Assignment 6: Embeddings

sh2432

Due: Thursday, November 14, 2019

This assignment involves two problems related to word embeddings. The first problem explores word embeddings constructed on Wikipedia data. The second problem involves constructing embeddings of music artists from radio station song playlists.

Lots of guidance and code are given for solving the problems--the assignment shouldn't take a long time to complete. The second problem should be straightfoward after you do the first problem.

Problem 1: Word embedding experiments (30 points)

In this problem you will run experiments on word embeddings using two different algorithms or configurations: word2vec embeddings trained on the "text8" Wikipedia corpus (2) GloVe embeddings pre-trained on a much larger corpus.

The text8 data are described here: http://mattmahoney.net/dc/textdata.html (http://mattmahoney.net/dc/textdata.html). You can download the data from http://mattmahoney.net/dc/text8.zip (http://mattmahoney.net/dc/text8.zip). Just unzip that file to obtain the text8 file, a 100MB excerpt of Wikipedia. This small dataset is sufficient for our exploratory purposes, but note that it is far too small for any real application. In the next few parts of this problem, you will construct word embeddings from the Wikipedia data.

`word2Vec` is a popular word embedding method. The following code will construct 100 dimensional embeddings on the text8 data.

In [1]:

```
import gensim
from gensim.models import word2vec
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', \
                    level=logging.INFO)
sentences = word2vec.Text8Corpus('text8')
model = word2vec.Word2Vec(sentences, size=100, window=10, min_count=10)
```

/

```
2019-11-14 10:31:59,632 : INFO : collecting all words and their counts
2019-11-14 10:31:59,668 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0
word types
2019-11-14 10:32:10,480 : INFO : collected 253854 word types from a corpus of 17005207
raw words and 1701 sentences
2019-11-14 10:32:10,481 : INFO : Loading a fresh vocabulary
2019-11-14 10:32:10,865 : INFO : effective_min_count=10 retains 47134 unique words (18%
of original 253854, drops 206720)
2019-11-14 10:32:10,865 : INFO : effective_min_count=10 leaves 16561031 word corpus (9
7% of original 17005207, drops 444176)
2019-11-14 10:32:11,105 : INFO : deleting the raw counts dictionary of 253854 items
2019-11-14 10:32:11,119 : INFO : sample=0.001 downsamples 38 most-common words
2019-11-14 10:32:11,121 : INFO : downsampling leaves estimated 12333563 word corpus (7
4.5% of prior 16561031)
2019-11-14 10:32:11,384 : INFO : estimated required memory for 47134 words and 100 dime
nsions: 61274200 bytes
2019-11-14 10:32:11,387 : INFO : resetting layer weights
2019-11-14 10:32:12,633 : INFO : training model with 3 workers on 47134 vocabulary and
100 features, using sg=0 hs=0 sample=0.001 negative=5 window=10
2019-11-14 10:32:13,651 : INFO : EPOCH 1 - PROGRESS: at 3.47% examples, 424157 words/s,
in_qsize 5, out_qsize 0
2019-11-14 10:32:14,664 : INFO : EPOCH 1 - PROGRESS: at 7.17% examples, 432617 words/s,
in_qsize 6, out_qsize 0
2019-11-14 10:32:15,665 : INFO : EPOCH 1 - PROGRESS: at 11.11% examples, 448417 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:16,672 : INFO : EPOCH 1 - PROGRESS: at 14.87% examples, 451631 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:17,681 : INFO : EPOCH 1 - PROGRESS: at 18.87% examples, 459282 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:18,685 : INFO : EPOCH 1 - PROGRESS: at 22.69% examples, 461169 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:19,699 : INFO : EPOCH 1 - PROGRESS: at 26.57% examples, 463531 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:20,702 : INFO : EPOCH 1 - PROGRESS: at 30.34% examples, 464333 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:21,704 : INFO : EPOCH 1 - PROGRESS: at 33.98% examples, 463161 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:22,722 : INFO : EPOCH 1 - PROGRESS: at 37.68% examples, 462043 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:23,728 : INFO : EPOCH 1 - PROGRESS: at 41.33% examples, 460706 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:24,732 : INFO : EPOCH 1 - PROGRESS: at 45.09% examples, 461098 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:25,743 : INFO : EPOCH 1 - PROGRESS: at 49.03% examples, 462788 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:26,747 : INFO : EPOCH 1 - PROGRESS: at 52.85% examples, 463354 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:27,751 : INFO : EPOCH 1 - PROGRESS: at 56.61% examples, 463487 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:28,759 : INFO : EPOCH 1 - PROGRESS: at 60.26% examples, 462490 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:29,760 : INFO : EPOCH 1 - PROGRESS: at 63.90% examples, 461769 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:30,766 : INFO : EPOCH 1 - PROGRESS: at 67.43% examples, 460157 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:31,788 : INFO : EPOCH 1 - PROGRESS: at 71.08% examples, 459232 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:32,805 : INFO : EPOCH 1 - PROGRESS: at 74.78% examples, 458813 words/
s, in_qsize 5, out_qsize 0
```

```
2019-11-14 10:32:33,817 : INFO : EPOCH 1 - PROGRESS: at 78.72% examples, 459017 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:34,826 : INFO : EPOCH 1 - PROGRESS: at 82.54% examples, 459216 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:35,838 : INFO : EPOCH 1 - PROGRESS: at 86.36% examples, 459548 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:36,844 : INFO : EPOCH 1 - PROGRESS: at 90.06% examples, 459409 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:37,844 : INFO : EPOCH 1 - PROGRESS: at 93.83% examples, 459340 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:38,862 : INFO : EPOCH 1 - PROGRESS: at 97.77% examples, 459937 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:39,401 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 10:32:39,410 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 10:32:39,416 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 10:32:39,417 : INFO : EPOCH - 1 : training on 17005207 raw words (12335160 e
ffective words) took 26.8s, 460627 effective words/s
2019-11-14 10:32:40,434 : INFO : EPOCH 2 - PROGRESS: at 3.70% examples, 452216 words/s,
in_qsize 6, out_qsize 0
2019-11-14 10:32:41,448 : INFO : EPOCH 2 - PROGRESS: at 7.64% examples, 461417 words/s,
in_qsize 6, out_qsize 0
2019-11-14 10:32:42,465 : INFO : EPOCH 2 - PROGRESS: at 11.46% examples, 460552 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:43,473 : INFO : EPOCH 2 - PROGRESS: at 15.29% examples, 462120 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:44,481 : INFO : EPOCH 2 - PROGRESS: at 19.05% examples, 461825 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:45,483 : INFO : EPOCH 2 - PROGRESS: at 22.69% examples, 459972 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:46,491 : INFO : EPOCH 2 - PROGRESS: at 26.16% examples, 455628 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:47,494 : INFO : EPOCH 2 - PROGRESS: at 29.75% examples, 454650 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:48,500 : INFO : EPOCH 2 - PROGRESS: at 33.33% examples, 453450 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:49,512 : INFO : EPOCH 2 - PROGRESS: at 36.98% examples, 452994 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:50,523 : INFO : EPOCH 2 - PROGRESS: at 40.86% examples, 454849 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:51,525 : INFO : EPOCH 2 - PROGRESS: at 44.68% examples, 456402 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:52,537 : INFO : EPOCH 2 - PROGRESS: at 48.32% examples, 455599 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:53,559 : INFO : EPOCH 2 - PROGRESS: at 52.15% examples, 456027 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:54,563 : INFO : EPOCH 2 - PROGRESS: at 56.03% examples, 457722 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:55,568 : INFO : EPOCH 2 - PROGRESS: at 59.67% examples, 457080 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:56,577 : INFO : EPOCH 2 - PROGRESS: at 63.43% examples, 457353 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:57,578 : INFO : EPOCH 2 - PROGRESS: at 67.02% examples, 456516 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:32:58,590 : INFO : EPOCH 2 - PROGRESS: at 70.78% examples, 456722 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:32:59,596 : INFO : EPOCH 2 - PROGRESS: at 74.49% examples, 456797 words/
```

```
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:00,605 : INFO : EPOCH 2 - PROGRESS: at 78.25% examples, 456088 words/
s, in_qsize 5, out_qsize 1
2019-11-14 10:33:01,625 : INFO : EPOCH 2 - PROGRESS: at 81.78% examples, 454616 words/
s, in_qsize 5, out_qsize 1
2019-11-14 10:33:02,638 : INFO : EPOCH 2 - PROGRESS: at 85.48% examples, 454467 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:03,649 : INFO : EPOCH 2 - PROGRESS: at 89.30% examples, 454988 words/
s, in_qsize 6, out_qsize 1
2019-11-14 10:33:04,655 : INFO : EPOCH 2 - PROGRESS: at 92.83% examples, 453971 words/
s, in_qsize 5, out_qsize 1
2019-11-14 10:33:05,668 : INFO : EPOCH 2 - PROGRESS: at 96.65% examples, 454322 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:06,504 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 10:33:06,518 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 10:33:06,522 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 10:33:06,523 : INFO : EPOCH - 2 : training on 17005207 raw words (12334019 e
ffective words) took 27.1s, 455089 effective words/s
2019-11-14 10:33:07,528 : INFO : EPOCH 3 - PROGRESS: at 3.64% examples, 450970 words/s,
in_qsize 6, out_qsize 0
2019-11-14 10:33:08,535 : INFO : EPOCH 3 - PROGRESS: at 7.52% examples, 458954 words/s,
in_qsize 5, out_qsize 0
2019-11-14 10:33:09,537 : INFO : EPOCH 3 - PROGRESS: at 11.41% examples, 463521 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:10,543 : INFO : EPOCH 3 - PROGRESS: at 15.29% examples, 466466 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:11,557 : INFO : EPOCH 3 - PROGRESS: at 18.87% examples, 460452 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:12,565 : INFO : EPOCH 3 - PROGRESS: at 22.69% examples, 461864 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:13,583 : INFO : EPOCH 3 - PROGRESS: at 26.46% examples, 461763 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:14,584 : INFO : EPOCH 3 - PROGRESS: at 30.16% examples, 461974 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:15,589 : INFO : EPOCH 3 - PROGRESS: at 33.98% examples, 463358 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:16,612 : INFO : EPOCH 3 - PROGRESS: at 37.74% examples, 462724 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:17,623 : INFO : EPOCH 3 - PROGRESS: at 41.33% examples, 460392 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:18,628 : INFO : EPOCH 3 - PROGRESS: at 44.74% examples, 457173 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:19,632 : INFO : EPOCH 3 - PROGRESS: at 48.50% examples, 457699 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:20,637 : INFO : EPOCH 3 - PROGRESS: at 52.20% examples, 457547 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:21,657 : INFO : EPOCH 3 - PROGRESS: at 56.03% examples, 458246 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:22,670 : INFO : EPOCH 3 - PROGRESS: at 59.73% examples, 457797 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:23,682 : INFO : EPOCH 3 - PROGRESS: at 63.61% examples, 458775 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:24,686 : INFO : EPOCH 3 - PROGRESS: at 67.37% examples, 458948 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:25,691 : INFO : EPOCH 3 - PROGRESS: at 71.19% examples, 459692 words/
s, in_qsize 5, out_qsize 0
```

```
2019-11-14 10:33:26,693 : INFO : EPOCH 3 - PROGRESS: at 75.19% examples, 461087 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:27,701 : INFO : EPOCH 3 - PROGRESS: at 78.95% examples, 460488 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:28,712 : INFO : EPOCH 3 - PROGRESS: at 82.89% examples, 461227 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:29,714 : INFO : EPOCH 3 - PROGRESS: at 86.71% examples, 461713 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:30,725 : INFO : EPOCH 3 - PROGRESS: at 90.36% examples, 461084 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:31,729 : INFO : EPOCH 3 - PROGRESS: at 94.30% examples, 461762 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:32,730 : INFO : EPOCH 3 - PROGRESS: at 98.18% examples, 462224 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:33,185 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 10:33:33,205 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 10:33:33,224 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 10:33:33,226 : INFO : EPOCH - 3 : training on 17005207 raw words (12334490 e
ffective words) took 26.7s, 462016 effective words/s
2019-11-14 10:33:34,246 : INFO : EPOCH 4 - PROGRESS: at 3.29% examples, 401435 words/s,
in_qsize 6, out_qsize 0
2019-11-14 10:33:35,253 : INFO : EPOCH 4 - PROGRESS: at 7.00% examples, 422700 words/s,
in_qsize 5, out_qsize 0
2019-11-14 10:33:36,275 : INFO : EPOCH 4 - PROGRESS: at 10.64% examples, 427858 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:37,278 : INFO : EPOCH 4 - PROGRESS: at 14.46% examples, 437944 words/
s, in_qsize 5, out_qsize 1
2019-11-14 10:33:38,284 : INFO : EPOCH 4 - PROGRESS: at 17.93% examples, 435588 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:39,287 : INFO : EPOCH 4 - PROGRESS: at 21.81% examples, 442453 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:40,297 : INFO : EPOCH 4 - PROGRESS: at 25.46% examples, 443515 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:41,311 : INFO : EPOCH 4 - PROGRESS: at 29.22% examples, 445992 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:42,314 : INFO : EPOCH 4 - PROGRESS: at 32.98% examples, 448556 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:43,340 : INFO : EPOCH 4 - PROGRESS: at 36.63% examples, 448037 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:44,346 : INFO : EPOCH 4 - PROGRESS: at 40.51% examples, 450538 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:45,363 : INFO : EPOCH 4 - PROGRESS: at 44.21% examples, 450627 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:46,375 : INFO : EPOCH 4 - PROGRESS: at 47.97% examples, 451429 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:47,386 : INFO : EPOCH 4 - PROGRESS: at 51.62% examples, 451027 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:48,388 : INFO : EPOCH 4 - PROGRESS: at 55.26% examples, 451034 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:49,406 : INFO : EPOCH 4 - PROGRESS: at 59.08% examples, 451880 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:50,418 : INFO : EPOCH 4 - PROGRESS: at 62.55% examples, 450249 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:51,423 : INFO : EPOCH 4 - PROGRESS: at 66.02% examples, 448899 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:52,441 : INFO : EPOCH 4 - PROGRESS: at 69.72% examples, 448985 words/
```

```
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:53,454 : INFO : EPOCH 4 - PROGRESS: at 73.13% examples, 447500 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:54,469 : INFO : EPOCH 4 - PROGRESS: at 77.13% examples, 448612 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:55,482 : INFO : EPOCH 4 - PROGRESS: at 81.13% examples, 450198 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:56,483 : INFO : EPOCH 4 - PROGRESS: at 84.71% examples, 449791 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:33:57,491 : INFO : EPOCH 4 - PROGRESS: at 88.30% examples, 449304 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:58,501 : INFO : EPOCH 4 - PROGRESS: at 92.06% examples, 449645 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:33:59,509 : INFO : EPOCH 4 - PROGRESS: at 95.94% examples, 450579 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:00,523 : INFO : EPOCH 4 - PROGRESS: at 99.76% examples, 451013 words/
s, in_qsize 4, out_qsize 0
2019-11-14 10:34:00,552 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 10:34:00,559 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 10:34:00,566 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 10:34:00,568 : INFO : EPOCH - 4 : training on 17005207 raw words (12334417 e
ffective words) took 27.3s, 451196 effective words/s
2019-11-14 10:34:01,573 : INFO : EPOCH 5 - PROGRESS: at 3.76% examples, 464837 words/s,
in_qsize 5, out_qsize 0
2019-11-14 10:34:02,575 : INFO : EPOCH 5 - PROGRESS: at 7.41% examples, 451993 words/s,
in_qsize 4, out_qsize 1
2019-11-14 10:34:03,583 : INFO : EPOCH 5 - PROGRESS: at 11.52% examples, 467526 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:04,590 : INFO : EPOCH 5 - PROGRESS: at 15.23% examples, 464094 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:05,591 : INFO : EPOCH 5 - PROGRESS: at 18.87% examples, 461192 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:06,610 : INFO : EPOCH 5 - PROGRESS: at 22.40% examples, 455589 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:07,621 : INFO : EPOCH 5 - PROGRESS: at 26.28% examples, 459091 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:08,622 : INFO : EPOCH 5 - PROGRESS: at 29.92% examples, 458643 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:09,627 : INFO : EPOCH 5 - PROGRESS: at 33.69% examples, 459530 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:10,642 : INFO : EPOCH 5 - PROGRESS: at 37.51% examples, 460338 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:11,652 : INFO : EPOCH 5 - PROGRESS: at 41.27% examples, 460316 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:12,674 : INFO : EPOCH 5 - PROGRESS: at 45.09% examples, 460684 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:13,679 : INFO : EPOCH 5 - PROGRESS: at 48.91% examples, 461476 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:14,683 : INFO : EPOCH 5 - PROGRESS: at 52.44% examples, 459493 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:15,692 : INFO : EPOCH 5 - PROGRESS: at 56.32% examples, 460835 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:16,696 : INFO : EPOCH 5 - PROGRESS: at 60.08% examples, 460885 words/
s, in_qsize 5, out_qsize 1
2019-11-14 10:34:17,713 : INFO : EPOCH 5 - PROGRESS: at 63.90% examples, 461149 words/
s, in_qsize 6, out_qsize 0
```

```
2019-11-14 10:34:18,727 : INFO : EPOCH 5 - PROGRESS: at 67.61% examples, 460553 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:19,741 : INFO : EPOCH 5 - PROGRESS: at 71.19% examples, 459459 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:20,748 : INFO : EPOCH 5 - PROGRESS: at 74.78% examples, 458508 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:21,772 : INFO : EPOCH 5 - PROGRESS: at 78.42% examples, 456767 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:22,782 : INFO : EPOCH 5 - PROGRESS: at 81.95% examples, 455450 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:23,793 : INFO : EPOCH 5 - PROGRESS: at 85.71% examples, 455639 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:24,817 : INFO : EPOCH 5 - PROGRESS: at 89.48% examples, 455534 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:25,825 : INFO : EPOCH 5 - PROGRESS: at 93.30% examples, 455930 words/
s, in_qsize 5, out_qsize 0
2019-11-14 10:34:26,842 : INFO : EPOCH 5 - PROGRESS: at 97.35% examples, 457180 words/
s, in_qsize 6, out_qsize 0
2019-11-14 10:34:27,543 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 10:34:27,550 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 10:34:27,554 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 10:34:27,555 : INFO : EPOCH - 5 : training on 17005207 raw words (12333225 e
ffective words) took 27.0s, 457094 effective words/s
2019-11-14 10:34:27,556 : INFO : training on a 85026035 raw words (61671311 effective w
ords) took 134.9s, 457094 effective words/s
```

The dictionary `model.wv` , keyed by words (as strings), has values which are the word embeddings (as numpy arrays).

We will also work with pre-trained GloVe embeddings. These embeddings were trained on a large corpus containing 6 billion tokens. You can load these embedding vectors using this code:

In [2]:
```
import gensim.downloader as gdl
glove = gdl.load("glove-wiki-gigaword-100")
```

```
2019-11-14 10:34:46,868 : INFO : loading projection weights from C:\Users\kavan/gensim-
data\glove-wiki-gigaword-100\glove-wiki-gigaword-100.gz
2019-11-14 10:36:29,841 : INFO : loaded (400000, 100) matrix from C:\Users\kavan/gensim
-data\glove-wiki-gigaword-100\glove-wiki-gigaword-100.gz
```

Here is a sample evaluation: puppy is to dog as what is to kitten?

In [3]:
```
glove.most_similar(positive = ['dog', 'kitten'], negative = ['puppy'])[0]
```

```
2019-11-14 10:36:29,855 : INFO : precomputing L2-norms of word weight vectors
```

Out[3]:
```
('cat', 0.6935379505157471)
```

Experiments

Conduct the following experiments with both sets of embeddings: the local word2vec embeddings, and the pre-trained GloVe embeddings. Based on the available memory on your computer, you may need to perform the experiments for each set of embeddings in a fresh python session. Comment on the qualitative differences in the results for each of the embeddings.

1.1 Find closest words

For each of the following words, find the 5 closest words in the embedding space, and report your results:

```
yale, physics, republican, einstein, algebra, fish
```

Here, "closest" means in terms of cosine similarity. See the gensim documentation; you might want to use the most similar function, or a related function. Choose five other query words yourself, and for each of them show the closest words in the embedding space. Comment on your findings.

In [4]:
```
#pre-trained GloVe embeddings:
glove.most_similar('yale') [0:5]
```

Out[4]:
```
[('harvard', 0.9161344766616821),
 ('princeton', 0.867539644241333),
 ('university', 0.8113802075386047),
 ('cornell', 0.801445484161377),
 ('stanford', 0.7877545356750488)]
```

In [5]:
```
glove.most_similar('physics') [0:5]
```

Out[5]:
```
[('chemistry', 0.8498000502586365),
 ('mathematics', 0.8340947031974792),
 ('science', 0.7914698719978333),
 ('biology', 0.7894973158836365),
 ('theoretical', 0.7342938780784607)]
```

In [6]:
```
glove.most_similar('republican') [0:5]
```

Out[6]:
```
[('gop', 0.9306843876838684),
 ('democrat', 0.8788903951644897),
 ('democratic', 0.8735148906707764),
 ('republicans', 0.8709758520126343),
 ('democrats', 0.8635865449905396)]
```

In [7]:
```
glove.most_similar('einstein') [0:5]
```

Out[7]:
```
[('relativity', 0.6908349394798279),
 ('freud', 0.6417257785797119),
 ('physics', 0.6145339012145996),
 ('bohr', 0.6144247055053711),
 ('theory', 0.6042598485946655)]
```

In [8]:
```
glove.most_similar('algebra') [0:5]
```

Out[8]:
```
[('algebras', 0.7539783716201782),
 ('geometry', 0.6790961027145386),
 ('algebraic', 0.6474162340164185),
 ('boolean', 0.6418250799179077),
 ('commutative', 0.6322142481803894)]
```

In [9]:
```
glove.most_similar('fish') [0:5]
```

Out[9]:
```
[('shrimp', 0.7793381214141846),
 ('salmon', 0.7608143091201782),
 ('tuna', 0.7485247254371643),
 ('meat', 0.7119255065917969),
 ('seafood', 0.6940564513206482)]
```

In [10]:
```
#chose my own words
glove.most_similar('meat') [0:5]
```

Out[10]:
```
[('beef', 0.8396860361099243),
 ('pork', 0.8260456323623657),
 ('chicken', 0.8187731504440308),
 ('milk', 0.7481759786605835),
 ('poultry', 0.7392504215240479)]
```

In [11]:
```
glove.most_similar('house') [0:5]
```

Out[11]:
```
[('office', 0.7581614851951599),
 ('senate', 0.7204986214637756),
 ('room', 0.7149738073348999),
 ('houses', 0.6888046264648438),
 ('capitol', 0.6851760149002075)]
```

In [12]:
```
glove.most_similar('amber') [0:5]
```

Out[12]:
```
[('violet', 0.6195389032363892),
 ('turquoise', 0.5459985733032227),
 ('tamblyn', 0.5357284545898438),
 ('jade', 0.5300266742706299),
 ('lily', 0.5154258012771606)]
```

In [13]:
```
glove.most_similar('machine') [0:5]
```

Out[13]:
```
[('machines', 0.7854336500167847),
 ('device', 0.6772987842559814),
 ('equipment', 0.6411972641944885),
 ('gun', 0.6409083604812622),
 ('guns', 0.6361788511276245)]
```

In [14]:
```
glove.most_similar('universe') [0:5]
```

Out[14]:
```
[('planet', 0.6928273439407349),
 ('earth', 0.6642659902572632),
 ('reality', 0.6588369011878967),
 ('realm', 0.6460943222045898),
 ('worlds', 0.6435551047325134)]
```

/

In [15]:
```
#local word2vec embeddings:
model.wv.most_similar('yale') [0:5]
```

```
2019-11-14 10:37:21,669 : INFO : precomputing L2-norms of word weight vectors
```

Out[15]:
```
[('harvard', 0.8705672025680542),
 ('cornell', 0.8208866119384766),
 ('rutgers', 0.8123234510421753),
 ('princeton', 0.8106819987297058),
 ('mcgill', 0.7560324668884277)]
```

In [16]:
```
model.wv.most_similar('physics') [0:5]
```

Out[16]:
```
[('mechanics', 0.8188358545303345),
 ('electromagnetism', 0.7710298299789429),
 ('optics', 0.7570593953132629),
 ('electrodynamics', 0.7450600266456604),
 ('quantum', 0.7370455265045166)]
```

In [17]:
```
model.wv.most_similar('republican') [0:5]
```

Out[17]:
```
[('republicans', 0.8349669575691223),
 ('democrat', 0.8342225551605225),
 ('democrats', 0.7954860925674438),
 ('incumbent', 0.7560781836509705),
 ('mps', 0.7549967765808105)]
```

In [18]:
```
model.wv.most_similar('einstein') [0:5]
```

Out[18]:
```
[('heisenberg', 0.7971362471580505),
 ('maxwell', 0.7942721247673035),
 ('planck', 0.7747113108634949),
 ('schr', 0.7679381370544434),
 ('electrodynamics', 0.7678996324539185)]
```

In [19]:
```
model.wv.most_similar('algebra') [0:5]
```

Out[19]:
```
[('algebraic', 0.8730374574661255),
 ('banach', 0.8227748870849609),
 ('associative', 0.8206220865249634),
 ('topology', 0.8183113932609558),
 ('boolean', 0.8051547408103943)]
```

In [20]:
```
model.wv.most_similar('fish') [0:5]
```

Out[20]:
```
[('shrimp', 0.8198156356811523),
 ('sheep', 0.8125563859939575),
 ('fruit', 0.809190034866333),
 ('foxes', 0.8078215718269348),
 ('nests', 0.8025256395339966)]
```

In [21]:
```
#my own words:
model.wv.most_similar('meat') [0:5]
```

Out[21]:
```
[('vegetables', 0.9014124870300293),
 ('beef', 0.8825104236602783),
 ('milk', 0.8787078857421875),
 ('vegetable', 0.8751095533370972),
 ('foods', 0.8657786846160889)]
```

In [22]:
```
model.wv.most_similar('house') [0:5]
```

Out[22]:
```
[('commons', 0.6864175796508789),
 ('lords', 0.59449303150177),
 ('manor', 0.5933786034584045),
 ('palace', 0.5657316446304321),
 ('houses', 0.559410572052002)]
```

In [23]:
```
model.wv.most_similar('amber') [0:5]
```

Out[23]:
```
[('alabaster', 0.7462334036827087),
 ('wool', 0.706028401851654),
 ('pale', 0.7046392560005188),
 ('amethyst', 0.6953625679016113),
 ('ivory', 0.6937078237533569)]
```

In [24]:
```
model.wv.most_similar('machine') [0:5]
```

Out[24]:
```
[('machines', 0.7225797772407532),
 ('device', 0.7169724106788635),
 ('calculator', 0.6875922083854675),
 ('enigma', 0.6692907810211182),
 ('controller', 0.668738842010498)]
```

In [25]:
```
model.wv.most_similar('universe') [0:5]
```

Out[25]:
```
[('galaxy', 0.7068586349487305),
 ('multiverse', 0.7044622302055359),
 ('primordial', 0.667279839515686),
 ('reality', 0.6498914957046509),
 ('universes', 0.6488914489746094)]
```

In general, the five closest words in the pre-trained GloVe embeddings are closer to the original word than the local word2vec embeddings. For some less commonly used words like 'amber', the closest words from the word2vec embeddings makes no sense.

1.2 Complete analogies

A surprising consequence of some word embedding methods is that they can be used to resolve analogies, like

```
france :  paris ::  england :  ?
```

You can "solve" this analogy by computing the nearest embedding vector to v where, $v = v_{paris} - v_{france} + v_{england}$.

Solve the following analogies with both sets of word embeddings and report your results:

```
france :  paris ::  england :  ?
france :  paris ::  germany :  ?
  queen :  woman ::  king :  ?
```

Choose five other analogies yourself, and report on the results.

In [26]:
```
glove.most_similar(positive=['paris', 'england'], negative=['france'])[0]
```

Out[26]:
```
('london', 0.8123227953910828)
```

In [27]:
```
glove.most_similar(positive=['paris', 'germany'], negative=['france'])[0]
```

Out[27]:
```
('berlin', 0.8846380710601807)
```

In [28]:
```
glove.most_similar(positive=['woman', 'king'], negative=['queen'])[0]
```

Out[28]:
```
('man', 0.7907768487930298)
```

In [29]:
```
glove.most_similar(positive=['singer', 'novel'], negative=['stage'])[0]
```

Out[29]:
```
('author', 0.716401219367981)
```

In [30]:
```
glove.most_similar(positive=['teacher', 'doctor'], negative=['student'])[0]
```

Out[30]:
```
('nurse', 0.7408320903778076)
```

In [31]:
```
glove.most_similar(positive=['cook', 'write'], negative=['recipes'])[0]
```

Out[31]:
```
('let', 0.60434490442276)
```

In [32]:
```
glove.most_similar(positive=['research', 'industry'], negative=['science'])[0]
```

Out[32]:
```
('companies', 0.7378610372543335)
```

In [33]:
```
glove.most_similar(positive=['water', 'vegetable'], negative=['drink'])[0]
```

Out[33]:
```
('irrigation', 0.6506858468055725)
```

some analogies are accurate based on the given information, some are not very close. for example, when put in receipes : cook :: write : ?, it returns 'let' which dosen't make any sense.

1.3 Visualize embeddings

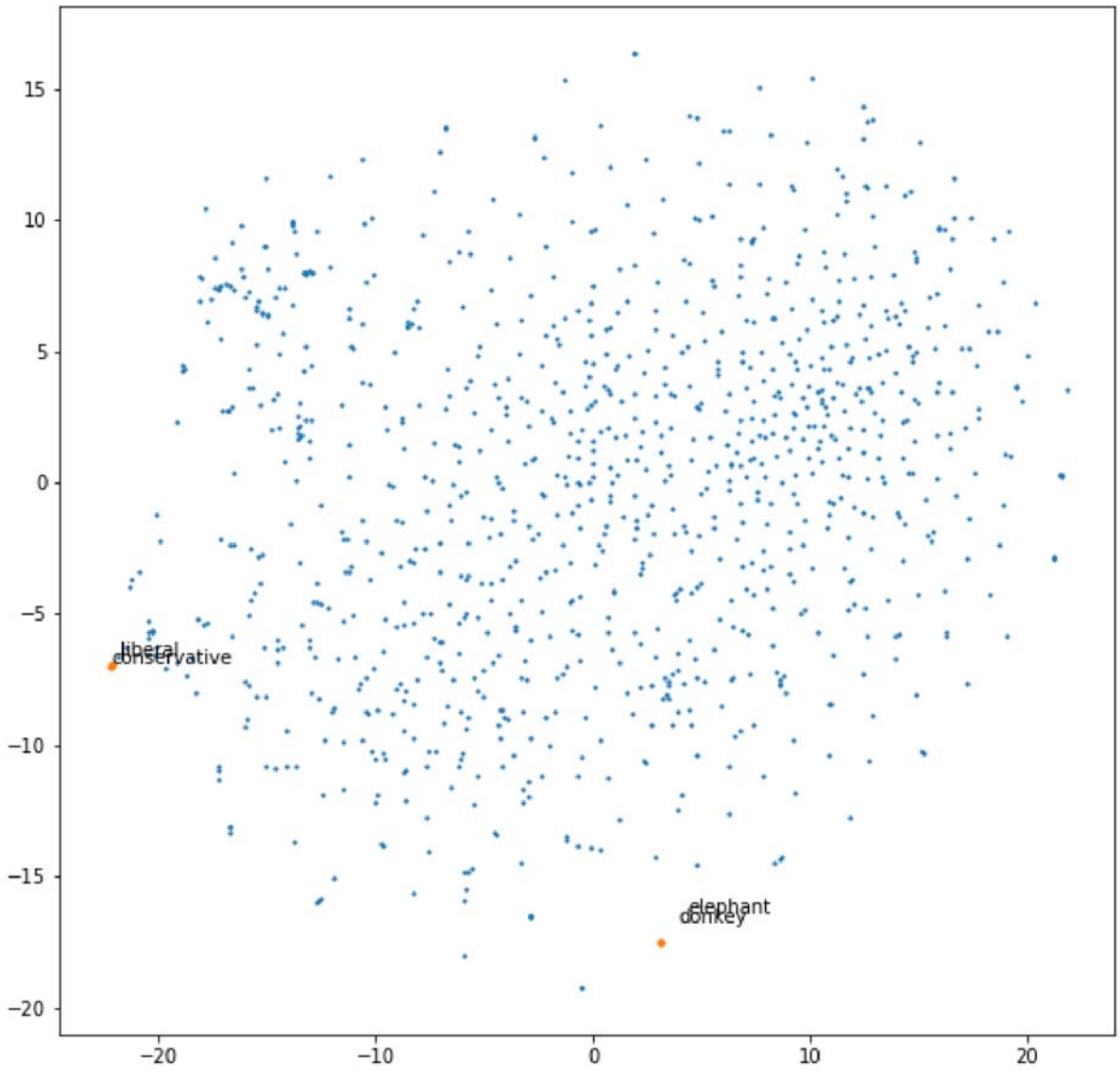
Use the t-SNE dimensionality reduction technique to visualize only the pre-trained GloVe embeddings in two dimensions. The code in `tsne_viz.ipynb` will perform the t-SNE method on a subset of the vocabulary. You can start with this code and modify it as you choose to construct visualizations of the embeddings. Or start with your own version of t-SNE. The code provided shows the relative positions of the words conservative, liberal, elephant, and donkey. Find at least three more examples that produce expected results and three examples that produce surprising results. Include the plots in your write-up. Give reasons why you might see surprising behavior here.

In [34]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# this functions computes and displays the 2-d t-SNE maps for a subset of the embedding
 vectors
# and displays them together with the points for a set of input words.

def display_tsne_neighborhood(model, input_word, nsample=1000, size1=2, size2=10, offset
=5):

    arr = np.empty((0,100), dtype='f')
    word_label = input_word

    # add the vector for each of the closest words to the array
    for w in range(len(input_word)):
        arr = np.append(arr, np.array([model[input_word[w]]]), axis=0)

    voc = [w for w in model.vocab]
    wrds = np.random.choice(range(len(voc)), size=nsample, replace=False)
    for w in wrds:
        wrd_vector = model[voc[w]]
        arr = np.append(arr, np.array([wrd_vector]), axis=0)

    # find tsne coords for 2 dimensions
    tsne = TSNE(n_components=2, random_state=0)
    np.set_printoptions(suppress=True)
    Y = tsne.fit_transform(arr)

    x_coord = Y[:, 0]
    y_coord = Y[:, 1]
    # display scatter plot
    size=2
    plt.scatter(x_coord, y_coord, s=size1)
    plt.scatter(x_coord[0:len(input_word)], y_coord[0:len(input_word)],s=size2)

    # label the input words
    for w in range(len(input_word)):
        plt.annotate(input_word[w], xy=(x_coord[w],y_coord[w]), \
                     xytext=(w*offset,w*offset), textcoords='offset points')
    plt.show()
```

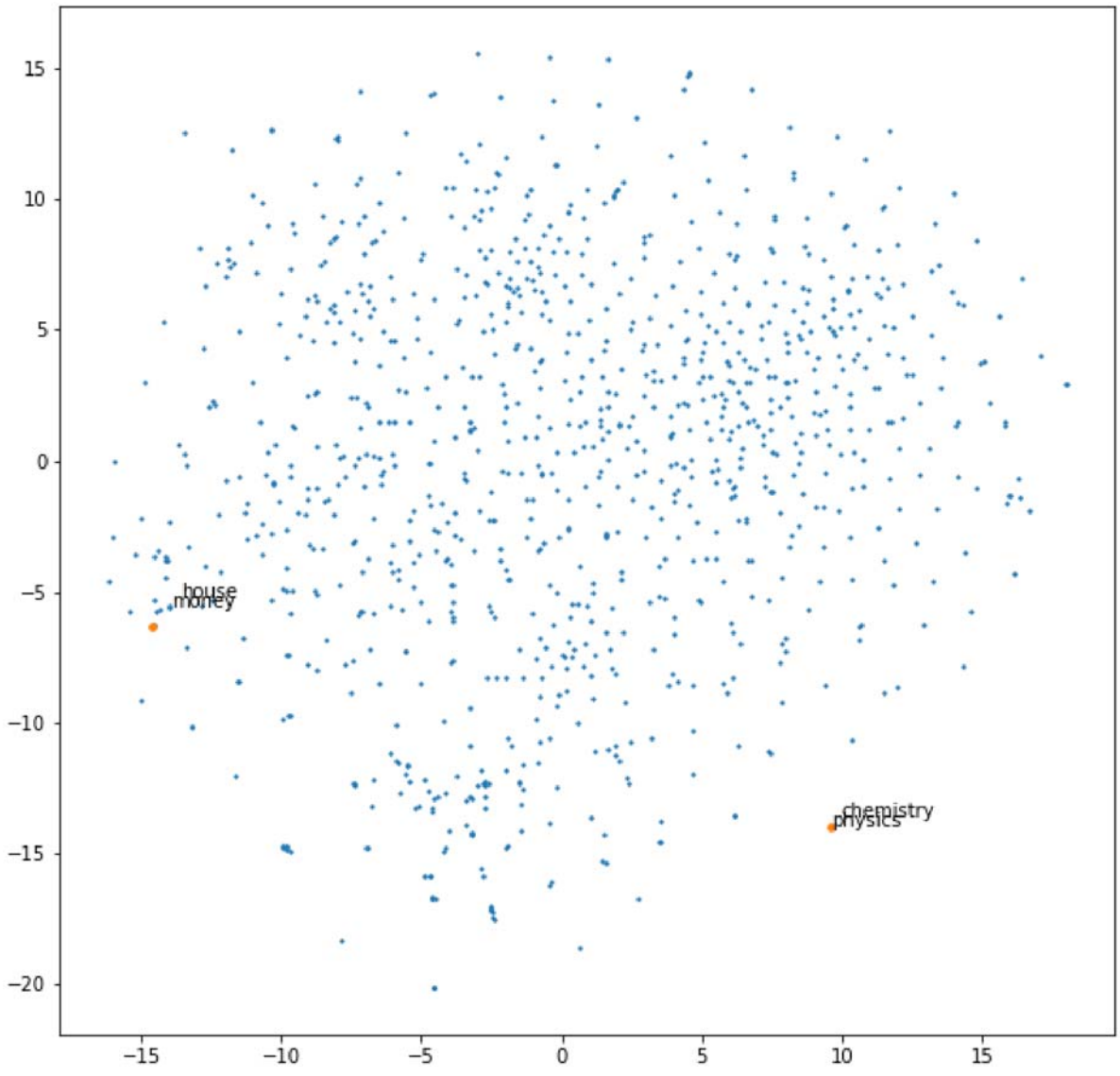
In [35]:
```
print('Graph 1')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['conservative', 'liberal', 'donkey', 'ele
phant'])
```

```
Graph 1
```



In [36]:
```
print('Graph 2')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['physics', 'chemistry', 'money', 'house'
])
```

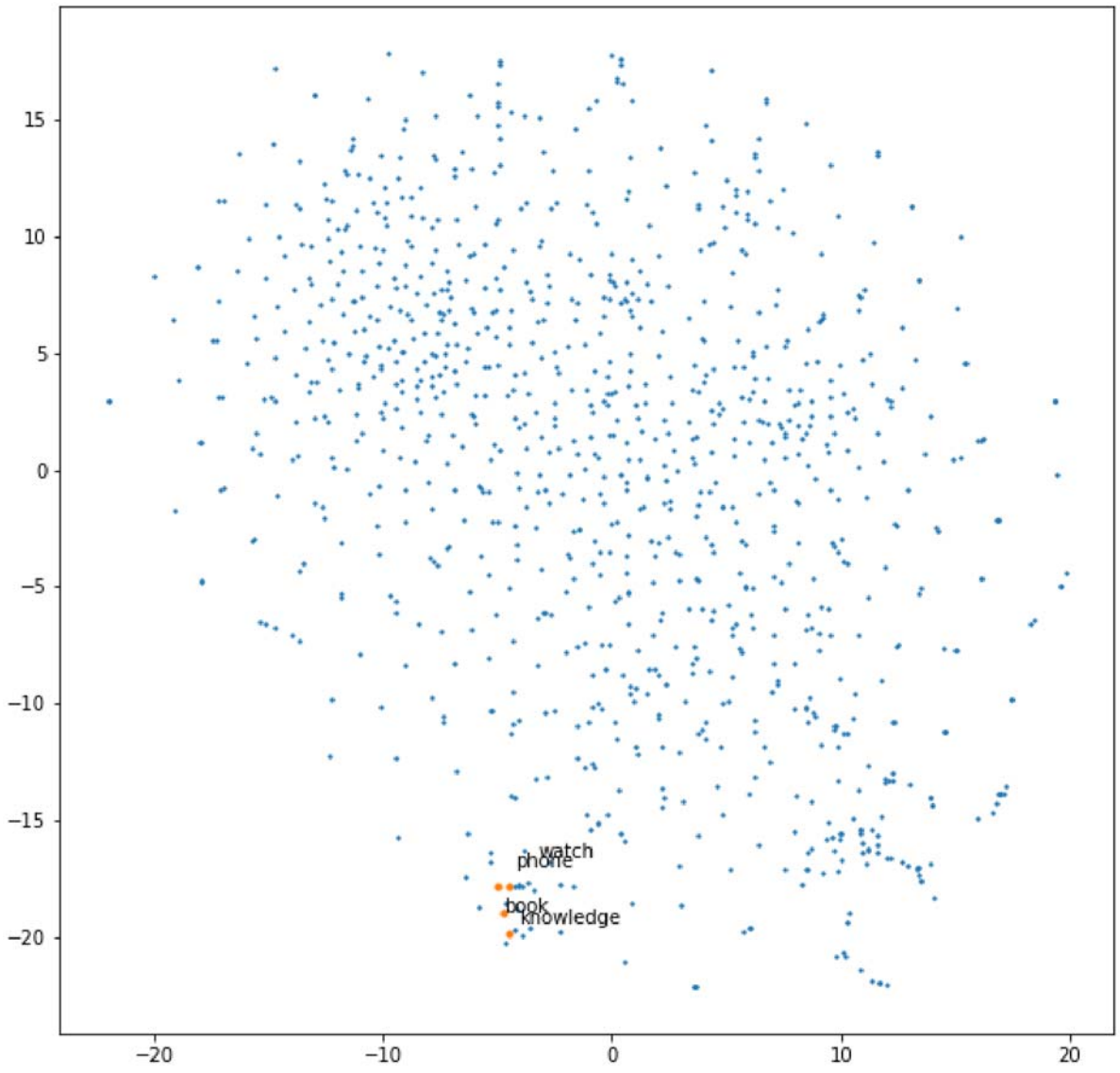
Graph 2



/

In [48]:
```
print('Graph 3')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['book', 'knowledge', 'phone','watch'])
```

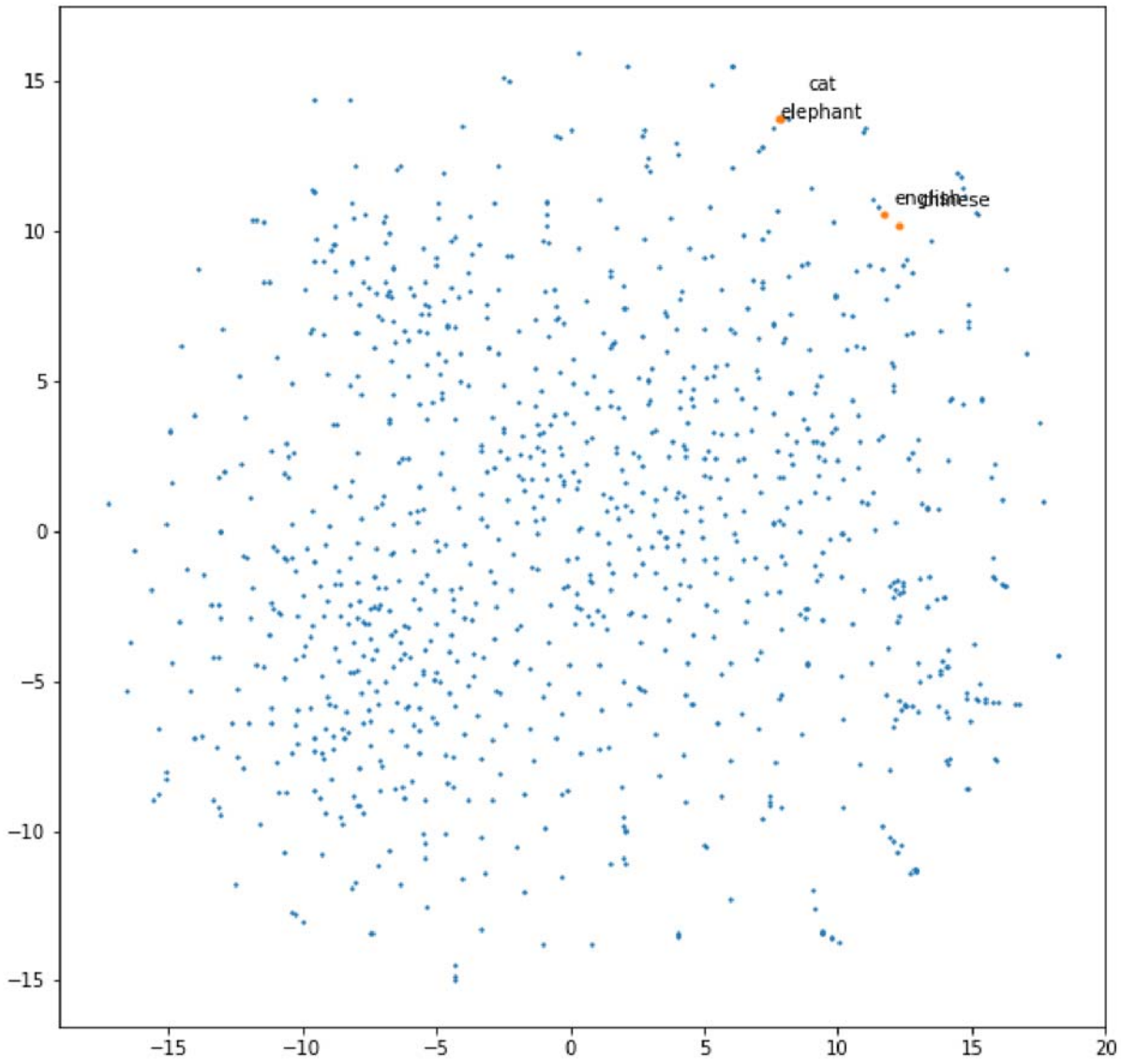
Graph 3



/

In [46]:
```
print('Graph 4')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['elephant','english', 'chinese', 'cat'])
```

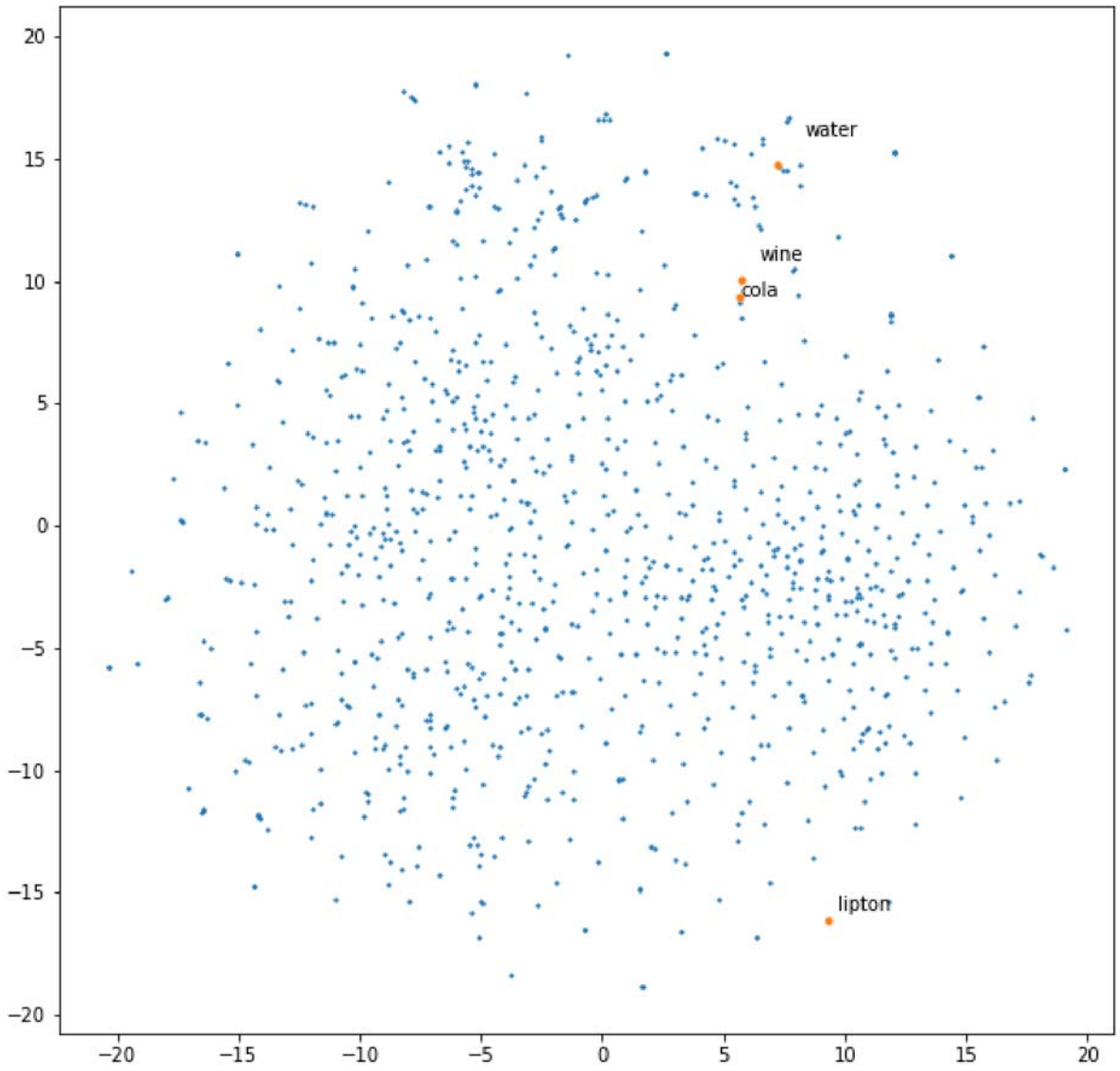
Graph 4



/

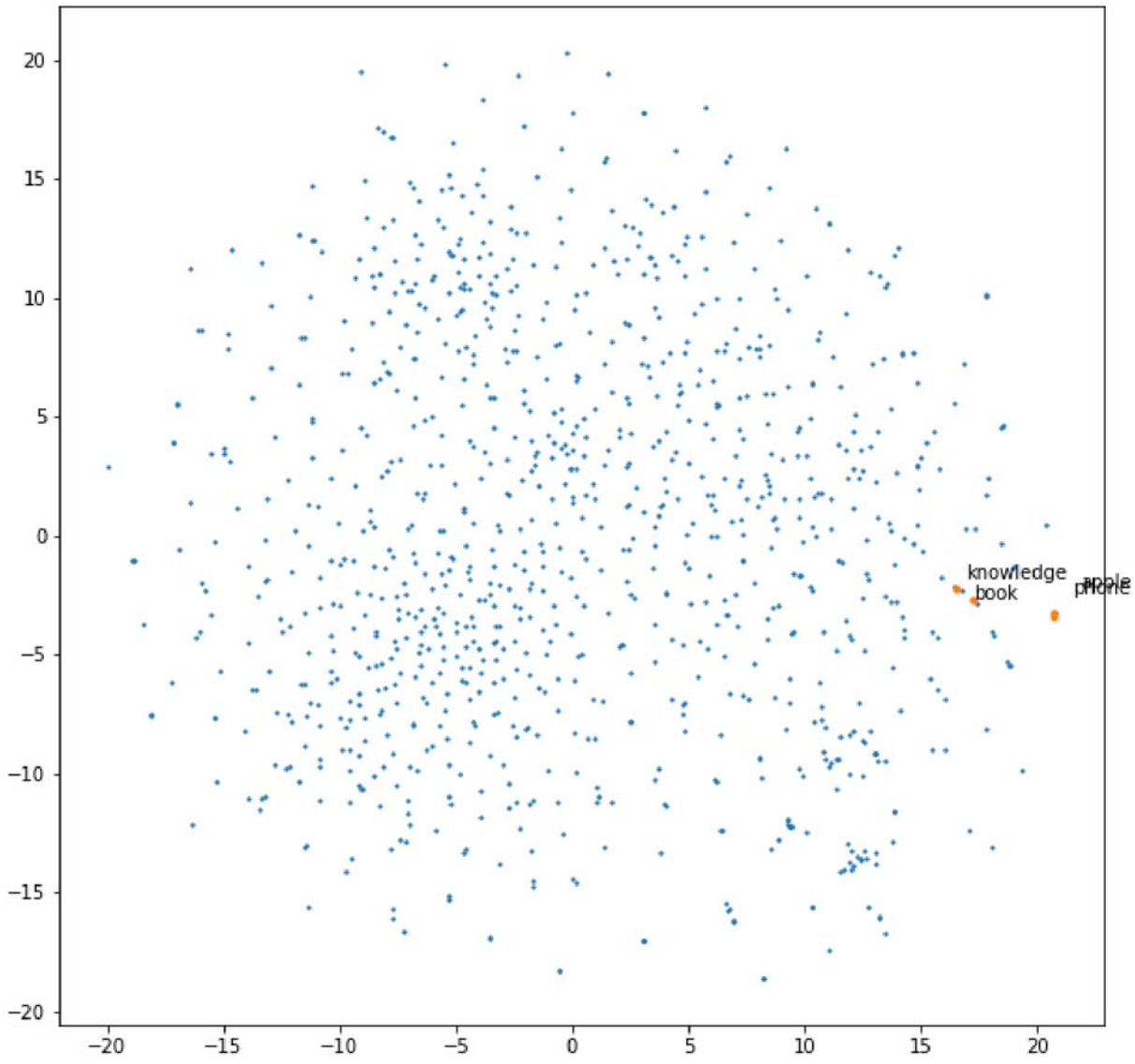
In [44]:
```
print('Graph 5')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['cola', 'lipton', 'wine', 'water'])
```

Graph 5



In [47]:
```
print('Graph 6')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['book', 'knowledge', 'phone','apple'])
```

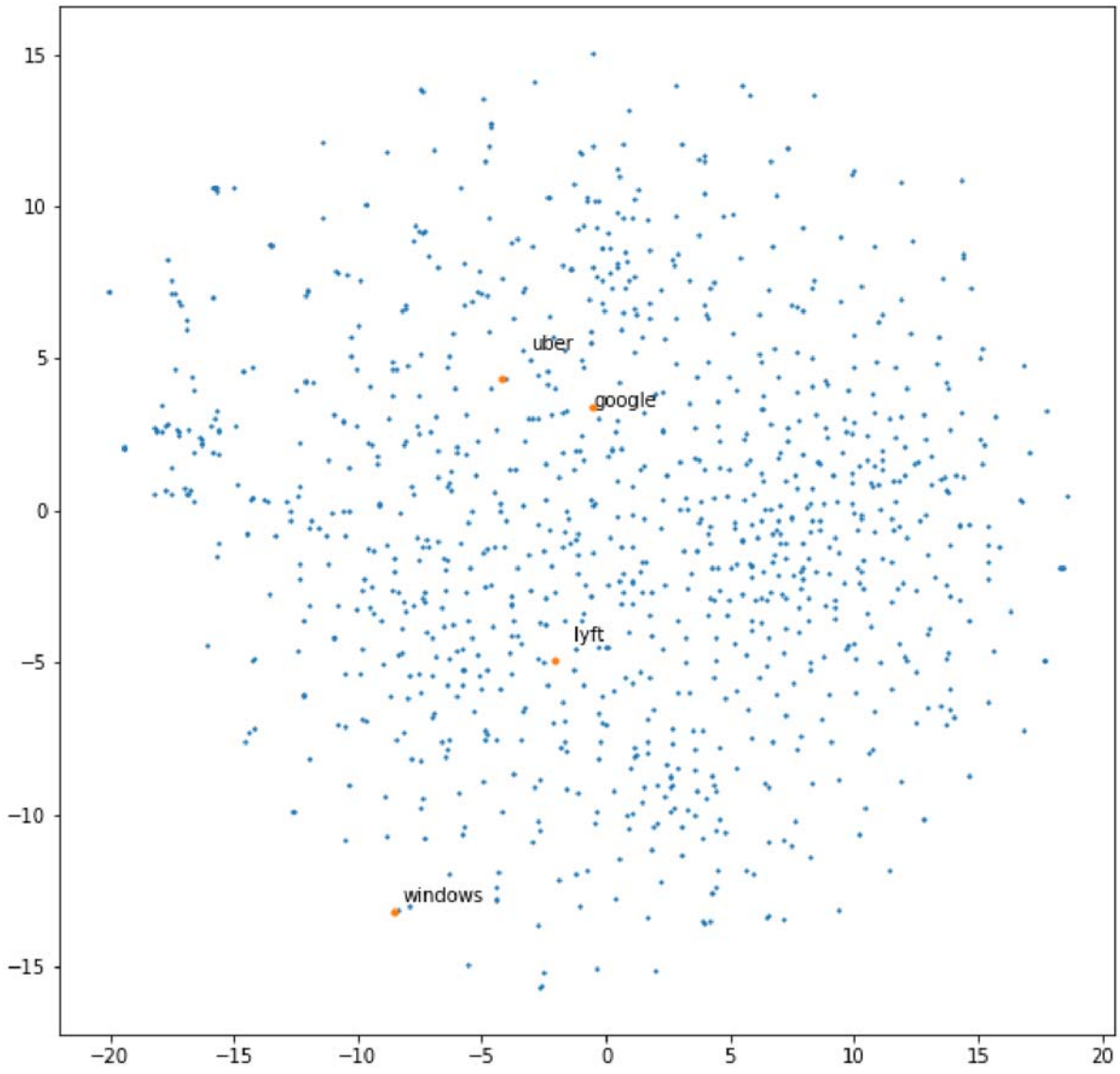
Graph 6



/

In [51]:
```
print('Graph 7')
%matplotlib inline
plt.figure(figsize=(10,10))
display_tsne_neighborhood(glove, input_word = ['google', 'windows', 'lyft', 'uber'])
```

Graph 7



/

Some surprising results:

1. In graph 5, it is suprising that Lipton is so distant from water, wine, and cola.
2. In graph 6, it is suprising that apple is every close to phone. The t-SNE algorithm of GloVe embeddings intelligently consider apple as a high-tech company rather than a type of fruit.
3. In graph 7, it is suprising to see that as high-tech companies, google, windows, lyft, and uber are not very related. Especially uber is very distant from lyft and google is very distant from windows.

This may because that "the t-SNE algorithm calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. It then tries to optimize these two similarity measures using a cost function." [1]

reference: [1] An Introduction to t-SNE with Python Example: https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1 (https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1)

Problem 2: Experiments with Musician Embeddings (15 points)

In this problem, we will use a collection of playlists obtained from last.fm (http://last.fm). We treat each playlist as a document, and each artist in the playlist as a word. By feeding this dataset to word2vec, we will be able to learn artist embeddings.

Artist Embeddings

The following experiments will be done with the playlist data file `playlists.txt`. Each line in this file is a playlist. The integers on each line are unique artist identifiers, indicating which artists were in each playlist. The artists are in `artists.txt`.

2.1 Construct embeddings

The code in `embed_artists.ipynb` constructs artist embeddings with word2vec. The artist names are mapped to id numbers in the playlists; the code maps them back to display the names. Copy that code here and run it.

In [55]:
```
import gensim
from gensim.models import word2vec
import logging

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

In [56]:
```
playlists = word2vec.LineSentence('playlists.txt')
music_model = word2vec.Word2Vec(playlists, size=64, window=100, min_count=10)
```

/

```
2019-11-14 11:15:47,474 : INFO : collecting all words and their counts
2019-11-14 11:15:47,485 : INFO : PROGRESS: at sentence #0, processed 0 words, keeping 0
word types
2019-11-14 11:15:47,533 : INFO : PROGRESS: at sentence #10000, processed 107891 words,
keeping 3250 word types
2019-11-14 11:15:47,568 : INFO : collected 3292 word types from a corpus of 189900 raw
words and 18111 sentences
2019-11-14 11:15:47,569 : INFO : Loading a fresh vocabulary
2019-11-14 11:15:47,573 : INFO : effective_min_count=10 retains 2228 unique words (67%
of original 3292, drops 1064)
2019-11-14 11:15:47,573 : INFO : effective_min_count=10 leaves 183255 word corpus (96%
of original 189900, drops 6645)
2019-11-14 11:15:47,580 : INFO : deleting the raw counts dictionary of 3292 items
2019-11-14 11:15:47,581 : INFO : sample=0.001 downsamples 59 most-common words
2019-11-14 11:15:47,582 : INFO : downsampling leaves estimated 173511 word corpus (94.
7% of prior 183255)
2019-11-14 11:15:47,589 : INFO : estimated required memory for 2228 words and 64 dimens
ions: 2254736 bytes
2019-11-14 11:15:47,589 : INFO : resetting layer weights
2019-11-14 11:15:47,619 : INFO : training model with 3 workers on 2228 vocabulary and 6
4 features, using sg=0 hs=0 sample=0.001 negative=5 window=100
2019-11-14 11:15:47,822 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 11:15:47,832 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 11:15:47,842 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 11:15:47,843 : INFO : EPOCH - 1 : training on 189900 raw words (173519 effec
tive words) took 0.2s, 787233 effective words/s
2019-11-14 11:15:48,033 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 11:15:48,043 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 11:15:48,052 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 11:15:48,053 : INFO : EPOCH - 2 : training on 189900 raw words (173475 effec
tive words) took 0.2s, 835648 effective words/s
2019-11-14 11:15:48,233 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 11:15:48,241 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 11:15:48,251 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 11:15:48,252 : INFO : EPOCH - 3 : training on 189900 raw words (173563 effec
tive words) took 0.2s, 881512 effective words/s
2019-11-14 11:15:48,565 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 11:15:48,580 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 11:15:48,590 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
2019-11-14 11:15:48,590 : INFO : EPOCH - 4 : training on 189900 raw words (173587 effec
tive words) took 0.3s, 517873 effective words/s
2019-11-14 11:15:48,819 : INFO : worker thread finished; awaiting finish of 2 more thre
ads
2019-11-14 11:15:48,824 : INFO : worker thread finished; awaiting finish of 1 more thre
ads
2019-11-14 11:15:48,834 : INFO : worker thread finished; awaiting finish of 0 more thre
ads
```

```
2019-11-14 11:15:48,835 : INFO : EPOCH - 5 : training on 189900 raw words (173535 effec
tive words) took 0.2s, 718198 effective words/s
2019-11-14 11:15:48,835 : INFO : training on a 949500 raw words (867679 effective word
s) took 1.2s, 714455 effective words/s
```

In [57]:
```
music_model.wv['299']
```

Out[57]:
```
array([-0.5889782 ,  0.04839313,  0.65038246, -0.2683968 , -0.80447567,
       -0.8799677 ,  0.16921292, -0.5570287 , -0.10233612,  0.3143244 ,
       -0.2765884 , -0.5444029 ,  0.48616862,  0.20114738, -0.43540907,
       -0.26947975, -1.3908005 , -1.3121594 ,  0.23237251,  0.7343331 ,
        0.00602113,  0.87225723,  0.00210551,  0.73249745,  0.38114515,
        0.4679564 , -0.36392632,  0.38617215, -0.12593956, -0.5920335 ,
        0.8502535 ,  0.21004896, -0.18975297, -0.43768352,  0.4356809 ,
       -0.3725518 ,  0.26975685, -0.271293  , -0.7668501 , -0.967311  ,
       -0.15593605,  0.8910003 , -0.61300117, -0.32896945, -1.5681725 ,
        0.15890098,  0.3262027 ,  0.6320881 ,  0.5222071 ,  0.26328048,
       -0.4999047 , -0.32450387, -0.5031259 ,  0.6236864 ,  0.7586574 ,
       -0.40636873, -0.24505413, -0.34643722,  0.42991439, -0.5839845 ,
        0.6673741 , -0.47845486,  0.35249674,  0.03966712], dtype=float32)
```

In [58]:
```
artist = [art.strip() for art in open('artists.txt', 'r')]
artist[0:10]
```

Out[58]:
```
['Everette Harp',
 'Bishop Paul S. Morton & Aretha Franklin',
 'Frankie Ballard',
 'Herb Alpert',
 'Rod Stewart & Chaka Khan',
 'Scars On 45',
 'New Radicals',
 'Crosby, Stills & Nash',
 'Ledisi',
 'La Quinta Estacion']
```

In [59]:
```
id2name = {}
name2id = {}
for w in range(len(artist)):
    id2name["%s" % w] = artist[w]
    name2id[artist[w]] = "%s" % w

id2name[name2id['Elton John']]
```

Out[59]:
```
'Elton John'
```

2.2 Similar artists

Find the 5 closest artist embedding vectors to the artists "The Beatles", "Lady Gaga", and "Nirvana". Comment on the results.

In [60]:
```
def similar_artists(model, artist, n=5):
    id = name2id[artist]
    out = model.wv.most_similar(id, topn=n)

    print("artists similar to '%s'\n" % artist)
    for i in range(n) :
        name = id2name[out[i][0]]
        print("\t%s" % name)

similar_artists(music_model, 'The Beatles')
```

```
2019-11-14 11:16:00,073 : INFO : precomputing L2-norms of word weight vectors

artists similar to 'The Beatles'

        The Electric Light Orchestra
        Grand Funk Railroad
        Steppenwolf
        T. Rex
        Creedence Clearwater Revival
```

In [61]:
```
similar_artists(music_model, 'Lady Gaga')
```

```
artists similar to 'Lady Gaga'

        Katy Perry
        Bruno Mars
        Black Eyed Peas
        Ke$ha
        Taio Cruz
```

In [62]:
```
similar_artists(music_model, 'Nirvana')
```

```
artists similar to 'Nirvana'

        Temple Of The Dog
        Stone Temple Pilots
        Bush
        Deftones
        Live
```

The 5 closest artist embedding vectors are very similar to the style of the artists I put in.

2.3 Visualize embeddings

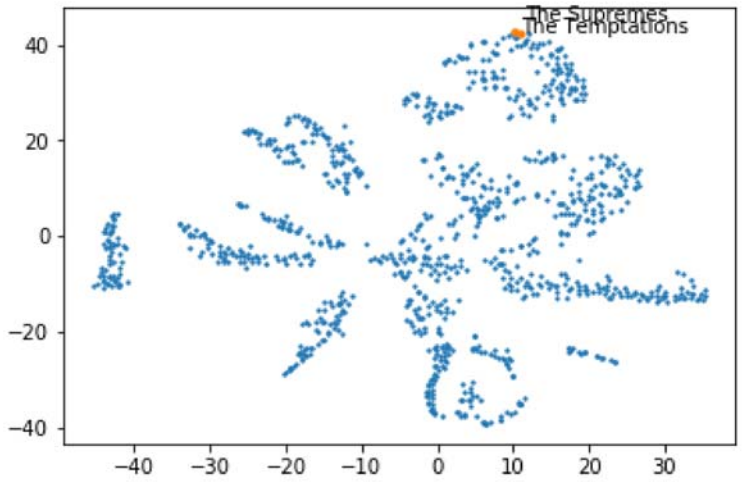
As for the word embeddings question, use the t-SNE dimensionality reduction technique to visualize the artist embeddings. After running t-SNE on the artist embeddings, try visualizing "The Temptations" and "The Supremes" together. Find a few more examples that you think are interesting and include the plots in your write-up. Comment on your findings.

In [63]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE


# this functions computes and displays the 2-d t-SNE maps for a subset of the embedding
 vectors
# and displays them together with the points for a set of input words.

def display_tsne_artists(model, artists, nsample=1000, size1=2, size2=10, offset=5):

    arr = np.empty((0,64), dtype='f')

    # add the vector for each of the closest words to the array
    for a in range(len(artists)):
        id = name2id[artists[a]]
        arr = np.append(arr, np.array([model[id]]), axis=0)

    voc = [w for w in model.wv.vocab]
    ids = np.random.choice(range(len(voc)), size=nsample, replace=False)
    for w in ids:
        wrd_vector = model[voc[w]]
        arr = np.append(arr, np.array([wrd_vector]), axis=0)

    # find tsne coords for 2 dimensions
    tsne = TSNE(n_components=2, random_state=0)
    np.set_printoptions(suppress=True)
    Y = tsne.fit_transform(arr)

    x_coord = Y[:, 0]
    y_coord = Y[:, 1]
    # display scatter plot
    size=2
    plt.scatter(x_coord, y_coord, s=size1)
    plt.scatter(x_coord[0:len(artists)], y_coord[0:len(artists)],s=size2)

    # label the input words
    for w in range(len(artists)):
        plt.annotate(artists[w], xy=(x_coord[w],y_coord[w]), \
                     xytext=(w*offset,w*offset), textcoords='offset points')
    plt.show()
```

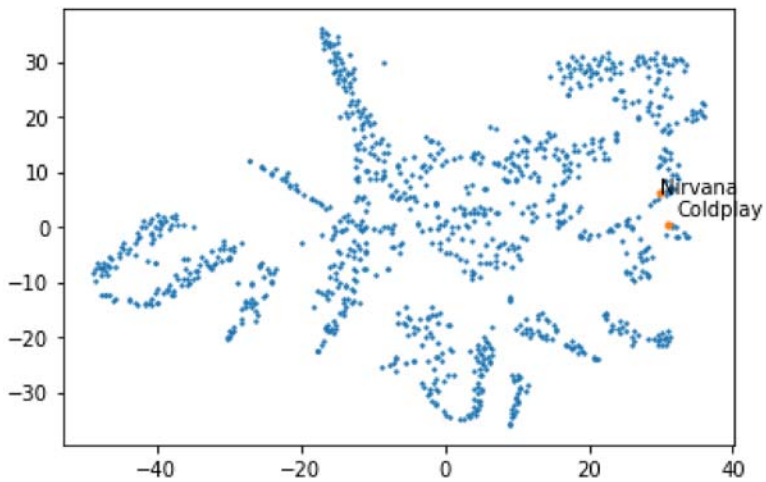
In [69]:
```
print("Graph 1")
display_tsne_artists(music_model, ['The Temptations', 'The Supremes'])
```

```
Graph 1

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



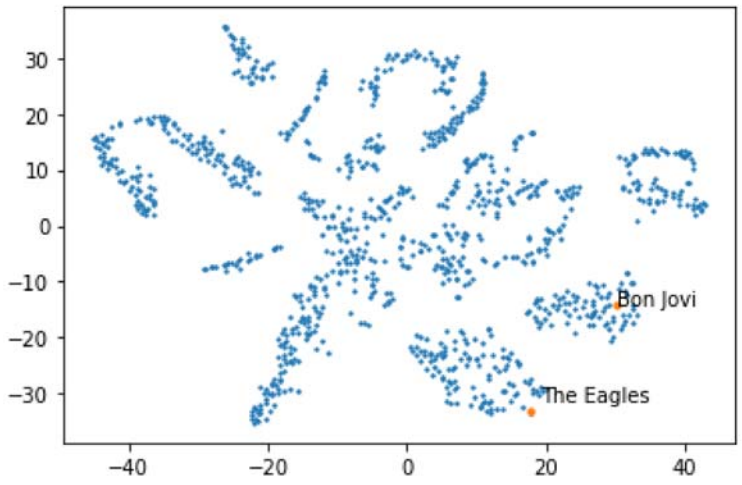
In [70]:
```
print("Graph 2")
display_tsne_artists(music_model, ['Nirvana', 'Coldplay'])
```

```
Graph 2

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



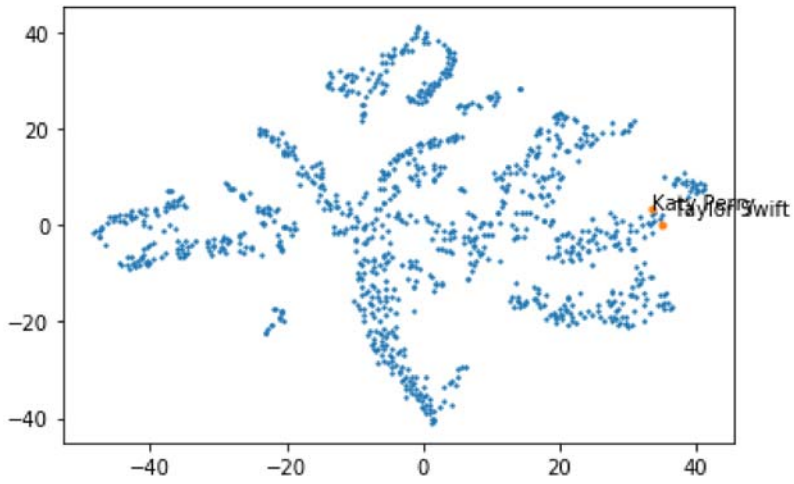
In [71]:
```
print("Graph 3")
display_tsne_artists(music_model, ['Bon Jovi', 'The Eagles'])
```

```
Graph 3

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



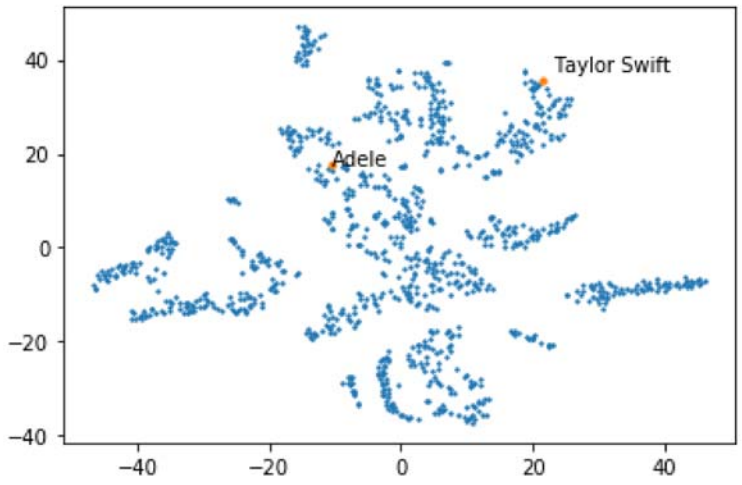
In [72]:
```
print("Graph 4")
display_tsne_artists(music_model, ['Katy Perry', 'Taylor Swift'])
```

```
Graph 4

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



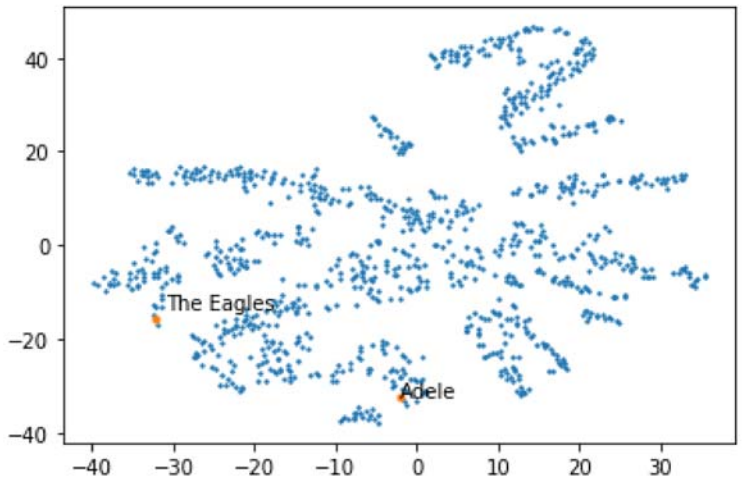
In [73]:
```
print("Graph 5")
display_tsne_artists(music_model, ['Adele', 'Taylor Swift'])
```

```
Graph 5

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



In [74]:
```
print("Graph 6")
display_tsne_artists(music_model, ['Adele', 'The Eagles'])
```

```
Graph 6

C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
C:\Users\kavan\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarnin
g: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__get
item__() instead).
```



Bands/singers with a similar music style are very close to each other in the graphs. Bands/singers with different music styles are more distant. Distance shown in the lower demension reflects the difference in the higher demension. For example, Taylor Swift and Katy Perry are sloser than Adele.