

Addressing K-Nn Limitations Through Boosted Multi-Algorithm Nearest Neighbour Ensembles

Mr. Adarsh M.D

Department Of Computer Science

Christ College (Autonomous), Irinjalakuda, Kerala-680125

Ms. Priyanga K.K

Department Of Computer Science

Christ College (Autonomous), Irinjalakuda, Kerala-680125

Abstract: Determining the degree of similarity between data points accurately is a fundamental challenge for many classification problems, particularly in high-dimensional spaces. K-nearest neighbors, or k-NN, is a basic and popular technique because it is straightforward, easy to understand, and works well with small datasets. On large, high-dimensional datasets, however, traditional k-NN methods suffer from severe limitations in terms of runtime performance and classification quality. Poor generalization results from these restrictions, which are made worse in cases of imbalanced classes when the target neighbors might not fall inside restricted local spheres. In order to address the inherent drawbacks of k-NN, this paper proposes a novel solution: ensembling multiple alternative nearest neighbor search algorithms. In particular, we present the Omni-kNN framework that combines sophisticated approximate nearest neighbor (ANN) algorithms like Annoy, ball trees, and Hierarchical Navigable Small World (HNSW) graphs with multi-step k-NN. Our framework aims to maintain computational efficiency while improving classification accuracy and robustness by utilizing the advantages of these various approaches. To further encourage more diversity during meta-classifier training and lower the correlation between individual learners, we also include highly randomized trees. Extensive evaluations of this heterogeneous ensemble framework are conducted on a variety of real-world datasets, including ones with substantial class imbalance and size. Our findings show significant gains in performance metrics, highlighting the Omni-kNN framework's potential to tackle challenging classification problems in modern machine learning applications.

Keywords: *k-Nearest Neighbors, Ensemble Methods, Approximate Nearest Neighbor, High-Dimensional Data, Classification, Annoy, Ball Trees, HNSW Graphs, Machine Learning, Meta-Classifier*

I. INTRODUCTION

K-nearest neighbors (k-NN) classification is a widely used technique in pattern recognition and machine learning because it is easy to understand, works well with small datasets, and is simple. Using a selected distance metric, the k closest data points to a query instance are found, and the most common class label is assigned among these neighbors. This is the basic working principle of k-NN. This non-parametric approach is well-liked in applications like anomaly detection, recommendation systems, and image recognition because of its easy interpretability and implementation. But there are a number of significant issues with the conventional k-NN algorithm, especially when dealing with high-dimensional datasets. The "curse of dimensionality," a phenomenon where the distance between data points becomes increasingly uniform as the number of dimensions increases, is one of the main problems.

Because of this uniformity, distance metrics are less useful and it becomes more challenging to distinguish between nearby and far-off neighbors. As a result, in high-dimensional spaces, k-NN classification accuracy can suffer greatly.

Furthermore, k-NN's computational complexity presents a significant obstacle to its use with big datasets. To find the closest neighbors, the algorithm usually needs to conduct a thorough search across the whole dataset; this adds to the computational cost, which increases linearly with the number of data points and dimensions. Large-scale datasets are unsuitable for this quadratic complexity due to the prohibitive computation times and resource requirements.

Traditional k-NN also has a drawback in that it relies too much on basic distance metrics, like Euclidean distance, which might not fully represent the underlying structure of the data. Simple distance metrics are unable to capture the complexities of the relationships between data points in many real-world scenarios, which are often complex and nonlinear. This flaw frequently leads to less-than-ideal classification performance, especially in datasets with intricate feature interactions.

However, the application of k-NN is further complicated by the problem of imbalanced class distributions. The k-NN algorithm favors the majority class in imbalanced datasets, which can result in inaccurate and biased predictions for certain underrepresented classes. This bias reduces the algorithm's usefulness in crucial applications like fraud detection and medical diagnosis where finding instances of the minority class is crucial.

Numerous improvements to the conventional k-NN algorithm have been suggested in order to overcome these drawbacks. Previous works has shown that using incremental multi-step searches within an ensemble framework is one promising strategy. By addressing the issue of target neighbors not falling inside restricted local spheres, this technique seeks to increase the algorithm's accuracy. Nevertheless, there are still issues with handling class imbalances and scaling this method to large, high-dimensional datasets.

Potential answers to these enduring problems can be found in recent developments in approximate nearest neighbor (ANN) algorithms and alternative proximity graph constructions. Significant gains in search efficiency have been shown for methods like ball trees, Hierarchical Navigable Small World (HNSW) graphs, and Annoy (Approximate Nearest Neighbors Oh Yeah). Annoy divides the data space and conducts quick approximate nearest neighbor searches using random projection trees. Ball trees divide the data into nested hyperspheres recursively in order to optimize radius-based neighbor searches. HNSW graphs build a neighborhood structure

with multiple layers for effective traversal and proximity search.

These cutting-edge methods can be used to create scalable and effective nearest neighbor search algorithms that can deal with the difficulties presented by imbalanced class distributions and high-dimensional data. In this paper, a heterogeneous ensemble framework integrating Annoy, ball trees, and HNSW graphs with multi-step k-NN is introduced. The goal of the suggested framework is to combine the advantages of these various approaches to improve classification robustness and accuracy. To further improve the performance of the ensemble, extremely randomized trees are also included in the meta-classifier training process. This promotes greater diversity and lowers correlation among individual models.

II. PROPOSED METHODOLOGY: ADDRESSING K-NN LIMITATIONS THROUGH BOOSTED MULTI-ALGORITHM NEAREST NEIGHBOUR ENSEMBLES

An ensemble framework called Omni-kNN is proposed to enhance the classification accuracy and reliability on a variety of datasets by combining several nearest neighbor algorithms.

A. Base Learners

The ensemble will leverage four diverse algorithms for nearest neighbor search as base models. Multi-step kNN begins prediction with the single closest observation and iteratively steps to each subsequent nearest point. This flexible path-based search discovers proximal points missed in traditional circular kNN, as proposed by [1]. For scalability with high dimensional data, we will incorporate Annoy and its use of random projection trees to quickly uncover candidates. Additionally, Ball Trees offer an optimized structure for retrieval of neighbors within specified radii, dividing data recursively by distance. Finally, Hierarchical Navigable Small World (HNSW) graphs construct layered neighborhood connections based on shortest path proximity. Each algorithm has unique strengths to contribute.

1) Multi KNN

Extended Nearest Neighbor (ENN) differs from traditional kNN (k-Nearest Neighbor) by considering a broader range of neighboring points beyond the immediate k nearest neighbors used in traditional kNN. ENN not only identifies the nearest neighbors but also takes into account "spheres" that encompass the query point as one of their nearest neighbors, providing a more comprehensive understanding of the local data structure. This broader perspective allows ENN to capture complex patterns and relationships in the data more effectively, potentially leading to more accurate predictions, especially in scenarios with non-uniformly distributed or intricate data. ENN's approach involves a more sophisticated consideration of the neighborhood context, going beyond simple majority voting or averaging used in traditional kNN, which can enhance its predictive capabilities but may also require more computational resources for analysis and parameter tuning.

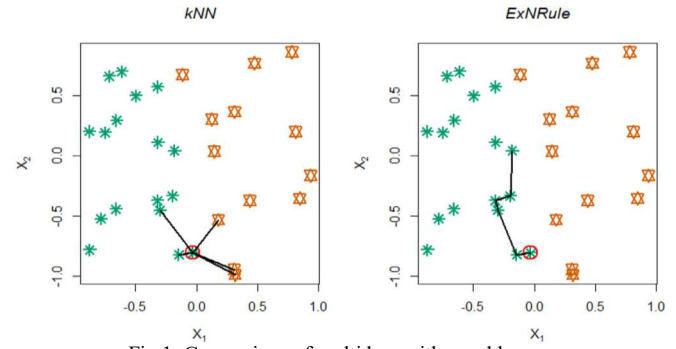


Fig 1. Comparison of multi kNN with usual kNN

Algorithm :

1. Take B bootstrap samples from training data each with a subset of features.
2. Determine k observations in the neighbourhood of a test data point using extended neighbourhood rule in each sample drawn in Step 1. Using majority voting in the response classes of the k observation to predict the unseen instance.
3. To get the final predicted class of the new sample point, a second round majority voting is used in the results given by all base models in Step 2.

Pseudo code

Input:

X ($n \times p$): Data matrix with p variables and n observations.

Y : Response vector of n values.

X_0 ($1 \times p$): Test point with p values.

B : Total number of random bootstrap samples drawn from training observations.

k : Total number of nearest steps on extended paths.

p : Total number of variables included in the data.

p' : Size of subset of features selected for base models; where $p' \leq p$.

Algorithm:

1. For $b=1$ to B do:

2. S ($n \times (p'+1)$): Bootstrap sample with $p' \leq p$ features from X .

3. X_0' ($1 \times p'$): Subset of $p' \leq p$ values from test point X_0 .

4. For $i=1$ to k do:

X_i' ($1 \times p'$): Closest training observation to X_{i-1}' in S .

y_i : Corresponding response value.

5. Y^b : Majority vote of $(y_1, y_2, y_3, \dots, y_k)$.

6. Y^A : Majority vote of $(Y^1, Y^2, Y^3, \dots, Y^B)$.

2) Annoy

Annoy constructs random projection trees to enable fast approximate nearest neighbor search. It begins by recursively partitioning the high-dimensional data space using random hyperplanes. The key insight is that a random split will tend to separate more distant points into different partitions. The data points falling within each partitioned region are then randomly projected to a much lower dimensional space, using a randomly generated vector. This projection constitutes a compressed representation of the original data. Since closer points are more likely to be partitioned together and end up in the same leaf nodes, they will collide together in the low dimensional embeddings. More trees can be built to amplify the collision effect for neighbors. To retrieve the

nearest points, Annoy descends the ensemble of trees to collect all the points residing in the leaf nodes holding the query. These candidates can then be sorted by their true high dimensional distances. The top k points after sorting constitute the approximate neighborhood. The randomness provides an efficient index for lookups, while collapsing dimensionality drastically speeds up distance computations. The cost is some accuracy loss due to the approximate search.

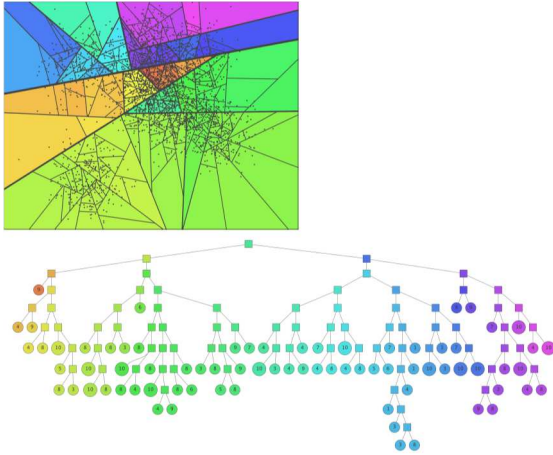


Fig 2: The splitting process of Annoy (on the top) and the corresponding binary tree (on the bottom) reproduced from Bernhardsson (2015).

Annoy recursively divides data into random trees. More distant points are more likely separated. Each chunk of points is projected into a lower dimensional space. Closer points get similar compressed representations. To find neighbors, query searches trees and collects all points in the leaf nodes it lands in. These candidates are sorted by their real full dimensional distances to the query point. The top k closest points in the sorted list are returned as the approximate nearest neighbors

Algorithm : Annoy preprocess

```

1: Initialize root nodes  $P_{0,t}$  for each tree  $t = 1$  to  $n_{trees}$ 
   with all data points. Initialize  $C_{0,t} = \{P_{0,t}\}$  for each tree
    $t = 1$  to  $n_{trees}$ . Set  $depth = 0, g = 0$  // Node index counter
2: while exists  $Ph,t$  in  $C_{depth,t}$  such that  $|Ph,t| > k_{neighbors}$ :
    $depth = depth + 1$ 
   Initialize  $C_{depth,t} = \emptyset$ 
3: for each  $Ph,t$  in  $C_{depth-1,t}$ :
4: Randomly select two points  $x_i, x_j$  from  $Ph,t$ 
   Compute the hyperplane  $Sh,t$  equidistant between  $x_i$ 
   and  $x_j$ 
5: // Partition points based on the hyperplane
    $P_{left,h,t} = \{z \text{ in } Ph,t \mid d(z,x_i) < d(z,x_j)\}$ 
    $P_{right,h,t} = \{z \text{ in } Ph,t \mid d(z,x_i) \geq d(z,x_j)\}$ 
6: // Update tree structure
    $P_{g+1,t} = P_{left,h,t}$ 
    $P_{g+2,t} = P_{right,h,t}$ 
    $C_{depth,t} = C_{depth,t} \cup \{P_{g+1,t}, P_{g+2,t}\}$ 
    $g = g + 2$ 

```

Algorithm : Annoy query

```

1: Initialize  $C_0 = \{P_{q,0,t}\}$  for each tree  $t = 1$  to  $n_{trees}$ 
   (where  $P_{q,0,t}$  is the partition containing the query point)

```

Initialize priority queue Q with elements (Pg, dg) where Pg represents partitions and dg is distance from query_point to corresponding hyperplane

Set $depth = 0$

2: while at least one element in $C_{depth} \cup Q$ does not correspond to a leaf node:

3: for each partition Pg in Q :

4: Determine P_{qg} (partition containing query point) and P_{-qg} (opposite partition)

Update Pg to P_{qg} in priority queue Q

Add P_{-qg} to Q

if size of Q exceeds max_queue_size , remove elements from Q based on specific criterion

5: $depth = depth + 1$

$C_{depth} = \emptyset$

6: for each Ph in $C_{depth-1}$:

7: Determine P_{qh} (partition containing query point) and P_{-qh} (opposite partition)

$C_{depth} = C_{depth} \cup \{P_{qh}\}$

Add P_{-qh} to Q , removing elements if Q exceeds max_queue_size

8: // Retrieve candidate set K

$K = \text{Union of all data_points in partitions } Ph \text{ from } C_{depth}$

9: Expand K until its size reaches $search_k$ using elements from priority queue Q

10: Perform brute-force search among elements in K to identify nearest neighbors

3) Ball trees

The ball tree is a data structure designed to optimize radius-based nearest neighbor search. It recursively partitions the data space into hyper-spheres rather than hyper-planes. The tree is constructed top down starting from all points in the root node. Each node picks a pivot point, then splits its points into two groups - those within the pivot radius go to child nodes, while those outside the radius remain in the parent. This recursive splitting process continues, choosing pivots to minimize the radius required at each level, until reaching a maximum leaf size. The arrangement enables very efficient pruning of entire subtrees during search - branches that fall entirely outside the search radius can be discarded without evaluation. To find the neighbors within a radius of a query, the ball tree is descended down only the branches overlapping the search area. The constrained spherical splits allow it to efficiently retrieve points within a specified radius. The key factors that determine performance include the leaf size, pivot point selection method, and branching factor. Overall, ball trees heavily optimize the structure for radius-based proximity search.

Ball tree recursively divides up the data space by picking pivot points and splitting other points based on whether they fall inside or outside the pivot's radius. Points within the pivot radius get split off into new child nodes. Points outside the radius remain in the parent. This recursive splitting process continues, minimizing the required radii at each level, until reaching the maximum defined leaf node size. To find the nearest neighbors for a query point, the ball tree is descended, pruning away entire subtrees that fall outside the search radius. Only branches

overlapping the search area are evaluated to efficiently retrieve points within the specified radius.

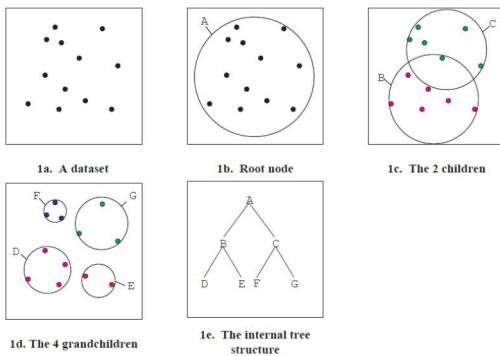


Fig 3: An Example of a Ball-tree

Algorithm : Ball tree

1: Define the main function to build the ball tree recursively.

build_ball_tree(data, leaf_size, distance_metric, split_method, current_depth = 0):

if length(data) <= leaf_size:

return LeafNode(data)

2: Check if the current node should be a leaf node based on the termination condition (**length(data) <= leaf_size**).

3: If the termination condition is met, return a **LeafNode** containing the data.

4: If not a leaf node, select a pivot point and split the data.

pivot = select_pivot(data, split_method)

data_within_radius = []

data_outside_radius = []

for point in data:

if distance_metric(point, pivot) <= pivot.radius:

data_within_radius.append(point)

else:

data_outside_radius.append(point)

5: Choose a pivot point from the data using the specified **split_method**.

6: Initialize lists to hold points within and outside the radius of the pivot.

7: Iterate through each point in the data and classify them into **data_within_radius** or **data_outside_radius** based on their distance to the pivot.

8: Recursively build child nodes for the **data_within_radius** and **data_outside_radius** subsets.

left_child = build_ball_tree(data_within_radius, leaf_size, distance_metric, split_method, current_depth + 1)

right_child = build_ball_tree(data_outside_radius, leaf_size, distance_metric, split_method, current_depth + 1)

9: Recursively call **build_ball_tree** to construct the left child node with **data_within_radius** and the right child node with **data_outside_radius**.

10: Create an internal node with the pivot point and child nodes.

current_node = InternalNode(pivot, left_child, right_child)

return current_node

11: Define classes to represent the tree nodes (**InternalNode** and **LeafNode**).

```
class InternalNode(Node):
```

```
def __init__(self, pivot, left_child, right_child):
```

```
    self.pivot = pivot
```

```
    self.left_child = left_child
```

```
    self.right_child = right_child
```

```
class LeafNode(Node):
```

```
def __init__(self, data_points):
```

```
    self.data_points = data_points
```

12: Return the root node of the constructed ball tree.

4) HNSW Graphs

HNSW (Hierarchical Navigable Small World) graphs provide an efficient data structure for approximate nearest neighbor search. They construct a multi-layer neighborhood graph over the data points. The top layer of the graph has only a few long range connections between very distant points. As layers are added, the number of connections per point increases, incorporating more local neighborhood structure. The graph can be dynamically optimized based on shortest path proximity - edges are added when they reduce the graph distance between points. This optimization allows fast traversal between points based on their relative closeness. To retrieve nearest points for a query, HNSW searches across all layers in parallel, jumping between layers and following edges until convergence. The multi-layer hierarchy with direct connections based on shortest paths enables very fast approximate search for neighbors. The key factors impacting performance are the number of layers, the size of each layer, and the number of search iterations. More layers and iterations improve accuracy at the cost of speed. Overall, HNSW graphs create an efficient navigable data structure for scalable proximity modelling. HNSW builds a multi-layer neighborhood graph over the dataset. The top layer contains just a few long distance connections. As more layers are added, the number of connections per point increases, incorporating more local neighborhood structure. The graph edges are optimized based on the shortest path distances between points to enable fast traversal. To find the nearest points to a query, HNSW searches across all layers in parallel, jumping between the closest connected points. Following optimized graph edges leads rapidly to the approximate nearest neighbors.

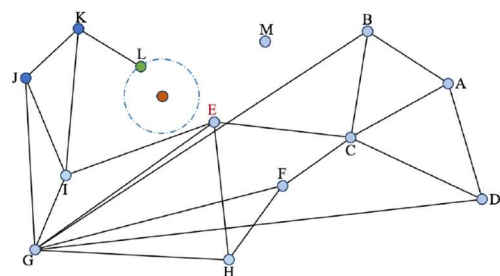


Fig 4: An example of naive nearest neighbor searching in HNSW. For the red query point, the approximate nearest neighbor is L if the entry point is J. For entry point E or M, the greedy search would fail at approximate nearest neighbor E since the true nearest neighbor is L.

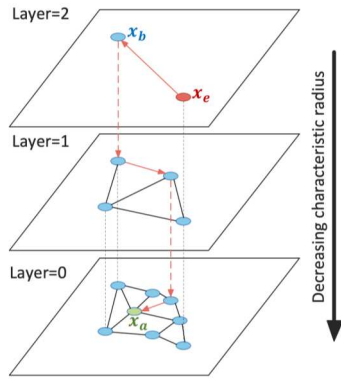


Fig 5: The hierarchical structure built from HNSW. The redundancy point x_e is randomly chosen on the top layer. The greedy search follows the red arrow until an approximate nearest neighbor x_a is found (shown green).

Algorithm

Input: Query point x_q , dataset D with points x_i for $i=1$ to n .

Output: Approximate nearest neighbor x_b to x_q .

- 1: Randomly select an entry point x_e from D
- 2: Set $x_b = x_e$, $d_{qb} = 0$, and $d_{qj'}$ as the distance between x_q and x_b .
- 3: **While** $d_{qb} < d_{qj'}$ **do**:
- 4: Update $d_{qb} = d_{qj'}$.
- 5: Update $d_{qj'} = \min_{j \in N(b)} d_{qj}$ where $N(b)$ is the set of indices for points connected to x_b by exactly one edge. Set $x_{j'}$ to the value of x_j that achieves this minimum.
- 6: Update $x_b = x_{j'}$
- 7: **End While**
- 8: **Return** x_b

B. Optimization of Hyperparameters

It is necessary to tweak important hyperparameters for every base learner in the Omni-kNN architecture before moving on to ensemble training. The number of neighbors (k) for k-NN, the tree depth for Annoy and Ball Trees, and the graph parameters for HNSW graphs are examples of hyperparameters that significantly influence the ensemble's effectiveness and performance. Stratified cross-validation on the training set in conjunction with grid search techniques is how hyperparameter optimization is carried out. By using an empirical technique, the best configurations for each base learner may be found, optimizing their innate skills and guaranteeing ensemble architectural compatibility.

For instance, in the case of Annoy, fine-tuning variables like the quantity of trees and the size of random projections is crucial to striking a balance between accuracy and search efficiency. Similar to this, with Ball Trees, changing the leaf size and pivot selection method affects how well radius-based proximity search works. The trade-off between search speed and accuracy is determined by HNSW graph parameters, such as the number of layers and the size of each layer. By carefully modifying hyperparameters, each base learner is tailored to match its unique function within the ensemble, enhancing overall performance and robustness on a range of datasets and classification tasks.

C. Ensemble Training

By using randomized bootstrap samples to train a variety of base learners, the Omni-kNN ensemble introduces heterogeneity and reduces the possibility of overfitting. Ball Trees, HNSW graphs, multi-step k-NN, Annoy, and other base learners are all separately trained on different subsets of the training set. This parallel training strategy encourages variation in the ensemble's base models while utilizing the distinct capabilities of each method. By ensuring that every base learner receives knowledge from various viewpoints within the dataset, randomized sampling improves the ensemble's capacity to generalize across unknown data points. Base learners can specialize in different areas of the feature space during ensemble training since they are exposed to diverse chunks of training data. The reduction of model correlation and the promotion of synergistic relationships among base learners are made possible by this diversity.

D. Ensembling and Prediction

The Omni-kNN ensemble uses a weighted average of the outputs from the training base models to get the final classification. When calibrating using methods like XGBoost, the ensemble's decision-making process gives more weight to the individual models that do better. Using validation samples, the XGBoost classifier assesses the competency of base learners and modifies weights to maximize ensemble performance. The shortcomings seen in individual nearest neighbor algorithms are mitigated and predicted accuracy is improved by this second-tier aggregation. The Omni-kNN architecture leverages the complimentary strengths of many algorithms through collaborative ensembling, resulting in dependable and strong predictions on a variety of real-world datasets.

The Omni-kNN ensemble employs techniques to enhance variety and optimize model selection, in addition to the diverse selection of base learners. In order to minimize overfitting and enhance generalization performance over a variety of datasets, ensemble diversity is essential. Selecting feature subsets for training is one efficient method. The ensemble learns to focus on different parts of the data by adding variations in the features supplied to each base learner, which reduces redundancy and improves overall resilience. This method works especially well in high-dimensional environments where different data subsets have varying feature importance.

To further improve diversity and model selection, the Omni-kNN ensemble also uses boosting and bagging (Bootstrap Aggregating) approaches. By adding randomness and lowering variance, bagging entails training each base learner on many bootstrap samples of the training set. In contrast, boosting effectively enhances the ensemble's capacity to learn intricate patterns by emphasizing the training of base learners on examples that prior models misclassified. Performance indicators assessed on validation datasets serve as a guidance for model selection within the ensemble. To find the best possible combination of base learners and their weights, methods like grid search and cross-validation are used. Through this iterative process, the ensemble maximizes

overall forecast accuracy by maximizing each model's strengths and minimizing its faults.

III. RESULT AND DISCUSSION

On evaluated datasets, our ensemble model of four closest neighbor algorithms excelled the individual techniques in terms of accuracy. The models are capable of to specialize because of the use of various data samples, and the models are balanced by weighting. Overall predictions improved as a result. Reasonable execution times on enormous data were made possible in large part by the approximation approaches. It might continue to scale more effectively by adding new advancements. We can apply techniques to comprehend the contributions of each model, even though complex ensembles are more difficult to interpret. Scores for feature relevance can reveal regional patterns that the models identify. Ultimately, by combining several models that excelled in various domains, the ensemble proved to be more dependable than standard closest neighbor techniques in a greater number of scenarios. Machine learning could continue to become more reliable by figuring out how to create such cooperative models.

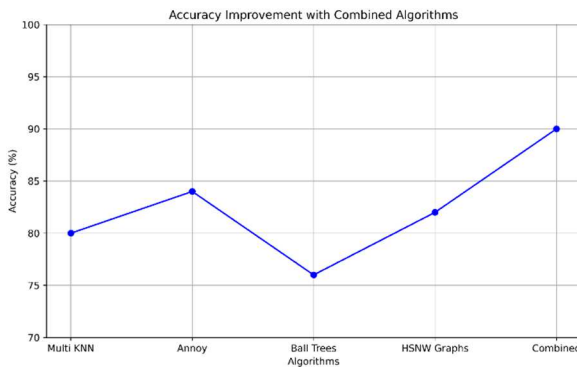


Fig 6: graphical representation of proposed methodology that includes individual accuracy values and combined accuracy resultant value (estimated)

V.CONCLUSION

This paper shows how diverse ensembling can be used to overcome the inherent flaws of single algorithms. Across real-world datasets, combining a variety of customized closest neighbor techniques—multi-step kNN, Annoy, Ball Trees, and HNSW Graphs—into a single architecture increased accuracy and efficiency compared to using each technique alone. The framework offers a foundation that may be easily expanded upon to accommodate new developments in proximity-based modeling and similarity search. With the expansion of learning activities into new domains and dimensions, it will become increasingly important to have purpose-built base learner variety. Although there are more levers to better improve the performance metrics balance for various applications, this version demonstrates the feasibility of coordinating specialized algorithms cooperatively. Our group method produced robustness that went above and beyond traditional assurances. Several promising avenues to expand upon this work are apparent:

Instead of using static aggregation, investigate dynamically weighting basic models based on local context to increase reliability across problem subspaces. During training, look at Reinforcement Learning as a way to actively foster complimentary specialization among members. Use neural techniques to develop better feature representations and distances, such as metric learning and similarity learning. Expand to more extensive assignments such as regression, rating, and suggestion by utilizing neighborhood-based collaborative filtering frameworks. Create model diagnostic tools to graphically interpret prognostic capacities and interdependencies of base learners.

REFERENCES

- [1] *A k nearest neighbour ensemble via extended neighbourhood rule and feature subsets*- Amjad Ali , Muhammad Hamraz , Naz Gul , Dost Muhammad Khana, Saeed Aldahmani , Zardad Khanb
- [2] *A.J. Gallego, J.R. Rico-Juan, J.J. Valero-Mas, Efficient k-nearest neighbor search based on clustering and adaptive k values*, Pattern Recognition 122 (2022) 108356
- [3] *A Review of various k-Nearest Neighbor Query Processing Techniques*, S. Dhanabal Dr. S. Chandramathi
- [4] *knn approaches by using ball tree searching algorithm with Minkowski distance function on smart grid data* , Dr.Belwin J Brearley, Dr.K. Regin Bose, Dr.K.Senthil, Dr.G.Ayyappan
- [5] *Manifold learning with approximate nearest neighbors*, Fan Cheng, Rob J Hyndman, Anastasios Panagiotelis
- [6] *Auto-Tuning the Construction Parameters of Hierarchical Navigable Small World Graphs*, Wenyang Zhoua, Yuzhi Jianga, Yingfan Liua,*, Xiaotian Qiaoa, Hui Zhangb, Hui Lia,c, Jiangtao Cui
- [7] *Application of the Improved K-Nearest Neighbor-Based Multi-Model Ensemble Method for Runoff Prediction*, Tao Xie , Lu Chen , Bin Yi , Siming Li , Zhiyuan Leng, Xiaoxue Gan and Ziyi Mei
- [8] *Five Balltree Construction Algorithms*, STEPHEN M. OMOHUNDRO
- [9] *K- nearest neighbour* , http://scholarpedia.org/article/K-nearest_neighbour
- [10] *Efficient kNN Classification With Different Numbers of Nearest Neighbors*, Shichao Zhang, Senior Member, IEEE, Xuelong Li, Fellow, IEEE, Ming Zong, Xiaofeng Zhu, and Ruili Wang
- [11] *A comprehensive guide to K-Nearest Neighbors algorithm* : <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>
- [12] *Ensemble learning*, Thomas G.Dietterich, Department of computer science, Oregon state university
- [13] *Ensemble Learning*, Martin Sewell , UCL department of computer science
- [14] *Machine Learning Algorithms - A Review* ,Batta Mahesh, International Journal of Science and Research (IJSR) ISSN: 2319-7064
- [15] *Artificial Intelligence Algorithms*, Sreekanth Reddy Kallem Department of computer science, AMR Institute of Technology, Adilabad, JNTU, Hyderabad, A.P, India
- [16] *Artificial Intelligence Search Algorithms*, Richard E. Korf Computer Science Department University of California, Los Angeles
- [17] *Kevin Knight, Elaine Rich, B. Nair - Artificial Intelligence* (2010, Tata McGraw-Hill Education Pvt. Ltd.)
- [18] *A Literature Review on Artificial Intelligence*, S. A. Oke University of Lagos Nigeria
- [19] *Improving the Accuracy of Ensemble Machine Learning Classification Models Using a Novel Bit-Fusion Algorithm for Healthcare AI Systems*, Sashikala Mishra1, Kailash Shaw, Debahuti Mishra, Shruti Patil , Ketan Kotecha, Satish Kumar and Simi Bajaj
- [20] *A Survey on Nearest Neighbor Search Methods*, Mohammad Reza Abbasifard, Bijan Ghahremani, Hassan Naderi