(MSc) Data Mining

Topic 12 : Text Mining

Part 01 : Overview

Dr Bernard Butler and Dr Kieran Murphy

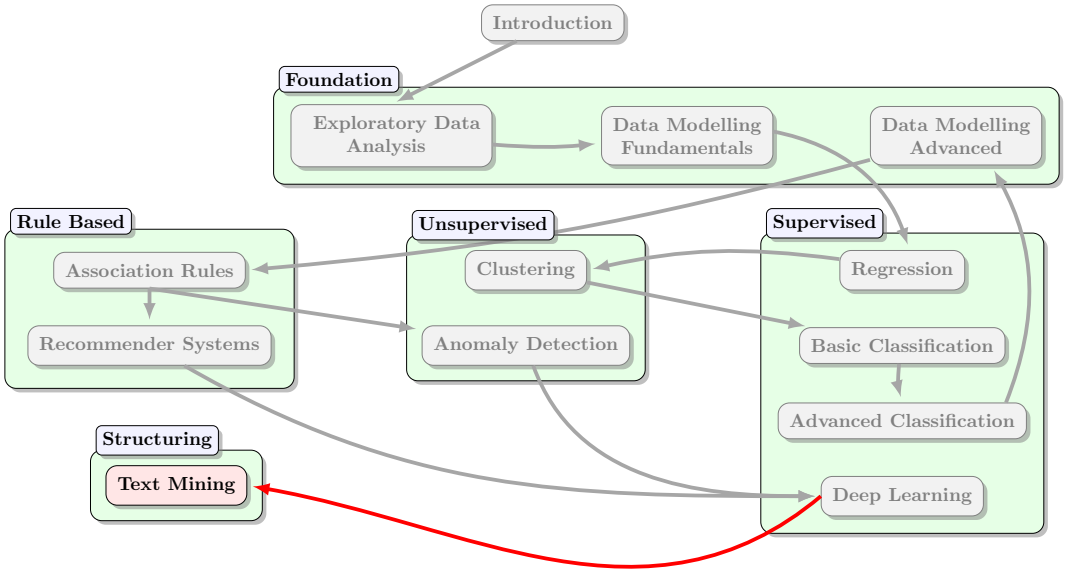
Department of Computing and Mathematics, WIT.
(bernard.butler@setu.ie; kmurphy@wit.ie)

Spring Semester, 2025

Outline

- What is text?
- Preparing text
- Analysing text
- Adding deep learning

Data Mining (Week 12)



Outline

1. Introduction 3

2. Natural Language 5

3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

6. Summary 61

This Week's Aim

This week's aim is to introduce the main concepts and representative algorithms of text mining, also known as text analytics, applied to *Natural Language Processing*.

- What is natural language?
- Pipeline models of text mining
- Insights from text
- Adding deep learning

Unstructured natural language requires a lot of processing before it can be used to make inferences about a topic, author or subject area.

Outline

1. Introduction 3

2. Natural Language 5

3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

6. Summary 61

Different types of languages

Formal languages

Expression-oriented languages, like those used in mathematics (for algebra and logic), regular expressions (for matching strings) and chemistry (for formulae like CH_3COOH) are examples of formal languages. Others formal language include general *programming languages* like python and Java, as well as *domain-specific languages* that range from application-specific configuration scripts, to SQL and HTML. Generally, humans *designed* formal languages to work with computers.

Natural Languages

These are languages used between humans, like English, Spanish and Mandarin Chinese, that *evolved* over time.

Example formal language

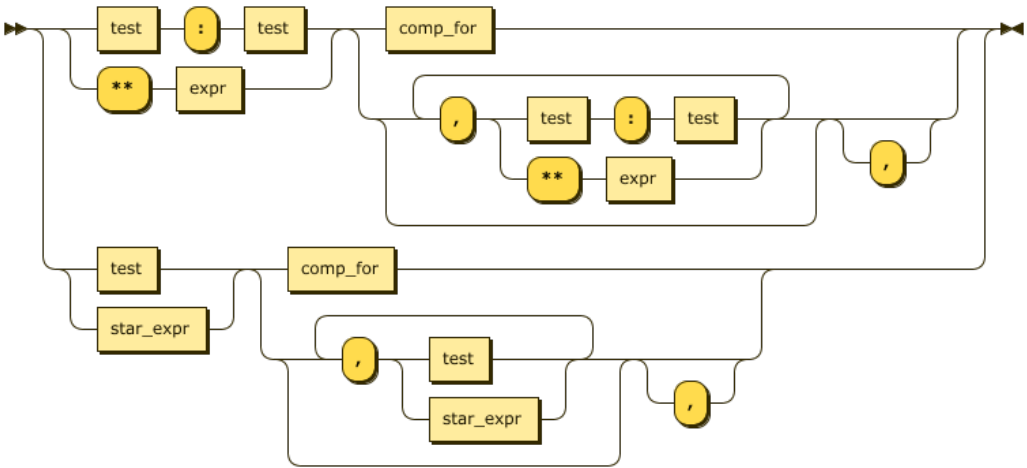
Example formal language

- Programming languages generally have very detailed grammar specifications.
- This makes them easy to parse and reduces the scope for ambiguity.
- Extended Backus-Naur Format is often used to express the grammar (syntax rules) of formal languages.
- Tools exist to generate a parser for a given grammar.
- The grammar (syntax rules) used to define a `dict` or `set` in python can be expressed as follows:

```
dictorsetmaker::= ( ((test ':' test | '**' expr)
       (comp_for | (',' (test ':' test | '**' expr))* (',')?)) |
       ((test | star_expr)
       (comp_for | (',' (test | star_expr))* (',')?)) )
```

Source:
https://discuss.python.org/t/railroad-diagrams-for-python-grammar/1017

Visualising a grammar using a railroad diagram



Source: `https://www.bottlecaps.de/rr/ui`

What would a grammar for a natural language like French look like?

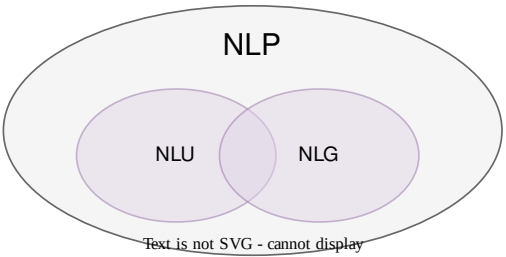
Comparing formal and natural languages

Criterion	Natural	Formal
Origination	Evolved	Designed
Objectives	Flexible, Expressive	Predictable, Unambiguous
Syntax	Loose, forgiving	Defined, rigid
Parsing	Difficult	Easier
Semantics	Context-sensitive	Explicit

- Formal languages need to do one job well: to capture knowledge in a way that is more easily interpreted by a computer
- Natural languages need much greater flexibility, from poetry, to textspeak, to political speeches, to academic writing

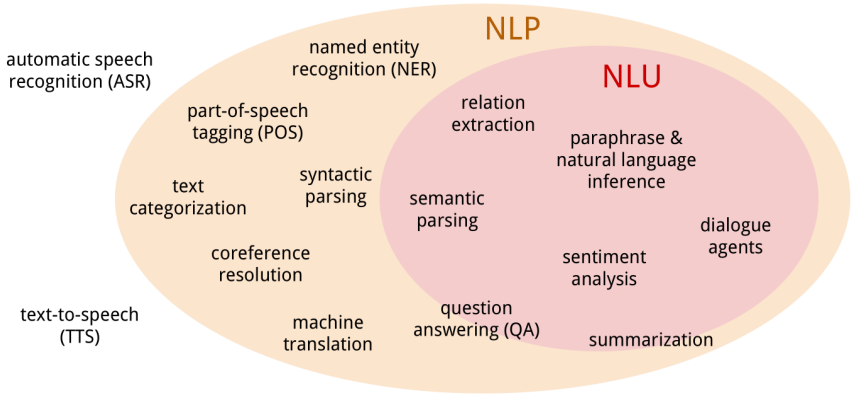
Natural language (unstructured text) is much more difficult for computers to handle!

Natural Language Processing, Understanding and Generation



- Natural Language Processing (NLP) includes all operations (including preprocessing) relating to text.
- Natural Language Understanding (NLU) derives *meaning* from unstructured text, for *computer* consumption.
- Natural Language Generation (NLG) generates text from internal computer models, for *human* consumption.
- The tools and techniques used by NLP are often categorised as *text mining*.

NLP, NLU and Speech



Source: https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf

- This diagram classifies selected tasks as NLP and/or NLU.
- By convention, speech tasks (recognition and generation) are not classed as NLP tasks.
- In today's talk, we do not cover speech tasks, just written text.

A text mining word cloud. . .

- The following is a word cloud of "text mining" terms used by presenters of the top 25 NLP lectures on the videolectures.net site.
- A word cloud is itself a popular NLP way to summarise text data. . .

Source: https://bit.ly/3vkP06n

Uses of NLP

- Natural Language interfaces to search engines, shopping sites, social media sites
- Language translation applications such as Google Translate
- Writing tools such as Microsoft Word and Grammarly that employ NLP to autocorrect spellings and/or to check grammatical accuracy of texts.
- Interactive Voice Response (IVR) applications used in call centers, or chatbots on support web pages, to respond to user requests.
- Personal assistant applications such as “OK Google”, Siri, Cortana, and Alexa that “converse with” humans
- Monitoring sentiment, trends and virality of posts on social media sites
- Information retrieval from document databases, CVs (resumés) and surveys (going beyond keyword search)
- “Voice of customer” analysis by email filtering, voice message analysis, etc.

Challenges for NLP interpretation

Challenge	Example
Words are misspelled	“belive”, “begining”
Words are inflected	“matrix” versus “matrices”
Tone is unclear	“The waiter was as friendly as all Parisian waiters.”
Varying context	“The food was good. However I cannot recommend…”
Idioms	“to take a rain check”; “to blow him away”
Jargon	“the device uses fractal resonance harmonics”*
Ambiguity	Several types: see next slide

*http://www.davidbarrow.com/psjg/

Examples of ambiguity

Type	Example	Analysis
Lexical	Don't get funny with me!	Funny: ha-ha or strange.
Syntactic	He hit the man with a bat	Who had the bat?
Referential	Mary called Jane because she knew.	Who knew?
Colloquial	It's grand.	It is big (or, in Ireland, barely adequate).

- Other homographs include "park", "tear", "wave", "fine", "lead", ...
- Some homographs (spelled the same) are also homonyms (sound the same), e.g., "bark"
- Syntactic and referential ambiguity are both examples of structural ambiguity

Translation failures

- A 1950s machine translation program translated “The spirit is willing but the flesh is weak” to Russian and back, resulting in “The vodka is good, but the meat is rotten” - Ooops!!
- On US TV, The Jimmy Fallon Show had a “Google Translates songs” segment with similar wacky translations for comic effect.
- US Immigration Service has used Google Translate to vet social media posts of incoming passengers, with embarrassing results
- Facebook Thailand sent a message on the King’s birthday that nearly saw it prosecuted according to *lèse majesté* laws there.
- A Tory MP used Google Translate for her Welsh language page, offering constituents (medical) surgeries and (botanical) plants to increase employment in her region

Of course, that is just a small sample!

Outline

1. Introduction 3

2. Natural Language 5

3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

6. Summary 61

NLP processes

Text Analytics

This focuses on producing *quantitative* results, e.g., how often a term appears in a corpus of documents, or the (statistical) distribution of bi-grams (two-word phrases) in a set of tweets.

Text Analysis

This *qualitative* procedure focuses on meaning, translation and interaction between humans and systems.

Text Analysis Techniques and Uses

- Text Classification
  - Sentiment analysis: Are customers happy with the service?
  - Topic detection: What are the latest news stories?
  - Intent detection: Are people supporting us or complaining?
- Text Extraction
  - Keyword extraction: What terms summarise recent news reports?
  - Entity recognition: Which persons or companies are trending?
- Word Frequency: Which words are most common for this author/genre?
- Collocation: Which phrases are significant for this domain?
- Concordance: How are keywords/phrases being used in context?
- Word Sense Disambiguation: How can we parse reliably?
- Clustering: What documents can be grouped by topic/author/genre?

The basics: NLTK vs spaCy

Advantages of NLTK

- NLTK is more mature (available since 2001), from academia (especially Stanford)
- Has more algorithms and tutorials/documentation
- Offers a toolbox approach so easily extended
- Generally uses less memory than spaCy for the same task

Advantages of spaCy

- spaCy is newer and returns objects rather than strings and arrays
- arguably has a more consistent API
- has fewer options but generally they are well-chosen, especially syntax analysis
- often runs faster than NLTK for the same task

We use NLTK in this module because it is easier to setup.

TextBlob provises a *pythonic* facade to NLTK - might be easier to use.

Preprocessing text

1. Load data: use standard python/pandas utilities or load document corpus prepared for NLTK
2. Tokenise to sentences or words: use NLTK's `sent_tokenize(...)` or `word_tokenize(...)`
3. Remove punctuation: use python's *regular expression* matcher
4. Standardise case: use Python's `.lower()` method on each word that was tokenized
5. Remove stop words: use list provided by NLTK and remove them with a python list comprehension
6. Stem: use NLTK's `PorterStemmer().stem(...)` applied using a python list comprehensions
7. Lemmatise: use NLTK's `WordNetLemmatizer().lemmatize(...)` applied using a python list comprehension

Stemming (faster) and lemmatisation (more reliable) are alternatives to each other

Terms in context - Concordance

Definition 1 (Concordance)

NLTK's `.concordance(keyword)` outputs a line for each instance of `keyword` in the document(s), surrounded by words before and after it. This shows how the author(s) use the given keyword(s) in context. If a keyword is potentially ambiguous, this provides a convenient way to check how it is being used in this case.

Concordance can be computed quickly and acts a quick check on the topics being considered. By providing the context, it is easier to test assumptions.

Term frequencies over time - Dispersion

Definition 2 (Dispersion)

If the text was collected over an extended period, the frequency of words might vary with time. In such cases, the placement of a word relative to the rest of the document is significant. A rugplot of a term's dispersion through the document can help to indicate whether a term is gaining or losing popularity. NLTK offers `.dispersion_plot(topicList)` for this purpose.

Dispersion is most often used to identify trending topics among social media posts.

Word frequencies relative to their peers

Definition 3 (Word Frequency)

If we ignore any temporal aspect, we can ask whether certain words appear more frequently than others. Indeed, a common "report" on text is to provide a list of words and their frequency of occurrence, sorted in decreasing order. The NLTK `.FreqDist(text).most_common(n)` function provides such a report, with only the top `n` words.

- Word frequency is the characteristic metric for a *bag of words model*.
- Word frequencies are helpful, but limiting to single words can make interpretation more difficult.

Adjacent Word frequencies

Definition 4 (Bi-gram (tri-gram) Frequency)

A bi-gram (tri-gram) is a pair (triple) of adjacent words occurring in the text. Often such adjacent word phrases (which are often *nouns* or *noun phrases*) carry more meaning than the individual words. The frequency distribution of such bi-grams (tri-grams) is often interesting and the top `n` bi-gram frequencies can be calculated by NLTK using `Counter(list(bigrams(text))).most_common(n)`.

Definition 5 (Bi-gram (tri-gram) Collocations)

Rather than considering just the frequency of bi-gram (tri-gram) occurrence, sometimes it is more informative to focus more on the word pairs (triples) that are more frequent than their individual words would suggest. Such pairs (triples) are called collocations and can be found by NLTK using `text.collocations()` for bi-gram collocations (with default settings) and `trigram_collocation = TrigramCollocationFinder.from_words(words); trigram_collocation.nbest(TrigramAssocMeasures.likelihood_ratio, n)` for tri-gram collocations, showing some of the optional settings.

Word frequency analysis

- Word, bi-gram and tri-gram frequencies in a single document or across a corpus of documents can reveal a lot of information
- In particular, such frequencies are indicative of the subject matter, and help to highlight *keywords*
- Anecdotally, the importance of a term is correlated with its frequency (other factors being equal).
- Thus it is a candidate feature of the document or corpus of such documents.
- However, this is not the whole story: stop words are very frequent (have a high *term frequency* score) but are not considered important.
- Perhaps a better feature exists...

Conditional word frequency analysis

- Term frequency has limitations when considering importance at a whole document or whole corpus level.
- Another consideration is how much focus there is on that term in
  - part of a document, say a chapter of a book, or a segment of a news articles, or
  - a document within a corpus, say a book within a series, or a single social media post.
- What is the frequency of a given term in other parts of a document, or documents in the corpus?
- If the term is limited to this text segment, that might make it more interesting
- Stop words are common across all document segments, this is one of the reasons we remove them!

The terms in the first article to cover a news story are more significant than articles without that story.

Term importance in context: its TF-IDF score

Definition 6 (Term Frequency (TF) score)

The term frequency (TF) measures the frequency of a term t in a document. It is often expressed as the ratio of the number of times the term occurs, divided by the number of such terms in the document.

Definition 7 (Inverse Document Frequency (IDF) score)

The inverse document frequency (IDF) is the ratio of the number of documents n in the corpus, divided by the number of documents containing the term t, which is df(t). The log is taken to prevent it growing too quickly with corpus size, so IDF = $\log(\frac{n}{\text{df}(t)})$. If no document in the corpus contains the term, the denominator would be zero, so 1 is added by convention: IDF = $\log(\frac{n}{1+\text{df}(t)})$. sklearn uses as "smoothed" version: IDF = $\log(\frac{1+n}{1+\text{df}(t)}) + 1$.

Definition 8 (TF-IDF score)

The TF-IDF score is the product of TF and IDF: TF $\times$ IDF.

TF and IDF measure term importance - their product is an even better feature for this purpose.

Uses of TF-IDF scores

Vectorisation

If every term in a corpus is assigned a TF-IDF score, the vector of such scores is unique to the document. This gives a way to compute the distance between documents, say by the cosine distance between their TF-IDF vectors.

Information retrieval

When searching for relevant documents associated with t, TF-IDF(t) can be used to rank the search results by relevance.

Keywords and Text summarisation

TF-IDF is efficient at finding key words. Adding context to terms with high TF-IDF means that text can be summarised.

TF-IDF for encoding words

Text needs to be *encoded* to be used in ML. TF-IDF is a frequency-based way to convert terms to numeric vectors.

Computing TF-IDF scores

- sklearn (not NLTK) offers TF-IDF calculation
- The calculation is computationally efficient, but ignores semantics, word order and context

Example code

```
# Initialize TfidfVectorizer with desired parameters (default
# smoothing and normalization)

tfidf_vectorizer = TfidfVectorizer(input='content', stop_words='english')

# Run TfidfVectorizer on the text in df.

tfidf_vector = tfidf_vectorizer.fit_transform(df["text"])

# Make a DataFrame out of the resulting tf-idf vector, setting the
# "feature names" or words as columns and the titles as rows

tfidf_df = pd.DataFrame(tfidf_vector.toarray(), index=Df['year_Name'],
  columns=tfidf_vectorizer.get_feature_names_out())
```

Word embedding

- One hot encoding of words is inefficient and does not capture semantic relationships
- TF-IDF scores convert words into numeric vectors based on relative frequencies
- Word2Vec (2013) is implemented in python using the gensim package and works well with NLTK
- GLoVE (2014) is an alternative, can work better
- Both use bags of words and derive context neighbourhoods using unsupervised learning
- Both Word2Vec and GLoVE embeddings have the nice property that similar concepts map to similar vectors

Outline

1. Introduction 3

2. Natural Language 5

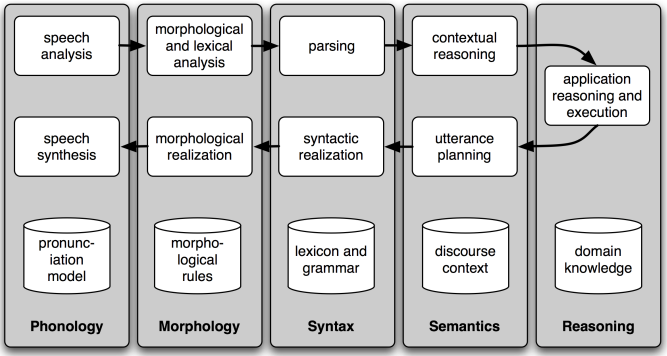
3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

6. Summary 61

NLP round trip

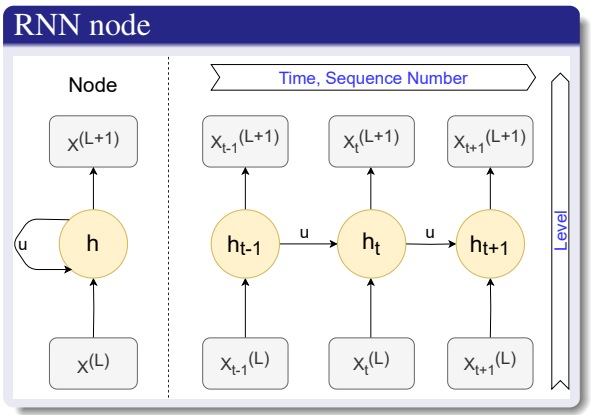


- The schematic above shows a conversational system enabled by NLP
- NLTK (and equivalent) help with the *Morphology* and *Syntax* phases
- These days, deep-learning based approaches show great promise for *Semantics* and *Reasoning*, and can help with all the other phases

circa 2018: DL and NLP come together

- Before 2018: State of the art NLU/NLG used RNN
- RNN: recurrent neural network: parse/build sentences additively “left to right”
- That and (relatively) limited training data...
- Resulting language models (that might express any valid form of text) were limited
  - Capability was good ($\sim$ bias was low): model fitted the training data well
  - Alignment was poor ($\sim$ variance was high): generated text was not consistent with intention
  - Result: well-structured but unreliable text output
  - Good enough for a chatbot in a restricted domain
- Enter GPT and BERT...

Recurrent Neural Networks and NLP



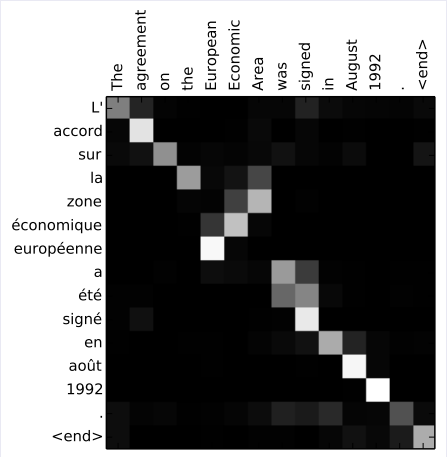
- Node has internal state (h_t, memory of sequence so far) and passes to itself in later instant
- h_t contributes to h_{t+1}
- Earlier words affect the (predictive) language model for words later in the sequence
- Language model is expressed in recurrent weights and biases of the neural network
- Nearby words tend to have the most effect

Recurrent architectures for NLP

- RNNs are simple to setup and relatively easily interpreted (as ANNs go!)
- The have two main problems
  1. Learning generates updates to weights and biases. These updates depend on the gradient of the loss function. But due to the recurrence on h_t, the gradients can *explode* or *vanish*. So learning can fail…
  2. Recurrence requires sequential learning (word by word) which is inefficient on multicore CPUs and especially on GPUs
- One solution to the first problem is to control the gradients by allowing to forget or emphasise
- Long Short Term Memory (LSTM) is an RNN variant where (input, output, forget) gates are added to each node
- Gives greater control over what previous data is remembered and what is forgotten for each input
- Gated Recurrent Units (GRU) is a simpler version of LSTM (lacks the output gates)
- Neither LSTM nor GRU addresses the second problem

Encoder-Decoder architectures for NLP

Machine translation



- Many NLP tasks are many (tokens) $\rightarrow$ many (tokens)
- Rather than sequential recurrence, a more scalable approach might be to
  1. *Encode* the input to an internal representation.
  2. *Decode* this representation to meaningful output.
- Transformers use this Encoder-Decoder principle.

GPT: Generative Pre-trained Transformer

- Latest is GPT-4 (paid ChatGPT only) and GPT-3.5
- Launched by OpenAI (founded by Elon Musk and others)
- Overview at GPT-3 demo and
- Trained on ever larger data sets - now based on billions of articles, tweets, etc
- GPT-3 claimed to use 175 billion(!) parameters; GPT-4 uses more
- Autoregressive, so unidirectional (lookback window size has been challenged)
- Typical use case: Give GPT-3.5 the starting point, it generates the rest
- Does not consider meaning, so can provide plausible but nonsense answers to questions: hallucination
- “Write a story” vs “Write an award-winning story” - latter filters the input better
- Not open source - access via OpenAI API (SaaS / PaaS)
- API is simple: little tuning needed - “text in, text out”

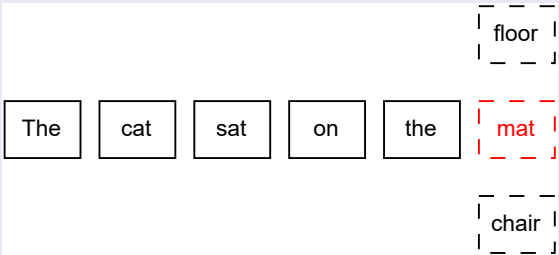
BERT, the contender from Google

Also see ELMO and ERNIE...

- BERT: Bidirectional Encoder Representations from Transformers
- Released as open source by Google
- Requires TPUs and GPUs - training time of the order of hours
- Considers latent relationships so is more sophisticated than GPT-3.5 (GPT-4?)
- Uses fewer parameters (less than 1 billion)
- More tuning required
- Fewer built-in models, so more effort for user
- Rise of companies like HuggingFace and MonkeyLearn with pre-built models

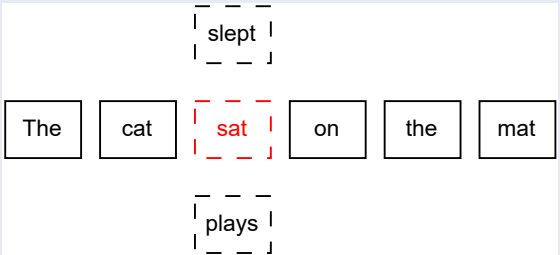
But first—Text Generation

Causal Language Modelling



- Given "The cat sat on the" what should the *next* word be?

Masked Language Modelling



- Given "The cat MASK on the mat" what should the MASKed word be?

- Generally: choose the word (token) with the highest probability from the training set.
- Use of *temperature* parameter to choose lower probability alternatives *sometimes*
- Analogous to accuracy vs diversity in recommender systems

Text Generation - from the ground up

Character-oriented

1. Add alphabetic characters, assuming they are uniformly distributed (e.g., Z is as common as A)
2. Now assume characters have a distribution based on the (target) language.
3. Include whitespace characters so the resulting word lengths are distributed per the language

Word-oriented

4. Generate words by sampling from a dictionary based on their frequency in the language.
5. Aside: such nonsense text is recommended for long but memorable passwords...
6. Given the previous word, pick the next word based on the frequency of word bi-grams in the language
7. Repeat with word trigrams, etc

Looking back gives better *context*, but how much is needed? And what about sentences, paragraphs...?

Enter. . . Large Language models

Step 1: Pre-training - Once-off

- Use unsupervised learning on unlabeled data to derive latent relationships between words

Step 2: Fine Tuning - Ongoing

- Use *self-supervised learning* using inferences from the pre-trained *relationship model* to improve the predictive ability of the *generative model*.
- Can distinguish between *zero-shot models* (general purpose, like ChatGPT) and domain-specific, like OpenAI Codex.

Step 3: Generation

- Given a prompt, or text to complete, or text to translate, use the *generative model* to generate new text

The generated text is not simply extracted from the training corpus, but is motivated by it!

Transformers - the new architecture for LLM and NLP

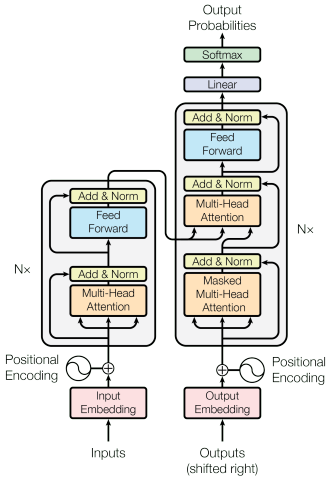
- State of the art for NLU and NLG
- Innovations include
- Positional encoding: ["hello", "world"] becomes [("hello", 1), ("world", 2)]
- Attention: Word combinations map to other Word combinations - easiest seen with translation
- Self-attention: Use attention mechanism in a “round trip” to resolve ambiguities
- Transformer concept was introduced by Google researchers in a *heavily cited* 2017 paper Attention Is All You Need.

Deep learning is used throughout, for both the relationship and generative models

Transformer architecture: Multiple Encoder-Decoders + Attention

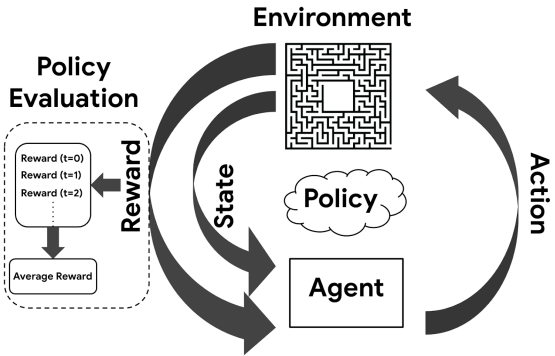
Transformer Architecture

- Note the use of Encoders and Decoders
- Also note the targeted use of recurrence within them
- Architecture is complex, but it is explained well by vcubingx and 3blue1brown on youtube



Source: Attention Is All You Need

Sidebar: Reinforcement Learning - learning from rewards

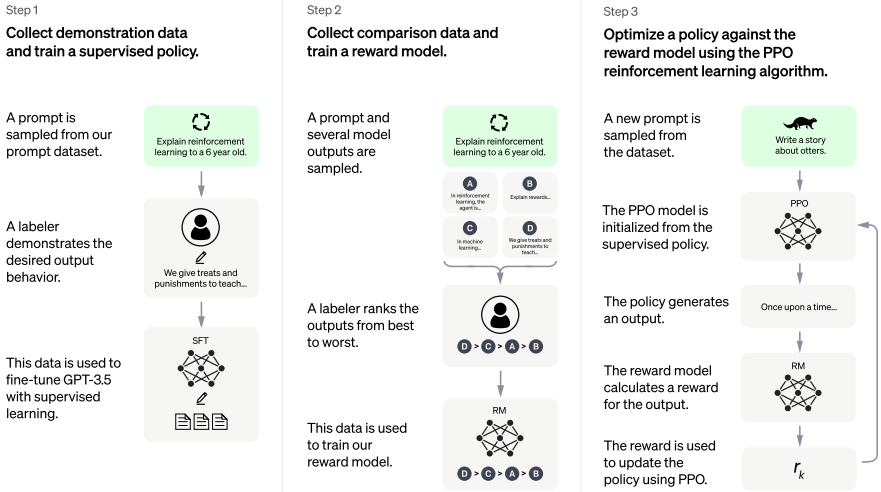


Source: https://bit.ly/3Ld8CzT

Overview - Mazes

- Agent (mouse) wishes to find cheese hidden in a maze (*goal*).
- Has knowledge of state (local *environment*).
- Makes navigation decisions based on heuristics encoded as *policies*.
- Each decision (*action*) has an associated *reward*.
- Aim is to maximise rewards and achieve the goal.

ChatGPT - Overview of its LLM and training



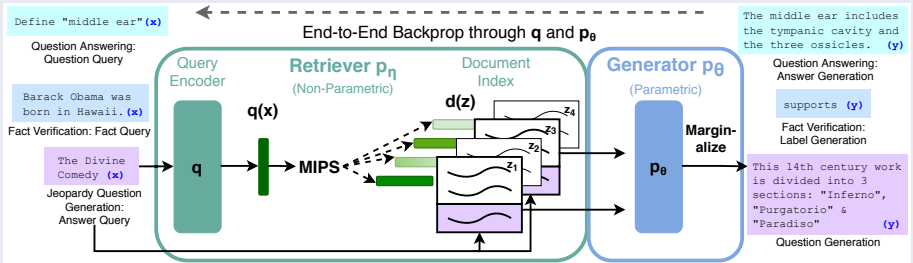
Source: https://bit.ly/3NlAche

ChatGPT challenges - Introducing RAG

ChatGPT faces 3 challenges that affect the quality of its generated text:

1. Limited training data, due to cutoff dates and blocking by content owners
2. Lack of subject-matter expertise and understanding, e.g., to perform calculations
3. Lack of traceability of generated text - no citations or other means of fact checking

Retrieval Augmented Generation (RAG) - from Meta Research (2021)



Source: Lewis et al. (2021) – Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

GPT and RAG can be used via langchain and similar tools.

Summary

- Text mining is closely linked to AI - think Turing's test for AI
- Computers have been working on languages and text for decades
- With Big Data and tools like NLTK, *discriminative* tasks like text and sentiment analysis have become mainstream
- Recent developments in LLMs mean that the next wave of AI will focus on *generative* tasks like conversational AI
- Many challenges to overcome: bias, data ownership, alignment issues, concentration of power, deep fakes, ...
- In the lab: text processing with NLTK, transformers and pytorch

Outline

1. Introduction 3

2. Natural Language 5

3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

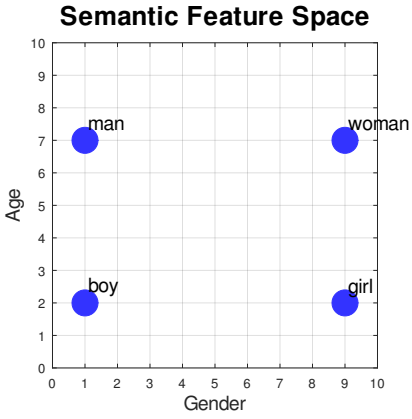
6. Summary 61

Dummy encoding

- Given `Woman`, `Man`, `Girl`, `Boy`
- Can dummy encode them, as `1000`, `0100`, `0010`, `0001`
- But this is
  1. Inefficient: length of code = number of words in the vocabulary
  2. No semantics: Impossible to determine whether word A is more similar to word B than it is to word C
- We can do better!

Descriptive Variable encoding

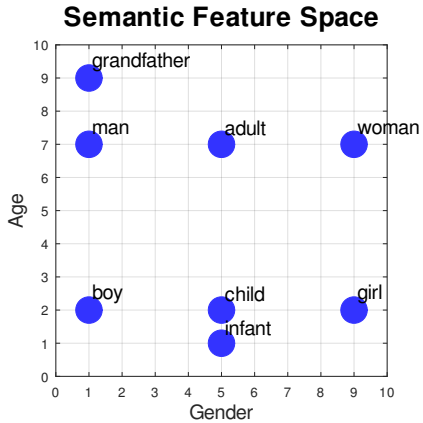
- Given `Woman`, `Man`, `Girl`, `Boy`, as before
- Two descriptive variables might be `Gender` and `Age`
- Each of the 4 words now has an “Age”-score and a “Gender”-score
- Hence, it can be represented with 2 floating point numbers, not 4 integers
- We can also define semantic distances, quantifying the similarity between words
- For example, `Woman` is more like `Girl` than `Boy`



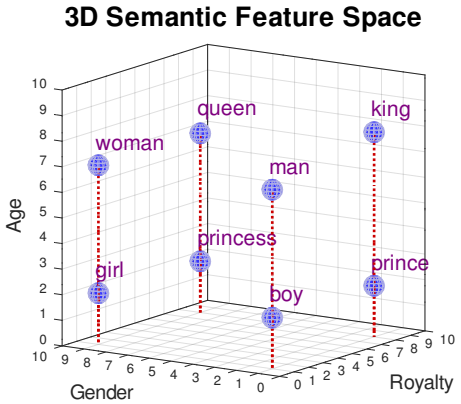
Acknowledgements

Much of this discussion was motivated by Word Embedding Demo: Tutorial and The Illustrated Word2vec.

Descriptive Variable encoding - Extended



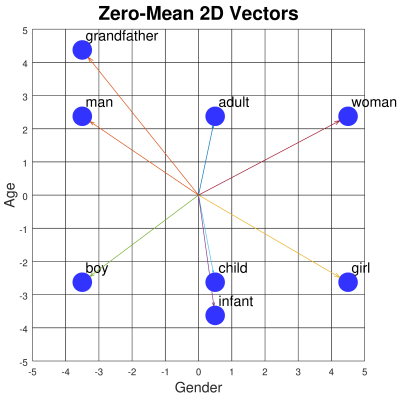
Adding more people, by Gender and Age



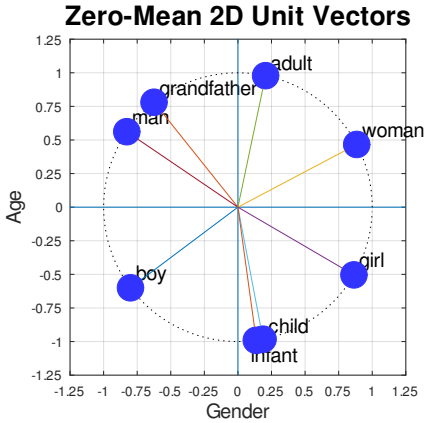
Adding *Royalty* variable, for a 3D Semantic space

Words as Vectors

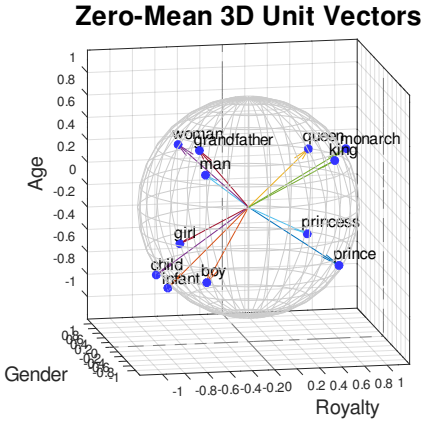
- With a restricted vocabulary of related words, a small number of descriptive features is enough
- It is convenient to think of words as vectors in some low-dimensional semantic space
- Distances can be computed using Euclidean, Manhattan or other norms



Scaling Word Vectors to unit length



2-D normalised word vectors: distance $\propto$ angle



3-D normalised word vectors: distance $\propto$ solid angle

Removed dependence on scaling, so cosine distance is useful

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!
- Need a richer set of dimensions - but how can we choose them?

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!
- Need a richer set of dimensions - but how can we choose them?
- And how can we map words to such vectors?

Generalising to larger text corpora

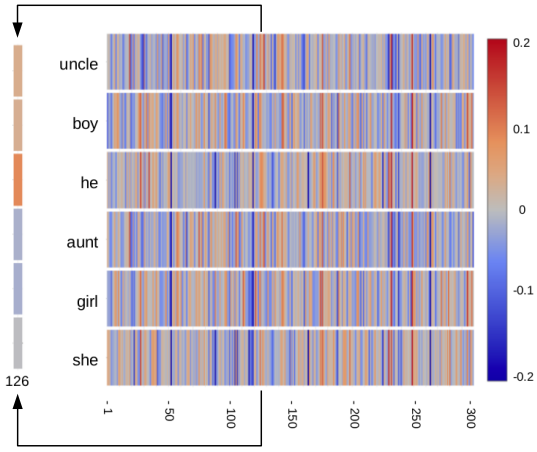
- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!
- Need a richer set of dimensions - but how can we choose them?
- And how can we map words to such vectors?
- This is where word embedding algorithms like `Word2Vec` and `GLoVE` come in...

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!
- Need a richer set of dimensions - but how can we choose them?
- And how can we map words to such vectors?
- This is where word embedding algorithms like `Word2Vec` and `GLoVE` come in...
- Key idea: words that occur together (as n-grams) carry meaning

Generalising to larger text corpora

- In a corpus such as the works of Shakespeare, or Wikipedia, the vocabulary is large and varied
- A small number of dimensions is not enough!!
- Need a richer set of dimensions - but how can we choose them?
- And how can we map words to such vectors?
- This is where word embedding algorithms like `Word2Vec` and `GLoVE` come in...
- Key idea: words that occur together (as n-grams) carry meaning



A subset of word vectors trained using wikipedia data

Skip-Gram with Negative Sampling (SGNS)

- `Word2Vec` comes in many variants, we consider one variant here
- Key considerations include:
  - Require a large corpus to train from; *word vectors are specific to that corpus*

Skip-Gram with Negative Sampling (SGNS)

- `Word2Vec` comes in many variants, we consider one variant here
- Key considerations include:
  - Require a large corpus to train from; *word vectors are specific to that corpus*
  - Need to decide on the number of words in the Vocabulary, say top-M by occurrence

Skip-Gram with Negative Sampling (SGNS)

- `Word2Vec` comes in many variants, we consider one variant here
- Key considerations include:
  - Require a large corpus to train from; *word vectors are specific to that corpus*
  - Need to decide on the number of words in the Vocabulary, say top-M by occurrence
  - What to do with proper nouns, stop words, roots, etc.?

Skip-Gram with Negative Sampling (SGNS)

- `Word2Vec` comes in many variants, we consider one variant here
- Key considerations include:
  - Require a large corpus to train from; *word vectors are specific to that corpus*
  - Need to decide on the number of words in the Vocabulary, say top-M by occurrence
  - What to do with proper nouns, stop words, roots, etc.?
  - Need to decide on the number of dimensions N in the word vectors (typically 300+)

Skip-Gram with Negative Sampling (SGNS)

- `Word2Vec` comes in many variants, we consider one variant here
- Key considerations include:
  - Require a large corpus to train from; *word vectors are specific to that corpus*
  - Need to decide on the number of words in the Vocabulary, say top-M by occurrence
  - What to do with proper nouns, stop words, roots, etc.?
  - Need to decide on the number of dimensions N in the word vectors (typically 300+)
  - Need to choose context window size C, i.e., C words before, and C words after the word to vectorise

SGNS Preparation

Step 1: Setup

1. Create an $M \times N$ tables `E` and `U`, where each row represents a vocabulary word and each column is a dimension used to represent a latent concept
2. Initialise each table with random numbers near 0

Note that

- `E` will contain vector embeddings of the words in the Vocabulary
- `U` is for the Vocabulary words that are used in context (and is not needed afterwards)

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus
- Look up its vector $\mathbf{e}_i$ in embedding table $\mathtt{E}$

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each Vocabulary context word, look up its vector $\mathbf{u}_j$ in context table `U`

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each Vocabulary context word, look up its vector $\mathbf{u}_j$ in context table `U`
- Compute the dot product $x_{ij} = \mathbf{e}_i \cdot \mathbf{u}_j$, which is just a scalar number.

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each Vocabulary context word, look up its vector $\mathbf{u}_j$ in context table `U`
- Compute the dot product $x_{ij} = \mathbf{e}_i \cdot \mathbf{u}_j$, which is just a scalar number.
- Use a sigmoid or logistic function to map x_{ij} so that $|\tilde{x}_{i,j}|$ lies within $(0, 1)$.

SGNS Training: Positive Cases

Step 2: Words within Context

- Start with the Vocabulary word at position $C + 1$ in the corpus
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each Vocabulary context word, look up its vector $\mathbf{u}_j$ in context table `U`
- Compute the dot product $x_{ij} = \mathbf{e}_i \cdot \mathbf{u}_j$, which is just a scalar number.
- Use a sigmoid or logistic function to map x_{ij} so that $|\tilde{x}_{i,j}|$ lies within $(0, 1)$.
- If $\tilde{x}_{i,j} < 0$ (*negative*), adjust $\mathbf{e}_i$ and $\mathbf{u}_j$ slightly (*gradient descent*)

SGNS Training: Negative Cases

Step 3: Words outside Context

- For each Vocabulary word, find at least 5 words that *never* appear in its $2C + 1$ context

SGNS Training: Negative Cases

Step 3: Words outside Context

- For each Vocabulary word, find at least 5 words that *never* appear in its $2C + 1$ context
- Look up its vector $\mathbf{e}_i$ in embedding table $\mathtt{E}$

SGNS Training: Negative Cases

Step 3: Words outside Context

- For each Vocabulary word, find at least 5 words that *never* appear in its $2C + 1$ context
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each of its 5+ non-context words, look up its vector $\mathbf{u}_j$ in context table `U`

SGNS Training: Negative Cases

Step 3: Words outside Context

- For each Vocabulary word, find at least 5 words that *never* appear in its $2C + 1$ context
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each of its 5+ non-context words, look up its vector $\mathbf{u}_j$ in context table `U`
- Use the same squashed dot product calcualtion as was used for the positive cases

SGNS Training: Negative Cases

Step 3: Words outside Context

- For each Vocabulary word, find at least 5 words that *never* appear in its $2C + 1$ context
- Look up its vector $\mathbf{e}_i$ in embedding table `E`
- For each of its 5+ non-context words, look up its vector $\mathbf{u}_j$ in context table `U`
- Use the same squashed dot product calcualtion as was used for the positive cases
- If $\tilde{x}_{i,j} > 0$ (*positive*), adjust $\mathbf{e}_i$ and $\mathbf{u}_j$ slightly (*gradient descent*)

SGNS Training: Overall

Step 4: Iteration to Convergence

- Perform Step 2 then Step 3

The resulting `E` can be used as input to a Large Language Model, which can be used to translate, summarise, or write text.

SGNS Training: Overall

Step 4: Iteration to Convergence

- Perform Step 2 then Step 3
- Check that `E` has *converged*

The resulting `E` can be used as input to a Large Language Model, which can be used to translate, summarise, or write text.

SGNS Training: Overall

Step 4: Iteration to Convergence

- Perform Step 2 then Step 3
- Check that `E` has *converged*
- Stop if converged, or if the number of iterations exceeds 50, say.

The resulting `E` can be used as input to a Large Language Model, which can be used to translate, summarise, or write text.

Outline

1. Introduction 3

2. Natural Language 5

3. NLP Tools and Techniques 17

4. Adding understanding and generation 32

5. Encoding words 49

6. Summary 61

Summary

- NLTK and friends can be used to perform basic text preprocessing
- Selected vocabulary words are then embedded as vectors: (word-level) knowledge
- Text naturally comprises varying lengths of token sequences
- So it is convenient to use a *transformer*-based deep neural network to represent the corpus-leve knowledge
- The model can then be used as a basis for tasks such as
  - translation
  - classification (e.g., sentiment analysis)
  - generation of new text (use the concepts and the style of the corpus the model was trained on)
- Note that prompts are needed to start the process of generating new text

Text mining/generation is a very active area, so this talk can only scratch the surface!