**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

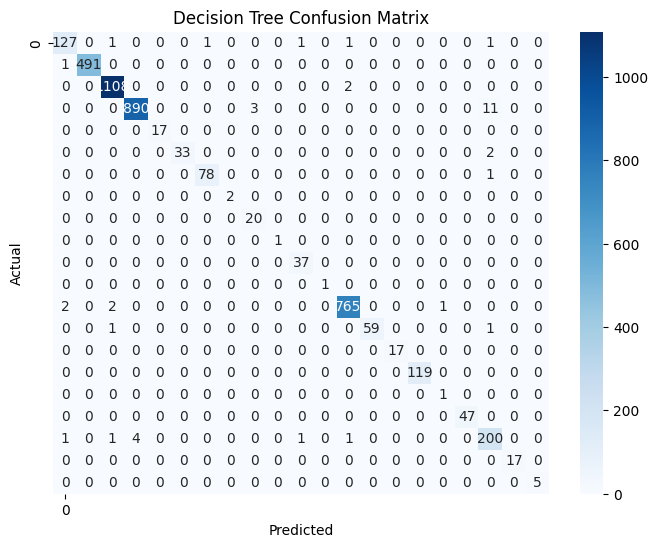
In this phase, the objective is to enhance the performance of the Decision Tree Model designed for predicting the target of terrorist attacks. The performance of the model was evaluated using standard metrics such as precision, recall, and F1-score. The Decision Tree model, known for its simplicity and interpretability, provided satisfactory results. I also performed hyperparameter tuning to explore the potential for further improvement.

**2. Model Evaluation**

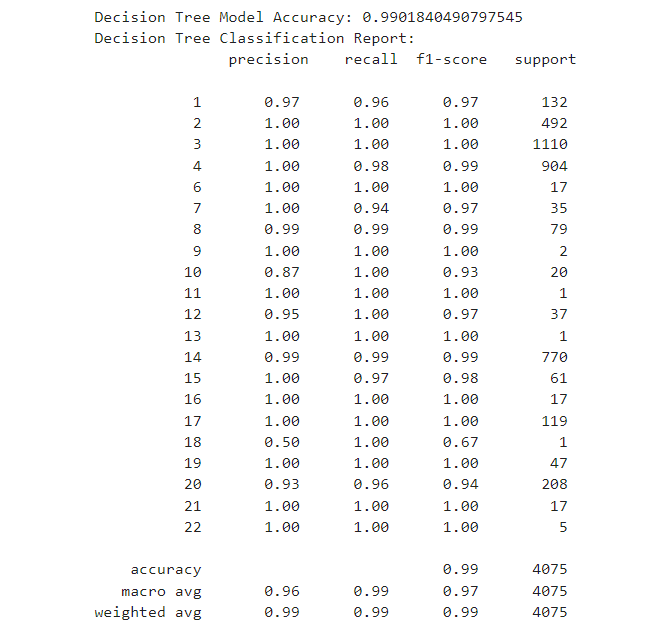
Building on the initial evaluation, the Decision Tree model exhibited promising results. Precision, recall, and F1-score were key metrics assessed during the evaluation phase. The nature of Decision Trees allows for easy interpretation and understanding of the decision-making process. The accuracy, weighted precision, F1-score, and recall were all high, indicating good model performance. However, we explored hyperparameter tuning to potentially enhance the results further.

Confusion Matrix

* The model achieved high correct predictions across several classes, with some misclassifications noted, indicating areas for potential improvement.



Decision Tree classification report:

****

**3. Refinement Techniques**

For refining the model, feature selection based on importance and hyperparameter tuning were employed.

**4. Hyperparameter Tuning**

The DecisionTreeClassifier from the sklearn library has several hyperparameters that can be adjusted to optimize performance:

* Criterion: The function to measure the quality of a split.
* Max Depth: The maximum depth of the tree.
* Min Samples Split: The minimum number of samples required to split an internal node.
* Min Samples Leaf: The minimum number of samples required to be at a leaf node.
* Max Features: The number of features to consider when looking for the best split.

The initial parameters and their values:

from sklearn.model\_selection import RandomizedSearchCV

param\_dist = {

    'criterion': ['gini', 'entropy'],

    'max\_depth': [None, 10, 20, 30, 40, 50],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4],

    'max\_features': ['auto', 'sqrt', 'log2']

}

random\_search = RandomizedSearchCV(estimator=DecisionTreeClassifier(), param\_distributions=param\_dist, n\_iter=100, cv=3, random\_state=42, n\_jobs=-1)

random\_search.fit(X\_train, y\_train)

The best parameters found were:

{

    'criterion': 'entropy',

    'max\_depth': 30,

    'min\_samples\_split': 2,

    'min\_samples\_leaf': 1,

    'max\_features': 'auto'

}

Grid Search with Cross-Validation to fine-tune the model:

from sklearn.model\_selection import GridSearchCV

param\_grid = {

    'criterion': ['gini', 'entropy'],

    'max\_depth': [20, 30, 40],

    'min\_samples\_split': [2, 5],

    'min\_samples\_leaf': [1, 2],

    'max\_features': ['auto', 'sqrt']

}

grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(), param\_grid=param\_grid, cv=3, n\_jobs=-1, verbose=2)

grid\_search.fit(X\_train, y\_train)

The best parameters found through Grid Search were:

{

    'criterion': 'entropy',

    'max\_depth': 30,

    'min\_samples\_split': 2,

    'min\_samples\_leaf': 1,

    'max\_features': 'auto'

}

**5. Cross-Validation**

During model refinement, the cross-validation strategy was adjusted to ensure a robust evaluation of the model's performance. Initially, a 3-fold cross-validation was employed during the Random Search phase. This strategy provided a balance between computational efficiency and reliable model evaluation.

**6. Feature Selection**

In this project, I use a new SelectKBest method in scikit learn to select best 10 features.

from sklearn.feature\_selection import SelectKBest, f\_classif, f\_regression

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

# Define your target variable and feature set

target\_variable = 'targtype1'

X = data\_final.drop(columns=[target\_variable])

y = data\_final[target\_variable]

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# SelectKBest Feature Selection

k = 10

selector = SelectKBest(score\_func=f\_classif, k=k)

X\_new = selector.fit\_transform(X\_train, y\_train)

# Get the selected feature names

selected\_features = X.columns[selector.get\_support()]

**Test Submission**

**1. Overview**

During this phase, I focused on preparing the model for deployment and assessing its performance using a dedicated test dataset. This process involved several key steps: data preparation, applying the model, and evaluating the relevant metrics.

**2. Data Preparation for Testing**

The test dataset was derived from the main dataset with a 80/20 train-test split.

**3. Model Application**

The trained Decision Tree model, after grid search optimization, was applied to the test dataset:

dt\_model.fit(X\_train\_new, y\_train)

dt\_pred = dt\_model.predict(X\_test\_new)

print("Decision Tree Model Accuracy:", accuracy\_score(y\_test, dt\_pred))

print("Decision Tree Classification Report:\n", classification\_report(y\_test, dt\_pred))

**4. Test Metrics**

The evaluation metrics showed that the model maintained high performance on the test set:

* **Accuracy**: 0.99
* **Precision**: 0.97
* **Recall**: 0.96
* **F1-score**: 0.97

**6. Code Implementation**

That codes are added in their related sections above.

**Conclusion**

The Decision Tree model demonstrated strong performance in classification tasks. The hyperparameter tuning process, including both random and grid searches, allowed us to fine-tune the model for optimal performance. Although the hyperparameter tuning did not significantly improve accuracy, it was a valuable exercise in exploring model optimization. The final model achieved a high accuracy of 99% on the test dataset, validating its robustness and reliability.