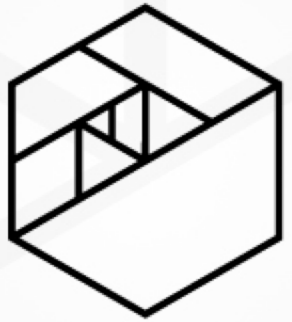
Natural Language Processing and Text Mining in Python

Michael Galvin
Executive Director of Data Science, Metis
galvin.mj@gmail.com
Twitter: @MikeJGalvin
https://github.com/galvin-mj

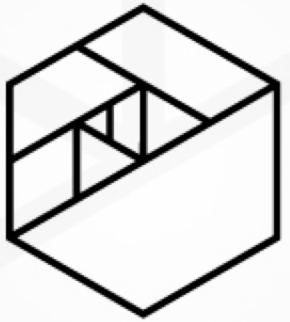


METIS

Metis
Data Science Training

- Data Science Bootcamp
  - 12 Weeks, in person
- Corporate Training
  - Machine Learning
  - Natural Language Processing
  - Spark
  - Python for Data Science
- Explore Data Science Online Training
- Evening Professional Development Courses

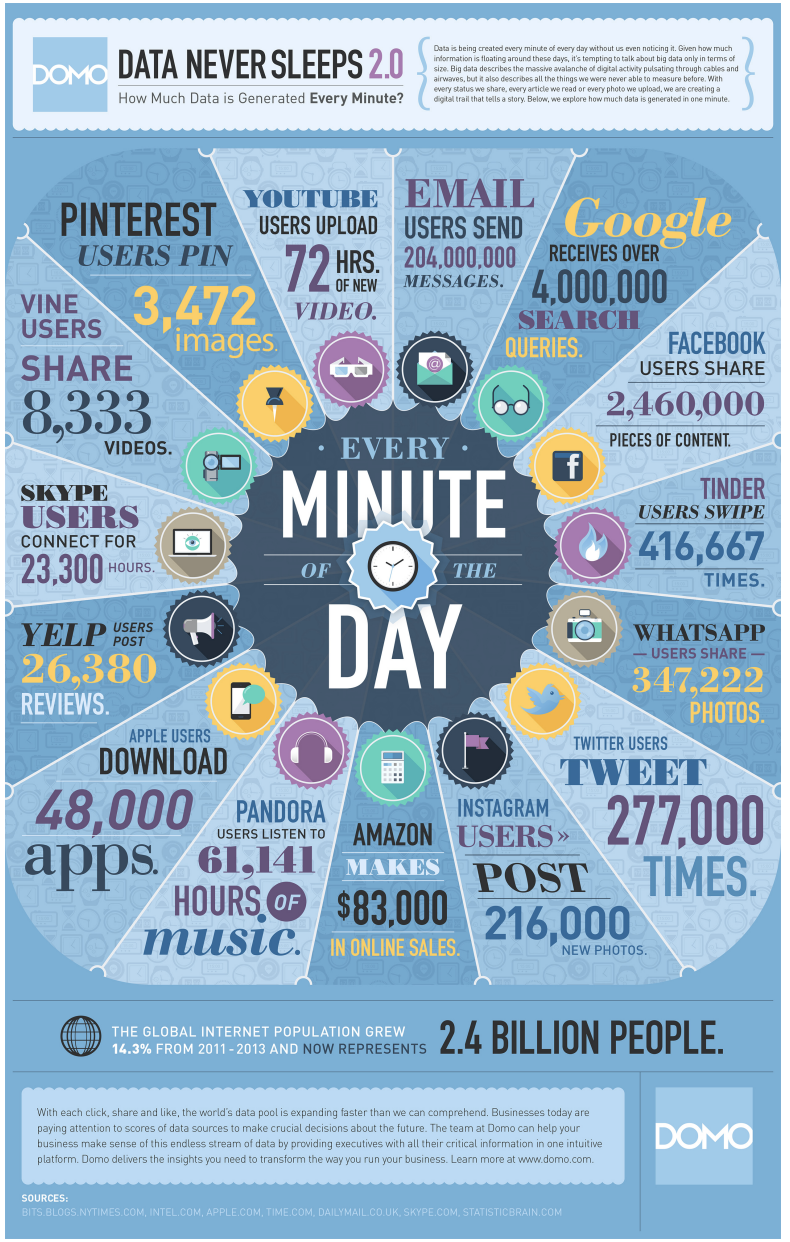
www.thisismetis.com



Large amounts of data are being generated every minute

- Lot's of this data is text
  - Email
  - Text messages
  - News articles
  - Blogs
  - Twitter
  - Reviews

- 204,000,000 Emails
- 4,000,000 Google Search Queries
- 277,000 Tweets
- 26,380 Yelp Reviews



https://www.domo.com/blog/2014/04/data-never-sleeps-2-0/

How is this useful?

- Classification
- Clustering (Organizing)
- Information Retrieval
- Topic Modeling
- Sentiment Analysis
- Parts of speech tagging
- Recommender systems
- Etc.

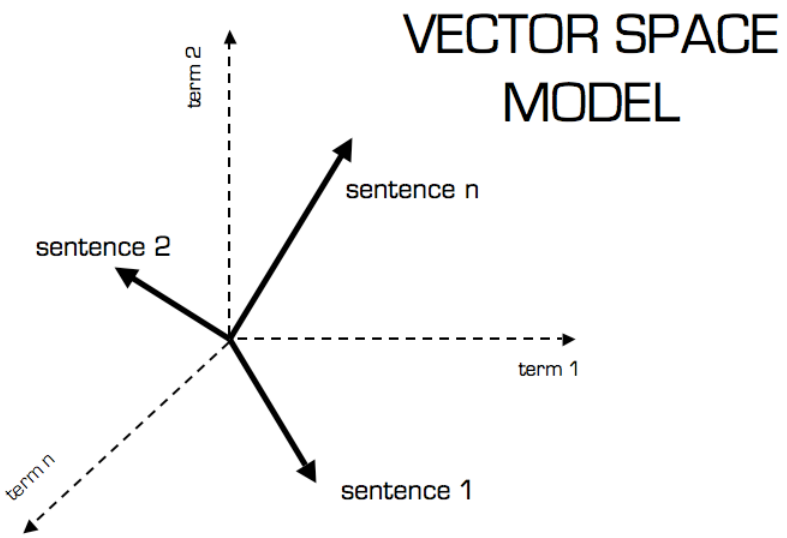
How to represent text?

Working with text

- For most analysis tasks we want structured numeric data
- Text
    - Unstructured
    - No usable numeric values

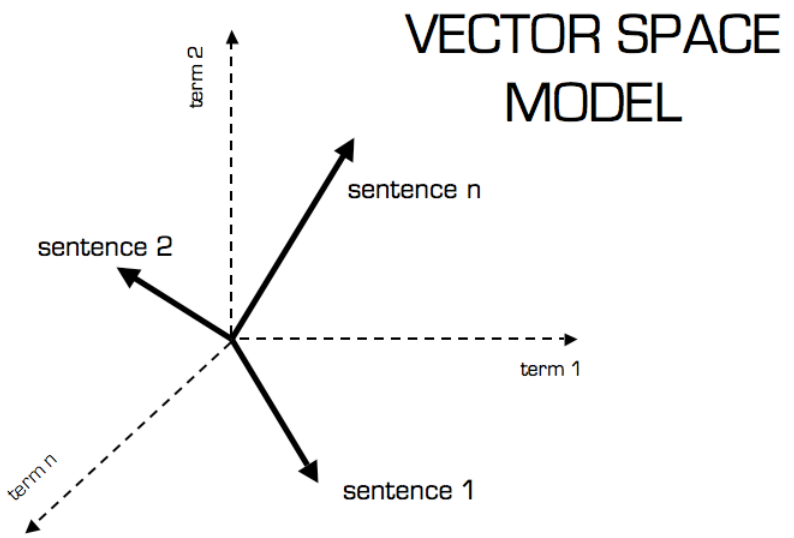
How to represent text?

- Tokenize text
  - “This is an example”
  - [This, is, an, example]
- One hot encoding
  - Vector indicating index of word
  - This = [1, 0, 0, 0]
  - is = [0, 1, 0, 0]
- Represent a document by it’s tokens



How to represent text?

- Tokenize text
  - “This is an example”
  - [This, is, an, example]
- One hot encoding
  - Vector indicating index of word
  - This = [1, 0, 0, 0]
  - is = [0, 1, 0, 0]
- Represent a document by it’s tokens




```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()

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

X = vectorizer.fit_transform(corpus)
pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
```

and	document	first	is	one	second	the	third	this
0	1	1	1	0	0	1	0	1
0	1	0	1	0	2	1	0	1
1	0	0	0	1	0	1	1	0
0	1	1	1	0	0	1	0	1

How to represent text?

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()

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

X = vectorizer.fit_transform(corpus)
pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
```

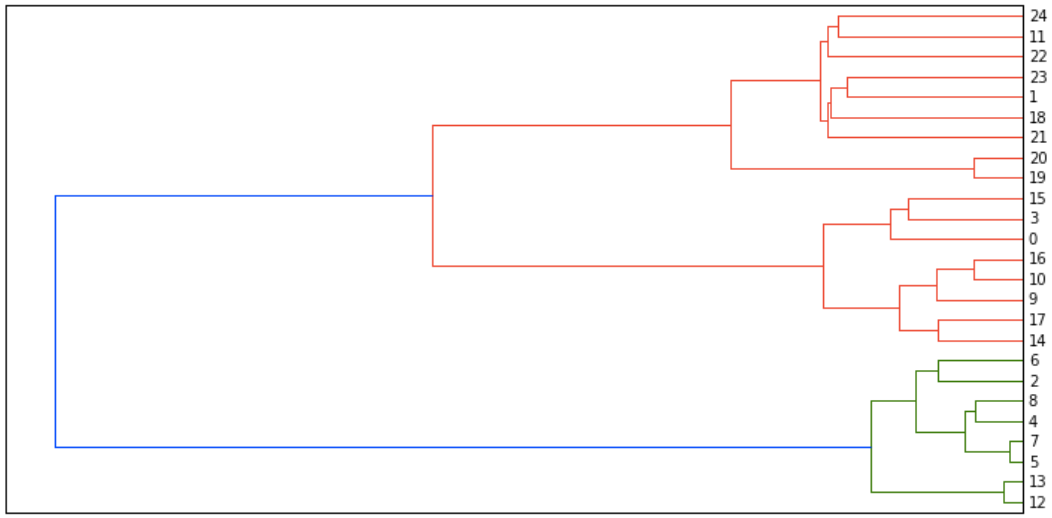
and	document	first	is	one	second	the	third	this
0	1	1	1	0	0	1	0	1
0	1	0	1	0	2	1	0	1
1	0	0	0	1	0	1	1	0
0	1	1	1	0	0	1	0	1

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(binary = True)

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

X = vectorizer.fit_transform(corpus)
pd.DataFrame(X.toarray(), columns = vectorizer.get_feature_names())
```

and	document	first	is	one	second	the	third	this
0	1	1	1	0	0	1	0	1
0	1	0	1	0	1	1	0	1
1	0	0	0	1	0	1	1	0
0	1	1	1	0	0	1	0	1

Clustering

```
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster.hierarchy import ward, dendrogram

dist = 1 - cosine_similarity(data[0:25])
linkage_matrix = ward(dist)
dendrogram(linkage_matrix, orientation="right");
```



Classification

```
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']

from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train',categories=categories, shuffle=True, random_state=42)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_counts, twenty_train.target)
```

http://scikit-learn.org/

What next?

- This seems simple.
    - How much information is contained in a word?
    - What else can we add to gain more information?
    - Are we making naive assumptions? What issues come with this?

Some Questions

- How to tokenize?
- Case
- Punctuation
- Common words
- Roots
- Word order and co-occurrence
- Misspellings
- Different Langauges
- Equal word weighting

Some Issues and Tricks

- Common Words
  - The, I, a
  - Stop Words

```
ex = CountVectorizer(stop_words='english')
x = ex.fit_transform(['this is an example with stop words'])
pd.DataFrame(x.toarray(), columns = ex.get_feature_names())
```

example	stop	words
1	1	1

- Word Roots
  - Run, running
  - Stemming

```
import nltk
stemmer = nltk.stem.porter.PorterStemmer()
print stemmer.stem('run')
run
print stemmer.stem('running')
run
```

- Word Order
  - The movie was not good
  - NY Times
  - N-grams

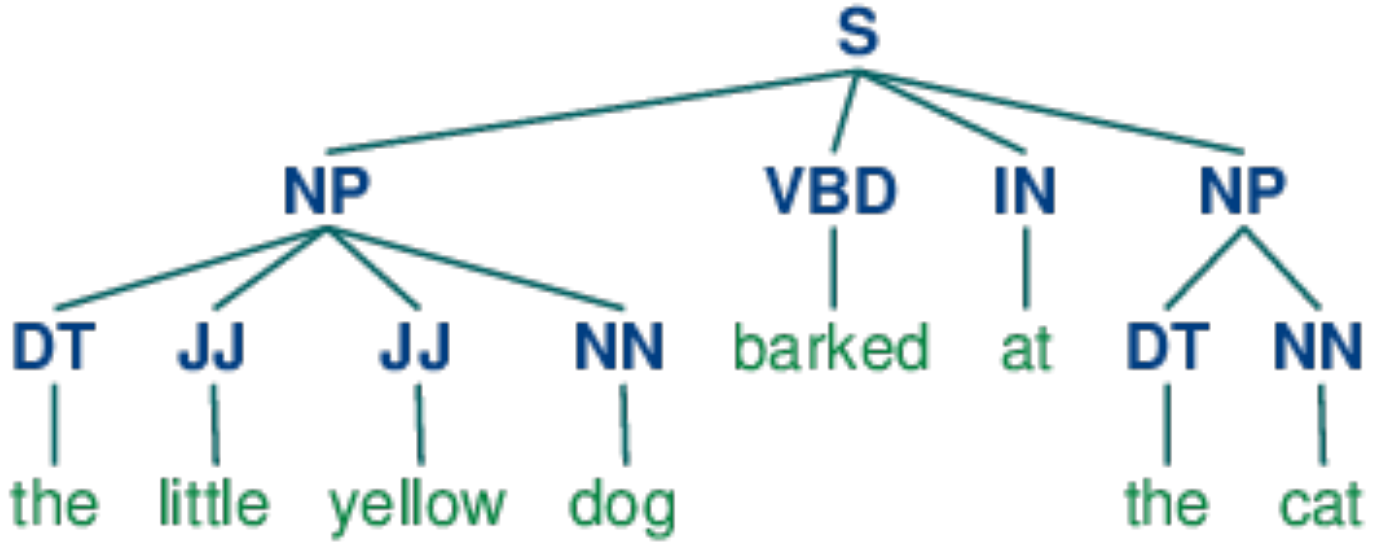
```
text = ['this is a bigram example']
bigram_vectorizer = CountVectorizer(ngram_range=(2,2))
X = bigram_vectorizer.fit_transform(text)
pd.DataFrame(X.toarray(), columns=bigram_vectorizer.get_feature_names())
```

bigram example	is bigram	this is
1	1	1

- Presupposition: Implicit Assumptions
  - Jane no longer writes fiction (Jane once wrote fiction)
- Word Relationships:
  - Substitution: Red and Blue, Monday and Tuesday
  - Co-occurrences: Car and Drive
- Sparsity
- High Dimensionality
- Similarity and Meaning

Chunking

- Chunking segments and labels multi-token sequences



Practice

Machine Learning

- Supervised: Prediction
    - Regression: Predict Continuous Value
    - Classification: Predict Categorical Value
- Unsupervised: Finding Structure
    - Clustering: Group Observations
    - Dimensionality Reduction: Reduce the Number of Dimension

Vocabulary

- Features
- Observations
- Labels

sepal length	sepal width	petal length	petal width	species
6.7	3.0	5.2	2.3	virginica
6.4	2.8	5.6	2.1	virginica
4.6	3.4	1.4	0.3	setosa
6.9	3.1	4.9	1.5	versicolor
4.4	2.9	1.4	0.2	setosa
4.8	3.0	1.4	0.1	setosa
5.9	3.0	5.1	1.8	virginica
5.4	3.9	1.3	0.4	setosa
4.9	3.0	1.4	0.2	setosa
5.4	3.4	1.7	0.2	setosa

Vocabulary

- Features
- Observations
- Labels

sepal length	sepal width	petal length	petal width	species
6.7	3.0	5.2	2.3	virginica
6.4	2.8	5.6	2.1	virginica
4.6	3.4	1.4	0.3	setosa
6.9	3.1	4.9	1.5	versicolor
4.4	2.9	1.4	0.2	setosa
4.8	3.0	1.4	0.1	setosa
5.9	3.0	5.1	1.8	virginica
5.4	3.9	1.3	0.4	setosa
4.9	3.0	1.4	0.2	setosa
5.4	3.4	1.7	0.2	setosa

Vocabulary

- Features
- Observations
- Labels

sepal length	sepal width	petal length	petal width	species
6.7	3.0	5.2	2.3	virginica
6.4	2.8	5.6	2.1	virginica
4.6	3.4	1.4	0.3	setosa
6.9	3.1	4.9	1.5	versicolor
4.4	2.9	1.4	0.2	setosa
4.8	3.0	1.4	0.1	setosa
5.9	3.0	5.1	1.8	virginica
5.4	3.9	1.3	0.4	setosa
4.9	3.0	1.4	0.2	setosa
5.4	3.4	1.7	0.2	setosa

Text Features

- Words
- n grams
- POS tagging
- Chunks
- Syntactic structure
- etc

Text Classification

- We want to predict a categorical variable
- Example: Predict if an email is spam or ham
- Given:
    - X: Email text
    - Y: Spam or not
- Goal: Build model that predicts spam or ham for a new email

Process

- We have historical email data with the labels
- First, let's process the email text data
    - Tokenize and get word counts
    - Remove stop words
    - Stem
    - etc
- Now we can take this dataset and put it into a classification algorithm
- Given a new email, process the text in the same way and put it through the model

Many Different Algorithms

- Logistic Regression
- Random Forest
- Naive Bayes
- Support Vector Machines
- Boosting
- Etc

Practice

TF-IDF

Term Frequency - Inverse Document Frequency

Bag of Words

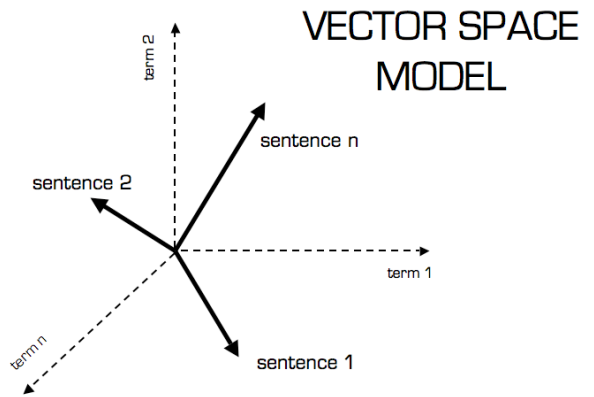
Process

- Tokenize
- Treat each token as a feature
- Represent a document by those features

"This is the first document document."

“This is the second document.”

	This	document	first	is	second	the
Sentence 1	1	2	1	1	0	1
Sentence 2	1	1	0	1	1	1

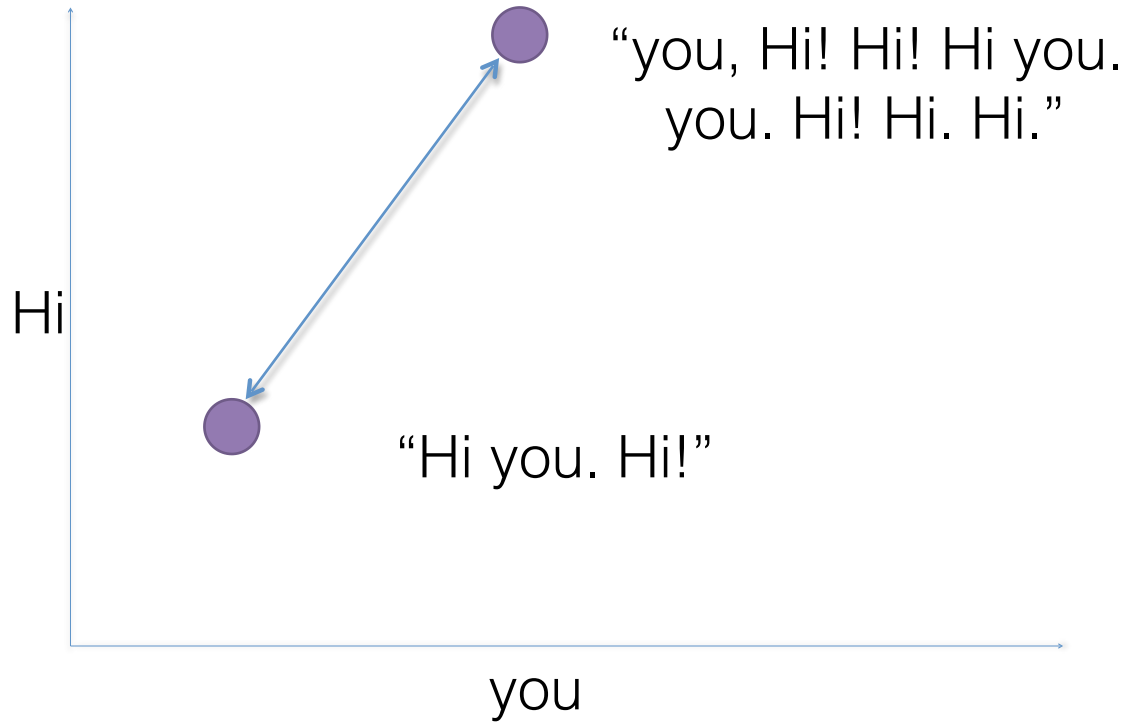


Some Issues

- High counts can dominate
    - Longer documents
    - High Frequency
- Each word is treated equally
    - Some terms might be more important than others

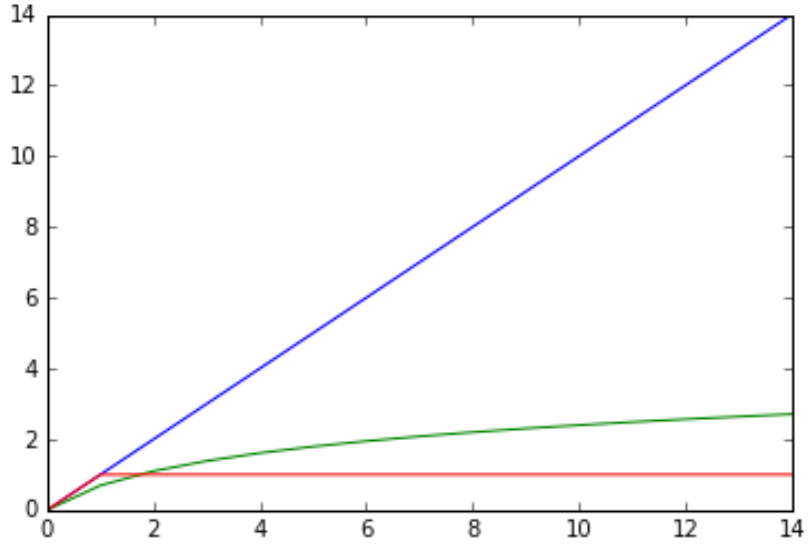
We want a metric that takes these issues into consideration

Term Frequency (TF)



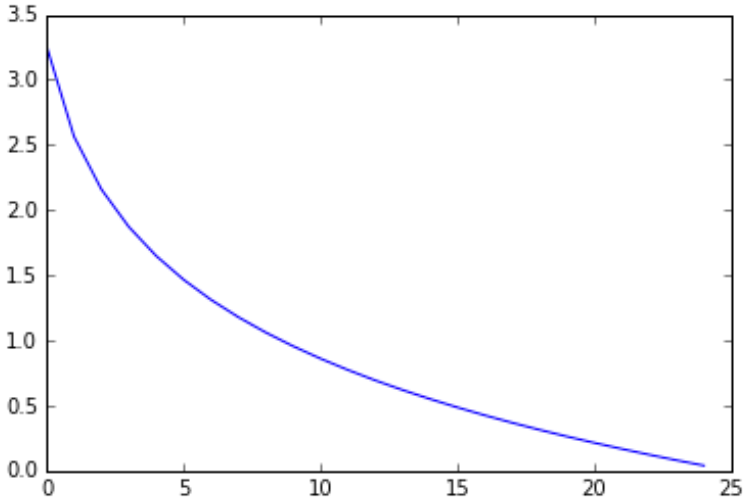
Term Frequency (TF)

- Two extremes
  - Did the term appear?
  - Frequency
- Balance with a transformation
  - One option is log(x+1) where x if the term count
  - There are others



Inverse Document Frequency (IDF)

- Previously we treated all words equally.
- We may want to discount common words and reward rare words
- IDF: Inverse Document Frequency
  - IDF = log[(D+1)/(d+1)]
- D: total number of documents
- d: Number of documents containing the term



Example

- IDF = log[(D+1)/(d+1)]
- D: total number of documents (D=2)
- d: total number of documents containing the term

"I like sports"
"I like technology"

- For "I" and "like" d = 2
  - log[(D+1)/(d+1)]= log[(2+1)/(2+1)] = log[3/3] = log[1] = 0
- For sports and technology. d = 1 f
  - log[(D+1)/(d+1)] = log[(2+1)/(1+1)] = log[3/2] = .4
- Sports and Technology are weighted more

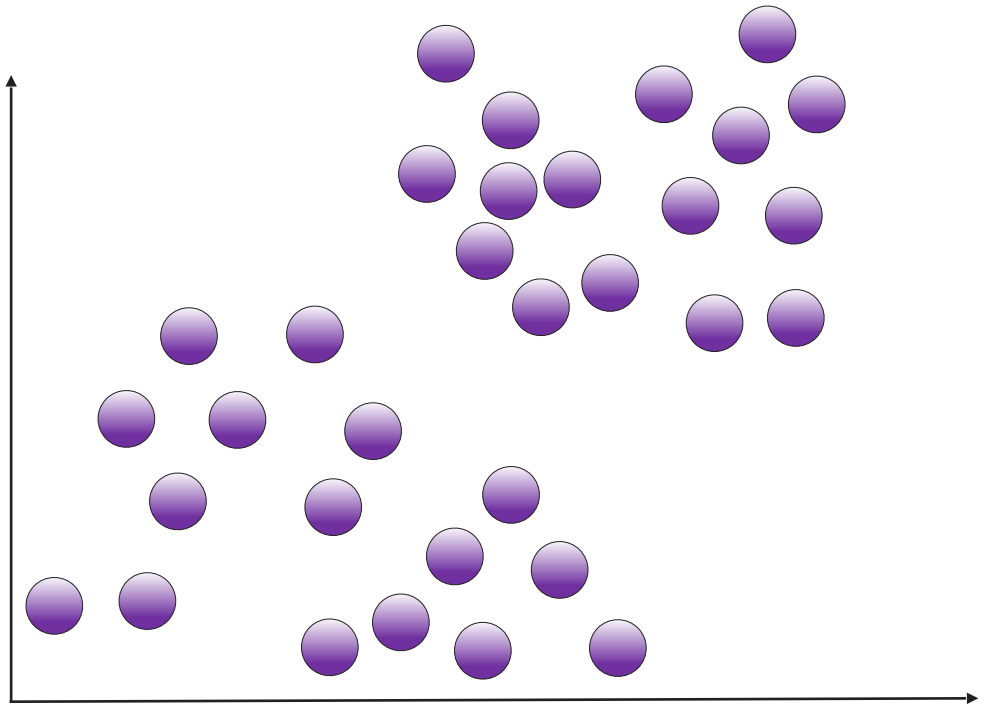
TFIDF

- We combine TF and IDF into a single metric
- TFIDF: Term Frequency Inverse Document Frequency
- TFIDF = TF*IDF

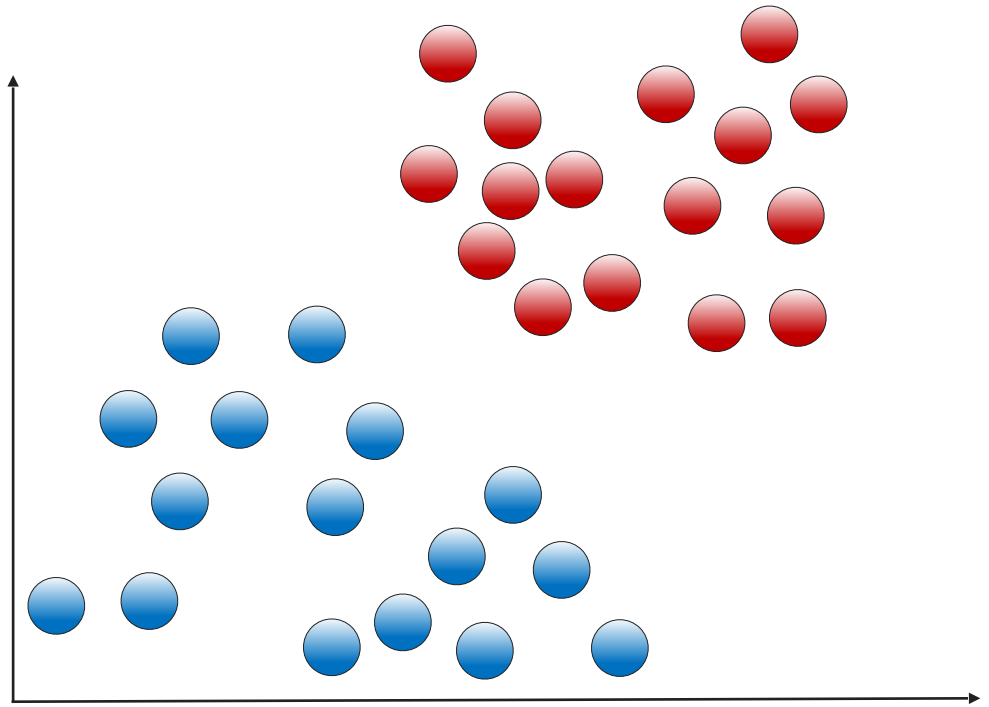
Practice

Clustering

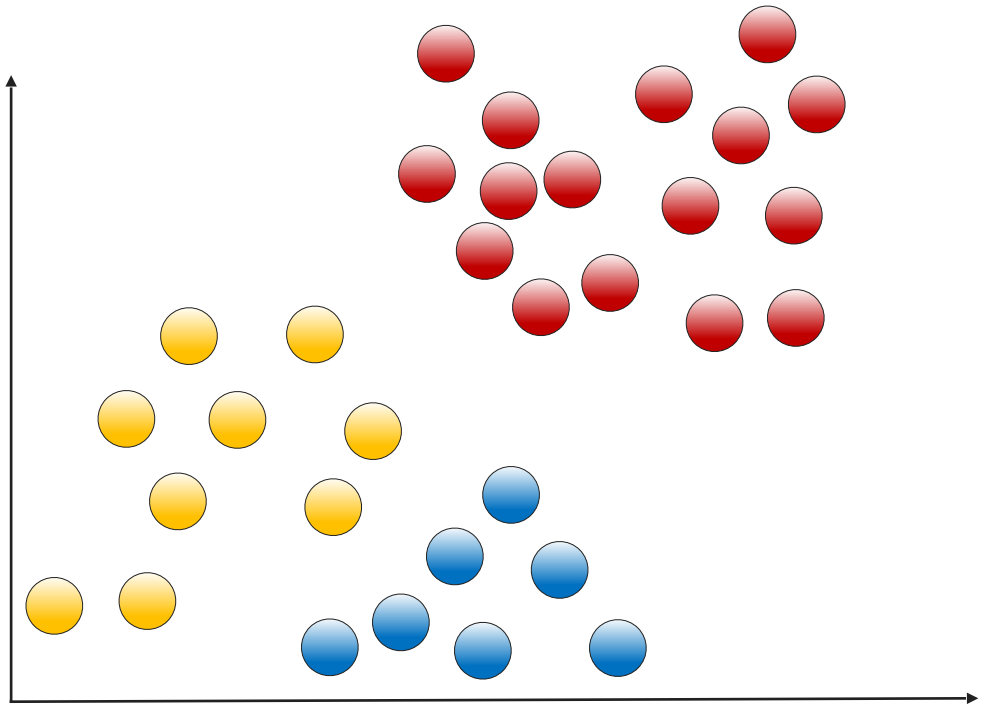
Clustering



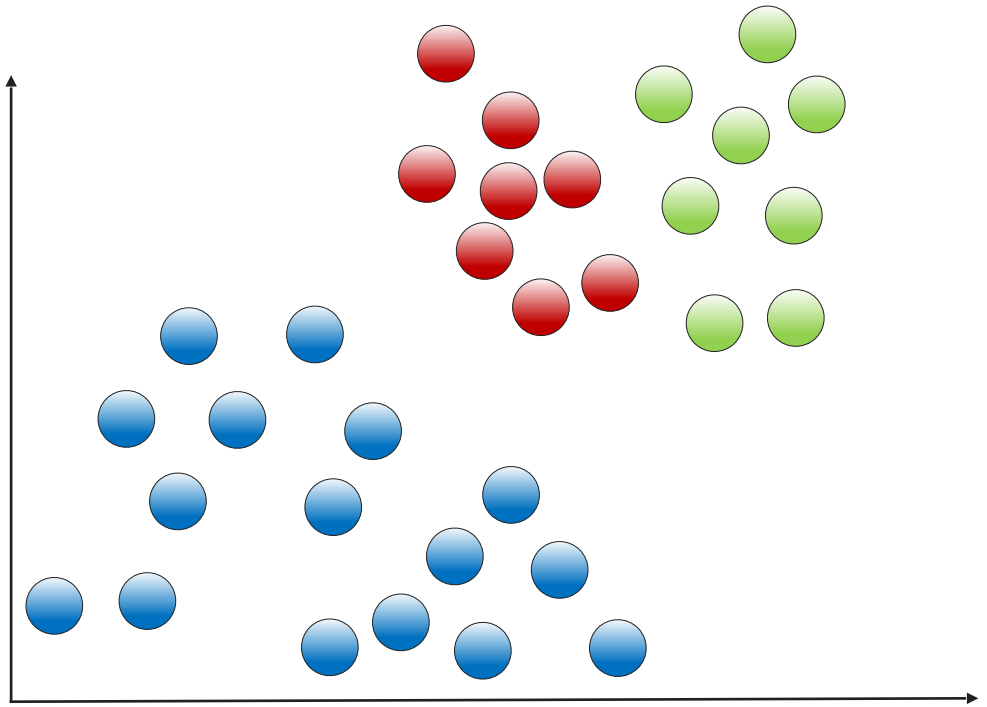
Clustering



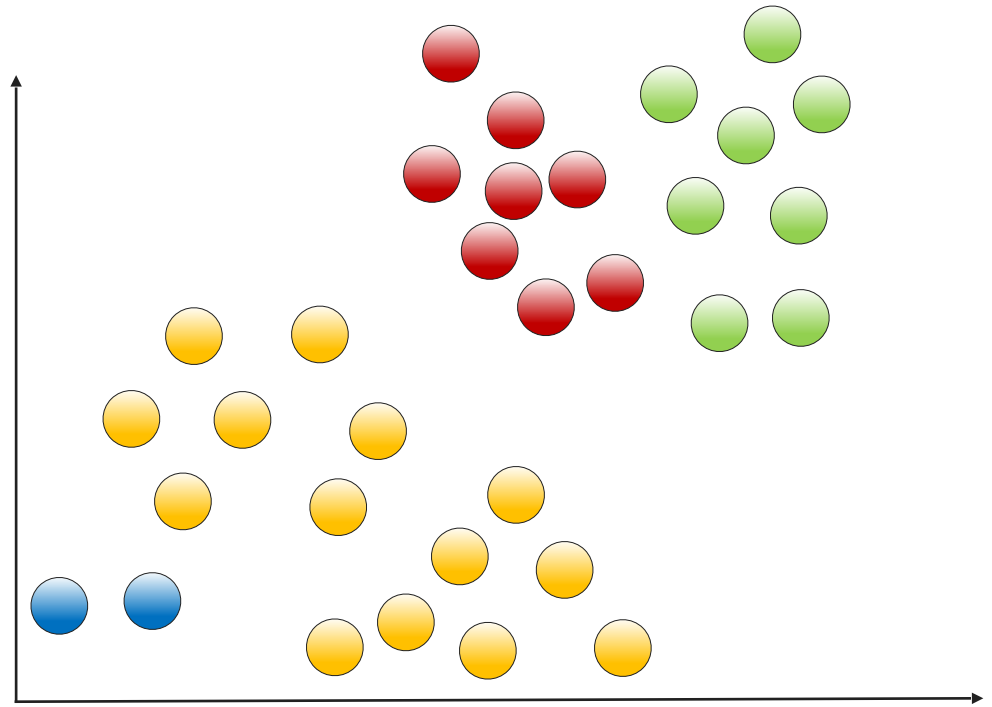
Clustering



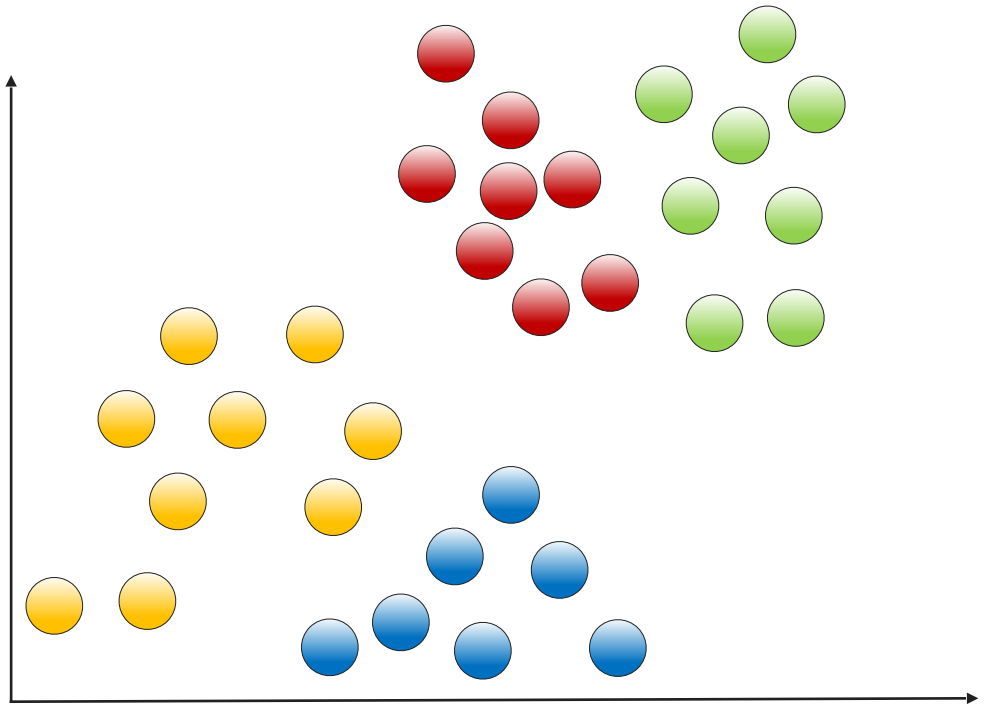
Clustering



Clustering



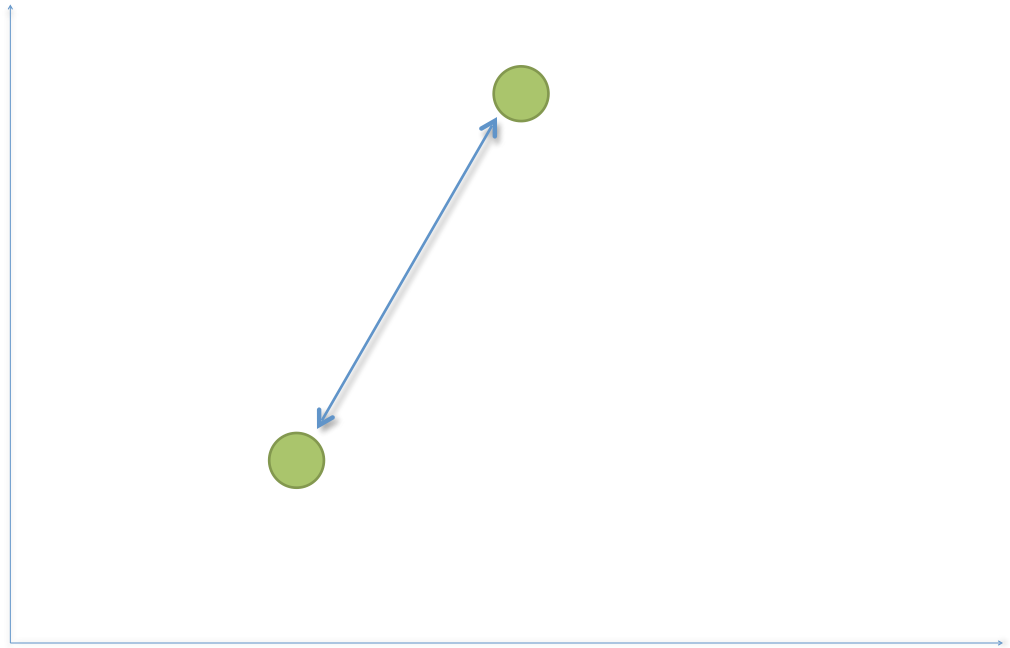
Clustering



Distance Measures

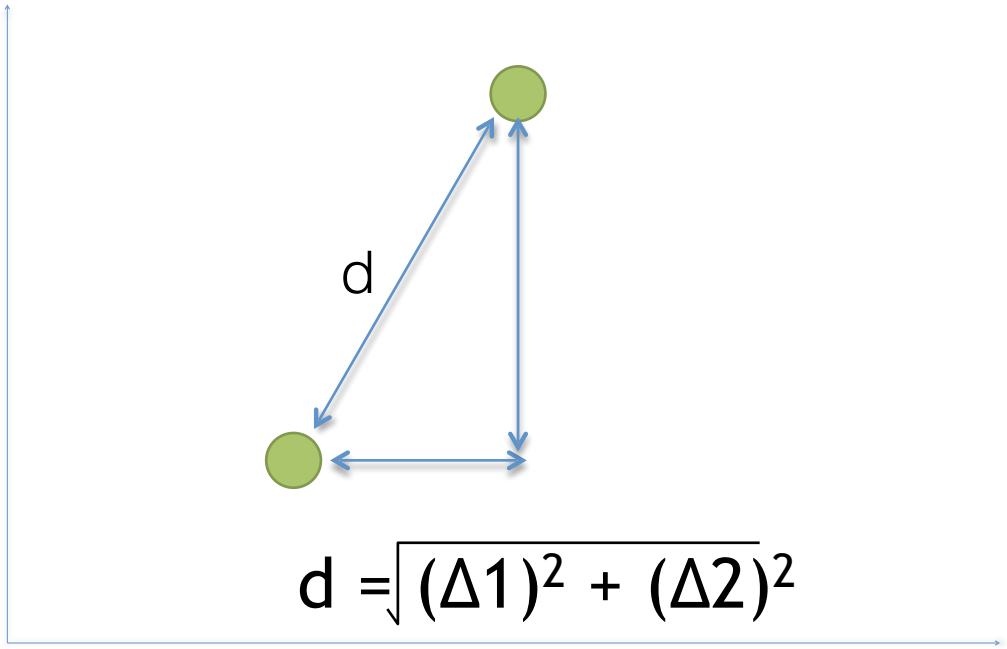
Distance

- We want to measure how close two observations are

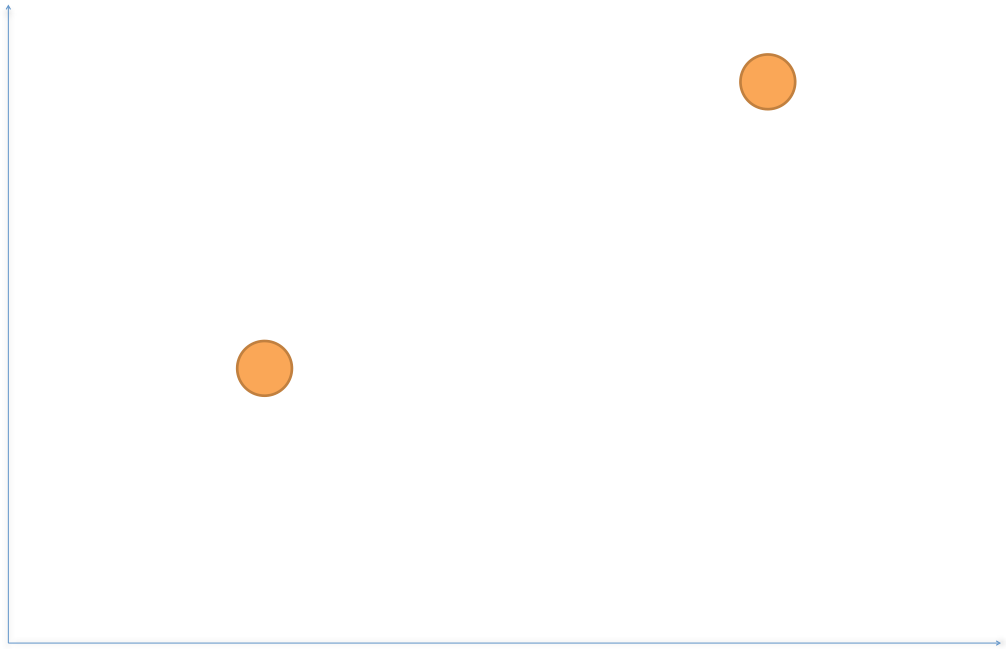


Euclidean Distance

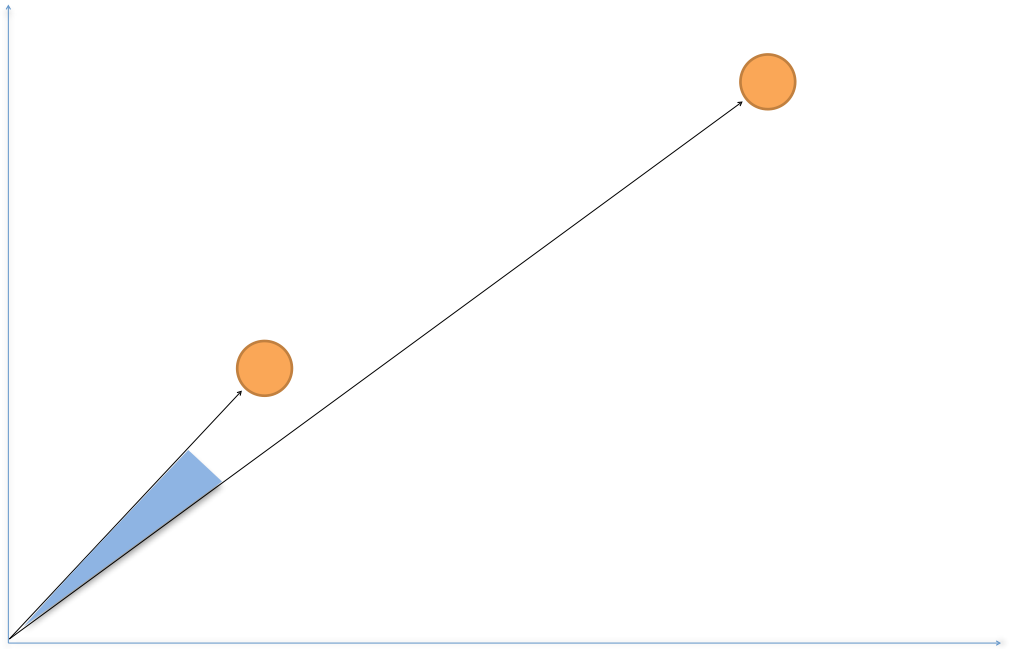
- We want to measure how close two observations are



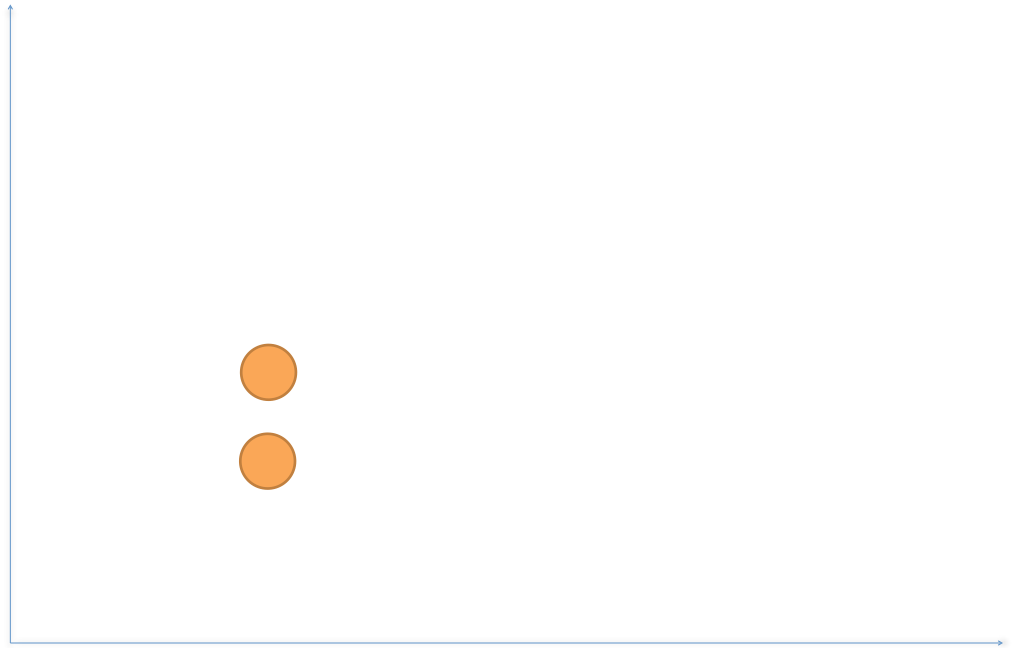
Cosine Distance



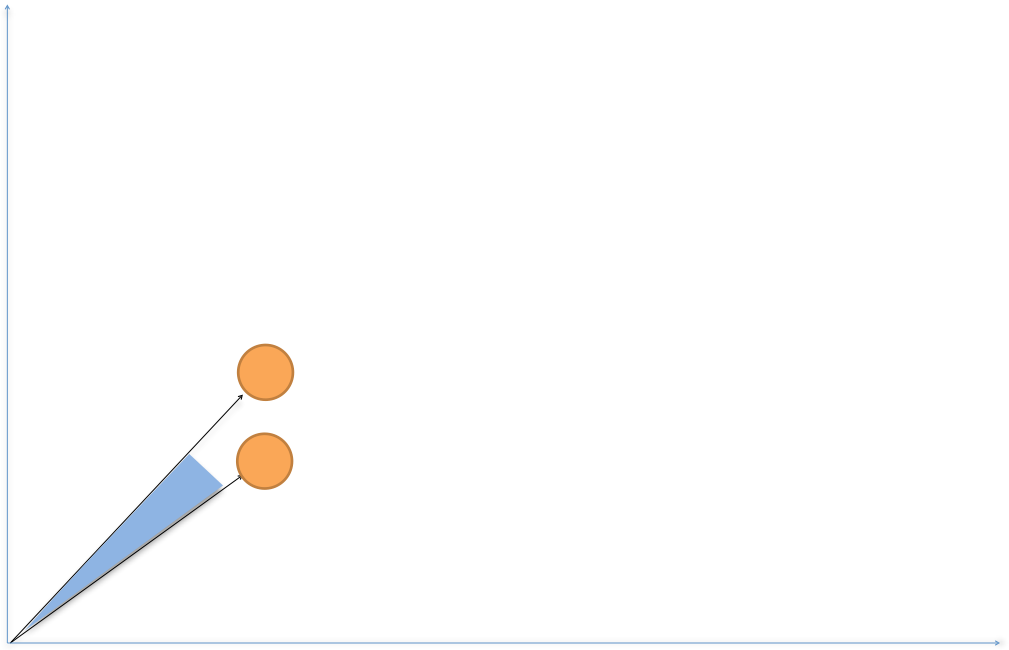
Cosine Distance



Cosine Distance

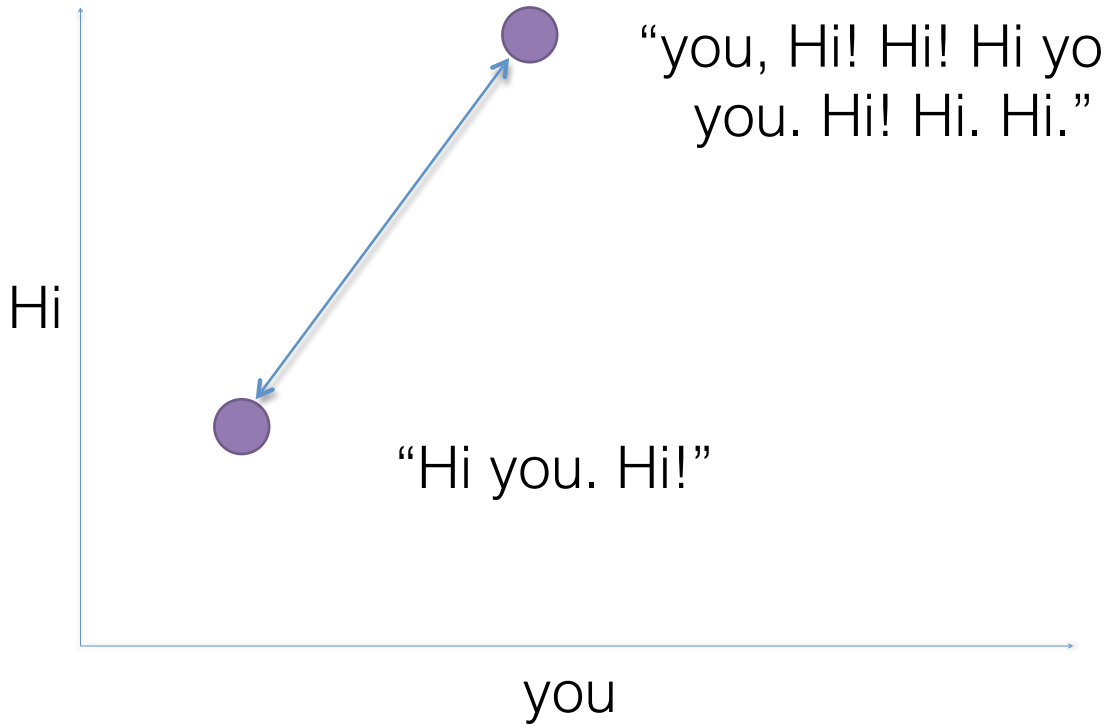


Cosine Distance



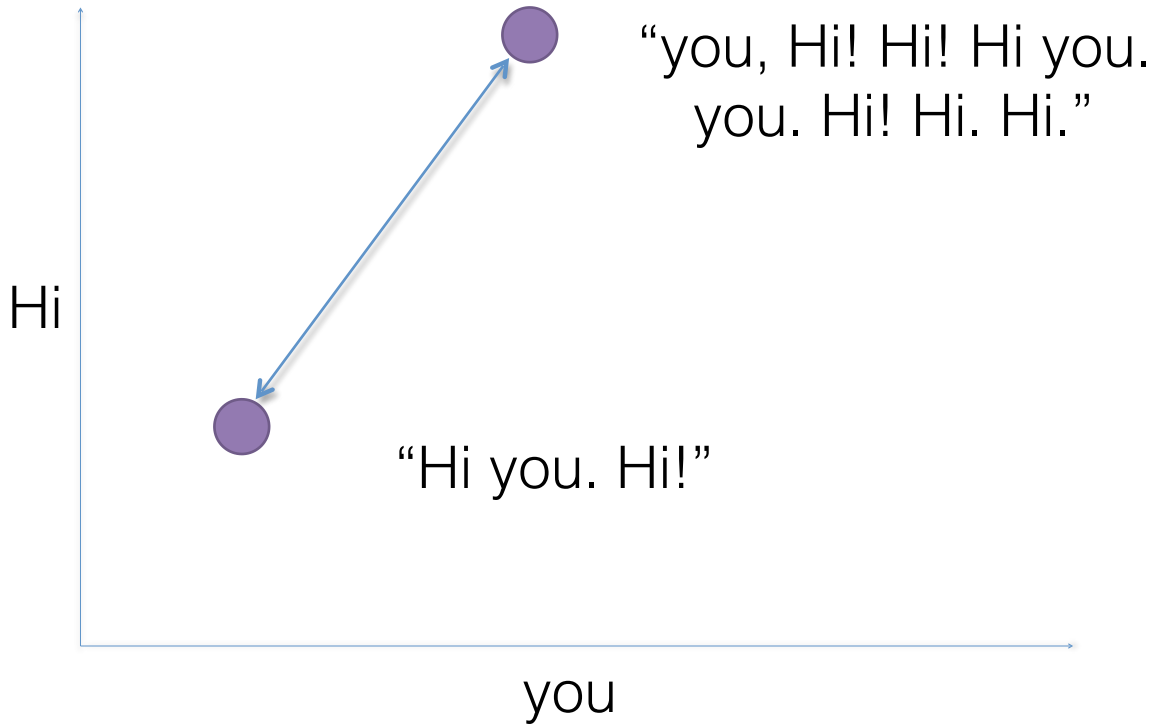
Euclidean Distance

Euclidean distance is sensitive to document length



Cosine Distance

Cosine distance is better for text



Practice

Word2Vec

Still Some Issues

- High Dimensional Sparse Vectors
- Similarity and Meaning
- Represent each words as a low dimensional dense vector

Red: [0,0,0,0,1,0,0,0 ,…, 0]
Blue: [0,0,0,0,0,0,1,0 ,…, 0]
Dog: [0,0,1,0,0,0,0,0 ,…, 0]

Word Embedding

- Represent each words as a low dimensional dense vector

Baseball: [0,0,0,0,1,0,0,0,…,0]
Politics: [0,0,0,0,0,0,1,0 ,…,0]
Football: [0,0,1,0,0,0,0,0 ,…,0]

->

Baseball: [-.5 ,.1, .65, -. 6,.125,.96,.74,-.364]
Politics : [-.433,.96, .54, -.84,.8,.245,.32, .4]
Football : [-.242 ,.153, .87, .452,.5,.84,.4,.3]

- Similar words are closer to one another
- Similarity is based on the neighbors they keep

  I sat on the chair and **read** a book.

  I sprained my **ankle** while playing basketball

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

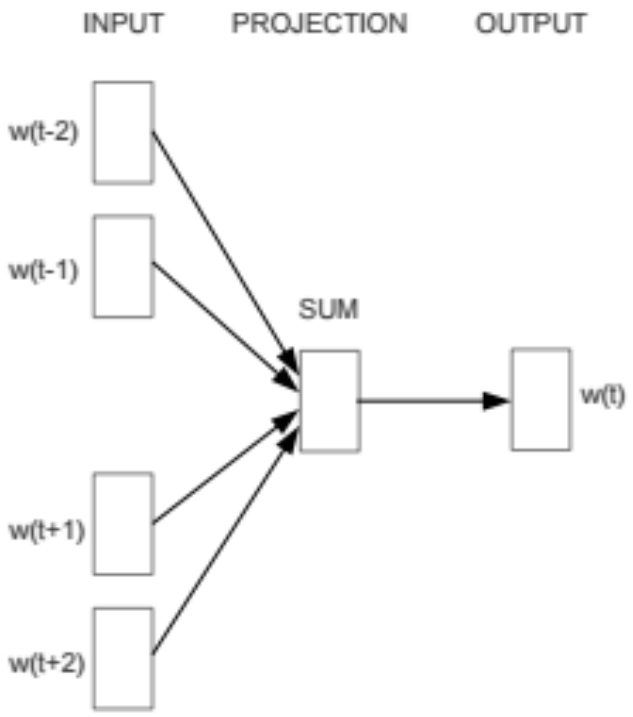
Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

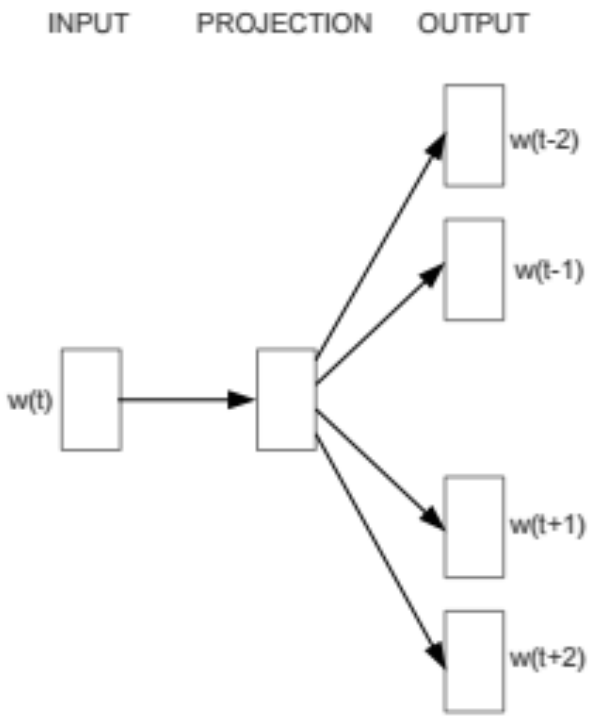
Word2Vec

- **Skip-gram:** Predict surrounding words given center

- **CBOW:** Predict center word given surrounding



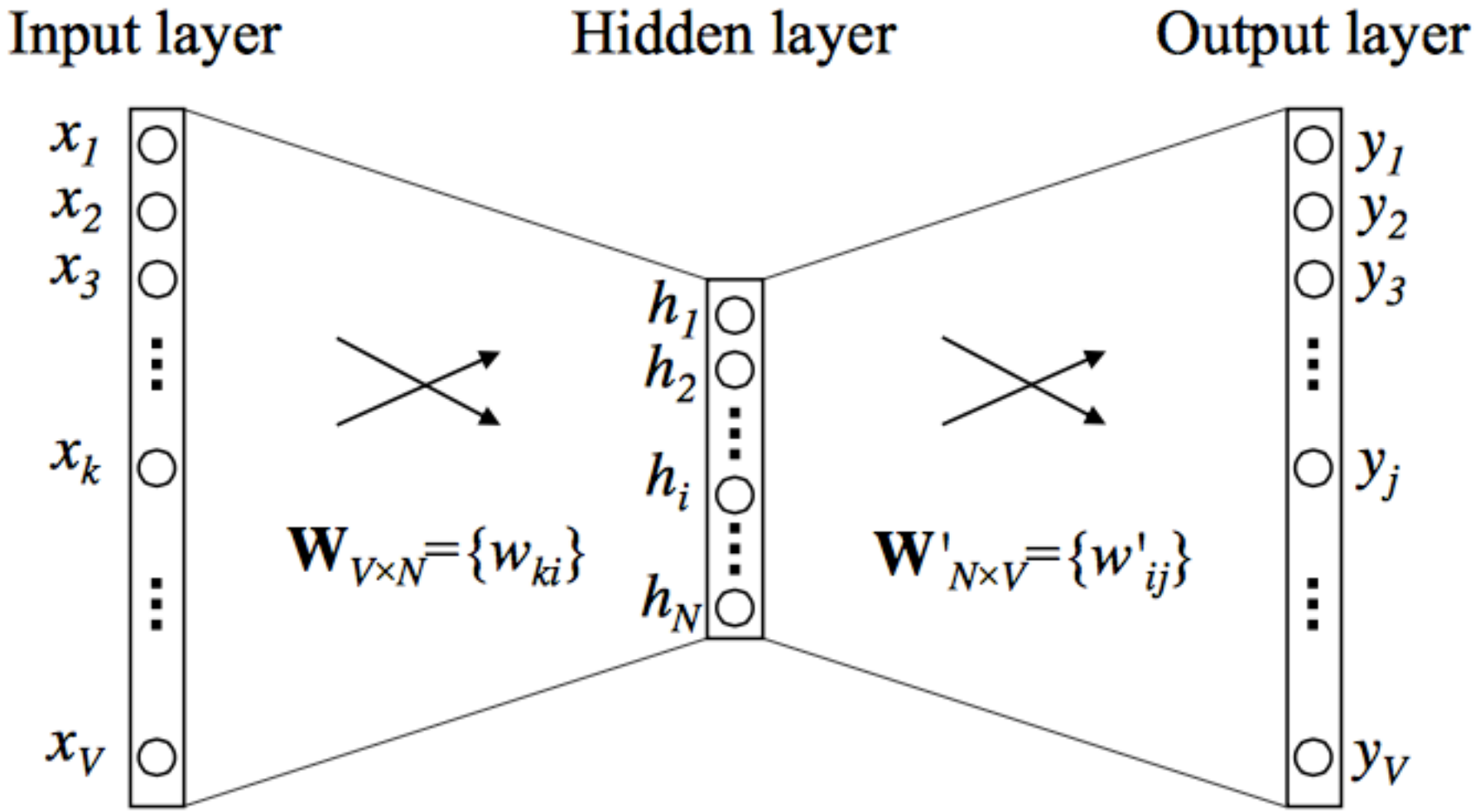
CBOW



Skip-gram

- Similarity is based on the neighbors they keep
  - I sat on the chair and **read** a book.
  - I sprained my **ankle** while playing basketball

Mikolov, et al. 2013


Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$

How do we do this?

- Objective:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j}|w_t)$$

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

- Stochastic Gradient Descent:
  - Initialize the weights to small random numbers (2dW)

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

How do we do this?

- Objective:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t)$$

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W}\exp\left(u_w^T v_c\right)}$$

- Stochastic Gradient Descent:
  - Initialize the weights to small random numbers (2dW)
  - $\theta^{new} = \theta^{old} - \alpha\nabla_\theta J(\theta)$ ← Proportional to W

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

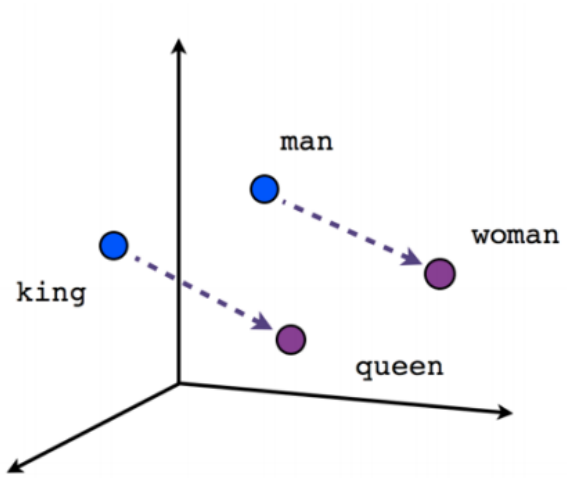
Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

- Negative Sampling (k~[2-20], depending on dataset size)

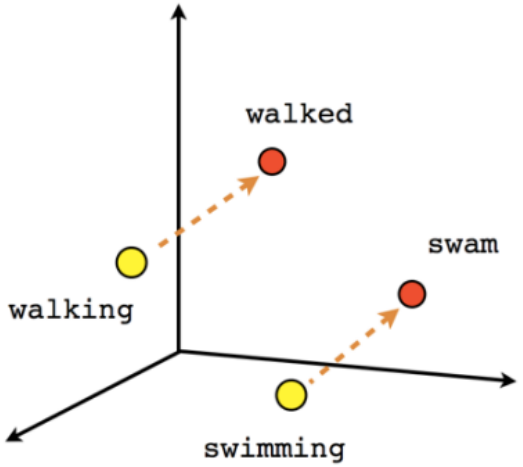
$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

Word Relationships

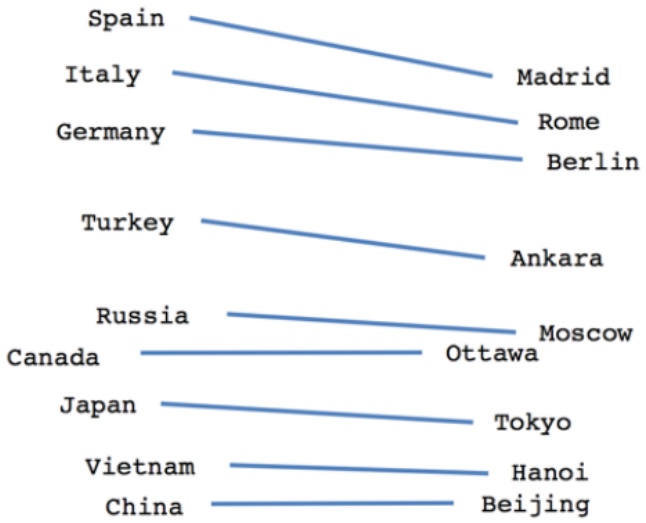
- Displacement vectors between points are representative of relationship
- Words with same relationship lead to the same vector



Male-Female



Verb tense



Country-Capital

https://www.tensorflow.org/versions/r0.8/tutorials/word2vec/index.html

Gensim

```
from gensim.models.word2vec import Word2Vec
model = Word2Vec(size=n_dim)
model.build_vocab(x_train)
model.train(x_train)
```

```
import gensim
from gensim.models.word2vec import Word2Vec

model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

```
print model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
```

```
'queen'
```

```
print model.most_similar(positive=['Canadiens', 'Toronto'], negative=['Montreal'], topn=1)
```

```
'Maple_Leafs'
```

https://code.google.com/archive/p/word2vec/

Gensim

```
model.most_similar(positive=['ate','speak'], negative=['eat'], topn=1)
```

```
    'spoke',
```

```
print model.most_similar(positive=['biggest','small'], negative=['big'], topn=1)
```

```
    'smallest'
```

```
print model.doesnt_match("breakfast cereal dinner lunch".split())
```

```
     cereal
```

```
model['king']
```

```
array([  1.25976562e-01,   2.97851562e-02,   8.60595703e-03,
         1.39648438e-01,  -2.56347656e-02,  -3.61328125e-02,
         1.11816406e-01,  -1.98242188e-01,   5.12695312e-02,
         3.63281250e-01,  -2.42187500e-01,  -3.02734375e-01,
        -1.77734375e-01,  -2.49023438e-02,  -1.67968750e-01,
        -1.69921875e-01,   3.46679688e-02,   5.21850586e-03,
         4.63867188e-02,   1.28906250e-01,   1.36718750e-01,
```

https://code.google.com/archive/p/word2vec/

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the code (licensed under the Apache License, Version 2.0)
- Unpack the files: unzip GloVe-1.2.zip
- Compile the source: cd GloVe-1.2 && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question
- The code is also available on GitHub

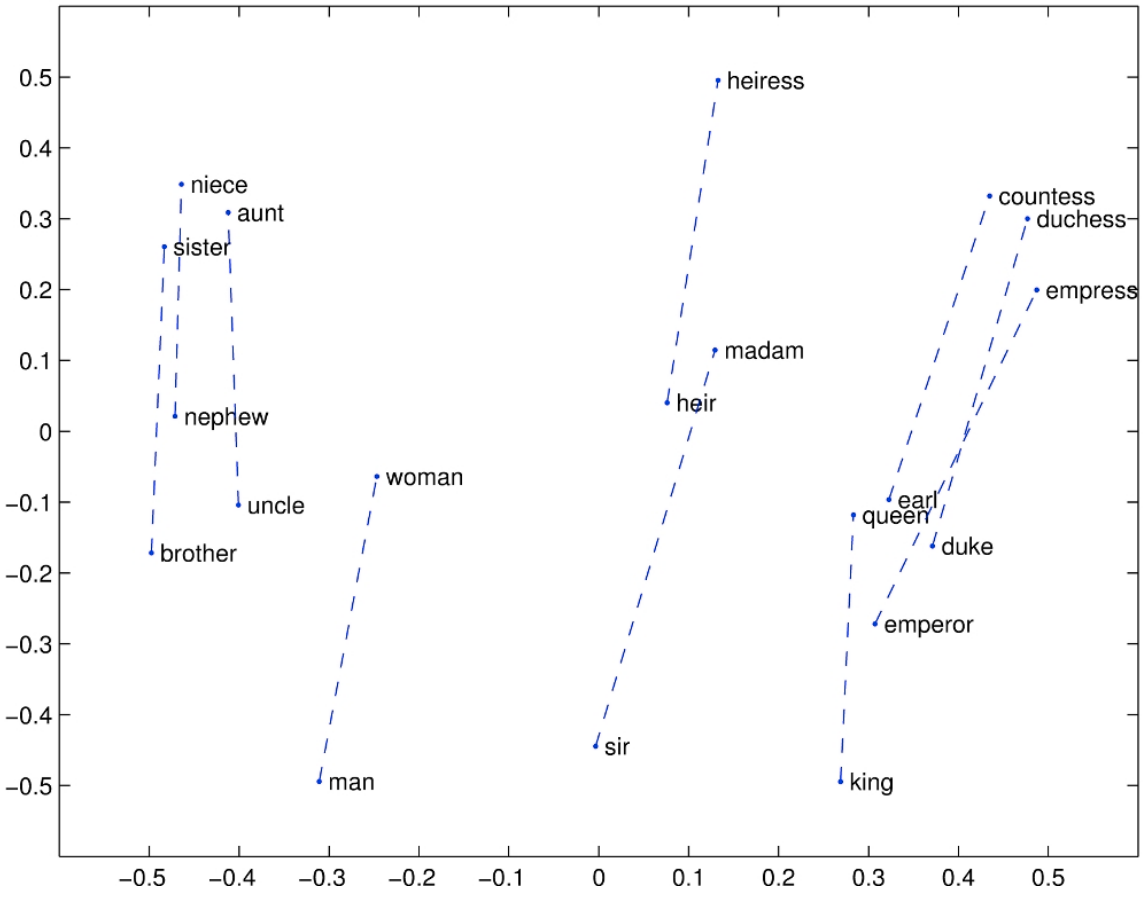
Download pre-trained word vectors

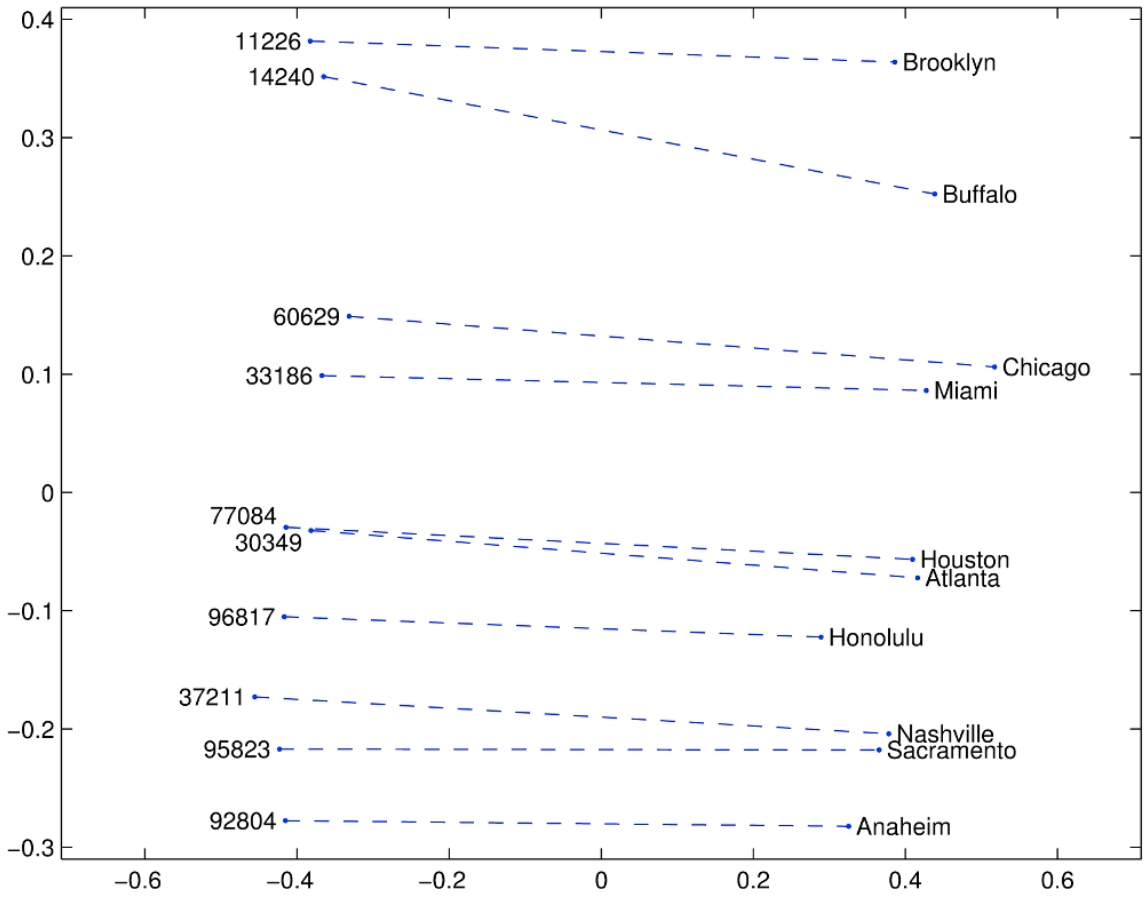
- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Citing GloVe

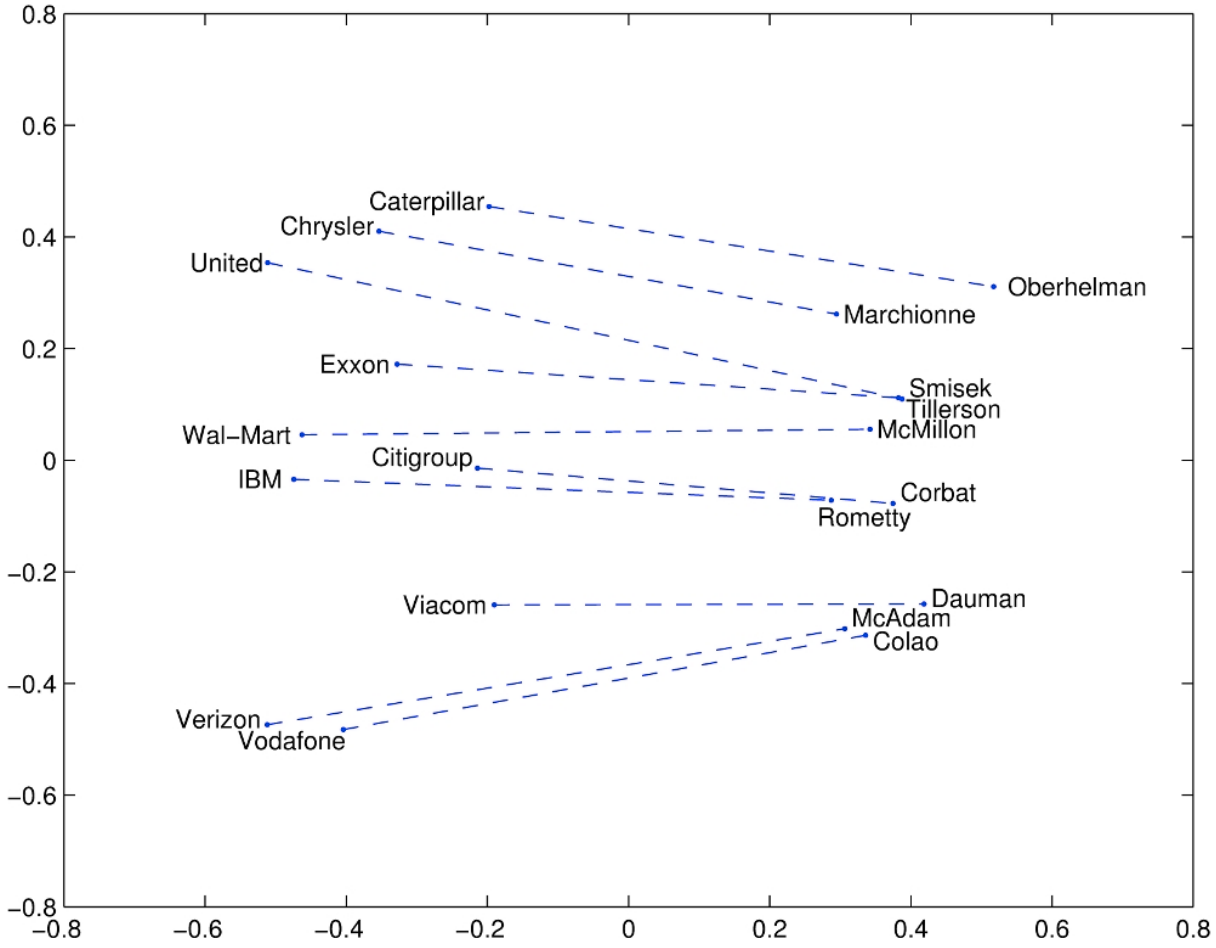
Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. [pdf] [bib]

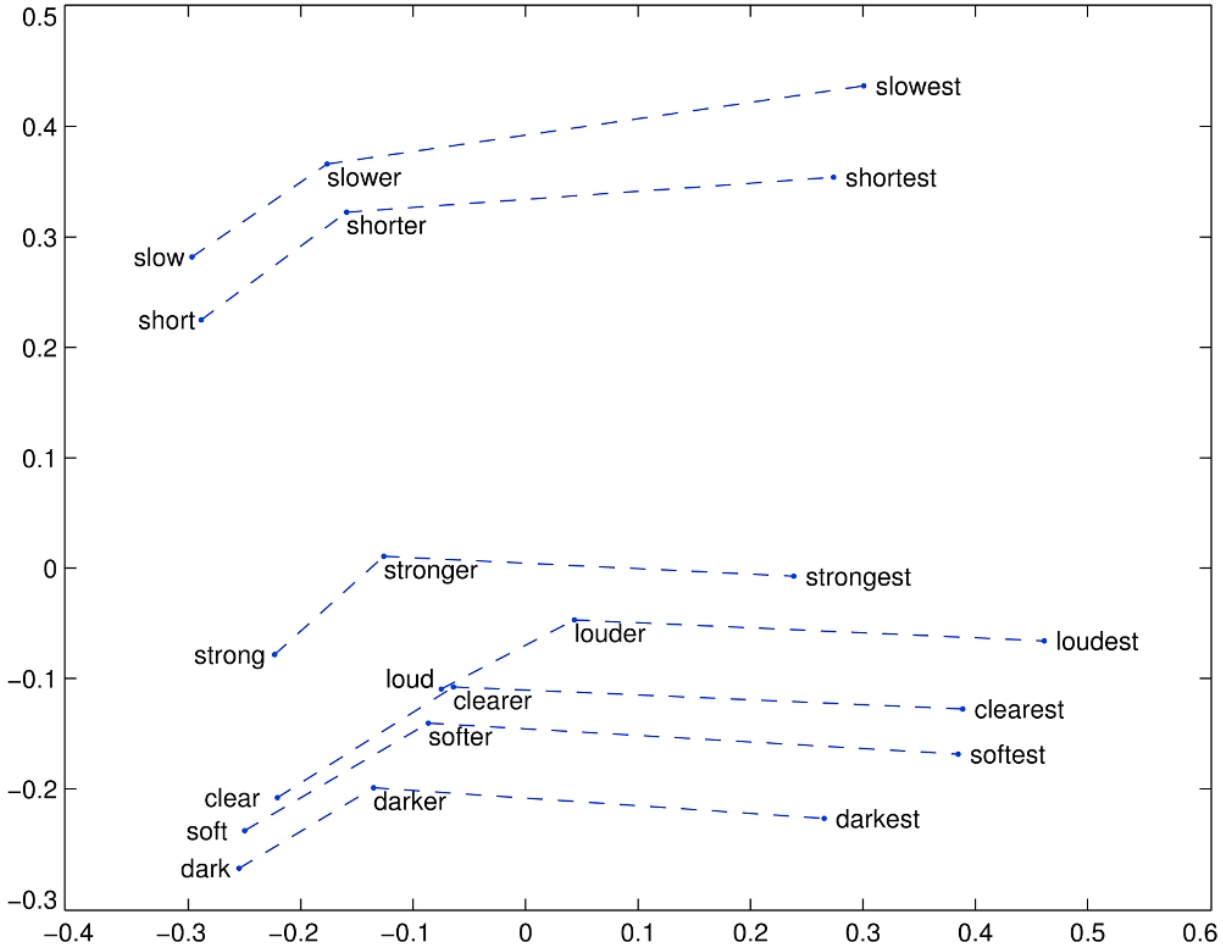
Some interesting Relationships





Some interesting Relationships





Topic Modeling

Topic Modeling

Goal: Discover the topics in a large collection of documents

For example, articles are arranged by topic on a news site

Document

- Document: A distribution of topics
- For example, an article can be 30% about business, 30% about tech, 40% about science, and 0.0% about the other topics.

Topic

- Topic: A distribution over words

Topic 1		Topic 2		Topic 3	
term	**weight**	**term**	**weight**	**term**	**weight**
game	0.014	space	0.021	drive	0.021
team	0.011	nasa	0.006	card	0.015
hockey	0.009	earth	0.006	system	0.013
play	0.008	henry	0.005	scsi	0.012
games	0.007	launch	0.004	hard	0.011

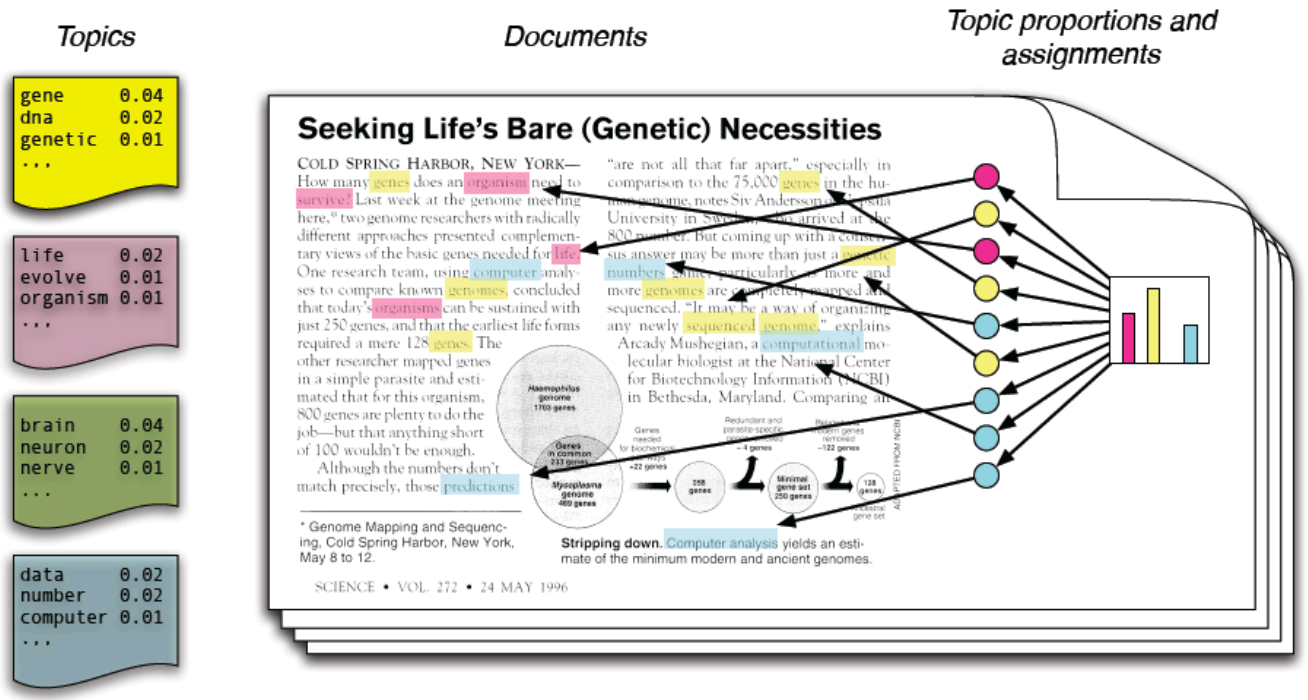
- Topic 1 is about sports so it uses words like “game” and “team” more than topic 2 which is about space

- Topic 2 can also use the words “team” or “game” but less frequently than topic 1

Goal

Find Document Topic Distribution

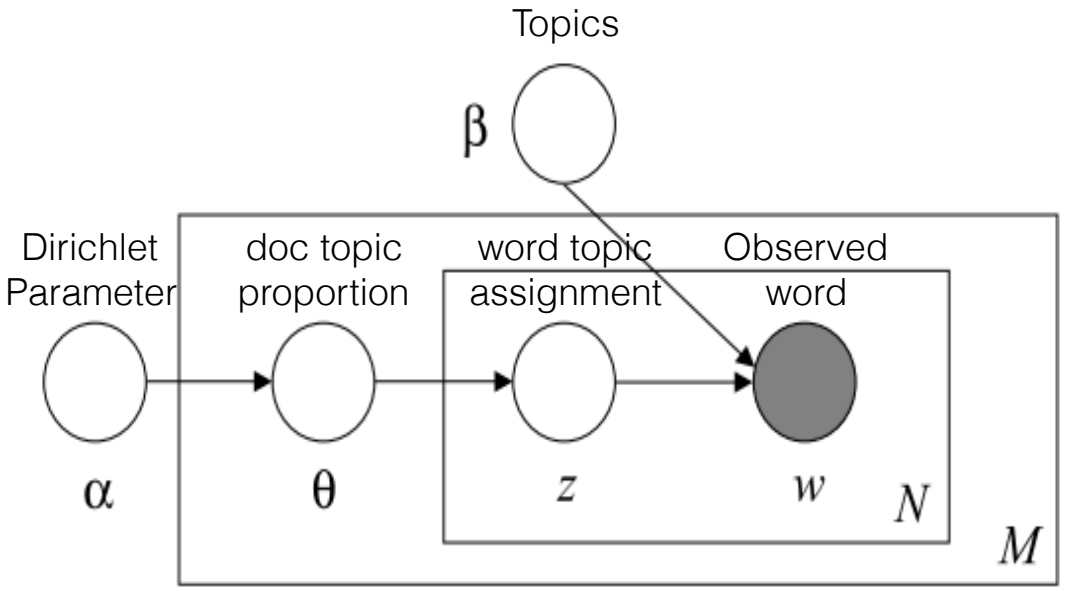
Find
Topic Word
Distribution



Latent Dirichlet Allocation (LDA)

- Generative Model: Models how the data is generated
- Assume a set of topics shared by all documents
- To generate a document we:
    - generate a topic from the document topic distribution
    - Generate a word from that topic
    - Repeat

More Formally



What now?

- We’re given a set of documents that we assumed were generated this way. Then the generative process is reversed
- We won’t go into the math
- Let’s practice in Python

Thank you

- michael@thisismetis.com
- @MikeJGalvin
- https://github.com/galvin-mj

- www.thisismetis.com
- @thisismetis