Google Summer of Code 2025

Proposal

GNU OCTAVE

Adding clustering *Searcher classes in statistics package

PERSONAL DETAILS:

Name: Swayam Shah (Sonu)

GitHub: Sonu0305

Discourse: Swayam

Telephone: +91 8291602906

Email: swayamshah66@gmail.com

LinkedIn: swayamshahh

Country: India

Time Zone: IST (GMT+5:30)

Language: English

PROJECT ABSTRACT:

The **GNU Octave** statistics package **lacks** extensible **clustering search classes** and has an **inefficient**, disabled KDTree implementation (**GitHub issue** #151). This project will **implement** `KDTreeSearcher`, `ExhaustiveSearcher`, and `hnswSearcher` **classes** with `knnsearch` and `rangesearch` **methods**, plus a `createns` **helper function**. Using a **C++-compiled** `.oct` library, the KDTree **will be optimized** for faster construction and queries, **surpassing** MATLAB compatibility. This will **enhance Octave's** clustering capabilities for **statistical computing**.

OVERVIEW:

GNU Octave is a premier open-source environment for **scientific computing**, celebrated for its **numerical prowess** and status as a free alternative to MATLAB. Its **rich package** ecosystem fuels diverse applications, from academic research to real-world problem-solving, with the **statistics package** playing a pivotal role in **statistical analysis**. Yet, deficiencies in its **clustering tools** reveal an **opportunity** to elevate its capabilities and retain users who might otherwise turn to proprietary alternatives.

This project rectifies this by introducing advanced searcher classes: **KDTreeSearcher** for **fast k-nearest neighbor** searches using a spatial tree to quickly locate the closest points, **ExhaustiveSearcher** for **brute-force** searches ensuring precision across all data, and **hnswSearcher** for **approximate nearest neighbor searches** leveraging hierarchical navigable small world graphs to balance speed and accuracy. These will include **knnsearch** to find the **k closest points** and **rangesearch** to retrieve all **points within a given radius**, enabling applications like pattern recognition, outlier detection, density estimation, and geospatial analysis. **Optimized** with compiled C++ code, these features will handle large datasets **efficiently**.

As an **active Octave contributor**, I've engaged in **community discussions**, **assisted users**, and **mastered** its **codebase**, preparing me to **meet user needs**.

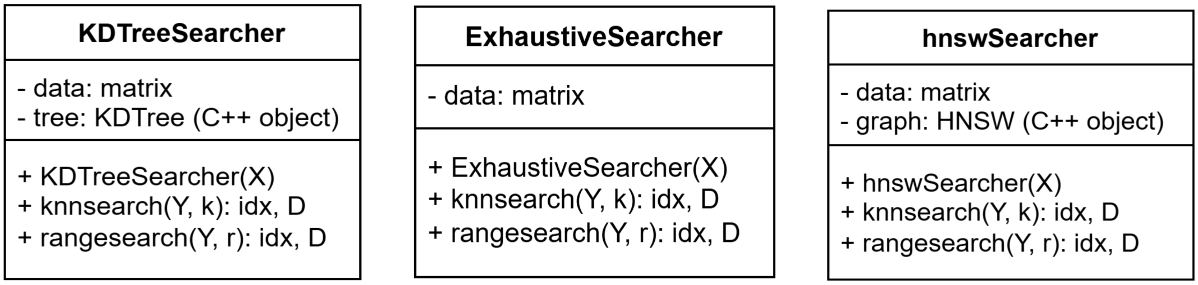
Through Google Summer of Code 2025, I'll apply my **Octave**, **C++**, and **clustering expertise**, collaborating with mentors to **enhance Octave's utility**, **broaden** its appeal, and deepen my role in its **vibrant community**. I am **motivated** to contribute and join the community. It's always great to see people use and appreciate your code.

DETAILED PROJECT DESCRIPTION:

I would like to focus on few important areas like:

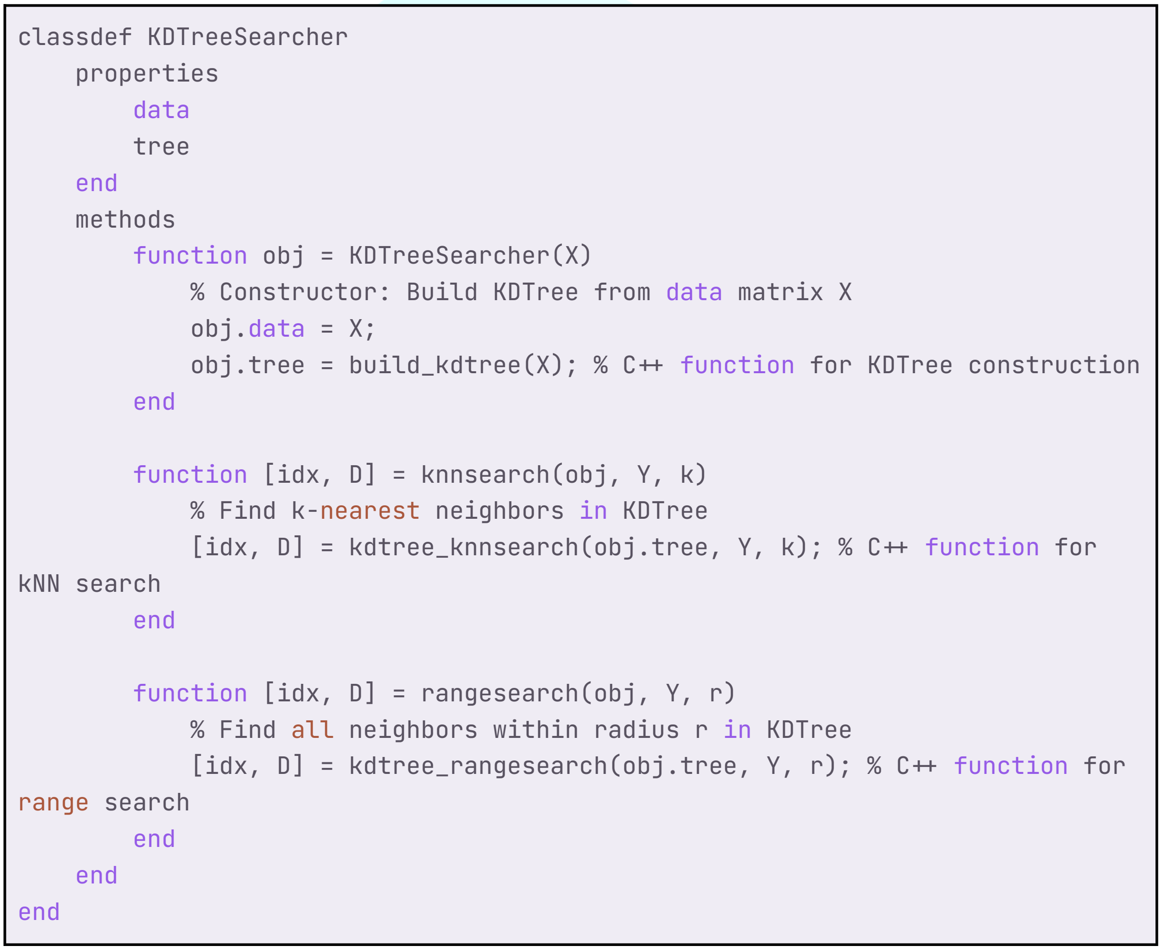
1. **Lack of Modularity**: The current implementation does not use object-oriented programming (OOP) principles. This makes it difficult to extend or modify the functionality.
2. **Inefficient KDTree Implementation**: The KDTree method is slow and poorly implemented.
3. **Missing hnswSearcher**: The current implementation does not include the Hierarchical Navigable Small World (HNSW) algorithm, which is a state-of-the-art method for approximate nearest neighbor searches.
4. **No Helper Function**: There is no createns helper function to create a nearest neighbor searcher object.

I would also like to share the **class diagrams** of proposed implementation of the classes:


KDTreeSearcher
- data: matrix
- tree: KDTree (C++ object)
+ KDTreeSearcher(X)
+ knnsearch(Y, k): idx, D
+ rangesearch(Y, r): idx, D
ExhaustiveSearcher
- data: matrix
+ ExhaustiveSearcher(X)
+ knnsearch(Y, k): idx, D
+ rangesearch(Y, r): idx, D
hnswSearcher
- data: matrix
- graph: HNSW (C++ object)
+ hnswSearcher(X)
+ knnsearch(Y, k): idx, D
+ rangesearch(Y, r): idx, D

I have also **researched** about how and decided about on how I would like the **method signatures** to be:

1. **KDTreeSearcher**:



```
classdef KDTreeSearcher
    properties
        data
        tree
    end
    methods
        function obj = KDTreeSearcher(X)
            % Constructor: Build KDTree from data matrix X
            obj.data = X;
            obj.tree = build_kdtree(X); % C++ function for KDTree construction
        end

        function [idx, D] = knnsearch(obj, Y, k)
            % Find k-nearest neighbors in KDTree
            [idx, D] = kdtree_knnsearch(obj.tree, Y, k); % C++ function for
kNN search
        end

        function [idx, D] = rangesearch(obj, Y, r)
            % Find all neighbors within radius r in KDTree
            [idx, D] = kdtree_rangesearch(obj.tree, Y, r); % C++ function for
range search
        end
    end
end
```

a. Using **KDTreeSearcher** (For Example):

```
X = rand(1000, 10); % 1000 points in 10D space
Y = rand(10, 10);   % 10 query points

% Create KDTreeSearcher
searcher = KDTreeSearcher(X);

% Find 5 nearest neighbors
[idx, D] = searcher.knnsearch(Y, 5);

% Find neighbors within radius 0.5
[idx, D] = searcher.rangesearch(Y, 0.5);
```

2. **ExhaustiveSearcher:**

```
classdef ExhaustiveSearcher
    properties
        data
    end
    methods
        function obj = ExhaustiveSearcher(X)
            % Constructor: Store data matrix X
            obj.data = X;
        end

        function [idx, D] = knnsearch(obj, Y, k)
            % Brute-force k-nearest neighbors search
            D = pdist2(obj.data, Y, 'euclidean'); % Use existing pdist2
function
            [D, idx] = sort(D, 2);
            D = D(:, 1:k);
            idx = idx(:, 1:k);
        end

        function [idx, D] = rangesearch(obj, Y, r)
            % Brute-force range search
            D = pdist2(obj.data, Y, 'euclidean');
            idx = cell(size(Y, 1), 1);
            for i = 1:size(Y, 1)
                idx{i} = find(D(:, i) <= r);
            end
        end
    end
end
```

3. **hnswSearcher:**

```
classdef hnswSearcher
    properties
        data
        graph
    end
    methods
        function obj = hnswSearcher(X)
            % Constructor: Build HNSW graph from data matrix X
            obj.data = X;
            obj.graph = build_hnsw(X); % C++ function for HNSW construction
        end

        function [idx, D] = knnsearch(obj, Y, k)
            % Approximate k-nearest neighbors search using HNSW
            [idx, D] = hnsw_knnsearch(obj.graph, Y, k); % C++ function for
HNSW kNN search
        end

        function [idx, D] = rangesearch(obj, Y, r)
            % Approximate range search using HNSW
            [idx, D] = hnsw_rangesearch(obj.graph, Y, r); % C++ function for
HNSW range search
        end
    end
end
```

a. Using **hnswSearcher** (For Example):

```
X = rand(100000, 100); % 100,000 points in 100D space
Y = rand(10, 100);     % 10 query points

% Create hnswSearcher
searcher = hnswSearcher(X);

% Find 10 approximate nearest neighbors
[idx, D] = searcher.knnsearch(Y, 10);
```

<u>BENEFITS TO OCTAVE COMMUNITY:</u>

1. **Closing the Feature Gap**: Adding **KDTreeSearcher**, **ExhaustiveSearcher**, and **hnswSearcher** aligns Octave's statistics package with **MATLAB**, attracting users needing advanced clustering tools.
2. **Enabling Advanced Research**: Enhanced search capabilities empower data scientists and researchers to **efficiently analyze** large datasets for machine learning and spatial tasks.

3. **Strengthening the Ecosystem**: Improved clustering functionality fosters development of higher-level tools, enriching Octave's open-source ecosystem.

DELIVERABLES:

1. **Searcher Classes**: **KDTreeSearcher**, **ExhaustiveSearcher**, **hnswSearcher** with **knnsearch** and **rangesearch**.
2. **Optimized KDTree**: C++-compiled .oct file **fixing** GitHub issue #151.
3. **createns Function**: Helper to instantiate searchers with options.
4. **Documentation**: Guides and examples for new features.
5. **Tests**: Suite ensuring compatibility and performance.

POSSIBLE DIFFICULTIES:

1. **MATLAB Parity**: Matching MATLAB's behavior might hit edge-case snags. I'll align with its docs and test rigorously.
2. **Octave Integration**: Linking .oct files and new classes could face build issues. My codebase knowledge and early tests will ease this.
3. **C++ Performance**: Optimizing KDTree in C++ for speed and memory may be tricky. I'll use deeper research & mentor input to refine it.

TIMELINE:

1. Once **coding begins**, I will work dedicatedly for **at least 4-5 hours** on **weekdays** (and **6-8 hours** on **weekends**) until the completion of the project. After the **submission** of my proposal (**on or before April 8**), I will start **researching more** about the classes and start noting down important aspects of it.
2. During the **start of** the community bonding period, I have my **end-semester examinations** which will start from **May 5 until May 17**. However, I **promise** to begin my work as early as possible to be on track. After the examinations, **I shall put in more time** to make up for any lost time or lag.
3. I have a **2-month long vacation after May 17** and **no other commitments** in hand, so I will be able to **devote ample time** to the project.

Up till May 8	**Proposal accepted or rejected** ➔ **Deepen familiarity** with Octave's statistics package codebase. ➔ Discuss project **details** with **mentor** Andreas Bertsatos, refining scope and **priorities**.
May 8 - June 1	**Community Bonding Period** ➔ Set up the development environment with the **latest Octave** version and C++ tools. ➔ **Finalize detailed plan**: **prioritize** KDTreeSearcher, identify **potential** HNSW **challenges**.

Week 1 & 2	**Coding Officially Begins!** ➔ Design **KDTreeSearcher** class using **classdef**. ➔ Start C++ **KDTree** implementation (tree-building logic), test basic queries.
Week 3 & 4	➔ **Complete** C++ **KDTree** as .oct file, **integrate** with **KDTreeSearcher**. ➔ Implement **knnsearch** and **rangesearch** for **KDTreeSearcher**, test performance.
Week 5	**Buffer Period** ➔ **Fix any** KDTree issues, **optimize** based on initial tests. ➔ **Review progress** with **mentor**, adjust **remaining tasks**.
Week 6 & 7	➔ Develop **ExhaustiveSearcher** with **knnsearch** and **rangesearch**, test **accuracy**. ➔ Implement **createns function**, **validate** with both KDTree and Exhaustive methods.
Week 8 & 9	➔ Build **hnswSearcher** using **HNSW algorithm**, add methods. ➔ Test hnswSearcher on **large datasets**, **tweak** parameters for **speed/accuracy**.
Week 10	**Buffer Period** ➔ **Resolve bugs** in **hnswSearcher** or prior classes. ➔ Ensure all classes **work seamlessly** with createns, **confirm deliverables** with **mentor**.
Week 11 & 12	➔ Write **comprehensive** documentation and usage examples for **all features**. ➔ **Finalize** testing suite (**performance**, **compatibility**), **prepare submission** (code, docs, report).
Until Pens Down & Post-GSoC	➔ Address **final mentor feedback**, submit well **before the deadline**. ➔ Continue refining features, **contributing to Octave community** long-term.

HISTORY OF MY CONTRIBUTIONS:

1. [**statistics**] **#173**: Adds new functionalities to glmfit
2. [**octave forge**] (**statistics**) **#66388** anovan: suggested help text example not working
3. **#66595** The qz function documentation was not updated with generalized eigenvalues are no longer returned (**Commit Applied**)
4. **#66629** print pdf does not work
5. **#66650** barh with nonzero baseline…
6. [**symbolic**] **#1311**: Modified Test Expectations
7. [**symbolic**] **#1316**: Fix compatibility with BSD tar by supporting both tar and gtar

8. [**symbolic**] **#1319**: Added the pol2cart functionality for Symbolic compatibility
9. [**symbolic**] **#1317**: warning('on', 'all') leads to many Octave warnings
10. [**Discourse**] **#6136**: CIE1931 to spectrum plot
11. [**Discourse**] **#6082**: Need Matlab test of constant functions

ACADEMIC EXPERIENCE AND OTHER ACTIVITIES:

I’m currently pursuing a **B.Tech** in **Computer Engineering** at **VJTI** (currently in Second Year), where I’ve developed **expertise** in **C++** and **Octave**, alongside a strong grasp of **clustering algorithms** and data structures. My academic projects have **sharpened** my skills in **Octave’s classdef system**, which I’ve applied to design extensible statistical tools. As **an active GNU Octave contributor**, I’ve participated in Octave **Discourse discussions**, **assisted users** with troubleshooting, and **gained deep familiarity** with the statistics package codebase. **Solving algorithmic challenges** on LeetCode has further refined my **C++ problem-solving** abilities, ideal for performance-driven implementations. This technical proficiency, paired with **my passion** for open-source and **dedication to Octave**, **equips me** to **enhance** the statistics package **effectively through GSoC**. Hereby, I am also attaching my resume: Swayam Shah.