Multivariate Time Series Classification

Practical Assignment 3

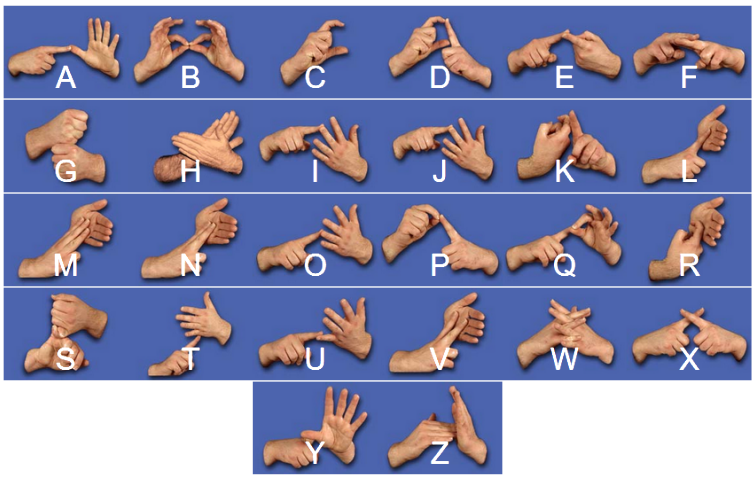
Informatics 2 for Biomedical Engineers

Graz University of Technology

Abstract

Classifying problems are common in every area of computer science. In this task we will take a look at recorded hand gestures in the Australian Sign Language (AUSLAN) Dataset. We classify the data based on the very simple nearest neighbour algorithm and use K-fold cross validation to evaluate the results. These are very straight forward methods providing decent results on this data and illustrate the difficulty of such problems and how to approach them.

Note, completing this assignment will require some computation time. Expect up to an hour of runtime for slower machines.



Two handed finger-spelling alphabet
Auslan Signbank, http://www.auslan.org.au/spell/twohanded.html

1. File Headers

All python source files (apart from the __init__.py) have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please feel free to copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces)!

2. Tasks

2.1. Reading the AUSLAN dataset (10pt)

Download the hand-sign recordings via the official resource[1] page or via the lecture git directory[2]. In the archive you will find 9 folders, each containing a week of recordings. In there, you will find each sign/word 3 times (these are also 3 different recordings). Each file consists of a sequence of lines consisting of 22 whitespace-separated numbers representing the 22 channels of information (sensors). pandas, with its *read_csv()* function[3], can be very helpful here. Your implementation should consider the following aspects:

(i) Abstract it in a similar fashion to assignment 1. Create a class serving as interface (you can use the abcmeta here). Then, implement a concrete class loading the data.

(ii) Do not hard-code the file paths. Your loader should receive a path where the data is stored. Then it should read all the folders representing the weeks. For each folder you should then load all the files inside.

[1] `https://kdd.ics.uci.edu/databases/auslan2/auslan.html`

[2] `https://github.com/pkasper/info2-bm/raw/master/assignments/tctodd.tar.gz`

[3] `http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html#pandas.read_csv`

(iii) Encode label names. Human readable labels are nice to understand your data but are bloat for computing. Your loader class should replace them with numeric values. However, you should store this mapping somewhere, to reconstruct the labels when you are done.

For every file you should check if contains valid data. This means you should be able to load 22 columns of data where every column has the same length. Note, the lengths can differ between the individual recordings, but that is intended. You should end up with some kind of data structure where you can select lists of recordings for each sign.

2.2. Classification and Cross-Fold Validation (10pt)

K-Fold. For this task we can use K-folding to evaluate our classifier. This means you split your entire data into K chunks. Setting K to 9 would be a smart choice, since we have 9 weeks of data. You start with taking the first chunk as test set and the remaining data as training set. Then you run your classification. You store the results and then repeat the process by taking the second chunk as test set and all others as training data. Repeat this until every chunk was the training set once.

Classification. To classify the data we will use 1-NN (1 nearest neighbour). For every entry in your test set, you look for the most similar sign in your train-set. There are multiple ways to calculate similarity between two series, such as Dynamic Time Warping. But the most straight forward way is to build a matrix for your series, deduct one from the other and then calculate the Frobenius norm. (Hint: numpy has a handy function for this!) To be able to calculate this difference matrix you require that both series have the same length. Again the simplest way is to cut the end of the longer series. The label you assign to your class in the test set should be label of the most similar training class. You should end up with 2 lists. One consisting of the labels returned by your algorithm for every entry in your test set, and one with their actual labels.

Cross Validation To validate your results you compare the two lists returned by the classifier (with your labelled results and the true labels). The metrics we are interested are precision, recall and the f1-score. sklearn provides some very helpful functions for this (look in particular for the classification report[4], which can be particular helpful during your testing). Since

[4] `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.`

you do this validation with each of your test-train set combinations your final values are simple the average precision, the average recall, and the average f1-score across all sets.

Interpret your results. See how well your algorithm classified the dataset and check if there are any classes with poor performance. Write down your results into a text file called `results.txt`. Please add the comment header to this file too!

2.3. Early Classification (5pt)

In some cases using reduced all the available data (here, the full time series) introduces a high amount of noise. We may get better results but just looking at the first parts of the data. When you compare two series you already strip the tail of the longer series. For early classification you then again, cut the tail of both series by a fraction. You can do this in 5% steps. Analyse (and visualise in a plot) how your performance (mean precision, recall and f1 score) change when you classify based on shorter series. Again, discuss your results in the `results.txt`.

3. Bonus Tasks: Multiprocessing (5*pt*)

For the 1-NN classification you have to take every item in your test set and compare it with every item in your train set. This can take some time if you have a lot of data. However, modern computers usually have multiple processor cores. To leverage this power python has the multiprocessing[5] module. Look at the documentation, create a pool of multiple workers and use the map (or probably more useful starmap) functions call your classification function and to classify multiple items from your test set at once.

4. File Headers

All python source files (apart from the __init__.py) have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number

`classification_report.html`

[5] `https://docs.python.org/3.6/library/multiprocessing.html`

- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please feel free to copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces)! You may want to adapt it to consider all members of your team. **Example:**

```
##############################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Here we import the package and do stuff
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
##############################################################################
```

5. External Code

If you use **any** code from any online source you must always make a note about this in the code. A short comment explaining where you took the code from and what it does is enough. Please note, this includes and code you take from any of the reference solutions. Else these lines of code will be considered plagiarism!

6. Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible
- Try to avoid *.__MACOSX* or *.DS_STORE* folders in your submission archives.

6.1. Allowed packages

Any that came bundled with anaconda! See this list: `https://docs.continuum.io/anaconda/pkg-docs#python-3-6`. Any packages marked as *In Installer* can be used. Note, this link leads to the 3.6 specifications. This is intended as 3.5 is technically outdated. However, it is safe to assume everything that is listed in the 3.6 package is also available in 3.5.

7. Coding Standard

For this lecture and the practicals we use PEP 8 [6] as a coding standard guideline. Please note that whilst we do not strictly enforce all these rules we will not tolerate messy or unreadable code. Please focus especially on the following three aspects.

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in python 3 (whilst allowing more freedom in python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all use a single underscore (_) for its name.

Max 72 characters. Do not have lines longer than 72 characters. Whilst PEP 8 allows 79 in some cases we ask you to use the lower cap of 72 characters per line. Please note that this includes indentation.

8. Automated Tests

We will monitor the standard output. For this task please make sure you do not have any other output and you use proper capitalisation (as defined by the task).
Please make sure your submission follows all the restrictions defined in this document.

9. Submission

9.1. Deadline

4^{th} of June 2017 at 23:59:59.

[6] `https://www.python.org/dev/peps/pep-0008/`

Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will automatically be pushed back for 24 hours.
If for whatever reason you are unable to submit your submission on time please contact your tutor as soon as possible.

9.2. Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your python source files please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. This is important to us as immediate feedback so we can adjust the tutor sessions. Please note that all submissions will be anonymised prior to being read. Hence please feel free to be honest (but do stay polite).
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every file (apart from the `readme.txt`)
- Coding standard upheld

9.3. Your submission file

The base directory structure of your submission should look as depicted below. If you wish to do so, you can add more files to your archive.