COURSE PROJECT 2 REPORT

Classification using SVM Model

Submitted by,

Vaibhav Kumar Jha, 201EE164

EE432 - Machine Learning

Under the guidance of

Dr. Yashwant Kashyap

Dept. of EEE, NITK Surathkal

Department of Electrical and Electronics Engineering,

National Institute of Technology Karnataka, Surathkal

15th November 2023

Abstract

This project delves into the realm of automated solar panel inspection using machine learning techniques. The primary goal was to develop an efficient system for classifying solar panels as either "clean" or "dusty" based on images captured in the field. Leveraging contour-based segmentation, we isolated the regions containing solar panels from the overall image, and subsequently employed K-means clustering to distinguish between clean and dusty panels using a threshold value. The Support Vector Machine (SVM) model was then trained on these labeled datasets, providing a robust solution for automated solar panel classification.

Introduction

The increasing reliance on solar energy necessitates the optimization and maintenance of solar panel installations to ensure optimal efficiency. One critical aspect of this maintenance is the periodic assessment of the cleanliness of solar panels, as dust accumulation can significantly impact their performance. Traditional manual inspection methods are time-consuming and labor-intensive. Therefore, the focus of this project is to introduce an automated system that can swiftly and accurately classify solar panels as clean or dusty.

The approach begins with contour-based segmentation, a method that effectively extracts the regions of interest (ROIs) containing solar panels from the input images. Following this, K-means clustering is applied to categorize these ROIs into clean and dusty panels based on a predefined threshold value. This preprocessing stage lays the foundation for the subsequent machine learning phase.

The heart of our classification system lies in the Support Vector Machine (SVM) model, a powerful supervised learning algorithm. The SVM is trained on labeled datasets generated through the segmentation and clustering steps, enabling it to learn the intricate patterns that distinguish between clean and dusty panels. The model's efficacy is then evaluated on new, unseen data, providing a reliable means of automated solar panel classification.

This report comprehensively details the methodology, experimental setup, results, and implications of the developed system, highlighting its potential contributions to the field of solar panel maintenance and efficiency optimization.

Support Vector Machines(SVM)

Support Vector Machines (SVMs) are a powerful class of supervised machine learning algorithms widely used for classification and regression tasks. SVMs are particularly well-suited for binary classification problems, such as the one tackled in this project: distinguishing between clean and dusty solar panels.

Theory Description:

SVMs work on the principle of finding an optimal hyperplane that best separates the data points of two classes while maximizing the margin between them. The key idea is to identify the support vectors, which are the data points closest to the decision boundary or hyperplane. These support vectors play a crucial role in defining the decision boundary and the margin. SVMs aim to find the hyperplane that not only separates the classes but also maximizes the distance between the hyperplane and the nearest support vectors, ensuring robustness and generalization to new data.

SVMs can handle both linear and non-linear classification tasks. In linear SVM, a linear decision boundary is sought, while in non-linear SVM, a kernel trick is applied to transform the feature space into a higher-dimensional space, making it possible to find a non-linear decision boundary. Common kernel functions include the linear, polynomial, radial basis function (RBF), and sigmoid kernels.

The SVM model is trained to minimize classification errors while also considering a regularization parameter (C) to control the trade-off between maximizing the margin and minimizing misclassifications. SVMs are known for their ability to handle high-dimensional data and effectively deal with outliers.

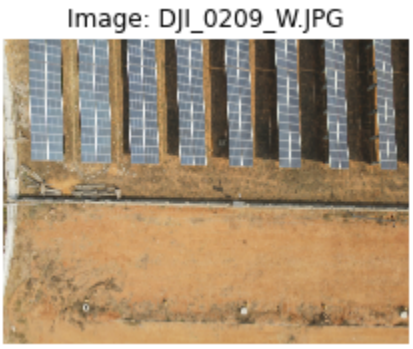
History:

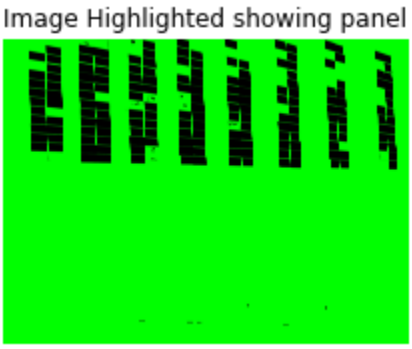
The concept of SVMs can be traced back to the early 1960s when Vladimir Vapnik and Alexey Chervonenkis began developing the theoretical foundations. However, it wasn't until the 1990s that SVMs gained prominence in the machine learning community.

Vapnik, in collaboration with other researchers, played a pivotal role in refining and popularizing SVMs.
The history of SVMs is marked by their successful application in various fields, including pattern recognition, image classification, and bioinformatics. Over the years, SVMs have evolved with advancements in kernel methods and optimization techniques, making them a fundamental tool in the machine learning toolbox. Their robustness and versatility continue to make SVMs a valuable asset in solving a wide range of classification and regression problems.

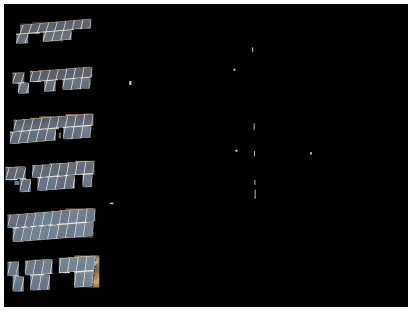
Different Steps taken

1. Apply contour based segmentation on the available images to extract the region of interest from the images.

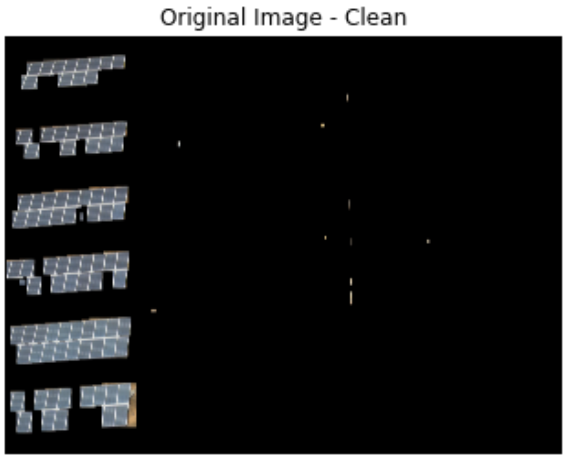


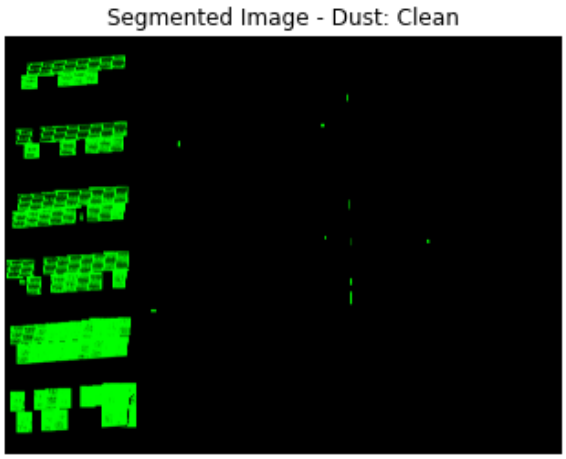


2. Create a new folder and save these images

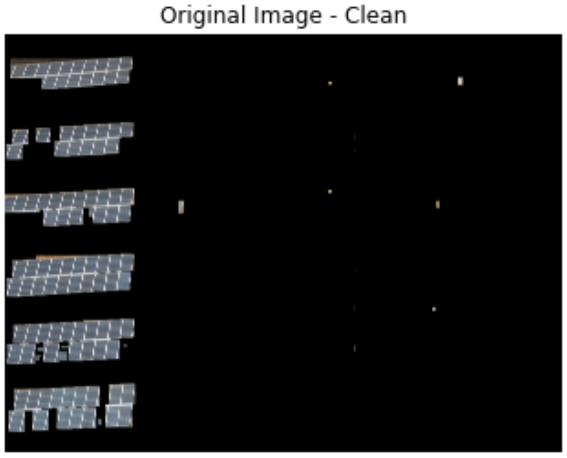


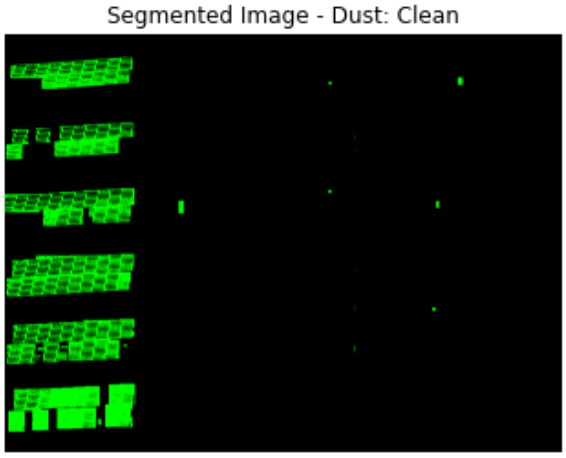
3. Apply K-Mean clustering on these images to label them as dusty or clean

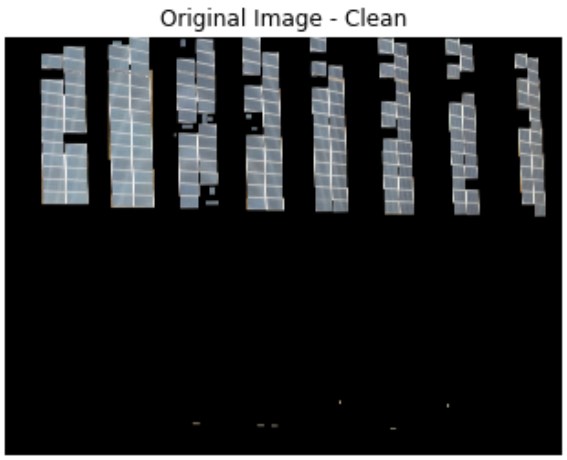


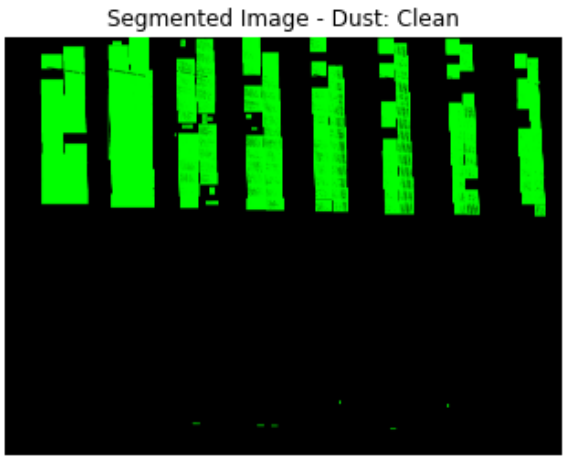


These are some examples of images labeled as clean

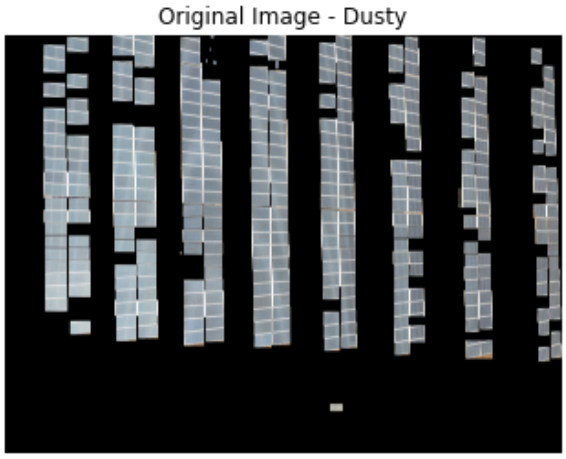


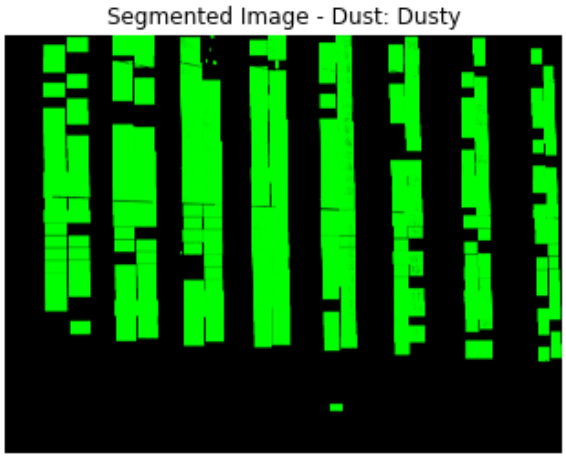


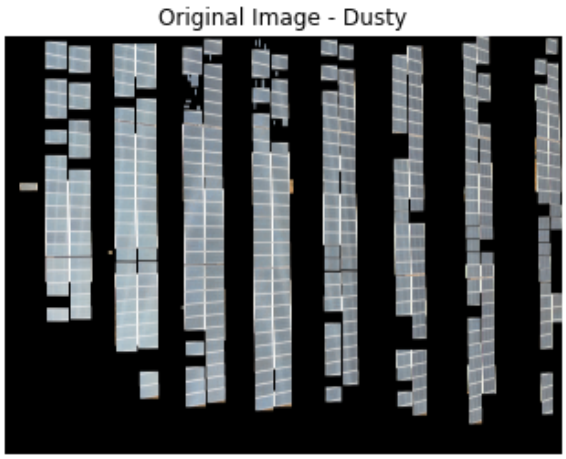


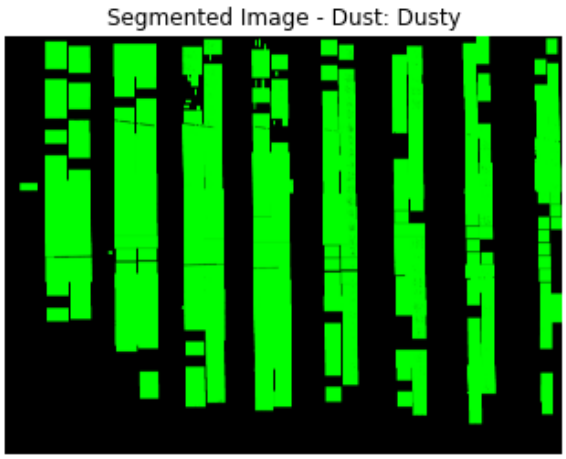


These are some examples of images labeled as dusty









4. Using this labeled data apply classification using SVM model

Portion of code used

- For segmentation

```
import os
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt


# Path to the folder containing the images
data_dir = "solar_panel_data"
# Create the "ML_cropped" folder if it doesn't exist
ML_cropped_dir = "segmented_images"
os.makedirs(ML_cropped_dir, exist_ok=True)

# Function to classify an image, return segmented image and bounding boxes
def classify_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    min_contour_area = 1000
    solar_panels = [cnt for cnt in contours if cv2.contourArea(cnt) >
min_contour_area]

    segmented_image = np.zeros_like(image)
    segmented_image[:] = (0, 255, 0)
    cv2.drawContours(segmented_image, solar_panels, -1, (0, 0, 0), -1)

    if len(solar_panels) > 0:
        return True, segmented_image, solar_panels
    else:
        return False, segmented_image, []


# Iterate through the images in the folder, classify, and save cropped
solar panels
for filename in os.listdir(data_dir):
    if filename.endswith(".JPG"):
        image_path = os.path.join(data_dir, filename)

        # Load the image
        image = cv2.imread(image_path)
```

```
        # Classify the image and get the segmented image and bounding
boxes
        has_solar_panel, segmented_image, solar_panels =
classify_image(image)

        if has_solar_panel:
            # Create a separate image for all the segmented panels
            all_panels_image = np.zeros_like(image)
            for panel in solar_panels:
                x, y, w, h = cv2.boundingRect(panel)
                all_panels_image[y:y+h, x:x+w] = image[y:y+h, x:x+w]

            # Define the path to save the image with all segmented panels
            save_path = os.path.join(ML_cropped_dir,
f"{filename}_all_panels.jpg")
            cv2.imwrite(save_path, all_panels_image)

        # Visualize the image and classification
        plt.figure(figsize=(8, 4))
        plt.subplot(121)
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title(f"Image: {filename}")
        plt.axis('off')

        plt.subplot(122)
        plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
        if has_solar_panel:
            plt.title("Image Highlighted showing panel")
        else:
            plt.title("Class: No Solar Panel")
        plt.axis('off')

        plt.show()
```

- For clustering

```
import numpy as np
from sklearn.cluster import KMeans
import cv2
```

```
import os
import matplotlib.pyplot as plt

# Function to extract features from an image (e.g., mean color)
def extract_features(image_path):
    image = cv2.imread(image_path)
    # Convert BGR to RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Calculate the mean color of the image
    mean_color = np.mean(image, axis=(0, 1))
    return mean_color

# Function to segment the image and label dust presence
def segment_and_label_dust(image_path, cluster_to_class, kmeans):
    image = cv2.imread(image_path)
    # Convert BGR to RGB
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply thresholding to segment dust
    _, binary_mask = cv2.threshold(gray, 100, 255,
cv2.THRESH_BINARY)  # Adjust threshold as needed

    # Create a colored mask for dust regions
    dust_mask = np.zeros_like(image_rgb)
    dust_mask[binary_mask == 255] = [0, 255, 0]  # Set dust regions
to green (you can change the color)

    # Overlay labels on the segmented image
    cluster_label =
cluster_to_class.get(kmeans.predict([extract_features(image_path)])[
0], "Unknown")

    return image_rgb, dust_mask, cluster_label

# Path to the folder containing the solar panel images
data_dir = "segmented_images"

# Create an empty list to store the features of each image
features = []
```

```
predicted_labels = []  # Initialize the predicted_labels list

# Create a dictionary to map cluster labels to class names
cluster_to_class = {0: "Clean", 1: "Dusty"}

# Create a dictionary to store the top ten images for each class
top_ten_images = {0: [], 1: []}

# Iterate through the images in the folder
for filename in os.listdir(data_dir):
    if filename.endswith(".jpg"):
        image_path = os.path.join(data_dir, filename)
        features.append(extract_features(image_path))

# Check if images were found
if len(features) == 0:
    print("No images found in the data directory.")
else:
    # Convert the list of features to a NumPy array
    features = np.array(features)

    # Number of clusters (classes) for K-Means
    num_clusters = 2

    # Apply K-Means clustering to classify the images
    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    kmeans.fit(features)

    # Iterate through the images again to display segmented images
with dust labels
    for i, filename in enumerate(os.listdir(data_dir)):
        if filename.endswith(".jpg"):
            image_path = os.path.join(data_dir, filename)
            image_rgb, dust_mask, cluster_label =
segment_and_label_dust(image_path, cluster_to_class, kmeans)
            predicted_class = cluster_to_class[kmeans.labels_[i]]
            predicted_labels.append(predicted_class)

            # Store the segmented image in the top_ten_images
dictionary
```

```
            class_label = cluster_to_class[kmeans.labels_[i]]
            if len(top_ten_images[kmeans.labels_[i]]) < 10:
                top_ten_images[kmeans.labels_[i]].append((filename,
image_rgb, dust_mask, cluster_label))

    # Display the top ten segmented images for each class
    for i in range(num_clusters):
        print(f"Top ten segmented images for class
'{cluster_to_class[i]}':")
        for filename, image_rgb, dust_mask, cluster_label in
top_ten_images[i]:
            plt.figure(figsize=(12, 5))
            plt.subplot(1, 2, 1)
            plt.imshow(image_rgb)
            plt.title(f"Original Image - {cluster_label}")
            plt.axis('off')
            plt.subplot(1, 2, 2)
            plt.imshow(dust_mask)
            plt.title(f"Segmented Image - Dust: {cluster_label}")
            plt.axis('off')
            plt.show()
```

- For training the model

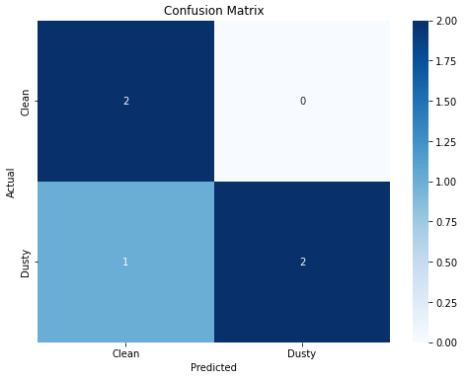
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have 'features' and 'predicted_labels' from your previous
code
# 'features' contain the mean color features
# 'predicted_labels' contain the cluster labels from K-Means (0 for
"Clean" and 1 for "Dusty")

# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(features,
predicted_labels, test_size=0.2, random_state=42)
```

```
# Initialize an SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)

# Train the SVM model on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# Calculate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=["Clean", "Dusty"], yticklabels=["Clean", "Dusty"])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

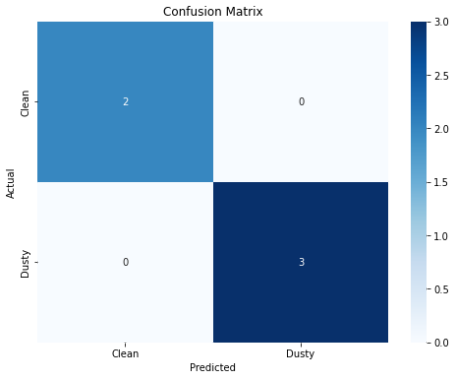
Results

The confusion matrix and accuracy for different kernel functions and regularization parameter(C)-

- For linear kernel
  C=1.0
  Accuracy=0.8

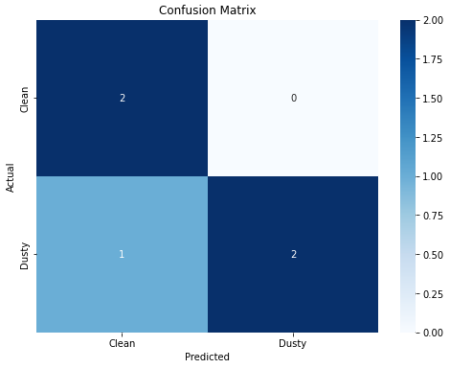


For C=0.02
Accuracy=1.0

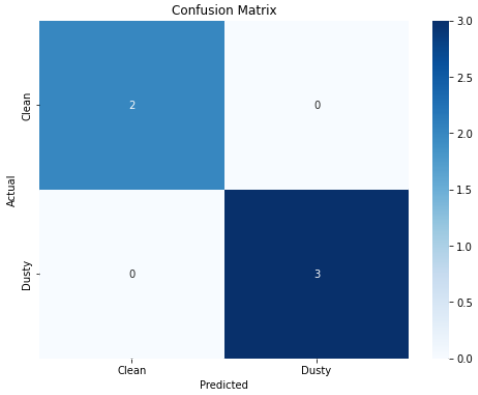


- For Radial bias function as Kernel function

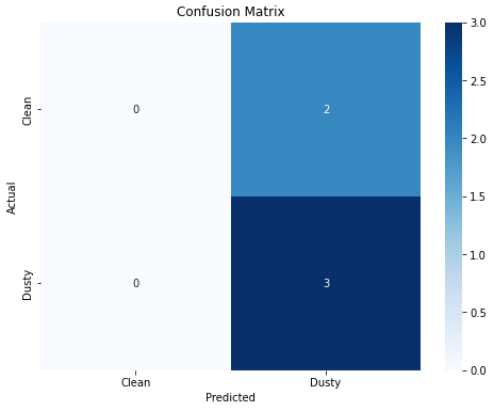
For C=1.0
Accuracy=0.8



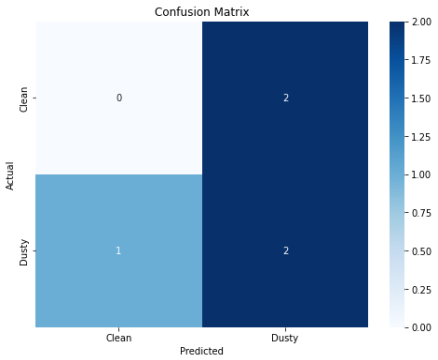
For C=0.1
Accuracy=1.0



- For sigmoid function
  C=1.0
  Accuracy=0.6



C=100
Accuracy=0.4



Here even for values of C that are very close to 0 the accuracy doesn't improve

suggesting that the sigmoid function is not a good choice for this application.