Cuckoo Search-Driven Feature Selection for Decision Tree Modelling

Rajkumar S. Bhosale[1], Archana R. Panhalkar[2]
[1]Assistant Professor, Amrutvahini College of Engineering, Sangamner, India
[2]Assistant Professor, Amrutvahini College of Engineering, Sangamner, India
[2]archana10bhosale@rediffmail.com

Abstract
Features are fundamental components of decision tree modeling, and their relevance, quality, and selection are crucial determinants of the model's effectiveness and performance. However, decision trees can be computationally expensive, requiring a significant amount of memory to store the trees and their associated data structures. To address this limitation, we present a novel approach that utilizes a Cuckoo Search-based feature selection algorithm to construct efficient and optimal decision trees. The Cuckoo Search algorithm, inspired by the behavior of cuckoo birds, is a powerful metaheuristic algorithm that effectively selects high-quality features and creates accurate decision trees in the subforest. We evaluate the proposed method on a variety of datasets from the standard UCI learning repository with different domains and sizes, and our results demonstrate that the algorithm creates optimal decision trees with high performance.
Keywords*:* Decision tree, Feature selection, Nature-inspired, Data mining, Cuckoo Search Optimization, C4.5, CART

1. Introduction

A decision tree is a flowchart-like structure used in decision-making and classification problems. It is a graphical representation of all the possible solutions to a decision, based on certain conditions and consequences. In a decision tree, the root node represents the initial decision or problem, and each branch represents a possible outcome or alternative solution. The branches are split based on certain conditions, which are represented by internal nodes, until a final outcome or solution is reached at the leaf nodes. Decision trees can be used for both classification and regression tasks. In classification tasks, the decision tree is used to categorize data into different classes, while in regression tasks, it is used to predict continuous numerical values. Decision trees are widely used in various fields, such as business, finance, medicine, and engineering, to aid in decision-making and problem-solving. They are also used in machine learning algorithms, such as Random Forests, Gradient Boosted Trees, and XGBoost.

Features are the building blocks of decision tree modeling, and their quality, relevance, and selection are critical factors that can determine the performance and usefulness of the model. Features play a crucial role in decision tree modeling as they are used to make decisions or predictions about the target variable. Each node in the decision tree represents a decision based on a particular feature, and each branch represents the possible values or outcomes of that feature. The quality and relevance of the features directly affect the accuracy and complexity of the decision tree model. A good feature should be informative and discriminatory, meaning that it should have a strong association with the target variable and be able to differentiate between different classes or values of the target variable. The importance of each feature can be quantified using measures such as information gain, Gini index, or chi-squared test. These measures evaluate the impact of each feature on the target variable and can be used to rank the features and select the most relevant ones. In decision tree modeling, it is important to select the right set of features that can maximize the accuracy and interpretability of the model, while avoiding overfitting and complexity. Feature selection and optimization techniques can be used to identify and select the best subset of features for the model, based on their relevance, redundancy, and interdependence.

When building a decision tree model, overfitting can occur when there are too many features, also known as the "curse of dimensionality" [2]. This can cause the model to become overly complex and make overly specific decisions that do not generalize well to new data. To avoid these we are selecting important features to construct decision tree using Cuckoo search optimization. Feature optimization is important in machine learning and data science, as it helps to improve the accuracy, efficiency, and interpretability of the model. Here are some reasons why feature optimization is important:

1. Reduce Overfitting: Including too many irrelevant or redundant features in the model can lead to overfitting, where the model fits the training data too closely and fails to generalize to new data. Feature optimization helps to identify and remove such features, thereby reducing the risk of overfitting.
2. Improve Model Performance: By selecting the most relevant and informative features for the model, feature optimization can improve the accuracy, precision, and recall of the model, and reduce the error rate.

3. Reduce Model Complexity: Feature optimization can also help to reduce the complexity of the model by removing features that do not contribute significantly to the target variable. This can make the model more interpretable and easier to understand.
4. Save Computation Time: By reducing the number of features used in the model, feature optimization can also reduce the computation time required for model training, testing, and deployment. This can be especially important for large datasets or real-time applications.
5. Improve Data Understanding: Feature optimization can also provide insights into the underlying data by identifying the most important variables and their relationships with the target variable. This can help to improve data understanding and guide further analysis and decision-making.

This paper is organized into the following sections. A brief idea about various methods of feature selection and limitations are discussed in Section-2. Section-3 explains on Cuckoo Search Optimization algorithm . In Section-4 the proposed method of feature selection using cuckoo search optimization is discussed. The detailed experimental results with discussion are explained in Section-5. Finally, Section-6 concludes the proposed method with remarks.

2. Literature Review

In many situations, groups of features are highly correlated. This can lead to overfitting and poor generalization of the decision tree to new data. One way to address this issue is to carefully choose exemplars, which are representative features from the group that can stand in for the rest.

Here are a few ways to prevent decision tree overfitting due to a large number of features:

1. Feature Selection: Select only the most important features that are likely to have a strong impact on the target variable. You can use feature selection techniques such as correlation, mutual information, or regularization methods to select a subset of features.
2. Feature Extraction: Transforming the features into a lower-dimensional space using techniques such as Principal Component Analysis (PCA) [4] or Linear Discriminant Analysis (LDA) [4]. This can help remove redundant or correlated features, which can reduce overfitting.
3. Regularization: Regularization is a technique that adds a penalty term to the decision tree model's objective function. This can help reduce overfitting by discouraging the model from making overly complex decisions.
4. Cross-validation: Use k-fold cross-validation to evaluate the decision tree model's performance on a validation set. This can help prevent overfitting by testing the model's ability to generalize to new data.
5. Pruning: Pruning is a technique that removes nodes from the decision tree that do not contribute to its performance. This can help reduce the model's complexity and prevent overfitting.

There are two common strategies for selecting features in machine learning: filters and wrappers. Filters aim to identify features that are related to or predictive of the target variable, and they do so independently of the learning algorithm. For example, Information Gain is a filter technique that was originally developed by Quinlan as a way to build concise decision trees but is now widely used for feature selection in general [5]. On the other hand, the wrapper method evaluates subsets of features based on the accuracy estimates provided by a classifier that was built with that feature subset. This approach is more computationally expensive than filtering techniques but can produce better results because it takes the bias of the classifier into account and evaluates features in context. A detailed presentation of the wrapper approach can be found in [6][7].

Hsu [8] discusses the development of a generic fitness function for validating input specifications and the use of this function to create two genetic algorithm wrappers. One wrapper is designed for variable selection in decision tree inducers, while the other is designed for variable ordering in Bayesian network structure learning. These methods aim to optimize the feature selection and feature ordering processes to improve the performance of these machine learning models. Theodoridis et. al. aims to leverage the strengths of both genetic algorithms and decision trees to improve the performance of the decision tree algorithm [9]. The genetic wrapper will optimize the feature selection and feature ordering process, while the decision tree algorithm will build a classification tree that is optimal for the given dataset.

The PSO algorithm is used to efficiently explore the feature space and identify the feature subset that maximizes the F-measure objective function [10]. By using this approach, the paper aims to improve the performance of the machine learning model by selecting the most important features for the given task.

In the traditional feature selection algorithm based on decision tree, the decision tree is easy to be influenced by the category and the irrelevant features. In such case, it is complex in constructing the decision

tree and is liable to be over fitting. Therefore, it is required to select the most relevant feature in building the decision tree. In this paper, we proposed feature selection using Cuckoo Search algorithm to construct accurate and efficient decision tree.

3. Cuckoo Search Optimization Algorithm for Feature Selection

Cuckoo search optimization (CSO) is a meta-heuristic algorithm that draws inspiration from the breeding behavior of cuckoo birds. Developed by Xin-She Yang and Suash Deb in 2009 [11], The CSO algorithm takes inspiration from the unique breeding behavior of cuckoo birds, where they lay their eggs in the nests of other birds to ensure the survival of their species. The algorithm mimics this behavior by generating a population of potential solutions, represented as "nests", and candidate solutions, represented as "eggs". The CSO algorithm then uses a set of search operators to iteratively update the candidate solutions and nests, with the objective of finding the optimal solution to a given problem. The algorithm's ability to efficiently explore the solution space and avoid local optima makes it a popular choice for solving complex optimization problems in various fields. Overall, the CSO algorithm's unique approach to optimization and its ability to adapt to changing environments have made it a powerful tool for solving challenging problems, from machine learning to engineering and beyond.

Algorithm- 1. Cuckoo Search Optimization (CSO)

```
Initialize population of n host nests randomly
Set maximum number of iterations
Set fraction of abandoned nests (pa)
Set step size factor (alpha)
Evaluate fitness of each host nest
While (not converged) do:
   Choose a cuckoo solution randomly
   Generate a new solution using Lévy flights
   Evaluate fitness of the new solution
   Choose a host nest randomly
   If (fitness of new solution > fitness of host nest) then
      Update the host nest with the new solution
   End If
   Sort the host nests by their fitness
   Replace pa fraction of the worst nests with new randomly generated ones
End While
Return the best host nest found
```

The pseudo-code of the Cuckoo search algorithm is as follows:

In this pseudocode, the algorithm begins by initializing a population of host nests randomly. It then sets various parameters, including the maximum number of iterations, the fraction of abandoned nests (pa), and the step size factor (alpha). The fitness of each host nest is then evaluated. The algorithm then enters a loop where it randomly chooses a cuckoo solution and generates a new solution using Lévy flights. It evaluates the fitness of the new solution and chooses a host nest randomly. If the fitness of the new solution is better than the fitness of the chosen host nest, the host nest is updated with the new solution.The host nests are then sorted by their fitness, and the pa fraction of the worst nests are replaced with new randomly generated ones. This process continues until a convergence criterion is met, and the best host nest found is returned as the solution.

Cuckoo search algorithm has been found to be promising for feature selection in various studies [12-14]. This is because the cuckoo search algorithm can effectively explore the search space and find good feature subsets that can improve the performance of a classification model. The algorithm can also handle the issue of feature redundancy and can select a compact set of features that can achieve high classification accuracy. Additionally, the cuckoo search algorithm has been shown to be computationally efficient and can handle high-dimensional feature spaces. We are using Binary Cuckoo Search optimization algorithm for important feature selection in decision tree.

4. Proposed System Methodology

In this paper we are applying Cuckoo search algorithm for selecting features which creates efficient decision trees. The basic idea is to represent each feature subset as a "bird's egg" and each feature subset as a "bird's nest." The fitness of each feature subset is evaluated using a classification algorithm. The Cuckoo search algorithm is then used to search for the optimal feature subset. We propose Cuckoo search based Decision Tree (CSDT) using optimized features.

Here is the pseudocode for CSDT:

1. Define the problem and input data:
   - Let $\boldsymbol{Dt}$ be the input data matrix of size $(r\ X\ c)$ where r is the number of samples and c is the number of features.
   - Let y be the corresponding class labels of size $(r\ X\ 1)$.
2. Initialize the cuckoo search algorithm:
   - Set the population size $\boldsymbol{nest}_{\ i}$ and the maximum number of iterations T.
   - Generate $\boldsymbol{nest}_{\ i}$ random binary feature vectors $\boldsymbol{x_i}$ of size $(m\ X\ 1)$representing the candidate solutions.
   - Evaluate the fitness function $\boldsymbol{f}(\boldsymbol{x_i})$ for each solution $\boldsymbol{x_i}$, where $\boldsymbol{f}$ is a performance metric such as classification accuracy.
3. Repeat for $t = 1\ to\ T$:
   - Generate a new candidate solution $\boldsymbol{x_{new}}$by performing Lévy flights on a randomly selected solution $\boldsymbol{x_i}$.
   - Evaluate the fitness function $\boldsymbol{f}(\boldsymbol{x_{new}})$for the new solution.
   - If $\boldsymbol{f}(\boldsymbol{x_{new}}) > f(\boldsymbol{x_i})$, then replace $\boldsymbol{x_i}$ with $\boldsymbol{x_{new}}$.
   - Abandon a fraction pa of the worst solutions and generate new solutions to replace them.
   - Sort the solutions in descending order of fitness and select the top N solutions to form the new population.
4. Output the best solution found after T iterations.

CSDT is effective in selecting the number of features while maintaining or improving the performance of the decision tree. However, like all metaheuristic algorithms, its performance depends on the problem and the parameter settings, and it may not always find the optimal subset of features. The entire CSDT algorithm is defined in Algorithm 2 as follows:

Algorithm-2:Cuckoo Search based Decision Tree (CSODForest)

Input: Training Dataset $\boldsymbol{Dt}$
Output: Global best position $\hat{\boldsymbol{g}}$.
Auxillary: Fitness vector $\boldsymbol{f}$ with size $\boldsymbol{m}$ and variables $\boldsymbol{acc\_t}$, $\boldsymbol{mfit}$, $\boldsymbol{gfit}$ and $\boldsymbol{mindex}$.
Begin
for each nest $\boldsymbol{nest}_{\boldsymbol{i}}$ do $(\forall \boldsymbol{i} = \boldsymbol{1, 2, \ldots . . m})$ do
  for each dimension $\boldsymbol{j}$ $(\forall \boldsymbol{j} = \boldsymbol{1, 2, \ldots . . d})$ do
    $\boldsymbol{x_j^i(0) \leftarrow Random\{0, 1\};}$
  end
  $\boldsymbol{f\, acc\_t_i \leftarrow -\infty}$ **;**
end
$\boldsymbol{gfit \leftarrow -\infty}$ **;**
for each iteration $\boldsymbol{k}$ $(\boldsymbol{k = 1, 2, \ldots . . K})$ do
  for each nest $\boldsymbol{nest}_{\boldsymbol{i}}$ $(\forall \boldsymbol{i} = \boldsymbol{1, 2, \ldots . . m})$ do
    Evaluate $\boldsymbol{nest}_{\boldsymbol{i}}$ in which $\boldsymbol{x_j^i(k) \neq 0}$ $\forall \boldsymbol{j = 1, 2, \ldots . . d}$ **;**
    Calculate $\boldsymbol{acc\_t}$
    if$\boldsymbol{(acc_t > facc_i)}$
      $\boldsymbol{facc_i \leftarrow acc\_t}$ **;**
      for each dimension $\boldsymbol{j}$ $(\forall \boldsymbol{j = 1, 2, \ldots . . d})$ do
        $\boldsymbol{\hat{x}_j^i = x_j^i(k)}$
      end
    ifend
  end
  $\boldsymbol{[mfit, mindex] = max(f)}$ **;**
  if $\boldsymbol{(mfit > gfit)}$
    $\boldsymbol{gfit \leftarrow mfit}$ **;**
    for each dimension $\boldsymbol{j}$ $(\forall \boldsymbol{j = 1, 2, \ldots . . d})$ do
      $\hat{g}^j = \boldsymbol{x_{mindex}^j(k)}$
    end
  ifend
  for each nest $\boldsymbol{nest}_{\boldsymbol{i}}$ do $(\forall \boldsymbol{i} = \boldsymbol{1, 2, \ldots . . m})$ do
    for each dimension $\boldsymbol{j}$ $(\forall \boldsymbol{j} = \boldsymbol{1, 2, \ldots . . d})$ do
      $\boldsymbol{x_j^i(k) \leftarrow x_j^i(k-1) + \alpha \oplus Lévy}$
      if $\left(\boldsymbol{\sigma < \frac{1}{1+e^{x_j^i(k)}}}\right)$ then
        $\boldsymbol{x_j^i(k) \leftarrow 1}$
      else
        $\boldsymbol{x_j^i(k) \leftarrow 0}$
      ifend
    end
  end
end
Evaluate tree using a testing set $\boldsymbol{T_{test}}$

CSDT algorithm is used to select the most important features in a binary feature space, where each feature can either be selected or not selected. The Binary Cuckoo Search algorithm works by first initializing a population of candidate feature subsets, where each subset is represented by a binary vector indicating which features are selected. Then, the algorithm performs a series of iterations to refine the feature subsets, guided by a fitness function that evaluates the performance of each subset. During each iteration, the algorithm randomly generates new candidate solutions by performing a random walk using the Lévy flight distribution. The algorithm then

evaluates the fitness of the new solutions and replaces the worst-performing solutions in the population with the new solutions. To enforce the binary constraint on the feature selection, the algorithm uses a threshold function to convert the real-valued solutions generated by the Lévy flight into binary vectors. The threshold function maps values above a certain threshold to 1 (selected), and values below the threshold to 0 (not selected). The algorithm continues iterating until a stopping criterion is met, such as a maximum number of iterations or convergence of the best solution. Finally, the best feature subset found by the algorithm is returned as the solution.

Overall, the Binary Cuckoo Search optimization algorithm provides a powerful and efficient way to perform feature selection in binary feature spaces, and has been shown to outperform other feature selection methods on a range of benchmark datasets.

5. Experimental Results

5.1 Dataset Description

The efficacy of the CSDT algorithm is assessed on six distinct benchmark datasets sourced from the UCI machine learning repository. The performance of pruned CSDT is evaluated on 6 standard datasets obtained from the UCI machine learning repository, which include numerical and categorical attributes from various domains. These datasets differ in size, and more information about them can be found in Table 1.

Table 1. Summary of datasets

Dataset	Instances	Attributes	Classes
Chess	3196	36	2
Credit Approval	653	15	2
Image Segmentation	2310	19	7
Ionosphere	351	34	2
Statlog Vehicle	846	18	4
Sonar	208	60	2

5.2 Comparison of Accuracy of CSDT with other decision tree algorithms

A comparison of the outcomes of CSDT is done with C4.5 [5], CART [16], and Random Tree[17] to demonstrate the effectiveness. To assess the performance of the proposed approach, a 10-fold cross-validation technique is employed, where each dataset is partitioned into ten subsets of equal size. In every fold, training is done on nine parts of the dataset and one part is used for evaluating the performance on the CSDT algorithm. Table 2 shows the Accuracy of the proposed CSDT with other methods.

Table 2. Comparison of Accuracy of CSDT with other methods

Dataset	**C4.5**	**CART**	**RandomTree**	**CSDT**
Chess	76.74	74.12	77.34	**79.24**
Credit Approval	86.09	86.01	80.57	**87.21**
Image Segmentation	96.92	**97.12**	95.40	97.23
Ionosphere	91.45	91.23	87.74	**90.86**
Statlog Vehicle	86.74	87.42	86.56	**88.42**
Sonar	72.57	72.36	73.12	74.38

From Table 2, we can observe that accuracy for many dataset is better in the CSDT algorithm. CSDT algorithm gives more than 2% greater accuracy datasets.

6. Conclusion

We present a novel approach to feature selection using the Cuckoo search optimization algorithm. The goal is to select a subset of features from a large set that can improve the accuracy of decision trees while reducing computational and storage costs. Our proposed algorithm, CSDT, uses the Cuckoo search algorithm to select high-quality features, which are then used to create efficient and accurate decision trees. We evaluate the performance of CSDT on six datasets from the UCI machine learning repository using a 10-fold cross-validation method. Results show that CSDT prunes decision trees with higher accuracy and avoids overfitting compared to other decision trees algorithms.

References

[1] Tirelli, T. and Pessani, D., 2011. Importance of feature selection in decision-tree and artificial-neural-network ecological applications. Alburnus alburnus alborella: A practical example. Ecological informatics, 6(5), pp.309-315.

[2] Ying, X., 2019, February. An overview of overfitting and its solutions. In Journal of physics: Conference series (Vol. 1168, p. 022022). IOP Publishing.

[3] Malhi, A. and Gao, R.X., 2004. PCA-based feature selection scheme for machine defect classification. IEEE transactions on instrumentation and measurement, 53(6), pp.1517-1525.

[4] Kasliwal, B., Bhatia, S., Saini, S., Thaseen, I.S. and Kumar, C.A., 2014, February. A hybrid anomaly detection model using G-LDA. In 2014 IEEE International Advance Computing Conference (IACC) (pp. 288-293). IEEE.

[5] Quinlan, J.R., 1993. Program for machine learning. C4. 5.

[6] Kohavi, R. and John, G.H., 1997. Wrappers for feature subset selection. Artificial intelligence, 97(1-2), pp.273-324.

[7] Loughrey, J. and Cunningham, P., 2005. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In Research and Development in Intelligent Systems XXI: Proceedings of AI-2004, the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 33-43). Springer London.

[8] Hsu, W.H., 2004. Genetic wrappers for feature selection in decision tree induction and variable ordering in Bayesian network structure learning. Information Sciences, 163(1-3), pp.103-122.

[9] Theodoridis, P.K. and Gkikas, D.C., 2020. Optimal feature selection for decision trees induction using a genetic algorithm wrapper-a model approach. In Strategic Innovative Marketing and Tourism: 8th ICSIMAT, Northern Aegean, Greece, 2019 (pp. 583-591). Springer International Publishing.

[10] Zhang, Y., Wang, S. and Wu, L., 2012. Spam detection via feature selection and decision tree. Advanced Science Letters, 5(2), pp.726-730.

[11] Yang, X.S. and Deb, S., 2009, December. Cuckoo search via Lévy flights. In 2009 World congress on nature & biologically inspired computing (NaBIC) (pp. 210-214). IEEE.

[12] Rodrigues, D., Pereira, L.A., Almeida, T.N.S., Papa, J.P., Souza, A.N., Ramos, C.C. and Yang, X.S., 2013, May. BCS: A binary cuckoo search algorithm for feature selection. In 2013 IEEE International symposium on circuits and systems (ISCAS) (pp. 465-468). IEEE.

[13] Gunavathi, C. and Premalatha, K., 2015. Cuckoo search optimisation for feature selection in cancer classification: a new approach. International journal of data mining and bioinformatics, 13(3), pp.248-265.

[14] Aziz, M.A.E. and Hassanien, A.E., 2018. Modified cuckoo search algorithm with rough sets for feature selection. Neural Computing and Applications, 29, pp.925-934.

[15] Dua, D., & Graff, C. (2017). UCI Machine Learning Repository. Retrieved from http://archive.ics.uci.edu/ml.

[16] L. Breiman, J. Friedman, R. Olshen, C. Stone, “Classification and Regression Trees”, Wadsworth International Group, CA, U.S.A, 1985.

[17] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.