Task-2 Detailed Report

Fraud Detection (Decision Tree from Scratch) & Pokémon DBSCAN Clustering

A Detailed Technical Report

1 Introduction

This report documents two machine learning tasks implemented **from scratch**:
(1) Fraud Detection using a custom Decision Tree classifier, and
(2) Unsupervised clustering of Pokémon data using PCA + DBSCAN.

The objective of this assignment was to avoid black-box ML libraries and instead implement core ML algorithms manually, gaining deeper understanding of the mathematical foundations behind them.

This report includes:

- Mathematical explanation of Decision Trees, PCA, and DBSCAN
- Python code snippets of the scratch implementations
- Evaluation results and visual outputs
- Comparison with Scikit-Learn equivalents
- Research-style discussion and observations

2 Part 1: Fraud Detection using a Scratch Decision Tree

2.1 Dataset Overview and Preprocessing

The dataset `onlinefraud.csv` contains financial transaction records labelled as fraud or non-fraud. The following preprocessing steps were performed:

- Removal of irrelevant or constant-value columns
- Median imputation for missing numeric values
- Factorization for categorical variables
- Undersampling to balance the fraud and non-fraud classes

2.2 Mathematics of Decision Trees (Gini Impurity)

A Decision Tree recursively splits the data using a feature and threshold that maximizes class separation.

Gini Impurity

Gini impurity measures how impure a node is:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \tag{1}$$

where p_i is the proportion of class i in the node.

Information Gain

A split is chosen to maximize the reduction in impurity:

$$Gain = Gini_{parent} - \left(\frac{N_L}{N} Gini_L + \frac{N_R}{N} Gini_R \right) \tag{2}$$

2.3 Scratch Code Snippet

Below is a simplified version of the code used to compute Gini and determine the best split:

Listing 1: Gini + Best Split Implementation

```
def gini(self, y):
    counts = np.bincount(y)
    probs = counts / len(y)
    return 1.0 - np.sum(probs ** 2)

def best_split(self, X, y):
    parent = self.gini(y)
    best_gain = 0
    for feat in range(X.shape[1]):
        for t in np.unique(X[:, feat]):
            left = X[:, feat] <= t
            right = ~left
            gain = compute_gain(...)
            if gain > best_gain:
                best_feat, best_thresh = feat, t
```

2.4 Model Evaluation Results

The model was trained using an 80-20 split. Evaluation metrics computed:

- Confusion Matrix
- Precision, Recall, F1-score
- ROC Curve and AUC
- Feature Importance based on Information Gain

Plots to Insert in Report:

- Confusion Matrix Heatmap
- ROC Curve
- Feature Importance Bar Graph

2.5 Comparison with Scikit-Learn Decision Tree

The scratch decision tree was compared with the Scikit-Learn implementation:

- Scratch model produced comparable accuracy and AUC
- Differences arise due to:
  - No pruning in scratch model
  - Threshold search limited to unique feature values
  - Scikit-Learn uses optimized C-based computations

3 Part 2: Pokémon Clustering using PCA + DBSCAN

3.1 Dimensionality Reduction with PCA

The Pokémon dataset contains stats and categorical attributes (types, abilities). One-hot encoding results in a high-dimensional feature space.

To reduce dimensionality while preserving variance, PCA projects data into a lower-dimensional space:

$$X_{pca} = (X - \mu)W_k^T \tag{3}$$

where W_k contains the top k eigenvectors of the covariance matrix.

20 components were used, retaining most of the variance.

3.2 DBSCAN Clustering

DBSCAN groups points based on density. It requires two parameters:

- ϵ (eps): neighborhood radius
- $minPts$: minimum points to form a dense region

A core point satisfies:

$$|N_\epsilon(p)| \geq minPts \tag{4}$$

Points not belonging to any cluster are labeled as noise.

Silhouette scores were computed for multiple values of eps and minPts.

3.3 Scratch DBSCAN Implementation Snippet

Listing 2: Core DBSCAN Expansion Logic

```
if len(neighbors) < min_samples:
    labels[i] = -1  # noise
else:
    labels[i] = cluster_id
    queue = list(neighbors)
    while queue:
        j = queue.pop(0)
        if labels[j] == -1:
            labels[j] = cluster_id
```

3.4 Results and Visualization

The outputs included:

- DBSCAN cluster scatter plot
- Silhouette score vs eps graph
- Silhouette score vs minPts graph
- Final Pokémon dataset saved with cluster labels

4 Conclusion

Both machine learning algorithms were successfully implemented from scratch and performed similarly to their Scikit-Learn equivalents. PCA + DBSCAN revealed meaningful Pokémon groupings based on abilities and stats, while the scratch Decision Tree effectively classified fraud cases.

These implementations demonstrate a strong understanding of core ML mathematics, algorithm mechanics, and evaluation methodology.