
ProVe - finding item replacements during inventory shortages

Experimental Protocol for XCS224U Spring 2020

Andrei Damian
Lummetry.AI
email: andrei.damian@lummetry.ai

April 16, 2020

1 The Hypothesis

In past years we have seen a lot of cross-domain applications from various deep learning areas to particular real-life cases with apparently little connection. One of those areas is that of applying deep representation learning based on neural language processing to business analytics systems and recommender systems in particular. The intuition behind our proposed deep learning pipeline is that we could, at least in theory, generate, through multiple modeling iterations, powerful-enough semantic vector space embeddings for each individual item (product) so that we can infer replacements items and propose them in the case on original product shortages – all of these in a self-supervised setting. Employing the same self-supervised approach and proposed experiment setting we are proposing a secondary generative model. This secondary objective is aiming at providing a cold-start for new products that are introduced in the inventory and, as such, do not have a transactional history and thus any kind of patterns.

The hypothesis is that we can apply post-processing fine-tuning after trained product embeddings - *ProVe* - that have been generated with GloVe [1] in a similiary way as Grbovic et al [2] use word2vec [3] in their *prod2-vec* [2] work. The fine tuning process will be either based on retrofitting or on actual re-construction of the vector space with new co-occurrence matrices in order to generate a product vector space model able to generate both item-replacement information as well as new embeddings for cold-start-ing items in inventories.

More precisely the hypothesis is that our proposed approach will reduce the distance between products that can actually replace each other in real life, while “pushing” apart dissimilar products in a similar manner as presented in the work of Faruqui et al [4], Mrkšić et al [5] and in the more recent work of Dingwall et al [6] and Lengerich et al [7] . Although the before mentioned work addresses natural language processing & understanding, we will assimilate product/items with words as well as other NLP/NLU objects with retail/business concepts.

2 The Data

For our experiment we decided to use a real-life transactional dataset that contains more than 2M transactions of a retailer with various locations over a period of more than 3 years. Although the dataset contains more than 15,000 different products in order to speed-up experiment time using full in-GPU training we reduced the number of products to a maximum of 13,000 top sold products.

Further information on the data can be observed in the data loading, preparation and minimal visualization of the experiment.

```
In [1]: # We load the required packages and setup the notebook environment
import numpy as np
from scipy import sparse
import itertools
import os
import pandas as pd
from datetime import datetime as dt
from time import time
import textwrap
from itertools import combinations
import matplotlib.pyplot as plt
%matplotlib inline
```

We prepare the majority of our global variables

```
In [25]: print_df = False
DEBUG = True # in-development flag
MODEL_NAME = 'exp_v1'
DATA_HOME = 'exp_data'
MODEL_HOME = 'exp_models'
DATA_FILE = os.path.join(DATA_HOME, 'df_tran_proc_top_13.5k.csv')
DATA_SLICE_FILE = os.path.join(DATA_HOME, 'df_tran_proc_top_13.5k_slice.csv')
META_FILE = os.path.join(DATA_HOME, 'df_items_top_13.5k.csv')
META_INFO = os.path.join(DATA_HOME, 'obfuscated_keys.txt')
MCO_OUT_FILE = os.path.join(MODEL_HOME, 'exp_mco.npz')
EMB_OUT_FILE = os.path.join(MODEL_HOME, MODEL_NAME + '_embeds.npy')
EMB128_10k = os.path.join(MODEL_HOME, 'exp_v2_embeds.npy')
EMB128_250k = os.path.join(MODEL_HOME, 'exp_v1_embeds.npy')
CHUNK_SIZE = 100 * 1024 * 2 # read 100MB chunks
MAX_N_TOP_PRODUCTS = 13000 # top sold products
_MCO_FILE = os.path.join(MODEL_HOME, 'mco_top_13.5k.npz') # debug pre-prepared mco
```

2.1 Data processing approach

Due to the size of transactional databases that often have billions of transactions each with many individual items we generate the proposed matrix of co-occurrence (MCO) with efficient batch reading of the transactional data.

For the purpose of efficiently processing the transactional records we use sparse arrays and a few helper functions within the `prove.py` module. We load our ProVe framework as well as other utility functions and pretty-prints settings for `Pandas` and `NumPy`. In order to ensure experiment replicability for all iterations we have setup a simple logging process (the class `Log`). All the source code in the current development stage for the `prove.py` module can be found at <https://github.com/andreionutdamian/cs224u/blob/develop/prove.py>

```
In [3]: from prove import Log
from prove import load_cats_from_json
from prove import generate_sparse_mco
from prove import show_neighbors
from prove import filter_cats
from prove import show_cats
```

```

from prove import ProVe
from prove import EmbedsEngine

In [29]: pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_colwidth', 500)
pd.set_option('display.width', 1000)
pd.set_option('precision', 4)
pd.set_option('display.float_format', '{:.3f}'.format)
pd.set_option('display.notebook_repr_html', True)

def _repr_latex_(self):
    str_latex = ''
    str_latex += '\\begin{center}\\n'
    str_latex += '\\scriptsize\\n'
    str_latex += self.to_latex(
        bold_rows=True,
        index_names=False,
    )
    str_latex += '\\end{center}\\n'
    return str_latex

pd.DataFrame._repr_latex_ = _repr_latex_ # monkey patch pandas DataFrame
np.set_printoptions(precision=3)
np.set_printoptions(suppress=True)
np.set_printoptions(linewidth=500)
plt.style.use('ggplot')
from IPython.display import Markdown, display
def printmd(string):
    display(Markdown(string))
log = Log(printmd)

```

2.2 Understanding the data

The proposed experimental real-life data comes within a few files generated by SQL commands and exports from an existing ERP system of our retailer. The most notable data files are the transactional database file and the metadata file. The metadata information taken directly from a real-life production system (ERP) contains raw information minimally describing each product-SKU IdemId with product name (ItemName) and other information such as number of item sales in observed in the selected period, a unique sequential item identifier (IDE) as well as as hierarchy information in two fields Ierarhie1 and Ierarhie2 that will be further used as a knowledge graph similar to WordNet [8]. There is also a secondary de-obfuscation data-source that contains for each hierarchy identified the actual name of that category. This information can also be used as a source for self-supervision in the process of creating the knowledge graph for the fine-tuning of our semantic vector space model.

The real-life transactional dataset, as previously mentioned, contains over 6M observations for over 2M different transactions. Each observation in the transactional database contains a BasketId identifier of the transaction as well as individual transaction detail information such as ItemId (and its counterpart IDE), a SiteId field that identifies the location of that particular transaction, a TimeStamp time-identification that is basically the same for all items of the same transaction, a quantity field, a customer identifier field and a product availability indicator.

For the metadata we load the whole file and analyse its content while for the transactional database we take a quick look at the first chunk of 15 rows

```

In [5]: def show_product(IDE, df):
rec = df[df.IDE==IDE].iloc[0]
log.P("    Name: {}".format(rec.ItemName))
log.P("    Id: {}".format(rec.ItemId))

```

```

log.P("  Freq: {}".format(rec.Freq))
log.P("  Hrchy: {}/{}".format(rec.Ierarhie1, rec.Ierarhie2))
return

log.P("Metatada information:")
df_meta = pd.read_csv(META_FILE)
df_meta = df_meta[df_meta.IDE < MAX_N_TOP_PRODUCTS]
log.P("  Total no of products: {}".format(df_meta.ItemId.unique().shape[0]))
log.P("  Total no of level 1 hierarchies:
{}".format(df_meta.Ierarhie1.unique().shape[0]))
log.P("  Total no of level 2 hierarchies:
{}".format(df_meta.Ierarhie2.unique().shape[0]))
log.P("  Most sold individual product over period:")
show_product(df_meta[df_meta.Freq == df_meta.Freq.max()].iloc[0]['IDE'], df_meta)
log.P("  Least sold individual product over period:")
show_product(df_meta[df_meta.Freq == df_meta.Freq.min()].iloc[0]['IDE'], df_meta)
df_meta[df_meta.ItemName.str.len()<30].iloc[:15,:-1]

```

Metatada information:

```

Total no of products: 13000
Total no of level 1 hierarchies: 24
Total no of level 2 hierarchies: 181
Most sold individual product over period:
Name: YUVAL NOAH HARARI / SAPIENS. SCURTA ISTORIE A OMENIRII
Id: 545535
Freq: 21248
Hrchy: 1/17
Least sold individual product over period:
Name: DAVID GROSSMAN / UN CAL INTRA ÎNTR-UN BAR
Id: 442552
Freq: 156
Hrchy: 1/0

```

	ItemId	IDE	Freq	ItemName	Ierarhie1	Ierarhie2
2	406083	2	13371	FELICITARI A 7331335123458	11	107
6	258901	6	8341	TURTA DULCE TIP INIMIOARE	20	269
13	527499	13	7676	TIBI USERIU / 27 DE PASI	1	9
15	493855	15	7354	IRINA BINDER / FLUTURI VOL. 3	1	0
23	651356	23	6718	SACOSA BIO CARTURESTI	11	100
24	522568	24	6531	IRINA BINDER / INSOMNII	1	9
30	599979	30	6006	MICHELLE OBAMA / POVESTEIA MEA	1	9
34	439986	34	5434	PAULA HAWKINS / FATA DIN TREN	1	0
40	258895	40	5143	TURTA DULCE POM DE CRACIUN	20	269
41	258898	41	5122	TURTA DULCE MOS CRACIUN MIC	20	269
46	355291	46	5061	SABOTEUR / 8+ / 747496	4	181
48	258361	48	4998	M35 WINTER COCKTAIL 50G	13	79
50	258270	50	4953	M161 MULTI FRUIT 50G	13	79
58	406255	58	4806	SENECA / ALT TIMP NU AM	1	43
59	258417	59	4765	M85 REDBUSH BAKED APPLE 50G	13	79

As we are working with a real-life normed relational database the hyerarchi fields do not tell much information other than a simple int identifier however we can deobfuscate them (Ierarhie1 and Ierarhie2) in our database fields with additional external data-source.

```

In [6]: df_meta, dct_categories = load_catags_from_json(df_meta, META_INFO)
df_meta[df_meta.ItemName.str.len()<30].iloc[:15,[3,7,8]]

```

	ItemName	Ierarhie1_name	Ierarhie2_name
2	FELICITARI A 7331335123458	PAPETARIE	Giftware
6	TURTA DULCE TIP INIMIOARE	GASTRONOMIE	Bakery
13	TIBI USERIU / 27 DE PASI	CARTE ROM	BIOGRAFII / MEMORII
15	IRINA BINDER / FLUTURI VOL. 3	CARTE ROM	FICTIUNE
23	SACOSA BIO CARTURESTI	PAPETARIE	TEMPORAR 1
24	IRINA BINDER / INSOMNII	CARTE ROM	BIOGRAFII / MEMORII
30	MICHELLE OBAMA / POVESTEA MEA	CARTE ROM	BIOGRAFII / MEMORII
34	PAULA HAWKINS / FATA DIN TREN	CARTE ROM	FICTIUNE
40	TURTA DULCE POM DE CRACIUN	GASTRONOMIE	Bakery
41	TURTA DULCE MOS CRACIUN MIC	GASTRONOMIE	Bakery
46	SABOTEUR / 8+ / 747496	JUCARII	CARD GAMES
48	M35 WINTER COCKTAIL 50G	GASTRONOMIE AA	Ceai
50	M161 MULTI FRUIT 50G	GASTRONOMIE AA	Ceai
58	SENECA / ALT TIMP NU AM	CARTE ROM	FILOSOFIE
59	M85 REDBUSH BAKED APPLE 50G	GASTRONOMIE AA	Ceai

```
In [7]: chunk_reader = pd.read_csv(DATA_SLICE_FILE, iterator=True)
        chunk_reader.get_chunk(15).iloc[:, :-2]
```

	BasketId	ItemId	SiteId	TimeStamp	Qty	ClientId
0	7130756	441093	26	2016-01-01 14:49:02.403	1.0	-103
1	7130756	464012	26	2016-01-01 14:49:02.403	1.0	-103
2	7130756	464013	26	2016-01-01 14:49:02.403	1.0	-103
3	7130802	377742	20	2016-01-01 15:49:03.860	1.0	-103
4	7130802	405083	20	2016-01-01 15:49:03.860	1.0	-103
5	7130802	405084	20	2016-01-01 15:49:03.860	1.0	-103
6	7130803	365473	26	2016-01-01 15:49:42.567	1.0	-103
7	7130803	381249	26	2016-01-01 15:49:42.567	1.0	-103
8	7130809	344274	26	2016-01-01 15:53:19.953	1.0	-103
9	7130809	393877	26	2016-01-01 15:53:19.953	1.0	-103
10	7130810	165748	20	2016-01-01 15:54:17.053	1.0	-103
11	7130810	363903	20	2016-01-01 15:54:17.053	1.0	-103
12	7130822	350149	20	2016-01-01 16:05:18.960	1.0	-103
13	7130822	447921	20	2016-01-01 16:05:18.960	1.0	-103
14	7130822	464013	20	2016-01-01 16:05:18.960	1.0	-103

```
In [8]: chunk_reader.close()
```

2.3 NLU meets BPA

In terms of natural language models analogy to Business Predictive Analytics systems, we consider each individual basket as a “sentence” and consider each individual product id as a “word”. Referring to our proposed method of constructing matrix of co-occurrence counts and applying GloVe approach for the vector space representation modeling we have two options:

- using a specific context window size controlled as a model hyperparameter with or without weighting between focal and each individual item;
- considering each transaction as the overall context with no distance weighting between the focal word and the context word;

Regarding the context window it is important to note that for online shopping systems the items ordering, in each individual transaction, is quite important as it is directly correlated to the user journey in the website. As such for the particular case the definition of a hyperparameter that will control the context window size is important. For classic brick-and-mortar retailers the order of each product in the receipt is not important as it is unlikely that it reflects the actual user sequence of actions. In our particular case of the real-life dataset we have a retailer that has over 30 locations and a website that seems to generate less than 10% of the revenue. As a result we decided to consider each transaction as a single-context for the computing of item co-counts.

Now the next logical step is to generate the matrix of item co-occurrence counts

```
In [9]: csr_mco, tran_sizes = generate_sparse_mco(
        DATA_FILE, mco_out_file=MCO_OUT_FILE,
        plot=True, return_counts=True, log=log,
        DEBUG=DEBUG, mco_file=_MCO_FILE)
```

Loading MCO...

Loading raw data...

MCO Processing done in 0.09 min (sparse mat creation: 0.09 min):

Start date: None

End date: None

Max co-occ: 5998.0

Co-occurrence distribution:

```
Counts:  00 6382178 913964 293414 138452 77970 49730 34124 24904 18758
Nr items: 0      1      2      3      4      5      6      7      8      9
```

Transactions size distribution:

```
Counts:  00 00 1210270 484388 194496 86254 41847 21134 11616 6430
Nr items: 0  1      2      3      4      5      6      7      8      9
```

Transactional data structure:

RangeIndex: 5767223 entries, 0 to 5767222

Data columns (total 8 columns):

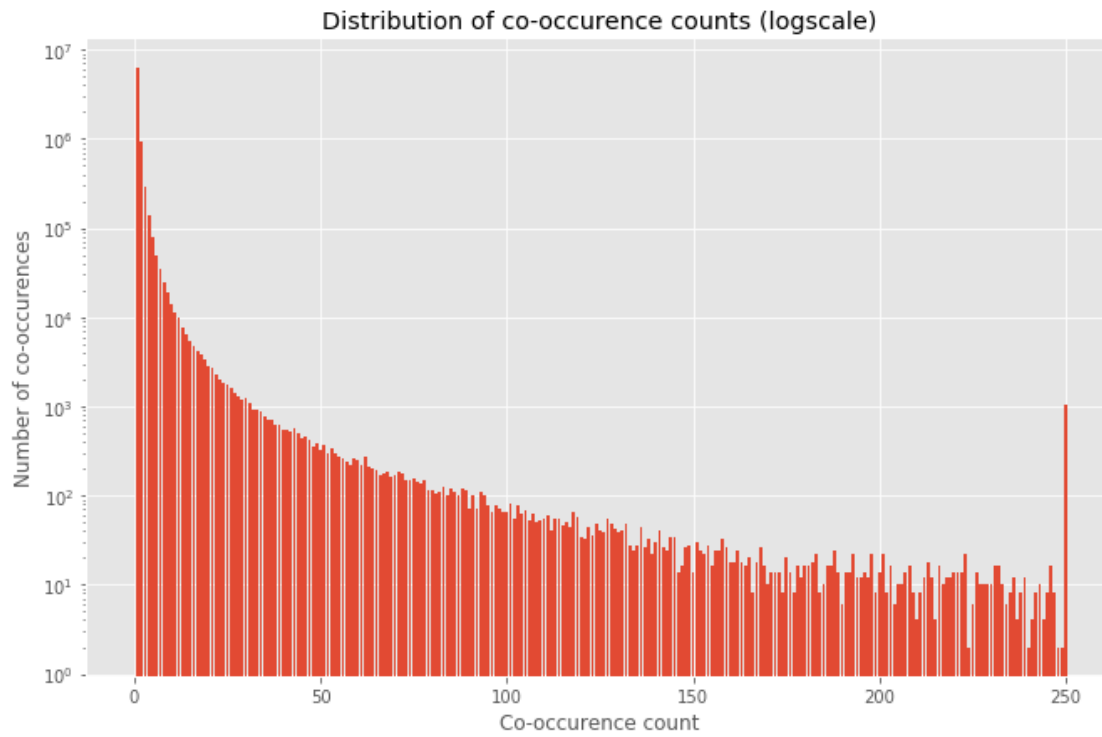
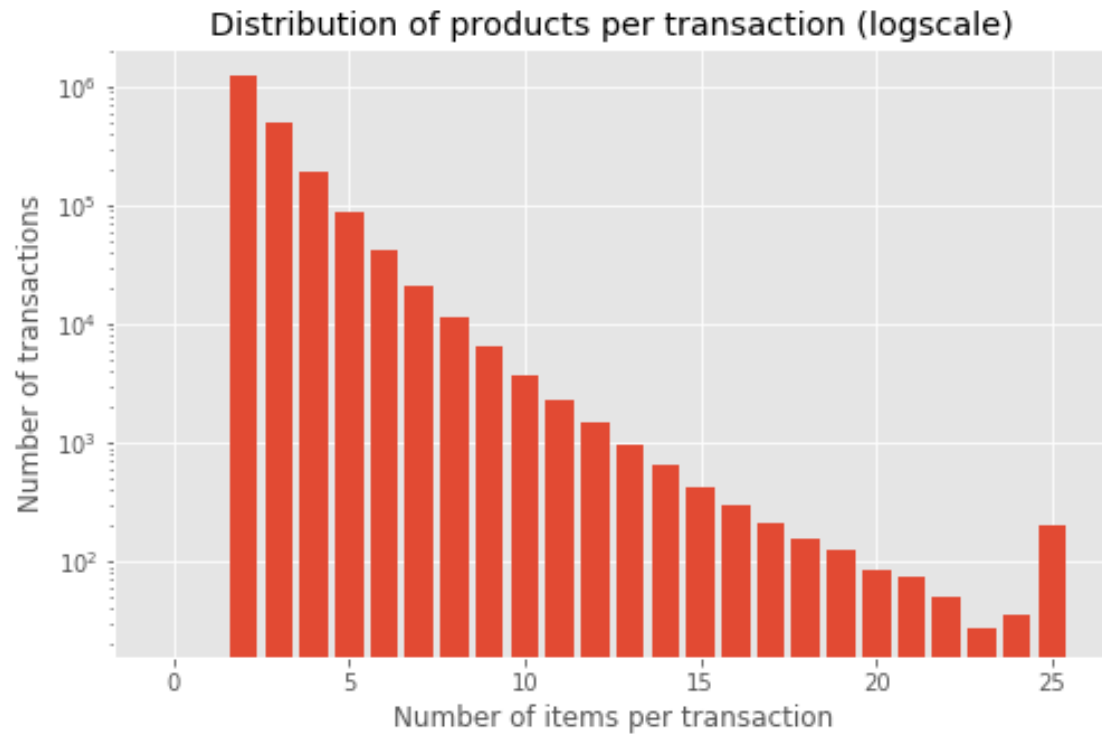
#	Column	Dtype
0	BasketId	int64
1	ItemId	int64
2	SiteId	int64
3	TimeStamp	object
4	Qty	float64
5	ClientId	int64
6	IsActive	int64
7	IDE	int64

dtypes: float64(1), int64(6), object(1)

memory usage: 352.0+ MB

MCO data: (only 7x7 values displayed)

```
[[ 0. 25. 49. 5998. 297. 85. 28.]
 [ 25. 0. 32. 19. 6. 9. 15.]
 [ 49. 32. 0. 22. 8. 15. 28.]
 [5998. 19. 22. 0. 165. 42. 13.]
 [ 297. 6. 8. 165. 0. 141. 11.]
 [ 85. 9. 15. 42. 141. 0. 11.]
 [ 28. 15. 28. 13. 11. 11. 0.]]
```



```
Saving 'exp_models\exp_mco.npz'
```

2.3.1 Meta-information & categories

With regard to the meta-data that can generate important knowledge graph information for our self-supervised approach we found out that the *detail category* - or the so called “level 2 hierarchy” - has a finer granularity than the first level hierarchical information. Further information regarding how this information is processed in order to obtain our knowledge graph will be presented in the modelling section.

```
In [10]: dct_i2n = {k:v for k,v in zip(df_meta.IDE, df_meta.ItemName)}

# we slice and convert the sparse matrix
np_mco = csr_mco.toarray()[:MAX_N_TOP_PRODUCTS,:MAX_N_TOP_PRODUCTS]
df_meta = df_meta.iloc[:MAX_N_TOP_PRODUCTS]
# raw sanity-check
for i in range(np_mco.shape[0]):
    assert i in df_meta.IDE

show_cats(df_meta, dct_categories, k=5, log=log)
```

Hierarchy level 1 'Ierarhie1' contains 32 categories

```
Category name: ACCESORII GASTRO AA   Id:   9  Prods:  523
Category name: ALTELE                 Id:  26  Prods:   9
Category name: CARTE ROM              Id:   1  Prods: 7202
Category name: CARTE STRAINA          Id:   0  Prods: 736
...
```

Hierarchy level 2 'Ierarhie2' contains 356 categories

```
Category name: ACCESORII              Id: 116  Prods:   3
Category name: ACCESORII FASHION      Id: 274  Prods:  21
Category name: ACCESORII MASA         Id: 249  Prods:  26
Category name: ALTELE                 Id:  60  Prods:  60
...
```

2.3.2 Analogy with word-vectors

As previously mentioned, the proposed ProVe model is expected to generate similar results with those of GloVe applied to word-vectors. Just as we load a pre-trained GloVe-100 embeddings matrix with the 400,000 words vector space representations and run a neighbourhood analysis for a word such as “beatles” and obtain “lennon”b, “mccartney” we expect to have similar outputs from our product semantic vector space model.

```
In [11]: if 'wemb' not in globals():
    log.P("Loading GloVe word embeddings")
    glove_words = os.path.join(DATA_HOME, 'glove_words_and_embeds_100d.npz')
    data = np.load(glove_words)
    np_words = data['arr_0']
    wemb = data['arr_1']

def show_word(word):
    show_neighbors(word, wemb, np_words, log=log)

show_word('beatles')
```


Loading GloVe word embeddings

Top neighbors for 9440: 'beatles':

```
beatles:    0.000
lennon:     0.246
mccartney:  0.276
dylan:      0.304
songs:      0.305
recordings: 0.325
albums:     0.326
elvis:      0.326
motown:     0.327
presley:    0.333
```

2.4 Self-supervised training and supervised testing

Although the training process is entirely self-supervised, one particular challenge that was faced was the preparation of test cases and actual evaluation metrics. For the development of the evaluation dataset a classic “supervised” approach has been adopted based on manual data exploration. After manual selection of a limited list of categories based on the available meta information a sample of products has been extracted and analysed. For each of the target selected categories - MUSIC, VYNIL, DVD, Board-games - multiple products have been manually selected. Following the manual selection of product candidates for evaluation data, for each individual chosen item the top neighbors items have been prepared using cosine distance.

```
In [12]: proposed_test_cats = {
        'Ierarhie1': {
            'COMICS&MANGA': 21, 'GASTRONOMIE': 20, 'HOME&DECO AA': 10,
            'LIFESTYLE': 12, 'MULTIMEDIA': 8, 'MUZICA': 2, 'NOVELTY': 23,
            'VINURI': 14,
        },
        'Ierarhie2': {
            'AUDIO & VIDEOBOOK': 259, 'Accesorii apple': 193, 'Activity': 294,
            'Audiobook CD': 111, 'BLU-RAY': 36, 'BLU-RAY 3D': 224, 'BOARD GAMES': 140,
            'Backtoschool': 108, 'Bakery': 269, 'Bath': 262, 'Blu-Ray Audio': 165,
            'Blu-Ray Video': 65, 'Body Care': 330, 'CD Clasica/Opera': 83, 'COMICS': 87,
            'Comics': 120, 'DRAW MANGA/COMICS/VIDEOGAMES': 72, 'FILOSOFIE': 43,
            'Film Blu-Ray': 136, 'Film DVD': 52, 'Film VHS': 343, 'GASTRO/BAKERY': 314,
            'Giftware': 107, 'Gourmet': 192, 'Jazz': 308, 'Kitchen': 118, 'LIFESTYLE': 283,
            'MANGA': 63, 'ROBOTICA': 300, 'SelfHelp': 198, 'TRAVELING COLLECTION': 139,
            'Travel': 243, 'Travel Mug': 144, 'Travelling collection': 75, 'UHD': 183,
            'Vinyl': 10,
        }
    }
dct_cat = filter_cats(
    proposed_test_cats, df_meta, dct_categories,
    max_n_top_products=MAX_N_TOP_PRODUCTS)

show_cats(df_meta, dct_cat, k=1000, log=log)
df_tpr = df_meta[
    df_meta.Ierarhie1.isin(list(dct_cat['Ierarhie1'].values())) |
    df_meta.Ierarhie2.isin(list(dct_cat['Ierarhie2'].values()))
]
```

Hierarchy level 1 'Ierarhie1' contains 8 categories

```
Category name: COMICS&MANGA  Id: 21  Prods: 135
Category name: GASTRONOMIE   Id: 20  Prods: 196
```

Category name:	HOME&DECO AA	Id:	10	Prods:	174
Category name:	LIFESTYLE	Id:	12	Prods:	239
Category name:	MULTIMEDIA	Id:	8	Prods:	411
Category name:	MUZICA	Id:	2	Prods:	219
Category name:	NOVELTY	Id:	23	Prods:	103
Category name:	VINURI	Id:	14	Prods:	23

Hierarchy level 2 'Ierarhie2' contains 14 categories

Category name:	Audiobook CD	Id:	111	Prods:	7
Category name:	BLU-RAY	Id:	36	Prods:	16
Category name:	BOARD GAMES	Id:	140	Prods:	165
Category name:	Backtoschool	Id:	108	Prods:	16
Category name:	Bakery	Id:	269	Prods:	10
Category name:	COMICS	Id:	87	Prods:	3
Category name:	Comics	Id:	120	Prods:	23
Category name:	FILOSOFIE	Id:	43	Prods:	144
Category name:	Giftware	Id:	107	Prods:	95
Category name:	LIFESTYLE	Id:	283	Prods:	11
Category name:	MANGA	Id:	63	Prods:	87
Category name:	Travel Mug	Id:	144	Prods:	66
Category name:	Travelling collection	Id:	75	Prods:	19
Category name:	Vinyl	Id:	10	Prods:	52

3 Metrics and overall evaluation

With regard to metrics and evaluation of this particular experiment it is important to decide how we view the model - as a classic classification or as a ranking problem. As a result the possible metrics that can be applied to our experiment are varied and the options range from the classification metrics such as accuracy, precision, recall, F-score up to recommender systems domain-specific metrics. The initially proposed metrics were `_Recall@K_` that measures the ability of our system to recommend viable items within the first K proposed candidates as well as the more importantly the `_MRR@k_` (Mean Reciprocal Rank) applied only for the first K candidates (if the target is ranked lower than the K then the score is 0).

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{r_i}$$

In the above formula Q represents all the tests performed while r_i represents the rank of the positive candidate (inf if no candidate matches the gold value). Finally we decided to use only MRR as evaluation metric with various small number of candidates ($K=1$ and $K=5$).

```
In [13]: def mrr_k(np_cands_batch, np_gold_batch, k=5):
        """
        will compute Mean Reciprocal Rank for a batch of predictions
        inputs:
            np_cands_batch: [batch, no_products] - the candidates for each experiment in batch
            np_gold_batch: [batch,] or [batch, 1] - the actual gold target product replacement
            k: int (default 5) the max number of candidates evaluated
        """
        assert len(np_cands_batch.shape) == 2
        np_gold_batch = np_gold_batch.reshape(-1,1)
        np_cands_batch = np_cands_batch[:, :k]
        n_obs, n_cands = np_cands_batch.shape
        matches = np_cands_batch == np_gold_batch
        _x = np.repeat(np.arange(1, n_cands+1).reshape(1,-1), n_obs, axis=0)
```

```

_y = np.ones((n_obs,n_cands)) * np.inf
np_res = 1 / np.where(matches, _x, _y).min(axis=1)
return np_res.mean()

```

4 The Models

For this work we decided to take the most straightforward approach at “classifying” viable replacement candidates: that of using cosine distance and take the first 5 candidates (or just the first one) as potential replacement items for a target item. Two main models form the base of our experiment - GloVe [1] in its new *ProVe* form as well as retrofitting techniques derived from the work of Faruqui et al [4], Mrkšić et al [5] and Dingwall et al [6].

4.1 Hyperparameters

The hyper-parameter selection is entirely based on the results of our exploratory data analysis on the real-life dataset combined with the know-how resulted from the natural language counterparts. For the vector space dimensionality we chose 128. We chose a maximum frequency of co-occurrences of 250 although as it can be observed from the distribution of counts we have a maximum of over 5000. The number of iterations is on-purpose set to a large number (250,000) as during our experimentation we concluded that after 1000 iterations the optimization begins to decrease slowly and we have decided a good-enough products vector representation is obtained when the loss starts to decrease at a rate of 0.1 per iteration. We save during our ProVe optimization after each 1000 iterations.

```

In [14]: EMBED_SIZE = 128
         MAX_FREQ = 250
         MAX_ITERS = 250000
         SAVE_ITERS = 1000

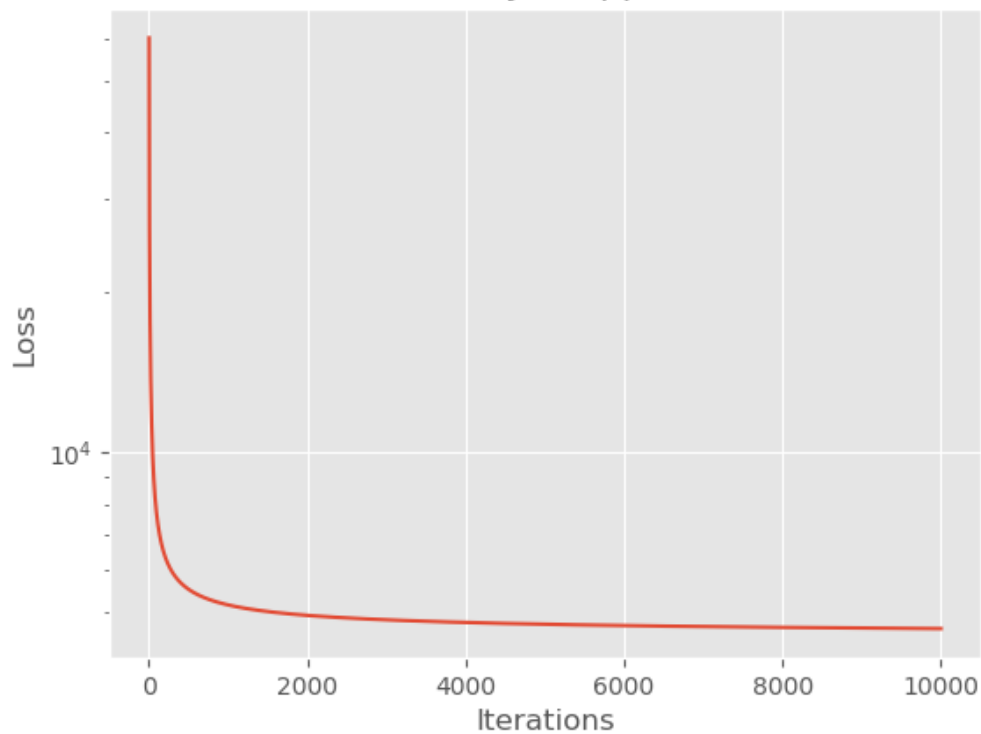
```

4.2 ProVe implementation

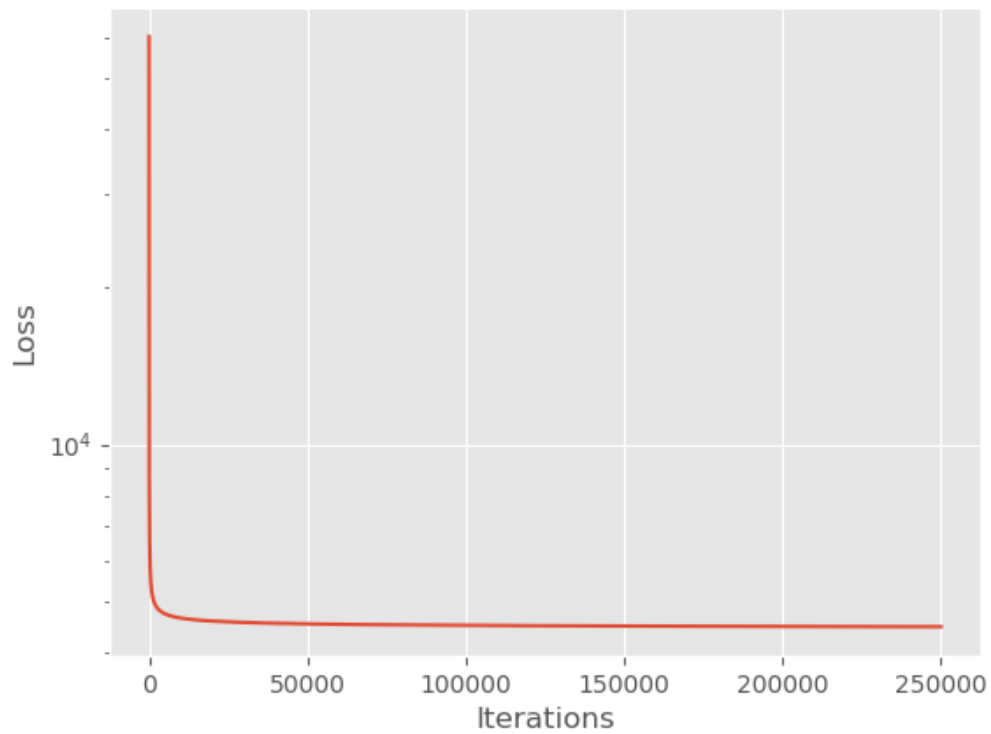
For the training of the *product* embeddings we are using the Mittens (<https://github.com/roamanalytics/mittens>) framework by Dingwall et al [6] on the prepared co-occurrences. As previously mentioned in order to speed-up model experimentation time we decided to drop the number of products down to 13,000 different items and gain optimization speed. The entire optimization procedure based on the Mittens [6] was modified so that it allows a finer control of the iterations as well as full in-GPU training without any RAM-to-VRAM transfer.

We	train	our	product	embeddings	for	10,000	epochs	(first
loss	graph	below)	and	for	250,000	epochs	(second	loss graph)

Mittens loss history (skipped first 5 iters)



Mittens loss history (skipped first 5 iters)



We compare the resulting embeddings based on our neighborhood similarity tests.

A secondary class entitled `EmbedsEngine` has been developed. It encapsulates methods for post-processing, visualization and high-end features based on pre-trained ProVe models

```
In [15]: prove_model = ProVe(
        default_embeds_file=EMB128_250k,
        name='exp_v1',
        log=log,
    )

    if prove_model.embeds is None:
        prove_model.fit(
            mco=csr_mco,
            embed_size=EMBED_SIZE,
            max_cooccure=MAX_FREQ,
            epochs=MAX_ITERS,
            save_folder=MODEL_HOME,
            save_iters=SAVE_ITERS,
        )

    embeds = prove_model.embeds

    prod_eng = EmbedsEngine(
        np_embeddings=embeds,
        df_metadata=df_meta,
        name_field='ItemName',
        id_field='IDE',
        log=log,
        categ_fields=['Ierarhie1', 'Ierarhie2'],
        dct_categ_names=dct_categories
    )
```

```
ProVe embeds (13000, 128) loaded.
Found category string in 'Ierarhie1_name'
Found category string in 'Ierarhie2_name'
Constructing items knowledge graph
  Retrieving products for category 'Ierarhie1'
  Retrieving products for category 'Ierarhie2'
Created products graph 100.0%
```

4.3 Refining evaluation data using trained ProVe

Following the full generation of the ProVe vector space model for product embeddings we can run few neighbors tests of our items expecting similar results as with the word embeddings. As previously mentioned we selected manually each individual evaluation item in order to find those that would clearly fail at proposing a good replacement candidate with just a cosine distance neighbor search within the products vector space.

Aside from the straightforward neighborhood search we also hand-picked two other products - one that is obviously a potential replacement for our product (“Best of David Bowie - 2002”) and another one that is clearly not a replacement (“NLP - by Ian McDermott”). For these two additional products we constantly measure the distances during the various phases of our experiment.

```
In [16]: test_items = [
        12071, 11418, 10088, 9251,
        9845, 8956, 6020,
        129, 1150, 3852
    ]
    exp_id = 12071
    pos_id = 11418
```

```
neg_id = 5312
```

```
prod_eng.analyze_item(exp_id, positive_id=pos_id, negative_id=neg_id)
prod_eng.get_similar_items(exp_id, filtered=False, show=print_df)
```

Analysis of 12071: 'CD / DAVID BOWIE / BLACKSTAR (2016)'

Distance from positive 11418: 0.921

Distance from negative 5312: 0.653

	ID	DIST	NAME	H1	H2
0	12071	0.0000	CD / DAVID BOWIE / BLACKSTAR (2016)	MUZICA	CD International
1	5312	0.6531	IAN MCDERMOTT / NLP PENTRU CARIERA SI VI	CARTE ROM	DEZVOLTARE PERSONALA
2	9997	0.6543	HOLLY SMALE / TOCILARA	CARTE ROM	FICTIUNE ADOLESCENTI
3	1115	0.6829	DAN LUNGU / SUNT O BABA COMUNISTA (TOP10	CARTE ROM	FICTIUNE
4	10047	0.6899	MAGNET METAL - BATMAN (JOKER)	COMICS&MANGA	Comics
5	2883	0.6930	HARRIET MUNCASTER / ISADORA MOON MERGE C	CARTE ROM	CARTI PENTRU COPII
6	12399	0.7012	YOU GIVE ME A REASON MINI CARD / KELLY H	PAPETARIE AA	Cadouri
7	7132	0.7027	SALLY GARDNER / ARIPI SI CO 3: MISTERIOA	CARTE ROM	CARTI PENTRU COPII
8	6443	0.7119	DVD / DUNKIRK (2017)	MULTIMEDIA	DVD
9	9334	0.7229	SET 6 COASTERS PARIS MONUMENTS	ACCESORII GASTR	Accesorii servit mas

As we can see from above table our proposed “anchor” product (12071) is clearly *closer* to a lot of products that might be in a complementarity relationship to it than to products from its own category (music, rock, etc). We can run the same similarity analysis using a filter that would give us only products from the same generic inventory hierarchical category.

```
In [17]: prod_eng.get_similar_items(12071, filtered=True, show=print_df)
```

	ID	DIST	NAME	H1	H2
0	12071	0.0000	CD / DAVID BOWIE / BLACKSTAR (2016)	MUZICA	CD International
140	10604	0.7962	LP / LED ZEPPELIN / PHYSICAL GRAFFITI -1	MUZICA	Vinyl
205	9991	0.8102	CD / LED ZEPPELIN / REMASTERS (1990) CD2	MUZICA	CD International
219	8599	0.8127	CD / AC/DC / BACK IN BLACK -1980- (2003)	MUZICA	CD International
453	3139	0.8368	CD / METALLICA / MASTER OF PUPPETS (1986	MUZICA	CD International
515	9333	0.8414	CD / GREEN DAY / AMERICAN IDIOT (2004)	MUZICA	CD International
638	9811	0.8509	LP / ELVIS PRESLEY / THE ESSENTIAL (2016	MUZICA	Vinyl
677	7932	0.8533	CD / METALLICA / METALLICA (1991)	MUZICA	CD International
690	6256	0.8541	LP / RED HOT CHILI PEPPERS / CALIFORNICA	MUZICA	Vinyl
710	7645	0.8552	CD / DELIA / LIVE (DIGIPACK) (2017)	MUZICA	CD Repertoriu Local

We also test the embeddings resulted from the 10,000 epochs training. Worth to mention is that the GPU 10,000 iterations took around 40' while the 250,000 took more than 12 hours. We discover that the lower epochs model behaves radically better than the 12 hrs trained model pointing to further investigation and the implementation of additional optimization heuristics (such as an early stopping mechanism)

```
In [30]: embeddings_10ke = np.load(EMB128_10k)
prod_eng.analyze_item(exp_id, positive_id=pos_id, negative_id=neg_id,
embeddings=embeddings_10ke)
prod_eng.get_similar_items(exp_id, filtered=False, show=print_df, embeddings=embeddings_10ke)
```

Analysis of 12071: 'CD / DAVID BOWIE / BLACKSTAR (2016)'

Distance from positive 11418: 0.664

Distance from negative 5312: 1.010

	ID	DIST	NAME	H1	H2
0	12071	0.000	CD / DAVID BOWIE / BLACKSTAR (2016)	MUZICA	CD International
1	11418	0.664	CD / DAVID BOWIE / BEST OF BOWIE (2002)	MUZICA	CD International
2	10035	0.688	CD / AEROSMITH / THE ESSENTIAL AEROSMITH	MUZICA	CD International
3	9040	0.689	CD / LEONARD COHEN / YOU WANT IT DARKER	MUZICA	CD International
4	5707	0.691	PLASTILINA AMOS 4/SET IC18P4 (B X 12)	JUCARII	Necatalogate
5	8639	0.699	KNOCK KNOCK 100 REASON TO PANIC BOOK ABO	CARTE STRAINA	GIFT BOOKS
6	1352	0.704	PLASTILINA AMOS PVCIC18G 18G/CUTIE GALB	JUCARII	Necatalogate
7	6964	0.708	CELLA SERGHI / PE FIRUL DE PAIANJEN AL M	CARTE ROM	BIOGRAFII / MEMORII
8	8807	0.709	SIMON SEBAG MONTEFIORE / SASENKA	CARTE ROM	FICTIUNE
9	12668	0.712	DAVID MCKEE / PRIMA MEA CARTICICA ELMER	CARTE ROM	CARTI PENTRU COPII

5 The full Pipeline

There are two main pipelines, one for each of our two individual objectives : that of finding product replacements and the second one of cold-starting new products in inventories:

Algorithm #1 GetProductReplacement - Product Replacement Modelling pipeline

1. Create MCO
2. Apply ProVe and obtain item embeddings
3. Extract knowledge graph information from meta-data
4. For each individual product
 - 4.1. Find connected products
 - 4.2. Retrofit the product using connected products

Algorithm #2 ColdStartProduct - cold-starting new products embeddings

1. Setup embeddings of existing products and extract knowledge graph information from meta-data
2. Input new product hierarchy information
3. Input similar products from proposed similar products
4. Create new product embedding based on proposed similar with simple mean over embeddings
5. Add all hierarchy-based products to the list of similar products
6. Retrofit the new product embedding to all similar products

At this point we have discussed regarding the MCO preparation and processing the count matrix with *ProVe* in order to obtain the initial products embedding space. The next logical step (2. in our algorithm) is to process the available hierarchical information in order to construct the knowledge graph. For this we are going to treat each category provided by the inventory's category management database as a potential source of information that can provide knowledge graph edges for each product in its domain. For our previous item example 12071 (David Bowie's "Blackstar") the product information is presented below:

```
In [18]: dct_prod_info = prod_eng.get_item_info(exp_id, verbose=True)
```

```
Product '12071' info:
```

```
NAME: CD / DAVID BOWIE / BLACKSTAR (2016)
Level 'Ierarhie1': 2
Level 'Ierarhie1_name': MUZICA
Level 'Ierarhie2': 4
Level 'Ierarhie2_name': CD International
EDGES: {10759, 1032, 12305, 3091, 4627, 10780, 9251, 1232}
```

At this point we can proceed with the final step presented in Algorithm #1, that of using the knowledge graph and pulling closer the products that have edges. Our proposed framework has

multiple flavors when it comes to selecting the part of the embedding space we want to modify and in this case, for the sake of the experiment and for a clearer intuition, we chose only to modify the vector space portion that is most relevant for our chose product (12071)

```
In [19]: new_embeddings = prod_eng.get_retrofitted_embeddings(prod_ids=exp_id, method='v1')
```

Performing retrofitting on (13000, 128) embedding matrix...

Stop tel: 5.0e-03

Start-edges: 1

Iteration 1; change was 3.5e+00

Iteration 2; change was 8.0e-03

Retrofitting converged at iteration 3; change was 0.0000

The next logical step is to verify the results of our retrofitting process re-running the past neighborhood tests both with and without filtering. First we run the search with no filtering:

```
In [20]: prod_eng.get_similar_items(exp_id, embeddings=new_embeddings, filtered=False, show=print_df)
```

	ID	DIST	NAME	H1	H2
0	12071	1.7881e-07	CD / DAVID BOWIE / BLACKSTAR (2016)	MUZICA	CD International
1	5312	6.6791e-01	IAN MCDERMOTT / NLP PENTRU CARIERA SI VI	CARTE ROM	DEZVOLTARE PERSONALA
2	9997	6.7252e-01	HOLLY SMALE / TOCILARA	CARTE ROM	FICTIUNE ADOLESCENTI
3	10047	6.8937e-01	MAGNET METAL - BATMAN (JOKER)	COMICS&MANGA	Comics
4	1115	6.9019e-01	DAN LUNGU / SUNT O BABA COMUNISTA (TOP10	CARTE ROM	FICTIUNE
5	7132	6.9034e-01	SALLY GARDNER / ARIPI SI CO 3: MISTERIOA	CARTE ROM	CARTI PENTRU COPII
6	6443	6.9186e-01	DVD / DUNKIRK (2017)	MULTIMEDIA	DVD
7	12399	7.0380e-01	YOU GIVE ME A REASON MINI CARD / KELLY H	PAPETARIE AA	Cadouri
8	2883	7.0732e-01	HARRIET MUNCASTER / ISADORA MOON MERGE C	CARTE ROM	CARTI PENTRU COPII
9	9334	7.2087e-01	SET 6 COASTERS PARIS MONUMENTS	ACCESORII GASTR	Accesorii servit mas

And then we run it with filtering using the hierarchy management database.

```
In [21]: df = prod_eng.get_similar_items(exp_id, embeddings=new_embeddings, filtered=True, show=print_df)
```

Now we can repeat our hand-picked positive vs negative examples test and observe an improvement in the distances (slightly lower for the positive and slightly higher for the negative). However, this improvement is far from our desired objective of determining the replacement (the actual positive example in this case) with the simple cosine distance measurement.

```
In [22]: prod_eng.analyze_item(exp_id, positive_id=pos_id, negative_id=neg_id, embeddings=new_embeddings)
```

Analysis of 12071: 'CD / DAVID BOWIE / BLACKSTAR (2016)'

Distance from positive 11418: 0.903

Distance from negative 5312: 0.668

6 Progress history

At this point in our research and development process we managed to obtain generate the product embeddings based starting from the real-life transactional database. A more important finding is that we have experimental proof that our *ProVe* model shows clear signs of capturing “semantic” information regarding product similarities as well as product complementarity, albeit without a clear cut between the two distinct categories. Similar findings can be discerned from the early work of Grbovic et al [2] who use the k-means based clusters in order to separate potential similarities from complementarities.

Progress has also been made with regard to the final stages of the proposed pipeline as the first iteration of experimentation with Faruqui et al [4] has been implemented as presented before.

There is still a extensive list of actions and problems that have to be tackled such as:

1. Review the ProVe training and find the required heuristics that would aid us in finind the best time-vs-embeddings quality ratio.
2. Review the knowledge graph preparation process as well as optimize the v1 retrofitting method
3. Tune the ProVe hyperparameters and scalability. Re-train the model with maximum energy efficiency vs performance
4. Evaluate these initial results with the proposed MRR metric on a larger set of manually chosen items
5. Add optimized versions of the retrofitting as well as functional retrofitting (Lengerich et al [7])
6. Evaluate the results for the advanced methods and compare with (1)
7. Complete the ColdStartProduct pipeline (Algorithm #2)

```
In [23]: printmd("END of Experimental Protocol.")
```

END of Experimental Protocol.

References

- [1] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [2] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1809–1818, 2015.
- [3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [4] Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*, 2014.
- [5] Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting word vectors to linguistic constraints. *arXiv preprint arXiv:1603.00892*, 2016.
- [6] Nicholas Dingwall and Christopher Potts. Mittens: an extension of glove for learning domain-specialized representations. *arXiv preprint arXiv:1803.09901*, 2018.
- [7] Benjamin J Lengerich, Andrew L Maas, and Christopher Potts. Retrofitting distributional embeddings to knowledge graphs with functional relations. *arXiv preprint arXiv:1708.00112*, 2017.
- [8] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.