Learning Word Embeddings for a Latin Corpus

Nate Stringham Advisor: Dr. Mike Izbicki

Pomona College

April 10, 2020

But First, a Story

Imagine the following scenario

But First, a Story

Imagine the following scenario

- You're visiting a long lost cousin in the U.S. state of Wisconsin

But First, a Story

Imagine the following scenario

- You're visiting a long lost cousin in the U.S. state of Wisconsin
- It's a nice sunny day so you've decided to spend some time at a local park

But First, a Story

Imagine the following scenario

- You're visiting a long lost cousin in the U.S. state of Wisconsin
- It's a nice sunny day so you've decided to spend some time at a local park
- Soon the heat starts to get to you and you find yourself in need of a drink

But First, a Story

Imagine the following scenario

- You're visiting a long lost cousin in the U.S. state of Wisconsin
- It's a nice sunny day so you've decided to spend some time at a local park
- Soon the heat starts to get to you and you find yourself in need of a drink
- Luckily you see a local and ask them if they could point you to a drinking fountain

But First, a Story

Imagine the following scenario

- You're visiting a long lost cousin in the U.S. state of Wisconsin
- It's a nice sunny day so you've decided to spend some time at a local park
- Soon the heat starts to get to you and you find yourself in need of a drink
- Luckily you see a local and ask them if they could point you to a drinking fountain
- Their response: 'There's a bubbler just over there!'

Morales of the Story

What does that story teach us?

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex
    - Semantics

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex
    - Semantics
    - Syntax

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex
    - Semantics
    - Syntax
    - ... and many more

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex
    - Semantics
    - Syntax
    - ... and many more
3. Context is key

Morales of the Story

What does that story teach us?

1. Humans are great at natural language processing (NLP)
2. Natural language data is complex
    - Semantics
    - Syntax
    - ... and many more
3. Context is key

Need a mathematical representation for natural language data!

Table of Contents

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

Word Embeddings

Definition

*A **word embedding** is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{we} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{are} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{going} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{to} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{create} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{some} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{latin} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{word} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{embeddings} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Types of Representations

Definition

We say a vocabulary of words has been **one-hot-encoded** *if*

- *each word is represented by a vector with dimension equal to the size of the vocabulary*
- *the entries of the vectors corresponds to a specific word in the vocabulary.*
- *the* i^{th} *word in our vocabulary is represented by a vector with a value of 1 in the* i^{th} *entry and 0 in all other entries.*

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{we} = \begin{pmatrix} 0.98 \\ -1.45 \\ 0.22 \\ 0.06 \\ -3.78 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{are} = \begin{pmatrix} 5.23 \\ 0.63 \\ 0.28 \\ 0.06 \\ 0.40 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{going} = \begin{pmatrix} -0.32 \\ 0.33 \\ 2.79 \\ 0.45 \\ 0.73 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{to} = \begin{pmatrix} 1.98 \\ 0.88 \\ 0.23 \\ 0.03 \\ 3.40 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{create} = \begin{pmatrix} 0.41 \\ 0.60 \\ -0.42 \\ 0.55 \\ 0.78 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{some} = \begin{pmatrix} 0.88 \\ -0.45 \\ -0.23 \\ 0.06 \\ 0.69 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{latin} = \begin{pmatrix} 3.20 \\ 0.51 \\ -0.72 \\ 0.08 \\ 1.50 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector* $w_i \in \mathbb{R}^n$ *where* w_i *represents the* i^{th} *word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{word} = \begin{pmatrix} -0.47 \\ 0.45 \\ 0.97 \\ 0.68 \\ -0.78 \end{pmatrix}$$

Word Embeddings

Definition

A **word embedding** *is a vector $w_i \in \mathbb{R}^n$ where w_i represents the i^{th} word in the vocabulary.*

$$V = \{\texttt{we}, \texttt{are}, \texttt{going}, \texttt{to}, \texttt{create}, \texttt{some}, \texttt{latin}, \texttt{word}, \texttt{embeddings}\}$$

$$\texttt{embeddings} = \begin{pmatrix} 6.23 \\ -0.78 \\ 0.93 \\ -0.03 \\ 0.44 \end{pmatrix}$$

Types of Representations

Definition

We say a vocabulary of words has been **one-hot encoded** *if*

- *the dimension of each vector is equal to the size of the vocabulary*
- *the i^{th} word in the vocabulary is represented by a vector with a value of 1 in the i^{th} entry and 0 in all other entries.*

Definition

An embedding is said to have a **distributed** *representation if each word is represented by a vector of weights where each entry is a real number.*

Types of Representations

Definition

We say a vocabulary of words has been **one-hot encoded** *if*

- *the dimension of each vector is equal to the size of the vocabulary*
- *the i^{th} word in the vocabulary is represented by a vector with a value of 1 in the i^{th} entry and 0 in all other entries.*

Definition

An embedding is said to have a **distributed** *representation if each word is represented by a vector of weights where each entry is a real number.*

Key Difference

One-hot are sparse and large, distributed are dense and small!

word2vec

word2vec

word2vec

word2vec

- method for creating ditributed word embeddings.

word2vec

word2vec

- method for creating ditributed word embeddings.
- ‘looks’ at words in context

word2vec

word2vec

- method for creating ditributed word embeddings.
- 'looks' at words in context

  `"Flectere si nequeo superos Acheronta movebo".`

word2vec

- method for creating ditributed word embeddings.
- ‘looks’ at words in context

    "Flectere si nequeo superos Acheronta movebo".

Architectures

word2vec

word2vec

- method for creating ditributed word embeddings.
- ‘looks’ at words in context

  "Flectere si nequeo superos Acheronta movebo".

Architectures

- Continuous Bag of Words (CBOW)

word2vec

word2vec

- method for creating ditributed word embeddings.
- 'looks' at words in context

```
"Flectere si nequeo superos Acheronta movebo".
```

Architectures

- Continuous Bag of Words (CBOW)
  - maximize the probability of predicting target words from context

word2vec

word2vec

- method for creating ditributed word embeddings.
- 'looks' at words in context

```
"Flectere si nequeo superos Acheronta movebo".
```

Architectures

- Continuous Bag of Words (CBOW)
  - maximize the probability of predicting target words from context
- Skipgram

word2vec

word2vec

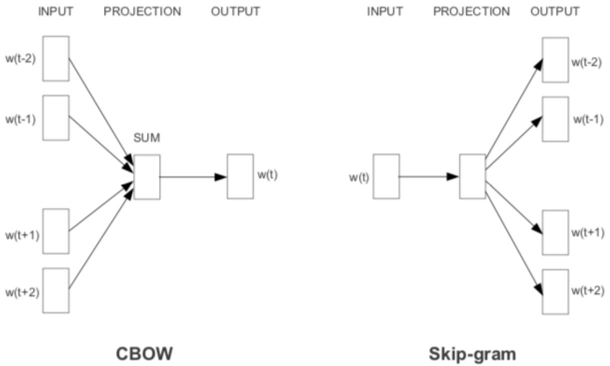
- method for creating ditributed word embeddings.
- ‘looks’ at words in context

  "Flectere si nequeo superos Acheronta movebo".

Architectures

- Continuous Bag of Words (CBOW)
  - maximize the probability of predicting target words from context
- Skipgram
  - maximize the probability of predicting context words from target

Visual Intuition



Visual Intuition

`"Flectere si nequeo superos Acheronta movebo".`

Visual Intuition

`"Flectere si nequeo superos Acheronta movebo".`

Visual Intuition

"Flectere si nequeo superos Acheronta movebo".

Visual Intuition

"Flectere si nequeo superos Acheronta movebo".

Visual Intuition

"Flectere si nequeo superos Acheronta movebo".

Visual Intuition

"Flectere si nequeo superos Acheronta movebo".

Learning using a skipgram

Given a corpus of words $w_1, w_2, \ldots, w_T$ the skipgram minimizes

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \tag{1}$$

Learning using a skipgram

Given a corpus of words $w_1, w_2, \ldots, w_T$ the skipgram minimizes

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t) \tag{1}$$

where

$$p(a|b) = \frac{\exp(v_a'^{\top} v_b)}{\sum_{w=1}^{W}\exp(v_w'^{\top} v_b)} \tag{2}$$

where W is the size of our vocabulary and v'_w and v_w are the input and output vector representations of word w.

Learning using a skipgram

Given a corpus of words $w_1, w_2, \ldots, w_T$ the skipgram minimizes

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{1}$$

where

$$p(a|b) = \frac{\exp(v_a'^{\top} v_b)}{\sum_{w=1}^{W}\exp(v_w'^{\top} v_b)} \tag{2}$$

where W is the size of our vocabulary and v'_w and v_w are the input and output vector representations of word w.
The weights for each word are updated using stochastic gradient descent

$$w^{t+1} = w^t + \eta_t \frac{\partial}{\partial w}\ell(w) \tag{3}$$

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
    - 5 declensions

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
    - 5 declensions
    - 4 tenses

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
    - 5 declensions
    - 4 tenses
    - 3 genders

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
  - 5 declensions
  - 4 tenses
  - 3 genders
- Latin text data is comparatively scarce

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
  - 5 declensions
  - 4 tenses
  - 3 genders
- Latin text data is comparatively scarce
  - Historical documents

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
  - 5 declensions
  - 4 tenses
  - 3 genders
- Latin text data is comparatively scarce
  - Historical documents
- Under-resourced and under-studied

Training a Latin Model

What (if anything) makes training Latin word embeddings different?

- Latin is "morphologically rich"
  - 5 declensions
  - 4 tenses
  - 3 genders
- Latin text data is comparatively scarce
  - Historical documents
- Under-resourced and under-studied
  - few benchmarks for comparing results

My Latin Models

3 Different Data Sources

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique
2. Christian Latin - 2.7 million tokens, 90 thousand unique

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique
2. Christian Latin - 2.7 million tokens, 90 thousand unique
3. Full Latin - 8.3 million tokens, 236 thousand unique

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique
2. Christian Latin - 2.7 million tokens, 90 thousand unique
3. Full Latin - 8.3 million tokens, 236 thousand unique

2 Model Types

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique
2. Christian Latin - 2.7 million tokens, 90 thousand unique
3. Full Latin - 8.3 million tokens, 236 thousand unique

2 Model Types

1. word2vec

My Latin Models

3 Different Data Sources

1. Medieval Latin - 1.7 million tokens, 75 thousand unique
2. Christian Latin - 2.7 million tokens, 90 thousand unique
3. Full Latin - 8.3 million tokens, 236 thousand unique

2 Model Types

1. word2vec
2. Fasttext (n-grams)

The Evaluation Problem

After a thorough training of your model(s) we are excited to see how 'good' our embeddings are. So, we inspect a few of them.

The Evaluation Problem

After a thorough training of your model(s) we are excited to see how 'good' our embeddings are. So, we inspect a few of them.

```
[ 1.09798884e+00  6.61867380e-01 -7.10478052e-02  1.08630925e-01
  1.90411139e+00 -1.34148002e-01  1.13010979e+00 -1.08128226e+00
  5.67232482e-02 -1.53587058e-01 -4.61597502e-01  2.20323277e+00
 -2.83151940e-02 -1.61308491e+00 -7.17172861e-01  1.03687203e+00
  9.62423146e-01 -1.09708142e+00 -2.44244242e+00  8.22242856e-01
 -5.69651365e-01 -1.57382798e+00  1.92259073e+00  1.61573005e+00
 -1.87173045e+00 -4.40778702e-01  8.33371878e-01 -3.92222732e-01
 -5.92744529e-01  8.38984668e-01  8.04344654e-01  1.07055962e-01
 -2.29991511e-01 -6.51594162e-01 -8.89630079e-01 -1.25787973e+00
 -2.66330659e-01 -8.97910058e-01 -1.34007168e+00 -1.63498962e+00
  2.19974113e+00  3.55621070e-01  1.38610697e+00  4.99659151e-01
  1.83429384e+00  1.45139420e+00  1.27718091e+00 -5.23355544e-01
 -1.48872185e+00 -8.93208265e-01 -2.76439548e-01 -1.15471005e+00
  5.90652406e-01 -1.46208286e+00 -1.07289457e+00  2.24772835e+00
 -6.43072533e-04 -8.49872053e-01  1.41535699e-01 -1.29253221e+00
  1.33586717e+00  1.43089890e+00 -7.10735202e-01  3.68400872e-01
  3.47683936e-01  1.00279522e+00 -9.98684585e-01  9.28291202e-01
  1.26067567e+00 -1.19633090e+00 -1.00435853e+00 -6.27937138e-01
  1.02863836e+00 -1.45649803e+00 -1.00423229e+00 -1.11265707e+00
 -1.33893037e+00  8.99451971e-01  8.59849989e-01  8.97351146e-01
  1.95721543e+00 -2.38950324e+00  6.98350430e-01 -2.54332781e+00
 -7.80221343e-01  1.34824312e+00 -1.30275965e+00  4.19737220e-01
  7.49867558e-01  1.43526709e-02 -3.36812735e-01  6.33926690e-01
 -1.28516543e+00 -9.78036046e-01 -9.70386505e-01  1.48103321e+00
  1.77720714e+00 -6.30774140e-01  6.73609793e-01  1.51666775e-01]
```

```
[-0.2927009  -2.3539     -3.082364   -2.0755804  -0.23387048  0.03116352
 -3.4240723   0.6804483   2.5297556   1.4924724  -2.4403913  -0.30286705
  3.3442798  -0.45448813 -0.8306715  -0.03494536  0.31276062  1.6929202
 -3.229696   -2.8762164   3.9146178  -0.32805213  0.6030931  -0.88103676
  1.0879899  -1.7850647   2.5750897  -1.7574708  -4.274682   -0.62027305
 -1.0076166   0.8643299  -0.8952144   3.1832883   3.9998395   3.2029138
 -1.5560957   0.27520418 -0.966296    1.0861672  -0.8834181  -2.5094414
 -1.9215351   1.0475558  -2.3829036  -0.17971425 -2.8454487   4.4128757
  0.23126438 -0.58600175  2.3757505  -0.08426343  0.6372437  -2.4128644
 -1.5134279  -0.44377014 -0.18394051 -3.542668   -2.3582625   1.5922571
  0.30302525  3.3288202   0.4322284  -0.3159594   2.7113795  -4.014404
  2.4380522  -2.5136967   3.5966437   3.757738    2.2040672  -0.1174969
 -0.42089543 -2.2368116  -0.10117842  2.6468835   2.606288   -1.93117
 -2.2637906  -1.4581017   0.891239    0.47737953  0.6579282   2.549008
 -0.7854796  -5.4956417  -0.9731244  -0.50446683 -0.5956649  -1.3335679
  3.1565733  -0.6824405   1.283595    2.0075195   1.0156357   0.34581745
 -2.6737473   1.6112593   2.4512193   0.465902  ]
```

```
[ 3.4993453  -0.75153375  2.7960842   1.0670502   1.7269794   2.849036
  2.3116152   0.28173736  0.7484232   1.8855542  -0.17695513  0.75033337
  3.0245388   2.2880986   0.62054646  3.8393073   1.3523717   1.1088774
  0.4298605   0.68544114  0.05531956  0.02740468 -0.06775122  0.7071461
  0.6200045   1.6111546  -1.6065401   0.29958934  2.1255572   3.9588451
  2.2673967  -2.420242   -1.5391635   1.5550674   3.338346    2.6921241
  4.594245    1.6169097   4.448282   -3.3279974   1.5572077   1.1702842
  0.8155618  -2.965235    1.2714497  -0.23951891  3.8141346  -0.52887446
 -1.9785266  -3.3365355  -2.4357328  -0.03013381  5.2083526  -0.47481996
 -2.44593     1.2577963  -1.8394886  -0.06880338  1.7498951  -1.2576197
  1.3084195  -2.5954428  -2.5381258  -1.4663447  -1.3752079  -1.2812922
  2.5025816   2.6445158  -0.25550297 -1.6185617  -0.34789237 -0.43534958
 -0.18697618  1.2172858  -3.026618   -1.9473408  -2.2698424   1.3054887
 -2.340305    4.3331575   0.38164425 -2.977643    2.29641    -7.2448053
 -0.6058418  -1.5438882   0.6236802   6.0693827   0.98486584  1.1576533
 -0.49797067 -1.5955713  -3.2062447  -0.39012623  0.4121893   2.5176299
  2.6070173  -1.0555307   3.642642    1.1389123 ]
```

The Evaluation Problem

After a thorough training of your model(s) we are excited to see how 'good' our embeddings are. So, we inspect a few of them.

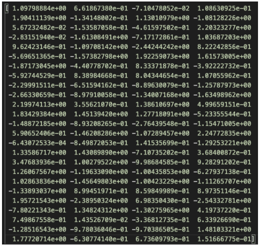
```
[ 1.09798884e+00  6.61867380e-01 -7.10478052e-02  1.08630925e-01
  1.90411139e+00 -1.34148002e-01  1.13010979e+00 -1.08128226e+00
  5.67232482e-02 -1.53587058e-01 -4.61597502e-01  2.20323277e+00
 -2.83151940e-02 -1.61308491e+00 -7.17172861e-01  1.03687203e+00
  9.62423146e-01 -1.09708142e+00 -2.44244242e+00  8.22242856e-01
 -5.69651365e-01 -1.57382798e+00  1.92259073e+00  1.61573005e+00
 -1.87173045e+00 -4.40778702e-01  8.33371878e-01 -3.92222732e-01
 -5.92744529e-01  8.38984668e-01  8.04344654e-01  1.07055962e-01
 -2.29991511e-01 -6.51594162e-01 -8.89630079e-01 -1.25787973e+00
 -2.66330659e-01 -8.97910058e-01 -1.34007168e+00 -1.63498962e+00
  2.19974113e+00  3.55621070e-01  1.38610697e+00  4.99659151e-01
  1.83429384e+00  1.45139420e+00  1.27718091e+00 -5.23355544e-01
 -1.48872185e+00 -8.93208265e-01 -2.76439548e-01 -1.15471005e+00
  5.90652406e-01 -1.46208286e+00 -1.07289457e+00  2.24772835e+00
 -6.43072533e-04 -8.49872053e-01  1.41535699e-01 -1.29253221e+00
  1.33586717e+00  1.43089890e+00 -7.10735202e-01  3.68400872e-01
  3.47683936e-01  1.00279522e+00 -9.98684585e-01  9.28291202e-01
  1.26067567e+00 -1.19633090e+00 -1.00435853e+00 -6.27937138e-01
  1.02863836e+00 -1.45649803e+00 -1.00423229e+00 -1.11265707e+00
 -1.33893037e+00  8.99451971e-01  8.59849989e-01  8.97351146e-01
  1.95721543e+00 -2.38950324e+00  6.98350430e-01 -2.54332781e+00
 -7.80221343e-01  1.34824312e+00 -1.30275965e+00  4.19737220e-01
  7.49867558e-01  1.43526709e-02 -3.36812735e-01  6.33926690e-01
 -1.28516543e+00 -9.78036046e-01 -9.70386505e-01  1.48103321e+00
  1.77720714e+00 -6.30774140e-01  6.73609793e-01  1.51666775e-01]
```

```
[-0.2927009  -2.3539     -3.082364   -2.0755804  -0.23387048  0.03116352
 -3.4240723   0.6804483   2.5297556   1.4924724  -2.4403913  -0.30286705
  3.3442798  -0.45448813 -0.8306715  -0.03494536  0.31276062  1.6929202
 -3.229696   -2.8762164   3.9146178  -0.32805213  0.6030931  -0.88103676
  1.0879899  -1.7850647   2.5750897  -1.7574708  -4.274682   -0.62027305
 -1.0076166   0.8643299  -0.8952144   3.1832883   3.9998395   3.2029138
 -1.5560957   0.27520418 -0.966296    1.0861672  -0.8834181  -2.5094414
 -1.9215351   1.0475558  -2.3829036  -0.17971425 -2.8454487   4.4128757
  0.23126438 -0.58600175  2.3757505  -0.08426343  0.6372437  -2.4128644
 -1.5134279  -0.44377014 -0.18394051 -3.542668   -2.3582625   1.5922571
  0.30302525  3.3288202   0.4322284  -0.3159594   2.7113795  -4.014404
  2.4380522  -2.5136967   3.5966437   3.757738    2.2040672  -0.1174969
 -0.42089543 -2.2368116  -0.10117842  2.6468835   2.606288   -1.93117
 -2.2637906  -1.4581017   0.891239    0.47737953  0.6579282   2.549008
 -0.7854796  -5.4956417  -0.9731244  -0.50446683 -0.5956649  -1.3335679
  3.1565733  -0.6824405   1.283595    2.0075195   1.0156357   0.34581745
 -2.6737473   1.6112593   2.4512193   0.465902  ]
```

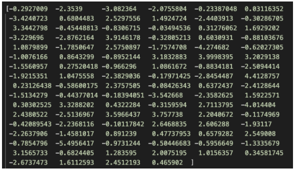
```
[ 3.4993453  -0.75153375  2.7960842   1.0670502   1.7269794   2.849036
  2.3116152   0.28173736  0.7484232   1.8855542  -0.17695513  0.75033337
  3.0245388   2.2880986   0.62054646  3.8393073   1.3523717   1.1088774
  0.4298605   0.68544114  0.05531956  0.02740468 -0.06775122  0.7071461
  0.6200045   1.6111546  -1.6065401   0.29958934  2.1255572   3.9588451
  2.2673967  -2.420242   -1.5391635   1.5550674   3.338346    2.6921241
  4.594245    1.6169097   4.448282   -3.3279974   1.5572077   1.1702842
  0.8155618  -2.965235    1.2714497  -0.23951891  3.8141346  -0.52887446
 -1.9785266  -3.3365355  -2.4357328  -0.03013381  5.2083526  -0.47481996
 -2.44593     1.2577963  -1.8394886  -0.06880338  1.7498951  -1.2576197
  1.3084195  -2.5954428  -2.5381258  -1.4663447  -1.3752079  -1.2812922
  2.5025816   2.6445158  -0.25550297 -1.6185617  -0.34789237 -0.43534958
 -0.18697618  1.2172858  -3.026618   -1.9473408  -2.2698424   1.3054887
 -2.340305    4.3331575   0.38164425 -2.977643    2.29641    -7.2448053
 -0.6058418  -1.5438882   0.6236802   6.0693827   0.98486584  1.1576533
 -0.49797067 -1.5955713  -3.2062447  -0.39012623  0.4121893   2.5176299
  2.6070173  -1.0555307   3.642642    1.1389123 ]
```

quod erat demonstrandum

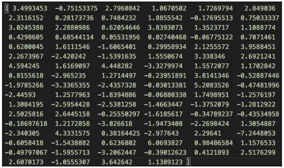
After a thorough training of your model(s) we are excited to see how 'good' our embeddings are. So, we inspect a few of them.



```
[ 1.09798884e+00  6.61867380e-01 -7.10478052e-02  1.08630925e-01
  1.90411139e+00 -1.34148002e-01  1.13010979e+00 -1.08128226e+00
  5.67232482e-02 -1.53587058e-01 -4.61597502e-01  2.20323277e+00
 -2.83151940e-02 -1.61308491e+00 -7.17172861e-01  1.03687203e+00
  9.62423146e-01 -1.09708142e+00 -2.44244242e+00  8.22242856e-01
 -5.69651365e-01 -1.57382798e+00  1.92259073e+00  1.61573005e+00
 -1.87173045e+00 -4.40778702e-01  8.33371878e-01 -3.92222732e-01
 -5.92744529e-01  8.38984668e-01  8.04344654e-01  1.07055962e-01
 -2.29991511e-01 -6.51594162e-01 -8.89630079e-01 -1.25787973e+00
 -2.66330659e-01 -8.97910058e-01 -1.34007168e+00 -1.63498962e+00
  2.19974113e+00  3.55621070e-01  1.38610697e+00  4.99659151e-01
  1.83429384e+00  1.45139420e+00  1.27718091e+00 -5.23355544e-01
 -1.48872185e+00 -8.93208265e-01 -2.76439548e-01 -1.15471005e+00
  5.90652406e-01 -1.46208286e+00 -1.07289457e+00  2.24772835e+00
 -6.43072533e-04 -8.49872053e-01  1.41535699e-01 -1.29253221e+00
  1.33586717e+00  1.43089890e+00 -7.10735202e-01  3.68400872e-01
  3.47683936e-01  1.00279522e+00 -9.98684585e-01  9.28291202e-01
  1.26067567e+00 -1.19633090e+00 -1.00435853e+00 -6.27937138e-01
  1.02863836e+00 -1.45649803e+00 -1.00423229e+00 -1.11265707e+00
 -1.33893037e+00  8.99451971e-01  8.59849989e-01  8.97351146e-01
  1.95721543e+00 -2.38950324e+00  6.98350430e-01 -2.54332781e+00
 -7.80221343e-01  1.34824312e+00 -1.30275965e+00  4.19737220e-01
  7.49867558e-01  1.43526709e-02 -3.36812735e-01  6.33926690e-01
 -1.28516543e+00 -9.78036046e-01 -9.70386505e-01  1.48103321e+00
  1.77720714e+00 -6.30774140e-01  6.73609793e-01  1.51666775e-01]
```



```
[-0.2927009  -2.3539     -3.082364   -2.0755804  -0.23387048  0.03116352
 -3.4240723   0.6804483   2.5297556   1.4924724  -2.4403913  -0.30286705
  3.3442798  -0.45448813 -0.8306715  -0.03494536  0.31276062  1.6929202
 -3.229696   -2.8762164   3.9146178  -0.32805213  0.6030931  -0.88103676
  1.0879899  -1.7850647   2.5750897  -1.7574708  -4.274682   -0.62027305
 -1.0076166   0.8643299  -0.8952144   3.1832883   3.9998395   3.2029138
 -1.5560957   0.27520418 -0.966296    1.0861672  -0.8834181  -2.5094414
 -1.9215351   1.0475558  -2.3829036  -0.17971425 -2.8454487   4.4128757
  0.23126438 -0.58600175  2.3757505  -0.08426343  0.6372437  -2.4128644
 -1.5134279  -0.44377014 -0.18394051 -3.542668   -2.3582625   1.5922571
  0.30302525  3.3288202   0.4322284  -0.3159594   2.7113795  -4.014404
  2.4380522  -2.5136967   3.5966437   3.757738    2.2040672  -0.1174969
 -0.42089543 -2.2368116  -0.10117842  2.6468835   2.606288   -1.93117
 -2.2637906  -1.4581017   0.891239    0.47737953  0.6579282   2.549008
 -0.7854796  -5.4956417  -0.9731244  -0.50446683 -0.5956649  -1.3335679
  3.1565733  -0.6824405   1.283595    2.0075195   1.0156357   0.34581745
 -2.6737473   1.6112593   2.4512193   0.465902  ]
```



```
[ 3.4993453  -0.75153375  2.7960842   1.0670502   1.7269794   2.849036
  2.3116152   0.28173736  0.7484232   1.8855542  -0.17695513  0.75033337
  3.0245388   2.2880986   0.62054646  3.8393073   1.3523717   1.1088774
  0.4298605   0.68544114  0.05531956  0.02740468 -0.06775122  0.7071461
  0.6200045   1.6111546  -1.6065401   0.29958934  2.1255572   3.9588451
  2.2673967  -2.420242   -1.5391635   1.5550674   3.338346    2.6921241
  4.594245    1.6169097   4.448282   -3.3279974   1.5572077   1.1702842
  0.8155618  -2.965235    1.2714497  -0.23951891  3.8141346  -0.52887446
 -1.9785266  -3.3365355  -2.4357328  -0.03013381  5.2083526  -0.47481996
 -2.44593     1.2577963  -1.8394886  -0.06880338  1.7498951  -1.2576197
  1.3084195  -2.5954428  -2.5381258  -1.4663447  -1.3752079  -1.2812922
  2.5025816   2.6445158  -0.25550297 -1.6185617  -0.34789237 -0.43534958
 -0.18697618  1.2172858  -3.026618   -1.9473408  -2.2698424   1.3054887
 -2.340305    4.3331575   0.38164425 -2.977643    2.29641    -7.2448053
 -0.6058418  -1.5438882   0.6236802   6.0693827   0.98486584  1.1576533
 -0.49797067 -1.5955713  -3.2062447  -0.39012623  0.4121893   2.5176299
  2.6070173  -1.0555307   3.642642    1.1389123 ]
```

quod   erat   demonstrandum   □

The Evaluation Problem

Problem

How to evaluate ‘goodness’?

The Evaluation Problem

Problem

How to evaluate 'goodness'?

Solution

1. Use as input to some other downstream NLP task

The Evaluation Problem

Problem

How to evaluate 'goodness'?

Solution

1. Use as input to some other downstream NLP task
2. Devise 'evaluation tasks' and test performance

The Evaluation Problem

Problem

How to evaluate ‘goodness’?

Solution

1. Use as input to some other downstream NLP task
2. Devise ‘evaluation tasks’ and test performance

Desirable Attributes

The Evaluation Problem

Problem

How to evaluate 'goodness'?

Solution

1. Use as input to some other downstream NLP task
2. Devise 'evaluation tasks' and test performance

Desirable Attributes

- capture semantic similarity

The Evaluation Problem

Problem

How to evaluate ‘goodness’?

Solution

1. Use as input to some other downstream NLP task
2. Devise ‘evaluation tasks’ and test performance

Desirable Attributes

- capture semantic similarity
- capture syntactic similarity

Odd-One-Out Evaluation

Examples

`pirum`, `pruna`, `baca`, `olea`, `denarius`

pear, **plum**, **berry**, **olive**, **denarius**

Can you pick the odd one out?

Odd-One-Out Evaluation

Examples

`consul, tribunus, praetor, magistris, episcopi`

consul, **tribune**, **praetor**, **magister**, **bishop**

Can you pick the odd one out?

Odd-One-Out Evaluation

Examples

`homines, feminae, liberi, vir, fratres`

men, **women**, **children**, **man**, **brothers**

Can you pick the odd one out?

The Odd-One-Out Task

To play the game with our model, we do the following:

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words
- Form a grouping of k in-words 1 out-word

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words
- Form a grouping of k in-words 1 out-word
- Find the mean of the vectors in the group (the center)

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words
- Form a grouping of k in-words 1 out-word
- Find the mean of the vectors in the group (the center)
- Compute the cosine distance between each word and the center

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words
- Form a grouping of k in-words 1 out-word
- Find the mean of the vectors in the group (the center)
- Compute the cosine distance between each word and the center
- Pick the word with the largest distance as the odd-one-out

The Odd-One-Out Task

To play the game with our model, we do the following:

- Pick 2 categories of words
- Form a grouping of k in-words 1 out-word
- Find the mean of the vectors in the group (the center)
- Compute the cosine distance between each word and the center
- Pick the word with the largest distance as the odd-one-out
- Check to see if chosen word is from the out-category

Top K Similarity

Another strategy: look at what's nearby

Top K Similarity

Another strategy: look at what's nearby

miles	**soldier**
militio	to be a soldier
centurio	centurion
exerceo	train
legio	legion
cohors	cohort/company
militaris	military
dux	leader
cohorto	to exhort
castra	camp
hostis	enemy

Table: Top 10 most similar words

Top K Similarity

Another strategy: look at what's nearby

denarius	**coin** (1/7 oz silver)
talentum	talent
nummus	money
uncia	ounce
sestertius	coin (1/4 of a denarius)
deni	group of ten
centum	one hundred
quinquageni	fifties
viceni	twenties
centeni	hundreds
ducenti	two hundred

Table: Top 10 most similar words

Top K Evaluation

Examples

caesar

Top K Evaluation

Examples

caesar

`miles`	`antonius`	`rex`	`roma`

Top K Evaluation

Examples

caesar

miles | antonius | rex | roma

caesaris | caesarem | caesare | pompeius | antonius | pompeium

Top K Evaluation

Examples

caesar

`miles` | `antonius` | `rex` | `roma`

`caesaris` | `caesarem` | `caesare` | `pompeius` | `antonius` | `pompeium`

Scores

Top K Evaluation

Examples

caesar

`miles` | `antonius` | `rex` | `roma`

`caesaris` | `caesarem` | `caesare` | `pompeius` | `antonius` | `pompeium`

Scores

category-in-topk accuracy $= \frac{1}{6}$

Top K Evaluation

Examples

caesar

`miles` | `antonius` | `rex` | `roma`

`caesaris` | `caesarem` | `caesare` | `pompeius` | `antonius` | `pompeium`

Scores

category-in-topk accuracy $= \frac{1}{6}$

topk-in-category accuracy $= \frac{1}{4}$

Top K Evaluation

To play the top k game

Top K Evaluation

To play the top k game

- Create a category of similar words

Top K Evaluation

To play the top k game

- Create a category of similar words
- Pick one word from the category and find the top k closest vectors

Top K Evaluation

To play the top k game

- Create a category of similar words
- Pick one word from the category and find the top k closest vectors
- Find matches by comparing top k words to the unused words in our category

Top K Evaluation

To play the top k game

- Create a category of similar words
- Pick one word from the category and find the top k closest vectors
- Find matches by comparing top k words to the unused words in our category
- Calculate a score by dividing by either

Top K Evaluation

To play the top k game

- Create a category of similar words
- Pick one word from the category and find the top k closest vectors
- Find matches by comparing top k words to the unused words in our category
- Calculate a score by dividing by either
    - k

Top K Evaluation

To play the top k game

- Create a category of similar words
- Pick one word from the category and find the top k closest vectors
- Find matches by comparing top k words to the unused words in our category
- Calculate a score by dividing by either
    - k
    - the size of the category - 1

From Tasks to Accuracy Scores

Problem

How to test the model as a whole?

From Tasks to Accuracy Scores

Problem

How to test the model as a whole?

Solution

We construct a test set of words separated into categories of interest. Those categories form the basis for performing our two evaluation tasks.

From Tasks to Accuracy Scores

Problem

How to test the model as a whole?

Solution

We construct a test set of words separated into categories of interest. Those categories form the basis for performing our two evaluation tasks.

religious titles	religious figures	countries	cities
papam	ambrosius	hispania	roma
archiepiscopus	augustini	gallia	mediolanum
episcopus	gregorius	italia	byzantium
apostolus	aquinas	germania	carthago
presbyter	hieronymus	graecia	troiae
propheta	eusebius	syria	NA

Table: A portion of my test set

Scores of Interest from my Models

Accuracies for k = 3

- Christian

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815

Scores of Interest from my Models
Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036

Scores of Interest from my Models

Accuracies for k = 3

- Christian
    - odd-one-out = .815
    - topk-in-category = .036
    - Category-in-topk = .053

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019
  - category-in-topk = .029

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019
  - category-in-topk = .029
- Full

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019
  - category-in-topk = .029
- Full
  - odd-one-out = .825

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019
  - category-in-topk = .029
- Full
  - odd-one-out = .825
  - topk-in-category = .052

Scores of Interest from my Models

Accuracies for k = 3

- Christian
  - odd-one-out = .815
  - topk-in-category = .036
  - Category-in-topk = .053
- Medieval
  - odd-one-out = .722
  - topk-in-category = .019
  - category-in-topk = .029
- Full
  - odd-one-out = .825
  - topk-in-category = .052
  - category-in-topk = .075

This is the End

DIXI