MAT 167: Applied Linear Algebra
Lecture 22: Text Mining

Naoki Saito

Department of Mathematics
University of California, Davis

May 19 & 22, 2017

Outline

Outline

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

What Is Text Mining?

- Text mining = Methods for extracting useful information from large and often unstructured collections of texts.
- It is also closely related to "information retrieval."
- In this context, keywords that carry information about the contents of a document are called *terms*.
- A list of all the terms in a document is called an *index*.
- For each term, a list of all the documents that contain that particular term is called an *inverted index*.
- A typical application is to search databases of scientific papers for given query terms.

- Because of Lecture 2 and HW #1, you should already be familiar with the concept of *term-document matrix*.
- Each column represents a document while each row represents a term.
- The *ij*th entry of such a matrix normally represents the frequency of occurrence of term *i* in document *j*.
- In reality, the size of such matrices are huge ($\gtrapprox 10^5 \times 10^6$).
- However, fortunately, most of the times, they are quite *sparse*.

- Because of Lecture 2 and HW #1, you should already be familiar with the concept of *term-document matrix*.
- Each column represents a document while each row represents a term.
- The *ij*th entry of such a matrix normally represents the frequency of occurrence of term *i* in document *j*.
- In reality, the size of such matrices are huge ($\gtrapprox 10^5 \times 10^6$).
- However, fortunately, most of the times, they are quite *sparse*.

- Because of Lecture 2 and HW #1, you should already be familiar with the concept of *term-document matrix*.
- Each column represents a document while each row represents a term.
- The *ij*th entry of such a matrix normally represents the frequency of occurrence of term i in document j.
- In reality, the size of such matrices are huge ($\gtrapprox 10^5 \times 10^6$).
- However, fortunately, most of the times, they are quite *sparse*.

- Because of Lecture 2 and HW #1, you should already be familiar with the concept of *term-document matrix*.
- Each column represents a document while each row represents a term.
- The *ij*th entry of such a matrix normally represents the frequency of occurrence of term i in document j.
- In reality, the size of such matrices are huge ($\gtrapprox 10^5 \times 10^6$).
- However, fortunately, most of the times, they are quite *sparse*.

- Because of Lecture 2 and HW #1, you should already be familiar with the concept of *term-document matrix*.
- Each column represents a document while each row represents a term.
- The *ij*th entry of such a matrix normally represents the frequency of occurrence of term i in document j.
- In reality, the size of such matrices are huge ($\gtrapprox 10^5 \times 10^6$).
- However, fortunately, most of the times, they are quite *sparse*.

The NIPS Dataset

- In this lecture, we will use the following 'Bags of Words' dataset available from the UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words` .
- This is a collection of 1500 ($= n$) articles (mostly in the field of machine learning and computational neuroscience) published in the proceedings of *Conference on Neural Information Processing Systems* (NIPS) over certain periods.
- The total number of terms (words) examined for these articles is 12419 ($= m$).
- More precisely, after tokenization (i.e., breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens), and removal of stop words (i.e., common words that do not give useful info; more about these in the next section), the vocabulary of unique words was truncated by only keeping words that occurred more than ten times.

The NIPS Dataset

- In this lecture, we will use the following 'Bags of Words' dataset available from the UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words` .
- This is a collection of 1500 ($= n$) articles (mostly in the field of machine learning and computational neuroscience) published in the proceedings of *Conference on Neural Information Processing Systems* (NIPS) over certain periods.
- The total number of terms (words) examined for these articles is 12419 ($= m$).
- More precisely, after tokenization (i.e., breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens), and removal of stop words (i.e., common words that do not give useful info; more about these in the next section), the vocabulary of unique words was truncated by only keeping words that occurred more than ten times.

The NIPS Dataset

- In this lecture, we will use the following 'Bags of Words' dataset available from the UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words` .
- This is a collection of 1500 ($= n$) articles (mostly in the field of machine learning and computational neuroscience) published in the proceedings of *Conference on Neural Information Processing Systems* (NIPS) over certain periods.
- The total number of terms (words) examined for these articles is 12419 ($= m$).
- More precisely, after tokenization (i.e., breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens), and removal of stop words (i.e., common words that do not give useful info; more about these in the next section), the vocabulary of unique words was truncated by only keeping words that occurred more than ten times.

The NIPS Dataset

- In this lecture, we will use the following 'Bags of Words' dataset available from the UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Bag+of+Words` .
- This is a collection of 1500 ($= n$) articles (mostly in the field of machine learning and computational neuroscience) published in the proceedings of *Conference on Neural Information Processing Systems* (NIPS) over certain periods.
- The total number of terms (words) examined for these articles is 12419 ($= m$).
- More precisely, after tokenization (i.e., breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens), and removal of stop words (i.e., common words that do not give useful info; more about these in the next section), the vocabulary of unique words was truncated by only keeping words that occurred more than ten times.

- First 10 words sorted in the alphabetical order: ‘a2i’, ‘aaa’, ‘aaai’, ‘aapo’, ‘aat’, ‘aazhang’, ‘abandonment’, ‘abbott’, ‘abbreviated’, ‘abcde’.
- 10 most frequently used words: ‘network’, ‘model’, ‘learning’, ‘function’, ‘input’, ‘neural’, ‘set’, ‘algorithm’, ‘system’, ‘data’.

- First 10 words sorted in the alphabetical order: ‘a2i’, ‘aaa’, ‘aaai’, ‘aapo’, ‘aat’, ‘aazhang’, ‘abandonment’, ‘abbott’, ‘abbreviated’, ‘abcde’.
- 10 most frequently used words: ‘network’, ‘model’, ‘learning’, ‘function’, ‘input’, ‘neural’, ‘set’, ‘algorithm’, ‘system’, ‘data’.

Outline

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', ...
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\overset{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in Wikipedia.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', ...
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\stackrel{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', . . .
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\stackrel{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', . . .
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\overset{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', . . .
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\stackrel{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', ...
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\overset{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Preprocessing the Documents and Queries

- Before the index (a list of terms contained in a given document) is made, we need to do the following two preprocessing steps:
  1. Elimination of stop words
  2. Stemming
- *Stop words* are words that can be found in virtually any document (i.e., most likely useless words to characterize the documents), e.g., 'a', 'able', 'about', 'above', 'according', 'accordingly', 'across', 'actually', 'after', . . .
- *Stemming* is the process of reducing each word that is conjugated or has a suffix to its stem. For example, 'fishing', 'fished', 'fish', 'fisher' $\overset{\text{stemming}}{\Longrightarrow}$ 'fish' (the root word).
- There are some public domain stemming software systems; see 'Stemming' page in `Wikipedia`.
- Note that stemming was not performed in the NIPS dataset, e.g., the terms include 'model', 'modeled', 'modeling', 'modelled', 'modelling'.

Outline

The Vector Space Model

- The main idea of this model is to create a term-document matrix, say, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where each document is represented by a column vector $\boldsymbol{a}_j$ that has nonzero entries in the position that correspond to terms found in that document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where that term can be found, i.e., the inverted index.
- In practice, a *text parser* (a program) is used to create term-document matrices, and does stemming and stop words removal too.
- The entry a_{ij} is normally set to the term frequency f_{ij}, i.e., the number of times term i appears in document j.
- Can have weights, e.g., $a_{ij} = f_{ij} \log(n/n_i)$ where n_i is the number of documents that contain term i. If term i occurs frequently in only a few documents, then the log factor becomes significant. On the other hand, if term i occurs many documents, the log factor makes $a_{ij} \approx 0$, i.e., term i is not useful. Stop words removal mitigates this to some extent.

The Vector Space Model

- The main idea of this model is to create a term-document matrix, say, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where each document is represented by a column vector $\boldsymbol{a}_j$ that has nonzero entries in the position that correspond to terms found in that document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where that term can be found, i.e., the inverted index.
- In practice, a *text parser* (a program) is used to create term-document matrices, and does stemming and stop words removal too.
- The entry a_{ij} is normally set to the term frequency f_{ij}, i.e., the number of times term i appears in document j.
- Can have weights, e.g., $a_{ij} = f_{ij} \log(n/n_i)$ where n_i is the number of documents that contain term i. If term i occurs frequently in only a few documents, then the log factor becomes significant. On the other hand, if term i occurs many documents, the log factor makes $a_{ij} \approx 0$, i.e., term i is not useful. Stop words removal mitigates this to some extent.

The Vector Space Model

- The main idea of this model is to create a term-document matrix, say, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where each document is represented by a column vector $\boldsymbol{a}_j$ that has nonzero entries in the position that correspond to terms found in that document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where that term can be found, i.e., the inverted index.
- In practice, a *text parser* (a program) is used to create term-document matrices, and does stemming and stop words removal too.
- The entry a_{ij} is normally set to the term frequency f_{ij}, i.e., the number of times term i appears in document j.
- Can have weights, e.g., $a_{ij} = f_{ij} \log(n/n_i)$ where n_i is the number of documents that contain term i. If term i occurs frequently in only a few documents, then the log factor becomes significant. On the other hand, if term i occurs many documents, the log factor makes $a_{ij} \approx 0$, i.e., term i is not useful. Stop words removal mitigates this to some extent.

The Vector Space Model

- The main idea of this model is to create a term-document matrix, say, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where each document is represented by a column vector $\boldsymbol{a}_j$ that has nonzero entries in the position that correspond to terms found in that document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where that term can be found, i.e., the inverted index.
- In practice, a *text parser* (a program) is used to create term-document matrices, and does stemming and stop words removal too.
- The entry a_{ij} is normally set to the term frequency f_{ij}, i.e., the number of times term i appears in document j.
- Can have weights, e.g., $a_{ij} = f_{ij} \log(n/n_i)$ where n_i is the number of documents that contain term i. If term i occurs frequently in only a few documents, then the log factor becomes significant. On the other hand, if term i occurs many documents, the log factor makes $a_{ij} \approx 0$, i.e., term i is not useful. Stop words removal mitigates this to some extent.

The Vector Space Model

- The main idea of this model is to create a term-document matrix, say, $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, where each document is represented by a column vector $\boldsymbol{a}_j$ that has nonzero entries in the position that correspond to terms found in that document.
- Consequently, each row represents a term and has nonzero entries in those positions that correspond to the documents where that term can be found, i.e., the inverted index.
- In practice, a *text parser* (a program) is used to create term-document matrices, and does stemming and stop words removal too.
- The entry a_{ij} is normally set to the term frequency f_{ij}, i.e., the number of times term i appears in document j.
- Can have weights, e.g., $a_{ij} = f_{ij} \log(n/n_i)$ where n_i is the number of documents that contain term i. If term i occurs frequently in only a few documents, then the log factor becomes significant. On the other hand, if term i occurs many documents, the log factor makes $a_{ij} \approx 0$, i.e., term i is not useful. Stop words removal mitigates this to some extent.

Usually, the term-document matrix is *sparse*. For example, in the NIPS dataset, the number of nonzero entries in the term-document matrix of size 12419×1500 is 746,316, which is only 4% of the whole matrix entries.

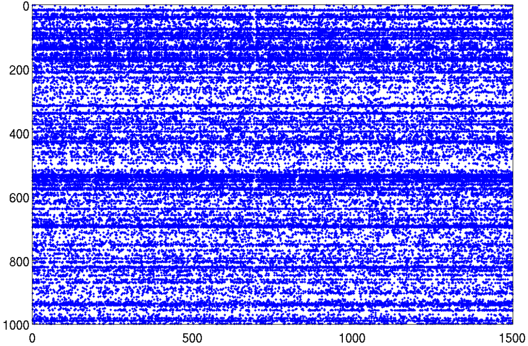

Figure: The first 1000 rows of the NIPS term-document matrix. Each dot represents a nonzero entry.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1 : n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^\top \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1:n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^\top \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1:n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^{\mathsf{T}} \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1 : n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^\mathsf{T} \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1 : n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^\mathsf{T} \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

Query Matching

- Query matching = a process of finding the relevant documents for a given query vector $\boldsymbol{q} \in \mathbb{R}^m$.
- We must define a distance or similarity between $\boldsymbol{q}$ and each document $\boldsymbol{a}_j \in \mathbb{R}^m$, $j = 1:n$.
- Often the following cosine distance (in fact, it would be better to say *similarity* rather than distance) is used:

$$\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) = \frac{\boldsymbol{q}^{\mathsf{T}} \boldsymbol{a}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{a}_j\|_2}.$$

- If $\theta(\boldsymbol{q}, \boldsymbol{a}_j)$ is small enough, then $\boldsymbol{a}_j$ is deemed relevant.
- More precisely, we set some predefined tolerance and if $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j)) > \mathrm{tol}$, then $\boldsymbol{a}_j$ is deemed relevant.
- The smaller the value of tol is, the more documents are retrieved and *considered* as relevant even if many of them are not really relevant.

A Query Matching Example

- Let's consider the NIPS dataset and set up $\boldsymbol{q} \in \mathbb{R}^{12419} = \boldsymbol{e}_{3528} + \boldsymbol{e}_{6700} + \boldsymbol{e}_{6932}$, i.e., only three nonzero entries that correspond to the three terms, 'entropy', 'minimum', 'maximum'.
- Compute $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j))$, $j = 1 : 1500$.

A Query Matching Example

- Let's consider the NIPS dataset and set up $\boldsymbol{q} \in \mathbb{R}^{12419} = \boldsymbol{e}_{3528} + \boldsymbol{e}_{6700} + \boldsymbol{e}_{6932}$, i.e., only three nonzero entries that correspond to the three terms, 'entropy', 'minimum', 'maximum'.
- Compute $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j))$, $j = 1:1500$.

A Query Matching Example

- Let's consider the NIPS dataset and set up $\boldsymbol{q} \in \mathbb{R}^{12419} = \boldsymbol{e}_{3528} + \boldsymbol{e}_{6700} + \boldsymbol{e}_{6932}$, i.e., only three nonzero entries that correspond to the three terms, 'entropy', 'minimum', 'maximum'.
- Compute $\cos(\theta(\boldsymbol{q}, \boldsymbol{a}_j))$, $j = 1:1500$.

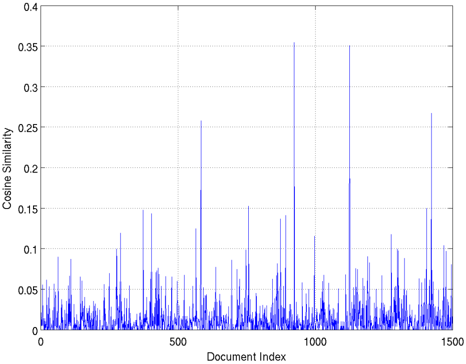

Figure: tol=0.2, 0.1, 0.05 correspond to 4, 15, 89 returned documents.

Performance Modeling

- Let us define the following quantities:

$$\text{Precision:} \quad P := \frac{D_r}{D_t};$$
$$\text{Recall:} \quad R := \frac{D_r}{N_r},$$

  where D_r, D_t, N_r are the number of relevant documents retrieved, the total number of documents retrieved, and the total number of relevant documents in the database, respectively.
- If we set tol large in the cosine similarity measure, then we expect to have high P but low R.
- On the other hand, if we set tol small, the situation is the other way around.
- Unfortunately, in the NIPS dataset, there is no information on the documents except those terms used in them. Hence, we cannot really compute "the Recall vs Precision plot" like those in the textbook.

Performance Modeling

- Let us define the following quantities:

$$\text{Precision:} \quad P := \frac{D_r}{D_t};$$
$$\text{Recall:} \quad R := \frac{D_r}{N_r},$$

where D_r, D_t, N_r are the number of relevant documents retrieved, the total number of documents retrieved, and the total number of relevant documents in the database, respectively.

- If we set tol large in the cosine similarity measure, then we expect to have high P but low R.
- On the other hand, if we set tol small, the situation is the other way around.
- Unfortunately, in the NIPS dataset, there is no information on the documents except those terms used in them. Hence, we cannot really compute "the Recall vs Precision plot" like those in the textbook.

Performance Modeling

- Let us define the following quantities:

$$\text{Precision:} \quad P := \frac{D_r}{D_t};$$
$$\text{Recall:} \quad R := \frac{D_r}{N_r},$$

  where D_r, D_t, N_r are the number of relevant documents retrieved, the total number of documents retrieved, and the total number of relevant documents in the database, respectively.
- If we set tol large in the cosine similarity measure, then we expect to have high P but low R.
- On the other hand, if we set tol small, the situation is the other way around.
- Unfortunately, in the NIPS dataset, there is no information on the documents except those terms used in them. Hence, we cannot really compute "the Recall vs Precision plot" like those in the textbook.

Performance Modeling

- Let us define the following quantities:

$$\text{Precision:} \quad P := \frac{D_r}{D_t};$$
$$\text{Recall:} \quad R := \frac{D_r}{N_r},$$

  where D_r, D_t, N_r are the number of relevant documents retrieved, the total number of documents retrieved, and the total number of relevant documents in the database, respectively.
- If we set tol large in the cosine similarity measure, then we expect to have high P but low R.
- On the other hand, if we set tol small, the situation is the other way around.
- Unfortunately, in the NIPS dataset, there is no information on the documents except those terms used in them. Hence, we cannot really compute "the Recall vs Precision plot" like those in the textbook.

Outline

Latent Semantic Indexing (LSI)

- Is an indexing and retrieval method that uses *SVD* to identify patterns in the relationships between the terms and documents.
- Is based on the principle that words that are used in the same contexts tend to have similar meanings.
- A key feature of LSI: its ability to *extract the conceptual content of a body of text* by establishing associations between those terms that occur in similar contexts.
- Could trace back its history to *factor analysis* applications in mid 1960s, but it started gaining the popularity in late 80s to early 90s. Nowadays, LSI is being used in many applications on a daily basis.

Latent Semantic Indexing (LSI)

- Is an indexing and retrieval method that uses *SVD* to identify patterns in the relationships between the terms and documents.
- Is based on the principle that words that are used in the same contexts tend to have similar meanings.
- A key feature of LSI: its ability to *extract the conceptual content of a body of text* by establishing associations between those terms that occur in similar contexts.
- Could trace back its history to *factor analysis* applications in mid 1960s, but it started gaining the popularity in late 80s to early 90s. Nowadays, LSI is being used in many applications on a daily basis.

Latent Semantic Indexing (LSI)

- Is an indexing and retrieval method that uses *SVD* to identify patterns in the relationships between the terms and documents.
- Is based on the principle that words that are used in the same contexts tend to have similar meanings.
- A key feature of LSI: its ability to *extract the conceptual content of a body of text* by establishing associations between those terms that occur in similar contexts.
- Could trace back its history to *factor analysis* applications in mid 1960s, but it started gaining the popularity in late 80s to early 90s. Nowadays, LSI is being used in many applications on a daily basis.

Latent Semantic Indexing (LSI)

- Is an indexing and retrieval method that uses *SVD* to identify patterns in the relationships between the terms and documents.
- Is based on the principle that words that are used in the same contexts tend to have similar meanings.
- A key feature of LSI: its ability to *extract the conceptual content of a body of text* by establishing associations between those terms that occur in similar contexts.
- Could trace back its history to *factor analysis* applications in mid 1960s, but it started gaining the popularity in late 80s to early 90s. Nowadays, LSI is being used in many applications on a daily basis.

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^\mathsf{T}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^\mathsf{T}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^\mathsf{T} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^\mathsf{T} A_k = \boldsymbol{q}^\mathsf{T} U_k H_k = (U_k^\mathsf{T} \boldsymbol{q})^\mathsf{T} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^\mathsf{T} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^\mathsf{T} \boldsymbol{q}.$$

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^{\mathsf{T}}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^{\mathsf{T}}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^{\mathsf{T}} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^{\mathsf{T}} A_k = \boldsymbol{q}^{\mathsf{T}} U_k H_k = (U_k^{\mathsf{T}} \boldsymbol{q})^{\mathsf{T}} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^{\mathsf{T}} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^{\mathsf{T}} \boldsymbol{q}.$$

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^\mathsf{T}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^\mathsf{T}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^\mathsf{T} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^\mathsf{T} A_k = \boldsymbol{q}^\mathsf{T} U_k H_k = (U_k^\mathsf{T} \boldsymbol{q})^\mathsf{T} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^\mathsf{T} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^\mathsf{T} \boldsymbol{q}.$$

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^{\mathsf{T}}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^{\mathsf{T}}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^{\mathsf{T}} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^{\mathsf{T}} A_k = \boldsymbol{q}^{\mathsf{T}} U_k H_k = (U_k^{\mathsf{T}} \boldsymbol{q})^{\mathsf{T}} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^{\mathsf{T}} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^{\mathsf{T}} \boldsymbol{q}.$$

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^\mathsf{T}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^\mathsf{T}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^\mathsf{T} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^\mathsf{T} A_k = \boldsymbol{q}^\mathsf{T} U_k H_k = (U_k^\mathsf{T} \boldsymbol{q})^\mathsf{T} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^\mathsf{T} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^\mathsf{T} \boldsymbol{q}.$$

- Let $A \in \mathbb{R}^{m \times n}$ be a term-document matrix, and let $A_k := U_k \Sigma_k V_k^{\mathsf{T}}$ be the rank k approximation of A using the first k singular values and singular vectors. Let $H_k := \Sigma_k V_k^{\mathsf{T}}$, i.e., $A_k = U_k H_k$.
- For an appropriate value of k, $A \approx A_k$. Hence, we have $\boldsymbol{a}_j \approx U_k \boldsymbol{h}_j$ where $\boldsymbol{a}_j$ and $\boldsymbol{h}_j$ are the jth column vectors of A and H_k, respectively.
- This means that $\boldsymbol{h}_j$ are the expansion coefficients of the best k-term approximation to $\boldsymbol{a}_j$ w.r.t. the ONB vectors $\{\boldsymbol{u}_1, \dots, \boldsymbol{u}_k\}$.
- Previously, for a given query vector $\boldsymbol{q}$, in order to compute the cosine similarities between $\boldsymbol{q}$ and $\boldsymbol{a}_j$, $j = 1:n$, we had to compute $\boldsymbol{q}^{\mathsf{T}} A$ followed by the normalization by $\|\boldsymbol{q}\|_2$ and $\|\boldsymbol{a}_j\|$.
- Now, let's replace A by its best k-term approximation A_k, i.e., we compute: $\boldsymbol{q}^{\mathsf{T}} A_k = \boldsymbol{q}^{\mathsf{T}} U_k H_k = (U_k^{\mathsf{T}} \boldsymbol{q})^{\mathsf{T}} H_k$.
- Hence, we can simplify the cosine similarity computation as follows:

$$\cos\theta_j := \frac{\boldsymbol{q}_k^{\mathsf{T}} \boldsymbol{h}_j}{\|\boldsymbol{q}\|_2 \|\boldsymbol{h}_j\|_2}, \quad \boldsymbol{q}_k := U_k^{\mathsf{T}} \boldsymbol{q}.$$

- Note that there is a typo in the textbook Eqn.(11.4). The formula in the previous page of this slide is correct. In the textbook formula, the author normalized it by $\|\boldsymbol{q}_k\|_2$ instead of $\|\boldsymbol{q}\|_2$. You can show that $\|\boldsymbol{q}_k\|_2 \neq \|\boldsymbol{q}\|_2$.
- The reason why we formed H_k and q_k is that there is no need to explicitly compute and store A_k once we have H_k and $\boldsymbol{q}_k$. Directly dealing with A_k by computing and storing it is wasteful and time-consuming particularly for a large A.

- Note that there is a typo in the textbook Eqn.(11.4). The formula in the previous page of this slide is correct. In the textbook formula, the author normalized it by $\|\boldsymbol{q}_k\|_2$ instead of $\|\boldsymbol{q}\|_2$. You can show that $\|\boldsymbol{q}_k\|_2 \neq \|\boldsymbol{q}\|_2$.
- The reason why we formed H_k and q_k is that there is no need to explicitly compute and store A_k once we have H_k and $\boldsymbol{q}_k$. Directly dealing with A_k by computing and storing it is wasteful and time-consuming particularly for a large A.

An LSI Query Example

- Let's use the NIPS dataset with $k = 100$.
- Then, the relative error of A_{100} and A in terms of the Frobenius norm, i.e., $\|A - A_{100}\|_F / \|A\|_F$ was 0.6074, which is still large.
- Nonetheless, we get the relatively good performance.

An LSI Query Example

- Let's use the NIPS dataset with $k = 100$.
- Then, the relative error of A_{100} and A in terms of the Frobenius norm, i.e., $\|A - A_{100}\|_F / \|A\|_F$ was 0.6074, which is still large.
- Nonetheless, we get the relatively good performance.

An LSI Query Example

- Let's use the NIPS dataset with $k = 100$.
- Then, the relative error of A_{100} and A in terms of the Frobenius norm, i.e., $\|A - A_{100}\|_F / \|A\|_F$ was 0.6074, which is still large.
- Nonetheless, we get the relatively good performance.

An LSI Query Example

- Let's use the NIPS dataset with $k = 100$.
- Then, the relative error of A_{100} and A in terms of the Frobenius norm, i.e., $\|A - A_{100}\|_F / \|A\|_F$ was 0.6074, which is still large.
- Nonetheless, we get the relatively good performance.

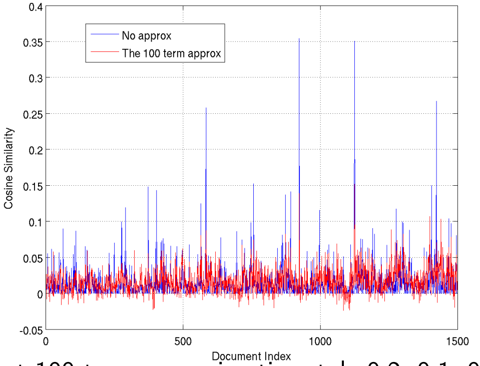

Figure: With the best 100 term approximation, tol=0.2, 0.1, 0.05 correspond to 0, 4, 72 returned documents; Compare with the no approximation case: 4, 15, 89.

- We know that the vector $\boldsymbol{u}_1$ is the most dominant basis vector representing the range of the term space (i.e., the column space of A).
- Hence it is of our interest to check what terms $\boldsymbol{u}_1$ represents (note that the entries of $\boldsymbol{u}_1$ are nonnegative for this matrix). The 10 terms corresponding to the largest entries of $\boldsymbol{u}_1$: 'network', 'model', 'learning', 'input', 'function', 'neural', 'set', 'training', 'data', 'unit'.
- Compare these with the top 10 frequently used terms: 'network', 'model', 'learning', 'function', 'input', 'neural', 'set', 'algorithm', 'system', 'data'. As you can see, they are very close.
- Let's check the entries of $\boldsymbol{u}_2$, which contains both positive and negative values. The top 5 positive entries of $\boldsymbol{u}_2$: 'network', 'unit', 'input', 'neural', 'output', while the top 5 negative entries of $\boldsymbol{u}_2$: 'model', 'data', 'algorithm', 'learning', 'parameter'.
- My interpretation: $\boldsymbol{u}_2$ tries to differentiate articles related to neuroscience from those related to machine learning algorithms.

- We know that the vector $\boldsymbol{u}_1$ is the most dominant basis vector representing the range of the term space (i.e., the column space of A).
- Hence it is of our interest to check what terms $\boldsymbol{u}_1$ represents (note that the entries of $\boldsymbol{u}_1$ are nonnegative for this matrix). The 10 terms corresponding to the largest entries of $\boldsymbol{u}_1$: 'network', 'model', 'learning', 'input', 'function', 'neural', 'set', 'training', 'data', 'unit'.
- Compare these with the top 10 frequently used terms: 'network', 'model', 'learning', 'function', 'input', 'neural', 'set', 'algorithm', 'system', 'data'. As you can see, they are very close.
- Let's check the entries of $\boldsymbol{u}_2$, which contains both positive and negative values. The top 5 positive entries of $\boldsymbol{u}_2$: 'network', 'unit', 'input', 'neural', 'output', while the top 5 negative entries of $\boldsymbol{u}_2$: 'model', 'data', 'algorithm', 'learning', 'parameter'.
- My interpretation: $\boldsymbol{u}_2$ tries to differentiate articles related to neuroscience from those related to machine learning algorithms.

- We know that the vector $\boldsymbol{u}_1$ is the most dominant basis vector representing the range of the term space (i.e., the column space of A).
- Hence it is of our interest to check what terms $\boldsymbol{u}_1$ represents (note that the entries of $\boldsymbol{u}_1$ are nonnegative for this matrix). The 10 terms corresponding to the largest entries of $\boldsymbol{u}_1$: 'network', 'model', 'learning', 'input', 'function', 'neural', 'set', 'training', 'data', 'unit'.
- Compare these with the top 10 frequently used terms: 'network', 'model', 'learning', 'function', 'input', 'neural', 'set', 'algorithm', 'system', 'data'. As you can see, they are very close.
- Let's check the entries of $\boldsymbol{u}_2$, which contains both positive and negative values. The top 5 positive entries of $\boldsymbol{u}_2$: 'network', 'unit', 'input', 'neural', 'output', while the top 5 negative entries of $\boldsymbol{u}_2$: 'model', 'data', 'algorithm', 'learning', 'parameter'.
- My interpretation: $\boldsymbol{u}_2$ tries to differentiate articles related to neuroscience from those related to machine learning algorithms.

- We know that the vector $\boldsymbol{u}_1$ is the most dominant basis vector representing the range of the term space (i.e., the column space of A).
- Hence it is of our interest to check what terms $\boldsymbol{u}_1$ represents (note that the entries of $\boldsymbol{u}_1$ are nonnegative for this matrix). The 10 terms corresponding to the largest entries of $\boldsymbol{u}_1$: 'network', 'model', 'learning', 'input', 'function', 'neural', 'set', 'training', 'data', 'unit'.
- Compare these with the top 10 frequently used terms: 'network', 'model', 'learning', 'function', 'input', 'neural', 'set', 'algorithm', 'system', 'data'. As you can see, they are very close.
- Let's check the entries of $\boldsymbol{u}_2$, which contains both positive and negative values. The top 5 positive entries of $\boldsymbol{u}_2$: 'network', 'unit', 'input', 'neural', 'output', while the top 5 negative entries of $\boldsymbol{u}_2$: 'model', 'data', 'algorithm', 'learning', 'parameter'.
- My interpretation: $\boldsymbol{u}_2$ tries to differentiate articles related to neuroscience from those related to machine learning algorithms.

- We know that the vector $\boldsymbol{u}_1$ is the most dominant basis vector representing the range of the term space (i.e., the column space of A).
- Hence it is of our interest to check what terms $\boldsymbol{u}_1$ represents (note that the entries of $\boldsymbol{u}_1$ are nonnegative for this matrix). The 10 terms corresponding to the largest entries of $\boldsymbol{u}_1$: 'network', 'model', 'learning', 'input', 'function', 'neural', 'set', 'training', 'data', 'unit'.
- Compare these with the top 10 frequently used terms: 'network', 'model', 'learning', 'function', 'input', 'neural', 'set', 'algorithm', 'system', 'data'. As you can see, they are very close.
- Let's check the entries of $\boldsymbol{u}_2$, which contains both positive and negative values. The top 5 positive entries of $\boldsymbol{u}_2$: 'network', 'unit', 'input', 'neural', 'output', while the top 5 negative entries of $\boldsymbol{u}_2$: 'model', 'data', 'algorithm', 'learning', 'parameter'.
- My interpretation: $\boldsymbol{u}_2$ tries to differentiate articles related to neuroscience from those related to machine learning algorithms.