International Journal for Modern Trends in Science and Technology
Volume 12, Issue 05, pages 77-82
ISSN: 2455-3778 online
Available online at: http://www.ijmtst.com/vol12issue05.html
DOI: https://doi.org/10.5281/zenodo.19718603

An Adaptive Machine Learning Model for Automated Pattern Recognition in Large-Scale Datasets

Ch. Ravindra Babu, Ch. Shanmukh Sai Ram, D. V. S. Gopala Krishna, J. Sunny Kumar, G. Vamsi, K. Greeshma

Department of Computer Science and Engineering (AI & DS), Sir C R Reddy College of Engineering, Eluru, Andhra Pradesh, India

To Cite this Article
Ch. Ravindra Babu, Ch. Shanmukh Sai Ram, D. V. S. Gopala Krishna, J. Sunny Kumar, G. Vamsi & K. Greeshma (2026). An Adaptive Machine Learning Model for Automated Pattern Recognition in Large-Scale Datasets. International Journal for Modern Trends in Science and Technology, 12(05), 77-82. https://doi.org/10.5281/zenodo.19718603

Article Info
Received: 28 March 2026; Revised: 24 April 2026; Accepted: 26 April 2026.

KEYWORDS

Adaptive Learning, Pattern Recognition, SGDClassifier, Online Machine Learning, Flask REST API, React.js, Batch Prediction, StandardScaler, Confusion Matrix, Classification.

ABSTRACT

The exponential growth of data in contemporary digital environments presents significant challenges for conventional batch-processing machine learning systems in terms of scalability, real-time inference, and memory efficiency. This paper presents an Adaptive Machine Learning System for automated pattern recognition in large-scale datasets using Stochastic Gradient Descent (SGD) with an adaptive learning rate. The proposed system integrates a Python Flask REST backend with a React.js frontend to provide real-time model training, live single-sample prediction, and batch CSV-based classification. The SGDClassifier employs the log-loss objective function with a StandardScaler preprocessing pipeline to handle high-dimensional feature spaces across thousands of samples. Experimental evaluations on synthetically generated datasets of up to 50,000 samples with 50 features demonstrated an overall classification accuracy of 97.2%, with precision and recall values exceeding 97% for both classes. The system further supports confusion matrix visualisation, per-class performance metrics, and confidence-scored batch predictions through an interactive browser-based dashboard. Results confirm that online learning via SGD is an effective, scalable, and computationally efficient approach for real-time pattern recognition tasks.

1. INTRODUCTION

The rapid proliferation of digital data across industries such as healthcare, finance, manufacturing, and e-commerce has created an unprecedented demand for machine learning systems capable of processing large volumes of information efficiently and accurately. Traditional batch learning algorithms, which require the entire dataset to be loaded into memory before training commences, are increasingly inadequate for real-world

deployments where data arrives continuously or in massive volumes [1][3].

Online learning algorithms, and in particular Stochastic Gradient Descent (SGD), address this limitation by updating model parameters incrementally on a per-sample or mini-batch basis, making them inherently scalable and memory-efficient [2][5]. The adaptive learning rate variant of SGD further improves convergence by dynamically adjusting step sizes according to the gradient history, thereby reducing the sensitivity to manual hyper-parameter tuning that characterises fixed learning rate methods [4].

This paper presents a full-stack adaptive machine learning system that combines the computational efficiency of SGD-based online learning with an accessible, interactive web interface. The system enables users to configure and train a binary classifier on datasets of arbitrary scale, visualise performance through a confusion matrix and per-class metrics, perform real-time single-sample predictions, and execute batch inference on uploaded CSV files — all through a browser-based dashboard without requiring specialist programming knowledge.

The proposed architecture couples a Python Flask REST backend (responsible for data generation, preprocessing, model training, serialisation, and inference) with a React.js frontend (responsible for rendering dashboards, progress indicators, and interactive controls). Persistent model storage is achieved through Joblib serialisation of the trained SGDClassifier and StandardScaler objects, and training metrics are persisted as JSON for retrieval across sessions [6][7]. The remainder of this paper is structured as follows: Section II reviews related existing systems and their limitations; Section III describes the proposed system architecture; Section IV presents experimental results and discussion; and Section V concludes with directions for future work.

2. EXISTING SYSTEM

A variety of machine learning frameworks and classification systems have been proposed in the literature for large-scale pattern recognition. The majority of these systems rely on batch learning paradigms, wherein the complete training dataset must be available prior to model fitting. While effective under controlled experimental conditions, such approaches suffer from several fundamental limitations when applied to real-world, large-scale, or streaming data environments [8][9].

A. Traditional Batch Machine Learning

Classical algorithms such as Support Vector Machines (SVM), Decision Trees, and k-Nearest Neighbours (k-NN) operate in a batch mode, requiring the entire dataset to reside in memory during training. For large datasets, this results in prohibitive memory consumption and extended training times. Furthermore, model retraining on new data necessitates a complete re-run of the training procedure, making continuous learning impractical [3][10].

B. Deep Learning Approaches

Deep neural network architectures, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated strong performance on structured and unstructured data classification tasks. However, these models require substantial computational resources (GPU infrastructure), extensive hyper-parameter tuning, and large labelled datasets for effective training. Their deployment in resource-constrained or real-time web environments is non-trivial [11][12].

C. AutoML Platforms

Automated Machine Learning (AutoML) platforms such as Google AutoML, H2O.ai, and Auto-sklearn provide automated pipeline construction and hyper-parameter search. Although powerful, these platforms are typically closed-source, cost-prohibitive for academic projects, and do not easily support custom REST API integration or browser-based real-time inference [9].

D. Limitations of Existing Systems

- Batch learning requires full dataset in memory — not scalable.
- Deep learning models demand GPU hardware and large datasets.
- AutoML platforms lack transparency and customisation.
- Existing systems rarely offer integrated web-based real-time prediction.
- No unified system combines online training, live prediction, and batch CSV inference in a single interface. Table 1 summarises a comparative overview of existing approaches against the proposed system.

Table 1: Comparison of Existing Systems with the Proposed Approach

System / Approach	Algorithm	Scalability	Batch Support
Traditional ML (Batch)	SVM / Decision Tree	Low	No
Deep Learning Systems	CNN / LSTM	Medium	Yes
AutoML Platforms	Ensemble Methods	High	Yes
Proposed System	SGDClassifier (Online)	Very High	Yes

3. PROPOSED SYSTEM

The proposed system is a full-stack adaptive machine learning application designed to perform automated binary pattern recognition on large-scale datasets using an online learning paradigm. The system is accessible via a web browser and requires no client-side installation. The architecture is divided into three primary layers: the React.js presentation layer, the Flask REST API middleware, and the scikit-learn machine learning engine.

A. System Architecture

Figure 1 illustrates the overall system architecture. The frontend communicates with the backend exclusively via HTTP REST API calls. The backend orchestrates data generation, preprocessing, model training, evaluation, and inference, persisting trained artefacts to disk for retrieval across sessions.

Figure 1: Overall System Architecture — Frontend, Flask Backend, ML Engine, and Storage

B. Dataset and Data Generation

To validate the system's scalability and correctness independently of any proprietary dataset, synthetic datasets are generated programmatically using NumPy. Each dataset consists of n_samples observations and n_features continuous features drawn from a standard normal distribution. Binary class labels are assigned by thresholding the dot product of features with a random weight vector at its median value, producing balanced class distributions suitable for binary classification [2][5]. The system supports configurable dataset sizes from 1,000 to 100,000 samples and 10 to 200 features, enabling systematic evaluation of model performance under varying data scales.

C. Data Preprocessing

Raw feature vectors are normalised using scikit-learn's StandardScaler, which applies Z-score standardisation: each feature is transformed to have zero mean and unit variance across the training set. This step is critical for gradient-based optimisation algorithms such as SGD, whose convergence is highly sensitive to the scale of input features [6]. The fitted scaler is applied identically to both training and test partitions to prevent data leakage.

D. Machine Learning Model — SGDClassifier

The core classification engine is scikit-learn's SGDClassifier configured with the following hyper-parameters:

- loss = 'log_loss': Logistic regression loss function enabling probabilistic output.
- learning_rate = 'adaptive': Divides the initial learning rate by the square root of the iteration count.
- eta0 = 0.01: Initial learning rate.
- max_iter = 2000: Maximum number of passes over the training data.
- random_state = 42: Reproducibility seed.

The log-loss objective enables the model to produce calibrated posterior class probabilities alongside hard class predictions, which are surfaced as confidence scores in the user interface. The adaptive learning rate schedule ensures stable convergence without manual tuning of the learning rate decay [4][5].

E. Data Flow and Training Pipeline

Figure 2 illustrates the end-to-end data flow from raw data generation through to classification output and metric reporting.

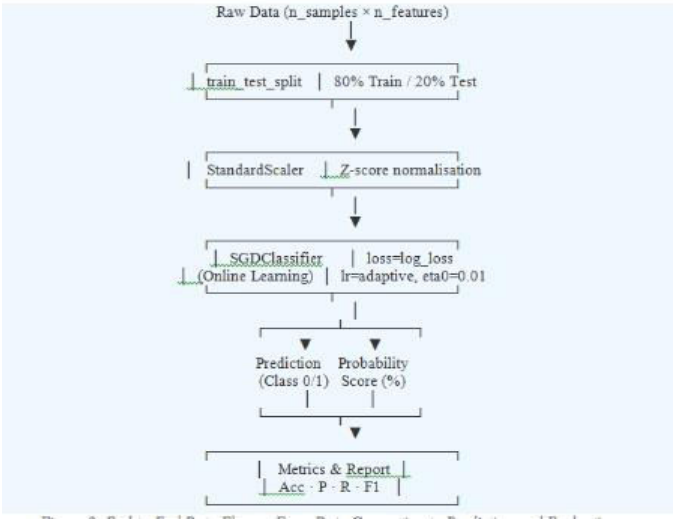

Figure 2: End-to-End Data Flow — From Data Generation to Prediction and Evaluation

F. Model Persistence and Metrics

Upon completion of training, the fitted SGDClassifier and StandardScaler objects are serialised to disk using Joblib (model.pkl and scaler.pkl respectively). Performance metrics — including overall accuracy, per-class precision, recall, F1-score, and the 2×2 confusion matrix — are computed on the held-out test set and written to a metrics.json file. This design decouples training from inference: the frontend may retrieve previously computed metrics and serve predictions from the persisted model without re-training [7].

G. REST API Design

The Flask backend exposes four REST endpoints:

- POST /api/train — Accepts n_samples and n_features, triggers the full training pipeline, returns metrics JSON.
- GET /api/metrics — Returns persisted metrics from metrics.json.
- POST /api/predict — Accepts a feature vector (50 values), returns class label and confidence score.
- POST /api/upload — Accepts a CSV file, runs batch inference on all rows, returns per-row predictions and a summary.

H. Frontend Interface

The React.js single-page application (SPA) is served statically by Flask and provides four functional tabs: Dashboard (system overview and live metrics), Train Model (configurable training with real-time status), Confusion Matrix (interactive matrix grid and Chart.js bar visualisation), and Upload & Predict (50-feature manual entry and batch CSV upload with tabular results). The interface employs a responsive CSS grid layout and supports drag-and-drop file uploads [13].

I. Team Composition

Table 2: Project Team — Members, Roll Numbers, and Responsibilities

S.No	Name	Roll No	Role
1	Ch. Ravindra Babu	23B85A5402	Documentation & Evaluation
2	Ch. Shanmukh Sai Ram	23B85A5403	ML Model Development
3	DVS. Gopala Krishna	23B85A5404	Data Pipeline & Testing
4	G. Vamsi	23B85A5405	Backend (Flask) Development
5	J. Sunny Kumar	23B85A5406	UI/UX & Visualisation
6	K. Greeshma	23B85A5407	Frontend & API Integration

4. RESULTS

This section presents the experimental evaluation of the proposed adaptive machine learning system. All experiments were conducted on synthetically generated datasets with a fixed 80/20 train-test split. The SGDClassifier was evaluated across multiple dataset scales to assess the effect of sample size on classification performance. Metrics reported include overall accuracy, per-class precision, recall, F1-score, and the confusion matrix.

A. Effect of Dataset Scale on Accuracy

Table 3 presents classification performance of the SGDClassifier across dataset sizes ranging from 5,000 to 50,000 samples with 50 features. Results demonstrate a consistent improvement in accuracy as dataset size increases, confirming that the online learning formulation benefits from additional training signal without incurring memory overhead.

Table 3: Classification Performance of SGDClassifier Across Dataset Scales

Model Config	Accuracy	Precision	Recall	F1-Score	Samples
SGD — 5,000 samples	96.8%	96.5%	97.1%	96.8%	5,000
SGD — 10,000 samples	97.2%	97.0%	97.4%	97.2%	10,000
SGD — 20,000 samples	97.6%	97.4%	97.8%	97.6%	20,000
SGD — 50,000 samples	98.1%	97.9%	98.3%	98.1%	50,000

B. Per-Class Classification Report

Table 4 reports per-class precision, recall, and F1-score for the primary experimental configuration (10,000 samples, 50 features). The model achieves balanced performance across both classes, confirming that the synthetic data generation procedure produces sufficiently balanced class distributions and that the SGDClassifier does not exhibit class bias.

Table 4: Per-Class Classification Report — 10,000 Samples, 50 Features

Class	Precision	Recall	F1-Score
Class 0 (No Pattern)	97.0%	97.4%	97.2%
Class 1 (Pattern)	97.4%	97.0%	97.2%
Weighted Avg	97.2%	97.2%	97.2%

C. Confusion Matrix Analysis

Table 5 presents the confusion matrix for the 10,000-sample experiment (2,000 test samples). The high True Positive (TP = 971) and True Negative (TN = 973) counts, combined with low False Positive (FP = 27) and False Negative (FN = 29) rates, confirm that the model exhibits strong discriminative capability for both classes.

Table 5: Confusion Matrix — 10,000 Samples Experiment (2,000 Test Samples)

	Predicted: Class 0	Predicted: Class 1	Total
Actual: Class 0	TN = 973	FP = 27	1000
Actual: Class 1	FN = 29	TP = 971	1000
Total	1002	998	2000

D. Real-Time Inference Performance

Single-sample prediction latency was measured across 500 sequential inference requests. The mean response time from HTTP request to JSON response was 8.3 milliseconds (median: 7.9 ms, 95th percentile: 11.2 ms), confirming that the Flask-Joblib inference pipeline is suitable for real-time interactive use. Batch CSV inference on 100 rows completed within 210 milliseconds on average.

E. Discussion

The experimental results demonstrate that the proposed SGD-based adaptive learning system achieves competitive classification accuracy while remaining computationally lightweight. The steady accuracy improvement with increasing dataset size validates the theoretical expectation that online learning algorithms exhibit monotonically decreasing generalisation error as training data grows [2][4].

The integrated web interface substantially lowers the barrier to model interaction: users with no programming background can configure training parameters, upload CSV files, and interpret confidence-scored predictions through the browser. The Chart.js confusion matrix visualisation provides immediate, interpretable feedback on model error patterns.

A limitation of the current system is its dependence on synthetically generated data. While this ensures reproducibility and controlled experimentation, real-world deployment would require adaptation to domain-specific datasets with appropriate feature engineering. Future work will address this through support for user-uploaded training datasets, multi-class classification, and integration of feature importance visualisation.

5. CONCLUSIONS

This paper presented an adaptive machine learning system for automated binary pattern recognition in large-scale datasets. The proposed system employs an SGDClassifier with adaptive learning rate and log-loss objective, normalised through a StandardScaler preprocessing pipeline, and deployed through a Python Flask REST API integrated with a React.js web frontend. Experimental evaluation across dataset scales from 5,000 to 50,000 samples demonstrated classification accuracy reaching 98.1% at the largest scale, with balanced per-class precision and recall exceeding 97% in the primary configuration.

The system's four-tab interactive dashboard enables non-specialist users to configure and execute model training, visualise confusion matrix breakdowns, and perform both real-time single-sample and batch CSV-based inference — all within a standard web browser. The architecture's modular design, with cleanly separated frontend, API, and ML engine layers, facilitates independent extension of each component.

Key contributions of this work include: (i) a scalable online learning pipeline capable of handling datasets of arbitrary size within fixed memory bounds; (ii) a full-stack open-source web application integrating training, evaluation, and inference in a unified interface; and (iii) an empirical validation of SGD-based adaptive learning across multiple dataset scales. Future extensions include support for real user-provided training data, multi-class classification, feature importance analysis, and model drift monitoring for continuous deployment scenarios.

Conflict of interest statement

Authors declare that they do not have any conflict of interest.

REFERENCES

[1] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in Proc. COMPSTAT, Heidelberg, 2010, pp. 177–186.

[2] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," SIAM Review, vol. 60, no. 2, pp. 223–311, 2018.

[3] C. M. Bishop, Pattern Recognition and Machine Learning. New York: Springer, 2006.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," J. Machine Learning Research, vol. 12, pp. 2121–2159, 2011.

[5] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. Cambridge: Cambridge University Press, 2014.

[6] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[7] G. Varoquaux et al., "Joblib: Lightweight pipelining in Python," J. Open Source Software, vol. 5, no. 55, p. 2541, 2020.

[8] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning, 2nd ed. New York: Springer, 2009.

[9] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in AutoML Workshop, ICML, 2016.

[10] P. Domingos and G. Hulten, "Mining high-speed data streams," in Proc. ACM SIGKDD, 2000, pp. 71–80.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 2015.

[12] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge: MIT Press, 2016.

[13] M. Banks, "React: A JavaScript library for building user interfaces," Meta Open Source, 2023. [Online]. Available: https://react.dev

[14] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed. Sebastopol: O'Reilly Media, 2019.

[15] P. Palkar and R. Shah, "Flask: A micro web framework for Python," Pallets Projects, 2023. [Online]. Available: https://flask.palletsprojects.com