NUS Fintech Society

Machine Learning

Natural Language Processing

Content

- Intro to Natural Language Processing
- Building Machine X
- Web Scraping
- A Better Machine X

Intro to Natural Language Processing

Problem Statement

- Wikipedia:
  Natural language processing (**NLP**) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

- Essentially, it's about making computers understand human language

- We've learned methods to make predictions given features
  - Independent variables as features, outcomes as dependent variables
  - For tabular data, features are a given

- Problem:
  - Not all problems have *obvious* features
  - Not all problems have *usable* features

Case Study: Image Processing

- Attempt 1:
  - Features: edges e.g. vertical lines, horizontal lines, diagonal lines etc.
  - Outcome: 0 or 1
  - Model: logistic regression, support vector machines (SVM) etc.
- Attempt 2:
  - Features: pixels [0, 255]
  - Outcome: 0 or 1
  - Model: neural network

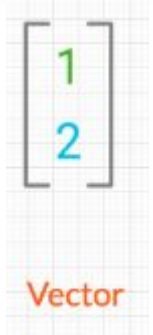
Natural Language Processing

Natural language processing (**NLP**) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

- Attempt 1:
  - Features: words (?)
  - Outcome: a model that understands words (??)
  - Model: ???

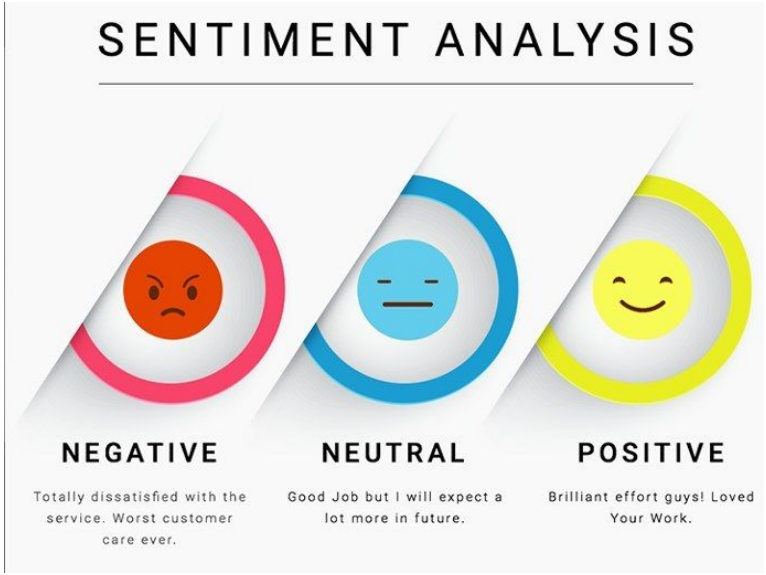
NLP

Natural language processing (**NLP**) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.



- Assumptions:
  - We have a machine X that, given words, produce features (*vectors)*
  - We want to make a machine Y that can do *sentiment analysis*

Case Study: Sentiment Analysis



1. Label each sentences as negative (-1), neutral (0), or positive (+1)
2. Pass through all the sentences through machine X to generate *set of features*
3. Train any model you can think of to predict *sentiment* from *features*

NLP Task A

- General Approach to NLP:
  a. Label each sentences
  b. Pass through all the sentences through machine X to generate *set of features*
  c. Train any model you can think of to predict outcome for A from *features*

  Note: replace underlined words with the appropriate context

Note:

- In ML, task A is is called *downstream task.*
- The *main / general task* is the task used to train machine X
- Here machine X is not trained again. This is called *using pre-trained models*

Possible Improvements

- General Approach to NLP:
  a. Label each sentences
  b. Pass through all the sentences through machine X to generate *set of features*
  c. Train any model you can think of to predict outcome for A from *features*

  Note: replace underlined words with the appropriate context

- Observation:
  - Machine X generate features that may or may not be useful for task A
    - Let backpropagation modify the model AND machine x together!
    - This process of (minimally) modifying machine X is called *fine-tuning*
    - In practice this means allowing for gradient updates to reach machine X

Problems

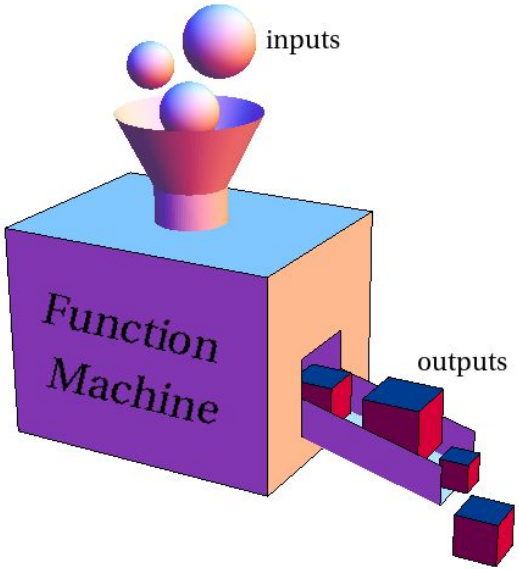
- Unreasonable assumption: How do we build X?
- High variance: the sentence “Today’s Fred’s birthday” can have multiple forms
- Low entropy:
  - Each character in a sentence doesn’t convey a lot of information
  - Each word in a sentence doesn’t convey a lot of information

The dog _____ small.
_____ cat is eating food.
I see a _____ girl.
She _____ play with me.
I _____ the blue sky.
The cat _____
dog are running.
Can I _____ it?
We go _____ the store.
I _____ happy.

Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Building Machine X

Recap: Machine X



- We want to *transform* words into vectors
  - Input: words
  - Output: vectors
- Word Embeddings
- Vector Representations

Preprocessing

- Sentence: “Today’s Fred’s buffday!”
- Method:
    1. Tokenisation: Break sentence into *tokens*
        a. “Today’s Fred’s buffday!” => [“Today’s”, “Fred’s”, “buffday”]
        b. Can be done with “.split()” in Python
    2. Normalisation: standardise words
        a. “buffday” => “birthday”
        b. More complicated
    3. Stemming and Lemmatisation:
        a. “Today’s” => “today”
        b. “Fred’s” => “fred”

Note: stemming is simple rules-based cutting, while lemmatisation uses language rules to obtain root word

Building Machine X

- What we've done so far:

  Today's Fred's buffday => ["today", "fred", "birthday"]

- Attempt 1:
  - "birthday" => 1
  - "fred" => 2
  - "today" => 3
  - Hence, "Today's Fred's buffday" => [3, 2, 1]

- Equivalently,
  - "birthday" => [1, 0, 0]
  - "fred" => [0, 1, 0]
  - "today" => [0, 0, 1]
  - Hence, "Today's Fred's buffday" => [[0, 0, 1], [0, 1, 0], [1, 0, 0]]

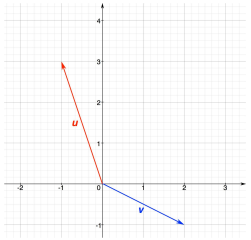
Getting Greedy….

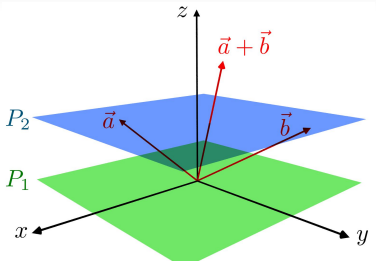
“Today’s Fred’s buffday” => [[0, 0, 1], [0, 1, 0], [1, 0, 0]]

- Adding more words:
  - “Today’s Fred’s buffday” => [[0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]]
  - Getting unwieldy with more words in our dictionary…
- Vectors don’t have semantic meaning:
  - “Today’s NOT Fred’s buffday” => [[0, 0, 1, 0], [0, 0, 0, 1], [0, 1, 0, 0], [1, 0, 0, 0]]
  - From the vectors alone, we cannot see that this sentence is the *opposite* of our original sentence

Ideally…

- Fixed embedding space:
  - Represent the entire language in $\mathbb{R}^d$
  - Vectors are of size d

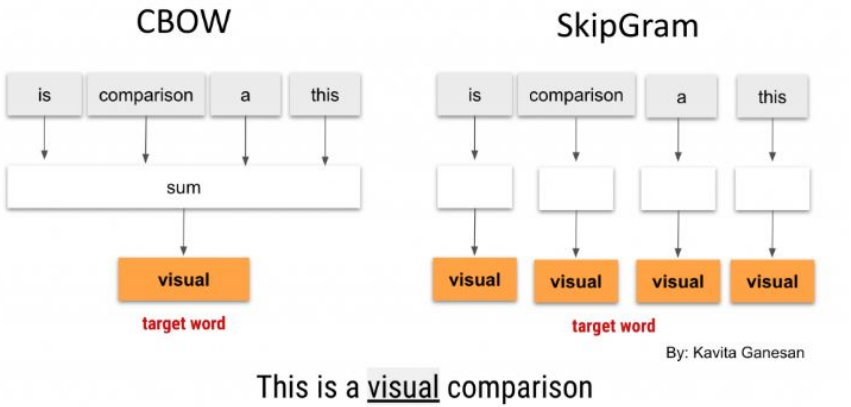




- Vector Operations
  - We want to have vectors that *make sense* to us
  - We can do vector operations on the *meaning of words:*
    - Vector additions:
      - “King” - “Man” + “Woman” = “Queen”
      - “School” + “Boss” = “Principal”
    - Scalar Multiplication:
      - 10 x “Sad” = “Very Sad”
      - -1 x “Sad” = “Happy”
    - Cosine Similarity
      - “happy” == “merry”

Approach: Word2Vec

- Unsupervised learning: a fill-in-the-blank approach to learning language
- Two approaches: CBOW and Skip-Gram
  - CBOW: Given all the N number of words around, predict the blank
  - SkipGram: Given a word in the sentence, predict the blank



Miscellaneous

- Other possible approach to Word2Vec?
  - GloVe, FastText
  - Cool model of the day: BERT, RoBERTa, etc.

- How does Word2Vec relate with Tokenisation, Normalisation, Lemmatisation?
  - Tokenisation, normalisation, and lemmatisation are *preprocessing* steps
  - They can be used before being passed into Word2Vec for training
  - They can also not be used i.e. all words are treated as individual tokens

- Python Libraries?
  - NLTK, Gensim, Hugging Face, and more

- Why do I need to know all these?
  - It’s important to know the source of the data for which the model was trained on
  - NLP on lawyers’ notes won’t benefit much from using model trained on Twitter

Web Scraping

Intro to Web Scraping

- Obtain data from the Internet
- There are *techniques* to do this:
  - Human copy-and-paste
  - Text pattern matching
  - HTML parsing
  - DOM parsing
  - Computer vision web-page analysis
  - And more…

- Most importantly:
  - Human + Googling e.g. Credit Risk Dataset
  - HTML parsing e.g. Cryptocurrency News

VERY brief introduction to HTML

- HTML “Code” looks like this:

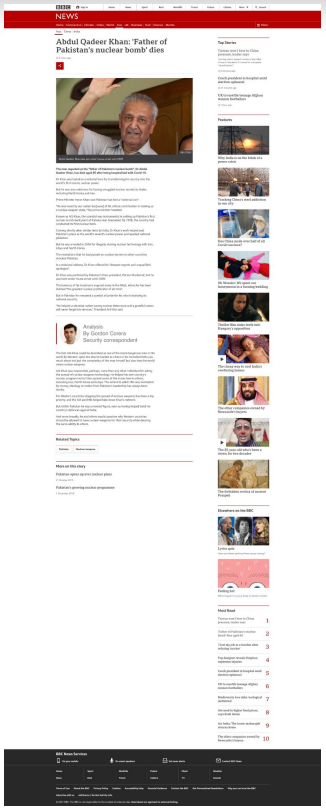
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

VERY brief introduction to HTML

- REAL Website

Abdul Qadeer Khan: 'Father of Pakistan's nuclear bomb' dies

```
<article class="ssrcss-1mc1y2-ArticleWrapper e1nh2i2l6">
  <header class="ssrcss-18mgjmu-HeadingWrapper e1nh2i2l5">
    <h1 tabindex="-1" id="main-heading" class="ssrcss-gcq6xq
    -StyledHeading e1fj1fc10">Abdul Qadeer Khan: 'Father of
    Pakistan's nuclear bomb' dies</h1> == $0
    <div>…</div>
```



Idea: Get new headlines by searching for tags with id "main-heading"

Intro to Web Scraping - HTML Parsing

1. Play with “target” website to identify regularity in patterns

2. Write code that does the following:
   - Access website (selenium)
   - Go to a page in the website:
     - Typically need to *search, click buttons etc.*
     - Access *elements* of the website by id (selenium)
   - Obtain HTML of page (beautifulsoup)
   - Extract information from HTML (beautifulsoup)
   - Repeat

3. Replace points 3 and 4 to apply other techniques of web scraping

A Better Machine X

Recipe to make Machine X

1. Define what do we want to act as *tokens* e.g. words, n-grams, etc.
2. Define *model architecture* of machine X
3. Define *pre-training tasks* e.g. fill-in-the-blank
4. Train machine X

- How to improve performance:
  - Find more useful tokens e.g. n-grams instead of words
  - Find a better model architecture
  - Find a more informative pre-training tasks

More Useful Tokens - FastText

- For languages like German with many composite words:
  - cup => *Tasse*, coffee cup => *Kaffeetasse*

- Solution: n-grams
  - For n = 2, the n-grams representation of the word “this” is:

```
<t
th
hi
is
s>
this
```

  - Observe that this would cause “Tasse” and “Kaffeetasse” to have overlapping n-grams

Model Architecture - Word2Vec

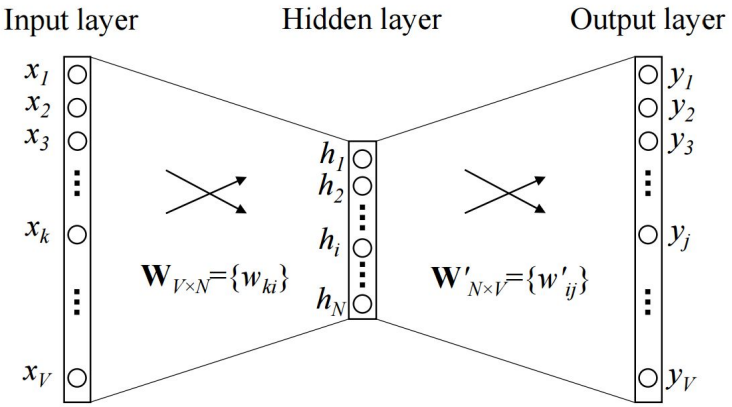
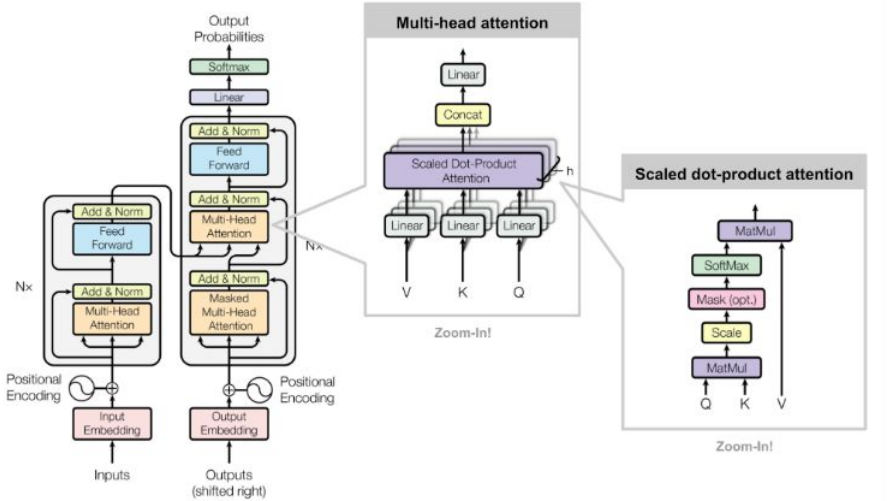


Figure 1: A simple CBOW model with only one word in the context

A Word2Vec model trained using CBOW approach. Note the absence of *non-linear activation functions*

Model Architecture - Transformers



A BERT model. Note the *softmax* activation function i.e. non-linear model

- Improvements from Word2Vec:
  - Attention Mechanism: model learns appropriate *context window size*
  - Non linearity in the model

Pre-Training Tasks

- *Traditional* Language Modelling:
  - Sliding window approach from left to right
  - Used in the GPT series by OpenAI
- Masked Language Modelling (MLM):
  - Essentially a fill-in-the-blank approach of Word2Vec
  - Put more blanks in the sentence + Randomly word replacement
  - Used in BERT series by Google
- Next Sentence Prediction:
  - Given sentence A and sentence B, predict if sentence B comes after A or not
  - Binary classification problem
  - Used in BERT series by Google
- Replaced Token Detection (RTD):
  - Replace random words, then predict which words have been replaced
  - Used in ELECTRA series by Google

Summary

Summary

- Intro to Natural Language Processing
  - Searching for a Machine X (Finding a pre-trained model)
  - Solving Downstream tasks using pre-trained model
- Building Machine X
  - Pre-processing of sentences
  - Obtaining vector embeddings with *unsupervised* learning
  - Case Study: Word2Vec
- Web Scraping
- A Better Machine X:
  - More useful tokens
  - Model Architectures
  - Pre-training tasks

What Now?

- Our discussion is *by no means exhaustive:*
  - NLP is a very large field currently in active development
  - Modern architectures we have not discussed:
    - GPT series: GPT-1, GPT-2, GPT-3
    - BERT series: RoBERTa, ALBERT, XLNet, etc.

- What is Attention Mechanism?
  - Attention is a vector representing *weights* e.g. how much to pay attention
    - General attention e.g. Additive Att., Multiplicative Att.
    - Self-attention e.g. QKV attention
  - Time complexity of QKV attention => O(n^2)
    - Reducing time complexity: Reformer, Linformer etc.

What Now?

- We use NLP as a *case study*; many of the points are relevant to other fields

- Model Architecture Standpoint:
  - Any (pre-trained) model that takes in input and produce vectors is good
  - Non-linearity is good (recall Rama's point last week)
  - Attention mechanism is good

- Training Perspective:
  - Pre-training + Fine-tuning is very common
  - Unsupervised / Self-supervised learning is very common with big data

Release of Project 2

- Problem Statement: Find and solve your own problem!

- What we look out for - 1:
  - How interesting it is:
    - # of people who have done the same problem before
    - Don’t do IMDB, CIFAR 10, etc. pls.. (Unless they’re interesting)
    - Bonus points for being FinTech related
    - Identify the *uniqueness* of your problem
    - Identify your *creativity* in solving the problem

  - How far have you explored?
    - Did you research your problem?
    - Did you go beyond what we covered?
    - # of models tried

Release of Project 2

- What we look out for - 2:
  - Good practices:
    - Documentation / Explanation of what you've done
    - Code organisation

- Requirements:
  - Must use machine learning one way or another
  - Don't have to focus on model building i.e. can be model deployment also (they're not easy :))
  - Showcase your project!
    - Upload *everything* onto your *public* github repo
    - For datasets / anything too big for github, put in google drive and share the link

Problem Ideas

- Stock price prediction using news data:
  - Given today's news => tomorrow's price
  - Given today's news + today's price => tomorrow's price

- Cryptocurrency Risk Assessment:
  - Given a cryptocurrency, predict how likely is it to be a scam / hacked / any other negative things

- Idea Generator: pick an ML model / technique you want to explore and find a use-case where the model may work:
  - GANs
  - Data Augmentation
  - Contrastive Learning
  - Etc.

Conclusion

- Today's the end of our "lecture" series

- Hope everyone had fun in the process and learned something!
  - We don't show *how* things are done e.g. what code to write etc.
  - The specific *Hows* can be Googled + YouTubed
  - Hopefully we've given you the *Whys* in ML
  - With the Whys in mind, you can do more creative and useful things!
    - Project 1 sets a specific constraint i.e. problem statement, datasets
      - **Analyst** will have to do this
    - Project 2 sets an open problem
      - **Analyst** will have to do this
      - **Project lead** will have to do this
    - Both projects encourage creativity why mix-and-match of techniques introduced in the workshops

- Thank you very much for being an amazing audience!

Thank You!