Seminar 1: Fun with Word Embeddings (3 points)

Today we gonna play with word embeddings: train our own little embedding, load one from gensim model zoo and use it to visualize text corpora.

This whole thing is gonna happen on top of embedding dataset.

Requirements: `pip install --upgrade nltk gensim bokeh` , but only if you're running locally.

```
# download the data:
!wget https://www.dropbox.com/s/obaitrix9jyu84r/quora.txt?dl=1 -O ./quora.txt
# alternative download link: https://yadi.sk/i/BPQrUu1NaTduEw
```

```
--2023-09-24 02:07:54--  https://www.dropbox.com/s/obaitrix9jyu84r/quora.txt?dl=1
Resolving www.dropbox.com (www.dropbox.com)... 162.125.70.18, 2620:100:6026:18::a27d:4612
Connecting to www.dropbox.com (www.dropbox.com)|162.125.70.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/dl/obaitrix9jyu84r/quora.txt [following]
--2023-09-24 02:07:55--  https://www.dropbox.com/s/dl/obaitrix9jyu84r/quora.txt
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com/cd/0/get/CET8E5mqLLk67mR6x3ricDaFU91LaZe9gKPxZFIDhFTbtm6Pqr2N9zXMuTPETgkN3o3J7S9ZA-gl8F16qFMBhvLicLm-6fj2AU3iH5W2y-TO9cu3j2-h9OWdQaUSkFwnxxY/file?dl=1# [following]
--2023-09-24 02:07:55--  https://ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com/cd/0/get/CET8E5mqLLk67mR6x3ricDaFU91LaZe9gKPxZFIDhFTbtm6Pqr2N9zXMuTPETgkN3o3J7S9ZA-gl8F16qFMBhvLicLm-6fj2AU3iH5W2y-TO9cu3j2-h9OWdQaUSkFwnxxY/file?dl=1
Resolving ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com (ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com)... 162.125.70.15, 2620:100:6026:15::a27d:460f
Connecting to ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com (ucb74a57ee744538b66124a6b9ac.dl.dropboxusercontent.com)|162.125.70.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 33813903 (32M) [application/binary]
Saving to: './quora.txt'

./quora.txt         100%[===================>]  32.25M  8.54MB/s    in 4.0s

2023-09-24 02:08:00 (8.15 MB/s) - './quora.txt' saved [33813903/33813903]
```

```
import numpy as np

with open("./quora.txt", encoding="utf-8") as file:
    data = list(file)

data[50]
```

```
"What TV shows or books help you read people's body language?\n"
```

Tokenization: a typical first step for an nlp task is to split raw data into words. The text we're working with is in raw format: with all the punctuation and smiles attached to some words, so a simple str.split won't do.

Let's use `nltk` - a library that handles many nlp tasks like tokenization, stemming or part-of-speech tagging.

```
from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()

print(tokenizer.tokenize(data[50]))
```

```
['What', 'TV', 'shows', 'or', 'books', 'help', 'you', 'read', 'people', "'", 's', 'body', 'language', '?']
```

```
# TASK: lowercase everything and extract tokens with tokenizer.
# data_tok should be a list of lists of tokens for each line in data.

data_tok = [tokenizer.tokenize(x.lower()) for x in data]
```

```
assert all(isinstance(row, (list, tuple)) for row in data_tok), "please convert each line into a list of tokens (strings)"
assert all(all(isinstance(tok, str) for tok in row) for row in data_tok), "please convert each line into a list of tokens (strings)"
is_latin = lambda tok: all('a' <= x.lower() <= 'z' for x in tok)
assert all(map(lambda l: not is_latin(l) or l.islower(), map(' '.join, data_tok))), "please make sure to lowercase the data"
```

```
print([' '.join(row) for row in data_tok[:2]])
```

```
["can i get back with my ex even though she is pregnant with another guy ' s baby ?", 'what are some ways to overcome a fast food addiction ?']
```

Word vectors: as the saying goes, there's more than one way to train word embeddings. There's Word2Vec and GloVe with different objective functions. Then there's fasttext that uses character-level models to train word embeddings.

The choice is huge, so let's start someplace small: **gensim** is another nlp library that features many vector-based models incuding word2vec.

```
from gensim.models import Word2Vec
model = Word2Vec(data_tok,
                 vector_size=32,      # embedding vector size
                 min_count=5,  # consider words that occured at least 5 times
                 window=5).wv  # define context as a 5-word window around the target word
```

```
# now you can get word vectors !
model.get_vector('anything')
```

```
array([-3.5460231 ,  1.7865689 ,  0.3841988 ,  2.7576609 ,  2.0146608 ,
        1.9734926 ,  1.8941041 , -5.0200357 ,  0.48848522,  2.4279432 ,
       -0.7554728 ,  2.0790002 ,  3.2052414 , -0.16385323,  2.6689763 ,
       -1.8514861 , -0.02959771, -1.6357832 ,  0.5121673 , -1.3257158 ,
       -2.1526604 , -0.4395907 , -1.0803162 , -3.99636   ,  2.299466  ,
       -2.3926432 ,  0.86943024,  1.9492018 ,  0.8349378 ,  0.50075203,
        0.8442198 ,  0.9906684 ], dtype=float32)
```

```
# or query similar words directly. Go play with it!
model.most_similar('bread')
```

```
[('rice', 0.956332266330719),
 ('sauce', 0.9481409192085266),
 ('vodka', 0.9390170574188232),
 ('cheese', 0.9327400326728821),
 ('butter', 0.9313923120498657),
 ('fruit', 0.920593798160553),
 ('beans', 0.9199300408363342),
 ('banana', 0.916680097579956),
 ('potato', 0.915650486946106),
 ('wine', 0.9098671674728394)]
```

Using pre-trained model

Took it a while, huh? Now imagine training life-sized (100~300D) word embeddings on gigabytes of text: wikipedia articles or twitter posts.

Thankfully, nowadays you can get a pre-trained word embedding model in 2 lines of code (no sms required, promise).

```
import gensim.downloader as api
model = api.load('glove-twitter-100')
```

```
model.most_similar(positive=["coder", "money"], negative=["brain"])
```

```
[('broker', 0.5820155739784241),
 ('bonuses', 0.5424473285675049),
 ('banker', 0.5385112762451172),
 ('designer', 0.5197198390960693),
 ('merchandising', 0.4964233338832855),
 ('treet', 0.4922019839286804),
 ('shopper', 0.4920562207698822),
 ('part-time', 0.4912828207015991),
 ('freelance', 0.4843311905860901),
 ('aupair', 0.4796452522277832)]
```

Visualizing word vectors

One way to see if our vectors are any good is to plot them. Thing is, those vectors are in 30D+ space and we humans are more used to 2-3D.

Luckily, we machine learners know about **dimensionality reduction** methods.

Let's use that to plot 1000 most frequent words

```
words = model.index_to_key[:1000]

print(words[::100])
```

```
['<user>', '_', 'please', 'apa', 'justin', 'text', 'hari', 'playing', 'once', 'sei']
```

```
# for each word, compute it's vector with model
word_vectors = np.array([model.get_vector(x) for x in words])
```

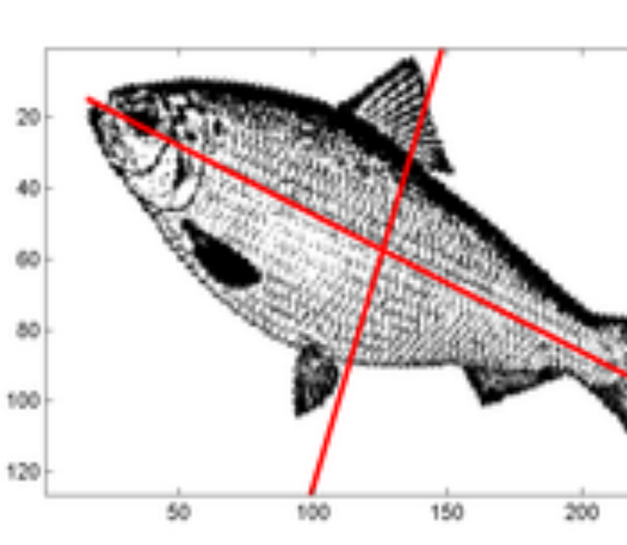
```
# word_vectors
```

```
assert isinstance(word_vectors, np.ndarray)
assert word_vectors.shape == (len(words), 100)
assert np.isfinite(word_vectors).all()
```

Linear projection: PCA

The simplest linear dimensionality reduction method is __P__rincipial __C__omponent __A__nalysis.

In geometric terms, PCA tries to find axes along which most of the variance occurs. The "natural" axes, if you wish.



Under the hood, it attempts to decompose object-feature matrix X into two smaller matrices: W and $\hat{W}$ minimizing *mean squared error*:

$$\|(XW)\hat{W} - X\|_2^2 \to_{W,\hat{W}} \min$$

- $X \in \mathbb{R}^{n \times m}$ - object matrix (**centered**);
- $W \in \mathbb{R}^{m \times d}$ - matrix of direct transformation;
- $\hat{W} \in \mathbb{R}^{d \times m}$ - matrix of reverse transformation;
- n samples, m original dimensions and d target dimensions;

```
from sklearn.decomposition import PCA

# map word vectors onto 2d plane with PCA. Use good old sklearn api (fit, transform)
# after that, normalize vectors to make sure they have zero mean and unit variance
word_vectors_pca = PCA(n_components=2).fit_transform(word_vectors)

# and maybe MORE OF YOUR CODE here :)
```

```
word_vectors_pca = (word_vectors_pca - word_vectors_pca.mean(axis=0)) / word_vectors_pca.std(axis=0)
```

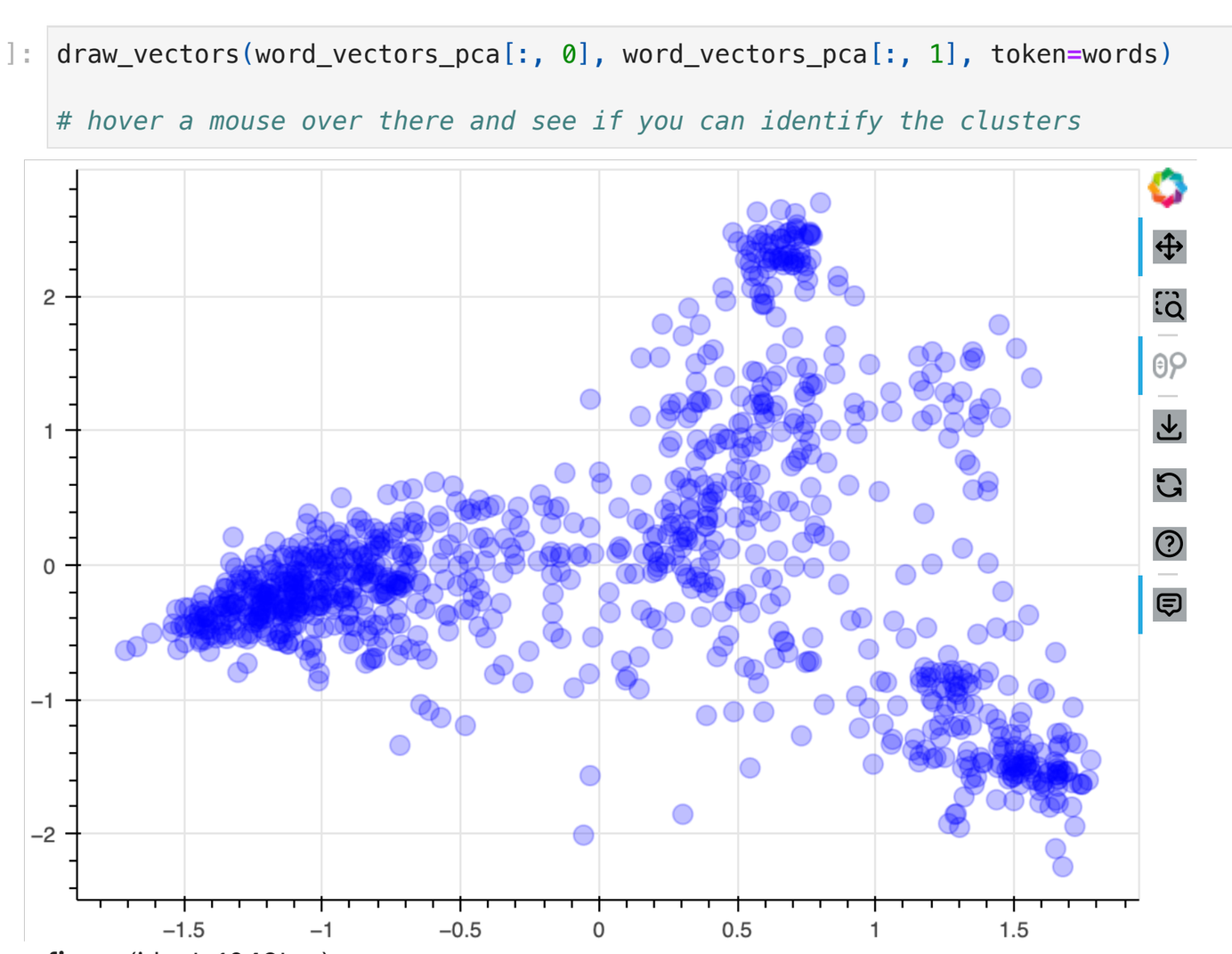
```
assert word_vectors_pca.shape == (len(word_vectors), 2), "there must be a 2d vector for each word"
assert max(abs(word_vectors_pca.mean(0))) < 1e-5, "points must be zero-centered"
assert max(abs(1.0 - word_vectors_pca.std(0))) < 1e-2, "points must have unit variance"
```

Let's draw it!

```
import bokeh.models as bm, bokeh.plotting as pl
from bokeh.io import output_notebook
output_notebook()

def draw_vectors(x, y, radius=10, alpha=0.25, color='blue',
                 width=600, height=400, show=True, **kwargs):
    """ draws an interactive plot for data points with auxilirary info on hover """
    if isinstance(color, str): color = [color] * len(x)
    data_source = bm.ColumnDataSource({ 'x' : x, 'y' : y, 'color': color, **kwargs })

    fig = pl.figure(active_scroll='wheel_zoom', width=width, height=height)
    fig.scatter('x', 'y', size=radius, color='color', alpha=alpha, source=data_source)

    fig.add_tools(bm.HoverTool(tooltips=[(key, "@" + key) for key in kwargs.keys()]))
    if show: pl.show(fig)
    return fig
```

BokehJS 3.2.2 successfully loaded.

```
draw_vectors(word_vectors_pca[:, 0], word_vectors_pca[:, 1], token=words)

# hover a mouse over there and see if you can identify the clusters
```



figure(id = 'p1049', ...)

Visualizing neighbors with t-SNE

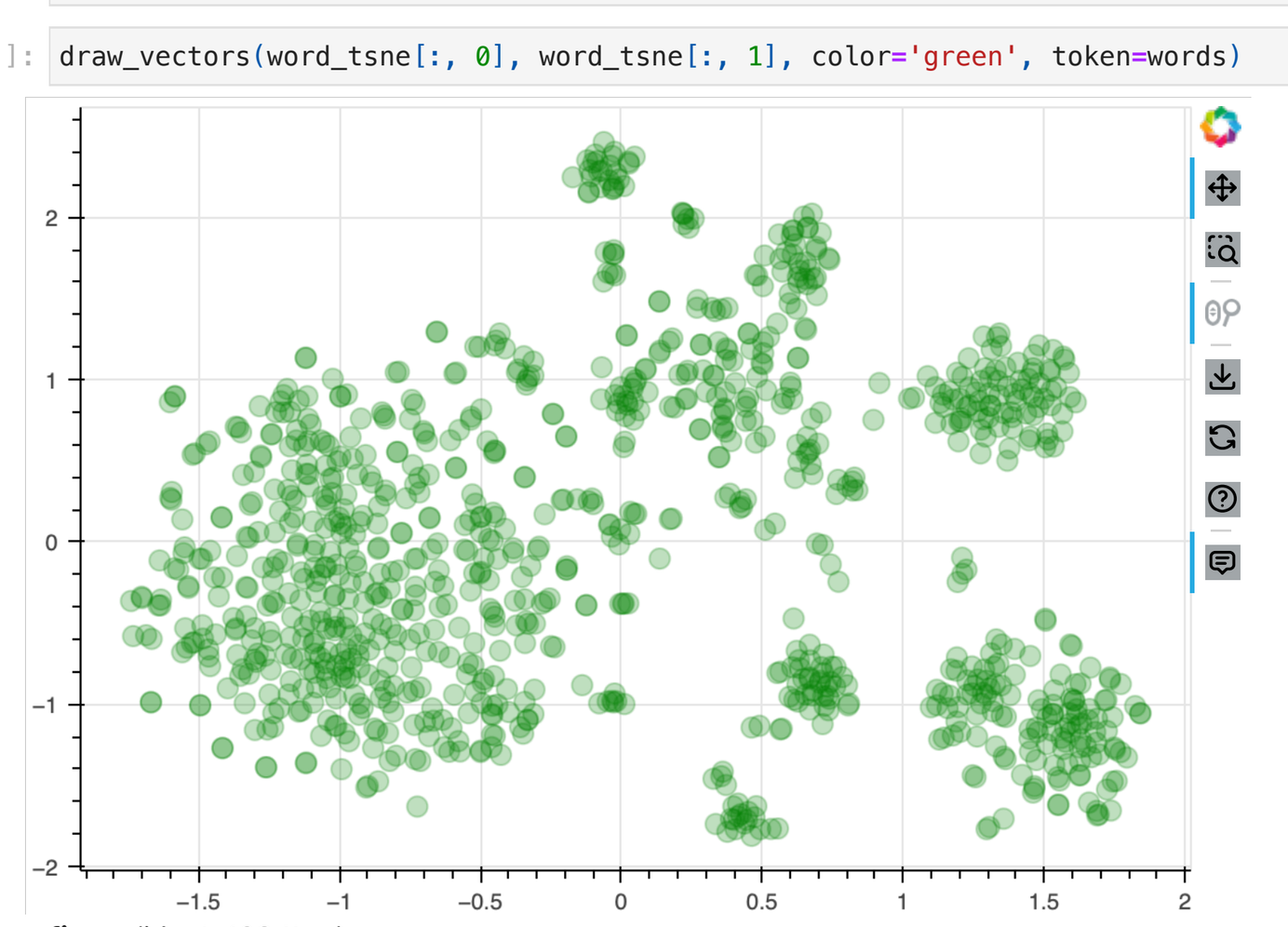
PCA is nice but it's strictly linear and thus only able to capture coarse high-level structure of the data.

If we instead want to focus on keeping neighboring points near, we could use TSNE, which is itself an embedding method. Here you can read **more on TSNE**.

```
from sklearn.manifold import TSNE

# map word vectors onto 2d plane with TSNE. hint: don't panic it may take a minute or two to fit.
# normalize them as just lke with pca


word_tsne = TSNE(n_components=2).fit_transform(word_vectors)
```

```
word_tsne = (word_tsne - word_tsne.mean(axis=0)) / word_tsne.std(axis=0)
```

```
draw_vectors(word_tsne[:, 0], word_tsne[:, 1], color='green', token=words)
```



figure(id = 'p1094', ...)

Visualizing phrases

Word embeddings can also be used to represent short phrases. The simplest way is to take **an average** of vectors for all tokens in the phrase with some weights.

This trick is useful to identify what data are you working with: find if there are any outliers, clusters or other artefacts.

Let's try this new hammer on our data!

```
def get_phrase_embedding(phrase):
    """
    Convert phrase to a vector by aggregating it's word embeddings. See description above.
    """
    # 1. lowercase phrase
    # 2. tokenize phrase
    # 3. average word vectors for all words in tokenized phrase
    # skip words that are not in model's vocabulary
    # if all words are missing from vocabulary, return zeros

    vector = np.zeros([model.vector_size], dtype='float32')

    word_vectors = np.array([model.get_vector(x, None) for x in tokenizer.tokenize(phrase.lower()) if x in model])
    if len(word_vectors)==0:
        return vector
    vector = word_vectors.mean(axis=0)

    return vector
```

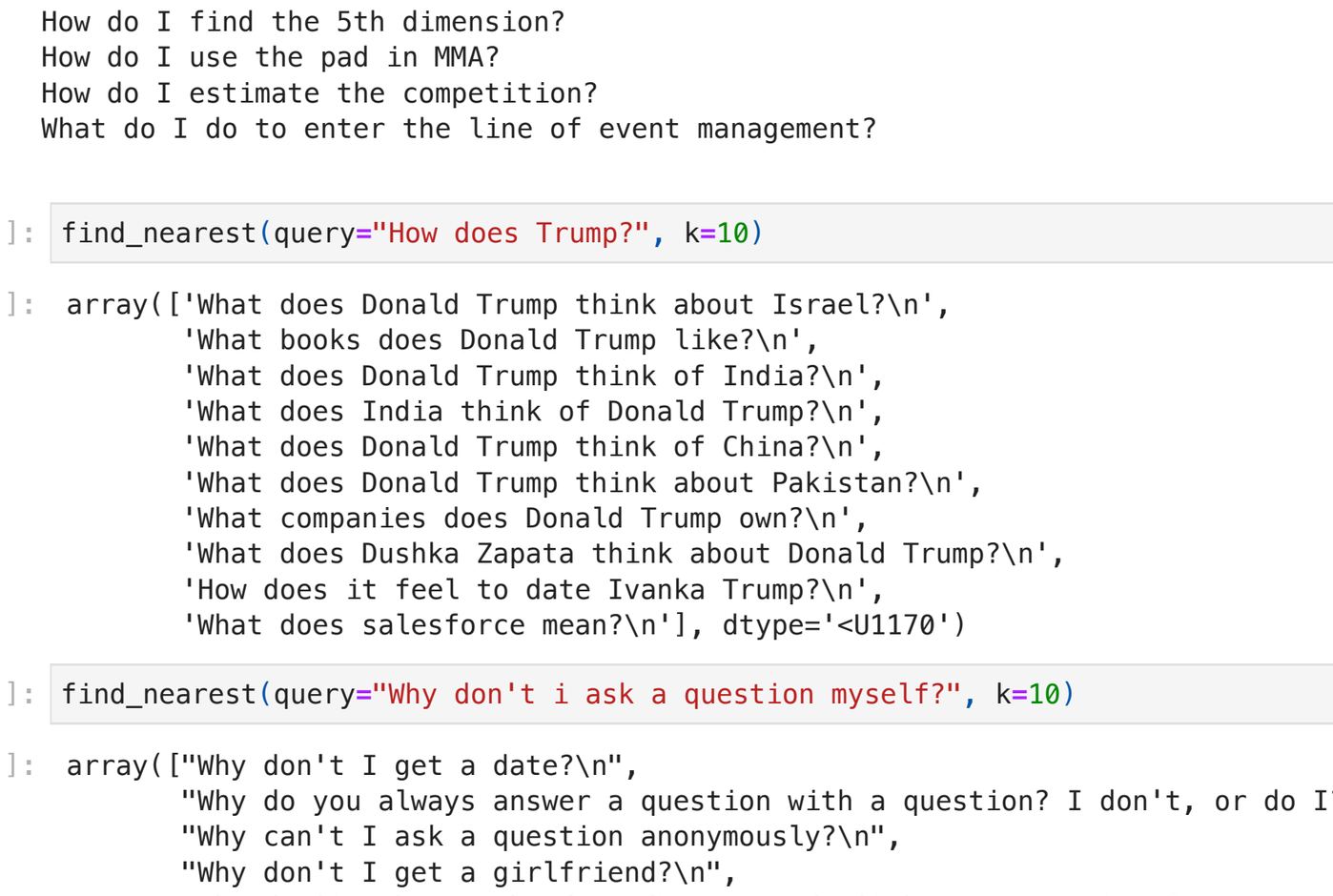
```
vector = get_phrase_embedding("I'm very sure. This never happened to me before...")

assert np.allclose(vector[::10],
                   np.array([ 0.31807372, -0.02558171,  0.0933293 , -0.1002182 , -1.0278689 ,
                             -0.16621883,  0.05083408,  0.17989802,  1.3701859 ,  0.08655966],
                              dtype=np.float32))
```

```
# let's only consider ~5k phrases for a first run.
chosen_phrases = data[::len(data) // 1000]

# compute vectors for chosen phrases
phrase_vectors = np.array([get_phrase_embedding(x) for x in chosen_phrases])
```

```
assert isinstance(phrase_vectors, np.ndarray) and np.isfinite(phrase_vectors).all()
assert phrase_vectors.shape == (len(chosen_phrases), model.vector_size)
```

```
# map vectors into 2d space with pca, tsne or your other method of choice
# don't forget to normalize

phrase_vectors_2d = TSNE().fit_transform(phrase_vectors)

phrase_vectors_2d = (phrase_vectors_2d - phrase_vectors_2d.mean(axis=0)) / phrase_vectors_2d.std(axis=0)
```

```
draw_vectors(phrase_vectors_2d[:, 0], phrase_vectors_2d[:, 1],
             phrase=[phrase[:50] for phrase in chosen_phrases],
             radius=20,)
```



figure(id = 'p1184', ...)

Finally, let's build a simple "similar question" engine with phrase embeddings we've built.

```
# compute vector embedding for all lines in data
data_vectors = np.array([get_phrase_embedding(l) for l in data])
```

```
model.similar_by_vector(get_phrase_embedding('Hello where are you'))
```

```
[('you', 0.949736475944519),
 ('there', 0.9299634099006653),
 ("'re", 0.9214767813682556),
 ('know', 0.9209698438644409),
 ('where', 0.9189091324806213),
 ('what', 0.9101167917251587),
 ('how', 0.9075482487678528),
 ('are', 0.9031805992126465),
 ("n't", 0.9030184745788574),
 ('why', 0.8966574668884277)]
```

```
from sklearn.metrics.pairwise import cosine_similarity

def find_nearest(query, k=10):
    """
    given text line (query), return k most similar lines from data, sorted from most to least similar
    similarity should be measured as cosine between query and line embedding vectors
    hint: it's okay to use global variables: data and data_vectors. see also: np.argpartition, np.argsort
    """
    # YOUR CODE
    query_emb = get_phrase_embedding(query)
    cosines = cosine_similarity(query_emb.reshape(1, -1), data_vectors)[0]
    top_k_indexes = np.argsort(cosines)[::-1][:k]
    top_k_phrases = np.array(data)[top_k_indexes]
    return top_k_phrases
```

```
results = find_nearest(query="How do i enter the matrix?", k=10)

print(''.join(results))

assert len(results) == 10 and isinstance(results[0], str)
assert results[0] == 'How do I get to the dark web?\n'
assert results[3] == 'What can I do to save the world?\n'
```

```
How do I get to the dark web?
What should I do to enter hollywood?
How do I use the Greenify app?
What can I do to save the world?
How do I win this?
How do I think out of the box? How do I learn to think out of the box?
How do I find the 5th dimension?
How do I use the pad in MMA?
How do I estimate the competition?
What do I do to enter the line of event management?
```

```
find_nearest(query="How does Trump?", k=10)
```

```
array(['What does Donald Trump think about Israel?\n',
       'What books does Donald Trump like?\n',
       'What does Donald Trump think of India?\n',
       'What does India think of Donald Trump?\n',
       'What does Donald Trump think of China?\n',
       'What does Donald Trump think about Pakistan?\n',
       'What companies does Donald Trump own?\n',
       'What does Dushka Zapata think about Donald Trump?\n',
       'How does it feel to date Ivanka Trump?\n',
       'What does salesforce mean?\n'], dtype='<U1170')
```

```
find_nearest(query="Why don't i ask a question myself?", k=10)
```

```
array(["Why don't I get a date?\n",
       "Why do you always answer a question with a question? I don't, or do I?\n",
       "Why can't I ask a question anonymously?\n",
       "Why don't I get a girlfriend?\n",
       "Why don't I have a boyfriend?\n", "I don't have no question?\n",
       "Why can't I take a joke?\n", "Why don't I ever get a girl?\n",
       "Can I ask a girl out that I don't know?\n",
       "Why don't I have a girlfriend?\n"], dtype='<U1170')
```

Now what?

- Try running TSNE on all data, not just 1000 phrases
- See what other embeddings are there in the model zoo: `gensim.downloader.info()`
- Take a look at FastText embeddings
- Optimize find_nearest with locality-sensitive hashing: use nearpy or `sklearn.neighbors`.

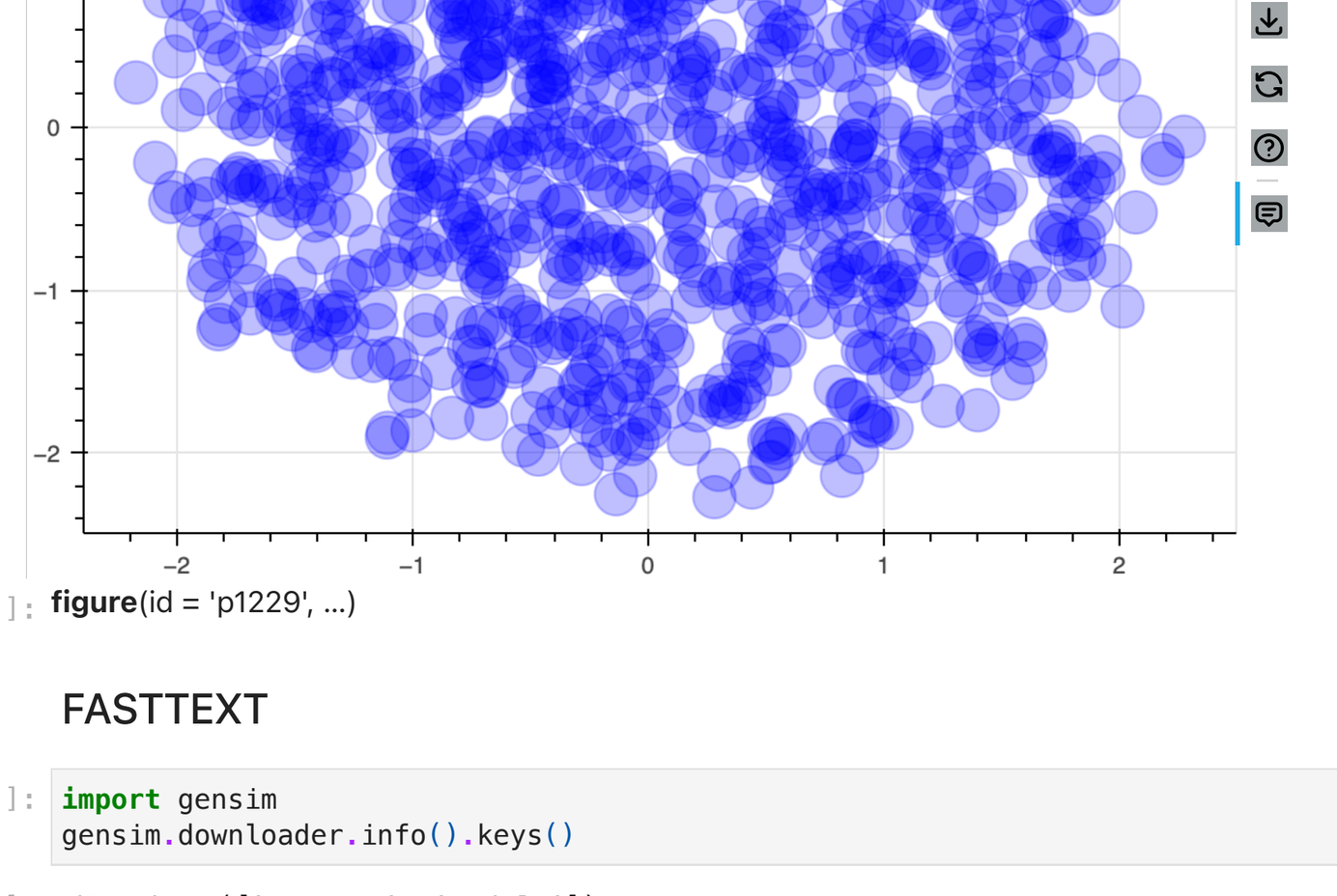
TSNE on all data

```
# TSNE
phrase_vectors = np.array([get_phrase_embedding(x) for x in data])
```

```
phrase_vectors_2d = TSNE().fit_transform(phrase_vectors)

phrase_vectors_2d = (phrase_vectors_2d - phrase_vectors_2d.mean(axis=0)) / phrase_vectors_2d.std(axis=0)
```

```
draw_vectors(phrase_vectors_2d[:, 0], phrase_vectors_2d[:, 1],
             phrase=[phrase[:50] for phrase in chosen_phrases],
             radius=20,)
```

```
BokehUserWarning: ColumnDataSource's columns must be of the same length. Current lengths: ('color', 537272), ('phrase', 1001), ('x', 537272), ('y', 537272)
```



figure(id = 'p1229', ...)

FASTTEXT

```
import gensim
gensim.downloader.info().keys()
```

```
dict_keys(['corpora', 'models'])
```

```
gensim.downloader.info()['models'].keys()
```

```
dict_keys(['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300', 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50', 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300', 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200', '__testing_word2vec-matrix-synopsis'])
```

```
model = api.load('fasttext-wiki-news-subwords-300')
```

```
get_phrase_embedding('Hello, how are you').shape
```

```
(300,)
```

```
data_vectors = np.array([get_phrase_embedding(l) for l in data])
```

```
results = find_nearest(query="How do i enter the matrix?", k=10)

print(''.join(results))
```

```
How do I evaluate the integral?
How do I estimate the competition?
How do I simplify the expression?
How do I manage the business?
How do I choose the proper profession?
How do I access the ExtraTorrent website?
How do I prepare the resume?
How do I increase the vocabulary?
How do I crack the GMAT?
How do I crack the CLAT?
```

```
data_vectors.shape
```

```
(537272, 300)
```

Local-Sensitivity Hashing (approximate neighbours)

```
import numpy
import tqdm

from nearpy import Engine
from nearpy.hashes import RandomBinaryProjections

# Dimension of our vector space
dimension = 300

# Create a random binary hash with 10 bits
rbp = RandomBinaryProjections('rbp', 10)

# Create engine with pipeline configuration
engine = Engine(dimension, lshashes=[rbp])

# Index 1000000 random vectors (set their data to a unique string)
for index, vec in tqdm.tqdm(enumerate(data_vectors)):
    engine.store_vector(vec, 'data_%d' % index)
```

```
537272it [00:05, 89959.33it/s]
```

```
def find_nearest_nearpy(query):
    """
    given text line (query), return k most similar lines from data, sorted from most to least similar
    similarity should be measured as cosine between query and line embedding vectors
    hint: it's okay to use global variables: data and data_vectors. see also: np.argpartition, np.argsort
    """
    # YOUR CODE
    query_emb = get_phrase_embedding(query)
    top_k_indexes = np.array([int(x[1].split('_')[1]) for x in engine.neighbours(query_emb)])
    top_k_neighbours = np.array(data)[top_k_indexes]
    return top_k_neighbours
```

```
find_nearest(query="How does Trump?", k=10)
```

```
array(['How does GCM generate registration_ids?\n',
       'How does Donald Trump persuade?\n', 'How does informatica?\n',
       'How does politics originate?\n', 'How does Websockets operate?\n',
       'How does PPF works?\n', 'How does ICEfaces works?\n',
       'How does anything exist?\n', 'How does TheTake work?\n',
       'How does SHAREit work?\n'], dtype='<U1170')
```

```
find_nearest_nearpy(query="How does Trump?")
```

```
array(['How does Donald Trump persuade?\n',
       'How does Donald Trump treat waitstaff?\n',
       'Why does everybody hate Trump?\n',
       'What does nationalism means?\n', 'What does Tiffany Trump do?\n',
       'What does nationalism mean?\n', 'What does manipulation mean?\n',
       'What does SENSEX mean?\n', 'What does さあひる mean?\n',
       'What does εἶμεν mean?\n'], dtype='<U1170')
```

```
find_nearest(query="Why don't i ask a question myself?", k=10)
```

```
array(["Why can't I ask a question anonymously?\n",
       "Why don't I get a girlfriend?\n",
       "Why don't I have a girlfriend?\n",
       "Why don't I have a boyfriend?\n",
       "Why don't I have a female friend?\n", "Why don't I get a date?\n",
       "Why don't I love myself?\n", "Why don't I ever get a girl?\n",
       "Why can't I take a joke?\n", "Why can't I love a person?\n"],
      dtype='<U1170')
```

```
find_nearest_nearpy(query="Why don't i ask a question myself?")
```

```
array(["Why can't I ask a question anonymously?\n",
       "Why don't I have a female friend?\n", "Why don't I get a date?\n",
       "Why don't I love myself?\n", "Why can't I take a joke?\n",
       "Why can't I love a person?\n", "Why don't I want friends?\n",
       "Why can't I understand myself?\n", "Why can't I feel myself?\n",
       "Why can't I keep a conversation going?\n"], dtype='<U1170')
```