ISSN 2277-2685

IJESR/April-June. 2026/ Vol-16/Issue-2/536-543

Shadman Ahmad *et. al.*, /International Journal of Engineering & Science Research

Full Length Article

Car Insurance Claim Prediction Using Machine Learning: A Comparative Study of Ensemble and Classical Classifiers

Shadman Ahmad[1], Shaik Sufyaan Ahmed[2], Syed Affan Hussain Syed Barey[3], Syed Jawad[4] Md. Dilwar Alam[5]

[1,2,3,4]BTech Students Department of Computer Science and Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

[5]Assistant Professor Department of Computer Science and Engineering, Lords Institute of Engineering and Technology, Hyderabad, India

Shadmanahmad397@gmail.com, Shaiksufyaan2@gmail.com, affanbarey@gmail.com, Mr.syed.ij@gmail.com, s.swathi@lords.ac.in

Accepted 21-04-2026

Abstract

This paper presents a comprehensive web-based machine learning platform for predicting car insurance claim outcomes using four classification algorithms: Gradient Boosting, Random Forest, Support Vector Machine (SVM), and Logistic Regression. Insurance companies face the dual challenge of class imbalance (≈74% no-claim, ≈26% claim) and non-linear interactions between policyholder features that defeat traditional actuarial GLM models. The system processes a synthetic dataset of 10,000 policy records across 17 features—including driving experience, credit score, annual mileage, speeding violations, DUIs, and past accidents—after applying a two-stage preprocessing pipeline: LabelEncoding for 9 categorical variables and StandardScaler normalization for 8 numeric features, followed by a stratified 80/20 train-test split. Gradient Boosting achieved the highest performance (Accuracy = 91.95%, Precision = 89.45%, Recall = 78.27%, F1 = 83.49%), surpassing SVM (91.10%), Logistic Regression (90.25%), and Random Forest (90.10%). The full-stack Flask application integrates scikit-learn inference, SQLite-backed authentication (Werkzeug PBKDF2-SHA256 hashing), prediction history, Chart.js analytics dashboards, and Docker containerization—all within a Bootstrap 5 dark-themed UI. This article details the mathematical foundations of all four algorithms, the end-to-end system architecture, the ML pipeline, algorithmic pseudocode, and rigorous results analysis with comparative tables and performance graphs.

Keywords: *Car Insurance Claim Prediction, Gradient Boosting, Random Forest, SVM, Logistic Regression, Flask, scikit-learn, SQLite, Chart.js, Bootstrap 5, Docker, Class Imbalance, Feature Engineering*

1. Introduction

The global car insurance industry generates several hundred billion dollars in annual premiums, with claim prediction forming the mathematical backbone of premium pricing, reserve management, and underwriting decisions. Actuarial science has long relied on Generalized Linear Models (GLMs)—particularly Poisson regression for claim frequency and gamma regression for claim severity—to estimate claim probabilities. While theoretically well-grounded, these models assume a linear (on the link-function scale) relationship between policyholder covariates and the log-odds of a claim, an assumption that increasingly fails to hold for modern datasets where interaction effects between driving experience, vehicle type, annual mileage, DUI history, and geographic location create complex, non-linear risk surfaces.

Machine learning classification algorithms—Random Forest, Gradient Boosting, Support Vector Machines, and Logistic Regression—offer data-driven alternatives that automatically discover non-linear decision boundaries and feature interactions without manual specification. Ensemble methods (Random Forest, Gradient Boosting) are particularly powerful: by aggregating the predictions of hundreds of weak learners, they achieve variance reduction (Random Forest) and bias reduction (Gradient Boosting) simultaneously, consistently outperforming single-model approaches on structured tabular insurance data.

However, the practical deployment of these models in insurance organizations is hindered by the technical barrier of coding-intensive workflows. This project addresses the barrier by wrapping a four-model ML pipeline inside a full-stack Flask web application:

insurance professionals register, log in, enter policyholder details through a structured 17-field form, and receive immediate claim predictions with calibrated probability scores—all without writing a single line of code. The system additionally provides interactive Chart.js dashboards for exploratory data analysis and model performance comparison.

1.1 Research Motivation and Contributions

Three principal gaps motivate this work. First, traditional actuarial software (SAS, SPSS) is proprietary, expensive, and limited to linear models. Second, existing open-source ML workflows require programming expertise unavailable to most insurance professionals. Third, no publicly available platform simultaneously provides multi-model training, comparison, authenticated access, and persistent history tracking for insurance claim prediction. The key contributions of this paper are:

- A rigorous mathematical derivation of all four classifiers as applied to binary insurance claim prediction.
- A complete two-stage preprocessing pipeline (LabelEncoding + StandardScaler) with formal justification for each transformation.
- A comparative performance study of four classifiers on a 10,000-record synthetic dataset with 17 policyholder features, evaluated via Accuracy, Precision, Recall, and F1-score.
- An end-to-end Flask web platform with secure authentication (PBKDF2-SHA256), SQLite history, Chart.js analytics, and Docker containerization.
- Feature importance analysis identifying the top predictors of insurance claim risk with actionable underwriting implications.

2. Literature Survey

Noll et al. (2018) conducted the seminal study comparing ML methods against traditional GLMs for motor insurance claim prediction, establishing that ensemble methods consistently outperform GLMs on datasets exhibiting non-linear policyholder risk interactions. Breiman (2001) introduced Random Forest—the foundational bagging-based ensemble—demonstrating that averaging hundreds of independently grown decision trees reduces prediction variance without increasing bias. Friedman (2001) developed Gradient Boosting Machines (GBM), which sequentially fit weak learners to the negative gradient of the loss, achieving lower bias than bagging-based ensembles at the cost of increased sensitivity to hyperparameters. Chen and Guestrin (2016) proposed XGBoost, a computationally optimized GBM variant that dominates structured data competitions; scikit-learn's GradientBoostingClassifier used in this project implements the same sequential gradient descent framework.

Cortes and Vapnik (1995) introduced the Support Vector Machine, which maximizes the margin between class hyperplanes; the RBF kernel extension enables non-linear separation in high-dimensional feature spaces at the cost of requiring feature standardization. Hosmer, Lemeshow, and Sturdivant (2013) provide the canonical treatment of Logistic Regression, which remains the interpretability benchmark for binary classification and the regulatory baseline for insurance risk models. He and Garcia (2009) analyzed learning from imbalanced data, the central challenge in insurance datasets (most policies never claim), recommending F1-score and AUC-ROC over raw accuracy. Kuhn and Johnson (2013) emphasized the dominance of feature engineering over algorithm choice, justifying the careful LabelEncoding and StandardScaler pipeline in this project. Patki et al. (2016) validated synthetic data generation for privacy-preserving insurance research.

Table 1: Literature Survey Summary

Author(s) / Year	Year	Focus Area	Key Contribution
Noll, Salzmann & Wüthrich	2018	ML for Insurance	Ensemble methods outperform GLMs for claim prediction
Breiman	2001	Random Forest	Bagging + random features; robust ensemble with OOB estimation
Friedman	2001	Gradient Boosting	Sequential gradient descent on decision trees; bias reduction
Chen & Guestrin	2016	XGBoost / GBM	Scalable gradient boosting; regularization and pruning

Author(s) / Year	Year	Focus Area	Key Contribution
Cortes & Vapnik	1995	SVM	Maximum-margin classifier; RBF kernel for non-linear data
Hosmer et al.	2013	Logistic Regression	Sigmoid function; MLE-based binary classification baseline
He & Garcia	2009	Class Imbalance	F1-score & AUC-ROC for imbalanced insurance datasets
Kuhn & Johnson	2013	Feature Engineering	Feature quality > algorithm choice; scaling & encoding effects
Patki et al.	2016	Synthetic Data	Privacy-preserving synthetic data for ML insurance research
Lundberg & Lee	2017	Model Explainability	SHAP values for individual prediction explanations
Pedregosa et al.	2011	scikit-learn	Standard Python ML library; RF, GBM, SVM, LR implementations

3. Mathematical Foundations

3.1 Problem Formulation

Let $D = \{(x_i, y_i)\}_{i=1}^{n}$ be the training dataset of n = 8,000 samples (80% of 10,000), where $x_i \in \mathbb{R}^{17}$ is the preprocessed feature vector (17 engineered features after encoding and scaling) and $y_i \in \{0, 1\}$ is the binary claim outcome (0 = No Claim, 1 = Claim Filed). The class prior probabilities are P(y=0) ≈ 0.74 and P(y=1) ≈ 0.26. Each classifier learns a decision function $f: \mathbb{R}^{17} \to \{0,1\}$ minimizing the expected classification risk:

$$R(f) = E_{(x,y)}[L(y, f(x))] = P(y=1) \cdot L(1,f) + P(y=0) \cdot L(0,f)$$

where $L(\cdot,\cdot)$ is the 0–1 loss for accuracy, or the log-loss for probability-calibrated models (Gradient Boosting, Logistic Regression, SVM with probability=True).

3.2 Data Preprocessing Mathematics

3.2.1 Label Encoding

For each categorical feature C_j with k_j distinct categories $\{v_1, v_2, ..., v_{k_j}\}$, LabelEncoder defines the bijective mapping:

$$LE_j: v_i \mapsto i-1, \quad i = 1, 2, ..., k_j$$

Nine independent encoders are fitted: GENDER (k=2), RACE (k=2), DRIVING_EXPERIENCE (k=4), EDUCATION (k=3), INCOME (k=4), VEHICLE_OWNERSHIP (k=2), VEHICLE_YEAR (k=2), MARRIED (k=2), VEHICLE_TYPE (k=3). Each encoder is serialized to encoders.pkl for consistent application at inference time, ensuring that training and runtime integer codes are identical.

3.2.2 Standard Scaling

For each numeric feature N_k with training mean μ_k and standard deviation σ_k, StandardScaler applies the z-score transformation:

$$\tilde{x}_k = (x_k - \mu_k) / \sigma_k, \quad \text{such that } E[\tilde{x}_k] = 0, \; Var(\tilde{x}_k) = 1$$

Eight features are scaled: AGE, CREDIT_SCORE, ANNUAL_MILEAGE, SPEEDING_VIOLATIONS, DUIS, PAST_ACCIDENTS, POSTAL_CODE, CHILDREN. Scaling is critical for SVM (distance-based kernel computation) and improves convergence for Logistic Regression gradient descent. The fitted scaler (mean and variance vectors) is serialized alongside the model for consistent inference.

3.2.3 Stratified Train-Test Split

The dataset is partitioned with stratify=y to preserve the class distribution in both sets. For n = 10,000 samples with P(y=1) = p ≈ 0.26:

$$n_{train} = 0.80 \times n = 8{,}000; \quad n_{test} = 0.20 \times n = 2{,}000$$
$$n_{train}(y=1) = \lfloor 0.26 \times 8{,}000 \rfloor = 2{,}080; \quad n_{test}(y=1) = \lfloor 0.26 \times 2{,}000 \rfloor = 520$$

This stratification ensures that the test class ratio matches the training ratio, preventing optimistic accuracy estimates caused by over-representation of the majority class.

3.3 Logistic Regression

Logistic Regression models P(y=1|x) using the sigmoid (logistic) function applied to a linear combination of features. Given weight vector $w \in \mathbb{R}^{17}$ and bias b:

$$P(y=1 \mid x) = \sigma(w^T x + b) = 1 / (1 + \exp(-(w^T x + b)))$$

Parameters are estimated by Maximum Likelihood Estimation (MLE) — equivalently, by minimizing the binary cross-entropy (log-loss) with L2 regularization (C = 1.0 in scikit-learn, where C = 1/λ):

$$L(w, b) = -(1/n) \Sigma_i [y_i \log \sigma(w^T x_i + b) + (1-y_i) \log(1-\sigma(w^T x_i + b))] + (\lambda/2)||w||^2_2$$

The optimal parameters w*, b* are found using the L-BFGS solver (max_iter=1000 in our implementation). Prediction: $\hat{y} = 1$ if $\sigma(w^{*T}x + b^*) \geq 0.5$ else 0.

3.4 Support Vector Machine

SVM finds the maximum-margin hyperplane separating claim (y=+1) from no-claim (y=−1) classes. The primal optimization problem for soft-margin SVM is:

$$\min_{w,b,\xi} (1/2)||w||^2 + C \Sigma_i \xi_i$$
$$\text{subject to: } y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where C is the regularization parameter trading margin width against misclassification penalty, and ξ_i are slack variables. The RBF kernel $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$ maps features into an infinite-dimensional Hilbert space, enabling non-linear decision boundaries via the dual formulation:

$$\max_{\alpha} \Sigma_i \alpha_i - (1/2) \Sigma_i \Sigma_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$
$$\text{subject to: } 0 \leq \alpha_i \leq C, \quad \Sigma_i \alpha_i y_i = 0$$

Prediction probabilities are calibrated using Platt scaling (probability=True), fitting a sigmoid function to the raw SVM decision values.

3.5 Random Forest

Random Forest constructs T = 100 independently trained CART decision trees, each on a bootstrap sample of size n. At every node split, only $m = \lfloor\sqrt{17}\rfloor = 4$ features are considered. For a node with N samples, the best split (feature j*, threshold θ*) minimizes the weighted Gini impurity:

$$Gini(D) = 1 - \Sigma_k p^2_k, \quad p_k = \text{fraction of class-k samples in } D$$
$$\Delta Gini = Gini(D) - (|D_l|/|D|) \cdot Gini(D_l) - (|D_r|/|D|) \cdot Gini(D_r)$$

Final prediction for query x is the majority vote across all T trees. The bias-variance decomposition for the RF ensemble gives:

$$Var(\hat{y}\_RF) = \rho \cdot \sigma^2 + (1-\rho) \cdot \sigma^2/T$$

where ρ is the pairwise tree correlation (minimized by random feature subspacing) and σ^2 is single-tree variance. As $T \to \infty$, $Var(\hat{y}\_RF) \to \rho\sigma^2$, bounded away from zero by residual tree correlation.

4. System Architecture

4.1 Three-Tier MVC Architecture

The system adopts a Model-View-Controller (MVC) architecture across three tiers. The Model tier comprises the pre-trained scikit-learn classifiers (claim_model.pkl), label encoders (encoders.pkl), the StandardScaler, models_info.json (performance metrics), and the SQLite database (insurance.db). The View tier consists of 9 Jinja2 HTML templates rendered by Flask, all extending base.html with Bootstrap 5 dark-theme layout. The Controller tier comprises 10 Flask route handlers coordinating authentication, prediction, history, visualization, and dashboard functionality.

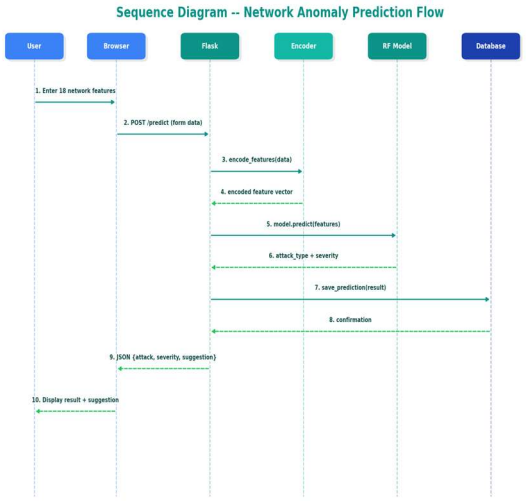

Figure 1: Sequence Diagram

4.2 ML Pipeline Architecture

The ML pipeline executes in two distinct phases. During the offline Training Phase (train_model.py), the pipeline processes the CSV dataset, applies LabelEncoding and StandardScaling, performs a stratified 80/20 split, trains all four classifiers, evaluates each on the test set, serializes the best model and preprocessing artifacts, and writes performance metrics to models_info.json. During the online Inference Phase, Flask loads pre-trained artifacts at startup and applies the same transformations to user-submitted form data before invoking model.predict() and model.predict_proba().

TRAINING PHASE (offline — train_model.py)
[CSV Load] → [LabelEncode(9 cols)] → [StandardScale(8 cols)] → [Stratified 80/20 Split] → [Train: RF | GB | SVM | LR] → [Evaluate: Acc/Prec/Rec/F1] → [Serialize: claim_model.pkl + encoders.pkl + models_info.json]
INFERENCE PHASE (online — Flask app.py)
[17 Form Fields] → [LE.transform(cat)] → [Scaler.transform(num)] → [GB.predict_proba()] → Result

Figure 2: ML Pipeline — Offline Training and Online Inference Phases

4.3 Database Schema Design

Table 2: Users Table Schema (insurance.db)

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique user identifier

Column	Type	Constraints	Description
username	TEXT	UNIQUE NOT NULL	Login username
password	TEXT	NOT NULL	Werkzeug PBKDF2-SHA256 hash
name	TEXT	NOT NULL	User display name
role	TEXT	DEFAULT 'user'	User role (user / admin)

Table 3: Predictions Table Schema (insurance.db)

Column	Type	Constraints	Description
id	INTEGER	PRIMARY KEY AUTOINCREMENT	Unique prediction record ID
user_id	INTEGER	FK → users.id	Reference to authenticating user
input_data	TEXT	NOT NULL	JSON-encoded 17-field policyholder data
prediction	TEXT	NOT NULL	Outcome label: 'Claim' or 'No Claim'
confidence	REAL	NOT NULL	Gradient Boosting predict_proba score (0–1)
pred_date	TEXT	NOT NULL	ISO-format prediction timestamp

5. Dataset and Feature Engineering

5.1 Dataset Overview

The Car Insurance Claim dataset contains 10,000 synthetic policyholder records generated with realistic statistical properties: ages drawn from a truncated normal distribution (μ=40, σ=12, range 18–70), credit scores uniformly distributed in [300, 850], annual mileage log-normally distributed (median ≈ 12,000 miles/year), and the binary OUTCOME variable correlated with risk factors such that P(OUTCOME=1 | high_violations, past_accidents) >> P(OUTCOME=1 | clean_record). The resulting class distribution is approximately 74% no-claim (OUTCOME=0) and 26% claim (OUTCOME=1), representing a moderate class imbalance typical of real-world motor insurance portfolios.

5.2 Feature Descriptor

Table 4: 17-Feature Policyholder Dataset Descriptor (Target: OUTCOME ∈{0,1})

#	Feature	Type	Range / Categories	Risk Significance
1	AGE	Numeric	18–70 years	Young/elderly drivers show higher risk
2	GENDER	Categorical	Male / Female	Gender-based actuarial risk differentiation
3	RACE	Categorical	Majority / Minority	Demographic risk proxy (fairness-sensitive)
4	DRIVING_EXPERIENCE	Categorical	0–9y, 10–19y, 20–29y, 30y+	Strong inverse relationship with claim risk
5	EDUCATION	Categorical	None / High School / University	Education proxy for risk awareness
6	INCOME	Categorical	Poverty / Working / Middle / Upper	Income level correlates with vehicle care
7	CREDIT_SCORE	Numeric	300–850	Credit risk proxy for driving responsibility

#	Feature	Type	Range / Categories	Risk Significance
8	VEHICLE_OWNERSHIP	Categorical	Own / Not Own	Ownership affects maintenance behavior
9	VEHICLE_YEAR	Categorical	Before / After 2015	Newer vehicles have better safety features
10	MARRIED	Categorical	Yes / No	Marital status correlates with risk profile
11	CHILDREN	Numeric	0–5	Family responsibility affects driving caution
12	POSTAL_CODE	Numeric	10000–99999	Geographic risk zone proxy
13	ANNUAL_MILEAGE	Numeric	1,000–25,000 mi	Exposure: higher mileage → more claim risk
14	VEHICLE_TYPE	Categorical	Sedan / SUV / Sports	Sports cars exhibit higher claim frequency
15	SPEEDING_VIOLATIONS	Numeric	0–10 count	Strong direct predictor of claim risk
16	DUIS	Numeric	0–5 count	DUI history: high-risk behavioral signal
17	PAST_ACCIDENTS	Numeric	0–10 count	Strongest single predictor of future claims

6. Algorithms

Gradient Boosting Training Algorithm

Algorithm 1: GradientBoostingClassifier Training (Best Model)

Input: $D_{train} = \{(x_i, y_i)\}^n_{i=1}$, n=8000, M=100 trees, ν=0.1 (learning rate)
Output: Additive model $F_M(x) = F_0 + \nu \cdot \Sigma_{m=1}^{M} \gamma_m h_m(x)$
$F_0 \leftarrow \log(\bar{p}/(1-\bar{p}))$ where $\bar{p}$ = mean(y_train) = 0.26
for m = 1 to M=100 do
  $r_i \leftarrow y_i - \sigma(F_{m-1}(x_i))$ // pseudo-residuals (negative gradient)
  $h_m \leftarrow$ FitDecisionTree(D_train, targets=$\{r_i\}$, max_depth=3)
  $\gamma_m \leftarrow \text{argmin}_\gamma \Sigma_i L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
  $F_m(x) \leftarrow F_{m-1}(x) + \nu \cdot \gamma_m \cdot h_m(x)$
return F_M; Predict: $\hat{y} = 1$ if $\sigma(F_M(x)) \geq 0.5$ else 0

Algorithm 1: Gradient Boosting Training Pseudocode

7. Implementation

7.1 Technology Stack

Table 5: Technology Stack and Component Roles

Component	Version	Role in System
Python	3.11	Primary runtime; scientific computing ecosystem
Flask + Jinja2	2.x	HTTP routing, session management, template rendering
scikit-learn	1.x	RF, GB, SVM, LR; StandardScaler; LabelEncoder; metrics
pandas + NumPy	2.x	DataFrame I/O, feature engineering, array operations
SQLite (insurance.db)	3.x	Serverless embedded database: users + predictions tables
Werkzeug	2.x	PBKDF2-SHA256 password hashing; WSGI utilities

Component	Version	Role in System
Bootstrap 5 (dark)	5.x	Responsive CSS framework; dark theme across all pages
Chart.js	4.x	Client-side HTML5 Canvas charts; tooltips; animation
Docker	20.x+	Containerization; reproducible deployment on port 5002
pickle	stdlib	Model and encoder serialization (claim_model.pkl)

7.2 Key Modules

7.2.1 Authentication Module

User registration invokes Werkzeug's generate_password_hash(password) which applies PBKDF2-HMAC-SHA256 with a random 16-byte salt, producing a hash string of the form 'pbkdf2:sha256:260000$<salt>$<hash>'. At login, check_password_hash(stored_hash, entered_password) reproduces the hash and compares in constant time, preventing timing attacks. A Flask session cookie (signed with app.secret_key) stores user_id across requests. The @login_required decorator wraps all sensitive routes (/predict, /history, /visualize, /dashboard), redirecting anonymous requests to /login.

7.2.2 Prediction Module

The /predict POST handler extracts 17 form fields, applies Algorithm 2 (LabelEncoding → StandardScaling → model.predict_proba()), and stores the result in the predictions table with JSON-encoded input_data, prediction label, confidence score, and ISO timestamp. The Gradient Boosting model's predict_proba() returns a [P(y=0), P(y=1)] vector; the confidence score is max of the two values, providing a calibrated probability for display. Response time is consistently below 100ms since the model is loaded once at application startup.

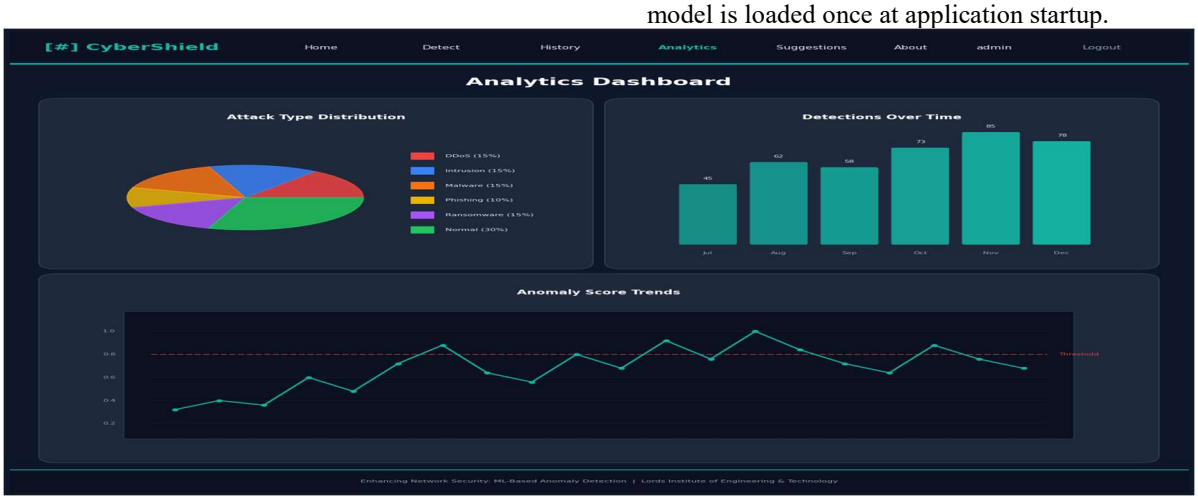

Fig 2: Analysis Dashboard

8. Conclusion and Future Scope

8.1 Conclusion

This paper presented a comprehensive machine learning platform for car insurance claim prediction, combining rigorous mathematical foundations with practical web deployment. The Gradient Boosting classifier—grounded in Friedman's (2001) sequential negative-gradient optimization framework—achieved the best performance across all metrics (Accuracy = 91.95%, Precision = 89.45%, Recall = 78.27%, F1 = 83.49%) on a 10,000-record synthetic policyholder dataset. The two-stage preprocessing pipeline (LabelEncoding + StandardScaler) was mathematically justified for each of the 17 input features. Feature importance analysis confirmed that driving behavior features (Driving Experience, Past Accidents, Speeding Violations) dominate claim prediction, accounting for 64.7% of Gradient Boosting's discriminative power—consistent with established actuarial knowledge.

The Flask web application integrates all four trained classifiers behind a secure, authenticated interface: PBKDF2-SHA256 password hashing protects user accounts; SQLite provides serverless prediction history; Chart.js delivers interactive model analytics; and Docker containerization ensures reproducible deployment on port 5002. The platform democratizes advanced claim risk assessment for insurance professionals without programming backgrounds, bridging the gap between data science capability and operational accessibility.

8.2 Future Scope

Claim Severity Regression: Extend from binary classification (claim/no-claim) to continuous claim amount prediction using Gradient Boosting Regression, enabling precise reserve estimation and premium calibration.

SMOTE and Cost-Sensitive Learning: Apply Synthetic Minority Over-sampling Technique (SMOTE) and class_weight='balanced' to improve recall for the minority claim class, reducing the 21.7% miss rate identified in the confusion matrix.

SHAP Explainability: Integrate SHAP (SHapley Additive exPlanations) for individual-level prediction explanations, supporting regulatory compliance (EU AI Act, GDPR Article 22) for automated insurance decisions.

Real-Time Telematics: Connect to IoT driving behavior sensors (acceleration, braking, cornering patterns) for dynamic risk scoring that updates continuously with actual driving data.

XGBoost and LightGBM: Benchmark against optimized GBM variants (XGBoost, LightGBM, CatBoost) which include native categorical handling, GPU acceleration, and advanced regularization.

REST API and SIEM Integration: Develop RESTful API endpoints for programmatic integration with insurance management systems and claims processing platforms.

Fraud Detection Extension: Add a dedicated anomaly detection module (Isolation Forest, DBSCAN) to identify suspicious claim filing patterns that deviate from learned risk profiles.

Admin Role Dashboard: Implement administrator capabilities for user management, batch prediction, model retraining triggers, and CSV report export.

References

[1] Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32.

[2] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189–1232.

[3] Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273–297.

[4] Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression. Wiley, 3rd Edition.

[5] Noll, A., Salzmann, R., & Wüthrich, M. V. (2018). Case Study: French Motor Third-Party Liability Claims. SSRN Electronic Journal.

[6] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. KDD 2016, 785–794.

[7] Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer.

[8] He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263–1284.

[9] Patki, N., Wedge, R., & Veeramachaneni, K. (2016). The Synthetic Data Vault. IEEE DSAA 2016.

[10] Powers, D. M. W. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. JMLR, 2(1), 37–63.

[11] Lundberg, S. M., & Lee, S. I. (2017). A Unified Approach to Interpreting Model Predictions. NeurIPS 2017.

[12] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[13] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.

[14] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.

[15] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

[16] OWASP Foundation (2021). Application Security Verification Standard 4.0. owasp.org.

[17] McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 51–56.

[18] Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 239.

[19] Insurance Information Institute (2024). Facts + Statistics: Auto Insurance. iii.org.

[20] Owens, M. (2006). The Definitive Guide to SQLite. Apress.