به نام خدا

پروژه چهارم درس هوش مصنوعی – ماشین بردار پشتیبان بهاره کاوسی نژاد – 99431217

بخش 1:

```python
import matplotlib.pyplot as plt
import numpy as np
import random
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

ابتدا کتابخانه های لازم را import می‌کنیم.

```python
def generate_random_dataset(size):
    X = []
    rand = random.randint(1, 199)
    for _ in range(size):
        if rand % 3 == 0:
            X.append([np.round(random.uniform(2, 4), 1),
np.round(random.uniform(0, 15), 1)])
        elif rand % 3 == 1:
            X.append([np.round(random.uniform(1, 5), 2),
np.round(random.uniform(20, 25), 2)])
        else:
            X.append([np.round(random.uniform(3, 8), 2),
np.round(random.uniform(5, 25), 2)])
    return X
```

از این تابع به منظور generate کردن یک dataset تصادفی استفاده می‌شود.

```python
def makePoints(dataset):
    Y = []
    data = sorted(dataset, key=lambda x: x[0])
    max_val = data[len(data) - 1][0]
    min_val = data[0][0]
    s1 = (max_val + min_val) / 2
    for ix in dataset:
        if ix[0] <= s1:
            Y.append(0)
        else:
            Y.append(1)
    return Y
```

از تابع makePoints برای تعیین برچسب‌های dataset تولید شده توسط تابع generate_random_dataset استفاده می‌شود. برای تعیین این برچسب‌ها از یک threshold استفاده می‌شود.

```python
def ClassifiyedPoints(X, Y):
    x0 = []
    x1 = []
    for i in range(len(X)):
        if Y[i] == 1:
            x1.append(X[i])
        else:
            x0.append(X[i])
    return x0, x1
```

بر اساس برچسب‌های تعیین شده در قسمت قبل، داده‌ها را classify می‌کنیم و آنها را به دو کلاس تقسیم می‌کنیم.

```python
def plot_decision_boundary(ax, pred_func, X, y):
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.3)
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
```

از تابع plot_decision_boundry برای ترسیم دیتاهای تولید شده و classify شدن آنها استفاده می‌شود.

```python
X_train = []
X_test = []
Y_train = []
Y_test = []
x0_vals = []
x1_vals = []
o = -1

size = 500
while o == -1:
    dataset = generate_random_dataset(size)
    label = makePoints(dataset)
    dataset = np.array(dataset)
    X_train, X_test, Y_train, Y_test = train_test_split(dataset,
label, test_size=0.2, random_state=42)
    x0_vals, x1_vals = ClassifiyedPoints(X_train, Y_train)

    if x0_vals == 0 or x1_vals == 0:
        o = -1
    else:
        o = 0
```

در این قسمت در یک حلقه while به تولید 500 دیتای تصادفی می‌پردازیم و پس از تعیین برچسب‌های آنها با استفاده از تابع makePoints، 80 درصد آن را به عنوان داده train و 20 درصد را به عنوان داده تست قرار می‌دهیم. با استفاده از تابع ClassifyedPoints آنها را classify می‌کنیم. این حلقه تا زمانی ادامه می‌یابد که مقادیر classify شده صفر نباشند. اگر هردو آنها دارای مقدار باشند، حلقه پایان می‌یابد.

```
# Instantiate the classifier
clf1 = SVC(kernel='linear')
clf2 = SVC(kernel='rbf')
clf3 = SVC(kernel='sigmoid')
clf4 = SVC(kernel='poly', gamma="auto")

# Fit the classifier on the training data
clf1.fit(X_train, Y_train)
clf2.fit(X_train, Y_train)
clf3.fit(X_train, Y_train)
clf4.fit(X_train, Y_train)

# Predictions and Accuracy
predictions = clf1.predict( X_test )
accuracy = accuracy_score(Y_test, predictions)
print( "linear kernel accuracy : " , accuracy )
predictions = clf2.predict( X_test )
accuracy = accuracy_score(Y_test, predictions)
print( "rbf kernel accuracy : " , accuracy )
predictions = clf3.predict( X_test )
accuracy = accuracy_score(Y_test, predictions)
print( "sigmoid kernel accuracy : " , accuracy )
predictions = clf4.predict( X_test )
accuracy = accuracy_score(Y_test, predictions)
print( "poly kernel accuracy : " , accuracy )
```

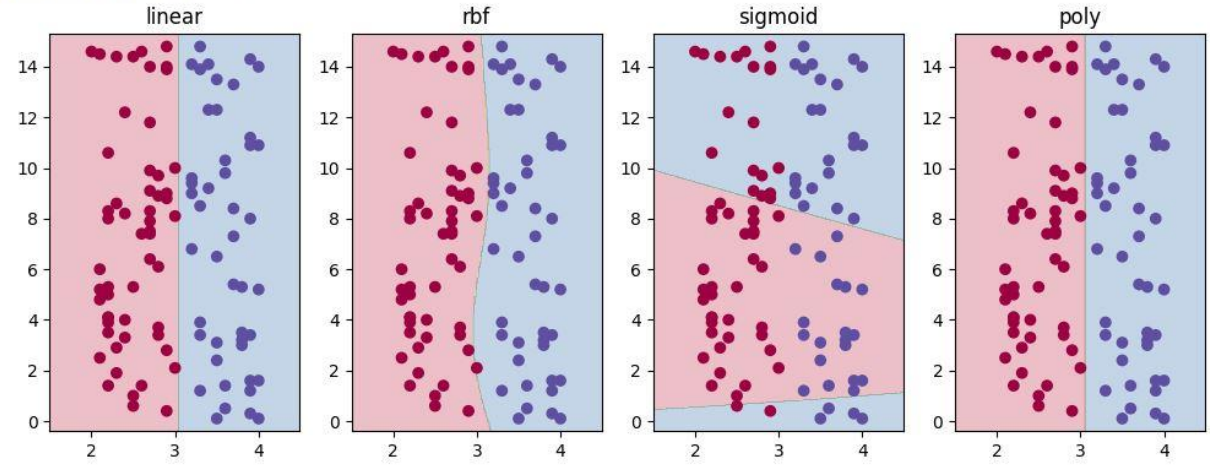
در این قسمت با استفاده از تابع آماده SVC، classifierهایی با چهار kernel متفاوت (linear، rbf، sigmoid و poly) تولید می‌کنیم. هر classifier روی داده‌های train با استفاده از تابع fit، train می‌شود. پس از train شدن، از classifierها برای پیش‌بینی روی داده‌های تست استفاده می‌شود و دقت با تابع accuracy محاسبه و چاپ می‌شود.

```
# Adjust figsize as needed
fig, axs = plt.subplots(1, 4, figsize=(10, 4))

# Plot decision boundary
plot_decision_boundary(axs[0], lambda X_test:
clf1.predict(X_test), X_test, Y_test)
axs[0].set_title('linear')


plot_decision_boundary(axs[1], lambda X_test:
clf2.predict(X_test), X_test, Y_test)
axs[1].set_title('rbf')

plot_decision_boundary(axs[2], lambda X_test:
clf3.predict(X_test), X_test, Y_test)
axs[2].set_title('sigmoid')


plot_decision_boundary(axs[3], lambda X_test:
clf4.predict(X_test), X_test, Y_test)
axs[3].set_title('poly')

plt.tight_layout()

plt.show()
```

در این قسمت plot انجام می‌شود.

نتایج:

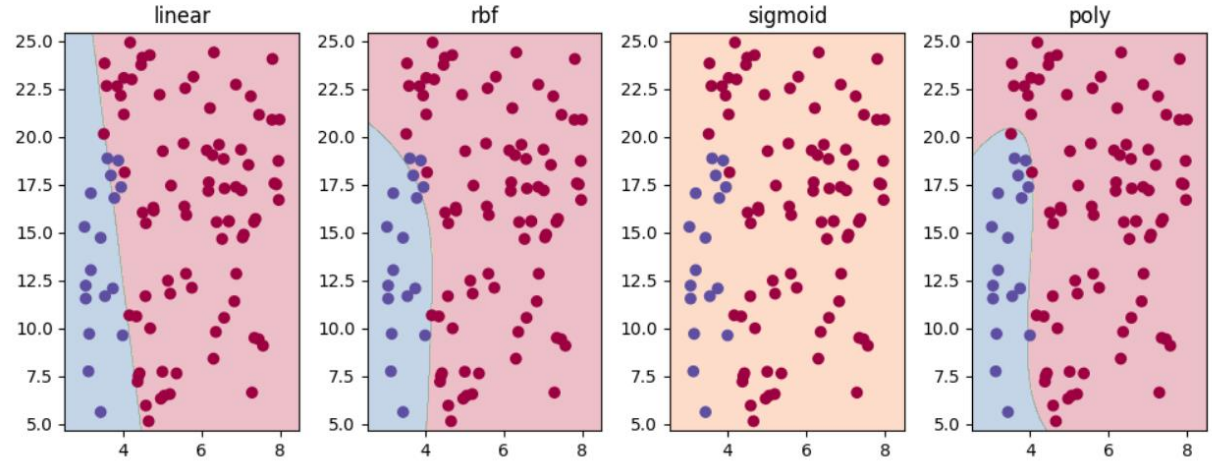
```
linear kernel accuracy :  1.0
rbf kernel accuracy :  0.99
sigmoid kernel accuracy :  0.6
poly kernel accuracy :  1.0
```



```python
def makePoints2(dataset):
  Y = []
  for ix in dataset :
      if(ix[0] <=  4 and ix[1] <= 20):
          Y.append(1)
      else:
          Y.append(0)
  return Y
```

یک تابع makePoints دیگر تعریف می کنیم تا نقاط ورودی تولید شده شکل دیگری داشته باشند و مراحل قبلی را طی می کنیم تا train انجام شود. نتایج به صورت زیر است:

```
linear kernel accuracy :  0.95
rbf kernel accuracy :  0.96
sigmoid kernel accuracy :  0.83
poly kernel accuracy :  0.97
```



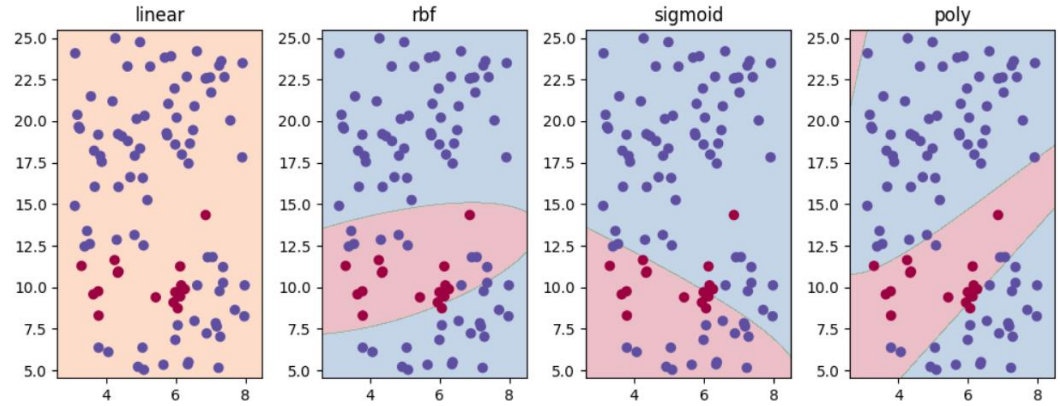
```python
def is_the_point_in_circle(ix , xceneter , ycenter , radious ) :
    rtmp = (ix[0] - xceneter)**2 + (ix[1]-ycenter)**2
    if( rtmp <= radious**2 ) :
        return True
    return False

def turn_points_2Circles(X, radious, xceneter, ycenter) :
    Y = []
    for ix in X:
        if(is_the_point_in_circle(ix, xceneter, ycenter, radious) or is_the_point_in_circle(ix,
xceneter + 3, ycenter + 4, radious - 0.5)) :
            Y.append(0)
        else:
            Y.append(1)
    return Y
```

این بار تابع تولید نقاط را به صورت دایره ای در نظر می گیریم؛ صدا کردن تابع به صورت زیر تغییر می کند:

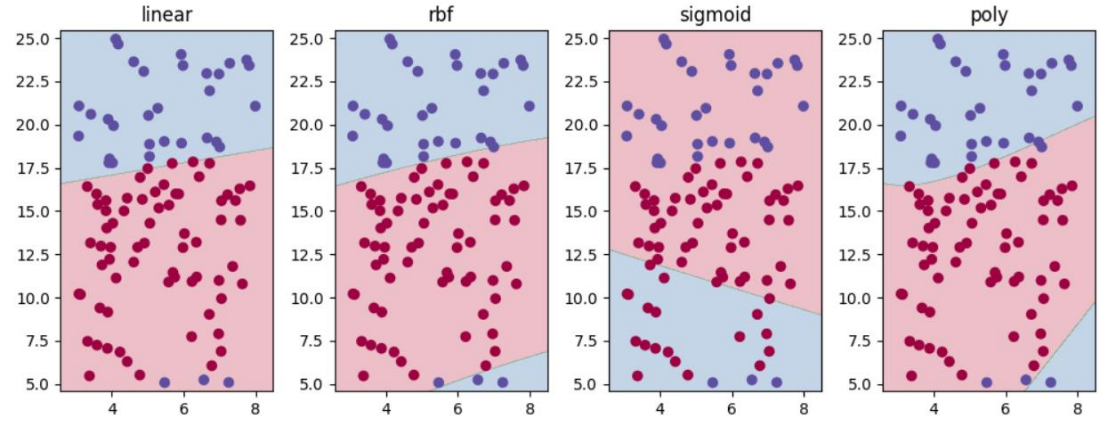
```python
label = turn_points_2Circles(dataset, 5, 4, 10)
```

نتایج به صورت زیر است:

```
linear kernel accuracy :  0.83
rbf kernel accuracy :  0.89
sigmoid kernel accuracy :  0.78
poly kernel accuracy :  0.93
```



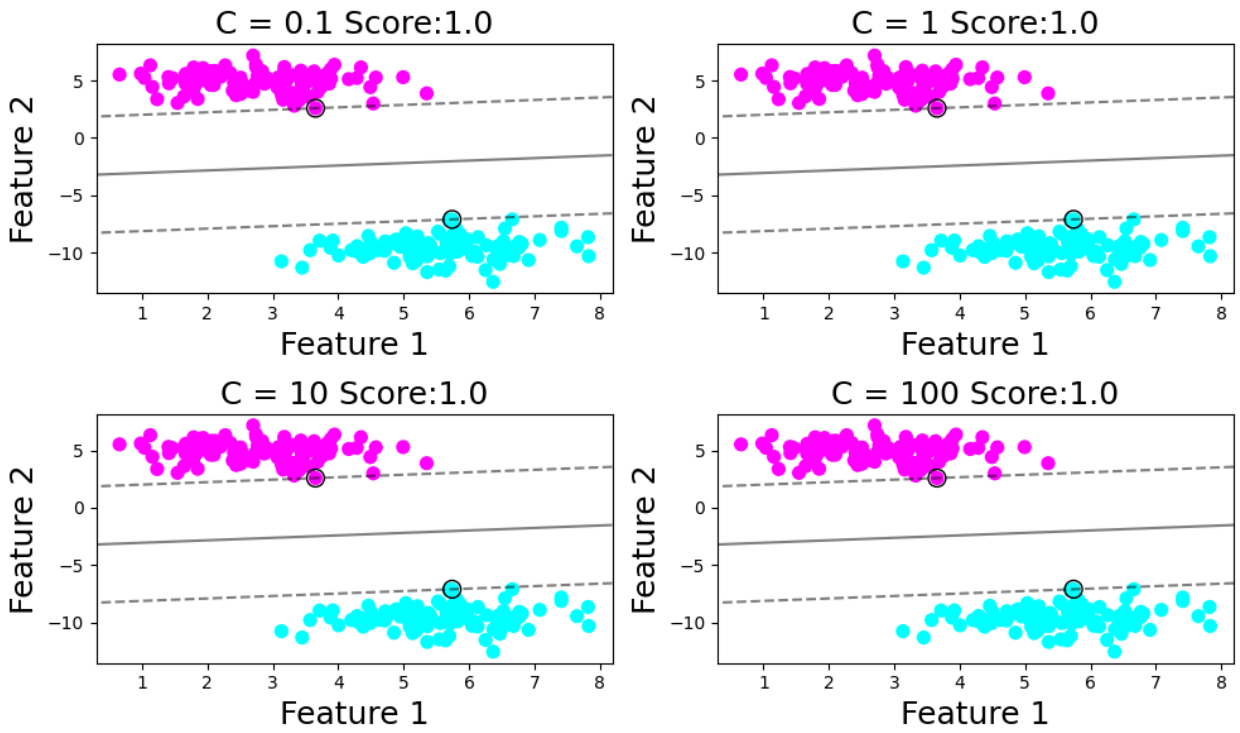

```
linear kernel accuracy :  0.95
rbf kernel accuracy :  0.99
sigmoid kernel accuracy :  0.52
poly kernel accuracy :  0.95
```



```python
def generate_data(num_samples, seed):
    X, y = make_blobs(
        n_samples=num_samples, n_features=2, centers=2, random_state=seed
    )
    return X, y

def train_svm_model(C, X, y):
    model = svm.SVC(kernel='linear', C=C)
    model.fit(X, y)
    return model

def plot_data(X, y):
    plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='cool')

def plot_decision_boundary(model):
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.decision_function(xy).reshape(XX.shape)
    plt.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

def plot_support_vectors(model):
    plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=100, linewidth=1,
facecolors='none', edgecolors='k')

def plot_svm_result(C, X, y):
    model = train_svm_model(C, X, y)
    plot_data(X, y)
    plot_decision_boundary(model)
    plot_support_vectors(model)
    plt.xlabel("Feature 1", fontsize=18)
    plt.ylabel("Feature 2", fontsize=18)
    plt.title("C = " + str(C) + " Score:" + str(model.score(X, y)), fontsize=18)


X, y = generate_data(num_samples=200, seed=10)

plt.figure(figsize=(10, 6))
for i, C in enumerate([0.1, 1, 10, 100]):
    plt.subplot(2, 2, i + 1)
    plot_svm_result(C, X, y)

plt.tight_layout()
plt.show()
```

در این قسمت تعدادی تابع جدید تعریف می کنیم تا داده های جدید را تولید، مدل svm را train و نتایج را رسم کنیم. در این قسمت support vector ها نیز رسم می شوند. Train کردن مدل svm با استفاده از یک linear kernel، پارامتر regularization تحت عنوان c، داده ورودی x و برچسب های y انجام می شود.

همچنین در این قسمت یک boundary نیز داریم که توسط تابع plot_decision_boundary ترسیم می شود.



در قسمت بعد با استفاده از GridSearchCV بهترین پارامتر برای SVM model را پیدا می کنیم:

- ابتدا با استفاده از تابع generate_linearly_separable_data با دو کلاس 0 و 1 تعدادی داده تولید می کنیم.
- داده های تولید شده را train می کنیم.
- در تابع calculate_best_params، یک grid از پارامترها به عنوان ورودی دریافت می شود. ابتدا یک SVM classifier یا clf بدون پارامتر تولید می شود. سپس یک GridSearchCV ساخته می شود که clf و grid پارامترها را به عنوان ورودی دریافت می کند. پارامتر cv تعداد foldهای cross-validation را مشخص می کند (در اینجا 3). سپس GridSearchCV، fit می شود.

بخش 2:

```
import tensorflow as tf
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm
import random
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.metrics import
accuracy_score
import seaborn as sns
```

ابتدا کتابخانه های لازم را import می کنیم.

```
feature_vector_length = 784
num_classes = 10

# Load the data
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_test = np.concatenate( [X_test , X_train[:20000]])
Y_test = np.concatenate([Y_test , Y_train[:20000] ])
X_train = X_train[20000:]
Y_train = Y_train[20000:]


# Convert into greyscale
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# Reshape the data - MLPs do not understand such things as '2D'.
# Reshape to 28 x 28 pixels = 784 features
X_train = X_train.reshape(X_train.shape[0],
X_test = X_test.reshape(X_test.shape[0], feature_vector_length)/255.0


print( X_train.shape , Y_train.shape)
```

دیتاست MNIST را با استفاده از (mnist.load_data، load کرده و آن را به داده های train و test تقسیم می کنیم.

دیتا را به grayscale تبدیل می کنیم و مقادیر پیکسل ها را نرمال سازی می کنیم. سپس دیتا را reshape می کنیم تا مطابق با ورودی مورد انتظار مدل باشد.

```
def show_imgs(x_test, decoded_imgs=None, n=10,s=28):
    plt.figure(figsize=(20, 4))
    for i in range(n):
        ax = plt.subplot(2, n, i+1)
        plt.imshow(x_test[i].reshape(s,s))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        if decoded_imgs is not None:
            ax = plt.subplot(2, n, i+ 1 +n)
            plt.imshow(decoded_imgs[i].reshape(s,s))
            plt.gray()
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)
    plt.show()
```

تابعی تحت عنوان show_imgs تعریف می کنیم و به کمک آن تصاویر دیتاست را نمایش می دهیم.

```
combined = list(zip(X_train , Y_train))
random.shuffle(combined)

X_train[:], Y_train[:] = zip(*combined)

clf = svm.SVC(kernel='rbf')
clf.fit(X_train, Y_train)

ypreds = clf.predict(X_test)
print("accuracy in test set : " , accuracy_score( Y_test , ypreds
))
```

دیتاست را به صورت تصادفی shuffle می کنیم. یک svm classifier یا clf با radial basis function (rbf) به عنوان kernel ساخته و دیتای train را fit می کنیم.

روی دیتای تست پیش بینی انجام می دهیم و دقت را با استفاده از accuracy_score می سنجیم.

```
show_imgs(X_test[10:20])
print("prediction of test set ")
print(ypreds[10 : 20 ])
```

```
ypreds_train = clf.predict(X_train[:2000])
print("accuracy in train set : " , accuracy_score( Y_train[:2000] , ypreds_train
ypreds_train
```

زیرمجموعه ای از داده های تست را با پیش بینی آن نمایش می دهیم.

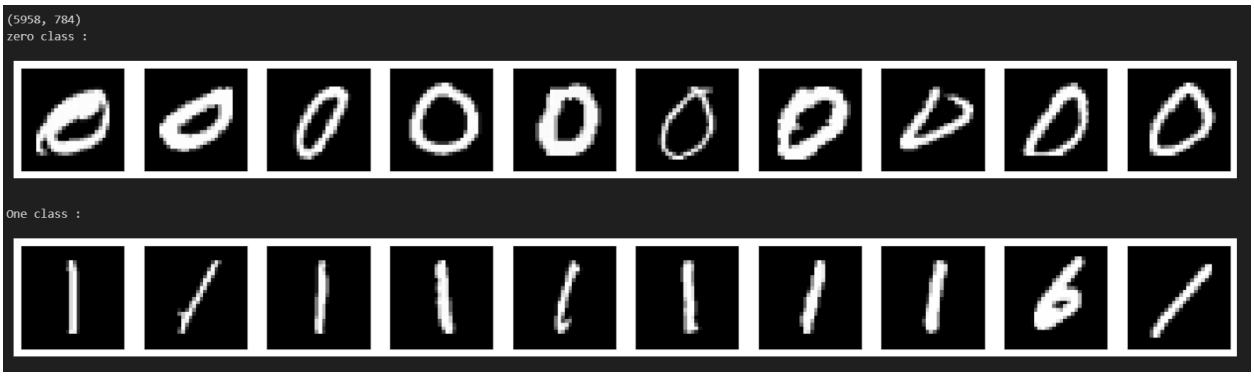
نتیجه:

```
accuracy in train set :  0.9525

array([2, 3, 7, ..., 5, 2, 9], dtype=uint8)
```

```python
def MakeDict( X , Y):
    Dict = {}
    for i in range(10):
        Dict[i] = []

    for i in range(X.shape[0]):
        Dict[Y[i]].append(X[i])

    for k in Dict.keys():
        Dict[k] = np.array(Dict[k])

    return Dict

image_data = MakeDict( X_train , Y_train )
print(image_data[2].shape)
print("zero class :")
show_imgs(image_data[0][10:20])

print("One class :")
show_imgs(image_data[1][10:20])
```

یک دیکشنری می سازیم و دیتای ورودی را بر اساس برچسب کلاس هایشان در این دیکشنری قرار می دهیم.

نتیجه:



بخش سوم:

```
def list_images(basePath, contains=None):
    return list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"),
contains=contains)
def list_files(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp"), contains=None):
    paths = []
    for (rootDir, dirNames, filenames) in os.walk(basePath):
        for filename in filenames:
            if contains is not None and filename.find(contains) == -1:
                continue
            ext = filename[filename.rfind("."):].lower()
            if ext.endswith(validExts):
                imagePath = os.path.join(rootDir , filename)
                paths.append( imagePath)
    return paths

Dict = {"cat" : 0 , "flower" :1}
```

```
def load_images(directory='', size=(64,64)):
    images = []
    labels = []
    label = 0
    imagePaths = list(list_images(directory))
    for path in imagePaths:
        if not('OSX' in path):
            image = Image.open(path)
            image_array = np.array(image)
            file_name = os.path.splitext(os.path.basename(path))[0]
            category = file_name.split()[0]
            labels.append(Dict[category])
            images.append(image_array)
    return images,labels

def rgb2grayscale(images) :
  gray_scale_images = []
  for rgb_image in images :
      pil_image = Image.fromarray(rgb_image)
      gray_image = pil_image.convert("L")
      gray_image_array = np.array(gray_image)
      gray_scale_images.append(gray_image_array)

  return gray_scale_images

images, labels = load_images('images')
gray_images = rgb2grayscale(images=images)
```

پس از import کردن کتابخانه های لازم، توابع مربوط به load کردن تصاویر را می نویسیم. پس از load کردن تصاویر با استفاده از load_images آنها را به grayscale تبدیل می کنیم.

```python
target_size = (64 , 64 )
resized_images = [resize(image, target_size) for image in gray_images]
array_of_images = np.array(resized_images)
print(array_of_images.shape)
plt.imshow( array_of_images[22000])
```

اندازه تصاویر را به 64 در 64 پیکسل تغییر می دهیم.

```python
def show_imgs(x_test, decoded_imgs=None, n=10,s=28):
    plt.figure(figsize=(20, 4))
    for i in range(n):
        ax = plt.subplot(2, n, i+1)
        plt.imshow(x_test[i].reshape(s,s))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        if decoded_imgs is not None:
            ax = plt.subplot(2, n, i+ 1 +n)
            plt.imshow(decoded_imgs[i].reshape(s,s))
            plt.gray()
            ax.get_xaxis().set_visible(False)
            ax.get_yaxis().set_visible(False)
    plt.show()
```

از تابع show_image برای نمایش تصاویر استفاده می کنیم.

```
combined = list(zip(array_of_images ,labels))
random.shuffle(combined)
array_of_images[:],labels[:] = zip(*combined)
feature_vector_length = 4096
X_train, X_test, Y_train, Y_test = train_test_split(array_of_images,labels, test_size=0.3 ,
random_state=42)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# # Reshape the data - MLPs do not understand such things as '2D'.
# # Reshape to 28 x 28 pixels = 784 features
X_train = X_train.reshape(X_train.shape[0], feature_vector_length)
X_test = X_test.reshape(X_test.shape[0], feature_vector_length)
```

```
clf = svm.SVC(kernel='rbf')
clf.fit(X_train[:6000], Y_train[:6000])
ypreds = clf.predict(X_test[:3000])
print("accuracy in test set : " , accuracy_score( Y_test[:3000] , ypreds ))
```

تصاویر را به صورت تصادفی shuffle کرده و به داده های train و test تقسیم می کنیم.

سپس با استفاده از rbf kernel در svm آنها را train می کنیم.

دقت:

```
accuracy in test set :  0.5
```

نتایج:

```
prediction of test set
                    cat cat flower flower flower cat cat
```