ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ASSIGNMENT

Nhận diện chữ số viết tay (0–9) trên tập dữ liệu MNIST (phiên bản Kaggle Digit Recognizer)

CO3117 - Học Máy

LỚP: L02

Giảng viên hướng dẫn: Võ Thanh Hùng
Danh sách thành viên:

STT	Họ và tên	MSSV	Lớp
1	Đoàn Đức Bình	2210302	L02
2	Nguyễn Anh Kiệt	2211758	L02

TP. Hồ Chí Minh, Tháng 11/2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

1 Giới thiệu

Trong Bài Tập Lớn này, nhóm lựa chọn bài toán nhận diện chữ số Viết tay (0 - 9) trên tập dữ liệu MNIST (phiên bản Kaggle Digit Recognizer). Ảnh xám 28×28 được trải phẳng thành 784 chiều. Từ đó xây dựng, huấn luyện và đánh giá ít nhất hai mô hình học máy khác nhau, so sánh hiệu năng bằng các chỉ số: Accuracy, Precision, Recall, F1-score, và phân tích nhầm lẫn bằng Confusion Matrix.

Mục tiêu của nhóm là xây dựng và đánh giá hai phương pháp học máy khác nhau: *Logistic Regression* (sklearn) và *Neural Network (MLP)* (PyTorch) nhằm so sánh hiệu năng, ưu nhược điểm và khả năng tổng quát hóa của từng mô hình.

Bài báo cáo trình bày toàn bộ quy trình từ phân tích, xử lý dữ liệu, huấn luyện mô hình, đánh giá đến thảo luận và kết luận.

2 Phân tích bài toán và dữ liệu

Tập dữ liệu được sử dụng là Kaggle Digit Recognizer, một biến thể của MNIST với tổng cộng 42.000 mẫu train và 28.000 mẫu test (Nguồn dữ liệu: tại đây).

2.1 Đặc điểm dữ liệu

- Mỗi mẫu là một chữ số viết tay 28x28 pixels.
- Ảnh được làm phẳng thành vector 784 chiều.
- Nhãn (label) gồm 10 lớp: 0 - 9.
- Pixel có giá trị trong khoảng [0, 255].

2.2 Phân tích sơ bộ

- Dữ liệu có phân bố lớp khá cân bằng.
- Một số chữ số dễ nhầm lẫn: 3 - 5, 4 - 9, 5 - 6.

2.3 Tiền xử lý

- Chuẩn hóa pixel bằng cách chia cho 255.0.
- Chia dữ liệu theo tỷ lệ 80% train - 20% test với **stratify** để giữ phân bố nhãn.
- Dữ liệu được giữ dạng vector 784 chiều.

3 Phương pháp và thiết kế thực nghiệm

Với bài toán nhận diện chữ số viết tay MNIST và đưa ra đánh giá, nhóm lựa chọn triển khai và so sánh hai mô hình học máy: *Logistic Regression* (thư viện sklearn) và *Neural Network* (sử dụng PyTorch).

Các metric cần đánh giá:

- Accuracy, Precision (macro), Recall (macro), F1-score (macro).

- Confusion Matrix: phân tích chi tiết lỗi nhầm giữa các lớp (với các cặp chữ số dễ nhầm lẫn)

- Thời gian: đo thời gian huấn luyện và dự đoán.

- Tài nguyên mô hình.

3.1 Thiết kế Logistic Regression

Logistic Regression là một mô hình tuyến tính được sử dụng phổ biến trong bài toán phân loại. Và trong bài toán nhận diện chữ viết tay, ta cần phân loại một mẫu ảnh số vào 1 trong 10 lớp (các chữ số 0-9), mô hình được thiết lập ở chế độ đa lớp (multiclass).

Trong Bài Tập Lớn, nhóm khởi tạo mô hình Logistic Regression từ thư viện scikit-learn (sklearn) với các tham số:

- `solver= 'lbfgs'`: thuật toán tối ưu phù hợp cho bài toán đa lớp.

- `max_iter= 1000`: quy định số vòng lặp tối đa khi solver chạy để tối ưu hóa hàm mất mát (loss function), tăng số vòng để đảm bảo hội tụ.

- `n_jobs= -1`: cho phép chạy song song trên các CPU cores của máy.

Quy trình tiền xử lý và chia tập dữ liệu:

1. Dữ liệu được đọc từ file CSV của Kaggle (train.csv, test.csv).
2. Tách nhãn (label) và dữ liệu ảnh (784 cột pixel).
3. Chuẩn hóa pixel:
$$x' = \frac{x}{255.0} \tag{1}$$
4. Chia dữ liệu thành tập train và test theo tỷ lệ 80/20 (sử dụng `train_test_split` của sklearn).

Thiết kế thực nghiệm:

1. Dùng dữ liệu vector 784 chiều đã được chuẩn hóa.
2. Khởi tạo mô hình với các tham số ở trên (solver, max_iter, n_jobs).
3. Gọi `fit(X_train, Y_train)` để huấn luyện.
4. Gọi `predict(X_test)` để lấy nhãn dự đoán.
5. Tính toán các metric bằng các hàm `accuracy_score`, `precision_score`, `recall_score`, `f1_score`, `confusion_matrix`.
6. Lưu trọng số mô hình vào file `logistic-regression-model.joblib` để phục vụ đánh giá và tái hiện kết quả.

3.2 Thiết kế Neural Network

Mạng neural nhiều lớp (MLP) là một mô hình học sâu cơ bản, có khả năng học các quan hệ phi tuyến giữa đầu vào và đầu ra. Trong bài toán nhận diện chữ số viết tay, nhóm xây dựng một MLP với các tầng ẩn và dropout nhằm tăng khả năng biểu diễn và giảm overfitting.

Kiến trúc mô hình:

- Đầu vào: vector 784 chiều (ảnh 28×28 trải phẳng).
- **fc-784-to-512**: 784 → 512, kích hoạt ReLU, Dropout(0.3).
- **fc-512-to-256**: 512 → 256, kích hoạt ReLU, Dropout(0.2).
- **fc-256-to-128**: 256 → 128, kích hoạt ReLU, Dropout(0.1).
- **Skip connection**: song song với các tầng chính, đầu ra của fc-784-to-512 được đưa qua một lớp tuyến tính (512 → 128) rồi cộng vào đầu ra fc-256-to-128.
- **fc-128-to-10**: 128 → 10, đầu ra là logits cho 10 lớp.
- Kích hoạt: ReLU cho các tầng ẩn, đầu ra dùng trực tiếp cho CrossEntropyLoss (không softmax).

Quy trình tiền xử lý và chia tập dữ liệu:

1. Dữ liệu được đọc từ file CSV (train.csv).
2. Tách nhãn (label) và dữ liệu ảnh (784 cột pixel).
3. Chuẩn hóa pixel:

$$x' = \frac{x}{255.0} \tag{2}$$

4. Chia dữ liệu thành tập train và test theo tỷ lệ 80/20 (sử dụng `train_test_split` của sklearn, stratify theo nhãn).

Thiết kế thực nghiệm:

1. Dùng dữ liệu vector 784 chiều đã được chuẩn hóa.
2. Định nghĩa lớp `MLP` kế thừa `nn.Module` với các tầng như trên.
3. Sử dụng hàm mất mát `CrossEntropyLoss`, tối ưu Adam với learning rate 0.0005.
4. Huấn luyện mô hình với batch size 512, 10 epochs, trên GPU (nếu có).
5. Sau huấn luyện, đánh giá mô hình trên tập test: tính `accuracy`, `precision`, `recall`, `f1-score`, `confusion matrix`.
6. Lưu trọng số mô hình vào file `neural-network-model.pth` để phục vụ đánh giá và tái hiện kết quả.

4 Kết quả

4.1 Kết quả của Logistic Regression

- Thời gian huấn luyện (s): t_train = 24,964s
- Thời gian dự đoán (s): t_pred = 0,0269s
- Thời gian dự đoán trung bình/mẫu (s): 0,000003s
- Accuracy: 0.9130
- Precision (macro): 0.9121
- Recall (macro): 0.9117
- F1-score (macro): 0.9118
- Số tham số mô hình: 7850
- Kích thước mô hình: 61.33KB (0.06MB)
- Confusion Matrix:

```
Hàng: nhãn thật, Cột: nhãn dự đoán
[[793   0   6   2   3   7  10   0   5   1]
 [  0 914   1   4   0   2   0   1  14   1]
 [  5  12 751  11   9   9   3  14  19   2]
 [  4   6  27 762   0  35   4   9  14   9]
 [  3   4   8   0 736   3   5   6   7  42]
 [  8   8  10  32  14 649   9   2  21   6]
 [  4   3   6   0   5  11 796   0   2   0]
 [  1   4   4   5   8   1   0 819   6  32]
 [  7  24   8  24   3  24   9   2 695  17]
 [  5   4   3   8  23   5   0  29   7 754]]
```

Hình 1: Confusion Matrix của Logistic Regression

4.2 Kết quả của Neural Network

- Thời gian huấn luyện (s): training_time = 27,173s
- Thời gian dự đoán (s): testing_time = 0,31452s
- Thời gian dự đoán trung bình/mẫu (s): 0,000037s
- Accuracy: 0.9719
- Precision (macro): 0.9718
- Recall (macro): 0.9716
- F1-score (macro): 0.9717
- Số tham số mô hình: 633098
- Kích thước mô hình: 2.4MB
- Confusion Matrix:

```
Confusion Matrix:
[[814   0   1   1   1   1   5   1   2   1]
 [  0 929   3   1   0   1   1   1   1   0]
 [  5   2 811   1   2   1   1   9   3   0]
 [  1   1  13 831   0   9   0   3   8   4]
 [  0   2   2   0 781   2   5   3   2  17]
 [  3   1   0   9   0 733   4   1   5   3]
 [  3   0   0   0   3   1 817   0   3   0]
 [  1   4   2   1   3   1   0 864   0   4]
 [  3   4   3   5   1   1   7   1 786   2]
 [  3   1   1   5  12   2   0  12   4 798]]
```

Hình 2: Confusion Matrix của MLP

4.3 Bảng so sánh tổng hợp

Mô hình	Accuracy	Precision (macro)	Recall (macro)	F1 (macro)	Train time (s)	Predict time (s)	Params	Model size
Logistic Regression	0.9130	0.9121	0.9117	0.9118	24,964s	0,0269s	7850	66.33 KB
MLP (PyTorch)	0.9719	0.9718	0.9716	0.9717	27,173s	0,31452s	633098	2.4 MB

Bảng 1: Bảng so sánh kết quả hai mô hình

5 Phân tích và thảo luận

5.1 So sánh hai mô hình

- Logistic Regression cho kết quả ổn định, tốc độ nhanh, phù hợp làm baseline.

- MLP đạt độ chính xác và F1-score cao hơn đáng kể nhờ khả năng học phi tuyến.

5.2 Phân tích nhầm lẫn (Confusion Matrix)

- Logistic Regression có xu hướng nhầm nhiều ở các cặp: 3-5, 4-9, 7-9 và chữ số 8.

- MLP giảm đáng kể các lỗi nhầm so với Logistic Regression.

5.3 Hạn chế

- Logistic Regression là mô hình tuyến tính nên không thể học được các đặc trưng phi tuyến tính và cấu trúc của ảnh chữ số viết tay, làm cho mô hình gặp khó khăn khi phân biệt các chữ số có nét tương tự nhau hay số 8 có nhiều kiểu viết khác nhau.

- MLP: số tầng, số neuron, dropout, lr, batch size, epochs đều quan trọng; tăng epochs có thể cải thiện thêm nhưng cần theo dõi overfitting.

- Chưa thử các kỹ thuật nâng cao: Hyperparameter tuning, Data augmentation, CNN (hiệu quả vượt trội trên MNIST).

5.4 Gợi ý hướng cải tiến

- Thử chuẩn hóa z-score theo kênh (mặc dù dữ liệu dạng pixel đã scale).

- Thêm data augmentation nhẹ (shift/rotate) để tăng tính khái quát.

- Áp dụng early stopping, ReduceLROnPlateau cho MLP.

- Tối ưu siêu tham số (GridSearch/Optuna) cho cả LR và MLP.

- Thử CNN đơn giản (LeNet-like) làm baseline DL mạnh hơn cho dữ liệu ảnh.

6 GitHub và Tài Liệu Tham Khảo

Source Code: https://github.com/akngg/ml_251

TÀI LIỆU THAM KHẢO

[1] Y. LeCun, C. Cortes and C. J. C. Burges, "MNIST handwritten digit database," *Yann LeCun's Website*.

Available: https://yann.lecun.org/exdb/mnist/index.html.

[2] Scikit-learn Developers, "LogisticRegression — scikit-learn documentation."

Available: https://scikit-learn.org/.

[3] PyTorch Team, "PyTorch: An open source machine learning framework."

Available: https://pytorch.org/.

[4] Kaggle Competitions, "Learn computer vision fundamentals with the famous MNIST data."

Available: https://www.kaggle.com/competitions/digit-recognizer/data