Digits Lab - Week 6

There are three files for this lab:

1. This pdf : `Digits.pdf`
2. The workbook : `digits.ipynb`
3. The images : `digits.zip`

All files are available on Moodle. Complete all work in the corresponding notebook. All written answers must be in **Markdown** cells.

You should use `GridSearchCV` when doing cross-validation.

If you don't have time for sections 4 and 5, skip to section 6.

When answering questions, a one-line answer is not sufficient. Interpretation and reasoning are just as important as model building.

Although this is an image classification problem (where CNNs would normally be used), we will investigate how well classical machine learning models perform.

Hand Written Digits

You are given 5,000 images of handwritten digits in the set

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

There are:

- 10 classes
- 500 samples per class
- Each image is 20×20 pixels

Your goal is to compare several classical classifiers and select the best model using validation performance.

1. Preparing the Data

You must get this section working before moving on.

1. Create the feature matrix X and target vector y.
   - Use `glob.glob('data/*/*.txt')` to generate a file list.
   - Load each image using:

     ```
     np.array(Image.open(filename))
     ```

     and `for fname in filelist` might help.
   - Each image is 20×20 — reshape so that X is a 2D array. `.flatten` or `.reshape` could be used.

- Ensure y is a 1D array aligned correctly with X.
  - Remember, you cannot guarantee filelist is ordered or there is an equal distribution of categories.
    - You can get the response variable from the filename, this may help (changes needed for Linux and Mac)

```
np.array([int(fname.split('\\')[-2]) for fname in filelist])
```

  - If you type y in its own cell it should look something like this `array([0, 0, 0, ..., 9, 9, 9])`

2. Check:
   - Shape of X
   - Shape of y
3. What range of values do the pixels take?
4. Scale the data so pixel values lie in $[0, 1]$.
5. If everything is set up correctly, the code commented as `##Printing Examples` will output a random selection of 10 examples for each class.
6. Split the dataset into:
   - Training set
   - Validation set
   - Test set

   Use `train_test_split` with `random_state` set. 60% / 20% / 20% could be a good split. You could use `stratify=y` to make sure the classes are split into train/valid/test evenly rather than completely random.

   You must use the same split for all models.

2. Linear Support Vector Machine

We begin with a linear SVM. This is often a strong baseline for high-dimensional data.

1. Train a model using:

```
SVC(kernel='linear')
```

2. Report:
   - Training accuracy
   - Validation accuracy

   Comment on whether overfitting appears to be present.

3. Print: (on validation set)

   - Confusion matrix
   - Classification report

   Which digits are most frequently confused?

4. Use 5-fold cross validation on the training set to tune the regularisation parameter C.

   - Test a sensible range of C values.
   - Select the best C based on cross-validation performance.

5. Retrain using the best C on the training set. Report the validation accuracy (save this score).

3. Random Forest

Now compare to a nonlinear ensemble model.

1. Train a RandomForestClassifier with default settings. Report training and validation accuracy.

2. Does this model appear to overfit?

3. Print confusion matrix and classification report.

4. Use 5-fold cross validation to tune:

   - `n_estimators`
   - `max_depth`

   Choose reasonable ranges.

5. Retrain the best model on the training set. Report the validation accuracy (save this score).

4. (If Time) k-Nearest Neighbours

1. Train kNN with $k = 5$. Report training and validation accuracy.

2. Does this model overfit?

3. Use 5-fold cross validation to choose the best k.

4. Retrain using the best k. Save the validation score.

5. (If Time) Logistic Regression

1. Train LogisticRegression (default settings).
2. Report training and validation accuracy.
3. Use 5-fold cross validation to tune regularisation.
4. Retrain best model and record validation score.

6. Model Selection

1. Which specific model achieved the best validation performance?
   Be precise. For example:
   "The best model was a Linear SVM with $C = 0.1$."
2. Combine the training and validation sets.
3. Retrain the chosen model.
4. Evaluate once on the test set.
5. Report:
   - Test accuracy
   - Confusion matrix
   - Classification report

7. Conclusion

Write a short discussion:

- Which model performed best and why?
- Which model overfit the most?
- Which model was computationally most expensive?
- Are some models better at distinguishing certain digits?
- Would you expect a neural network to outperform these models?