plant disease classification

Name Ahmed sadek id: 2305355
Basmala ibrahim 2305229
Hana ahmed 2305211
Toqa osama 2305253

Introduction

Plant diseases are a major threat to agricultural productivity worldwide. Timely identification of these diseases is critical for preventing crop loss. In this project, we aim to develop a machine learning model using the K-Nearest Neighbors (KNN) algorithm to classify plant diseases. The classification can be based on features such as leaf images or environmental data like temperature, humidity, and soil conditions.

KNN:

Dataset Preparation

- Subset Creation: The code selects a specific number of images per class (1,000 for training and 300 for validation) from the original dataset. This step ensures a manageable dataset size for experimentation.
- Directory Structure: The selected images are organized into a new directory structure, separating training and validation sets, which is essential for model evaluation

Feature Extraction and Classification Methods

Raw Pixel Features with PCA and KNN:

- Process: Images are resized to 16x16 pixels, flattened into vectors, and then reduced in dimensionality using Principal Component Analysis (PCA).

- Classification: A K-Nearest Neighbors (KNN) classifier is trained on the PCA-transformed features.

Raw Pixel Features with PCA and Random Forest:

- Process: Similar to the previous method, but images are resized to 64x64 pixels. PCA is applied for dimensionality reduction.

- Classification: A Random Forest classifier is trained on the reduced features.

Histogram of Oriented Gradients (HOG) with PCA and KNN (Grid Search):

- Process: HOG features are extracted from 128x128 grayscale images to capture edge and texture information. PCA reduces feature dimensionality.

- Classification: KNN classifier is optimized using GridSearchCV to find the best hyperparameters.

Color Histograms with KNN (Grid Search):

- Process: Color histograms are computed for each RGB channel, capturing color distribution in images.

- Classification: KNN classifier is optimized using GridSearchCV to determine optimal parameters.

Wavelet Transform Features with KNN:

- Process: Wavelet transforms (specifically using the 'haar' wavelet) are applied to grayscale images to extract frequency and location information.

- Classification: A KNN classifier is trained on the wavelet-transformed features..

Pros:

- Simple, no feature engineering.

Cons:

- High-dimensional; images may contain irrelevant background pixels.
- PCA reduces the noise and redundancy, and helps avoid overfitting.

Classifiers Used

K-Nearest Neighbors (KNN):

- Simple and intuitive.
- Non-parametric: good for small datasets.
- Can be slow with large test sets unless optimized with KD-tree or Ball-tree.

Random Forest:

- Ensemble method: combines multiple decision trees.
- Handles non-linear data well.
- Resistant to overfitting, especially with many features.

Hyperparameter Tuning with GridSearchCV:

- Applied to KNN: helps find optimal k (neighbors), distance metric (euclidean, manhattan), and weight strategy.
- Avoids manual guesswork and improves model performance.

Evaluation Metrics

- **Accuracy**: Overall percentage of correct predictions.
- **Precision**: Accuracy of positive predictions per class.
- **Recall**: Ability to detect all positive instances per class.
- **F1-score**: Balance between precision and recall.
- **Confusion Matrix**: Visual representation of prediction errors per class.

All these metrics are computed using sklearn.metrics.classification_report and confusion_matrix.

Visualization Techniques

- **Wavelet Decomposition Plots: Shows how images are split into LL, LH, HL, HH bands.**
- **Confusion Matrix Heatmaps: Generated using seaborn.heatmap to see class-wise performance.**
- **Sample Image Plots: Randomly selected train/val images for sanity check.**

Potential Improvements

✅ A. Data Augmentation

- **Add rotation, flipping, zoom, etc., to increase diversity.**
- **Libraries: imgaug, Albumentations, or tf.keras.preprocessing.image.ImageDataGenerator.**

✅ B. Deep Learning Models

- **Pre-trained CNNs like ResNet, EfficientNet, MobileNet can extract high-level features.**
- **Use transfer learning to fine-tune on plant disease dataset.**

✅ C. Feature Fusion

- **Combine multiple features (e.g., HOG + color histogram) into a single feature vector.**
- **Improve classifier performance using a richer representation.**

✅ D. Ensemble Classifiers

- **Combine KNN, Random Forest, and others using voting or stacking for better performance.**

DECISION_TREE:

Dataset and Preprocessing

- Dataset Used:
- Source: New Plant Diseases Dataset (Augmented)
- Contains images of plant leaves categorized by disease type.

Preprocessing Steps:

Subsampling:

- 500 images per class for training.
- 100 images per class for validation.
- Randomly sampled to ensure class balance.
- Path: The subset is saved in structured folders (/train and /valid), each with subfolders per class.
- def create_subset(...)

This function copies a specified number of images per class from the original dataset into a new reduced dataset for training and validation purposes.

Feature Extraction

Feature vectors are generated for each image using the following methods:

1. Color Histogram:

- A 3D color histogram in RGB space with 8 bins per channel.
- Captures the distribution of colors in the image.
- Normalized and flattened to form part of the feature vector.

2. Laplacian Variance:

- Measures the variance of the Laplacian (second derivative) of the grayscale image.
- Captures the level of detail or "blurriness" in the image.
- A higher value implies a sharper, more detailed image — often indicative of visible disease symptoms.
- These two are combined into a single feature vector using np.hstack

Data Organization

Image paths are organized by class using dictionaries:

- training_images_path and validation_images_path: store paths to each class's images.
- prepared_training_data and prepared_validation_data: store the extracted feature vectors for each class.

Model Training

Model:

- DecisionTreeRegressor from sklearn.tree
- Although typically used for regression, it's used here with integer-labeled classes.
- The regressor is trained to output class indices (0, 1, 2, ..., N).
- Not ideal for classification — DecisionTreeClassifier would be more appropriate, but this serves as an experimental baseline.

Training and Validation:

- Features and labels (x, y) are built from the prepared_training_data.
- The model is trained using clf.fit(x, y).
- Predictions are made on validation data: y_pred = clf.predict(x_test).

EvaluationMetrics

- **Classification Report**: Includes precision, recall, and F1-score per class.
- **Accuracy**: Overall percentage of correct predictions.
- **Confusion Matrix**: Visual heatmap showing actual vs predicted class labels.
- **Encoding:**LabelEncoder is used to handle class label visualization in the confusion matrix

Observations

- This pipeline establishes a baseline using handcrafted features and classical ML.
- Using DecisionTreeRegressor instead of a classifier introduces approximation errors (float predictions).
- You could improve predictions by rounding the outputs of the regressor or using DecisionTreeClassifier.

Recommendations for Improvement

- Use DecisionTreeClassifier for classification tasks.
- Convert predictions to integers if using regressors (e.g., np.round()).
- Add feature normalization/scaling to ensure uniform feature importance.
- Test additional classifiers: SVM, KNN, Random Forest.
- Use CNN-based deep learning models for better accuracy.

SVM-RANDOM FOREST:

Dataset and Preprocessing

Dataset Used:

- Source: New Plant Diseases Dataset (Augmented)
- Contains images of plant leaves categorized by disease type.

Preprocessing Steps:

Subsampling:

- 500 images per class for training.
- 100 images per class for validation.
- Randomly sampled to ensure class balance.
- Path: The subset is saved in structured folders (/train and /valid), each with subfolders per class.
- def create_subset(...)

This function copies a specified number of images per class from the original dataset into a new reduced dataset for training and validation purposes.

Feature Extraction

Color Histogram Features:

- Each image is converted to RGB format, resized to 128×128 pixels, and processed to extract color histograms for each channel (Red, Green, Blue) using 32 bins. The histograms are normalized and concatenated to form a feature vector.
- def extract_color_histogram(image, bins=32)

This provides a compact and informative representation of color distribution, which can help differentiate between disease types that manifest as discoloration patterns.

Machine Learning Models

Training & Validation:

Data is loaded and labeled using:

- load_data(image_dir, img_size=(128, 128))
- This returns feature vectors (X_train, X_val) and their corresponding encoded labels (y_train, y_val).

Models Implemented

1. Support Vector Machine (SVM):

- Used for multiclass classification.
- Applied Grid Search with cross-validation to find optimal hyperparameters (C, kernel, gamma).
- Best model selected via:
- GridSearchCV(SVC(), param_grid, cv=5)
- Evaluation: Accuracy and Confusion Matrix.
-

2. Logistic Regression:

- Uses multi_class='multinomial' with lbfgs solver for multiclass output.
- Trained on histogram features.
- Quick baseline model for comparison.
-

3. Naive Bayes (GaussianNB):

- Probabilistic model assuming feature independence.
- Simple and efficient baseline model.

4. Random Forest:

- Ensemble method based on decision trees.
- Grid search used to tune hyperparameters (n_estimators, max_depth, min_samples_split).Often performs well in real-world classification tasks.
- Evaluation and Results

Evaluation Metrics

- **Accuracy**: Overall percentage of correctly classified images.
- **Confusion Matrix**: Visual comparison of predicted vs actual classes.
- **Classification Report** (used in earlier code): Includes precision, recall, and F1-score.

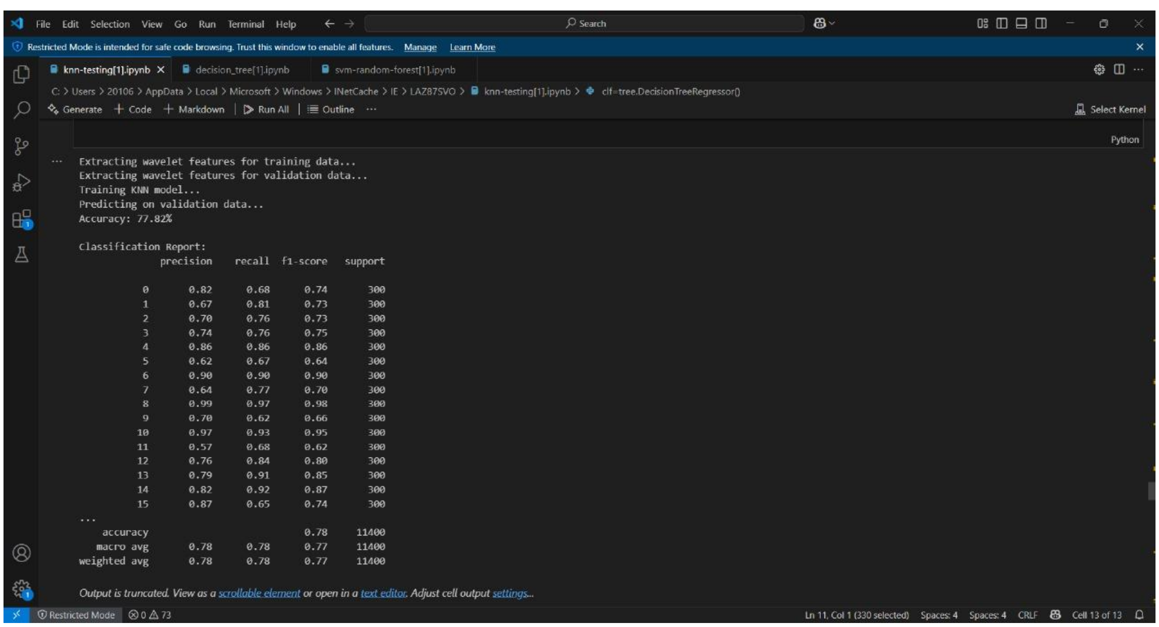
Visualization

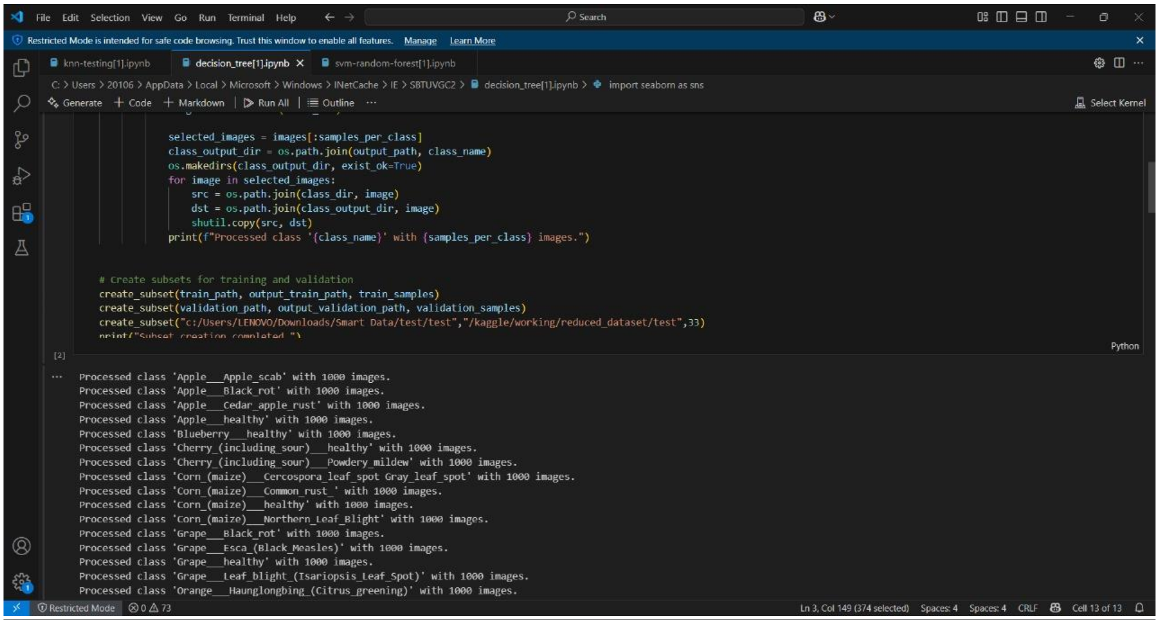
- Confusion matrices are plotted using Seaborn for interpretability.

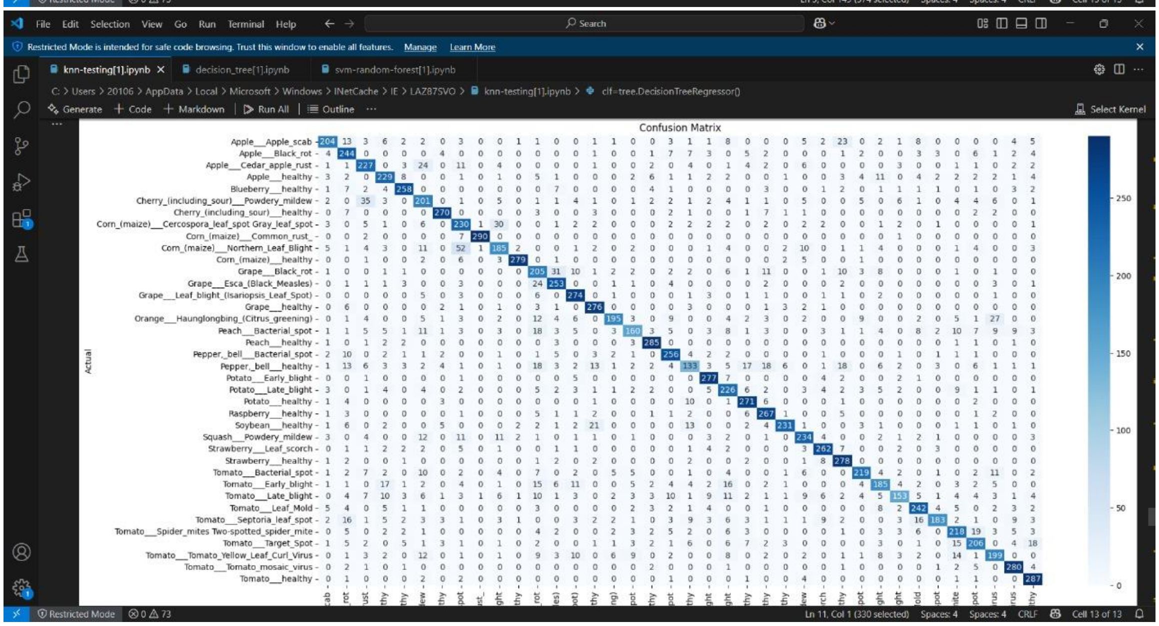
Recommendations for Improvement

- Add more features: Combine color, texture (e.g., HOG, LBP), and shape features.
- Use deep learning: Apply CNNs (e.g., ResNet, EfficientNet) for better feature learning.
- Augment data: Use more image augmentation to generalize better.
- Evaluate with F1-score: Especially for imbalanced class scenarios.

Output:




knn-testing[1].ipynb
decision_tree[1].ipynb
svm-random-forest[1].ipynb
selected_images = images[:samples_per_class]
class_output_dir = os.path.join(output_path, class_name)
os.makedirs(class_output_dir, exist_ok=True)
for image in selected_images:
src = os.path.join(class_dir, image)
dst = os.path.join(class_output_dir, image)
shutil.copy(src, dst)
print(f"Processed class '{class_name}' with {samples_per_class} images.")
# Create subsets for training and validation
create_subset(train_path, output_train_path, train_samples)
create_subset(validation_path, output_validation_path, validation_samples)
Processed class 'Apple___Apple_scab' with 1000 images.
Processed class 'Apple___Black_rot' with 1000 images.
Processed class 'Apple___Cedar_apple_rust' with 1000 images.
Processed class 'Apple___healthy' with 1000 images.
Processed class 'Blueberry___healthy' with 1000 images.
Processed class 'Cherry_(including_sour)___healthy' with 1000 images.
Processed class 'Cherry_(including_sour)___Powdery_mildew' with 1000 images.
Processed class 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot' with 1000 images.
Processed class 'Corn_(maize)___Common_rust_' with 1000 images.
Processed class 'Corn_(maize)___healthy' with 1000 images.
Processed class 'Corn_(maize)___Northern_Leaf_Blight' with 1000 images.
Processed class 'Grape___Black_rot' with 1000 images.
Processed class 'Grape___Esca_(Black_Measles)' with 1000 images.
Processed class 'Grape___healthy' with 1000 images.
Processed class 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)' with 1000 images.
Processed class 'Orange___Haunglongbing_(Citrus_greening)' with 1000 images.


knn-testing[1].ipynb
decision_tree[1].ipynb
svm-random-forest[1].ipynb
Confusion Matrix
Actual