ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO ĐỒ ÁN 2
LOGISTIC REGRESSION

Bộ môn: Thống Kê Máy Tính Và Ứng Dụng
Giảng viên hướng dẫn: Nguyễn Văn Quang Huy

2020 - 2021

MỤC LỤC

I. THÔNG TIN NHÓM VÀ MỨC ĐỘ ĐÓNG GÓP

STT	Họ và tên	MSSV	Tự đánh giá	Mức độ đóng góp
1	Du Chí Nhân	18120492	100%	100%
2	Lê Hoàng Phương Nhi	18120496	100%	100%
3	Phan Văn Võ Quyền	18120529	100%	100%
4	Lê Thị Như Quỳnh	18120530	100%	100%
5	Phạm Minh Sỹ	18120540	100%	100%

II. NỘI DUNG CHI TIẾT

1. Tập dữ liệu MNIST

- ✓ Nhóm tải xuống và đọc toàn bộ tập dữ liệu MNIST bao gồm cả phần training set và test set thông qua thư viện Tensorflow:

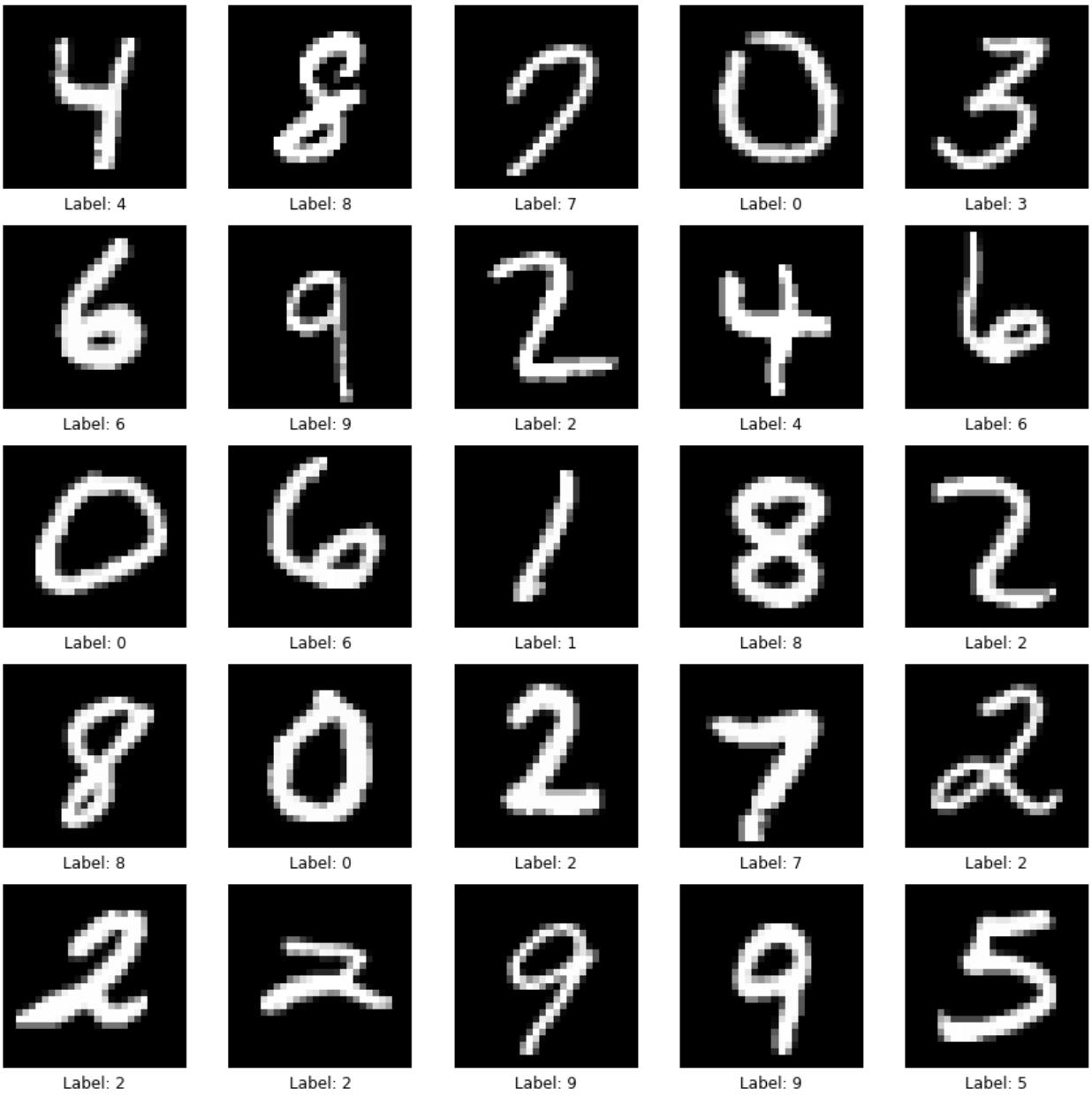
```
(X, y), (X_test, y_test) = mnist.load_data()
```

Trong đó:

- X, X_test có dạng (60000, 28, 28) và (10000, 28, 28) là mảng các mảng 2 chiều có kích thước 28 x 28 chứa các giá trị từ 0 đến 255 (mảng các bức ảnh).
- y, y_test có dạng (60000,) và (10000,) là mảng một chiều chứa các giá trị từ 0 đến 9 (mảng các nhãn).

- ✓ Lấy ngẫu nhiên một số mẫu bất kỳ trong tập dữ liệu, hiển thị hình ảnh và nhãn của mẫu:

```
np.random.seed(26)
rand_arr = np.random.choice(len(df),size=25,replace=False)
fig, axs = plt.subplots(5,5,figsize=(15,15))
j = 0
for i in rand_arr:
    ax = axs[j//5 , j%5]
    ax.imshow((X_df.iloc[i].values).reshape(28,28),cmap='gray')
    ax.axis('off')
    ax.set_title('Label: {}'.format(y_df.iloc[i]),fontsize=12,y=-0.15)
    j += 1
plt.show()
```



Hình 1: Một số mẫu trong tập dữ liệu MNIST

2. Tiền xử lý dữ liệu

2.1. Cách biến đổi dữ liệu đầu ra

- ✓ Ý tưởng:

  Sử dụng 10 ma trận thay thế dùng để phân lớp cho mỗi class với chỉ số tương ứng, mỗi ma trận dùng để phân lớp cho 1 nhãn. Với mỗi phần tử trong ma trận, nếu nhãn của ma trận đầu ra ban đầu trùng với nhãn class thì sẽ được đánh dấu là 1, ngược lại đánh dấu là 0.
- ✓ Cách thực hiện:

Sử dụng phương thức **get_dummies** của **pandas** để chuyển đổi biến phân loại thành chỉ số. Khi đó, dữ liệu đầu ra sẽ chuyển về dạng 1 **dataframe** có 10 cột (tương tự như 10 ma trận đã nêu ở ý tưởng) tương ứng với các nhãn của ảnh từ 0 – 9 và dữ liệu đầu ra ban đầu tương ứng với cột nào thì giá trị cột đó sẽ là 1, các cột còn lại là 0.

```
def output_preprocessing(y_train):
  y_train_preprocessed = pd.get_dummies(y_train)
  return y_train_preprocessed
```

2.2. Loại bỏ các điểm ảnh có thể không có ý nghĩa trong việc phân loại

- ✓ Tính giá trị trung bình ở từng pixels trên từng cột, những pixels nào cho giá trị trung bình bé hơn hoặc bằng ngưỡng cho trước thì xóa đi. Ở đây, nhóm chọn ngưỡng là 1.
- ✓ Số lượng pixels đã loại bỏ là 283*X.shape[0] với X là đầu vào (ví dụ tập train của bộ dữ liệu MNIST có 60000 ảnh thì số lượng pixels đã loại bỏ là 283*60000)

```
] print('We will drop {} pixels'.format((X_train.mean(axis=0) <= 1).sum()))
  dropidx = np.argwhere(X_train.mean(axis=0) <= 1).reshape(-1)

  We will drop 283 pixels
```

- ✓ Dùng ý tưởng này để tạo 1 class Meanless_Pix_Dropper để đưa vào pipeline khi cần sử dụng:

```
class Meanless_Pix_Dropper(BaseEstimator, TransformerMixin):
    def __init__(self, threshold=1):
        self.threshold = threshold

    def fit(self, X, y=None):
        self.drop_idx = np.argwhere(X.mean(axis=0) <= self.threshold).reshape(-1)
        return self

    def transform(self, X, y=None):
        return np.delete(X, self.drop_idx, axis=1)

    def set_params(self, **params):
        for param, value in params.items():
            setattr(self,param,value)
        return self
```

2.3. Hiện tượng đa cộng tuyến

- ✓ Đa cộng tuyến là gì:

  Đa cộng tuyến là hiện tượng các biến độc lập trong mô hình hồi qui phụ thuộc tuyến tính lẫn nhau, thể hiện dưới dạng hàm số (vi phạm giả định 5 của mô hình hồi qui tuyến tính).

- ✓ Hậu quả của đa cộng tuyến:

Mặc dù sự hiện diện của đa cộng tuyến không ảnh hưởng đến tính nhất quán của các ước tính OLS của các hệ số hồi qui. Tuy nhiên, các ước tính sẽ trở nên không chính xác và không đáng tin cậy.

Hơn nữa, thực tế không thể phân biệt các tác động riêng lẻ của các biến độc lập lên biến phụ thuộc. Hậu quả là sai số chuẩn của tham số hồi qui tăng cao. Từ đó dẫn đến t-tests trên các hệ số có ít khả năng bác bỏ giả thuyết không.

Trong đó, OLS (Ordinary least squares) là phương pháp bình phương nhỏ nhất. Đây là một phương pháp ước tính dựa trên tiêu chí tối thiểu hóa tổng phần dư bình phương của hồi qui.

✓ Phát hiện đa cộng tuyến:

Dấu hiệu nổi bật của đa cộng tuyến là hệ số xác định R^2 cao mặc dù t-statistics về các hệ số của độ dốc ước tính không đáng kể, thể hiện sai số chuẩn tăng cao. Mặc dù các hệ số có thể được ước tính thiếu chính xác, nhưng các biến độc lập có vai trò giải thích biến phụ thuộc, điều này thể hiện thông qua việc R^2 cao. Thông thường người ta sử dụng hệ số VIF (Variance Inflaction Factor) để chứng minh cho hiện tượng đa cộng tuyến, đối với các data về kỹ thuật, khoa học,... VIF > 10 thì được xem là xảy ra đa cộng tuyến, đối với các data về xã hội thì VIF > 2 được xem là xảy ra đa cộng tuyến. Giả sử ta có $R^2 = 0.9$, thì hệ số Tolerance (dung sai) = $1 - R^2 = 0.1$, khi đó VIF = 1/ Tolerance = 10. Nhận thấy R^2 và VIF đồng biến, do vậy ở đồ án này nhóm chọn R^2 để đánh giá hiện tượng đa cộng tuyến của dữ liệu để dễ dàng tiếp cận.

✓ Xử lí hiện tượng đa cộng tuyến trong bài toán MNIST:

- Phát hiện hiện tượng đa cộng tuyến trong dữ liệu:
  Dựa vào phần lý thuyết ở trên ta tiến hành sử dụng ước tính OLS thông qua thư viện statsmodels để cho ra kết quả R^2 :

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.616
Model:                            OLS   Adj. R-squared:                  0.611
Method:                 Least Squares   F-statistic:                     111.2
Date:                Mon, 21 Jun 2021   Prob (F-statistic):               0.00
Time:                        07:48:35   Log-Likelihood:            -1.0004e+05
No. Observations:               50000   AIC:                         2.015e+05
Df Residuals:                   49287   BIC:                         2.078e+05
Df Model:                         712
Covariance Type:            nonrobust
==============================================================================
```

Ta nhận thấy $R^2 = 0.616$, do đó ta nghi vấn mô hình đã gặp phải vấn đề đa cộng tuyến (R^2 càng cao, và càng gần 1 thì mức độ phụ thuộc giữa các biến càng lớn, do đó mà hiện tượng đa cộng tuyến càng dễ xảy ra)

✓ Xử lý hiện tượng đa cộng tuyến bằng PCA:

- PCA là một thủ tục thống kê (statistical procedure) biến đổi một tập các biến tương quan thành một tập hợp các biến ít/không tương quan được gọi là thành phần chính. PCA loại bỏ đa cộng tuyến bằng cách chuyển đổi các feature ban đầu thành các feature có hiệu suất cao hơn với ít nhiễu hơn.
- Trong sklearn, việc điều chỉnh số chiều dữ liệu được giữ lại được thực hiện thông qua thông số n_component, nếu n_components nguyên và lớn hơn 1, pca sẽ hiểu đây là số chiều được giữ lại, nếu 0 < n_components < 1, pca sẽ lựa chọn số chiều sao cho số phương sai giải thích được >= n_components, hay tổng phương sai được giữ lại >= n_components, với phương sai càng lớn, tức là dữ liệu có độ phân tán cao, thể hiện lương thông tin càng lớn. Trong một hệ trục tọa độ, tổng phương sai của dữ liệu là như nhau và bằng tổng các giá trị riêng của ma trận hiệp phương sai $\sum_{i=1}^{D} \lambda_i$. Thêm nữa, PCA giúp giữ lại lượng thông tin là: $\sum_{i=1}^{K} \lambda_i$. Vậy ta có thể xem biểu thức $r_K = \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{D} \lambda_j}$ là lượng thông tin được giữ lại khi số chiều mới sau PCA là K. Việc lựa chọn K sao cho loại bỏ tốt đa cộng tuyến cũng như khi đưa vào train mô hình cho ra kết quả tốt thì phải trải qua quá trình lựa chọn, thử nghiệm.
- Thử nghiệm với pca(n_components = .98) ta thu được kết quả như sau:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.607
Model:	OLS	Adj. R-squared:	0.605
Method:	Least Squares	F-statistic:	294.7
Date:	Mon, 21 Jun 2021	Prob (F-statistic):	0.00
Time:	07:48:47	Log-Likelihood:	-1.0063e+05
No. Observations:	50000	AIC:	2.018e+05
Df Residuals:	49738	BIC:	2.041e+05
Df Model:	261		
Covariance Type:	nonrobust		

- Thử nghiệm với pca(n_components = 40, random_state = 1), ta thu được kết quả như sau:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.535
Model:	OLS	Adj. R-squared:	0.535
Method:	Least Squares	F-statistic:	1439.
Date:	Mon, 21 Jun 2021	Prob (F-statistic):	0.00
Time:	07:48:53	Log-Likelihood:	-1.0483e+05
No. Observations:	50000	AIC:	2.098e+05
Df Residuals:	49959	BIC:	2.101e+05
Df Model:	40		
Covariance Type:	nonrobust		

→ Rõ ràng ta thấy khi đi qua mô hình PCA, hiện tượng đa cộng tuyến giữa các điểm dữ liệu đã được giảm đi. Đối với PCA có n_components = 0.98 thì tức là nó sẽ giữ lại 98% phương sai dữ liệu nên tùy vào dữ liệu mà số chiều giảm sẽ khác nhau ví dụ ở đây đối với tập train (sau khi đã tách tập dữ liệu train thành 2 tập train và validation) thì số chiều giảm xuống còn 261, khi đó số pixels bị mất đi là (784 – 261)*50000. Đối với PCA có n_components = 40 thì tức là nó sẽ làm giảm chiều dữ liệu xuống còn 40, khi đó số pixels tương đương bị mất đi là (784 – 40) * X.shape[0] với X là đầu vào.

2.4. Gia tăng hiệu quả của mô hình

Để gia tăng hiệu quả của mô hình, nhóm sử dụng thêm một số phương pháp: Deskewing, Standard Scaler (chuẩn hóa dữ liệu), Polynomial Features, Multi zoning.

- ✓ Deskewing:

  Khi viết, đôi khi chúng ta sẽ viết nghiêng so với mặt giấy điều này sẽ làm cho các chữ cái và con số bị lệch. Và không giống như mắt người, máy tính sẽ gặp khó khăn khi nhận dạng các chữ cái và con số này. Vì vậy, người ta đưa ra một quá trình gọi là **Deskewing** để giải quyết vấn đề này.

  Về mặt hình thức, **Deskewing** là quá trình làm thẳng hình ảnh đã được quét hoặc viết một cách lệch lạc – hình ảnh bị nghiêng quá xa về một hướng hoặc bị lệch.

  Người ta mô hình hóa quá trình **Deskewing** như một phép biến đổi affine (là một phép biến đổi gồm một biến đổi tuyến tính và một vector dịch – phép tịnh tiến).

  Quá trình gồm các bước sau:

  - Tìm khối tâm của ảnh để tính xem cần bù ảnh bằng bao nhiêu.
  - Tìm ma trận hiệp phương sai của cường độ pixel hình ảnh (có thể sử dụng ma trận này để ước tính độ lệch của góc).
  - Tính toán ma trận để làm thẳng hình ảnh bằng công thức sau:

$$\begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

với $\alpha = \frac{Cov(X,Y)}{Var(X)}$

- ✓ Standard Scaler: Chuẩn hóa dữ liệu bằng cách trừ đi mean và chia cho độ lệch chuẩn giúp thuật toán hội tụ nhanh hơn, kết quả tốt hơn.
- ✓ Polynomial Features: Tạo đa thức và các tính năng tương tác. Sẽ tạo ra ma trận đối tượng mới bao gồm tất cả các tổ hợp đa thức của các đối tượng có bậc nhỏ hơn hoặc bằng bậc đã chỉ định.
- ✓ Multi zoning: Cắt hình ảnh thành lưới tự định nghĩa, ở đây nhóm chọn lưới 8x8, hình ảnh sẽ được chia thành 64 ô vuông nhỏ. Dữ liệu ban đầu sẽ được reshape lại thành (50000, 24, 24). Số liệu ở lưới 8x8 tự định nghĩa sẽ được cập nhật bằng cách lấy trung bình theo các pixels ở các cột và các dòng 0, 3, 6, 9, 12, 15, 18, 21, 24 của dữ liệu ban đầu. Giúp giảm chiều dữ liệu, từ đó mô hình sẽ xử lý nhanh hơn.

→ Sau khi sử dụng kết hợp các phương pháp trên thì kết quả trên tập validation đã được cải thiện được tầm 4 – 8%.

3. Logistic Regression

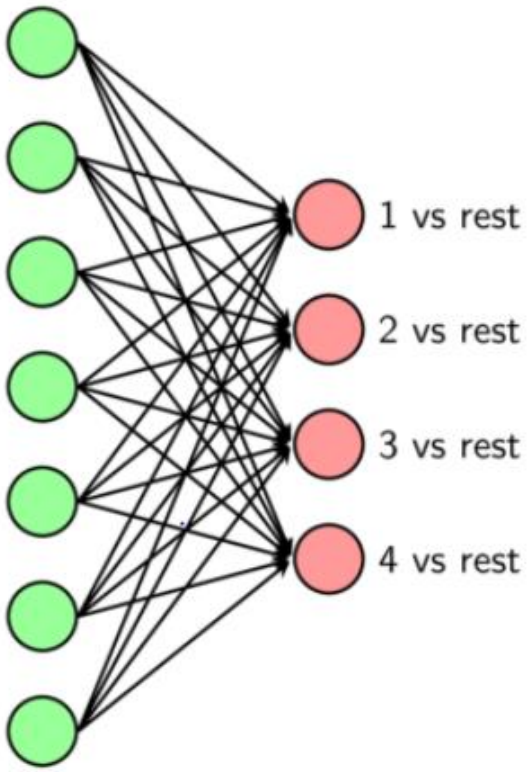
3.1. Dữ liệu đầu vào và đầu ra (khi chưa sử dụng các phương pháp khác để gia tăng hiệu quả mà chỉ sử dụng Meanless_Pix_Dropper và PCA)

- ✓ Dữ liệu đầu vào:
  - Nếu sử dụng PCA với n_components = 0.98:
    Mảng các mảng một chiều có kích thước (252,) gồm 252 số nguyên có giá trị từ 0 đến 255 (các điểm ảnh của bức ảnh).
  - Nếu sử dụng PCA với n_components = 40:
    Mảng các mảng một chiều có kích thước (40,) gồm 40 số nguyên có giá trị từ 0 đến 255 (các điểm ảnh của bức ảnh).
- ✓ Dữ liệu đầu ra: Mảng chứa nhãn của bức ảnh (được xác định bằng cách lấy nhãn có phần trăm xảy ra là lớn nhất trong 10 nhãn từ 0 – 9).

3.2. Cấu trúc và cách thiết kế mô hình Logistic Regression

3.2.1. Cấu trúc và cách thiết kế

Để thực hiện việc phân lớp cho bộ MNIST, nhóm thực hiện theo phương pháp one-vs-rest. Cụ thể, nếu có 10 classes thì nhóm sẽ xây dựng 10 classifiers, mỗi classifier tương ứng với một class. Classifier thứ nhất giúp phân biệt **class 1** vs **not class 1**, tức xem một điểm có thuộc class 1 hay không, hoặc xác suất để một điểm rơi vào class 1 là bao nhiêu. Tương tự như thế, classifier thứ hai sẽ phân biệt **class 2** vs **not class 2**, … Kết quả cuối cùng có thể được xác định bằng cách xác định class mà một điểm rơi vào với xác suất cao nhất. Phương pháp này còn được gọi là one hot coding vì với cách mã hóa trên, giả sử có 4 classes, class 1, 2, 3, 4 sẽ lần lượt được mã hóa dưới dạng nhị phân bởi **1000, 0100, 0010** hoặc **0001**. One-hot vì chỉ có *one* bit là *hot* (bằng **1**). Hàm Logistic Regression trong thư viện sklearn có thể được dùng trực tiếp để áp dụng vào các bài toán multi-class classification với phương pháp one-vs-rest. Chúng ta có thể hình dung phương pháp one-vs-rest như sau:



one-vs-rest

Đầu tiên khởi tạo một list gồm 10 bộ phân lớp Logistic Regression (max_iter=200)

```
ovr_classifiers = []
for col in y_train_preprocessed:
    ovr_classifiers.append(LogisticRegression(max_iter = 200))
```

Xây dựng hàm prediction theo nguyên tắc đã nêu trên đây, xác định class bằng xác suất mà điểm cần dự đoán rơi vào cao nhất thông qua phương thức predict_proba sẵn có của sklearn.logisticregression. Về cơ chế xác suất này được xác định thông qua một hàm sigmoid

$f(s) = \frac{1}{1+e^{-s}}$

```
def predictions(models,test_X):

    preds_y = []

    for model in models:
        ## chọn xác suất thuộc về  lớp
        y = model.predict_proba(test_X)[:,1]

        if len(preds_y) == 0:
            preds_y = y
        else:
            preds_y = np.vstack((preds_y,y))

    preds_y = preds_y.T

    return np.argmax(preds_y, axis=1)
```

Xây dựng hàm train cho 10 bộ phân lớp, với đầu vào là list 10 bộ phân lớp Logistic Regression, một pipeline tiền xử lí, X_train và y_train

```
def train(models, preprocess_pipeline, X_train, y_train):
    '''
    Train logistic regressions to classify multiclass
    models: A list contain 10 Logistic Regression models
    X_train, X_test: Data with shape of (n_images, width, height)
    y_train: One Hot Encoded train label
    y_test: Labels with shape of (n_images,)
    '''
    X_train_preprocessed = preprocess_pipeline.fit_transform(X_train)

    for model, label in zip(models, y_train):
        model.fit(X_train_preprocessed, y_train[label])
```

Cuối cùng thực hiện transform lên tập test và dự đoán và in ra ma trận hỗn loạn:

```
def train_n_summary(models, preprocess_pipeline, X_train, y_train, X_test, y_test):
    '''
    Train logistic regressions to classify multiclass
    models: A list contain 10 Logistic Regression models
    X_train, X_test: Data with shape of (n_images, width, height)
    y_test: Labels with shape of (n_images,)
    '''
    train(models, preprocess_pipeline, X_train, y_train)

    X_test_preprocessed = preprocess_pipeline.transform(X_test)
    y_pred = predictions(models, X_test_preprocessed)

    score = metrics.accuracy_score(y_test, y_pred)
    confusion_matrix_(y_test, y_pred, score)
```

Như vậy có thể hình dung cấu trúc của mô hình Logistic Regression được xây dựng như sau: Xây dựng 10 bộ phân lớp ứng với 10 chữ số -> Train cho 10 bộ phân lớp này bằng cách fit vào tập train -> Transfrom kết quả vào tập test, tính toán xác suất và dự đoán lớp dựa vào xác suất lớn nhất.

3.2.2. Các thí nghiệm

a. Thí nghiệm 1:

- ✓ Chia dữ liệu tập train thành 2 tập: train và validation.
- ✓ Sử dụng mô hình Logistic Regression với max_iter = 200.
- ✓ Các bước thực hiện:
  - Bước 1: Tạo pipeline gồm meanless_pixels_dropper và pca1 (pca có n_components = 0.98) .
  - Bước 2: Huấn luyện và đem mô hình đi dự đoán kết quả trên tập validation.
- ✓ Dữ liệu đầu vào lúc này là mảng các mảng một chiều có kích thước (252,) gồm 252 số nguyên có giá trị từ 0 đến 255 (các điểm ảnh của bức ảnh).
- ✓ Dữ liệu đầu ra sẽ là mảng chứa nhãn của bức ảnh (được xác định bằng cách lấy nhãn có phần trăm xảy ra là lớn nhất trong 10 nhãn từ 0 – 9).

b. Thí nghiệm 2:

- ✓ Chia dữ liệu tập train thành 2 tập: train và validation.
- ✓ Sử dụng mô hình Logistic Regression với max_iter = 200.
- ✓ Các bước thực hiện:
  - Bước 1: Tạo pipeline gồm deskewing, meanless_pixels_dropper, pca1 (pca có n_components = 0.98) và standard scaler .
  - Bước 2: Huấn luyện và đem mô hình đi dự đoán kết quả trên tập validation.
- ✓ Dữ liệu đầu vào lúc này là mảng các mảng một chiều có kích thước (190,) gồm 190 số thực (vì sử dụng standard scaler nên các giá trị ban đầu từ 0 đến 255 đã được chuẩn hóa).
- ✓ Dữ liệu đầu ra sẽ là mảng chứa nhãn của bức ảnh (được xác định bằng cách lấy nhãn có phần trăm xảy ra là lớn nhất trong 10 nhãn từ 0 – 9).

c. Thí nghiệm 3:

- ✓ Chia dữ liệu tập train thành 2 tập: train và validation.
- ✓ Sử dụng mô hình Logistic Regression với max_iter = 200.
- ✓ Các bước thực hiện:
  - Bước 1: Tạo pipeline gồm deskewing, multi zoning, polynomial features và standard scaler.
  - Bước 2: Huấn luyện và đem mô hình đi dự đoán kết quả trên tập validation.
- ✓ Dữ liệu đầu vào lúc này là mảng các mảng một chiều có kích thước (2081,) gồm 2081 số thực (vì sử dụng standard scaler nên các giá trị đã được chuẩn hóa). Và khi sử dụng multi zoning thì dữ liệu đã được giảm số chiều nhưng sau đó lại sử dụng polynomial features nên số chiều dữ liệu lại tăng lên.
- ✓ Dữ liệu đầu ra sẽ là mảng chứa nhãn của bức ảnh (được xác định bằng cách lấy nhãn có phần trăm xảy ra là lớn nhất trong 10 nhãn từ 0 – 9).

d. Thí nghiệm 4:

- ✓ Chia dữ liệu tập train thành 2 tập: train và validation.
- ✓ Sử dụng mô hình Logistic Regression với max_iter = 200.
- ✓ Các bước thực hiện:
  - Bước 1: Tạo pipeline gồm deskewing, pca2 (pca có n_components = 40), polynomial features và standard scaler.
  - Bước 2: Huấn luyện và đem mô hình đi dự đoán kết quả trên tập validation.
- ✓ Dữ liệu đầu vào lúc này là mảng các mảng một chiều có kích thước (821,) gồm 821 số thực (vì sử dụng standard scaler nên các giá trị đã được chuẩn hóa). Và khi sử dụng pca2 thì dữ liệu đã được giảm số chiều xuống còn 40 nhưng sau đó lại sử dụng polynomial features nên số chiều dữ liệu lại tăng lên.
- ✓ Dữ liệu đầu ra sẽ là mảng chứa nhãn của bức ảnh (được xác định bằng cách lấy nhãn có phần trăm xảy ra là lớn nhất trong 10 nhãn từ 0 – 9).

e. Thí nghiệm 5:

- ✓ Chia dữ liệu tập train thành 2 tập: train và validation.
- ✓ Sử dụng mô hình Logistic Regression mặc định nhưng đổi bộ phân lớp thành One vs One:

```
ovo_classifiers = OneVsOneClassifier(LogisticRegression())
```

One-vs-one sẽ xây dựng các bộ binary classifiers cho từng cặp classes. Bộ thứ nhất phân biệt class 1 và class 2, bộ thứ hai phân biệt class 1 và class 3,...Khi có một dữ liệu mới vào, sẽ đưa nó vào toàn bộ các bộ binary classifiers đã xây dựng. Kết quả cuối cùng có thể được xác định bằng cách xem class nào mà điểm dữ liệu đó được phân vào nhiều nhất. Ở đây, đối với Logistic Regression thì có thể tính tổng các xác suất tìm được sau mỗi bộ binary classifiers.

- ✓ Các bước thực hiện:
  - Bước 1: Tạo pipeline gồm deskewing, pca2 (pca có n_components = 40), polynomial features và standard scaler.
  - Bước 2: Huấn luyện và đem mô hình đi dự đoán kết quả trên tập validation.
- ✓ Dữ liệu đầu vào lúc này là mảng các mảng một chiều có kích thước (821,) gồm 821 số thực (vì sử dụng standard scaler nên các giá trị đã được chuẩn hóa). Và khi sử dụng pca2 thì dữ liệu đã được giảm số chiều xuống còn 40 nhưng sau đó lại sử dụng polynomial features nên số chiều dữ liệu lại tăng lên.
- ✓ Dữ liệu đầu ra sẽ là mảng chứa nhãn của bức ảnh (được xác định bằng cách trong thời gian dự đoán lớp nào nhận được nhiều vote nhất sẽ được chọn).

3.3. Các tham số đặc biệt

- ✓ Ở mô hình Logistic Regression thì nhóm sử dụng các tham số mặc định là chủ yếu, chỉ đặt chỉ số max_iter. Và dưới đây là thông tin về một số tham số quan trọng trong mô hình Logistic Regression:
  - max_iter: số lần lặp tối đa để các bộ giải hội tụ, ở đồ án này nhóm đặt bằng 200 để giúp bộ giải hội tụ.

- solver: thuật toán (bộ giải) được áp dụng, ở đồ án này nhóm đặt mặc định tức là solver = 'lbfgs' – một phương pháp tối ưu dựa trên gradient, vì tính đa nhiệm của nó.
- penalty: chỉ định mức được sử dụng trong hình phạt, ở đồ án này nhóm đặt mặc định tức là penalty = 'l2' – bổ sung thêm một hình phạt L2 bằng bình phương độ lớn của các hệ số, L2 sẽ không mang lại các mô hình thưa thớt và tất cả các hệ số bị thu hẹp theo cùng một hệ số (không hệ số nào bị loại bỏ), để giúp tránh Overfitting.
- C: tham số cân bằng của hồi quy logistic xác định độ mạnh của chính quy được gọi là C và các giá trị cao hơn của C tương ứng với độ chính quy ít hơn, ở đồ án này nhóm đặt mặc định tức là C = 1.0.

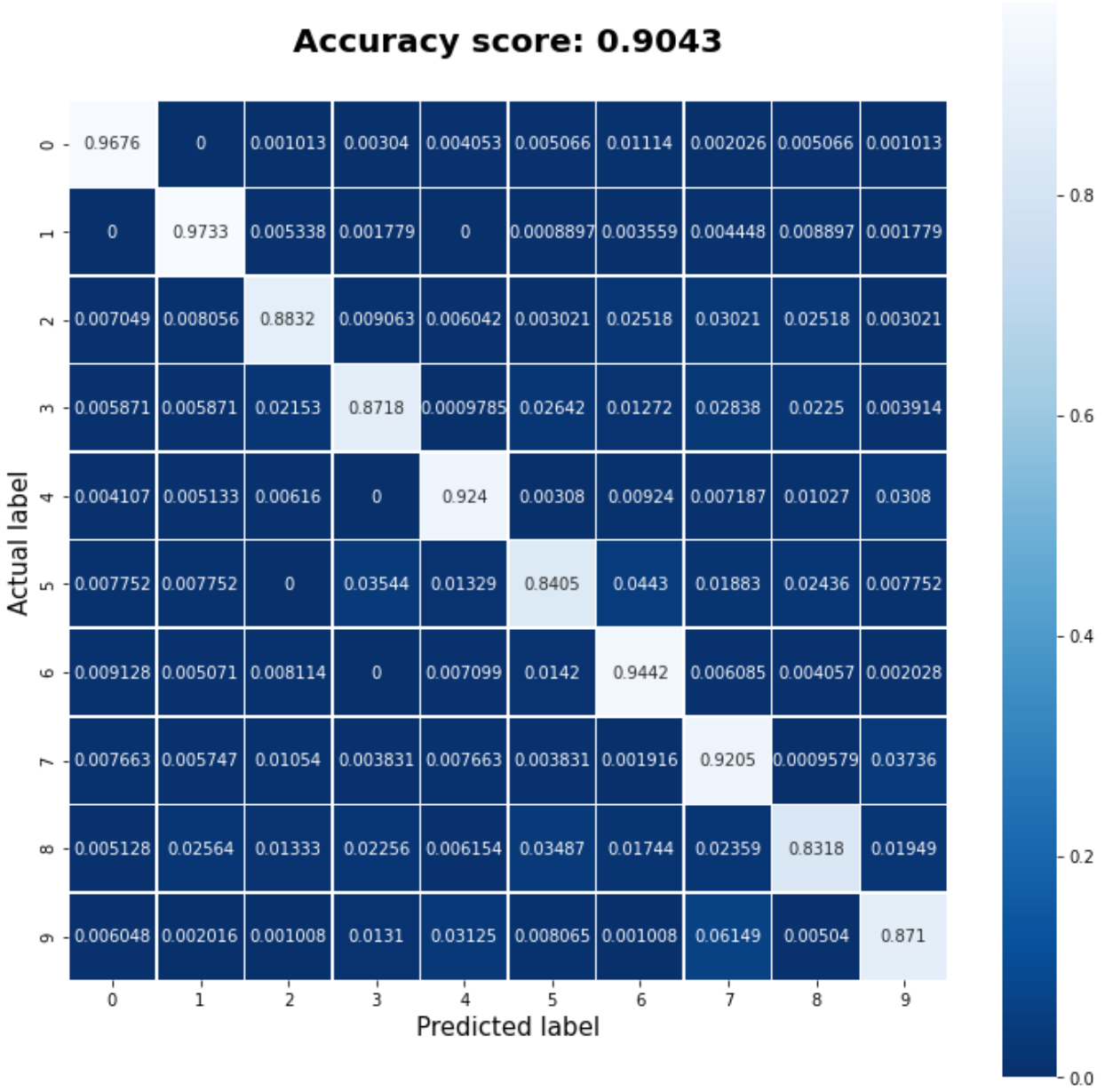
✓ Polynomial Features:
Nếu sử dụng phương pháp này theo mặc định thì số lượng tính năng đầu vào sẽ khá nhiều. Vì vậy nhóm đặt tham số **interaction_only = True** để giảm số lượng tính năng đầu vào, giảm bớt tình trạng overfitting.

✓ PCA:
- Sử dụng PCA với n_components = 0.98 để giảm chiều dữ liệu nhưng vẫn giữ lại được 98% phương sai của dữ liệu. Giúp giữ lại được các tính năng quan trọng nhưng giảm được chiều dữ liệu để việc chạy mô hình nhanh hơn, hiệu quả hơn.
- Sử dụng PCA với n_components = 40 để giảm chiều dữ liệu xuống còn 40. PCA này sẽ được sử dụng trong các pipeline có sử dụng polynomial features để việc chạy mô hình nhanh hơn vì polynomial features sẽ làm tăng đáng kể số lượng tính năng đầu vào.

3.4. Kết quả đạt được

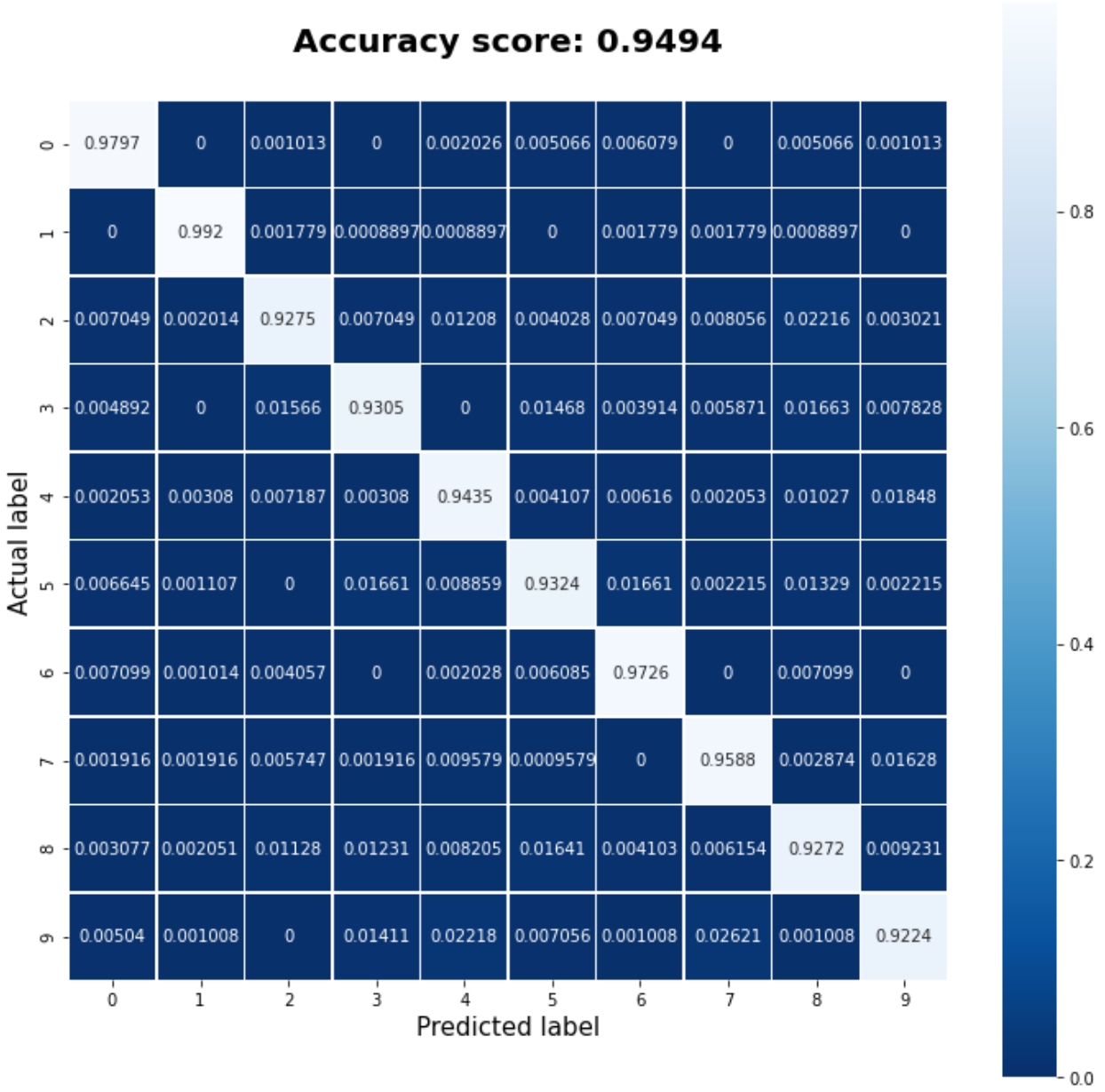
Trên tập Validation:

✓ Thí nghiệm 1:
- Độ chính xác của mô hình đối với toàn bộ tập là: 0.9043.
- Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9676, 0.9733, 0.8832, 0.8718, 0.924, 0.8405, 0.9442, 0.9205, 0.8318, 0.871.
- Confusion Matrix:



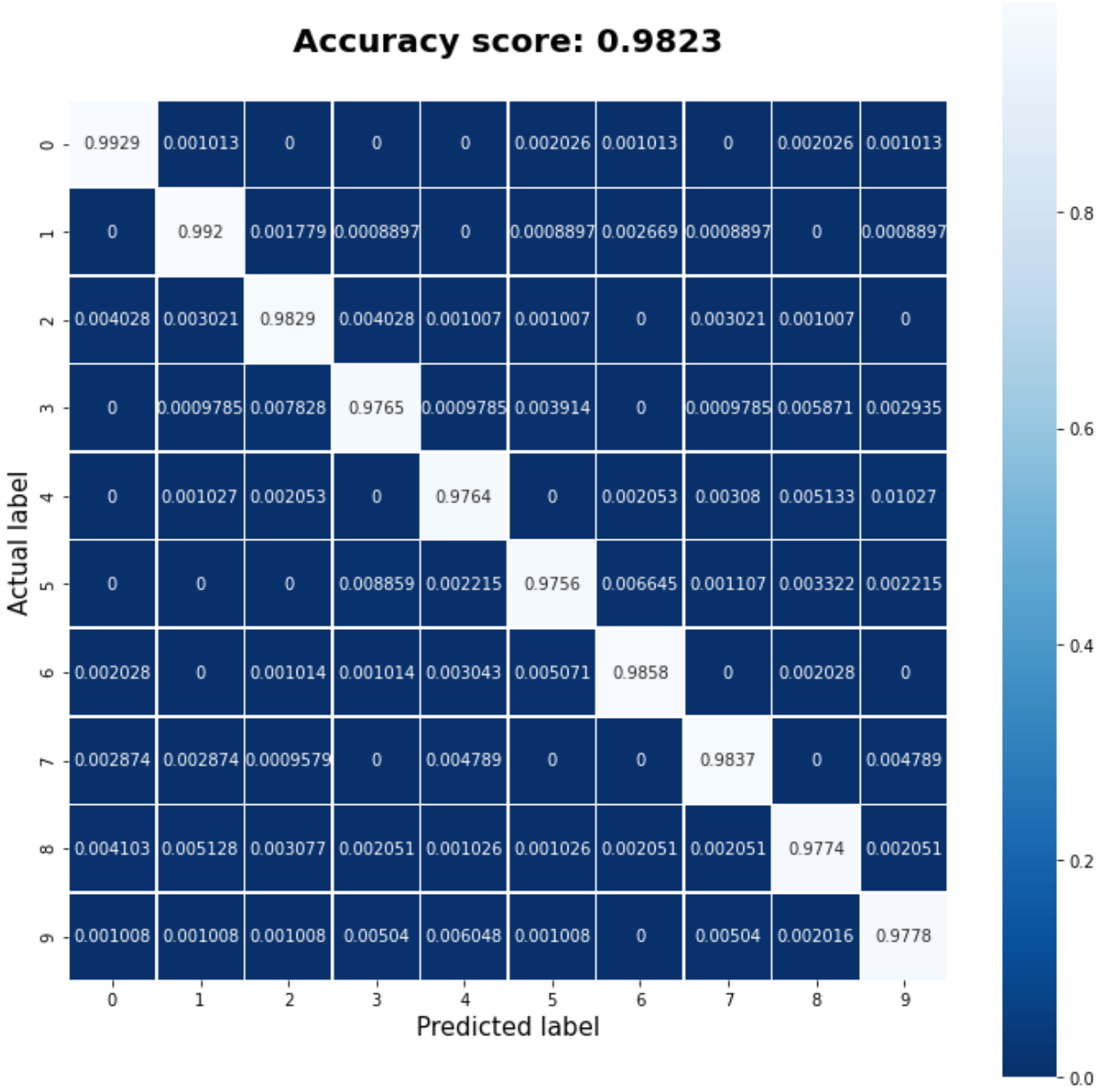
✓ Thí nghiệm 2:

- Độ chính xác của mô hình đối với toàn bộ tập là: 0.9494.
- Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9797, 0.992, 0.9275, 0.9305, 0.9435, 0.9324, 0.9726, 0.9588, 0.9272, 0.9224.
- Confusion Matrix:



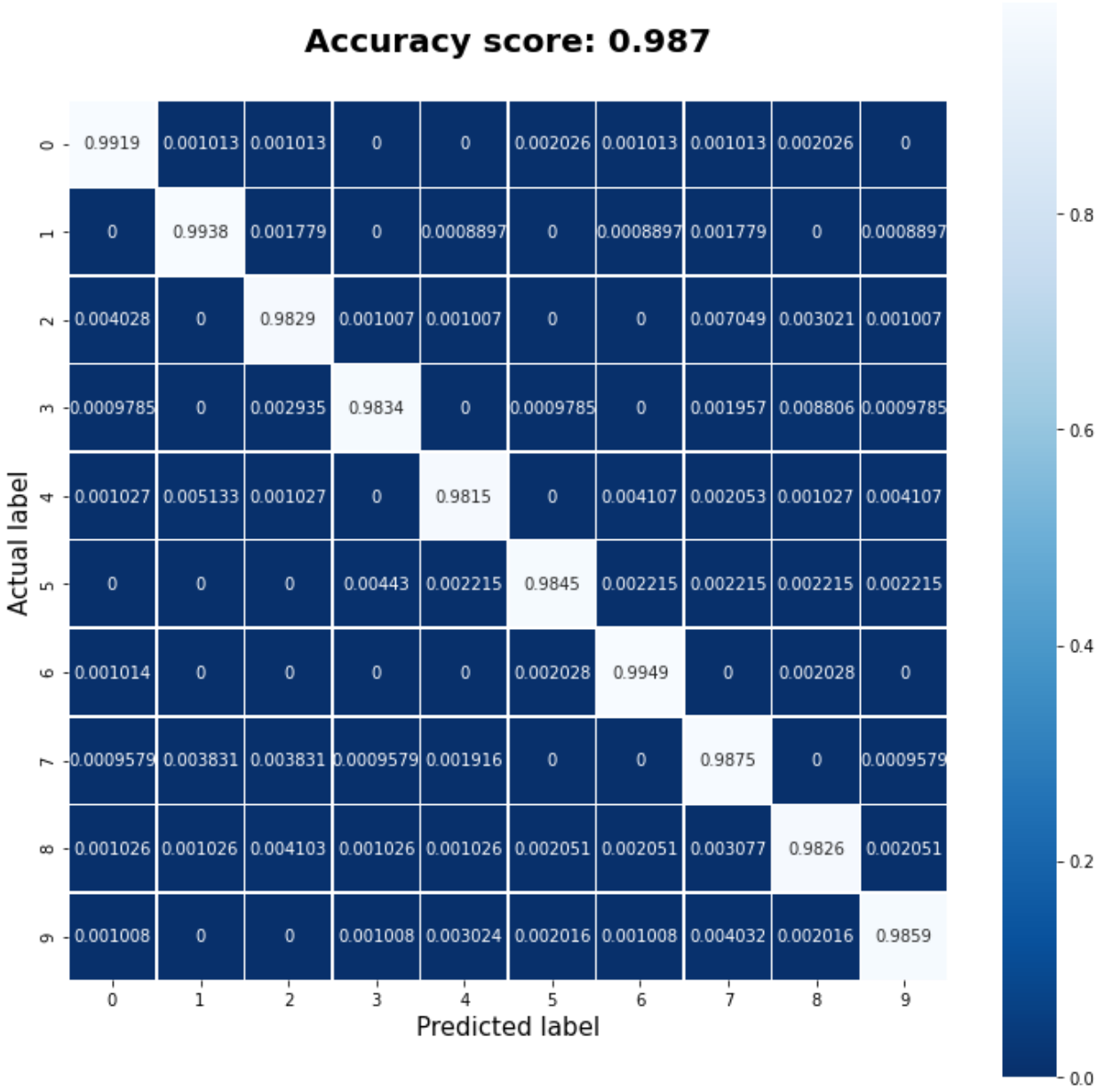
✓ Thí nghiệm 3:

- Độ chính xác của mô hình đối với toàn bộ tập là: 0.9823.
- Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9929, 0.992, 0.9829, 0.9765, 0.9764, 0.9756, 0.9858, 0.9837, 0.9774, 0.9778.
- Confusion Matrix:



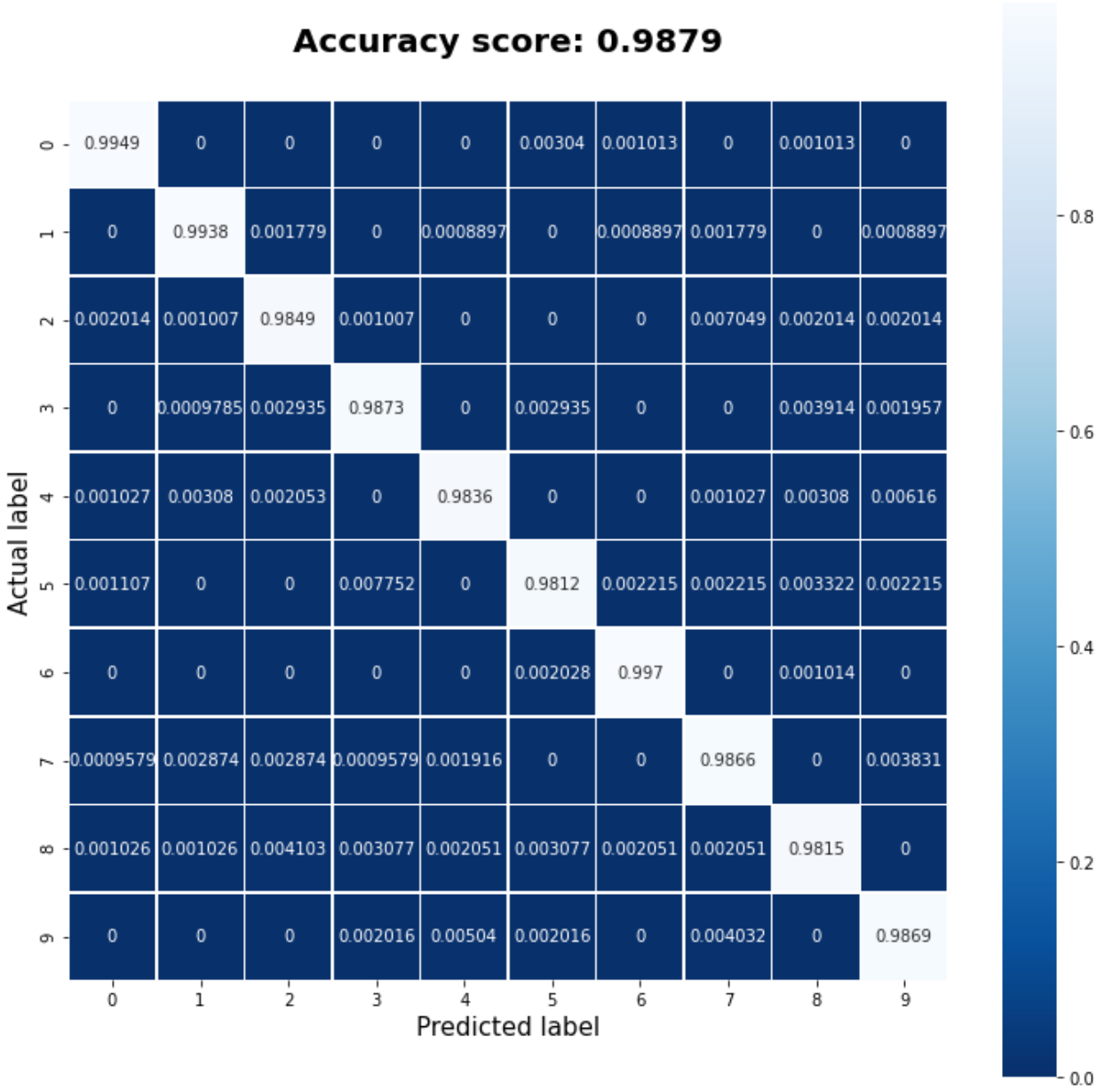
✓ Thí nghiệm 4:

- Độ chính xác của mô hình đối với toàn bộ tập là: 0.987.
- Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9919, 0.9938, 0.9829, 0.9834, 0.9815, 0.9845, 0.9949, 0.9875, 0.9826, 0.9859.
- Confusion Matrix:



✓ Thí nghiệm 5:

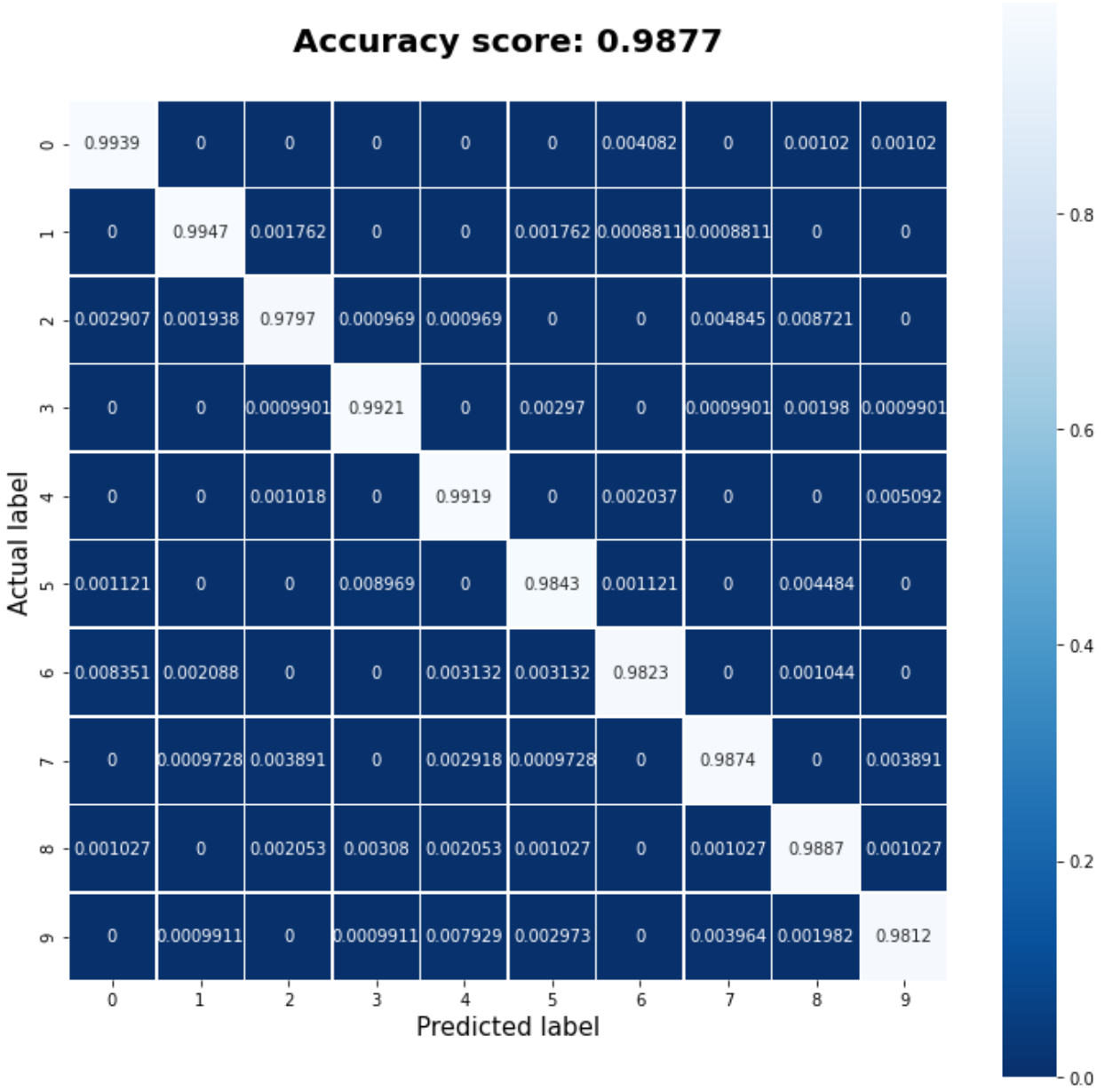
- Độ chính xác của mô hình đối với toàn bộ tập là: 0.9879.
- Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9949, 0.9938, 0.9849, 0.9873, 0.9836, 0.9812, 0.997, 0.9866, 0.9815, 0.9869.
- Confusion Matrix:



Trên tập Test:

Từ kết quả đạt được trên tập validation ta thấy thí nghiệm 5 cho kết quả cao nhất, vì vậy lấy model ở thí nghiệm 5 huấn luyện trên toàn bộ tập dữ liệu train ban đầu (train + validation) và dự đoán kết quả trên tập test.

- ✓ Độ chính xác của mô hình đối với toàn bộ tập là: 0.9877.
- ✓ Độ chính xác của mô hình đối với từng lớp (từ 0 – 9) lần lượt là: 0.9939, 0.9947, 0.9797, 0.9921, 0.9919, 0.9843, 0.9823, 0.9874, 0.9887, 0.9812.
- ✓ Confusion Matrix:



4. Thử nghiệm thực tế

- ✓ Sử dụng model cho kết quả cao nhất huấn luyện cho toàn bộ tập dữ liệu train + test để dự đoán ảnh tự vẽ.

```
selected_pipeline = deskew_pca_poly_scale_pipeline
ovo_train(ovo_classifiers, selected_pipeline, pixel_cols.reshape(-1,28,28), label_col)
```

- ✓ Hàm quan trọng:
  - Hàm **event_function**: hàm cho phép thực hiện vẽ, create_oval cho phép vẽ số bằng chuột, chữ số được hiện trong khung vẽ, img_draw.ellipse cho phép vẽ số bằng chuột, chữ số được lưu vào ảnh.

```
def event_function(event):
    x = event.x
    y = event.y

    x1 = x - 16
    y1 = y - 16

    x2 = x + 16
    y2 = y + 16

    canvas.create_oval((x1,y1,x2,y2), fill = 'black')
    img_draw.ellipse((x1,y1,x2,y2), fill = 'white')
```

- Hàm **clear** dùng để xóa những gì được vẽ bằng chuột, để tiếp tục thực hiện dự đoán.

```
def clear():
    global img, img_draw, label_status

    img = Image.new('RGB',(500, 500),(0,0,0))
    img_draw = ImageDraw.Draw(img)
    canvas.delete('all')
    label_status.config(text = 'Waiting...')
```

- Hàm **predict**: dự đoán số vẽ tay, chuyển ảnh về mảng các pixel, hàm cv2.cvtColor chuyển ảnh từ BGR sang gray, đưa ảnh về kích thước 28x28, sử dụng model để dự đoán số.

```
def predict():
    global label_status
    img_array = np.array(img)
    img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2GRAY)

    img_array = cv2.resize(img_array, (28,28))
    img_array = img_array.reshape(784,)

    result = ovo_predictions(ovo_classifiers, selected_pipeline.transform(img_array.reshape(1,-1)))
    label_status.config(text = ('PREDICTION: ' + str(result[0])))
```

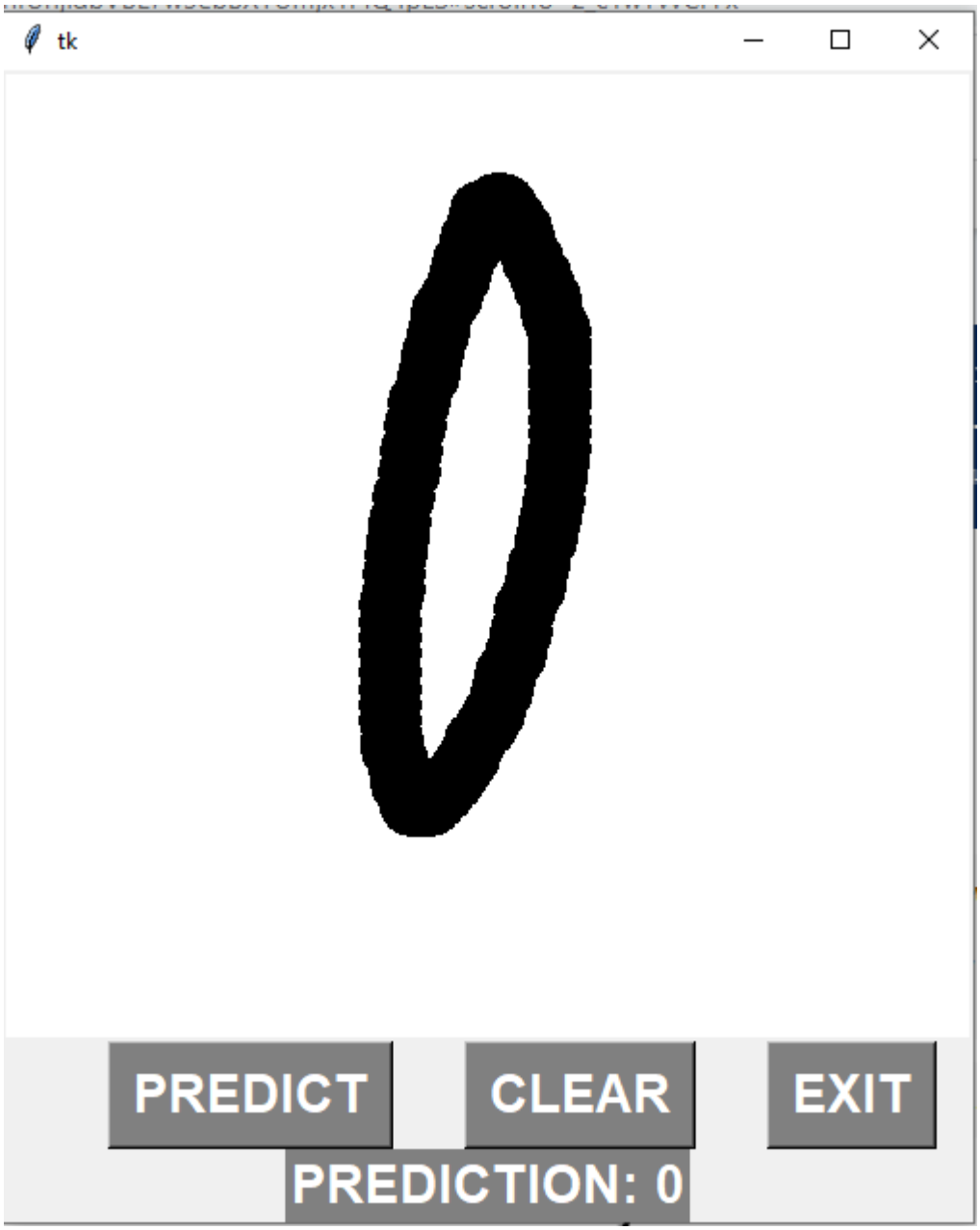
✓ Khung vẽ:

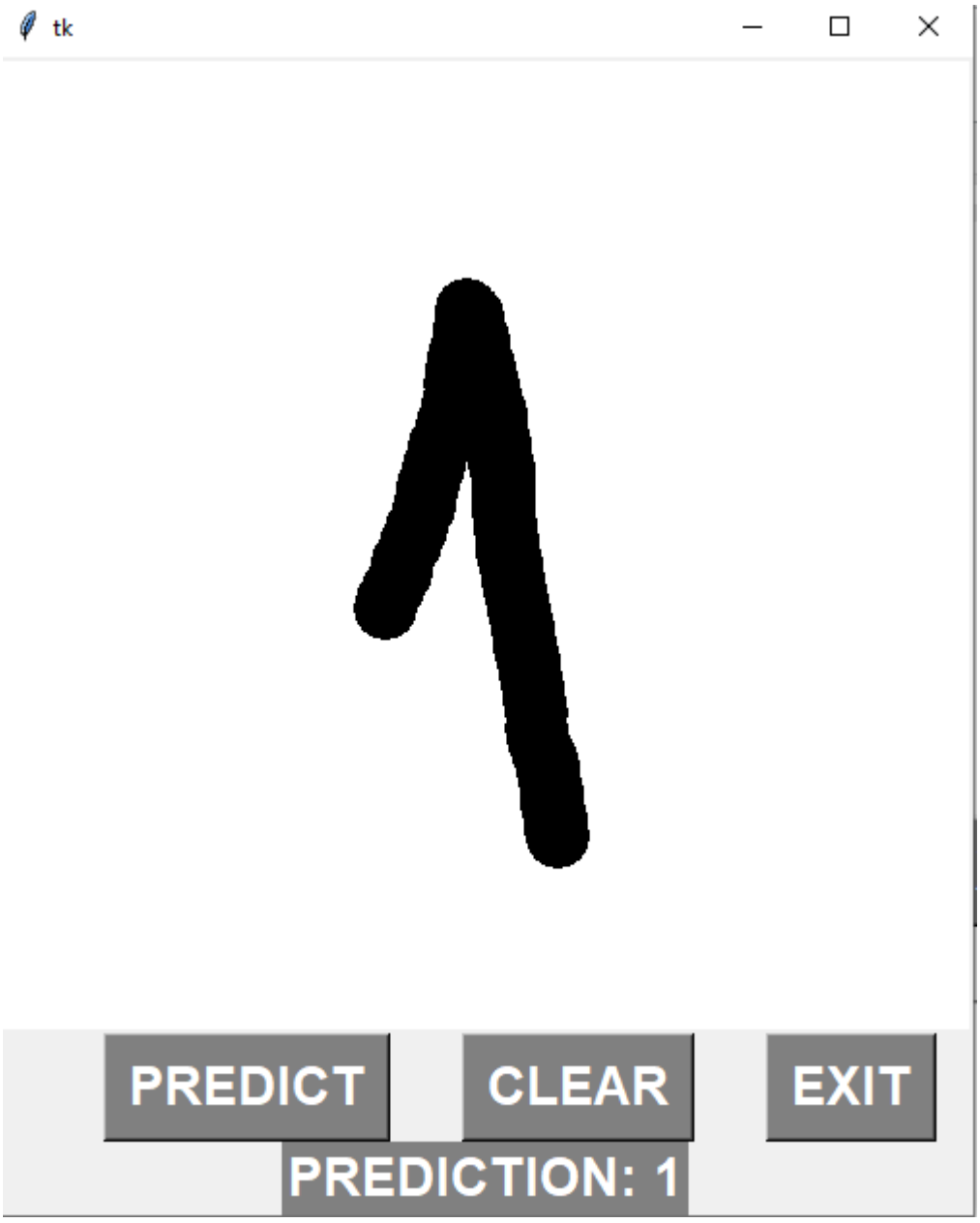
✓ Có 3 lựa chọn:

- Predict: dự đoán số vừa vẽ.
- Clear: xóa số vừa vẽ để tiếp tục vẽ.
- Exit: thoát khỏi ô vẽ.

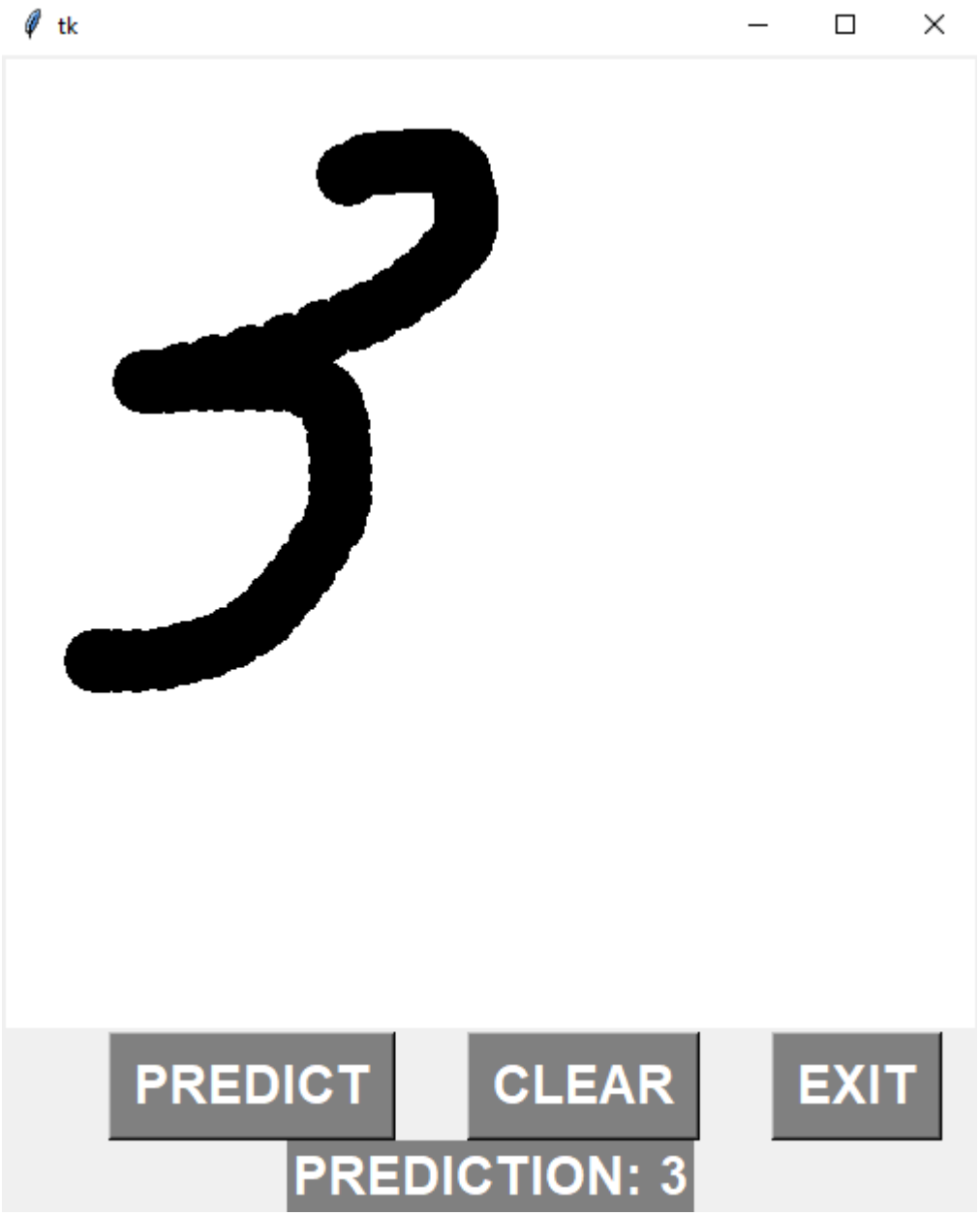
Và Prediction: sẽ hiện số mà model dự đoán được.

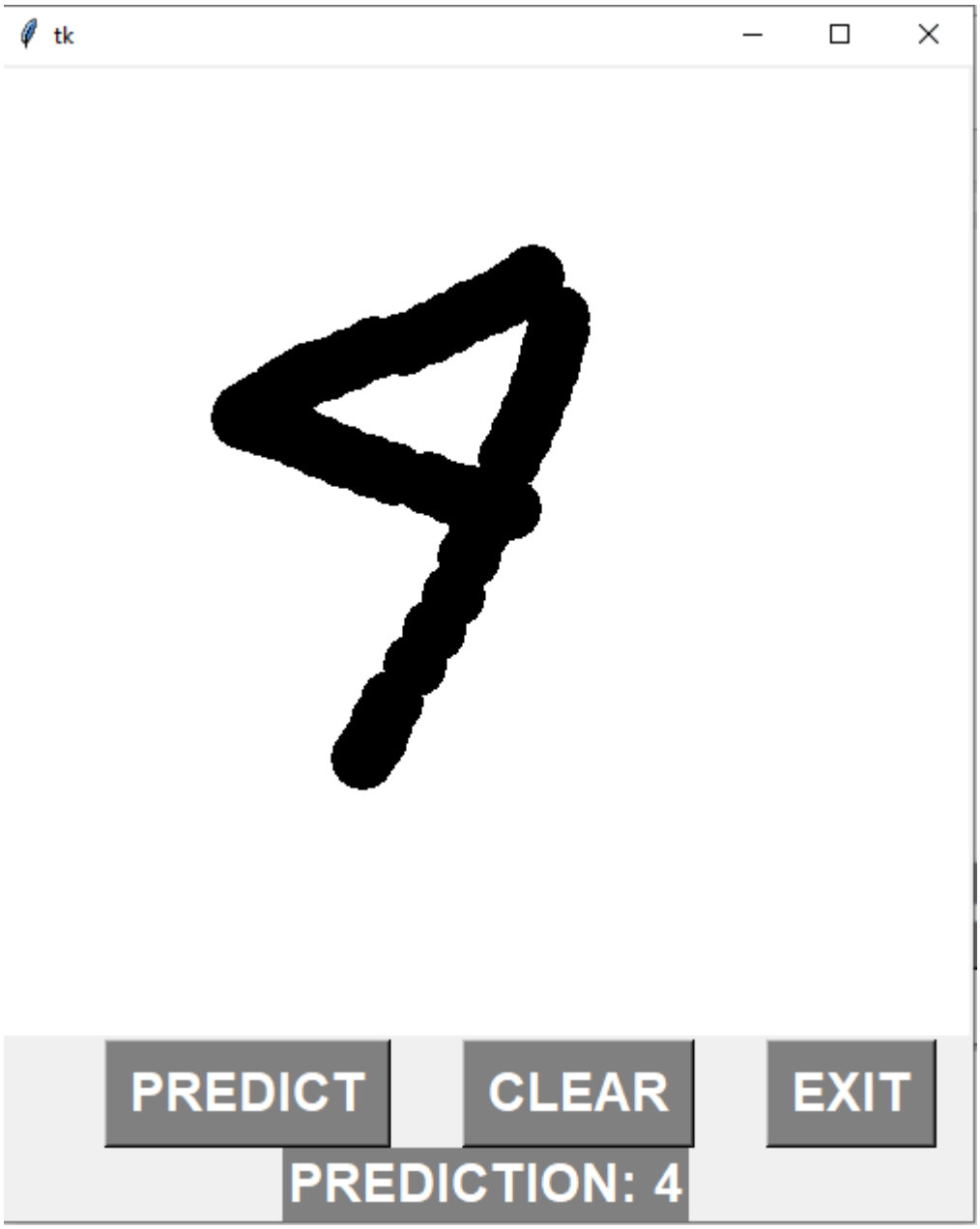
✓ Thử dự đoán một số ảnh:


tk
PREDICT
CLEAR
EXIT
PREDICTION: 0


tk
PREDICT
CLEAR
EXIT
PREDICTION: 1


tk
PREDICT
CLEAR
EXIT
PREDICTION: 2


tk
PREDICT
CLEAR
EXIT
PREDICTION: 3



5. Hướng dẫn chạy code

- ✓ Vì phần tự vẽ ảnh có sử dụng giao diện TK nên code chỉ có thể chạy trên local.
- ✓ Vì chạy trên local nên có một số thư viện cần cài đặt trước khi chạy:
  - Tensorflow.
  - Keras.
  - CV2.

III. TÀI LIỆU THAM KHẢO

https://machinelearningcoban.com/2017/06/21/pca2/

https://vietnambiz.vn/da-cong-tuyen-multicollinearity-trong-mo-hinh-hoi-qui-la-gi-hau-qua-20200107181856717.htm
https://stats.stackexchange.com/questions/365964/can-pca-be-used-for-detecting-multicollinearity
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
https://www.youtube.com/watch?v=nGWN6nm8QRk
https://machinelearningcoban.com/2017/02/11/binaryclassifiers/#one-vs-rest-hay-one-hot-coding
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://fsix.github.io/mnist/Deskewing.html?fbclid=IwAR1eY35ZZ4Wyqprn3UX5_J7K3RhPX82oixEKKGgMTMxXBwbZGGcP1Vac3qs