**Report on the Random Forest Algorithm Implementation**

**Introduction**

In this report, we discuss the implementation of the Random Forest algorithm to predict survival outcomes in the Titanic dataset. Random Forest is an ensemble learning method that constructs multiple decision trees and combines their predictions to enhance performance and reduce overfitting. This model is widely used in classification tasks due to its ability to handle large datasets, its robustness against overfitting, and its capacity for handling missing values.

**Motivation**

The Titanic dataset offers an excellent opportunity to explore machine learning models in a real-world context. This dataset contains information about passengers aboard the Titanic, such as age, class, gender, and survival status, which makes it ideal for classification tasks. By using Random Forest, we aim to predict whether a passenger survived based on these features.

The motivation for using Random Forest specifically stems from:

**Ensemble learning:** Random Forest builds multiple decision trees, increasing predictive power by reducing variance and bias.

**Feature importance:** Random Forest can be used to identify important features influencing the survival of passengers, offering valuable insights.

**Handling of missing data:** The algorithm works well with missing values, which makes it particularly useful in real-world scenarios where data is often incomplete.

**Theory Behind the Random Forest Algorithm**

Random Forest is an ensemble learning method, meaning it combines multiple models (in this case, decision trees) to make more accurate predictions.

**Bootstrapping:** A Random Forest model builds multiple decision trees by sampling the training data with replacement. This process, known as bootstrapping, ensures that each tree is trained on a slightly different subset of the data.

**Random Feature Selection:** At each decision node, instead of considering all features, Random Forest selects a random subset of features to decide on the split. This introduces diversity among the trees and reduces correlation, which in turn improves the generalization of the model.

**Majority Voting:** For classification problems, Random Forest uses majority voting to combine predictions from all the individual trees. The prediction for a data point is the class that is predicted by the majority of the trees.

**Numerical Implementation of Random Forest**

In this implementation, follow these key steps:

Step 1: **Data Preprocessing**

**Feature Selection:** We selected relevant features for prediction, including Survived, Pclass, Sex, Age, SibSp, Parch, and Fare. We removed irrelevant columns, like Name, Ticket, and Cabin, which are not helpful in our model.

**Handling Missing Values**: The dataset had some missing values in the Age column, which were removed using the drop\_na() function. Alternatively, missing values could be imputed, but for simplicity, we chose to exclude rows with missing data.

**Data Transformation:** We converted the Survived and Sex columns into factor variables since the target variable is categorical, and Sex is an important feature for classification.

Step 2: **Bootstrap Sampling**

We implemented a function, bootstrap\_sample(), which generates a bootstrap sample of the dataset. This sample is drawn with replacement, ensuring that some observations may appear multiple times while others may be excluded.

Step 3: **Building Decision Trees**

For each bootstrap sample, we build a decision tree. The function build\_tree() selects a random subset of features (mtry) for splitting the data at each node. This randomness reduces correlation between individual trees and is key to the strength of Random Forest.

Step 4: **Combining Trees in a Random Forest**

We train a set of decision trees by repeating the bootstrapping and tree-building steps n\_trees times. The train\_random\_forest() function stores these trees in a list.

Step 5: **Prediction via Majority Voting**

Once the Random Forest is trained, we predict the class for each observation in the test set by aggregating the predictions of all individual trees. The function predict\_random\_forest() uses majority voting to determine the final class for each observation.

Step 6: **Evaluation**

We evaluated the model's performance using:

**Confusion Matrix:** To assess the number of true positives, true negatives, false positives, and false negatives.

**Accuracy:** The proportion of correct predictions.

**ROC Curve:** To measure the true positive rate versus the false positive rate at various classification thresholds.

**Discussion**

Based on the results from the Titanic dataset, the Random Forest model achieved an accuracy of 78.99%, which is a solid result for this classification task. Let's dive into the evaluation metrics and discuss the model's performance in detail.

**Confusion Matrix Analysis:**

The confusion matrix for the model's predictions is as follows:

Predicted Survived (0) Predicted Not Survived (1)

Actual Survived (0) 408 134

Actual Not Survived (1) 16 156

True Positives (TP): 156 (These are the passengers correctly predicted to have survived)

True Negatives (TN): 408 (These are the passengers correctly predicted to have not survived)

False Positives (FP): 134 (These are the passengers incorrectly predicted to have survived)

False Negatives (FN): 16 (These are the passengers incorrectly predicted to have not survived)

**Key Performance Metrics:**

**Accuracy:** The model's accuracy is 78.99%, which means the model correctly predicted the survival status for approximately 79% of the passengers.

**Sensitivity** (Recall): The sensitivity, which is the proportion of actual survivors correctly identified, is very high at 96.23%. This indicates that the model is very good at detecting survivors, which is critical in scenarios where we want to minimize false negatives (i.e., ensuring survivors are not missed).

**Specificity:** The specificity, or the proportion of non-survivors correctly identified, is 53.79%. While this is lower than the sensitivity, it suggests that the model is more focused on detecting survivors rather than non-survivors. This can be acceptable depending on the application's requirements, but improving specificity may be valuable if we need a more balanced model.

**Positive Predictive Value** (PPV): The PPV, or precision, is 75.28%. This means that when the model predicts a passenger survived, there's a 75% chance that the prediction is correct.

**Negative Predictive Value** (NPV): The NPV is 90.70%, which indicates that when the model predicts a passenger did not survive, it is correct 90.7% of the time.

**Balanced Accuracy:** The balanced accuracy, which takes into account both sensitivity and specificity, is 75.01%. This score gives us an overall understanding of how well the model is performing across both classes.

**Model Insights:**

No Information Rate (NIR): The No Information Rate is 59.38%, meaning that if we were to predict the majority class (non-survived), we would get this accuracy just by chance. The fact that our model's accuracy is significantly higher than this value (78.99%) indicates that the Random Forest model is doing a good job at learning from the data and is not just guessing the majority class.

**Kappa:** The Kappa statistic is 0.5346, which indicates moderate agreement between the predicted and actual values beyond what would be expected by chance.

**Mcnemar's Test**: The p-value from McNemar's test is less than 2.2e-16, suggesting that the discrepancies between predicted and actual values are statistically significant.

**Discussion Points:**

Sensitivity vs Specificity: The high sensitivity (96.23%) indicates that the model is good at identifying passengers who survived, which is important in survival analysis tasks where we want to avoid missing any survivors. However, the lower specificity (53.79%) suggests that the model may not be as effective at identifying passengers who did not survive. Balancing these metrics may improve the model’s performance for both groups. Adjusting the decision threshold or tuning hyperparameters might help achieve better specificity.

**Model Bias:** The model is somewhat biased toward predicting survival, as evidenced by the higher sensitivity compared to specificity. This is likely due to the dataset's class imbalance (around 59% of passengers did not survive). To address this, techniques like class weighting or resampling could be employed to balance the training data and reduce bias towards the majority class.

**Feature Importance:** Random Forest inherently provides a measure of feature importance, which can be used to identify the most influential variables in predicting survival. Based on this, we could further explore the impact of variables like Sex, Pclass, Age, and Fare, which are known to be important factors in survival analysis.

**Conclusion:**

Overall, the Random Forest model is effective in predicting Titanic survival outcomes, particularly in detecting survivors (high sensitivity). However, the model could benefit from balancing sensitivity and specificity, possibly through further tuning of hyperparameters. The performance is solid, with an accuracy of 79%, and additional improvements could come from exploring more sophisticated techniques like hyperparameter optimization or ensemble methods (e.g., Gradient Boosting). Additionally, further exploration of the feature importance could lead to insights into which passenger characteristics most strongly predict survival, providing more interpretability in the context of survival analysis.

**Reflection and Generalization**

Random Forest is a highly effective and flexible algorithm for classification tasks, especially when there are many features and missing values. The implementation presented here can be generalized to other binary classification problems where interpretability is important, but predictive performance is the priority.  
  
  
  
**Script:**  
  




