Lecture 4

Support vector machine (SVM)

Support vector machine (SVM)

One of the most popular learning methods used to solve classification and regression problems.

The main idea of the method is **to construct a hyperplane that separates the sample objects in an optimal way**.

The algorithm operates under the assumption that the greater the distance (gap) between the separating hyperplane and the objects of the separable classes, the smaller the average classifier error will be.

Linear classification

Consider a binary classification problem in which $X = R^d$ $-$ object space;

$Y = \{-1, +1\}$ $-$ set of valid answers (target feature);

$X = \{(x_i, y_i)\}_{i=1}^{l}$ $-$ training set;

Let's call class $+1$ positive, and class -1 negative.

Here d $-$ dimension of feature space, l $-$ number of examples in the training set.

Linear classification

1. Problem Statement

Suppose we have a dataset $X = \{(x_i, y_i)\}_{i=1}^{l}$.

The goal of SVM is to find a hyperplane that separates the data into two classes.

The hyperplane is defined by the equation $(\langle w, x\rangle + b)$, where $b \in R -$ bias, it determines how far the hyperplane is from the origin. If $b = 0$, the hyperplane passes through the point $(0,0)$, if $b \neq 0$, the hyperplane is biased.

Linear classification

$w \in R^d -$ vector of weights, it specifies the direction in which the hyperplane divides the data.

2. Optimal Hyperplane

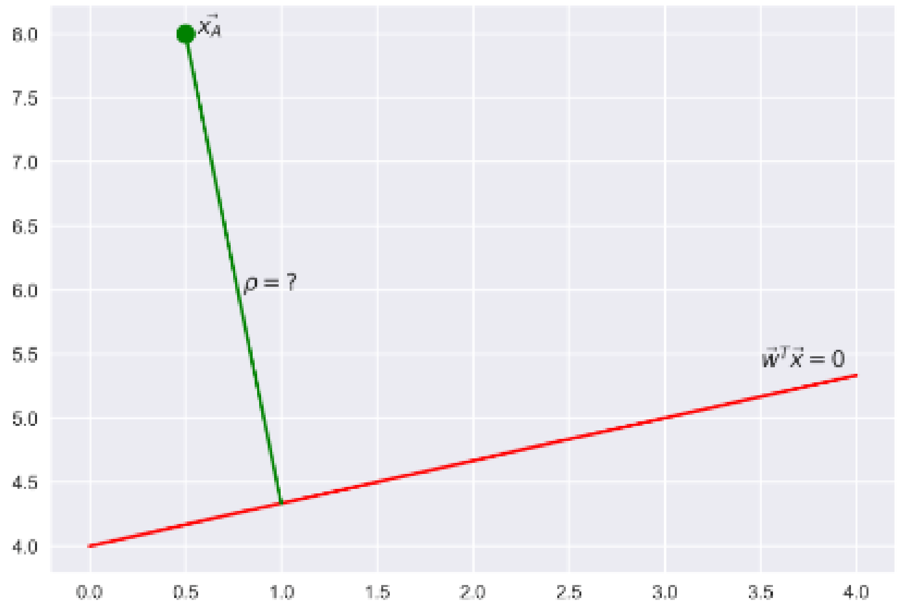
The optimal hyperplane should maximize the margin between the two classes. The margin is the distance from the hyperplane to the nearest points of each class (support vectors).

Linear classification

To be precise, the distance from a point with a radius vector x_A to plane $\langle w, x \rangle = 0$:

$$\rho(x_A, \langle w, x \rangle = 0) = \frac{\langle w, x \rangle}{\|w\|}$$

To maximize the margin, we need to minimize $\|w\|$, as the margin is inversely proportional to the norm of w.



SVM for linearly separable sampling

So, the corresponding optimization problem (hard margin support vector machine):

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w, x_i\rangle + b) \geq 1, i = 1, \dots, l \end{cases}$$

The functional is strictly convex, and the constraints are linear.

Solving the support vector machine problem

If the data is not perfectly separable, a regularization parameter C is introduced, which allows some points to violate the margin boundary **(soft margin support vector machine):** :

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Solving the support vector machine problem

Let us construct a dual problem to the support vector machine problem. Let's write the Lagrangian:

$$L(w, b, \xi, \lambda, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\lambda_i\left[y_i(\langle w, x_i\rangle + b) - 1 + \xi_i\right] - \sum_{i=1}^{l}\mu_i\,\xi_i$$

Solving the support vector machine problem

Depending on the values of ξ_i and λ_i objects x_i are divided into three categories:

•$\xi_i = 0, \lambda_i = 0$. Such objects do not influence the decision and lie outside the dividing line. Objects in this category are called peripheral.

•$\xi_i = 0$, $0 < \lambda_i < C$. From condition $y_i(\langle w, x_i\rangle + b) = 1 - \xi_i$ we have $y_i(\langle w, x_i\rangle + b) = 1$. The object lies strictly on the border of the dividing strip.

•$\xi_i = 0, \lambda_i = C$. Such objects may lie inside the dividing strip or extend beyond it. If $0 < \xi_i < 1$ then the object is classified correctly, otherwise - incorrectly. Objects in this category are called support intruders.

Support Vector Machine in Python

Solving the problem of binary (when there are only two classes) classification:

•the algorithm is trained on objects from the training set for which the class labels are known in advance

•the already trained algorithm predicts the class label for each object from the delayed/test sample

Class labels can take values $Y = \{-1, +1\}$.

Object is a vector with n features $x = (x_1, x_2, \dots, x_n)$ in R^n.

.

Support Vector Machine in Python

The main goal of SVM as a classifier is to find the equation of the separating hyperplane $w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_0 = 0$.

General view of the function: $F(x) = sign(w^T x + b)$

After adjusting the weights of the algorithm, all objects falling on one side of the constructed hyperplane will be predicted as the first class, and objects falling on the other side will be predicted as the second class.

Support Vector Machine in Python

We configure the separating hyperplane in such a way that class objects lie as far as possible from the separating hyperplane.

The algorithm maximizes the gap between the hyperplane and the class objects that are closest to it.

Support Vector Machine in Python

So, we need to maximize gap between two classes. Vector w is the normal vector to the separating hyperplane.

Distance between classes: $\frac{2}{\|w\|} \to max \Rightarrow \|w\| \to min$

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w, x_i \rangle + b) \geq 1, i = 1, \dots, l \end{cases}$$

We got a default SVM setting with a hard-margin SVM, when no object is allowed to fall into the separation strip

Support Vector Machine in Python

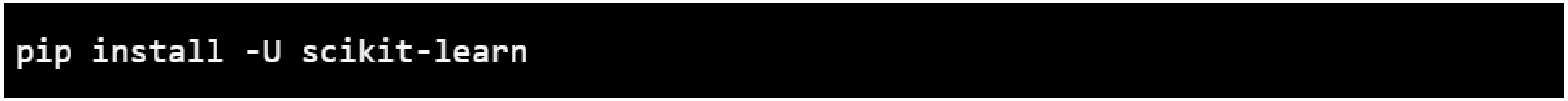
For linearly inseparable classes, we will allow the algorithm to make errors on training objects, but at the same time we will try to ensure that there are fewer errors:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i, i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases}$$

Let’s substitute it into the functional and get an unconditional optimization problem:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\max\big(0,1 - y_i(\langle w, x_i\rangle + b)\big) \to \min_{w,b}$$

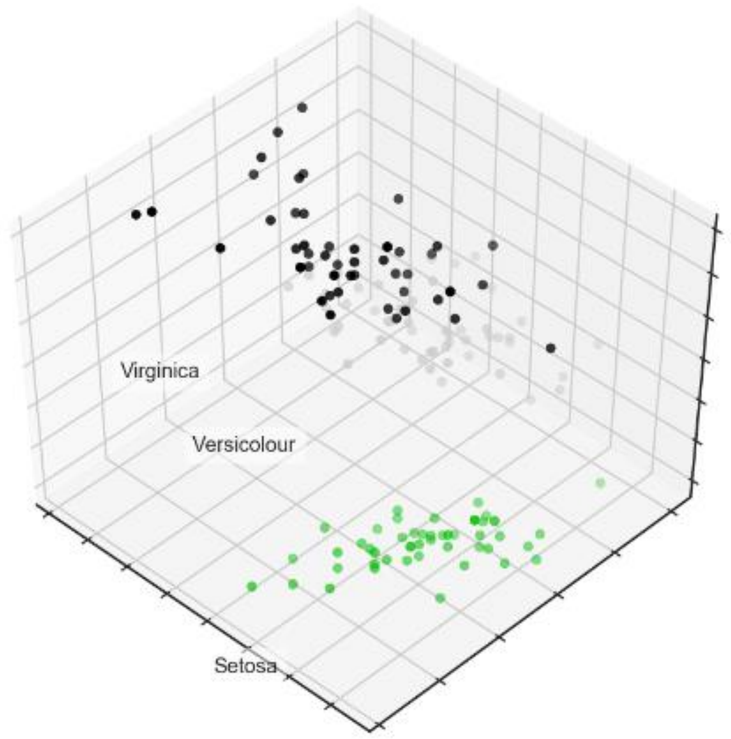
Support Vector Machine in Python



```
pip install -U scikit-learn
```

Support Vector Machine in Python

Let's start by loading all the necessary modules and spinning the usual dataset with **irises** following the example from the documentation of the scikit-learn package.



Support Vector Machine in Python

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper.

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Support Vector Machine in Python

Description of the dataset.

1. Classes (Iris species)

The dataset contains three classes:

-Iris setosa,

-Iris versicolor,

-Iris virginica.

Each class is represented by 50 samples.

Support Vector Machine in Python

2. Features

For each sample, 4 features were measured:

-Sepal length in cm,

-Sepal width in cm,

-Petal length in cm,

Petal width in cm.

Thus, each sample is described by 4 numerical features.

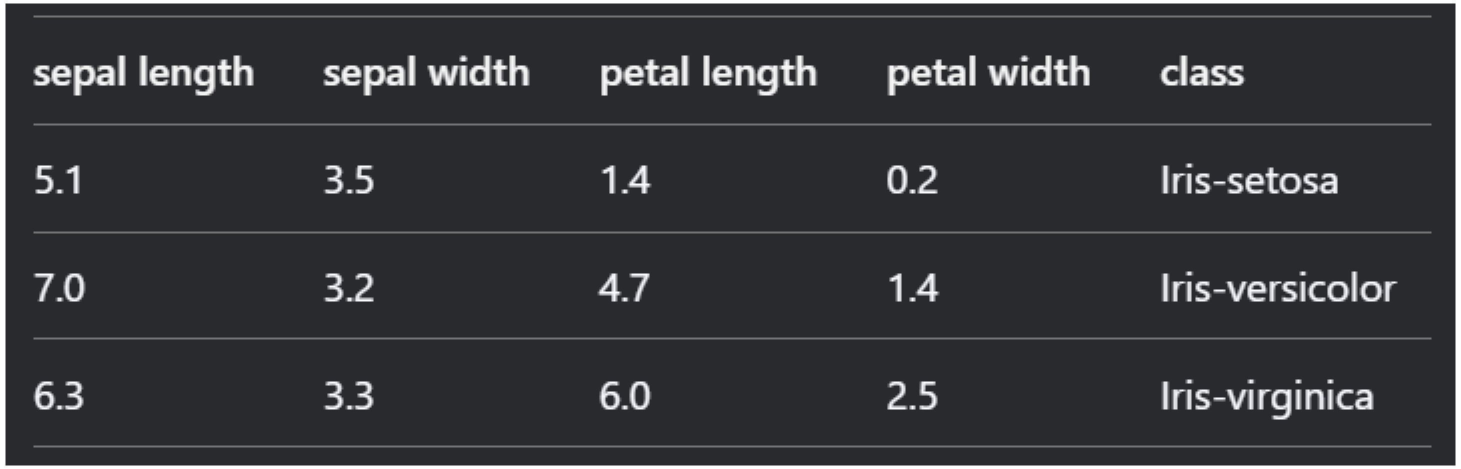
Support Vector Machine in Python

3. Data structure

The data set is usually represented as a table, where:

-Rows correspond to samples (150 rows),

-Columns correspond to features and class label (5 columns).



sepal length	sepal width	petal length	petal width	class
5.1	3.5	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica

Example of data

Support vector machine (SVM) for image classification

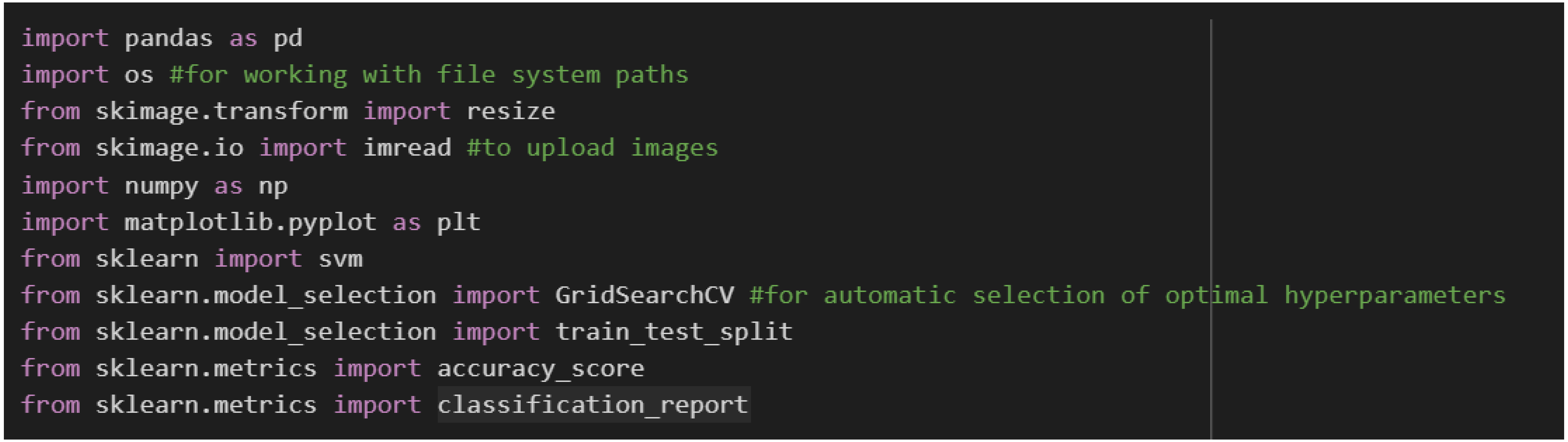
Let’s consider using SVMs for image classification.

Image is a two-dimensional array of pixels. The size of the array corresponds to the resolution of the image, for example, if the image is 200 pixels wide and 200 pixels tall, the array will have the dimensions 200 x 200 x 3. The third dimension represents the RGB color channels.

One of the main advantages of using SVMs for image classification is that they can effectively handle high-dimensional data, such as images.

Support vector machine (SVM) for image classification

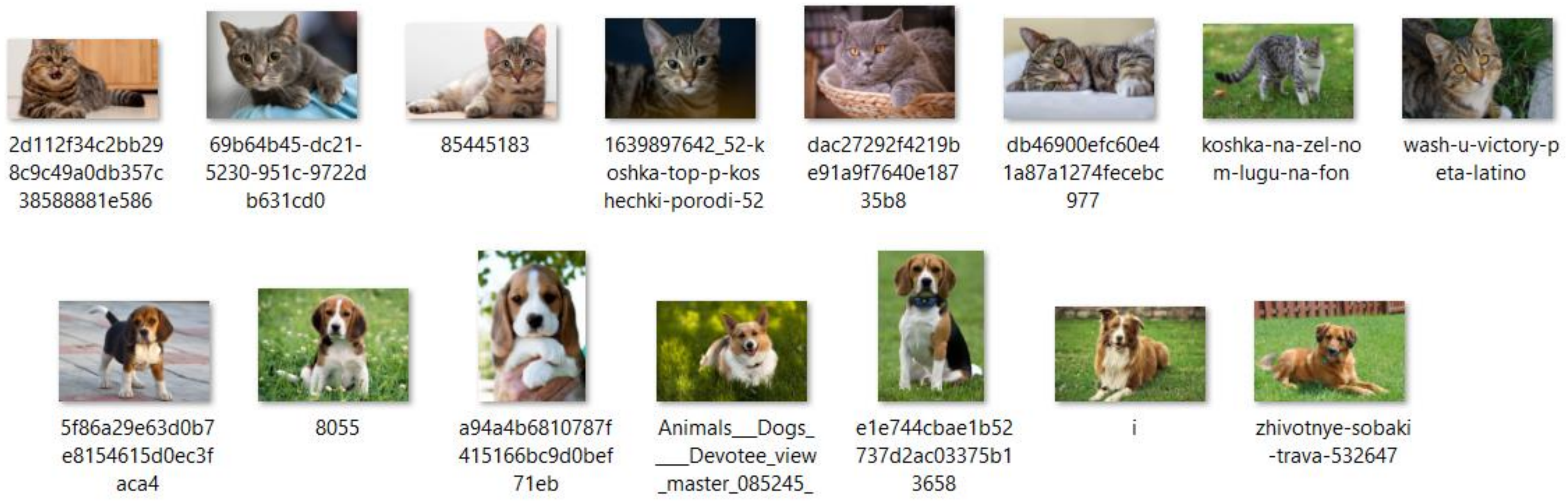
Step 1:Import required libraries.



```
import pandas as pd
import os #for working with file system paths
from skimage.transform import resize
from skimage.io import imread #to upload images
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import GridSearchCV #for automatic selection of optimal hyperparameters
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

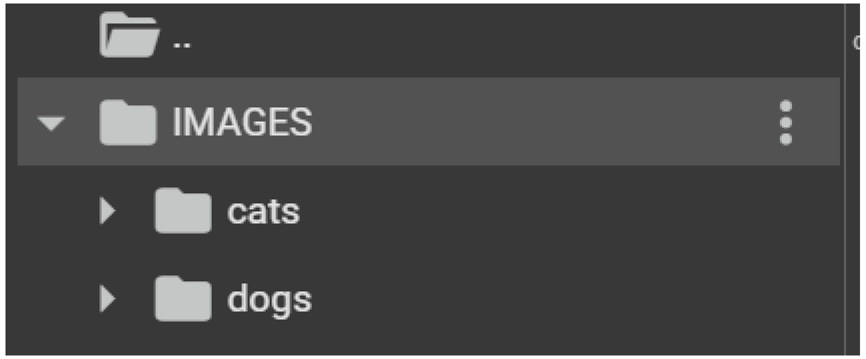
Support vector machine (SVM) for image classification

Step 2: Load the image and convert it to a data frame.

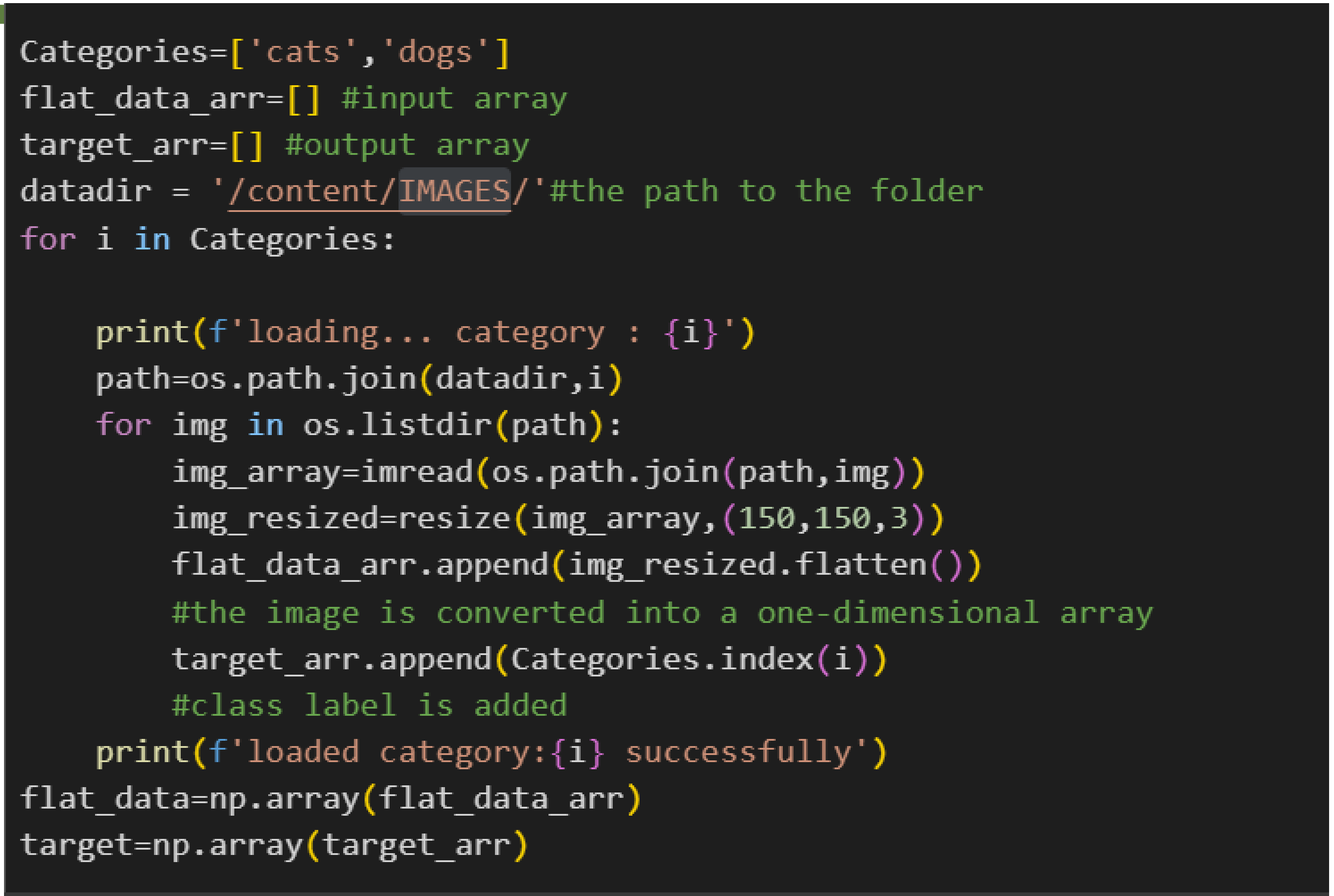

2d112f34c2bb298c9c49a0db357c38588881e586
69b64b45-dc21-5230-951c-9722db631cd0
85445183
1639897642_52-koshka-top-p-koshechki-porodi-52
dac27292f4219be91a9f7640e18735b8
db46900efc60e41a87a1274fecebc977
koshka-na-zel-nom-lugu-na-fon
wash-u-victory-peta-latino
5f86a29e63d0b7e8154615d0ec3faca4
8055
a94a4b6810787f415166bc9d0bef71eb
Animals__Dogs____Devotee_view_master_085245_
e1e744cbae1b52737d2ac03375b13658
i
zhivotnye-sobaki-trava-532647

Support vector machine (SVM) for image classification

Step 2: Load the image and convert it to a data frame.



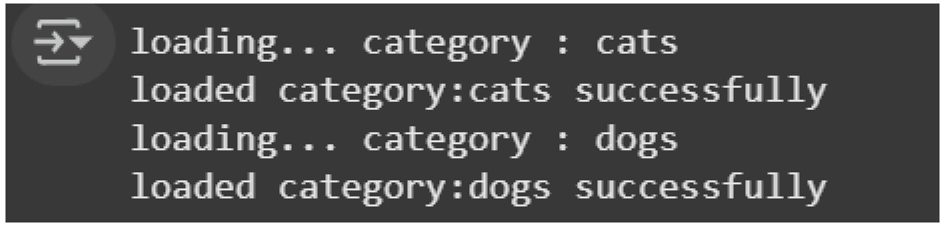
Support vector machine (SVM) for image classification



```python
Categories=['cats','dogs']
flat_data_arr=[] #input array
target_arr=[] #output array
datadir = '/content/IMAGES/'#the path to the folder
for i in Categories:

    print(f'loading... category : {i}')
    path=os.path.join(datadir,i)
    for img in os.listdir(path):
        img_array=imread(os.path.join(path,img))
        img_resized=resize(img_array,(150,150,3))
        flat_data_arr.append(img_resized.flatten())
        #the image is converted into a one-dimensional array
        target_arr.append(Categories.index(i))
        #class label is added
    print(f'loaded category:{i} successfully')
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
```

Support vector machine (SVM) for image classification

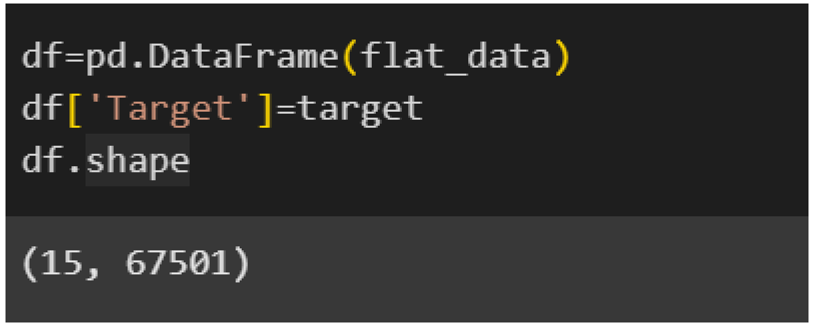
Step 2: Load the image and convert it to a data frame.



```
loading... category : cats
loaded category:cats successfully
loading... category : dogs
loaded category:dogs successfully
```

Support vector machine (SVM) for image classification

Step 2: Load the image and convert it to a data frame.

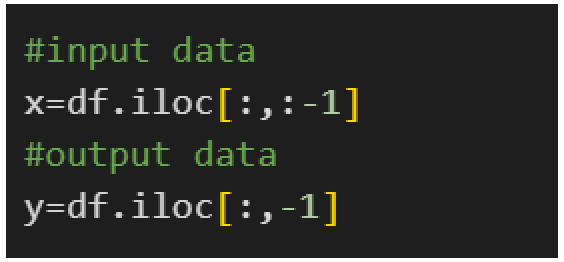


```
df=pd.DataFrame(flat_data)
df['Target']=target
df.shape
```

```
(15, 67501)
```

Support vector machine (SVM) for image classification

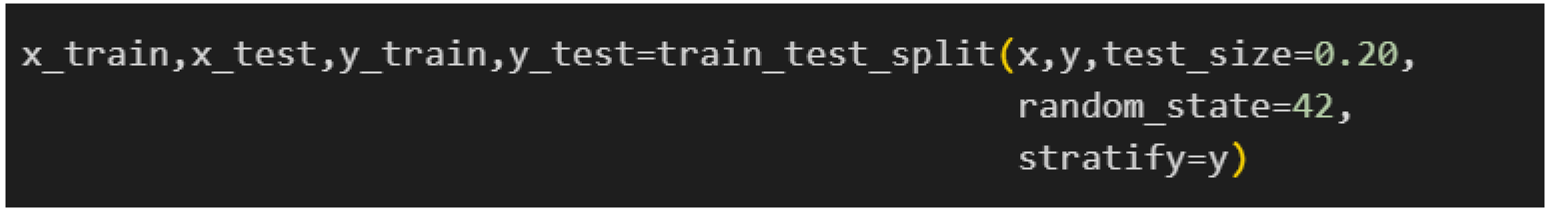
Step 3: Separate input features and targets.



```
#input data
x=df.iloc[:,:-1]
#output data
y=df.iloc[:,-1]
```

Support vector machine (SVM) for image classification

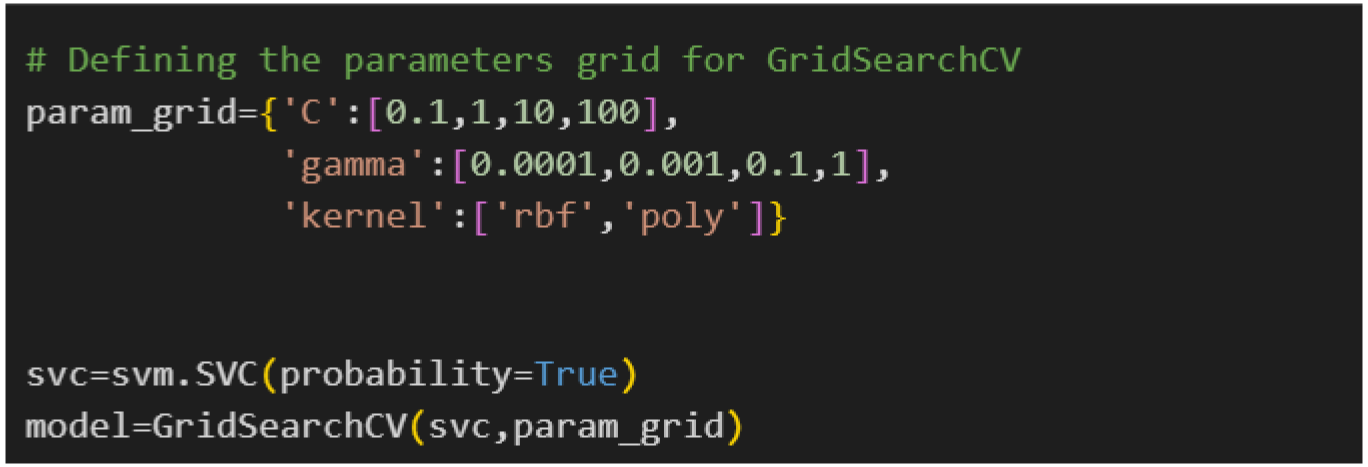
Step 4: Separate input features and targets.



```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,
                                                random_state=42,
                                                stratify=y)
```

Support vector machine (SVM) for image classification

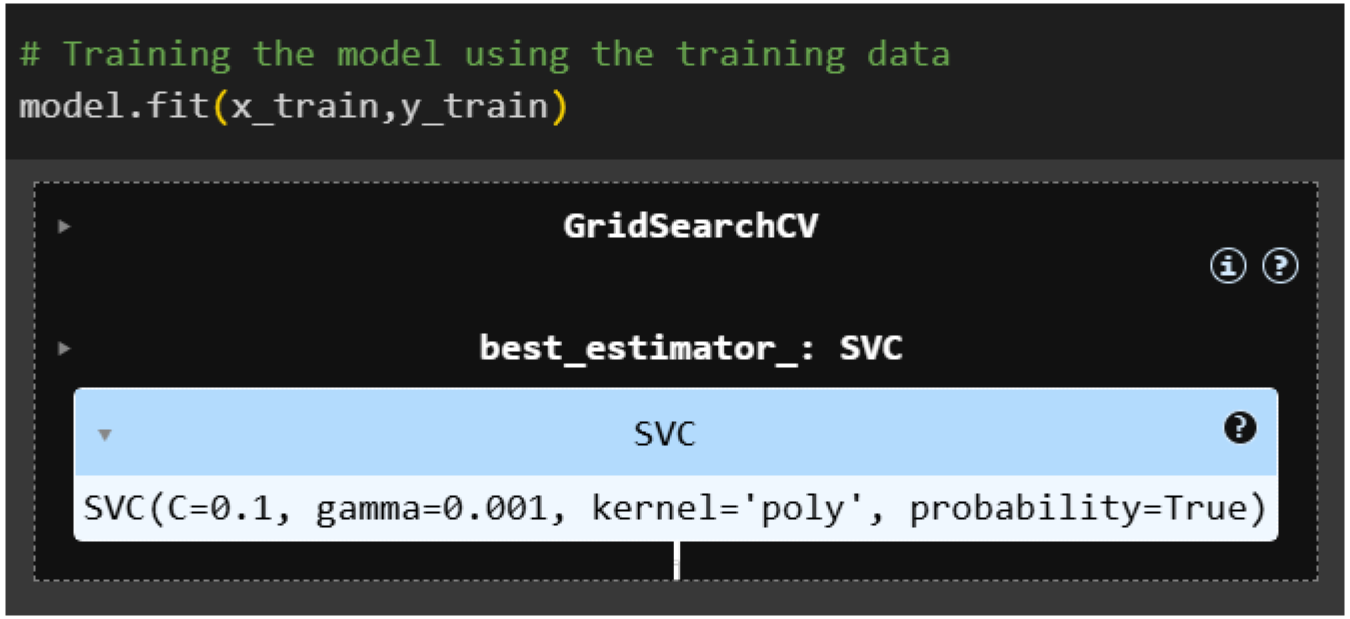
Step 5: Build and train the model



```
# Defining the parameters grid for GridSearchCV
param_grid={'C':[0.1,1,10,100],
            'gamma':[0.0001,0.001,0.1,1],
            'kernel':['rbf','poly']}


svc=svm.SVC(probability=True)
model=GridSearchCV(svc,param_grid)
```

Support vector machine (SVM) for image classification

Model training



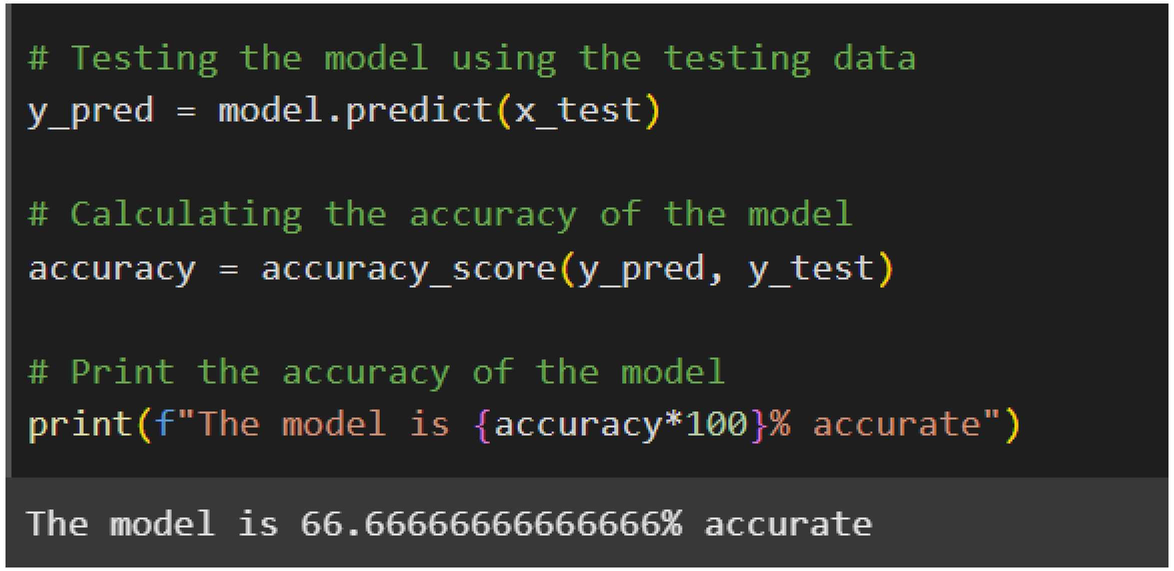
```
# Training the model using the training data
model.fit(x_train,y_train)
```

```
GridSearchCV
best_estimator_: SVC
SVC
SVC(C=0.1, gamma=0.001, kernel='poly', probability=True)
```

Support vector machine (SVM) for image classification

Step 6: Model evaluation

Now the model is tested using testing data in this way **model.predict()** and the accuracy of the model can be calculated using the **accuracy_score()** method from **sklearn.metrics.**



```
# Testing the model using the testing data
y_pred = model.predict(x_test)

# Calculating the accuracy of the model
accuracy = accuracy_score(y_pred, y_test)

# Print the accuracy of the model
print(f"The model is {accuracy*100}% accurate")
```

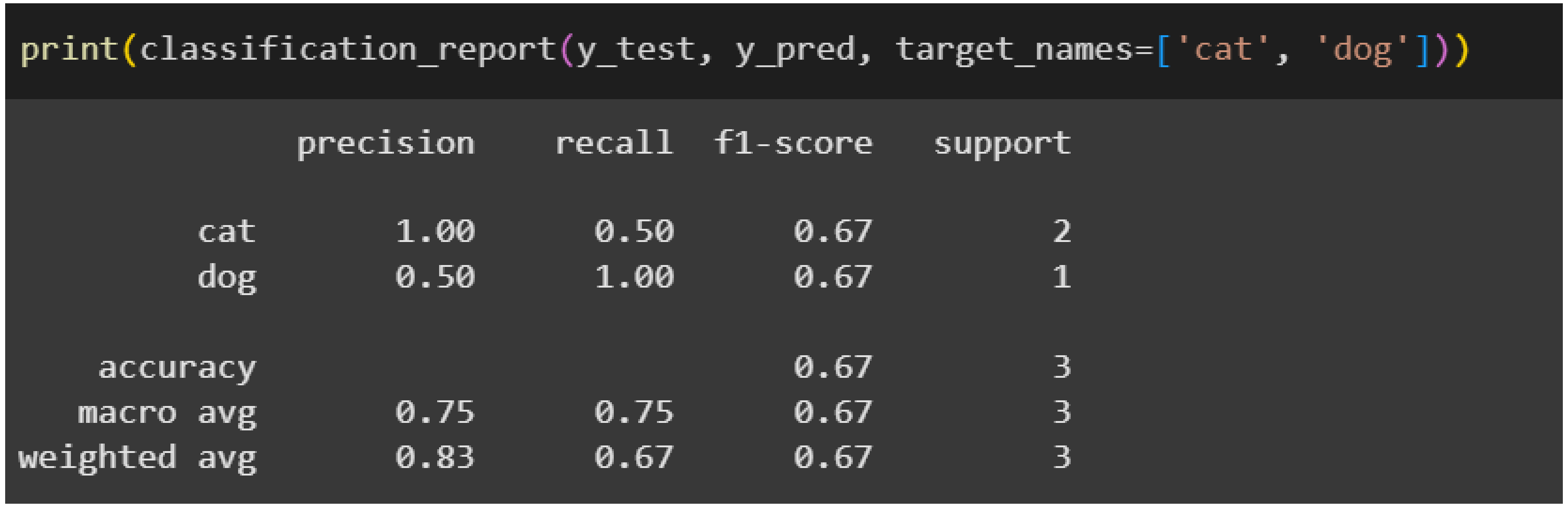
```
The model is 66.66666666666666% accurate
```

Support vector machine (SVM) for image classification

Classification Report.

Precision: The proportion of correctly predicted positive classes among all predicted positive classes

Recall: The proportion of correctly predicted positive classes among all actual positive classes.



```
print(classification_report(y_test, y_pred, target_names=['cat', 'dog']))
```

```
              precision    recall  f1-score   support

         cat       1.00      0.50      0.67         2
         dog       0.50      1.00      0.67         1

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

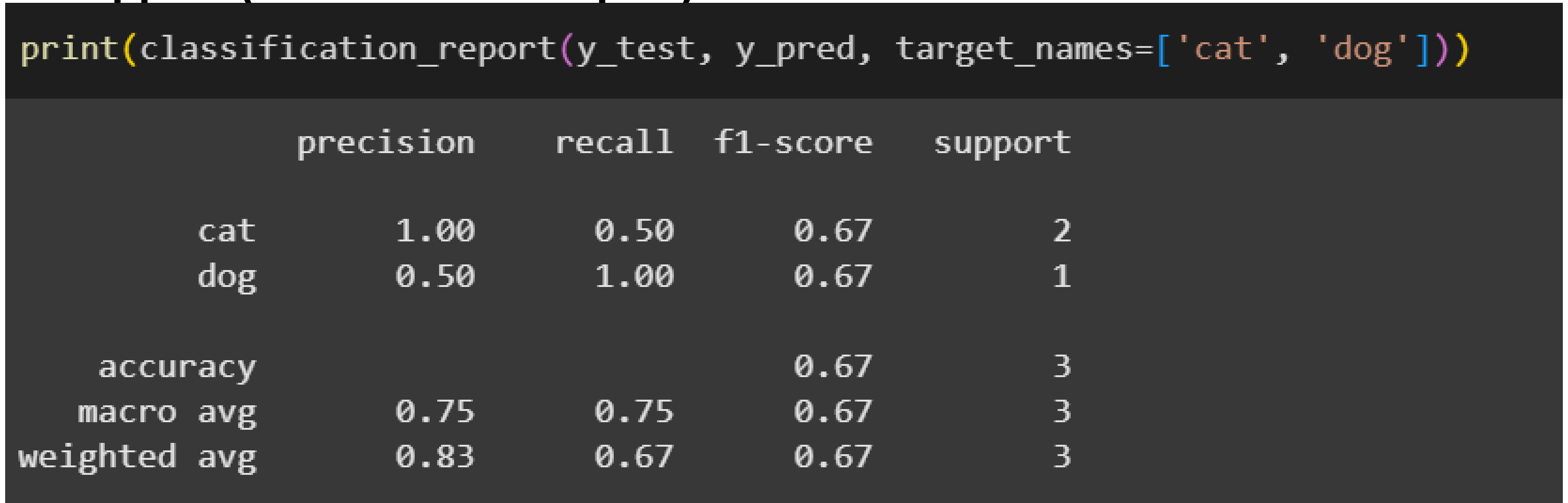
$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Support vector machine (SVM) for image classification

Support: The number of actual examples for each class in the test data.

Macro Avg (Average macro): Average value of metrics (precision, recall, F1-score) for all classes without taking into account class imbalance.

Weighted Avg: Average metrics (precision, recall, F1-score) across all classes, weighted by the support (number of examples) of each class



```
print(classification_report(y_test, y_pred, target_names=['cat', 'dog']))
```

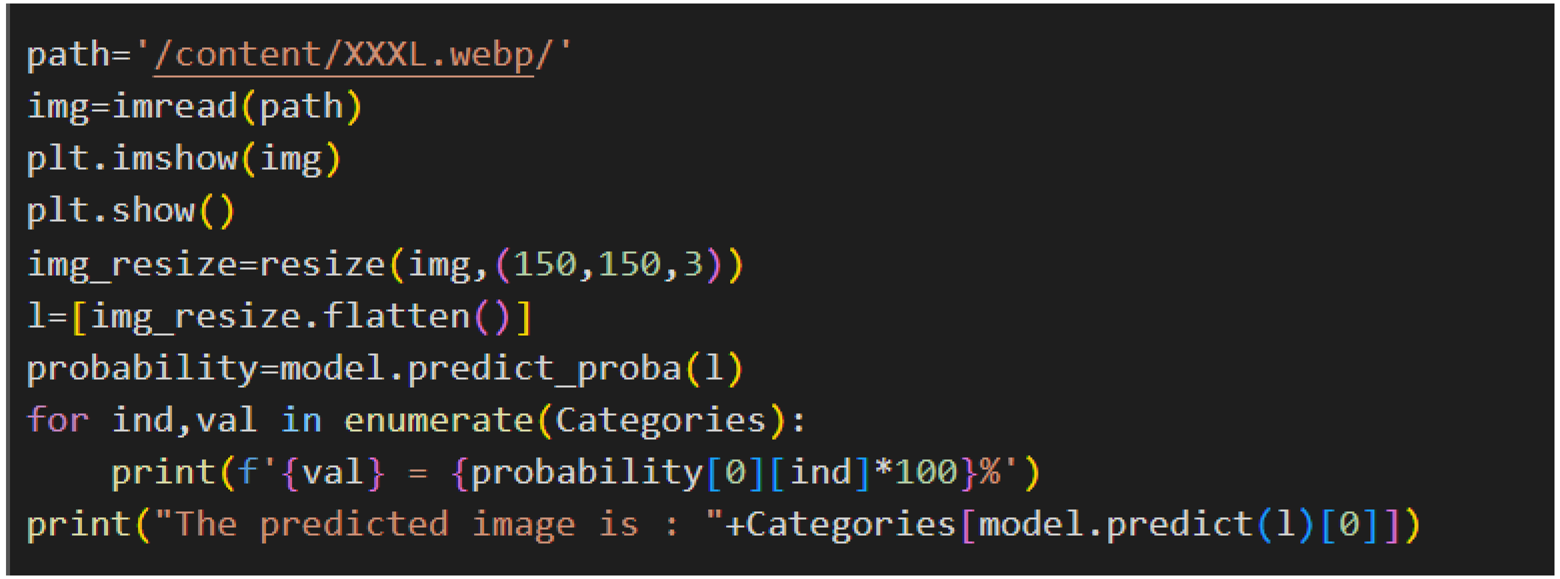
```
              precision    recall  f1-score   support

         cat       1.00      0.50      0.67         2
         dog       0.50      1.00      0.67         1

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

Support vector machine (SVM) for image classification

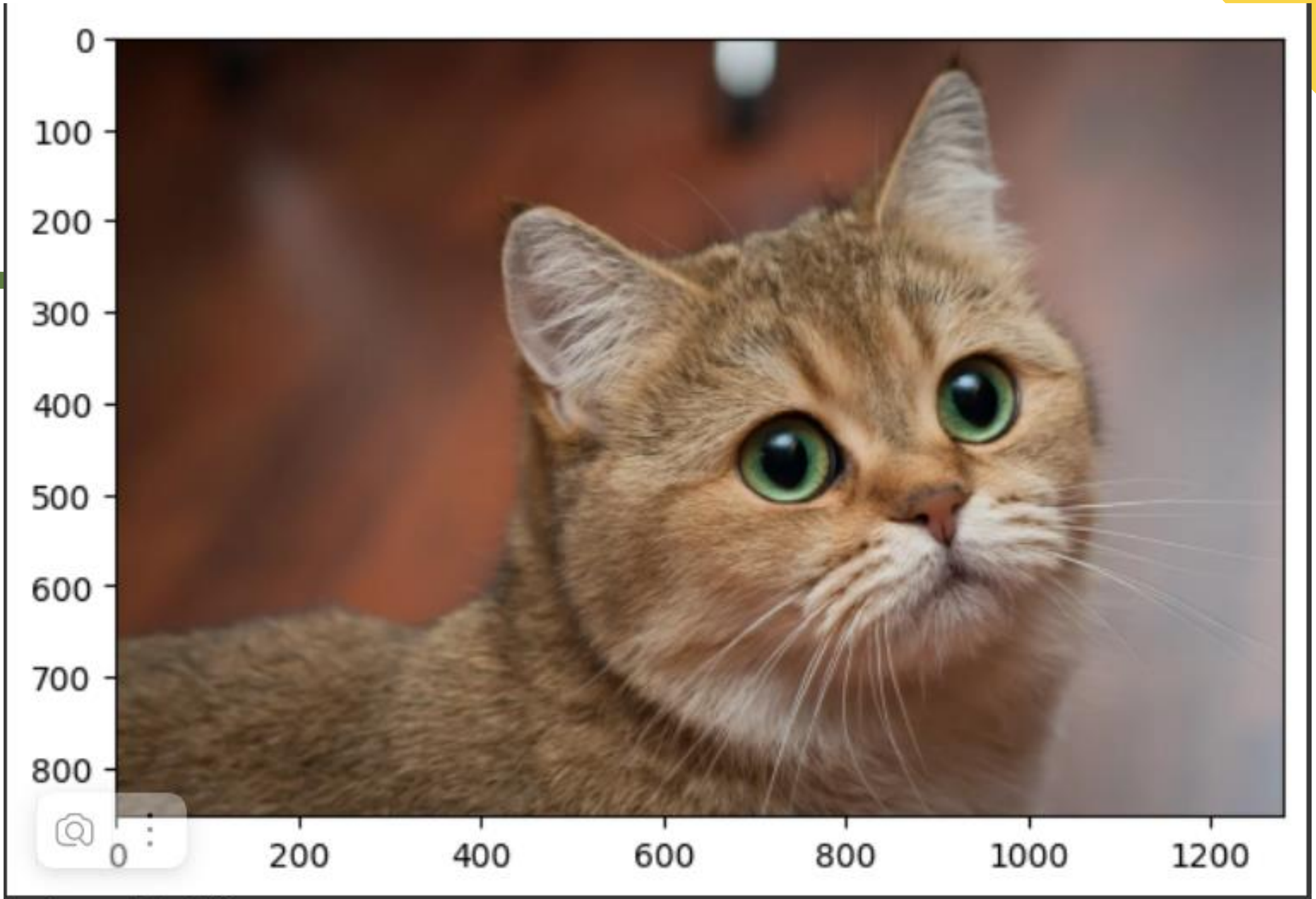
Step 7: Prediction

Now we will give a new image to our model and it will predict whether the given image is of cat or dog

Support vector machine (SVM) for image classification



```python
path='/content/XXXL.webp/'
img=imread(path)
plt.imshow(img)
plt.show()
img_resize=resize(img,(150,150,3))
l=[img_resize.flatten()]
probability=model.predict_proba(l)
for ind,val in enumerate(Categories):
    print(f'{val} = {probability[0][ind]*100}%')
print("The predicted image is : "+Categories[model.predict(l)[0]])
```

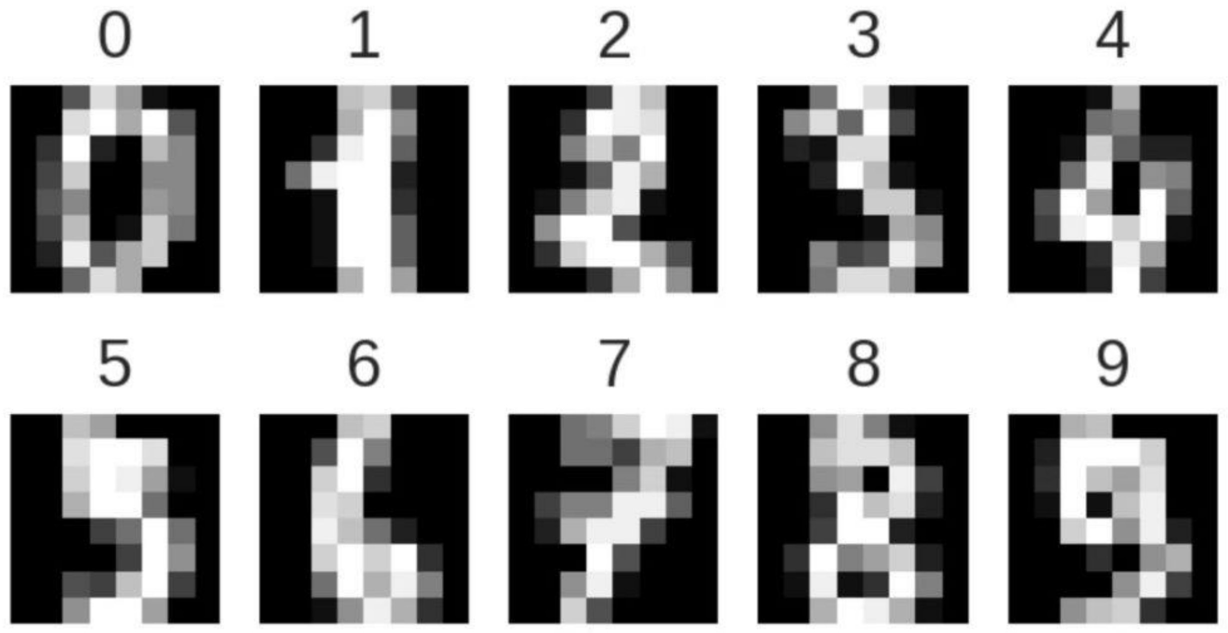


```
cats = 50.0%
dogs = 50.0%
The predicted image is : cats
```

Support vector machine (SVM) for image classification

Image Classification Using MNIST Dataset as an Example.

Let's take the classic MNIST dataset, which contains images of handwritten digits from 0 to 9.

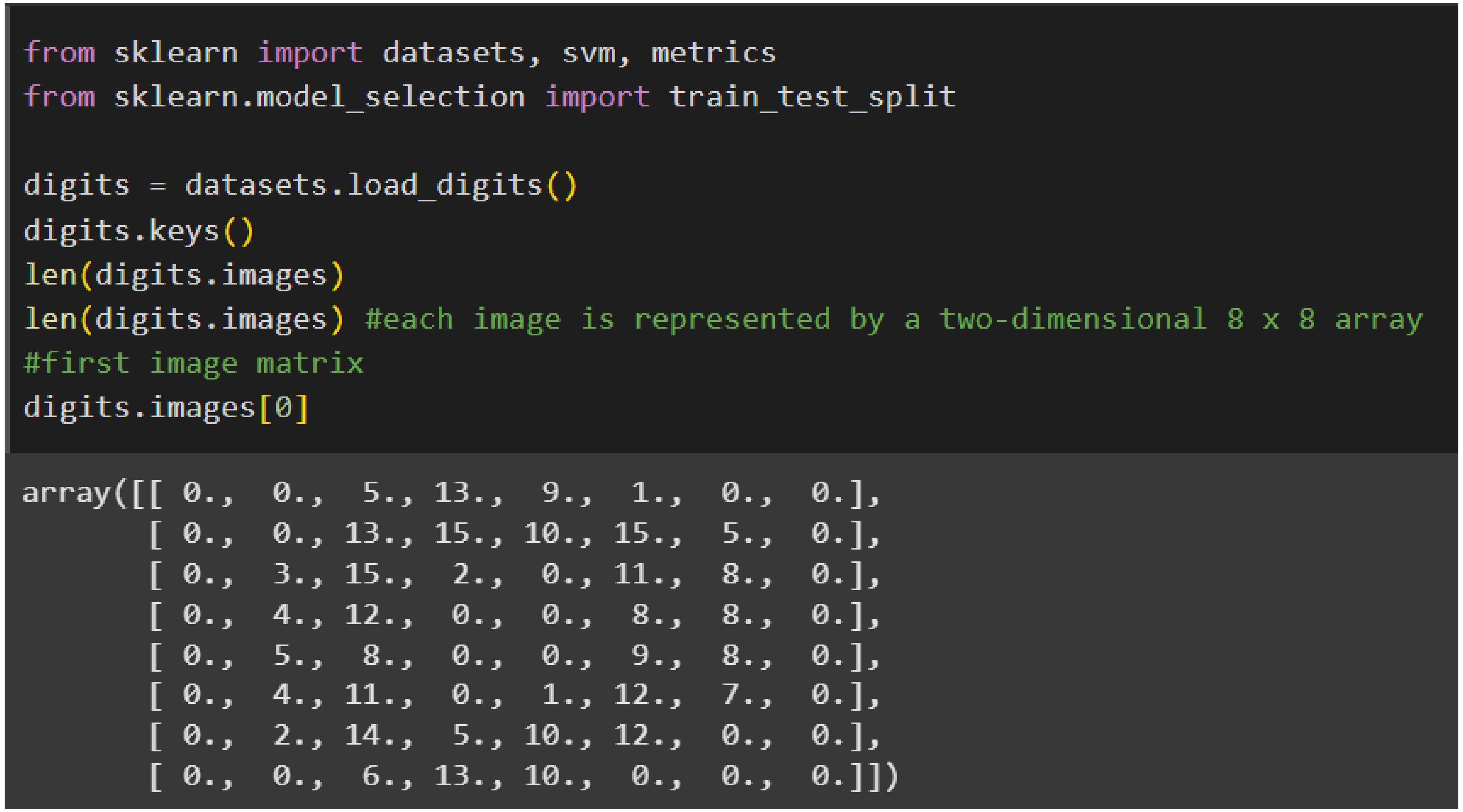


Support vector machine (SVM) for image classification

After training, the model will be able to independently determine what is presented in the image.
In this case, there are not two classes, but ten.

First, we import the necessary libraries and the dataset itself from the Scikit-learn library.

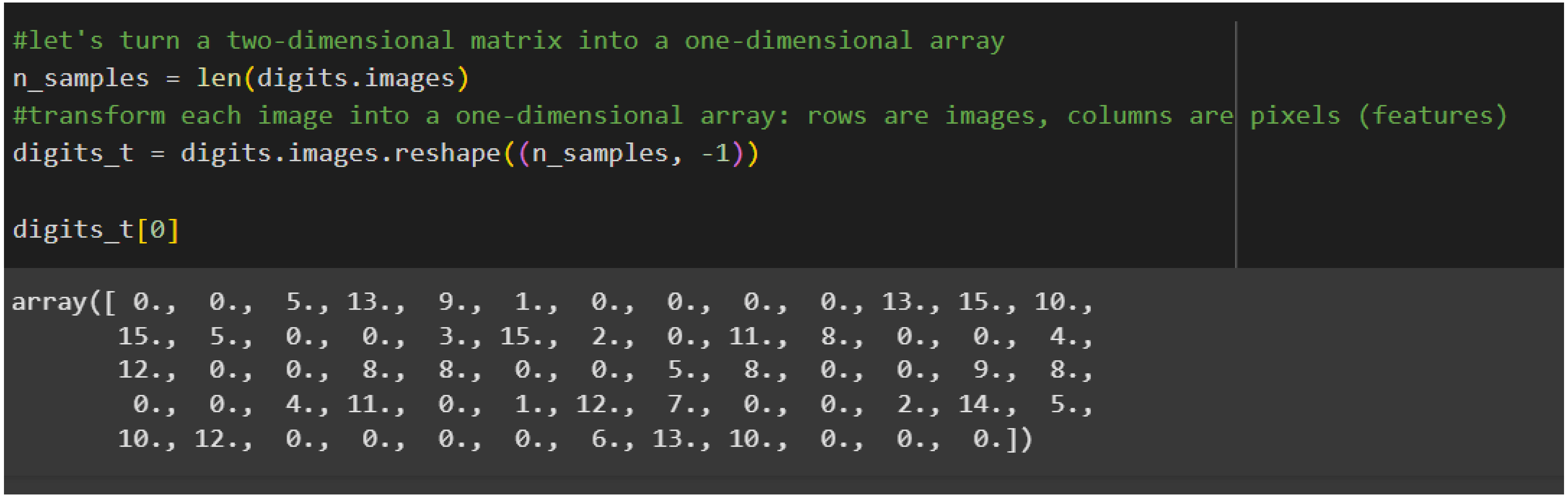
Support vector machine (SVM) for image classification



```
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()
digits.keys()
len(digits.images)
len(digits.images) #each image is represented by a two-dimensional 8 x 8 array
#first image matrix
digits.images[0]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

Support vector machine (SVM) for image classification

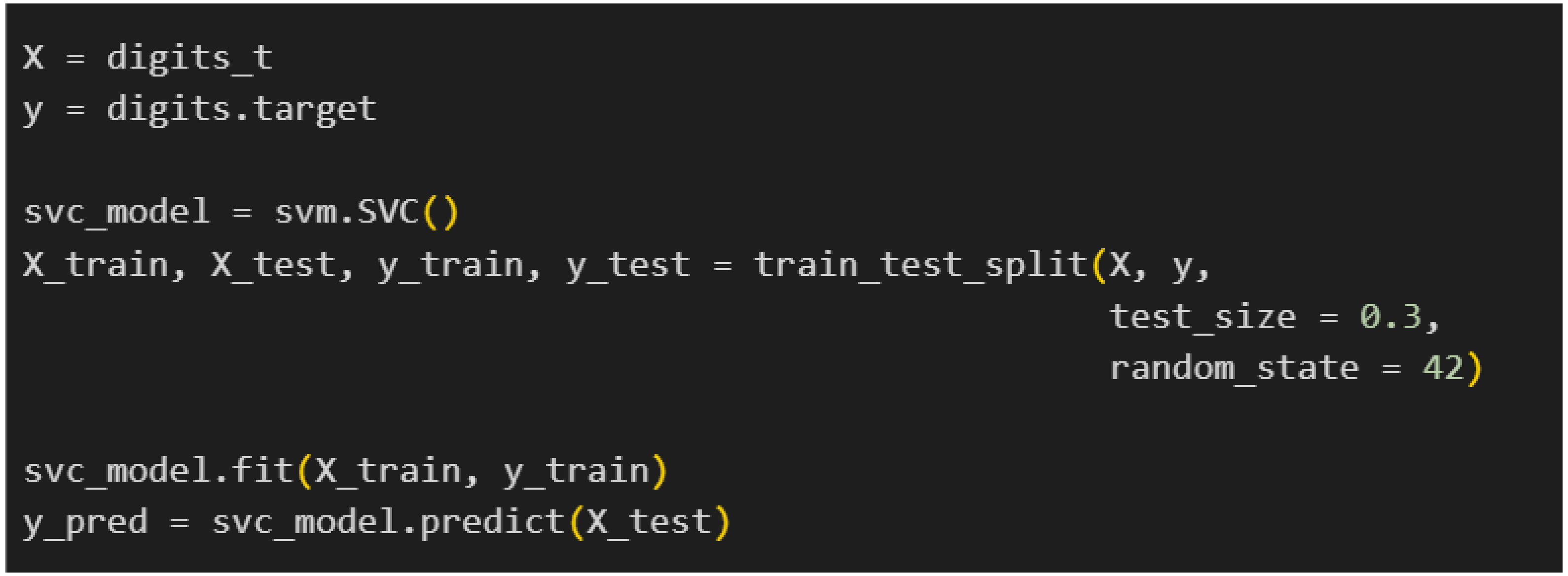


```
#let's turn a two-dimensional matrix into a one-dimensional array
n_samples = len(digits.images)
#transform each image into a one-dimensional array: rows are images, columns are pixels (features)
digits_t = digits.images.reshape((n_samples, -1))

digits_t[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

Support vector machine (SVM) for image classification

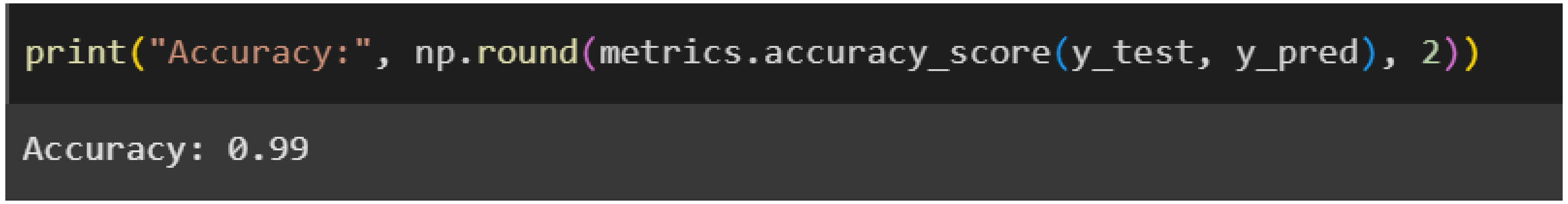
In the process of training the model, we first split the data into training and test samples. We then use support vector machines to train the model and make a forecast.



```
X = digits_t
y = digits.target

svc_model = svm.SVC()
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 42)


svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)
```

Support vector machine (SVM) for image classification

To evaluate the quality of the model, we will first use the accuracy metric. It shows the proportion of correct predictions. In the case of multi-class classification, it is calculated as the arithmetic mean of accuracy across all classes.

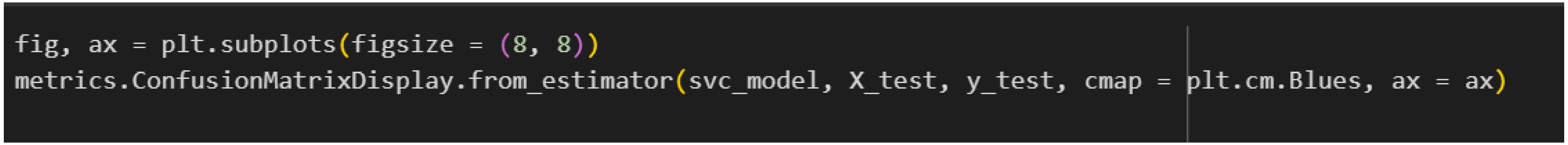


```
print("Accuracy:", np.round(metrics.accuracy_score(y_test, y_pred), 2))
```

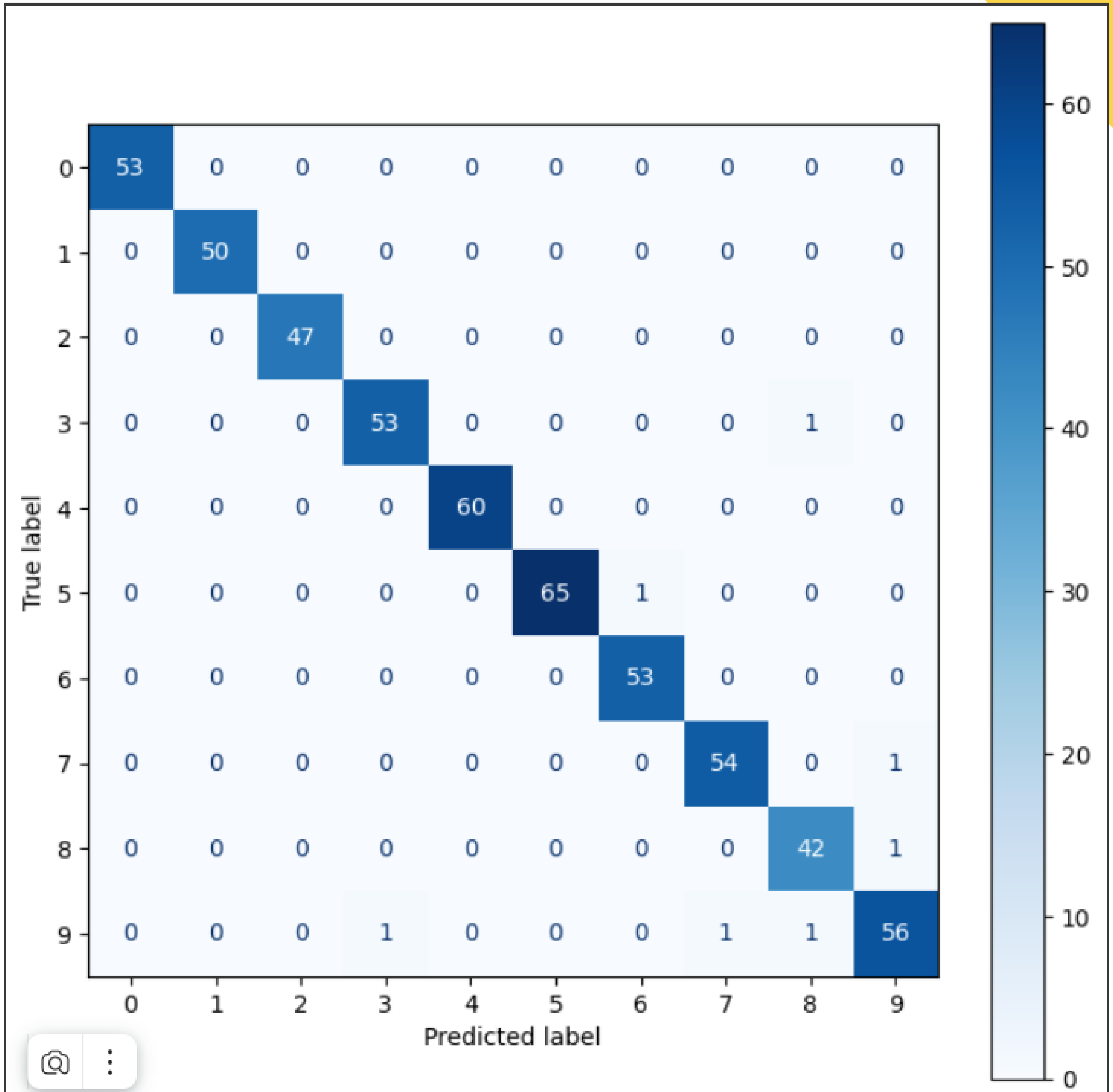
```
Accuracy: 0.99
```

Support vector machine (SVM) for image classification

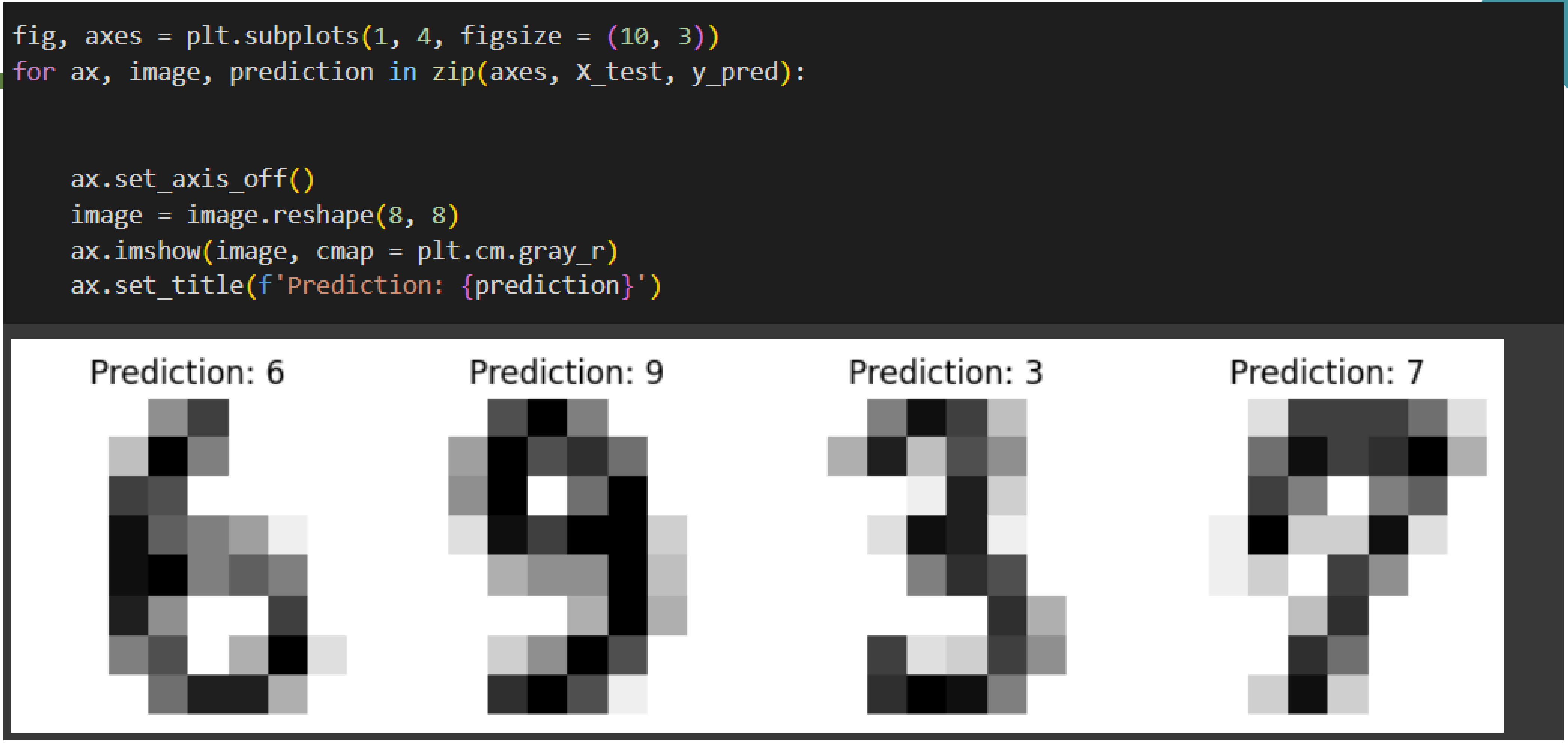
Accuracy for each class can be conveniently viewed using the error matrix.



```
fig, ax = plt.subplots(figsize = (8, 8))
metrics.ConfusionMatrixDisplay.from_estimator(svc_model, X_test, y_test, cmap = plt.cm.Blues, ax = ax)
```



	0	1	2	3	4	5	6	7	8	9
0	53	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	47	0	0	0	0	0	0	0
3	0	0	0	53	0	0	0	0	1	0
4	0	0	0	0	60	0	0	0	0	0
5	0	0	0	0	0	65	1	0	0	0
6	0	0	0	0	0	0	53	0	0	0
7	0	0	0	0	0	0	0	54	0	1
8	0	0	0	0	0	0	0	0	42	1
9	0	0	0	1	0	0	0	1	1	56



```
fig, axes = plt.subplots(1, 4, figsize = (10, 3))
for ax, image, prediction in zip(axes, X_test, y_pred):


    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap = plt.cm.gray_r)
    ax.set_title(f'Prediction: {prediction}')
```