CW2: A Memory of Wine

Davide Moltisanti & Michael Wray

March, 2019

1 Introduction

Year 2400. Because of global warming, grape vines, along with the majority of plant life, **are but a distant memory**. An archaeological dig found a so-called "hard drive" from 1991 AD buried beneath the tiers of past civilisations and miles of one-use plastics. The device contains precious data about a long lost drink called "wine". All those stories about ancestors who enjoyed their lives with a sip of wine drive you to investigate how to bring it back to Earth (and ship it to Mars)...

2 Overview

In this coursework you will be asked to classify a set of samples into a certain number of classes. You will first extract discriminative features from the data samples, then build a classifier that uses such features to recognise what class a given sample belongs to.

For this coursework you will need to provide a report along with your code. You will work in pairs for this coursework.

Read this document carefully before attempting any coding. Your code will be auto-marked, so check input/output formats carefully.

3 Dataset

In this coursework we will use the Wine Dataset. This dataset consists of 178 samples, each corresponding to a wine. Each wine is produced in the same region in Italy, and each wine derives from one of three different cultivars (or type of plant). We will treat the three different cultivars as classes.

Each sample contains 13 features/dimensions, corresponding to a chemical constituent of the wine: 1) Alcohol, 2) Malic acid, 3) Ash, 4) Alkalinity of ash, 5) Magnesium, 6) Total phenols, 7) Flavanoidsm, 8) Nonflavanoid phenols, 9) Proanthocyanins, 10) Color intensity, 11) Hue, 12) OD280/OD315 of diluted wines, 13) Proline.

Now, we are neither wine experts nor do we know much (if anything) about chemistry, but still we want to tell what cultivar each wine derives from. The beauty of machine learning is that we can do that (to some extent) without being an expert!

Training and testing We split the dataset in a 70/30 proportion to create the training and testing sets. These sets are labelled, i.e. the class each sample belongs to is known. You will need to use the training set to train the classifier, and use the test set to check your code is working correctly, as well as to report results. We also generated a private test set, which we will use to auto-mark your code. The private test set will contain the same features as the training and public test sets. We will not release the private test set.

4 Implementation

4.1 Feature Selection

You will have to **select two features only**, out of the 13 available features. To choose your features, do a scatter plot of all the pairwise combinations of the features (as we did already during the lab sessions), using only the training set. Observe the scatter plots and determine which pair of features best separates the data points into the three classes.

As you will notice, there is not just a single good pair, but rather multiple pairs provide equally good clustering. This adds to the challenge of selecting only two features, but fear not: we are not concerned about creating the best classifier in the world. Rather, we would like you to understand the difficulties involved in taking such decision, and to discuss about them accordingly in your report.

Include the plots in your report to facilitate your discussion on the feature selection. Once you have selected your pair of features, make sure you use only such two features to train and test your classifier.

4.2 K-Nearest Neighbours

Implement the K-Nearest Neighbour classifier to predict the class of a given unknown sample. Use the standard classification metrics **accuracy** to evaluate your classifier on the **test** set. Use $k \in \{1, 2, 3, 4, 5, 7\}$ and discuss how results vary according to k on your report. Calculate the confusion matrix to discuss how the classifier behaves according to the three different classes. You may either choose a single k for your discussion or show how the confusion matrix changes according to k. Both approaches are valid and we leave this decision to you.

Note You will have to code this yourself. Any calls to Scipy's K-NN or any other similar library are forbidden and will be treated as **not implemented**. Recall that we implemented the **Nearest Centroid** algorithm during our lab sessions. You should be able to adapt your Nearest Centroid code to get a working K-NN classifier with few changes.

4.3 Alternative Classifier

Using the same selected **two** features, implement another classifier of your choice (e.g. decision tree, Naïve Bayes, etc.). Compare and discuss in your report the obtained results with the K-NN ones. It is fine if the alternative classifier delivers poorer results than the K-NN's ones, as long as you provide an explanation for it in your report. In fact, what we are interested in here is your understanding of the difference between K-NN and your alternative classifier, and the interpretation of the respective results.

Note Again, you will have to code this yourself. Any calls to libraries implementing a classifier will be treated as not implemented. However, you are allowed to use some libraries to facilitate your implementation (e.g. you can use Scipy to generate a normal distribution whenever you need). If you are not sure about what is allowed and what is not, feel free to contact us.

4.4 Use Three Features

Try now to manually select **three** features as opposed to only two, adding an extra feature to your selected pair. Using your new set of features, run again K-NN with the same ks listed above. Rather than just attempting to improve the accuracy of the classifier, we would like you to discuss about the challenges you faced in selecting the features, as well as the benefit (if any) of using an extra dimension for the classification task.

Hint The only thing you should take care of in your code to make your classifier work with three features is how you calculate the distance between data points. If you use NumPy correctly you don't even need to change anything in your classifier's code.

4.5 Principal Component Analysis (PCA)

Instead of using the previously selected features, use PCA to reduce the original 13 features to only 2. Produce a scatter plot like the ones you produced in Section 4.1 for the PCA-reduced training set, and include the figure in your report. Compare the PCA plot to the scatter plot you obtained with your manually selected features.

Run again your K-NN classifier (using k as listed above) using the PCA-transformed data. Compare and discuss the PCA results with those obtained with your manually selected features and K-NN.

Note You may use Scipy's PCA implementation for this task. Be mindful of fitting the PCA model on the training set **only**, and use the fitted model to reduce the test set.

5 Marking

The code will be worth a maximum of 10 marks, while the report will be worth a maximum of 15 marks. The code will be automatically marked as detailed below.

Code marking

We will assign a mark according to how your accuracy compares with that we obtained with our coded solution on the private test set, **using the same features as your selected ones**. This means that regardless of the selected features you are able to get a full mark if your classifier is correctly implemented. This is because accuracy may vary considerably according to the selected features. We provide more details on the expected input/output of the program in Section 6.

Report marking

As you will have noticed, the report has a higher weight than the code. This is because we care more about your understanding of the concepts involved in this coursework.

The report should be **4 pages** long. Include as many figures as needed to support your observations. Be concise and demonstrate that you not only are able to code the solution, but more importantly that you can understand and elaborate on the obtained results. As a rule, we would **not** like to see any reference to the code in the report, whilst we would like to see brief clear statements that showcase your understanding.

6 Input/Output

We provide a skeleton Python script that will facilitate the use of the auto-marker. The file is called `wine_classifier.py` and contains some code that will help you to get started, such as data loading and command line arguments parsing. We also provide an `utilities.py` file that contains the code to load the data and, importantly, to print your results to the standard output in the format expected by the auto-marker. You should implement your solution adding your code to our skeleton script. Do not change the file name.

You can download the data and the skeleton code here: `https://github.com/UoB-COMS21202/lab_sheets_public/tree/master/Coursework_2`.

Data

The Wine dataset is split in a training and a test set. These are provided as CSV files named respectively `wine_train.csv` and `wine_test.csv`. Both contain 13 columns, corresponding to the 13 wine features. The training set contains 125 rows, each corresponding to a wine sample, while the test set contains 53 rows.

The class labels for the two sets are provided as CSV files named `wine_train_labels.csv` and `wine_test_labels.csv`. These files contain respectively 125 and 53 rows, and one column only. Each row contains a single number - either $(1, 2, 3)$ - indicating the class label of the corresponding sample in the training/test set - i.e. the sample at the same row index.

Running the code

The script expects a **mandatory parameter** `mode`. The parameter `mode` sets which of the 5 coursework points should be run when invoking the script. There are 5 possible self-explanatory modes: **`feature_sel, knn, alt, knn_3d, knn_pca`**. The auto-marker will run your code using the `mode` parameter to retrieve your various results. Your code should print only the relevant output for each mode, i.e. :

`python wine_classifier.py feature_sel` should print the indices of your selected pair of features. This is so that we can compare your classifier accuracy to our solution using the same features. Print only the features **pair**, i.e. do not print the three features we asked you to choose for comparison. For example:

```
$ python wine_classifier.py feature_sel
[0, 1]
```

`python wine_classifier.py knn` should print the class predictions obtained with your **K-NN** classifier and the selected pair of features, for each sample in the test set. For example:

```
$ python wine_classifier.py knn
[2, 1, 3, 1, 2, 3, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2,
1, 3, 2, 3, 2, 1, 2, 1, 3, 2, 2, 2, 1, 1, 3, 2, 1, 3, 3, 2,
2, 2, 3, 1, 3, 2, 2, 2, 1, 3, 2, 1, 3]
```

`python wine_classifier.py alt` should print the class predictions obtained with your alternative classifier, for each sample in the test set.

`python wine_classifier.py knn_3d` should print the class predictions obtained with your K-NN classifier and the three selected features, for each sample in the test set.

`python wine_classifier.py knn_pca` should print the class predictions obtained with your K-NN classifier and the PCA-reduced data, for each sample in the test set.

The script also accepts the following optional parameters:

- `--k`. The k parameter for the K-NN classifier. For example:

  ```
  $ python wine_classifier.py knn --k 3
  ```

  Will run the K-NN classifier with $k = 3$. Default is $k = 1$;

- `--train_set_path`. Path to the training set CSV file. For example:

  ```
  $ python wine_classifier.py knn --train_set_path
  /tmp/my_path/train_wine.csv
  ```

  will load the training set from `/tmp/my_path/train_wine.csv`. Default is `data/wine_train.csv`;

- `--train_labels_path`. Path to the training labels CSV file. Default is `data/wine_train_labels.csv`;

- `--test_set_path`. Path to the test set CSV file. Default is `data/wine_test.csv`;

- `--test_labels_path`. Path to the test labels CSV file. Default is `data/wine_test_labels.csv`.

Note that invoking the script with no parameters other than `mode` will attempt to load the data from a folder called `data` located under the working directory.

Hint The above results were obtained on the test set with our coded solution. You may use them to check your code.

Adding your Code

You will find 5 functions in the skeleton script, one for each of the coursework points. These functions are invoked according to the aforementioned `mode` parameter. At the moment these functions do nothing. We recommend you to write your code inside each function.

You will notice that the functions are already taking in input the necessary parameters. You will also notice that there is a special parameter called `**kwargs` in each function. This a Pythonic way of passing more optional parameters to a function. You can find a tutorial on how to use these here. You should not be worried about this special argument anyway, since we defined the functions so that they already take all the necessary parameters in input.

Make sure you use our functions `print_features(x)` and `print_predictions(y)` to print your results to the command line (both functions are already imported in the skeleton script). You can pass your features and predictions either as Python lists or as NumPy arrays. This will ensure a smooth auto-marking experience as the output format will be the expected one.

7 Submission

Submit only your `wine_classifier.py` Python file and your report in PDF format (we suggest learning and using LaTeX since it will be a useful skill for the future). Do **NOT** submit your code as a Jupyter notebook file. Do not submit any results in the form of text files either. Good luck!