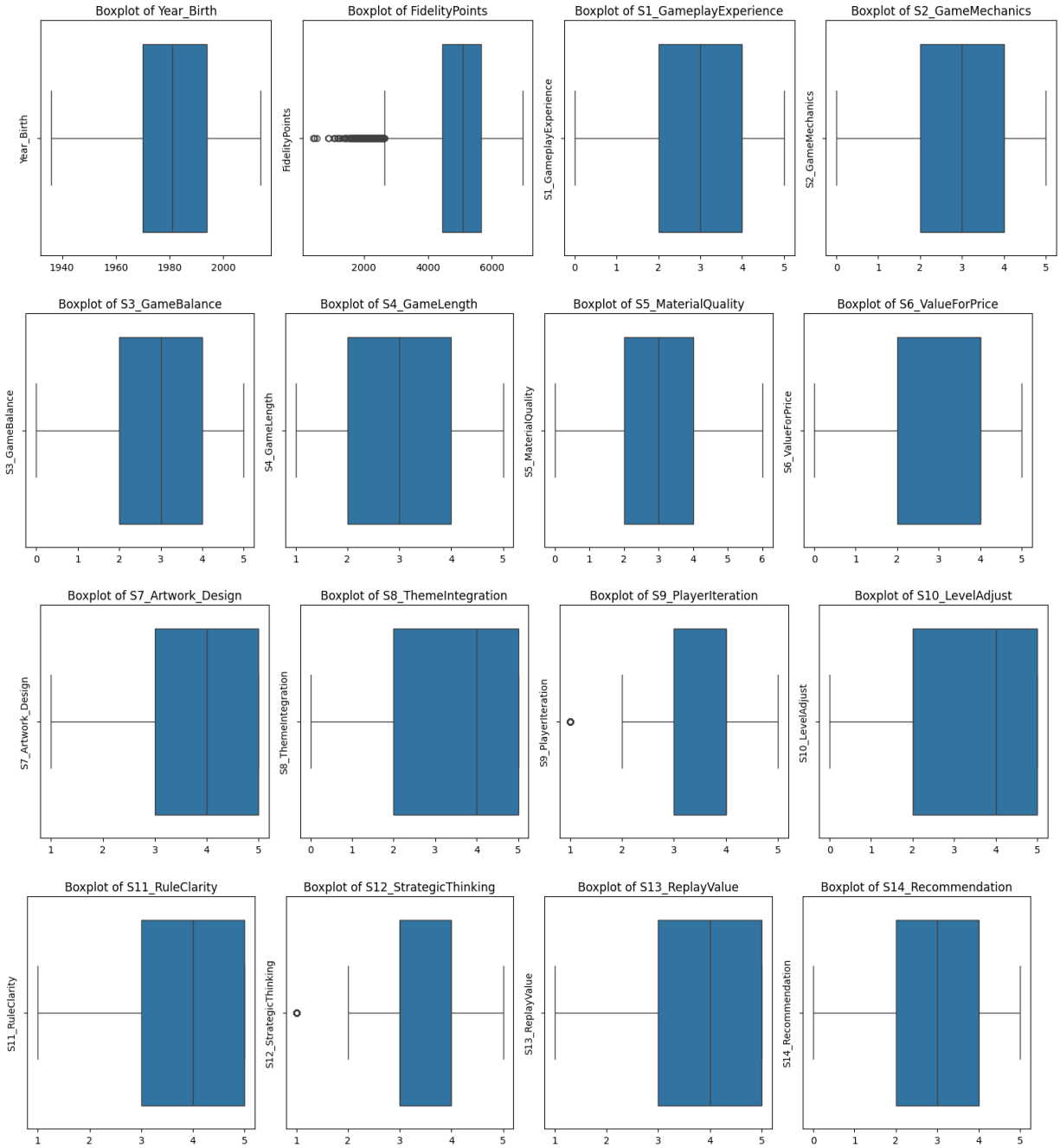

Figure 2. Boxplots of the variables of the dataset

The Spearman correlation matrix visualized the relationships between survey responses, revealing moderate to strong correlations among several variables, particularly those related to game enjoyment and engagement such as **“S7_Artwork_Design”** and **“S8_ThemeIntegration”.**

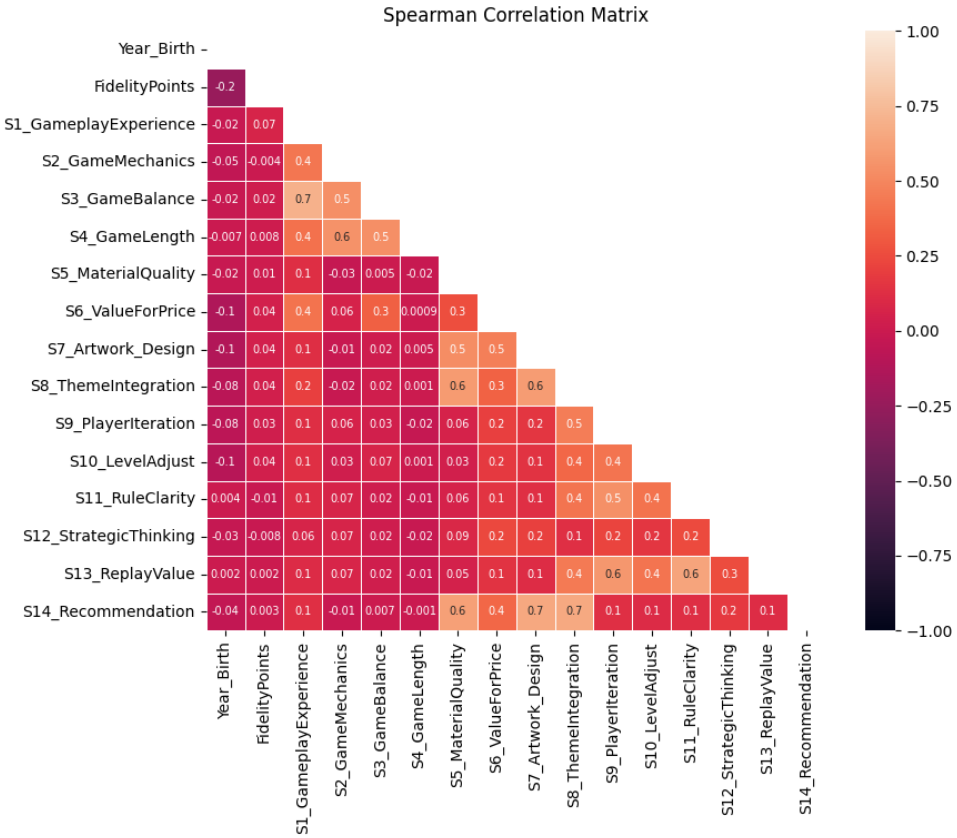

Figure 3. Spearman Correlation Matrix.

2. Data Preprocessing

2.1 Data Cleaning: Duplicates, Missing Values, Data Inconsistencies and Outliers

To ensure data accuracy and reliability, we began by **removing duplicates** from the dataset. Initially containing 15,589 records, this process reduced the total to 15,586, ensuring each entry represents a unique data point. The **"Year_Birth"** variable had missing entries, crucial for demographic analyses. Instead of discarding these rows, which constitute a significant portion of our dataset, we opted for median imputation. This method is robust to outliers and preserves the distribution of the dataset. To improve data integrity, we converted the **"Year_Birth"** variable from float to integer, aligning it with its logical data type and facilitating age-related calculations and groupings.

We corrected inconsistencies identified during the data audit phase:

- **S5_MaterialQuality**: Some entries incorrectly recorded as '6' were adjusted to '5', the nearest valid category within the expected range {0, 1, 2, 3, 4, 5}.

- **Newsletter_Subscription**: Standardized entries by converting 'y' to 'yes', thus homogenizing the data for clearer analysis and preventing classification errors.

In-depth analysis of variables like **“FidelityPoints”, “S9_PlayerIteration”,** and **“S12_StrategicThinking”** revealed outliers. For FidelityPoints, instead of removing the outliers, we considered applying a power transform later in the project to address any potential skewness in the data. For the latter two variables, despite initial appearances in boxplots, the distribution frequency justified retaining these values as they were not infrequent or extreme enough to distort overall trends.

2.2 Feature Engineering

New variables were engineered to add value to our analysis and model building:

- **Male**: Derived from customer titles, providing a binary classification of gender.
- **Age**: Calculated from Year_Birth, offering insights into customer demographics.
- **Global_Satisfaction**: A composite index created from all survey responses, intended to represent overall customer satisfaction.

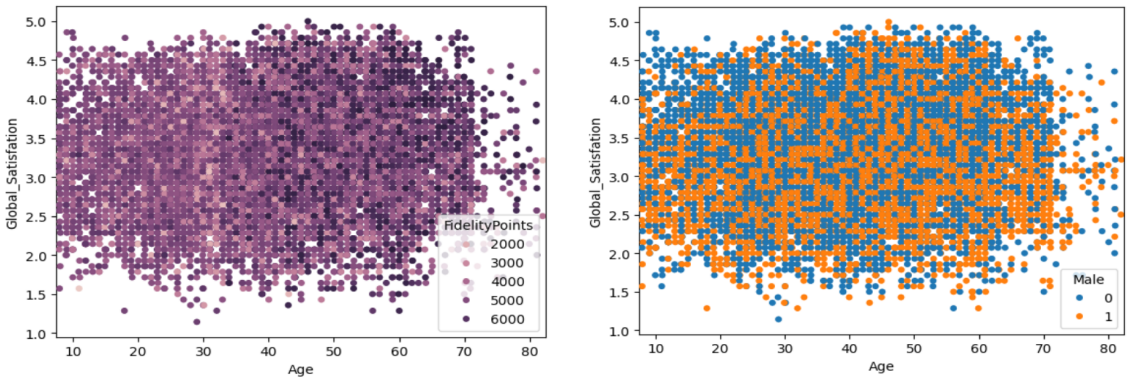

Figure 4. Global Satisfaction Variable

- **Age and Global Satisfaction by Gender**. Both males and females across various ages show a broad range of satisfaction levels, with no noticeable difference in satisfaction between genders.
- **Age and Global Satisfaction by Fidelity Points**. It suggests that higher fidelity points tend to correlate with the age variable, indicating that older customers generally have more fidelity points.

2.3 Data Transformation and Reduction

Categorical variables were transformed via one-hot encoding to fit into our predictive models. This includes converting categorical data like **“Membership”** and **“Newsletter_Subscription”** into a format suitable for regression analysis. We also performed significant variable reduction:

- Irrelevant Variables: We removed variables that offered no predictive power or were redundant, such as “Name” and “Year_Birth”.
- Encoding: After one-hot encoding on the variables “Membership” and “Newsletter_Subscription”, we removed the less informative binary variables

“Membership_Non-Premium” and “Newsletter_Subscription_no” to simplify the model input space.

- Power Transform: We applied square root transformation to the “FidelityPoints” variable to stabilize variance and make the data more normally distributed.

2.4 Feature Selection

Feature selection is essential to enhance the performance and interpretability of our models by focusing on the most relevant predictors. We employed a combination of filter, wrapper, and embedded methods to refine our feature set.

2.4.1 Filter Methods

Our initial approach involved applying filter methods to reduce redundancy and improve model efficiency:

- **Chi-Square Test**: This test evaluated the importance of categorical variables such as “Preferred_Game_Genre”, “Male”, “Membership_Premium” and “Newsletter_Subscription _yes”. Our findings across multiple dataset splits consistently highlighted these variables as significant predictors, confirming their importance and justifying their retention in the model.
- **ANOVA (Analysis of Variance)**: We employed ANOVA to scrutinize the effects of numerical features on our model, looking for features whose inclusion could significantly impact the model’s outcomes. This approach was critical in identifying features like "FidelityPoints", "Age", and "Global_Satisfaction" among others, each showing significant variance between groups. Features passing the ANOVA test demonstrated that they could contribute meaningfully to model differentiation and were thus retained.
- **Kendall**: Kendall's tau is a statistic used to measure the ordinal association between two measured quantities. It is a non-parametric test that assesses how well the relationship between two variables can be described using a monotonic function. The Kendall method results show that most variables are irrelevant for predicting the target variable, with 16 out of 17 numerical variables marked as "Discard." Only one variable was retained, S6_ValueForPrice with a significance value of 10. This indicates minimal correlation between the discarded variables and the target, suggesting they do not provide substantial predictive power and can be excluded to simplify the model.
- **Variance Thresholding**: We assessed the variance of numerical features to exclude any with insufficient variability. All features displayed adequate variance, thus none were excluded based on this criterion.
- **Spearman Correlation**: This analysis was particularly crucial for identifying features that exhibited high correlations with one another, potentially leading to multicollinearity in our predictive models. From our analysis, the Spearman correlation identified that the variables

"Male", "Membership_Premium", and "Newsletter_Subscription_yes" were highly correlated with other variables in the dataset.

2.4.2 Wrapper Methods

Wrapper methods assess the usefulness of subsets of features by directly measuring their impact on the performance of a predictive model. We utilized the following techniques:

- **Recursive Feature Elimination (RFE)**: helped in systematically reducing the feature set by iteratively removing the least impactful features, which sharpened our model's focus on influential predictors like "S2_GameMechanics", "S3_GameBalance", "S6_ValueForPrice", and "Global_Satisfaction". These features consistently improved model accuracy, underscoring their importance.
- **Sequential Feature Selection (SFS)**: This method complemented RFE by incrementally testing feature combinations to identify the optimal subset that maximized model performance. This method not only confirmed the significance of key features but also helped prevent overfitting and improved the transferability of our model.

2.4.3 Embedded Methods

Embedded methods integrate feature selection within the model training process, providing insights directly from the model:

- **Decision Trees**: This method helped us understand the importance of variables like "S3_GameBalance" and "S6_ValueForPrice" by measuring their information gain. This method clearly identified the variables that were most effective at splitting the data, thus enhancing model interpretability and decision-making.
- **Lasso Regression**: Effectively minimized the influence of less significant variables by reducing their coefficients to zero. For example, it identified less impactful variables within the set and eliminated them from the model, such as "S1_GameplayExperience" and "S4_GameLength", ensuring that only the most relevant features, like "Global_Satisfaction" and "FidelityPoints", were included.
- **Random Forest**: Employing this method across ten data splits has provided significant insights into the predictive power of various features. Key findings include: **Dominant Predictors** such as "S6_ValueForPrice", "S1_GameplayExperience", and "Global_Satisfaction" that emerged as the most influential features, consistently displaying high importance across all data splits. **Additional variables** like "S8_ThemeIntegration", "FidelityPoints", and "Age" also demonstrated considerable importance, albeit to a lesser extent than the dominant predictors. These features contribute valuable information that enhances the model's accuracy. Variables such as "Preferred_Game_Genre", "Membership_Premium", and "Newsletter_Subscription" showed **no impact** on the model's performance. Their lack of

significance suggests they could be candidates for removal to streamline the feature set and improve model efficiency.

2.4.4 Consolidation of Feature Selection Results

After running all the different methods, we built the following table, that was used a starting point for the selection of the variables to consider for training the models in the next phase.

	Filter Methods			Wrapper Methods		Embedded Methods		
Predictor	**Chi-Quare**	**ANOVA**	**Kendall**	**SFS**	**RFE**	**Lasso**	**DT**	**RF**
Preferred_Game_Genre	10	NA	NA	NA	NA	NA	NA	NA
Male	10	NA	NA	NA	Discard	Discard	Discard	NA
Membership_Premium	10	NA	NA	NA	Discard	Discard	Discard	NA
Newsletter_Subscription	10	NA	NA	NA	10	Discard	Discard	NA
FidelityPoints	NA	10	Discard	Keep	Discard	Discard	10	Keep
Age	NA	10	Discard	Discard	Discard	Discard	4	Keep
Global_Satisfation	NA	10	Discard	Keep	4	Discard	Discard	Keep
S1_GameplayExperience	NA	10	Discard	Keep	Discard	Discard	10	Keep
S2_GameMechanics	NA	10	Discard	Keep	4	Discard	Discard	Discard
S3_GameBalance	NA	10	Discard	Keep	4	Discard	Discard	Keep
S4_GameLength	NA	Discard	Discard	Keep	Discard	Discard	Discard	Discard
S5_MaterialQuality	NA	10	Discard	Discard	Discard	Discard	Discard	Discard
S6_ValueForPrice	NA	10	10	Keep	10	10	10	Keep
S7_Artwork_Design	NA	10	Discard	Keep	Discard	Discard	Discard	Keep
S8_ThemeIntegration	NA	10	Discard	Keep	6	10	10	Keep
S9_PlayerIteration	NA	10	Discard	Discard	2	10	Discard	Discard
S10_LevelAdjust	NA	10	Discard	Keep	6	10	5	Keep
S11_RuleClarity	NA	10	Discard	Keep	Discard	Discard	Discard	Discard
S12_StrategicThinking	NA	10	Discard	Keep	4	10	1	Discard
S13_ReplayValue	NA	10	Discard	Keep	Discard	Discard	Discard	Discard
S14_Recommendation	NA	10	Discard	Discard	Discard	10	Discard	Discard

Figure 5. Feature Selection – Summary table

By combining the insights from filter, wrapper, and embedded methods along with the MIC analysis, we established a robust set of features that includes both categorical and numerical variables critical for our predictive model. Important variables like “Preferred_Game_Genre”, “Male”, “Membership_Premium”, and “Newsletter_Subscription_yes” were affirmed across multiple feature selection techniques as essential.

3. Modelling and Fine-Tuning

3.1 Model Selection: First Approach

The process begins by selecting the relevant features for modeling. Three feature subsets are created: keep_data, all_data, and try_data.

- **keep_data:** subset contains 'Preferred_Game_Genre', 'Newsletter_Subscription_yes', 'S6_ValueForPrice', 'S8_ThemeIntegration', 'S10_LevelAdjust', and 'Buy_product'.
- **all_data:** subset includes all features from keep_data plus 'FidelityPoints', 'S1_GameplayExperience', 'S9_PlayerIteration', and 'S12_StrategicThinking'.
- **try_data:** subset includes all features from keep_data plus 'S9_PlayerIteration' and 'S12_StrategicThinking'.

Several algorithms are evaluated using these feature subsets, namely K-Nearest Neighbors (KNN), Decision Trees (DT) and Neural Networks (NN). The models are evaluated using 10-fold stratified cross-validation, applying MinMaxScaler for data scaling and calculating the F1-score for each fold.

The performance of the models is evaluated using all_data feature subset. The results are presented in the following table:

all_data

Model	Train F1-Score	Validation F1-Score
KNN	0.924	0.89
DT	0.862	0.86
NN	0.913	0.908

Table 1. Model Performance on all_data

The NN model achieves the highest validation F1-score of 0.908, followed closely by the KNN with a validation F1-score of 0.89. The models are also compared using the keep_data and try_data feature subsets. The results are as follows:

keep_data

Model	Train F1-Score	Validation F1-Score
KNN	0.915	0.878
DT	0.853	0.851
NN	0.905	0.902

Table 2. Model Performance on keep_data

try_data

Model	Train F1-Score	Validation F1-Score
KNN	0.917	0.88
DT	0.855	0.854
NN	0.908	0.905

Table 3. Model Performance on try_data

Based on the model evaluation results: all_data feature subset leads to better results on the validation dataset compared to keep_data and try_data. Considering overfitting, KNN has the highest difference between train and validation scores (0.034), followed by NN (0.005) and DT (0.002). NN appear to be the most promising model when considering both overfitting and F1-score performance.

3.1.1 Model Optimization

The KNN model is optimized by tuning the number of neighbors (n_neighbors) using values [1, 3, 5, 7, 9, 11]. The results show the train and validation F1-scores for each n_neighbors value.
The best value for n_neighbors, considering the validation F1-score, is 7. The final optimized KNN model is created with n_neighbors=7.

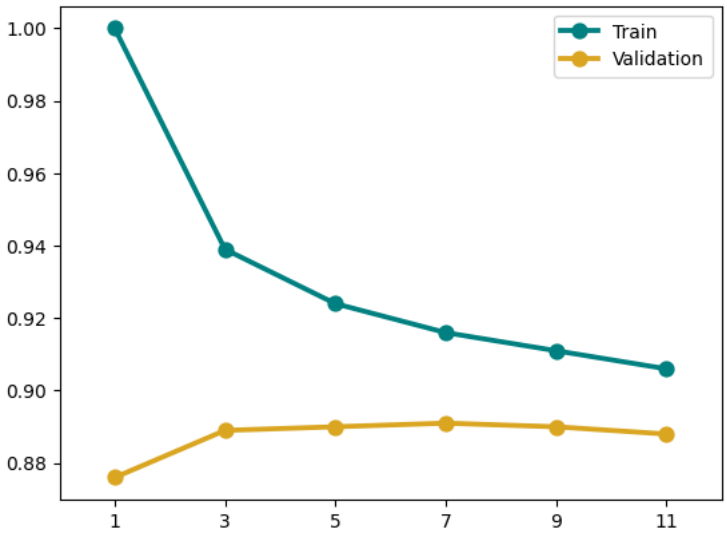

Figure 4. Compare results for KNN model optimization

The Decision Tree model is optimized by tuning the maximum depth (max_depth) using values [1, 2, 3, 4, 5, 6, 7, 8, 9]. The results visualize the impact of max_depth on train and validation F1-scores.
The best value for max_depth is 4. The final optimized Decision Tree model is created with max_depth=4 and random_state=99.

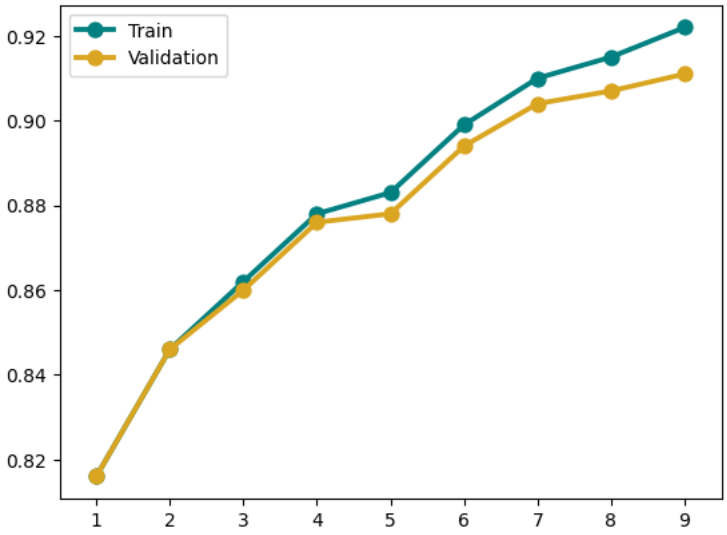

Figure 5. Compare results for DT model optimization

The Neural Network model is optimized by tuning the hidden layer sizes (hidden_layer_sizes) using values [(10), (100), (150), (200), (100, 100)]. The results visualize the impact of hidden layer sizes on train and validation F1-scores.

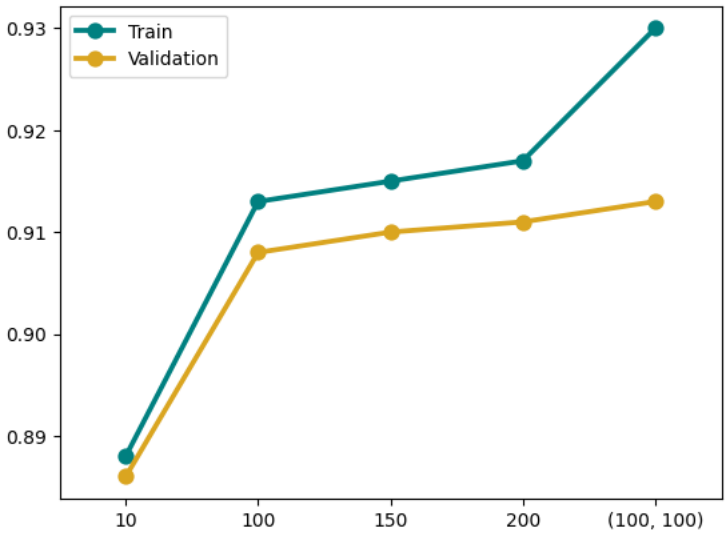

Figure 6. Compare results for NN model optimization

3.1.2 Random Search

We decided to apply the to the NN, since it was the most promising model. We used the RandomizedSearchCV to perform a random search over a specified hyperparameter space. The hyperparameter space includes hidden_layer_sizes, learning_rate_init, learning_rate, solver, and activation. The best hyperparameters found by the random search are:

- solver: 'adam'
- learning_rate_init: 0.01
- learning_rate: 'invscaling'
- hidden_layer_sizes: (320)
- activation: 'relu'

The best F1-score obtained through the random search is 0.9073.

3.1.3 Grid Search

After the random search, a grid search is conducted using GridSearchCV to further fine-tune the hyperparameters of the NN model. The best hyperparameters found by the grid search are:

- activation: 'relu'
- hidden_layer_sizes: (320)
- learning_rate: 'invscaling'
- learning_rate_init: 0.005
- solver: 'adam'

The best F1-score obtained through the grid search is 0.9096.

3.1.4 ROC Curve

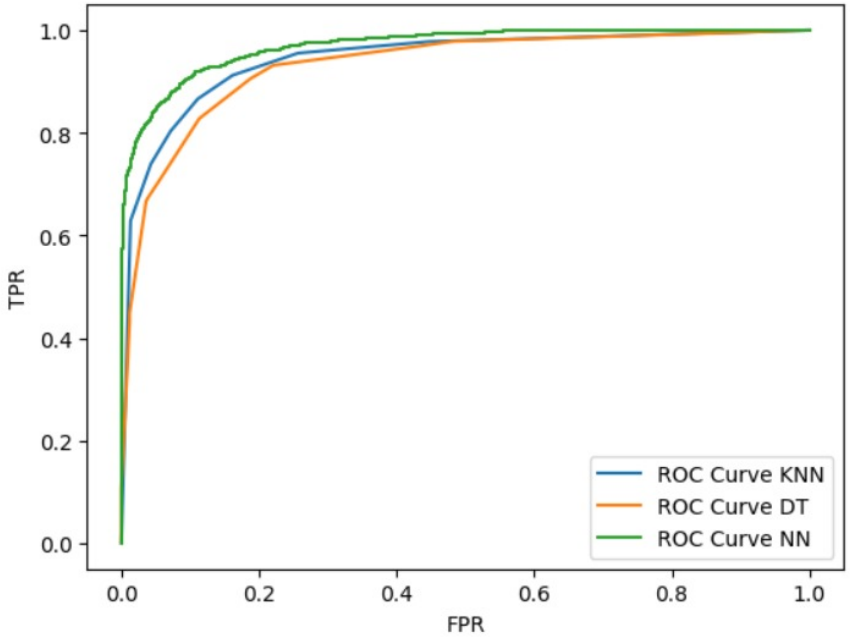

Figure 9. ROC Curve for KNN, DT and NN models.

AUC values:

0.9438767504072452 (KNN)

0.9309561586604165 (DT)

0.9696479840323339 (NN)

While considering the roc curve and the AUC, it seems that the NN is the one that achieves a better result. A higher AUC value generally indicates a better performing model. It means the model is better at distinguishing between the positive and negative classes across different threshold levels.

3.1.5 Precision Recall Curve

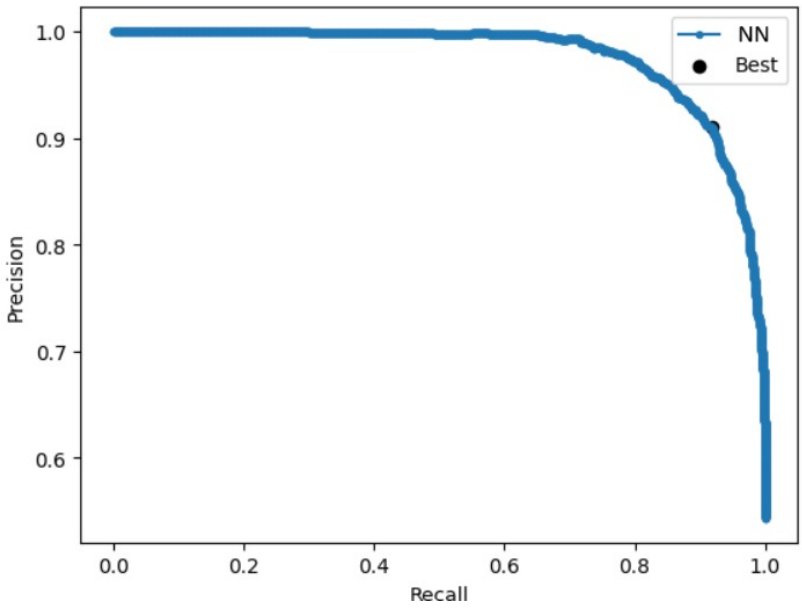

Figure 10. Precision Recall Curve.

Running the precision_recall_curve function we obtained for the NN the threshold of 0.427276, and we were able to improve the f1-score of our final model.

3.1.6 Kaggle Performance

We then deployed the NN model to our test dataset, uploaded the .CSV to Kaggle and obtained the initial performance of 0.89812. Considering that this was not a satisfactory result, this led us to revisit our variable selection strategy and experiment with including additional variables, and at the same time test with additional models.

3.2 Model Selection: Second Approach

We expanded our modeling approach to include more variables, and tested several combinations, and converged in a final dataset where we removed only 2 specifically “S4_GameLength” and “S5_MaterialQuality”.

In the modeling phase, we employed a diverse set of machine learning models to determine the best predictive performance. The additional models used were:

- Logistic Regression (LogisticRegression)
- Gradient Boosting (GradientBoostingClassifier)
- Random Forest (RandomForestClassifier)
- AdaBoost (AdaBoostClassifier)
- Bagging (BaggingClassifier)
- Extra Trees (ExtraTreesClassifier)
- Support Vector Classifier (SVC)

4. Performance Assessment

We conducted a thorough comparison of these models using both training and validation datasets. The performance results were as follows:

- **KNN:** Train 0.938, Validation 0.913
- **Decision Tree:** Train 0.862, Validation 0.862
- **Neural Network:** Train 0.954, Validation 0.938
- **Logistic Regression:** Train 0.845, Validation 0.845
- **Gradient Boosting:** Train 0.933, Validation 0.930
- **Random Forest:** Train 1.000, Validation 0.941
- **AdaBoost:** Train 0.904, Validation 0.902
- **Bagging:** Train 0.996, Validation 0.935
- **Extra Trees:** Train 1.000, Validation 0.942
- **SVC:** Train 0.939, Validation 0.930

The Random Forest and Extra Trees classifiers achieved perfect accuracy on the training set but demonstrated slight overfitting, as indicated by their validation scores. The Neural Network and its variant (NN_MLP) also performed exceptionally well on both datasets. Overall, the best-performing models in terms of validation accuracy were Random Forest (RF), Extra Trees (ET), and Neural Networks, making them the top candidates for our final predictive model. This comprehensive evaluation ensures that we select a robust model with strong generalization capabilities.

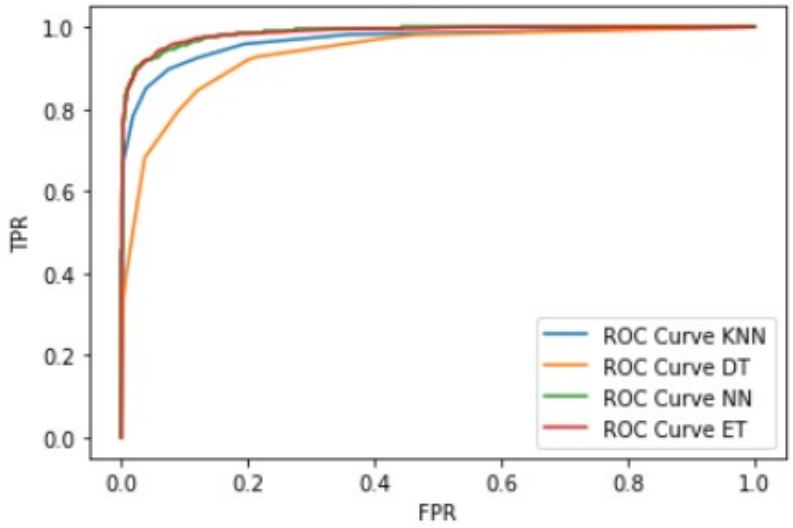

Figure 11. ROC Curve for KNN, DT, NN and ET models.

AUC values:

0.9665302644916947 (KNN)

0.9353103356778752 (DT)

0.9867598679096262 (NN)
0.986676741356186 (ET)

Again, when considering the ROC curve and de AUC values, the NN model seems to be the way to go. With the Precision Recall Curve we obtained the threshold of 0.369006.

The final optimized Neural Network model is trained on the train using the selected features. The test dataset (test.csv) is loaded and undergoes the same preprocessing steps as the training data, including feature selection, one-hot encoding, and scaling.

The final model was applied to the test dataset to predict customer purchasing behaviour. After uploaded again the .CSV to Kaggle we obtained the final performance of 0.94825 that we consider satisfactory.

4.1 Summary of Predictions

- **Total Customers Analyzed**: 5,195
- **Predicted Buyers (Buy_product = 1)**: 2,497 (48.07%)
- **Predicted Non-Buyers (Buy_product = 0)**: 2,698 (51.93%)

4.2 Key Insights

- **Distribution of Predictions**:

The predictions are almost evenly split, with a slight majority of customers predicted not to buy the product. This indicates a balanced dataset with a near-equal chance of a customer being classified as a buyer or non-buyer.

- **Potential Buyer Segments:**

A significant number of customers (48.07%) are predicted to make a purchase, suggesting a strong potential for market segment.

5. Conclusions

This project aimed to enhance the efficiency of QuestDice's marketing strategy for the second product launch by developing predictive models to identify potential buyers. By analyzing a dataset of 15,589 initial responses, we targeted a subset (test dataset) of 5,195 customers, predicting 2,497 buyers (48.07%) and 2,698 non-buyers (51.93%).

Key Points:

- Predictive Accuracy: With the Kaggle performance of 0.94825, our models accurately distinguished between likely buyers and non-buyers, enabling a more focused marketing approach.

- Resource Optimization: By targeting the predicted 48.07% likely buyers, QuestDice can significantly reduce advertising costs. For example, advertising to 2,497 predicted buyers at €3 per customer would cost €7,491, compared to advertising to all 5,195 customers, which would cost €15,585.

- Financial Impact: The targeted approach is expected to increase revenue. With a profit of €8 per sale, targeting the 2,497 predicted buyers could yield a **potential** profit of €19,976 (2,497 * €8), resulting in a net profit of €12,485 (€19,976 - €7,491). This is more efficient than the previous broad approach, which resulted in a profit of €21,049 from a larger advertising spend.

In conclusion, the application of these predictive models positions QuestDice to implement a more cost-effective and impactful marketing strategy, driving higher revenue and fostering sustainable growth.

Annexes

Annex 1: MIC (Mutual Information Criterion) Analysis

We also employed the Mutual Information Criterion (MIC) to quantify the dependency between variables. MIC captures both linear and non-linear relationships, providing a more comprehensive understanding of feature importance.

- Data Preparation: The dataset was prepared by encoding categorical variables using one-hot encoding and ensuring all features were in a suitable format for analysis.
- MIC Calculation: We calculated the mutual information scores for all features with respect to the target variable. This involved using both mutual_info_classif for classification tasks and mutual_info_regression for regression tasks.

The MIC analysis highlighted the following top features: "S6_ValueForPrice", "Global_Satisfaction", "S1_GameplayExperience", "S8_ThemeIntegration", "S7_Artwork_Design".

These features demonstrated the highest mutual information scores, indicating a strong dependency with the target variable. Consequently, they were selected for further model training and validation.

Feature	MI Score
S6_ValueForPrice	0.231214
Global_Satisfation	0.149697
S1_GameplayExperience	0.136971
S8_ThemeIntegration	0.111677
S7_Artwork_Design	0.106903
S9_PlayerIteration	0.074869
S14_Recommendation	0.069988
S10_LevelAdjust	0.067941
S13_ReplayValue	0.049181
Newsletter_Subscription_yes	0.047655
Preferred_Game_Genre_Strategy	0.044257
S12_StrategicThinking	0.041743
S3_GameBalance	0.040519
S11_RuleClarity	0.039369
Preferred_Game_Genre_Party	0.031174
Age	0.030149
sqrt_FidelityPoints	0.025312
Male	0.025099
Membership_Premium	0.006761
S2_GameMechanics	0.000000

Figure 11. MIC – Feature Importance for our dataset.

Annex 2: Other predictive Models (excluding Ensembles)

The models utilized include Logistic Regression, Neural Network (MLPClassifier), and Support Vector Machine (SVC).

Logistic Regression

A statistical method for binary classification that models the probability of a binary response based on one or more predictor variables. It applies the logistic function to the linear combination of the input features, transforming the output to fall between 0 and 1, which can be interpreted as a probability.

The primary hyperparameter used here is the `random_state`, which is set to 1 to maintain consistency across different runs. Logistic Regression is straightforward and interpretable, making it a good baseline model.

Neural Network (MLPClassifier)

A Multi-Layer Perceptron (MLP) is a type of artificial neural network consisting of at least three layers: an input layer, one or more hidden layers, and an output layer. MLPs can capture complex patterns in data through non-linear activation functions and backpropagation to minimize the error.

Hyperparameters:

- hidden_layer_sizes: Defines the number of neurons in the hidden layer.
- activation: Specifies the activation function used for hidden layers.
- solver: An optimization algorithm for weight updates.
- random_state: Ensures reproducibility.

The chosen hyperparameters provide a balance between model complexity and training time. The “Relu” activation function and “Adam” solver are widely used due to their efficiency and ability to handle large datasets effectively.

Support Vector Machine (SVC)

A powerful classification method that finds the hyperplane which maximally separates the data points of different classes. In cases where the data is not linearly separable, SVM can use kernel functions to project the data into higher-dimensional spaces where a linear separator may exist.

Hyperparameters:

- kernel: Uses the Radial Basis Function (RBF) kernel to handle non-linear relationships.
- C: Regularization parameter that controls the trade-off between achieving a low training error and a low testing error.
- gamma: Defines the influence of a single training example.

The RBF kernel is selected for its flexibility in capturing non-linear patterns. The regularization parameter C and gamma are set to default values, which work well for a variety of datasets.

Comparative Analysis

In comparing Logistic Regression, Neural Network (MLPClassifier), and Support Vector Machine (SVC) models, we explore their distinct characteristics and applications.

Logistic Regression is valued for its simplicity, interpretability, and computational efficiency. It performs optimally on datasets that are linearly separable, making it suitable for foundational modeling tasks and scenarios where understanding the impact of predictors is crucial. However, its efficacy diminishes when confronted with complex, non-linear relationships within data.

The MLPClassifier, or Neural Network, offers robust capabilities in modeling intricate and non-linear relationships due to its flexible architecture. It excels in tasks that demand high-dimensional data analysis and pattern recognition, such as image recognition and complex data analytics. Nevertheless, leveraging its capabilities requires substantial computational resources, and without proper regularization, neural networks are prone to overfitting and lack the interpretability of simpler models like Logistic Regression.

On the other hand, the Support Vector Machine (SVC) stands out in managing high-dimensional data spaces and demonstrating resilience against overfitting, especially when utilizing appropriate kernel functions. It finds extensive application in fields requiring sophisticated data separability, such as text classification and bioinformatics. Despite its effectiveness, SVC can be computationally intensive, particularly with larger datasets, and tends to be less interpretable compared to simpler models.

Each of these models—Logistic Regression, MLPClassifier, and SVC—presents unique strengths and limitations, catering to different aspects of data analysis and predictive modeling. Their suitability varies based on the nature of the data, the complexity of relationships within it, and the specific requirements of the application domain.

Annex 3: Ensamble Models

Ensemble techniques combine multiple individual models to improve predictive performance, leveraging the strengths of various algorithms to mitigate their weaknesses. In this analysis, we explore three prominent ensemble methods available in the sklearn library: Gradient Boosting, Random Forest, and AdaBoost.

Gradient Boosting

Gradient Boosting builds an ensemble of decision trees sequentially, where each subsequent tree corrects errors made by the previous ones. It optimizes a loss function by adding weak learners (shallow trees typically) iteratively. Gradient Boosting is robust against overfitting due to its sequential nature and is effective in capturing complex relationships in data. Hyperparameters such as learning rate, tree depth, and number of estimators are crucial in controlling model complexity and generalization ability.

Random Forest

Random Forest constructs an ensemble of decision trees, where each tree is trained independently using a subset of the data and features (bagging). It aggregates predictions across multiple trees to improve accuracy and generalization. Random Forest is less prone to overfitting compared to individual decision trees and performs well on large datasets with high-dimensional features. Key hyperparameters include the number of trees (n_estimators), tree depth, and feature subset size.

AdaBoost

AdaBoost (Adaptive Boosting) works by sequentially fitting models to correctly classify instances that previous models have misclassified. It assigns higher weights to misclassified data points to prioritize learning from errors. AdaBoost is effective in enhancing the performance of weak learners and is particularly useful in scenarios with clean, structured data. Hyperparameters like the number of estimators and learning rate influence its performance and convergence.

Comparative Analysis

In comparing Gradient Boosting, Random Forest, and AdaBoost, each technique offers unique strengths suited to different data characteristics and modeling objectives.

Gradient Boosting excels in handling complex relationships and is robust against overfitting, making it suitable for tasks where accuracy and model interpretability are crucial. Random Forest, with its ensemble of independently trained trees, balances bias and variance well, performing

admirably on high-dimensional datasets. AdaBoost, focusing on iterative learning from errors, enhances the performance of weak learners, particularly in structured datasets.

Each ensemble method - Gradient Boosting, Random Forest, and AdaBoost - presents distinct advantages and considerations. The choice among them hinges on factors such as dataset size, complexity, interpretability needs, and the trade-off between bias and variance.

References:

[1] https://www.ibm.com/topics/random-forest

[2] https://scikit-learn.org/stable/modules/ensemble

[3] https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/

[4] https://aws.amazon.com/what-is/boosting/

[5] https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/

[6] https://scikit-learn.org/stable/modules/svm.html

[7] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[8] https://pub.towardsai.net/gaussian-naive-bayes-explained-and-hands-on-with-scikit-learn-4183b8cb0e4c

[9] https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes

[10] https://scikit-learn.org/stable/modules/sgd.html#sgd

[11] https://michael-fuchs-python.netlify.app/2019/11/11/introduction-to-sgd-classifier/

[12] https://builtin.com/data-science/gradient-descent

[13] https://scikit-learn.org/stable/modules/feature_selection.html#rfe