CAB320 Week 11 Prac Sheet – Neural Network

Adapted by Thierry Peynot (2024) from a prac sheet written by Frederic Maire (2022-2023)

In this practical, you will build and train different classifiers for the classification of handwritten digits. You will also use some metrics to assess the performance of your classifiers.

Exercise 1 - Retrieve and inspect the MNIST dataset

Use the scaffolding code in the provided file `sklearn_nn.py`, load the MNIST dataset. (Note: this dataset is a 'classic' in the machine learning community. The challenge of classifying those digits led to what many consider as the beginnings of deep learning).

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
```

- Print the keys of the `mnist` object (which is an instance of a dictionary-like class called "`Bunch`").
- What are the entries "`data`" and "`target`"?
- What are their shapes?

For convenience, alias the inputs and the class labels with

```
X, y = mnist["data"], mnist["target"]
```

Plot the example indexed 99 with the following code:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
some_digit = X[99]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = mpl.cm.binary, interpolation="nearest")
plt.axis("off")
plt.show()
```

Exercise 2 – Linear classifier

Let us keep the same X and y from the previous exercise.

- What is the type of the entries of y?
- Convert the entries of y to `uint8` (hint: use ndarray method 'astype')
- Create a training and test set using
  `X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]`

Let us build a binary classifier to distinguish between '5' and 'non 5'

- Create boolean arrays (masks) to identify the digits that correspond to a '5' in the arrays `y_train` and `y_test`.

We will build a classifer for the digit class '5' that implements a regularized linear model with

stochastic gradient descent (SGD) learning.

- Create an instance `sgd_clf` of the class `sklearn.linear_model.SGDClassifier`
- Train `sgd_clf` to discriminate between '5' and 'non 5' (hint: use the `fit` method)
- Predict the classes of the batch `X_test[50:80]` with `sgd_clf`. Check the results by plotting the digits.
- Convert the entries of y to `uint8` (hint: use ndarray method '`astype'`)

A good way to evaluate a model is to use cross-validation. Let us use the `cross_val_score` function to evaluate your SGDClassifier model using K-fold cross-validation, with three folds. Remember that K-fold cross-validation means splitting the training set into K-folds (in this case, three), then making predictions and evaluating them on each fold using a model trained on the remaining folds.

- Import the function `cross_val_score` from the module `sklearn.model_selection`
- Call the `cross_val_score` with the appropriate parameters (you can use scoring="accuracy")

Exercise 3 – Confusion Matrix

In a previous lecture we have introduced the confusion matrix, which is fairly comprehensive way to visualise the performance of a classifier. To compute the confusion matrix, you first need to have a set of predictions, so they can be compared to the actual targets. You could make predictions on the test set, but let's keep it untouched for now (remember that you want to use the test set only at the very end of your project, once you have a classifier that you are ready to launch). Instead, you can use the `cross_val_predict` function from the module `sklearn.model_selection`.

Just like the `cross_val_score` function, `cross_val_predict` performs K-fold cross-validation, but instead of returning the evaluation scores, it returns the predictions made on each test fold. This means that you get a clean prediction for each instance in the training set ("clean" meaning that the prediction is made by a model that never saw the data during training).

• Compute the confusion matrix by combining the functions `cross_val_predict` and `confusion_matrix`.

• What do the rows and columns of the confusion matrix represent?

Exercise 4 – Random Forest classifiers

Decision trees are still a popular type of classifiers when the inputs have discreate values. This week you can use them as a black box.

A **random forest** is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Random Forest classifiers can directly classify instances into multiple classes.

You can call `predict_proba()` to get the list of probabilities that the classifier assigned to each instance for each class. Let us train a RandomForestClassifier. The `predict_proba()` method returns an array containing a row per instance and a column per class, each containing the probability that the given instance belongs to the given class.

- Create a random forest classifier by instantiating an object `forest_clf` of the class `RandomForestClassifier` (hint: look at the `sklearn.ensemble` module).
- Train the forest on the 10 digit classes (multiclass classification).
- Print the confusion matrix

Exercise 5 – Multi-layer Neural Network in sklearn

The MLPClassifier and MLPRegressor are sklearn implementations of neural networks. In this exercise, you will be training a neural network based multiclass classifier for the MNIST dataset (downloaded in Exercise 1). As neural networks are sensitive to the scale of the input, we will first pre-process the data.

- Scale the data with a `StandardScaler` from the `sklearn.preprocessing` module
- Build a classfier `mlp` using the class `MLPClassifier` from the `sklearn.neural_network` module. Suggestion use “`hidden_layer_sizes=[10,10,10], verbose=True, max_iter=1000`” as the parameter list of the constructor.
- Fit `mlp` to the train data and plot the confusion matrix (don’t forget to apply the `StandardScaler` to `X_test`!)

Exercise 6 – Multi-layer Neural Network in Keras

In this exercise, use Keras to implement and train the neural network described in Exercise 6. Some scaffolding code is provided in the file `keras_nn.py`