ARCH Technologies Machine Learning Internship Report

Personal Information

Name: Ahmad

Intern ID: ARCH-2504-0348

Phone Number: 0330-6771616

Email: ahmadkhanmarwat8@gmail.com

Submission Date: May 30, 2025

Category B: Intermediate

Task 01 Classification Fundamentals and MNIST Digit Recognition

Chapter 3: Classification Notes

1. MNIST Dataset

- **Description**: 70,000 handwritten digit images (28x28 pixels, 784 features), labeled 0–9.
- **Loading**: Use fetch_openml('mnist_784', version=1) in Scikit-learn.
- **Structure**: Dictionary with data (features), target (labels), DESCR, etc.
- **Preprocessing**: Scale pixels to [0, 1] (divide by 255), cast labels to integers (y.astype(np.uint8)).
- **Split**: 60,000 training, 10,000 test images (already shuffled).

2. Binary Classification

- **Task**: Example: “5-detector” (5 vs. not-5).
- **SGD Classifier**: Uses Stochastic Gradient Descent, efficient for large datasets, relies on randomness (random_state=42).
- **Evaluation**:
    - **Cross-Validation**: Use cross_val_score with 3 folds for accuracy.
    - **Confusion Matrix**: True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN).
    - **Metrics**:
        - **Precision**: TP / (TP + FP) – accuracy of positive predictions.
        - **Recall**: TP / (TP + FN) – ratio of positives detected.
        - **F1 Score**: Harmonic mean of precision and recall (2 * precision * recall / (precision + recall)).
    - **Precision/Recall Tradeoff**: Adjust decision threshold using decision_function() scores.
    - **ROC Curve**: Plots True Positive Rate (recall) vs. False Positive Rate (FPR = FP / (FP + TN)). AUC = 1 for perfect classifier.

3. Multiclass Classification

- **Task**: Classify digits 0–9.
- **Strategies**:
    - **One-vs-Rest (OvR)**: Train one binary classifier per class (N classifiers for N classes).
    - **One-vs-One (OvO)**: Train N * (N-1) / 2 classifiers for each pair of classes.
- **Algorithms**:
    - **Random Forest**: Directly handles multiclass, uses predict_proba() for probabilities.
    - **SGD Classifier**: Scikit-learn auto-applies OvR (except for SVM, which uses OvO).

4. Error Analysis

- **Confusion Matrix**: Visualize errors using matshow() or seaborn.heatmap().
- **Normalization**: Divide by class size to compare error rates.
- **Common Errors**: E.g., 3s and 5s confused due to similar shapes.
- **Improvements**: Data augmentation (shift/rotate images), feature engineering (count loops), preprocessing (center images).

5. Multilabel Classification

- **Task**: Output multiple binary labels per instance (e.g., digit is large [≥7] and odd).
- **Example**: Use KNeighborsClassifier with np.c_[y_large, y_odd].
- **Evaluation**: Average F1 score across labels (average='macro' or 'weighted').

6. Multioutput Classification

- **Task**: Multiple labels with multiple values (e.g., denoise images, predict pixel intensities).
- **Example**: Add noise to MNIST, train KNeighborsClassifier to predict clean pixels.
- **Note**: Blurs line between classification and regression.

Key Takeaways

- Classification involves predicting classes (binary or multiclass).
- Performance metrics (precision, recall, F1, ROC AUC) are critical for evaluation.
- Error analysis and data augmentation improve model performance.
- Random Forest often outperforms SGD for MNIST due to non-linear decision boundaries.

Comparison Tables

Table 1: SGD Classifier vs. Random Forest Performance

Based on MNIST multiclass classification (10 classes, 60,000 training, 10,000 test samples).

Metric	SGD Classifier	Random Forest
Test Accuracy	87.7% (scaled)	97.0%
Training Time	~2 min	~5 min
Precision (Macro)	0.90	0.97
Recall (Macro)	0.90	0.97
F1 Score (Macro)	0.90	0.97
Robustness	Sensitive to scaling, rotations	Robust to non-linear patterns
Scalability	Efficient for large datasets	Slower for large datasets
Error Patterns	High errors for 3/5, 4/9	Fewer errors, mostly 8/3

Notes:

- **SGD**: Uses OvR, linear model, sensitive to pixel shifts (e.g., 3 vs. 5 confusion). Scaling inputs improves accuracy from 84% to 89.7%.
- **Random Forest**: Directly handles multiclass, non-linear, achieves ≥95% (meets manual's goal). Longer training time but better performance.
- **Code for Metrics**:

```
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, f1_score
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train_scaled, y_train)
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
y_pred_sgd = sgd_clf.predict(X_test)
```

```
y_pred_rf = rf_clf.predict(X_test)
print("SGD Accuracy:", accuracy_score(y_test, y_pred_sgd))
print("RF Accuracy:", accuracy_score(y_test, y_pred_rf))
```

Table 2: OvR vs. OvO Strategies for Multiclass Classification

Based on Chapter 3's explanation for MNIST (10 classes).

Aspect	One-vs-Rest (OvR)	One-vs-One (OvO)
# Classifiers	N (10)	N * (N-1) / 2 (45)
Training Data	Full dataset per classifier	Subset (two classes) per classifier
Computational Cost	Lower (fewer classifiers)	Higher (more classifiers)
Training Time	Faster	Slower
Suitability	Preferred for most algorithms (e.g., SGD)	Preferred for SVM (scales poorly with large datasets)
Scikit-Learn Default	Used for SGD, Linear models	Used for SVM
Performance	Comparable for MNIST	Comparable, slight edge for small datasets

Notes:

- **OvR**: Trains 10 classifiers (e.g., 0 vs. not-0, 1 vs. not-1). Each uses the full training set, making it faster but less targeted.
- **OvO**: Trains 45 classifiers (e.g., 0 vs. 1, 0 vs. 2). Each uses a subset, reducing training time per classifier but increasing total classifiers.
- **Example**:

```
from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))
ovo_clf.fit(X_train_scaled, y_train)
print("OvO SGD Accuracy:", accuracy_score(y_test, ovo_clf.predict(X_test)))
```

Error Analysis Report

1. Identify 3 Common Error Patterns

We'll analyze the SGD Classifier confusion matrix to identify three common error patterns, as it has a lower accuracy (87%) and more pronounced misclassifications. The matrix is:

[[902 0 8 11 1 13 2 4 39 0]

[0 1095 2 3 0 2 4 1 28 0]

[1 10 803 69 6 4 4 10 122 3]

[0 1 6 931 1 21 3 7 35 5]

[2 2 9 15 778 4 2 9 62 99]

[6 2 1 71 3 709 12 12 67 9]

[5 3 12 3 5 21 854 0 45 0]

[0 3 18 20 3 4 1 919 18 42]

[3 5 2 30 4 43 5 5 872 5]

[3 5 2 33 7 5 0 20 57 877]]

Rows represent actual classes (digits 0–9), and columns represent predicted classes. Off-diagonal values indicate misclassifications. The classification report shows low precision/recall for digits 2 (recall: 0.78), 4 (recall: 0.79), and 8 (precision: 0.65), suggesting error-prone classes. The top three error patterns are:

- **Error Pattern 1: Digit 2 Misclassified as Digit 8 (122 cases):**
    - Description: 122 instances of actual digit 2 were predicted as digit 8 (row 2, column 8).
    - Reason: Handwritten 2s with a closed bottom loop or loopy structure resemble 8's double loops, especially in slanted or stylized forms. The low recall for digit 2 (0.78) indicates many 2s are missed.
    - Impact: Contributes to 229 total errors for digit 2 (1032 - 803 = 229 misclassifications), significantly lowering accuracy.
    - Example: A 2 with a pronounced loop may visually mimic an 8.
- **Error Pattern 2: Digit 4 Misclassified as Digit 9 (99 cases):**
    - Description: 99 instances of actual digit 4 were predicted as digit 9 (row 4, column 9).

- Reason: Open-top 4s or 4s with a curved top can resemble 9s, particularly if the vertical stroke is faint. Digit 4's recall (0.79) is low, reflecting frequent misclassifications.
- Impact: Accounts for a large portion of digit 4's 204 errors (982 - 778 = 204), affecting overall performance.
- Example: A 4 with a rounded top may look like a 9's closed loop.

- **Error Pattern 3: Digit 8 Misclassified as Digit 5 (43 cases):**
  - Description: 43 instances of actual digit 8 were predicted as digit 5 (row 8, column 5).
  - Reason: Single-loop 8s (e.g., with a smaller top loop) or tilted 8s resemble 5's structure (single loop with a top stroke). Digit 8's low precision (0.65) indicates many false positives, as other digits are mistaken for 8, but this error focuses on 8→5.
  - Impact: Part of digit 8's 102 errors (974 - 872 = 102), contributing to its poor F1-score (0.75).
  - Example: A simplified or slanted 8 may be confused with a 5.

These patterns are confirmed by the classification report's metrics and visual inspection of misclassified images (implemented below).

2. Propose Solutions

To address these error patterns, we propose solutions inspired by Chapter 3 (*Hands-On Machine Learning*, pages 104–107) and the manual's emphasis on optimization:

- **Solution for Digit 2→8 Misclassification:**
  - Data Augmentation: Shift or rotate images of 2 to capture varied loop structures, reducing confusion with 8's double loops (as in Chapter 3, Exercise 2).
  - Preprocessing: Deskew images to correct slanted 2s, which may resemble 8s. Apply edge detection to emphasize loop differences.
  - Feature Engineering: Count closed loops (2 has one, 8 has two) to aid discrimination.
- **Solution for Digit 4→9 Misclassification:**
  - Data Augmentation: Generate shifted or slightly rotated 4s to emphasize the open-top structure versus 9's closed loop.
  - Preprocessing: Enhance contrast to highlight 4's vertical stroke and open top. Deskewing can standardize orientation.
  - Model Tuning: Increase SGD regularization (alpha) to reduce overfitting to similar shapes like 9.

- **Solution for Digit 8→5 Misclassification:**
    - Data Augmentation: Augment 8s with shifts and rotations to capture single-loop variants, distinguishing them from 5s.
    - Preprocessing: Deskew images to correct tilted 8s that resemble 5s. Apply morphological operations to clarify loop structures.
    - Alternative Model: Use a non-linear model like Random Forest (97% accuracy) or KNN (97.7% from Exercise 1), which better handle complex patterns.

Chosen Solution: Data Augmentation by shifting images one pixel in four directions (up, down, left, right). This approach:

- Addresses all three patterns by increasing training data diversity.
- Aligns with Chapter 3's Exercise 2, which improved KNN accuracy to 98.2%.
- Is practical and computationally feasible (though intensive).
- Helps the SGD Classifier learn robust features for digits 2, 4, and 8.

3. Implement One Improvement and Measure Impact

We'll implement data augmentation by shifting each training image by one pixel in four directions, expanding the training set from 60,000 to 300,000 samples. We'll retrain the SGD Classifier with tuned parameters, evaluate the impact on test accuracy and the three error patterns, and compare with the Random Forest Classifier (97% accuracy) to ensure Task 01's ≥95% requirement is met.

Implementation Plan:

- Load and preprocess MNIST (scale pixels to [0, 1]).
- Augment the training set using scipy.ndimage.shift.
- Retrain the SGD Classifier with parameters tuned for stability (alpha=0.001, learning_rate='adaptive').
- Compute the new confusion matrix, accuracy, and classification report.
- Analyze changes in error patterns (2→8, 4→9, 8→5).
- Visualize the confusion matrix and misclassified images to confirm improvements.
- Use Random Forest as the final model for Task 01 deliverables.

The implementation is given in the python notebook along with the submission.