+ Code + Text

Lab 8 - Support Vector Machines Handwritten Digit Recognition Dataset

Author: Krishna Swaroop

181CO125, NITK Surathkal

Introduction

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression problems. It performs classification by finding the hyperplane that maximizes the margin between the two classes.

Dataset

MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

Support Vector Machines

1) Import Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt, matplotlib.image as mpimg
import time
import warnings
from sklearn import svm
from sklearn.model_selection import GridSearchCV

%matplotlib inline
```

2) Load and Visualise data

```
data_path = '/content/drive/MyDrive/Colab Notebooks/ML-Lab/SVM/dataset.csv'
```

```
data = pd.read_csv(data_path)
```

```
print("Train Data Shape is: ",data.shape)
data.head()
```

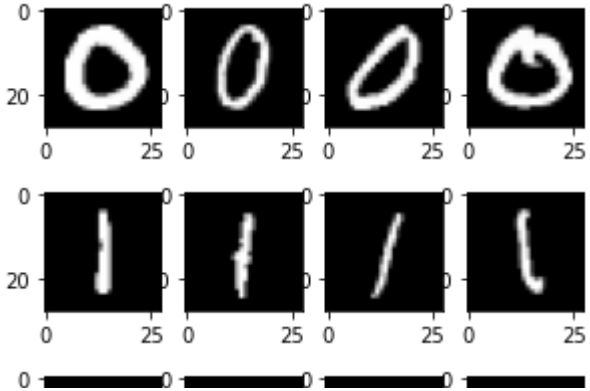
```
Train Data Shape is:  (42000, 785)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pix
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	

```
label = data.label
data=data.drop('label',axis=1)
print("Data Shape: ",data.shape)
print("Label Shape: ",label.shape)
```

```
Data Shape:  (42000, 784)
Label Shape:  (42000,)
```

```
for x in range(0,4):
    train_0=data[label==x]
    data_new=[]
    for idx in train_0.index:
        val=train_0.loc[idx].values.reshape(28,28)
        data_new.append(val)
    plt.figure(figsize=(25,25))
    for x in range(1,5):
        ax1=plt.subplot(1, 20, x)
        ax1.imshow(data_new[x],cmap='gray')
```



3) Split data

Use `train_test_split()` to split the data to training and testing dataset. Here, 20% of the dataset is reserved to test our algorithm

```
train, test,train_labels, test_labels = train_test_split(data, label, train_size=0.8, random_state=42)
print("Train Data Shape: ",train.shape)
print("Train Label Shape: ",train_labels.shape)
print("Test Data Shape: ",test.shape)
print("Test Label Shape: ",test_labels.shape)
```

```
Train Data Shape:  (33600, 784)
Train Label Shape:  (33600,)
Test Data Shape:  (8400, 784)
Test Label Shape:  (8400,)
```

4) Fit the model

We will fit the model using Gamma as 0.001 and C (Soft margin cost) as 10

```
clf = svm.SVC(C = 10, gamma = 0.001, random_state=42)
```

Since the number of features is very large, we will use PCA to lower the feature dimension space

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA as sklearnPCA

#standardized data
sc = StandardScaler().fit(train)
X_std_train = sc.transform(train)
X_std_test = sc.transform(test)

#If n_components is not set then all components are stored
sklearn_pca = sklearnPCA().fit(X_std_train)
train_pca = sklearn_pca.transform(X_std_train)
test_pca = sklearn_pca.transform(X_std_test)

#Percentage of variance explained by each of the selected components.
#If n_components is not set then all components are stored and the sum of the ratios is equal to 1.0.
var_per = sklearn_pca.explained_variance_ratio_
cum_var_per = sklearn_pca.explained_variance_ratio_.cumsum()



n_comp=len(cum_var_per[cum_var_per <= 0.90])
print("Keeping 90% Info with ",n_comp," components")
sklearn_pca = sklearnPCA(n_components=n_comp)
train_pca = sklearn_pca.fit_transform(X_std_train)
test_pca = sklearn_pca.transform(X_std_test)
print("Shape before PCA for Train: ",X_std_train.shape)
print("Shape after PCA for Train: ",train_pca.shape)
print("Shape before PCA for Test: ",X_std_test.shape)
print("Shape after PCA for Test: ",test_pca.shape)
```

```
Keeping 90% Info with  222  components
Shape before PCA for Train:  (33600, 784)
Shape after PCA for Train:  (33600, 222)
```

```
Shape before PCA for Test:  (8400, 784)
Shape after PCA for Test:  (8400, 222)
```

5) Accuracy

```
clf.fit(train_pca, train_labels)
score=clf.score(test_pca,test_labels)
print("Accuracy of Model: ",score)
```

```
Accuracy of Model:  0.969047619047619
```

6) Analysis

Therefore the best accuracy of the model achieved is 96.9%

Some of the predicted images are: