MTech DSML (PES)

Arun Sharma K

Day Plan

Need and Applications

Workflow

NLP fundamentals

Hands on

Typical questions

Q&A

Participate

Sessions

1

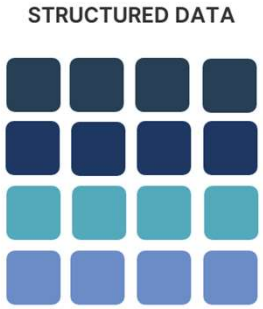
- Unstructured Vs structured
- Need for NLP
- Typical applications
- Aspects of NLP
- Workflow
- Clean-up
- EDA
- Hands-on

2

- Feature engineering
- Frequency based
- Sematic based
- Hands-on

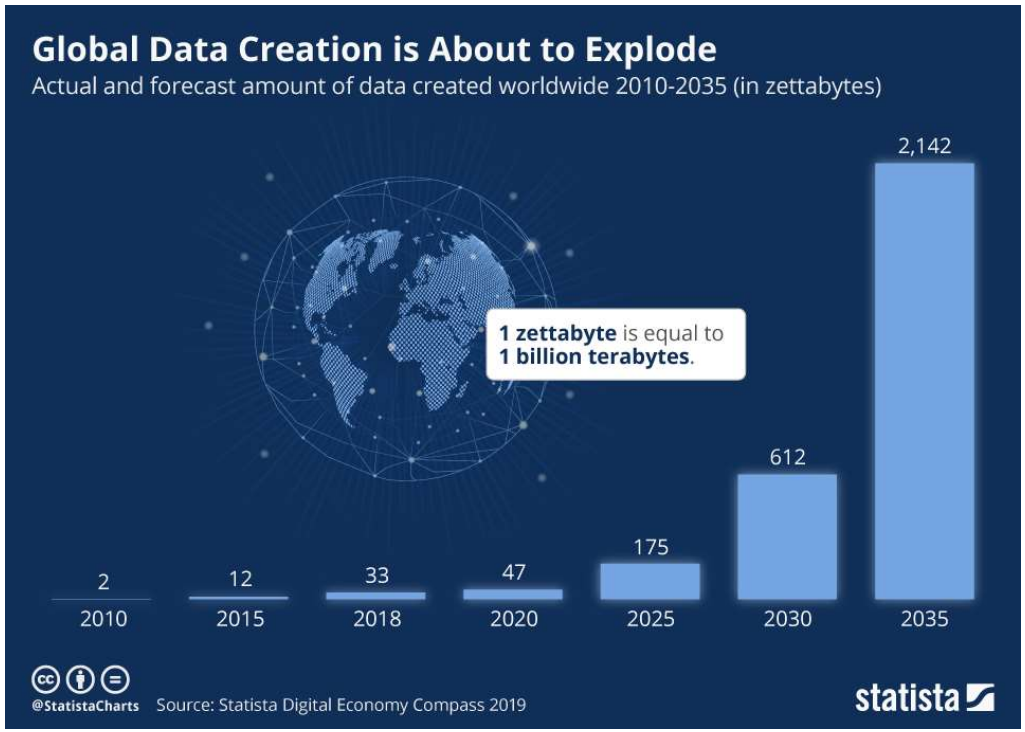
NLP:Session-1

Structured Vs Unstructured



Fits neatly into a database	Lacks a clear structure
Easy to retrieve and analyze	Needs special tools to be useful
Goes into a data warehouse	Requires more complex storage
Enables quick decision-making	Takes longer to analyze
Uses basic tools like spreadsheets, SQL	Requires advanced tools like NLP, ML

Need for NLP



Growing volume of text, streaming audio, video, clickstream, sensor and log data→ ~90% of worldwide data will be unstructured by 2025 [IBM]

Need for NLP

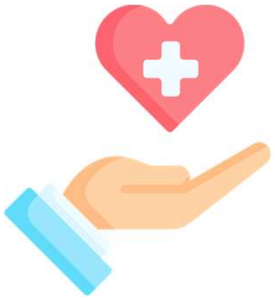
Huge emphasize from Organizations to focus on develop robust strategies for effectively

- Managing
- Storing
- Analyzing

Harness the data for better, data-driven decisions

Value

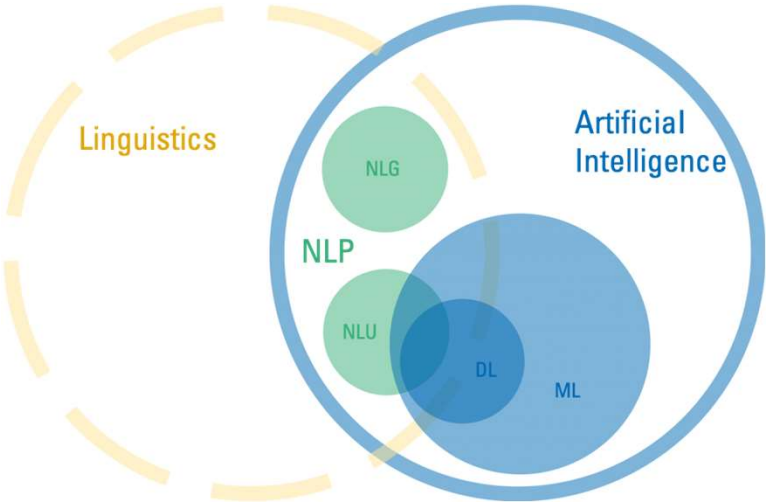
- 25% possible annual savings
- 31% reduction in 30-day readmission for certain patients
- USD 500,000 savings per year

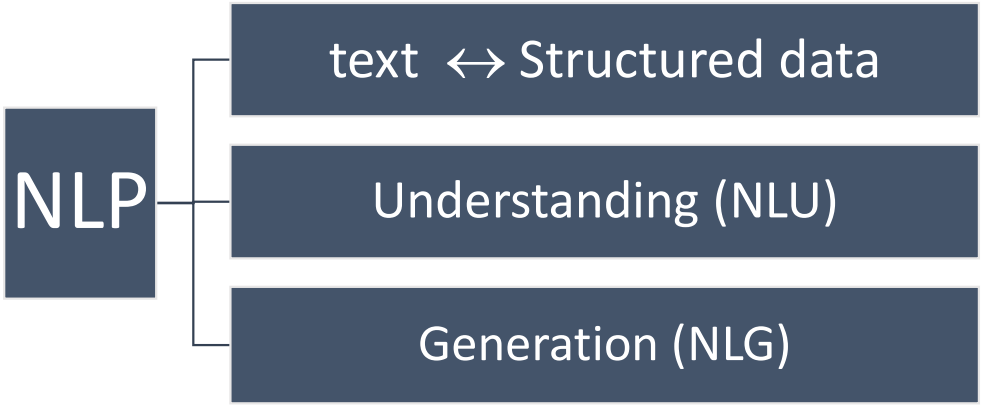


- 30% fewer fraud incidents
- Nearly USD 4 million reduction in expenditure
- 90% quicker time-to-value for big data analytics

https://www.ibm.com/think/insights/managing-unstructured-data#_edn2

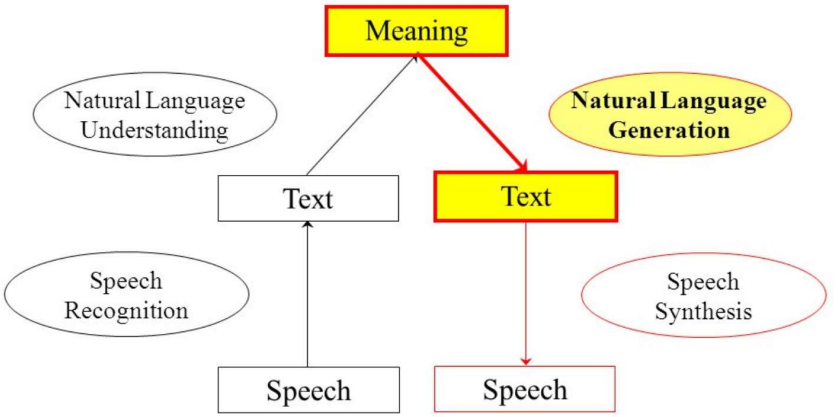
What is NLP


Linguistics
Artificial Intelligence
NLG
NLP
NLU
DL
ML


NLP
text ↔ Structured data
Understanding (NLU)
Generation (NLG)

NLU Vs NLG

- NLG is the inverse of NLU
- NLG maps from meaning to text, while NLU maps from text to meaning
- NLG is easier than NLU because a NLU system cannot control the complexity of the language structure it receives as input while NLG links the complexity of the structure of its output.



NLU
Ambiguity in input
ill-formed input
Under-specification

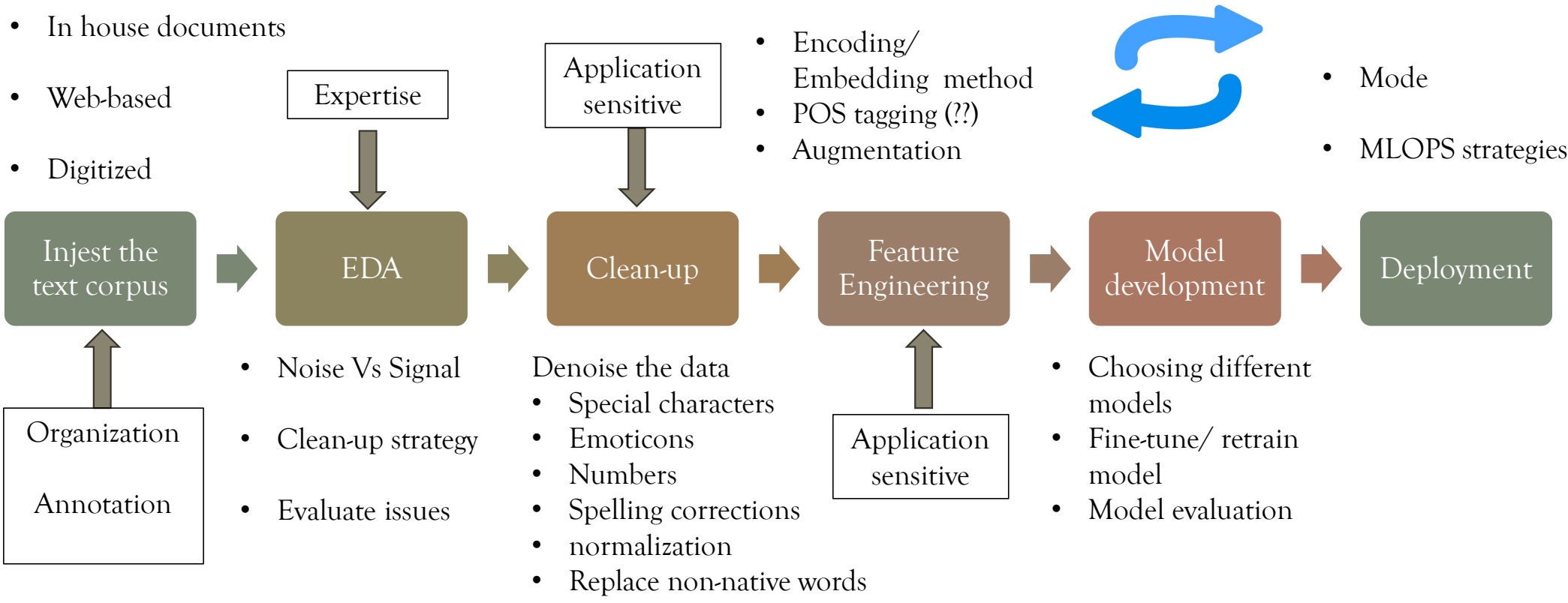
NLG
Relatively Unambiguous
Well-formed
Well-specified

http://www.lacsc.org/papers/PaperA6.pdf

Typical Applications of NLP

- Spell Checking
- Text Classification
- Sentiment Analysis/opinion mining
- Question Answering
- Automatic Summarization
- Text suggestion
- Machine Translation(statistical machine translation)
- Speech Recognition

Workflow


In house documents
Web-based
Digitized
Expertise
Application sensitive
Encoding/ Embedding method
POS tagging (??)
Augmentation
Mode
MLOPS strategies
Injest the text corpus
EDA
Clean-up
Feature Engineering
Model development
Deployment
Organization
Annotation
Noise Vs Signal
Clean-up strategy
Evaluate issues
Denoise the data
Special characters
Emoticons
Numbers
Spelling corrections
normalization
Replace non-native words
Application sensitive
Choosing different models
Fine-tune/ retrain model
Model evaluation

Steps in preprocessing

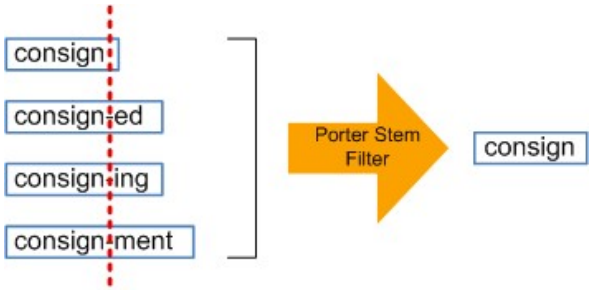
- Removing punctuations
- Tokenization
- Remove stop words
- Stemming
- Lemmatizing
- POS tagging
- Vectorizing Data
  - Bag of words/CountVectorizer
  - N-Grams
  - TF-IDF

Tokenization

- Sentence tokenization→ split collection of sentences to individual sentences
- Word tokenization→ split sentence into words
- Stop words→ words that do not hold much importance in processing compared to keywords
- Nltk stop words
- Adding new stop words

Stemming

- Stemming: heuristically removing the affixes of a word, to get its stem (root) word
- Stem (root) is the part of the word to which you add inflectional (changing/deriving) affixes such as (-ed,-ize, -s,-de,mis)
- Stemming :reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language



https://www.datacamp.com/community/tutorials/stemming-lemmatization-python

Lemmatization

- Lemmatization: unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma.

- A lemma is the dictionary form, or citation form of a set of words.

- *runs, running, ran* are all forms of the word *run*, therefore *run* is the lemma of all these words

REGEX

Regex

Sequences of characters that form search patterns

Regex	Description	Example Matches
.	Any single character except newline	a.b → "acb", "a_b"
\d	Any digit (0-9)	\d → "5", "123"
\D	Any non-digit character	\D → "a", "@"
\w	Any word character (a-z, A-Z, 0-9, _)	\w → "a", "1", "_"
\W	Any non-word character	\W → "!", " "
\s	Any whitespace (space, tab, newline)	\s → " ", "\t"
\S	Any non-whitespace character	\S → "a", "9"
^	Start of a string	^Hello → "Hello World"
$	End of a string	end$ → "The end"
*	0 or more repetitions	a* → "", "aaaa"
+	1 or more repetitions	a+ → "a", "aaa"
{n}	Exactly n repetitions	a{3} → "aaa"
()	Grouping	(abc)+ → "abcabc"
[]	Character set	[a-z] → "a", "z"

Examples

Regex to extract email

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

- ^ asserts the start of the string.
- [a-zA-Z0-9._%+-]+ matches the username (1+ alphanumeric, dots, underscores).
- @ matches the literal "@" character.
- [a-zA-Z0-9.-]+ matches the domain name.
- \.[a-zA-Z]{2,} matches the top-level domain (e.g., ".com").
- $ asserts the end of the string.

Regex to extract phone#

```
^\d{10}$
```

- ^ asserts the start.
- \d{10} matches exactly 10 digits.
- $ asserts the end.

Feature engineering

Encoding and Embedding types

Unstructured to structured

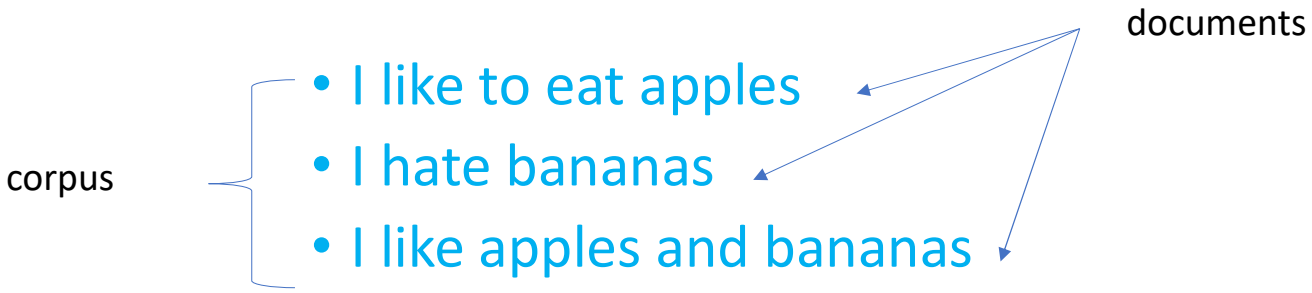
Frequency based

- Count vectorization
- TF-IDF vectorization
- N-gram

Semantic based

- word2Vec
- Glove
- ELMO
- BERT, SBERT
- GPT
- Custom embedding

Bag of words



Vocabulary ={and, apples, bananas, eat, hate, I, like, to}

	And	Apples	bananas	eat	hate	I	like	to	TARGET
1	0	1	0	1	0	1	1	1	1
2	0	0	1	0	1	1	0	0	0
3	1	1	1	0	0	1	1	0	1

I like to eat apples → [0,1,0,1,0,1,1,1]

TF-IDF

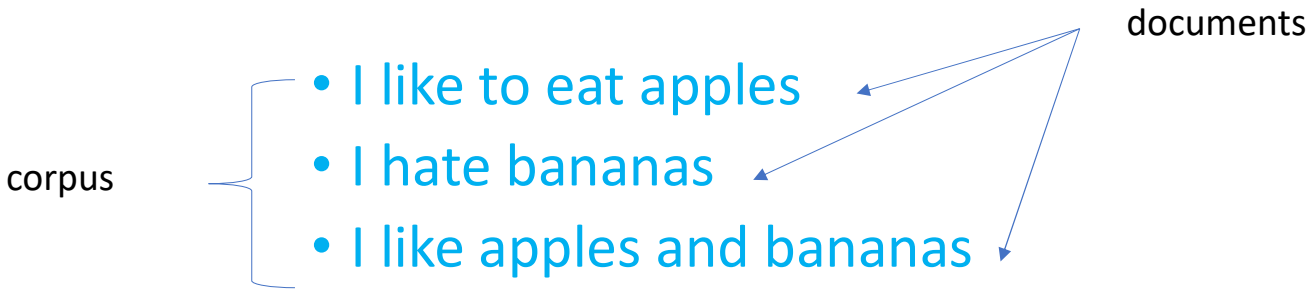
Statistical measure used to evaluate how important a word is to a document in a collection or corpus.

- TF(t,d) = count of ‘t’ in d / number of words in d
- IDF(t) = log(N/(df(t) + 1))
- df(t) = occurrence of t in documents
- N= total number of documents in the corpus
- **TF-IDF(t, d) = tf(t, d) * log(N/(df + 1))**

Term Frequency
This measures the frequency of a word in a document. This highly depends on the length of the document and the generality of word

Inverse Document Frequency
This measures the importance of the document in the whole corpus. DF is the count of occurrences of term t in the document set N

Bag of words



Vocabulary ={and, apples, bananas, eat, hate, I, like, to}

	And	Apples	bananas	eat	hate	I	like	to	TARGET
1	0	1	0	1	0	1	1	1	1
2	0	0	1	0	1	1	0	0	0
3	1	1	1	0	0	1	1	0	1

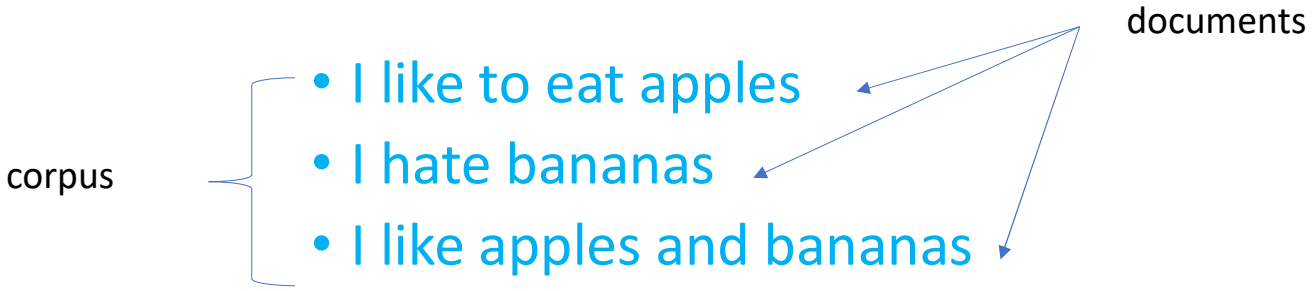
TF (I)= 1/5 =0.2
IDF (I)= ln(3/(3+1))=-0.28
TF-IDF(I)=0.2*-0.28=-0.05754

TF (Apples)= 1/5 =0.2
IDF (Apples)= ln(3/(2+1))=0
TF-IDF(Apple)=0.2*0=0

TF (to)= 1/5 =0.2
IDF (to)= ln(3/(1+1))=0.4
TF-IDF(to)=0.2*0.4=0.08

I like to eat apples → [0, 0, 0, 0.08, 0, -0.05754, 0, 0.08]

Bag of words-N grams



Vocabulary ={and, apples, bananas, eat, hate, I, like, to, I like, like to, to eat, eat apples, I hate, hate bananas, like apples, apples and, and bananas }

#	And	Apples	bananas	eat	hate	I	like	to	I like	like to	to eat	eat apples	I hate	hate bananas	like apples	apples and	and bananas	TARGET
1	0	1	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1
2	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
3	1	1	1	0	0	1	1	0	1	0	0	0	0	0	1	1	1	1

I like to eat apples → [0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]

NLP: Text embeddings

Session-2

Word embedding

- **Basic CountVectors→ limitations**
  - The product is good→ [1,1,1,1]
  - Is the product good→ [1,1,1,1]
  - Vocabulary →[good, is, product, this]
- **Count-Based Vector Space Model**
- Arrangement of same words in a different order (syntactic) could change the meaning. Which is not captured by CountVectors
- Sequencing issue

Word embedding

- Have a great time→ [0,1,1,1]
- Have a good time→ [1,0,1,1]
- Vocabulary→[good, great, have, time]
- Semantically similar sentences.
- Different representations by CountVectors/OHE
- Each word is treated as an orthogonal dimension and similarity between words is not captured
- This technique leads to a high dimensional sparse matrix

Context matters

- this is a river bank
- this is icici bank

- he is a rich man
- he is rich manuel

Adjacent words add to/ modify the meaning of the word

Word embedding

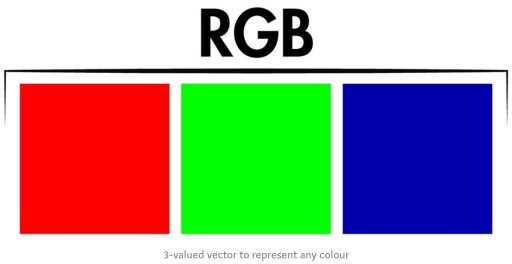
- Word embeddings are word representation that allows words with similar meaning to have a similar representation
- leads to a low dimensional dense matrix
- Each word is represented by a real-valued vector, often tens or hundreds of dimensions.
- This is contrasted to the thousands or millions of dimensions required for sparse word representations
- Convert input OHE matrix (HDS) to real-valued vector(LDD)
- **semantic-Based Vector Space Model**

Word embedding: the idea

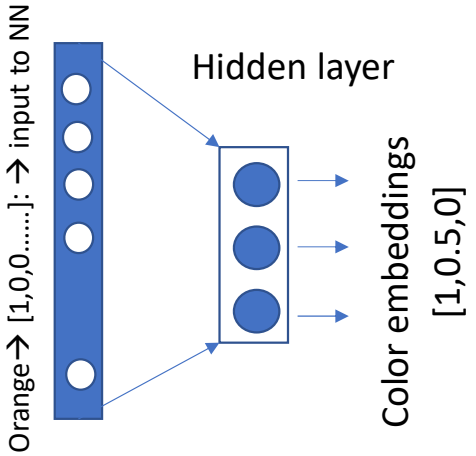
16 colors

Each location in the vector represents a different colour

Orange→ [1,0,0……]: the OHE way



Orange→ [R,G,B]:[1,0.5,0] the feature vector way



- LD representation
- Captures similarity→
  - compare orange and Blue → [1,0.5,0]-[0,0,1]→ large difference
  - compare orange and red → [1,0.5,0]-[1,0,0]→ less difference
- Essentially a FA type DR method

Word embedding→ A DR technique applied to text data

Word embedding: the idea

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

Each word gets a 1x9 vector representation

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

lower dimensional embedding

	Femininity	Youth	Royalty
Man			
Woman			
Boy			
Girl			
Prince			
Princess			
Queen			
King			
Monarch			

	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1

Each word gets a 1x3 vector

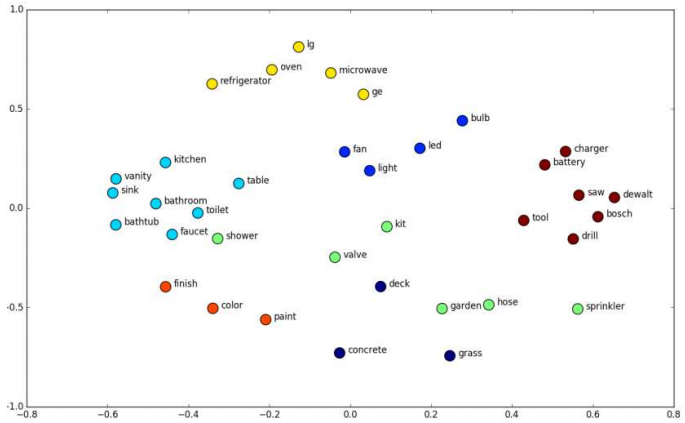
Similar words... similar vectors

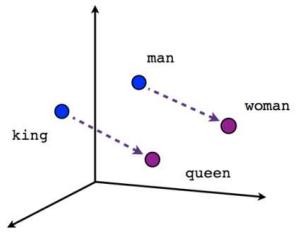
1. Each word is represented with a lower-dimensional vector (3 instead of 9).
2. Similar words have similar vectors here. There's a smaller distance between the embeddings for "girl" and "princess", than from "girl" to "prince". In this case, distance is defined by Euclidian distance.
3. The embedding matrix is much less sparse (less empty space),
4. We could add more words to the vocabulary without increasing the dimensionality. For instance, the word "child" might be represented with [0.5, 1, 0].
5. Relationships between words are captured and maintained, e.g. the movement from king to queen, is the same as the movement from boy to girl, and could be represented by [+1, 0, 0].

https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/

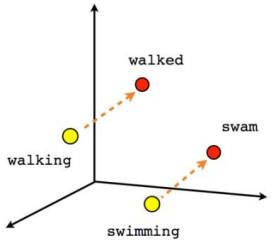
Outcomes of WE

- Recognizes words that are similar,
- Naturally captures the relationships between words as we use them

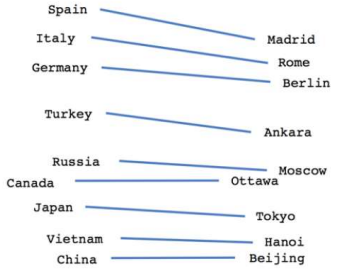




Male-Female



Verb tense



Country-Capital

- The linear relationships between words in the embedding space lend themselves to unusual word algebra, allowing words to be added and subtracted, and the results actually making sense. For instance, in a well defined word embedding model, calculations such as (where [[x]] denotes the vector for the word ‘x’)

[[king]] – [[man]] + [[woman]] = [[queen]]

[[Paris]] – [[France]] + [[Germany]] = [[Berlin]]

Extracting word embedding

- The focus is to optimize the embeddings such that the core meanings and the relationships between words is maintained

- The central idea of word embedding training is that similar words are typically surrounded by the same “context” words in normal use

Different Embedding approaches

1. Word2Vec
2. Glove
3. ELMO (Sequential model-based)
4. BERT (Transformer-based)

$$\left[\begin{bmatrix}0.1\\0.3\\0.8\\0.9\\|\\|\\|\\0\end{bmatrix},\begin{bmatrix}0\\0.4\\0.6\\0.2\\|\\|\\|\\0.5\end{bmatrix},\begin{bmatrix}0.2\\0.3\\0.4\\0.7\\|\\|\\|\\0.1\end{bmatrix},\begin{bmatrix}0\\0.1\\0\\0.7\\|\\|\\|\\0.2\end{bmatrix}\right]$$

n × m Tensor for ‘m’ words/tokens

‘n’ dimensional matrix for each word

How to use this Tensor as input?

1) Aggregation→ Not a smart idea
2) Sequential models
3) Transformers