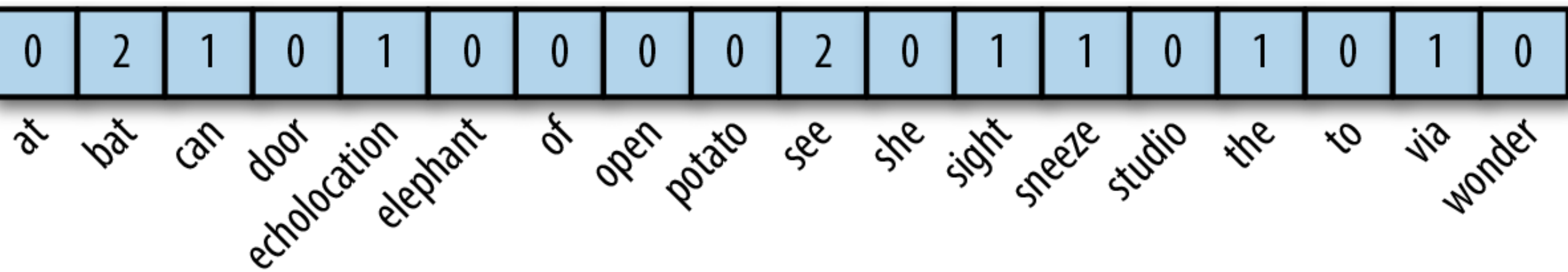
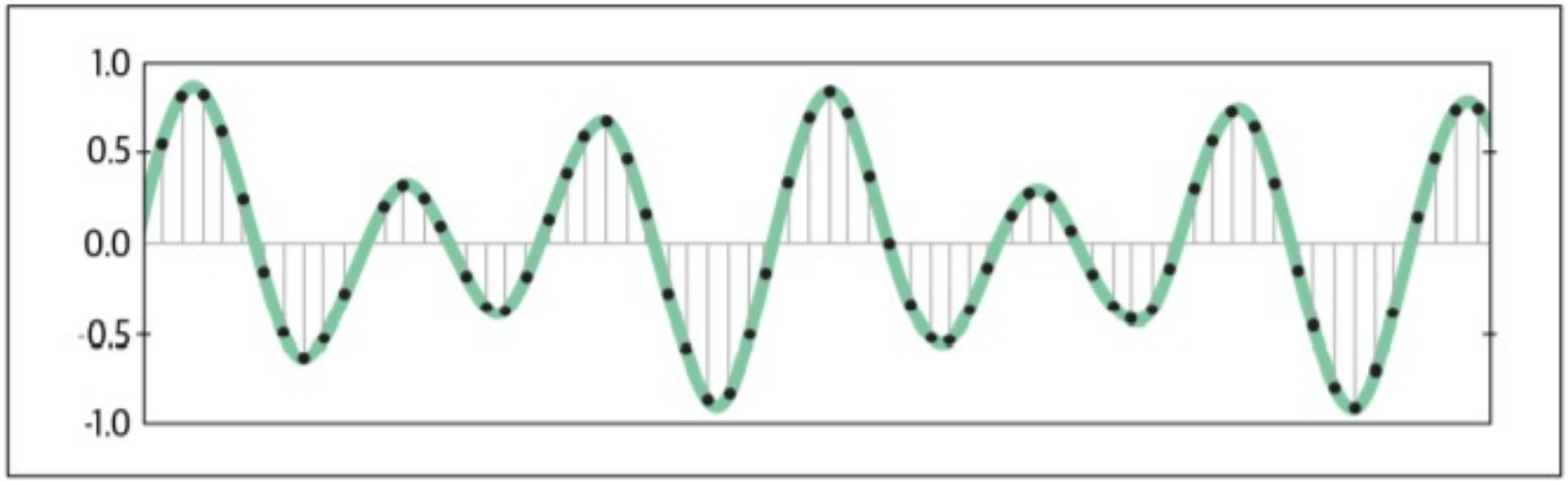
Topic 3
Text Representation



Speech




```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41,
-169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448,
-397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451,
1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499,
-488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148,
-1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325,
350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```

Figure 3-4. Speech signal represented by a numerical vector

Imaging

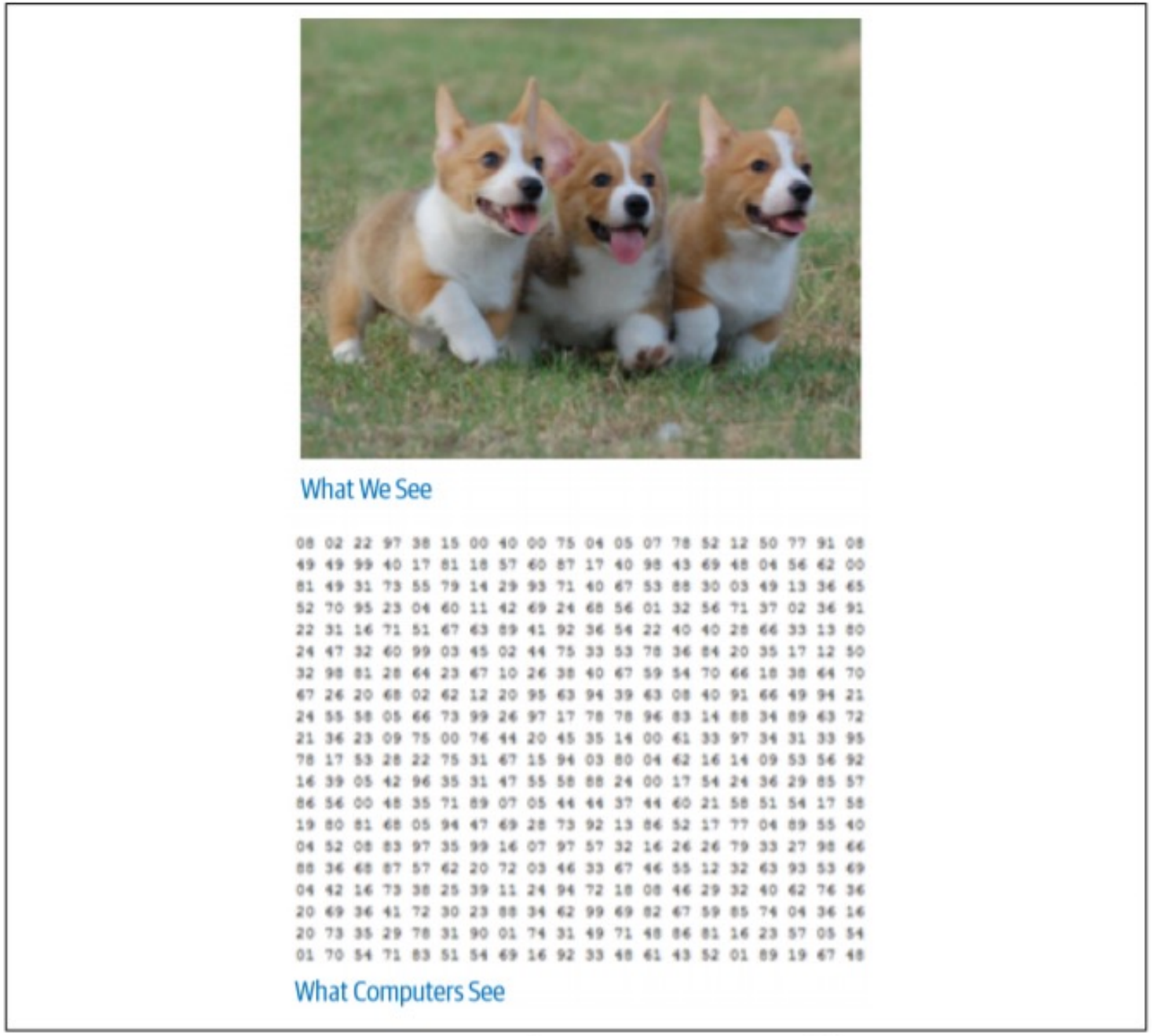


Figure 3-2. How we see an image versus how computers see it [1]

What can we do with a string?

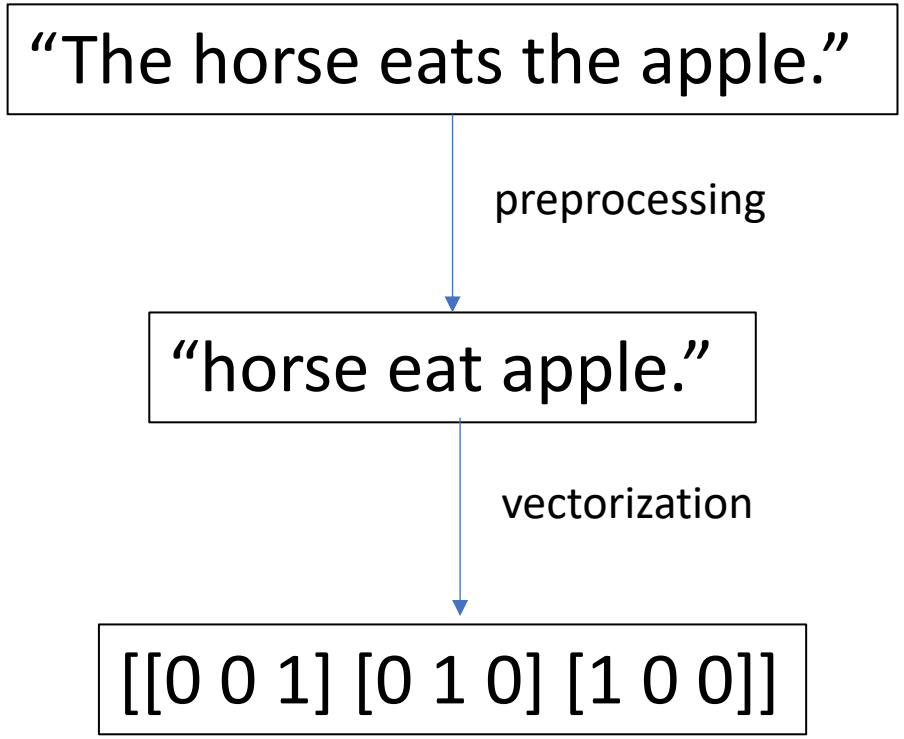
- Many methods cannot process strings or plain text
- Need to work on a numeric feature space
- Feature extraction, also known as *vectorization*
  - Character-level representation
  - Word-level representation
  - Document-level representation

Vector Representation

- Mathematical model that represents text units as vectors
  - What is a vector?
- Different strategies that capture the linguistic properties of a text to different degrees (but the strategies we cover will fall under vector space models - VSM)

One Hot Encoding

apple	eat	horse
1	0	0
0	1	0
0	0	1



One Hot Encoding

```
#Get one hot representation for any string based on this vocabulary.
#If the word exists in the vocabulary, its representation is returned.
#If not, a list of zeroes is returned for that word.
def get_onehot_vector(somestring):
    onehot_encoded = []
    for word in somestring.split():
        temp = [0]*len(vocab)
        if word in vocab:
            temp[vocab[word]-1] = 1 # -1 is to take care of the fact indexing
        onehot_encoded.append(temp)
    return onehot_encoded
```

if word not present, it's 0

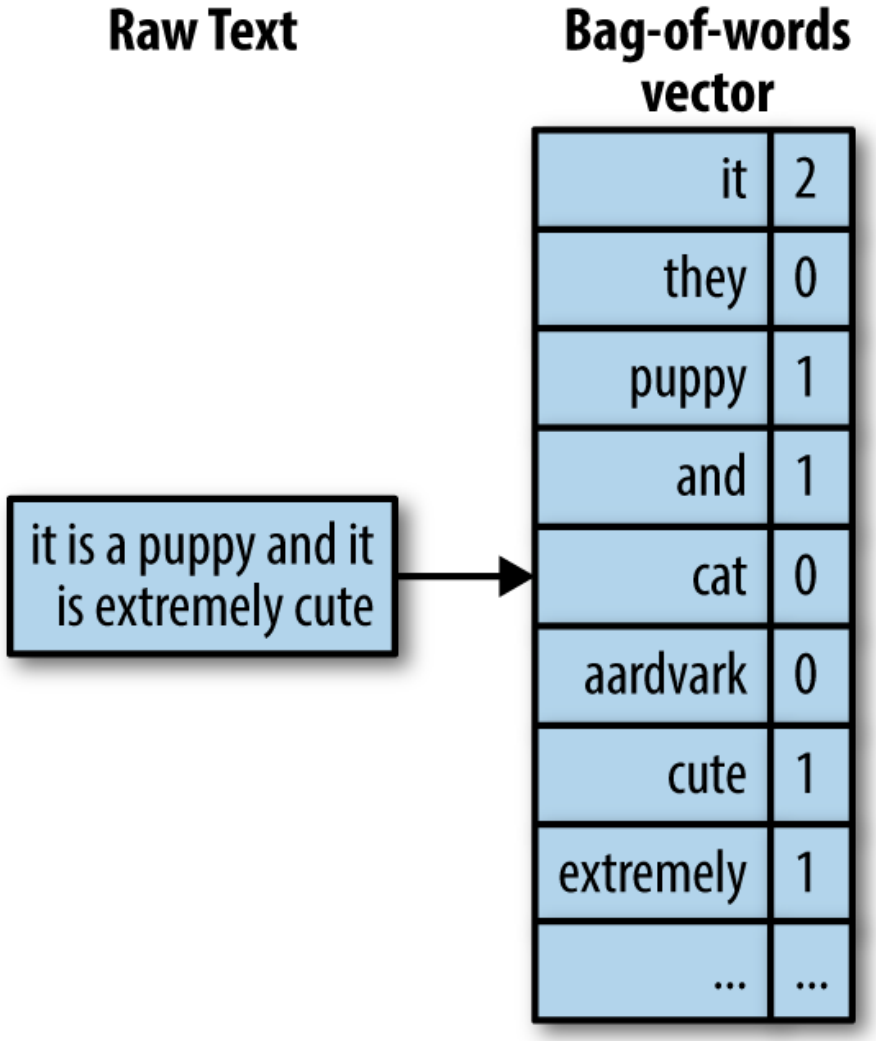
drawback: if sth not in vocab, it can't be represented

What happens if we add new documents?

Drawbacks

- All words equidistant, i.e. normalization extra important
- Relationships between tokens lost
- Scalability
- Sparsity
- Out of vocabulary words (OOV)

Bag-of-Words

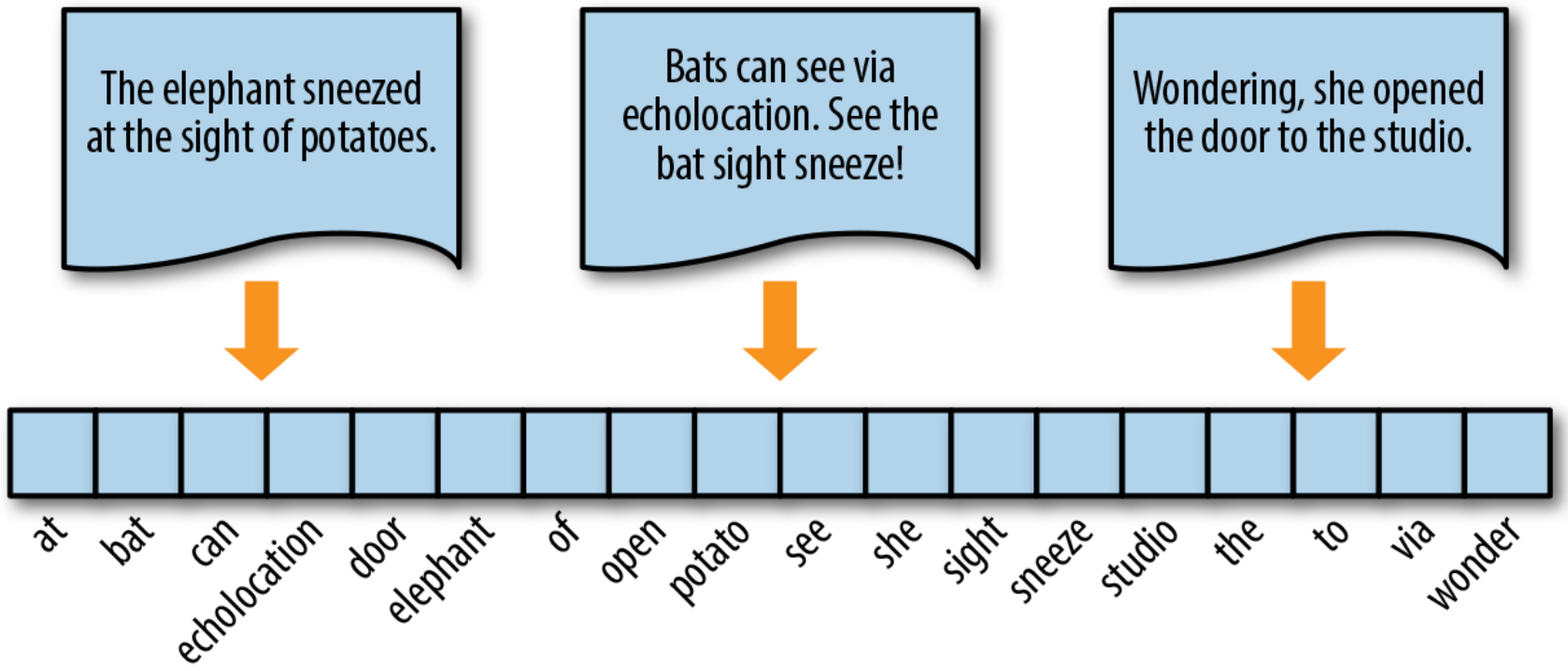


- Represent text as the bag (multiset) of its words
  - Keep multiplicity
  - Commonly used for classification problems

Text Book ch 3 GitHub have examples of how to build vocabulary

Bag-of-Words

- What numeric values should be put into this vector?



Drawbacks

"dog bites man" and "man bites dog" means the same thing

- The size of the vector still increases with the size of the vocabulary
  - Sparsity
- OOV words
- Disregarding grammar, word order
- Still can't capture information about words that mean the same thing

Bag of N-grams representation

deal with frequency of n words instead of 1 words

"The cat sat on the mat"

We could break this up into:

- **1-grams**: The, cat, sat, on, the, mat
- **2-grams**: The cat, cat sat, sat on, on the, the mat
- **3-grams**: The cat sat, cat sat on, sat on the, on the mat
- etc.

What will happen as N increases for an N-gram representation?

What will happen as N increases for an N-gram representation?

- We still have sparsity and OOV problems

Term frequency (TF) Representation

- TF representation weights words proportionally to their frequency
- Most frequent words not always most informative

- $TF(t, d) = \frac{\text{(\# of occurrences of term t in d)}}{\text{(total \# of terms in d)}}$

t = term, d = document

TF-IDF

- Inverse-Document-Frequency
- Basic idea: Common words may not add anything to our understanding of a specific text. Conversely, if a word occurs less frequently we might want to give it a larger weight in our representation

$$IDF(t) = \log\left(\frac{\text{total documents}}{\text{\# documents with t}}\right)$$

TF-IDF = TF * IDF

Activity

- Calculate the TF-IDF of term for the following toy corpus:
- **Document 1:** "The cat sat on the mat.”
- **Document 2:** “The mat was red.”
- **Document 3:** “The cat liked the mat."

$\frac{1}{6}$

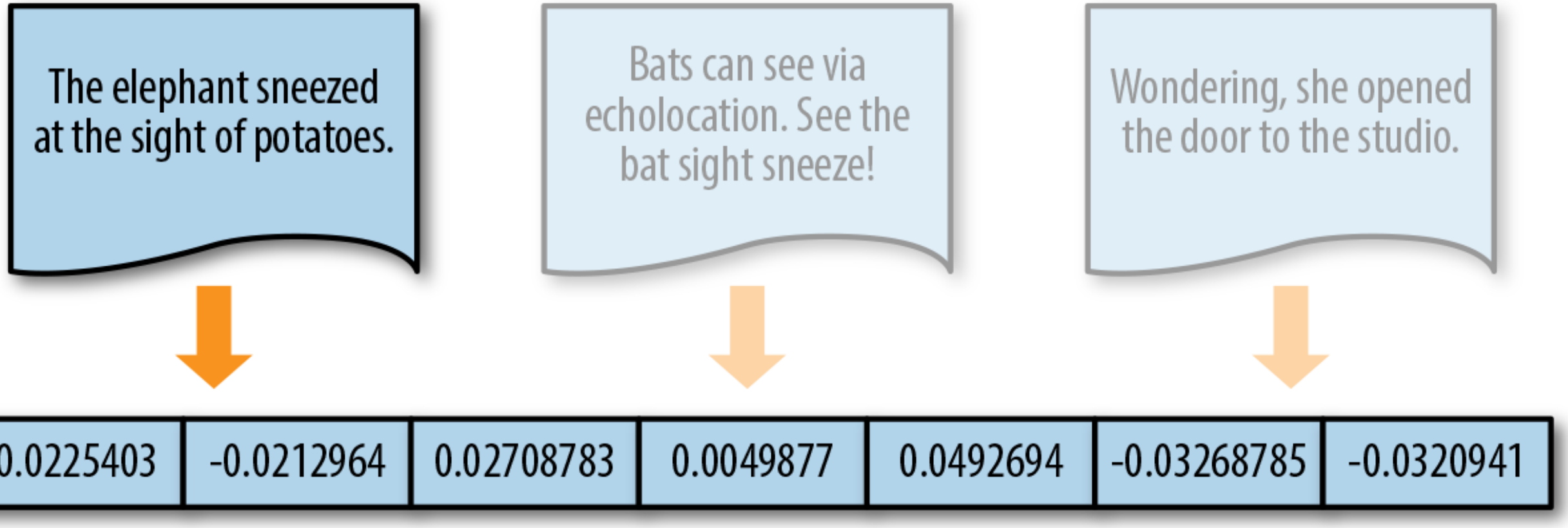
$$cat = TF \cdot IDF = \frac{1}{6} \cdot \log\left(\frac{3}{2}\right)$$

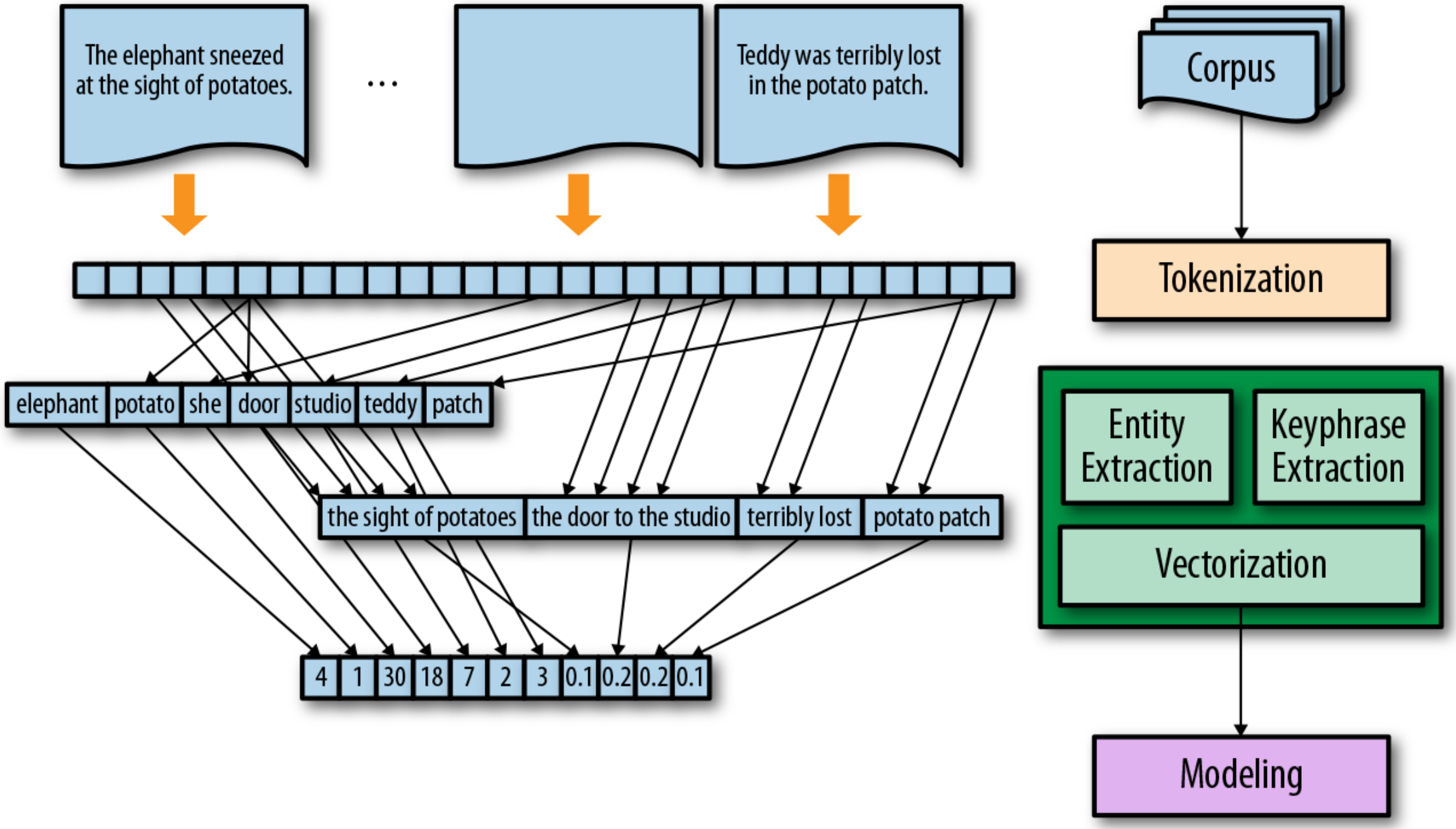
$$mat = \frac{1}{6} \cdot \log\left(\frac{3}{3}\right) = 0$$

scikit-learn change it also so TF-IDF of mat not to be 0

Distributed Representations

- Encode text along a continuous scale with a distributed representation
- Example: Word2Vec




The elephant sneezed at the sight of potatoes.
...
Teddy was terribly lost in the potato patch.
elephant
potato
she
door
studio
teddy
patch
the sight of potatoes
the door to the studio
terribly lost
potato patch
4
1
30
18
7
2
3
0.1
0.2
0.2
0.1
Corpus
Tokenization
Entity Extraction
Keyphrase Extraction
Vectorization
Modeling

Representing text as tensors

- What is a tensor?

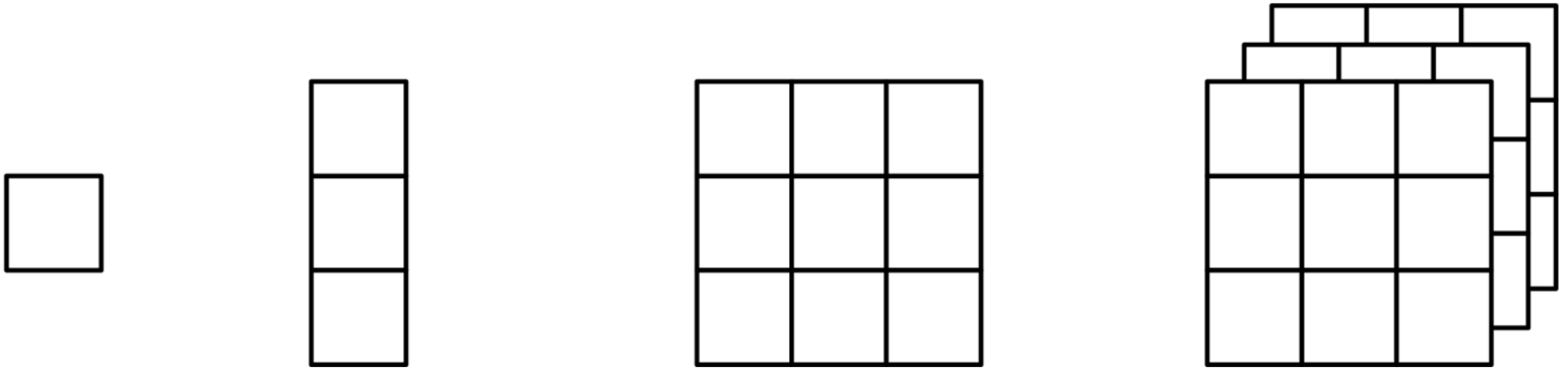
NumPy Array: [1 2 3]
PyTorch Tensor: tensor([1, 2, 3])

Scalar
Rank 0 Tensor

Scalar
Rank 1 Tensor

Scalar
Rank 2 Tensor

Rank 3 Tensor



Why use tensors?

- More efficient
- That is what many architectures expect

CUDA Tensors

- Using GPU instead of CPU
- Allocate the tensor on the GPU’s memory

```
torch.cuda.is_available()
torch.device()

x = torch.rand(3, 3).to(device)
describe(x)
```

Next time:

- Crash course in neural networks and autoencoders

1, CountVectorizer : Create dictionary

-> count_vect = CountVectorizer()

count_vect.fit_transform(text)

2, Create dictionary for series of words, not only single word

count_vect = CountVectorizer(ngram_range=(1, 3))

3, TF-IDF

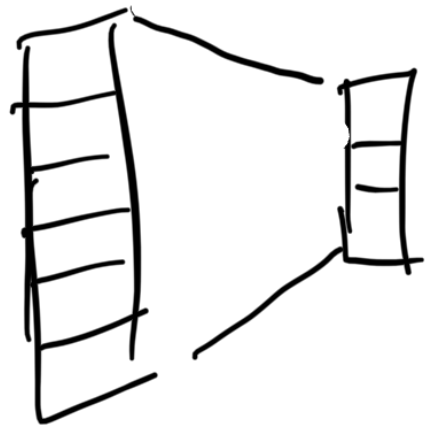
Create dictionary for words but with normalize

(*) If words doesn't exist, can't be represented (as not exist in dictionary)

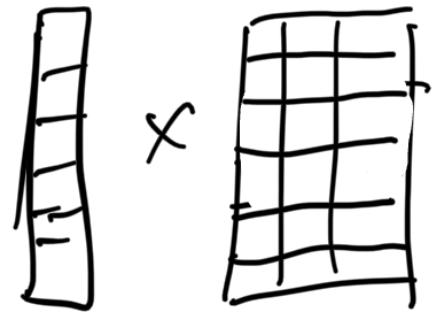
In Bag of word : Dog eats man and man eats dog is the same

In Bag of n_gram : They are different as it conta the representation of phrase in addition to representation of words

In TF-IDF, not exist = 1 al not exist = 0 but also include the interest ig



6x3



$= (6 \times 1) \times$ =