COMP2002 - Artificial Intelligence

Week 3 - Neural Network Exercises - Model Answer

Introduction

This set of model answers is intended to show you possible ways of solving the neural network exercises in week 3 – there are others. Please attempt the exercises before looking at the answers.

Activities

Your task is to go through the following tasks. Please note, that you are expected to complete some work on this outside of the timetabled sessions.

Exercise 1

This exercise has the following elements:

- Import the digits dataset.
- Train a neural network to classify the digits using Scikit's *MLPClassifier* object.
- Construct a confusion matrix to illustrate the results.

Before we do anything, we need to import the packages we'll be using:

```python
from sklearn.datasets import load_digits
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

The first step is to load the data. This is the same as in last week's exercises; recall that the Scikit package provides the data in three parts – the data, the images of the digits, and the target values. As in last week's load the data as follows:

```python
# Load the Iris data.
data = load_digits ()
inputs = data.data
targets = data.target
```

The next step is to classify the data. We do this in three steps:

1. Instantiate the object.
2. Fit the training inputs to the corresponding targets.
3. Capture the outputs by using the classifier's predict method.

Here is the code that trains the classifier:

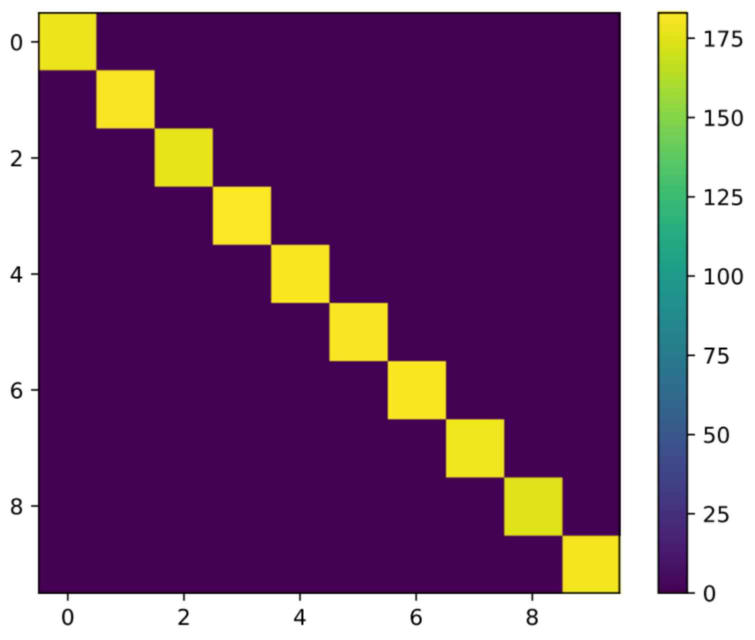
```python
# Classify the data.
classifier = MLPClassifier()
classifier.fit(inputs, targets)
outputs = classifier.predict(inputs)
```

Finally we need to assess the predictive quality. We will build a confusion matrix to show the frequency that target values were correctly predicted. Recall from last week's lecture that in the confusion matrix C, the element $C_{i,j}$ shows the number of times that target i was classified as output j. So, if element $C_{2,6} = 47$ then on 47 occasions an input corresponding to the digit 2 was classified as a 6. What we want is for the biggest values to be in the diagonal entries, $C_{i,i}$, as this means that the majority of targets were predicted to have the same output. You can see that this is the case in the output from this short program below.

Here is the code to construct the confusion matrix:

```python
# Compute and visualise the confusion matrix.
C = confusion_matrix(targets, outputs)
plt.imshow(C)
plt.colorbar()
plt.savefig("confusion.pdf", bbox_inches="tight")
plt.show()
```

Here is the confusion matrix from running the program. You can see that the highest numbers are in the diagonal entries, meaning that in most cases the predictions were accurate.



Exercise 2

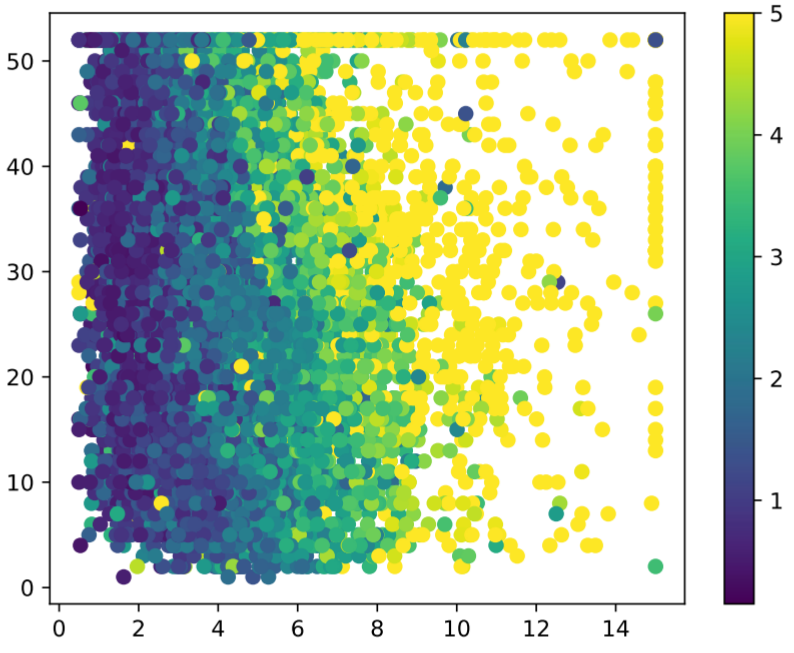
Your task in the subsequent exercises was to use the California House Price dataset, also available from Scikit. As discussed in this week's lecture, this is a regression problem in which the challenge is to predict the price of houses in California based on characteristics of the properties. This exercise required you to load the data and plot two of the inputs against each other. As with the other datasets we've seen, each input is in its own column, so your task is to take any two columns in the input data and plot them against each other. I've done this for the first two inputs, 0 and 1. I've coloured the points according to the target value – the following code produces the plot shown below. Remember, the target values lie in the region 0.15 to 5.

```python
from sklearn.datasets import fetch_california_housing
import matplotlib.pyplot as plt

# Load the data.
```

```
data = fetch_california_housing()
inputs = data["data"]
targets = data["target"]

# Plot columns 0 and 1 (can replace with any pair of inputs).
plt.scatter(inputs[:,0], inputs[:,1], c=targets, cmap="viridis")
plt.colorbar()
plt.savefig("california_data.pdf", bbox_inches="tight")
plt.show()
```



Exercise 3

As you can see from the plot in Exercise 2, the various inputs are on different scales. To make sure that the model gives equal weight to each input we need to pre-process them by normalising. We'll use the *MinMaxScaler* within Scikit to do this, which normalises the values so that they lie between 0 and 1. as follows:

```
from sklearn.preprocessing import MinMaxScaler

# Scale the data.
scaler = MinMaxScaler()
scaled = scaler.fit_transform(inputs)

# Print the range of the variables to show the normalisation effect.
print(inputs.ptp(axis=0))
print(scaled.ptp(axis=0))
```

The *ptp* method shows you the distance between two variables. In this case, it will show you the distance between the minimum and maximum value in each column. The first print will show the original values, the second will show all 0s and 1s.

Exercise 4

The final exercise requires you to train the model. You were given code that will do this in the lecture, training the MLP and then assessing it with the mean_absolute_error function. Here is the code again:

```python
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error

regressor = MLPRegressor()
regressor.fit(scaled, targets)
outputs = regressor.predict(scaled)
print(mean_absolute_error(targets, outputs))
```