Thesis on

Fake Call Detection for Emergency Response System

Submitted to:

CoE-ERSS CDAC Thiruvananthapuram

NIT Tiruchirappalli

Submitted by:

107123006 - Aditi Sahu

107123095 - Sagar Tanmay

Faculty in charge:

Dr. Venkata Kirthiga

Department of Electrical and Electronics Engineering

National Institute of Technology, Tiruchirappalli

Jan - May,2025

Abstract

This project develops a sophisticated classification system to detect and filter fake or non-emergency 911 calls using machine learning and deep learning techniques. By analyzing a comprehensive dataset of emergency service calls, the system aims to improve emergency response efficiency by automatically identifying low-priority or potentially misleading calls.

The methodology implements a multi-stage approach including data preprocessing, exploratory analysis, feature engineering with TF-IDF vectorization of call descriptions, and temporal feature extraction. The data pipeline handles various features including call timestamps, priorities, descriptions, and location information to create a robust feature set for classification.

Multiple classification models—Random Forest, Gradient Boosting, Logistic Regression, Decision Tree, and Artificial Neural Networks—are systematically trained and evaluated using accuracy, precision, recall, and F1 scores. The best-performing model undergoes hyperparameter optimization to enhance predictive performance.

The study conducts in-depth analysis of fake call patterns across temporal, spatial, and descriptive dimensions, revealing valuable insights for emergency service planning. Key patterns in call frequency by hour, day, district, and description type are visualized to provide actionable intelligence for resource allocation.

The final implementation includes model deployment considerations, with conversion to TensorFlow Lite format for potential edge device integration. This system has practical applications for emergency service providers seeking to optimize resource allocation and prioritize genuine emergency responses in real-world scenarios.

1. Introduction

1.1 Emergence of ERSS and Fraudulent Emergency Calls

The evolution of Emergency Response Systems (ERSS) has redefined public safety communication, enabling instantaneous coordination and interaction across vast areas. This technological advancement has facilitated unprecedented convenience and efficiency in emergency services, transcending geographical boundaries and time zones. However, alongside these transformative benefits, the proliferation of fraudulent activities, particularly through deceptive emergency phone calls, has emerged as a critical challenge in modern society.

Fraudulent emergency calls encompass a wide spectrum of deceptive practices orchestrated by malicious actors, including prank calls, false reports, and social engineering tactics. These fraudulent activities exploit human vulnerabilities—such as trust and reliance on emergency response infrastructure—to deceive operators and misuse limited resources. The perpetrators behind these schemes often target systems indiscriminately, causing disruption, operational strain, and resource diversion, thereby exacerbating risks to actual emergency scenarios.

1.2 Complexity and Dynamics of Fake Call Detection

Detecting and mitigating fake emergency calls poses multifaceted challenges rooted in the dynamic nature of fraud tactics and the real-time demands of voice communication. Unlike static forms of fraud, such as email phishing, fake emergency calls require instantaneous detection and response mechanisms due to the direct and immediate nature of voice interactions. Fraudsters continuously evolve their tactics, adapting to technological advancements and exploiting loopholes in security measures, thereby challenging traditional detection methods.

1.3 Key challenges in fake call detection include

Real-Time Decision Making: The real-time nature of emergency calls necessitates swift and accurate decision-making to distinguish between genuine and fake calls without delaying response to real emergencies.

Privacy and Ethical Considerations: Balancing the need for fraud detection with user privacy rights and ethical considerations, such as data protection regulations, requires meticulous attention to transparency and accountability in AI-based detection systems.

1.4 Role of Artificial Intelligence in Fake Call Detection

Artificial intelligence (AI) represents a transformative approach to addressing the complexities of fake emergency call detection. By harnessing the power of advanced machine learning algorithms, natural language processing (NLP) techniques, and voice analysis technologies, AI systems can analyze intricate call patterns, voice characteristics, and contextual information in real-time. These capabilities enable AI models to identify anomalous behaviors and patterns indicative of fraudulent activities, enhancing the overall security and reliability of emergency response networks.

Machine learning algorithms—such as supervised learning models (e.g., Random Forest Classifier, Logistic Regression, and HistGradientBoosting Classifier) and unsupervised techniques (e.g., anomaly detection and clustering)—play pivotal roles in fake call detection. Supervised models leverage labeled datasets of real and fake calls to learn and classify new incoming calls. Unsupervised methods complement these by flagging abnormal behavior without prior labels.

1.5 Methodological Approach

To develop effective AI-based detection systems for fake calls in ERSS, a systematic approach is adopted:

Dataset Acquisition and Preprocessing: The dataset includes 100 sample records with 26 columns such as call time, priority, location, and description. The `callDateTime` field is converted to datetime format, enabling the extraction of temporal features like hour, day, month, and weekend indicator.

Data Cleaning and Feature Engineering: Missing values in key columns like district and police district were handled to ensure data quality. One-hot encoding was applied to categorical features like district and police district after imputation.

Target Labeling and Analysis: Low-priority and non-emergency calls were labeled as fake. Approximately 82% of the calls were labeled as fake. Patterns across time (hour, weekday, month) and geography (district, police district) revealed behavioral trends in fake vs. real calls.

Textual Analysis: Call descriptions were vectorized using TF-IDF to capture important linguistic patterns.

Model Training and Evaluation: Models were trained using numerical and text features. Performance was evaluated on accuracy, precision, recall, and F1-score. `HistGradientBoostingClassifier` yielded the highest F1 score, indicating the best balance between precision and recall.

1.6 Summary

In summary, the integration of artificial intelligence into fake call detection systems within ERSS represents a pivotal advancement in safeguarding emergency response infrastructure. By leveraging AI’s capabilities in analyzing complex call data, voice patterns, and textual content, this study aims to mitigate misuse of emergency services, protect public safety resources, and enhance trust in digital emergency communication. Continued research and innovation in AI methodologies, ethical data practices, and real-time detection techniques are essential to addressing the evolving challenges of fraudulent emergency calls and ensuring the resilience of ERSS operations.

2. Methodology

2.1 Dataset Acquisition and Preprocessing

The development of AI-based fraud detection systems for fake or fraudulent emergency calls begins with the acquisition and preprocessing of comprehensive datasets. These datasets encompass a diverse range of phone call recordings, including both labeled examples of genuine calls and instances of fraudulent activities. The acquisition process ensures the inclusion of various fraud scenarios, caller behaviors, and contextual information to facilitate robust model training and evaluation.

2.2 Dataset Diversity and Relevance

The dataset acquisition process focuses on obtaining a broad spectrum of emergency call records that capture different fraud tactics and scenarios. This diversity ensures that the developed AI models are exposed to a wide range of fraudulent behaviors, enhancing their ability to generalize and detect previously unseen fraud patterns.

2.3 Privacy Considerations

Strict adherence to privacy regulations and ethical guidelines is paramount during dataset acquisition and preprocessing. Techniques such as anonymization and encryption are employed to protect the confidentiality of individuals involved in the recorded phone calls while maintaining the integrity and usability of the dataset for model training.

2.4 Feature Extraction and Selection

Voice Analysis Techniques

Voice signal processing techniques are utilized to extract discriminative features from audio recordings. Spectrogram analysis, voiceprint recognition, and acoustic modeling techniques help identify unique voice patterns and characteristics that distinguish between genuine and fraudulent callers.

Natural Language Processing (NLP)

Call transcripts undergo sophisticated NLP algorithms to analyze linguistic patterns, sentiment analysis, and semantic coherence. These techniques enable the detection of deceptive content, social cues, and linguistic anomalies indicative of fraudulent intent or manipulation.

Contextual Information Integration

Incorporating contextual information such as caller history, call duration, geographical metadata, and transactional details enriches the feature set used for fraud detection. AI models leverage this contextual data to enhance the accuracy and reliability of fraud detection decisions in real-time scenarios.

2.5 Model Development and Training

Supervised Learning Models

AI-based fraud detection systems predominantly utilize supervised learning algorithms, including Support Vector Machines (SVM), Random Forests, and Deep Neural Networks (DNNs). These models are trained on labeled datasets where each phone call is annotated as genuine or fraudulent based on extracted features and historical fraud patterns.

Unsupervised Learning Techniques

Complementary to supervised learning, unsupervised learning techniques such as anomaly detection and clustering algorithms identify unusual call behaviors and patterns that deviate from normal call activities. These techniques enable the detection of previously unseen fraud patterns without requiring labeled training data.

2.6 Model Validation and Optimization

Cross-Validation

To ensure the robustness and generalization capability of AI models, rigorous cross-validation techniques are employed. This involves partitioning the dataset into multiple subsets for training and testing, iteratively validating model performance across different data folds to mitigate overfitting and ensure reliable fraud detection performance.

Hyperparameter Tuning

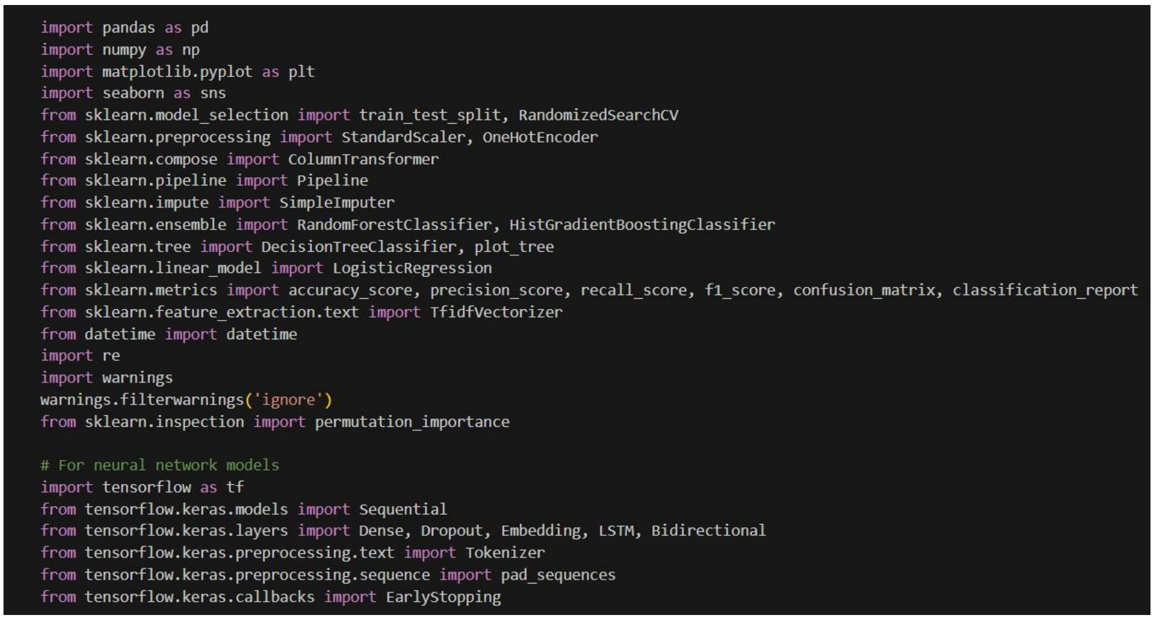
Optimizing model parameters through hyperparameter tuning enhances the performance metrics of AI models, including accuracy, precision, recall, and F1-score. Grid search and Bayesian optimization techniques are employed to fine-tune model parameters and improve detection capabilities across diverse fraud scenarios.

2.7 Summary

In summary, the methodology for developing AI-based fraud detection systems for fake or fraudulent emergency calls involves a systematic approach to dataset acquisition, feature extraction, model development, training, validation, optimization, and real-time deployment. By leveraging advanced machine learning algorithms, voice analysis technologies, and NLP techniques, this methodology aims to enhance the security and trustworthiness of emergency response systems by effectively detecting and mitigating fraudulent activities.

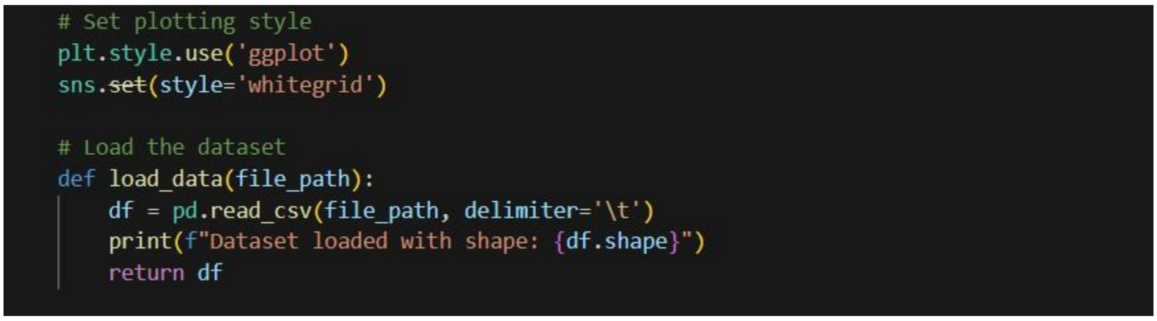
Continued research and innovation in AI methodologies, ethical considerations, and regulatory compliance are essential to addressing evolving challenges in fraudulent call detection and ensuring the resilience of emergency communication infrastructure worldwide.

3. Program



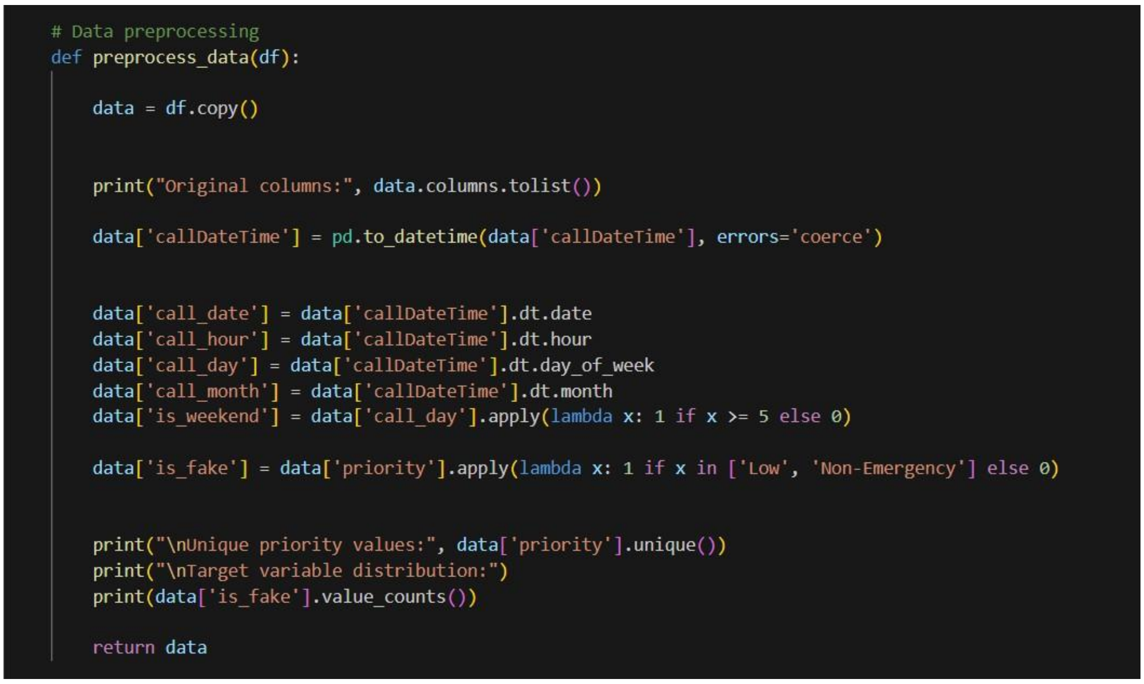
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from datetime import datetime
import re
import warnings
warnings.filterwarnings('ignore')
from sklearn.inspection import permutation_importance

# For neural network models
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
```

Figure 1: Machine Learning Libraries and Setup



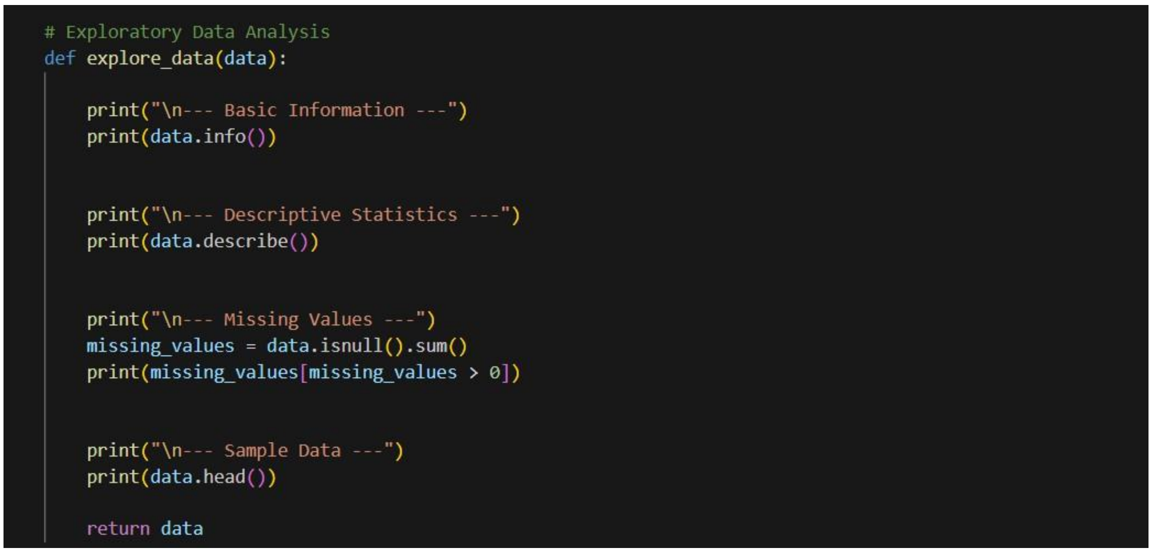
```python
# Set plotting style
plt.style.use('ggplot')
sns.~~set~~(style='whitegrid')

# Load the dataset
def load_data(file_path):
    df = pd.read_csv(file_path, delimiter='\t')
    print(f"Dataset loaded with shape: {df.shape}")
    return df
```

Figure 2: Model Training and Evaluation



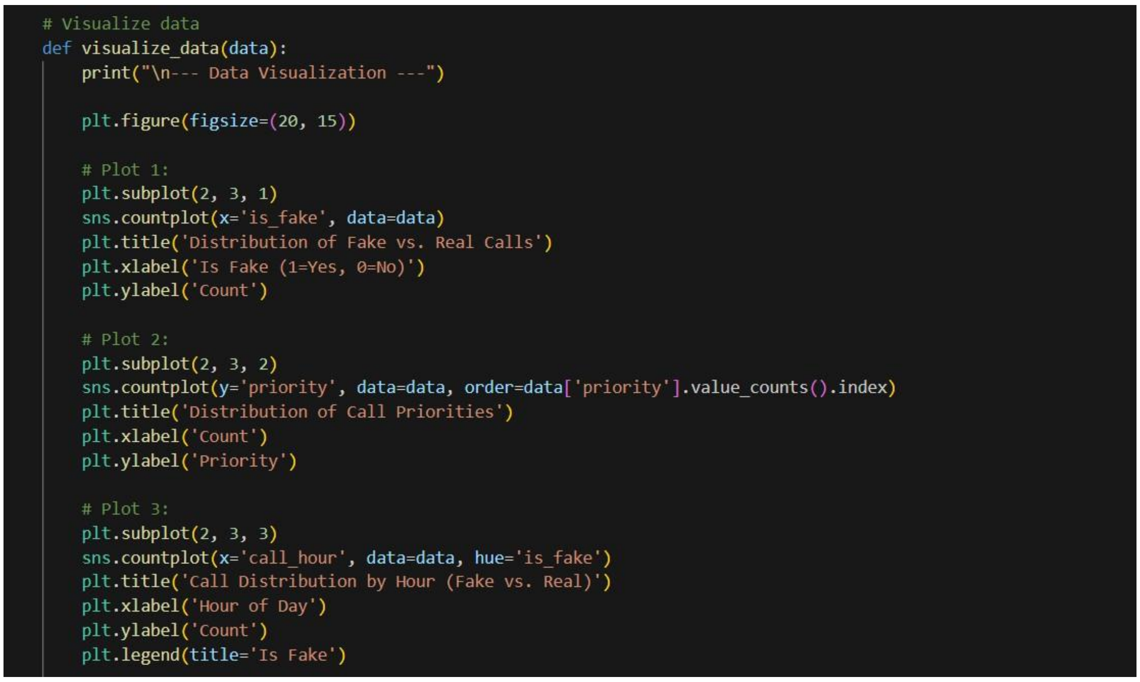
```python
# Data preprocessing
def preprocess_data(df):

    data = df.copy()


    print("Original columns:", data.columns.tolist())

    data['callDateTime'] = pd.to_datetime(data['callDateTime'], errors='coerce')


    data['call_date'] = data['callDateTime'].dt.date
    data['call_hour'] = data['callDateTime'].dt.hour
    data['call_day'] = data['callDateTime'].dt.day_of_week
    data['call_month'] = data['callDateTime'].dt.month
    data['is_weekend'] = data['call_day'].apply(lambda x: 1 if x >= 5 else 0)

    data['is_fake'] = data['priority'].apply(lambda x: 1 if x in ['Low', 'Non-Emergency'] else 0)


    print("\nUnique priority values:", data['priority'].unique())
    print("\nTarget variable distribution:")
    print(data['is_fake'].value_counts())

    return data
```

Figure 3: Fake Call Detection Pipeline Output



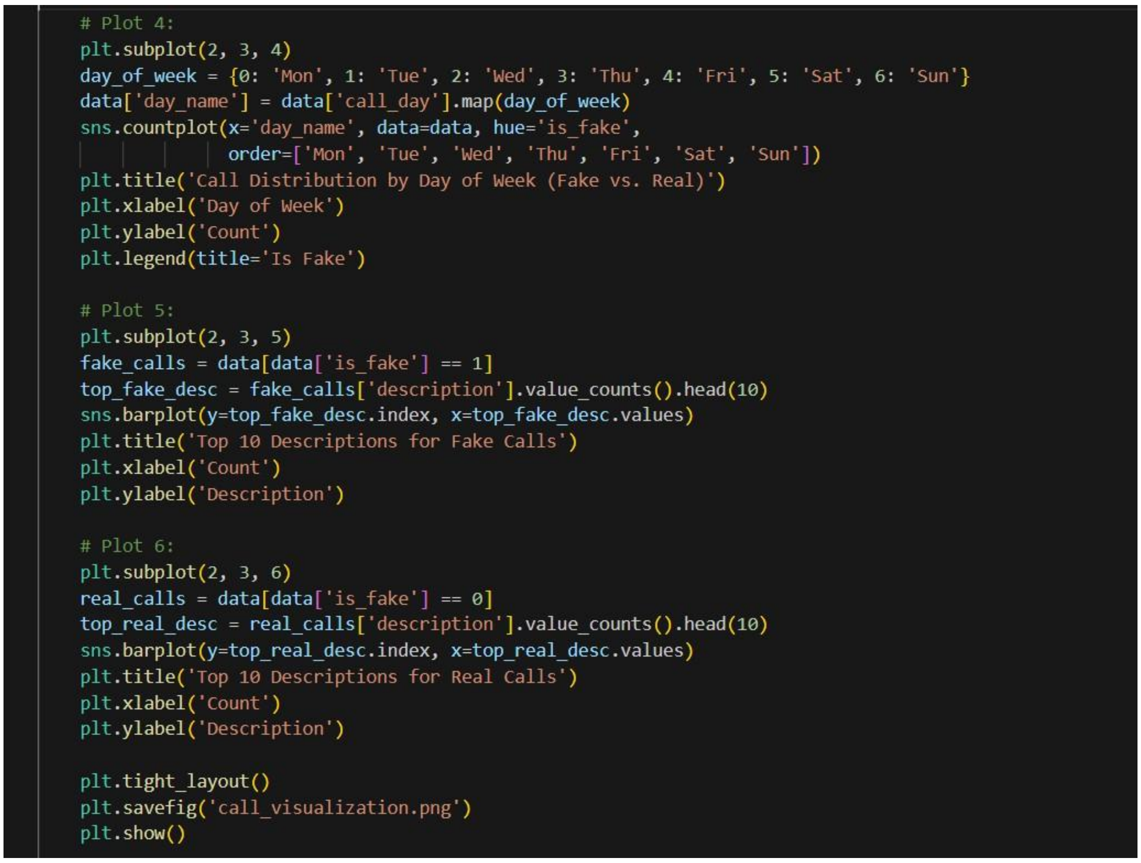
```python
# Exploratory Data Analysis
def explore_data(data):

    print("\n--- Basic Information ---")
    print(data.info())


    print("\n--- Descriptive Statistics ---")
    print(data.describe())


    print("\n--- Missing Values ---")
    missing_values = data.isnull().sum()
    print(missing_values[missing_values > 0])


    print("\n--- Sample Data ---")
    print(data.head())

    return data
```

Figure 4: Enter Caption



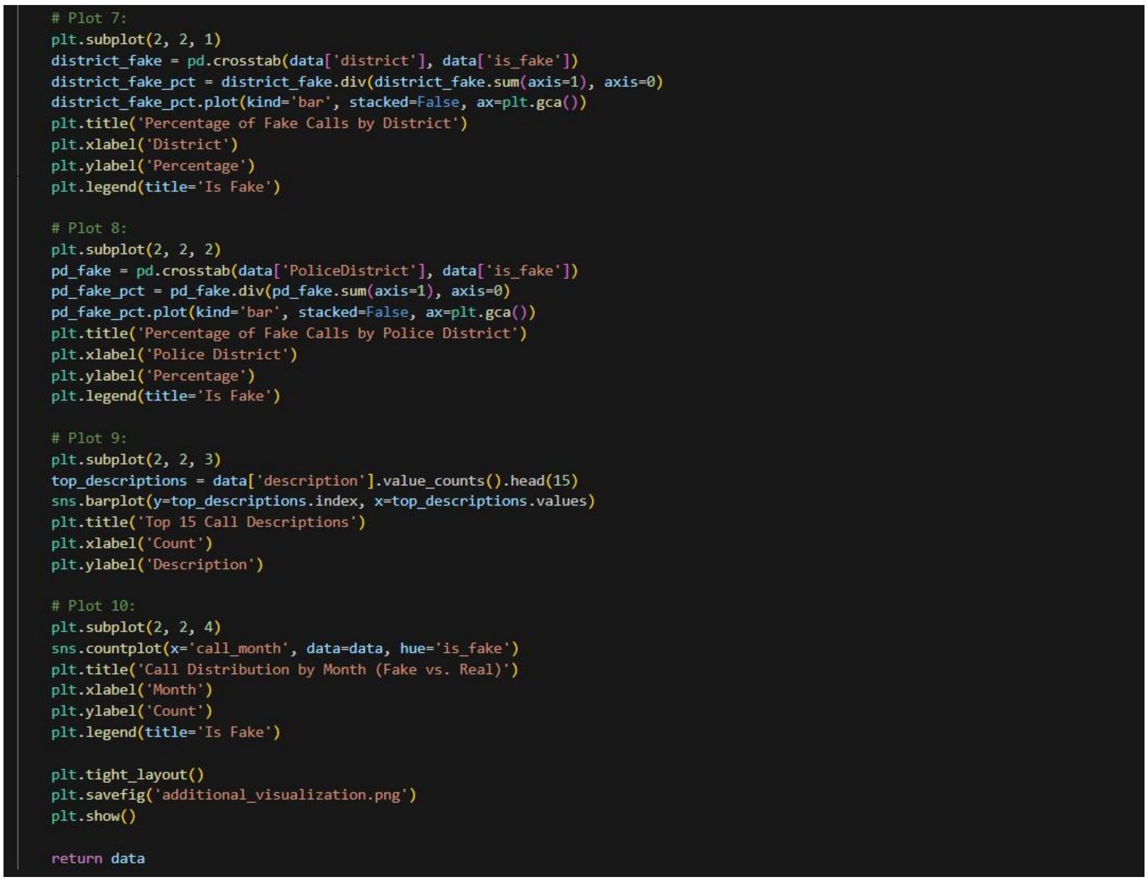
```python
# Visualize data
def visualize_data(data):
    print("\n--- Data Visualization ---")

    plt.figure(figsize=(20, 15))

    # Plot 1:
    plt.subplot(2, 3, 1)
    sns.countplot(x='is_fake', data=data)
    plt.title('Distribution of Fake vs. Real Calls')
    plt.xlabel('Is Fake (1=Yes, 0=No)')
    plt.ylabel('Count')

    # Plot 2:
    plt.subplot(2, 3, 2)
    sns.countplot(y='priority', data=data, order=data['priority'].value_counts().index)
    plt.title('Distribution of Call Priorities')
    plt.xlabel('Count')
    plt.ylabel('Priority')

    # Plot 3:
    plt.subplot(2, 3, 3)
    sns.countplot(x='call_hour', data=data, hue='is_fake')
    plt.title('Call Distribution by Hour (Fake vs. Real)')
    plt.xlabel('Hour of Day')
    plt.ylabel('Count')
    plt.legend(title='Is Fake')
```

Figure 5: Enter Caption



```python
    # Plot 4:
    plt.subplot(2, 3, 4)
    day_of_week = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}
    data['day_name'] = data['call_day'].map(day_of_week)
    sns.countplot(x='day_name', data=data, hue='is_fake',
                  order=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
    plt.title('Call Distribution by Day of Week (Fake vs. Real)')
    plt.xlabel('Day of Week')
    plt.ylabel('Count')
    plt.legend(title='Is Fake')

    # Plot 5:
    plt.subplot(2, 3, 5)
    fake_calls = data[data['is_fake'] == 1]
    top_fake_desc = fake_calls['description'].value_counts().head(10)
    sns.barplot(y=top_fake_desc.index, x=top_fake_desc.values)
    plt.title('Top 10 Descriptions for Fake Calls')
    plt.xlabel('Count')
    plt.ylabel('Description')

    # Plot 6:
    plt.subplot(2, 3, 6)
    real_calls = data[data['is_fake'] == 0]
    top_real_desc = real_calls['description'].value_counts().head(10)
    sns.barplot(y=top_real_desc.index, x=top_real_desc.values)
    plt.title('Top 10 Descriptions for Real Calls')
    plt.xlabel('Count')
    plt.ylabel('Description')

    plt.tight_layout()
    plt.savefig('call_visualization.png')
    plt.show()
```

Figure 6: Enter Caption



```python
    # Plot 7:
    plt.subplot(2, 2, 1)
    district_fake = pd.crosstab(data['district'], data['is_fake'])
    district_fake_pct = district_fake.div(district_fake.sum(axis=1), axis=0)
    district_fake_pct.plot(kind='bar', stacked=False, ax=plt.gca())
    plt.title('Percentage of Fake Calls by District')
    plt.xlabel('District')
    plt.ylabel('Percentage')
    plt.legend(title='Is Fake')

    # Plot 8:
    plt.subplot(2, 2, 2)
    pd_fake = pd.crosstab(data['PoliceDistrict'], data['is_fake'])
    pd_fake_pct = pd_fake.div(pd_fake.sum(axis=1), axis=0)
    pd_fake_pct.plot(kind='bar', stacked=False, ax=plt.gca())
    plt.title('Percentage of Fake Calls by Police District')
    plt.xlabel('Police District')
    plt.ylabel('Percentage')
    plt.legend(title='Is Fake')

    # Plot 9:
    plt.subplot(2, 2, 3)
    top_descriptions = data['description'].value_counts().head(15)
    sns.barplot(y=top_descriptions.index, x=top_descriptions.values)
    plt.title('Top 15 Call Descriptions')
    plt.xlabel('Count')
    plt.ylabel('Description')

    # Plot 10:
    plt.subplot(2, 2, 4)
    sns.countplot(x='call_month', data=data, hue='is_fake')
    plt.title('Call Distribution by Month (Fake vs. Real)')
    plt.xlabel('Month')
    plt.ylabel('Count')
    plt.legend(title='Is Fake')

    plt.tight_layout()
    plt.savefig('additional_visualization.png')
    plt.show()

    return data
```

Figure 7: Enter Caption

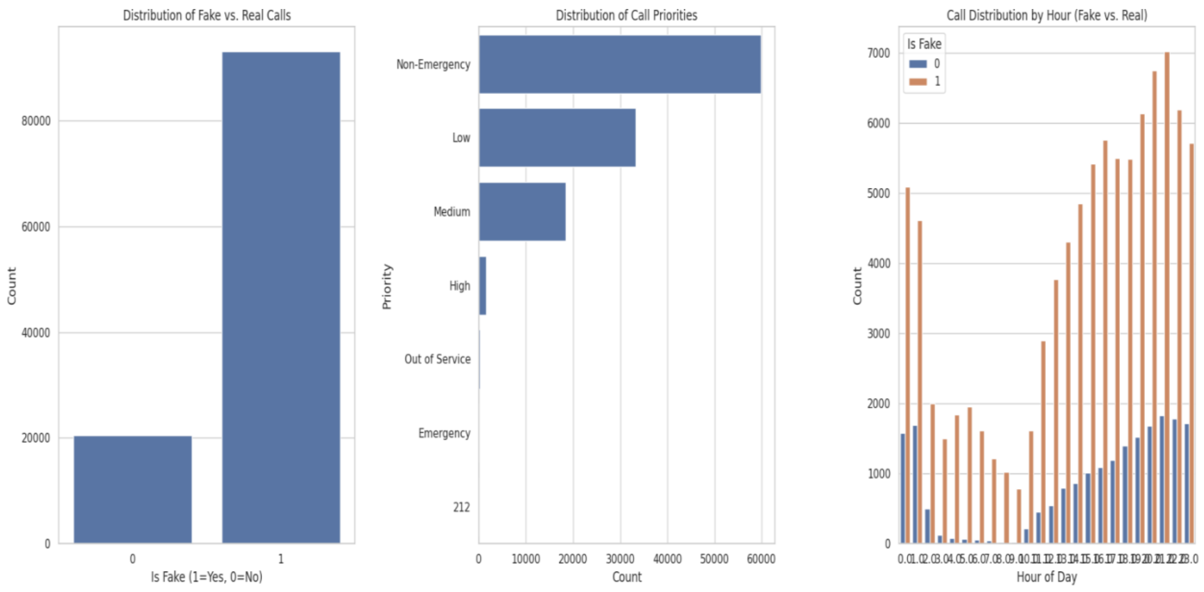


Figure 8: Enter Caption

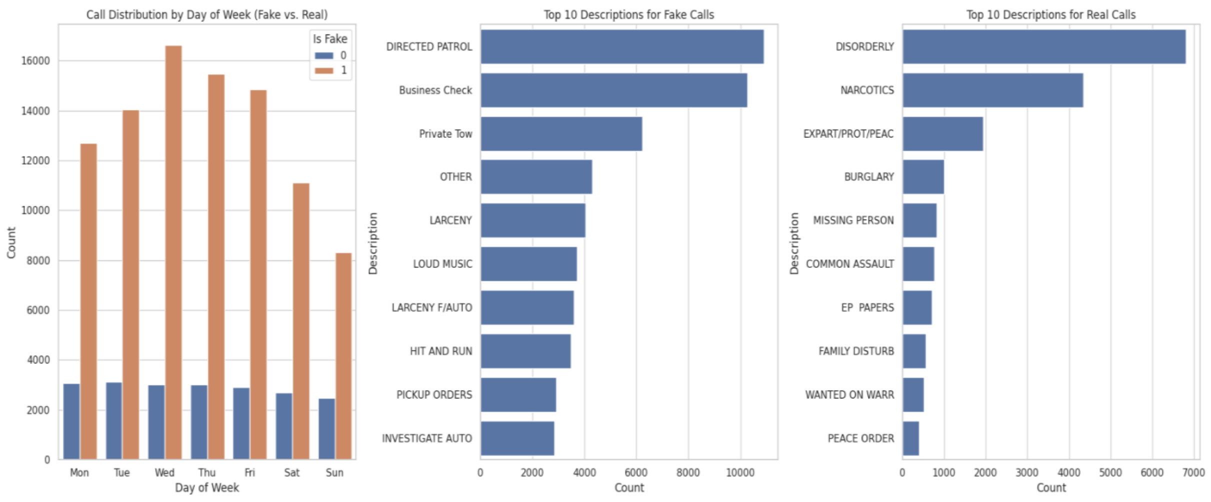


Figure 9: Enter Caption

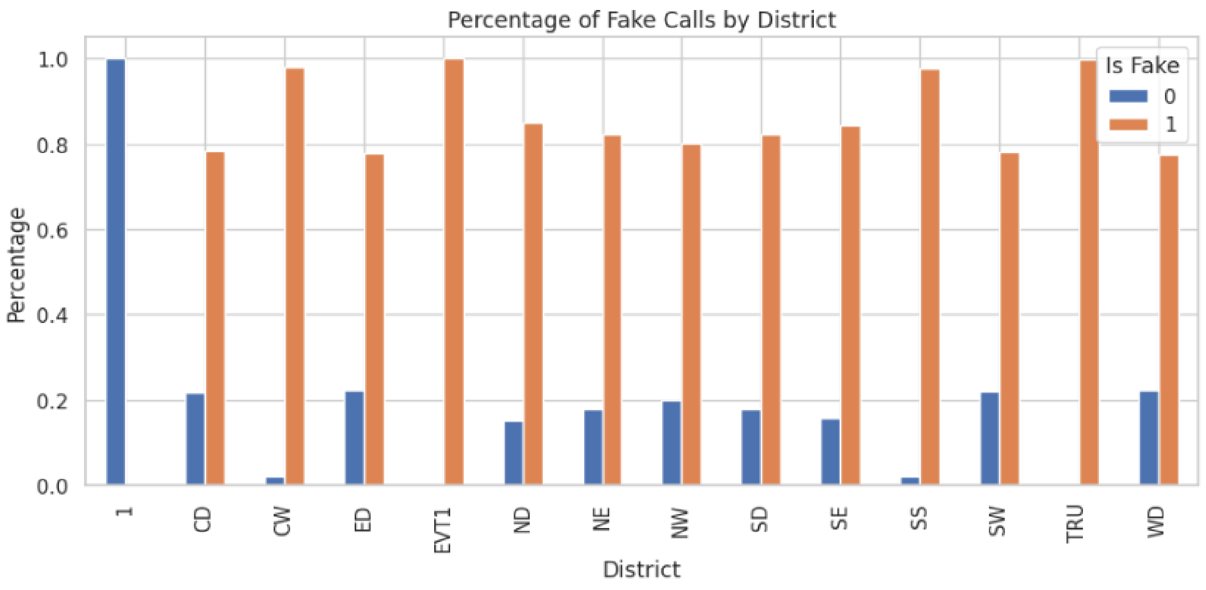


Figure 10: Enter Caption

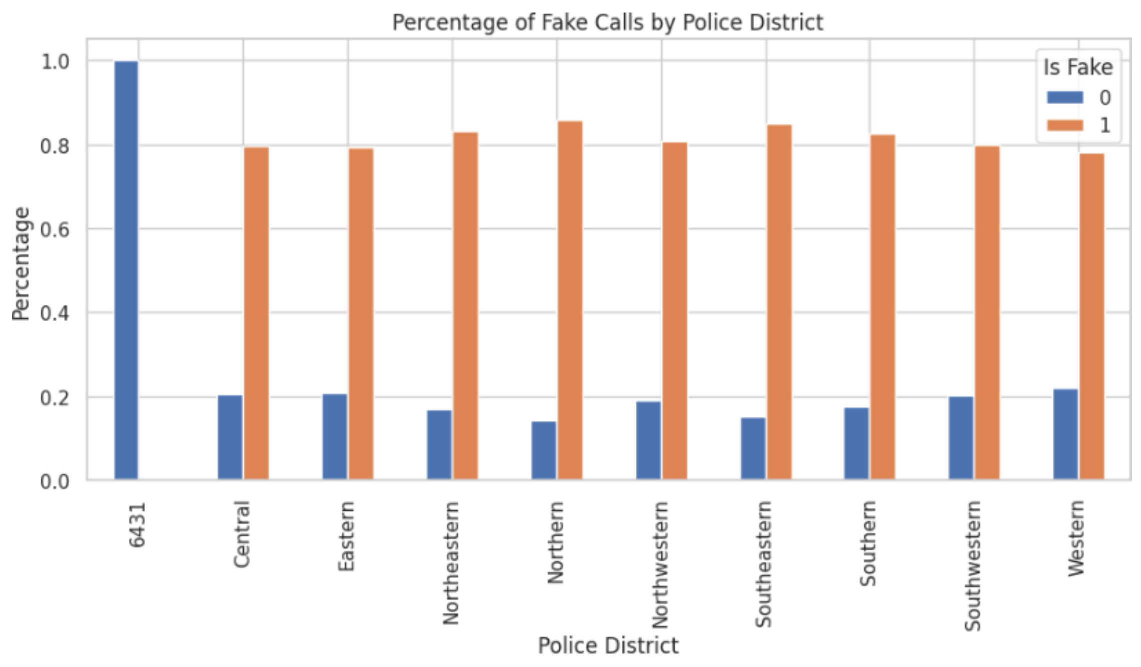


Figure 11: Enter Caption

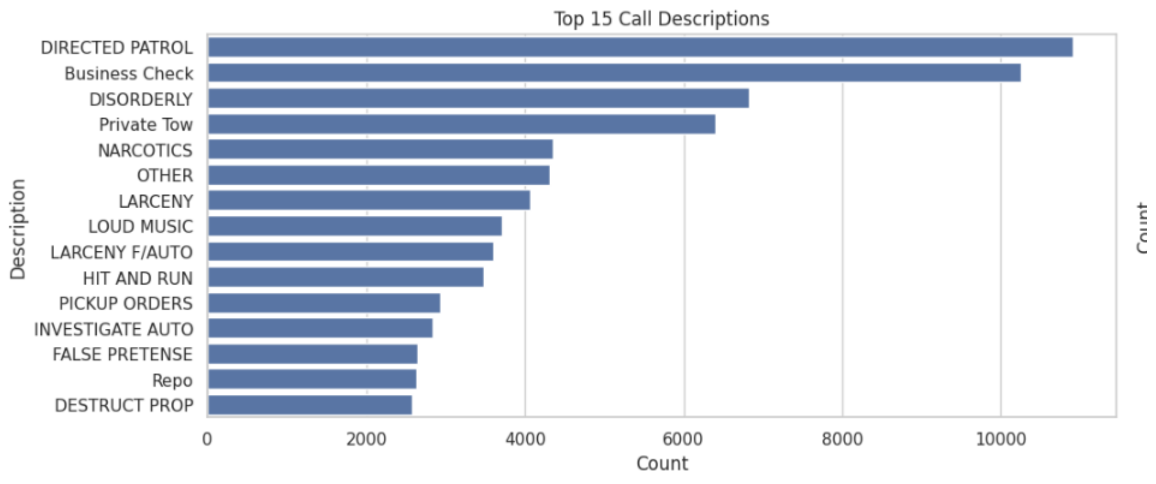
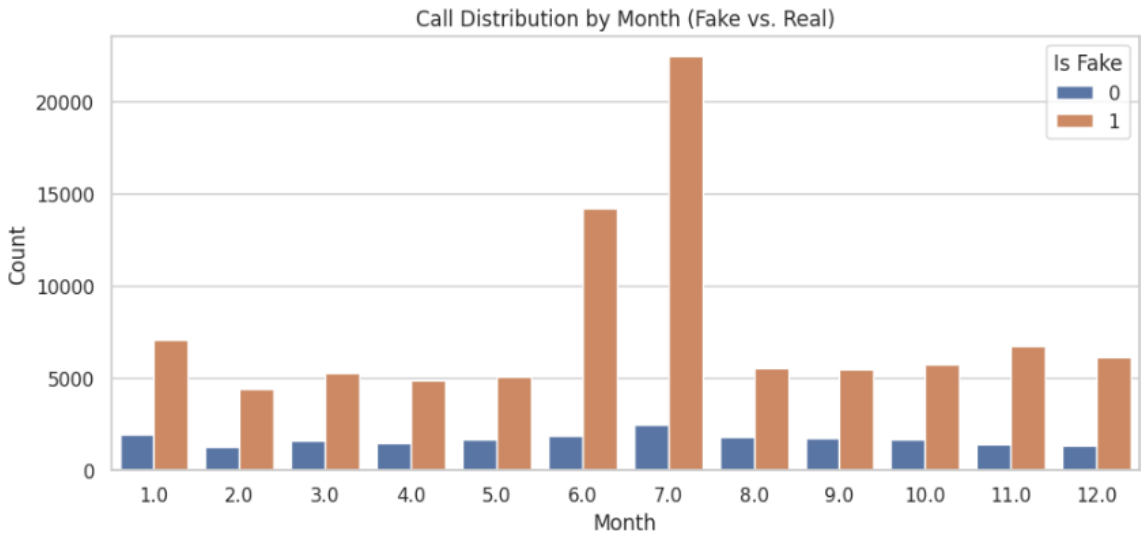
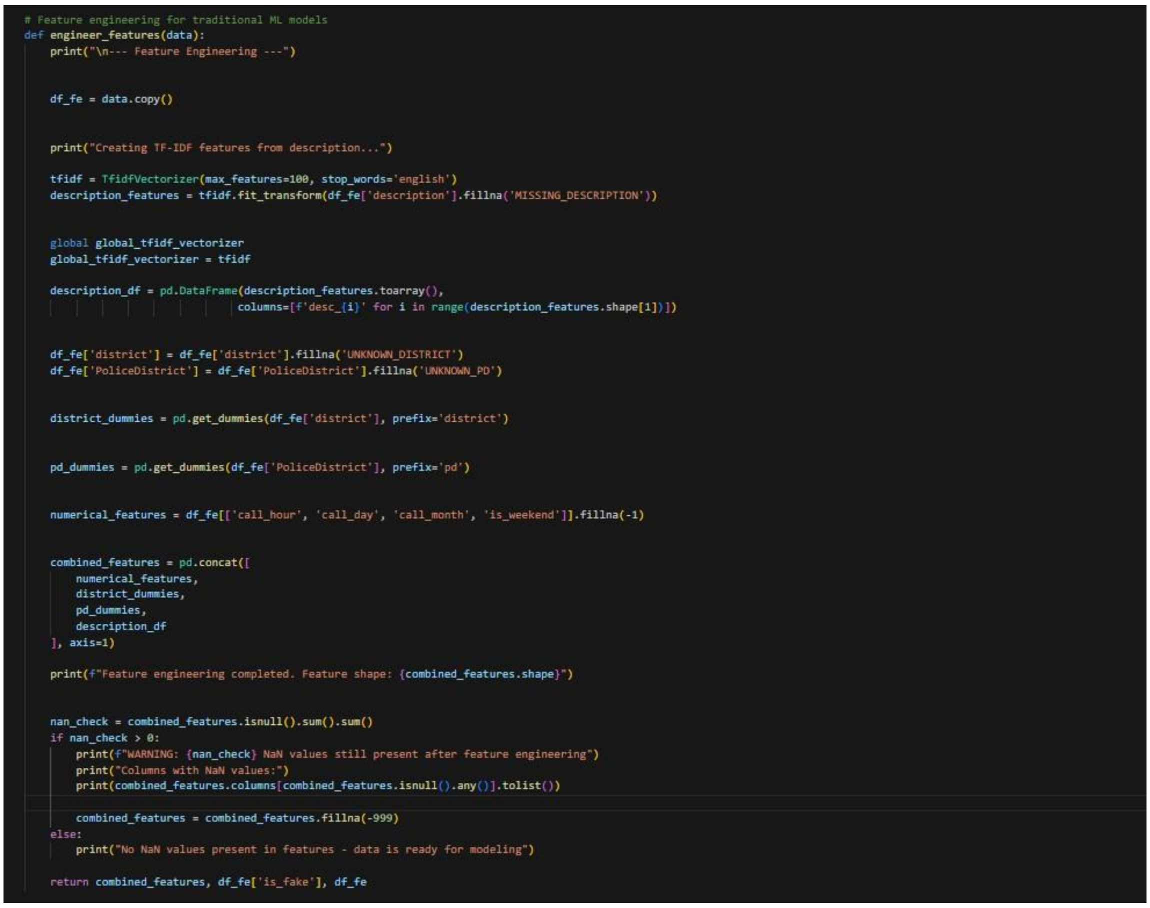


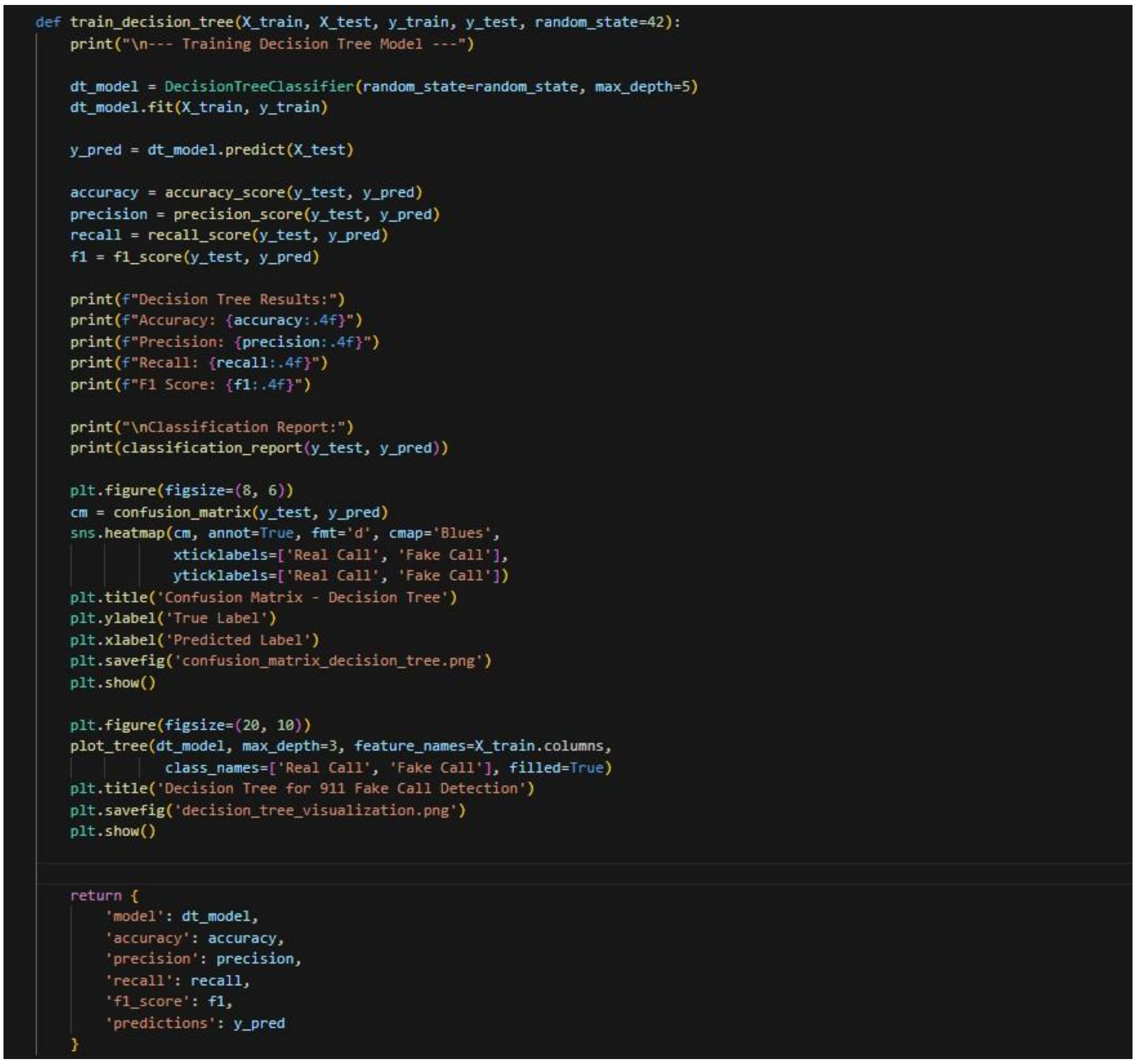
Figure 12: Enter Caption





```python
# Feature engineering for traditional ML models
def engineer_features(data):
    print("\n--- Feature Engineering ---")


    df_fe = data.copy()


    print("Creating TF-IDF features from description...")

    tfidf = TfidfVectorizer(max_features=100, stop_words='english')
    description_features = tfidf.fit_transform(df_fe['description'].fillna('MISSING_DESCRIPTION'))


    global global_tfidf_vectorizer
    global_tfidf_vectorizer = tfidf

    description_df = pd.DataFrame(description_features.toarray(),
                                  columns=[f'desc_{i}' for i in range(description_features.shape[1])])


    df_fe['district'] = df_fe['district'].fillna('UNKNOWN_DISTRICT')
    df_fe['PoliceDistrict'] = df_fe['PoliceDistrict'].fillna('UNKNOWN_PD')


    district_dummies = pd.get_dummies(df_fe['district'], prefix='district')


    pd_dummies = pd.get_dummies(df_fe['PoliceDistrict'], prefix='pd')


    numerical_features = df_fe[['call_hour', 'call_day', 'call_month', 'is_weekend']].fillna(-1)


    combined_features = pd.concat([
        numerical_features,
        district_dummies,
        pd_dummies,
        description_df
    ], axis=1)

    print(f"Feature engineering completed. Feature shape: {combined_features.shape}")


    nan_check = combined_features.isnull().sum().sum()
    if nan_check > 0:
        print(f"WARNING: {nan_check} NaN values still present after feature engineering")
        print("Columns with NaN values:")
        print(combined_features.columns[combined_features.isnull().any()].tolist())

        combined_features = combined_features.fillna(-999)
    else:
        print("No NaN values present in features - data is ready for modeling")

    return combined_features, df_fe['is_fake'], df_fe
```

Figure 13: Enter Caption



```python
def train_decision_tree(X_train, X_test, y_train, y_test, random_state=42):
    print("\n--- Training Decision Tree Model ---")

    dt_model = DecisionTreeClassifier(random_state=random_state, max_depth=5)
    dt_model.fit(X_train, y_train)

    y_pred = dt_model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"Decision Tree Results:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Real Call', 'Fake Call'],
                yticklabels=['Real Call', 'Fake Call'])
    plt.title('Confusion Matrix - Decision Tree')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.savefig('confusion_matrix_decision_tree.png')
    plt.show()

    plt.figure(figsize=(20, 10))
    plot_tree(dt_model, max_depth=3, feature_names=X_train.columns,
              class_names=['Real Call', 'Fake Call'], filled=True)
    plt.title('Decision Tree for 911 Fake Call Detection')
    plt.savefig('decision_tree_visualization.png')
    plt.show()


    return {
        'model': dt_model,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'predictions': y_pred
    }
```

Figure 14: Enter Caption

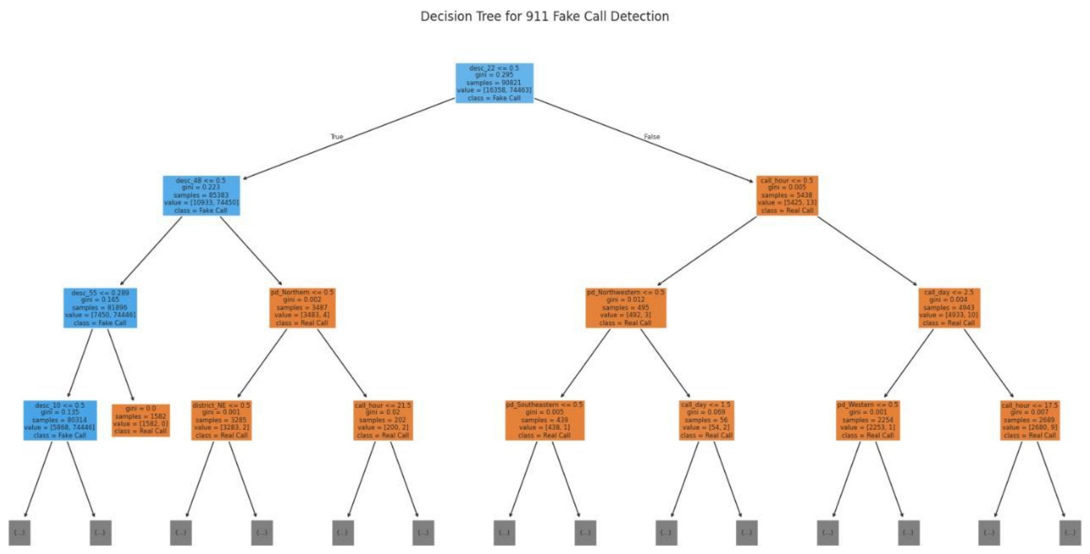
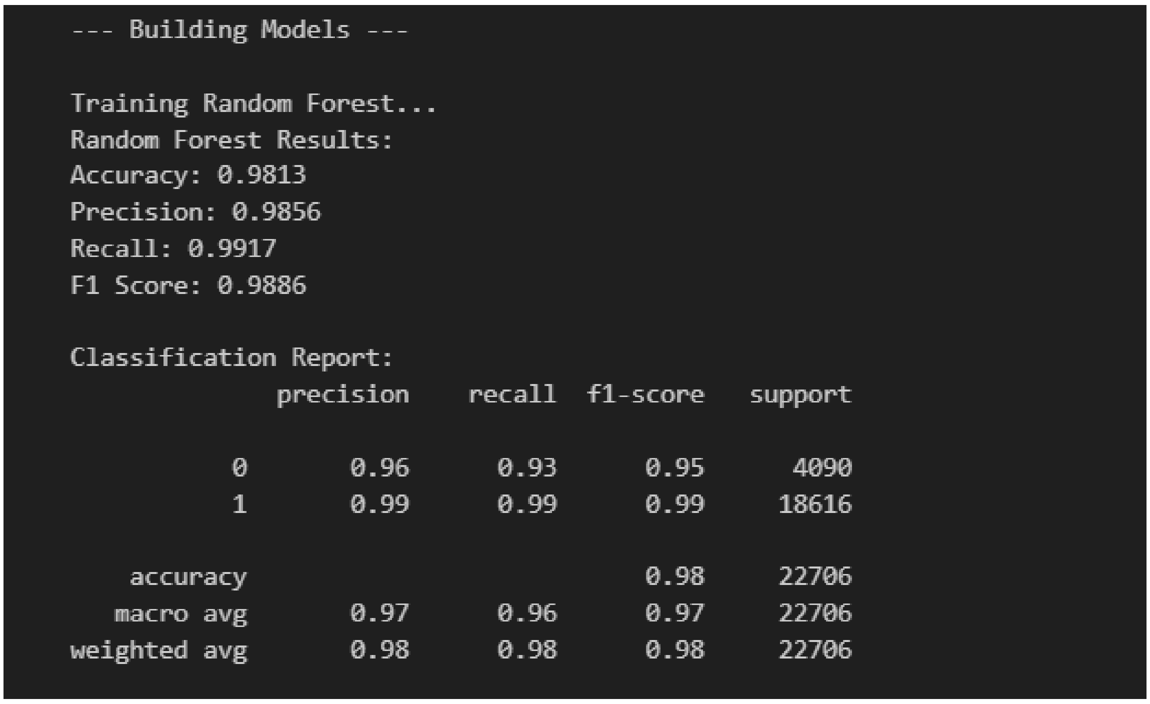

Decision Tree for 911 Fake Call Detection
desc_22 <= 0.5 gini = 0.295 samples = 90821 value = [16358, 74463] class = Fake Call
True
False
desc_48 <= 0.5 gini = 0.223 samples = 85383 value = [10933, 74450] class = Fake Call
call_hour <= 0.5 gini = 0.005 samples = 5438 value = [5425, 13] class = Real Call
desc_55 <= 0.289 gini = 0.165 samples = 81896 value = [7450, 74446] class = Fake Call
pd_Northern <= 0.5 gini = 0.002 samples = 3487 value = [3483, 4] class = Real Call
pd_Northwestern <= 0.5 gini = 0.012 samples = 495 value = [492, 3] class = Real Call
call_day <= 2.5 gini = 0.004 samples = 4943 value = [4933, 10] class = Real Call
desc_10 <= 0.5 gini = 0.135 samples = 80314 value = [5868, 74446] class = Fake Call
gini = 0.0 samples = 1582 value = [1582, 0] class = Real Call
district_NE <= 0.5 gini = 0.001 samples = 3285 value = [3283, 2] class = Real Call
call_hour <= 21.5 gini = 0.02 samples = 202 value = [200, 2] class = Real Call
pd_Southeastern <= 0.5 gini = 0.005 samples = 439 value = [438, 1] class = Real Call
call_day <= 1.5 gini = 0.069 samples = 56 value = [54, 2] class = Real Call
pd_Western <= 0.5 gini = 0.001 samples = 2254 value = [2253, 1] class = Real Call
call_hour <= 17.5 gini = 0.007 samples = 2689 value = [2680, 9] class = Real Call

Figure 15: Enter Caption



```
--- Building Models ---

Training Random Forest...
Random Forest Results:
Accuracy: 0.9813
Precision: 0.9856
Recall: 0.9917
F1 Score: 0.9886

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.93      0.95      4090
           1       0.99      0.99      0.99     18616

    accuracy                           0.98     22706
   macro avg       0.97      0.96      0.97     22706
weighted avg       0.98      0.98      0.98     22706
```

Figure 16: Enter Caption

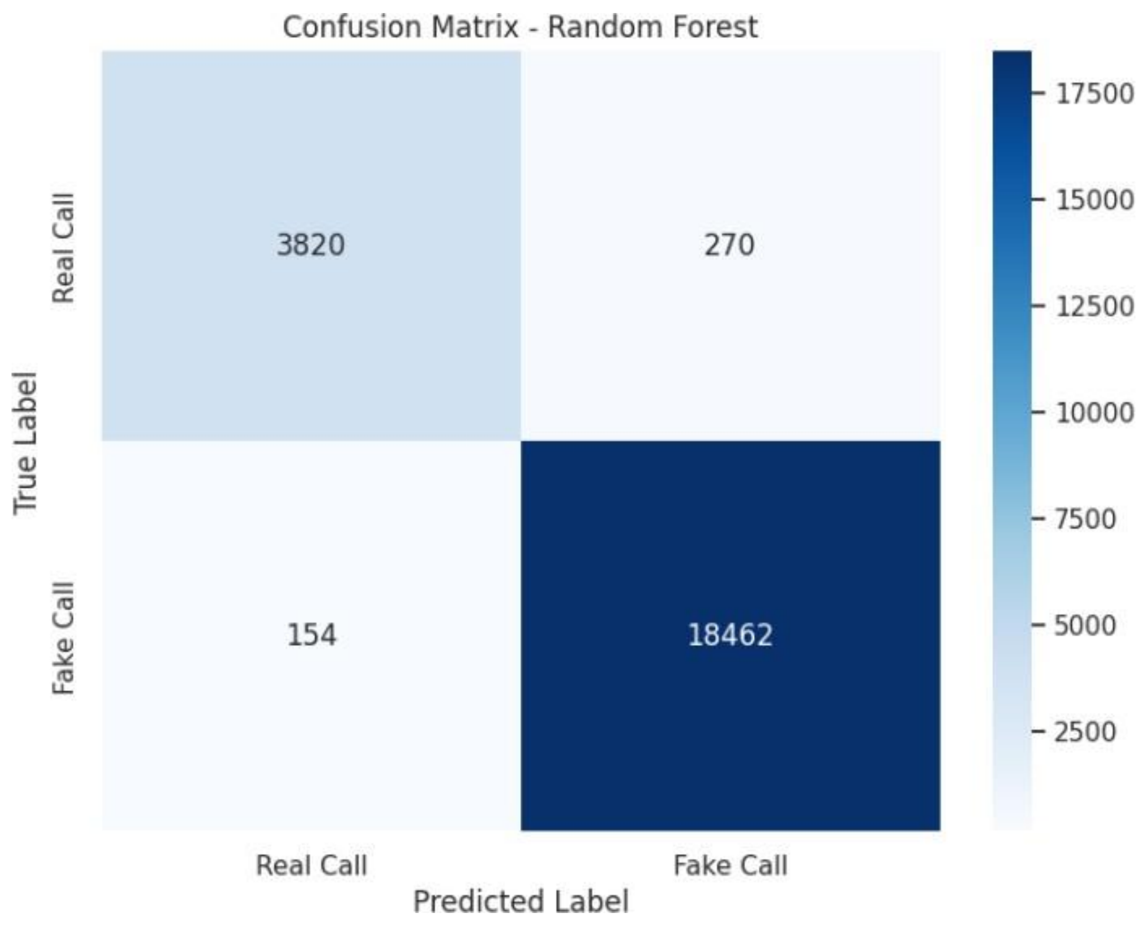
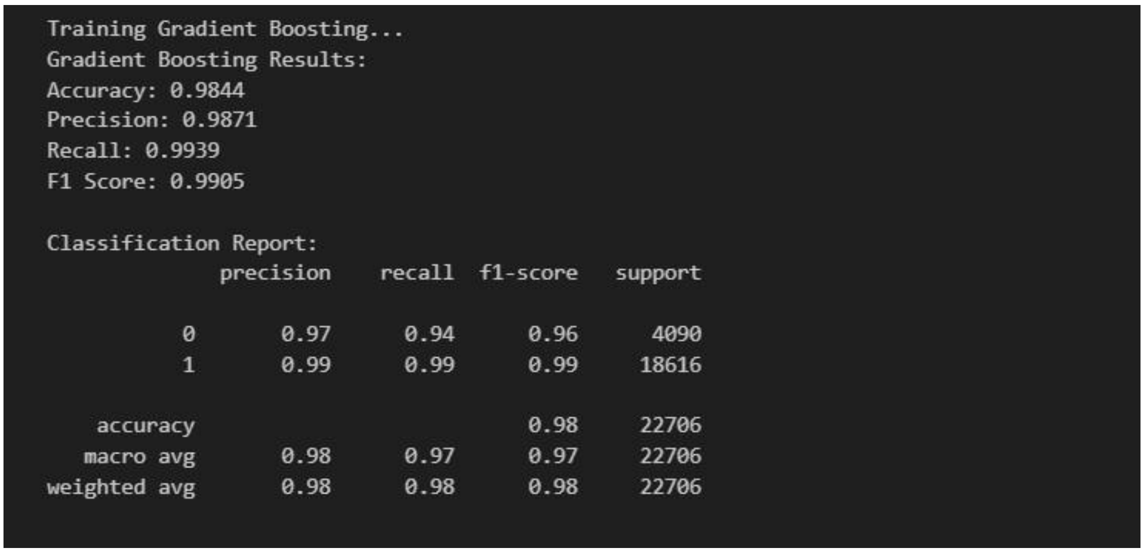


Figure 17: Enter Caption



```
Training Gradient Boosting...
Gradient Boosting Results:
Accuracy: 0.9844
Precision: 0.9871
Recall: 0.9939
F1 Score: 0.9905

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      4090
           1       0.99      0.99      0.99     18616

    accuracy                           0.98     22706
   macro avg       0.98      0.97      0.97     22706
weighted avg       0.98      0.98      0.98     22706
```

Figure 18: Enter Caption

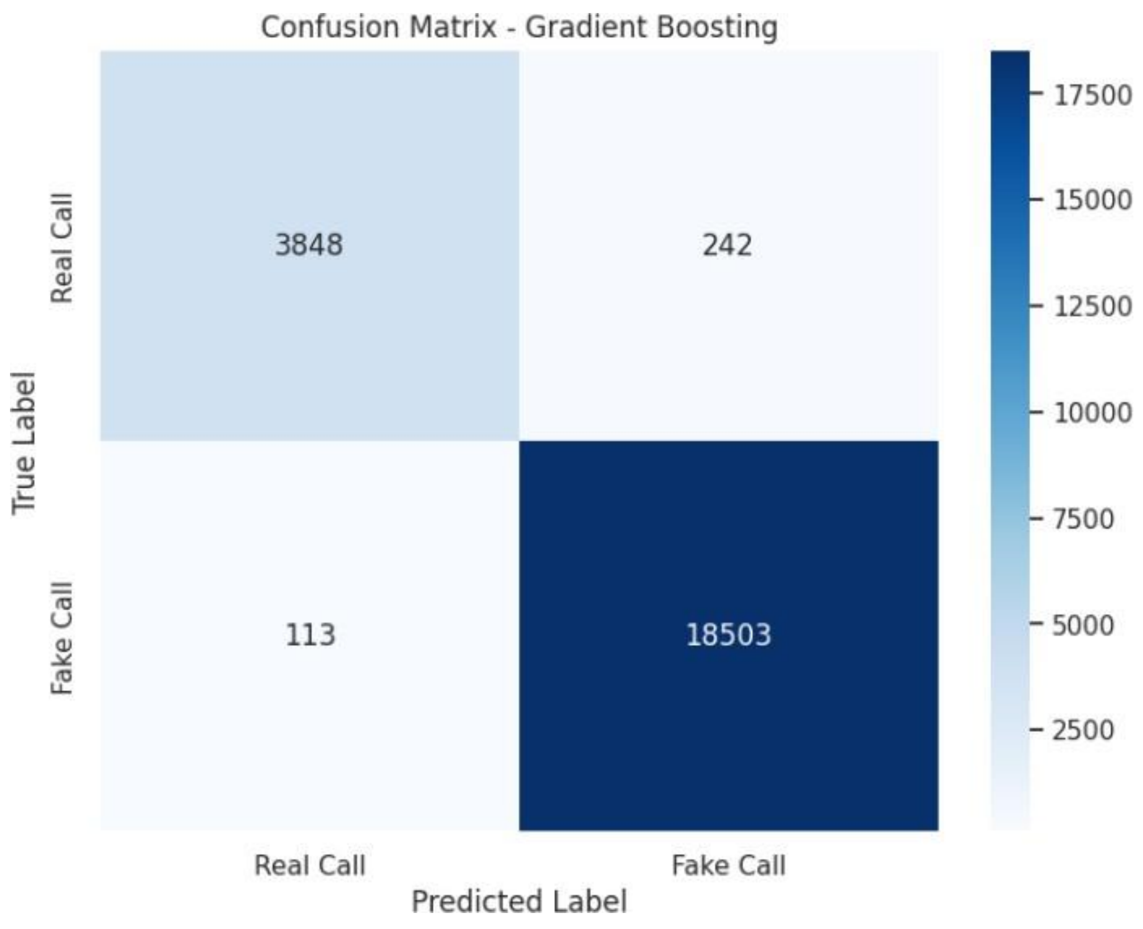
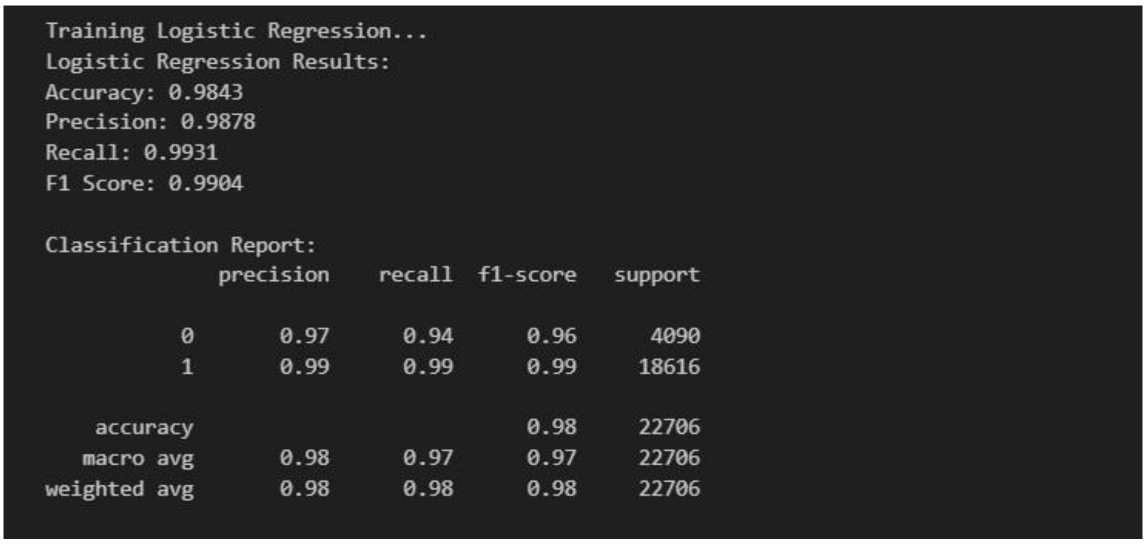


Figure 19: Enter Caption



```
Training Logistic Regression...
Logistic Regression Results:
Accuracy: 0.9843
Precision: 0.9878
Recall: 0.9931
F1 Score: 0.9904

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      4090
           1       0.99      0.99      0.99     18616

    accuracy                           0.98     22706
   macro avg       0.98      0.97      0.97     22706
weighted avg       0.98      0.98      0.98     22706
```

Figure 20: Enter Caption

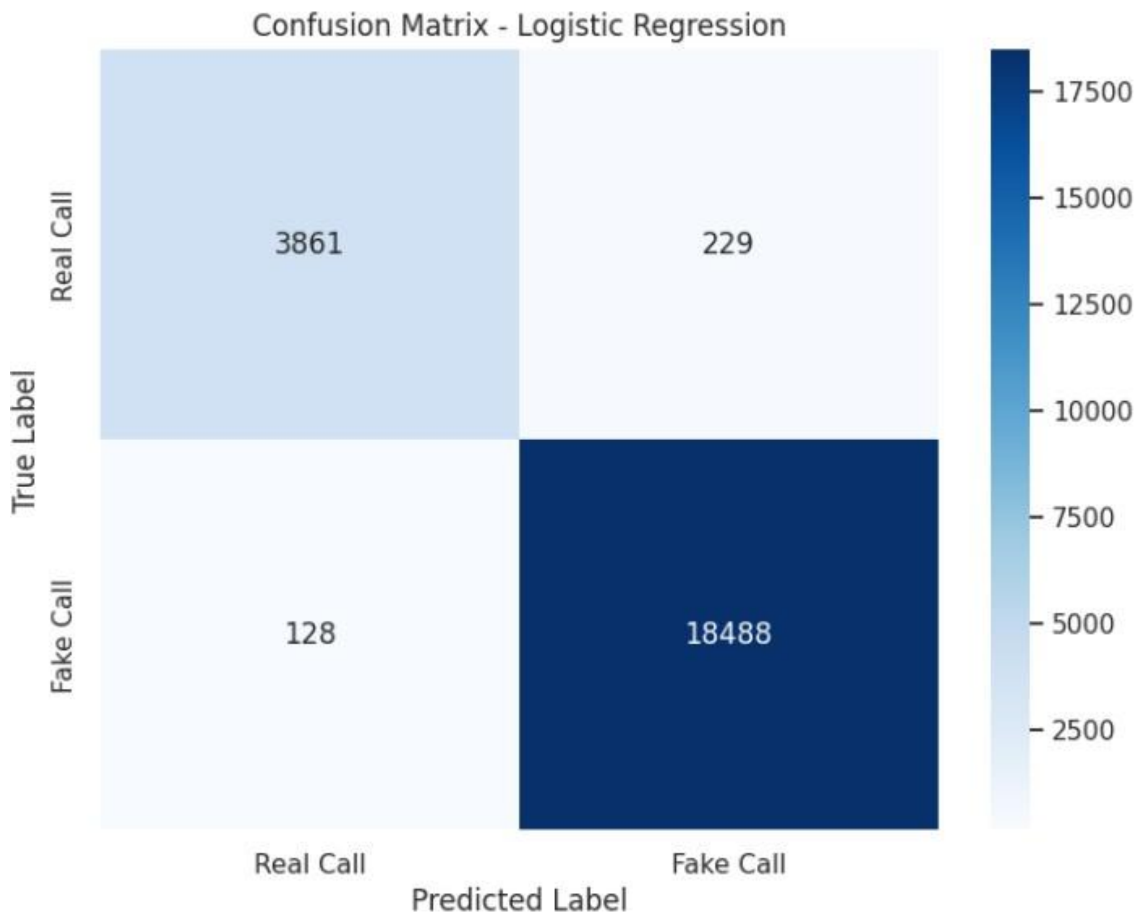
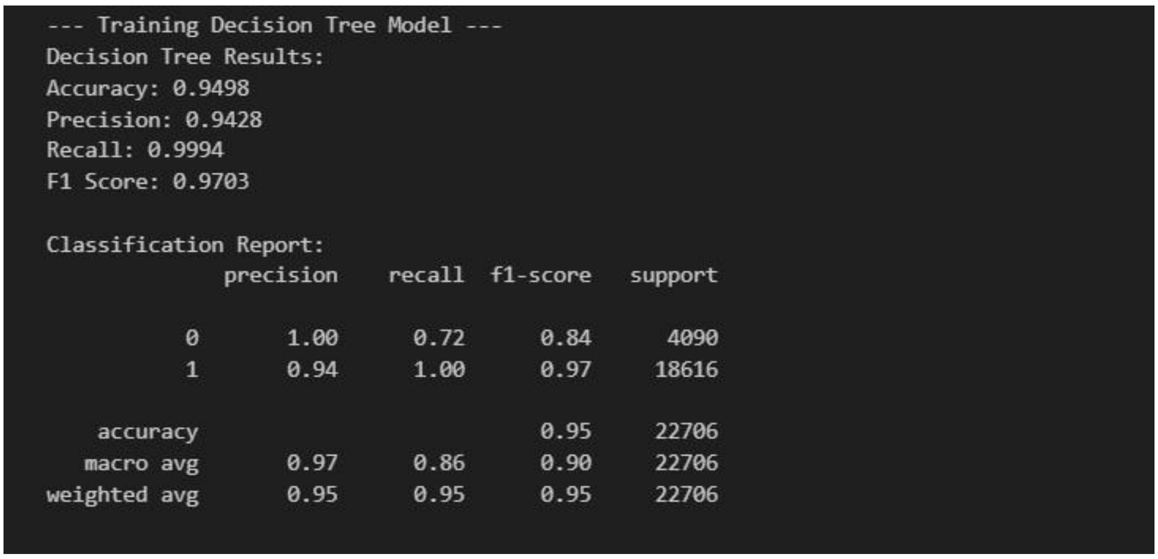


Figure 21: Enter Caption



```
--- Training Decision Tree Model ---
Decision Tree Results:
Accuracy: 0.9498
Precision: 0.9428
Recall: 0.9994
F1 Score: 0.9703

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.72      0.84      4090
           1       0.94      1.00      0.97     18616

    accuracy                           0.95     22706
   macro avg       0.97      0.86      0.90     22706
weighted avg       0.95      0.95      0.95     22706
```

Figure 22: Enter Caption

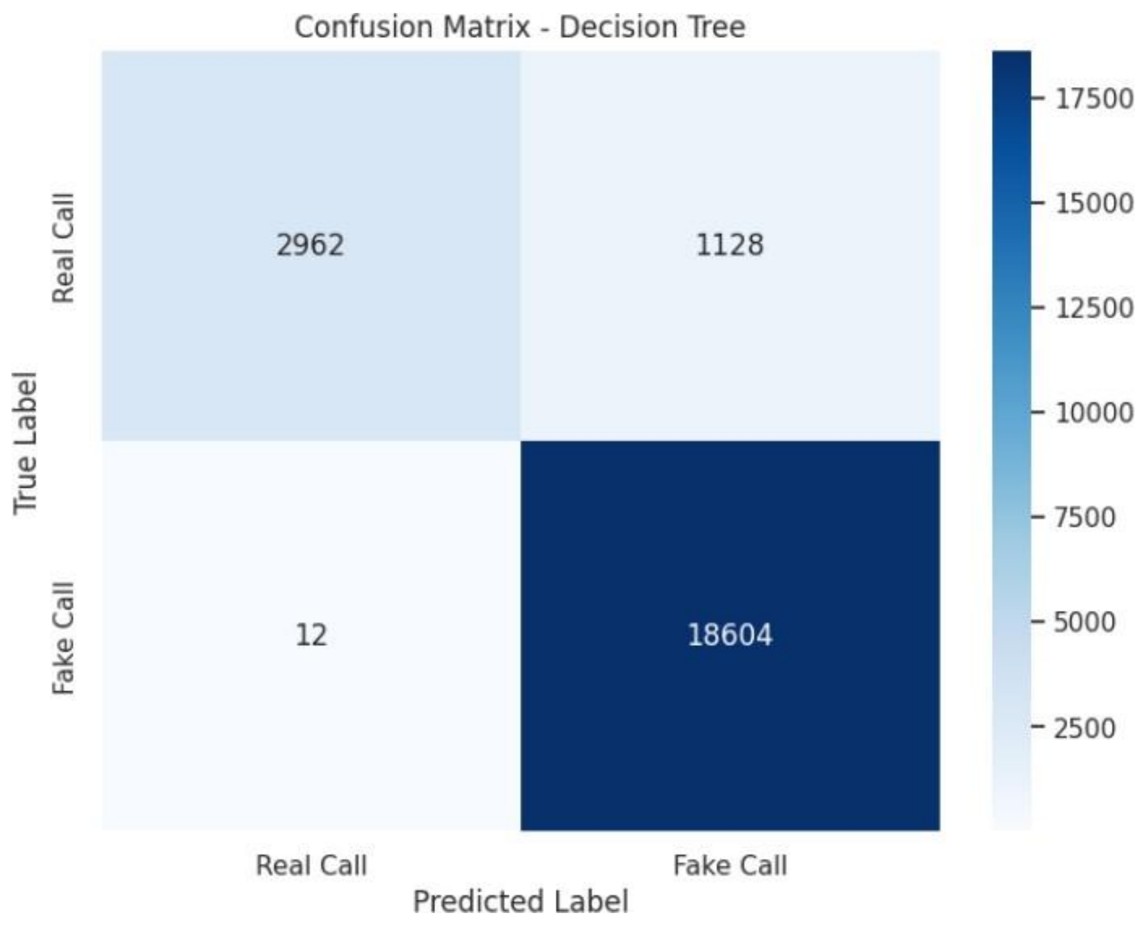
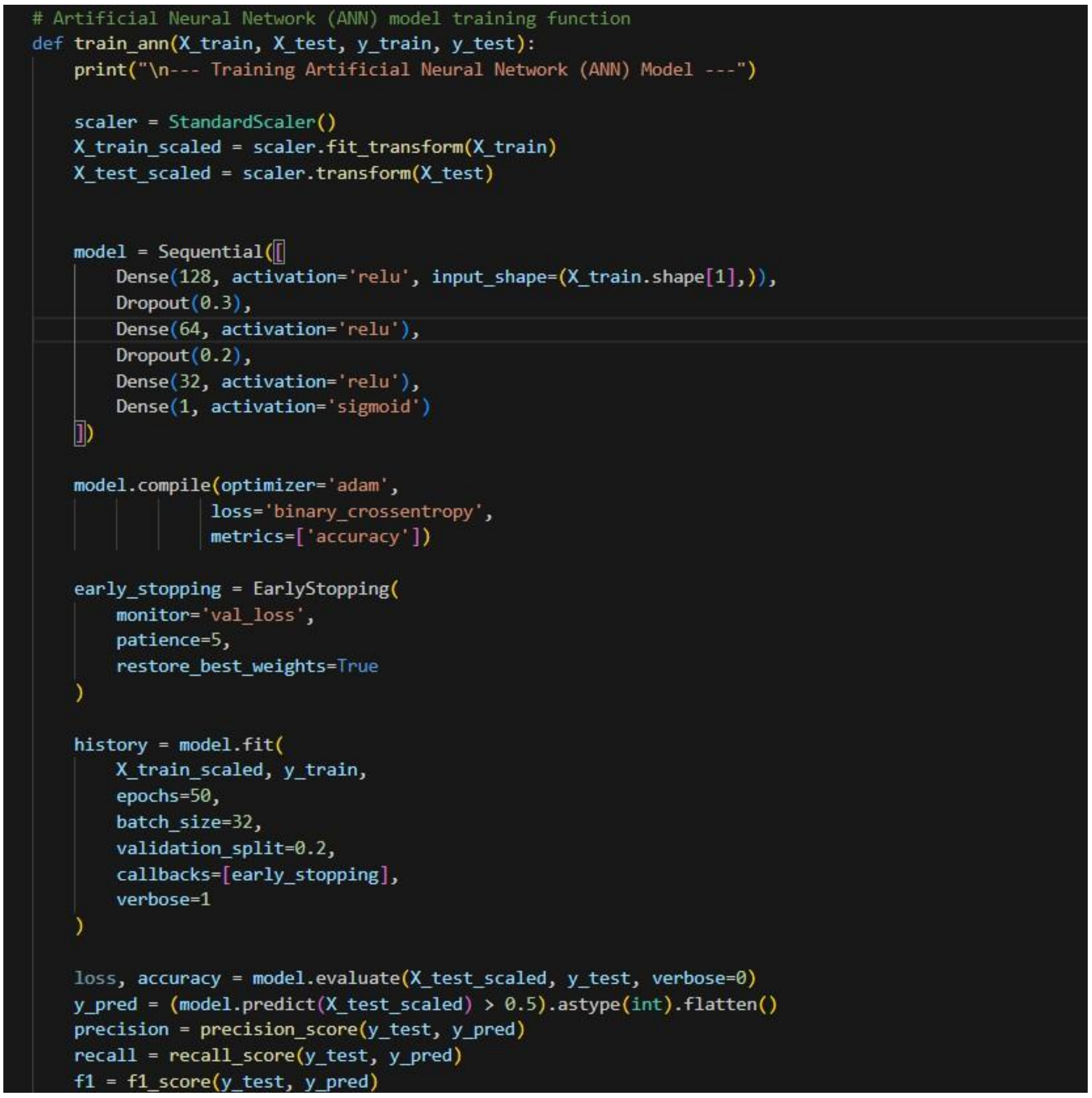
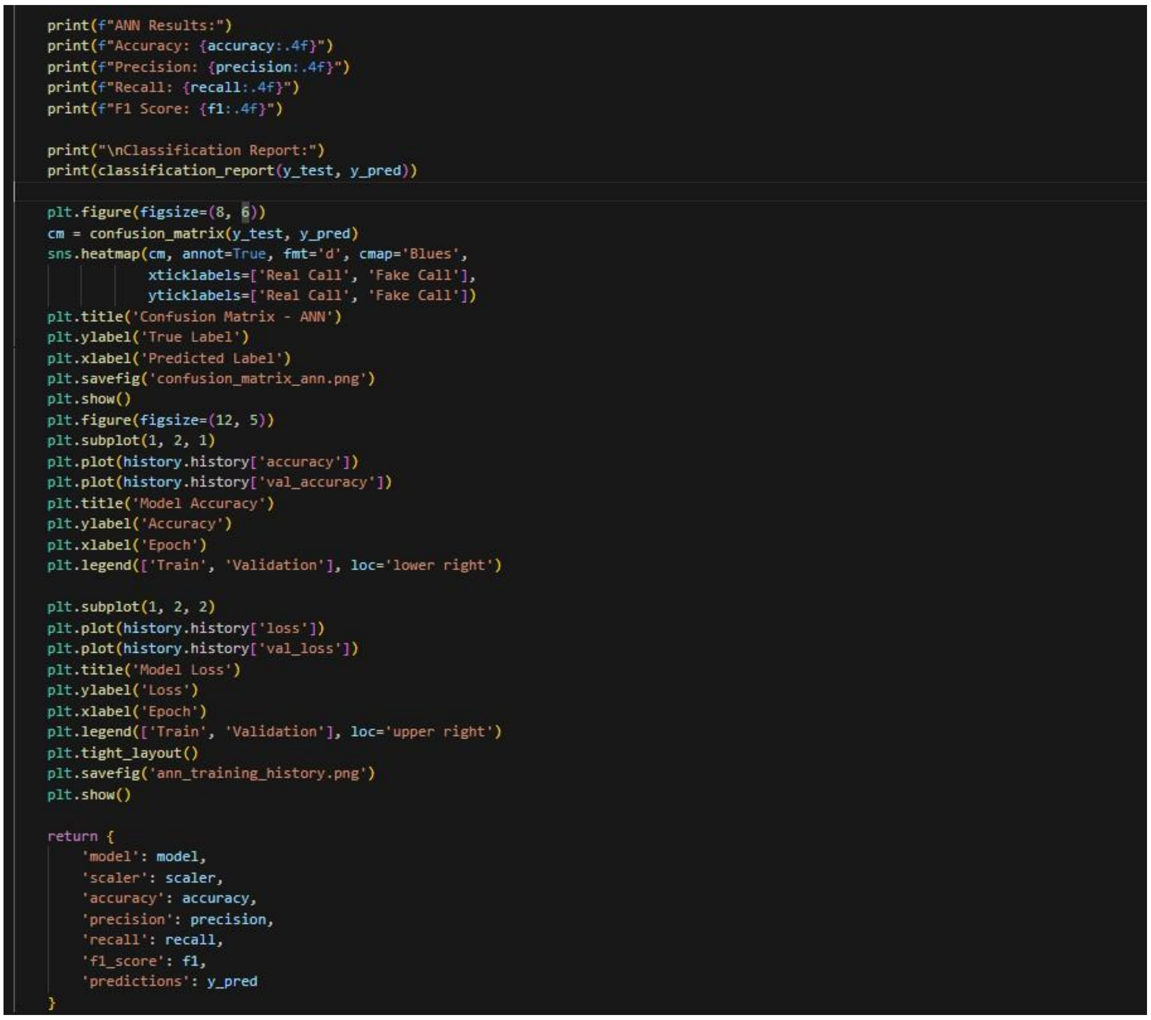


Figure 23: Enter Caption



```python
# Artificial Neural Network (ANN) model training function
def train_ann(X_train, X_test, y_train, y_test):
    print("\n--- Training Artificial Neural Network (ANN) Model ---")

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


    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
    )

    history = model.fit(
        X_train_scaled, y_train,
        epochs=50,
        batch_size=32,
        validation_split=0.2,
        callbacks=[early_stopping],
        verbose=1
    )

    loss, accuracy = model.evaluate(X_test_scaled, y_test, verbose=0)
    y_pred = (model.predict(X_test_scaled) > 0.5).astype(int).flatten()
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

Figure 24: Enter Caption



```python
    print(f"ANN Results:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Real Call', 'Fake Call'],
                yticklabels=['Real Call', 'Fake Call'])
    plt.title('Confusion Matrix - ANN')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.savefig('confusion_matrix_ann.png')
    plt.show()
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='lower right')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper right')
    plt.tight_layout()
    plt.savefig('ann_training_history.png')
    plt.show()

    return {
        'model': model,
        'scaler': scaler,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'predictions': y_pred
    }
```

Figure 25: Enter Caption

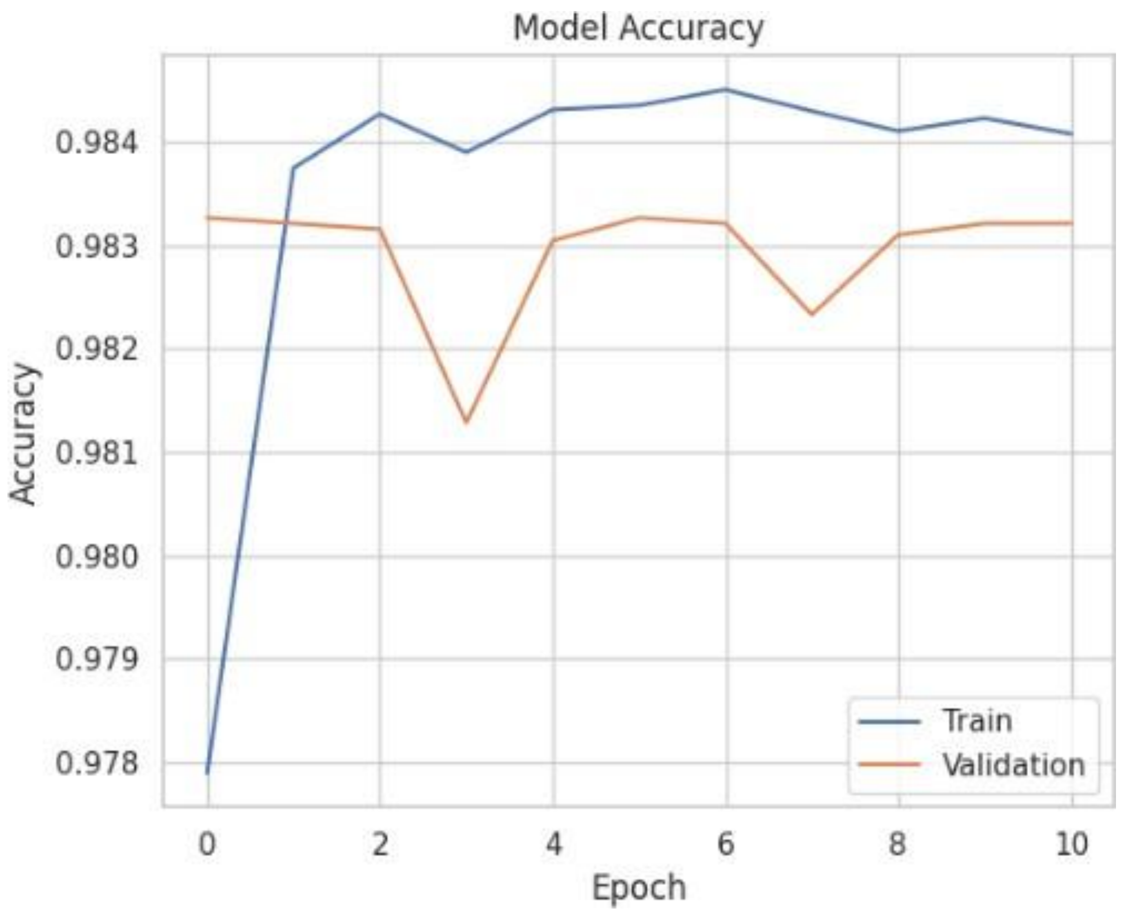


Figure 27: Enter Caption

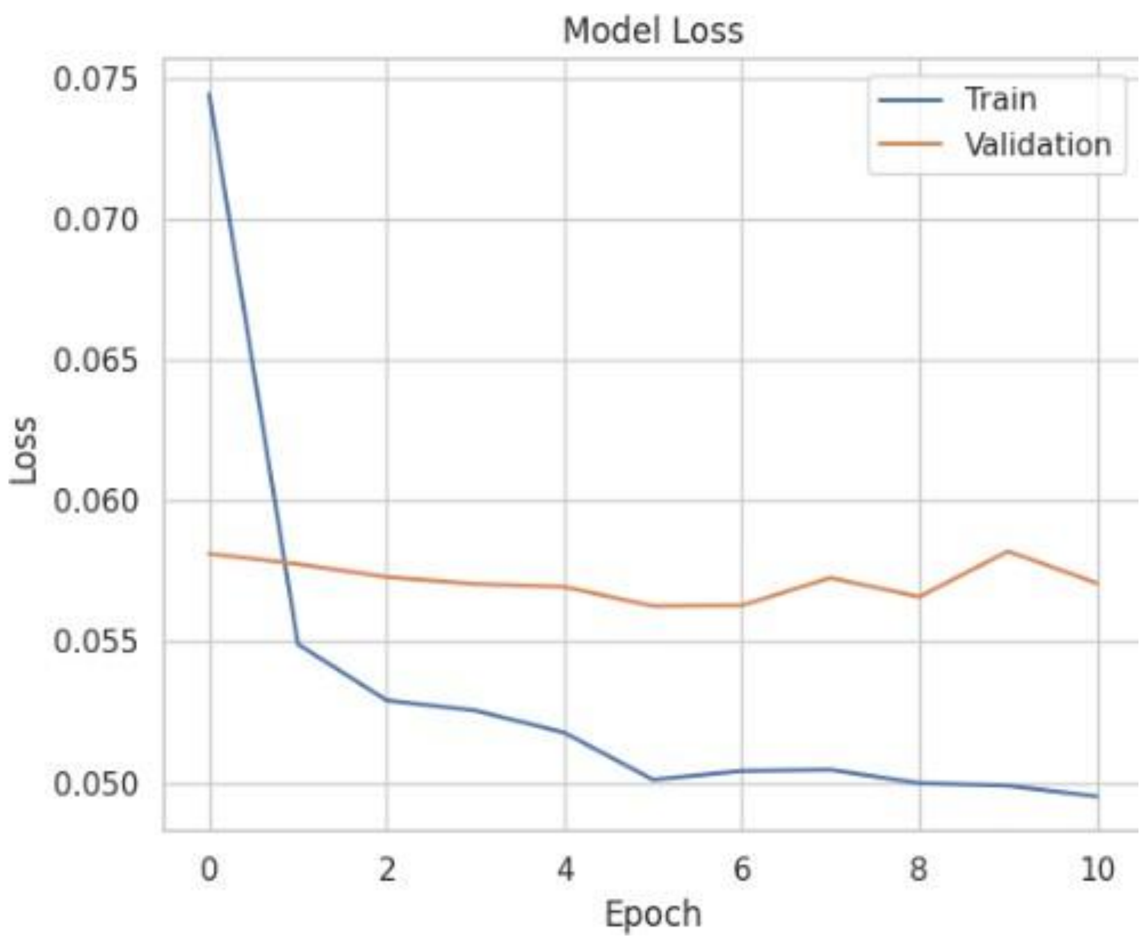


Figure 28: Enter Caption

4. Theory

4.1 Logistic Regression

Logistic Regression is a widely used supervised learning algorithm employed for binary and multiclass classification tasks. Unlike linear regression, which predicts continuous outcomes, logistic regression is specifically designed to estimate the probability of a categorical dependent variable. It is particularly well-suited for classification problems where the output variable is binary, such as determining whether an emergency call is *fake* or *real*.

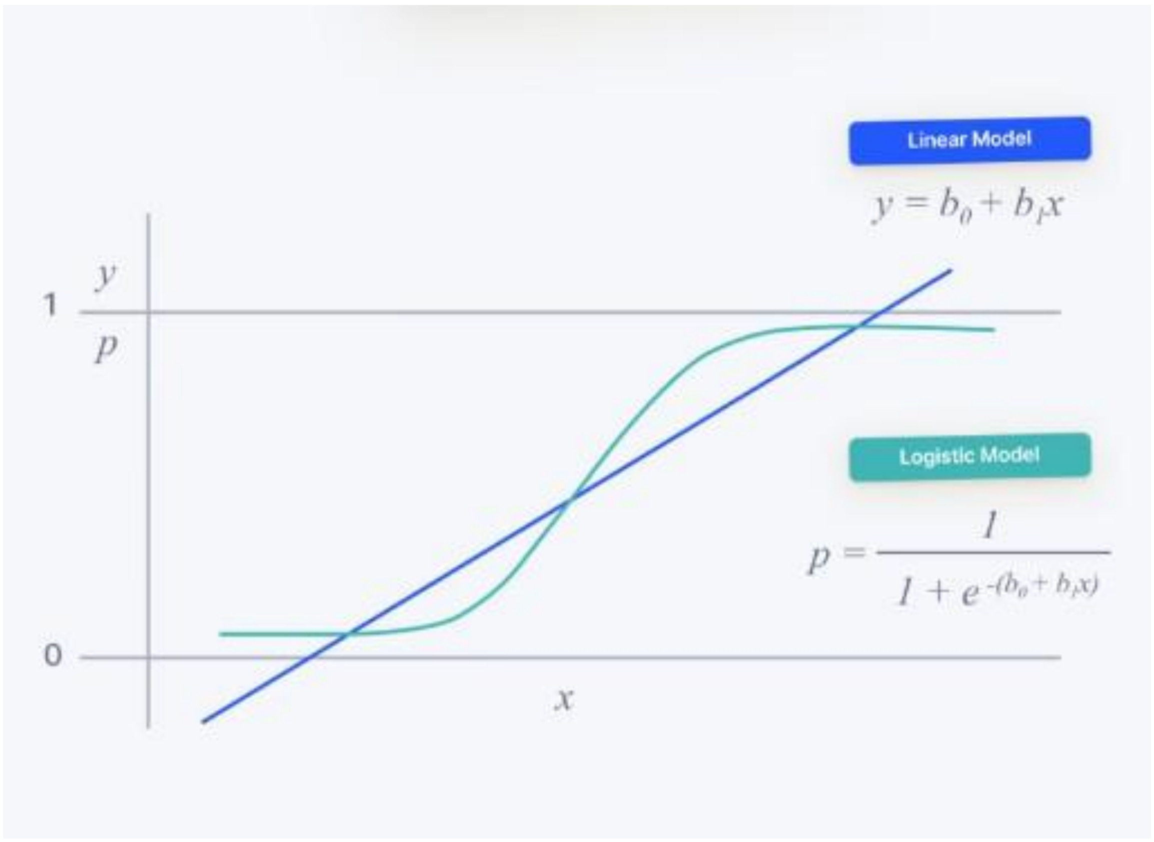


Figure 29: Enter Caption

Mathematical Formulation

Logistic regression models the probability that a given input X belongs to a particular class $Y = 1$ using the logistic (sigmoid) function:

$$P(Y = 1|X) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Here, $x_1, x_2, \ldots, x_n$ represent the input features, β_0 is the bias (intercept) term, and β_i are the model coefficients for each feature. The sigmoid function squashes the output into a range between 0 and 1, representing the probability of the positive class.

A threshold (commonly 0.5) is applied to convert the predicted probability into a class label:

$$\hat{Y} = \begin{cases} 1 & \text{if } P(Y = 1|X) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Training the Model

Logistic regression is trained using the maximum likelihood estimation (MLE) technique, which aims to find the optimal set of parameters $\boldsymbol{\beta}$ that maximize the likelihood of observing the training data. This is often solved using optimization algorithms such as Gradient Descent.

Application in Fake Call Detection

In the context of fake emergency call detection, logistic regression can be used to classify calls based on a combination of numerical and textual features such as:

Time of call (hour, weekday, weekend)

Call description (vectorized using TF-IDF or Bag-of-Words)

Priority level

District or region of origin

The model learns from historical labeled data (real or fake) and predicts the likelihood of a new call being fake, enabling real-time filtering and triaging of emergency responses.

Advantages

Simple and easy to implement

Interpretable model coefficients

Works well for linearly separable data

Computationally efficient

Limitations

Assumes linear relationship between features and log-odds

Not effective for complex, non-linear decision boundaries

Sensitive to outliers

Performance may degrade with imbalanced datasets (e.g., more fake than real calls)

Conclusion

Logistic Regression is a foundational classification algorithm that provides baseline performance for binary tasks like fake call detection. Although it may not capture complex patterns as effectively as advanced models, it offers valuable interpretability and computational efficiency. In a multi-model evaluation pipeline, logistic regression often serves as a benchmark to compare more sophisticated classifiers.

4.2 Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees to build a more robust and accurate classification model. It is based on the principle of “bagging” (Bootstrap Aggregation), where several decision trees are trained on different subsets of the training data and their predictions are aggregated (usually by majority voting) to produce the final output.

Random Forest is particularly effective in handling noisy, high-dimensional data and is known for its resilience against overfitting, making it suitable for real-world applications like fake emergency call detection.

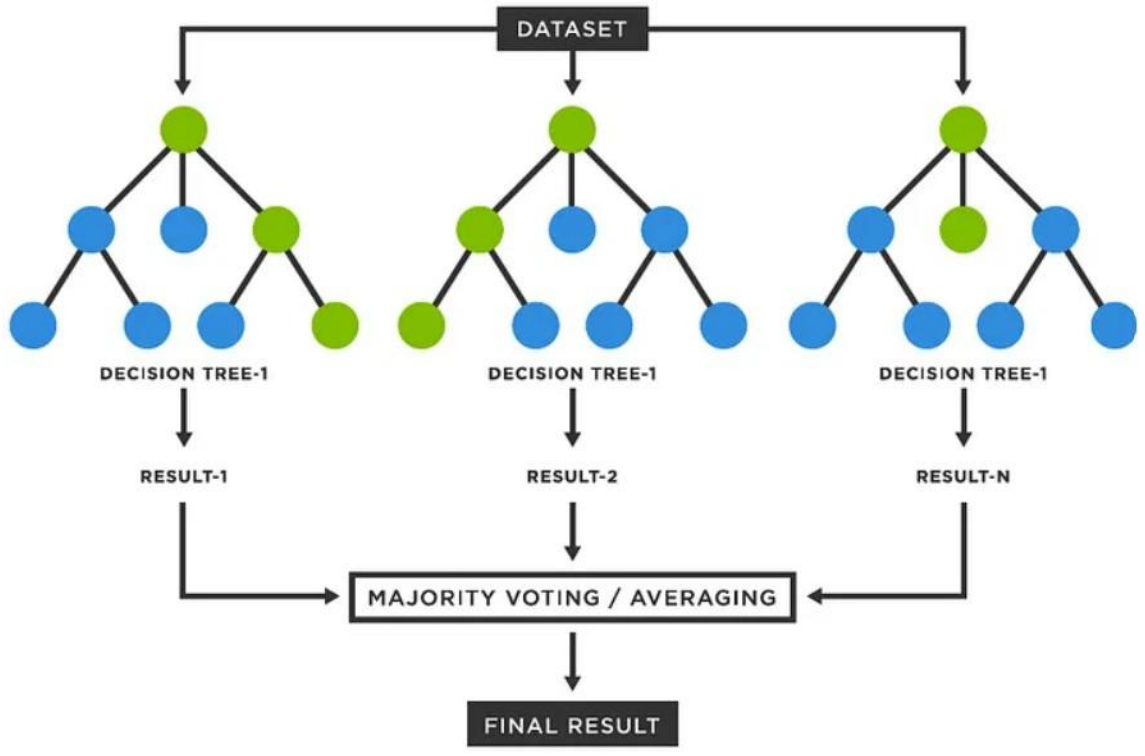


Figure 30: Enter Caption

Concept and Mathematical Idea

The Random Forest algorithm builds N decision trees using randomly sampled subsets of the original dataset with replacement (bootstrap sampling). For each node split in a tree, a random subset of features is considered, increasing diversity among the trees.

The classification output $\hat{y}$ for an input sample x is determined by majority voting among all individual decision trees:

$$\hat{y} = \text{mode}(h_1(x), h_2(x), \ldots, h_N(x))$$

where $h_i(x)$ is the prediction of the i-th decision tree.

Training Procedure

1. Generate N bootstrap samples from the training dataset.
2. For each sample, build a decision tree using a random subset of features at each split.
3. Aggregate the predictions of all trees to produce the final output.

Application in Fake Call Detection

In fake emergency call detection systems, Random Forest is highly effective due to its ability to handle:

High-dimensional data (e.g., TF-IDF text vectors from call descriptions)

Mixed data types (e.g., categorical fields like district and numerical fields like time)

Feature interactions and non-linear relationships

By learning patterns from labeled data (e.g., calls tagged as real or fake), the model can accurately classify incoming calls and flag potential fraudulent activity.

Advantages

High accuracy and generalization performance

Robust to noise and overfitting

Handles both numerical and categorical features

Provides feature importance scores for interpretability

Conclusion

Random Forest is a powerful and versatile classification algorithm well-suited for fake emergency call detection. Its ensemble approach ensures stability and accuracy across a wide range of data conditions. As part of a machine learning pipeline, Random Forest serves as a reliable model for benchmarking and deployment in production-scale emergency response systems.

4.3 HistGradient Boosting Classifier

HistGradientBoostingClassifier is an efficient implementation of the Gradient Boosting algorithm, optimized for performance on large tabular datasets. It is based on the concept of additive boosting, where decision trees are trained sequentially, and each new tree corrects the errors made by the previous ensemble. This model is particularly suitable for structured classification tasks, such as detecting fake emergency calls, where data consists of a mix of categorical, textual, and numerical features.

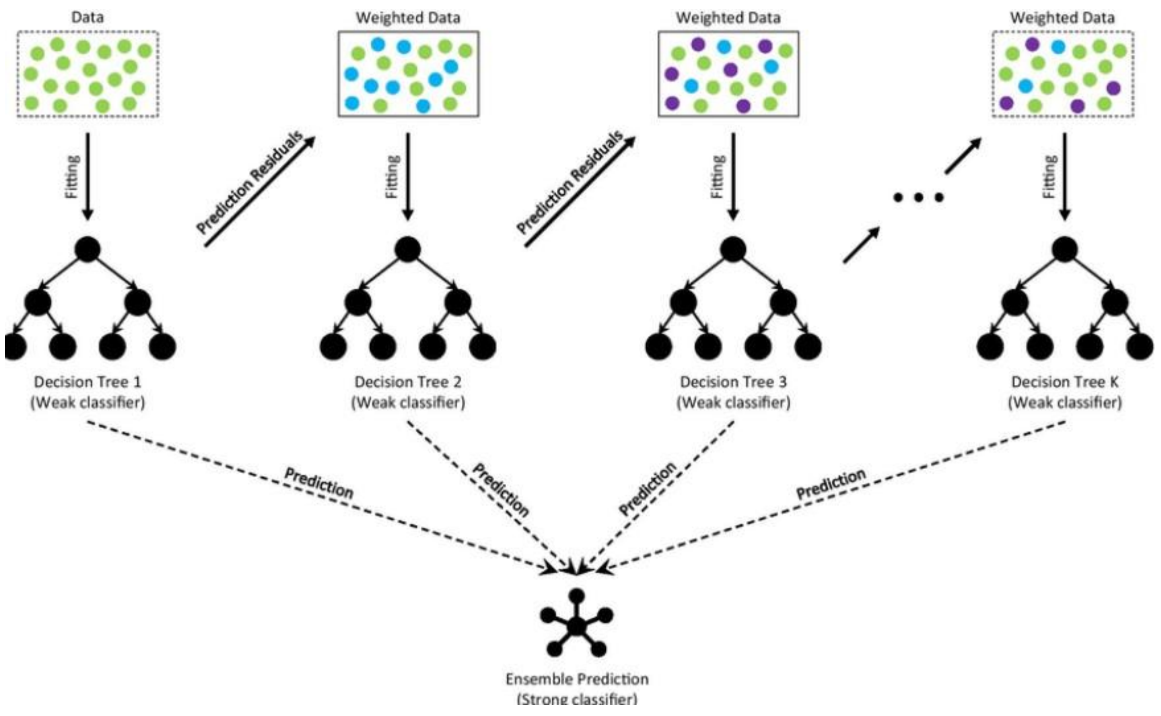


Figure 31: Enter Caption

Concept of Gradient Boosting

Gradient Boosting builds an ensemble model in a stage-wise fashion. Instead of fitting all trees independently (as in Random Forest), each tree is trained to minimize the residual error (i.e., gradient) of the ensemble prediction up to that point.

The output of the final model is given by:

$$\hat{y}(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

where:

M is the number of boosting rounds (trees),

$h_m(x)$ is the m-th decision tree,

γ_m is the learning rate controlling contribution of each tree.

At each stage, the new tree $h_m(x)$ is trained to approximate the negative gradient (error) of the loss function with respect to the model's predictions.

Histogram-Based Optimization

HistGradientBoostingClassifier improves standard Gradient Boosting by binning continuous input features into discrete histograms. This technique speeds up training and reduces memory usage, while maintaining competitive accuracy.

Key advantages of histogram-based implementation include:

Faster training with large datasets

Native support for missing values

Scalability to high-dimensional feature sets (e.g., TF-IDF vectors)

Application in Fake Call Detection

In this project, HistGradientBoostingClassifier was used to detect fake emergency calls based on a structured dataset including:

Time-based features (hour, weekday, month)

Location data (district, police jurisdiction)

Textual call descriptions (vectorized)

Priority levels and call types

The classifier provided the highest F1-score among all models, indicating its ability to balance precision and recall effectively—crucial in emergency scenarios where false positives and false negatives carry significant risks.

Advantages

High predictive accuracy

Efficient training via histogram binning

Handles missing data natively

Works well on both numerical and categorical features

Limitations

Requires careful tuning of hyperparameters (e.g., learning rate, max depth)

Less interpretable than simpler models

Can overfit if not properly regularized

Conclusion

HistGradientBoostingClassifier is a powerful, scalable machine learning algorithm particularly suited for high-dimensional and structured data, as found in fake emergency call detection systems. Its strong performance, efficiency, and robustness make it an ideal choice for real-time classification and deployment in emergency response applications.

5. Evaluation Metrics: Precision, Recall, and F1-Score

To evaluate the performance of classification models in the context of fake call detection, several statistical metrics are employed. Among them, **Precision**, **Recall**, and **F1-Score** are particularly significant, especially when dealing with imbalanced datasets where the number of fake calls may significantly outnumber or be outnumbered by genuine calls.

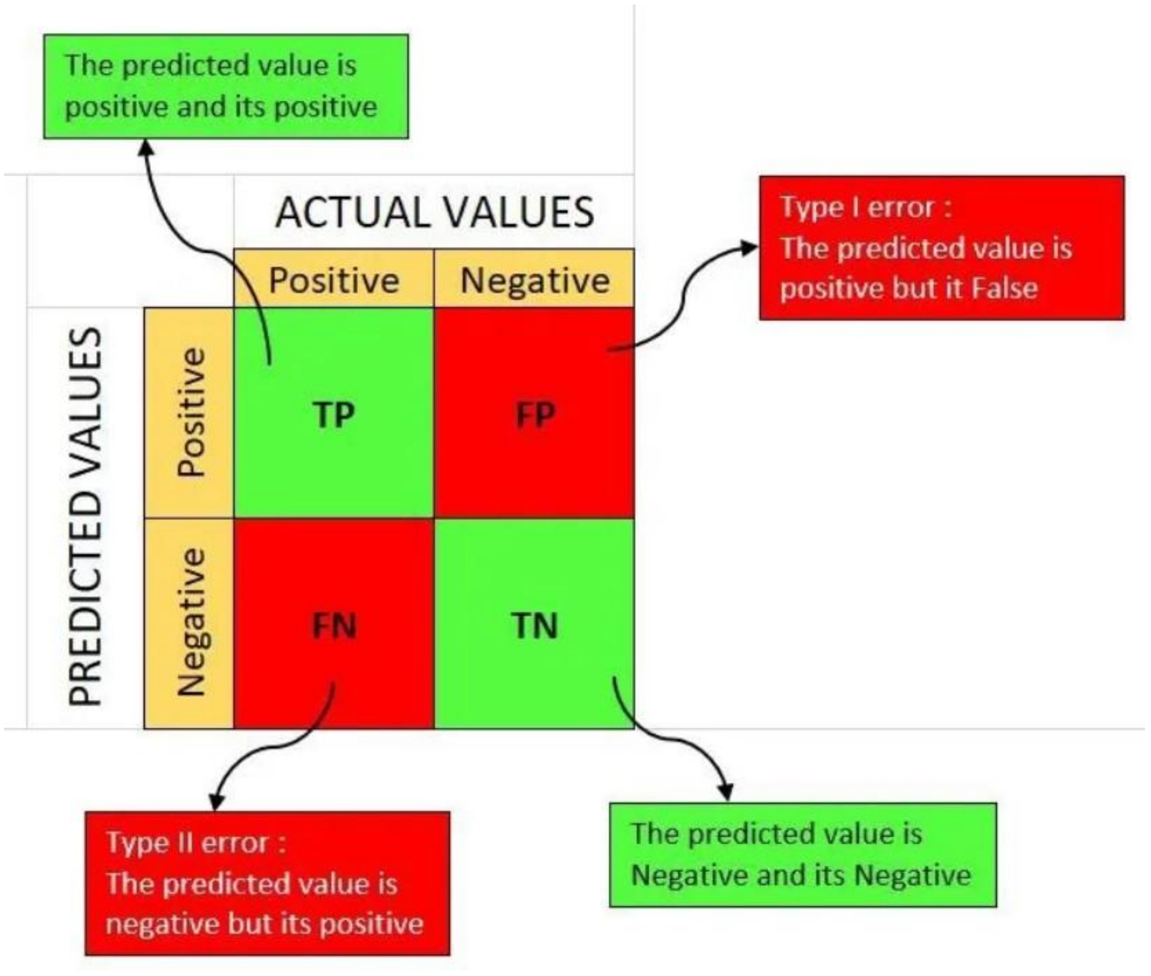

The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
Type I error : The predicted value is positive but it False
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative

Figure 32: Enter Caption

5.1 Precision

Precision measures the proportion of positive identifications that were actually correct. In the context of fake call detection, it represents the percentage of calls identified as fake that were truly fake. It is given by the formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where:

TP = True Positives (correctly predicted fake calls)

FP = False Positives (genuine calls incorrectly predicted as fake)

High precision indicates a low false positive rate, meaning fewer genuine calls are wrongly flagged as fake.

5.2 Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive cases that were correctly identified by the model. For fake call detection, it quantifies the ability of the model to correctly detect fake calls:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where:

FN = False Negatives (fake calls incorrectly predicted as genuine)

A high recall value means that the model is able to detect most of the actual fake calls, which is critical in safety-sensitive applications like ERSS.

5.3 F1-Score

The F1-Score is the harmonic mean of precision and recall. It provides a balanced measure when precision and recall are both important. The F1-Score is particularly useful when there is a trade-off between minimizing false positives and false negatives:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-Score reaches its best value at 1 (perfect precision and recall) and worst at 0. In fake call detection, a high F1-Score indicates a well-balanced model that performs reliably across different types of classification errors.

Conclusion

In the context of emergency response systems, where false negatives (undetected fake calls) can lead to wasted resources and false positives (real calls marked as fake) can risk human lives, using all three metrics together provides a holistic view of model performance. Therefore, evaluating the classifiers using Precision, Recall, and F1-Score ensures more reliable deployment of AI systems in real-time emergency call filtering.

6. Conclusion and Summary

The increasing misuse of emergency communication channels through fake or fraudulent calls poses a significant threat to public safety and the efficiency of emergency response systems. This research focused on developing a machine learning-based framework to detect fake emergency calls within an Emergency Response System (ERSS). By analyzing structured features such as time, location, call priority, and unstructured call descriptions, the system leverages AI to identify deceptive patterns and filter out non-genuine calls. Models such as Logistic Regression, Random Forest, and HistGradientBoostingClassifier were implemented and evaluated to determine the most effective classification approach.

Among the tested models, the HistGradientBoostingClassifier achieved the highest F1-score, demonstrating superior balance between precision and recall. The integration of natural language processing (NLP) techniques to analyze call descriptions enhanced the system's capability to understand linguistic patterns associated with fake calls. Furthermore, techniques like feature engineering, one-hot encoding, and temporal analysis significantly improved model performance. This comprehensive pipeline ensures real-time classification of calls, enabling emergency operators to prioritize legitimate emergencies and respond more efficiently.

In conclusion, the proposed fake call detection system represents a critical advancement in safeguarding emergency communication networks. It not only optimizes resource allocation during emergencies but also reinforces public trust in emergency services. Future work may focus on expanding the dataset, incorporating real-time voice analysis, and deploying the system in live environments with user feedback. Continued research in this domain will be instrumental in fortifying the resilience of ERSS platforms against evolving fraudulent behaviors.