3 Kernel Perceptron

(1) Each of 20 runs used a randomly (and uniquely) split ‘zipcombo‘ dataset into 80% train and 20% test.
Each run for each polynomial degree d from 1 to 7 was performed on train and test datasets.
The mean train and test error rate percentages and standard deviations across the 20 runs are shown in Table 1 (to 2 d.p.):

d	**train**	**test**
1	7.58±0.84	1.84±0.22
2	1.35±0.47	0.85±0.13
3	0.47±0.01	0.69±0.11
4	0.25±0.15	0.63±0.10
5	0.13±0.05	0.60±0.09
6	0.09±0.04	0.60±0.06
7	0.07±0.03	0.62±0.07

Table 1.

(2) A “best” parameter d^* is determined 20 times by 5-fold cross-validation of the 80% training data split.
Using this d^*, we re-train weights with the full 80% training set and run tests on the remaining 20% errors.
Of the 20 d^*s, the mean and standard deviation was 5.55 ± 0.67. Of the same 20 for which tests were run (on the 20% dataset), the mean and standard deviation was 0.60 ± 0.09.

3 Kernel perceptron contin.

(3) The same operations as question 2 were repeated, replacing mistake counts with confusion matrices. The values in this matrix are calculated by counting the number of instances whereby a particular digit ('true y') was misclassified ('predicted y'), such that every combination of misclassifications is added up and divided by the total count of that particular digit, giving the mean values (Figure 1).

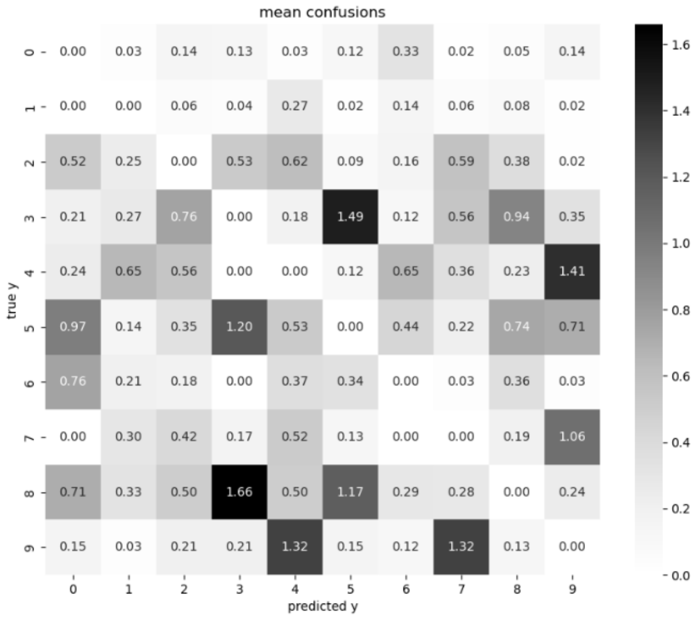

Figure 1. Mean confusions

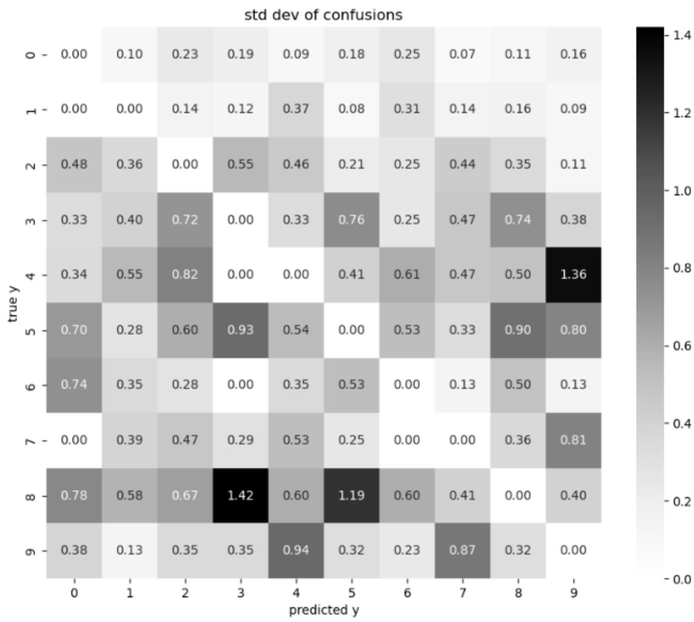

Figure 2. Standard deviations of confusions

3 Kernel perceptron contin.

(4) The sum of confusion error rates for each digit, according to the 20% test set (and 3 epochs) is shown in Table 2 below:

0	1	2	3	4	5	6	7	8	9
0.99	0.69	3.16	4.88	4.22	5.30	2.28	2.79	5.68	3.64

Table 2. Hence the five hardest-to-predict numbers according to the confusion matrix are: 8, 5, 3, 4, 9 (in order from the hardest).

However, the wording of the question seems to suggest that it is individual images (rather than one of the 10 classes), and potentially from within the full unsplit dataset, that the five hardest-to-predict should be identified. It is not completely clear how to find the five hardest by this definition. The approach taken was as follows:

The weights were trained on the full data set, 3 epochs were used as before, with degree 5 (rounded down from the mean d^* calculated in question 2. ($d = 5.55$ was tried but unsurprisingly - in hindsight - it caused numerical instability issues). Predictions were then made, again on the full dataset, using the trained weights.

19 digits were misclassified out the 9298 digits that are in the full dataset, shown in Figure 3 below:

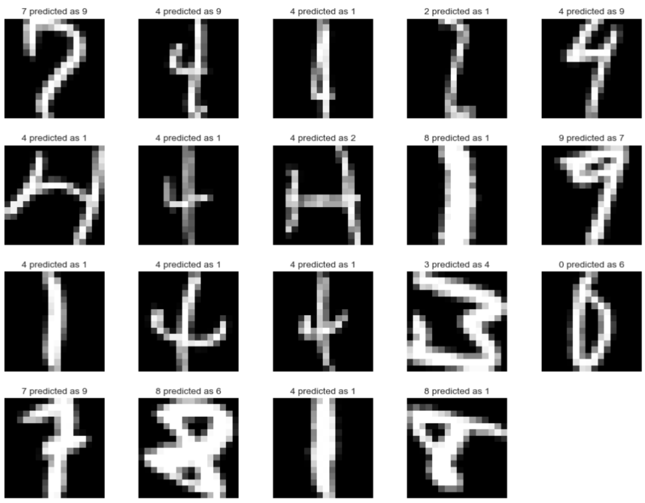

Figure 3. 19 incorrectly predicted images from the full dataset of 9298 digits.

3 Kernel perceptron contin.

(4) contin.

Some of these misclassifications are more surprising than others.
The high prevalence of 4s, numbering at 10 of the 19, is not surprising given how varied their appearances are in this set.
The 7s and 9s are confused twice, which is not so surprising.
A 6 and 8 confused is not all that surprising.
There are several examples where the number other than 1 looks like a 1, and so it is not surprising when these are 'wrongly' predicted as 1s.
The 8 in the 4th row, 4th column is predicted to be a 1 though it looks more like either a 9, 4 or an 8.
Predictions for row 2, column 1 and row 2, column 3 are more surprising being predicted as a 1 and 2.
Regardless, it seems remarkable that only 19 of 9298 were misclassified given such a simple algorithm - there are no convolutional neural networks here, just a 'kernel trick' and a basic perceptron.

Table 3 below indicates the percentage rate of misclassification per digit.
'total' is the sum of occurences of each digit in the full dataset.
'misclassified' is the sum of misclassified occurences (these are the 19 misclassified digits shown in Figure 3 above):
'rate (%)' is the misclassified divided by the total, multiplied by 100.

digit	**total**	**misclassified**	**rate** (%)
0	1553	1	0.06
1	1269	0	0
2	929	1	0.11
3	824	1	0.12
4	852	10	1.17
5	716	0	0
6	834	0	0
7	792	2	0.25
8	708	3	0.42
9	821	1	0.12

Table 3. Rates of misclassifications with full dataset (%)

3 Kernel perceptron contin.

(5) TODO ...

Repeating 1 and 2 with a Gaussian kernel (d^* is now c and 1,...,7 is now S): From 20 runs for $c = 1, ..., 7$.

Basic results:
Each run uses a randomly split ‘zipcombo‘ into 80% train and 20% test.
The mean train and test error rates and standard deviations are shown in Table 2 below:

d	**train**	**test**
1	00.00±0.0	00.00±0.0
2	00.00±0.0	00.00±0.0
3	00.00±0.0	00.00±0.0
4	00.00±0.0	00.00±0.0
5	00.00±0.0	00.00±0.0
6	00.00±0.0	00.00±0.0
7	00.00±0.0	00.00±0.0

Table X

Cross-validation:
A “best” parameter c^* is determined by 5-fold cross-validation of the 80% training data split.
Using this c^*, we retrain on the full 80% training set. The test errors are computed on the remaining 20% using the same c^*.
The test error for this value of c^* is recorded and the process above is repeated 19 more times, resulting in 20 c^* and 20 test errors.
The mean test error and mean c^* is 00.00 ± 0.0 and 0.0 respectively.

3 Kernel perceptron contin.

(6) TODO ...

An alternate method to generalise the kernel perceptron to k-classes might be to use ...:

Basic results:
Each run uses a randomly split 'zipcombo' into 80% train and 20% test.
The mean train and test error rates and standard deviations are shown in Table 3 below:

d	**train**	**test**
1	00.00±0.0	00.00±0.0
2	00.00±0.0	00.00±0.0
3	00.00±0.0	00.00±0.0
4	00.00±0.0	00.00±0.0
5	00.00±0.0	00.00±0.0
6	00.00±0.0	00.00±0.0
7	00.00±0.0	00.00±0.0

Table 3. Error rates (%)

Cross-validation:
A "best" parameter c^* is determined by 5-fold cross-validation of the 80% training data split.
Using this c^*, we retrain on the full 80% training set. The test errors are computed on the remaining 20% using the same c^*.
The test error for this value of c^* is recorded and the process above is repeated 19 more times, resulting in 20 c^* and 20 test errors.
The mean test error and mean c^* is 00.00 ± 0.0 and 0.0 respectively.

3 Kernel perceptron contin.

(6) TODO ...

A.

A discussion of any parameters of your method which were not cross-validated over.

B.

A discussion of the two methods chosen for generalising 2-class classifiers to k-class classifiers.

C.

A discussion comparing results of the Gaussian to the polynomial Kernel.

D.

A discussion of our implementation of the kernel perceptron:

$$\mathbf{w}(\cdot) = \sum_{i=0}^{m} \alpha_i K(\mathbf{x}_i, \cdot)$$

represents a function $\mathbf{w}$ which takes some input (which here is expected to be the flattened array of MNIST image pixels for a digit and computes the weighted sum of kernel evaluations for this input and the image pixels for all the other digits in some dataset.
Alpha, the classifier, is an array of coefficients which are learned during training, according to mistakes made. The original mathematica code uses variable 'GLBcls' for alpha. We replaced this with 'alphas'.
The result of the weighted sum

(i) was represented by....
(ii) was evaluated by ...
(iii) had new terms added to it during training by ...

E.

(Any table produced in 1-6 above should also have at least one sentence discussing the table.)