PreProcessing

May 31, 2019

1 Proof of Concept - Pre-Processing

This brief report summarises the method used to preprocess the raw gestures data into manageable, database ready data. It is important that the data is free from errors at this stage as well as have values in a form that is also manageable for direct import into the database management system (MSSQL in this case).

1.1 Imports

The python libraries used for this pre-processing are well established in the data science field. We are creating and managing data with the use of the pandas DataFrame functionality and are also using the os library built-in to python to manage access to the local filesystem.

```
In [1]: import pandas as pd
        from os import listdir
        from os.path import isfile, join, isdir
```

1.2 Process

The first step in developing the pre-processing system is to manage access to the raw data. For the Gestures data, each candidate's records were placed in a folder associated with a number. The files themselves are timestamped and associated with their parent folder. In order to obtain a database of individual records to access, a small helper function is required to traverse and collate each file.

The code below establishes the location of the raw data and the output path in respect to the execution directory of the python code. The loop below simply prints, the file directory for these files which aligns to the filestructure of the raw data as explained above.

```
In [2]: path_to_data = "./Raw Data/gestures/"
        output_path = "./Raw Data/Formatted_Files/"

        for dir in listdir(path_to_data):
            print(dir, end=',')
```

```
01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
```

The following code now produces a list of directories for the system to traverse to retreive data from. The difference between the direct listdir above and this code is that only subdirectories of the parent directory are listed. You can see this as the raw data's README.txt is not listed.

This step was added as a simple way of ensuring that the raw data can be directly obtained and pre-processed before use without any need to manually modify any information

```
In [3]: folders = [dir for dir in listdir(path_to_data) if isdir(join(path_to_data, dir))]

        for dir in folders:
            print(dir, end=',')
```

```
01,02,03,04,05,06,07,08,09,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,
```

Below is the code that was used to prepare data for importing to the MSSQL database. The only difference being that the file was exported as a single csv file before importing. The basic process of this function is to obtain the list of files present in each folder as defined above. Once obtained, the system then reads the information directly as a tab seperated csv file (As the raw data is expressed). Also note that the usecols function strips the time information from the data. As this final system was a classification task, the requirement of time space information was deemed not necessary and has hence been excluded from the system.

Once the system has completed compiling all of the data into a single pandas DataFrame, some processing was performed on the data. Firstly, this involved removing the subset of the data with the class value equal to 0 as this corresponds to unlabeled data (as stated in the README). As these classes are not useful for training/validation of the chosen neural network, they are cleanly removed from every file. After this set of processing, the result was then shuffled to ensure that a random distribution of each subjects data was obtained. For the final neural network, this ensures that specific epochs are not biased towards the gestures of a single individual, rather a random subset of gestures from the entire pool.

```
In [4]: output_df = pd.DataFrame()


        for folder in folders:
            files = [f for f in listdir(path_to_data + folder + '/') if isfile(join(path_to_data

            for file in files:
                df = pd.read_csv(path_to_data + folder + '/' + file, sep='\t', header=0, usecols
                df.drop(df.loc[df['class']==0].index, inplace=True)
                output_df = output_df.append(df)

        output_df = output_df.sample(frac=1).reset_index(drop=True)
        output_df.head()
```

Out[4]:

	channel1	channel2	channel3	channel4	channel5	channel6	channel7	\
0	-0.00001	-0.00003	-0.00001	0.00001	0.00001	0.00000	-0.00002	
1	-0.00005	-0.00005	-0.00005	-0.00005	-0.00003	-0.00003	-0.00001	
2	0.00005	-0.00002	0.00001	0.00004	0.00008	0.00009	0.00007	
3	0.00056	0.00006	0.00002	0.00000	-0.00001	-0.00001	0.00006	
4	-0.00001	0.00001	0.00018	-0.00033	0.00041	0.00031	0.00025	

	channel8	class
0	0.00000	1.0

```
1  -0.00004    1.0
2   0.00011    3.0
3   0.00024    3.0
4   0.00013    5.0
```

The final task is to analyse the final dataset. With the following info function available in pandas.DataFrame, we are able to see some high level information about the data. The total amount of records amounted to over 1.5 million with all values in every channel and class non-null. Although the values are expressed here as float64, the class variable will be modified to be an ungsigned integer in the database as a full 64 bit float value is considerably more storage than required for this variable.

```
In [5]: output_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1512751 entries, 0 to 1512750
Data columns (total 9 columns):
channel1    1512751 non-null float64
channel2    1512751 non-null float64
channel3    1512751 non-null float64
channel4    1512751 non-null float64
channel5    1512751 non-null float64
channel6    1512751 non-null float64
channel7    1512751 non-null float64
channel8    1512751 non-null float64
class       1512750 non-null float64
dtypes: float64(9)
memory usage: 103.9 MB
```