Twitter Data Wrangling Report

Author: Simon Thornewill von Essen
Date: 13.02.2018

Data Input

The first dataset that was meant to be downloaded manually was relatively easy to import. This was done by clicking on the hyperlink provided to me, moving the data into the relevant directory and importing using Python's Pandas package using the `pandas.read_csv()` function.

The next dataset was meant to be downloaded programatically using the `Requests` package. The package takes a URL and saves the response to a variable which can then be saved or written to a relevant file before being imported again with `Pandas`.

The final dataset was not trivial to download. I used some code from another repository to help me a little with this
section and referenced it in the README file.

Instead of putting the twitter token files inside of the Python file, created a CSV file which contained them and make sure that git ignores them so that they do not get uploaded to github using the `.gitignore` file.

The connection to the twitter API was done using `tweepy` the response was written to `data/tweet_json.txt` using the `json` library and the `open` command as can be seen at `In [5]` in the ipynb/html file. The problem encountered here was that I wasn't using the `json` library at first which made writing the file a little more difficult than it needed to be. I had to make sure to add `wait_on_rate_limit=True, wait_on_rate_limit_notify=True` into the API constructor within the code to make sure that the connection would not time out from the server.

The text file that was created was not very easy to handle because of how big it is. For some reason I have a hard time opening it in any kind of notepad file or within Atom. However, importing it and selecting the information I required using pandas was relatively straightforward as above.

After importing all the data, I made sure to create copies of the data frames using `pandas.DataFrame.copy()` so that nothing strange would happen to the original dataframes that I imported.

Data Assessment

Next, the imported data needed to be assessed for quality issues. This was done programmatically within my ipynb file and visually using Calf from the OpenLibre suite. Visual assessment was not very well documented not only because it is difficult to point directly to what you are viewing without messing around with image files, but also because I kind of neglected this part of the assessment a little bit. This meant that I ran into more problems during the cleaning stage later in the wrangling process.

Visual Assessment

I could recreate what I wrote in my jupyter notebook file, but this would be a waste of time so I'll just keep this brief.

Basic problems I found during the visual assessment were that some IDs that were handed to the API did not come back with any result, which might cause problems for me down the line. It was also noted that many of the images that were sent to @WeRateDogs were not actually dogs at all! Removing these images would be important for assessment.

One thing that I missed as well was that the columns containing the dog stage not only was structured as a sort of "sparse matrix" when it could just be one column (tidiness issue), but it also was the case that roughly 90/200 of the values were present, which wasn't immediately noticeable because the missing values were denoted with a "None" string instead of a `NaN` value. There were other columns as well which were not super important so I just removed them.

Programmatic Assessment

The programatic assessment is where I found most of the issues that I wanted to address during the wrangling phase. It is worth noting however that there is not a hard line btween what constitutes programmatic and visual, as i noted within the notebook itself. The general problems that I found was missing values using `pandas.DataFrame.info()`, which I used quite a lot throughout the wrangling process. The `tweet_id` was always an integer when it should have been a string (see Ordinal vs Categorical values).

I used `pandas`'s filtering capabilities to help me figure out where to draw the line between which pictures were and werent dogs. This means that I had to look at the top three predictions that were either predicted to be a dog or not and then look at the image visually in order to assert if this categorisation was true or not. I found that images were unlikely to be dogs if the top three predictions were false or if there was a `True` prediction with a probability of 0.2.

Data Cleaning

During the cleaning I solved the problems that I outlined in the Visual Assessment. Some of the issues I had were due to not thoroughly assessing the data visually. This meant that I had to skip a lot throughout the document to make sure all data entries were properly updated, which was cumbersome.

There were also a lot of times when commands that I was using werent working for one reason or another. One time I tried to remove duplicate rows using `pandas.DataFrame.drop_duplicates()` but this did not work because the rows that sometimes contained the same names of dogs sometimes had different `tweet_id`s.

However, having defined the small "step-by-step" tasks helped me a lot to focus on each individual one and execute them, which meant that this step took much less time than it might have if I did not have a plan, so I am very happy about that.

Data Output

After the data cleaning was completed, I created a CSV file and a SQLite database using `pandas` and `sqlalchemy` respectively.

The SQLite database was made by creating an engine object using the `sqlalchemy.create_engine()` constructor and then using the `pandas.DataFrame.to_sql()` command using the engine I had just created to export the data.

The CSV creation was much less interesting, that was just done using `pandas.DataFrame.to_csv()`.