**Decision Tree Classifier with Entropy-Based Splitting**

This Python script is developed to build a Decision Tree Classifier using entropy as a measure to split nodes and make decisions. The goal is to predict the restaurant 'Type' based on different restaurant features in the dataset.

**Libraries Used**

- **pandas**: Used for data manipulation and analysis. In this script, we use it to convert the data dictionary into a DataFrame, which provides a more convenient data structure for data manipulation.

- **sklearn.preprocessing.LabelEncoder:** This class in the sklearn library is used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

- **sklearn.model\_selection.train\_test\_split:** This function is used to split the dataset into random train and test subsets.

**- sklearn.tree.DecisionTreeClassifier:** This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. The classifier is set to use entropy as the measure of the quality of a split.

- **sklearn.metrics:** The sklearn.metrics module implements several loss, score, and utility functions to measure classification performance.

- **matplotlib.pyplot:** A plotting library used for 2D graphics in python programming language. It provides an object-oriented API for embedding plots into applications.

**Data Overview**

The data used in this script is provided as a hardcoded dictionary. The dictionary is converted into a pandas DataFrame for ease of data manipulation. The data consists of different restaurant features and the restaurant 'Type' which is our target variable.

**Data Preprocessing**

Label Encoding is used to convert categorical text data into model-understandable numerical data. We have applied Label Encoding on each column of the DataFrame using a loop.

**Data Splitting**

The DataFrame is split into two sets, the features (X) and the target (y). The 'Type' column is our target, and the rest of the columns are our features. The dataset is further split into a training set and a test set with a 70%-30% ratio using sklearn's train\_test\_split function. The training set is used to train the model, and the test set is used to evaluate the model's performance.

**Model Training**

We instantiate a Decision Tree Classifier with the criterion set to "entropy". This means the model uses entropy to measure the quality of a split. The model is then trained using the training data.

**Making Predictions**

Once the model is trained, it can be used to make predictions. We use the test data (X\_test) to let the model predict the target values. These predicted values can be compared to the actual values (y\_test) to evaluate how well the model performs.

**Model Evaluation**

The model's performance is evaluated by comparing the predicted target values with the actual target values using the accuracy\_score function from the sklearn.metrics module. The accuracy score is the simplest way to evaluate a classification model, but keep in mind that it might not be the best metric for all situations, especially for imbalanced datasets.

**Decision Tree Visualization**

Finally, the decision tree is visualized using sklearn's plot\_tree function and matplotlib. This visual representation can help understand how the model is making decisions.

**Running the Script**

To run the script, Python 3 and the necessary libraries need to be installed on your machine. Once the environment is set up, you can navigate to the directory containing the script in the terminal/command prompt and type `python script\_name.py`, replacing `script\_name.py` with the name of the script file.

**Limitations and Future Considerations**

The dataset used in this script is quite simple. For a more complex, real-world dataset, more sophisticated preprocessing steps might be needed, including handling missing values, dealing with outliers, and potentially scaling features.

Moreover, the Decision Tree Classifier is a basic model that might not provide the best performance for complex datasets. Consider using more advanced models or ensemble methods for such datasets. Parameter tuning can also help to optimize the model's performance.