Digits_v2

March 6, 2021

0.1 MNIST Digits Computer Vision Projekt

0.1.1 Author: Georg Zhelev

In this projekt a support vector classifier (SVC) and a fully connected neural network (MLP) are applied on the MNIST Digits dataset with the goal of predicting written digits. Each image is represented by a matrix with elements representing pixels made up of gray scale values. The closer the value 0 the more white the pixel, the closer to 255 the more black the pixel. This format allows for a machine learning model to process the image information.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 25 12:48:01 2021

@author: zhele
"""

import pandas as pd
import matplotlib.pyplot as plt, matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
%matplotlib inline
```

[ ]:
```
# Load the data
train = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Econ/Digits/train.
 ↪csv")
test = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Econ/Digits/test.
 ↪csv")
```

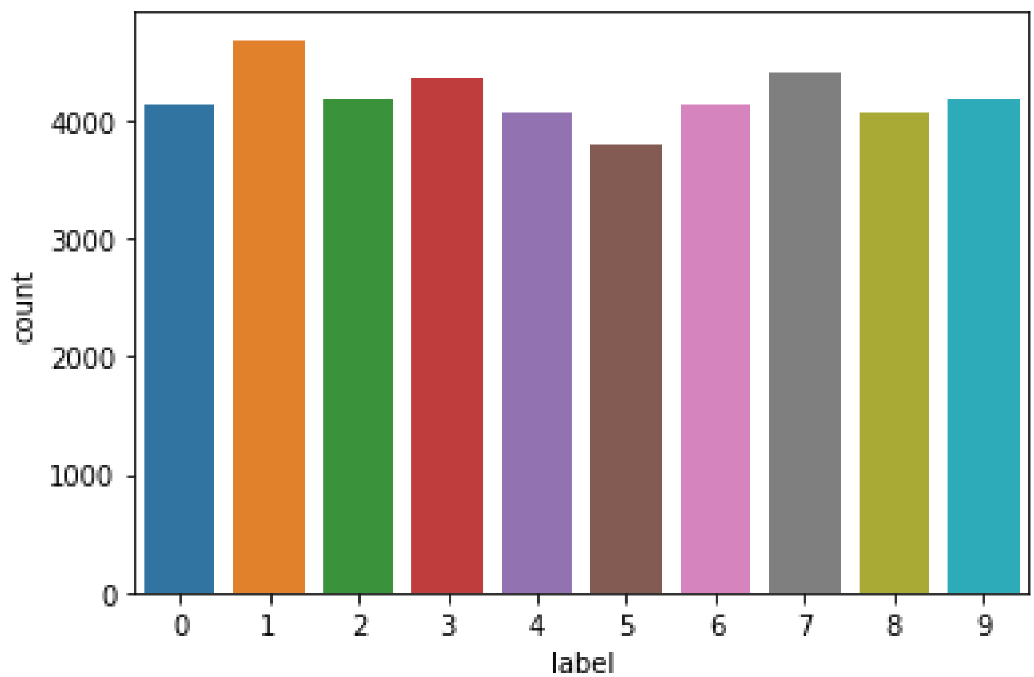
[ ]:
```
#The training data set, (train.csv), has 785 columns. The first column,
#called "label", is the digit that was drawn by the user. The rest of the
```

```
#columns contain the pixel-values of the associated image.

#Each pixel column in the training set has a name like pixelx, where x is
#an integer between 0 and 783, inclusive. To locate this pixel on the image,
#suppose that we have decomposed x as x = i * 28 + j, where i and j are
#integers between 0 and 27, inclusive. Then pixelx is located on row i and
#column j of a 28 x 28 matrix, (indexing by zero).

#For example, pixel31 indicates the pixel that is in the fourth column from
#the left, and the second row from the top, as in the ascii-diagram below.

#000 001 002 003 ... 026 027
#028 029 030 031 ... 054 055
#056 057 058 059 ... 082 083
# |   |   |   |  ...  |   |
#728 729 730 731 ... 754 755
#756 757 758 759 ... 782 783
```

[ ]:
```
Y_train = train["label"]

# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

import seaborn as sns
g = sns.countplot(Y_train)

Y_train.value_counts()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

[ ]:
```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```




```
[ ]: Y_train.shape
```

```
[ ]: (42000,)
```

```
[ ]: X_train.shape
```

```
[ ]: (42000, 784)
```

```
[ ]: test.shape
```

```
[ ]: (28000, 784)
```

```
[ ]: Y_train.value_counts().sum()
```

```
[ ]: 42000
```

```
[ ]: # Check the data
     X_train.isnull().any().describe()
```

```
[ ]: count       784
     unique        1
     top       False
     freq        784
     dtype: object
```

[ ]:
```
test.isnull().any().describe()
```

[ ]:
```
count       784
unique        1
top       False
freq        784
dtype: object
```

I check for corrupted images (missing values inside).

There is no missing values in the train and test dataset. So we can safely go ahead.

###MLP: Reduce Sample

[ ]:
```
#take first 5000 obs. with all features after the first column
images = train.iloc[0:5000,1:]

#take first 5000 obs. of all features until the second column
labels = train.iloc[0:5000,:1]
```

[ ]:
```
# free some space
#del train
```

###MLP: Split

[ ]:
```
#Split

train_images, test_images,train_labels, test_labels = train_test_split(images,␣
 ↪labels, train_size=0.8, random_state=0)
```

[ ]:
```
#Observe Data

#notice that the image features is flattened into a single row 28*28=784
train_images.shape
train_labels.shape
test_images.shape
test_labels.shape
```

[ ]:
```
(1000, 1)
```

###MLP: Reshape

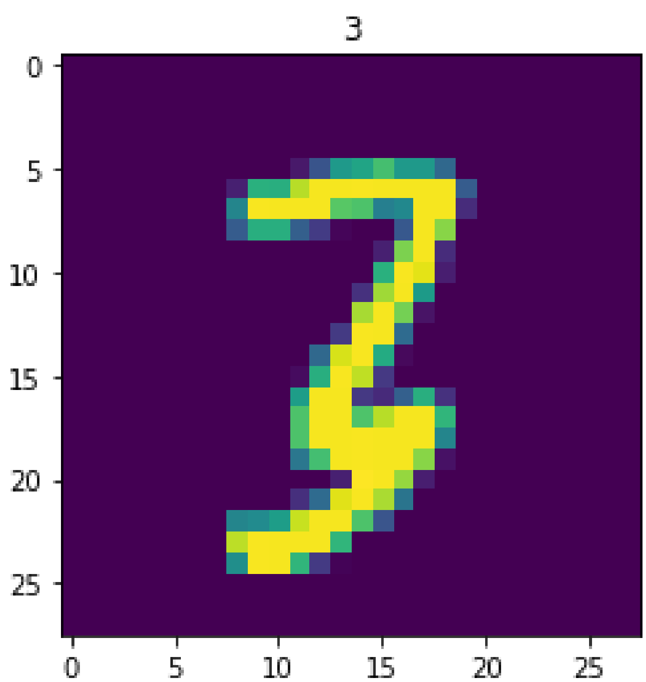
[ ]:
```
#choose a single observation i from data set
i=2
#convert to np array
img=train_images.iloc[i].to_numpy()
#reshape it to a two dimentional 28x28 so it can be viewed by a naked eye
img=img.reshape((28,28))
```

[ ]:
```
#single observation, see variable explorer
train_images.iloc[i]
```

[ ]:
```
pixel0      0
pixel1      0
pixel2      0
pixel3      0
pixel4      0
           ..
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Name: 775, Length: 784, dtype: int64
```

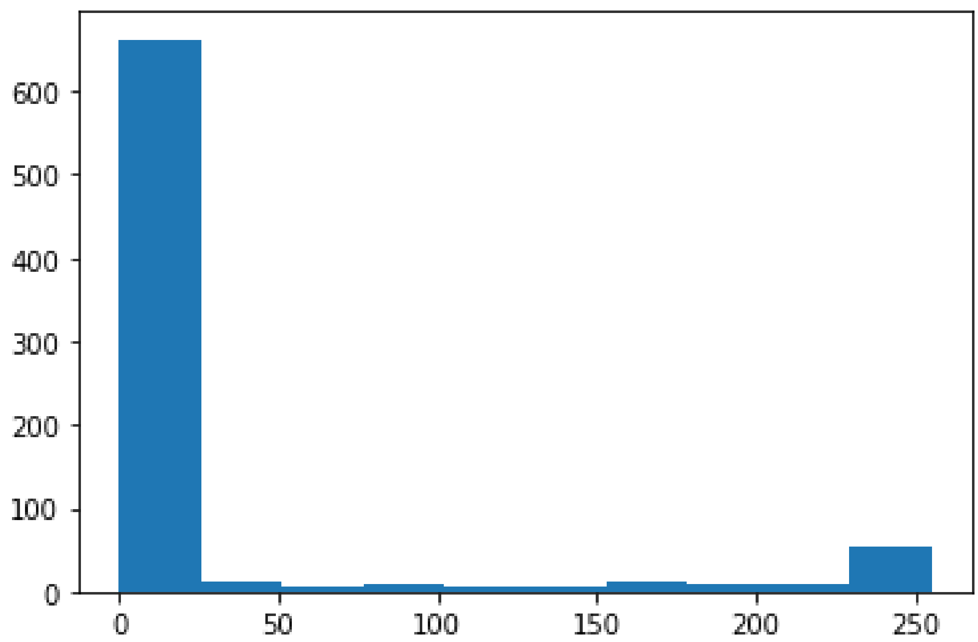
[ ]:
```
#plot features
plt.imshow(img)
#plot label
plt.title(train_labels.iloc[i,0])
```

[ ]:
```
Text(0.5, 1.0, '3')
```



```
[ ]: #histogram of the image pixel values
     #it is a gray scale values (from 0 = white to 255=black and everything in␣
      ↪between)
     plt.hist(train_images.iloc[i])
     #many pixel values are around 0 or low close to 0
     #some are the maximal 255
     #the pixels where the number is drawn are the darkest
```

```
[ ]: (array([662.,  11.,   5.,   8.,   7.,   7.,  11.,   9.,   9.,  55.]),
      array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
             229.5, 255. ]),
      <a list of 10 Patch objects>)
```



0.1.2 MLP:Scale

```
[ ]: #Pre-Process
     #scaling will remove possibility to observe image by viewing the data
     test_images = preprocessing.scale(test_images)
     train_images = preprocessing.scale(train_images)
```

###Train SVC

```
[ ]: #Train SVC Model
     clf = svm.SVC(C=7, gamma=0.009)
     #ravel() flattens an array (the ,1 column is removed, see variabel explorer)
     #.values  displays a list of all values in a given dictionary.
```

```
clf.fit(train_images, train_labels.values.ravel())
clf.score(test_images,test_labels)
#svc scores very well, but that is actually overfitting

#Explanations of commands
#hast no index
#ravel.shape
#has index
#train_labels.shape
```

[ ]: 0.743

Above is the accuracy on the train test (for test set accuracy see kaggle results below)

###Train MLP

[ ]:
```
#Train MLP Model
clf2 = MLPClassifier()
clf2.fit(train_images, train_labels.values.ravel())
clf2.score(test_images,test_labels)

#switch to binary from gray scale: any pixel with a value simply
#becomes 1 and everything else remains 0.
#test_data[test_data>0]=1
```

[ ]: 0.922

Above is the accuracy on the train test (for test set accuracy see kaggle results below)

0.1.3 Make Predictions

[ ]:
```
#Make Predictions

def pred(classifier):
    #predict just the first 5000 entries (because its quicker)
    results=classifier.predict(test_data[0:5000])
    return results

def pred_full(classifier):
    #predict all obs.
    results=classifier.predict(test)
    return results

#results_svc = pred(clf)
results_scv = pred_full(clf)

results_mlp = pred_full(clf2)
```

[ ]:
```python
from google.colab import files

#Save Predictions for Submission
def save(results, name):
    #convert to pd data frame
    df = pd.DataFrame(results)
    #rename index column
    df.index.name='ImageId'
    #start index from 1 instead of 0
    df.index+=1
    #name second column
    df.columns=['Label']
    df.to_csv(f"{name}.csv", header=True)
    files.download(f"{name}.csv")
    return

save(results_mlp, "results_mlp")
save(results_scv, "results_svc")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[ ]:
```python
#Kaggle Prediction Competiton Results

#1: SVC prediction accuracy on the test set is 20% without any pre-processing
#2: SVC with scalling 24,7%
#3: with MLP 81%
#4: with CNN 91% (separate notebook)
```