MATH 250: Mathematical Data Visualization

Applications: Text Mining

Peter A. Gao

2025-04-23

San José State University

Outline

- Text mining
- Bag of words model
- Word embeddings

Other readings

- Text Mining with R
- Linear Algebra for Data Science Ch. 16
- Speech and Language Processing

Text mining

How do we turn books, websites, and other texts into **data**?

Much of this class has focused on **structured data** (typically organized into tables), but today we will focus on **unstructured data**, such as text data, which often has no fixed length or format.

We will focus on two strategies for converting this unstructured data into structured matrices: the bag of words model and newer word embedding approaches.

Working with text data in R

Before diving into typical models for text data, it may be useful to quickly review tools for working with and summarizing text data in R.

Below, we use the `janeaustenr` package to carry out some basic text processing, following the examples outlined in Text Mining with R.

Basic text processing

```
library(janeaustenr)
library(dplyr)
library(stringr)
austen_books() |> head()
```

```
# A tibble: 6 x 2
  text                    book
  <chr>                   <fct>
1 "SENSE AND SENSIBILITY" Sense & Sensibility
2 ""                      Sense & Sensibility
3 "by Jane Austen"        Sense & Sensibility
4 ""                      Sense & Sensibility
5 "(1811)"                Sense & Sensibility
6 ""                      Sense & Sensibility
```

Basic text processing

```
original_books <- austen_books() |>
  group_by(book) |>
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,
                                     regex("^chapter [\\divxlc]",
                                           ignore_case = TRUE)))) |>
  ungroup()
original_books |> head()
```

```
# A tibble: 6 x 4
  text                    book                linenumber chapter
  <chr>                   <fct>                    <int>   <int>
1 "SENSE AND SENSIBILITY" Sense & Sensibility          1       0
2 ""                      Sense & Sensibility          2       0
3 "by Jane Austen"        Sense & Sensibility          3       0
4 ""                      Sense & Sensibility          4       0
5 "(1811)"                Sense & Sensibility          5       0
6 ""                      Sense & Sensibility          6       0
```

Basic text processing

The `unnest_tokens()` function splits text data into individual tokens, which are typically words:

```
library(tidytext)
tidy_books <- original_books |>
  unnest_tokens(word, text)
tidy_books |> head()
```

```
# A tibble: 6 x 4
  book                linenumber chapter word
  <fct>                    <int>   <int> <chr>
1 Sense & Sensibility          1       0 sense
2 Sense & Sensibility          1       0 and
3 Sense & Sensibility          1       0 sensibility
4 Sense & Sensibility          3       0 by
5 Sense & Sensibility          3       0 jane
6 Sense & Sensibility          3       0 austen
```

Basic text processing

We can remove common **stop words** (common words with limited/no meaning) as follows:

```
data(stop_words)
tidy_books <- tidy_books |>
  anti_join(stop_words)
```

Basic text processing

```
tidy_books |>
  count(word, sort = TRUE)
```

```
# A tibble: 13,914 x 2
   word       n
   <chr>  <int>
 1 miss    1855
 2 time    1337
 3 fanny    862
 4 dear     822
 5 lady     817
 6 sir      806
 7 day      797
 8 emma     787
 9 sister   727
10 house    699
# i 13,904 more rows
```

Bag-of-words model

The bag-of-words model represents text as a set of words (tokens) without paying attention to their order. In other words, under a bag-of-words representation, the phrases “humans like cats” and “cats like humans” are indistinguishable without context.

The bag-of-words representation is obviously simpler than the way text is represented in a large language model. However, for many applications, the bag-of-words model may still be sufficient (and computationally cheap).

`tm`: Text Mining in R

In R, the `tm` package (Feinerer, Hornik, and Meyer 2008) provides an interface to perform many preprocessing tasks for text mining.

In particular, the package contains functions of loading data, **corpus** management, and creating term-document matrices.

A collection of documents is often called a **corpus**: in `tm`, there are two ways to store such data: as a `VCorpus` (volatile corpus) or `PCorpus` (permanent corpus).

Example: *Reuters*-21587

The *Reuters*-21578 dataset is a collection of 21578 documents that appeared on the *Reuters* newswire in 1987 (Lewis 1987).

The `tm.corpus.Reuters21578` package provides this dataset as a VCorpus object for use with `tm`.

```
library(tm)
# install.packages("tm.corpus.Reuters21578",
#                  repos = "http://datacube.wu.ac.at")
library(tm.corpus.Reuters21578)
data(Reuters21578)
```

Example: *Reuters*-21587

```
meta(Reuters21578[[1]])
```

```
  author       : character(0)
  datetimestamp: 1987-02-26 15:01:01
  description  :
  heading      : BAHIA COCOA REVIEW
  id           : 1
  language     : en
  origin       : Reuters-21578 XML
  topics       : YES
  lewissplit   : TRAIN
  cgisplit     : TRAINING-SET
  oldid        : 5544
  topics_cat   : cocoa
  places       : c("el-salvador", "usa", "uruguay")
  people       : character(0)
  orgs         : character(0)
  exchanges    : character(0)
```

Example: *Reuters*-21587

```
library(stringr)
options(width = 75)
strwrap(str_sub(as.character(Reuters21578[[1]]), 1, 600))
```

```
 [1] "Showers continued throughout the week in the Bahia cocoa zone,"
 [2] "alleviating the drought since early January and improving"
 [3] "prospects for the coming temporao, although normal humidity levels"
 [4] "have not been restored, Comissaria Smith said in its weekly"
 [5] "review.  The dry period means the temporao will be late this year."
 [6] "Arrivals for the week ended February 22 were 155,221 bags of 60"
 [7] "kilos making a cumulative total for the season of 5.93 mln against"
 [8] "5.81 at the same stage last year. Again it seems that cocoa"
 [9] "delivered earlier on consignment was included in the arrivals"
[10] "figures.  Comissaria S"
```

Preprocessing

Typically, text data must be preprocessed for optimal results. Below, we remove unnecessary whitespace and convert all words to lowercase.

```
Reuters21578 <- tm_map(Reuters21578,
                       stripWhitespace)
Reuters21578 <- tm_map(Reuters21578,
                       content_transformer(tolower))
```

Removing stop words

Stop words are words that do not hold much semantic meaning, including common words like “to”, “the”, etc. Such words are not necessarily relevant for tasks like categorizing texts. Below, we remove the stop words.

```
Reuters21578 <- tm_map(Reuters21578,
                       removeWords,
                       stopwords("english"))
```

Stemming

Stemming is the process of converting words that may have the same meaning into a common token/stem. For example, s*wimming* and s*wim* may both be treated as s*wim*.

In `tm`, the stemming function requires installing the `SnowballC` package.

Stemming: *Reuters*-21587

```
# requires package SnowballC
Reuters21578 <- tm_map(Reuters21578, stemDocument)
strwrap(str_sub(as.character(Reuters21578[[1]]), 1, 600))
```

```
 [1] "shower continu throughout week bahia cocoa zone, allevi drought"
 [2] "sinc earli januari improv prospect come temporao, although normal"
 [3] "humid level restored, comissaria smith said week review. dri"
 [4] "period mean temporao will late year. arriv week end februari 22"
 [5] "155,221 bag 60 kilo make cumul total season 5.93 mln 5.81 stage"
 [6] "last year. seem cocoa deliv earlier consign includ arriv figures."
 [7] "comissaria smith said still doubt much old crop cocoa still avail"
 [8] "harvest practic come end. total bahia crop estim around 6.4 mln"
 [9] "bag sale stand almost 6.2 mln hundr thousand bag still hand"
[10] "farmers, middlemen, expor"
```

Document-term matrix: *Reuters*-21587

The `DocumentTermMatrix` function can be used to convert a `VCorpus` object into a DTM with n rows and p columns, where n is the number of observations and p is the number of distinct terms.

```
dtm <- DocumentTermMatrix(Reuters21578)
dim(dtm)
```

```
[1] 21578 89935
```

Document-term matrix: *Reuters*-21587

```
inspect(dtm)
```

```
<<DocumentTermMatrix (documents: 21578, terms: 89935)>>
Non-/sparse entries: 1187801/1939429629
Sparsity           : 100%
Maximal term length: 61
Weighting          : term frequency (tf)
Sample             :
       Terms
Docs    bank billion compani dlrs mln pct reuter said said. will
  11224    2       0       0    0   1   2       6    8     5    4
  15871    0       0       0    0   0   0       1    0     0    1
  15875    0       0       0   10   9   0       1    0     0    0
  17396   10       0       0    0   0   8       1   14    12    1
  17474    2       1       0    0   3   5       1    6     4    4
  17953    3       2       0    0   0   0       1    7    10    2
  5214     0       0       1    0  12   2       2   10     9    1
  5985     2       4       7    1   1  11       0   16     8   14
  6657     0       1       0    1   8   0       1    7     5    2
  7135     1       9       0    4   1   6       1   12     2    9
```

Document-term matrix: *Reuters*-21587

```
options(width = 60)
inspect(removeSparseTerms(dtm, 0.99))
```

```
<<DocumentTermMatrix (documents: 21578, terms: 941)>>
Non-/sparse entries: 709012/19595886
Sparsity           : 97%
Maximal term length: 13
Weighting          : term frequency (tf)
Sample             :
       Terms
Docs    bank billion compani dlrs mln pct reuter said said.
  11083    2       0       0    1   0  22      1    8     2
  11224    2       0       0    0   1   2      6    8     5
  15875    0       0       0   10   9   0      1    0     0
  17396   10       0       0    0   0   8      1   14    12
  17953    3       2       0    0   0   0      1    7    10
  4944     0       0       0    2   2  19      1   14     8
  5214     0       0       1    0  12   2      2   10     9
  5985     2       4       7    1   1  11      0   16     8
  6657     0       1       0    1   8   0      1    7     5
  8746     1       2       0    0   0  12      1    8     5
       Terms
```

TF-IDF: *Reuters*-21587

The `weighting` argument allows us to specify TF-IDF weighting for the entries of the DTM.

```
dtm_tfidf <-
  DocumentTermMatrix(Reuters21578,
                     control = list(weighting = weightTfIdf))
dim(dtm_tfidf)
```

```
[1] 21578 89935
```

TF-IDF: *Reuters*-21587

```
options(width = 60)
inspect(removeSparseTerms(dtm_tfidf, 0.99))
```

```
<<DocumentTermMatrix (documents: 21578, terms: 941)>>
Non-/sparse entries: 709012/19595886
Sparsity           : 97%
Maximal term length: 13
Weighting          : term frequency - inverse document frequency (norma
Sample             :
       Terms
Docs    billion       cts dlrs loss mln net       pct rev
  15474       0 0.1336806    0    0   0   0 0.00000000   0
  17835       0 0.0000000    0    0   0   0 0.00000000   0
  19196       0 0.4861114    0    0   0   0 0.00000000   0
  19309       0 0.3145427    0    0   0   0 0.00000000   0
  19376       0 0.5347225    0    0   0   0 0.00000000   0
  19396       0 0.4113250    0    0   0   0 0.00000000   0
  20242       0 0.5347225    0    0   0   0 0.00000000   0
  20993       0 0.2056625    0    0   0   0 0.00000000   0
```

TF-IDF: *Reuters*-21587

```
options(width = 60)
# find terms with at least 0.25 correlation with "war"
findAssocs(dtm_tfidf, "war", 0.25)
```

```
$war
     iraq   iranian      iran     iraqi    offens    "crush
     0.41      0.39      0.34      0.34      0.32      0.30
   blow."     kamal kharrazi,  safely,"     swamp     aveng
     0.30      0.30      0.30      0.30      0.30      0.28
   front.      irna     front
     0.28      0.28      0.25
```

Classification: *Reuters*-21587

```
options(width = 60)
table(unlist(meta(Reuters21578, "topics_cat"))) |>
  sort(decreasing = T) |> head(20)
```

```
        earn          acq     money-fx         crude
        3987         2448          801           634
       grain        trade     interest         wheat
         628          552          513           306
        ship         corn          dlr      oilseed
         305          254          217           192
money-supply        sugar          gnp       coffee
         190          184          163           145
     veg-oil         gold      nat-gas      soybean
         137          135          130           120
```

Classification: *Reuters*-21587

The *Reuters* data includes topic labels. Suppose we focus on two topics with a substantial number of instances: `crude` (for crude oil) and `money-fx` (for exchange rates).

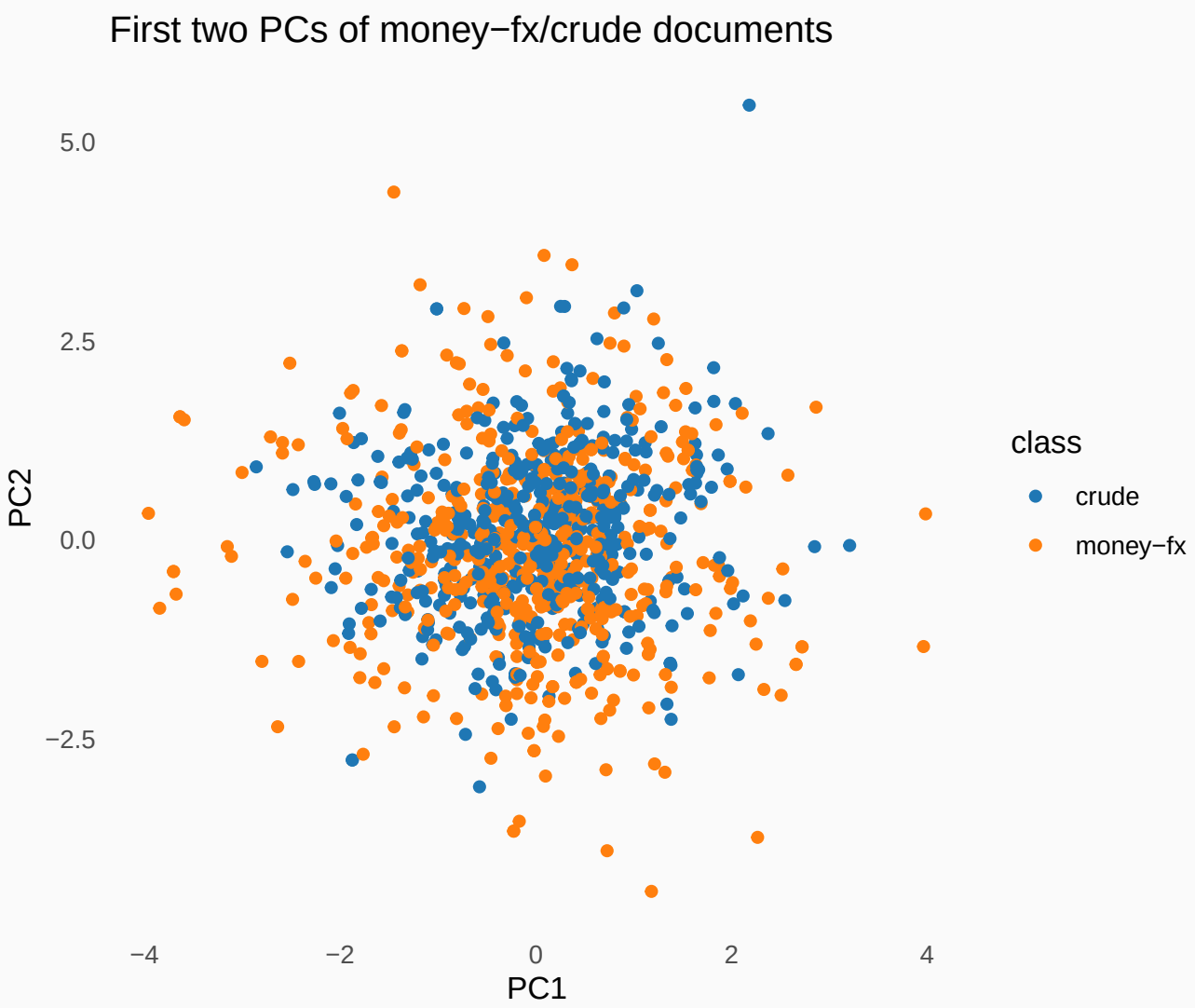
```
idx <- sapply(meta(Reuters21578, "topics_cat"), function(x) x[1])
idx <- idx %in% c("crude", "money-fx")
reuters_ex <- Reuters21578[idx]
```

Classification: *Reuters*-21587

```
# create tf-idf matrix
class_tfidf <-
  DocumentTermMatrix(reuters_ex,
                     control = list(weighting = weightTfIdf))
# remove terms that appear in fewer than 5% of documents
class_tfidf <- removeSparseTerms(class_tfidf, 0.95)

# create vector of first class labels
class_vec <- sapply(meta(reuters_ex, "topics_cat"),
                    function(x) x[1])
```

Classification: *Reuters*-21587

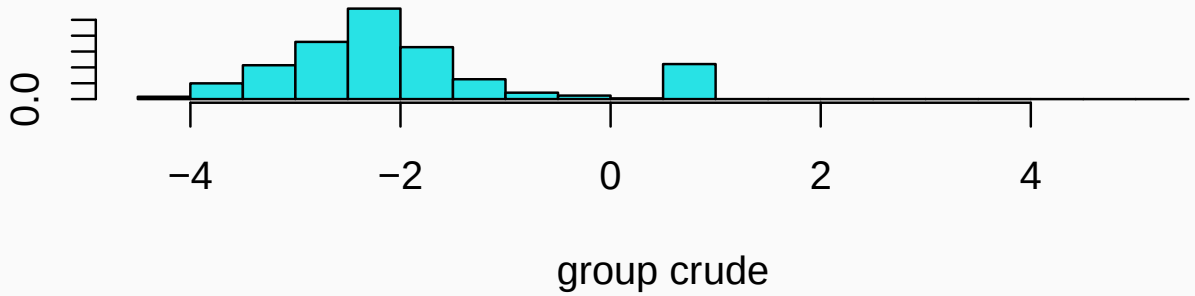


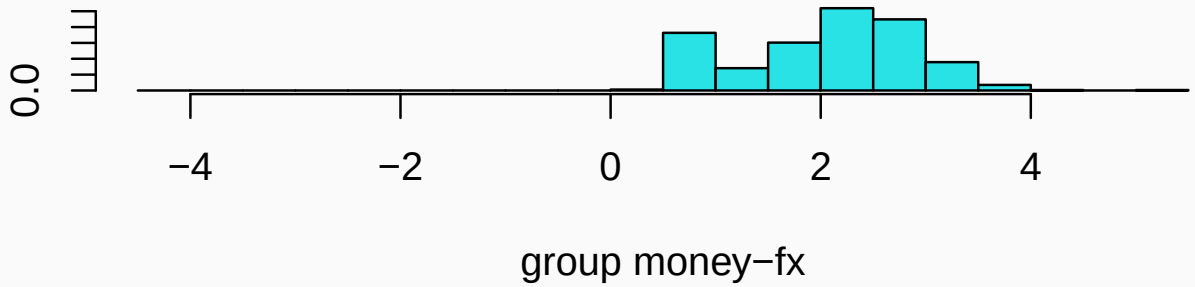
Classification: *Reuters*-21587

```
pca_res <- prcomp(class_tfidf, scale = T)
plot_dat <- data.frame(PC1 = pca_res$x %*% pca_res$rotation[, 1],
                       PC2 = pca_res$x %*% pca_res$rotation[, 2],
                       class = class_vec)
library(tidyverse)
library(ggsci)
ggplot(plot_dat, aes(x = PC1, y = PC2, color = class)) +
  geom_point() +
  theme_minimal() +
  scale_color_d3(name = "class", scale_name = "category10") +
  ggtitle("First two PCs of money-fx/crude documents") +
  xlab("PC1") + ylab("PC2") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Classification: *Reuters*-21587

```
library(MASS)
lda_res <- lda(class_tfidf, grouping = class_vec)
plot(lda_res)
```





Classification: *Reuters*-21587

```
lda_res$scaling |> as.data.frame() |> arrange(LD1) |> head(5)
```

```
               LD1
said     -21.38227
oil      -18.83797
industri -11.17192
forc     -10.98521
said.    -10.63338
```

```
lda_res$scaling |> as.data.frame() |> arrange(-LD1) |> head(5)
```

```
               LD1
monetari 14.896443
currenc  12.960497
dollar   12.208735
treasuri 10.769660
exchang   9.636657
```

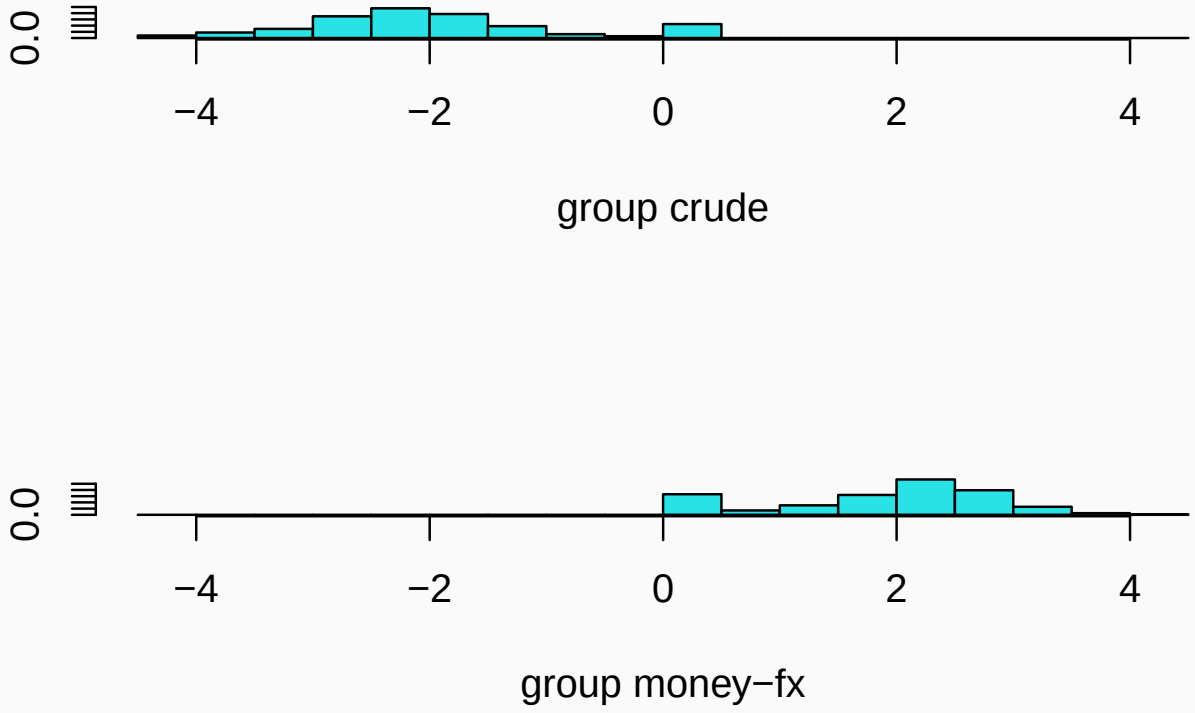
Classification: *Reuters*-21587

Recall that we can also first use PCA to reduce the dimension of our data before applying LDA.

```
library(MASS)
pca_95_idx <- which(cumsum(pca_res$sdev^2) /
                      sum(pca_res$sdev^2) < .95)
pca_dat <- as.data.frame(pca_res$x[, pca_95_idx]) |>
  mutate(label = class_vec)
lda_res <- lda(label ~., pca_dat)
plot(lda_res)
```

Classification: *Reuters*-21587

Recall that we can also first use PCA to reduce the dimension of our data before applying LDA.



N-grams

The bag-of-words model cannot be used directly to **generate** text as it does not consider word order. **Language models** are models that can be used to predict words or sentences in order.

A simple language model is the n-gram model. An **n-gram** is a sequence of words of length n.

For example, a 2-gram (bigram) is a sequence of two words (“thank you”) while a 3-gram (trigram) is a sequence of three words (“please be seated”).

N-grams

The typical strategy is to model $P(X_n \mid X_1, \dots, X_{n-1})$, the probability of word X_n given previous words $X_1, \dots, X_{n-1}$:

$$P(X_1, \dots, X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_1, X_2) \cdots P(X_n \mid X_1, \dots, X_{n-1})$$

The bigram model simplifies this model by approximating the history using only the last word:

$$P(X_1, \dots, X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_2) \cdots P(X_n \mid X_{n-1})$$

For more on the N-gram model, you can review Chapter 3 of Speech and Language Processing

Word embeddings

While n-gram models capture correlation between sequential words, they do not encode any semantic information, or information on the meaning of the words.

Language models increasingly rely on word embeddings, which represent words as vectors in a vector space. This vector space representation of words/tokens captures relationships between words.

Such methods have gradually led to some of the recent developments in large language model-powered tools like ChatGPT.

Word embeddings

Note that document-term matrices also induces a representation of words (and documents) as vectors. This vectors are typically sparse and long (length being the number of unique words/tokens or documents).

Today, the term **word embeddings** typically refers to methods that represent words as shorter, dense vectors (and the entries typically do not represent frequencies/counts).

Word similarity and relatedness

Word embeddings are vector representations of the meaning of words.

Given large amounts of unstructured (text) data, word embedding algorithms provide a “self-supervised” way to learn these vector representations.

How do we define similarity between words? One approach is to simply ask people whether two words have the same or similar meanings.

Vector semantics

Word embedding methods typically define a word's meaning by studying the distribution of its neighboring words.

Words that frequently appear in similar contexts are treated as having similar meanings.

For example, *tomatoes* and *carrots* may both appear frequently near words like *salad* or *eat*.

Vector semantics

These methods represent each word as a point in some multidimensional vector space.

The individual coordinate values may not be interpretable: the embedding derives its usefulness by capturing relationships between words.

Speech and Language Processing, Figure 6.1

Semantle

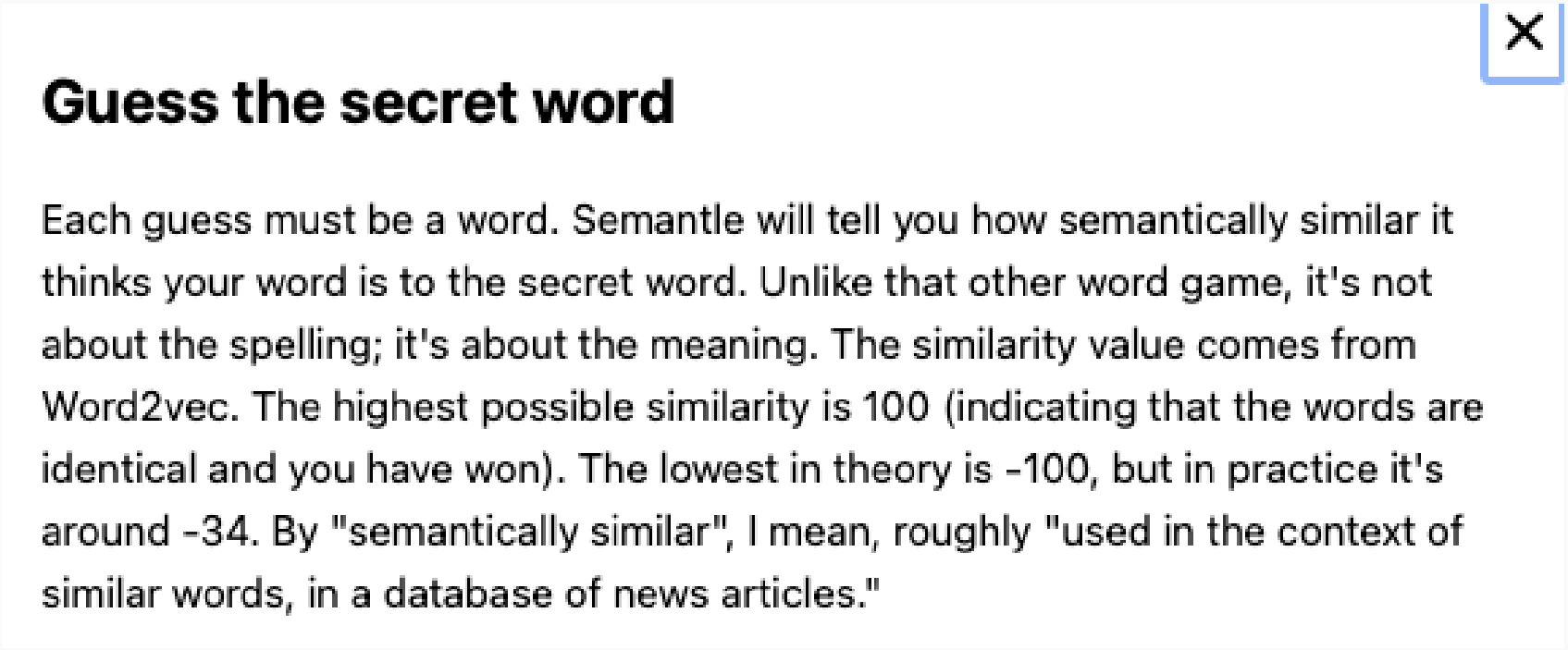


Figure 1: Semantle

Word2vec

Word2vec is a software package that includes two methods for computing word embeddings. One of these methods is **skip-gram with negative sampling**.

This is an example of a static embedding, so this method yields one fixed vector representation of each word in the vocabulary.

Intuitively, the strategy is to train a classifier on the binary prediction task: is word a likely to appear near word b?

Word2vec

Let w be a target word (ex. *water*). Suppose we define two words as being "near" each other if they are adjacent or one word apart.

the elephant [drank the water in the] lake

How do we get training data? For any word w, we need both **positive** cases (words that appear near w) and **negative** cases (words that do not appear near w).

Word2vec

The general strategy is as follows:

1. Based on existing text, identify the target word and its neighboring words as positive cases.
2. Randomly sample other words in the vocabulary to get negative cases.
3. Train a binary classifier (using logistic regression). Use the estimated model parameters as an embedding.

Word2vec

Given a target word w and proposed context word c, we use logistic regression to estimate $P(Y = 1 \mid w, c)$, where $Y = 1$ represents the event that c is a context word for w.

To model this probability, we define $\mathbf{w}$ to be an embedding vector for word w and $\mathbf{c}$ to be an embedding vector for word c.

The probability that c is a true context word for w should be higher if the two vectors are "similar."

Word2vec

We quantify similarity using the dot product

$$\mathbf{c} \cdot \mathbf{w} = |\mathbf{c}||\mathbf{w}| \cos \theta$$

Then we model the probability as follows:

$$P(Y = 1 \mid w, c) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

This yields the probability that c is a context word for w.

Word2vec

Given a window of L context words, our goal will be to select w to maximize the probability

$$P(Y = 1 \mid w, c_{1:L}) = \prod_{i=1}^{L} \frac{1}{1 + \exp(-\mathbf{c}_i \cdot \mathbf{w})}$$

Estimating vector representations

How do we estimate $\mathbf{w}$ and $\mathbf{c}$? The skip-gram method actually estimates two vector representations of each word, one representation as a target word, and one as a context word.

Given a body of text and a vocabulary size, word2vec randomly initializes an embedding for each word.

A unified loss function based on the complete corpus is minimized via gradient descent, gradually moving dissimilar words apart and similar words together.

Word2vec example: Reuters

```
library(word2vec)
reuters_ex_content <- sapply(reuters_ex, function(x) x$content)
reuters_ex_content <- tolower(reuters_ex_content)
w2v_model <-
  word2vec(reuters_ex_content,
           type = "skip-gram",
           dim = 15, iter = 20)
embedding <- as.matrix(w2v_model)
```

Word2vec example: Reuters

```
predict(w2v_model, "dollar", type = "nearest", top_n = 5)
```

```
$dollar
   term1      term2 similarity rank
1 dollar        yen  0.9669455    1
2 dollar undervalu  0.9473589    2
3 dollar        148  0.9442537    3
4 dollar        149  0.9434962    4
5 dollar     weaken  0.9424641    5
```

```
predict(w2v_model, "markets", type = "nearest", top_n = 5)
```

```
$markets
    term1     term2 similarity rank
1 markets   continu  0.9384213    1
2 markets moreover  0.9273792    2
3 markets   depreci  0.9242573    3
4 markets   financi  0.9211671    4
5 markets concentr  0.9177634    5
```

Word2vec example: Reuters

```
options(width = 60)
rownames(embedding)[1:10]
```

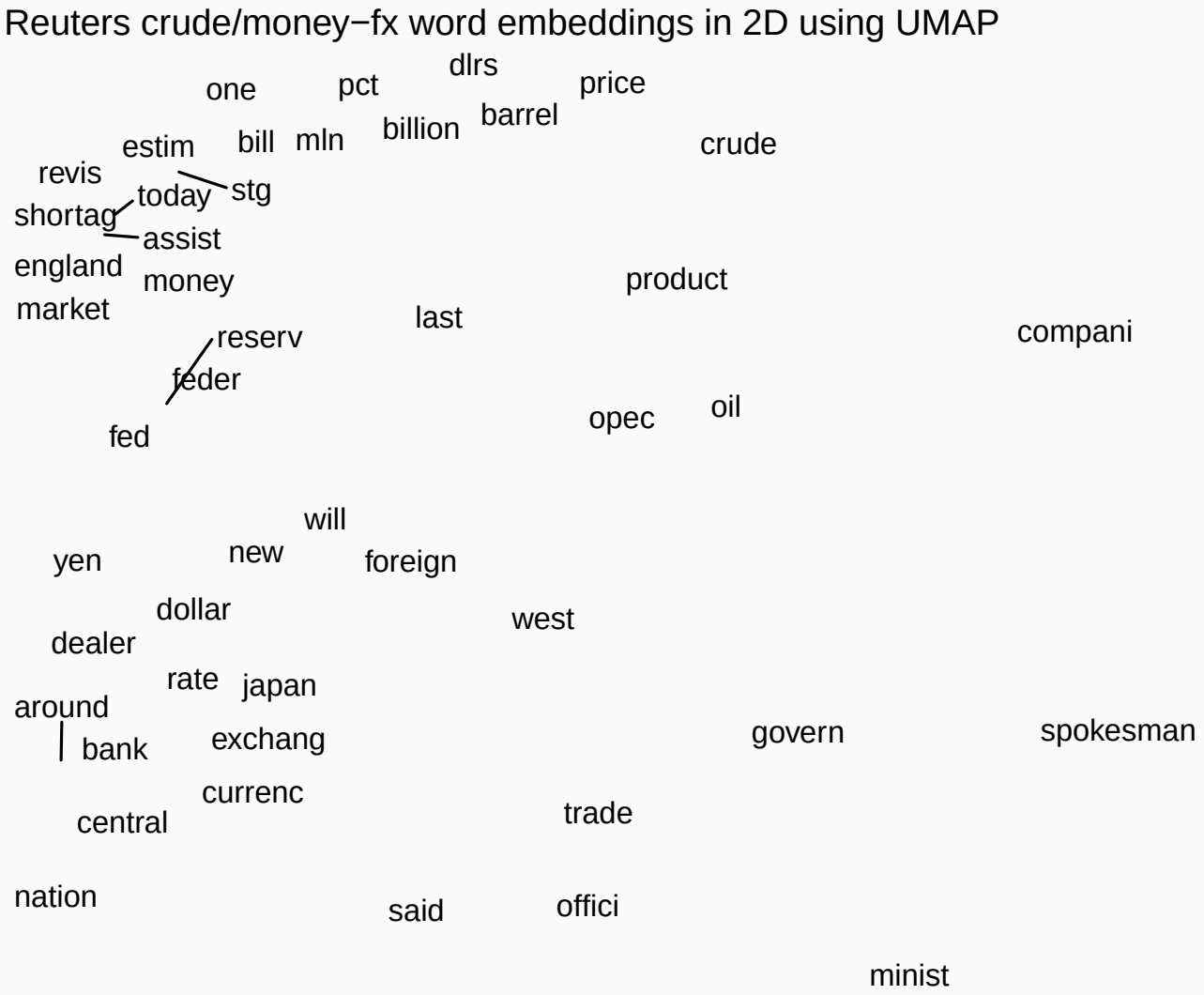
```
 [1] "round"          "worldwide"      "bpd"
 [4] "administration" "event"          "bahrain"
 [7] "auction"        "per"            "brother"
[10] "probabl"
```

Word2vec example: Reuters

```
options(width = 60)
dollar_emb <- predict(w2v_model, c("dollar"), type = "embedding")
dollar_emb
```

```
            [,1]         [,2]       [,3]    [,4]      [,5]
dollar 0.9840165 0.0008567185 -0.1715956 1.06186 0.3715606
             [,6]      [,7]      [,8]      [,9]     [,10]
dollar -0.9459425 0.7405589 0.6088247 0.6953234 -1.347762
          [,11]      [,12]     [,13]     [,14]      [,15]
dollar 1.162594 -0.6376502 0.8105136 -2.486393 -0.1597338
```

Word2vec example: Reuters



Word2vec example: Reuters

```
library(umap)
library(ggrepel)
umap_emb <- umap(embedding, method = "naive")
freq_terms <- colSums(as.matrix(class_tfidf)) |>
  sort(decreasing = T) |>
  names() |>
  head(50)
freq_terms_idx <- match(freq_terms, rownames(embedding))
plot_dat <- data.frame(word = freq_terms,
                       x = umap_emb$layout[freq_terms_idx, 1],
                       y = umap_emb$layout[freq_terms_idx, 2])
ggplot(plot_dat, aes(x = x, y = y, label = word)) +
  geom_text_repel() + theme_void() +
  labs(title = "Reuters crude/money-fx embeddings in 2D")
```

Transformers and LLMs

In the past five years, large language models, which are **pretrained** on large amounts of text, have emerged and outperformed previous methods on many natural language tasks.

Like word2vec, LLMs rely on vector embeddings of word meanings.

While the details of LLMs are beyond the scope of this course, we conclude with a brief discussion of the key ideas and how they relate to dimension reduction.

Autoencoders

Recall that when we apply PCA to a data matrix X, we project the data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to a lower dimensional representation $\mathbf{y}_1, \ldots, \mathbf{y}_n$. We can call this transformation $f_e : \mathbb{R}^p \to \mathbb{R}^q$ the **encoder:**

$$\mathbf{y}_i = f_e(\mathbf{x}_i)$$

At the same time, we learn another linear mapping $f_d : \mathbb{R}^q \to \mathbb{R}^p$ that maps principal components back to the original vector space. We can call this the **decoder**.

Autoencoders

The overall reconstruction function is

$$r(\mathbf{x}) = f_d(f_e(\mathbf{x}))$$

Note that when we keep $k < \text{rank}(X)$ dimensions, $r(\mathbf{x}) \neq \mathbf{x}$.

PCA minimizes the loss in the 2-norm:

$$\mathcal{L}(\theta) = ||r(\mathbf{x}) - \mathbf{x}||_2^2$$

However, we can consider different choices of $f_e, f_d, \mathcal{L}$.

These yield different **autoencoders**. PCA is closely related to simple autoencoders.

A brief introduction to neural networks

We will briefly discuss neural networks. Suppose we wish to model the relationship between inputs $x_1, \dots, x_n$ and output y (typically for a classification/prediction task).

One way to combine information from all the inputs is via a weighted sum

$$z = \mathbf{w}^\top \mathbf{x} + b$$

where $\mathbf{w}$ is a **weight vector** and b is a scalar bias term.

A brief introduction to neural networks

Instead of modeling $y = \mathbf{w}^\top \mathbf{x} + b$ directly, we can apply a non-linear transformation (or **activation function**):

$$y = g(z)$$

For example, some possible activation functions are the sigmoid

$$y = \frac{1}{1 + \exp(-z)}$$

or rectified linear unit (ReLU)

$$y = \max(z, 0)$$

Unlike typical regression methods used in statistics, note that this strategy is deterministic: we do not assume any distribution on y.

A single neuron

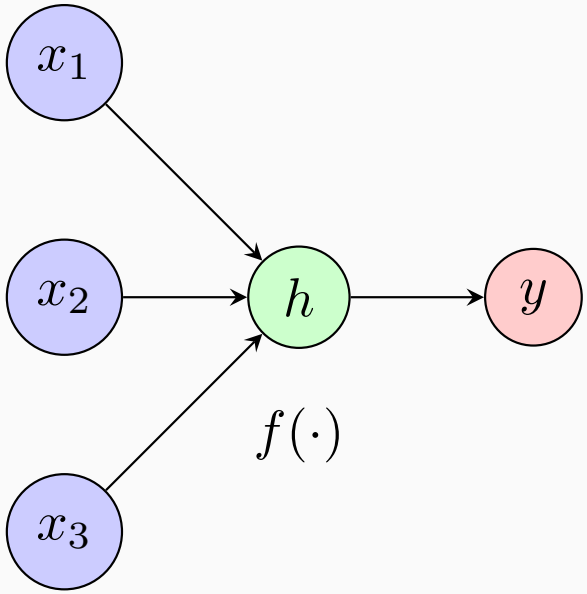


Figure generated by ChatGPT.

A brief introduction to neural networks

Note that we are limited by the shape of the non-linear transformation we use. What if the selected transformation $g(\mathbf{x}; \mathbf{w}, b)$ is a bad approximation of the true relationship between y and $\mathbf{x}$?

Instead of constructing a single linear combination z and then applying the nonlinear transformation, neural networks create multiple such linear combinations $z_1, \dots, z_k$, yielding features $g(z_1), \dots, g(z_k)$. These features can then be used to model outputs $y_1, \dots y_m$.

A single hidden layer

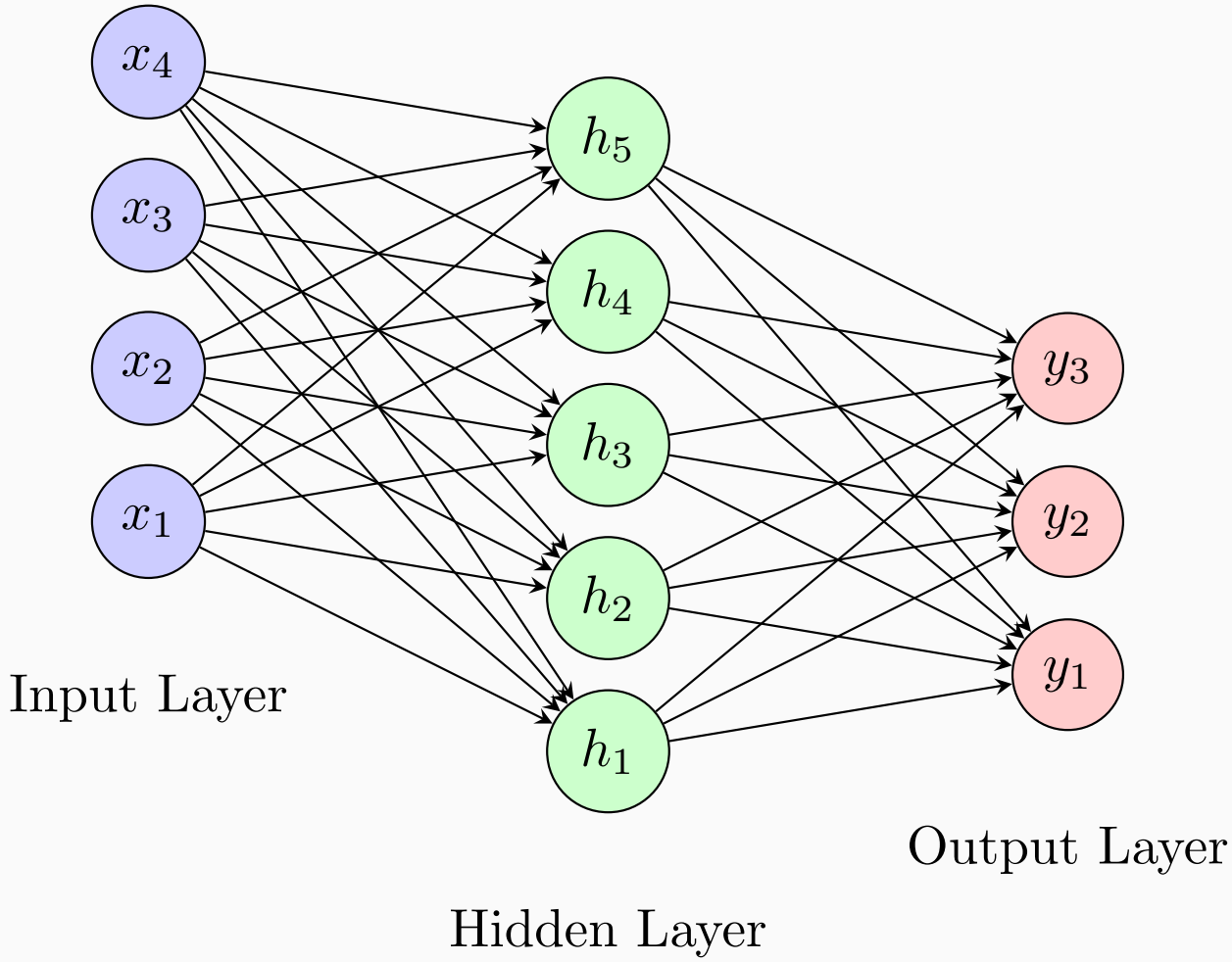


Figure generated by ChatGPT.

A brief introduction to neural networks

For a simple classification problem, a neural network with one hidden layer is analogous to multinomial logistic regression, where the input data are not the original data, but instead the **features** that are developed in the hidden layer.

Instead of developing the features by hand, the neural network is designed to automatically learn useful features which can then be used for classification.

By introducing additional hidden nodes/layers, it is possible to build arbitrarily flexible models for the relationship between outputs and inputs.

Estimation via backpropagation

After specifying the neural network's **architecture** (number of layers and nodes) we can estimate the bias parameters and weight vectors (as well as any other parameters for the activation functions) via **backpropagation**.

Loosely speaking, we specify a loss function based on the outputs of the neural network. Backpropagation approximates the gradient of the loss function (with respect to the weight parameters) iteratively, computing the gradient layer by layer.

Gradient descent algorithms can then be used to obtain parameter estimates.

Transformers

Transformers are a type of neural network architecture that are designed to represent a set of inputs (words) in terms of their contexts.

As opposed to a static embedding method like Word2vec, the embeddings proposed by a transformer are designed to represent contextualized meanings: i.e. what is the meaning of a word in its particular context?

For example, consider the following examples:

1. The **keys** to the car **are** in my pocket.
2. **The chicken** crossed the road because **it** wanted to get to the other side.

These examples indicate that words have contextual meanings. How do we represent the linkage between **the chicken** and **it**?

Attention

Transformers extend neural networks using a mechanism called attention, which provides a way for the network to represent the importance of a particular word in its context. This helps develop a contextualized represntation of the word's meaning.

Transformers are neural netowrks that incorporate **self-attention** layers.

For more on transformers, see Speech and Language Processing, Ch. 10] or you may enjoy this video from 3blue1brown.

Attention

Speech and Language Processing, Figure 10.1

Transformer embeddings

Transformers produce an initial vector representation for each word. Transformer layers then transform these embeddings, producing contextualized embeddings related to how words are used.

Different input sentences thus yield different embeddings, which can then be used to generate predictions and new text.

BERT, GPT-3, and more

Today, many LLMs are built upon these transformer architectures, achieving the best performance yet on many text generation tasks.

Tools like Gemini and ChatGPT provide user-friendly interfaces to the pre-trained transformers, which embed the user input and use the generated embeddings to predict the best output.

For information on ChatGPT's embeddings, you can see their documentation here.

Retrieval augmented generation

A popular new use of LLM embeddings is **retrieval augmented generation (RAG)**. For tasks such as document retrieval, an LLM can be used to embed the **query** and the **documents** and then find the documents whose vector representations are closest to the query vector.

These documents can then be used to produce user-interpretable output.

RAG has become popular for producing more reliable output (using information from verified sources rather than purely LLM output).

Potential concerns with using LLMs

- Privacy violations and data leakage
- Hallucination/fabrication
- Toxic language
- Bias amplification
- Lack of interpretability

Dimension reduction for visualizing embeddings

- Embedding Projector
- Latent Scope

References

Feinerer, Ingo, Kurt Hornik, and David Meyer. 2008. “Text Mining Infrastructure in R.” *Journal of Statistical Software* 25 (March): 1–54. https://doi.org/10.18637/jss.v025.i05.

Lewis, David. 1987. “Reuters-21578 Text Categorization Collection.” [object Object]. https://doi.org/10.24432/C52G6M.