COMP2002 - Artificial Intelligence

Week 2 - Machine Learning Exercises - Model Answer

Introduction

This set of model answers is intended to show you possible ways of solving the Machine Learning exercises in week 2 – 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 - Classify the Iris data

There are five parts to the code for this solution:

1. Load the required modules.
2. Load the Iris data.
3. Project the data into two dimensions using PCA.
4. Train a classifier and predict the class labels.
5. Render the plot.

The solutions here do the model training on the 4-dimensional data; you could you the PCA projection for this too. The first step is to import the modules we'll need, as follows:

```python
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
```

Fortunately the Iris data is included with the Scikit packages, so we can simply load it using a Python function.

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

The *load_iris* function returns two things – the data to be used for classification and the target values. We'll separate them into a separate variable to keep our code readable.

Having loaded the data, it can be projected into two dimensions using principal component analysis. We do this by setting up the PCA object, and initialise it to project onto two components. The main reason for this is so that we can visualise the data – if you want a more accurate representation you can use more components, but you won't be able to make a 2D plot of it.

PCA works by mapping your data into a low-dimensional space (in this case, 2-dimensional). Therefore, there are two steps to using it: (1) pass in the data to construct the mapping (fit the data) and (2) transform it into the low-dimensional space. The *fit_transform* method does both of these steps in one go for us. So the compressed variable is our 2-dimensional representation of the data.

```
# Project the data into two dimensions using PCA.
pca = PCA(n_components=2)
compressed = pca.fit_transform(inputs)
```

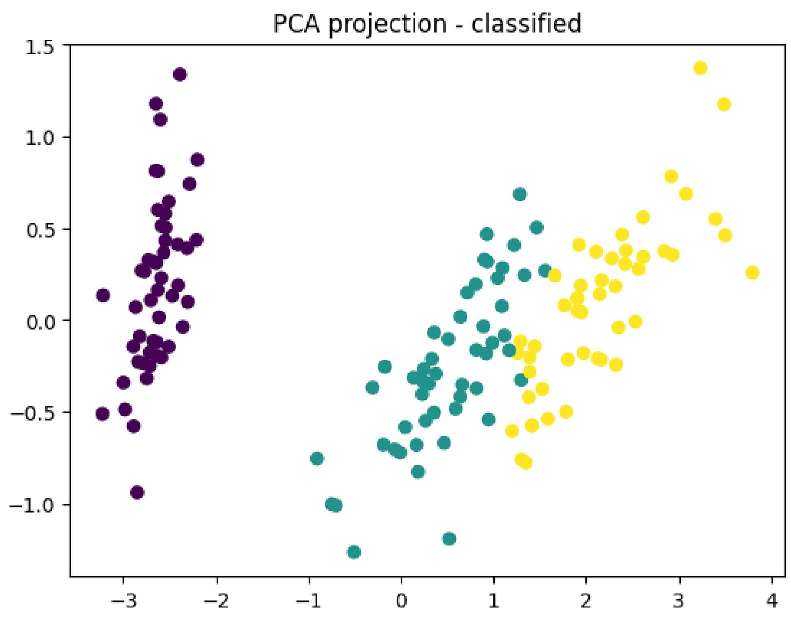
We can now train the classifier. We're going to use the *k-nearest neigbour* classifier included within Scikit. Instantiate the classifier (you'll need to determine the best number of neighbours – this usually takes some experimentation to see what gives the best result) and then *fit* the data to the targets. Using the *predict* method will then give you the predicted labels, and if you pass in some unseen inputs you'll get their classifications.

```
# Train the classifier.
classifier = KNeighborsClassifier(n_neighbors=10).fit(inputs, targets)
classifiedData = classifier.predict(inputs)
```

One of the advantages of using Scikit is that all of its prediction tools – for classification, clustering and regression – follow a very similar pattern of use. So once you've learned how to use one of them you will be able to use any of them.

Finally, produce a plot showing the classification using the following code:

```
# Plot the results.
plt.figure()
plt.scatter(compressed[:,0], compressed[:,1], c=classifiedData)
plt.title("PCA projection - classified")
plt.savefig("iris_pca_classified.png", bbox_inches="tight")
plt.show()
```



Exercise 2 - Loading the data with Pandas

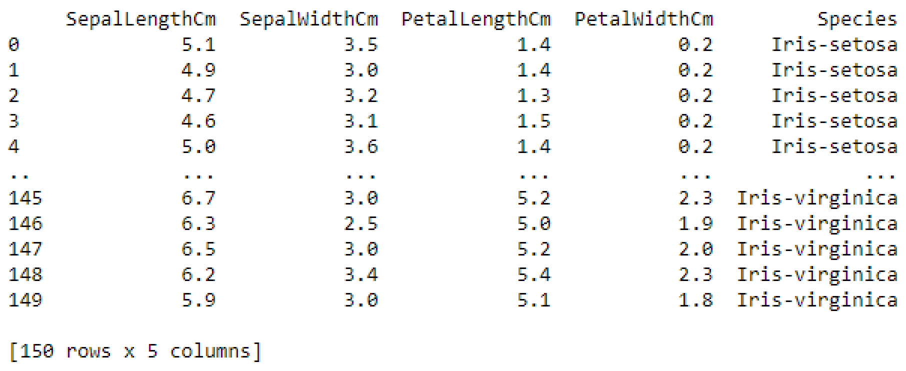
Exercise 1 required you to use the Iris data from within Scikit. Here you need to load it using the Pandas package, and convert the data for use within Numpy arrays. Begin by loading the two packages:

```
import pandas
import numpy as np
```

Having imported the packages, load the data using Pandas. The data is provided in a comma separated format (*iris.csv*, available on the DLE). The *read_csv* method is used to load the data, and I've printed the *data* variable so that you can see how it's presented – this isn't strictly necessary, but it's always useful to do a quick visual check that the data has been imported correctly once you've loaded it.

```
# Load the data using Pandas. Print it to view it.
data = pandas.read_csv("iris.data")
print(data)
```

The result of your print statement should look something like this:



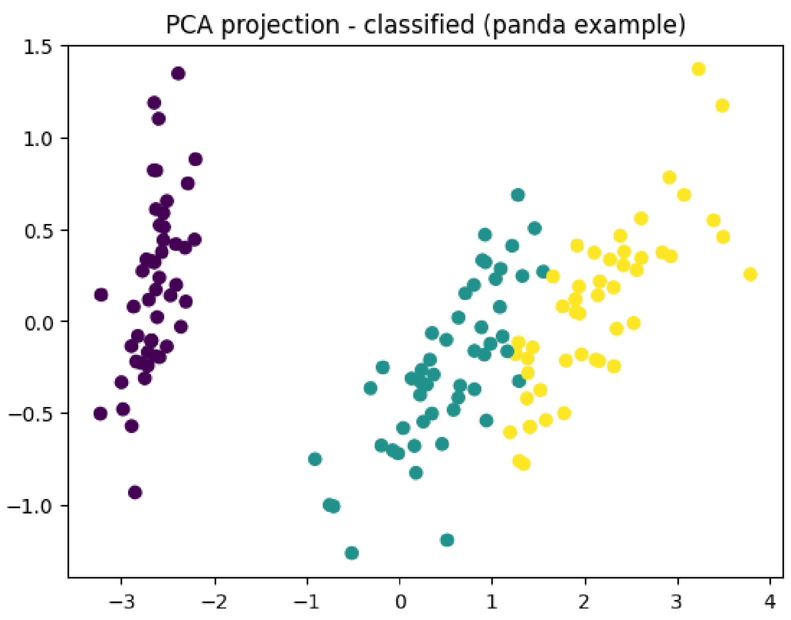
```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm         Species
0              5.1           3.5            1.4           0.2     Iris-setosa
1              4.9           3.0            1.4           0.2     Iris-setosa
2              4.7           3.2            1.3           0.2     Iris-setosa
3              4.6           3.1            1.5           0.2     Iris-setosa
4              5.0           3.6            1.4           0.2     Iris-setosa
..             ...           ...            ...           ...             ...
145            6.7           3.0            5.2           2.3  Iris-virginica
146            6.3           2.5            5.0           1.9  Iris-virginica
147            6.5           3.0            5.2           2.0  Iris-virginica
148            6.2           3.4            5.4           2.3  Iris-virginica
149            5.9           3.0            5.1           1.8  Iris-virginica

[150 rows x 5 columns]
```

We're going to convert the *DataFrame* object (the object Pandas puts data into by default) into a pair of Numpy arrays – one for the inputs and one for the targets. We'll also need to convert the targets to numerical values (they are strings withing the file – either "Iris-setosa", "Iris-versicolor" or "Iris-virginica").

```
# Extract the inputs.
inputs = data.values[:,:-1].astype(float)
# Extract the targets - convert to numerical values to help with colouring
# when we plot the results.
classes = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
targets = [classes.index(cls_str) for cls_str in data.values[:,-1].astype(str)]
targets = np.array(targets)
```

This code makes use of *slicing* – extracting specific parts of the array. For example, for the inputs, [:,:-1] extracts all rows and everything up to (but not including) the final column (shown by :-1). In the case of the targets, we only want the final column, so we use [:,-1] to retrieve all rows and only the column with the index -1. **Having added the Pandas code you should ensure that your plot works as in Exercise 1. The resulting figure should look like this:**



Exercise 3 - Classifying Digit Images

This exercise requires you to complete a similar set of steps to classify some written digit data. As with the Iris dataset, you could load this from a file on your computer but I'm using the version that comes with Scikit.

```
from sklearn.datasets import load_digits
```

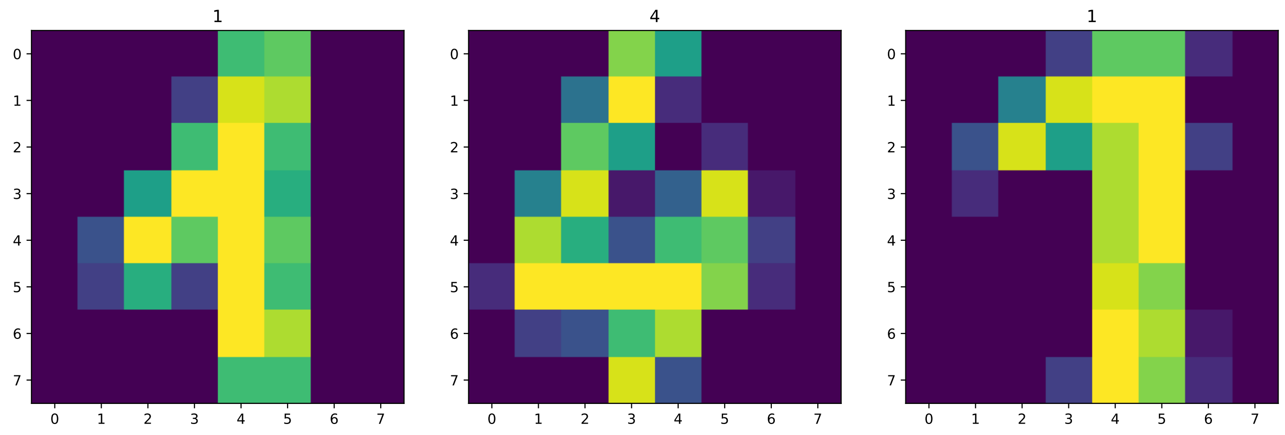
Second, the code that actually extracts the data and parses into three variables (data, images and targets):

```
# Load the data
digits = load_digits ()
# Extract the three parts of the data.
data = digits.data
images = digits.images
target = digit.target
```

The data comes in three parts - the data you can use for machine learning, a set of images (one representing each digit in the data, conveniently formatted for visualising) and the target values.

First I will plot some of the data to check I have loaded it in correctly (it is good practice to check this):

```
# Plot some of the digits
for idx in [200, 800, 1500]:
 plt.figure()
 plt.imshow(images[idx,:])
 plt.title(targets[idx])
 plt.savefig(f"digit_{idx}.pdf", bbox_inches="tight")
```

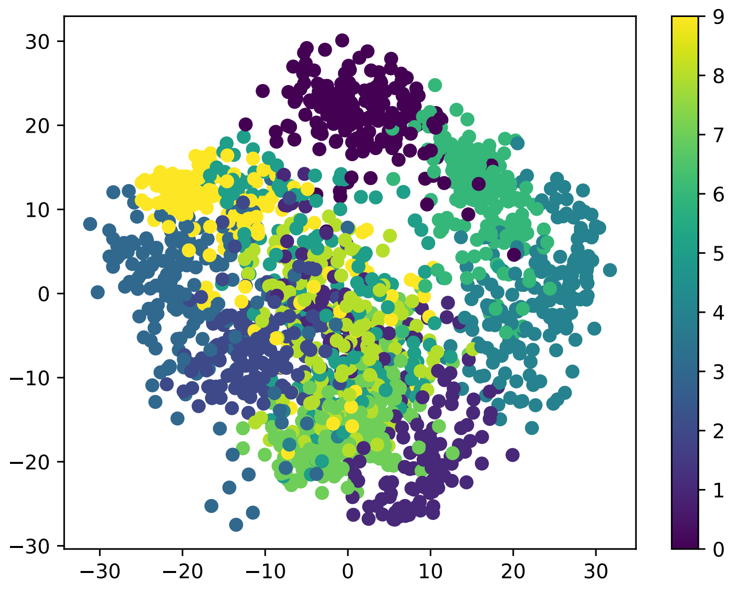


To create these images I used a list that defined three of the images and looped over it. The f at the start indicates that I'm going to include a variable in the sting which in this case was idx.

Now we have checked that the data was loaded in correctly, we prepare the data for classification. We are going to reduce the dimensionality with PCA so that we can visualise the whole dataset.

```
# Reduce the dimensionality with PCA
compressed = PCA(n_components=2).fit_transform(data)
```

```
# Plot the resulting PCA embedding.
plt.figure()
plt.scatter(compressed[:,0],compressed[:,-1],c=targets, cmap="viridis")
plt.colorbar()
plt.savefig("digits_PCA.pdf",bbox_inches="tight")
```



The colour of each point indicates the digit it represents and we can see that digits are broadly kept together. Looking at which digits are close together also reveals reassuring information about what has been done by PCA, in that numbers which are similar (e.g. 1 and 7) are placed close together.

Now the data has been compressed into two dimensions, we can start on the machine learning part. Now we split the data into two sets - training and testing. The code for this is shown below:

```
# Split the data into training and testing sets.
xtrain, xtest, ttrain ,ttest= train_test_split(compressed,targets)
```

We are going to use an SCV object and train it with the training data after which we will test the model with the test data.

```
# Train the classifier and run the test data through it.
classifier = SVC(gamma="auto")
classifier.fit(xtrain,ttrain)
ytrain = classifier.predict(xtrain)
ytest = classifier.predict(xtest)
```

Finally, we wish to know how well our model is working. For this, we will use the mean absolute error on both the training and test sets.

```
# Compute and print the model accuracy.
trainAccuracy = accuracy_score(ttrain,ytrain)
testAccuracy = accuracy_score(ttest,ytest)
print(f"{trainAccuracy} training accuracy, {testAccuracy} testing accruacy")
```

You are looking for a result of approximately 0.8 for the training data and 0.6 for the testing data. Note: these values are for this dataset and model, other datasets and models will differ drastically!