Multi-class Logistic Regression

Contents


```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, "savefig.dpi":300})
sns.set_context("notebook")
sns.set_style("ticks")
```

Objectives

- To demonstrate multi-class logistic regression

Handwritten Digits

We will demonstrate multi-class logistic regression using a handwritten digits dataset. The data are in scikit-learn and our example follows very closely this example.

First, let's load the dataset.

```
from sklearn import datasets

digits = datasets.load_digits()

print(digits.DESCR)
```

```
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

The images are in a 3D array:

```
print(digits.images.shape)
```

```
(1797, 8, 8)
```

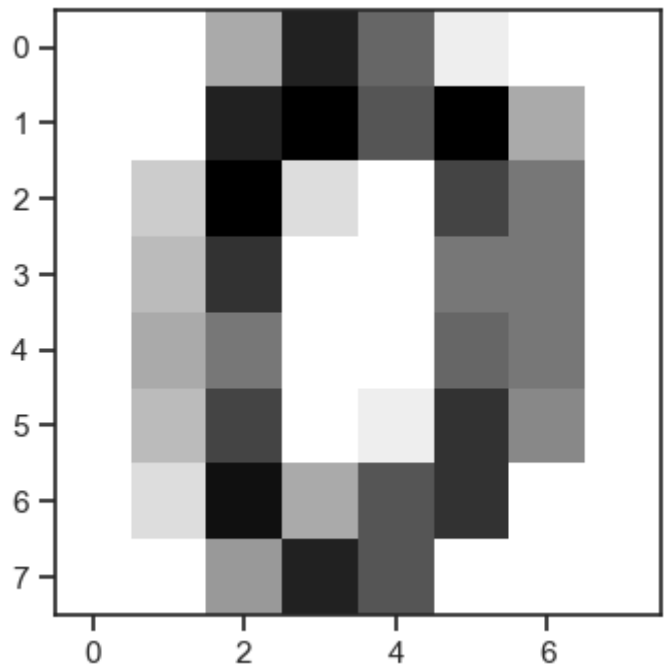
Each row of this array is an 8x8 image (which is just a matrix). Here is the first image as just numbers:

```
print(digits.images[0])
```

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

These numbers correspond to the darkness of each pixel. The greater the value the darker the pixel. Here is how we can visualie the first image:

```
fig, ax = plt.subplots()
ax.imshow(
    digits.images[0],
    cmap=plt.cm.gray_r,
    interpolation='nearest'
);
```



That's clearly a 0. Now each one of the images comes we predetermined labels that we can use to train models. Here is where you can find the labels:

```
print(digits.target)
```

```
[0 1 2 ... 8 9 8]
```

and notice that the first label is a 0, which is great. Let's now plot several images just to gain some intuition about them:

```
fig, axes = plt.subplots(4, 4)
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(
    axes.flatten(),
    images_and_labels[:16]
):
    ax.set_axis_off()
    ax.imshow(
        image,
        cmap=plt.cm.gray_r,
        interpolation='nearest'
    )
    ax.set_title(f'Training: {label}')
plt.tight_layout();
```

We are going to apply the multi-class logistic regression classifier with 64 linear features, one per pixel. This assumes that the images are vectorized. That is, we turn them from 8×8 matrices to 64-dimensional arrays. Here is how we can do this:

```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
print(data.shape)
```

```
(1797, 64)
```

Let's split the dataset into training and validation sets. We will use the functionality of scikit learn for this:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    data,
    digits.target,
    test_size=0.5,
    shuffle=True
)
```

The model we are going to fit is:

$$p(y = k|\mathbf{x}, \mathbf{W}) = \text{softmax}_k \left(\mathbf{w}_1^T\mathbf{x}, \ldots, \mathbf{w}_K^T\mathbf{x}\right),$$

where $\mathbf{x}$ is the vectorized version of the image. Let's do it:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(
    max_iter=2000,
    penalty='none',
    fit_intercept=True
)
model.fit(X_train, y_train);
```

Here is how you can get the matrix of all weights $\mathbf{W}$:

```
print(model.coef_.shape)        not the weights themselves, just the shape of W
```

```
(10, 64)
```

Here are point predictions for (picking the label with the highest probability):

```
predicted = model.predict(X_test)
print('#\tTruth\tPrediction')
print('-' * 26)
for i, (yt, yp) in enumerate(zip(y_test, predicted)):
    print(f'{i}\t{yt}\t{yp}')
```

#	Truth	Prediction
0	7	7
1	7	7
2	7	7
3	3	3
4	2	2
5	6	6
6	4	4
7	7	7
8	2	2
9	5	5
10	1	1
11	6	6
12	5	9
13	2	2
14	0	0
15	9	9
16	6	6
17	6	6
18	8	8
19	6	6
20	9	9
21	0	0
22	2	2
23	1	1
24	9	9
25	5	5
26	4	4
27	7	7
28	0	0
29	4	4
30	9	9
31	4	4
32	0	0
33	3	3
34	4	4
35	6	6
36	3	2
37	2	2
38	3	3
39	0	0
40	9	9
41	3	3
42	6	6
43	3	3
44	7	7
45	4	4
46	4	4
47	5	5
48	5	5
49	1	1
50	8	8
51	3	3
52	8	6
53	4	4
54	9	5
55	8	8
56	5	5
57	9	9
58	4	4
59	4	4
60	6	6
61	8	1
62	9	9
63	1	1
64	6	6
65	3	3
66	2	2
67	5	5
68	7	7
69	4	4
70	0	0
71	2	2
72	5	5
73	7	7
74	8	8
75	5	5
76	1	1
77	7	7
78	3	3
79	8	8
80	9	9
81	7	7

82	6	6
83	3	3
84	1	1
85	6	6
86	4	4
87	9	9
88	8	4
89	5	5
90	1	1
91	4	4
92	4	4
93	2	2
94	8	8
95	8	8
96	9	9
97	6	6
98	1	1
99	2	2
100	6	4
101	6	6
102	0	0
103	8	8
104	5	5
105	3	3
106	4	4
107	8	8
108	3	3
109	3	3
110	5	5
111	8	8
112	5	5
113	9	9
114	0	0
115	6	4
116	3	3
117	4	4
118	7	7
119	5	5
120	6	6
121	2	2
122	1	1
123	7	7
124	1	1
125	4	4
126	4	1
127	0	0
128	0	0
129	5	5
130	0	0
131	9	9
132	8	8
133	0	0
134	9	9
135	5	5
136	6	6
137	3	3
138	4	4
139	3	3
140	8	8
141	4	7
142	6	6
143	9	9
144	3	3
145	5	5
146	7	7
147	5	5
148	6	6
149	7	7
150	6	6
151	0	0
152	5	5
153	2	2
154	9	9
155	3	3
156	6	6
157	0	0
158	1	1
159	8	8
160	0	0
161	8	8
162	1	1
163	0	0
164	5	5
165	2	1
166	2	2
167	1	1

168	7	7
169	0	0
170	9	9
171	9	1
172	4	0
173	0	0
174	8	8
175	4	4
176	1	1
177	3	3
178	4	4
179	8	8
180	3	3
181	6	6
182	3	3
183	3	3
184	5	5
185	1	8
186	1	1
187	9	9
188	3	3
189	5	5
190	3	3
191	0	0
192	7	7
193	7	7
194	0	0
195	5	5
196	0	0
197	0	0
198	3	3
199	2	2
200	9	9
201	6	6
202	1	1
203	9	9
204	0	0
205	4	4
206	9	9
207	5	5
208	3	3
209	9	9
210	8	8
211	1	1
212	7	7
213	2	2
214	5	5
215	9	9
216	4	4
217	8	4
218	7	7
219	2	2
220	2	2
221	0	0
222	8	8
223	7	7
224	4	4
225	9	9
226	6	6
227	8	8
228	9	9
229	9	9
230	0	0
231	5	5
232	8	8
233	9	9
234	9	9
235	3	5
236	8	8
237	6	6
238	9	9
239	8	8
240	7	7
241	8	8
242	1	1
243	0	0
244	8	8
245	0	0
246	8	8
247	8	8
248	2	2
249	4	4
250	4	4
251	8	2
252	2	2
253	8	1

254	1	1
255	0	0
256	6	6
257	6	6
258	3	3
259	0	0
260	5	5
261	4	4
262	6	6
263	6	6
264	8	8
265	6	6
266	0	0
267	8	8
268	5	5
269	6	6
270	8	8
271	0	0
272	3	3
273	2	2
274	5	5
275	0	0
276	6	6
277	2	2
278	8	1
279	8	8
280	8	8
281	3	5
282	7	7
283	7	7
284	0	0
285	9	9
286	3	3
287	7	7
288	6	6
289	1	1
290	8	8
291	0	0
292	7	7
293	5	5
294	1	1
295	1	1
296	4	4
297	5	5
298	5	5
299	1	1
300	8	8
301	8	8
302	0	0
303	1	1
304	0	0
305	5	5
306	1	1
307	8	6
308	5	5
309	2	2
310	8	8
311	1	1
312	7	7
313	3	3
314	8	8
315	9	9
316	4	4
317	9	9
318	4	1
319	9	9
320	9	9
321	3	3
322	9	9
323	5	5
324	7	7
325	3	3
326	8	8
327	0	0
328	6	6
329	5	5
330	2	2
331	7	9
332	9	9
333	4	4
334	3	3
335	2	2
336	0	0
337	7	7
338	9	9
339	0	0

340	1	8
341	9	9
342	7	7
343	5	5
344	7	7
345	0	0
346	1	1
347	0	0
348	5	5
349	8	5
350	7	7
351	6	6
352	5	5
353	1	1
354	9	8
355	0	0
356	9	9
357	9	9
358	4	4
359	0	0
360	1	1
361	5	5
362	8	8
363	6	6
364	3	3
365	1	1
366	0	0
367	6	6
368	7	7
369	9	9
370	9	9
371	8	1
372	2	2
373	2	2
374	1	1
375	0	0
376	1	1
377	4	4
378	6	6
379	8	8
380	4	4
381	4	4
382	2	2
383	7	7
384	3	3
385	3	3
386	4	4
387	6	6
388	3	3
389	7	7
390	5	5
391	1	1
392	5	5
393	0	0
394	7	7
395	5	5
396	6	6
397	2	2
398	0	0
399	7	7
400	6	6
401	5	5
402	1	6
403	7	7
404	9	9
405	6	6
406	4	4
407	4	4
408	4	4
409	8	8
410	0	0
411	4	4
412	6	6
413	4	4
414	7	7
415	5	5
416	5	5
417	7	7
418	2	2
419	9	9
420	6	6
421	3	3
422	3	3
423	4	4
424	3	3
425	2	2

426	7	7
427	6	6
428	0	0
429	2	2
430	6	6
431	1	1
432	6	6
433	4	4
434	4	4
435	1	1
436	1	1
437	6	6
438	3	3
439	8	8
440	6	6
441	6	6
442	6	6
443	0	0
444	2	2
445	5	5
446	6	6
447	7	7
448	2	2
449	5	5
450	0	0
451	3	3
452	8	8
453	2	2
454	2	2
455	1	1
456	8	8
457	0	0
458	8	8
459	8	8
460	2	2
461	9	8
462	0	0
463	1	1
464	6	6
465	9	9
466	2	2
467	2	2
468	3	3
469	8	8
470	4	4
471	8	8
472	5	5
473	7	7
474	6	6
475	0	0
476	3	3
477	5	5
478	6	6
479	5	5
480	5	5
481	2	2
482	2	2
483	1	1
484	9	9
485	3	3
486	3	3
487	7	7
488	9	9
489	5	5
490	7	7
491	7	7
492	2	2
493	8	8
494	8	8
495	3	3
496	1	1
497	7	7
498	5	5
499	2	2
500	1	1
501	9	9
502	0	0
503	4	4
504	0	0
505	8	8
506	2	2
507	6	6
508	1	1
509	0	0
510	9	9
511	4	4

512	0	0
513	9	9
514	4	4
515	0	0
516	7	7
517	8	8
518	9	9
519	2	2
520	2	2
521	8	1
522	2	2
523	9	9
524	5	5
525	7	7
526	4	4
527	7	7
528	5	5
529	4	4
530	9	9
531	3	3
532	0	0
533	4	4
534	5	5
535	8	1
536	7	7
537	1	1
538	0	0
539	3	3
540	1	1
541	6	6
542	2	2
543	0	0
544	1	6
545	7	7
546	6	6
547	5	5
548	7	7
549	9	9
550	8	4
551	0	0
552	9	9
553	4	4
554	3	3
555	7	7
556	7	7
557	8	8
558	4	4
559	1	1
560	6	6
561	4	4
562	7	7
563	3	3
564	4	4
565	0	0
566	3	3
567	8	8
568	2	3
569	2	2
570	9	9
571	6	1
572	7	7
573	9	9
574	1	1
575	9	9
576	8	8
577	1	1
578	8	8
579	1	1
580	6	6
581	8	8
582	6	6
583	7	7
584	5	5
585	7	7
586	7	7
587	0	0
588	8	8
589	9	9
590	6	6
591	9	9
592	1	1
593	5	5
594	2	2
595	0	0
596	2	2
597	5	5

598	2	2
599	0	0
600	3	3
601	9	9
602	6	6
603	2	2
604	3	5
605	5	1
606	1	1
607	2	2
608	5	5
609	4	4
610	7	7
611	3	3
612	4	1
613	1	1
614	8	8
615	0	0
616	8	8
617	5	5
618	6	6
619	6	6
620	6	6
621	7	7
622	8	8
623	5	5
624	5	5
625	5	5
626	4	4
627	3	3
628	3	3
629	4	4
630	4	4
631	2	2
632	1	1
633	3	3
634	6	6
635	8	8
636	3	3
637	5	5
638	8	8
639	6	6
640	9	9
641	4	4
642	0	0
643	6	6
644	1	1
645	8	8
646	4	4
647	1	1
648	3	3
649	5	5
650	7	7
651	7	7
652	9	9
653	2	2
654	7	7
655	7	7
656	0	0
657	0	0
658	3	3
659	8	8
660	7	7
661	4	4
662	4	4
663	0	0
664	0	0
665	1	1
666	6	6
667	2	2
668	8	8
669	8	8
670	9	9
671	5	5
672	7	7
673	2	2
674	1	1
675	4	4
676	2	2
677	3	3
678	0	0
679	4	4
680	3	3
681	0	0
682	2	2
683	2	2

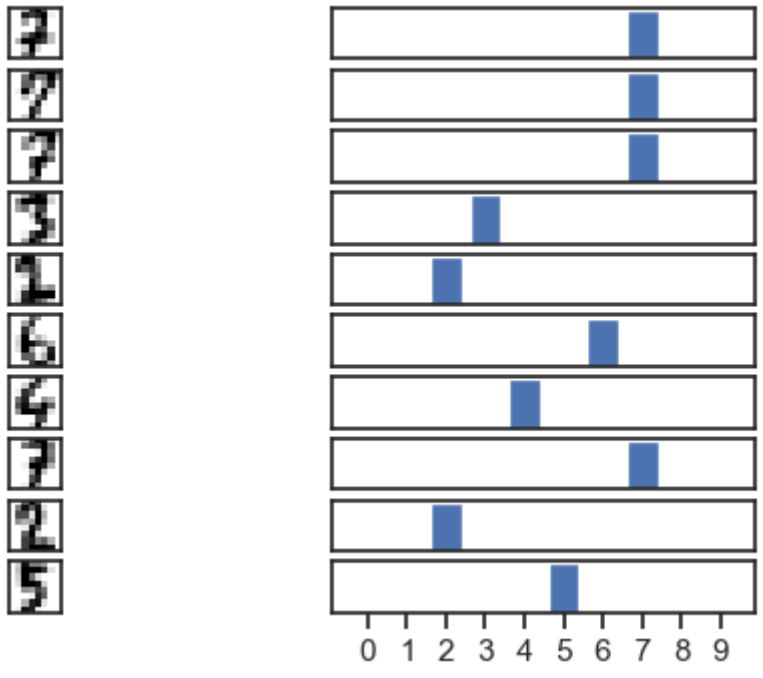
684	1	1
685	0	0
686	4	4
687	6	6
688	5	5
689	9	8
690	7	7
691	7	7
692	0	0
693	8	1
694	4	4
695	1	1
696	6	6
697	7	7
698	7	7
699	8	8
700	6	6
701	3	3
702	8	8
703	4	4
704	1	1
705	8	8
706	1	1
707	0	0
708	8	8
709	2	2
710	8	8
711	9	9
712	5	5
713	2	2
714	5	5
715	1	9
716	8	8
717	7	7
718	7	7
719	3	5
720	2	2
721	0	0
722	1	1
723	1	1
724	1	1
725	3	3
726	4	4
727	0	0
728	1	1
729	5	5
730	8	8
731	3	3
732	2	2
733	9	9
734	3	3
735	5	5
736	2	2
737	7	7
738	9	1
739	5	5
740	9	9
741	5	5
742	7	7
743	5	5
744	4	4
745	9	9
746	0	0
747	6	6
748	7	7
749	4	4
750	3	3
751	1	1
752	8	8
753	2	2
754	1	1
755	9	9
756	3	3
757	4	4
758	8	8
759	8	8
760	8	8
761	0	0
762	4	4
763	9	9
764	6	6
765	1	1
766	1	1
767	0	0
768	9	9
769	4	4

770	6	6
771	6	6
772	1	1
773	2	2
774	0	0
775	2	8
776	2	2
777	8	8
778	5	5
779	8	8
780	8	8
781	3	3
782	4	4
783	6	6
784	3	3
785	3	3
786	4	4
787	8	8
788	5	5
789	1	1
790	7	7
791	3	3
792	5	5
793	9	9
794	2	2
795	2	2
796	9	9
797	4	4
798	4	4
799	1	1
800	5	5
801	6	6
802	4	4
803	3	3
804	7	7
805	0	0
806	5	5
807	6	6
808	1	1
809	7	7
810	6	6
811	1	1
812	1	1
813	2	2
814	3	3
815	1	1
816	0	0
817	9	9
818	5	5
819	4	4
820	3	3
821	7	7
822	2	2
823	0	0
824	1	1
825	1	1
826	6	6
827	1	1
828	7	7
829	3	7
830	4	4
831	4	4
832	0	0
833	4	4
834	6	6
835	1	1
836	8	8
837	4	4
838	0	0
839	2	1
840	0	0
841	6	6
842	1	1
843	1	1
844	4	4
845	8	8
846	5	5
847	5	5
848	5	5
849	8	5
850	5	5
851	2	2
852	7	7
853	6	6
854	0	0
855	1	1

```
856     9       9
857     3       3
858     0       0
859     3       3
860     9       9
861     4       4
862     7       7
863     3       3
864     2       2
865     8       8
866     0       0
867     6       0
868     3       3
869     0       0
870     6       6
871     8       8
872     3       3
873     9       9
874     6       6
875     0       0
876     5       9
877     1       1
878     3       3
879     2       2
880     5       5
881     8       8
882     8       8
883     8       8
884     1       1
885     4       4
886     4       4
887     6       6
888     2       2
889     0       0
890     9       9
891     4       4
892     4       4
893     1       1
894     7       7
895     3       3
896     2       2
897     2       2
898     4       4
```

But we can also make probabilistic predictions:

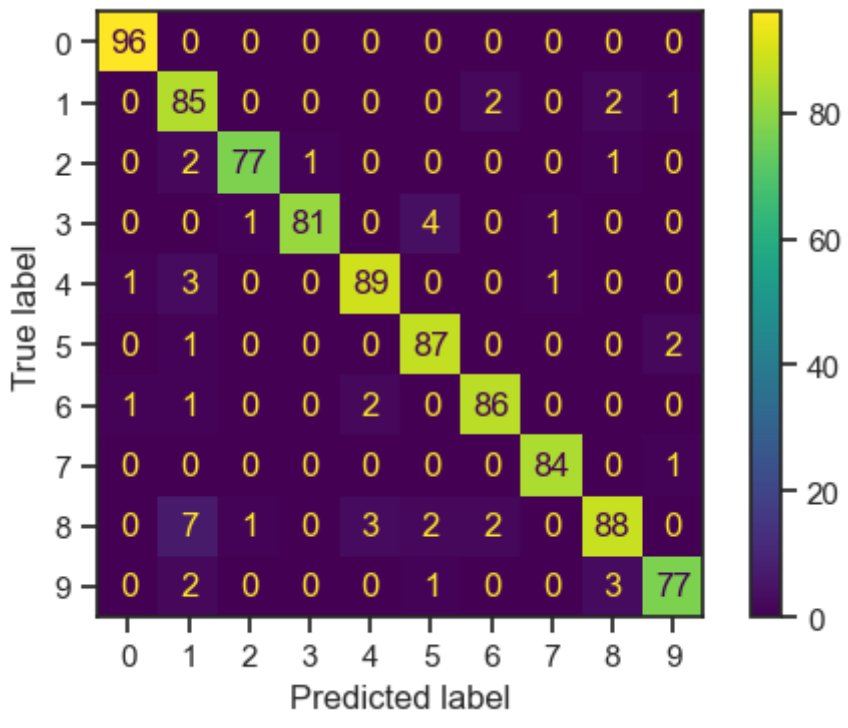
```
prob_predict = model.predict_proba(X_test)

fig, axes = plt.subplots(10, 2)
for i in range(10):
    axes[i, 0].imshow(
        X_test[i].reshape((8, 8)),
        cmap=plt.cm.gray_r,
        interpolation='nearest'
    )
    axes[i, 0].set_yticks([])
    axes[i, 0].set_xticks([])
    axes[i, 1].set_xticks([])
    axes[i, 1].bar(
        np.arange(10),
        prob_predict[i, :]
    )
    axes[i, 1].set_yticks([])
axes[-1, 1].set_xticks(np.arange(10))
axes[-1, 1].set_xticklabels(model.classes_);
```



Scikit-learn has the capability to run many accuracy metrics at once for you. Here is everything including the precision matrix:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

print(f"Classification report for model {model}")
print(
    metrics.classification_report(y_test, predicted)
)

cm = confusion_matrix(
    y_test,
    predicted,
    labels=model.classes_
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=model.classes_
)
disp.plot();
```

```
Classification report for model LogisticRegression(max_iter=2000, penalty='none')
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        96
           1       0.84      0.94      0.89        90
           2       0.97      0.95      0.96        81
           3       0.99      0.93      0.96        87
           4       0.95      0.95      0.95        94
           5       0.93      0.97      0.95        90
           6       0.96      0.96      0.96        90
           7       0.98      0.99      0.98        85
           8       0.94      0.85      0.89       103
           9       0.95      0.93      0.94        83

    accuracy                           0.95       899
   macro avg       0.95      0.95      0.95       899
weighted avg       0.95      0.95      0.95       899
```



Questions

- Look at the precision matrix carefully and identify the digits for which the most mistakes are made. Why does this happen? Write code below to visualize some of the wrong predictions.

```
# Your code below this point
```

By Ilias Bilionis (ibilion[at]purdue.edu)