Frailty Classifier Project
Modelling Strategy

Andrew Crane-Droesch

February 4, 2020

Intro

Document matrix

Start with a sparse matrix "one-hot" encoded representation of a document:

- The rows correspond to positions in a sequence
- The columns correspond to unique words
- The sum of the matrix equals the number of rows

$\boldsymbol{D} \equiv$

	I	am	Sam	$would$	you	$like$
I	1	0	0	0	0	0
am	0	1	0	0	0	0
Sam	0	0	1	0	0	0
Sam	0	0	1	0	0	0
I	1	0	0	0	0	0
am	0	1	0	0	0	0
$would$	0	0	0	1	0	0
you	0	0	0	0	1	0
$like$	0	0	0	0	0	1

Call it $\boldsymbol{D}$. It has dimensions $N \times V$ – total words by unique words.

Document matrix

- Document matrices are *sparse*. Lots of zeros.
- They are high-dimensional. Using $\boldsymbol{D}$ directly as a design matrix in a model is generally inefficient.
- A document matrix doesn't explicitly encode any informaiton about how words might be similar to each other. Synonyms are wholly different from each other.
  - The euclidian distance between any two word vectors (rows in the matrix) will always be $\sqrt{2}$
- Summing them column-wise makes a unigram matrix:

	I	am	Sam	$would$	you	$like$
$1-gram$	2	2	2	1	1	1

Embeddings

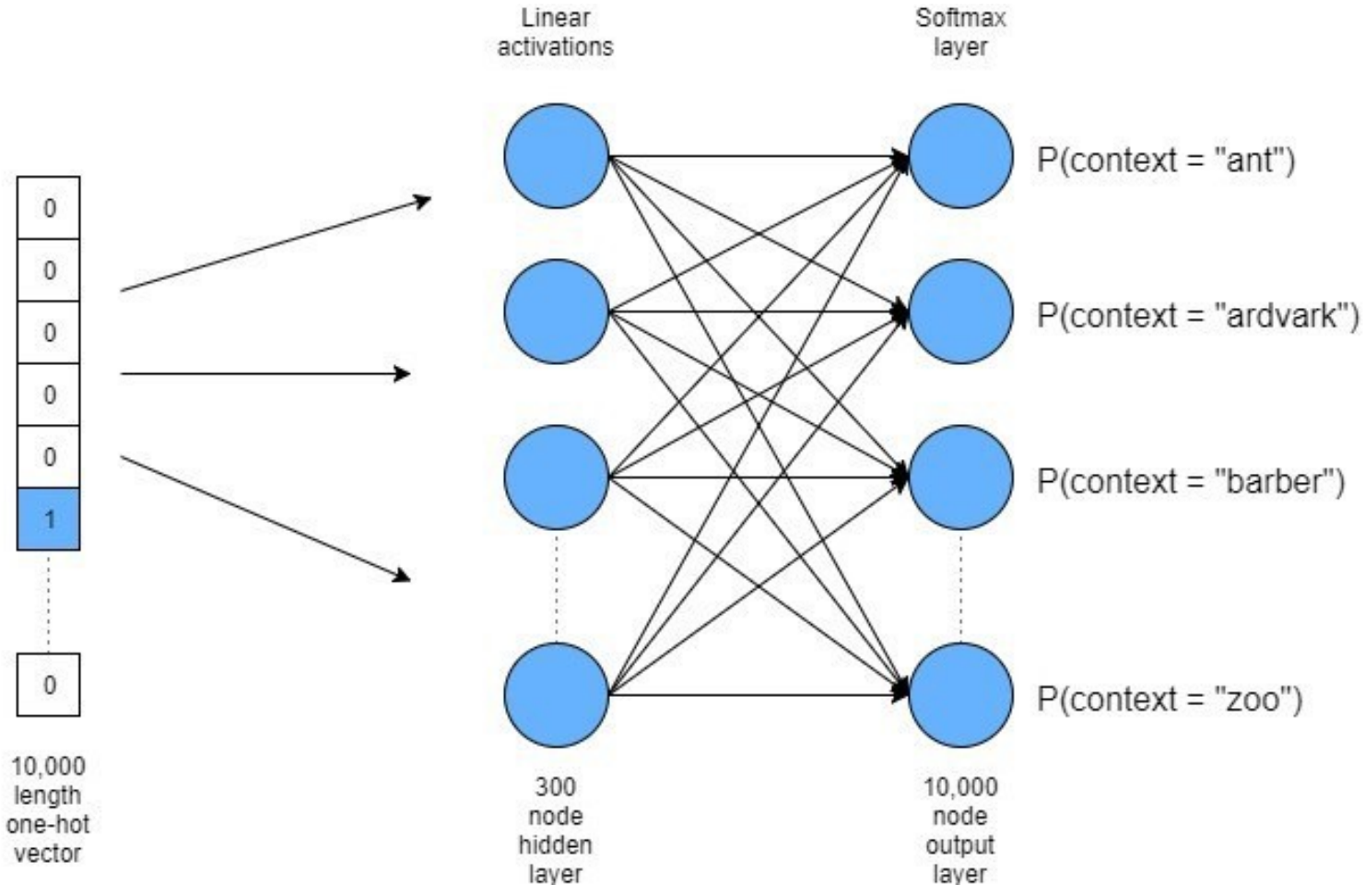
Embeddings attempt to do the following:

- Reduce the dimensionality of $\boldsymbol{D}$ without losing too much information
- Capture the similarity between similar words

Different embedding methods do this in different ways, but all create analogous output:

- word2vec, fasttext, GloVe

Embeddings in pictures



Embeddings in math

(a simple) Embedding model:

$$\boldsymbol{D} = s\left(\boldsymbol{D}\boldsymbol{\Gamma}_0\boldsymbol{\Gamma}_1\right) + \epsilon$$

where:

- $\boldsymbol{D}$ is the document matrix, dimension $N \times V$
- $\boldsymbol{\Gamma}_0$ is the first weight matrix. It has dimension $V \times U$, where $U << V$. It is responsible for "embedding" $\boldsymbol{D}$ in a lower-dimensional space
- $\boldsymbol{\Gamma}_1$ is the second weight matrix. Its dimension is $U \times V$. It is responsible for taking the lower-dimensional representation and bringing it back to the original resolution.
- $s()$ is the softmax function. It maps the maps real numbers to a vector of probabilities summing to 1.
- ϵ is the error to be minimized. As the bottleneck gets skinnier (i.e.: as U gets smaller), ϵ will have to increase.

Embeddings in math

(a simple) Embedding model:

$$\boldsymbol{D} = s\left(\boldsymbol{D}\boldsymbol{\Gamma}_0\boldsymbol{\Gamma}_1\right) + \epsilon$$

where:

- The embedded text, call it $\boldsymbol{E}$, is defined as $\boldsymbol{D}\boldsymbol{\Gamma_0}$ (dimension $N \times U$)
- (most) Embedding models are trained by computing the derivatives of the parameter matrices $\boldsymbol{\Gamma}_0, \boldsymbol{\Gamma}_1$ with respect to a loss function, changing the parameters to make the loss smaller, and then recomputing the derivatives and repeating. This is backpropagation.
- the error ϵ is implicit. The goal is to minimize it, to get the best possible $\boldsymbol{\Gamma}$. But as the dimension of $\boldsymbol{\Gamma}$ gets smaller, this gets harder to do.

Embeddings in practice

Training and using embeddings are separate processes. Trained embedding models give you $\mathbf{\Gamma}_0$. You then supply your own text to multiply it against. For example, take $\boldsymbol{D}$ to be our doctor Seuss example from before. The embeddings $\mathbf{\Gamma}_0$ might look like this:

$$\mathbf{\Gamma}_0 \equiv$$

	$u1$	$u2$
I	1	2
am	3	4
Sam	5	6
$would$	7	8
you	9	10
$like$	11	12

(Note that the entries of the matrix are totally unrealistic in this example)

Embeddings in practice

Say you wanted to embed the phrase "I like Sam" into this two-dimensional space:

$$\left[\begin{array}{c|cccccc} I & 1 & 0 & 0 & 0 & 0 & 0 \\ like & 0 & 0 & 0 & 0 & 0 & 1 \\ Sam & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c|cc} & u1 & u2 \\ \hline I & 1 & 2 \\ am & 3 & 4 \\ Sam & 5 & 6 \\ would & 7 & 8 \\ you & 9 & 10 \\ like & 11 & 12 \end{array}\right] = \left[\begin{array}{c|cc} & u1 & u2 \\ \hline I & 1 & 2 \\ like & 11 & 12 \\ Sam & 5 & 6 \end{array}\right]$$

Because the the document matrix $\boldsymbol{D}$ has only one "1" per row, this matrix multiplicaton is the same as a "lookup" operation.

Still, the matrix notation will come in handy later.

Word embeddings to sentence or span embeddings

Supervised learning jobs can all be conceptualized as

$$y = f(\boldsymbol{X}) + \epsilon$$

where y is some label and $\boldsymbol{X}$ is a matrix of features. So let's add a label to our toy example:

	I	*am*	*Sam*	*would*	*you*	*like*	**label**
I	1	0	0	0	0	0	0
am	0	1	0	0	0	0	0
Sam	0	0	1	0	0	0	1
Sam	0	0	1	0	0	0	0
I	1	0	0	0	0	0	0
am	0	1	0	0	0	0	0
would	0	0	0	1	0	0	1
you	0	0	0	0	1	0	1
like	0	0	0	0	0	1	0

(I labeled words that come after a verb)

Word embeddings to sentence or span embeddings

If we fit

$$\textbf{label} = f(\boldsymbol{D}) + \epsilon$$

or

$$\textbf{label} = f(\boldsymbol{E}) + \epsilon$$

our model would do nothing but capture the correlations between individual words and labels. But what if we want to capture the surrounding context?

First, let's remember that the matrix equation above is the same as

$$label_i = f(E_i) + \epsilon$$

where i indexes rows from 1 to N. A natural strategy could be to include lags and leads:

$$label_i = f(E_i, E_{i-1}, E_{i+1}) + \epsilon$$

But lots of other possibilities exist.

Word embeddings to sentence or span embeddings

Consider the span "I am Sam". Its embedding is:

	$u1$	$u2$	**label**
I	1	2	0
am	3	4	0
Sam	5	6	1

The centroid word is "am", which is associated with a label of zero. How do we associate information about the surrounding words with that zero label? Several options, all of which involve "windowing":

- Lags and leads
- span-wise maxes and mins, averages

	$u1$	$u2$	**label**	lag		$lead$		min		max	
am	3	4	0	1	2	5	6	1	2	5	6

In this toy example, the lags and leads happen to be the same as the maxes and mins, but that won't generally be the case.

Word embeddings to sentence or span embeddings

We've been working with a bandwidth of 1 (as measured from the centroid.). This is the same thing as a window size of 3.

Taking an average over this window is the same as doing the following:

$$[.33, .33, .33] \left[\begin{array}{c|cc} & u1 & u2 \\ \hline I & 1 & 2 \\ am & 3 & 4 \\ Sam & 5 & 6 \end{array}\right] = [3, 4]$$

But why weight evenly? We could do

$$[.2, .7, .1] \left[\begin{array}{c|cc} & u1 & u2 \\ \hline I & 1 & 2 \\ am & 3 & 4 \\ Sam & 5 & 6 \end{array}\right] = [2.8, 3.8]$$

Optimal weighting schemes could be hyperparameters. Or they could be *directly estimated* at the same time as model parameters if we're using a neural network. This is very similar to the concept of *attention*.

Embedded Ngrams

Ngrams are usually computed over the whole of a document, but they could also be computed over a window. Here's bigrams for our dummy example:

$\boldsymbol{G}^2 =$

	(I, I)	(I, am)	(I, sam)	$\ldots$	(am, Sam)	(Sam, Sam)
I	0	1	0	$\ldots$	0	0
am	0	1	0	$\ldots$	1	0
Sam	0	0	0	$\ldots$	1	0

These bigrams are going to be much too high-dimensional to use as-is, but we can embed them into a lower-dimensional space:

$$E^2 = \underbrace{\boldsymbol{G}^2}_{N \times V^2} \left[\underbrace{\boldsymbol{\Gamma}_0 \otimes \boldsymbol{\Gamma}_0}_{V^2 \times U^2} \right]$$

Doing so could help capture topology-dependent phenomena like negation and adjectives/adverbs. The dimensionality is too high to do this with the whole vocab even at $U = 100$, but this could be reduced through TF-IDF weighting of bigrams and PCA on the matrix $\boldsymbol{\Gamma}$.

Models

Given a label and a transformation that takes the document matrix $\boldsymbol{D}$ and returns $\boldsymbol{X}$, fitting a statistical model is generally a fairly straightforward exercise of tuning hyperparameters.

Neural nets are the exception, because they effectively create their own design matrices through representation learning. I'll discuss those later.

We'll try:

- Penalized logistic regression
- Random forests
- XGboost (with trees, probably)