

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐÀO TẠO CHẤT LƯỢNG CAO
NGÀNH CÔNG NGHỆ THÔNG TIN



BÁO CÁO ĐỒ ÁN 3

ỨNG DỤNG NHẬN DIỆN KÝ TỰ

SVTH: NGUYỄN THÀNH TÂM **17110219**

SVTH: LÊ QUỐC NGUYỄN VƯƠNG **17115253**

SVTH: PHAN SƠN **17110216**

GVPT: NGUYỄN THIÊN BẢO

TP. Hồ Chí Minh, tháng 12 năm 2020

MỤC LỤC

LỜI MỞ ĐẦU	1
Danh mục các hình	2
Danh mục các bảng.....	4
CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI.....	5
1.1. Giới thiệu đề tài	5
1.2.1. Lý do, mục đích chọn đề tài	5
1.2.2. Mục tiêu.....	5
CHƯƠNG 2: NỘI DUNG	6
1. Giới thiệu bộ dữ liệu.....	6
2. Bảng phân bố dữ liệu.....	8
2. Thư viện sử dụng	9
2. Thuật toán sử dụng	10
2.1. Support Vector Machines (SVM).....	10
3. Xử lý dữ liệu	12
3.1. Import dữ liệu	12
3.2. Reshape dữ liệu	13
3.3. Feature scaling.....	13
4. Tạo model bằng random forest.....	14
4.1. Training data.....	14
4.2. Đánh giá model.....	14
4.2.3. Dự đoán thử	18
4.3. Phân tích lỗi	19
4.4. Tối ưu model	25
4.5. Đánh giá model sau khi xử lý lại dữ liệu.....	26
5. Tạo model bằng SVM classifier	28
5.1. Tiến hành train model.....	28
5.2. Đánh giá model.....	29
5.3. Parameter tuning.....	30
CHƯƠNG 3: KẾT LUẬN	36
TÀI LIỆU THAM KHẢO	37

LỜI MỞ ĐẦU

Nhóm xin chân thành cảm ơn sự hướng dẫn tận tình của thầy Nguyễn Thiên Bảo, cả về chuyên môn lẫn tinh thần, đã giúp đỡ nhóm trong quá trình thực hiện project.

Với mỗi sinh viên nói chung, việc tích lũy kiến thức qua giáo trình, bài giảng trên lớp là rất quan trọng và cần thiết, tuy nhiên sinh viên lại chưa có nhiều cơ hội để áp dụng các kiến thức đó vào project. Cho nên, việc thực hiện Đồ án 3 chính là cơ hội cho nhóm được thử sức mình tạo ra những project thực tế, được hoạt động theo nhóm một cách chuyên nghiệp dưới sự hướng dẫn của các giảng viên có dày dặn kinh nghiệm trong lĩnh vực trí tuệ nhân tạo, cụ thể là machine learning.

Vì kiến thức và kỹ thuật còn hạn hẹp, nên trong quá trình thực hiện có xảy ra sai sót, rất mong thầy góp ý để nhóm có thể hoàn thiện project tốt hơn.

Được sự hướng dẫn tận tình của thầy, nhóm xin chân thành cảm ơn!

Danh mục các hình

Hình 1:	Biểu đồ phân bố dữ liệu của các class	8
Hình 2:	Thông tin margin của SVM.....	10
Hình 3:	Xác định class sau khi thêm feature bậc cao	12
Hình 4:	Công thức thực hiện SVM với Gaussian RBF Kernel	12
Hình 5:	Lấy bộ dữ liệu.....	13
Hình 6:	Reshape dữ liệu	13
Hình 7:	Thực hiện scaling dữ liệu	14
Hình 8:	Học với thuật toán RandomForestClassifier	14
Hình 9:	Đo hiệu suất validation accuracy.....	15
Hình 10:	Xem kết quả score của tập train và test	15
Hình 11:	Tính giá trị Predict, Accuracy	16
Hình 12:	Điểm số Precision, Recall, F1-score.	16
Hình 13:	Giá trị trung bình accuracy từng class.....	16
Hình 14:	Biểu đồ đường Precision của RandomForest	17
Hình 15:	Biểu đồ đường Recalls của RandomForest	17
Hình 16:	Biểu đồ đường F1-scores của RandomForest	18
Hình 17:	Dự đoán thử với 20 samples.....	18
Hình 18:	Confusion matrix	19
Hình 19:	Confusion matrix ẩn đường chéo chính	21
Hình 20:	Xem class 0 và 24.....	22
Hình 21:	Xem class 1 và 21	23
Hình 22:	Xem class 9 và 44.....	24
Hình 23:	Xem class 15 và 40.....	25

Hình 24:	Kết hợp class 15 và 40.....	26
Hình 25:	Accuracy sau khi học lại RandomForest.....	26
Hình 26:	Giá trị R2 và RMSE	27
Hình 27:	Confusion matrix sau khi tính lại	27
Hình 28:	Confusion matrix ảnh đường chéo chính sau khi tính lại.....	28
Hình 29:	Lấy 100000 samples	28
Hình 30:	Fit model với 100k samples.....	29
Hình 31:	Root mean squared error	29
Hình 32:	Validation accuracy	29
Hình 33:	Accuracy on test set.....	29
Hình 34:	Precision, recall, f1 score.....	30
Hình 35:	Tunning part 1	30
Hình 36:	Tunning part 2	31
Hình 37:	Tunning part 3	31
Hình 38:	Tunning part 4	31
Hình 39:	Fit model again	32
Hình 40:	Root mean squared error	32
Hình 41:	Validation accuracy	32
Hình 42:	Accuray on test set.....	32
Hình 43:	Precision chart	33
Hình 44:	Recall chart.....	34
Hình 45:	F1 score chart	35

Danh mục các bảng

Bảng 1:	Danh sách các class	7
---------	---------------------------	---

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu đề tài

Chủ đề của nhóm là xây dựng ứng dụng nhận diện ký tự văn bản với machine learning.

Ngày nay, các ứng dụng thông minh xuất hiện rất nhiều, đó là sự bùng nổ của lĩnh vực trí tuệ nhân tạo, nhóm đã quyết định xây dựng một ứng dụng có khả năng dự đoán ký tự mà người dùng vẽ ra là ký tự gì và show kết quả dự đoán lên màn hình.

1.2.1. Lý do, mục đích chọn đề tài

Với mong muốn tạo một ứng dụng thực tiễn, nhóm đã quyết định chọn đề tài này làm tiền đề để phát triển sau này.

Đây là một chủ đề có thể xem là nền tảng trong các bài toán sau này như quét văn bản từ hình ảnh, ...

1.2.2. Mục tiêu

Mục tiêu nhóm đặt ra là nhóm phải biết được quy trình thực hiện một dự án machine learning, hiểu về cách hoạt động của thuật toán, biết được các kỹ thuật xử lý dữ liệu, lựa chọn model, đánh giá model và phân tích model.

Mục tiêu đề ra là model xây dựng được phải có accuracy score trên 0.85, và ứng dụng tạo ra có thể sử dụng được model đã xây dựng để dự đoán.

CHƯƠNG 2: NỘI DUNG

1. Giới thiệu bộ dữ liệu

Nhóm sử dụng bộ dữ liệu EMNIST, đây là một tập hợp các chữ số ký tự viết tay bắt nguồn từ Cơ sở dữ liệu đặc biệt NIST 19 và được chuyển đổi sang định dạng hình ảnh 28x28 pixel và cấu trúc tập dữ liệu khớp trực tiếp với tập dữ liệu MNIST.

Tóm tắt tập dữ liệu:

Có sáu phần tách khác nhau được cung cấp trong tập dữ liệu này. Dưới đây là một bản tóm tắt ngắn về tập dữ liệu:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

Phần bổ sung đầy đủ của EMNIST có sẵn trong phần tách ByClass và ByMerge. Tập dữ liệu Balanced EMNIST chứa một tập hợp các ký tự với số lượng mẫu bằng nhau cho mỗi lớp. Tập dữ liệu EMNIST Letters kết hợp một tập hợp cân bằng giữa chữ hoa và chữ thường thành một tác vụ 26 lớp. Bộ dữ liệu EMNIST Digits và EMNIST MNIST cung cấp bộ dữ liệu chữ số viết tay cân bằng tương thích trực tiếp với bộ dữ liệu MNIST gốc. [1]

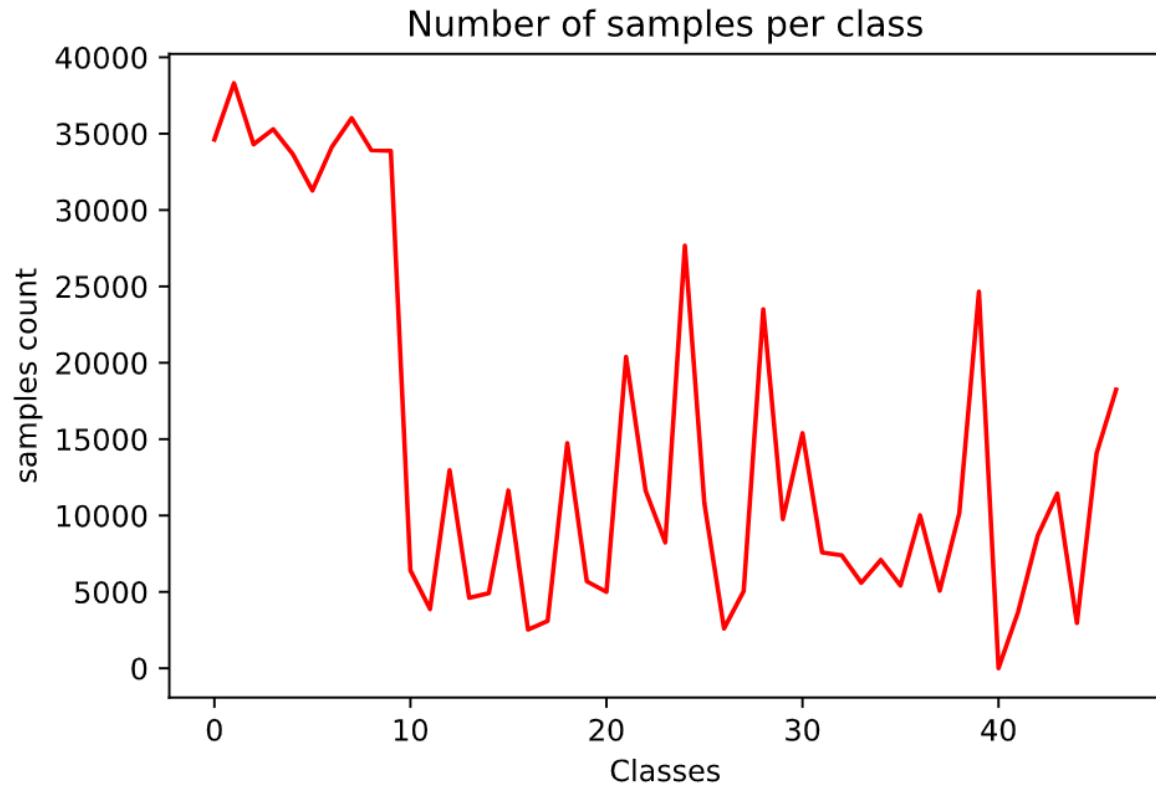
Ở trong project này, nhóm sử dụng phần tách ByMerge. Nhóm sử dụng phần này vì đây là bản đầy đủ của EMNIST. Hơn nữa, nó đã được merge một số class chữ hoa và chữ thường, vì có một số chữ in hoa và in thường được viết giống nhau, nên được gộp label lại, độ chính xác sẽ cao hơn.

Các class được đánh số từ 0 đến 46 với thứ tự như sau:

Class	Ký tự	Class	Ký tự
0	'0'	24	O, o
1	'1'	25	P, p
2	'2'	26	Q
3	'3'	27	R
4	'4'	28	S, s
5	'5'	29	T
6	'6'	30	U, u
7	'7'	31	V, v
8	'8'	32	W, w
9	'9'	33	X, x
10	A	34	Y, y
11	B	35	Z, z
12	C, c	36	a
13	D	37	b
14	E	38	d
15	F	39	e
16	G	40	f
17	H	41	g
18	I, i	42	h
19	J, j	43	n
20	K, k	44	q
21	L, l	45	r
22	M, m	46	t
23	N		

Bảng 1: Danh sách các class

2. Bảng phân bố dữ liệu



Hình 1: Biểu đồ phân bố dữ liệu của các class

- Ta thấy dữ liệu phân bố không đồng đều giữa các class, 10 class đầu được base trên bộ dữ liệu MNIST, nên có số lượng sample rất cao, trên 30000 samples trên mỗi class. Các class còn lại thì được thu thập sau, số lượng ít hơn rất nhiều, một số class chỉ từ 5000 samples.

2. Thư viện sử dụng

2.1. Thư viện Scikit-learn

Scikit-learn (Sklearn) là thư viện mạnh mẽ nhất dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Thư viện cung cấp một tập các công cụ xử lý các bài toán machine learning và statistical modeling gồm: classification, regression, clustering, và dimensionality reduction. [2]

Project lần này của nhóm thuộc về classification.

2.2. Thư viện emnist

Đây là thư viện giúp lấy danh sách các datasets trong bộ dữ liệu EMNIST, trong project này nhóm sử dụng bộ By Merge như đã giới thiệu.

2.3. Thư viện numpy

Đây là thư viện giúp xử lý tính toán trên mảng có kích thước lớn và nhiều chiều, được sử dụng xuyên suốt project.

2.4. Thư viện matplotlib

Matplotlib là một thư viện sử dụng để vẽ các đồ thị trong Python, chính vì vậy nó là thư viện cực phổ biến của Python. Nhóm sử dụng matplotlib để vẽ lên các đồ thị và confusion matrix.

2.5. Thư viện joblib

Đây là thư viện giúp nhóm lưu các model tìm được và các thông số tính toán.

2. Thuật toán sử dụng

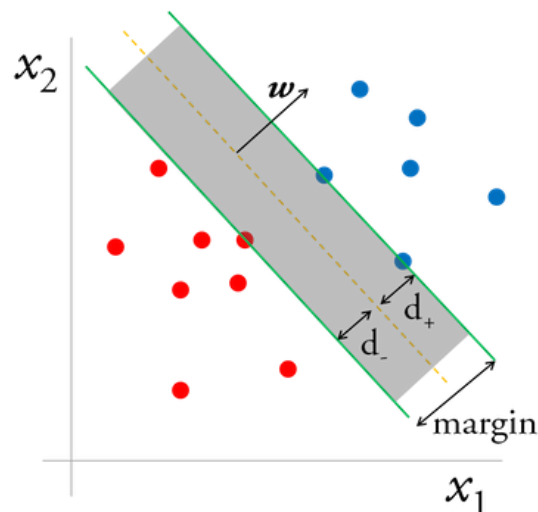
2.1. Support Vector Machines (SVM)

SVM là một thuật toán thuộc nhóm Supervised Learning (Học có giám sát) dùng để phân chia dữ liệu (Classification) thành các nhóm riêng biệt.

SVM là một thuật toán rất mạnh, chạy rất nhanh, trước khi deep learning ra đời thì nó là một trong những thuật toán tốt nhất. Trong một số trường hợp SVM nó còn tốt hơn deep learning vì deep learning sinh ra model rất phức tạp và nó cần phải có một lượng lớn dữ liệu nếu không nó chạy không tốt.

2.1.1. Margin

Xét trong không gian hai chiều, Margin là khoảng cách giữa siêu phẳng đến 2 điểm dữ liệu gần nhất tương ứng với 2 phân lớp.



Hình 2: Thông tin margin của SVM

SVM cố gắng tối ưu thuật toán bằng cách tìm cách maximize giá trị margin này, từ đó tìm ra siêu phẳng đẹp nhất để phân 2 lớp dữ liệu.

2.1.2. Support vectors.

Bài toán của chúng ta trở thành tìm ra 2 đường biên của 2 lớp dữ liệu (ở hình bên trên là 2 đường xanh lá cây) sao cho khoảng cách giữa 2 đường này là lớn nhất.

Đường biên của lớp xanh sẽ đi qua một (hoặc một vài) điểm xanh.

Đường biên của lớp đỏ sẽ đi qua một (hoặc một vài) điểm đỏ.

Các điểm xanh, đỏ nằm trên 2 đường biên được gọi là các support vector, vì chúng có nhiệm vụ support để tìm ra siêu phẳng.

Đó cũng là lý do của tên gọi thuật toán Support Vector Machines.

2.1.3. Soft Margin.

Margin được chia thành 2 loại đó là Soft Margin và Hard Margin.

Để tránh overfitting, nhiều khi để muốn có margin cao, ta chấp nhận việc một vài data có thể không được chia chính xác (ví dụ như 1 bóng xanh bị lọt sang vùng của bóng đỏ). Data này được gọi là nhiễu.

⇒ Margin trong trường hợp này gọi là Soft margin.

Hard margin ám chỉ việc tìm được Margin mà không nhiễu (tất cả các data đều thỏa mãn sự phân chia). Với các bài toán thực tế, việc tìm được Hard Margin nhiều khi là bất khả thi, vì thế việc chấp nhận sai lệch ở một mức độ chấp nhận được là vô cùng cần thiết.

Trong cài đặt SVM, người ta giới thiệu tham số C với quy ước:

- C là một số dương vô cùng: đồng nghĩa với việc không cho phép sai lệch, đồng nghĩa với Hard Margin.

- C lớn: cho phép sai lệch nhỏ, thu được margin nhỏ.

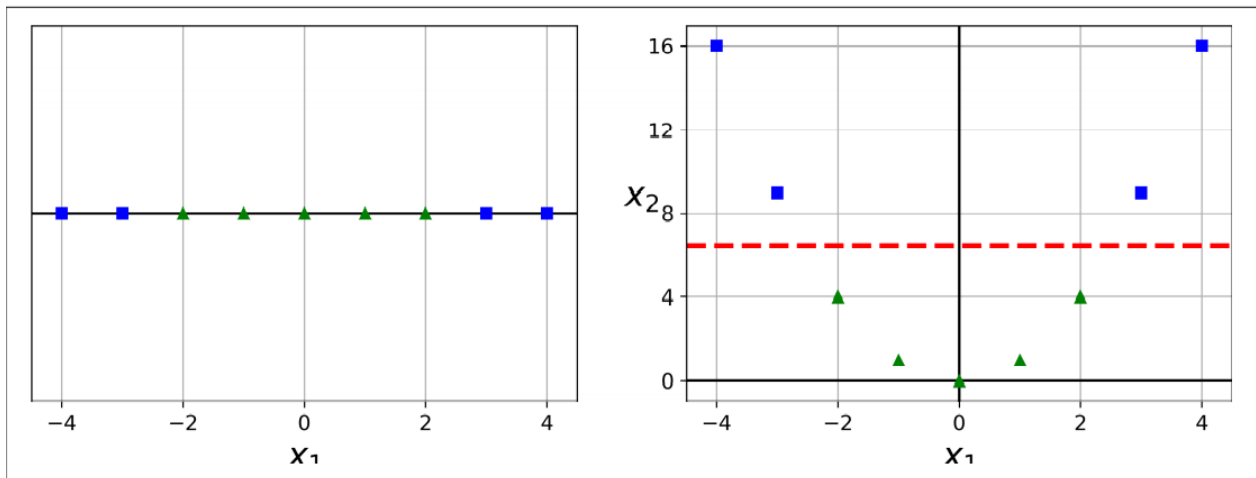
- C nhỏ: cho phép sai lệch lớn, thu được margin lớn.

⇒ Tùy bài toán cụ thể mà ta cần điều chỉnh tham số C này để thu được kết quả tốt nhất.

2.2. Nonlinear SVM Classification.

2.2.1. Cách thực hiện.

Đối với nonlinear dataset thì ta chỉ cần thêm những features (polynomial feature). Ví dụ với hình bên trái, khi ta chỉ có một feature x_1 thì dataset không được phân chia rõ ràng. Nhưng nếu ta thêm một feature x_2 sẽ cho ra kết quả như tấm hình thứ 2, dataset có ranh giới phân chia rõ ràng.



Hình 3: Xác định class sau khi thêm feature bậc cao

2.2.2. Gaussian RBF Kernel.

$$\phi_{\gamma}(\mathbf{x}, \ell) = \exp \left(-\gamma \| \mathbf{x} - \ell \|^2 \right)$$

Hình 4: Công thức thực hiện SVM với Gaussian RBF Kernel

2.3. Random Forest.

Random Forest là một sự kết hợp của nhiều những Decision Trees. Thuật toán Decision Tree sẽ phân tập dữ liệu ra làm 2 nhánh dựa vào threshold của mỗi feature, nếu dữ liệu không tìm thấy threshold thì nó sẽ tiếp tục phân nhánh nhờ dùng đệ quy. Cuối cùng nó sẽ cho ra những tập class khác biệt với độ chính xác khá là cao.

2.3.1. Extra-Trees.

Thay vì tìm threshold tốt nhất trong dữ liệu như Decision Trees, Extremely Randomized Tree hay gọi tắt là Extra-Trees sẽ tìm threshold bằng cách random threshold đó với mỗi feature trong models. Nhờ điều này khiến cho Extra-Trees nhanh hơn so với Random.

3. Xử lý dữ liệu

3.1. Import dữ liệu

Ta cần install thư viện emnist, sau đó chọn phần tách, ở đây nhóm chọn phần tách ByMerge như đã trình bày ở trên, và lấy ra bộ train set và test set, X là các feature, còn y là label.

Bộ train set có tổng 697932 samples, mỗi sample có kích thước 28x28 pixel.

Bộ test set có tổng 116323 samples.

```
[17] ▶ ▶ MI

#Get train test.
from emnist import extract_training_samples
X_train, y_train_before = extract_training_samples('bymerge')

#Get test set.
from emnist import extract_test_samples
X_test, y_test_before = extract_test_samples('bymerge')

#In ra kiểm tra số lượng image train và test.
#X_train, X_test là samples, y_train, y_test là nhãn.
print("x_train shape:", X_train.shape, "y_train shape:", y_train_before.shape)
print("x_test shape:", X_test.shape, "y_test shape:", y_test_before.shape)]

x_train shape: (697932, 28, 28) y_train shape: (697932,)
x_test shape: (116323, 28, 28) y_test shape: (116323,)
```

Hình 5: Lấy bộ dữ liệu

3.2. Reshape dữ liệu

Ta cần đưa mỗi sample từ mảng 2 chiều 28x28 pixel thành một mảng một chiều 784 features, thực hiện trên cả hai bộ test set và train set.

```
[19] ▶ ▶ MI

#Đổi matrix 28*28 pixel, thành một mảng 1 chiều 784 phần tử.
X_train = X_train.reshape(697932,784)
X_test = X_test.reshape(116323,784)
print(X_train.shape)

(697932, 784)
```

Hình 6: Reshape dữ liệu

3.3. Feature scaling

Trước khi đưa dữ liệu vào train, ta cần thực hiện chuẩn hoá dữ liệu (Feature scaling).

Dữ liệu có thể đến từ nhiều nguồn, với đơn vị, các thành phần có giá trị chênh lệch lớn. Để tính toán ta cần đưa tất cả dữ liệu về một chuẩn chung, các thành phần có giá trị nằm cùng trong 1 khoảng như [0, 1].

Sử dụng StandardScaler có sẵn trong sklearn để thực hiện feature scaling, sau khi xong thì ta lưu dữ liệu đã được scale lại để sử dụng lần sau.

```
from sklearn.preprocessing import StandardScaler
if 0:
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
    joblib.dump(X_train_scaled, 'saved_var/data/X_train_scaled')
else:
    X_train_scaled = joblib.load('saved_var/data/X_train_scaled')
print('done')
```

done

Hình 7: Thực hiện scaling dữ liệu

4. Tạo model bằng random forest

4.1. Trainning data

Đối với riêng thuật toán Random forest, dữ liệu không cần phải scale trước khi train, vì nó đã rất tốt rồi, thậm chí có trường hợp scale dữ liệu sẽ bị giảm accuracy đối với bộ dữ liệu MNIST.

```
#===== RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
#Fit với data chưa được scale
if 0:
    rf_clf = rf_clf.fit(X_train, y_train)
    joblib.dump(rf_clf, 'saved_var/rf_clf')
else:
    rf_clf = joblib.load('saved_var/rf_clf')
print('học xong RandomForestClassifier')
```

học xong RandomForestClassifier

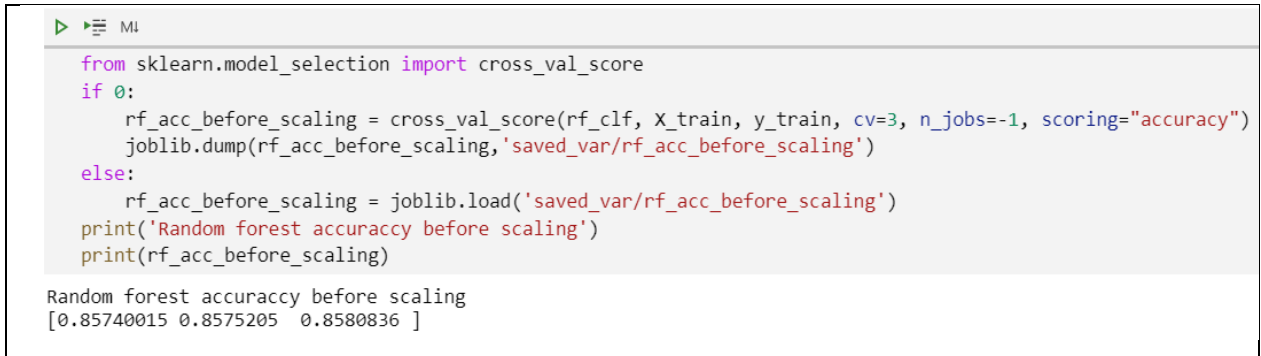
Hình 8: Học với thuật toán RandomForestClassifier

4.2. Đánh giá model

4.2.1. Accuracy score

Nhóm thực hiện tính accuracy score bằng phương pháp cross validation với 3 folds.

Ta thấy sau khi chạy 3 folds, accuracy của model tầm khoảng 0.857, đối với bộ dữ liệu này thì 0.857 là một con số khá cao, và cao hơn nhiều so với các thuật toán SGD và Decision tree.

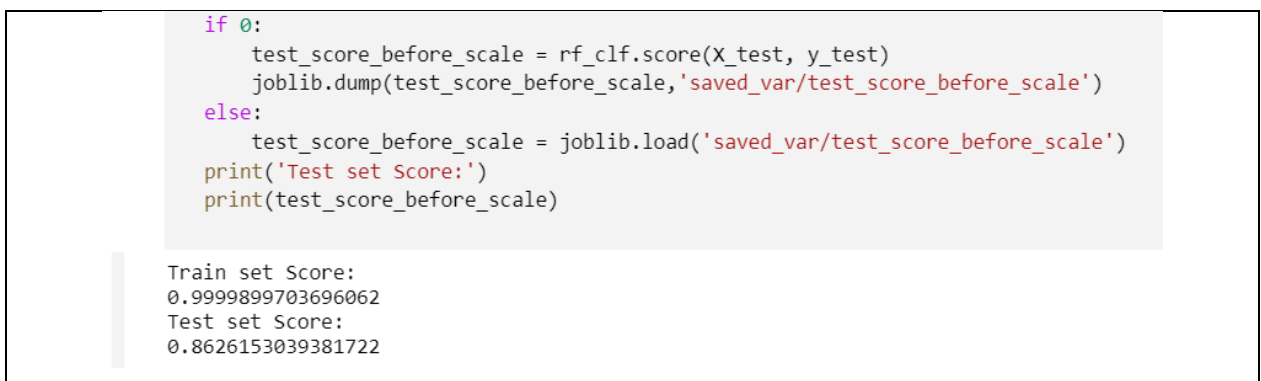


```
from sklearn.model_selection import cross_val_score
if 0:
    rf_acc_before_scaling = cross_val_score(rf_clf, X_train, y_train, cv=3, n_jobs=-1, scoring="accuracy")
    joblib.dump(rf_acc_before_scaling, 'saved_var/rf_acc_before_scaling')
else:
    rf_acc_before_scaling = joblib.load('saved_var/rf_acc_before_scaling')
    print('Random forest accuraccy before scaling')
    print(rf_acc_before_scaling)

Random forest accuraccy before scaling
[0.85740015 0.8575205 0.8580836 ]
```

Hình 9: Đo hiệu suất validation accuracy.

Tiếp theo ta sẽ tính accuracy trên tập test.



```
if 0:
    test_score_before_scale = rf_clf.score(X_test, y_test)
    joblib.dump(test_score_before_scale, 'saved_var/test_score_before_scale')
else:
    test_score_before_scale = joblib.load('saved_var/test_score_before_scale')
    print('Test set Score:')
    print(test_score_before_scale)

Train set Score:
0.9999899703696062
Test set Score:
0.8626153039381722
```

Hình 10: Xem kết quả score của tập train và test

- Score trên tập train set có mức chính xác rất cao, gần như bằng 1.
- Score trên tập test set có mức chính xác cũng rất tốt, khoảng 0.862.

⇒ Ta thấy score cho ra với random forest là rất tốt.

4.2.2. Precision và recall và f1 scores

- Tiếp theo ta sẽ tính precision và recall và f1 scores cho mỗi class.
- Ta sẽ import metrics từ sklearn để tính các điểm precision, recall và F1 score cho mỗi class.

```
from sklearn import svm, metrics
f1_scoresprint("PREDICT & RESULT")

if 0:
    predict = rf_clf.predict(X_test_scaled)
    ac_score = metrics.accuracy_score(y_test, predict)
    cl_report = metrics.classification_report(y_test, predict)
```

Hình 11: Tính giá trị Predict, Accuracy

- Kết quả 10 class đầu:

	Score = 0.8626153039381722				
	precision	recall	f1-score	support	
0	0.67	0.80	0.73	5745	
1	0.68	0.90	0.77	6400	
2	0.90	0.96	0.93	5765	
3	0.95	0.98	0.96	5827	
4	0.92	0.96	0.94	5498	
5	0.89	0.89	0.89	5326	
6	0.93	0.97	0.95	5787	
7	0.97	0.98	0.97	5873	
8	0.90	0.95	0.93	5655	
9	0.89	0.97	0.93	5651	
10	0.86	0.90	0.88	1058	

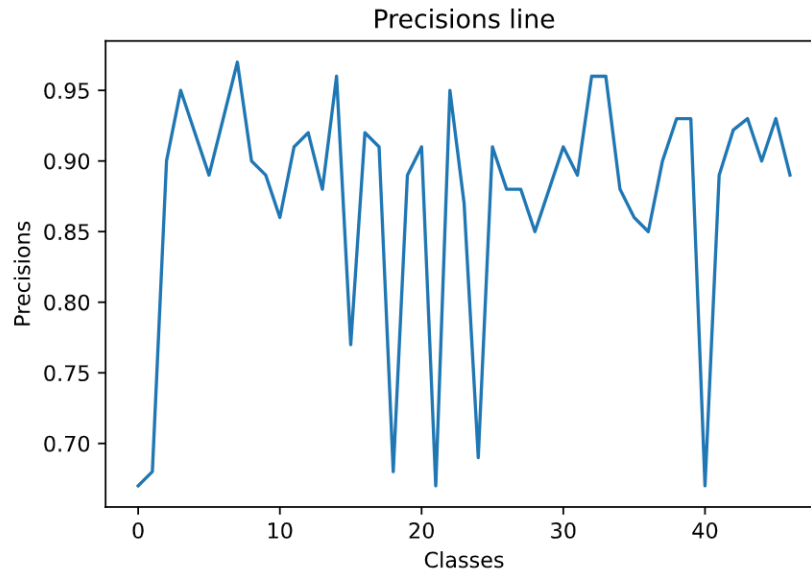
Hình 12: Điểm số Precision, Recall, F1-score.

Ngoài ra metrics còn trả về một số thông số như macro - average và weighted - average. Macro-average precision, recall là trung bình cộng của các precision, recall cho từng class.

accuracy				0.86	116323
macro avg	0.87	0.79	0.81		116323
weighted avg	0.86	0.86	0.86		116323

Hình 13: Giá trị trung bình accuracy từng class.

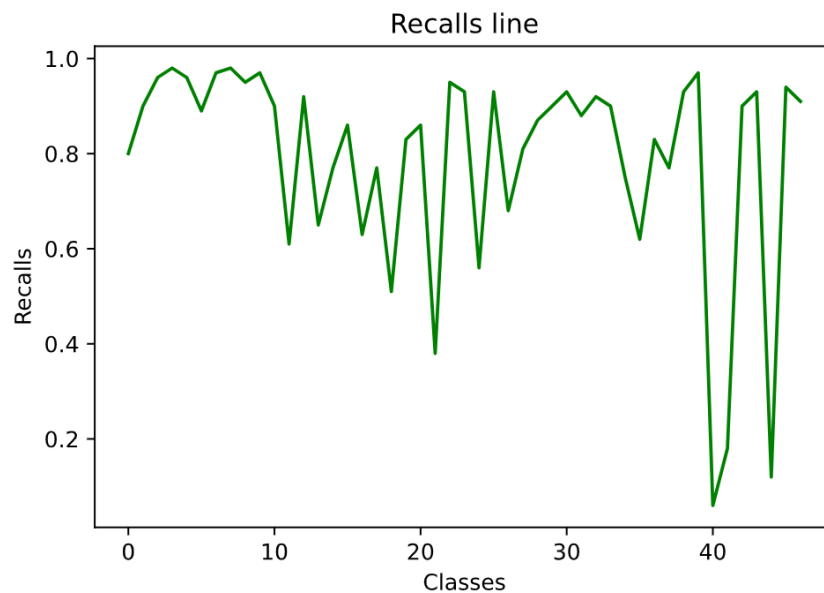
- Biểu đồ cho đường precision.



Hình 14: Biểu đồ đường Precision của RandomForest

Nhìn chung precision của các class khá cao trung bình nằm vào khoảng gần 0.9, tuy nhiên có một vài class chưa được tốt, tương ứng là class 0, 1, 15, 18, 12, 24 và 40.

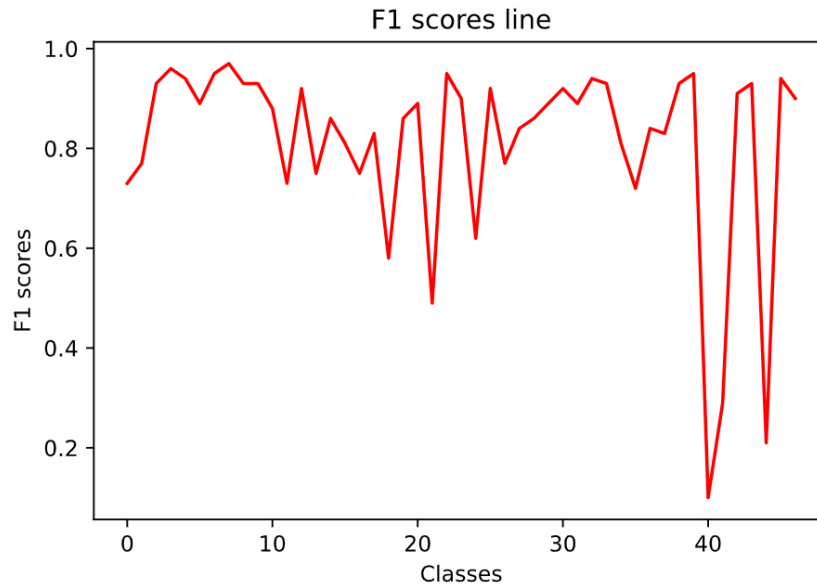
- Biểu đồ cho đường recall.



Hình 15: Biểu đồ đường Recalls của RandomForest

Trung bình điểm recall của các class nằm khoảng trên 0.8, ở mức tương đối tốt, tuy nhiên có một số class có recall thấp và cực thấp, đến mức không chấp nhận được như class 18, 21, 24 và đặc biệt là class 40 và 44 có recall chỉ ở mức khoảng 0.1.

- Biểu đồ cho đường F1 scores:



Hình 16: Biểu đồ đường F1-scores của RandomForest

Ta thấy class 18, 21, 24 có điểm số khá thấp, đặc biệt là class 40 và 44 cực thấp, không thể dùng để dự đoán.

4.2.3. Dự đoán thử

Ta tiến hành predict thử 20 samples đầu trong tập test set.

```
# [5] Try Predict
print("Random forest predict:")
print(rf_clf.predict(x_test[:20]))
print("Actual result:")
print(y_test[:20])
```

Random forest predict:
[0. 22. 28. 0. 45. 8. 5. 1. 4. 3. 22. 45. 19. 1. 5. 24. 6. 1.
39. 6.]

Actual result:
[24. 22. 28. 0. 5. 8. 5. 1. 4. 3. 22. 45. 19. 1. 5. 24. 6. 21.
39. 6.]

Hình 17: Dự đoán thử với 20 samples

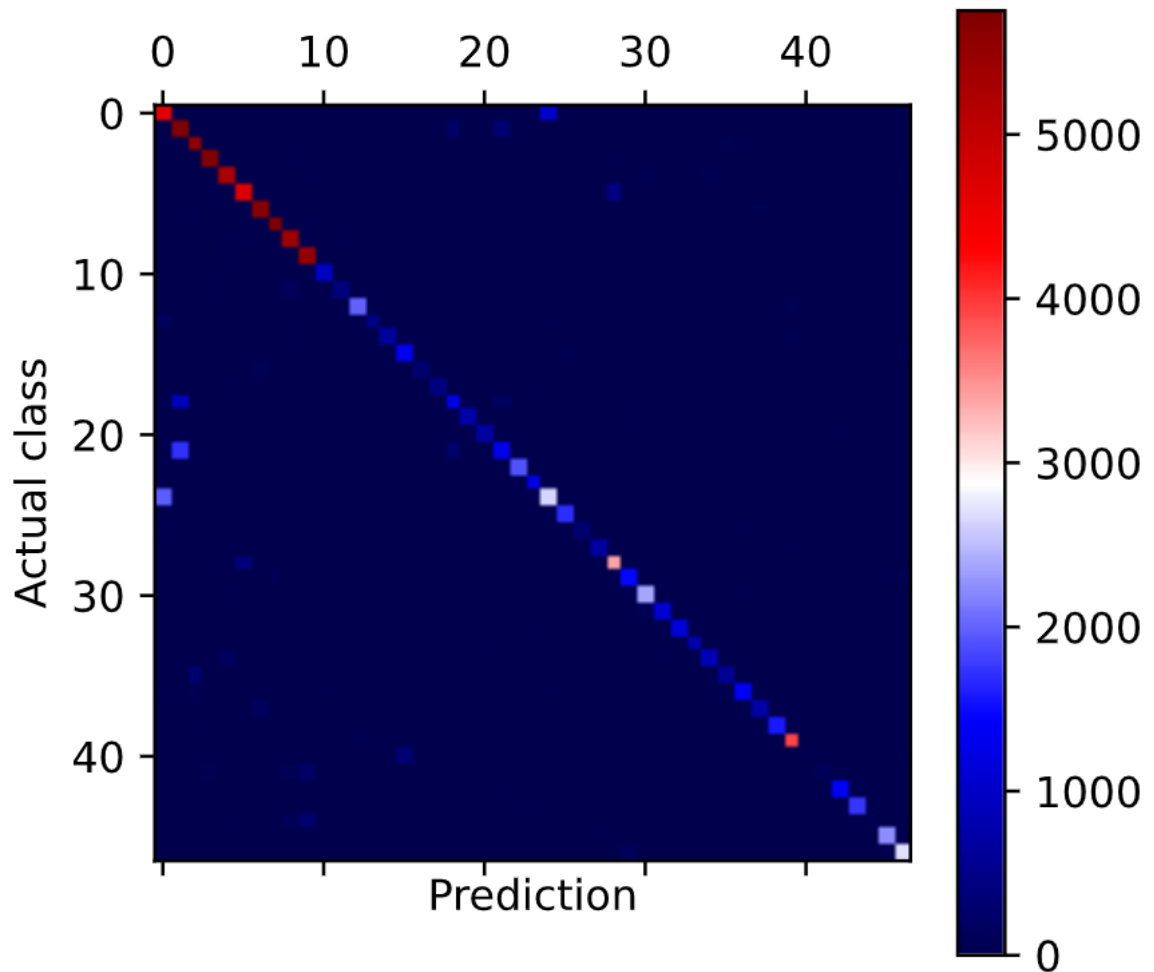
Ta thấy model của chúng ta dự đoán tương đối tốt.

4.3. Phân tích lỗi

4.3.1. Confusion matrix

Để phân tích lỗi, ta cần predict hết các samples trong tập test set và kiểm tra với label thật sự, sau đó lưu lại kết quả predict.

Công cụ để chúng ta có thể phân tích lỗi hiệu quả, chi tiết và được dùng phổ biến là Confusion matrix.

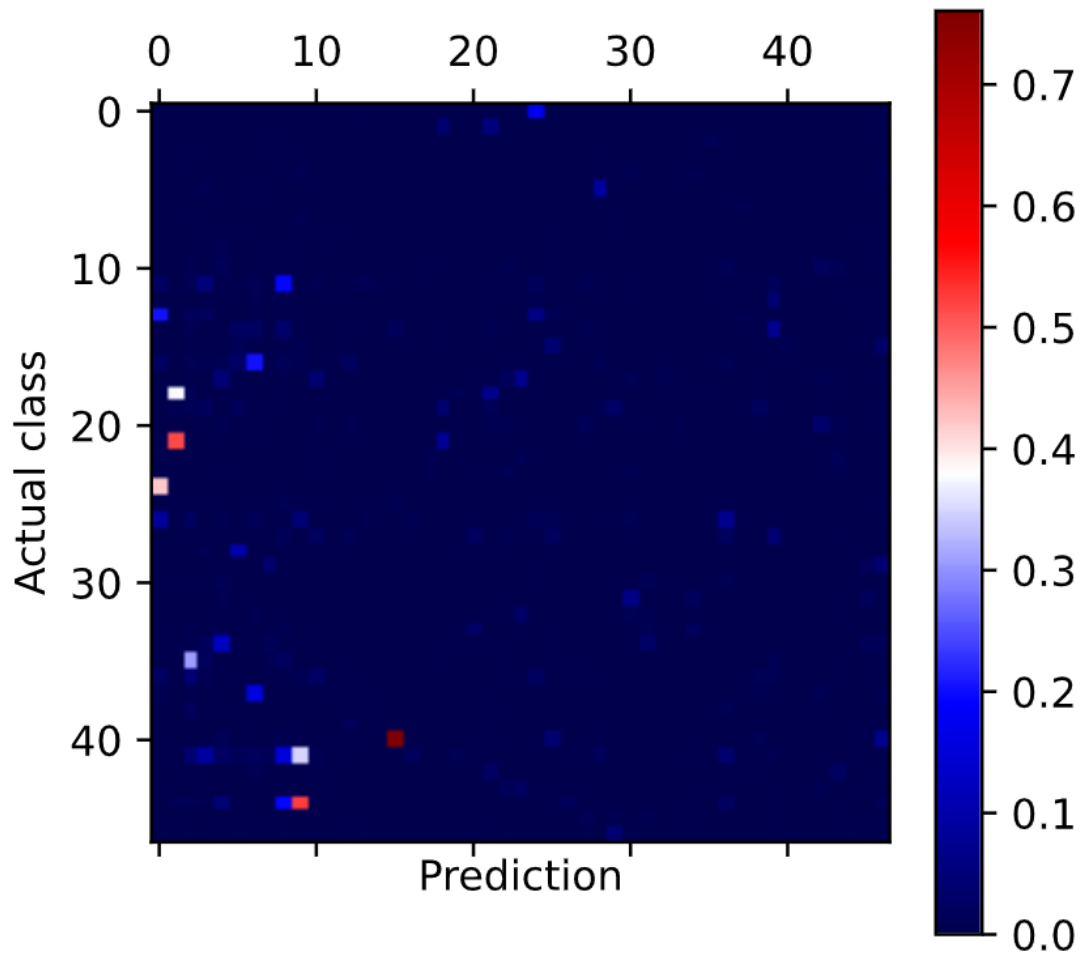


Hình 18: Confusion matrix

- Confusion matrix được vẽ trên thể hiện mức độ chính xác với từng class, mức độ chính xác tăng tương ứng từ gang màu xanh lên màu đỏ.
- Trục X là dự đoán còn trục Y là class thật sự.

Báo cáo thực tập tốt nghiệp

- Trong hình ta thấy đường chéo chính nó nổi bật lên, tức là chúng ta đã train được model tốt, dự đoán khá chính xác, đặc biệt là dự đoán các con số từ 0 đến 9, màu của chúng nổi bật lên hẳn.
- Tuy nhiên vẫn có một số ô trên đường chéo chính rất mờ nhạt như:
 - Class 40: chữ 'f'.
 - Class 41: chữ 'g'.
 - Class 44: chữ 'q'.
- Cũng có một số ô không nằm trên đường chéo chính nhưng nổi bật lên trong matrix như:
 - Hàng 0 cột 24, tức số 0 dự đoán là chữ 'o, O'.
 - Hàng 18 cột 1, tức chữ 'I, l' dự đoán là số 1.
 - Hàng 24 cột 0, tức chữ 'O, o' dự đoán là số 0.
- Ta thấy ở đây có 2 ô đối xứng nhau qua đường chéo chính đó là class 0 là class 24 – tức số 0 và chữ 'O, o', hai class này khiến model của chúng ta bị 'bối rối', nó nhầm lẫn giữa hai class này rất nhiều.
- Để cho dễ quan sát và phân tích lỗi hơn thì ta sẽ đưa đường chéo chính không còn nổi lên nữa, ta sẽ đưa nó về bằng 0. và các lỗi sẽ nổi lên.
- Giờ ta sẽ tính theo tỉ lệ để so sánh sai khác giữa các class khác nhau, bỏ qua số lượng phần tử của nó, lấy tỉ lệ từ 0 đến 1. ví dụ số 0 nhầm qua số 1 bao nhiêu phần trăm.



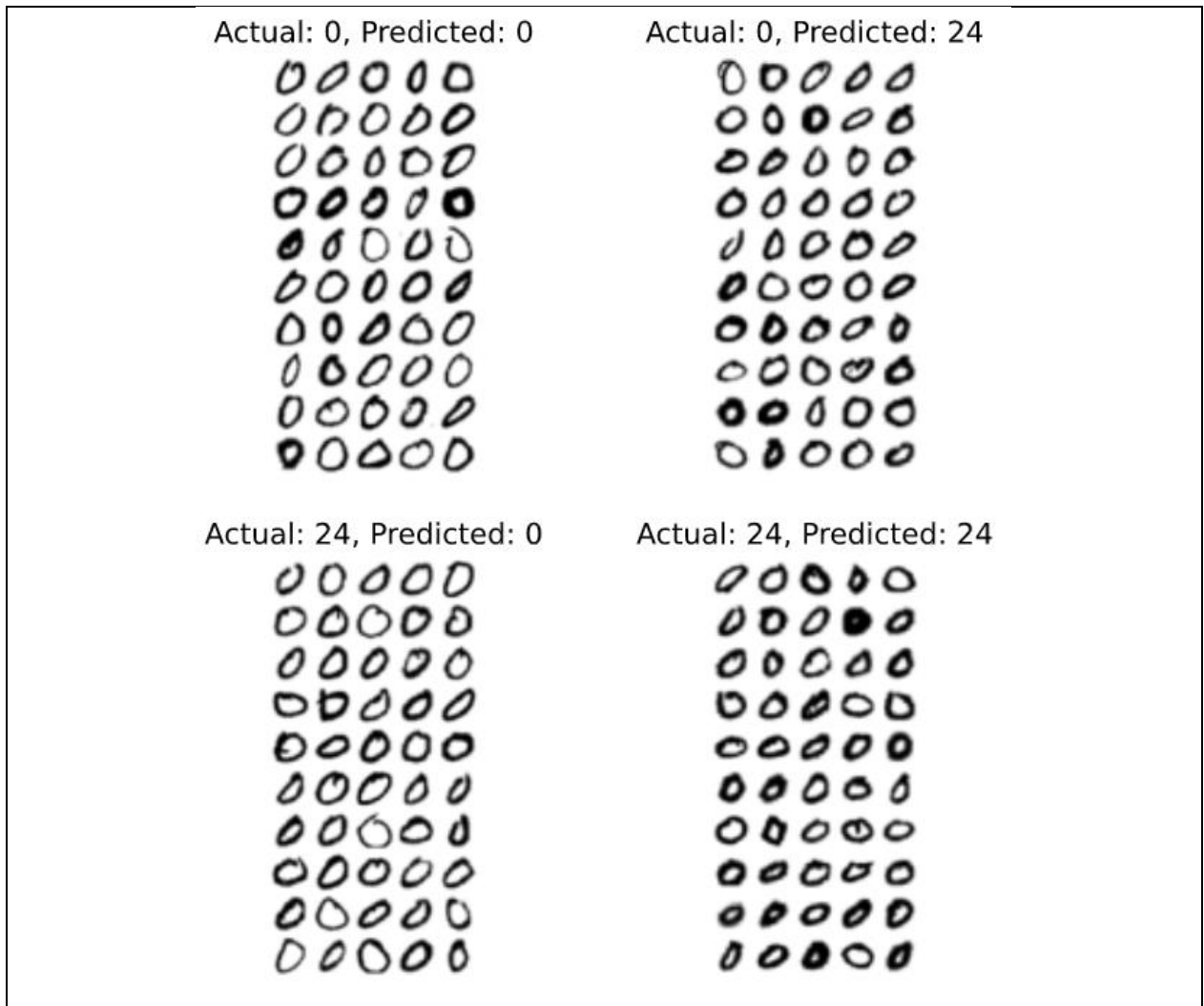
Hình 19: Confusion matrix ẩn đường chéo chính

- Ta thấy ở đây model dự đoán class 40 nhầm sang class 15 là rất nhiều, ở mức khoảng 0.75.

4.3.2. Kiểm tra dữ liệu

Ta sẽ kiểm tra xem có vấn đề gì với các class này khiến model predict sai như vậy.

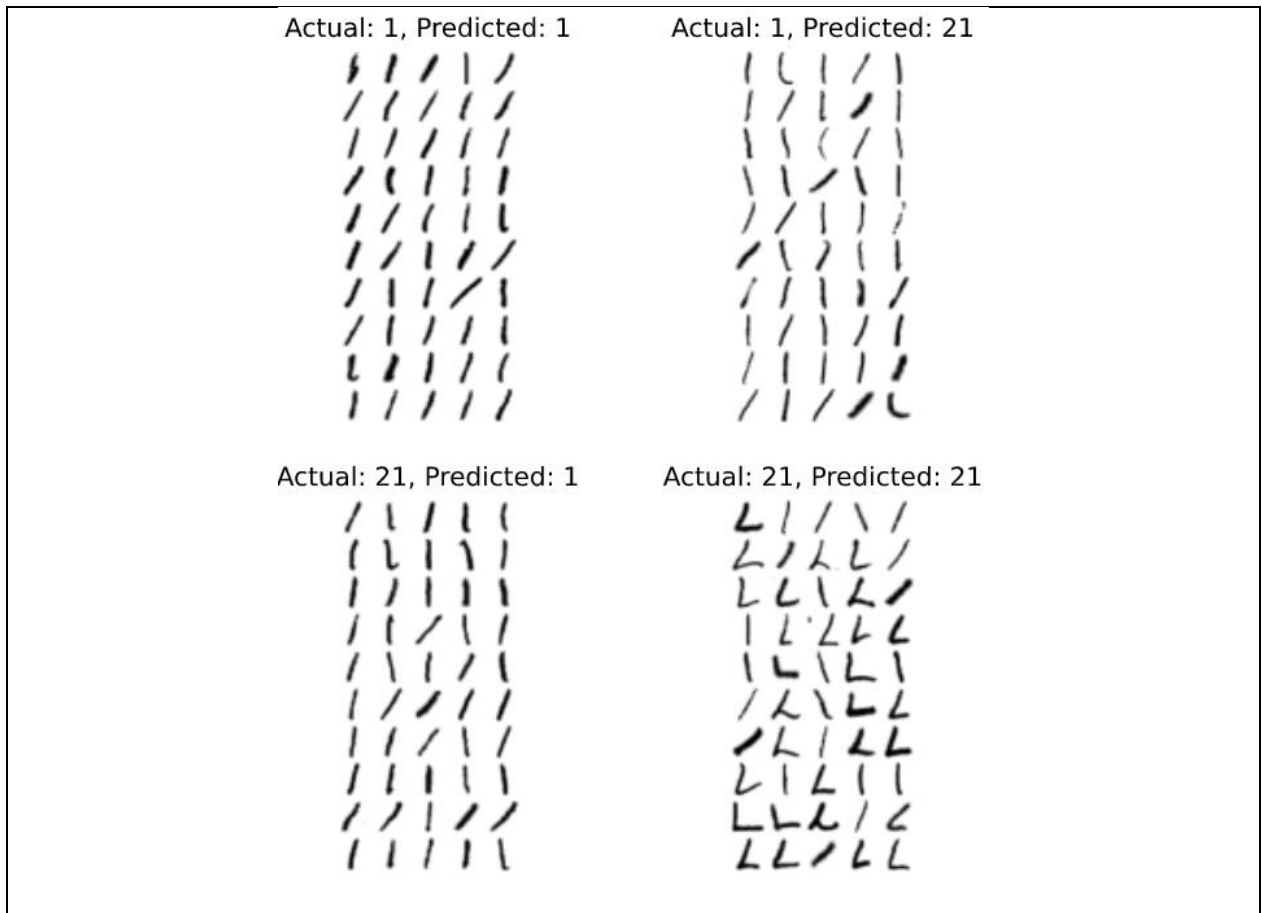
- Class 0 và class 24:



Hình 20: Xem class 0 và 24

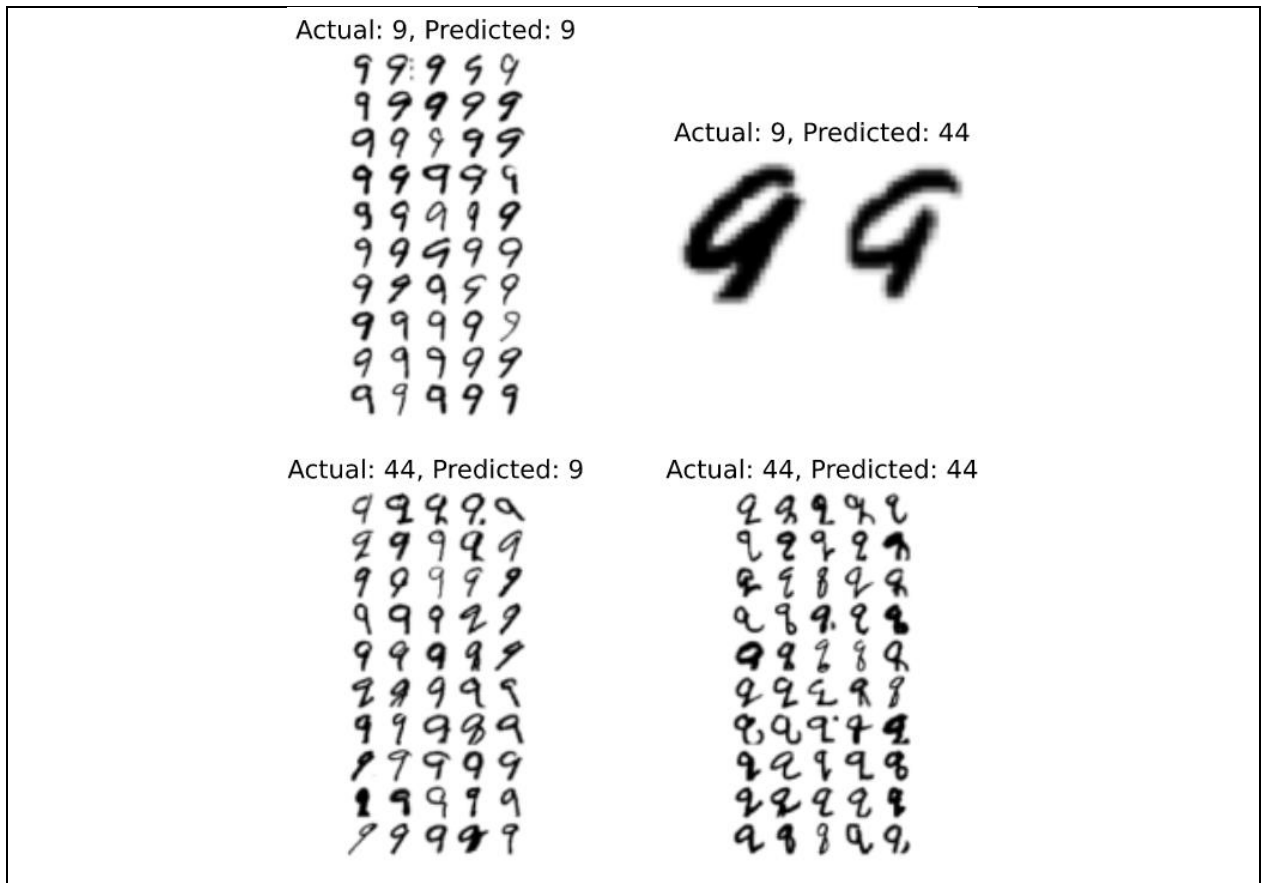
Ta thấy thật sự 2 class này rất giống nhau, khiến model của chúng ta không thể phân biệt được 2 class này.

- Class 1 và class 21:



Hình 21: Xem class 1 và 21

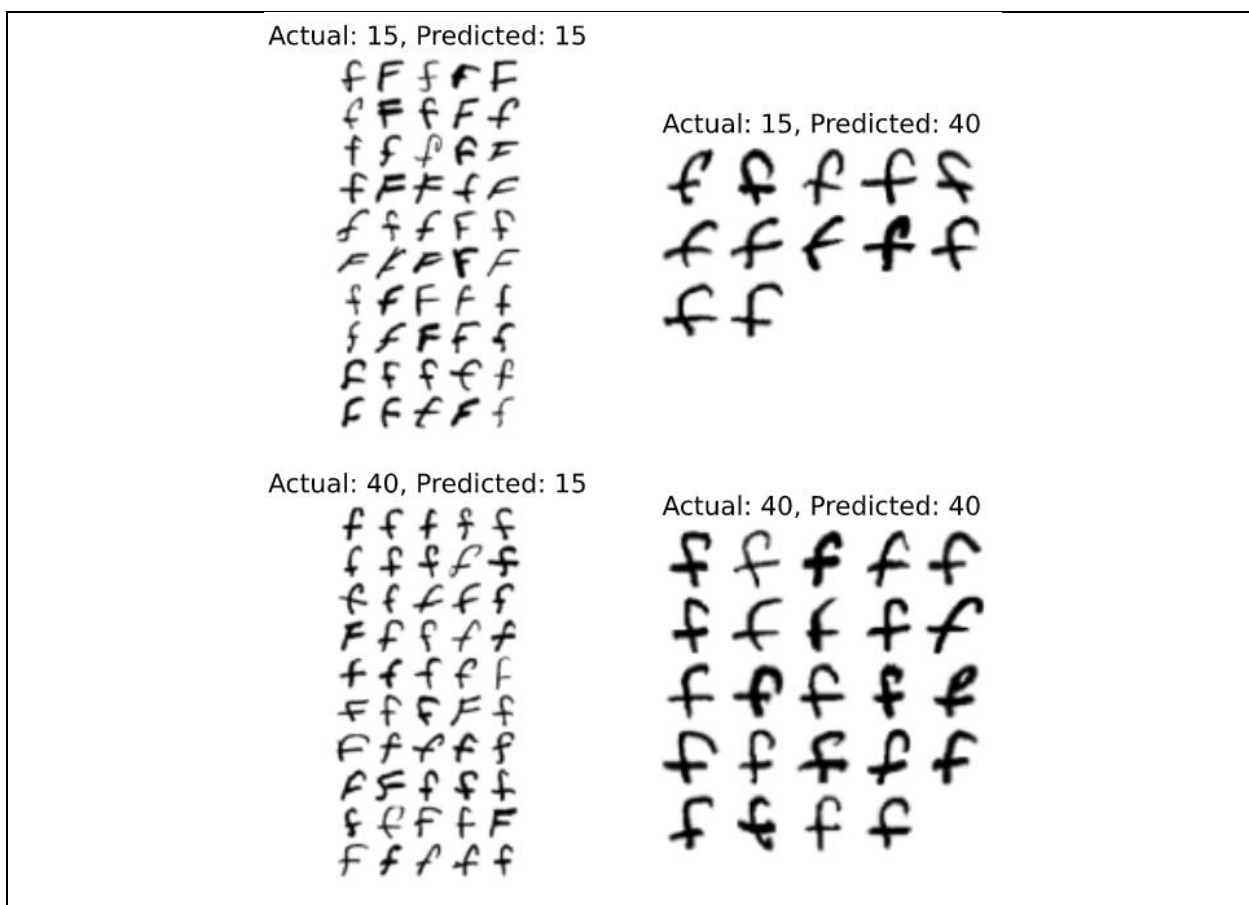
- Đối với class 1 và class 21 (chữ '1', 'L'), ta thấy dường như model chỉ phân biệt được ký tự 'L', còn '1' thì dự đoán sai sang số 1 rất nhiều.
- Class 9 và 44:



Hình 22: Xem class 9 và 44

Với thống kê trên thì ký tự ‘q’ có độ chính xác rất thấp, phần lớn nó bị nhầm qua class 9, ta thấy trong bộ dữ liệu, rất nhiều hình ảnh số 9 và ký tự ‘q’ rất giống nhau, không thể phân biệt được, chỉ có thể phân biệt các ký tự ‘q’ có móc.

- Class 15 và 40:



Hình 23: Xem class 15 và 40

Ta thấy, ký tự ‘f’ bị nhầm qua ký tự ‘F’ rất nhiều, nó chỉ dự đoán cho đúng cho class ‘f’ 24 lần trên toàn tập test, và kiểu viết ‘f’ và ‘F’ trong tập tài liệu này cũng rất giống nhau trong rất nhiều trường hợp.

4.4. Tối ưu model

Ta thấy, với class 15 và 40 tương ứng với ‘F’ và ‘f’, model của chúng ta dự đoán sai rất nhiều, cả precision và recall của class 40 đều rất thấp, ‘f’ và ‘F’ thật sự giống nhau trong nhiều trường hợp, khiến model của chúng ta bị “bối rối” không biết nó thật sự là class nào.

Nhóm đưa ra giải pháp là Merge 2 class này thành một, sau đó train lại data và xem score của model có được cải thiện không.

Ta sẽ duyệt qua bộ train set và test set, nếu label là 40 thì ta sẽ chuyển label về thành 15, sau đó lưu lại label của train set và test set để tiện sử dụng.

```
[3] ▶ MI

# MERGE CLASS 15 AND 40 (F & f)

if 0:
    y_train = np.array([])
    y_test = np.array([])
    for i in range(0, y_train_before.size):
        if y_train_before[i] == 40:
            y_train = np.append(y_train, [15])
        else:
            y_train = np.append(y_train, [y_train_before[i]])
    joblib.dump(y_train, 'saved_var/data/y_train')
    for i in range(0, y_test_before.size):
        if y_test_before[i] == 40:
            y_test = np.append(y_test, [15])
        else:
            y_test = np.append(y_test, [y_test_before[i]])
    joblib.dump(y_test, 'saved_var/data/y_test')
else:
    y_train = joblib.load('saved_var/data/y_train')
    y_test = joblib.load('saved_var/data/y_test')

y_test_before = None
y_train_before = None
print('done')
#

done
```

Hình 24: Kết hợp class 15 và 40

Sau đó ta sẽ tiến hành học lại model.

4.5. Đánh giá model sau khi xử lý lại dữ liệu

Đây là accuracy sau khi tiến hành học lại.

```
▶ MI

from sklearn.model_selection import cross_val_score
if 0:
    rf_acc = cross_val_score(rf_clf, X_train, y_train, cv=3, n_jobs=-1, scoring="accuracy")
    joblib.dump(rf_acc, 'saved_var/rf_acc_before_scaling')
else:
    rf_acc = joblib.load('saved_var/rf_acc_before_scaling')
print('Random forest accuracy')
print(rf_acc)

Random forest accuracy
[0.86137188 0.86108819 0.86168567]
```

Hình 25: Accuracy sau khi học lại RandomForest

Ta thấy sau khi merge hai class ‘f’ và ‘F’ lại thì accuracy đã tăng lên được 1% so với trước.

R2 score và Root Mean Square Error trên testing data:

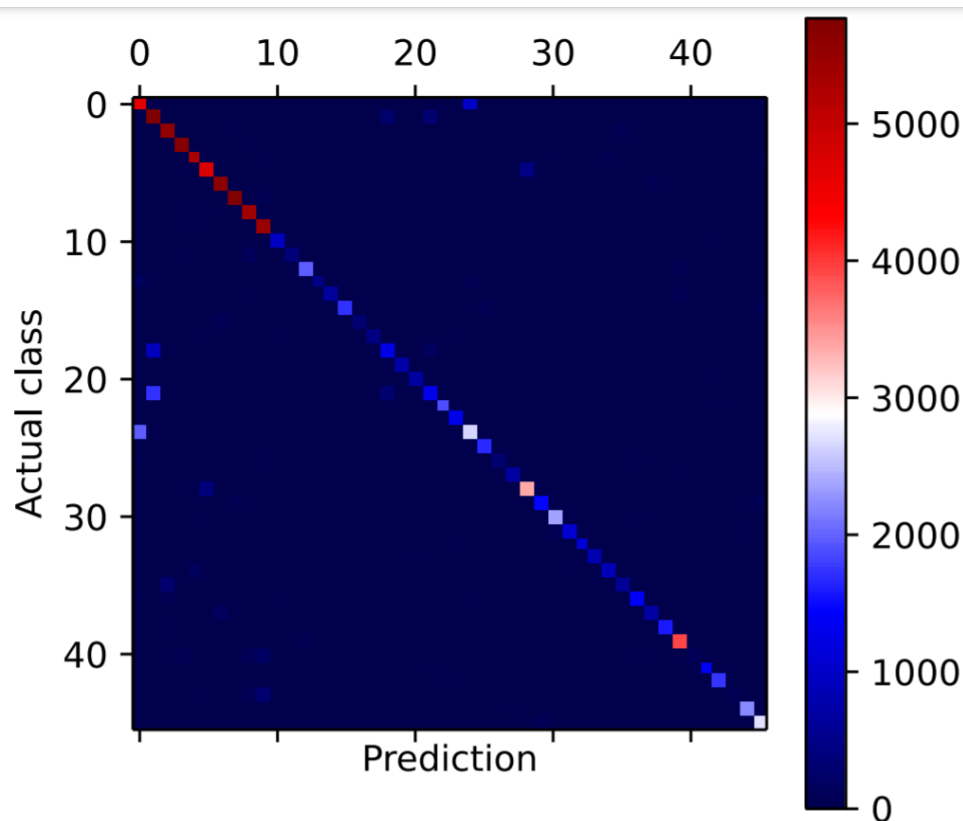
```
Calculate on Test set  
R2 score (on testing data, best=1): 0.866122778814164  
Root Mean Square Error: 8.0  
=====
```

Hình 26: Giá trị R2 và RMSE

R2 score cũng được tăng lên 1%, Root Mean Square Error tăng lên 8.

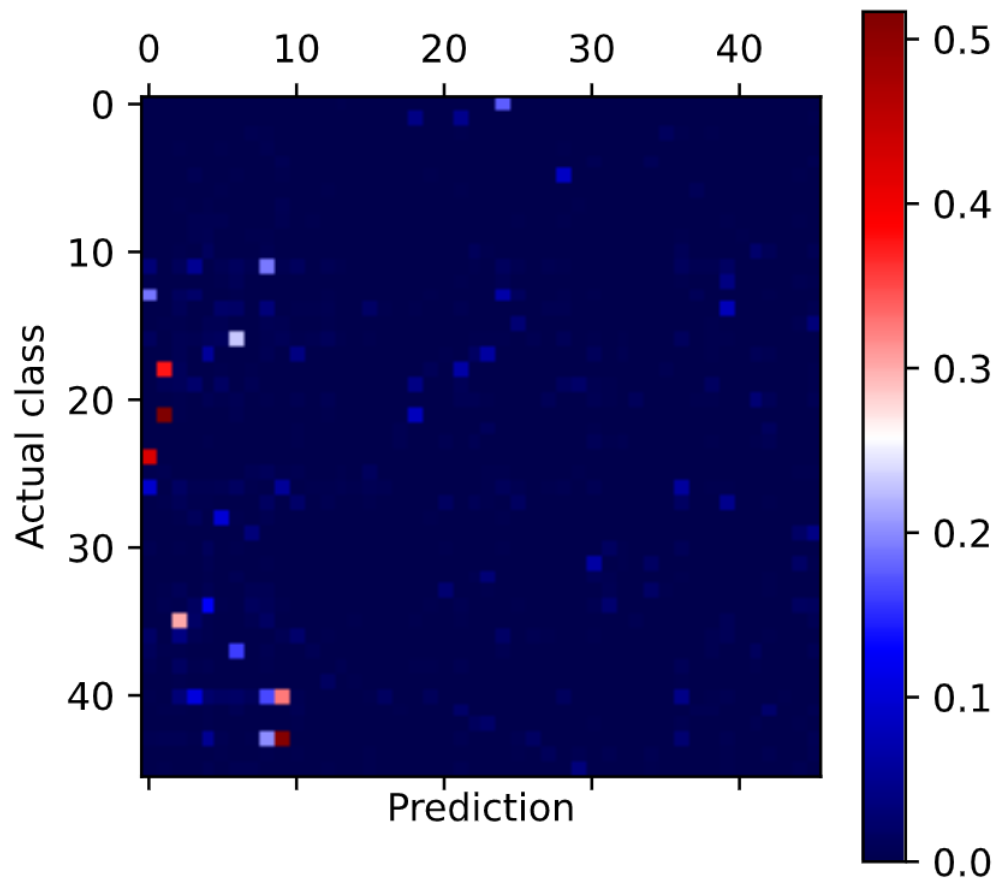
Ta thấy sau khi merge 2 class 'F' và 'f' lại thì accuracy của model đã tốt hơn trước.

(*) **Lưu ý:** khi merge class 40 vào class 15 thì nó không còn class 40 nữa, ta chỉ còn 45 class, trong các thông kê dưới thì thứ tự các class 41 đến 46 sẽ giảm xuống 1 đơn vị.



Hình 27: Confusion matrix sau khi tính lại

Dưới đây là confusion matrix khi làm nổi bật lỗi lên.



Hình 28: Confusion matrix ảnh đường chéo chính sau khi tính lại

Ta thấy lỗi đã được giảm xuống đáng kể, từ trên 0.7 xuống còn mức khoảng 0.52.

5. Tạo model bằng SVM classifier

5.1. Tiến hành train model

Sử dụng lại dữ liệu đã xử lý ở trên.

Ở đây nhóm chỉ sử dụng 100,000 sample trong tập training data để học, ở đây nhóm muốn khảo sát trước độ chính xác của thuật toán, vì thực hiện training trên SVC rất tốn thời gian, độ phức tạp rất cao.

```
MI
x_train_100000 = x_train_scaled[:100000]
y_train_100000 = y_train[:100000]
print('done')
```

Hình 29: Lấy 100000 samples

Tiếp theo ta sẽ tiến hành train data với SVC, mặc định SVC sẽ train với option là “ova” tức là one vs all, nhưng nhóm chỉ định option “ovo” để cải thiện tốc độ training.

```
from sklearn import svm, metrics
clf = svm.SVC(decision_function_shape='ovo')
clf.fit(X_train_100000, y_train_100000)
```

Hình 30: Fit model với 100k samples

5.2. Đánh giá model

5.2.1. Root Mean Square Error

```
Root mean squared Error:
8.003404126892244
```

Hình 31: Root mean squared error

5.2.2. Accuracy

5.2.3. Cross validation

```
SVM validation accuracy
[0.85316 0.85457 0.85431]
```

Hình 32: Validation accuracy

5.2.4. Accuracy trên tập testing data

```
Accuracy score:
Score = 0.8355097444185586
```

Hình 33: Accuracy on test set

Ta thấy model khi train với 100000 samples (gần 1/2 của bộ dữ liệu train set), thì accuracy trên tập testing data đã đạt được 0.84, gần bằng random forest khi train với 100% training data (0.86).

5.2.5. Precision và recall.

- Trên 10 class đầu tiên:

	precision	recall	f1-score	support
0.0	0.67	0.80	0.73	5745
1.0	0.63	0.95	0.76	6400
2.0	0.88	0.94	0.91	5765
3.0	0.95	0.96	0.95	5827
4.0	0.90	0.94	0.92	5498
5.0	0.87	0.86	0.87	5326
6.0	0.92	0.96	0.94	5787
7.0	0.96	0.96	0.96	5873
8.0	0.90	0.93	0.91	5655
9.0	0.89	0.96	0.92	5651
10.0	0.84	0.78	0.81	1058
11.0	0.81	0.56	0.66	652
12.0	0.91	0.88	0.89	2156
13.0	0.79	0.64	0.71	735
14.0	0.93	0.74	0.83	860
15.0	0.87	0.83	0.85	1950
accuracy			0.84	116323
macro avg	0.83	0.76	0.78	116323
weighted avg	0.84	0.84	0.83	116323

Hình 34: Precision, recall, f1 score

Ta thấy trung bình precision, recall và f1 score của tất cả các class nằm mở mức khá, chỉ chạy với 1/2 dữ liệu tập train mà có kết quả như vậy là khá tốt.

5.3. Parameter tuning.

5.3.1. Tuning tham số C.

Nhóm thực hiện tuning tham số C sao cho model cho ra kết quả tốt nhất.

Các parameter candidates là: $C = \{0.1, 1, 4, 5, 6, 7, 8, 10\}$. Vì thời gian tuning quá lâu, nên nhóm chia từng part để tuning, mỗi lần chỉ tuning 2 giá trị.

- Part 1, $C = \{0.1, 1\}$

```

if 0:
    tuning_01 = GridSearchCV(estimator=svm.SVC(decision_function_shape='ovo'), param_grid=parameter_candidates_part_1, n_jobs=-1)
    tuning_01.fit(X_train_100000, y_train_100000)
    joblib.dump(tuning_01, 'tuning/clf_tuning_01')
else:
    tuning_01 = joblib.load('tuning/clf_tuning_01')

print('Best score:', tuning_01.best_score_)
print('Best C:', tuning_01.best_estimator_.C)

tuning C part 1
C = {0.1, 1}
Best score: 0.8314199999999999
Best C: 1

```

Hình 35: Tuning part 1

Ta thấy với 2 giá trị 0.1 và 1 thì 1 có giá trị lớn hơn, nên ta sẽ thử tăng C lên để tiếp tục tuning.

- Part 2, $C = \{5, 10\}$

```
tunning C part 2
C = {5, 10}
Best score: 0.84588
Best C: 5
```

Hình 36: Tuning part 2

Ta thấy với 2 giá trị 5 và 10 thì 5 có giá trị lớn hơn, nên ta sẽ cho C around 5 và tìm kiếm C tốt hơn.

Part 3, $C = \{4, 6\}$

```
tunning C part 3
C = {4, 6}
Best score: 0.84612
Best C: 6
```

Hình 37: Tuning part 3

Ta thấy với 2 giá trị 4 và 6 thì 6 có giá trị lớn hơn, nên ta sẽ tăng C và tìm kiếm C tốt hơn.

- Part 4, $C = \{7, 8\}$

```
tunning C part 4
C = {7, 8}
Best score: 0.8461399999999999
Best C: 7
```

Hình 38: Tuning part 4

Ta thấy với $C = 7$ thì ta có score tốt nhất.

⇒ Lựa chọn parameter = 7.

5.3.2. Tuning tham số Gama.

Nhóm đã thực hiện tuning tham số gamma với mong muốn tăng hiệu suất của model, nhưng thực hiện tuning với tham số gamma thật sự rất mất thời gian, dù chỉ thực hiện tuning với 1/6 training data và với 2 giá trị gamma, nhưng các hệ thống hỗ trợ training online như Kaggle, Google Colab đều không thể cho ra kết quả vì vượt quá runtime limit.

Báo cáo thực tập tốt nghiệp

Nhóm sử dụng máy tính có cấu hình khá cao, nhưng sau 12 tiếng vẫn không thể cho ra kết quả, nhóm quyết định dừng việc tuning với tham số gamma vì chưa đủ điều kiện thực hiện.

5.4. Training lại model với best parameters

Sau khi tìm được tham số tốt nhất cho model thì nhóm tiến hành training lại với 100% dữ liệu training, hi vọng rằng nó sẽ cho ra kết quả tốt.

```
from sklearn import svm, metrics
clf = svm.SVC(decision_function_shape='ovo', C=7, kernel='rbf', gamma='scale')
clf.fit(X_train_300000, y_train)
joblib.dump(clf, 'saved_var/svm_clf')
```

Hình 39: Fit model again

Root Mean Square Error

```
Root Mean Square Error:  7.6
=====
```

Hình 40: Root mean squared error

Cross validation accuracy

```
SVM validation accuraccy
[0.86381 0.86505 0.86709]
```

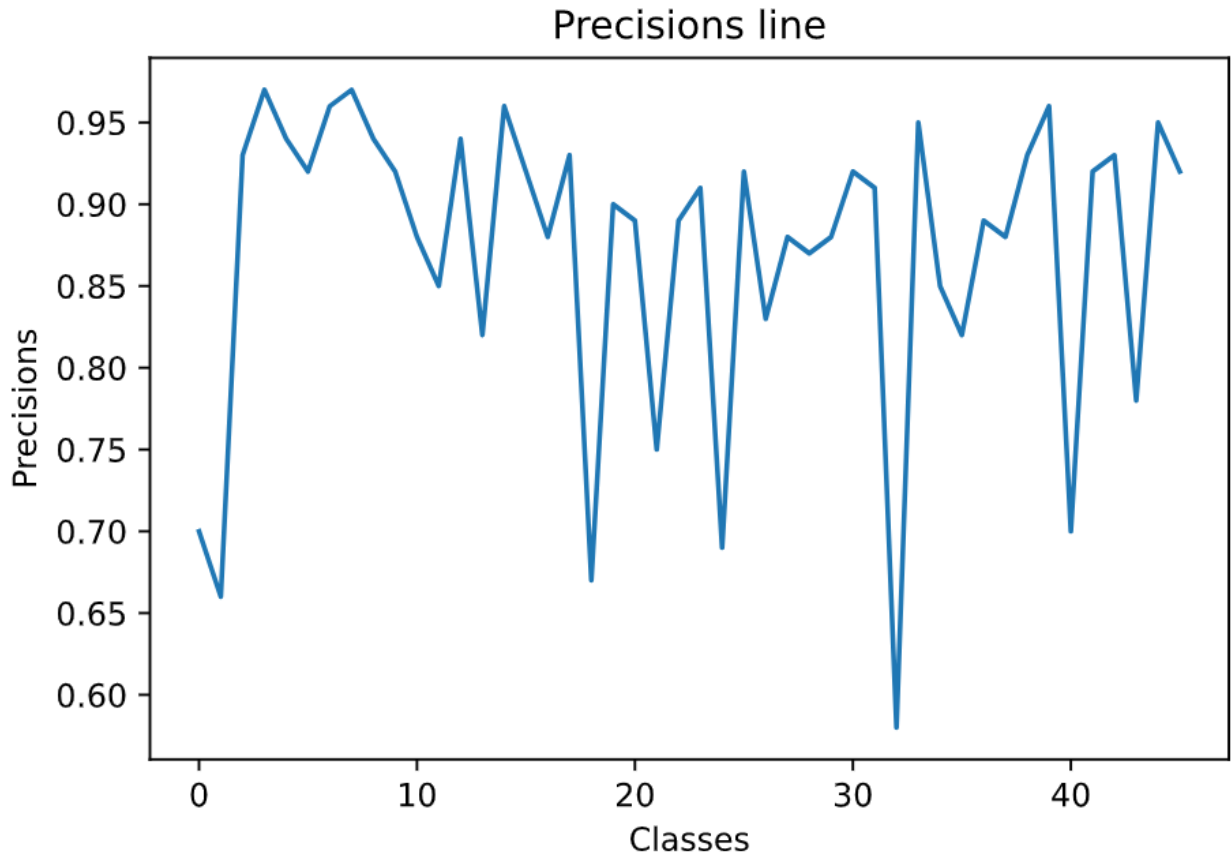
Hình 41: Validation accuracy

Accuracy trên tập testing data

```
Accuracy on test set:
0.8719857637784445
```

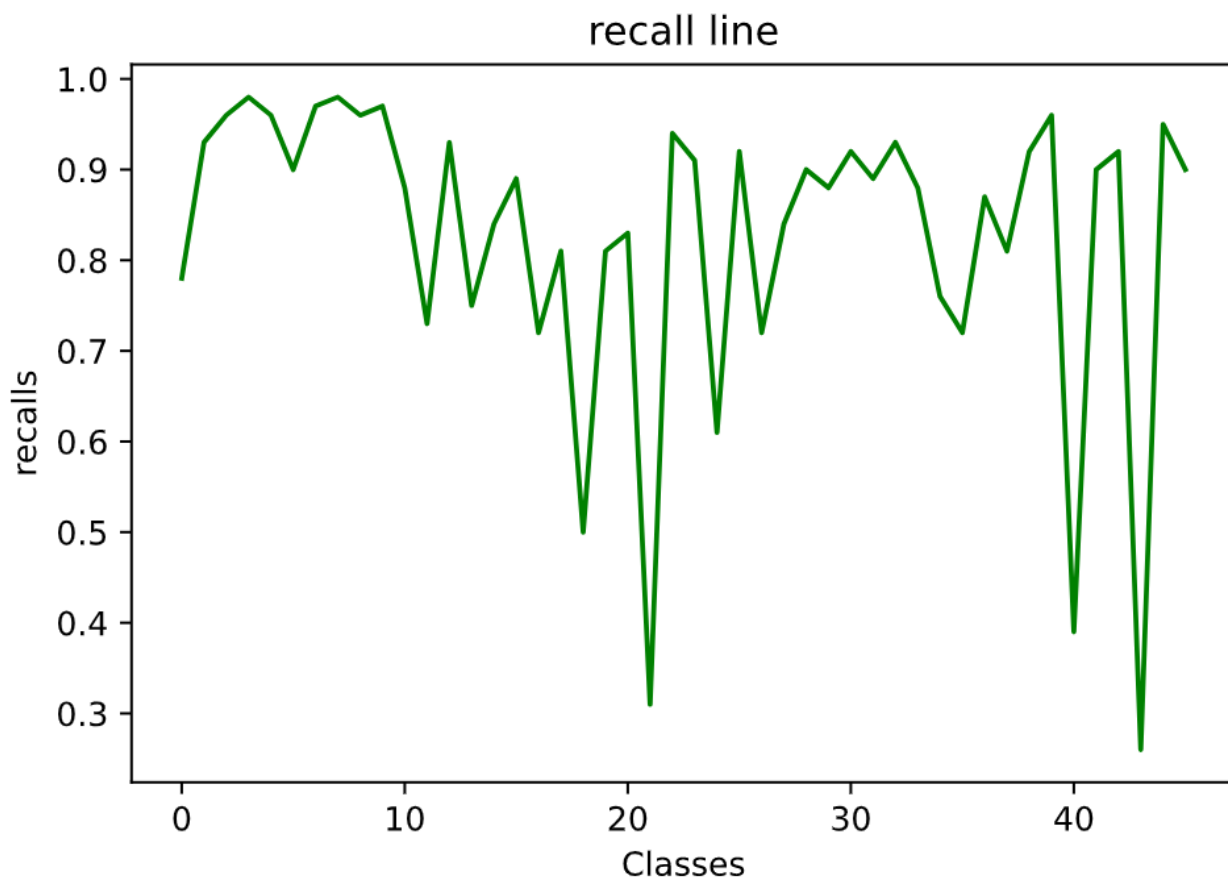
Hình 42: Accuray on test set

Precision, recall và f1 score

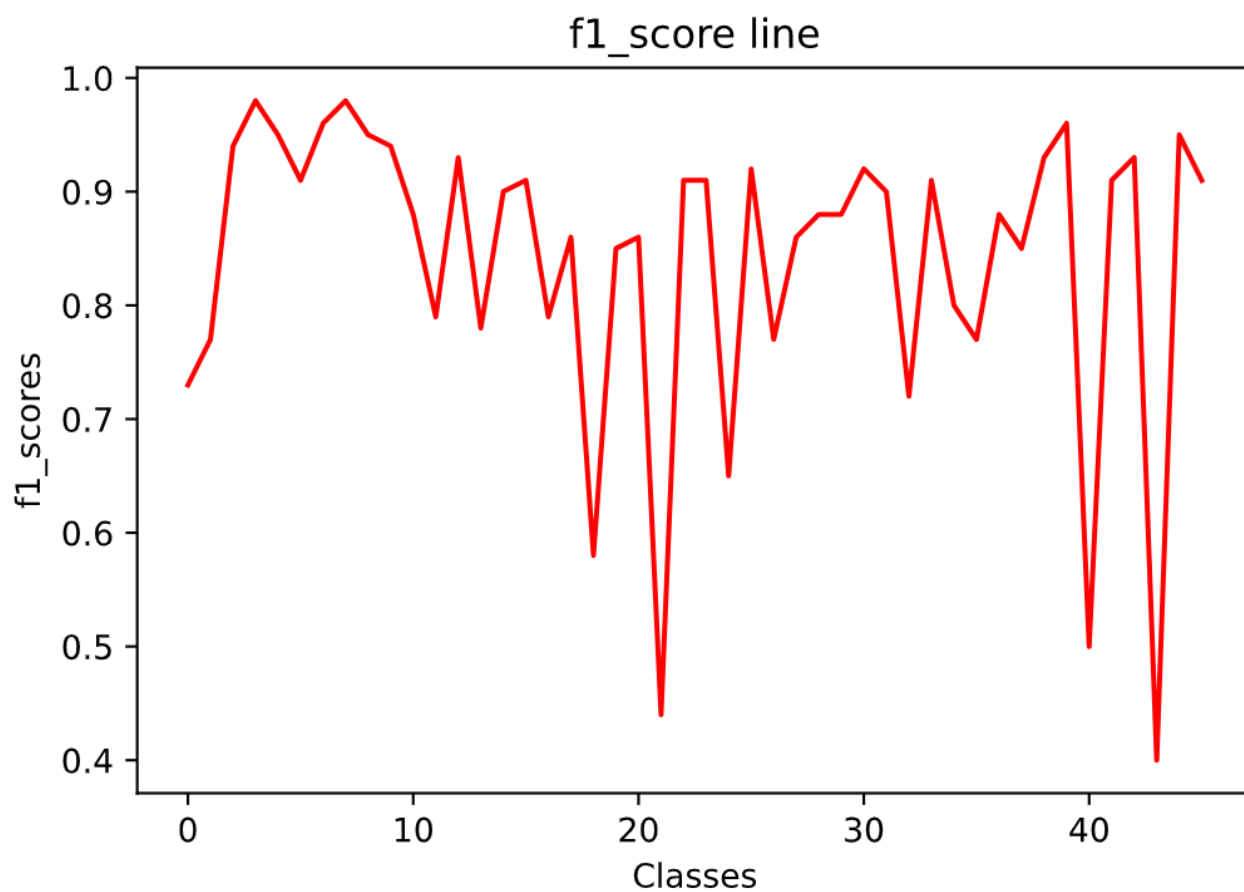


Hình 43: Precision chart

Precisions của hầu hết các class là rất tốt, đạt tới ngưỡng khá cao, nhưng có một số class lại rất thấp, đặc biệt là class



Hình 44: Recall chart



Hình 45: F1 score chart

CHƯƠNG 3: KẾT LUẬN

Với mục tiêu đề ra từ đầu là xây dựng một ứng dụng dự đoán ký tự sử dụng machine learning, nhóm đã hoàn thành được khoảng 90%. Accuracy của model tìm đạt trên 0.87, đạt được mục tiêu đề ra ban đầu là 0.85.

Sau khi hoàn thành project, nhóm thấy mình đã nắm vững kiến thức về machine learning nhiều hơn, biết cách xử lý dữ liệu sao cho hợp lý, có khả năng đánh giá model qua các điểm đánh giá, biết phân tích lỗi và tối ưu model để hiệu suất của model tăng lên.

Ngoài ra, nhóm còn tìm hiểu được nhiều thuật toán hơn khi thực hiện train với nhiều thuật toán khác nhau như Random Forest, Support Vector Machine, đây sẽ là những kiến thức vô giá cho nhóm áp dụng vào thực tế sau này.

TÀI LIỆU THAM KHẢO

[1]. <https://www.nist.gov/itl/products-and-services/emnist-dataset#:~:text=The%20EMNIST%20dataset%20is%20a,directly%20matches%20the%20MNIST%20dataset%20>.

- Status: Active

- Last visited: 09/01/2021

[2]. <https://machinelearningcoban.com/2017/04/09/smv/>

- Status: Active

- Last visited: 09/01/2021