# Machine Learning Algorithm Visualizer (AlgoViz)

Darsh Patel Priyanshu Shah Chanchal Yadav Tanisha Saini
Bhargav Shekokar Arun Kumar

April 14, 2025

**Abstract**

AlgoViz is an interactive tool designed to make machine learning (ML) algorithms more accessible and comprehensible through dynamic visualizations. The platform enables users to explore, modify, and visualize the behavior of various ML algorithms in real time. It supports supervised learning algorithms like Polynomial Regression, K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), and Artificial Neural Networks (ANN), as well as unsupervised techniques such as K-Means Clustering, DBSCAN, and Principal Component Analysis (PCA). By offering features like real-time parameter tuning, 2D visualizations and guided tutorials, AlgoViz creates an immersive and engaging learning experience for users ranging from beginners to advanced practitioners. This report outlines the development process, architecture, implemented algorithms, interactive features, and future scope of the project.

## 1. Introduction

Machine learning is a rapidly growing field that adds up upon with computational models to solve complex problems. However, the precise nature of ML concepts often makes them difficult for learners to understand. Personally as well it was not that easy to picturize how my model actually is working. For example, understanding how decision boundaries change with different parameters or how clustering algorithms group data points can be difficult without proper steps and labels.

The aim of Algoviz is to provide with an interactive platform where users can visualize the behavior of the algorithm and experiment with model parameters and data in real time. The main focus of AlgoViz is to simplify learning by enabling users to visualize how ML models process data and make predictions.

## 2. Project Architecture

AlgoViz uses client-server architecture in a modular fashion that is easy to scale, modify, and keep up to date over time. Separation of the user interface and the underlying logic allows for smooth data flow and easier, more versatile development.

### 2.1. Backend (Flask)

The backend is developed using Flask, a lightweight Python web framework that facilitates quick development and integration of machine learning functionalities.

- It is responsible for executing various supervised and unsupervised learning algorithms, handling API requests, managing dataset generation, data preprocessing, metric computation, and serving visualization updates.
- The backend processes requests from the frontend, runs the appropriate ML models, and sends back data necessary for real-time visualization and interaction.

To ensure robust performance and high availability, the backend was deployed on Google Cloud, which offers scalable compute resources and reliable uptime. This deployment setup enables the application to handle multiple users and large datasets efficiently, with the potential to scale based on future traffic and feature additions.

### 2.2. Frontend (React)

The frontend of AlgoViz is built using React, which provides a dynamic and responsive interface for users. It features sliders for parameter tuning, dropdowns for algorithm selection, and input fields for working with custom datasets.

- The user interface is designed to offer real-time visual feedback through interactive 2D plots, performance metric displays, and intuitive animations that help users understand algorithm behavior.
- Additionally, the frontend includes guided tutorials and interactive quizzes to support learning and engagement. For deployment, the frontend was hosted on FireBase, which provided an easy-to-use platform for web deployment and hosting.

One of the main advantages of Firebase was its ability to customize the domain name, unlike some other platforms which offered domains in a randomized numeric format. This feature allowed the project to maintain a more professional and user-friendly web address.

## 3. Implemented Algorithms

### 3.1. Polynomial Regression

Polynomial Regression is like linear regression's cooler cousin—it fits a curvy line, using an nth-degree polynomial, to catch those tricky non-linear patterns in your data. In AlgoViz, you can go wild: throw in a random dataset or just plop your own points on the graph. Then, mess around with the learning rate, the max iterations, and the polynomial degree to get the model humming.

As you tweak things, AlgoViz shows the regression curve coming together right before your eyes, plus the equation it's using, the $R^2$ score (how snug the fit is), and the Mean Squared Error (how much it's missing by). It's all there on the graph, making it super easy to see how your changes shape the way the model handles those curvy trends

## 3.2. K-Nearest Neighbors (KNN)

Decision Trees are like a game of 20 questions—they split data step by step based on feature cutoffs, creating a tree-like structure that can handle both classification and regression tasks. In AlgoViz, you're in control: you can tweak settings like the maximum depth of the tree, the minimum samples needed to split a node, and the splitting criterion—choosing between Gini Impurity or Entropy for classification.

You can label training points as class 0, 1, or 2, then toss in interaction or prediction points to see how the model figures things out. AlgoViz brings it to life by showing the tree as it grows and mapping out the decision regions in the feature space, so you can clearly see how the data gets carved up.

Whether you're sorting points into groups or predicting numbers, AlgoViz updates in real time, displaying decision boundaries for classification or regression surfaces for continuous predictions, depending on what mode you're in. It's a hands-on way to understand how tweaking parameters or changing the data shapes the tree's structure and how well it performs!

## 3.3. Decision Tree

Decision Trees split data recursively based on feature thresholds to build a hierarchical tree structure capable of both classification and regression. In AlgoViz, users can interactively adjust parameters like the maximum depth of the tree, minimum samples required to split a node, and the splitting criterion (Gini Impurity or Entropy for classification).

Users can assign training points to classes 0, 1, or 2 and then add interaction and prediction points to observe how the model generalizes. AlgoViz visually presents the evolving tree structure and overlays the corresponding decision regions in the feature space, providing a clear understanding of how the algorithm partitions data.

Whether classifying points into distinct groups or predicting continuous values, the tool offers real-time visualization of decision boundaries or regression surfaces, depending on the selected mode. This interactive setup helps users grasp how changes in parameters and data affect the structure and performance of the tree.

## 3.4. Support Vector Machine (SVM)

Support Vector Machines, or SVMs, are nifty supervised learning tools that work by finding the best possible "line" (or hyperplane) to split data points into groups, aiming to keep the widest gap—or margin—between them. In AlgoViz, you can toggle between Training and Prediction Modes and sort your data into two classes (0 and 1) for classification.

The setup lets you pick from different kernels—Linear, Polynomial, RBF, or Sigmoid—so you can tackle data that's either neatly lined up or trickier to separate. You can also tweak the margin width, which directly shapes the decision boundary the model draws.

While training, AlgoViz makes it visual: the support vectors—those key points that pin down the hyperplane—pop out in dark blue. The decision boundary stands out clearly on the graph, so you can easily see how the model's carving up the classes. Switch to prediction mode, and you can toss in test points to see how the model sorts them based on what it's

learned. It's a great way to get a feel for how SVMs handle everything from simple to complex data splits!

## 3.5. Artificial Neural Network (ANN)

Artificial Neural Networks, or ANNs, are like digital brains made up of layers of interconnected "neurons" that learn complex patterns by tweaking the strength of their connections and using activation functions to decide what gets passed along. In AlgoViz, you're in the driver's seat—you can build your own network from scratch. Want to add hidden layers? Go for it. Need more or fewer neurons in a layer? Your call. You can even pick the activation function—ReLU, Tanh, or Sigmoid—to shape how the network processes info.

You also get to fine-tune the training process by playing with settings like the learning rate (how big the network's steps are when learning), the number of epochs (how many times it runs through the data), and the batch size (how much data it chews on at once). The model lets you switch between Training and Prediction Modes, and your training points can belong to one of three classes (0, 1, or 2).

As you train, AlgoViz keeps things lively by predicting test points on the fly, based on where your network's at with its layers and training data. The interface updates in real time, showing you predicted regions so you can see exactly how your choices—architecture, settings, all of it—affect the network's performance. It's a hands-on way to get a feel for how neural nets learn and make decisions!

## 3.6. K-Means Clustering

K-Means is a neat unsupervised learning trick that sorts data into k clusters by trying to keep points as close as possible to the center (or centroid) of their cluster. It does this by minimizing the total squared distances between each point and its centroid.

In AlgoViz, you get to mess around with the setup. You can pick the type of dataset, decide how many actual clusters there should be, tweak the spread of the clusters, and choose how many data points to throw in. For the clustering itself, you set k (the number of clusters) and decide how many rounds the algorithm should run to settle things down. While it's working, AlgoViz shows you the action live—points getting grouped based on your k, and centroids shifting around with each step. When it's done, you get some handy stats to see how well it worked: the Inertia score tells you how tight your clusters are, and the Silhouette score shows how cleanly separated they are from each other. It's a super visual way to figure out if your clustering is on point!

## 3.7. DBSCAN

DBSCAN, or Density-Based Spatial Clustering of Applications with Noise, is a clever unsupervised algorithm that groups tightly packed data points into clusters while spotting outliers in sparser areas. It relies on two key settings: $\varepsilon$, which sets the radius around a point to check for neighbors, and `MinPoints`, the minimum number of points needed within that radius for a point to be a *core* point.

Points get labeled in three ways: core points have enough neighbors (at least `MinPoints`) within their $\varepsilon$-radius; border points are close enough to a core point but don't have enough neighbors themselves; and noise points are loners, not fitting into any cluster.

In `AlgoViz`, you can play with $\varepsilon$ and `MinPoints` to watch clusters take shape in real time. You can also tweak the animation speed to follow the process step by step. Dotted lines show each point's $\varepsilon$-radius, making it easy to see how clusters form and grow. When it's done, `AlgoViz` gives you a breakdown: how many points were processed, how many clusters formed, and counts of core, border, and noise points. It's a hands-on way to grasp how DBSCAN finds clusters of any shape and picks out noise, all without needing to guess the number of clusters upfront.

### 3.8. Principal Component Analysis (PCA)

Let's talk about Principal Component Analysis (PCA)—it's like a magic trick for simplifying complex data without losing the good stuff. Imagine you've got a ton of data points floating around in a high-dimensional space. PCA swoops in, finds the most important patterns, and shrinks everything down to a simpler form, keeping as much of the data's "vibe" (or variance) as possible.

In AlgoViz, you get to play around with this process hands-on. You can drop in your own data points or let the tool whip up some random ones for you. Want to spice things up? Tweak the noise level to see how it shakes up the data's spread. Once you're ready, hit the Compute PCA button, and AlgoViz works its magic. It figures out the first two principal components (PC1 and PC2), which are like the superstars of your data's story, and shows you their eigenvalues and eigenvectors—fancy terms for how much "weight" and "direction" these components carry.

The platform doesn't stop there. You'll get a bunch of cool stats, like the explained variance ratio for PC1 and PC2 (basically, how much of the data's story each one captures), the cumulative variance (the total story preserved), and the mean vector (the data's center point). Plus, there's a covariance matrix to show you how the data's features play together—like who's best friends and who's not.

The best part? AlgoViz makes it super visual, so you don't have to wrestle with numbers alone. Your original data points show up as blue dots. If you go for a 1D projection, the transformed points pop up in red along the principal component axis. A yellow dot marks the data's average, grounding everything. The black line is PC1, the direction where your data stretches the most, and the purple line is PC2, catching the next biggest stretch. Dotted lines connect the original points to their new spots, so you can see exactly how PCA moves things around.

It's like watching your data get a makeover—simplified but still totally itself. AlgoViz lets you poke around and really get how PCA works, all while making it fun and easy to follow.

## 4. Interactive Features

AlgoViz enhances user engagement through:

- Real-time parameter tuning with instant visual feedback.
- Guided tutorials explaining theoretical concepts.
- The ability for users to plot their own data points or generate random sample datasets.
- Used Librariees like Pyplot and tools like graphviz.
- Visualization of the impact of parameter changes directly in the resulting graph for better conceptual clarity.

# 5. Future Scope

Future enhancements include:

- Incorporating additional algorithms such as Convolutional Neural Networks (CNN) and Random Forest to broaden the range of machine learning techniques available to users.
- Mobile app development to provide a seamless experience across devices, making the platform more accessible on the go.
- Improved GUI to enhance user interaction with more intuitive controls, better visualization features, and smoother navigation.
- Support for user-uploaded custom datasets to allow users to work with their own data, increasing the flexibility of the platform.
- Integration of model performance comparison tools to allow users to easily compare the results of different algorithms and parameter settings.

# 6. Conclusion

AlgoViz successfully transforms abstract ML concepts into tangible, interactive experiences. Through a modular architecture and dynamic visualizations, it serves as a versatile educational resource for learners and educators alike, democratizing machine learning education globally.

# References


- **Mantra Labs**, *10 Most Important Interaction Design Principles*, *Mantra Labs Blog*, 2024.
- **Infrasity Learning**, *ReactJS Best Practices for Developers*, *DEV Community*, 2024.
- **Machine Learning Mastery**, *5 Tools for Visualizing Machine Learning Models*, *Machine Learning Mastery Blog*, 2024.

- **PMC NCBI**, *Interactive Machine Learning by Visualization: A Small Data Solution*, *NCBI*, 2019.

- **Universitas Putra Indonesia YPTK Padang**, *The Use of Hyperparameter Tuning in Model Classification: A Scientific Work Area Identification*, *arXiv Preprint*, 2024.


# 7. Contributions

- Darsh Patel:Frontend and Backend Development.

  Worked on Project Architecture and Coordination, Decision Tree Visualization and SVM implementation, API development for algorithm execution

- Priyanshu Shah:Frontend and Backend Development.

  Worked on Neural network and PCA implementation and visualization,Algorithm optimization and performance tuning,Data preprocessing utilities

- Bhargav Shekokar:Backend Development

  KNN and DBSCAN implementation, Animation and transition effect, Cross browser compatibility

- Tanisha Saini: Frontend Development and Documentation

  Techinal Documentation and tutorials, Testing and Quality ensuration, Polynomial Regression implementaion

- Chanchal Yadav: Frontend Development

  Home page design,KMeans clustering implementation, UI/UX design and responsiveness

- Arun kumar: Backend Development

  SVM and PCA implementation, Performance Metrics calculations, Sample Dataset Curation

# 8. Acknowledgments


- We extend our heartfelt gratitude to our professor, Dr. Anand Mishra, for his invaluable guidance, mentorship, and support throughout the development of this project. His expertise and encouragement have been instrumental in shaping the vision and execution of AlgoViz.

- We also wish to thank our peers and mentors for their constructive feedback, insightful suggestions, and unwavering support, which have greatly contributed to the success of this project. Their collaboration and encouragement have been a source of inspiration.

- A special thanks to the open-source community for providing the libraries, tools, and resources that made this project possible. The contributions of countless developers and researchers have been pivotal in enabling us to build a robust and feature-rich platform.

- Lastly, we are deeply grateful for the cloud credits provided by Dr. Mishra, which allowed us to deploy and scale our application seamlessly. This support has been critical in ensuring the accessibility and performance of AlgoViz for all users.