GR5241_Project1_Lili_Tan

April 5, 2022

[1]:
```
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd

sns.set_style('darkgrid')
%matplotlib inline

# Data download and preprocessing
DOWNLOAD_MNIST = True # If already download, set as False
train_data = torchvision.datasets.MNIST(
    root = './mnist/',
    train = True, # this is training data
    #transform=torchvision.transforms.ToTensor(),
    download = DOWNLOAD_MNIST ,
)
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

# change the features to numpy
X_train = train_data.train_data.numpy()
X_test = test_data.test_data.numpy()

# change the labels to numpy
Y_train = train_data.train_labels.numpy()
Y_test = test_data.test_labels.numpy()
```

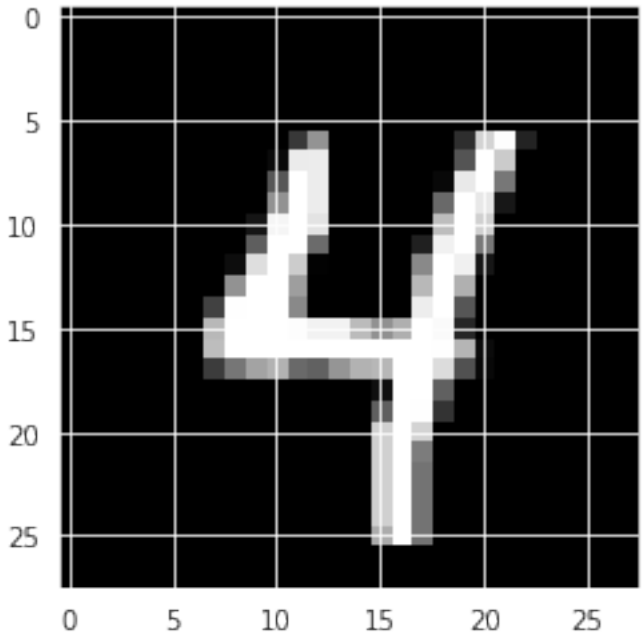
```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./mnist/MNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting ./mnist/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./mnist/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/28881 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz

  0%|          | 0/1648877 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist/MNIST/raw


/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:62:
UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:67:
UserWarning: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:52:
UserWarning: train_labels has been renamed targets
  warnings.warn("train_labels has been renamed targets")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:57:
UserWarning: test_labels has been renamed targets
  warnings.warn("test_labels has been renamed targets")
```

[2]:
```python
#Q1.(a)
import random
i = random.randint(0, X_train.shape[0])
# change i to a fix number or fix a seed to get fix result everytime
plt.imshow(X_train[i], cmap = plt.get_cmap('gray'))
plt.show()
print(f'Y_train is {Y_train[i]}')
print('It matches the label in Y_train')
```




```
Y_train is 4
It matches the label in Y_train
```

[3]:
```
#Q1.(b)
#dimension of X_train
print(f'X_train dimension: {X_train.shape}')

#dimension of X test
print(f'X_test dimension: {X_test.shape}')

#normalize X_train
norm = np.linalg.norm(X_train)
X_train_norm = X_train/norm

#normalize X_test
norm = np.linalg.norm(X_test)
X_test_norm = X_test/norm
```

```
X_train dimension: (60000, 28, 28)
X_test dimension: (10000, 28, 28)
```

[4]:
```
#Q1.(c)
#one-hot embedding Y train
Y_train_onehot = np.array(pd.get_dummies(Y_train))
#one-hot embedding Y test
Y_test_onehot = np.array(pd.get_dummies(Y_test))
```

Q1(c) benefit of such transformation: By one-hot embedding method, we vectorize the value in Y_test and Y_train. It is simple and fast to create and update the vectorization, just add a new entry in the vector with a one for each new category which means it can be rescaled easily. Moreover, the one-hot implementation is known for being the fastest one, allowing a state machine to run at a faster clock rate than any other encoding of that state machine.

[5]:
```
#Q2.(a)
#reshape x_test and x_train
nsamples, nx, ny = X_train.shape
X_train_reshape = X_train.reshape((nsamples,nx*ny))
nsamples, nx, ny = X_test.shape
X_test_reshape = X_test.reshape((nsamples,nx*ny))
```

[6]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=27).fit(X_train_reshape,Y_train_onehot)
print(f'knn test set error: {(1-knn.score(X_test_reshape,Y_test_onehot))*100:0.
 ↪2f}%')
```

```
knn test set error: 4.80%
```

[7]:
```
#AdaBoost
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=30), random_state=0).
 ↪fit(X_train_reshape,Y_train)
print(f'AdaBoost test set error: {(1-ada.score(X_test_reshape,Y_test))*100:0.
 ↪2f}%')
```

```
AdaBoost test set error: 3.19%
```

[8]:
```
#SVM
from sklearn import svm
clf = svm.SVC(C=4, kernel='rbf',random_state=123)
clf.fit(X_train_reshape, Y_train)

print(f'SVM with Gaussian Kernel test set error: {(1-clf.score(X_test_reshape,␣
 ↪Y_test))*100:0.2f}%')
```

```
SVM with Gaussian Kernel test set error: 1.57%
```

[9]:
```
#Q2.(b)
#Pick favorite classifier
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
#train the model
model.fit(X_train, Y_train_onehot, validation_data=(X_test, Y_test_onehot),␣
 ↪epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 205s 109ms/step - loss: 0.2237 -
accuracy: 0.9492 - val_loss: 0.0877 - val_accuracy: 0.9720
Epoch 2/3
1875/1875 [==============================] - 203s 108ms/step - loss: 0.0701 -
accuracy: 0.9784 - val_loss: 0.0810 - val_accuracy: 0.9774
Epoch 3/3
1875/1875 [==============================] - 204s 109ms/step - loss: 0.0461 -
accuracy: 0.9858 - val_loss: 0.1033 - val_accuracy: 0.9729
```

[9]: <keras.callbacks.History at 0x7f30a1f9ae50>

Accuracy for CNN method is 98.58%