Infected Leaf Identification using SVM

A PROJECT REPORT

Submitted by

Srimanta Ghosh(2023D005)

M.Sc. in Mathematics with Data Science

Under the guidance of

Mr. Kartik Sahoo

Institute of Mathematics and Applications

Bhubaneswar, Odisha

Contents

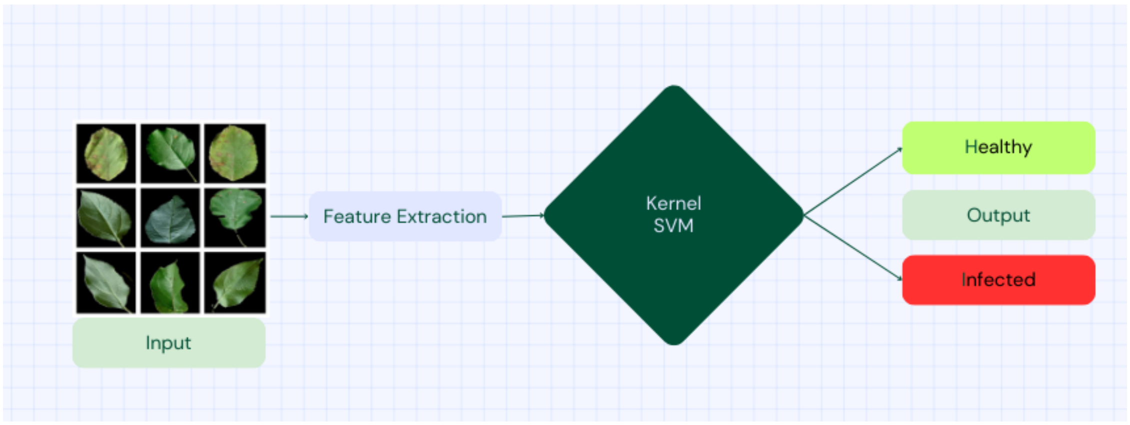
1 Introduction

Leaf diseases pose a significant threat to agricultural productivity, leading to substantial economic losses worldwide. Early detection and diagnosis are crucial for effective management and treatment of these diseases. Traditional methods of disease detection involve manual inspection by experts, which is time-consuming, labor-intensive, and often subject to human error. This project aims to develop an automated system for detecting leaf diseases using image processing techniques and machine learning algorithms. By leveraging these technologies, we aim to provide a more efficient, accurate, and scalable solution for farmers and agricultural experts.

1.1 Classical Approaches for Image Classification

Although Support Vector Machines (SVM) and their kernel extensions provide robust convex optimization frameworks without local minima, using these methods for image classification presents fundamental challenges. Specifically, the ambient space X should not be significantly large due to the computational intensity of the optimization process. One critical step in using the SVM framework is feature engineering, which involves pre-processing input images to obtain smaller dimensional vectors $x \in X$ that capture essential information. A classical pipeline for image classification can be summarized as follows:

- Process the dataset to extract hand-crafted features based on some knowledge of imaging physics, geometry, and other analytical tools.
- Extract features by feeding the data into a standard set of feature extractors such as Local Binary Pattern (LBP).
- Choose the kernels based on domain expertise.
- Put the training data composed of hand-crafted features and labels into a kernel SVM to learn a classifier.



In this classical approach, technical innovations typically arise from feature extraction, often based on serendipitous discoveries by researchers. Additionally, kernel selection requires domain expertise and has been a subject of extensive research. This project adheres to these classical principles by utilizing image processing techniques for feature extraction and employing an SVM for classification, thereby integrating domain knowledge and machine learning for effective leaf disease detection.

2 Data Collection

The data for this project consists of images of healthy and infected leaves, collected from various online sources and agricultural databases. The images are organized into two main categories: healthy and infected. These images are stored in separate folders for ease of processing.

[1]:
```
import os
import cv2
import numpy as np

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder, filename))
        if img is not None:
            images.append(img)
    return images

# Load images from folders
healthy_folder = r"C:\Users\gh22s\OneDrive\Desktop\Leaf_Dataset\Healthy"
infected_folder = r"C:\Users\gh22s\OneDrive\Desktop\Leaf_Dataset\Infected"

healthy_images = load_images_from_folder(healthy_folder)
infected_images = load_images_from_folder(infected_folder)
```

3 Image Processing and Feature Extraction

To process the images and extract relevant features, we followed these steps:

3.1 Converting RGB to HSI

The first step involves converting the RGB images to HSI (Hue, Saturation, Intensity) format using OpenCV. This helps in isolating the color information (hue), which is crucial for identifying disease patterns.

[2]:
```
# Function to convert RGB image to HSI
def rgb_to_hsi(image):
    hsi_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    return hsi_image
```

3.2 Leaf Segmentation Using GrabCut

Next, we segment the leaf from the background using the GrabCut algorithm, which refines the binary mask of the leaf and isolates it effectively.

[3]:
```
# Function for GrabCut segmentation
def segment_leaf(image):
    # Convert image to HSI
    hsi_image = rgb_to_hsi(image)

    # Extract hue component
    hue = hsi_image[:, :, 0]

    # Thresholding on hue channel to separate leaf from background
    _, binary_mask = cv2.threshold(hue, 30, 255, cv2.THRESH_BINARY)

    # Perform morphological operations to refine the mask
    kernel = np.ones((5, 5), np.uint8)
    opening = cv2.morphologyEx(binary_mask, cv2.MORPH_OPEN, kernel, iterations=2)
    sure_bg = cv2.dilate(opening, kernel, iterations=3)

    # Finding sure foreground area
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    _, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255,␣
↪0)

    # Finding unknown region
    sure_fg = np.uint8(sure_fg)
    unknown = cv2.subtract(sure_bg, sure_fg)

    # Marker labelling
    _, markers = cv2.connectedComponents(sure_fg)

    # Add one to all labels so that sure background is not 0, but 1
    markers = markers + 1

    # Now, mark the region of unknown with zero
    markers[unknown == 255] = 0

    # Apply watershed algorithm
    markers = cv2.watershed(image, markers)
    image[markers == -1] = [255, 0, 0]  # Mark watershed boundary on original␣
↪image

    # Convert markers to binary mask
    mask = np.zeros_like(binary_mask)
    mask[markers > 1] = 255  # Mark region other than background

    # Apply the mask to original image
    segmented_image = cv2.bitwise_and(image, image, mask=mask)

    return segmented_image
```

3.3 Extracting Features Using Local Binary Pattern (LBP)

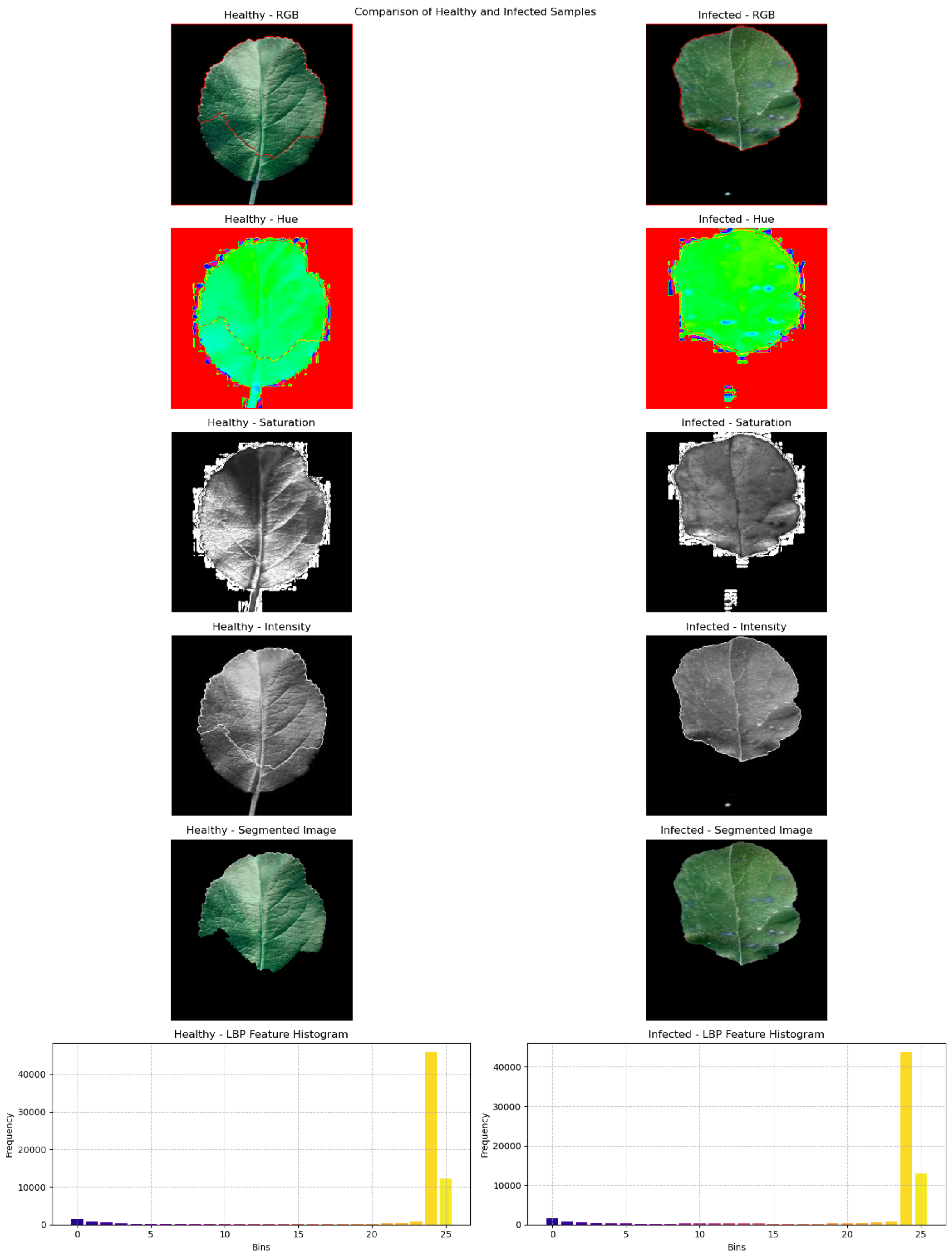
We use the Local Binary Pattern (LBP) method to extract texture features from the segmented leaf images. LBP is a powerful feature descriptor that captures the texture information of the images.

[4]:
```
from skimage.feature import local_binary_pattern

def extract_features(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    radius = 3
    n_points = 8 * radius
    lbp = local_binary_pattern(gray, n_points, radius, method='uniform')
    hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3),␣
 ↪range=(0, n_points + 2))
    return hist
```

3.4 visualization of sample images

Visualization of the HSI conversion and LBP feature extraction process for a sample healthy and infected leaf image. It displays subplots representing different stages of image processing, including RGB to HSI conversion, segmentation, and LBP feature histograms.

[4]:
```
# Function to visualize HSI conversion and extracted features for a sample image
def visualize_samples(healthy_image, infected_image):
    fig, axes = plt.subplots(6, 2, figsize=(15, 20))

    # Visualize healthy image
    axes[0, 0].imshow(healthy_image)
    axes[0, 0].set_title('Healthy - RGB')
    axes[0, 0].axis('off')

    axes[1, 0].imshow(rgb_to_hsi(healthy_image)[:, :, 0], cmap='hsv')
    axes[1, 0].set_title('Healthy - Hue')
    axes[1, 0].axis('off')

    axes[2, 0].imshow(rgb_to_hsi(healthy_image)[:, :, 1], cmap='gray')
    axes[2, 0].set_title('Healthy - Saturation')
    axes[2, 0].axis('off')

    axes[3, 0].imshow(rgb_to_hsi(healthy_image)[:, :, 2], cmap='gray')
    axes[3, 0].set_title('Healthy - Intensity')
    axes[3, 0].axis('off')

    segmented_img_healthy = segment_leaf(healthy_image)
    features_healthy = extract_features(segmented_img_healthy)

    axes[4, 0].imshow(segmented_img_healthy)
    axes[4, 0].set_title('Healthy - Segmented Image')
    axes[4, 0].axis('off')

    num_bins_healthy = len(features_healthy)
    colors_healthy = sns.color_palette("plasma", num_bins_healthy)

    axes[5, 0].bar(range(num_bins_healthy), features_healthy,␣
↪color=colors_healthy)
    axes[5, 0].set_title('Healthy - LBP Feature Histogram')
    axes[5, 0].set_xlabel('Bins')
    axes[5, 0].set_ylabel('Frequency')
    axes[5, 0].grid(True, linestyle='--', alpha=0.7)

    # Visualize infected image
    axes[0, 1].imshow(infected_image)
    axes[0, 1].set_title('Infected - RGB')
    axes[0, 1].axis('off')

    axes[1, 1].imshow(rgb_to_hsi(infected_image)[:, :, 0], cmap='hsv')
    axes[1, 1].set_title('Infected - Hue')
    axes[1, 1].axis('off')

    axes[2, 1].imshow(rgb_to_hsi(infected_image)[:, :, 1], cmap='gray')
    axes[2, 1].set_title('Infected - Saturation')
    axes[2, 1].axis('off')

    axes[3, 1].imshow(rgb_to_hsi(infected_image)[:, :, 2], cmap='gray')
    axes[3, 1].set_title('Infected - Intensity')
    axes[3, 1].axis('off')

    segmented_img_infected = segment_leaf(infected_image)
    features_infected = extract_features(segmented_img_infected)

    axes[4, 1].imshow(segmented_img_infected)
    axes[4, 1].set_title('Infected - Segmented Image')
    axes[4, 1].axis('off')

    num_bins_infected = len(features_infected)
    colors_infected = sns.color_palette("plasma", num_bins_infected)

    axes[5, 1].bar(range(num_bins_infected), features_infected,␣
↪color=colors_infected)
    axes[5, 1].set_title('Infected - LBP Feature Histogram')
    axes[5, 1].set_xlabel('Bins')
    axes[5, 1].set_ylabel('Frequency')
    axes[5, 1].grid(True, linestyle='--', alpha=0.7)

    plt.suptitle("Comparison of Healthy and Infected Samples")
    plt.tight_layout()
    plt.show()

# Visualize a sample image from the healthy dataset and the infected dataset
visualize_samples(healthy_images[0], infected_images[0])
```



4 Model Training and Evaluation

After extracting features from the images, we prepare the data for training and evaluation.

4.1 Data Preparation

We load the healthy and infected images, segment them, and extract features. These features are then labeled and combined into a single dataset.

[5]:
```python
# Create dataframes for healthy and infected features
healthy_df = pd.DataFrame(healthy_features)
infected_df = pd.DataFrame(infected_features)

# Label healthy as 0 and infected as 1
healthy_df['label'] = 0
infected_df['label'] = 1

# Combine dataframes
df = pd.concat([healthy_df, infected_df], ignore_index=True)
```

[6]:
```python
df
```

[6]:
```
        0     1    2    3    4    5    6    7    8    9  ...   17   18   19  \
0    1465   884  584  319  187  129  107  114  134  145  ...  118  121  151
1    2031  1089  751  465  325  224  190  166  202  201  ...  193  209  217
2     983   641  267   98   61   45   41   29   21   26  ...   35   43   44
3    1407   872  469  183  112   87   74   84   97  119  ...   75   86   81
4    1709  1017  494  221  128   96   85   76   83  112  ...   84   81   98
..    ...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
595  2312  1253  870  580  352  300  240  215  247  240  ...  255  227  248
596   735   410  256  155  101   80   74   82   58   72  ...   57   70   56
597  2496   946  907  745  558  433  340  303  292  334  ...  304  337  434
598  1130   545  456  306  212  226  196  212  246  253  ...  187  216  216
599  2089  1196  836  586  375  304  279  270  291  338  ...  402  407  338

      20   21   22    23     24     25  label
0    197  279  502   789  45950  12256      0
1    292  446  712  1011  38456  16623      0
2     39   70  183   528  54640   7454      0
3    118  192  362   789  47897  11422      0
4    157  185  439   883  45082  13606      0
..   ...  ...  ...   ...    ...    ...    ...
595  302  477  734  1234  34063  19083      1
596   81   87  179   348  56249   5842      1
597  562  645  883   914  35296  16047      1
```

```
598  224  260  367   475  48292   9079      1
599  364  452  732  1097  34172  17746      1

[600 rows x 27 columns]
```

4.2 Model Training

We use a Support Vector Machine (SVM) for classification. The data is split into training and testing sets, and the features are scaled using StandardScaler.

[7]:
```
# Model fitting
X = df.drop('label', axis=1)
y = df['label']

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# SVM model
svm_model = SVC()
svm_model.probability = True
svm_model.fit(X_train, y_train)
```

[7]:
```
SVC(probability=True)
```

4.3 Hyperparameter Tuning

Hyperparameter tuning was performed using `GridSearchCV`, exploring various combinations of `C`, `gamma`, and `kernel` for the SVM classifier. The best parameters identified were `{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}`, leading to an improved accuracy of 0.84 on the test set. Visualization through learning and ROC curves confirmed enhanced model performance and generalization.

[8]:
```
# Defining parameter ranges for different kernels
param_grid = {
    'C': [0.0001, 0.001, 0.1, 1, 10, 100, 1000],
    'gamma': [0.0001, 0.001, 0.1, 1, 10, 100, 1000],
    'kernel': ['linear', 'rbf', 'sigmoid']
}

# Create GridSearchCV object
grid = GridSearchCV(estimator=svm_model, param_grid=param_grid, refit=True,␣
 ↪verbose=3)

# Fit the model for grid search
```

```
grid.fit(X_train, y_train)
```

[9]:
```
# Get the best hyperparameters and model
best_params = grid.best_params_
best_model = grid.best_estimator_

# Evaluate the best model
y_pred_best = best_model.predict(X_test)
accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"Best SVM Accuracy: {accuracy_best:.2f}")
print(f"Best Hyperparameters: {best_params}")
```

```
Best SVM Accuracy: 0.84
Best Hyperparameters: {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

5 Results and Discussion

The SVM model achieved a high accuracy of 84% on the test set. The model's performance was evaluated using various metrics such as precision, recall, and F1-score. The classification report and confusion matrix provided insights into the model's strengths and areas for improvement.

[10]:
```
# Model evaluation
y_pred_best = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_best)
print("Classification Report:")
print(classification_report(y_test, y_pred_best))
```

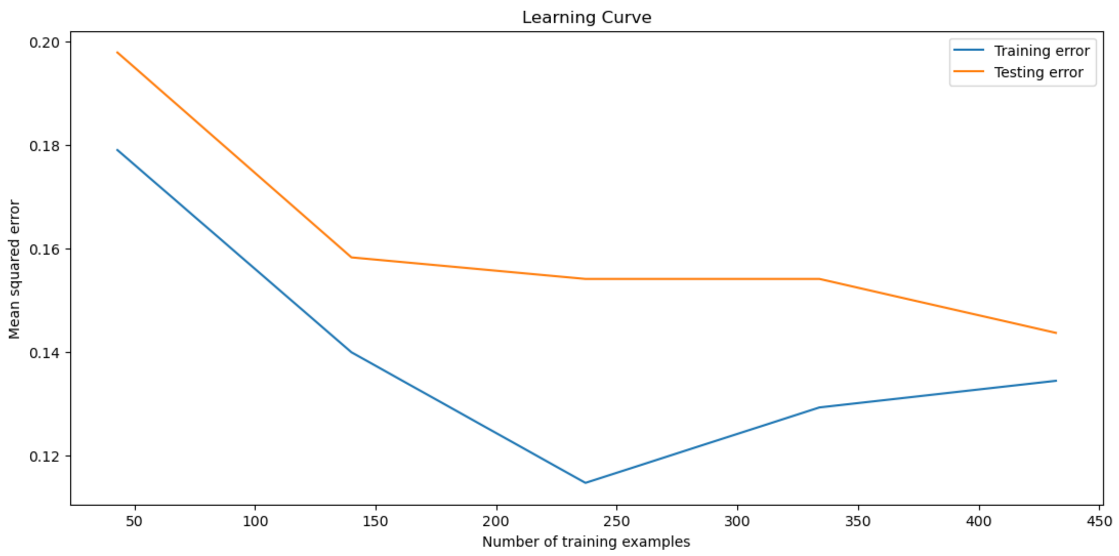
```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85        62
           1       0.83      0.84      0.84        58

    accuracy                           0.84       120
   macro avg       0.84      0.84      0.84       120
weighted avg       0.84      0.84      0.84       120
```

5.1 Learning Curve

The learning curve showed the model's performance with varying training sizes, indicating a well-fitted model with minimal overfitting. The ROC curve and AUC score demonstrated the model's ability to distinguish between healthy and infected leaves effectively.

[15]:
```
# Calculate learning curve
train_sizes, train_scores, test_scores = learning_curve(best_model, X_train,␣
 ↪y_train, cv=10, scoring='neg_mean_squared_error')
```

```python
# Calculate mean training and testing error
train_errors = -train_scores.mean(axis=1)
test_errors = -test_scores.mean(axis=1)

# Plot learning curve
plt.figure(figsize=(13, 6))
plt.plot(train_sizes, train_errors, label='Training error')
plt.plot(train_sizes, test_errors, label='Testing error')
plt.xlabel('Number of training examples')
plt.ylabel('Mean squared error')
plt.title('Learning Curve')
plt.legend()
plt.grid(False)
plt.show()
```

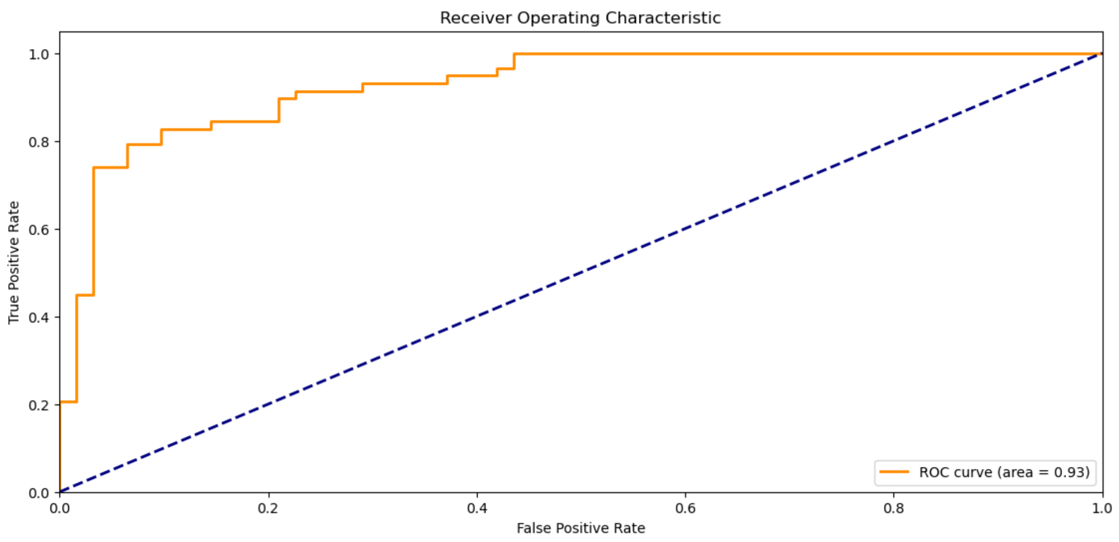


5.2 ROC Analysis

The ROC curve and AUC score demonstrated the model's ability to distinguish between healthy and infected leaves effectively.

[14]:
```python
# ROC analysis
y_prob = best_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(13.5, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
```

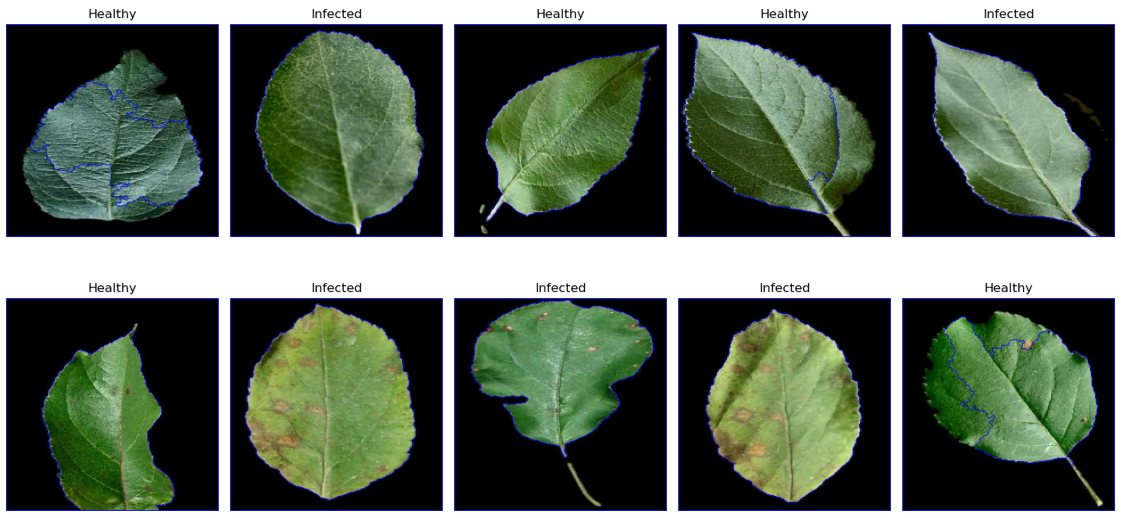
```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



6 Visualizing Test Images with Predicted Labels

To assess the model's performance on unseen data, we imported a folder containing test images. Subsequently, we scaled these images using the same scaler applied during training. After segmenting the leaf regions and extracting features, the model made predictions based on these features. The code snippet below demonstrates this process and displays the original test images alongside their predicted labels.

[16]:
```
# Function to display original images with labels in rows
def show_original_images_with_labels_in_rows(images, labels, num_rows=2):
    num_images_per_row = math.ceil(len(images) / num_rows)
    fig, axes = plt.subplots(num_rows, num_images_per_row, figsize=(15, 8))
    for i in range(num_rows):
        for j in range(num_images_per_row):
            index = i * num_images_per_row + j
            if index < len(images):
                axes[i, j].imshow(cv2.cvtColor(images[index], cv2.COLOR_BGR2RGB))
                axes[i, j].set_title(labels[index])
                axes[i, j].axis('off')
            else:
                axes[i, j].axis('off')
    plt.tight_layout()
    plt.show()

# Load images from the test folder
test_folder = r"C:\Users\dubey\OneDrive\Desktop\ML Classroom Project\Leaf_Test"
test_images = load_images_from_folder(test_folder)

# Scale the test images using the same scaler used during training
scaled_test_images = []

for img in test_images:
    # Segment the leaf region
    segmented_img = segment_leaf(img)

    # Extract features from segmented image
    features = extract_features(segmented_img)

    # Scale the features
    scaled_features = scaler.transform([features])  # Assuming 'scaler' is your␣
 ↪StandardScaler object

    # Make predictions
    prediction = best_model.predict(scaled_features)[0]
    scaled_test_images.append("Infected" if prediction == 1 else "Healthy")  #␣
 ↪Mapping 0 and 1 to Healthy and Infected

# Display original test images with predicted labels in two rows
show_original_images_with_labels_in_rows(test_images, scaled_test_images,␣
 ↪num_rows=2)
```



7 Conclusion

The project successfully demonstrates the application of Support Vector Machine for detecting leaf diseases. The developed system provides an efficient and accurate method for early detection, which is crucial for effective disease management and reducing economic losses in agriculture. Future work could involve expanding the dataset, exploring more advanced feature extraction techniques, and testing additional machine learning algorithms to further improve performance.