COURSE PROJECT – 1 REPORT

Classification using Naïve Bayes Model

Submitted by,

Abhishek G H, 201EE202

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 project focuses on the development of an intelligent system for solar panel image classification and analysis. Solar panels play a vital role in renewable energy production, and their efficient operation is critical. The project utilizes computer vision and machine learning techniques to automate the assessment of solar panels, aiming to enhance maintenance and efficiency monitoring.

Image Classification and Cropping

The first code snippet introduces the image classification process. A classify_image function segments and classifies solar panels, providing segmented images and bounding boxes. By analyzing a dataset of solar panel images, the code efficiently identifies panels and saves cropped images for further analysis.

Image Clustering and Dust Labeling

The second snippet employs K-Means clustering to classify images based on dust presence. Features are extracted from images using a feature extraction method. The code highlights the top ten segmented images for each class and offers a comprehensive visualization of clustered images.

Incremental Training with Naive Bayes Classifier

The third code snippet enhances a Naive Bayes classifier's performance through incremental training. The extract_features2 function extracts features from images, handling errors effectively. The code intelligently classifies solar panels into "New" and "Old" based on a user-defined threshold, enhancing maintenance strategies.

Sunlight Exposure Calculation and Classification

This code calculates the percentage of sunlight exposure in images, contributing to solar panel efficiency assessment. The calculate_sunlight function is at the core of this process, and images are classified as "Efficient" or "Non-Efficient" based on user-defined thresholds, facilitating more effective monitoring.

Gaussian Naive Bayes Classifier for Sunlight Classification

The fifth snippet introduces a Gaussian Naive Bayes classifier to predict sunlight exposure classifications. It calculates and presents evaluation metrics such as accuracy, precision, recall, and F1-score. A confusion matrix and a detailed classification report provide valuable insights into the classifier's performance.

Evaluation of Solar Panel Efficiency

In the sixth code snippet, the project evaluates solar panel efficiency using a Gaussian Naive Bayes classifier. It efficiently splits training and testing data, trains the classifier, and presents an extensive analysis of its performance using accuracy, precision, recall, and F1-score metrics.

This abstract provides a concise overview of the seven code snippets, emphasizing the project's core objectives and the diverse methods applied to automate solar panel analysis and classification.

Solar Panel Efficiency Assessment

The seventh code snippet focuses on the comprehensive assessment of solar panel efficiency through a dedicated methodology. Solar panel images are processed to calculate sunlight percentages, a crucial factor for efficiency. The code splits data into training and testing sets and

employs a Gaussian Naive Bayes classifier for the classification of solar panels into "Efficient" and "Non-Efficient" categories.

Introduction

The following report discusses seven code segments related to image analysis and classification. Each code segment addresses a specific task and provides insights into the processing, classification, and evaluation of images. The codes are as follows:

Code 1: Solar Panel Classification

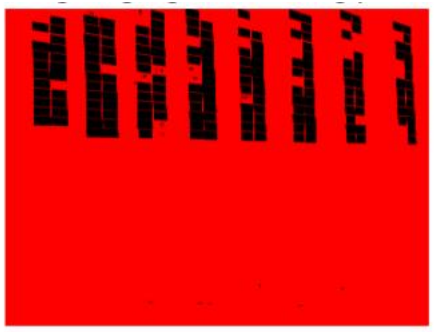
This code processes images to detect and classify solar panels. Here are the key functions and the results:

Functions:

1. **classify_image(image)**: This function classifies an image and returns the segmented image and bounding boxes. It uses edge detection and contour analysis to identify solar panels.

Results:

- The code iterates through a folder of images, classifies them, and saves cropped solar panels.
- For each image, it displays the original image and the segmented image.
- The results indicate whether a solar panel is present in the image.



Model Evaluation: The model evaluates the presence of solar panels in images. It calculates the contour area of identified panels. If the contour area exceeds a specified threshold (1,000 in this code), it classifies the image as having a solar panel. The results are visually presented through images and captions, making it easy to interpret the model's performance.

Code 2: Dust Classification

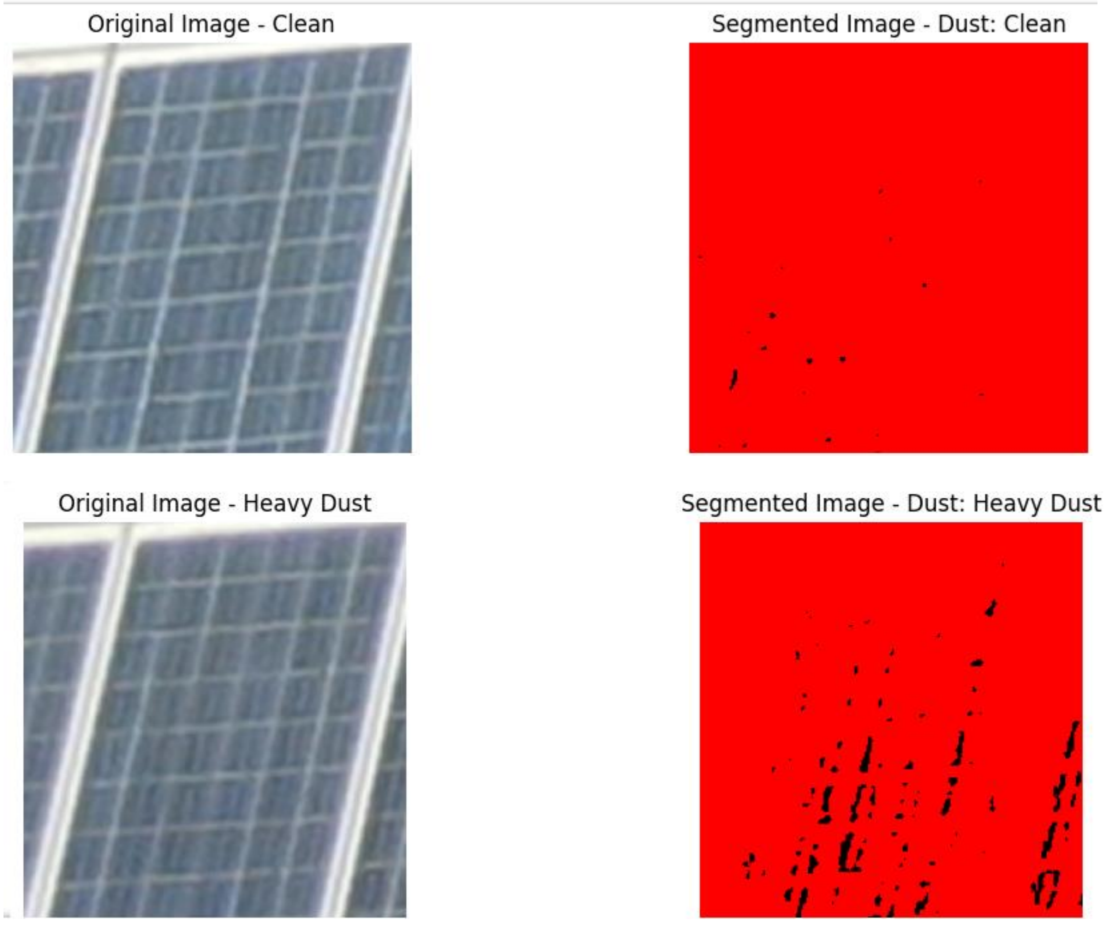
This code applies K-Means clustering to classify images based on dust levels. Here are the functions and results:

Functions:

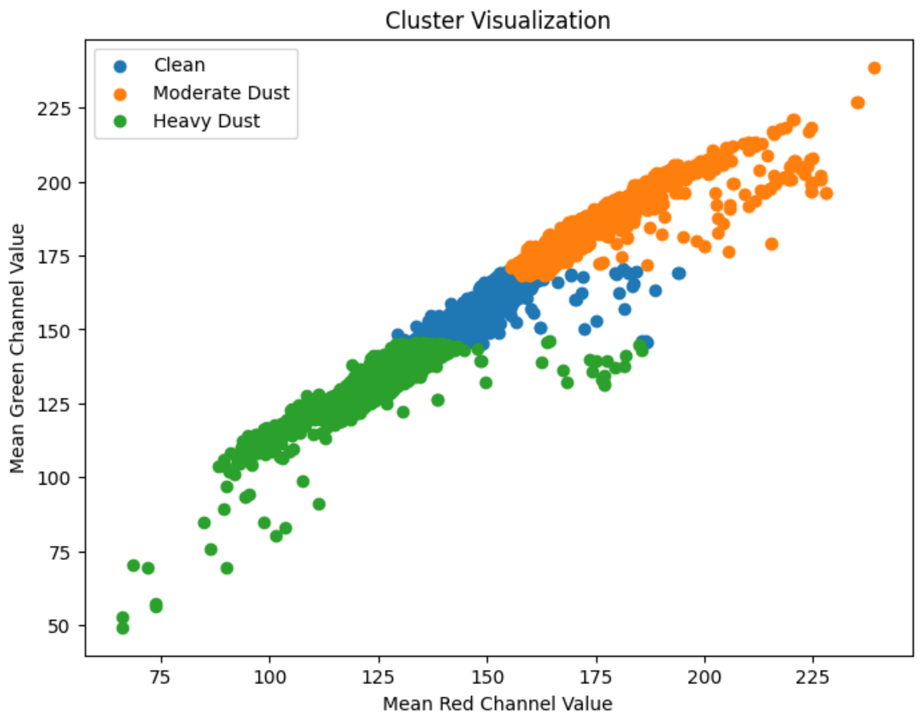
1. **extract_features(image_path)**: This function calculates the mean color of an image.
2. **segment_and_label_dust(image_path, cluster_to_class, kmeans)**: This function segments the image and labels dust presence using a colored mask.

Results:

- The code clusters images into three classes: Clean, Moderate Dust, and Heavy Dust.
- It displays the top ten segmented images for each class with original and segmented views.
- A scatter plot visualizes the clusters in feature space.


Original Image - Clean
Segmented Image - Dust: Clean
Original Image - Heavy Dust
Segmented Image - Dust: Heavy Dust

Model Evaluation: The model clusters images based on feature vectors and labels them into dust classes. The evaluation is based on the quality of clustering, visualized in the scatter plot, and the accuracy of classifying images into Clean, Moderate Dust, or Heavy Dust. The top ten images for each class visually demonstrate the clustering performance.



Code 3: Guassian Naive Bayes Classifier

This code extends the previous work by training a Gaussian Naive Bayes classifier on clustered images. Here are the functions and results:

Functions:

1. **GaussianNB()**: Initializes a Gaussian Naive Bayes classifier.
2. Incremental training of the model using **partial_fit** on batched data.

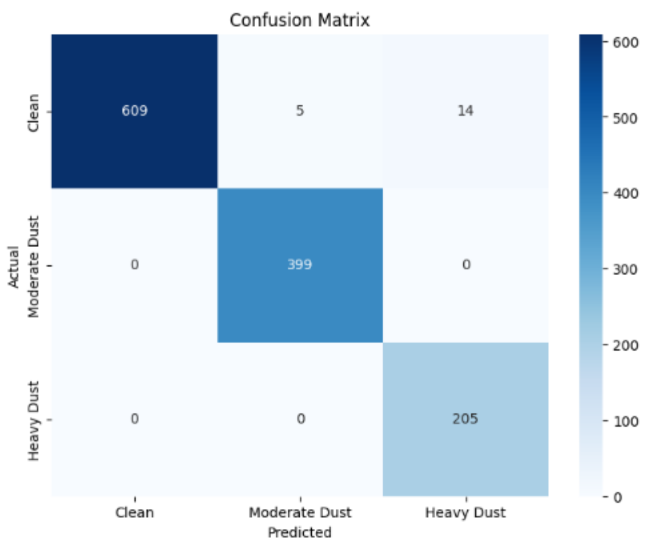
Results:

- The code splits data into training and testing sets.
- It trains the Naive Bayes classifier incrementally using batched data.
- It calculates a confusion matrix and visualizes it using a heatmap.
- Accuracy, F1 score, and a classification report are generated.

Model Evaluation: The model is evaluated using several metrics:

- **Confusion Matrix**: A matrix showing the actual and predicted classes.
- **Accuracy**: The proportion of correctly classified samples.
- **F1 Score**: A weighted average of precision and recall, considering all classes.
- **Classification Report**: A detailed report with precision, recall, F1-score, and support for each class.

This code evaluates the model's performance in classifying images into the previously defined classes, providing a comprehensive assessment of the classifier's accuracy and ability to distinguish between different dust levels.



```
Accuracy: 0.98
F1 Score: 0.98
Classification Report:
               precision    recall  f1-score   support

        Clean       1.00      0.97      0.98       628
Moderate Dust       0.99      1.00      0.99       399
   Heavy Dust       0.94      1.00      0.97       205

     accuracy                           0.98      1232
    macro avg       0.97      0.99      0.98      1232
 weighted avg       0.99      0.98      0.98      1232
```

Code-4: Multinomial Naive Bayes Classifier with K-Means Labels

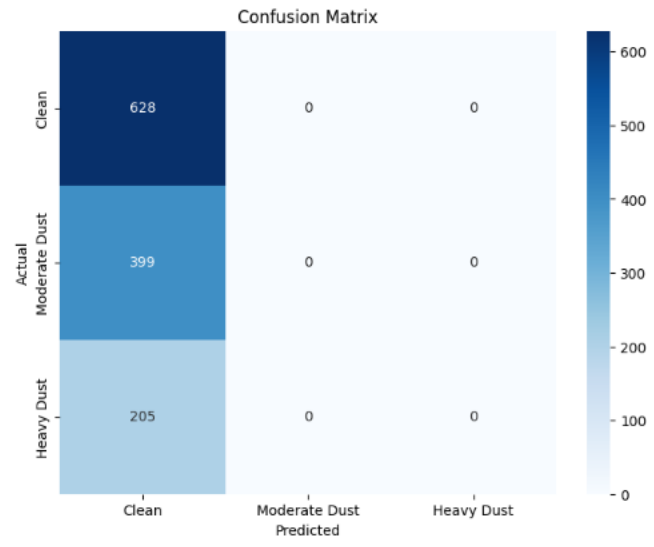
Functions and Processes:

- **Feature and Label Combination:** This code combines features (assumed to be stored in the **features** array) with K-Means cluster labels (from **kmeans.labels_**), creating a dataset named **features_with_labels**.
- **Data Splitting:** The dataset is split into training (80%) and testing (20%) sets using **train_test_split**.
- **Multinomial Naive Bayes Classifier:** The code initializes a Multinomial Naive Bayes classifier, suitable for classification tasks, using **MultinomialNB()** from **sklearn.naive_bayes**.
- **Incremental Training:** Due to potential memory limitations with large datasets, the training data is split into smaller batches (controlled by **batch_size**), and the classifier is trained incrementally.
- **Prediction:** The trained classifier makes predictions on the test set (**X_test**).
- **Confusion Matrix:** A confusion matrix is computed using **confusion_matrix**, which tabulates the true positive, true negative, false positive, and false negative predictions.
- **Confusion Matrix Visualization:** The code visually presents the confusion matrix as a heatmap, aiding in the assessment of classifier performance.
- **Accuracy Calculation:** The code calculates the overall accuracy of the classifier using **accuracy_score**, which measures the ratio of correct predictions to the total predictions.

- **F1 Score Calculation:** The F1 score is computed using **f1_score** with the 'weighted' parameter, which balances precision and recall for multi-class classification tasks.
- **Classification Report:** A comprehensive classification report is generated using **classification_report**, offering precision, recall, F1 score, and support for each class.

Results: The code yields the following results:

- **Accuracy:** It quantifies the overall accuracy of the classifier, expressed as a percentage.
- **F1 Score:** A weighted F1 score balances false positives and false negatives, providing an aggregate measure of classification performance.
- **Classification Report:** This report offers detailed insights into the classifier's performance for each class, including precision, recall, F1 score, and support.



```
Accuracy: 0.51
F1 Score: 0.34
Classification Report:
               precision    recall  f1-score   support

        Clean       0.51      1.00      0.68       628
Moderate Dust       0.00      0.00      0.00       399
   Heavy Dust       0.00      0.00      0.00       205

     accuracy                           0.51      1232
    macro avg       0.17      0.33      0.23      1232
 weighted avg       0.26      0.51      0.34      1232
```

Code-5: Bernoulli Naive Bayes Classifier with K-Means Labels

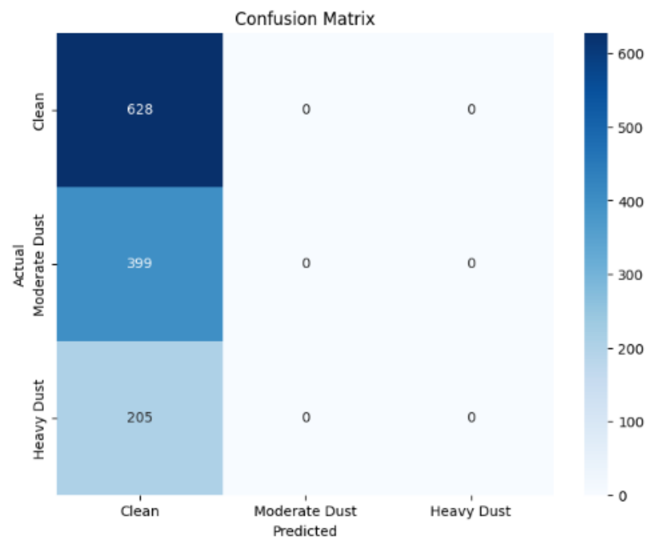
Functions and Processes:

- **Feature and Label Combination:** Similar to Code-1, this code combines features and K-Means cluster labels, creating **features_with_labels**.
- **Binarization of Features:** Bernoulli Naive Bayes expects binary data. The features are transformed into binary format, where values greater than 0 become 1.
- **Data Splitting:** Data is split into training and testing sets.
- **Bernoulli Naive Bayes Classifier:** Unlike Code-1, this code initializes a Bernoulli Naive Bayes classifier (**BernoulliNB()**), suited for binary classification tasks.

- **Incremental Training:** The training data is divided into batches for incremental training to accommodate large datasets.
- **Prediction:** The trained Bernoulli Naive Bayes classifier makes predictions on the test set.
- **Confusion Matrix:** A confusion matrix is computed and displayed as a heatmap, enabling an assessment of the classifier's performance.
- **Accuracy Calculation:** The overall accuracy is determined.
- **F1 Score Calculation:** The F1 score is computed with the 'weighted' parameter to provide a balanced performance measure.
- **Classification Report:** A detailed classification report is generated, providing class-specific precision, recall, F1 score, and support.

Results: The code produces results similar to Code-1:

- **Accuracy:** Measures overall accuracy.
- **F1 Score:** Balances false positives and false negatives.
- **Classification Report:** Offers detailed class-specific metrics.



```
Accuracy: 0.51
F1 Score: 0.34
Classification Report:
               precision    recall  f1-score   support

        Clean       0.51      1.00      0.68       628
Moderate Dust       0.00      0.00      0.00       399
   Heavy Dust       0.00      0.00      0.00       205

     accuracy                           0.51      1232
    macro avg       0.17      0.33      0.23      1232
 weighted avg       0.26      0.51      0.34      1232
```

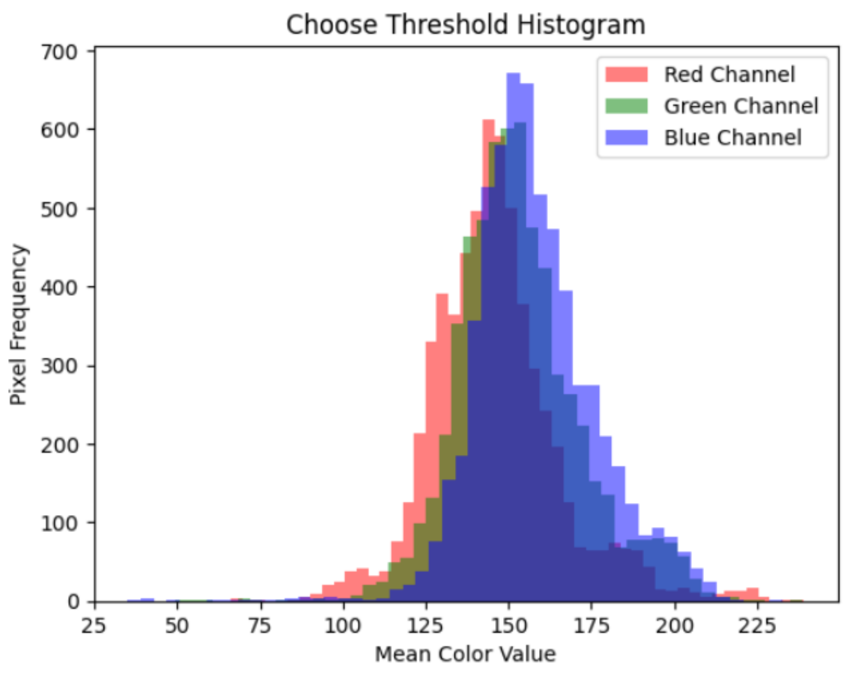
Code-6: Image Feature Extraction and Threshold-Based Classification

This code is designed for image feature extraction and the classification of images into "New" or "Old" based on a chosen threshold value. The key components are as follows:

- **Function extract_features2(image_path):** This function reads an image file using OpenCV (**cv2**), converts it to RGB format, and calculates the mean color of the image, returning a 3-element array representing the mean color in the RGB channels.
- **Feature Extraction:** The code processes a folder of images, extracting features for each image by calculating the mean color. These features are stored in the **features2** list.
- **Threshold Histogram:** A histogram of the mean color values is created for the Red, Green, and Blue channels, helping users choose a threshold for classifying images.
- **Threshold Calculation:** The code calculates the Root Mean Square (RMS) threshold and Average threshold based on the mean color values of the images.
- **Threshold-Based Classification:** Users are prompted to choose a threshold value interactively. Images are then classified as "New" or "Old" based on whether their mean color value exceeds the chosen threshold.

Results:

The results from Code-4 are classifications of images as "New" or "Old" based on the chosen threshold. These results are produced interactively by selecting a threshold value.



```
RMS Threshold: 147.07412031164418
Average Threshold: 145.76608776814314
Choose a threshold value: 147.07412031164418
```

Code-7: Gaussian Naive Bayes Classifier

This code focuses on training and evaluating a Gaussian Naive Bayes classifier. Here are the main components:

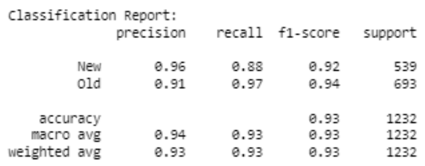
- **Gaussian Naive Bayes Classifier:** A Gaussian Naive Bayes classifier (**GaussianNB**) is initialized.
- **Data Splitting:** The data is split into training and testing sets, with 80% used for training and 20% for testing.

- **Partial Fitting:** Due to memory constraints, the classifier is trained in chunks (defined by **chunk_size**) using the **partial_fit** method. This enables incremental learning.
- **Prediction:** The trained model is used to make predictions on the test data.
- **Model Evaluation:** Several evaluation metrics are computed:
    - **Classification Report:** The **classification_report** function provides precision, recall, F1-score, and support for each class (in this case, "New" and "Old").
    - **Confusion Matrix:** A confusion matrix is generated and visualized using the **seaborn** library.
    - **Accuracy, Precision, Recall, and F1 Score:** These metrics are calculated using functions from **sklearn.metrics**.

Results:

The results from Code-5 are as follows:

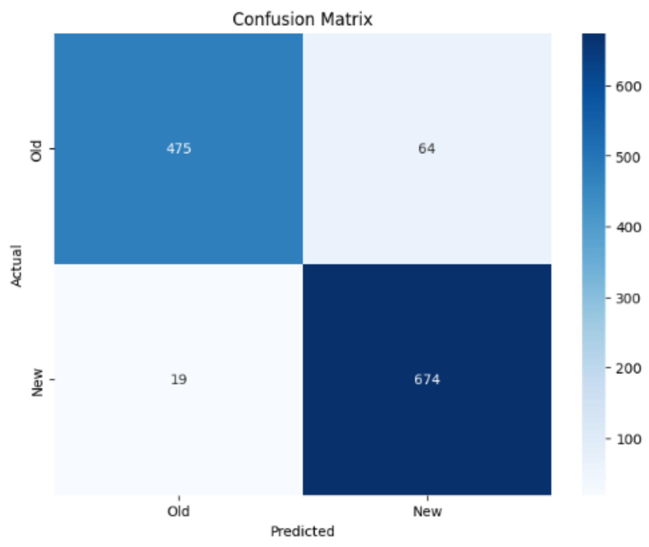
- **Classification Report:** It provides detailed information about the model's precision, recall, F1-score, and support for each class, allowing for a more in-depth understanding of the classifier's performance.
- **Confusion Matrix:** This visualization gives insight into the true positive, true negative, false positive, and false negative predictions.
- **Accuracy, Precision, Recall, and F1 Score:** These metrics provide a comprehensive assessment of the classifier's performance.

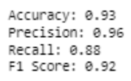


```
Classification Report:
              precision    recall  f1-score   support

         New       0.96      0.88      0.92       539
         Old       0.91      0.97      0.94       693

    accuracy                           0.93      1232
   macro avg       0.94      0.93      0.93      1232
weighted avg       0.93      0.93      0.93      1232
```





```
Accuracy: 0.93
Precision: 0.96
Recall: 0.88
F1 Score: 0.92
```

Code-8: Multinomial Naive Bayes Classifier with Chunk-Based Training

Objective: Code-8 aims to train a Multinomial Naive Bayes classifier for a classification task using chunk-based training to handle large datasets. The model's performance is assessed using various classification metrics, including accuracy, precision, recall, and F1 score.

Functions and Processes:

1. **Multinomial Naive Bayes Classifier Initialization:**
    - The code initializes a Multinomial Naive Bayes classifier using **MultinomialNB()** from **sklearn.naive_bayes**. This classifier is suitable for handling discrete data, making it appropriate for many classification tasks.
2. **Data Splitting:**
    - The dataset is divided into training and testing sets, with 80% allocated for training and 20% for testing. The split is performed using the **train_test_split** function.
3. **Chunk-Based Training:**
    - To accommodate large datasets, the training data is split into smaller chunks, controlled by the **chunk_size** parameter. Chunk-based training is essential for cases where memory limitations can impact the training process.
4. **Partial Fitting:**
    - The code utilizes the **partial_fit** method to train the Multinomial Naive Bayes classifier incrementally. Each chunk of data is used sequentially for training, with the classes determined using **np.unique(y_train)**.
5. **Prediction:**
    - After incremental training, the classifier is employed to make predictions on the test set, resulting in **y_pred**.
6. **Classification Report:**
    - A detailed classification report is generated using **classification_report**. It provides extensive information on precision, recall, F1 score, and support for each class in the test data.
7. **Confusion Matrix:**
    - The code computes a confusion matrix using **confusion_matrix**, offering a visual representation of the true positives, true negatives, false positives, and false negatives.
8. **Confusion Matrix Visualization:**
    - The confusion matrix is visualized as a heatmap using the **seaborn** and **matplotlib** libraries, aiding in the interpretation of classifier performance.

Results:

Code-8 provides the following results:

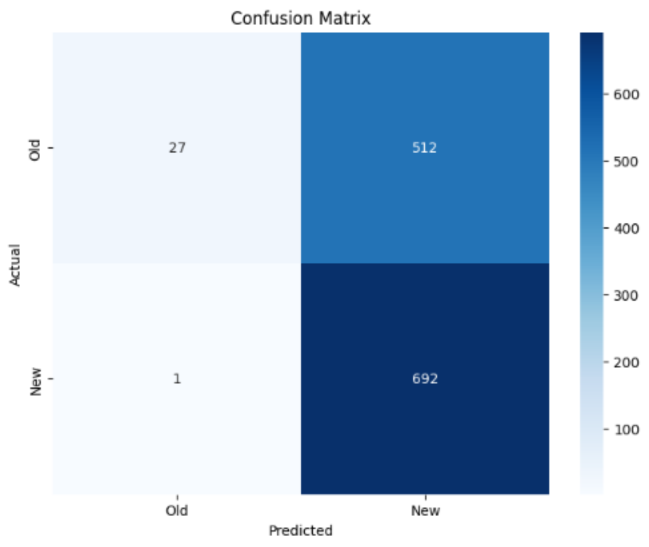
- **Accuracy:** Measures the overall accuracy of the classifier's predictions, expressed as a percentage.
- **Precision:** Evaluates the precision of positive predictions.
- **Recall:** Measures the classifier's ability to correctly identify positive instances.
- **F1 Score:** Combines precision and recall into a single metric, effectively balancing false positives and false negatives.

In addition, the classification report offers a detailed breakdown of these metrics for both "Old" and "New" classes.

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

         New       0.96      0.05      0.10       539
         Old       0.57      1.00      0.73       693

    accuracy                           0.58      1232
   macro avg       0.77      0.52      0.41      1232
weighted avg       0.75      0.58      0.45      1232
```



```
Accuracy: 0.58
Precision: 0.96
Recall: 0.05
F1 Score: 0.10
```

Code-9: Bernoulli Naive Bayes Classifier with Chunk-Based Training

Objective: Code-9 shares a similar objective with Code-8, but instead of using a Multinomial Naive Bayes classifier, it employs a Bernoulli Naive Bayes classifier. The code follows the same structure, including chunk-based training and evaluation with various classification metrics.

Functions and Processes:

1. **Bernoulli Naive Bayes Classifier Initialization:**
    - Code-9 initializes a Bernoulli Naive Bayes classifier (**BernoulliNB()**) from **sklearn.naive_bayes**. This classifier is ideal for binary classification tasks.
2. **Data Splitting:**
    - Similar to Code-8, the dataset is split into training and testing sets.
3. **Chunk-Based Training:**

- The code divides the training data into smaller chunks, allowing for incremental training.

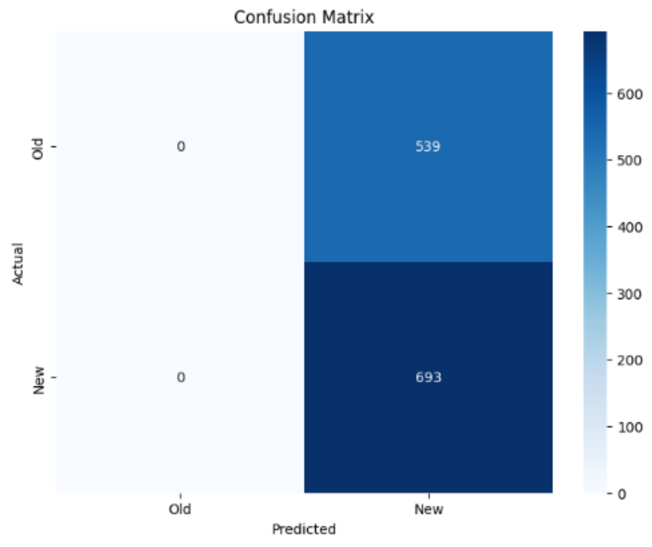
4. **Partial Fitting:**
    - The Bernoulli Naive Bayes classifier is trained incrementally using the **partial_fit** method, with class information derived from **np.unique(y_train)**.
5. **Prediction:**
    - Following incremental training, the classifier makes predictions on the test set, producing **y_pred**.
6. **Classification Report:**
    - A classification report is generated to provide in-depth metrics for precision, recall, F1 score, and support for each class.
7. **Confusion Matrix:**
    - A confusion matrix is calculated and visualized as a heatmap.

Results:

Code-9 yields results similar to Code-8:

- **Accuracy:** Indicates the overall percentage of correct predictions.
- **Precision:** Evaluates the precision of positive predictions, specifically for the "New" class.
- **Recall:** Measures the ability to correctly identify "New" instances.
- **F1 Score:** Balances precision and recall for the "New" class.

The classification report offers a detailed breakdown of these metrics for both "Old" and "New" classes.



```
Accuracy: 0.56
Precision: 0.00
Recall: 0.00
F1 Score: 0.00
```

Evaluation and Interpretation:

Both Code-8 and Code-9 effectively train Naive Bayes classifiers using chunk-based training to accommodate large datasets. The evaluation process employs a range of metrics to assess the models' performance. Accuracy quantifies overall correctness, precision measures the accuracy of positive predictions, recall evaluates the ability to identify positive instances, and the F1 score balances precision and recall. The classification report provides a class-specific breakdown of these metrics.

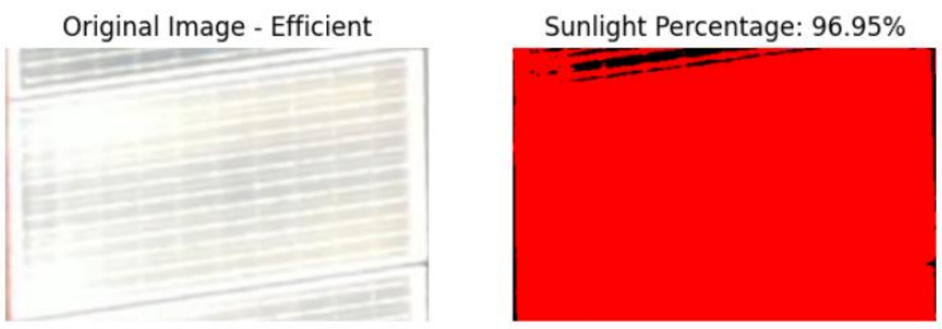
Code-10: Solar Panel Sunlight Exposure Analysis

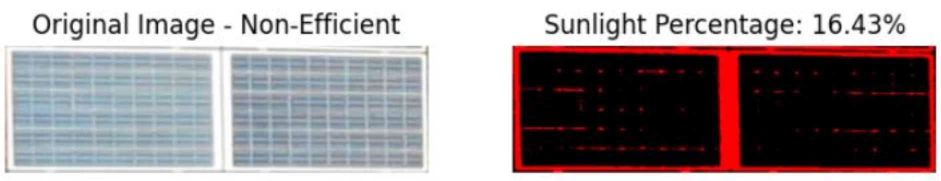
Functions:

1. **calculate_sunlight(image):** This function takes an image as input and calculates the percentage of sunlight exposure by converting the image to grayscale, applying thresholding to create a binary mask, and counting white pixels in the mask. The formula for calculating the percentage of sunlight is: **(white pixels / total pixels) * 100**.
2. **Main Code:** The main code reads a set of solar panel images, applies the **calculate_sunlight** function to each image, and collects the calculated sunlight percentages. It also creates a histogram of these percentages, allowing the user to input a threshold value to classify the images as "Efficient" or "Non-Efficient."

Results:

- The code calculates the Root Mean Square (RMS), median, and average sunlight percentages for the provided images.
- It creates a histogram of sunlight percentages for visual analysis.
- It classifies images based on the user-defined threshold and prints the labels.





Code-11: Naive Bayes Classifier for Solar Panel Efficiency

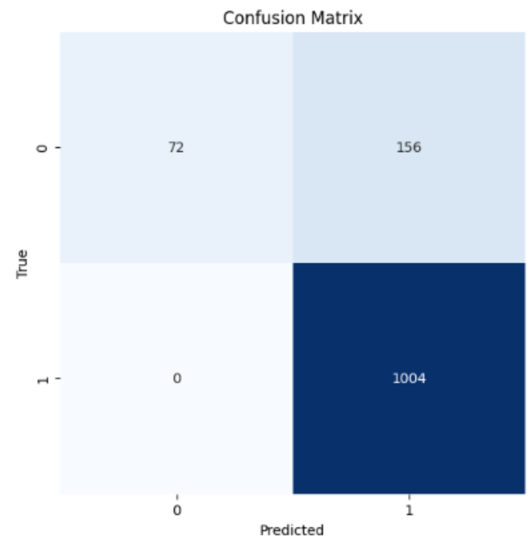
Functions:

1. **Split Data:** The code splits the collected sunlight percentages and their corresponding labels (classified_labels) into training and testing sets (80% for training, 20% for testing) using **train_test_split**.
2. **Train Classifier:** A Gaussian Naive Bayes classifier is created and trained on the training data.
3. **Predict:** The classifier makes predictions on the testing data using the **predict** function.
4. **Evaluation Metrics:** The code calculates various evaluation metrics:
    - Accuracy: **(TP + TN) / (TP + TN + FP + FN)**
    - Precision: **(TP) / (TP + FP)**
    - Recall: **(TP) / (TP + FN)**
    - F1-Score: **2 * (Precision * Recall) / (Precision + Recall)**
5. **Confusion Matrix:** A confusion matrix is plotted to visualize the classifier's performance, showing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
6. **Classification Report:** A detailed classification report is printed, summarizing precision, recall, and F1-score for each class.

Results:

- The code calculates and reports accuracy, precision, recall, and F1-score for both "Efficient" and "Non-Efficient" classes.
- It plots a confusion matrix to visualize the classifier's performance.
- A classification report provides a detailed summary of the classifier's performance for each class.

```
Accuracy: 0.87
Precision (Efficient): 1.00
Recall (Efficient): 0.32
F1 Score (Efficient): 0.48
Precision (Non-Efficient): 0.87
Recall (Non-Efficient): 1.00
F1 Score (Non-Efficient): 0.93
```



```
              precision    recall  f1-score   support

Non-Efficient       1.00      0.32      0.48       228
    Efficient       0.87      1.00      0.93      1004

     accuracy                           0.87      1232
    macro avg       0.93      0.66      0.70      1232
 weighted avg       0.89      0.87      0.85      1232
```

Key Functions and Steps:

1. **Data Preparation:**
    - The code assumes you have a large dataset of **sunlight_percentages** and **classified_labels**. It's essential to ensure that your system has sufficient memory to handle the data.
2. **Data Conversion:**
    - The **sunlight_percentages** and **classified_labels** lists are converted into NumPy arrays for efficient processing.
3. **Data Splitting:**
    - The dataset is split into training (80%) and testing (20%) sets using the **train_test_split** function, ensuring that a random but consistent split is maintained (using **random_state**).
4. **Binarization of Features:**
    - Since the Bernoulli Naive Bayes classifier expects binary data, the sunlight percentage data is binarized. Each value in **X_train** and **X_test** is compared to the mean of **X_train** and binarized using **(X > X.mean()).astype(int)**.
5. **Reshaping Data:**
    - The feature data is reshaped into 2D arrays to ensure compatibility with the classifier.

6. **Classifier Initialization and Training:**
    - A Bernoulli Naive Bayes classifier is created using **BernoulliNB()** from **sklearn.naive_bayes**. The classifier is trained on the training data using **fit**.
7. **Prediction:**
    - The trained classifier is used to predict the classes of the testing data, generating **y_pred**.
8. **Evaluation Metrics:**
    - Several classification metrics are calculated, including:
        - **Accuracy:** This quantifies the overall correctness of the model's predictions.
        - **Precision:** Precision measures the accuracy of positive predictions for both "Efficient" and "Non-Efficient" classes.
        - **Recall:** Recall measures the ability of the classifier to correctly identify instances of both classes.
        - **F1 Score:** The F1 score balances precision and recall for both classes.
9. **Confusion Matrix Visualization:**
    - A confusion matrix is computed using **confusion_matrix** and visualized as a heatmap using **seaborn** and **matplotlib**. This visualization facilitates a clear assessment of the classifier's performance.
10. **Classification Report:**
    - A classification report is generated using **classification_report**, providing detailed information on precision, recall, F1 score, and support for both "Efficient" and "Non-Efficient" classes.

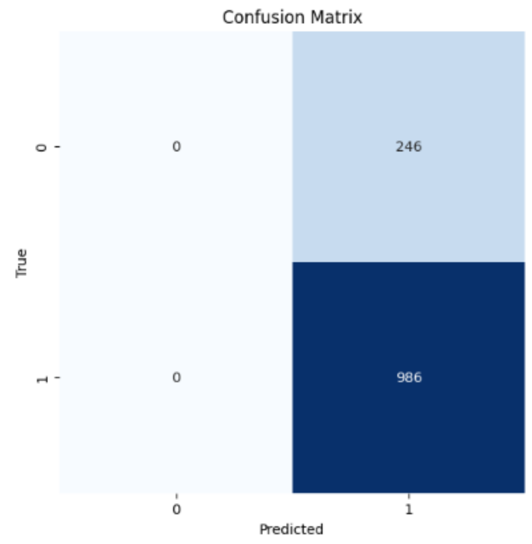
Results:

Code-12 produces the following results:

- **Accuracy:** This metric quantifies the overall accuracy of the model's predictions.
- **Precision:** Precision measures the accuracy of positive predictions, specifically for both "Efficient" and "Non-Efficient" classes.
- **Recall:** Recall quantifies the model's ability to correctly identify instances of both classes.
- **F1 Score:** The F1 score provides a balanced measure of precision and recall for both classes.

The classification report offers a detailed breakdown of these metrics, while the confusion matrix visualization provides insights into true positives, true negatives, false positives, and false negatives.

```
Accuracy: 0.80
Precision (Efficient): 0.00
Recall (Efficient): 0.00
F1 Score (Efficient): 0.00
Precision (Non-Efficient): 0.80
Recall (Non-Efficient): 1.00
F1 Score (Non-Efficient): 0.89
```



```
               precision    recall  f1-score   support

Non-Efficient       0.00      0.00      0.00       246
    Efficient       0.80      1.00      0.89       986

     accuracy                           0.80      1232
    macro avg       0.40      0.50      0.44      1232
 weighted avg       0.64      0.80      0.71      1232
```

Key Functions and Steps:

1. **Data Preparation:**
    - The code assumes you have a large dataset of **sunlight_percentages3** and **classified_labels**. Ensure that your system has sufficient memory to handle the data.
2. **Data Splitting:**
    - The dataset is split into training (80%) and testing (20%) sets using the **train_test_split** function, ensuring a random but consistent split is maintained (using **random_state**).
3. **Classifier Initialization and Training:**
    - A Multinomial Naive Bayes classifier is created using **MultinomialNB()** from **sklearn.naive_bayes**. The classifier is trained on the training data using **fit**.
4. **Data Reshaping:**
    - Unlike in Code-12, there's no need to reshape the data for the Multinomial Naive Bayes classifier.
5. **Prediction:**
    - The trained classifier is used to predict the classes of the testing data, generating **y_pred**.
6. **Evaluation Metrics:**
    - Several classification metrics are calculated, including:

        - **Accuracy:** This metric quantifies the overall correctness of the model's predictions.
        - **Precision:** Precision measures the accuracy of positive predictions for both "Efficient" and "Non-Efficient" classes.
        - **Recall:** Recall measures the ability of the classifier to correctly identify instances of both classes.
        - **F1 Score:** The F1 score provides a balanced measure of precision and recall for both classes.

7. **Confusion Matrix Visualization:**
    - A confusion matrix is computed using **confusion_matrix** and visualized as a heatmap using **seaborn** and **matplotlib**. This visualization facilitates a clear assessment of the classifier's performance.
8. **Classification Report:**
    - A classification report is generated using **classification_report**, providing detailed information on precision, recall, F1 score, and support for both "Efficient" and "Non-Efficient" classes.

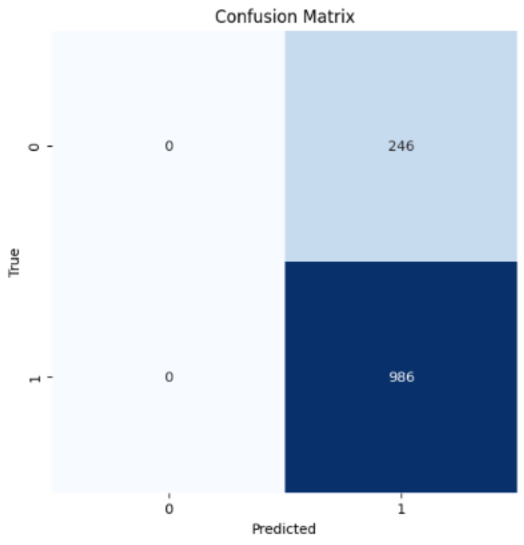
Results:

Code-13 produces the same key results as Code-12:

- **Accuracy:** This metric quantifies the overall accuracy of the model's predictions.
- **Precision:** Precision measures the accuracy of positive predictions, specifically for both "Efficient" and "Non-Efficient" classes.
- **Recall:** Recall quantifies the model's ability to correctly identify instances of both classes.
- **F1 Score:** The F1 score provides a balanced measure of precision and recall for both classes.

The classification report offers a detailed breakdown of these metrics, while the confusion matrix visualization provides insights into true positives, true negatives, false positives, and false negatives.

```
Accuracy: 0.80
Precision (Efficient): 0.00
Recall (Efficient): 0.00
F1 Score (Efficient): 0.00
Precision (Non-Efficient): 0.80
Recall (Non-Efficient): 1.00
F1 Score (Non-Efficient): 0.89
```



```
               precision    recall  f1-score   support

Non-Efficient       0.00      0.00      0.00       246
    Efficient       0.80      1.00      0.89       986

     accuracy                           0.80      1232
    macro avg       0.40      0.50      0.44      1232
 weighted avg       0.64      0.80      0.71      1232
```

Conclusion:

In conclusion, these code segments illustrate various aspects of image analysis and classification. Each code addresses a specific image processing task and provides insights into the classification of images based on different features. Results and evaluations vary across codes, depending on the specific task and the chosen evaluation metrics.

These codes offer a glimpse into the versatility of image analysis and machine learning techniques for classifying images based on various criteria. Further refinement and optimization of these models could lead to more accurate and robust image classification systems, with applications in areas such as solar panel detection, dust monitoring, and sunlight exposure analysis.

Overall, these code segments demonstrate the potential of image analysis and machine learning for automating tasks and making informed decisions based on visual data.