MINI PROJECT 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

25th October 2023

Abstract

This report presents the results of a machine learning classification project that aimed to classify images extracted from a video using a Support Vector Machine (SVM) model. The project's objective was to develop a robust system for distinguishing between two distinct categories: cloud-covered sky and background.

The project encompassed several key phases, including video frame extraction, image color segmentation, labeling of data, SVM model training, and the application of the model to classify each frame into cloudy sky and background . The SVM model was chosen for its effectiveness in handling complex image data and its suitability for binary classification tasks.

The report outlines the methodology employed, including data preprocessing techniques, feature extraction, and model training parameters. It presents the findings of the classification process, including accuracy and performance metrics, and discusses insights gained from the analysis of results.

Additionally, this report highlights the challenges encountered during the project, such as data labeling, and provides insights into potential improvements and future directions for the classification system.

Ultimately, the successful implementation of the SVM-based classification model demonstrates its potential for real-world applications, such as weather monitoring and video analytics, where the ability to distinguish between cloud and clear sky conditions is of paramount importance.

Introduction

This project involves several crucial stages, including extracting frames from the video,applying color based segmentation on each frame,augmenting these segmented frames, carefully labeling the data, and training a strong Support Vector Machine (SVM) model. The SVM model, known for its effectiveness in handling complex image data and making binary classifications, plays a central role in this classification project.

In this report, we will detail the methodological approach we took, which encompasses data preprocessing methods, techniques for extracting relevant features, and the process of finding the most suitable parameters for the SVM model. Additionally, we will share insights gained from the classification results, emphasizing the SVM model's precision in distinguishing between cloudy and clear sky conditions.

Furthermore, this report will delve into the challenges we faced, particularly those related to the labeling of data, and provide glimpses of potential ways to enhance and expand this classification system. The successful implementation of the SVM-based model highlights its potential usefulness in real-world applications, such as weather monitoring and video analytics, where accurately identifying cloud cover versus clear sky conditions is of significant value.

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 cloudy and clear sky conditions.

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.

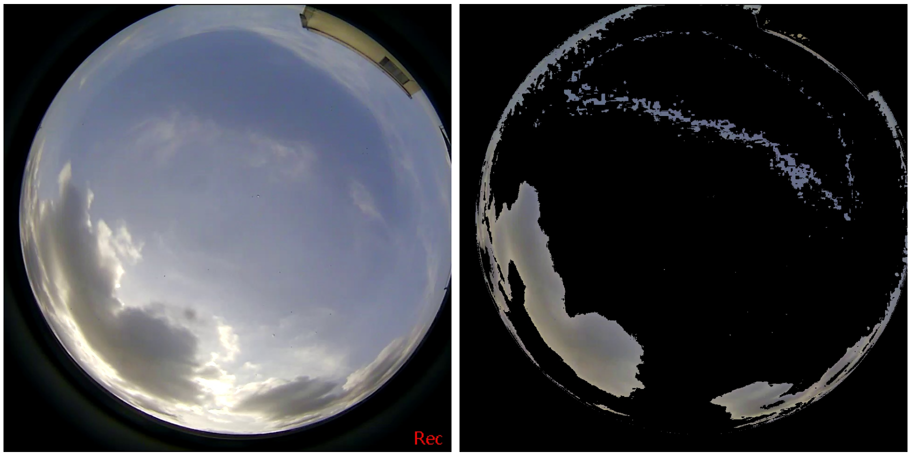
The SVM model is trained to minimize classification errors while also considering a regularization parameter (C) to control the trade-off between maximizing the margin and minimizing misclassifications. SVMs are known for their ability to handle high-dimensional data and effectively deal with outliers.

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.

<u>Different Steps taken</u>

1. First of all collected different frames from the provided video at the rate of 30 sec per frame
2. The applied color based segmentation on these images in which labeled two different classes in each frame named cloud coverage and background.And stored the percentage coverage of cloud and background of each frame in the name of image to compare it with final results.
   Ex-



3. Then applied Augmentation on these segmented images to make our dataset more diverse making 3 copies of each image at 90 deg rotation.
4. Finally trained and tested the model for these two classes taking three different values of C using feature extraction from each segmented image i.e. regularization parameter and stored the model in .pkl file.
5. Got the percentage of cloud and background for each image as a result for each of these models.

Portion of code used

1. For getting frames

```
import cv2
import os

# Path to the input video file
video_file = 'Data.avi'

# Create a directory for storing frames if it doesn't exist
output_dir = 'raw-frames'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Open the video file
cap = cv2.VideoCapture(video_file)

# Initialize variables
frame_count = 0
frame_rate = 30  # Number of seconds per frame
```

```
while True:
    # Read a frame from the video
    ret, frame = cap.read()

    # If the video has ended, break the loop
    if not ret:
        break

    # Calculate the time in seconds
    time_in_seconds = frame_count / cap.get(cv2.CAP_PROP_FPS)

    # Check if it's time to save a frame (every 10 seconds)
    if time_in_seconds % frame_rate == 0:
        # Construct the output filename
        output_file = os.path.join(output_dir,
f'frame_{frame_count}.jpg')

        # Save the frame as an image
        cv2.imwrite(output_file, frame)

    # Increment the frame count
    frame_count += 1

# Release the video capture object and close any open windows
cap.release()
cv2.destroyAllWindows()

print(f"Frames extracted and saved in '{output_dir}'")
```

2. For segmentation

```
import os
import cv2
import numpy as np

def extract_image_features(image_path):
    # Read the image
    image = cv2.imread(image_path)
```

```
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Calculate the average color of the sky (blue)
    average_sky_color = np.mean(image[0:100, 0:100], axis=(0, 1))

    # Apply Canny edge detection to detect edges in the image
    edges = cv2.Canny(gray, threshold1=30, threshold2=120)

    # Calculate the total number of edge pixels
    total_edge_pixels = np.sum(edges > 0)

    # Calculate the percentage of edge pixels in the image
    edge_pixel_percentage = (total_edge_pixels / (image.shape[0] *
image.shape[1])) * 100

    # Extract additional features as needed, such as texture, shape,
or cloud cover percentage

    # For example, you could use color segmentation to estimate
cloud cover percentage

    # Define regions with blue color (sky)
    lower_blue = np.array([90, 100, 100], dtype=np.uint8)
    upper_blue = np.array([140, 255, 255], dtype=np.uint8)

    # Create a mask for the blue regions
    mask = cv2.inRange(image, lower_blue, upper_blue)

    # Calculate the cloud cover percentage
    cloud_cover_percentage = (np.sum(mask > 0) / (image.shape[0] *
image.shape[1])) * 100

    # Return the extracted features as a dictionary
    features = {
        "average_sky_color": average_sky_color,
        "edge_pixel_percentage": edge_pixel_percentage,
        "cloud_cover_percentage": cloud_cover_percentage,
    }
```

```
    return features

# Define the folder containing the images
image_folder = "raw-frames"

# Create the output directory if it doesn't exist
output_folder = "segmented_images"
os.makedirs(output_folder, exist_ok=True)

# Iterate through the images in the folder
i=0
for filename in os.listdir(image_folder):
    if filename.endswith(('.jpg')):
        image_path = os.path.join(image_folder, filename)
        image_features = extract_image_features(image_path)

        # Read the image for segmentation
        image = cv2.imread(image_path)

        # Perform segmentation (for example, using color
segmentation)
        lower_blue = np.array([90, 100, 100], dtype=np.uint8)
        upper_blue = np.array([140, 255, 255], dtype=np.uint8)
        mask = cv2.inRange(image, lower_blue, upper_blue)
        segmented_image = cv2.bitwise_and(image, image, mask=mask)

        # Extract the cloud cover percentage from features
        cloud_cover_percentage =
int(image_features["cloud_cover_percentage"])

        # Construct the new filename with label and percentage of
cloud
        new_filename =
f"label_cloud{i}_{cloud_cover_percentage}.jpg"
        i=i+1

        # Save the segmented image in the output directory with the
new filename
        output_path = os.path.join(output_folder, new_filename)
        cv2.imwrite(output_path, segmented_image)
```

```
# Print a message indicating the completion
print("Segmented images saved in the 'segmented_images' folder with
label and percentage of cloud as names.")
```

3. Augmentation

```
import os
import cv2

# Define the input folder containing the segmented images
segmented_image_folder = "segmented_images"

# Create the output directory for augmented images if it doesn't
exist
augmented_image_folder = "augmented_images"
os.makedirs(augmented_image_folder, exist_ok=True)

# Function to apply image augmentation
def augment_image(image):
    # Define the augmentation operations you want to apply here.
    # For example, you can apply operations like rotation, flipping,
brightness adjustment, etc.
    # Here, we'll demonstrate rotation and horizontal flipping as
examples.

    # Rotate the image (you can specify the angle)
    rotated_image = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)

    # Horizontal flip the image
    flipped_image = cv2.flip(image, 1)

    return [rotated_image, flipped_image]

# Iterate through the segmented images in the folder
for filename in os.listdir(segmented_image_folder):
    if filename.endswith('.jpg'):
        segmented_image_path = os.path.join(segmented_image_folder,
filename)

        # Read the segmented image
```

```
        segmented_image = cv2.imread(segmented_image_path)

        # Augment the image
        augmented_images = augment_image(segmented_image)

        # Save augmented images with new filenames
        for i, augmented_image in enumerate(augmented_images):
            augmented_filename = f"{filename.replace('.jpg',
'')}%augmented_{i}.jpg"
            augmented_image_path =
os.path.join(augmented_image_folder, augmented_filename)
            cv2.imwrite(augmented_image_path, augmented_image)

# Print a message indicating the completion
print("Augmented images saved in the 'augmented_images' folder.")
```

4. Model testing and training(in the last part just replace the relative path of image to get its %cloud pixel percentage)

```
import cv2
import numpy as np
import os
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

# Define the folder containing your segmented images.
IMAGE_FOLDER = 'augmented_images'

# Set the threshold for classifying as "cloud" or "background."
threshold = 128

# Define the blue color range for cloud detection
lower_blue = np.array([100, 50, 50], dtype=np.uint8)
upper_blue = np.array([140, 255, 255], dtype=np.uint8)


# here change the relative path to get the cloud coverage of any
image
```

```python
sample_image =
cv2.imread('augmented_images\label_cloud1_8_%cloud_91_%background%au
gmented_0.jpg', cv2.IMREAD_COLOR)


def preprocess_image(image):
    if image is not None:
        # Create a mask for the blue regions in the image
        hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv_image, lower_blue, upper_blue)

        # Create "cloud" and "background" images based on the mask
        cloud = cv2.bitwise_and(image, image, mask=mask)
        background = cv2.bitwise_and(image, image, mask=1 - mask)

        return cloud, background
    else:
        return None, None

def calculate_coverage(cloud_image, background_image):
    total_pixels = cloud_image.size + background_image.size
    cloud_coverage = (cloud_image.size / total_pixels) * 100
    background_coverage = 100 - cloud_coverage
    return cloud_coverage, background_coverage

# Load and preprocess the dataset
cloud_images = []
background_images = []

for filename in os.listdir(IMAGE_FOLDER):
    image = cv2.imread(os.path.join(IMAGE_FOLDER, filename),
cv2.IMREAD_COLOR)

    cloud_image, background_image = preprocess_image(image)

    if cloud_image is not None and background_image is not None:
        cloud_images.append(cloud_image.flatten())
        background_images.append(background_image.flatten())

# Create labels for cloud and background (corrected)
```

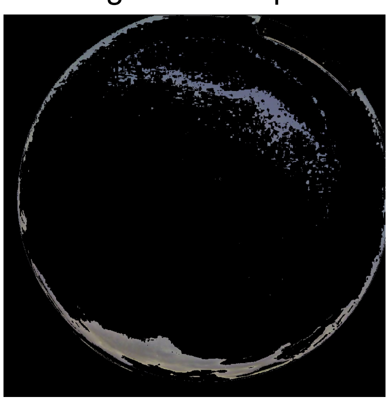
```
labels = [0] * len(cloud_images) + [1] * len(background_images)  # 
Swap the labels for cloud and background

# Combine the data for training
X = cloud_images + background_images

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, 
test_size=0.2, random_state=42)

# Create an SVM classifier with a custom C value
custom_C = 1.0  # You can adjust this value as needed
clf = SVC(C=custom_C, kernel='linear')

# Train the SVM model
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print('Accuracy:', accuracy)

# Print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)


if sample_image is not None:
    # Create a mask for the blue regions in the sample image
    hsv_sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2HSV)
    sample_mask = cv2.inRange(hsv_sample_image, lower_blue, 
upper_blue)

    # Calculate the percentage coverage of each class based on the 
number of pixels
```

```
    cloud_pixels = np.sum(sample_mask)  # Count the number of 
"cloud" pixels in the mask
    total_pixels = sample_image.size  # Total number of pixels in 
the image

    cloud_coverage = (cloud_pixels / total_pixels) * 100
    background_coverage = 100 - cloud_coverage

    # Print the results
    print('Cloud coverage:', cloud_coverage, '%')
    print('Background coverage:', background_coverage, '%')
    print(sample_image)
else:
    print('Sample image not loaded or invalid.')
```

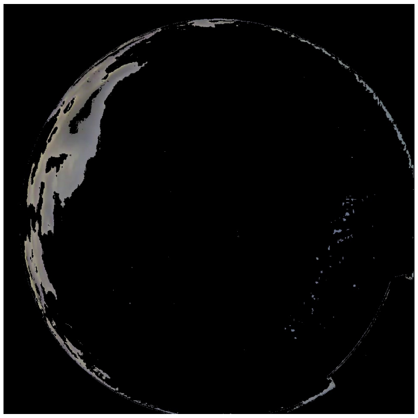
5. Cloud coverage percentage is added in the image name of each sample

Results

- The cloud coverage percentage of each image is added in the name of image after segmentation part Ex-

  For this image the relative path is "segmented_images\label_cloud0_11.jpg" and its cloud coverage percentage is 11%.
- For getting the Cloud pixel percentage, in the model just add the relative path in as the sample_image and in the end the code will give the result Ex-



For this image the result of cloud pixel percentage -
`Cloud coverage: 4.095052083333333 % Background coverage: 95.90494791666667 %`(for model with C=1.0 and linear kernel)

- Results for different kernel functions
    1. For Sigmoid kernel function
        - C=0.1, test train split=0.2
          `Accuracy: 0.5384615384615384 Confusion Matrix: [[6 0] [6 1]]`
        - `C=0.15` test train split=0.2
          `Accuracy: 0.9230769230769231 Confusion Matrix: [[6 0] [1 6]]`
        - `C=0.2 test train split=0.2`
          `Accuracy: 1.0 Confusion Matrix: [[6 0] [0 7]]`
    2. For linear kernel function
        - For this kernel function the accuracy remains 1.0 for a very wide range of C which suggests that this model is classifying the data set with high efficiency.
        - C=1.0, test train split=0.2
          `Accuracy: 1.0 Confusion Matrix: [[6 0] [0 7]]`
    3. For RBF kernel function
        - C=0.05, test train split=0.2
          `Accuracy: 0.46153846153846156 Confusion Matrix: [[6 0] [7 0]]`
        - C=0.07, test train split=0.2
          `Accuracy: 1.0 Confusion Matrix: [[6 0] [0 7]]`
        - C=0.1, test train split=0.2
          `Accuracy: 0.9230769230769231 Confusion Matrix: [[5 1] [0 7]]`