Machine Learning Model Complete Documentation

Healthcare Fraud Detection System

Includes: All Functions, Terms, and Definitions

Table of Contents

1. Model Overview

This document provides a complete explanation of the Machine Learning model used in the Healthcare Fraud Detection System. The model analyzes healthcare claims to identify potentially fraudulent billing patterns.

Model Summary

Property	Value
Model Type	Gradient Boosting Classifier
Training Data	5,410 providers
Original Claims	558,211 claims
Test Accuracy	94.82%
ROC-AUC Score	0.9683
Features Used	28
Training Approach	Provider-level aggregation

2. Key Terms & Definitions

A. Machine Learning Basic Terms

Machine Learning (ML)

A subset of Artificial Intelligence (AI) where computers learn patterns from data without being explicitly programmed. Instead of writing rules, we show the computer examples and it learns the rules automatically.

Model

A mathematical representation of a real-world process. In our case, the model is a mathematical function that takes claim data as input and outputs a fraud probability (0% to 100%).

Training

The process of teaching a model to recognize patterns by showing it many examples. The model adjusts its internal parameters to minimize prediction errors on the training data.

Testing

Evaluating how well a trained model performs on new, unseen data. This tells us if the model will work in the real world.

Features

The input variables or attributes used to make predictions. In our model, features include: total_claims, amount_mean, num_diagnoses_mean, etc. Features are the "X" in the equation.

Target Variable (Label)

The output variable we want to predict. In our model, the target is "is_fraud" (True or False). This is the "y" in the equation.

Prediction

The output generated by a model for new input data. Our model predicts the probability that a provider is fraudulent.

Classification

A type of ML problem where the goal is to assign data to discrete categories. Our model classifies providers as "Fraudulent" or "Legitimate".

Binary Classification

Classification with exactly two classes. Our problem is binary: Fraud (1) or Not Fraud (0).

B. Data Terms

Dataset

A collection of data organized in rows and columns. Each row is one example (claim), each column is one feature (amount, age, etc.). Our dataset has 558,211 claims with 15 columns.

Training Set

The portion of data used to train the model (80% in our case). The model learns patterns from this data.

Test Set

The portion of data held back to evaluate the model (20% in our case). The model never sees this during training, so it provides an unbiased evaluation.

Train-Test Split

Dividing the dataset into training and testing portions. We use 80-20 split: 4,328 providers for training, 1,082 for testing.

Stratification

Ensuring the class distribution in train and test sets matches the original data. If 38% of providers are fraudulent, we ensure both sets have ~38% fraud.

Class Imbalance

When one class has significantly more examples than another. We have 38% fraud vs 62% legitimate - moderate imbalance that we handle with stratification.

Feature Engineering

The process of creating new features from raw data. We create "claims_per_patient" by dividing total_claims by unique_patients.

Aggregation

Combining multiple rows into summary statistics. We aggregate 558,211 claims into 5,410 provider-level statistics (mean, sum, max, etc.).

Missing Values (NaN)

Empty or undefined values in the dataset. We fill these with 0 using fillna(0).

Infinity Values (Inf)

Infinitely large values that occur from division by zero. We replace these with 0.

C. Model Algorithm Terms

Decision Tree

A flowchart-like model that makes decisions by splitting data based on feature values. Like a series of yes/no questions: "Is amount > 10000?" If yes, go left; if no, go right.

Ensemble

A method that combines multiple models to make better predictions than any single model. Like asking multiple experts and taking a vote.

Random Forest

An ensemble of many decision trees trained on random subsets of data. Each tree votes, and the majority wins. We train 200 trees in parallel.

Gradient Boosting

An ensemble method that builds trees sequentially. Each new tree corrects the errors of previous trees. We use Gradient Boosting Classifier with 200 trees.

Boosting

Training models sequentially where each model focuses on fixing mistakes of previous models. Like a student who studies their wrong answers.

Learning Rate

How much each tree contributes to the final prediction. A smaller learning rate (0.1) means each tree has less influence but the overall model is more robust.

n_estimators

The number of trees in the ensemble. We use 200 trees. More trees = better accuracy but slower training.

max_depth

Maximum depth of each decision tree. Depth 6 means at most 6 levels of decisions. Prevents overfitting.

min_samples_split

Minimum number of samples required to split a node. If a node has fewer samples, it becomes a leaf. Prevents overfitting.

min_samples_leaf

Minimum number of samples required in a leaf node. Ensures each prediction is based on sufficient data.

Subsample

Fraction of samples used to train each tree. 0.8 means each tree sees 80% of the data, adding randomness.

D. Data Preprocessing Terms

Feature Scaling

Transforming features to a similar scale. Different features have different ranges (amount: 0-50000, ratio: 0-1). Scaling puts them on equal footing.

Standardization (Z-Score Normalization)

Scaling features to have mean=0 and standard deviation=1. Formula: z = (x - mean) / std. Example: If amount mean=5000, std=2000, then 7000 becomes (7000-5000)/2000 = 1.0

StandardScaler

A sklearn class that performs standardization. It computes mean and std from training data, then applies the transformation.

fit()

Learn the parameters (mean, std) from training data. Called once on training set.

transform()

Apply the learned parameters to data. Called on both training and test sets.

fit_transform()

Convenience method that does fit() followed by transform() in one step.

Label Encoding

Converting categorical labels to integers. Example: "Inpatient"=0, "Outpatient"=1. Required because ML models work with numbers.

One-Hot Encoding

Converting categories to binary columns. "Inpatient" becomes [1,0], "Outpatient" becomes [0,1]. We did not use this.

3. Model Evaluation Metrics

Evaluation metrics tell us how well our model performs. Different metrics capture different aspects of performance.

A. Basic Metrics

Accuracy

Percentage of correct predictions out of total predictions.

Formula: (True Positives + True Negatives) / All Predictions

Our model: 94.82% accuracy means it correctly classifies ~95 out of 100 providers.

True Positive (TP)

Model correctly predicts fraud when there is actually fraud. The best outcome! Our model achieved 406 TPs.

True Negative (TN)

Model correctly predicts legitimate when there is no fraud. Also good! Our model achieved 620 TNs.

False Positive (FP)

Model incorrectly predicts fraud when provider is actually legitimate. A false alarm. We have 38 FPs.

False Negative (FN)

Model fails to catch actual fraud. The worst outcome! We have 18 FNs - 18 fraudulent providers we missed.

Confusion Matrix

A table showing TP, TN, FP, FN counts. Helps visualize model performance:

```
                Predicted
             Legit    Fraud
Actual Legit [  620  ,   38  ]
       Fraud [   18  ,  406  ]
```

B. Advanced Metrics

Precision

Of all providers we flagged as fraud, how many are actually fraud?

Formula: TP / (TP + FP) = 406 / (406 + 38) = 91.4%

High precision means few false alarms.

Recall (Sensitivity)

Of all actual fraud cases, how many did we catch?

Formula: TP / (TP + FN) = 406 / (406 + 18) = 95.8%

High recall means we catch most fraud.

F1-Score

Harmonic mean of Precision and Recall. Balances both metrics.

Formula: 2 * (Precision * Recall) / (Precision + Recall)

Our F1: 2 * (0.914 * 0.958) / (0.914 + 0.958) = 93.5%

Specificity

Of all legitimate providers, how many did we correctly identify?

Formula: TN / (TN + FP) = 620 / (620 + 38) = 94.2%

ROC Curve

Receiver Operating Characteristic curve. Plots True Positive Rate (Recall) vs False Positive Rate at different thresholds.

Shows trade-off between catching fraud and false alarms.

AUC (Area Under Curve)

Area under the ROC curve. Ranges from 0 to 1.

1.0 = Perfect classifier

0.9+ = Excellent

0.8-0.9 = Good

0.5 = Random guess

Our AUC: 0.9683 (Excellent!)

Threshold

The probability cutoff for classification. Default is 0.5.

If probability >= 0.5, predict fraud.

Lower threshold = catch more fraud but more false alarms.

Higher threshold = fewer false alarms but miss more fraud.

C. Cross-Validation Terms

Cross-Validation

A technique to evaluate model performance by training and testing on different subsets of data multiple times. More reliable than a single train-test split.

K-Fold Cross-Validation

Divide data into K equal parts (folds). Train on K-1 folds, test on remaining fold. Repeat K times. Average the results.
We use 5-fold: Each fold is tested once while other 4 are used for training.

Fold

One subset of data in cross-validation. With 5,410 providers and 5 folds, each fold has ~1,082 providers.

Mean CV Score

Average performance across all folds. Our mean: 94.14%

Standard Deviation (CV)

Variation in performance across folds. Our std: 0.71%.
Low std means consistent performance across different data subsets.

D. Overfitting & Underfitting

Overfitting

When a model learns training data too well, including noise and outliers. It performs great on training data but poorly on new data.
Signs: Training accuracy >> Test accuracy
Our model: Training 100%, Test 94.82% - slight overfitting, acceptable.

Underfitting

When a model is too simple to capture patterns in the data. Performs poorly on both training and test data.
Signs: Both training and test accuracy are low.

Generalization

How well a model performs on new, unseen data. A well-generalized model works in the real world, not just on training data.

Regularization

Techniques to prevent overfitting by penalizing model complexity. Examples: limiting tree depth, requiring minimum samples in leaves.

Bias

Error due to overly simplistic assumptions. High bias = underfitting. The model is too simple to learn patterns.

Variance

Error due to sensitivity to small fluctuations in training data. High variance = overfitting. Model learns noise.

Bias-Variance Tradeoff

Balancing bias and variance. Too simple = high bias. Too complex = high variance. Goal: Find the sweet spot.

4. Sklearn Functions Reference

A. Model Selection Functions

train_test_split()

Purpose: Splits data into training and testing sets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,              # Features
    y,              # Target
    test_size=0.2,  # 20% for testing
    random_state=42 # Reproducibility
)
```

cross_val_score()

Purpose: Performs K-fold cross-validation and returns scores for each fold.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5)
print(f"Mean: {scores.mean():.2%}")
```

B. Preprocessing Functions

StandardScaler()

Purpose: Standardizes features to mean=0, std=1.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

LabelEncoder()

Purpose: Converts categorical labels to integers.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
encoded = le.fit_transform(["Cat", "Dog", "Cat"])
# Result: [0, 1, 0]
```

C. Classifier Functions

GradientBoostingClassifier()

Purpose: Ensemble classifier that builds trees sequentially.

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier(
    n_estimators=200,     # Number of trees
    learning_rate=0.1,    # Step size
    max_depth=6           # Tree depth
)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

RandomForestClassifier()

Purpose: Ensemble of decision trees trained in parallel.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    n_jobs=-1  # Use all CPU cores
)
```

D. Evaluation Functions

accuracy_score()

Purpose: Calculate fraction of correct predictions.

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_true, y_pred)
# Returns: 0.9482 (94.82%)
```

roc_auc_score()

Purpose: Calculate Area Under ROC Curve.

```
from sklearn.metrics import roc_auc_score

auc = roc_auc_score(y_true, y_prob)
# Returns: 0.9683
```

confusion_matrix()

Purpose: Generate confusion matrix (TP, TN, FP, FN).

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_true, y_pred)
# Returns: [[TN, FP], [FN, TP]]
```

classification_report()

Purpose: Generate precision, recall, F1 report.

```
from sklearn.metrics import classification_report


print(classification_report(y_true, y_pred))
```

5. Custom Functions

Function 1: create_provider_features(df)

Location: ml/train_model.py (Lines 57-238)

Purpose: Aggregates individual claims into provider-level statistics. This is the KEY function that improves accuracy from 62% to 94%.

Parameters:

df (pandas.DataFrame): Claims data with 558,211 rows

Returns:

pandas.DataFrame: One row per provider (5,410 rows) with 28 features

What it does:

1. Groups claims by provider_id
2. Calculates statistics: mean, sum, std, max, min
3. Creates derived features: claims_per_patient, revenue_per_patient
4. Calculates inpatient_ratio

Function 2: train_enhanced_model()

Location: ml/train_model.py (Lines 245-543)

Purpose: Main entry point for training the fraud detection model.

Parameters:

None (uses hardcoded file paths)

Returns:

None (saves model to ml/model.pkl)

What it does:

1. Loads claims data
2. Creates provider features
3. Splits into train/test
4. Scales features
5. Trains Gradient Boosting and Random Forest
6. Selects best model
7. Saves model to disk

6. Complete ML Workflow

Training Pipeline

Step 1: Load Data

Load 558,211 claims from claims.csv

Step 2: Aggregate

Group by provider_id, creating 5,410 provider records

Step 3: Feature Engineering

Create 28 features per provider

Step 4: Split Data

80% training (4,328), 20% testing (1,082)

Step 5: Scale Features

Standardize to mean=0, std=1

Step 6: Train Models

Train Gradient Boosting and Random Forest

Step 7: Evaluate

Compare accuracy and ROC-AUC

Step 8: Select Best

Choose model with higher AUC

Step 9: Save

Export to ml/model.pkl

Prediction Pipeline (At Runtime)

Step 1: Load Model

Load trained model from model.pkl

Step 2: Extract Features

Calculate 28 features from claim

Step 3: Scale

Apply same scaling as training

Step 4: Predict

Get fraud probability (0-100%)

Step 5: Classify

Assign risk level based on probability

Summary

This document covered:

1. Machine Learning fundamentals and terminology
2. Data preprocessing techniques
3. Feature engineering for fraud detection
4. Model training with ensemble methods
5. Evaluation metrics and their interpretations
6. All sklearn functions used
7. Custom functions in our codebase
8. Complete training and prediction workflows

The Healthcare Fraud Detection model achieves 94.82% accuracy using Gradient Boosting Classifier trained on provider-level aggregated features from 558,211 claims.

Document generated for Healthcare Fraud Detection Project