PYTHON PROGRAMMING AND MACHINE LEARNING

INTRODUCTION TO TEXT PROCESSING

Yunghans Irawan (yirawan@nus.edu.sg)

Objectives

- Understand the basic tasks of text processing in Python

- Able to implement a simple text processing using machine learning

Text Processing

- Structured vs. Unstructured Data
- Text Data Preparation
  - Tokenization
  - Stemming / Lemmatization
  - Stop words
- Text Featurization
  - Count Vectorization
  - TF-IDF
  - Word Embeddings

Why Text Processing?

Text data is string with varying lengths

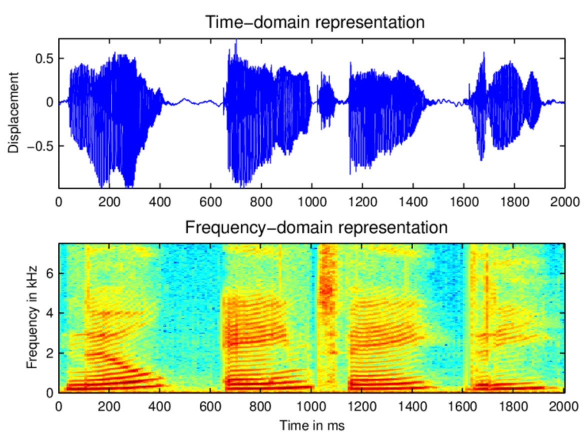
Machine Learning applies **mathematical models**

Therefore, need to **convert text to numbers** via

- Text Processing
- Text Featurization

Structured vs. Unstructured Data

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1



“zero, one, two, three”

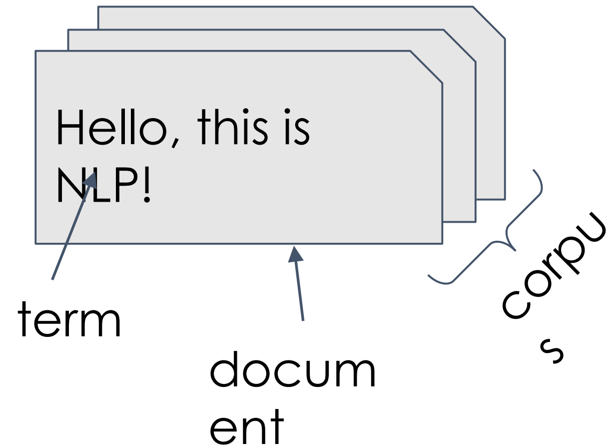
Structured vs. Unstructured Data

Structured	**Semi-structured**	**Unstructured**
Fixed format / size	Data with semantic tags	No format
Tabular data	XML, JSON	Text, Audio, Video, Speech

Objective: Convert unstructured data into structured vectors

Terminology

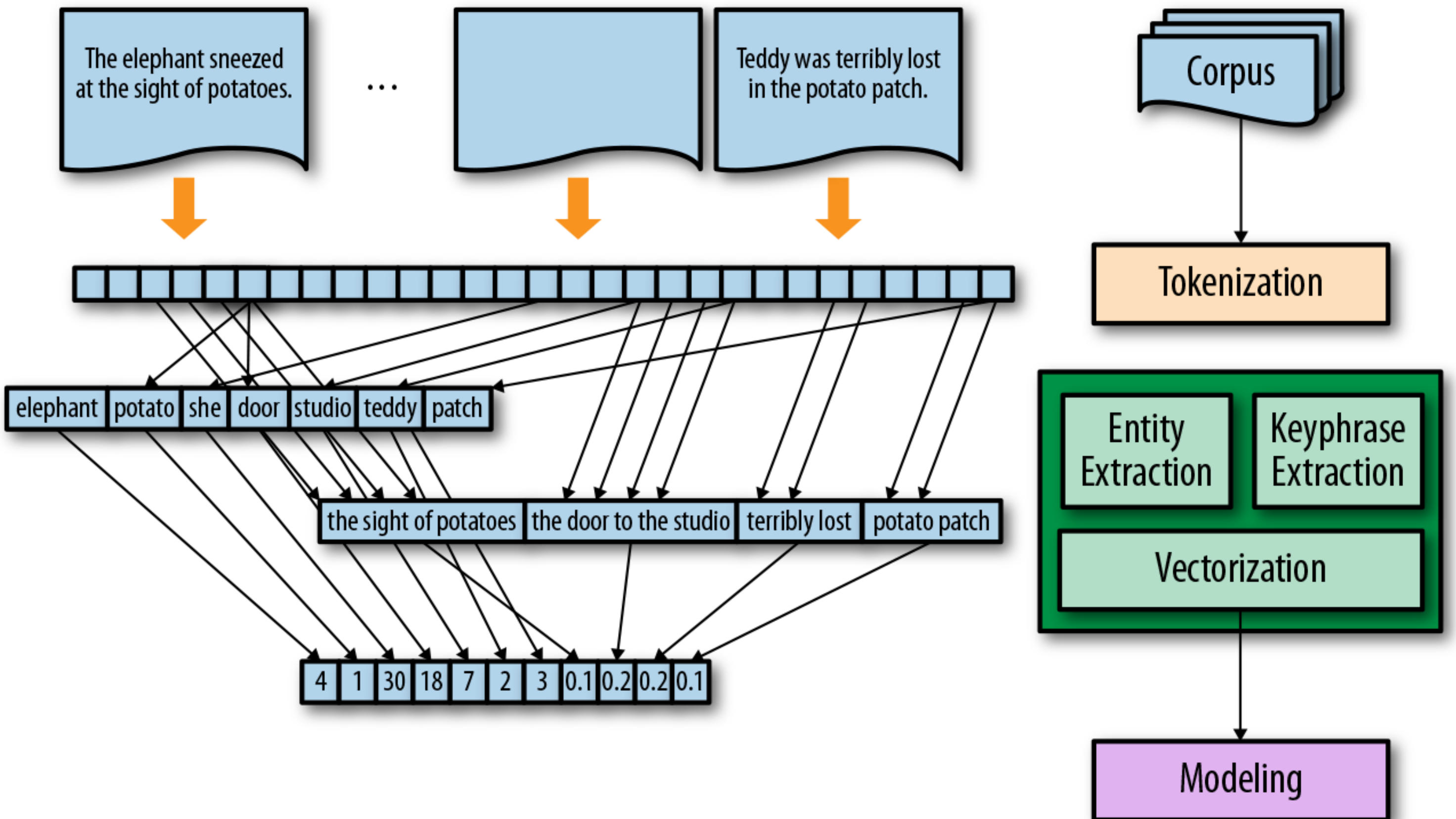
- Document
  - Contains a collection of words / bag of words
  - Article, email, SMS, word document, sentence, ...
- Corpus
  - Collection of documents
- Term / word / token
  - Text entity
- N-gram
  - **Terms** consisting of N consecutive, overlapping sequences of words
- Vocabulary
  - **Set** of unique terms
  - Feature dimensions = vocabulary size



Vocabulary (list of 3 bi-grams):

- hello this
- this is
- is nlp

Text Processing Example



https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html

Text Processing Libraries

NLP libraries

- Natural Language Toolkit (NLTK)
- SpaCy
- Gensim

More complete, linguistic features

ML libraries

- Scikit-Learn
- Keras
- PyTorch

Simple but more limited

TEXT DATA PREPARATION

Text Data Preparation

- Cleaning
- Tokenization
- Stemming / Lemmatization
- Stop words

Cleaning

- Remove punctuation
- Convert all to lowercase
- Remove non-ASCII characters
- …

Note: choose the appropriate cleaning for your task

Note: language specific rules

Tokenization

- Split document into terms
- Use libraries or write custom regex

```
from nltk import word_tokenize

text = 'Hello this is a test.'

word_tokenize(text)
```

```
['Hello', 'this', 'is', 'a', 'test', '.']
```

Lemmatization vs Stemming

```python
from nltk.stem import WordNetLemmatizer

text = 'he liked cats and dogs, and teaching machines to learn'

lm = WordNetLemmatizer()

print([lm.lemmatize(token) for token in word_tokenize(text)])
```

```
['he', 'liked', 'cat', 'and', 'dog', ',', 'and', 'teaching', 'machine', 'to', 'learn']
```

Looks at word form (verb, noun, ...)

```python
from nltk.stem import SnowballStemmer

text = 'he liked cats and dogs, and teaching machines to learn'

stem = SnowballStemmer(language='english')

print([stem.stem(token) for token in word_tokenize(text)])
```

```
['he', 'like', 'cat', 'and', 'dog', ',', 'and', 'teach', 'machin', 'to', 'learn']
```

Blunt knife chops off affixes for any word

Usually either is ok (depends on your task)

Stop words

Stop words are words that are very commonly in use in any sentence

Usually can be **removed without changing meaning**

English stop words

```
{'very', 'itself', 'does', 'nor', 'as', 'had', 'not', 'ours', "shan't", 'out', 'yourself', "hadn't", "hasn't", 'him', 'ma', 'o
ver', 'each', 'is', "that'll", 'she', 'to', "she's", 'but', 'should', 'shouldn', 'needn', 'when', 'those', "weren't", 'don',
'didn', 's', 'if', 'did', 'into', 'more', 'no', 'it', 'doing', "didn't", 'these', 'just', 'then', 'what', 'a', 'ain', 'now',
've', "mightn't", 'his', 'them', 'up', 'he', 'was', 'won', "won't", 'such', 'wasn', 'were', 'theirs', 'or', 'from', 'yours',
"needn't", 'few', 'once', 'd', 'can', 'during', 'they', 'own', 'will', "haven't", "isn't", 'there', 'some', 'y', 'at', 'on',
"don't", 'we', "you'd", 'against', 'both', 'aren', 'shan', 're', 'himself', 'be', 'have', 'being', 'hadn', 'any', "wouldn't",
'of', 'under', 'why', 'which', 'after', 'has', 'between', 'again', 'further', 'me', 'do', 'all', 'you', 'and', 'same', 'so',
'than', "you've", 'down', 'weren', 'an', 'most', 'couldn', 'o', 'are', "wasn't", 'who', 'because', 'her', 'before', 'wouldn',
'mightn', 'its', 'this', 'for', "you're", 'i', 'with', 'here', 'above', "should've", "couldn't", 'ourselves', 'where', 'm', 'o
ther', 'in', 'by', 'yourselves', 'themselves', 'hasn', "mustn't", "it's", 'off', "aren't", 'the', 'doesn', 'through', 't', 'yo
ur', "you'll", 'herself', 'whom', 'mustn', 'that', 'am', 'until', 'isn', 'having', 'how', 'about', 'll', 'haven', 'myself', 'h
ers', 'my', 'while', "doesn't", 'our', 'only', "shouldn't", 'their', 'below', 'too', 'been'}
```

Chinese stop words

Stop words

```python
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))

text = 'he liked cats and dogs, and teaching machines to learn'

print([token for token in word_tokenize(text) if token not in stop])
```

```
['liked', 'cats', 'dogs', ',', 'teaching', 'machines', 'learn']
```

Note: customize stop words depending on your task

Note: alternative is to apply a maximum threshold on word frequency

TEXT FEATURIZATION

Common Featurization Methods

- Word Count
- TF-IDF
- Word Embeddings

Objective: Convert a word to a **meaningful number** or a **vector of numbers**

Word Count Vectorization

1. Create vocabulary from the unique words
2. Count how often each word appears in a document
3. Create a feature vector with the word count as the entry

```
corpus = [
  'This is the first document.',
  'This document is the second document.',
  'And this is the third one.',
  'Is this the first document?',
]
```

```
vocabulary = [
'and', 'document', 'first', 'is',
'one', 'second', 'the', 'third', 'this'
]
```

Word Count Vectorization

1. Create vocabulary from the unique words
2. Count how often each word appears in a document
3. Create a feature vector with the word count as the entry

and	document	first	is	one	second	the	third	this	text
0	1	1	1	0	0	1	0	1	This is the first document.
0	2	0	1	0	1	1	0	1	This document is the second document.
1	0	0	1	1	0	1	1	1	And this is the third one.
0	1	1	1	0	0	1	0	1	Is this the first document?

Feature vector

Assume more frequent = more important

TF-IDF

Address shortcoming of Word Count Vectorization

- Penalize words that occur in lots of documents
- If a word appears all the time, it does not contain much information

Combines two measures

- Term Frequency = Word Count (as before)
- Inverse Document Frequency = Count of documents containing word

TF-IDF (textbook definition)

Word count (number of times term t appears in document d)

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

$$\text{idf}(t) = log\frac{n_d}{1+\text{df}(d,t)}$$

Document count (number of documents that contain term t)

TF-IDF vs. Word Count

Word Count

and	document	first	is	one	second	the	third	this	text
0	1	1	1	0	0	1	0	1	This is the first document.
0	2	0	1	0	1	1	0	1	This document is the second document.
1	0	0	1	1	0	1	1	1	And this is the third one.
0	1	1	1	0	0	1	0	1	Is this the first document?

TF-IDF

and	document	first	is	one	second	the	third	this	text
0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085	This is the first document.
0.000000	0.687624	0.000000	0.281089	0.000000	0.538648	0.281089	0.000000	0.281089	This document is the second document.
0.511849	0.000000	0.000000	0.267104	0.511849	0.000000	0.267104	0.511849	0.267104	And this is the third one.
0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085	Is this the first document?

More weight to rare words like “first, third”

Word Embeddings

Word count and TF-IDF are **statistical models**

Word Embeddings is a **neural probabilistic model**

Objectives:

1. Infer meaning of a word in terms of its neighbours
2. Compress sparse, high-dimensional data into lower dimensions

Sparsity and high-dimension

Each word is a column

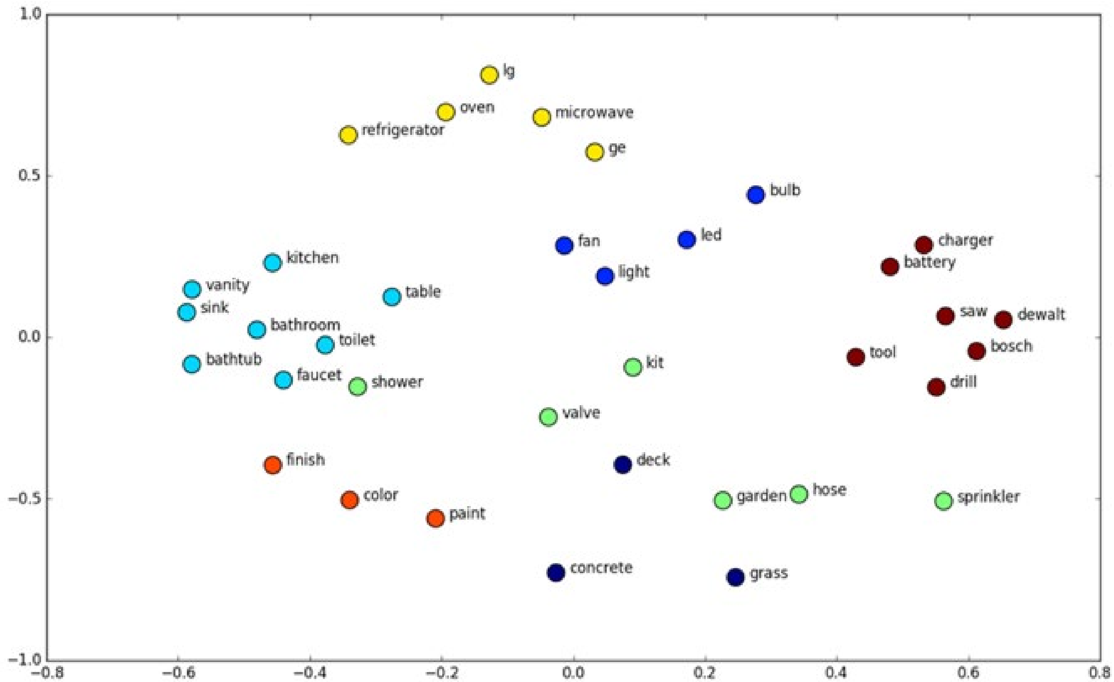
- E.g. 1000 word vocabulary => 1000 columns => 1000 dimensions

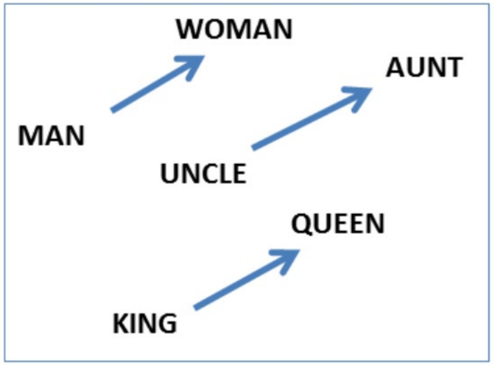
Columns are sparse: contain a lot of zeros

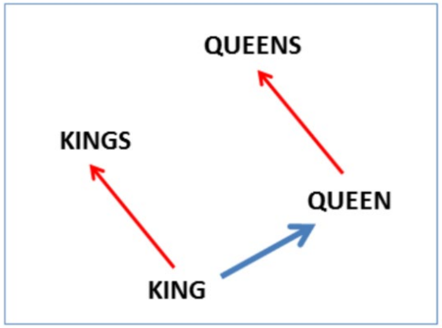
- Most words don't appear frequently

and	document	first	is	one	second	the	third	this	text
0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085	This is the first document.
0.000000	0.687624	0.000000	0.281089	0.000000	0.538648	0.281089	0.000000	0.281089	This document is the second document.
0.511849	0.000000	0.000000	0.267104	0.511849	0.000000	0.267104	0.511849	0.267104	And this is the third one.
0.000000	0.469791	0.580286	0.384085	0.000000	0.000000	0.384085	0.000000	0.384085	Is this the first document?

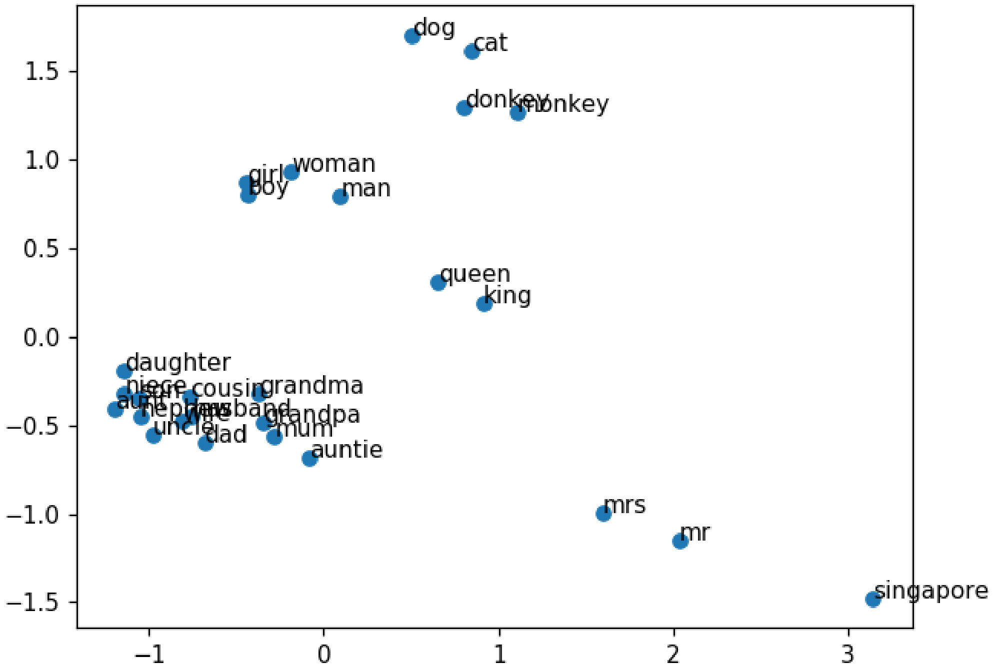
Word Meaning in Vector Space







Google News Embeddings




```python
model['man']
```

```
array([ 0.32617188,  0.13085938,  0.03466797, -0.08300781,  0.08984375,
       -0.04125977, -0.19824219,  0.00689697,  0.14355469,  0.0019455 ,
        0.02880859, -0.25      , -0.08398438, -0.15136719, -0.10205078,
        0.04077148, -0.09765625,  0.05932617,  0.02978516, -0.10058594,
       -0.13085938,  0.001297  ,  0.02612305, -0.27148438,  0.06396484,
       -0.19140625, -0.078125  ,  0.25976562,  0.375     , -0.04541016,
        0.16210938,  0.13671875, -0.06396484, -0.02062988, -0.09667969,
        0.25390625,  0.24804688, -0.12695312,  0.07177734,  0.3203125 ,
        0.03149414, -0.03857422,  0.21191406, -0.00811768,  0.22265625,
       -0.13476562, -0.07617188,  0.01049805, -0.05175781,  0.03808594,
       -0.13378906,  0.125     ,  0.0559082 , -0.18261719,  0.08154297,
       -0.08447266, -0.07763672, -0.04345703,  0.08105469, -0.01092529,
        0.17480469,  0.30664062, -0.04321289, -0.01416016,  0.09082031,
```

```python
model.distance('man', 'woman')
```

```
0.23359877690046482
```

```python
model.most_similar("man")
```

```
[('woman', 0.7664012312889099),
 ('boy', 0.6824870109558105),
 ('teenager', 0.6586930751800537),
 ('teenage_girl', 0.6147903203964233),
 ('girl', 0.5921714305877686),
 ('suspected_purse_snatcher', 0.571636438369751),
 ('robber', 0.5585119128227234),
 ('Robbery_suspect', 0.5584409236907959),
 ('teen_ager', 0.5549196004867554),
 ('men', 0.5489763021469116)]
```

Word Distance

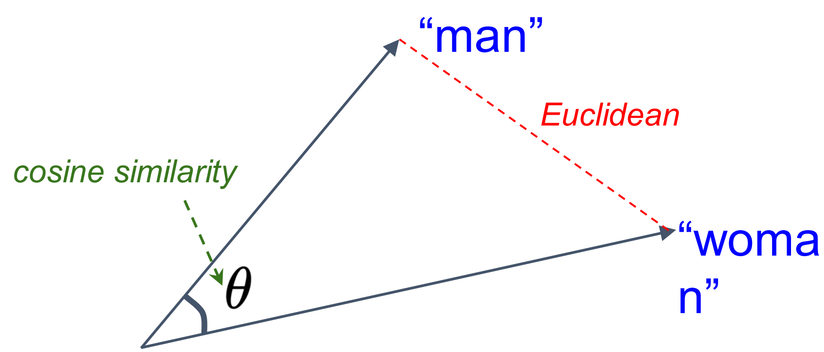
$$similarity(A,B) = cos(\theta) = \frac{AB^\top}{\|A\|\|B\|}$$

$$euclidean(A,B) = \|A - B\| = \sqrt{\|A\|^2 + \|B\|^2 - 2A.B}$$

$$\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2}$$

More common: cosine similarity

- Has direction
- Range [-1, 1]

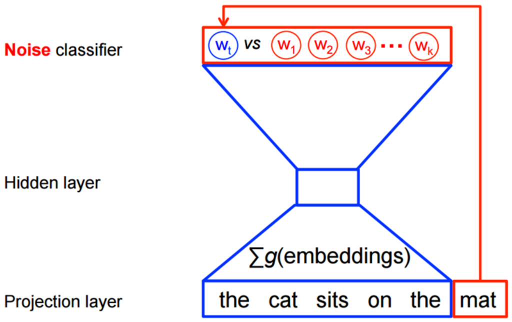


Pre-trained Word Embeddings

- GloVe - trained using global co-occurrence statistics

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

- Word2Vec - trained using negative sampling

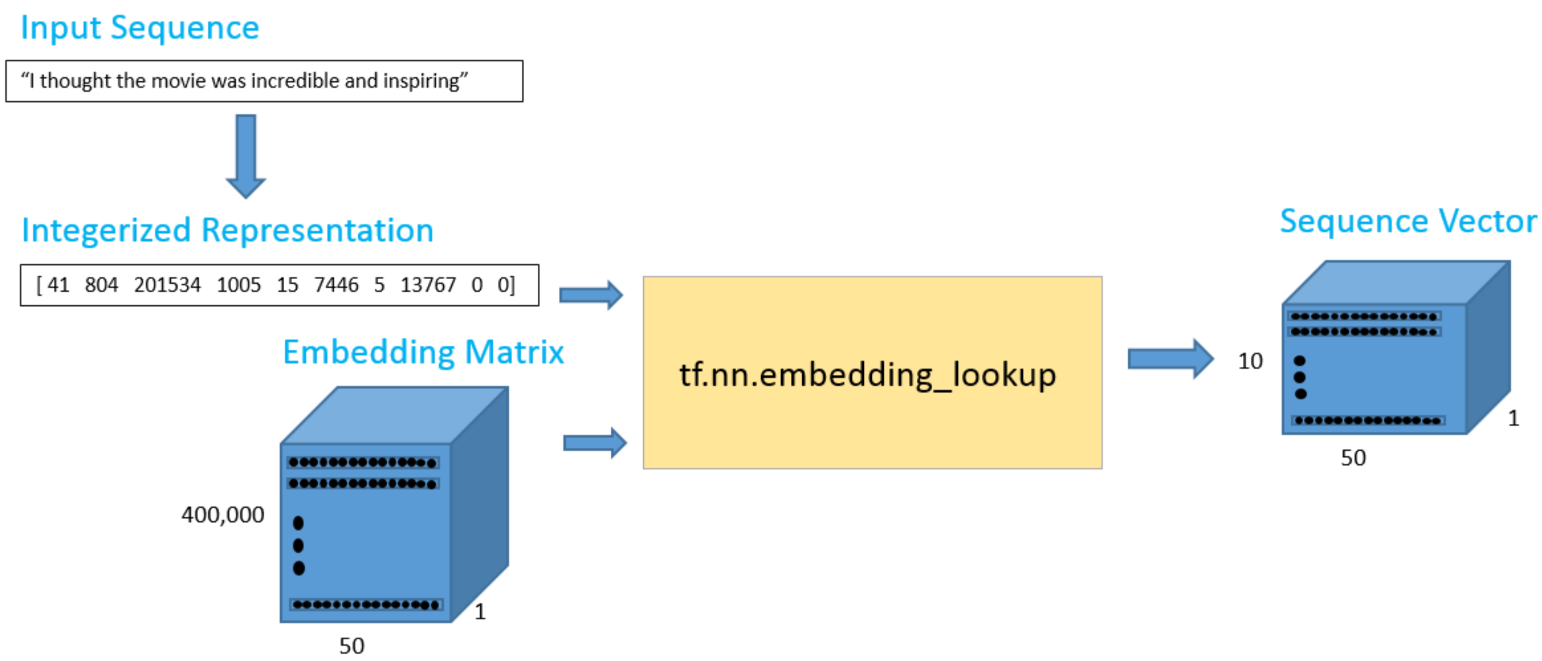


Pre-trained Word Embeddings

- fastText - each word is a bag of character-level n-grams. Available in 157 languages

- Custom corpus
  - Train your own word-embeddings using gensim.Word2Vec
  - Alternatively, you can try updating an existing word embedding

Embedding Layer



When to use...

Word Counts	Small corpus and vocabulary When you need a quick baseline
TF-IDF	Documents cover similar vocabulary e.g. movie reviews
Pre-trained Word Embedding	Documents cover large, diverse vocabulary e.g. wikipedia articles
Custom Word Embedding	Domain-specific vocabulary Want to encode word meanings

Where are we?

Text Corpus → Pre-processing (clean, tokenize, lemmatize, ...) → Feature extraction (word count, TF-IDF, word embeddings) → Model (classification, regression, clustering)

raw text

terms

```python
from nltk.corpus import stopwords
stop = set(stopwords.words('english'))

text = 'he liked cats and dogs, and teaching machines to learn'

print([token for token in word_tokenize(text) if token not in stop])
```

```
['liked', 'cats', 'dogs', ',', 'teaching', 'machines', 'learn']
```

vectors

```python
model['cat'] # vector
```

```
array([-0.96419 , -0.60978 ,  0.67449 ,  0.35113 ,  0.41317 , -0.21241 ,
        1.3796  ,  0.12854 ,  0.31567 ,  0.66325 ,  0.3391  , -0.18934 ,
       -3.325   , -1.1491  , -0.4129  ,  0.2195  ,  0.8706  , -0.50616 ,
       -0.12781 , -0.066965,  0.065761,  0.43927 ,  0.1758  , -0.56058 ,
        0.13529 ], dtype=float32)
```

Applications of Text Processing

Retail: Suggest product pricing based on item description, etc

Finance: Predict stock price movements from market news, etc

Social: Classifying Wikipedia comments for toxicity

Education: Predict acceptance of teacher project proposals

Business: Predict if a Kickstarter project gets funding

Further study

Latent Semantic Analysis - TF-IDF + SVD dimensionality reduction

King - man + woman is queen, but why?

Gensim Tutorials

Stanford CS224n

How to use Word Embedding Layers in Keras

Practitioner's Guide