ML Lab 5 Pipeline

Group Partner: Abdullah Rihawi

Submitted by: Abdul Malik

Introduction

In this lab, we explored how to build a complete machine learning classification pipeline using Scikit-Learn. A pipeline helps organize the steps needed to train and test models including preprocessing, model fitting, evaluation, and tuning. We used the digit recognition dataset, which includes small grayscale images of handwritten numbers from 0 to 9. Each image is 8x8 pixels and is already converted into numerical format for easy use in machine learning.

Objective

The main goals of this lab were:

- To understand how to preprocess and explore a real-world dataset
- To build a classification pipeline using Scikit-learn
- To compare multiple classification models
- To evaluate model performance using metrics and visualizations
- To optimize a model using hyperparameter tuning

Implementation

We loaded the digits dataset using load_digits() from Scikit-Learn. The dataset contains 1,797 rows (each representing a digit image) and 64-pixel features, with one additional column for the target value, which ranges from 0 to 9. Each image is originally 8x8 pixels, and has already been flattened into a one-dimensional array.

To begin understanding the data, we visualized some sample digit images. These helped us see the kind of handwriting variation and image quality present in the dataset. Next, we explored the data using statistical summaries, boxplots, and a heatmap to identify any missing values, outliers, or feature correlations. The dataset was clean, with no missing values, and we observed that some features showed variation in pixel intensity.

We then built classification pipelines using Scikit-Learn's Pipeline functionality. Each pipeline included preprocessing steps such as StandardScaler to normalize features, and PCA (Principal Component Analysis) to reduce the number of features from 64 to 30. This helped reduce computation time while keeping most of the data's important information. After preprocessing, we trained five different classifiers: Logistic Regression, Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Gradient Boosting. Each model was trained and tested using a split of 80% training data and 20% testing data. After this, we compared their performance using evaluation metrics like accuracy, precision, recall, and F1-score.

Finally, we used GridSearchCV to tune the hyperparameters of the Random Forest model and checked whether the performance improved after tuning.

Results

Below we showed the results of our Pipeline implementation and discussed each result thoroughly.

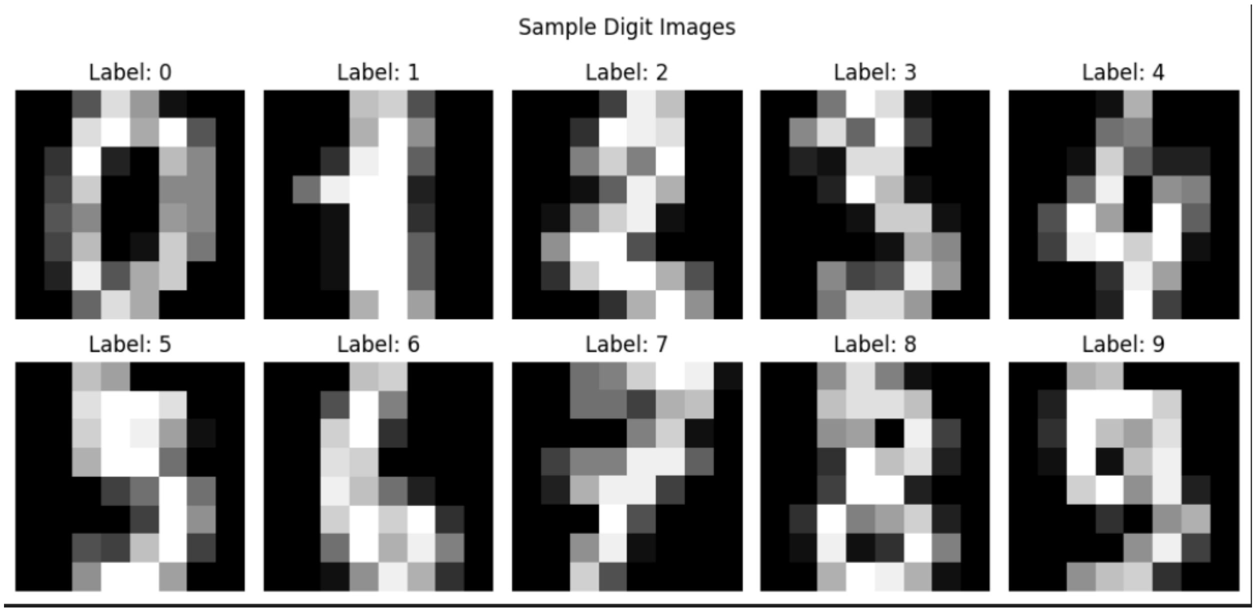
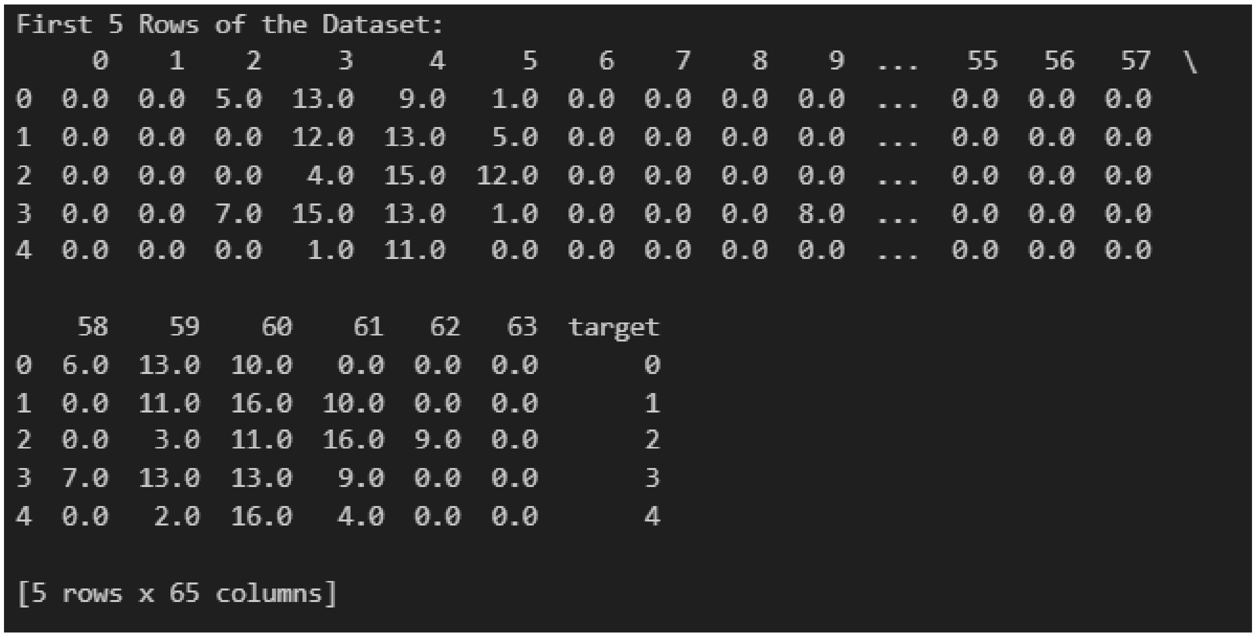


Figure 1: Sample Digit Images (grid of 0–9)

This figure shows a selection of 8×8 grayscale images from the dataset, each representing a handwritten digit between 0 and 9. Looking at these images gives us a visual understanding of the variation in how different people write the same digit. For example, the number "5" might be written with or without a curve at the top, and "1" might have a horizontal base or be a simple vertical line. These variations are important because they make the classification task more challenging and realistic. Models trained on this data need to learn to recognize patterns despite these small differences.



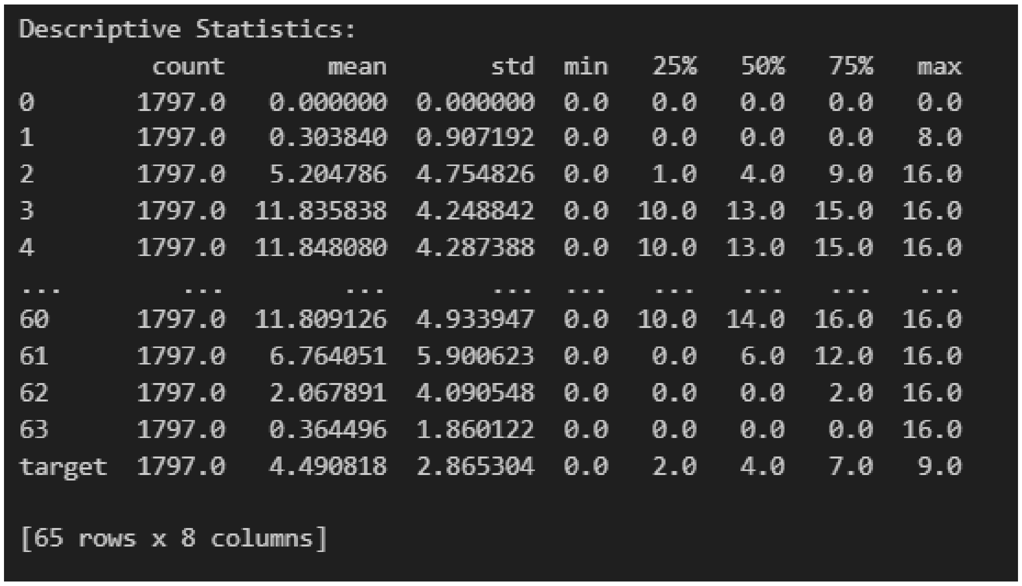
```
First 5 Rows of the Dataset:
     0    1    2     3     4     5    6    7    8    9  ...   55   56   57  \
0  0.0  0.0  5.0  13.0   9.0   1.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
1  0.0  0.0  0.0  12.0  13.0   5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
2  0.0  0.0  0.0   4.0  15.0  12.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
3  0.0  0.0  7.0  15.0  13.0   1.0  0.0  0.0  0.0  8.0  ...  0.0  0.0  0.0
4  0.0  0.0  0.0   1.0  11.0   0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0

    58    59    60    61   62   63  target
0  6.0  13.0  10.0   0.0  0.0  0.0       0
1  0.0  11.0  16.0  10.0  0.0  0.0       1
2  0.0   3.0  11.0  16.0  9.0  0.0       2
3  7.0  13.0  13.0   9.0  0.0  0.0       3
4  0.0   2.0  16.0   4.0  0.0  0.0       4

[5 rows x 65 columns]
```

Figure 2: First 5 rows of the dataset

This table displays the first five records of the dataset. Each row represents one digit image, with 64 numerical columns representing pixel intensities (ranging from 0 to 16) and one target column that indicates the correct digit label (0–9). These numerical values are the flattened version of the 8×8 image. By viewing the raw data in this form, we can see that the model doesn't directly "see" the images, but instead works with numbers that describe how dark or light each pixel in the image is.

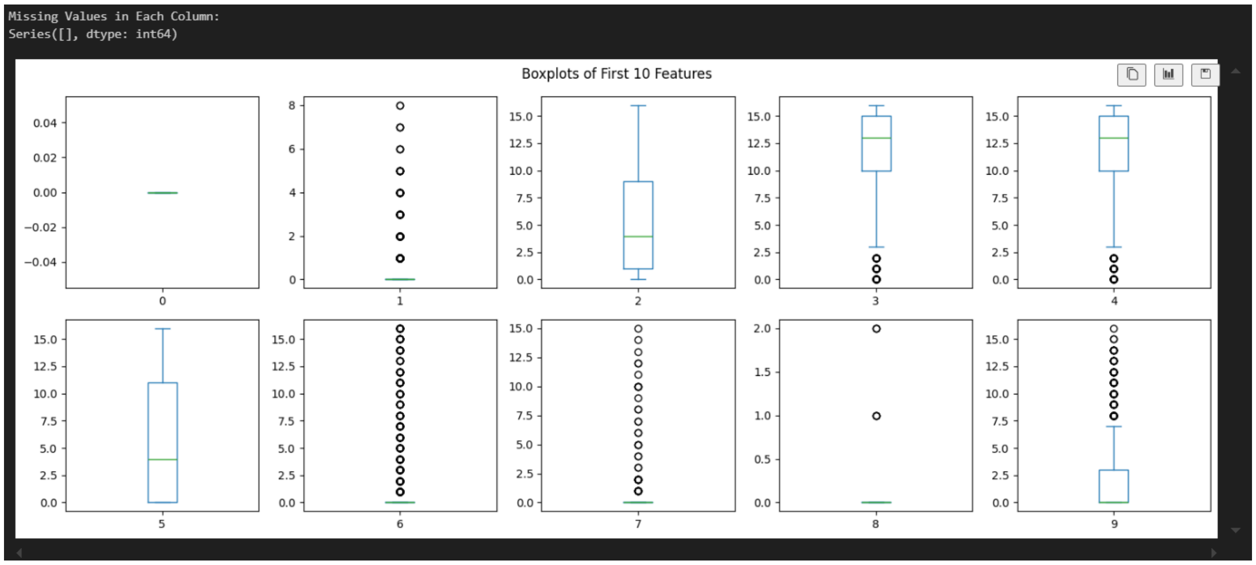


```
Descriptive Statistics:
         count       mean       std  min   25%   50%   75%   max
0       1797.0   0.000000  0.000000  0.0   0.0   0.0   0.0   0.0
1       1797.0   0.303840  0.907192  0.0   0.0   0.0   0.0   8.0
2       1797.0   5.204786  4.754826  0.0   1.0   4.0   9.0  16.0
3       1797.0  11.835838  4.248842  0.0  10.0  13.0  15.0  16.0
4       1797.0  11.848080  4.287388  0.0  10.0  13.0  15.0  16.0
...        ...        ...       ...  ...   ...   ...   ...   ...
60      1797.0  11.809126  4.933947  0.0  10.0  14.0  16.0  16.0
61      1797.0   6.764051  5.900623  0.0   0.0   6.0  12.0  16.0
62      1797.0   2.067891  4.090548  0.0   0.0   0.0   2.0  16.0
63      1797.0   0.364496  1.860122  0.0   0.0   0.0   0.0  16.0
target  1797.0   4.490818  2.865304  0.0   2.0   4.0   7.0   9.0

[65 rows x 8 columns]
```

Figure 3: Descriptive statistics table

This table summarizes key statistics (such as mean, standard deviation, min, max) for each of the 64-pixel features. It helps us understand how the pixel values are distributed across the dataset. For instance, a pixel that is frequently part of the digit stroke will have a higher average value, while pixels near the corners (often background) may have low average values. This statistical view also helps us identify which features (pixels) might be more informative for the model and whether some pixels are almost always zero (and could be removed or down-weighted).



```
Missing Values in Each Column:
Series([], dtype: int64)
```

Figure 4: Boxplots of first 10 features

This figure shows boxplots for the first 10-pixel features. A boxplot is a graphical way to show the spread and distribution of data. The box represents the middle 50% of values (from the first quartile to the third), while the line inside shows the median. Points outside the box are considered outliers. In our case, these boxplots show how much pixel intensity varies for certain pixels across different digit images. Some features have wider boxes and more outliers, indicating that they capture areas of the image where handwriting varies a lot, while others remain more stable.

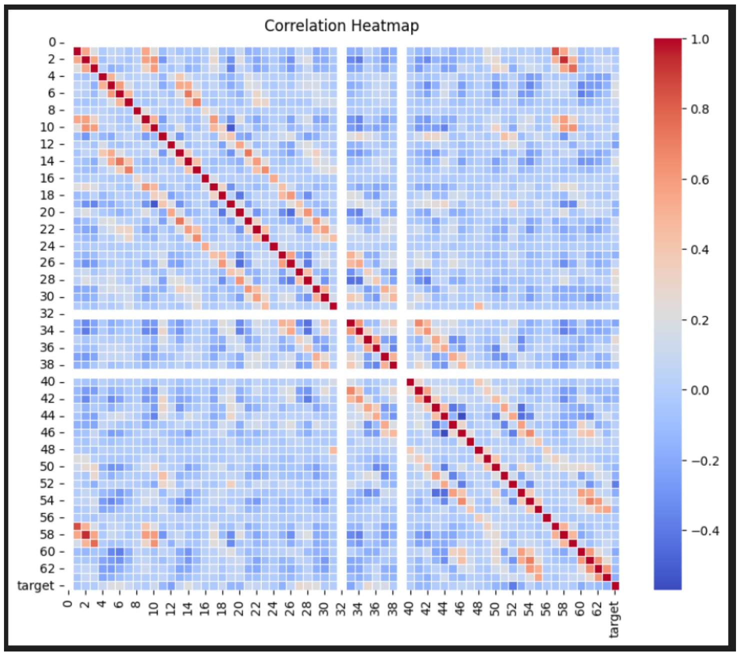
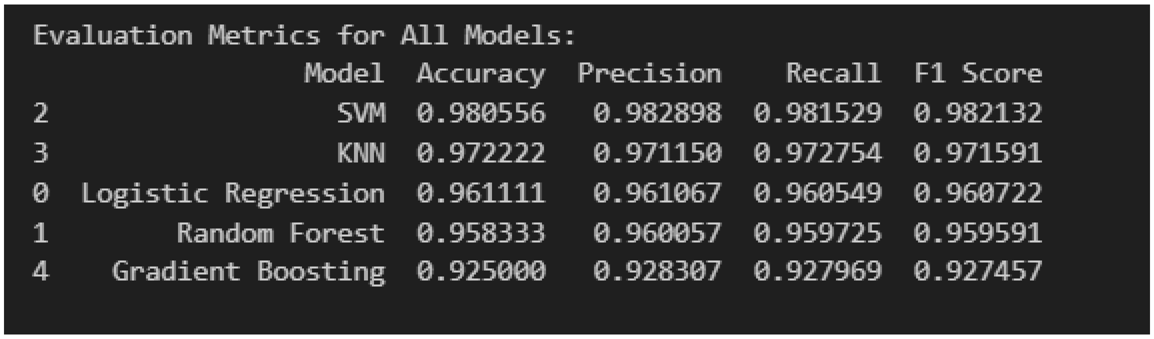


Figure 5: Correlation heatmap

This figure presents a heatmap of the correlation between the first few features. A heatmap uses colors to show how strongly two features are related darker or brighter colors indicate higher or lower correlation. A value close to 1 means that two pixels often have similar values, while a value near 0 means no relation. In digit images, adjacent pixels often have similar values (for example, pixels along the stroke of the digit), so some features are moderately correlated. Understanding these correlations is useful before applying dimensionality reduction techniques like PCA.



```
Evaluation Metrics for All Models:
                  Model  Accuracy  Precision    Recall  F1 Score
2                   SVM  0.980556   0.982898  0.981529  0.982132
3                   KNN  0.972222   0.971150  0.972754  0.971591
0   Logistic Regression  0.961111   0.961067  0.960549  0.960722
1         Random Forest  0.958333   0.960057  0.959725  0.959591
4     Gradient Boosting  0.925000   0.928307  0.927969  0.927457
```

Figure 6: Table of Evaluation Metrics for All Models

This table shows the overall performance of five different machine learning models: Logistic Regression, Random Forest, SVM, KNN, and Gradient Boosting. For each model, we measured accuracy (how many predictions were correct), precision (how many predicted positives were actually correct), recall (how many actual positives were correctly identified), and F1-score (a balance between precision and recall). The SVM model achieved the best results across all metrics, indicating that it is the most reliable for this classification task. This comparison helps us decide which model is most suitable.

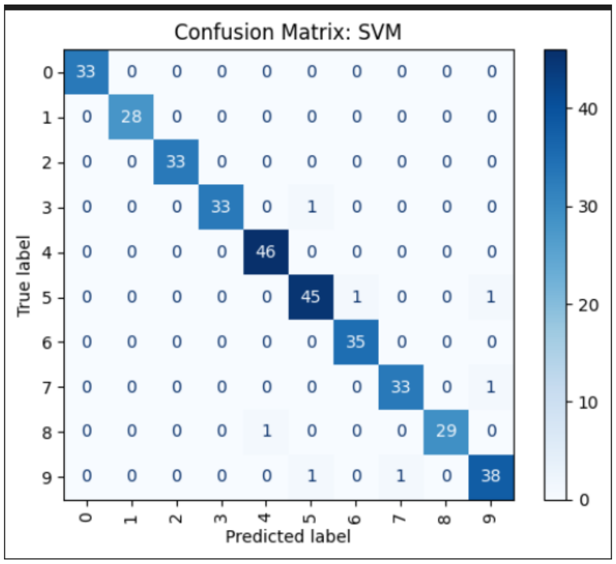
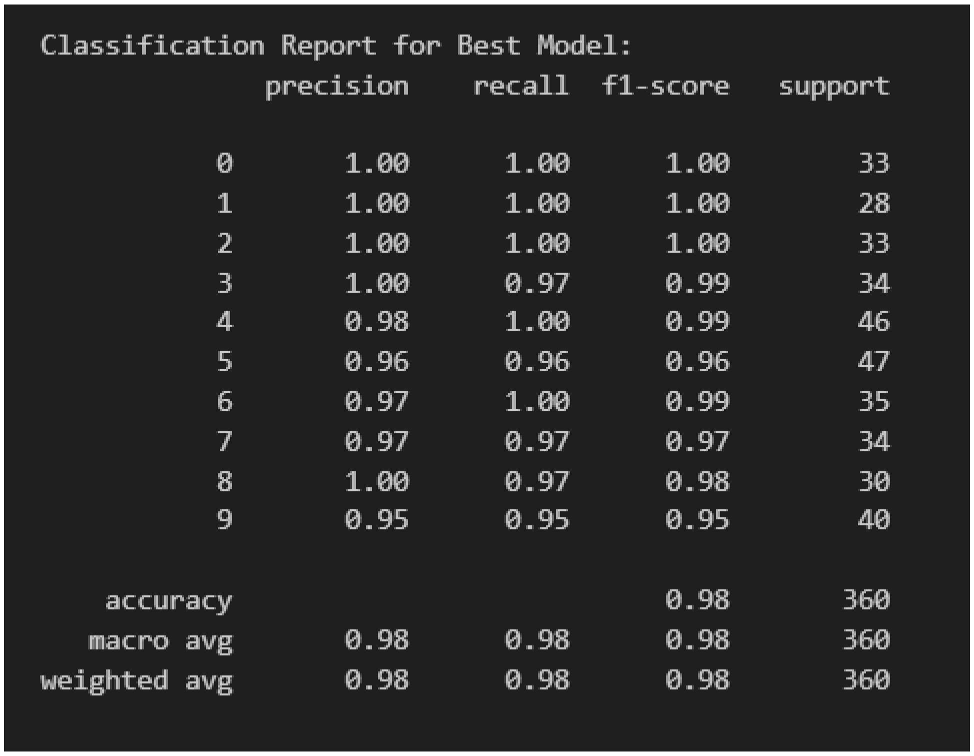


Figure 7: Confusion Matrix for SVM

This is a confusion matrix, which gives a detailed view of how well the SVM model predicted each class. The rows show actual digit labels, and the columns show the model's predictions. Ideally, all values should fall along the diagonal, which means the model predicted the correct digit. This matrix is mostly diagonal with very few off-diagonal values, showing that SVM made very few mistakes. It helps us understand where the model might still be confused for example, if it predicts a "9" when the digit is actually an “8”.



```
Classification Report for Best Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        33
           3       1.00      0.97      0.99        34
           4       0.98      1.00      0.99        46
           5       0.96      0.96      0.96        47
           6       0.97      1.00      0.99        35
           7       0.97      0.97      0.97        34
           8       1.00      0.97      0.98        30
           9       0.95      0.95      0.95        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

Figure 8: Table of Classification Report for SVM

This report breaks down the performance of the SVM model per digit class. For each digit from 0 to 9, it shows precision, recall, and F1-score. Most of the digits have scores above 97%, showing that the model performed consistently across all classes. This kind of table is especially useful when checking whether the model favors or struggles with specific digits (e.g., it might confuse “1” and “7” more than others).

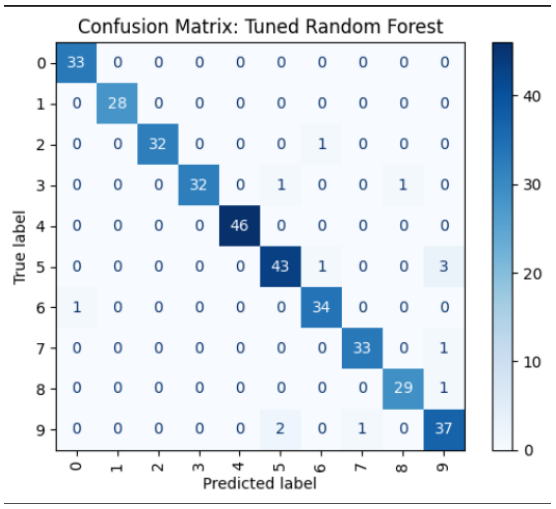
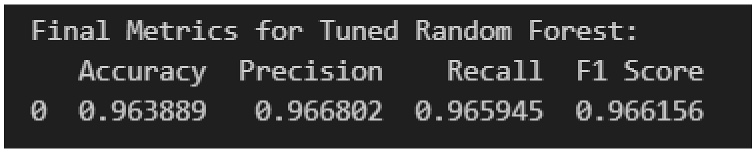


Figure 9: Confusion Matrix for Random Forest

Like the earlier confusion matrix, this one shows how well the Random Forest model performed. While the matrix still shows strong diagonal values, we can see slightly more off-diagonal entries than with SVM. That means Random Forest made more classification errors. Most of the misclassifications involve digits that are visually similar (such as 3 and 5 or 4 and 9), which is common in digit recognition problems.



```
Final Metrics for Tuned Random Forest:
   Accuracy  Precision    Recall  F1 Score
0  0.963889   0.966802  0.965945  0.966156
```

Figure 10: Final Evaluation of Random Forest

This table shows the performance of the Random Forest model after tuning its hyperparameters using GridSearchCV. The accuracy and F1-score improved slightly, showing that tuning helped the model learn better. However, the improvement wasn't large enough to outperform the SVM. This confirms that while hyperparameter tuning is useful, model selection also plays a big role in achieving high accuracy.

Discussion

From the results, it's clear that the Support Vector Machine model gave the best overall performance. It had the highest F1-score and showed the least number of misclassifications. The use of PCA helped simplify the feature space without reducing accuracy, which is useful for datasets with many features like image pixels.

One limitation we noticed is that some models, like Random Forest and Gradient Boosting, were slightly less accurate, especially before tuning. Additionally, some digits were confused with others due to similarities in handwriting, which is a common issue in digit recognition tasks.

Another limitation is that we only used traditional machine learning models. While they performed well, deep learning models like Convolutional Neural Networks (CNNs) are known to perform even better on image data. However, using CNNs would require more computing power and a different setup.

To improve in the future, we could:

- Use a larger dataset like MNIST (28x28 images) for more challenging classification
- Try neural networks that can better capture spatial patterns in images
- Explore ensemble methods that combine predictions from multiple models

Conclusion

This lab helped us understand how to build a full machine learning pipeline using Scikit-Learn. We learned how to explore and preprocess data, build pipelines, train different models, evaluate them using metrics and visualizations, and tune hyperparameters to improve performance. The digit recognition dataset was a good example for applying these techniques. We found that the SVM model worked best for this task. With further improvements and advanced models, even higher accuracy can be achieved.