**Part 2: Stage 1 - Baseline Prediction (SVM and KNN Classifiers)**

**Step 2.1: Implement Support Vector Machine (SVM)**

**Objective**: Train an SVM classifier using the extracted features, incorporating robust hyperparameter tuning.

The Support Vector Machine (SVM) algorithm is widely used for classification tasks. It works by mapping data to a higher-dimensional feature space and finding a linear hyperplane that maximizes the separation between classes. While primarily designed for binary classification, SVM can be extended to handle multi-class classification. The performance of an SVM classifier is heavily influenced by its learning parameters: kernel, gamma, and C.

For this implementation, we will use the Radial Basis Function (RBF) kernel, which is often preferred for high-dimensional feature sets due to its ability to efficiently model complex, non-linear relationships without significant computational overhead.

The RBF kernel has two key parameters:

* **Cost (C)**: This parameter regulates the trade-off between maximizing the margin and minimizing classification errors. A high C value emphasizes minimizing errors, potentially leading to overfitting, while a low C value allows for a wider margin and more misclassifications, leading to a more generalized model.
* **Gamma (gamma)**: This defines the influence of a single training example on the decision boundary. A high

gamma value results in a narrow decision boundary with localized effects, increasing the risk of overfitting. Conversely, a low

gamma value leads to a smoother, more generalized decision boundary, potentially causing underfitting.

To optimize these parameters and avoid model overfitting, a grid search algorithm combined with five-fold cross-validation will be applied. The model's final performance will be determined by averaging the results across these five iterations.

**Explanation and Possible Parameter Tuning Options:**

1. **Data Loading (Placeholder)**: The code starts with creating dummy data (X, y) because the actual feature loading depends on the output format from your MATLAB feature extraction. **You would replace this dummy data generation with your actual data loading code.** For instance, if your MATLAB script saves features to mango\_leaf\_features.csv where the last column is the label:

Python

# Example for actual data loading

# file\_path = 'path/to/your/mango\_leaf\_features.csv'

# df = pd.read\_csv(file\_path)

# X = df.iloc[:, :-1] # All columns except the last one are features

# y = df.iloc[:, -1] # The last column is the label

1. **param\_grid for GridSearchCV**:
   * 'C': [0.1, 1, 10, 100, 1000]
     + **Interpretation**: Controls the penalty for misclassified training observations. A smaller C creates a larger margin but allows more misclassifications (more regularization). A larger C creates a smaller margin but aims to classify all training data correctly (less regularization).
     + **Tuning Strategy**: Start with a wide range (e.g., powers of 10) and then narrow down if the optimal C is found at the boundaries of your initial range.
   * 'gamma': [1, 0.1, 0.01, 0.001, 0.0001]
     + **Interpretation**: Defines how far the influence of a single training example reaches. A high gamma means the influence is very localized, leading to potential overfitting. A low gamma means a wider influence, leading to a smoother decision boundary and potential underfitting.
     + **Tuning Strategy**: Similar to C, use powers of 10 initially. If the best gamma is at an extreme, expand the range in that direction.
   * 'kernel': ['rbf']
     + **Interpretation**: The kernel function to be used. The paper specifically mentions using 'rbf' for large feature sets. While 'linear' and 'poly' are also options for SVM, 'rbf' is chosen here as it generally performs well with complex, non-linear relationships often found in image features. If you wanted to test other kernels (though the paper specifies RBF), you could add

'linear', 'poly' here and also add 'degree' to param\_grid for the poly kernel.

* + - * 'linear': C
      * 'poly': C, degree, gamma, coef0
      * 'sigmoid': C, gamma, coef0
      * 'precomputed': requires a precomputed kernel matrix
    - **Tuning Strategy**: Stick to 'rbf' as per the paper's findings, but keep in mind other kernels exist if results are not satisfactory.

1. **GridSearchCV Parameters**:
   * estimator=svm: The model to tune.
   * param\_grid=param\_grid: The dictionary of parameters to search.
   * cv=5: Specifies 5-fold cross-validation, as stated in the paper. This helps in getting a more reliable estimate of the model's performance and prevents overfitting to a single train-test split.
   * verbose=3: Controls the verbosity of the output. Higher values mean more detailed output during the search.
   * n\_jobs=-1: Uses all available CPU cores, significantly speeding up the grid search process.
2. **calculate\_fnr Function**: This custom function calculates the False Negative Rate. The provided implementation for multi-class assumes class 0 is 'healthy' and all other classes are 'diseased'. **You might need to adjust this logic based on your actual label encoding if 'healthy' is not consistently class 0 or if you need FNR for specific disease classes.** The paper presents class-wise FNR values in Table 3, which would require a loop through each class for its specific FNR calculation.