```
1: ////////////////////////////////////////////
2: maXbox Starter 83 MNIST Single Prediction
3: ////////////////////////////////////////////
4:
5: Machine Learning XI_____________________________
6:
7:
8: For this tutor we'll explore one of the classic machine learning datasets - hand
   written digits classification.
9:
10: https://github.com/maxkleiner/maXbox4/blob/master/MNISTSinglePredict.ipynb
11:
12: Task with matrix solution:
13:
14: https://github.com/maxkleiner/maXbox4/blob/master/MNISTSinglePredict2Test.ipynb
15:
16: We have set up a very simple SVC to classify the MNIST digits to make one single
   predict. First we load the libraries and the dataset:
17:
18: #sign:max: MAXBOX8: 13/03/2021 07:46:37
19: import numpy as np
20: import pandas as pd
21: import matplotlib.pyplot as plt
22: from sklearn import tree
23: from sklearn.svm import SVC
24: from sklearn.ensemble import RandomForestClassifier
25: from sklearn import datasets
26: from sklearn.metrics import accuracy_score
27: from sklearn.model_selection import train_test_split
28:
29: The dataset is available either for download from the UCI ML repository or via a
   Python library scikit-learn dataset.
30:
31: # [height, weight, 8*8 pixels of digits 0..9]
32: dimages = datasets.load_digits()
33: print(type(dimages), len(dimages.data))
34:
35: <class 'sklearn.utils.Bunch'> 1797 samples
36:
37: The dataset consists of a table - columns are attributes, rows are instances
   (individual observations). In order to do computations easily and efficiently and
   not to reinvent wheel we can use a suitable tool - pandas. So the first step is
   to obtain the dataset and load it in a second step for train and test-split into
   a DataFrame.
38:
39: Then we setup the Support Vector Classifier with the training data X and the
   target y:
40:
41: #Support Vector Classifier
42: sclf = SVC(gamma=0.001, C=100, kernel='linear')
43:
44: X= dimages.data[:-10]
45: y= dimages.target[:-10]
46: print(len(X))
47:
48: train set samples: 1787
49:
50:
51: Gamma is the learning rate and the higher the value of gamma the more precise the
   decision boundary would be. C (regularization) is the penalty of the fault
   tolerance.
```

Having a larger C will lead to smaller values for the slack variables. This means that the number of support vectors will decrease. When you run the prediction, it will need to calculate the indicator function for each support vector. Now we train (fit) the samples:

```
sclf.fit(X,y)
print('ONLY train score ',sclf.score(X,y))

SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In the last step we predict a specific digit from the test set (only the last 10 samples are unseen), means we pass an actual image and SVC makes the prediction of which digit belongs to the image:

```
testimage = -6

s_prediction = sclf.predict([dimages.data[testimage]])
print ('the image maybe belongs to ',s_prediction)
plt.imshow(dimages.images[testimage],cmap=plt.cm.gray_r,interpolation="nearest")
plt.show()

the image maybe belongs to  [4]

ASUVORK5CYII=%0A
as 4 of MNIST

https://maxbox4.files.wordpress.com/2021/04/4index.png

/QZQIaIGkiFqIBmiBpIhaiAZogaSIWogmf8B0Eylpk5WooEAAAAASUVORK5CYII=%0A
```

The same fit we try with a Random Forest Classifier to finish the first step of this lesson:

```
#RandomForestClassifier
rfc_clf = RandomForestClassifier()
rfc_clf.fit(X,y)
rfc_prediction = rfc_clf.predict([predictimage_test])
print ('predict with RFC ',rfc_prediction)
```

There are many ways to improve this predict, including not using a vector classifier and go further with a neural classifier, but here's a simple one to start what we do. Let's just simplify our images by making them true black and white and stack an array.

```
https://maxbox4.files.wordpress.com/2021/04/athenauml.gif?w=125&zoom=2
UML Unified Machine Learning.


-------------------------------------------------
MNIST Multi Prediction
-------------------------------------------------
```

Now we split explicit data in train- and test-set. Splitting the given images in 80:20 ratio so that 80% image is available for training and 20 % image is available for testing. We consider the data as pixels and the target as labels.

```
Convert and create the dataframe from datasets. We are using support vector
machines for classification. Fit method trains the model and score will test it
against the given test set and score.

A Support Vector Machine (SVM) is a discriminative classifier formally defined by
a separating hyperplane. In other words, given labeled training data (supervised
learning), the algorithm outputs an optimal hyperplane which categorizes new
examples. In two dimensional space this hyperplane is a line dividing a plane in
two parts where in each class lay in either side.

df = pd.DataFrame(data=dimages.data, columns=dimages.feature_names)
print(df.head(5))
df['target'] = pd.Series(dimages.target)
#df['pixels'] = dimages.data[:,1:64] #pd.Series(dimages.data[:,1:785])
print(df['target'])
print(df.shape) #print(df.info)

pixels = df
labels = df.target
print('pixels ',pixels)

test_images,train_images, test_labels,train_labels = \
             train_test_split(pixels,labels,train_size=0.8,random_state=2);

print('train size: ',len(train_images), len(train_labels))

sclf.fit(train_images, train_labels)
print('test score ',sclf.score(test_images,test_labels))

This gives us the score of 97 percent (0.97633959638135) which is at all a good
score. We would try to increase the accuracy but this is sort of challenge.

The dataset description of our primer says: Each image is 8 pixels in height and
8 pixels in width, for a total of 64 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.

pixel_0_0 pixel_0_1 pixel_0_2 … pixel_7_5 pixel_7_6 pixel_7_7
0 0.0 0.0 5.0 … 0.0 0.0 0.0
1 0.0 0.0 0.0 … 10.0 0.0 0.0
2 0.0 0.0 0.0 … 16.0 9.0 0.0
3 0.0 0.0 7.0 … 9.0 0.0 0.0
4 0.0 0.0 0.0 … 4.0 0.0 0.0

[5 rows x 64 columns]
0 0
1 1
2 2
3 3
4 4
..
1792 9
1793 0
1794 8
1795 9
1796 8
Name: target, Length: 1797, dtype: int32
(1797, 65)
pixels pixel_0_0 pixel_0_1 pixel_0_2 … pixel_7_6 pixel_7_7 target
```

```
150:
151: 0 0.0 0.0 5.0 … 0.0 0.0 0
152: 1 0.0 0.0 0.0 … 0.0 0.0 1
153: 2 0.0 0.0 0.0 … 9.0 0.0 2
154: 3 0.0 0.0 7.0 … 0.0 0.0 3
155: 4 0.0 0.0 0.0 … 0.0 0.0 4
156: … … … … … … … …
157: 1792 0.0 0.0 4.0 … 0.0 0.0 9
158: 1793 0.0 0.0 6.0 … 0.0 0.0 0
159: 1794 0.0 0.0 1.0 … 0.0 0.0 8
160: 1795 0.0 0.0 2.0 … 0.0 0.0 9
161: 1796 0.0 0.0 10.0 … 1.0 0.0 8
162:
163: [1797 rows x 65 columns]
164: train size: 360 : 360
165: test score 0.97633959638135
166: predict with RFC [9]
167:
168: C:\maXbox\mX46210\DataScience>
169:
170: Would be nice to get the confusion matrix of MNIST dataset to get an impression
     of the score.
171:
172: from sklearn.metrics import confusion_matrix
173: test_predictions = sclf.predict(test_images)
174: #print(confusion_matrix(test_labels,np.argmax(test_predictions,axis=1)))
175: print(confusion_matrix(test_labels, test_predictions))
176:
177: test score 0.97633959638135
178:
179: [[146 0 0 0 0 0 0 0 0 0] 0
180: [ 0 138 0 0 0 0 0 0 0 0] 1
181: [ 1 7 137 0 0 0 1 0 0 0] 2
182: [ 0 0 0 146 0 0 0 0 1 0] 3
183: [ 0 1 0 0 145 0 0 0 0 0] 4
184: [ 0 0 0 0 1 135 1 0 0 2] 5
185: [ 2 0 0 0 0 0 144 0 0 0] 6
186: [ 0 0 0 0 0 0 0 138 0 1] 7
187: [ 0 1 0 0 1 3 1 4 126 2] 8
188: [ 0 0 0 0 0 1 0 0 3 148]] 9
189:
190: Splitting the given images in 70:30 ratio shows a slight different confusion
     matrix so that 70% image is available for training and 30 % image as unseen or
     unknown is available for testing. Number 8 has probably most problems to get
     recognized! So disguise as 8 you can be a 6 or 9 and thats logical cause the 8 is
     in a 7-segment LCD display the base pattern! In german we say that with the word
     Achtung ;-).
191:
192: train_images,test_images,train_labels,test_labels = \
193:              train_test_split(pixels,labels,train_size=0.7,random_state=2);
194:
195: sclf.fit(train_images, train_labels)
196: print('test score ',sclf.score(test_images,test_labels))
197: test_predictions = sclf.predict(test_images)
198: print(confusion_matrix(test_labels, test_predictions))
199:
200: test score  0.975925925925926
201:
202: [[54  0  0  0  0  0  0  0  0  0]
203:  [ 0 56  0  0  0  0  0  0  0  0]
```

```
 [ 0  0 54  0  0  0  0  0  0  0]
 [ 0  0  0 60  0  2  0  0  0  0]
 [ 0  0  0  0 50  0  0  1  2  0]
 [ 0  0  0  0  0 58  0  0  0  1]
 [ 0  1  0  0  0  0 55  0  0  0]
 [ 0  0  0  0  0  0  0 55  0  0]
 [ 0  2  0  0  0  0  1  0 43  1]
 [ 0  0  0  0  0  1  0  1  0 42]]


----------------------------------------
Log Regression Schema
----------------------------------------

import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
X=np.array([[10000,80000,35],[7000,120000,57],[100,23000,22],[223,18000,26]])
y=np.array([1,1,0,0])

cls=LogisticRegression(random_state=12)
cls.fit(X,y)
LogisticRegression(random_state=12)
print(cls.predict([[6500,50000,26]]))
>>> [1]
>>>

Could be a predictive maintenance task to predict next failure of the machine type! Numbers could be 6500 operation hours, 50000 items produced, 26 as age. [1] means could fail or need maintenance next time.

Note: Train and test split is mistaken (muddle up or confuse), that's the right one:
train_images, test_images, train_labels, test_labels = \
train_test_split(pixels,labels,train_size=0.8,random_state=2);

train size: 1437 1437
test size: 360 360
test score 0.9777777777777777


for Reference:

https://maxbox4.wordpress.com/2021/04/06/ml-primer/
https://maxkleiner1.medium.com/ml-primer-aa181ea1acd0

https://colab.research.google.
com/github/maxkleiner/maXbox4/blob/master/MNISTSinglePredict2Test.
ipynb#scrollTo=YAcgQEaPoMjZ

----app_template_loaded_code----
----File newtemplate.txt not exists - now saved!----
```