Report

1. Datasets, general statistics:

topics:

	id	name
0	7	Electronics & Computers
1	8	Toys, Kids & Baby
2	5	Beauty, Health & Grocery
3	9	Automotive & Industrial
4	3	Clothing, Shoes & Jewelry (Fashion)
5	4	Movies, Music & Games
6	1	Books & Audible
7	2	Home, Garden & Tools
8	6	Sports & Outdoors

lists_topics_sample – to training model*:*

Overview

Dataset info

Number of variables	3
Number of observations	1982
Total Missing (%)	0.0%
Total size in memory	61.9 KiB
Average record size in memory	32.0 B

Variables types

Numeric	2
Categorical	1
Date	0
Text (Unique)	0
Rejected	0

Warnings

- `list_name` has a high cardinality: 1978 distinct values Warning

sample for pred – to predict topic_id*:*

Overview

Dataset info

Number of variables	2
Number of observations	485
Total Missing (%)	0.0%
Total size in memory	11.4 KiB
Average record size in memory	24.0 B

Variables types

Numeric	1
Categorical	1
Date	0
Text (Unique)	0
Rejected	0

Warnings

- `list_name` has a high cardinality: 421 distinct values Warning
- Dataset has 63 duplicate rows Warning

2. Basic analysis

The first idea is to compare the words in the *list_name* for train dataset and predict dataset in order to see if it is the similar product namings.

The basic analysis shows that number of unique words in train dataset is 1843, and number of unique words in dataset to predict is 626. The number of common words is 354. It shows that there are a lot of different words between training and predicting data. It leads to the conclusion that information in given datasets is not enough to make efficient and correct predictions.

3. Model

It was decided to use vector representations for all the words in the *list_name* which would help to find similarities between different words. In general, with the task to predict topic for the product it makes good sense to find similar by context words.

That is why the next steps were covered:

- pre-trained word vectors were loaded at http://nlp.stanford.edu/projects/glove/ (Wikipedia 2014 + Gigaword 5)
- 300 dimensional vectors were chosen for analysis, and were loaded to Python from .txt file
- some words from train and predict data were not found in loaded corpus of words, so they were transformed to the most close words that exist in the corpus: 1st – plural converted to singular, 2nd – *difflib* used to find approximated words (two similar words, and result vector is an average of two vectors)
- when we get 300d vectors for all the words in train and predict *list_name* we calculate 300d vectors for list_names (phrases) by averaging between words from the *list_name*
- when we have 300d vectors for all the list_names we started to use cosine similarity to find most similar for each in predict list_name dataset among all the list_names in train dataset
- we chose to find 15 closest list_names for each predict list_name, and summing up distances by topics to see which topic is the most influential
- some results:

```
Best predictions for  "Pocket Projectors"
{Home, Garden & Tools: 0.53006689353283709,
Electronics & Computers: 7.7825366768783955}

Best predictions for  "Hard Drive Enclosures"
{Toys, Kids % Baby: 0.51570887883784244, 9: 0.95973683704673951, Home,
Garden & Tools: 1.0786450708662851, Electronics & Computers:
5.605584031786945}

Best predictions for  "Herb Grinders"
{Home, Garden & Tools: 5.9703077296338254,
Beauty, Health: 0.68274063725665246}
```

Other results can be find in ipython notebook.

- was decided to repeat similar procedure, but to find vectors for the words from *topic_name*, then for each *list_name* from predict dataset find the most similar topics – direct way, without similarities between list_names. As a result, there are differences in predictions between these two approaches, from ~500 predictions only ~200 predictions are the same – look at the **Predicted topics.png**. However, visualizations show that similarities between list_names are much higher on average, than similarities between predicting list_names and

topic_names – look at the **Distribution of closest distances.png**. That is why 1st approach should be more consistent and reliable.

4. To improve the predictions we could divide train data on train and validate data, and try to see how we can improve our predictions by changing some of hyper parameters in our solution, e.g.
   - change number of closest neighbors to average vectors between them
   - instead of averaging vectors take some weighted linear average with different weights,
   - change dimensionality of vectors (300d should be still enough), etc.

   Also, more training data would improve our predictions of course.