**Dataset Description**

The dataset consists of **15,000 labeled grayscale images** of clothing items sourced from the Zalando website, often used for fashion-related machine learning tasks. Each image is a **28x28 pixel** grayscale image, resulting in **784 pixel values** per image, where each pixel has an intensity value between 0 and 255.

The dataset is categorized into five distinct classes:

* **T-shirt/top** (label 0)
* **Trousers** (label 1)
* **Pullover** (label 2)
* **Dress** (label 3)
* **Shirt** (label 4)

Each image has an associated label, representing the clothing type. The dataset is divided as follows:

* **Training Set**: 10,000 images
* **Test Set**: 5,000 images

The data is stored in **NPY format** in two files (fashion\_train.npy and fashion\_test.npy). In these files, each row contains:

* **First 784 columns**: Pixel values of the 28x28 grayscale image.
* **785th column**: Clothing category label, which is an integer from 0 to 4, representing the type of clothing.

**Steps to Build a Decision Tree for Clothing Classification**

1. **Preprocess the Data**:
   * Load the NPY files and separate the **features** (pixel values) from the **labels** (clothing categories).
   * Optionally, normalize pixel values by dividing by 255, scaling them to a range of 0 to 1 to potentially improve model performance.
   * If needed, split the training data further into a training and validation set to fine-tune hyperparameters.
2. **Select the Splitting Criterion**:
   * Use **Information Gain** or **Gini Impurity** to evaluate the best splits at each node.
   * Since each feature is a pixel, the model will assess individual pixels and their values (intensity levels) to determine the best split. For example, darker or lighter pixel regions may indicate specific clothing types (e.g., trousers might be darker in the lower part of the image).
3. **Build the Tree Recursively**:
   * **Base Cases**: Stop further splitting and create a leaf node if:
     + All samples in a subset belong to the same category.
     + Maximum depth is reached (controlled by max\_depth).
     + Number of samples in the node is below a threshold (set by min\_samples\_leaf).
   * **Recursive Splitting**:
     + For each pixel, calculate the selected criterion (e.g., information gain) to find the most informative splits.
     + Divide the data based on the selected pixel value and continue splitting recursively for each subset until base cases are met.
4. **Pruning to Prevent Overfitting**:
   * Limit the depth of the tree (max\_depth) or set a minimum number of samples per leaf (min\_samples\_leaf) to avoid overfitting to the training data.
   * Use cross-validation to adjust these parameters and improve generalization.
5. **Train and Evaluate**:
   * Train the decision tree on the 10,000-image training set.
   * Evaluate the model on the 5,000-image test set, calculating metrics like accuracy, precision, recall, F1-score, and confusion matrix to understand its performance.

**Key Functions in the DecisionTree Class**

1. **\_\_init\_\_**: Initializes the decision tree classifier with parameters to control its complexity:
   * **max\_depth**: Limits the depth of the tree to avoid overfitting.
   * **min\_samples\_leaf**: Specifies the minimum number of samples required in a leaf node.
   * **min\_information\_gain**: Sets a threshold for the minimum information gain required for a split.
2. **entropy**: Computes the entropy of a set of class probabilities, measuring impurity in data subsets. Lower entropy means the subset is more homogeneous.
3. **class\_probabilities**: Calculates the probability distribution of each class in a given set of labels. This is used to determine data purity.
4. **data\_entropy**: Uses the class probabilities to compute entropy for an entire dataset or subset. It helps in understanding the impurity of the dataset before splitting.
5. **partition\_entropy**: Computes the weighted entropy across multiple subsets, essential for evaluating the quality of a split.
6. **split**: Splits the data based on a given feature (pixel) and a threshold, creating two subsets. This function is used repeatedly to create branches in the tree.
7. **find\_best\_split**: Iterates through all features (pixels) and calculates the best possible split point by evaluating entropy or information gain, identifying the feature that most effectively reduces impurity.
8. **find\_label\_probs**: Calculates probabilities of each class label in a subset. This can help determine the likelihood of each clothing type at a node.
9. **create\_tree**: Recursively builds the tree by selecting the best splits at each node. This function calls itself to add branches to nodes until the stopping criteria are met.
10. **predict\_one\_sample**: Takes a single image and predicts its category by traversing the tree from the root to a leaf node, based on the pixel values.
11. **train**: Trains the decision tree by calling create\_tree on the training data, building the model structure.
12. **predict\_proba**: Returns the probability distribution over classes for each sample, providing insight into the model’s confidence in its predictions.
13. **predict**: Predicts the clothing categories for a test set of images by applying predict\_one\_sample to each image in the set.
14. **print\_recursive**: Recursively prints the structure of the tree for each node and level. This is helpful for visualizing the decision-making process.
15. **print\_tree**: Prints the entire tree from the root node, providing a summary of the tree structure and splits.