INFO370 Lab 9: ML workflow

March 7, 2021

Introduction

This lab asks you to play with ML modeling workflow, in particular you have to select the model that gives you the best accuracy. This task differs in many ways from the previous tasks, you will get quite a bit of code and methods as black box, and operate with those.

More specifically, the task is to recognize handwritten digits. So it is an image recognition task. We use the sklearn-provided curated and standardized 8×8 pixel digits.[1]

1 Categorize MNIST hand-written digits

1. Load the data. The digits data is built into sklearn and you can use the following code

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

##
mnist = load_digits()
# the result contains attributes .data for the design matrix
# and .target for the lables.
X = mnist.data  # design matrix X
y = mnist.target  # labels, 0-9 denoting the corresponding numbers

print("Image data:", X.shape)

## Image data: (1797, 64)

print("First 10 labels:", y[:10])

## First 10 labels: [0 1 2 3 4 5 6 7 8 9]
```

   The digits pixel data are "flattened" into rows of X. Each pixel value is coded as intensity between 0 and 16.

2. How many digits do we have in the data? How many pixels per digit do we have?

3. Plot a few digits to have an idea how do these look like. Can you recognize the digit? Check the corresponding label to see if your guess was correct.

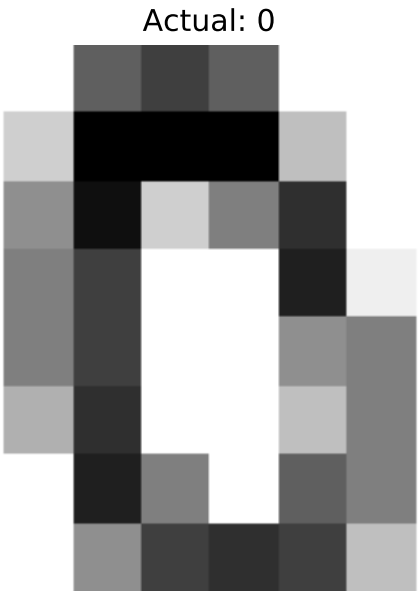
   You can plot a digit as follows:

[1] There are different version of this data, there are datasets with higher-resolution images and more observations.

```
i = 78  # plot digit #77
plt.imshow(X[i].reshape((8, 8)), cmap='gray_r')
# because these are flattened, we have to reshape it back to 8x8 image
_ = plt.axis("off")  # no need for axis annotation
plt.title(f"Actual: {y[i]}")  # show the correct label in title

## Text(0.5, 1.0, 'Actual: 0')

plt.show()
```



What do you think, can you recognize the digits easily?

Your next task is to develop the best model to categorize the digits using logistic regression. As we have 10 different categories here, we actually use a version of logistic regression (multinomial logit) that can handle more than two categories, but from coding perspective you don't see much difference.

4. Set up a logistic regression model, and 10-fold cross-validate the accuracy. How good an accuracy do you get? Can you explain in plain words what does it mean?

   Note: you may get convergence warnings. To suppress those you can set `max_iter=500` and suppress warnings with

```
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning

simplefilter("ignore", category=ConvergenceWarning)
```

5. Logistic regression has little parameters to tune. The only one that matters here a little bit is the regularization parameter `C`. Try out different values of C in range 10^{-6} to 10^{6}. So you set up your model along the lines

```
m = LogisticRegression(C=1000, multi_class="multinomial")
```

Try all regularization parameters $10^{-6}, 10^{-5}, 10^{-4}, \ldots, 10^{6}$. Which one gives you the best accuracy?

Hint: you can create such a logarithmic sequence with `np.logspace` like

```
import numpy as np

np.logspace(-6, 6, 13)

## array([1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01,
##        1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06])
```

6. Make a plot where you show how accuracy depends on C. Use log-scale on your x-axis with `plt.xscale("log")`

Solution 1

1. load data: code given
2. How many

```
X.shape

## (1797, 64)
```

Rows of X gives the number of digits (1797) cases, columns gives the pixels per digit (64).

3. Plot: code given.

I would probably guess fairly well, but probably not perfectly as the images are quite sketchy.

4. Cross-validate logistic accuracy

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

m = LogisticRegression(solver="lbfgs", multi_class="multinomial")
cv = cross_val_score(m, X, y, scoring="accuracy", cv=10)
print("CV results:", cv)

## CV results: [0.90810811 0.96174863 0.8839779  0.95555556 0.94413408 0.96648045
##  0.94413408 0.93820225 0.8700565  0.9375    ]

print(cv.mean())

## 0.9309897545814041
```

I get $A = 0.93$. So 93 digits out of 100 are put in the correct category.

5. find best C:

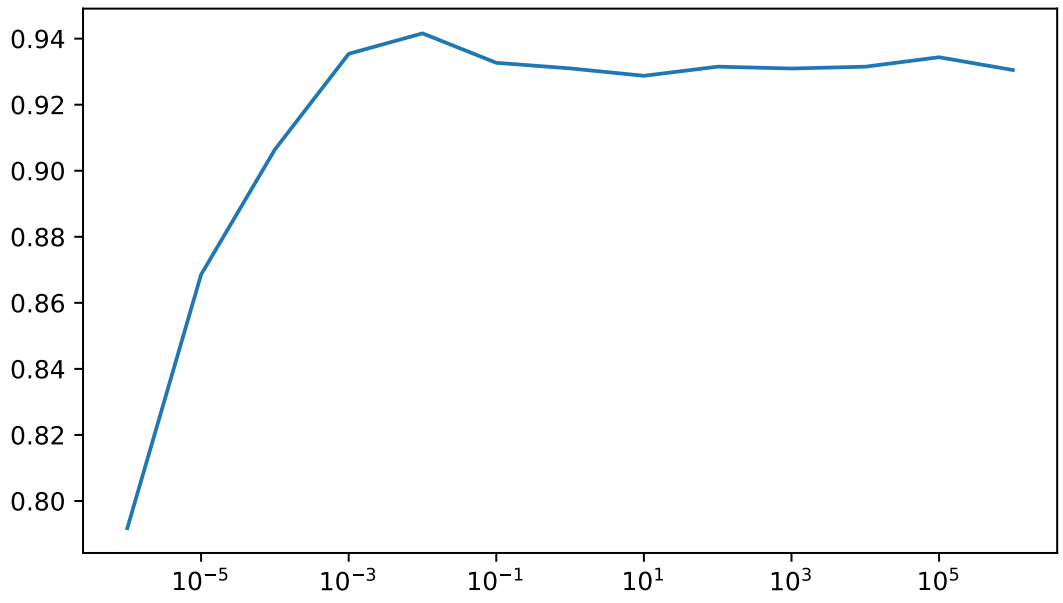
```
Cs = np.logspace(-6, 6, 13)
As = pd.Series(0.0, index=Cs)
for C in Cs:
    m = LogisticRegression(solver="lbfgs", C=C, multi_class="multinomial")
    cv = cross_val_score(m, X, y, scoring="accuracy", cv=10)
    As[C] = cv.mean()
    print(C, As[C])

## 1e-06 0.7917518531606996
## 1e-05 0.868561440547501
## 0.0001 0.9064214770068689
## 0.001 0.9354028120519775
## 0.01 0.9415964039437792
## 0.1 0.932684494868466
## 1.0 0.9309897545814041
## 10.0 0.9287489985496643
## 100.0 0.9315123096006713
## 1000.0 0.93095324539615
## 10000.0 0.9315059970671362
## 100000.0 0.9343642561952461
## 1000000.0 0.9304995258311667
```

The best result seems to be around $\mathsf{C} = 0.01$ with accuracy 0.94

6. plot

```
_ = plt.plot(Cs, As)
plt.xscale("log")
plt.show()
```



2 Challenge (not graded)

If you still have time and interest, try a different type of model. Sklearn is made in a way that it is fairly easy to switch models. Let's try k-NN. k-NN finds k "most similar" images to the given image, and categorizes the current image based on what they are. It has one central parameter, k, i.e. how many similar images to consider. It may be 1 or more. You can set up k-NN as

```
from sklearn.neighbors import KNeighborsClassifier

m = KNeighborsClassifier(1)  # consider the single most similar image
```

Rest of the code is more or less the same as for logistic regression

1. Find the best k between 1 and 30 by 10-fold cross-validating the model. What is the best accuracy? Is it better or worse than for logistic regression?
2. Make a similar plot as above where you display the accuracy as a function of k.

If you are interested, you can try even more model types, say classification trees, random forests, support vector machines and neural networks. But you have to figure out which parameters are worth of tuning.

Solution 2.0

1. k-NN:

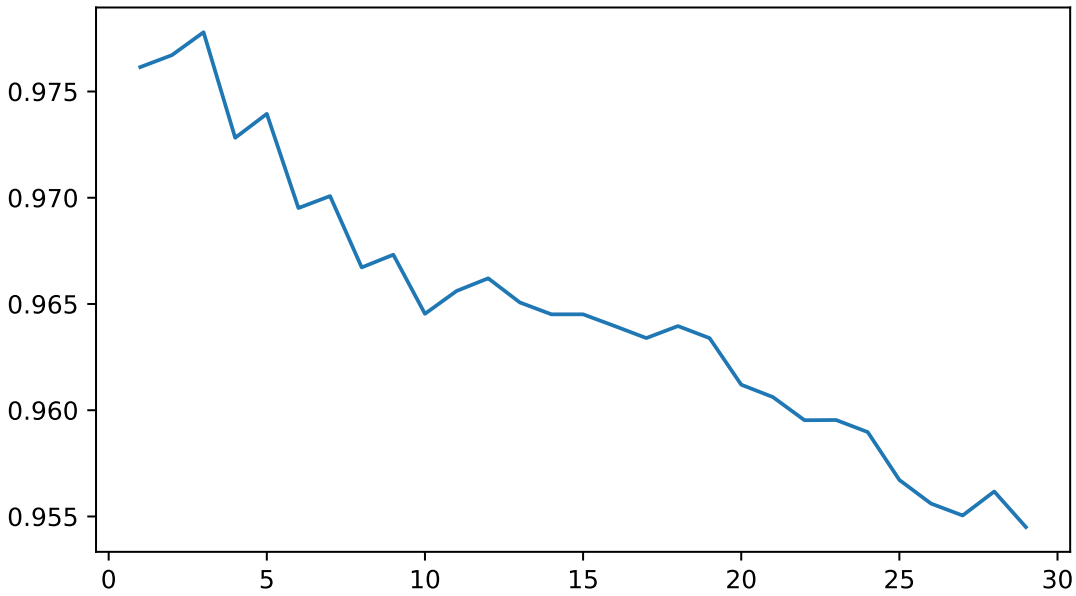
```
ks = np.arange(1, 30)
As = pd.Series(0.0, index=ks)
for k in ks:
    m = KNeighborsClassifier(k)
    cv = cross_val_score(m, X, y, scoring="accuracy", cv=10)
    print(k, cv.mean())
    As[k] = cv.mean()

## 1 0.9761493860252022
## 2 0.9767119759627271
## 3 0.9777892113798643
## 4 0.9728216572539727
## 5 0.9739482872546906
## 6 0.9695148279797406
## 7 0.9700794639117305
## 8 0.9667209856266528
## 9 0.9673191512876572
## 10 0.9645378919526033
## 11 0.9656123246807266
## 12 0.9662080401765462
## 13 0.965071784140186
## 14 0.9645127199356647
## 15 0.9645131249223089
## 16 0.9639608137016387
## 17 0.9633958419502262
## 18 0.9639608137016387
## 19 0.9633926679516229
```

```
## 20 0.9612005460957125
## 21 0.9606205250346738
## 22 0.9595304670389913
## 23 0.9595396436743368
## 24 0.9589715333879925
## 25 0.9567182664480555
## 26 0.9556071553369444
## 27 0.9550458620367263
## 28 0.9561758067351066
## 29 0.9544998290814753
```

2. plot

```
_ = plt.plot(ks, As)
plt.show()
```



The best $k = 3$, the best accuracy is 0.977. This is better than for logistic