The HCAN project

Goal ….

# The original HCAN models

tf\_cnn.py …

tf\_hcan.py …

tf\_han.py …

tf\_traditional\_ml.py

These model scripts are referred to as the *original implementations*.

# The Yelp Data

The raw data we consider here consists of approximately 6M Yelp reviews of various businesses and services including bars, restaurants, cleaners, tattoo parlors and sports/hobby venues, etc. Reviews of dental and medical services are very common as well. The Yelp Open Dataset website (ref. <https://www.yelp.com/dataset>) states the purpose for publishing this information.

The Yelp dataset is a subset of our businesses, reviews, and user data for use in personal, educational, and academic purposes. Available as JSON files, use it to teach students about databases, to learn NLP, or for sample production data while you learn how to make mobile apps.

The various datasets described on that page can be found in the Yelp-dataset subdirectory. We will focus on the “reviews” dataset (reviews.json). The format of each JSON entry is described in <https://www.yelp.com/dataset/documentation/main>. Our models are concerned with three features that are guaranteed to appear in each entry.

* *text* – a string of sentences exactly as written by the reviewer. The format of the text is pretty much anything goes. We cannot make assumptions about language, grammar, punctuation, etc.
* *stars* – an integer ranking from 1 to 5 where 1 is the lowest/worst and 5 is the highest/best.
* *date* – the date (in “yyyy-mm-dd” format) of the posting. This is used in the data filtering layer discussed later to reduce the volume of data for those model implementations that process memory-resident data.

## Distribution by year of posting

The review posting dates range from in 2014 up to sometime in 2018. We can see that the number of reviews greatly increases year to year, perhaps indicating the more widespread use and popularity of the app over time. The slight dip in 2018 is possibly the result of incomplete data for that year, i.e. the extraction may have been capped prior to year’s end. The following chart depicts the distribution of the data over all 15 years of available data – 6,685,900 reviews in total according to the Web site, but data cleansing in the TFRecord generator (discussed later) reduces that number to 6,664,886.

|  |  |
| --- | --- |
| **Year** | **Number of reviews** |
| 2018 | 1,177,662 |
| 2017 | 1,217,673 |
| 2016 | 1,098,786 |
| 2015 | 952,400 |
| 2014 | 704,862 |
| 2013 | 491,294 |
| 2012 | 367,090 |
| 2011 | 302,867 |
| 2010 | 187,387 |
| 2009 | 101,173 |
| 2008 | 57,347 |
| 2007 | 21,389 |
| 2006 | 5,081 |
| 2005 | 876 |
| 2004 | 13 |

Table 1. Number of reviews available by year

# Review document preparation

In the original implementations the desired Yelp data subset must be extracted and preprocessed by feature\_extraction\_yelp.py (the *preprocessor* going forward). The preprocessor converts the review text to lower case, isolating sentence delimiters including periods, question marks and exclamation points, and transforms that result into a list of sentences, each consisting of a list of words with the sentence delimiters removed.

The preprocessor then generates two files in the data subdirectory subsequently “imported” by the original models. The first is yelp\_data.pkl, a python pickle file that contains a dictionary-of-dictionaries encapsulating all of the selected Yelp reviews in both text and digitized formats, plus the “star” rating, the label for supervised learning. The digitized text is derived from the second output, the word “embedding matrix” generated by the ginsim.models Word2Vec function.

The key of each entry of the former is simply the review record number / order of occurrence – a sequence of integers ranging from 0 to N. The corresponding values are dictionaries containing the keys: *text*, *idx*, and *label*. *text* is an array of sentences, each sentence an array of words in natural language. idx is an array of array of integers, isomorphic to *text*. The dictionary-of-dictionaries is later replaced by a simpler array-of-dictionaries as discussed under TFRecords below.

The second file, yelp.embeddings, is produced by Word2Vec based on the vocabulary captured in yelp\_data.pkl These “embeddings” are used to construct CONV1D layers in the various models.

The preprocessor extracts only those reviews from a particular year – the comment says “to reduce data size”. Currently, that year is 2016 (hard-coded) exposing 1,098,786 examples. The governor makes sense because the entire dictionary-of-dictionaries is memory-resident in the current implementations.

tf\_cnn.py, tf\_hcan.py and tf\_han.py each operate on the two files produced by the preprocessor. These model scripts contain logic to partition the pickled review documents into training and test sets.

# Tensorflow Estimators and the tf.data.Dataset API

The Cerebras CS-1 system can currently execute (train, validate, predict) Tensorflow models written to the Estimator API, acquiring its input via the tf.data.Dataset API backed by disk-resident TFRecord datasets. You can find introductory material here: https://www.tensorflow.org/guide/estimator

Migrating the input source of the HCAN models from a pickled memory-resident dictionary structure to TFRecords has the additional benefit that the input examples can grow to any arbitrary size.

The following paragraphs will discuss how we replaced the preprocessor with a similar utility called make\_tfrecords.py that generates embedding files using Word2Vec as before, but creates TFRecord datasets containing the training and validation examples rather than a python pickle file.

# The HCAN TFRecord generator

make\_tfrecords.py is an extension of the original feature\_extraction\_yelp.py. Like the latter, it generates examples in a format readily consumed by the model scripts (or nearly so). However, the TFRecord examples can be read piecemeal and batched and there is no practical limit to the size of an example dataset.

## HCAN TFRecord format

Individual TFRecords are essentially dictionaries associating a value or array of values with each key. The HCAN TFRecord keys are:

* *data* – an array A[] of int64, where each entry is the digitized value of the corresponding word in the review text. All of the sentence delimiters have been removed. The tf\_cnn model will process this feature directly. No sentence awareness is needed.
* *breaks* – an array A[] of int64, where each entry A[i] is offset to the end of the sentence beginning at A[i-1]. The tf\_hcan model is sensitive to sentence structure. It will reconstruct sentences using this feature together with *data*.
* *label* – an array A[5] of int64 which is a one-hot encoding of the stars rating of one through five.

I was not able to find a technique for simply representing the text as an array of sentence arrays. TFRecords support feature nesting, but each record must have an identical nesting structure as best as I can tell. Representing the text and sentence boundaries using two separate, flat vectors seemed like a reasonable choice. I found the following supporting post on stack overflow.

<https://stackoverflow.com/questions/53125730/how-to-save-list-of-lists-of-varying-length-in-a-tfrecord>

The *label* is a one-hot array. The original model implementations all convert the *stars* rating to this format so it is done here in advance as a simplification.

The int64 datatypes might appear to be overkill but that is the only integer datatype supported by TFRecord construction methods.

The convert\_to\_tfr function writes separate train and test files based on the split= argument currently hard-coded as 80/20. This should be parameterized. This approach is different than that taken in the original models. In those scripts, the example data is split into train, validation and test sets – something I have not implemented in the TFRecord versions.

# Better understanding the Yelp data

The –print\_samples argument was added to the TFRecord generator so that I could get a better idea of what the reviews look like, organized by rating/stars then length. When specified, reviews are categorized by their length in increments of 100 words. For each rating the text of a single review having a unique length increment is captured and printed at the end of TFRecord generation. Examination of those samples was enlightening. Sample output is provided as an attachment.

## Languages

I noticed that a fair number of reviews are written in French. English dominates, followed by French. I was not sure how non-English entries would affect the embeddings and/or the training and validation. I added specific traps for French and German language vocabulary and discarded suspect records. This is very crude but reasonably effective. In a pass over all of the available Yelp data 19,838 French reviews and 1,120 German reviews were discarded.

I then (just recently) attempted to identify non-English, rather than targeting specific languages where I had some knowledge of basic vocabulary. I tried to find examples that did NOT contain any of a set of common English words. The bad news is that a large number of very terse English reviews were mistakenly flagged. However, it did turn up a fairly substantial number of entries in Chinese and Japanese, and a smattering in Spanish and Dutch. I believe that number of combined Chinese and Japanese entries are significant, far exceeding the number of German reviews but certainly fewer that those in French.

I have not pursued this any further. But it occurs to me that if we could reliably identify and classify the non-English entries, we could translate and include rather than simply exclude. That might be a fun project for someone with a lot of time on their hands.

## Rating / stars distribution

The following table depicts the distribution of all available reviews over their associated rating, the number of stars (1 – 5) awarded by the reviewer.

|  |  |  |
| --- | --- | --- |
| **Rating (stars)** | **Number of reviews** | **Percent of total** |
| 1 | 1,000,854 | 15 |
| 2 | 540,833 | 8 |
| 3 | 735,349 | 11 |
| 4 | 1,461,258 | 22 |
| 5 | 2,926,592 | 44 |

Table 2. Distribution by rating/stars over ALL reviews

We can readily see that there is significant skew in the data. Only 23% of the reviews are negative and of those only 8% of those are “not so good”. This doesn’t surprise me. Many consumers are probably not motivated to take the time to write a review unless they have strong feelings, either positive or negative. The distribution of ratings over a subset of the examples, notably those from 2016 only (1,098,783 examples), suggest that this skew permeates the data. This subset is particularly important because it is what the original models process and will be used for base line comparisons with the new implementations.

|  |  |  |
| --- | --- | --- |
| **Rating (stars)** | **Number of reviews** | **Percent of total** |
| 1 | 179,830 | 16 |
| 2 | 85,036 | 8 |
| 3 | 104,068 | 9 |
| 4 | 207,160 | 19 |
| 5 | 522,689 | 48 |

Table 3. Distribution by rating/stars 2016 reviews

## Sentence demarcation

The original preprocessor “normalizes” the input reviews by translating text to lower case, removing quoted strings and isolating sentence delimiters. tf\_cnn operates on word sequences only and is not concerned with sentence boundaries. tf\_hcan, however, operates on sentences so we presume it is more important to do the segmentation accurately. As mentioned previously, there are a significant number of documents that discuss medical/dental doctors and facilities. Tracing thru the parse of several examples I saw that a sentence like ‘I think Dr. Smith is one of the best…’ became two sentences: ‘I think dr’ and ‘smith is one of the best…’. Fortunately, this was very easy to remedy by expanding the text substitution operations demonstrated in the preprocessor. The TFRecord generator translates common abbreviations to tokens that do not prematurely terminate the sentences they occur in. It isn’t perfect but seems to work well most of the time. Other typographical substitutions are aimed at expanding contractions and contracting possessives. All substitutions are made in context. Some require leading, trailing white space, or both.

|  |  |  |
| --- | --- | --- |
| **Abbrev** | **Replacement** | **Notes** |
| dr. | dr | Lots of these |
| vs. | vs |  |
| mr. | mr |  |
| mrs. | mrs |  |
| ms. | ms |  |
| inc. | inc |  |
| llc. | llc |  |
| ltd. | ltd |  |
| approx. | approx |  |
| appt. | appt |  |
| apt. | apt |  |
| i.e. | ie |  |
| e.g. | eg |  |
| p.s. | ps |  |
| p.s | ps |  |
| a.m. | AM | Hours of operation, etc. |
| p.m | PM | Hours of operation, etc. |
| ‘re | ‘ are ‘ | we’re, you’re, they’re |
| (s) | s | A bunch, surprisingly |
| ` | ‘’ | joe`s |
| - | ‘’ | e-mail, etc. |

Table 4. Sentence cleansing

## Review length

A Yelp review document, like any document, is a sequence of sentences, and each sentence is a sequence of words. Some models, like tf\_hcan and tf\_han , consume inputs in the form of a fixed-size sentence matrix, where each row contains the words of a single sentence padded as necessary, up to row maximum. The original implementations determine these dimensions by choosing the maximum words-per-sentence and sentences-per-review encountered in the data. Thus, each reformatted example is large and sparse (1245, 681). The impacts on training time are discussed later.

We see from that distribution below, reviews of one to 15 sentences dominate (87% of total). Sampling some of the outliers I found that some reviewers like to punctuate every word as in “I! love! this! place!”, or add punctuation going for the dramatic affect as in “I. l.o.v.e. t.h.i.s. p.l.a.c.e. !.!.!.!.!.!).

I propose that we implement a –max\_sentences (maximum sentences per review) argument to the TFRecord generator to optionally specify a limit, a complement to the –trunc (maximum words per sentence) argument discussed below. (The question is “which sentences” to include… the first, last, middle, etc.)

|  |  |  |
| --- | --- | --- |
| # Sentences per Review | All Available Data | 2016 Data |
| 0 - 4 | 2,001,690 | 354,954 |
| 5 - 9 | 2,712,612 | 454,976 |
| 10 - 14 | 1,054,497 | 169,181 |
| 15 - 19 | 448,901 | 77,827 |
| 20 - 24 | 209,115 | 29,164 |
| 25 - 29 | 104,134 | 7,259 |
| 30 - 34 | 55,494 | 1,423 |
| 35 - 39 | 30,951 | 251 |
| 40 - 44 | 18,489 | 42 |
| 45 - 49 | 11,049 | 6 |
| 50 - 54 | 6,835 | 3 |
| 55 - 59 | 4,324 | 2 |
| 60 - 64 | 2,726 | 2 |
| … | etc | etc |

Table 5. Number of sentences per review

The following tables summarize the number of words-per-sentence found in the 2016 review extract.

|  |  |
| --- | --- |
| # Words per Sentence | # Sentences |
| 0 - 19 | 6557164 |
| 20 - 39 | 1568895 |
| 40 - 59 | 99768 |
| 60 - 79 | 12383 |
| 80 - 99 | 3247 |
| 100 - 119 | 1189 |
| 120 - 139 | 560 |
| 140 - 159 | 274 |
| 160 - 179 | 216 |
| 180 - 199 | 111 |
| 200 - 219 | 73 |
| 220 - 239 | 38 |
| 240 - 259 | 19 |
| 260 - 279 | 21 |
| 280 - 299 | 17 |
| 300 319 | 25 |

## Metadata

The TFRecord generator produces a JSON file summarizing the training and validation datasets it creates. Here, reformatted is the all\_available-HCAN-metadata.json file produced when processing the entire review dataset.