ADS2 Practical 2.10
Supervised learning: MNIST digit classification

Facilitators: GL, DS, JC, WZ

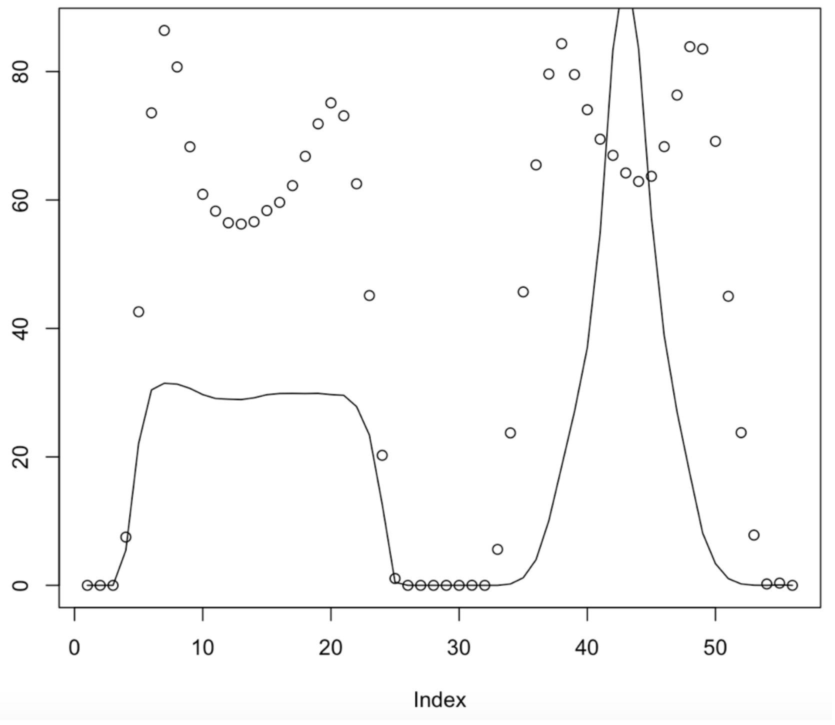
April 23, 2024

Training a supervised learning model for digit classification

Based on results from the previous session, hopefully now you have a `features` dataframe with 1000 rows/observations and with the first column (`features$label`) representing class label (0..9) and other columns (e.g. 56 of them, if you choose the features suggested in part 1) representing features.

Note: calculating features may take a while (like minutes or even hours), but once you have them, everything else should be pretty quick! (that's one reason why we use them)

To ensure that your features make sense, first plot their statistics for different labels to see that they seem correct as intended. For example, if you use mean values per row and then per column as your features, their mean values for digit 0 (circles) and digit 1 (lines) should look as follows:



Once you are confident about your features, we can proceed with training a supervised learning model. As we discussed quite a bit about neural networks in the lecture, I would suggest using a neural network, but you are free to use anything else you feel comfortable with. I will provide instructions how to do it with neural networks, but some of them apply to other methods as well. For them, use `library(nnet)`.

First of all, we should normalize our data. This can be achieved for example by dividing all feature values by 255 to ensure that they fall between 0 and 1 (in this way, normalisation will also not be dependent on statistics of the training data, so can also be equally applied to the testing data).

Once feature values are normalized, we should split our data into training and validation sets, e.g. with 700 examples for training and 300 for validation. It's best to select examples randomly, e.g. with `rows <- sample(1:1000, 700)` (after initialising the random number generator), but then use the same selection throughout your training.

We also have testing data, but normally we don't touch it until we finished adjusting our model parameters (like number of hidden layer neurons in a neural network).

Now we should extract training data and labels. For example, with the format described above it could be done as follows:

```
train_labels <- features[rows, 1]
valid_labels <- features[-rows, 1]
train_data <- features[rows, -1]
valid_data <- features[-rows, -1]
```

Now, as we have 10 classes (for digits from 0 to 9), we need to generate an appropriate output representation for a neural network (and some other methods too) with a N x 10 matrix where the correct class is denoted as 1 and other classes are denoted as 0 (and N is the number of examples, so for example 700 for training data).
This can be done using the following function:

```
train_labels_matrix = class.ind(train_labels)
```

So, for example, the following train_labels

```
> head(train_labels)
[1] 5 0 1 7 7 9
```

will produce the following train_labels_matrix

```
> head(train_labels_matrix)
     0 1 2 3 4 5 6 7 8 9
[1,] 0 0 0 0 0 1 0 0 0 0
[2,] 1 0 0 0 0 0 0 0 0 0
[3,] 0 1 0 0 0 0 0 0 0 0
[4,] 0 0 0 0 0 0 0 1 0 0
[5,] 0 0 0 0 0 0 0 1 0 0
[6,] 0 0 0 0 0 0 0 0 0 1
```

Now we are ready to train our neural network.

We can do this using the following function

```
nn = nnet(train_data, train_labels_matrix, size = 4, softmax = TRUE)
```

It will train a network with 4 hidden units (in one layer), performing 100 iterations by default. You will get a result like the following,

```
# weights:  278
initial  value 1676.371429
iter  10 value 1397.234493
iter  20 value 1086.076845
iter  30 value 866.645778
iter  40 value 697.143178
iter  50 value 615.241827
iter  60 value 568.235269
iter  70 value 538.730898
iter  80 value 521.986354
iter  90 value 507.359435
iter 100 value 488.741360
final  value 488.741360
stopped after 100 iterations
```

It indicates the number of weights (parameters) and development of errors with training.

To use your train model for classifying data, use the predict command. For example,

```
pred_train = predict(nn, train_data, type="class")
pred_valid = predict(nn, valid_data, type="class")
```

will compute predicted classes for training and validation data.

You can calculate your classification accuracy by using

```
mean(pred_train == train_labels)
mean(pred_valid == valid_labels)
```

for training and validation data.
What is the chance level? How does your result compare to it?

Now experiment training your model with a different number of parameters (e.g. 1 to 12 hidden layer neurons). How does classification accuracy for training and validation data change with that? Does it look similar to the bias-variance dilemma plot from the lecture?

Finally, take your best performing model and apply it to the testing data (e.g. again its first 1000 examples). Remember to calculate their features and normalize them in the same way. How does classification performance there compare to training and validation performance?