# AntiFungalML: A Machine learning based prediction of anti-fungal activity in small molecules

# Author: Gopal Srivastava

# Course: BIOL 4800

# Course Adviser: Dr. Jeremy Brown

# Submission Date: May 8, 2024

**Abstract**

AntiFungalML is a machine learning-based tool for predicting the anti-fungal activity of small molecules. The tool uses a binary classification algorithm that takes SDF files as input and returns prediction probabilities for the input compound to have anti-fungal properties. The data sets used to generate the classification algorithm were collected from various sources, including PubChem, ChEMBL, and ceuMassMediator. The positive data set consisted of 544 unique anti-fungal compounds, while the negative data set consisted of 544 unique compounds consisting of secondary metabolites collected from MeFSAT. The tool uses Random Forest as the machine learning algorithm to train models, which outperformed SVM, kNN and CART methods with an ROC value of **0.97**, sensitivity value of **0.89**, and specificity value of **0.87**. The tool also performs feature selection using a mean decrease in accuracy test and optimizes the number of features for training. The best model was found to be top 30 features with mtry 6, which was used for blind testing and achieved a balanced accuracy of **0.8349**, sensitivity of **0.7248**, specificity of **0.9450**, precision of **0.9294**, recall of **0.7248**, and F1 score of **0.8144**.

## Pre-requisites

* Python3
* Pandas
* Rdkit
* ggplot2
* R v4.3

## 1. Data Curation

The data sets used to generate the classification algorithm were collected from following sources

### Positive Set:

* The anti-fungal compounds were collected from following databases:
  + PubChem (<https://pubchem.ncbi.nlm.nih.gov>)
  + ChEMBL (<https://www.ebi.ac.uk/chembl/>)
  + ceuMassMediator (<https://github.com/albertogilf/ceuMassMediator>)
* We collected 544 unique anti-fungal compounds from the aforementioned databases.
* The uniqueness of the compounds were ensured by Similarity search using Rdkit.

The all vs all similarity search can be performed using following python script

python Scripts/DataPreperation/4.RdKitSimilaritySearch.py -infile

The input file consisted of CompoundID and SMILES as instance separated by Tab.

**Example:**

**Table 1**: The table below represents the three examples taken from the input file for Canonical SMILES generation python script.

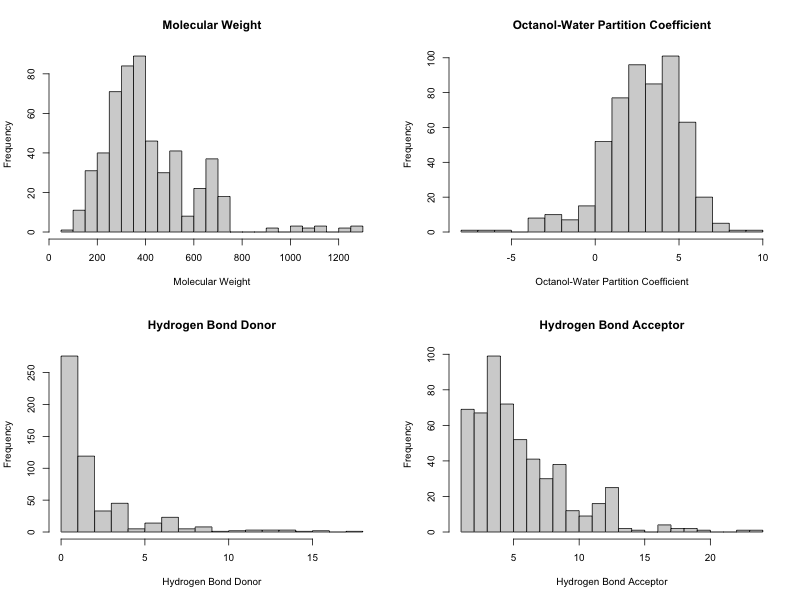
|  |  |
| --- | --- |
| CompoundID | SMILES |
| CHEMBL2104873 | CCCNCC(O)COc1ccc2c(=O)cc(-c3ccccc3)oc2c1.O=C(O)/C=C\C(=O)O |
| CHEMBL2105183 | CC1CC(=O)NN=C1c1ccc(NCC(C)(C)NCC(O)COc2cc(Cl)ccc2C#N)cc1 |
| CHEMBL2104058 | COC(=O)Nc1cc(N2CC=CCC2)nc2nc(=O)on12 |

# The SMILES can be converted to Canonical SMILES using following command line  
# The script also calculates Lipinski Rule of 4  
python Scripts/DataPreperation/2.RdKitLipinskiandCanonicalSmiles.py -infile Path/Filenames -outfile Path/FinalProcessedSmilesCompound.tsv

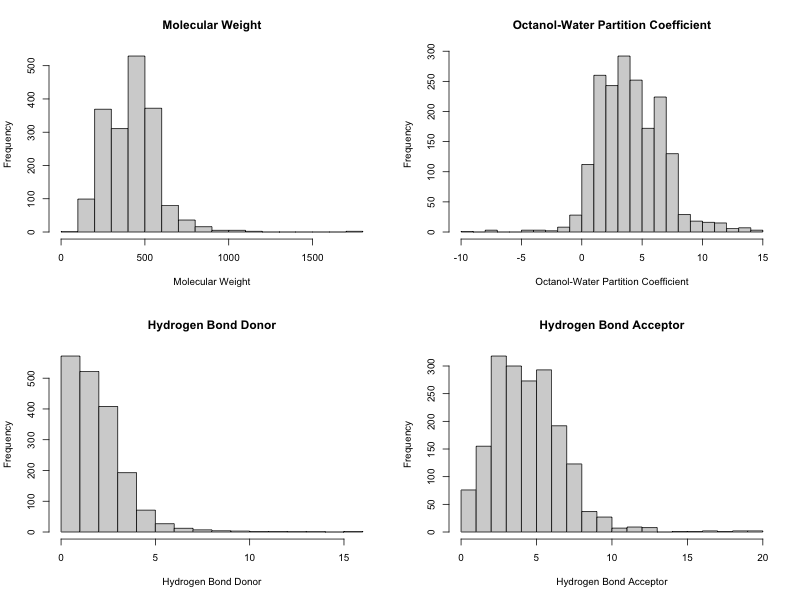
The Lipinski Rule of 4 were calculated to ensure that the compounds in the positive data set follow at least 3 out of 4 rules. It has been shown by (Bickerton et al. 2012) that the drugs following at least 3 out of 4 rules show best druggability.

To plot Lipinski Rule of 4 following process can be followed.

# To calculate Lipinski Rule of 4 for postive dataset  
Rscript Scripts/DataPreperation/3.PlotLipinskiProperties.R DataCuration/Positivedataset/FinalProcessedSmilesCompound.tsv Figures/PositiveMoleculeLipinskiProperties.png  
  
#To calculate Lipinski Rule of 4 for negative dataset  
Rscript Scripts/DataPreperation/3.PlotLipinskiProperties.R DataCuration/Negativedataset/FinalProcessedSmilesCompound.tsv Figures/NegativeMoleculeLipinskiProperties.png



**Figure 1:** Distribution of molecular properties (Lipinski Rule of Four) for compounds in positive data.



**Figure 2:** Distribution of molecular properties (Lipinski Rule of Four) for compounds in negative data.

### Negative Set:

* The negative set was constructed by collecting the compounds that did not show anti-fungal activity from <https://cb.imsc.res.in/mefsat/home>. (Vivek-Ananth et al. 2021)
* There were a total of **1833** compounds from the MeFAST database. The SMILES of these compounds were converted to Canonical SMILES before further calculation of Lipinski Properties of Four.

### Structural Similarity between Positive and Negative Sets

* To ensure least structural overlap between the positive and negative sets, a Tanimoto Similarity Search was performed using RdKit.
* To perform the similarity search between positive and negative sets

#### **Step1**

# Remove the header from negative set SMILES file and then contatenate the positive and negative sets to create new total SMILES dataset  
# Run these commands on terminal.  
sed 1d Negativedataset/FinalProcessedSmilesCompound.tsv | cut -f2- > Negativedataset/FinalProcessedSmilesCompound.tsv1   
cat Positivedataset/FinalProcessedSmilesCompound.tsv Negativedataset/FinalProcessedSmilesCompound.tsv1 > TotalSMILESPositiveNegative.tsv   
grep -v "Not a valid SMILES" TotalSMILESPositiveNegative.tsv > TotalSMILESPositiveNegative.tsv1  
mv TotalSMILESPositiveNegative.tsv1 TotalSMILESPositiveNegative.tsv   
# TotalSMILESPositiveNegative.tsv contains all the SMILES from both positive and negative sets.

#### **Step2**

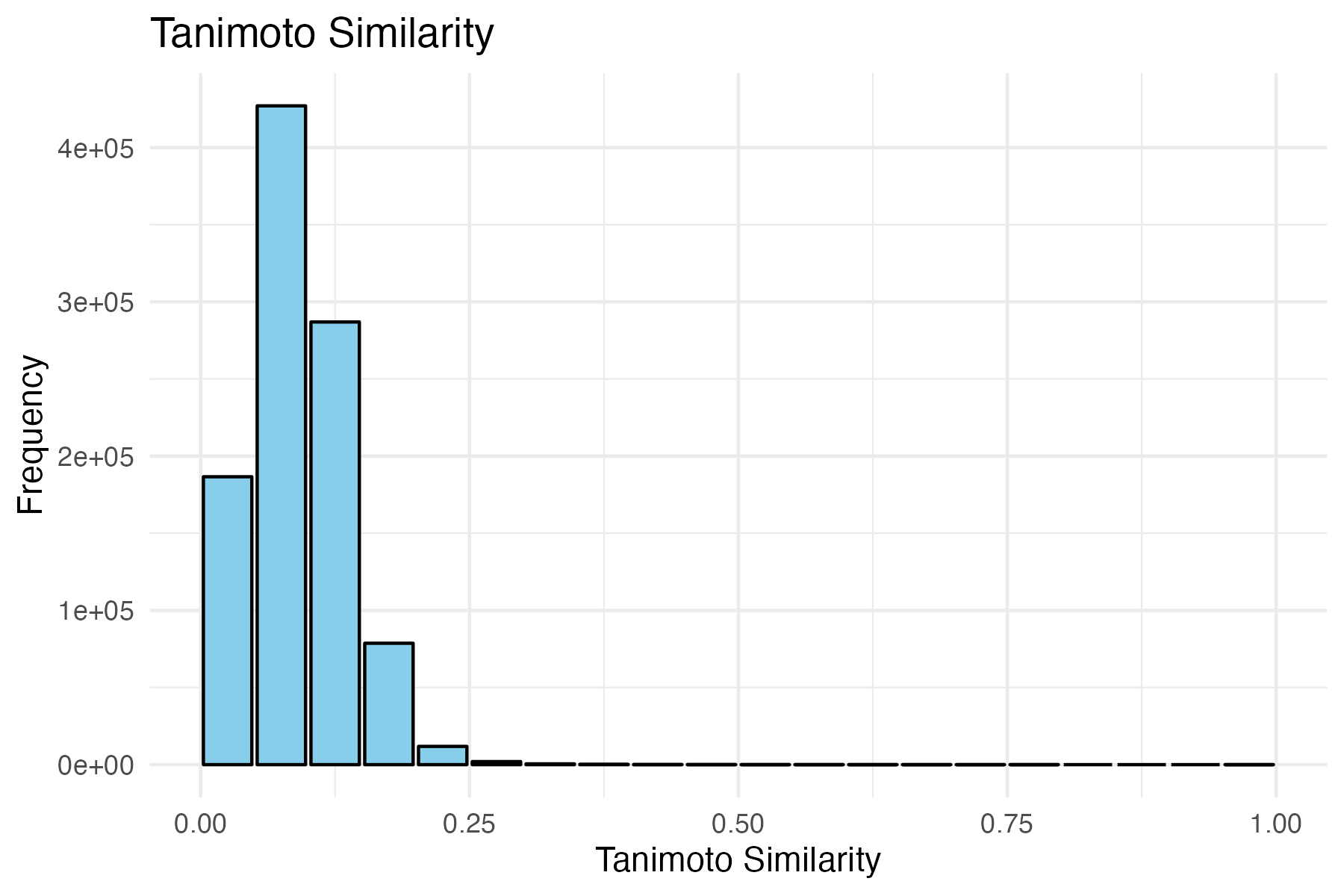
Run similarity search all vs all in TotalSMILESPositiveNegative.tsv

# To run the script please provide infile and outfile names (required)  
python Scripts/DataPreperation/4.RdKitSimilaritySearch.py -infile TotalSMILESPositiveNegative.tsv -outfile TotalSMILESSimilaritySearchOutput.tsv

#### **Step3**

Plot Tanimoto Similarity Between Positive and Negative sets.

# Here the first argument is input file and second is the output file  
Rscript Scripts/DataPreperation/5.PlotTanomiotoSimilarityHistograms.R TotalSimilaritySearchOutputs.tsv SimilaritySearchTotal.png



**Figure 3:** Distribution of structural similarity between molecules in positive and negative data. The similarity search was conducted using RdKit.

* Since all the pair-wise similarities have a mean of 0.091, we can use randomly selected 544 compounds with similarity less than mean.

# Load the pandas library  
# Run these commands in python terminal  
import pandas as pd  
  
# Load the Similarity search output data generated in previous step  
SimilarityOutput = pd.read\_csv("DataCuration/TotalSimilaritySearchOutputs.tsv", sep="\t")  
  
SimilarityOutput.Query = SimilarityOutput.Query.astype(str)  
SimilarityOutput.Target = SimilarityOutput.Target.astype(str)  
  
# Only keep instances with positive compound ID as query  
SimilarityOutput = SimilarityOutput[~(SimilarityOutput.Query.str.contains("MSID"))]  
# Keep instances with negative compound ID as target  
SimilarityOutput = SimilarityOutput[SimilarityOutput.Target.str.contains("MSID")]  
# Keep the instances with high dis-similarity  
SimilarityOutput = SimilarityOutput[SimilarityOutput.Similarity<=0.091]  
  
# Selecting 544 negative compounds. Keep random\_state=42 for reproducibility  
SimilarityOutput = pd.Series(SimilarityOutput.Target.unique())  
SimilarityOutput = SimilarityOutput.sample(544, random\_state=42)  
  
# Save the list in a output file  
SimilarityOutput.to\_csv("DataCuration/ListofNegativeCompounds.txt",index=False)

## 2. Data Preparation

For machine to be able to read chemical information, we need to represent chemical structures as Descriptors and Fingerprints using RdKit.

### Descriptor Calculation

#### **Descriptors of positive data**

# To calculate RdKit descriptors use command below  
# Use infile with CompoundID and CanonicalSMILES columns  
# Give outfile name to save the outputs and label shows the category for the machine learning.  
python Scripts/DataPreperation/6.DescriptorCalculation.py -infile DataCuration/Positivedataset/FinalProcessedSmilesCompound.tsv -outfile DataPreparation/PositiveDesc.csv -label "positive"

#### **Descriptors of negative data**

# To calculate RdKit descriptors use command below  
# Use infile with CompoundID and CanonicalSMILES columns  
# Give outfile name to save the outputs and label shows the category for the machine learning.  
python Scripts/DataPreperation/6.DescriptorCalculation.py -infile DataCuration/Negativedataset/FinalProcessedSmilesCompound.tsv -outfile DataPreparation/NegativeDesc.csv -label "negative"

To get the final 544 compounds out of total negative compounds, use following command lines.

Then split the data into training and testing sets (80:20 % split).

# Extract the randomly selected negative compounds chosen at Data curation steps  
head -1 DataPreparation/NegativeDesc.csv > DataPreparation/negative\_header # Get header of descriptor file  
grep -w -f DataCuration/ListofNegativeCompounds.txt DataPreparation/NegativeDesc.csv > DataPreparation/NegativeDesc.csv1  
cat DataPreparation/negative\_header DataPreparation/NegativeDesc.csv1 > DataPreparation/NegativeDesc.csv2  
mv DataPreparation/NegativeDesc.csv2 DataPreparation/NegativeDesc.csv # Final Negative data  
rm DataPreparation/NegativeDesc.csv1  
  
# Split the positive and negative data into training and testing data  
# There are 544 compounds per datasets  
# Split is 80:20% => 544\*0.8 = ~435 training and 109 testing  
  
# Positive taining/testing set  
head -n 436 DataPreparation/PositiveDesc.csv > DataPreparation/Positivetraining.csv  
tail -n 109 DataPreparation/PositiveDesc.csv > DataPreparation/Positivetesting.csv  
cat DataPreparation/negative\_header DataPreparation/Positivetesting.csv > DataPreparation/Positivetesting.csv1  
mv DataPreparation/Positivetesting.csv1 DataPreparation/Positivetesting.csv  
  
# Negative taining/testing set  
head -n 436 DataPreparation/NegativeDesc.csv > DataPreparation/Negativetraining.csv  
tail -n 109 DataPreparation/NegativeDesc.csv > DataPreparation/Negativetesting.csv  
cat DataPreparation/negative\_header DataPreparation/Negativetesting.csv > DataPreparation/Negativetesting.csv1  
mv DataPreparation/Negativetesting.csv1 DataPreparation/Negativetesting.csv

## 3. Machine Learning

### a. Method Comparisons

Here we will be looking into different types of classical machine learning algorithms and comparing them with each other using various statistical measures.

Concatenate the Positive and Negative Data training and testing files to create Training and Testing sets for machine learning.

Use following Python based code to create Training and Testing data.

# Run these in python terminal  
import pandas as pd  
  
# Load positive and negative training dataset  
positive = pd.read\_csv("DataPreparation/Positivetraining.csv")  
negative = pd.read\_csv("DataPreparation/Negativetraining.csv")  
  
Training = pd.concat([positive,negative], ignore\_index=True)  
# Save the training data  
Training.to\_csv("DataPreparation/TrainingData.csv", index=False)  
  
# Load positive and negative testing dataset  
positive = pd.read\_csv("DataPreparation/Positivetesting.csv")  
negative = pd.read\_csv("DataPreparation/Negativetesting.csv")  
  
Testing = pd.concat([positive,negative], ignore\_index=True)  
  
# Save the training data  
Testing.to\_csv("DataPreparation/TestingData.csv", index=False)

We tested Four different classical machine learning algorithms (CART, Support Vector Machine, Kth Nearest Neighbor and Random Forest).

Let’s look into details of each of these methods. What are these methods and how do they function.

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It is based on the concept of ensemble learning, which is a process of combining multiple algorithms to solve a particular problem. The main idea behind ensemble learning is that a group of weak models can come together to form a powerful model.

Random Forest can be used for both Classification and Regression problems in ML. It creates a set of decision trees from a randomly selected subset of the training set, which then aggregates the votes from different decision trees to decide the final class of the test object. This is known as the “wisdom of the crowd” approach, where the final prediction is based on the majority vote of the individual decision trees.

The Random Forest algorithm works by creating multiple decision trees and then aggregating their predictions. This is done by randomly selecting a subset of the training data and using it to train each decision tree. The randomness in the selection of the training data helps to reduce the correlation between the decision trees, which in turn helps to improve the overall performance of the model.

The key benefits of using Random Forest include:

* Improved accuracy: By aggregating the predictions of multiple decision trees, Random Forest can achieve higher accuracy than a single decision tree.
* Reduced overfitting: Random Forest is less prone to overfitting than a single decision tree, as the randomness in the selection of the training data helps to reduce the correlation between the decision trees.
* Handling of missing values: Random Forest can handle missing values in the data by using the median value of the variable for regression problems and the mode for classification problems.
* Feature importance: Random Forest can provide information on the importance of each feature in the model, which can be useful for feature selection and understanding the underlying data. (Breiman 2001)

Support Vector Machine (SVM)

Support Vector Machine (SVM) is one of the classical machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. SVM is based on the concept of finding a hyperplane that can best separate two classes of data.

The main idea behind SVM is to find the hyperplane that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the nearest data points from each class. These nearest data points are called support vectors, hence the name Support Vector Machine.

SVM uses a kernel function to transform the data into a higher dimensional space, where it is easier to find a hyperplane that can separate the two classes. The most commonly used kernel functions are linear, polynomial, and radial basis function (RBF) (Cristianini and Shawe-Taylor 2000).

There are several types of kernel functions that can be used in Support Vector Machine (SVM), including:

1. Linear kernel: This is the simplest kernel function, which maps the data into a higher dimensional space by preserving the original feature space. The linear kernel is defined as:

where x and y are the input vectors, and is the transpose of x.

1. Polynomial kernel: This kernel function maps the data into a higher dimensional space by applying a polynomial function to the original feature space. The polynomial kernel is defined as:

where gamma, coef0, and degree are the kernel parameters.

1. Radial Basis Function (RBF) kernel: This kernel function maps the data into an infinite dimensional space, and is defined as:

where gamma is the kernel parameter, and ||x - y|| is the Euclidean distance between x and y.

1. Sigmoid kernel: This kernel function is similar to the polynomial kernel, but with a different activation function. The sigmoid kernel is defined as:

where gamma and coef0 are the kernel parameters. Even the form of the sigmoid function same as hyperbolic function’s kernel, the internal function form is different.

1. Hyperbolic Tangent kernel: This kernel function is similar to the sigmoid kernel, but with a different activation function. The hyperbolic tangent kernel is defined as:

where gamma and coef0 are the kernel parameters.

1. Exponential Chi-Square kernel: This kernel function is used for data that has been normalized, and is defined as:

where gamma is the kernel parameter, and $x\_i $and are the i-th elements of the input vectors x and y.

The key benefits of using SVM include:

* Improved accuracy: SVM can achieve high accuracy, especially in cases where the data is not linearly separable.
* Handling of high-dimensional data: SVM can handle high-dimensional data, as it uses a kernel function to transform the data into a higher dimensional space.
* Robustness to noise: SVM is less sensitive to noise in the data, as it focuses on finding the hyperplane that maximizes the margin between the two classes.
* Flexibility: SVM can be used for both classification and regression problems, and can handle different types of kernel functions.

Kth Nearest Neighbor (kNN)

K-Nearest Neighbor (kNN) is a simple and effective instance-based learning algorithm used for classification and regression tasks. In kNN, the output of a new instance is determined by a majority vote of its k-nearest neighbors in the training set, where k is a positive integer. For classification tasks, the class label of the new instance is assigned based on the majority class of its k-nearest neighbors. For regression tasks, the output value of the new instance is calculated as the average of the output values of its k-nearest neighbors.

The performance of kNN depends on the choice of k, the distance metric used, and the curse of dimensionality. A small value of k may result in overfitting, while a large value of k may result in underfitting. The distance metric used in kNN can be Euclidean, Manhattan, or any other metric that is appropriate for the data. The curse of dimensionality refers to the phenomenon where the performance of kNN degrades as the dimensionality of the data increases. This is because the volume of the feature space increases so rapidly with the number of dimensions that the available data become sparse.

Despite these challenges, kNN has several advantages, such as ease of implementation, robustness to noisy data, and the ability to handle multi-class problems. It is often used as a baseline model for comparison with other more complex models. In summary, kNN is a simple yet powerful algorithm that can be used for classification and regression tasks, and its performance depends on the choice of k, the distance metric, and the dimensionality of the data (Cover and Hart 1967).

Classification And Regression Trees (CART)

Classification and Regression Trees (CART) is a popular decision tree algorithm used for both classification and regression tasks. In CART, the data is recursively split into subsets based on the values of the input features until a stopping criterion is met. The tree is constructed by selecting the best split at each node, which maximizes the homogeneity of the resulting subsets. For classification tasks, the homogeneity is measured using a criterion such as Gini impurity or entropy, while for regression tasks, the homogeneity is measured using the mean squared error.

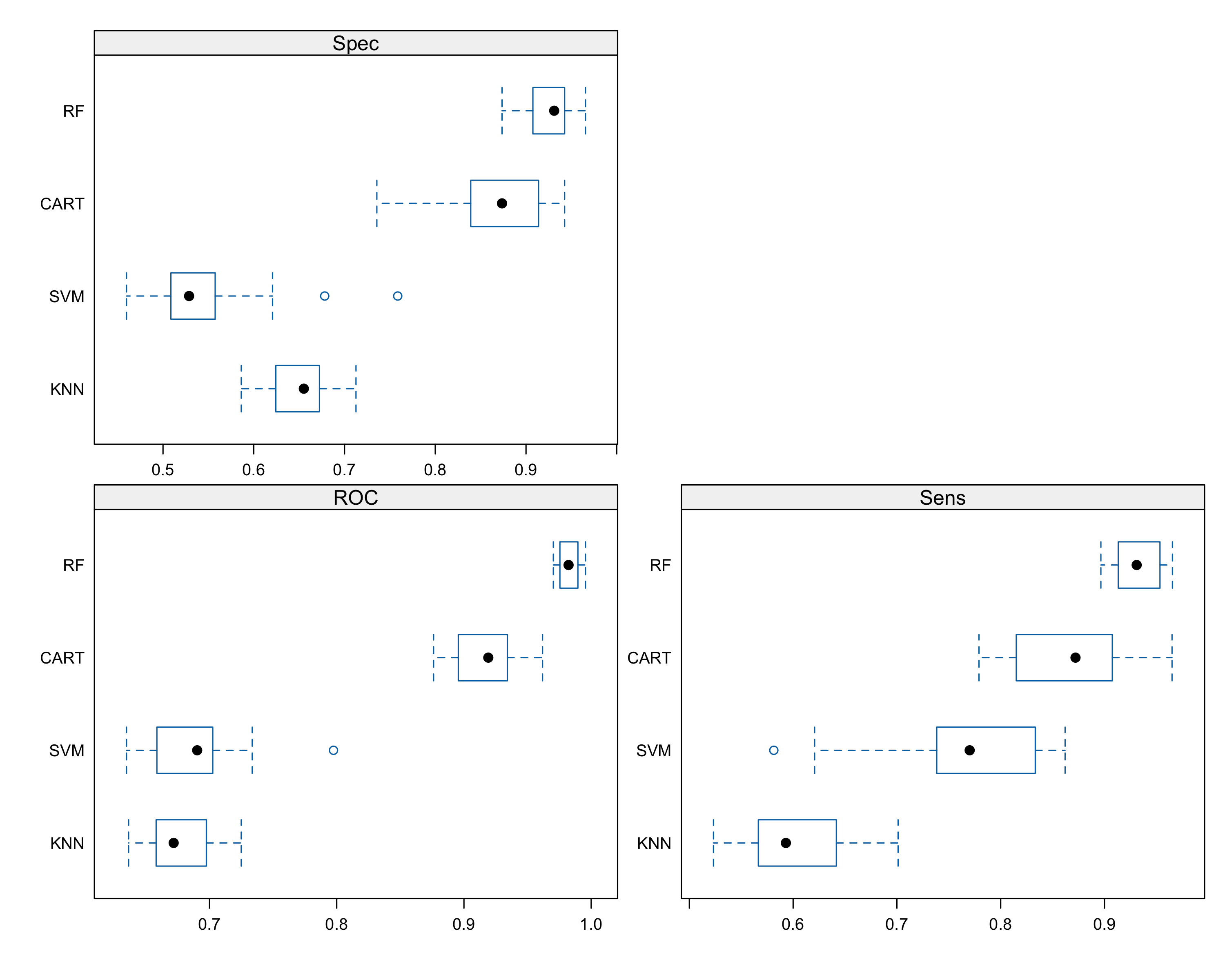
CART has several advantages, such as ease of interpretation, ability to handle both categorical and numerical data, and robustness to outliers. The resulting tree can be visualized and interpreted, providing insights into the relationships between the input features and the output variable. CART can also handle missing values by using surrogate splits, which are alternative splits that are used when the primary split is not applicable.

However, CART also has some limitations, such as a tendency to overfit the data, sensitivity to small changes in the data, and the potential for unstable tree structures. To address these limitations, various techniques such as pruning, cross-validation, and ensemble methods can be used. In summary, CART is a powerful and interpretable algorithm for classification and regression tasks, but it requires careful tuning and regularization to avoid overfitting and ensure stability (Breiman et al. 2017).

### Method comparison results

The results revealed that the Random Forest outperformed all the other methods with an Receiver Operating Curve (ROC) value of 0.97, Sensitivity value of 0.89 and Specificity value of 0.87.

So, in the further analysis, we will be using Random Forest as the machine learning algorithm to train models.

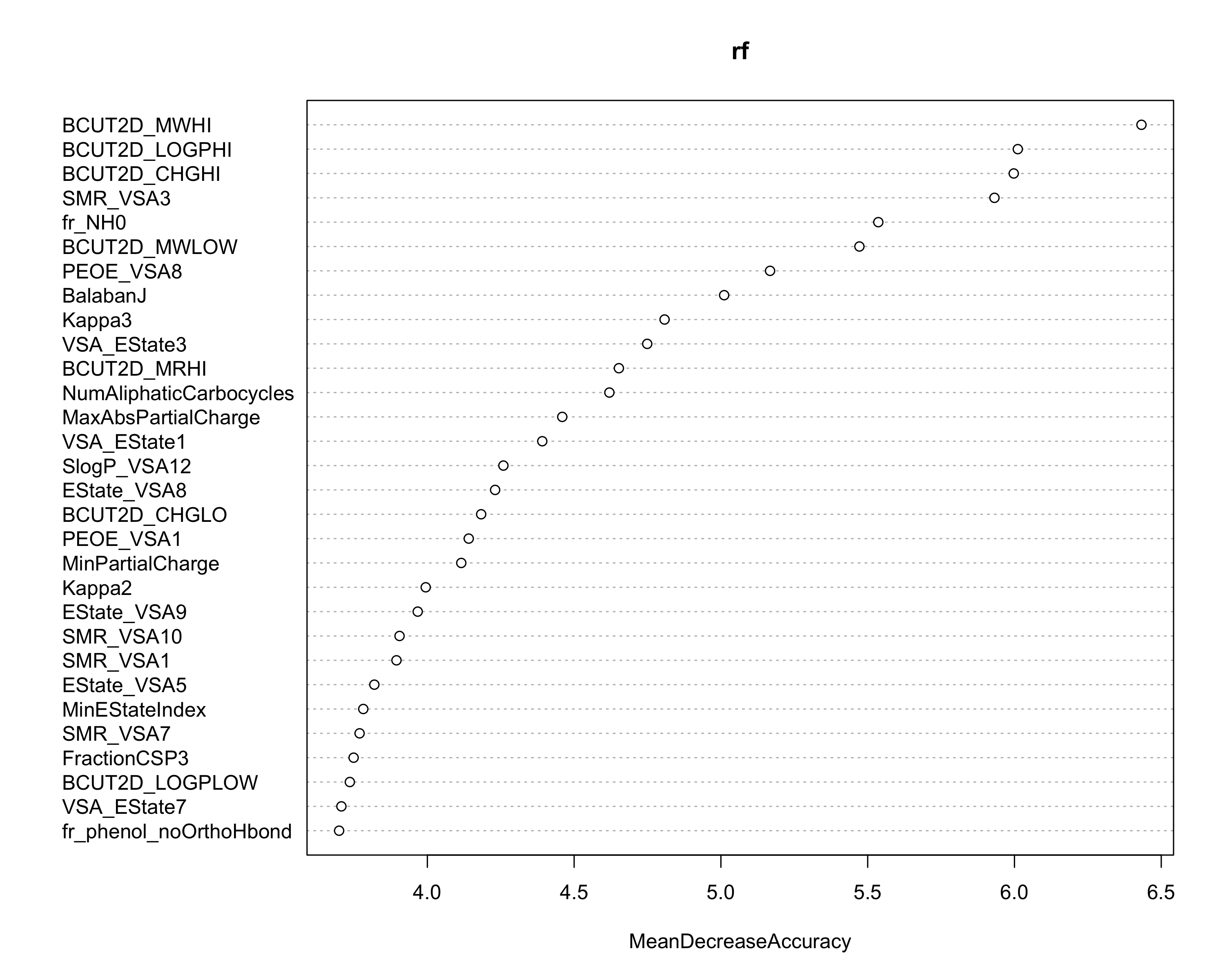


**Figure 4:** Box plots of Specificity (Spec), Sensitivity (Sens) and ROC for RF, CART, SVM and kNN algorithms. The box shows the distribution of values and black dot in the middle represents mean of the metrics.

### b. Feature Selection:

For an optimal machine learning model, it is extremely important to select most informative features. To accomplish this goal, we performed a mean decrease in accuracy test on training data. The Mean Decrease in Accuracy (MDA) looks at the decrease in model performance (accuracy) when a feature is remove from the training model. So, the features with highest importance will show highest decrease in accuracy.

The mean decrease in accuracy for top 30 features is then plotted as scatter plot to represent MDA.



**Figure 5:** Mean Decrease in Accuracy per features (descriptor). The x-axis represents the decrease in accuracy and y-axis represents the name of the best features. According to the figure, BCUT2D\_MWHI is the most important feature as it shows the highest decrease in accuracy.

### c. Optimizing number of features for training

To train the machine learning model with best performance, it is important to select subset of the features and check for performance on each subset of the data.

Here, we will be looking at top **10, 20, 30, 40 and 50** features then we will use these to check for performances.

#### **Step 1**

Create descriptor lists with top 10, 20, 30, 40 and 50 features collected from Mean Decrease in Accuracy (MDA).

# From the Mean Decrease in Accuracy get top 10, 20, 30, 40 and 50 features and create text files for each of these features  
  
# Top 10 features  
cut -d"," -f1 MachineLearning/FeatureSelection/MeanAcc.csv | sed 1d | head -10 > MachineLearning/FeatureSelection/top10  
  
# Top 20 features  
cut -d"," -f1 MachineLearning/FeatureSelection/MeanAcc.csv | sed 1d | head -20 > MachineLearning/FeatureSelection/top20  
  
# Top 30 features  
cut -d"," -f1 MachineLearning/FeatureSelection/MeanAcc.csv | sed 1d | head -30 > MachineLearning/FeatureSelection/top30  
  
# Top 40 features  
cut -d"," -f1 MachineLearning/FeatureSelection/MeanAcc.csv | sed 1d | head -40 > MachineLearning/FeatureSelection/top40  
  
# Top 50 features  
cut -d"," -f1 MachineLearning/FeatureSelection/MeanAcc.csv | sed 1d | head -50 > MachineLearning/FeatureSelection/top50

#### **Step2**

Prepare subset of data with these aforementioned top features.

# For each of the top features list perform getting subset from Training data  
for subset in MachineLearning/FeatureSelection/top\*; do sh Scripts/MakeSubsetData/MakeSubsetData.sh DataPreparation/TrainingData.csv $subset; done

#### **Step3**

**Optimize mtry value for each of the subset training data.**

In random forest, **mtry** is a hyperparameter that determines the number of features randomly selected at each node for splitting. It is a key parameter that can significantly impact the performance of the random forest model.

The **mtry** value is a positive integer that specifies the number of features to be randomly selected at each node. By selecting a subset of features at each node, random forest reduces the correlation between the trees in the forest, which can lead to better generalization performance.

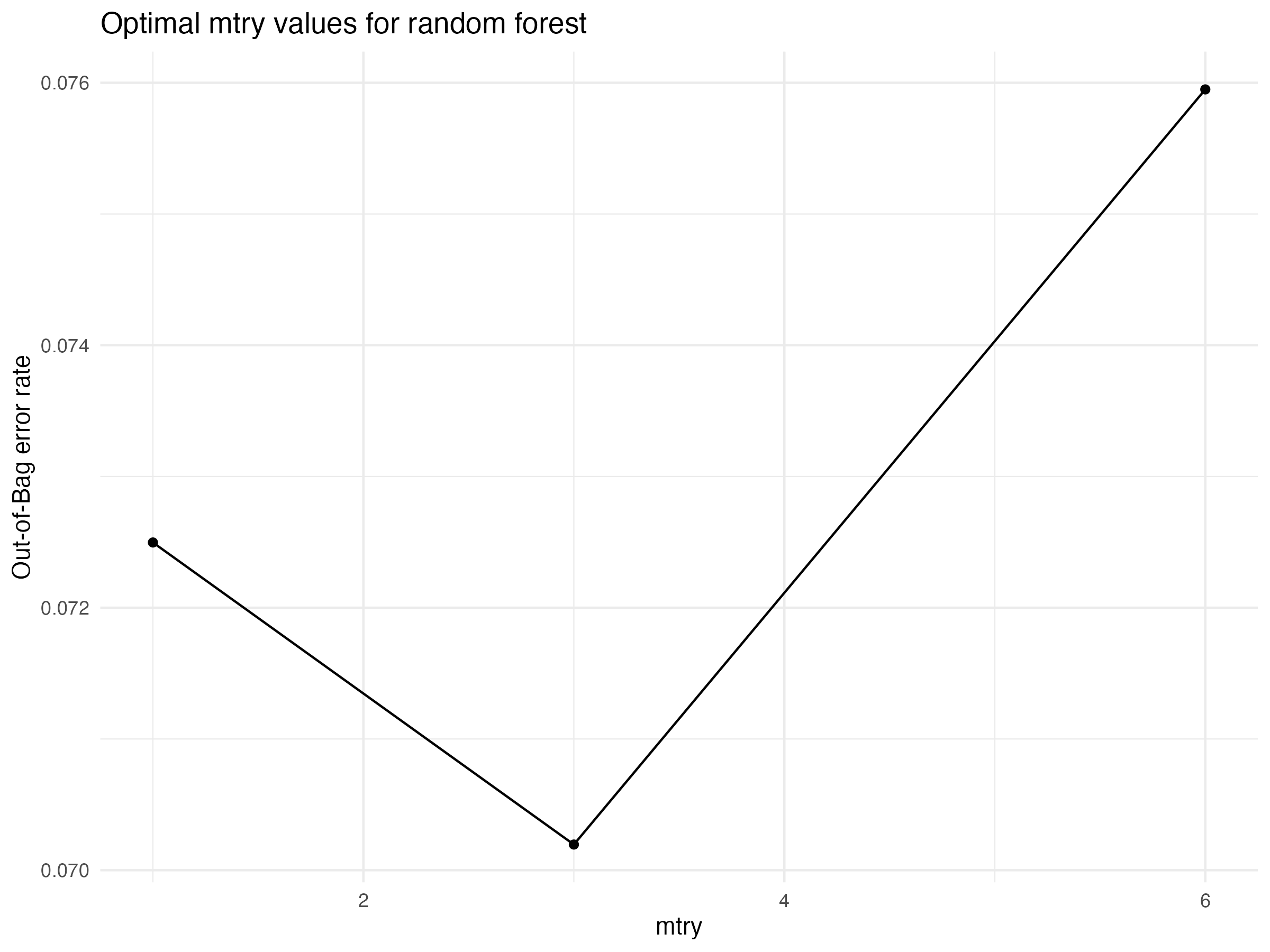
The optimal value of **mtry** depends on the dataset and the problem being solved. In general, a good starting point is to set **mtry** to the square root of the total number of features in the dataset. For example, if the dataset has 10 features, a good starting value for **mtry** would be 3 (i.e., sqrt(10) ≈ 3.16).

If the **mtry** value is too low, the random forest model may become overfit to the training data, leading to poor generalization performance. On the other hand, if the **mtry** value is too high, the model may not capture the underlying patterns in the data, leading to poor performance.

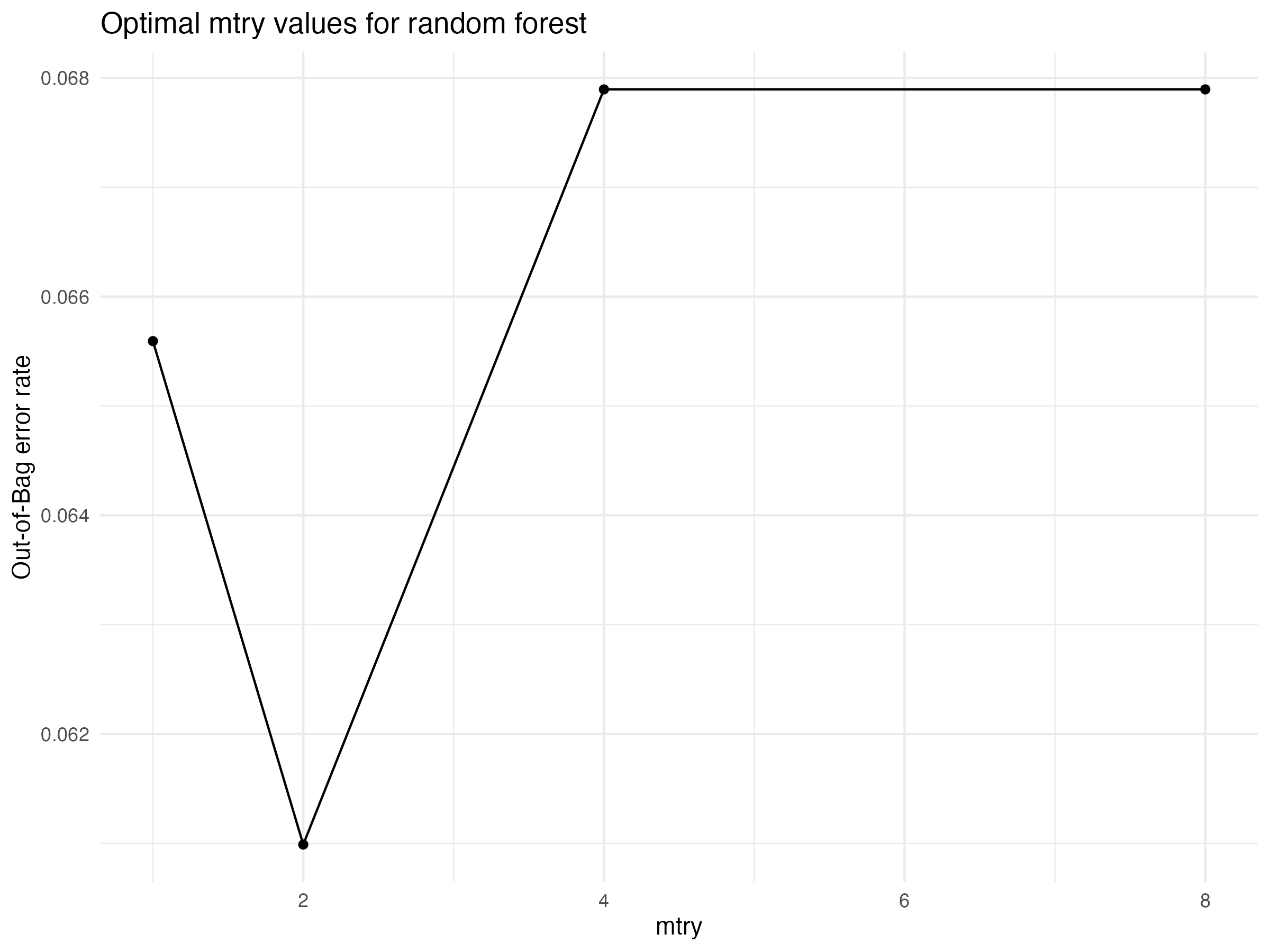
In practice, it is often recommended to try different values of **mtry** and evaluate the performance of the model using cross-validation or other evaluation metrics. This can help to identify the optimal value of **mtry** for a given dataset and problem.

# Mtry runs  
# Top 10  
Rscript Scripts/MtryOptimization/mtrySelection.R DataPreparation/TrainingDatatop10.csv MachineLearning/MtryOptimization/mtrytop10.png  
  
# Top 20  
Rscript Scripts/MtryOptimization/mtrySelection.R DataPreparation/TrainingDatatop20.csv MachineLearning/MtryOptimization/mtrytop20.png  
  
# Top 30  
Rscript Scripts/MtryOptimization/mtrySelection.R DataPreparation/TrainingDatatop30.csv MachineLearning/MtryOptimization/mtrytop30.png  
  
# Top 40  
Rscript Scripts/MtryOptimization/mtrySelection.R DataPreparation/TrainingDatatop40.csv MachineLearning/MtryOptimization/mtrytop40.png  
  
# Top 50  
Rscript Scripts/MtryOptimization/mtrySelection.R DataPreparation/TrainingDatatop50.csv MachineLearning/MtryOptimization/mtrytop50.png

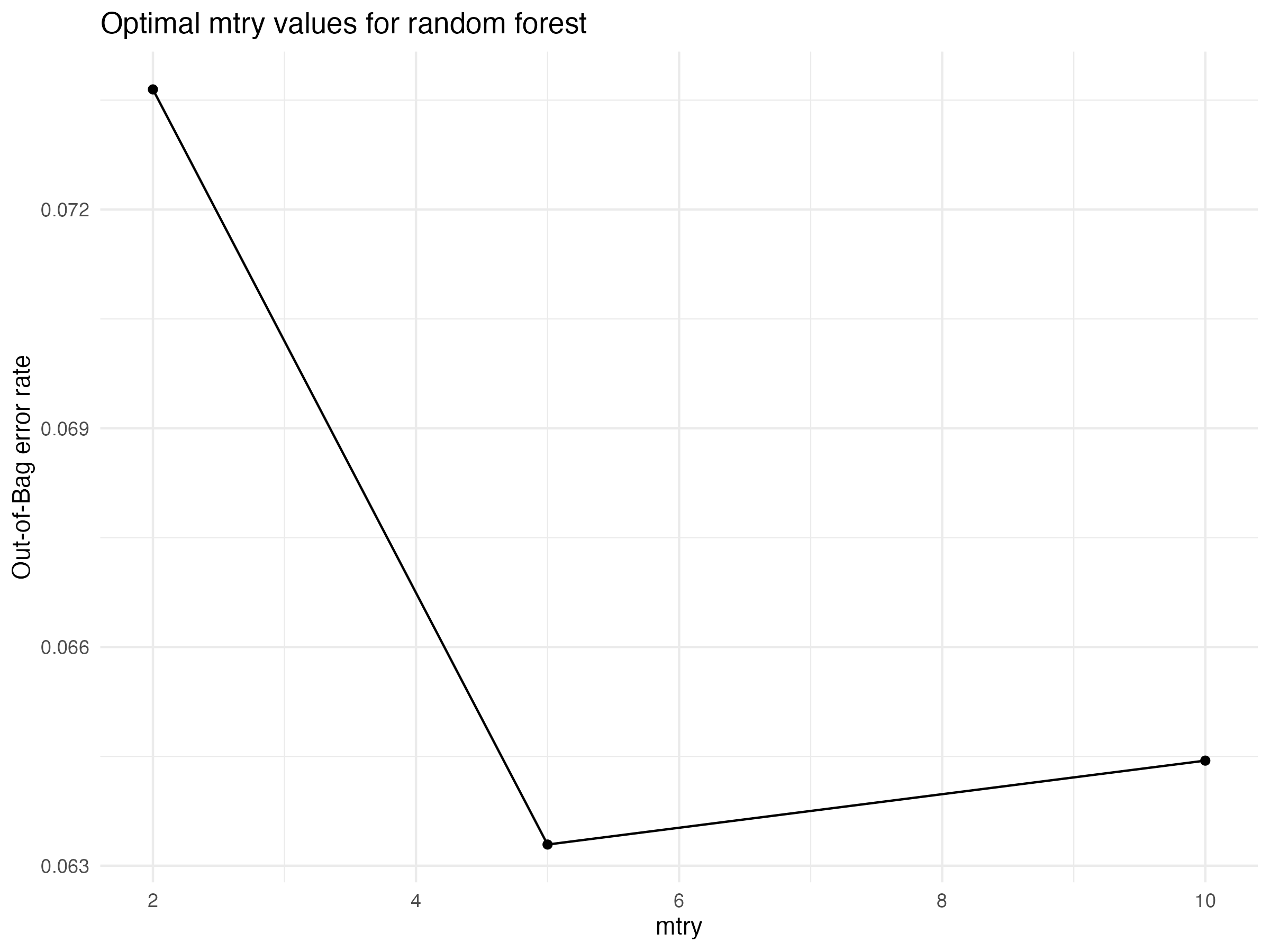
The prediction is estimated based on the out of bag error (OOB error). Out of bag error, also known as out of bag estimate, is a method used to measure the prediction error of machine learning models such as random forests, boosted decision trees, and other models that utilize bootstrap aggregating (bagging). Bagging involves sub-sampling with replacement to create training samples for the model to learn from. The out of bag error is calculated by evaluating predictions on those observations that were not used in the building of the next base learner. It can be useful for evaluating the performance of the model on unseen data and can provide an indication of how well the model is performing, although it is not always a reliable estimate of the generalization error of the model. The out of bag error is calculated by finding all models that are not trained by the out of bag instance, taking the majority vote of these models’ results for the out of bag instance, and compiling the out of bag error for all instances in the out of bag dataset.



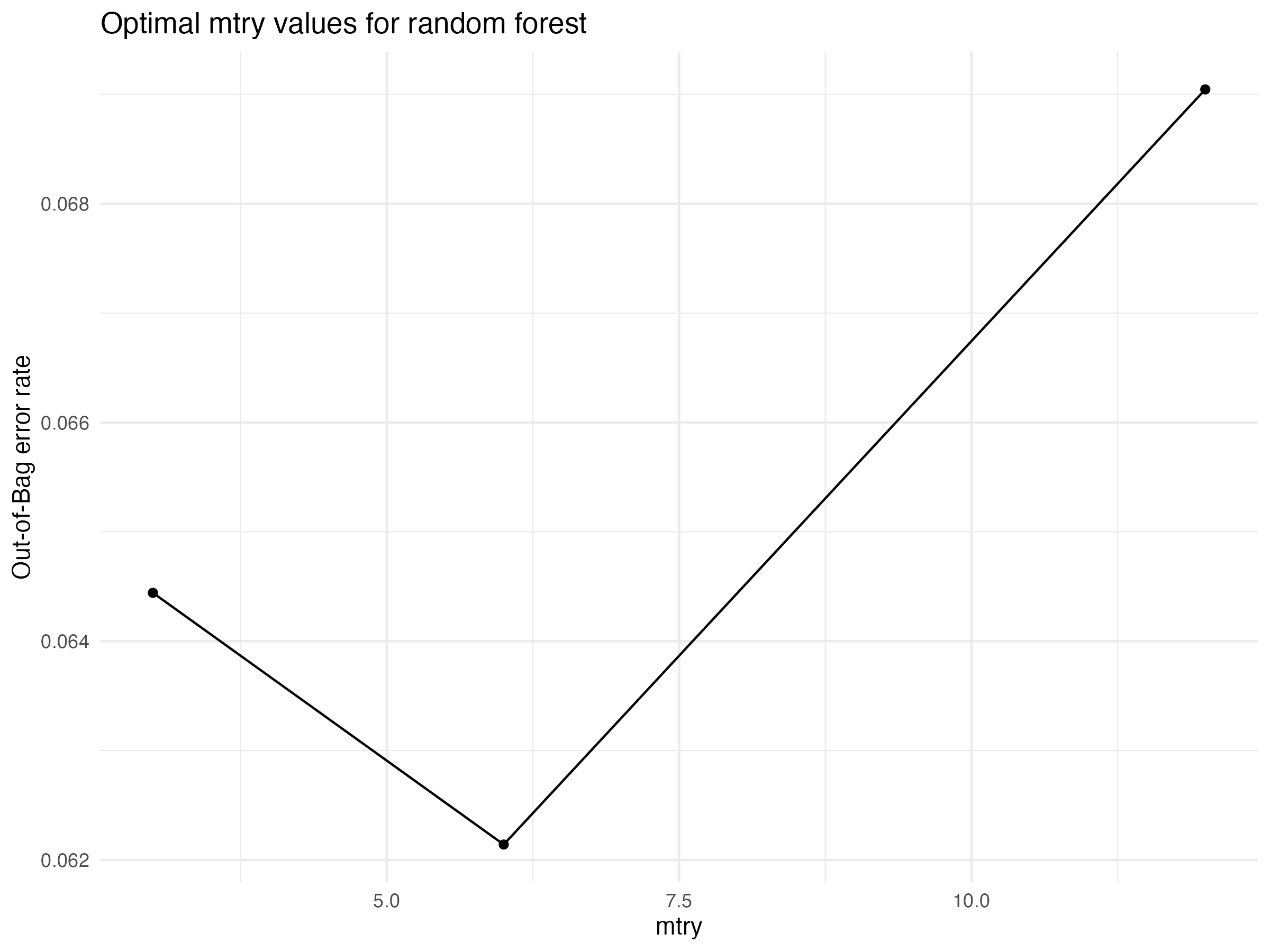
**Figure 6:** Mtry optimization for Top 10 features.



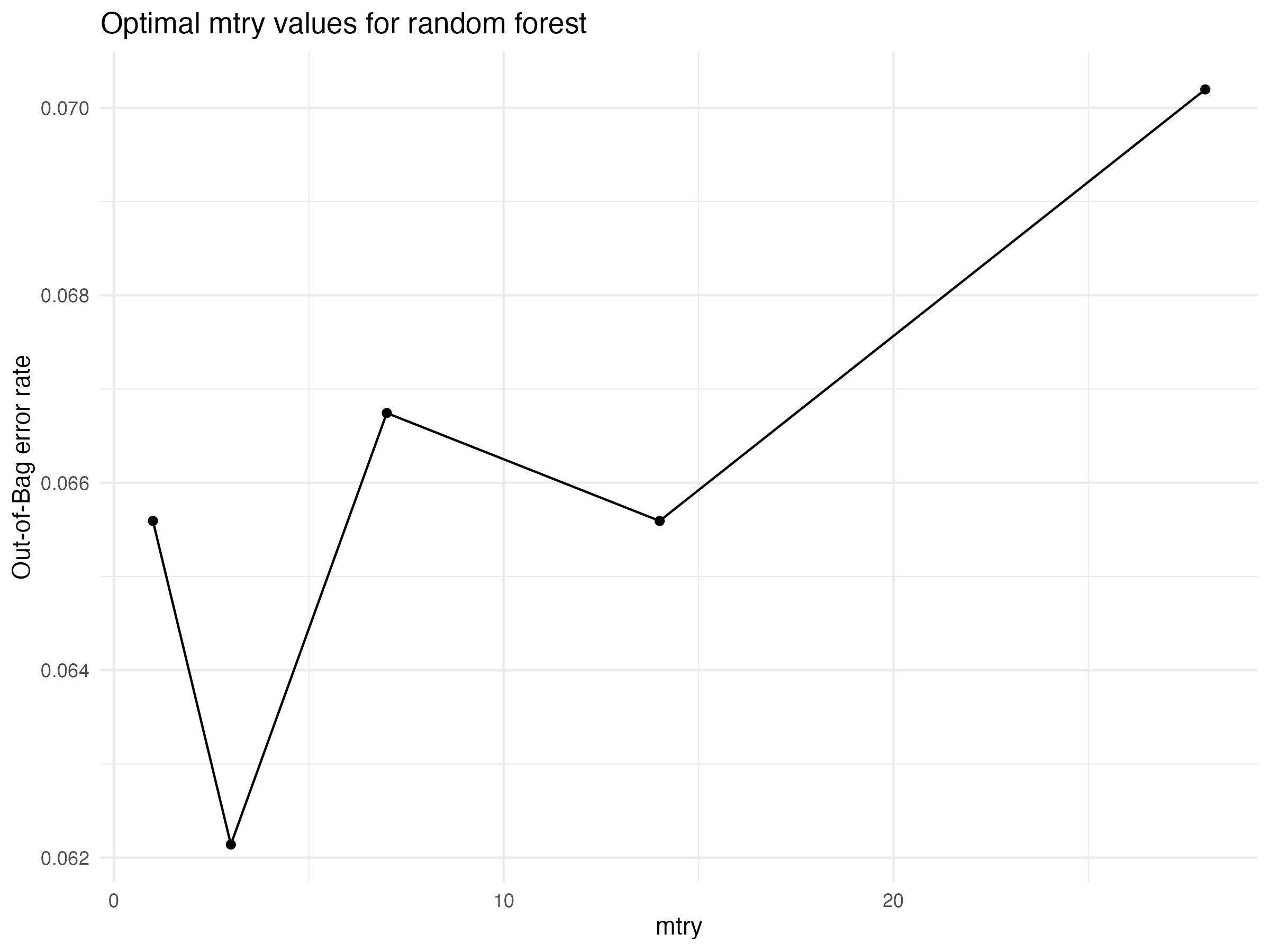
**Figure 7:** Mtry optimization for Top 20 features.



**Figure 8:** Mtry optimization for Top 30 features.



**Figure 9:** Mtry optimization for Top 40 features.



**Figure 10:** Mtry optimization for Top 50 features.

### d. Finding the best model

* The next step will be to choose top 3 mtry values from the Mtry optimization to train models. These models are expected to give best performances because of least OOB error.

# Following code can be utilized for testing the performance of RF method on Top features at best mtry values  
  
# Top10  
Rscript Scripts/ModelTraining/Top10/ModelTrainingTop10.R DataProcessing/TrainingDatatop10.csv > MachineLearning/Modeltraining/TrainingTop10.results  
  
# Top20  
Rscript Scripts/ModelTraining/Top20/ModelTrainingTop20.R DataProcessing/TrainingDatatop20.csv > MachineLearning/Modeltraining/TrainingTop20.results  
  
# Top30  
Rscript Scripts/ModelTraining/Top30/ModelTrainingTop30.R DataProcessing/TrainingDatatop30.csv > MachineLearning/Modeltraining/TrainingTop30.results  
  
# Top40  
Rscript Scripts/ModelTraining/Top40/ModelTrainingTop40.R DataProcessing/TrainingDatatop40.csv > MachineLearning/Modeltraining/TrainingTop40.results  
  
# Top50  
Rscript Scripts/ModelTraining/Top50/ModelTrainingTop50.R DataProcessing/TrainingDatatop50.csv > MachineLearning/Modeltraining/TrainingTop50.results

**Table 2:** Table of Model performance comparison for top 10, 20, 30, 40, and 50 features at the various mtry values. The best model top 30 and it is highlighted in bold.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Features | Mtry | Accuracy | Sensitivity | Specificity | MCC |
| Top 10 | 2 | 0.935 | 0.935 | 0.933 | 0.869 |
| Top 10 | 3 | 0.929 | 0.933 | 0.925 | 0.857 |
| Top 10 | 4 | 0.931 | 0.935 | 0.927 | 0.862 |
| Top 20 | 1 | 0.935 | 0.933 | 0.938 | 0.871 |
| Top 20 | 2 | 0.932 | 0.935 | 0.929 | 0.864 |
| Top 20 | 3 | 0.937 | 0.936 | 0.938 | 0.873 |
| Top 30 | 4 | 0.940 | 0.940 | 0.940 | 0.880 |
| Top 30 | 5 | 0.942 | 0.944 | 0.941 | 0.885 |
| **Top 30** | **6** | **0.945** | **0.949** | **0.941** | **0.890** |
| Top 40 | 5 | 0.939 | 0.942 | 0.936 | 0.878 |
| Top 40 | 6 | 0.937 | 0.946 | 0.928 | 0.874 |
| Top 40 | 7 | 0.939 | 0.942 | 0.936 | 0.878 |
| Top 50 | 3 | 0.937 | 0.942 | 0.932 | 0.873 |
| Top 50 | 7 | 0.940 | 0.946 | 0.934 | 0.880 |
| Top 50 | 14 | 0.938 | 0.944 | 0.932 | 0.876 |

To define the accuracy, sensitivity, specificity, and Matthews correlation coefficient (MCC) metrics, following formulas can be used:

* Accuracy: The proportion of correct predictions out of the total number of predictions. It is calculated as:

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

* Sensitivity: The proportion of true positives out of the total number of actual positives. It is calculated as:
* Specificity: The proportion of true negatives out of the total number of actual negatives. It is calculated as:
* Matthews correlation coefficient (MCC): A measure of the quality of binary classifications. It takes into account true and false positives and negatives, and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes. It is calculated as:

### e. Blind testing (Model Evaluation/ Independent Validation)

The best model at Top 30 , mtry 6, will be used to predict model performance using independent testing test.

# Copy the best Model to the MachineLearning/IndependentTesting to be used as model  
cp MachineLearning/Modeltraining/rf\_mtry6.Rdata MachineLearning/IndependentTesting

Extract the top 30 features from Testing Data to make subset of the validation data as the best model has top 30 features.

# Subset testing data to have top 30 features  
sh Scripts/MakeSubsetData/MakeSubsetData.sh DataPreparation/TestingData.csv MachineLearning/FeatureSelection/top30  
cp DataPreparation/Testingdatatop30.csv MachineLearning/IndependentTesting

Run validation script to get prediction on testing data.

# Arg1: Model to be loaded, Arg2: Testing file  
Rscript Scripts/IndependentTesting/ValidationPerformance.R MachineLearning/IndependentTesting/rf\_mtry6.Rdata MachineLearning/IndependentTesting/Testingdatatop30.csv

Note: The validation script creates a confusion matrix (2\*2) for testing data based on true labels and predicted labels and along with the confusion matrix the script also calculates are metric of performance including sensitivity, specificity, prevalence, PPV, NPC, detection rate, detection prevalence, balanced accuracy, precision, recall and F1 value.

The formula for prevalence, PPV, NPC, detection rate, detection prevalence, balanced accuracy, precision, recall and F1 value are as follows for a given confusion matrix:

|  |  |  |
| --- | --- | --- |
|  | **Reference** |  |
| **Predicted** | **Positive** | **Negative** |
| **Positive** | A | B |
| **Negative** | C | D |

where beta = 1 for this function (Kuhn 2008).

### Performance Table

For the calculation of the final results matrices <https://onlineconfusionmatrix.com> was used.

During the validation process the positive label was set as “positive.”

**Table 3:** Performance table of Random Forest Model tested on independent (20%) validation set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Balanced Accuracy | Sensitivity | Specificity | Precision | Recall | F1 |
| 0.8349 | 0.7248 | 0.9450 | 0.9294 | 7248 | 0.8144 |

### Process the final prediction results table:

# To make the prediction table look like in good format. Run these commands in python terminal.  
  
import pandas as pd  
testingdata = pd.read\_csv("MachineLearning/IndependentTesting/Testingdatatop30.csv", sep=",")  
predicteddata = pd.read\_csv("MachineLearning/IndependentTesting/TestingDataPrediction.csv", sep=",")  
predicteddata.insert(0, "CompoundID", testingdata.CompoundID.tolist())  
predicteddata.to\_csv("MachineLearning/IndependentTesting/TestingDataPrediction.csv", index=False)

## Conclusion

AntiFungalML is a powerful tool for predicting the anti-fungal activity of small molecules. The tool uses a binary classification algorithm that takes SMILES of compounds as input and returns prediction probabilities for the input compound to have anti-fungal properties. The tool uses Random Forest as the machine learning algorithm to train models, which outperformed other methods with high accuracy, sensitivity, and specificity. The tool also performs feature selection and optimizes the number of features for training, resulting in a highly accurate and reliable model. The best model was found to be top 30 features with mtry 6, which achieved high performance metrics during blind testing. Even though, we were able to achieve high performance compare to other algorithms, there might still be some more improvemnets that can be made. For example, we can use more number of top features, include fingerprints as features etc. Overall, AntiFungalML can be a valuable tool for researchers and scientists working in the field of anti-fungal drug discovery.

**Reference:**

Bickerton, G. Richard, Gaia V. Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L. Hopkins. 2012. “Quantifying the Chemical Beauty of Drugs.” *Nature Chemistry* 4 (2): 90–98. <https://doi.org/10.1038/nchem.1243>.

Breiman, Leo. 2001. *Machine Learning* 45 (1): 5–32. <https://doi.org/10.1023/a:1010933404324>.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 2017. *Classification and Regression Trees*. Routledge. <https://doi.org/10.1201/9781315139470>.

Cover, T., and P. Hart. 1967. “Nearest Neighbor Pattern Classification.” *IEEE Transactions on Information Theory* 13 (1): 21–27. <https://doi.org/10.1109/tit.1967.1053964>.

Cristianini, Nello, and John Shawe-Taylor. 2000. “An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods,” March. <https://doi.org/10.1017/cbo9780511801389>.

Kuhn, Max. 2008. “Building Predictive Models in*R*Using the**caret**Package.” *Journal of Statistical Software* 28 (5). <https://doi.org/10.18637/jss.v028.i05>.

Vivek-Ananth, R. P., Ajaya Kumar Sahoo, Kavyaa Kumaravel, Karthikeyan Mohanraj, and Areejit Samal. 2021. “MeFSAT: A Curated Natural Product Database Specific to Secondary Metabolites of Medicinal Fungi.” *RSC Advances* 11 (5): 2596–2607. <https://doi.org/10.1039/d0ra10322e>.