wrangle_act

October 25, 2020

1 WeRateDogs Twitter Feed

This project looks at various data sources for Tweets from the WeRateDogs Twitter account, specifically:

1. the `twitter-archive-enhanced.csv` which contains the tweet text, as is the core data set
2. the Twitter API is used to access the original tweets to retrieve missing fields such as the retweet and favorite counts
3. an image prediction file containing the top 3 predictions for each of the (up to 4) dog pictures in the tweet

---

1.1 GATHER DATA

We use a number of data assets including local files, remote files on web servers, and JSON payloads returned by the Twitter API.

1.1.1 Read the CSV data

Pandas `read_csv()` function is quite versatile when uploading data, and can be configured to handle different date formats, numeric data types, not available (NA) markers, etc. Getting this right upfront can save time, but requires the raw data in files to be eyeballed first. For this we can use command line tools like head & tail, or Excel, which allows column headings to be frozen, data to be sorted and searched, etc.

Having looked at the raw data, we make the following observations:

1. tweet Ids are large integers, we need to select an approriate integer datatype so no accuracy is lost
2. some tweet Ids use floats, e.g.: `in_reply_to_status_id`, `in_reply_to_user_id`, with NaNs used as a Not Available marker, as mentioned above these need to be converted to integers
3. time stamps are close to ISO 8601 format, and are GMT
4. some strings have embeded new line characters, `read_csv()` handles this, but it messes up commands like `wc -l`

Actions taken to address above observations:

- convert floating point tweets Ids to a 64-bit integer, retaining the Not Available representation
- specifcally tell Pandas which columns are dates

Load the enhanced Twitter archive, using explicit data types for fields, instead of letting Pandas infer them. The Twitter API will define the data types for the Twitter fields, to which I add the data types for the "enhanced" fields.

To get around the fact that nullable numeric fields, by default, are interpreted by `read_csv()` as floats (so as to include NaN to represent null or Not Available), I am mapping optional tweet Ids to Pandas nullable integer data type.

1.1.2 Examine the core data set

The first discrepancy we note is that, according to the project motivation document, the main "archive contains basic tweet data for all 5000+ of their tweets" however that is clearly not the case as, having loaded it, the number of tweets is less than half that. As this is the master data set we have been provided with, this is the data we will go with, since it has been previously enhanced.

Just to double check this row count, we will run a line count on the input file, which should roughly match the number of rows in the data frame. Any discrepancy on counts is due to those embeded new line (NL) characters in the tweet text, as was previously mentioned.

Now we can double check the column data types, against the data type mapping provided to `read_csv()`.

1.1.3 Use Twitter API to enrich the core data set

Next we want to use the Twitter API to retrieve the original tweets, so that we can enrich our enhanced tweets data with the missing attributes previously dientified (`retweet_counts`, `favorite_counts`).

Having registered with Twitter as a developer, and obtained credentials and keys, we stored these in a private project directory and configuration file (which are excluded from our git repo, and thus won't be visible online in github).

We now use those credentials to authenticate with Twitter for API access.

Next we will load the enrichment data in batches, for better performance, as API invocations are subject to significant network latency. Twitter also applies rate limiting to their APIs, so it is necessary to throttle the rate at which we make requests, and to retry any failed requests. Luckily, this can be handled automatically by the Tweepy library, by setting the `wait_on_rate_limit_notify` flag when configuring API connection.

Although we are enriching the core enhanced tweets archive, we will initially load the API data into a separate data frame, cleanup as necessary, and then merge into the main table.

Next we review the structure of the merged data frame. In particular, the number of rows should be unchanged (as you'd expect given it is a left join), and the 2 additional columns have been added on at the end, with some NA values reflecting tweets that have never been retweeted or favorited.

1.1.4 Read in and normalise image data set

Finally we need to load the image predictions data, so we can tidy it. We will read this data from the CloudFront URL, a opposed to the local file, to ensure we get the most up-to-date version.

We can now briefly review the structure of this data frame:

1. each row refers to an image
2. each image is numbered, as it is selected as one of up to 4 dog images that may be associated with each tweet
3. we then have the top 3 predictions for the image

Each prediction consists of the following information:

1. a predicted label or class (e.g.: the dog breed) tht describes that image
2. a confidence factor associated with the previous prediction, in the range 0.0 -> 1.0
3. a boolean indicator confirming if the predicted label is a dog breed, or some other object

Looking at the confidence factors for predictions p1 - p3, they appear to be listed in most confident to least confident order. Therefore we will use the column name numeric suffix to generate a ranking column, which we can later sort by (to preserve this decreasing confidence order).

This last attribute confirms that the image classifier used to generate these prediction was trained on a broad set of images, only a subset of which are dog images labelled with their corresponding dog breed. But on occasions the classifier may have interpreted a dog image as an object other than a dog.

1.2 ASSESS DATA

We have already assessed the core tweet data, and fixed some issues at load time. Lets now look at data quality and structural issues.

1.2.1 Remedy data quality issues

Some of the issues we now want to remedy are:

1. dog names and stages are extracted when found, otherwise the value *None* is used
2. dog stages are predefined, so storing the stage name into the relevant stage columnis redundant information, all we require is a binary marker
3. the 'source' column is an HTML anchor with a link to http://twitter.com/download/iphone which is repeated for all rows, therefore we will drop the `source` column

So next we convert those dog stage columns into a boolean data type.

1.2.2 Remedy structural issues

The `expanded_urls` column, which stores the full length URL for shortened URLs that appear in the tweet, has a couple of issues, both quality and structure:

1. it can store multiple URLs per cell, as a comma separated string
2. the same URL can appear multiple times

As looking at the tweet text it is not obvious why the same URL can appear more than once, instead of storing a repetition count, we will just drop any duplicates. Since these URLs can now have a many-to-one relationship with the tweet in which they appear, we will store them in a separate data frame.

Finally we will tidy up the image predictions data. By applying the Tidy Data principles, we will remove variables (the prediction number) from the relevant column names.

We are repeating the same image URL and image number, on each of the predictions, just for simplicity (the alternative is to split the image data into 2 data frames).

As we are removing variables from the column names, we will end up with more rows. Since all top 3 predictions are always generated, we will have exactly 3 times the number of rows we started with, as reflected by the row counts above and below.

1.3 CLEAN DATA

We have already performed a few cleaning tasks, including correcting data types and eliminating redundant data.

1.3.1 Address specific data cleaning requirements

Next we will clean up the data as specified under the section **Key Points**, in the Project Motivation page, specifically:

1. We drop retweets, by definition these are tweets where the `retweet_status` fields are populated
2. Since none of the remaining rows are retweets, any column related to retweets will only hold NA and are now redundant, so we drop these columns
3. We drop columns without images, i.e.: where the `tweet_id` does not appear in the images data frame

1.3.2 Keep persistent copy of wrangled clean data

As there appear to be no tweets beyond August 1st, 2017 (most likely since we dropped tweets without images) we are now done with cleaning. We end the wranggling process by writing the clean data to a new set of clean data files.

1.4 Data insights and visualisation

In this section we look at the data and query it to obtain some insights. Specifically, we are interested in:

1. Finding the number of tweets with a score above 10/10, versus tweets with a score under 10/10
2. Identify the tweets where more than one dog stage appears
3. Finding the number of top breed predictions from the image classifier, with a predictionconfidence below 0.5

Count number of scores above and below 10/10

```
(1152, 835)
```

Show tweets with more than one dog stage in the tweet text

```
                    text  \
tweet_id
733109485275860992
Like father (doggo), like son (pupper). Both 12/10 https://t.co/pG2inLaOda
741067306818797568
This is just downright precious af. 12/10 for both pupper and doggo
https://t.co/o5J479bZUC
751583847268179968                              Please stop sending it pictures
that don't even have a doggo or pupper in them. Churlish af. 5/10 neat couch tho
https://t.co/u2c9c7qSg8
759793422261743616                      Meet Maggie &amp; Lila. Maggie is the
doggo, Lila is the pupper. They are sisters. Both 12/10 would pet at the same
time https://t.co/MYwR4DQKll
785639753186217984  This is Pinot. He's a sophisticated doggo. You can tell by
the hat. Also pointier than your average pupper. Still 10/10 would pet
cautiously https://t.co/f2wmLZTPHd
801115127852503040                      This is Bones. He's being haunted by
another doggo of roughly the same size. 12/10 deep breaths pupper everything's
fine https://t.co/55Dqe0SJNj
802265048156610565
Like doggo, like pupper version 2. Both 11/10 https://t.co/9IxWAXFqze
808106460588765185                      Here we have Burke (pupper) and Dexter
(doggo). Pupper wants to be exactly like doggo. Both 12/10 would pet at same
time https://t.co/ANBpEYHaho
817777686764523521  This is Dido. She's playing the lead role in "Pupper Stops
to Catch Snow Before Resuming Shadow Box with Dried Apple." 13/10 (IG:
didodoggo) https://t.co/m7isZrOBX7
854010172552949760  At first I thought this was a shy doggo, but it's actually a
Rare Canadian Floofer Owl. Amateurs would confuse the two. 11/10 only send dogs
https://t.co/TXdT3tmuYk
855851453814013952        Here's a puppo participating in the #ScienceMarch.
Cleverly disguising her own doggo agenda. 13/10 would keep the planet habitable
for https://t.co/cMhq16isel

                    doggo  floofer  pupper  puppo
tweet_id
733109485275860992   True    False    True  False
741067306818797568   True    False    True  False
751583847268179968   True    False    True  False
759793422261743616   True    False    True  False
785639753186217984   True    False    True  False
801115127852503040   True    False    True  False
802265048156610565   True    False    True  False
808106460588765185   True    False    True  False
817777686764523521   True    False    True  False
854010172552949760   True     True   False  False
855851453814013952   True    False   False   True
```

Count tweets where the top scoring breed prediction is below 0.5

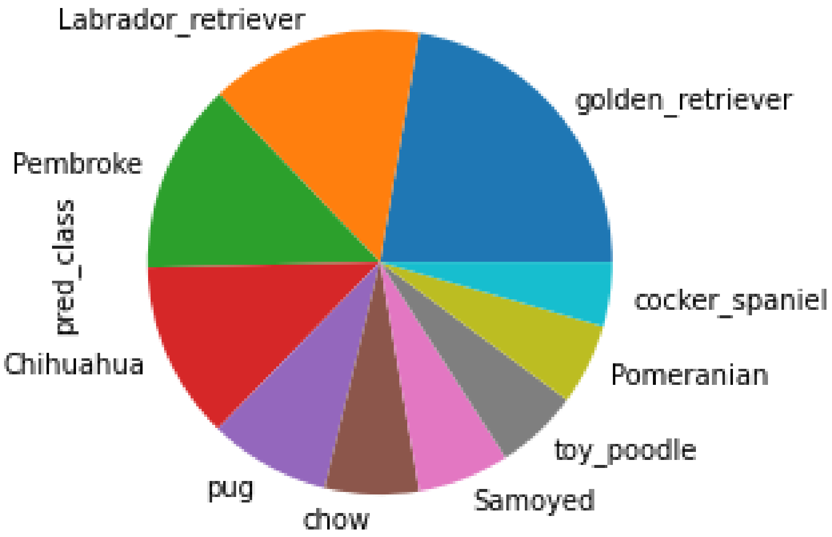
```
559
```

Now we are going to generate some visualisations:

1. First, based on the top image prediction, look at the frequency distribution for the top 10 breeds only, based on number of tweets
2. Now look at the frequency distribution for the top 10 breeds only, based on aggregate number of favorites

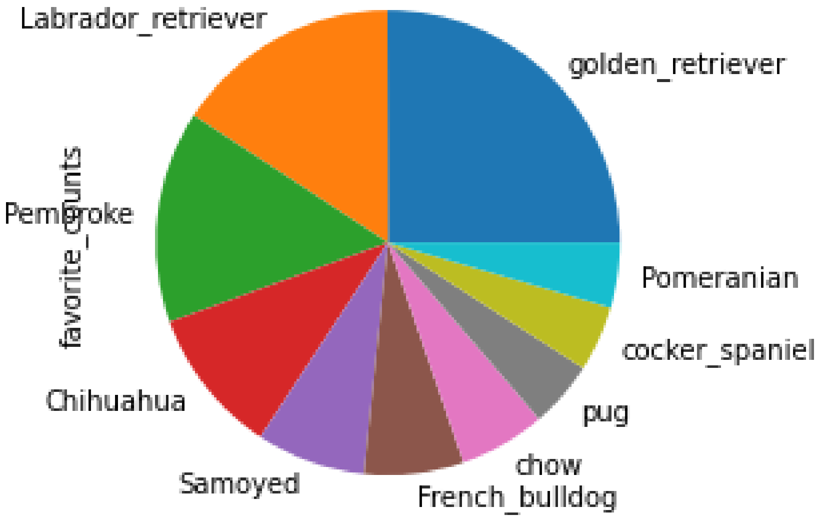
Distribution by number of tweets

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffd40552c50>
```



Distribution by number of favorites

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffd715d3810>
```



1.4.1 Generate internal report

Having cleaned the data, and generated data insights, we can now generate the internal documentation from this notebook's markdown cells.

(you probably want to clear all output previous to the data insights output generated in the last section, and then SAVE the notebook)

```
[NbConvertApp] Converting notebook wrangle_act.ipynb to pdf
[NbConvertApp] Support files will be in wrangle_act_files/
[NbConvertApp] Making directory ./wrangle_act_files
[NbConvertApp] Making directory ./wrangle_act_files
[NbConvertApp] Writing 37997 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 82271 bytes to wrangle_act.pdf
```