STA 414/2104:
Statistical Methods for Machine Learning II

Week 9 Natural Language Processing

Michal Malyska

University of Toronto

Overview

- Natural Language Processing
- Embeddings
  - Bag of Words
  - Term Frequency - Inverse Document Frequency
  - Word2Vec
- NLP concepts
- NLP tasks
  - Document Classification
  - Token Classification
  - Natural Language Generation

Machine Learning on Text

So far in the class (and probably all other STA classes) we have been operating under the assumptions that all the inputs to our models are numbers. Today we will focus on another kind of input - text. In particular, Natural Language.

- How is natural language represented?

NLP concepts

Some names we will be using throughout this lecture:

- **Token** - A single "atom" of text, usually a word
- **Document** - A complete datapoint of text
- **Span** - A subset of a document, a group of Tokens
- **Vocabulary** - A list of all possible tokens

NLP Tasks

What do we want to do with text?

- Classification (Documents, Spans, Tokens)
  - Hate speech detection
  - Spam filtering
  - Social Media drug adverse effect identification

NLP Tasks

What do we want to do with text?

- Generation (Question Answering, Summarization, Free-text generation, . . .)
  - Translating natural language into SQL queries
  - "Hey siri what is the weather like?"
  - Chatbots
  - Talk to a transformer

NLP Tasks

What do we want to do with text?

- Regression (Essay scoring, like count prediction)
  - How many retweets will this tweet get?
- Information Extraction
  - Who are the people mentioned in this text?
- Document Retrieval
  - Google search
  - Automatic literature review

Spam

We begin by revisiting a familiar example from earlier lectures: determining whether an email is spam or not.

- What kind of task is it ?
- How would you approach it?

More formally: Consider a set of observations $(x_i, y_i)_{i=1:N}$ where $y_i = 1$ means datapoint i was spam, and x_i is the text of the email.
Our goal is to create / learn a function f_θ such that $f_\theta(x_i) = p(y_i|x_i)$

Heuristics

Perhaps it is possible to find some heuristics that work:

- If x_i contains any prescription medication name $\hat{y}_i = 1$
- If x_i is is mostly capital letters $\hat{y}_i = 1$
- If x_i contains "Make **Amount** every **Time Period**" $\hat{y}_i = 1$

```python
import re

SPAM = 1
MEDICINE_LIST = ["Acetylcysteine", "Apixaban","..."]

def spam_classifier_heuristic(text: str) -> bool:
    """
    Heuristics to classify whether a string is a spam email or not
    """
    for medicine in MEDICINE_LIST:
        if text.contains(medicine):
            return SPAM
    if text.upper() == text:
        return SPAM
    if re.match(r"((M|m)ake)\s*((\w*)\s*)*(\$\d*)\s*((per|for\sjust)\s*(week|day|year))|(weekly|daily|monthly)", text) is not None:
        return SPAM
    else:
        return 1 - SPAM
```

Can we learn simple heuristics?

To apply any kind of learning algorithms we have seen before we need to convert x into a numerical representation h.

- Given a vocabulary $V_{j=1:M}$, determine h such that: $h_j = 1$ whenever token j from the vocabulary is present in x.
- Each datapoint x is represented by an M dimensional vector of 0's and 1's
- How do we determine the vocabulary?
- Just list and count all the words in all of the documents, and then only keep the top M.
- We have numerical features, so just plug them as input into any "standard" algorithm: Logistic Regression

Bag of Words

This simple binary representation is called a **(binary) Bag of Words**.

- What is included in the representation h of x?
- What if we care about more than just the presence / abscence of a specific word?
- We could just include the count of each word turning h from a vector of 1's and 0's into a vector of counts.
- What about phrases? "Polyethylene Glicol"?

N-Grams

Instead of considering single words as entries in a vocabulary, perhaps we would want to consider phrases.
An **N-gram** is a contiguous sequence of N tokens from a given text.
Under $N = 1$; also called unigrams

$$V = (\text{"This"}, \text{"is"}, \text{"a"}, \text{"sentence"})$$

$$\text{"This is a sentence"} = (1, 1, 1, 1)$$

$$\text{"A sentence"} = (0, 0, 1, 1)$$

Under $N = 2$; also called bigrams

$$V = (\text{"This is"}, \text{"is a"}, \text{"a sentence"})$$

$$\text{"This is a sentence"} = (1, 1, 1)$$

$$\text{"A sentence"} = (0, 0, 1)$$

Common words

Some words are incredibly common, but do not contribute a lot in terms of distinguishing between texts.

- Virtually any text in English will contain "the" or "a"
- Should we include those in the vocabulary?
- Can we learn which words to include in the vocabulary?
- We can better represent documents by the **relative frequency** of words in them.
- Note: in practice we often remove some words from all text and ignore them completely. You will see those referred to as **stopwords**

Term Frequency

We can represent the "count" (**Term Frequency**) of a word in many different ways:

- Raw count of times it is present in x: **BoW**
- Binarized count of times it is present in x: **binary BoW**
- Count of times it is present in x divided by number of tokens in x
- Raw count scaled by number of other terms (not count of) in x
- Log of the raw count

Term Frequency example

$V = (\text{"This"}, \text{"is"}, \text{"a"}, \text{"sentence"}, \text{"another"}, \text{"not"}, \text{"Yet"})$
$x_1 =$"This is a sentence. This is another sentence."
$x_2 =$"This is not a sentence. Yet another not a sentence"

Inverse Document Frequency

Just like with term frequency, we can represent the "relative prevalence" (**Inverse Document Frequency**) of a word in many different ways:
Denote $N_j = \sum_{i=1}^{N} I\left(V_j \in x_i\right)$ count of datapoints that include j-th word in the vocabulary.

- $$\frac{1}{N_j}$$
- $$\frac{N}{N_j}$$
- $$\log(\frac{N}{N_j})$$

You can think of it as a scaling factor for each of the words in the vocabulary.

TF-IDF

By combining Term Frequency with Inverse Document Frequency we can measure how common the word is in a particular datapoint relative to other documents.

$$\text{TF-IDF}(x) = TF(x) \times IDF(x)$$

TF-IDF example

$V = (\text{"This"}, \text{"is"}, \text{"a"}, \text{"sentence"}, \text{"another"}, \text{"not"}, \text{"Yet"})$
$x_1 =$"This is a sentence. This is another sentence."
$x_2 =$"This is not a sentence. Yet another not a sentence"

Embeddings

All of the methods we talked about can be used to generate numerical representations of whole documents, by the use of just the word occurences, and simple functions. We say that h_i is the **embedding** of x_i, and we call $g(x) = h$ an embedding function. We can then re-frame our problem of learning $f_\theta(x) = y$ as:

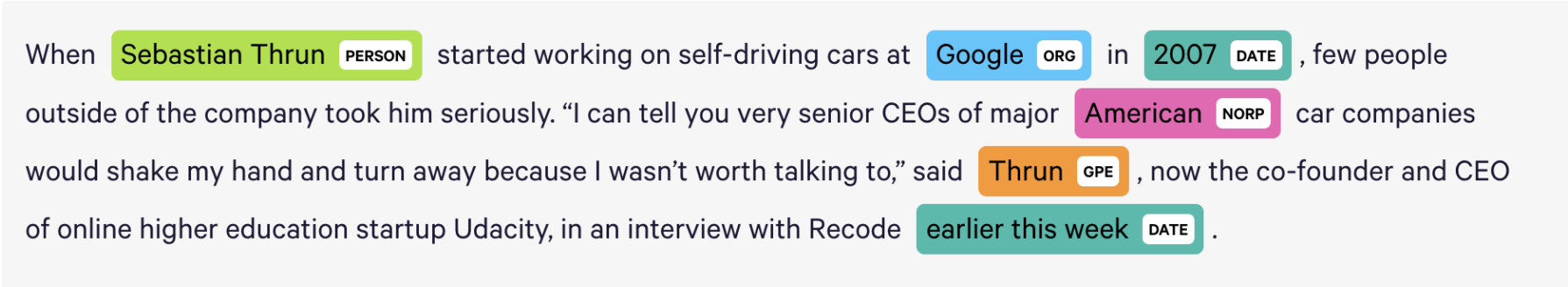
$$f_\theta(x) = c_{\theta_1}(h) = c_{\theta_1}(g_{\theta_2}(x)) = y$$

Where c_θ is any classification / regression function, and g_θ is an embedding function.

- Embeddings are not restricted to documents.
- How could we embed an image?
- What if our datapoint consists of an image and text?

Token classification

Sometimes we care about something more granular than just the whole document. Perhaps we want to identify each parts of text that correspond to certain concepts:



- What should we get a represantation of?
- How could we do this?

[1] from SpaCy: https://explosion.ai/demos/displacy-ent

Word2Vec

We start with a fairly strong assumption: "Words that have similar meanings will occur in simmilar contexts" Based on that we define a **context** of size **k** of token $x_{i,j}$ [1]as a set of tokens:

$$\text{context}(x_{i,j}) = \{x_{i,j-k}, x_{i,j-(k-1)}, \ldots, x_{i,j-1}, x_{i,j+1}, \ldots x_{i,j+k}\}$$

Then given a set of datapoints $x_{i=1:N,j=1:M_i}$ and a vocabulary $V_{r=1:R}$ we define an unsupervised learning task of predicting what words occur in the context of each word in the vocabulary. More formally, given a sequence of training words $x_1, \ldots, x_T$ we want to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in\text{context}(t)} logp(w_j|w_t)$$

[1]This is also called a **skip-gram**

Skip Gram Continued

The basic formulation of $p(x|w)$ uses the softmax function:

$$p(x|w) = \frac{\exp\left((u(w)^T(v(x)\right)}{\sum_{r=1}^{R}\exp\left((u(w))^T(v(V_r))\right)}$$

where $u(w)$ is the "word" and $v(w)$ is the "context" representation of word w. What are those representations?

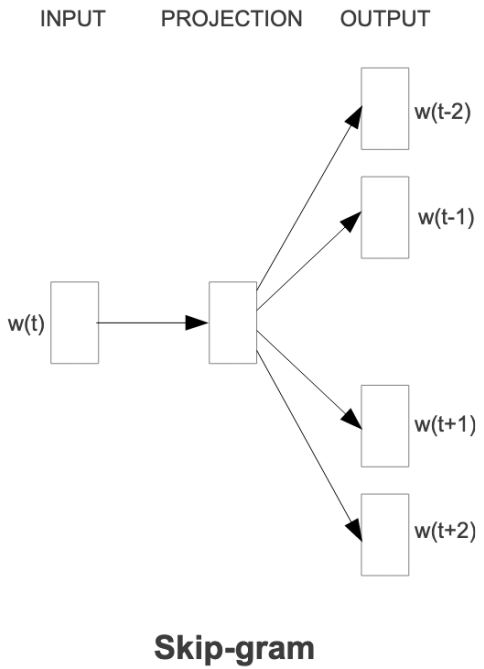
- In our particular case we will take u and v to be simple linear projections of the **one-hot** (binary BoW) encoding of the word, and context respectively.

$$u(w) = bBoW(w)U^T$$

The matrix U will be of size $R \times e$ where **e** is the embedding dimension, which is a hyperparameter you chose.

Skip Gram Continued

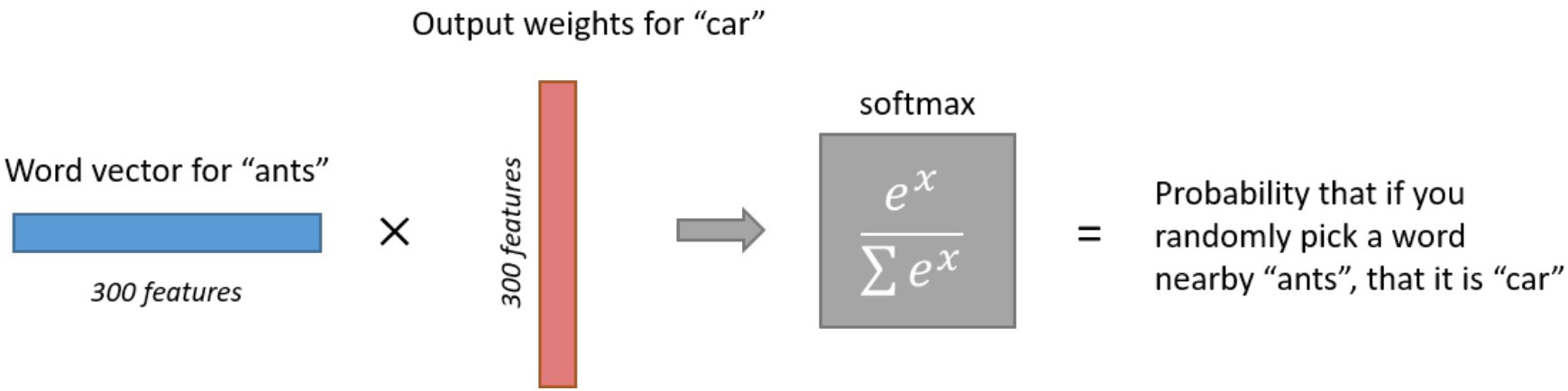
Notice that the the bBoW(w) is a binary vector with all 0's and a single 1 at the index of word w in the Vocabulary!
Similarily we define $v(w)$ to be a linear projection that back from $u(w)$ to predict each word in the context.



[1]"Efficient Estimation of Word Representations in Vector Space", Mikolov et al

Skip Gram Vis

How to estimate $p(\text{"car"}|\text{"ants"})$?



[1] Image from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2Vec

The training process is then as follows:

- Initialize the two matrices
- Sample pairs of words (w_i, w_j) and compute the objective
- Gradient Descent based on the objective.
- After convergence keep only the matrix U
- Embedding of word at vocabulary index i is just the i-th row of U

$$\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}17 & 24 & 1\\ 23 & 5 & 7\\ 4 & 6 & 13\\ 10 & 12 & 19\\ 11 & 18 & 25\end{bmatrix} = \begin{bmatrix}10 & 12 & 19\end{bmatrix}$$

[1] Image from:
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2Vec notes

- In practice this training procedure is not feasible - we would have to compute softmax over the entire vocabulary at every step.
- There are a lot of tricks and improvements over the years - really worth reading the original paper.
- There is another possible objective called **Continuous Bag of Words (CBoW)** that is the exact opposite of the Skip Gram Objective - estimate the word given context instead of context given word.
- What are we missing?

Token Classification and Embeddings continued

- To classify tokens, we can just take the Word2Vec embedding of each token as an input to e.g. Linear Regression / Multinomial Naive Bayes, and estimating probabilities that the token belongs to a certain category.
- We can combine Word2Vec representations into document level representations.
- We can combine Different embedding methods! Nothing is stopping you from taking TF-IDF vector of a document and "stacking" the average of all Word2Vec vectors in the document.

Sequence 2 Sequence Tasks

- What if our output should also be in the form of text?
- Re-frame the "context" to only feature words before the input
- Train in the unsupervised setting of CBoW, with the modified context.
- Idea: Sample the next word, conditional on the previous k based on the CBoW softmax.
- What kind of model is that?

Similarity

- What does it mean for 2 words to be similar?
- What does it mean for 2 datapoints to be similar?
- The most common way to measure similarity in NLP is via the cosine similarity

We define **cosine similarity** between two vectors to be:

$$\text{cosine sim}(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

This is especially convenient for binary BoW.

Some Cool properties of W2V

Let $h(x)$ be the Word2Vec embedding of the word x. We can perform some vector algebra to find that:

- $$h(\text{"Athens"}) - h(\text{"Greece"}) + h(\text{"Germany"}) \approx h(\text{"Berlin"})$$
- $$h(\text{"Mice"}) - h(\text{"Mouse"}) + h(\text{"Dollar"}) \approx h(\text{"Dollars"})$$
- In the original paper they propose a set of 14 categories and evaluate accuracy on these kinds of "algebraic" operations to find an accuracy of $\sim 55 - 60\%$

You can just download the original Word2Vec embeddings and play around!

Summary

- The first modelling step with any data should be to convert it to a numerical representation.
- We can learn embeddings from unlabelled data.
- We can easily combine different kinds of embeddings to improve how we represent data.
- We have learned a number of different document, and token embedding algorithms.
- Human Language is hard!