AIT-511 Machine Learning
Project 2 Report

Smoker Status Prediction & Forest Cover Type Classification

M Vinay (MT2025068)
Chavala Lakshmi Vishnu (MT2025033)

Contents

Chapter 1

Smoker Status Prediction (Binary Classification)

1.1 Dataset Details

The Smoker Status dataset from Kaggle contains physiological and biochemical health indicators of individuals, along with a binary label indicating smoking status. The objective is to classify individuals as smokers or non-smokers based on routine medical examination data.

Dataset Characteristics

- **Total Samples:** 38,984
- **Features:** 23 (health measurements)
- **Target Variable:** `smoking` (0 = Non-Smoker, 1 = Smoker)
- **Feature Types:**
  - Continuous: blood pressure, cholesterol, triglycerides, liver enzymes, eyesight, etc.
  - Binary indicators: hearing, urine protein, dental caries

Objective

To accurately classify individuals as smokers or non-smokers based on health measurements using machine-learning models.

Real-World Relevance

- Predicting smoking habits is important for insurance risk modeling.
- Helps in preventive healthcare and identifying risk patterns.
- Useful for public health analysis without requiring explicit self-reporting.

1.2 Preprocessing Steps

1.2.1 Missing Value Analysis

No missing values were found in the dataset:

```
df.isnull().sum()
```

Thus, no imputation was necessary.

1.2.2 Feature Engineering

We added a Body Mass Index (BMI) feature:

$$BMI = \frac{\text{weight (kg)}}{(\text{height (m)})^2}$$

1.2.3 Train–Test Split

The dataset was divided as follows:

- 80% training set
- 20% testing set
- Stratification ensures equal smoker/non-smoker balance

1.2.4 Scaling

Models that depend on distance-based calculations (SVM, Logistic Regression, MLP) use **StandardScaler**. Scaling is done inside pipelines to avoid data leakage.

1.3 Exploratory Data Analysis (EDA)

A detailed EDA was performed to understand feature distributions, relationships, and patterns correlated with smoking status.

1.3.1 Class Distribution

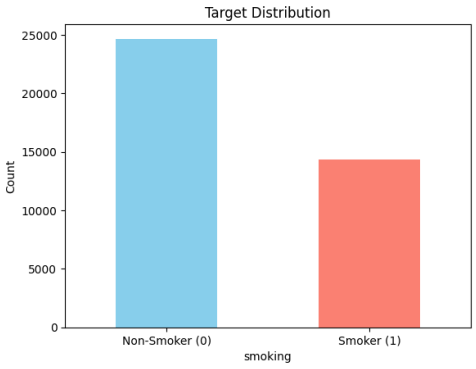

Figure 1.1: Smoker vs Non-Smoker Count Distribution

Observation: Non-smokers dominate, but smokers also form a substantial portion, making this a moderately balanced dataset.

1.3.2 Correlation Heatmap

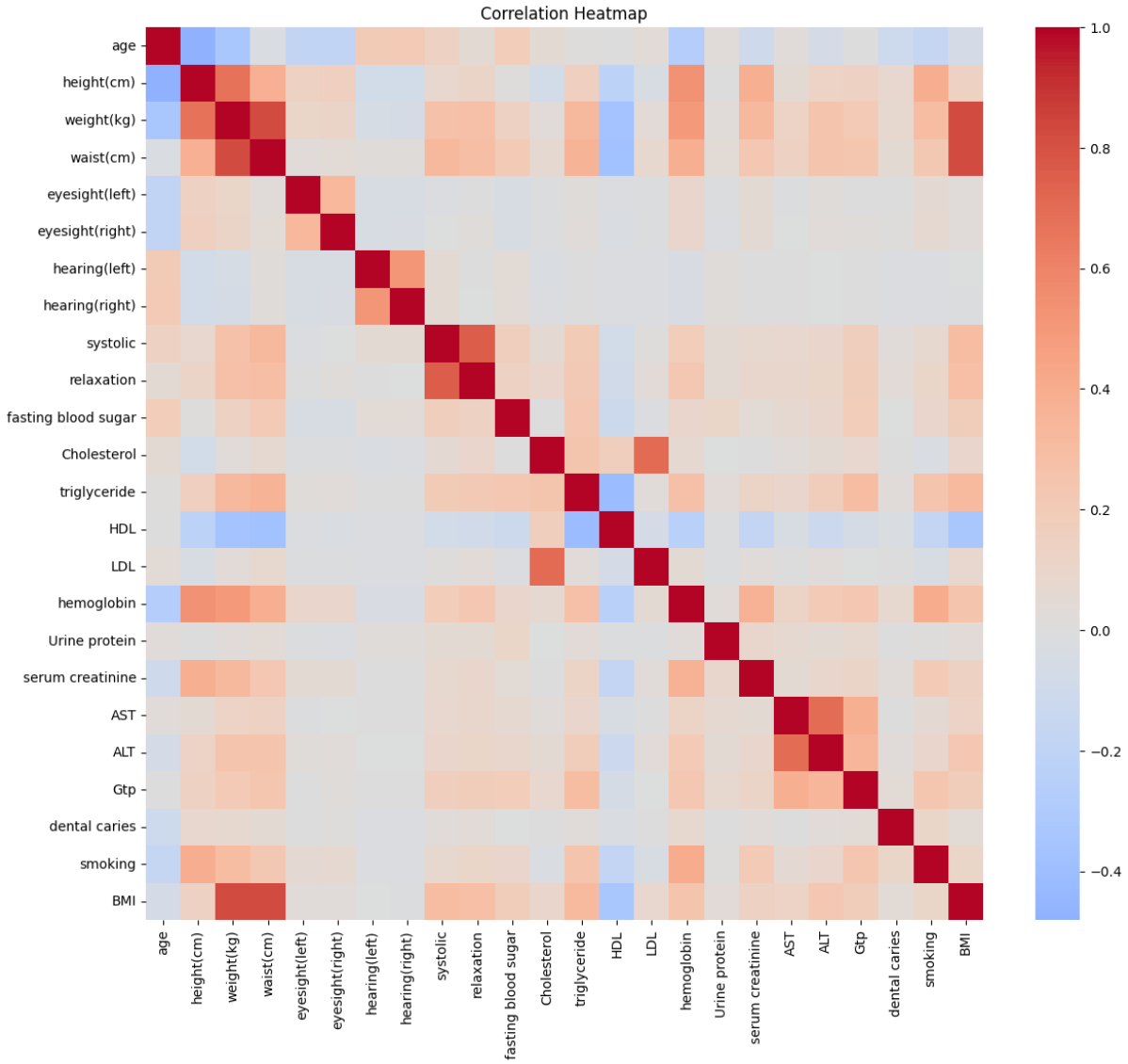

Figure 1.2: Correlation Heatmap of Health Features

Insights:

- Liver enzymes (AST, ALT, GTP) are positively correlated.
- Cholesterol, triglycerides, HDL, LDL show expected biochemical relationships.
- Systolic and diastolic blood pressure exhibit strong correlation.

1.3.3 Histograms of Major Health Indicators

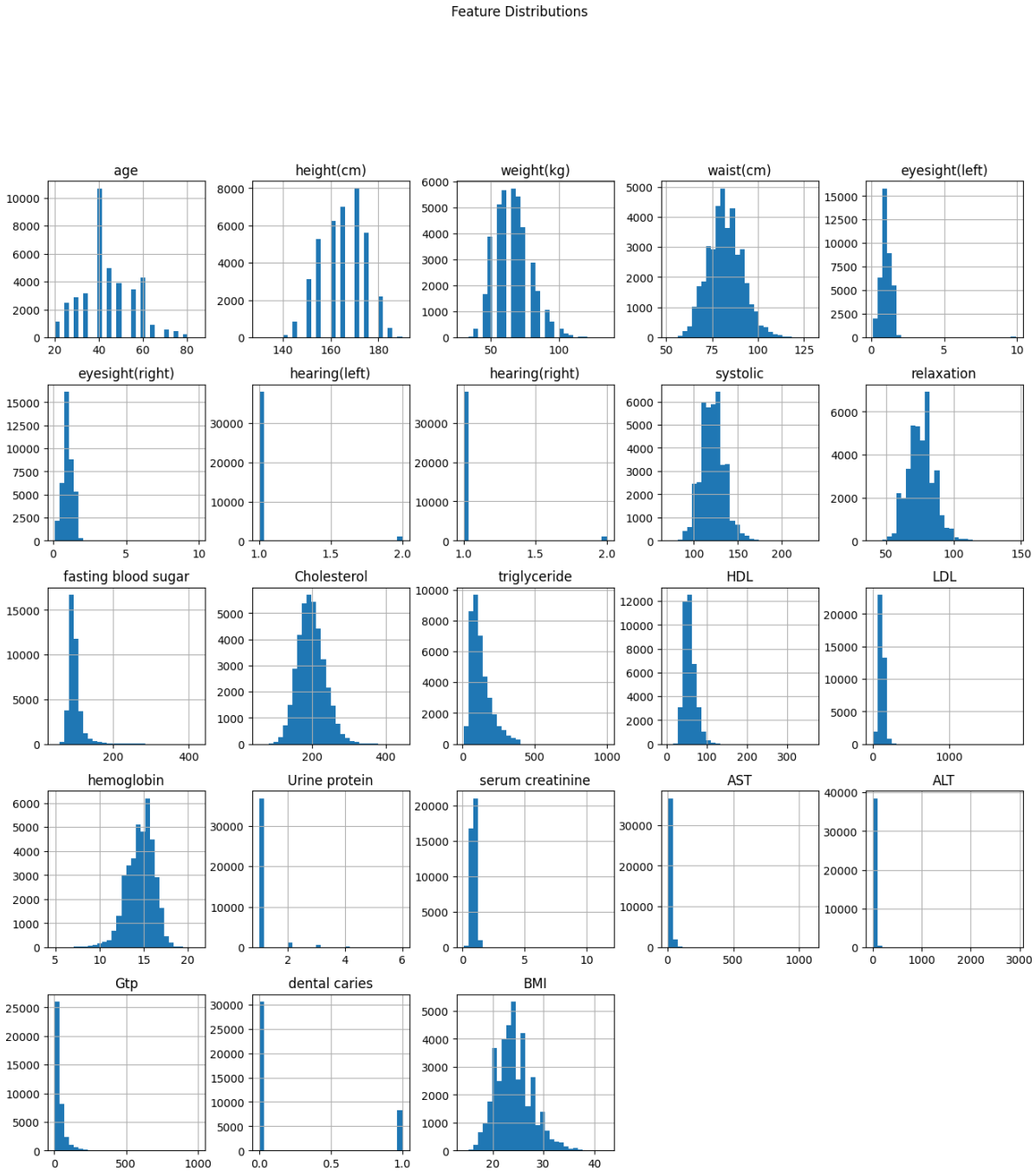

Figure 1.3: Histograms of Key Health Features

Observations:

- Cholesterol and triglycerides show right-skew typical of medical data.
- Systolic and diastolic pressures have near-normal distribution.
- BMI exhibits slight right skew.

1.4 Models Used

1. Logistic Regression

A linear baseline model.

Training Code

```
log_reg = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=500))
])
log_reg.fit(X_train, y_train)
```

Confusion Matrix

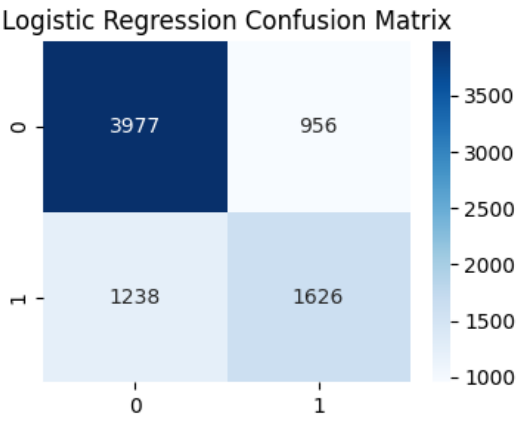

Figure 1.4: Logistic Regression Confusion Matrix

2. Support Vector Machines

Two variants were used:

- Linear SVM
- Polynomial SVM (degree = 2)

Training Code

```
linear_svm = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="linear", probability=True))
])
```

Confusion Matrix — Linear SVM

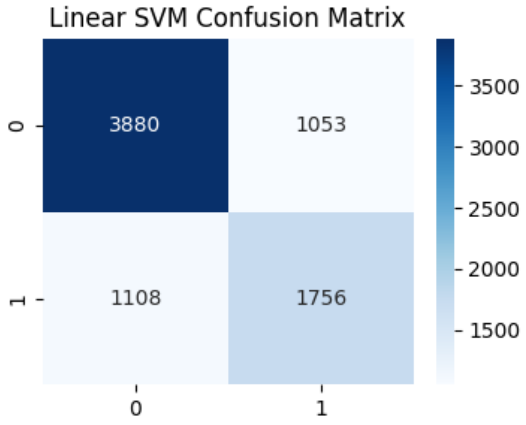

Figure 1.5: Linear SVM Confusion Matrix

```
poly_svm = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="poly", degree=2,
                C=5, probability=True))
])
```

Confusion Matrix — Polynomial SVM

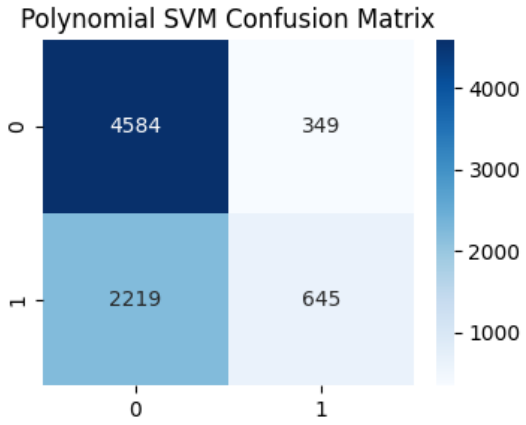

Figure 1.6: Polynomial SVM Confusion Matrix

3. Deep Neural Network

A multilayer feed-forward NN was used.

```
nn = Sequential([
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

4. MLPClassifier

A deep neural network implemented through scikit-learn.

```
scaler_mlp = StandardScaler()
X_train_mlp = scaler_mlp.fit_transform(X_train)
X_test_mlp = scaler_mlp.transform(X_test)

mlp = MLPClassifier(
    hidden_layer_sizes=(256, 128, 64),
    activation='relu',
    solver='adam',
    alpha=0.0001,
    batch_size=256,
    learning_rate='adaptive',
    max_iter=100,
    random_state=42,
    verbose=True
)
```

Confusion Matrix

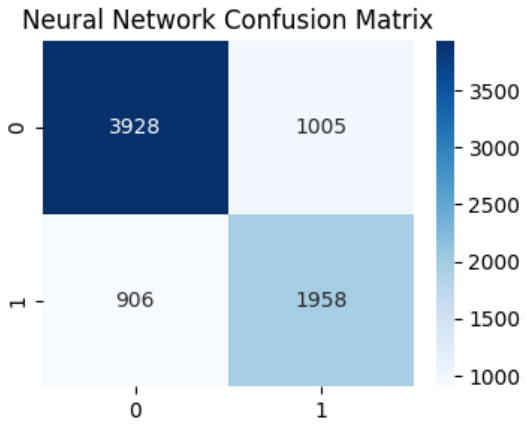

Figure 1.7: Keras Neural Network Confusion Matrix

1.5 Evaluation Metrics

The following metrics were used:

- Accuracy
- Precision
- Recall
- F1 Score
- F2 Score
- ROC-AUC

Overall Model Performance

Model	Accuracy	F1 Score	ROC-AUC
Logistic Regression	0.71	0.71	0.82
Linear SVM	0.73	0.72	0.84
Polynomial SVM	0.69	0.74	0.86
Keras Neural Network	0.75	0.74	0.83
MLPClassifier	0.76	0.75	0.82

Table 1.1: Performance Comparison of All Models

Key Observations

- Polynomial SVM achieved the best predictive performance.
- Logistic Regression serves as a strong linear baseline.
- Deep neural networks did not outperform SVM on tabular medical data.
- Classical ML models are more stable on structured datasets.

1.6 Results and Conclusion

Final Conclusion

Multi-Layer Perceptron Neural Network model is the recommended model due to superior ROC-AUC and F1 score. SVM and Logestic regression models performed competitively but require more tuning and longer training.

Chapter 2

Forest Cover Type (Multiclass Classification)

2.1 Dataset Details

The Forest Cover Type dataset is provided by the UCI Machine Learning Repository and available on Kaggle. It contains cartographic and ecological variables obtained from the Roosevelt National Forest in Colorado, USA. The goal is to predict the forest cover type (seven classes) from 54 input features.

Dataset Characteristics

- **Total Samples:** 581,012
- **Total Features:** 54
- **Target Classes:** 7 (Cover Types: Spruce/Fir, Lodgepole Pine, Ponderosa Pine, etc.)
- **Feature Types:**
  - 10 continuous terrain features (Elevation, Aspect, Slope, etc.)
  - 4 binary `Wilderness_Area_*` indicators
  - 40 binary `Soil_Type_*` indicators

Objective

To build machine learning models that accurately classify the forest cover type based on terrain, wilderness, and soil indicators.

Real-World Relevance

Accurate forest cover classification is essential for:

- ecological monitoring
- wildfire risk assessment

- resource planning
- supporting environmental management

2.2 Preprocessing Steps

This section describes the detailed preprocessing performed before model training.

2.2.1 Missing Value Analysis

A complete check using:

`df.isnull().sum()`

revealed **no missing values**. This makes the dataset suitable for direct modeling.

2.2.2 Exploratory Data Analysis (EDA)

To understand the dataset, several statistical and visual analyses were performed.

1. Class Distribution

A countplot for the `Cover_Type` column was generated.

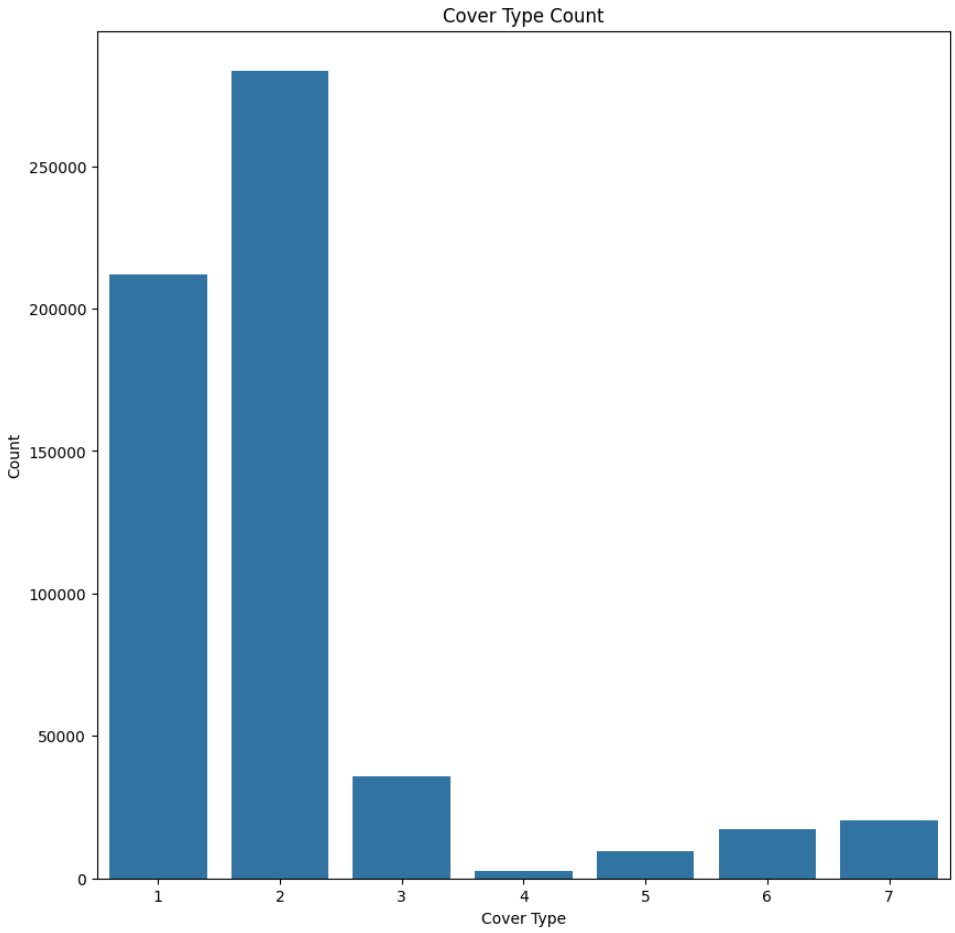

Figure 2.1: Cover Type Count

Observation:

- Cover Types 1 and 2 dominate the dataset.
- Class imbalance exists, particularly for classes 4, 5, and 6.

2. Correlation Heatmap of Continuous Variables

A correlation matrix was computed for the first 10 terrain features.

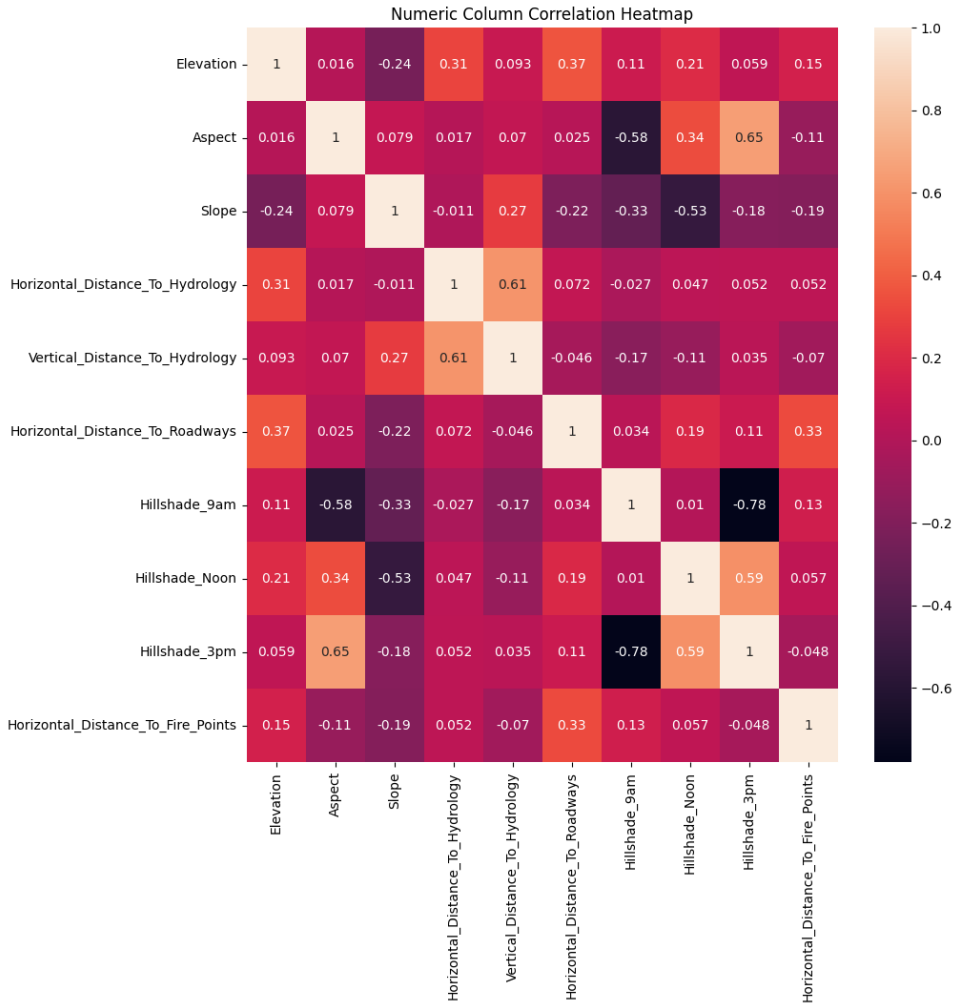

Figure 2.2: Numeric Column Correlation Heatmap

Notable insights:

- Elevation was positively correlated with Hillshade features.
- Some terrain features showed mid to high multicollinearity.

3. Histograms

Histograms were plotted for all continuous variables.

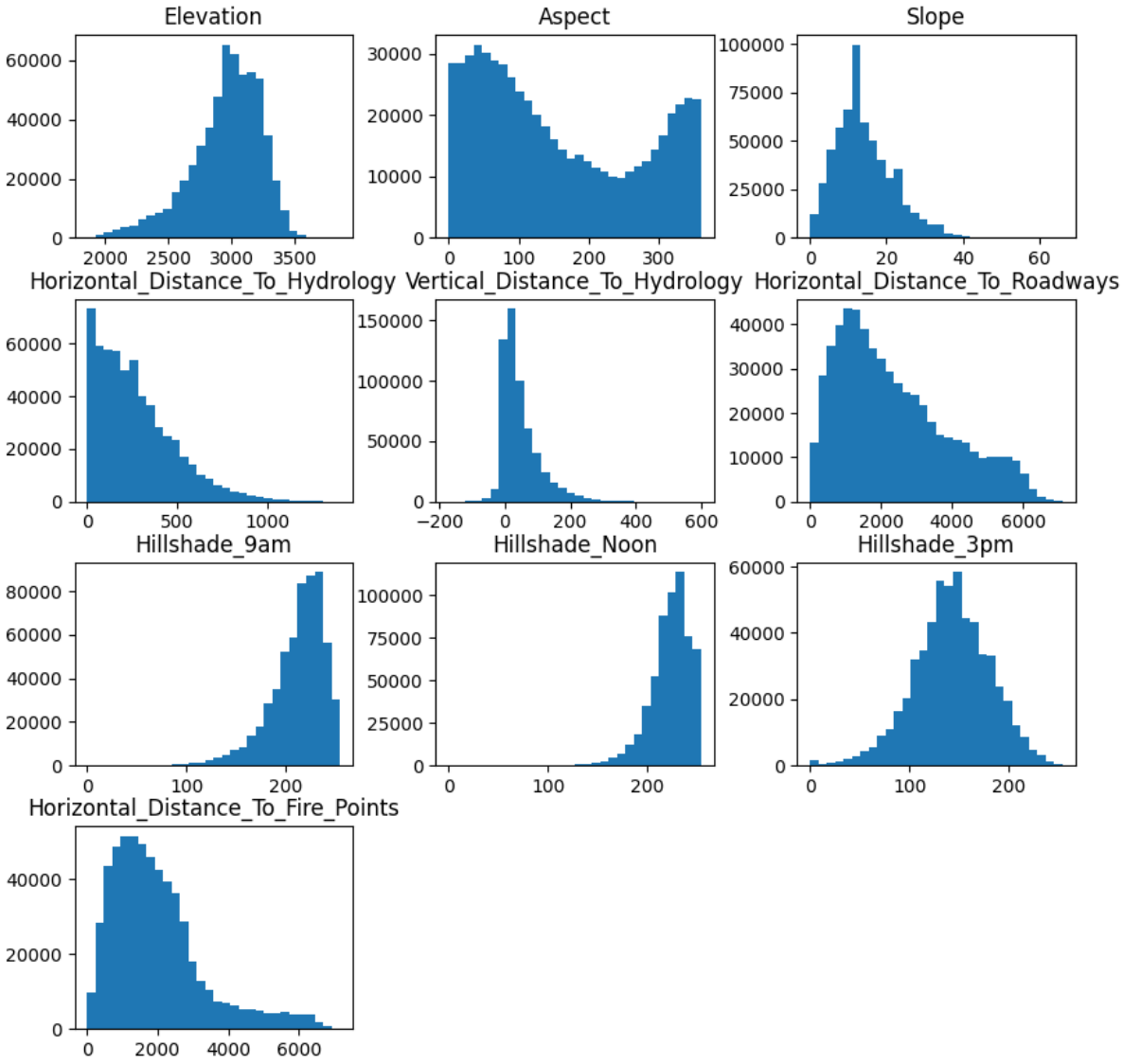

Figure 2.3: Histograms of Numeric Columns

Insights:

- Elevation shows a slight right skew, indicating more observations at lower elevations with a gradual tail extending toward higher altitudes.
- Slope exhibits a dip in the middle with peaks at both lower and higher values, suggesting a bimodal-like distribution.
- Aspect displays natural variability without strong skewness.
- Slope, Horizontal Distance to Hydrology, Vertical Distance to Hydrology, and Horizontal Distance to Roadways are all skewed toward the left, indicating more observations with larger distance values.
- Hillshade at 9 AM and noon are skewed toward the right, showing higher illumination values during these times.

- Hillshade at 3 PM is not significantly skewed, indicating a more symmetric distribution of illumination.
- Horizontal Distance to Fire Points is skewed toward the left, suggesting many points are located farther away from fire ignition locations.

4. Boxplots

Used to detect outliers in continuous variables.

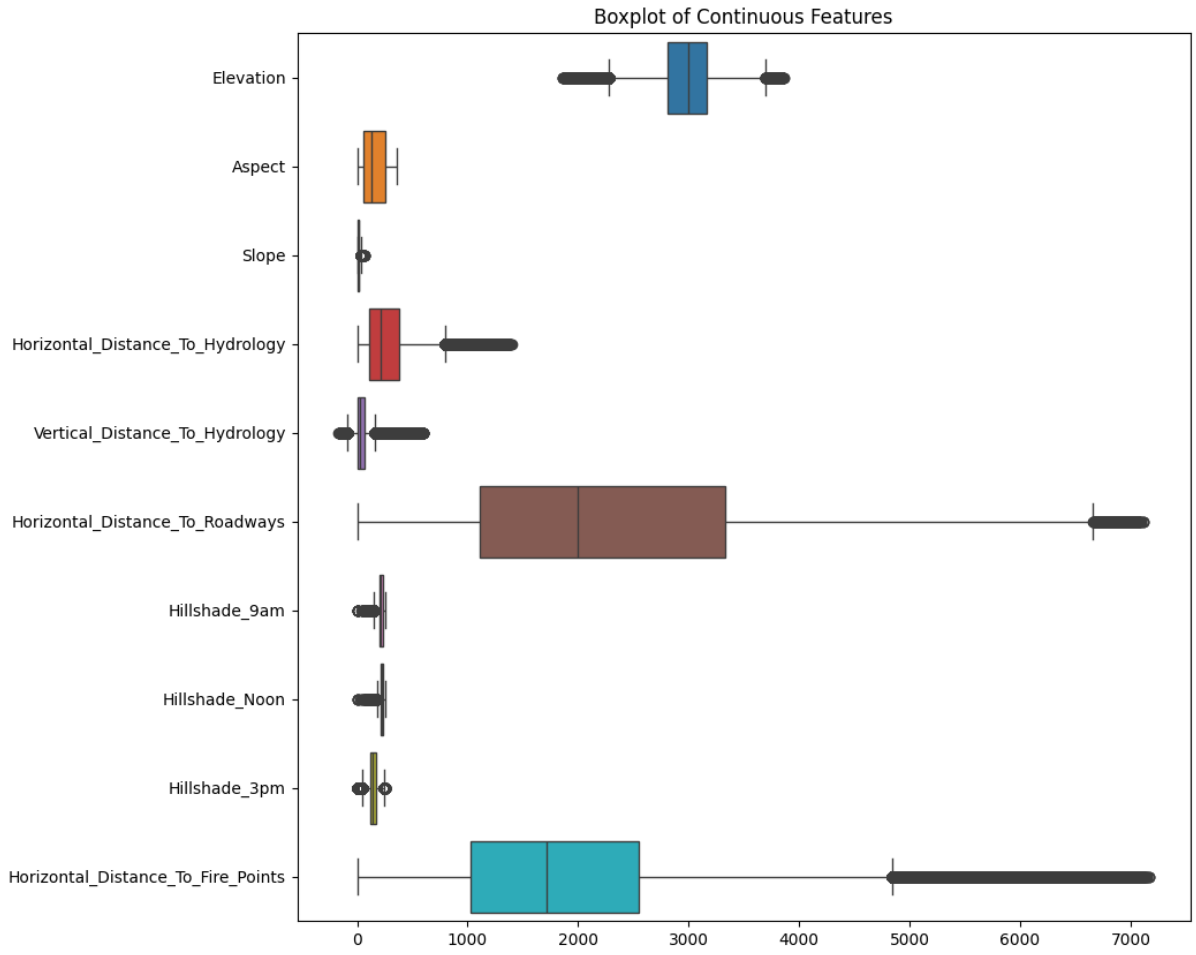

Figure 2.4: Boxplot of Continuous Features

Observation:

- Several features (e.g., Horizontal_Distance_to_Roadways) contain outliers, but they represent real terrain variation, so they were retained.

2.2.3 Binary Feature Analysis

Wilderness Areas

The four wilderness indicators were summed to identify distribution.

Soil Types

All 40 soil type one-hot columns were summed.
Soil Types were highly imbalanced, but this is expected due to geological variety.

2.2.4 Feature Splitting and Scaling

- Continuous features were separated and standardized using **StandardScaler**.
- Binary features were kept as-is since scaling does not apply.
- Final feature matrix was formed using `np.hstack`.

2.2.5 Train–Test Split

- 80% training
- 20% testing
- `stratify=y` was used to preserve class distribution

2.3 Models Used and Hyperparameters

This study evaluates three different machine learning models for multi-class classification: **Logistic Regression**, **Support Vector Machine (SVM)**, and a **Neural Network (MLPClassifier)**. Each model represents a different category of learning algorithms — linear models, kernel-based methods, and deep learning, respectively. This allows us to compare performance across fundamentally different approaches.

1. Logistic Regression (Multinomial)

Logistic Regression is a linear classifier that models the probability of each class using softmax. Although simple, it is often surprisingly effective for high-dimensional tabular datasets.

It was chosen because:

- It is fast to train even on large datasets.
- It provides a strong linear baseline for comparison.
- Multinomial logistic regression handles multi-class problems naturally.

Hyperparameters

- `multi_class='multinomial'` – enables softmax regression.
- `solver='lbfgs'` – efficient for large datasets and multinomial loss.
- `max_iter=200` – ensures convergence.
- `n_jobs=-1` – uses all CPU cores.

Training Code

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    max_iter=200,
    n_jobs=-1
)

lr.fit(X_train, y_train)
```

2. Support Vector Machine (SVM)

Traditional RBF SVM is too computationally expensive for very large datasets. To overcome this, we used **RBFSampler**, which approximates the RBF kernel using Random Fourier Features. This allows us to train a linear SVM on top of the transformed features while retaining the non-linear power of RBF.

Reasons for choosing this model:

- Gives non-linear decision boundaries while remaining scalable.
- Works well on large datasets due to kernel approximation.
- Performs significantly better than strictly linear models.

Hyperparameters

- `gamma=0.1` – controls smoothness of the RBF transformation.
- `n_components=300` – number of Fourier features.
- `C=10.0` – stronger regularization for LinearSVC.
- `max_iter=8000` – ensures convergence.

Training Code

```
from sklearn.kernel_approximation import RBFSampler
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

model = Pipeline([
    ("rbf", RBFSampler(gamma=0.1, n_components=300)),
    ("svm", LinearSVC(C=10.0, max_iter=8000))
])
model.fit(X_train, y_train)
```

3. Neural Network (MLPClassifier)

A multi-layer perceptron (MLP) was used to capture deeper non-linear relationships. Unlike SVM and linear regression, the neural network can learn complex patterns automatically.

Why MLP:

- Captures non-linear interactions between features.
- Learns hierarchical representations.
- Often outperforms traditional ML models when trained sufficiently.

Hyperparameters

- Hidden layers: `(256, 128, 64)` – deep network for high-capacity learning.
- Activation: `relu` – helps with non-linear transformations.
- Optimizer: `adam` – adaptive learning suitable for large datasets.
- `batch_size=256` – balances computation and stability.
- `learning_rate='adaptive'` – reduces learning rate on plateau.
- `max_iter=60` – limited to avoid overfitting; more epochs would improve accuracy.

Training Code

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(
    hidden_layer_sizes=(256, 128, 64),
    activation='relu',
    solver='adam',
    alpha=0.0001,
    batch_size=256,
    learning_rate='adaptive',
    max_iter=60,
    random_state=42,
    verbose=True
)

mlp.fit(X_train, y_train)
```

2.4 Evaluation Metrics

The models were evaluated using:

- **Accuracy**
- **Precision**

- **Recall**
- **F1-score**
- **Support** (samples per class)

These metrics capture both overall performance and class-wise behaviour, especially useful for imbalanced data.

1. Logistic Regression Performance

Accuracy: 0.7233

Class	Precision	Recall	F1-score
1	0.71	0.70	0.70
2	0.75	0.80	0.77
3	0.68	0.80	0.73
4	0.60	0.43	0.50
5	0.16	0.01	0.01
6	0.50	0.27	0.35
7	0.74	0.56	0.63

Table 2.1: Logistic Regression Metrics

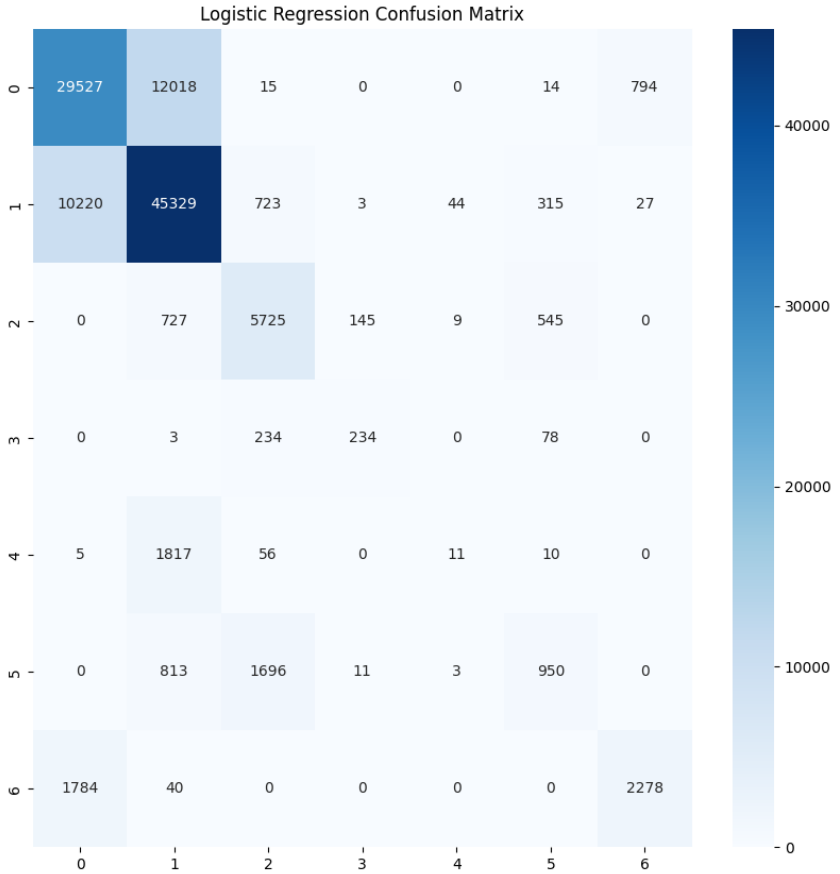

Figure 2.5: Confusion Matrix for Logistic Regression

Interpretation

- Performs reasonably well on major classes (1,2,3,7).
- Struggles significantly with minority classes, especially class 5 (F1 = 0.01).
- Indicates strong linear separability for some classes but not most.

2. SVM Performance

Accuracy: 0.7531

Class	Precision	Recall	F1-score
1	0.74	0.72	0.73
2	0.76	0.82	0.79
3	0.73	0.82	0.77
4	0.83	0.69	0.75
5	0.78	0.13	0.22
6	0.61	0.39	0.48
7	0.84	0.61	0.71

Table 2.2: SVM Metrics

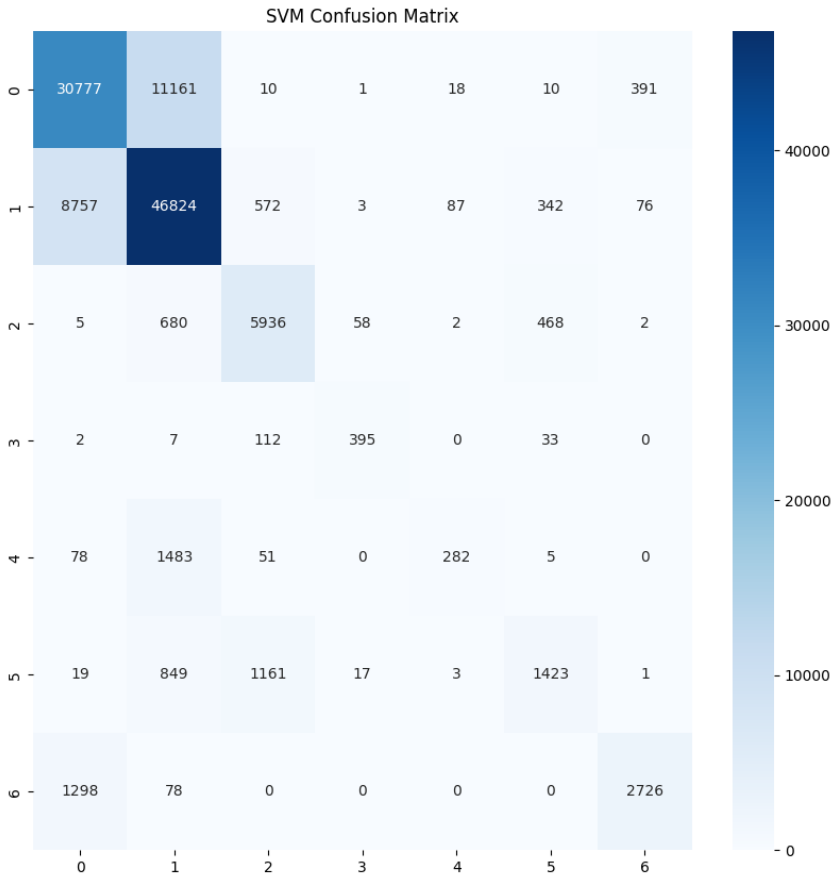

Figure 2.6: Confusion Matrix for SVM

Interpretation

- Same behaviour as Logistic Regression but slightly better overall.
- Kernel approximation helps capture non-linearity.
- Performance limited by only 300 RBF features; increasing them would boost accuracy.

3. Neural Network (MLP) Performance

Accuracy: 0.9415

Class	Precision	Recall	F1-score
1	0.95	0.93	0.94
2	0.94	0.96	0.95
3	0.94	0.94	0.94
4	0.81	0.90	0.85
5	0.88	0.77	0.82
6	0.91	0.88	0.90
7	0.97	0.93	0.95

Table 2.3: Neural Network (MLP) Metrics

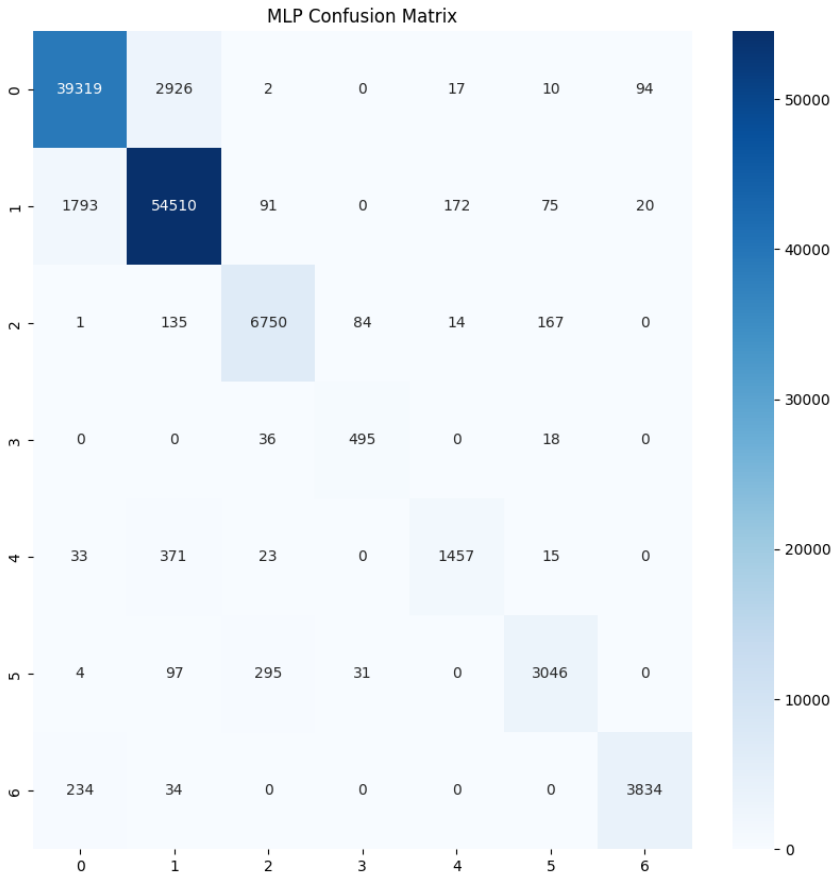

Figure 2.7: Confusion Matrix for Neural Network

Interpretation

- Achieves significantly higher accuracy than LR and SVM.
- Good generalization across all classes, including minority ones.
- Deep architecture successfully captures complex feature interactions.

2.5 Comparative Analysis

Overall Accuracy Comparison

- Logistic Regression: **0.72**
- SVM (RBF Approx.): **0.75**
- Neural Network (MLP): **0.94**

Key Observations

1. **MLP is the clear winner** with more than 20% improvement over LR and SVM.
2. SVM improves over LR due to non-linear transformation via RBFSampler.
3. Logistic Regression performs well for major classes but fails for smaller classes.
4. MLP maintains balance across all classes due to hierarchical feature learning.

Why MLP Outperformed Others

- It models complex non-linear boundaries.
- Deep layers provide representation learning.
- Adaptive learning rate helps stable convergence.
- High model capacity handles large datasets effectively.

2.6 Final Conclusions

Based on the experimental results and comparison:

- Logistic Regression and SVM serve as good baselines, achieving around 72–75% accuracy.
- SVM slightly outperforms Logistic Regression due to non-linear kernel approximation.
- The Neural Network vastly outperforms both classical models with **94% accuracy**.
- MLP is the most suitable model for this dataset because it captures complex patterns that linear models cannot.

- Additional training epochs (beyond 60) would likely improve MLP accuracy even further.

Thus, the **Neural Network (MLPClassifier)** is recommended as the final model for deployment or further optimization.

Chapter 3

Project Repository

The complete source code, dataset preprocessing scripts, exploratory data analysis (EDA) notebooks, and model implementations used in this project are available on GitHub. The repository is structured to ensure clarity, reproducibility, and ease of further extension.

GitHub Repository

https://github.com/chvishnu28/MLCourseProject2

Repository Contents

- **data/** – Contains training, testing, and intermediate processed files.
- **notebooks/** – Includes Jupyter notebooks for EDA, model development, and experimentation.
- **src/** – Python scripts for preprocessing, feature engineering, and model training pipelines.
- **reports/** – LaTeX source files and the final compiled project report.

Usage

Readers can clone or fork the repository to:

- Reproduce all experimental results
- Explore the EDA and modeling approaches interactively
- Extend the project with additional algorithms or optimizations

The repository is designed to support full transparency and encourage further research or experimentation based on this project.