1495 UNIVERSITY OF ABERDEEN | CELEBRATING 525 YEARS 1495 – 2020

ABERDEEN 2040

Text Mining – 2

Data Mining & Visualisation

Lecture 28

2025

Today…

- Information Retrieval
- Jaccard Coefficient
- TF-IDF
- Word Embeddings

Text Mining

So far, we have outlined some text preprocessing techniques, as well as outlining one form of text mining (sentiment analysis).

Now, let's explore some more of the theory that we can apply to text mining more broadly.

Information Retrieval

Information Retrieval

Information Retrieval (IR) is a scientific discipline relating to the identification and retrieval of resources (files, websites, databases, etc.).

It is widely used in contexts where we want to ‘query’ a larger set of data, and return similar (i.e. *relevant*) data in as little time as possible.

It is the cornerstone of many technologies, such as web search engines and document retrieval systems.

Information Retrieval – Core Components

Query: A query is what we are looking for, and what we input into the IR system. It might be a set of keywords, phrases, a question, etc.

Document: Any piece of content that can be retrieved in response to a query. This might include text documents, PDFs, web pages, files, etc.

Index: A data structure that improves the speed of data retrieval operations by providing quicker access to the records.

Relevance: The measure of how closely a document matches the user's query, in ensuring that the results meet the user's needs.

Ranked Retrieval Models

With ranked retrieval models, rather than returning a set of documents that satisfy some query expression, the system returns an ordering over the (top) documents in the collection for a query.

Ranked retrieval is typically associated with **free-text queries**, where, rather than using a structured query language (like SQL), the user's query might include just one or more words in a human language.

Note that this is where text preprocessing (stop word removal, stemming, lemmatization, normalization, etc.) can play a key role.

Ranked Retrieval Models – Advantage

The main advantage of ranked retrieval models is that, when a system produces a ranked result set, large result sets are not an issue.

Indeed, we can just show the top ~10 results to the user, so as not to overwhelm them.

Ranking Scores

So how do we determine ‘relevance’ within ranked result sets?

We need a way of assigning a *score* to a query-document pair, indicating how much the latter ‘matches’ with the former.

Ranking Scores – One-Term Query

One simple way of doing this could be with a '**one-term query**' – i.e. a single word or token that we use to search for document.

If the document does not contain this query term, the score will be 0.

The more frequently this query term appears in the document, the higher the retrieval score will be.

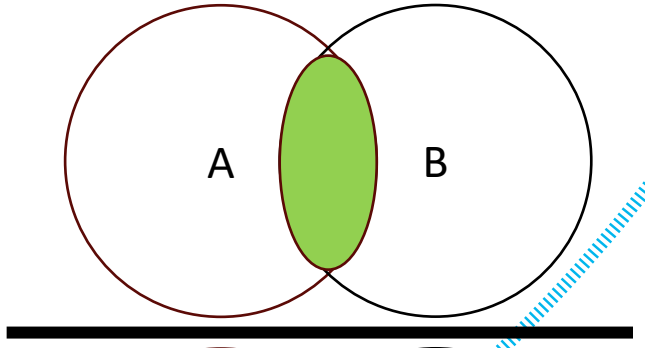
Such an approach works, and is fast, but not always very effective.

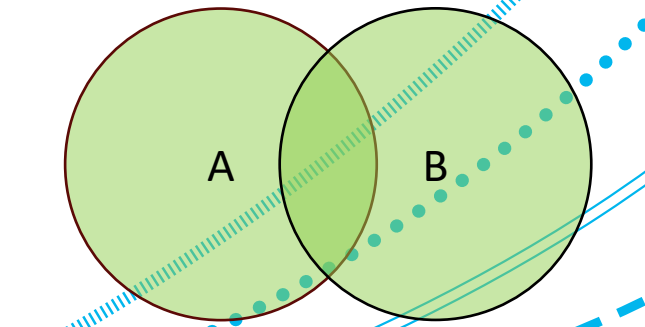
Ranking Scores – Jaccard Coefficient

So let's say we want to 'score' our documents based on some query.

One approach might be to use to Jaccard Coefficient, which measures the overlap of two sets: A and B.

The Jaccard coefficient is always ≥ 0, and ≤ 1.





Ranking Scores – Jaccard Coefficient

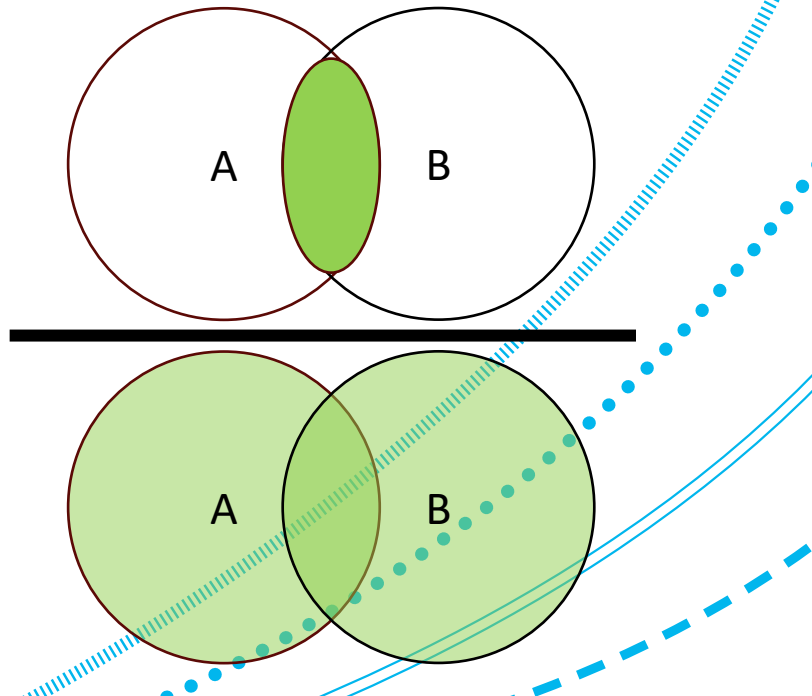
Jaccard Coefficient:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

If A = B, then J(A, B) = 1

And if A ∩ B = ∅, then J(A, B) = 0

J(A, B) = J(B, A)



Ranking Scores – Jaccard Coefficient

Example:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

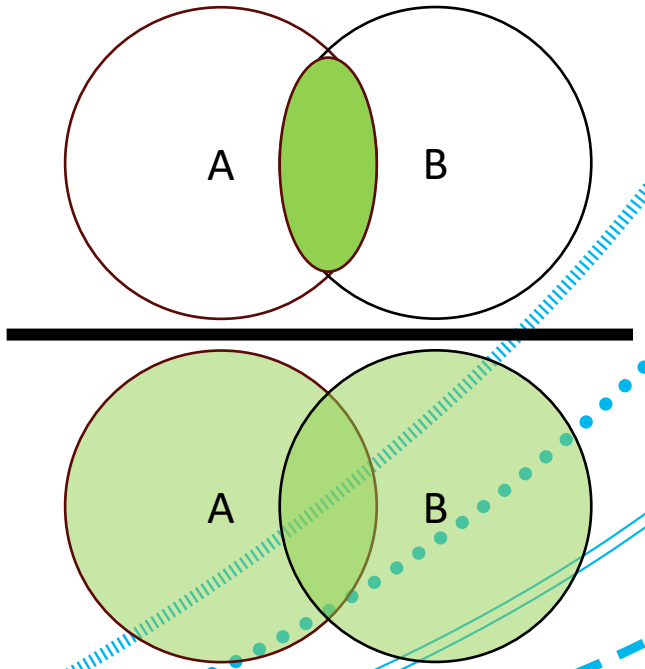
q = { ‘review’; ‘for’; ‘the’; ‘latest’; ‘iphone’ }

d1 = { ‘this’, ‘is’, ‘the’, ‘review’, ‘for’, ‘the ’, ‘latest ’, ‘iphone’ }

d2 = { ‘this’, ‘is’, ‘the’, ‘review’, ‘for’, ‘the ’, ‘latest ’, ‘huawei’ }

J(q, d1) = (5) / (7) = 0.71 ✅

J(q, d2) = (4) / (8) = 0.5



Ranking Scores – Jaccard Coefficient

This works reasonably well for measuring the similarity between a query and a given document.

However, it doesn't take into account how many times the query actually appears within the document.

E.g. think what would happen if the Huawei review contained the word 'iPhone' to compare the two devices, and contained fewer unique words than the iPhone review…

Ranking Scores – Jaccard Coefficient

q = { ‘review’; ‘for’; ‘the’; ‘latest’; ‘iphone’ }

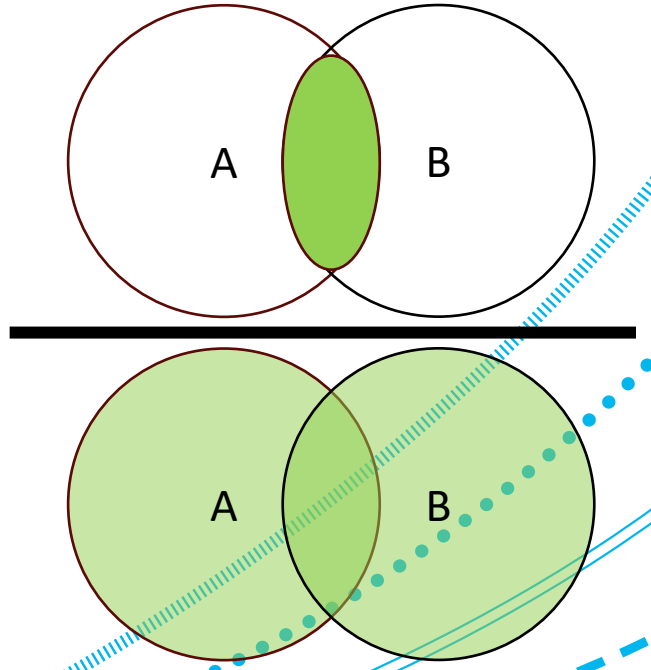
d1 = { ‘this’, ‘is’, ‘the’, ‘review’, ‘for’, ‘the ’, ‘latest ’, ‘iphone’, ‘and’, ‘it’, ‘is’, ‘the’, ‘fastest’, ‘iphone’, ‘ever ’ }

d2 = { ‘move’, ‘over’, ‘iphone’, ‘this’, ‘is’, ‘the’, ‘review’, ‘for’, ‘the’, ‘latest ’, ‘huawei’ }

J(q, d1) = (5) / (11) = 0.46

J(q, d2) = (5) / (10) = 0.5 ✅

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Ranking Scores – Term Frequency

If we search for a particular word or term, we want our IR system to take the frequency that it appears in the documents into account.

Think how many times the word *'python'* might appear within a book on *data mining* vs how many times it might appear within a book on *reptiles*.

This concept is known as *'term frequency'*, which plays a significant role in modern-day IR systems.

Ranking Scores – Term Frequency

More formally, the term frequency $tf(t, d)$ is simply defined as the frequency f of term t occurring within document d, over the total number of terms t' within d.

$$tf(t,d) = \frac{f_t}{\sum_{t' \in d} f_{t',d}}$$

In other words, it represents an (adjusted) measure of importance that any given term plays within a given document.

Ranking Scores – Document Frequency

However, we also want our IR system to recognize that some terms are more or less informative than others.

E.g. think of the proportional value of the word *'regression'* to a data mining document, compared to the word *'the'* (which might appear in almost every document).

One way of measuring this is through *'document frequency'*.

Ranking Scores – Document Frequency

The document frequency $df(t, D)$ is the number of documents within the document set D that contain the term t.

$$df(t, D) = |\{d : d \in D \text{ and } t \in d\}|$$

However, df is high when the term is common across all documents.

What we're really looking for is a measure of the *informativeness* of t, which is the inverse of this metric.

Ranking Scores – Inverse Document Frequency

Given that DF represents an *inverse* measure of the informativeness of t, we can also calculate the 'inverse document frequency'.

$$idf(t, D) = \log\left(\frac{|D|}{|\{d : d \in D \text{ and } t \in d\}|}\right)$$

The intuition behind df is that rarer (i.e. more specialized) terms will likely be more informative than more common terms.

The $\log$ is typically (but not always) included, to help scale the values.

Ranking Scores – TF-IDF

We can bring these terms together to come up with some form of overall 'score' of a query, taking into account:

- The frequency that t appears in d – i.e. tf
- The importance of t within D – i.e. idf

$$\text{TF-IDF}(t, d, D) = tf(t, d) \cdot idf(t, D)$$

We call this Term Frequency-Inverse Document Frequency, or TF-IDF (sometimes TF.IDF or TF*IDF).

Ranking Scores – TF-IDF

So in very simple terms, TF-IDF can be used to come up with a score for any given document, given a particular query.

We can then use these to ‘rank’ our most relevant documents, and display this to the user.

TF-IDF – Text Mining

But let's think back to our original context: Text Mining.

Let's say we have a column of text data representing apartment adverts. Think about how we might evaluate particular terms against the text.

E.g. we retrieve an TF-IDF score for our apartment descriptions based on the terms 'parking', 'highway', and 'commute', and then extract that as its own numerical column.

TF-IDF – Text Mining

What this might give us is each data point (i.e. advert) containing a set of columns for different query terms, containing their TF-IDF scores.

id	text	pool	parking, highway, commute	cats, dogs	…
1	This apartment…	0.4	0	0.05	
2	This property…	0	0.2	0.05	
3	New to the market…	0.1	0	0.3	
…	…				

TF-IDF – Text Mining

Each advert now contains a vector of TF-IDF scores, for whichever terms we think might be interesting to explore.

We could then use these values *downstream* in our analysis, e.g. via clustering, as a set of ML predictors, or for some data visualisation!

Word Embeddings

Word Embeddings

Another concept within text mining is the idea of **word embeddings**.

Word embeddings are a way to represent words as **dense, multi-dimensional vectors**, where the distance and direction between vectors corresponds to similarity and relationships between those words.

Word embeddings are created by training a neural network on a large amount of text data, and adjusting the weights of the vector representations based on the context in which words appear.

Word Embeddings

Each word is a row of data, with each subsequent column corresponding to some latent dimensions of the embedding space.

These vectors are '**dense**' in that we limit the number of columns to e.g. 50–300.

This is opposed to dummy coding, which would require a column for every possible word.

Word Embeddings

These dimensions are **latent**, in that they do not correspond to any trained labels; the features emerge from the co-occurrence of patterns within the training corpus.

In other words, they can be hard to interpret and understand, but word embeddings are a proven technique that work very well.

Word Embeddings

E.g. imagine we have two columns (rather than 50–300), and that these columns happen to map to some representation of *Gender* and *Age*.

For each word, the values/weights of each column correspond to that word's placement within this semantic feature space.

word	Gender	Age
man	1	7
woman	9	7
boy	1	2
girl	9	2

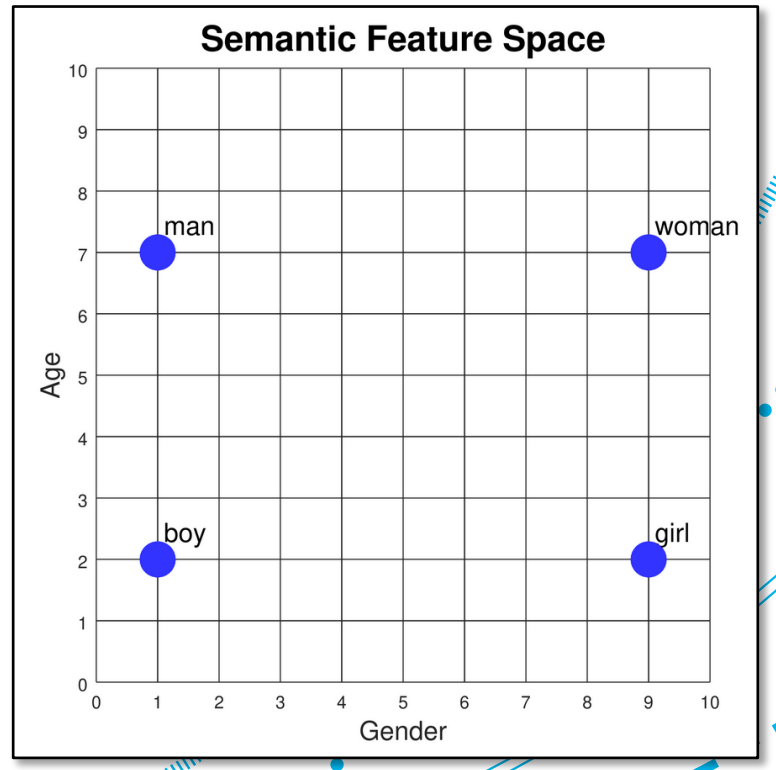


Image source: https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

Word Embeddings

This can be a very powerful approach, as we can use these embeddings to perform vector arithmetic!

E.g. boy – man + woman

= [1, 2] - [1, 7] + [9, 7]

= [9, 2] (i.e. girl)

word	Gender	Age
man	1	7
woman	9	7
boy	1	2
girl	9	2

Word Embeddings

We can also use the cosine similarity to compare the angle between two vectors.

By doing so, we can evaluate similarities between different vectors (words).

E.g. man and grandfather, boy and adult

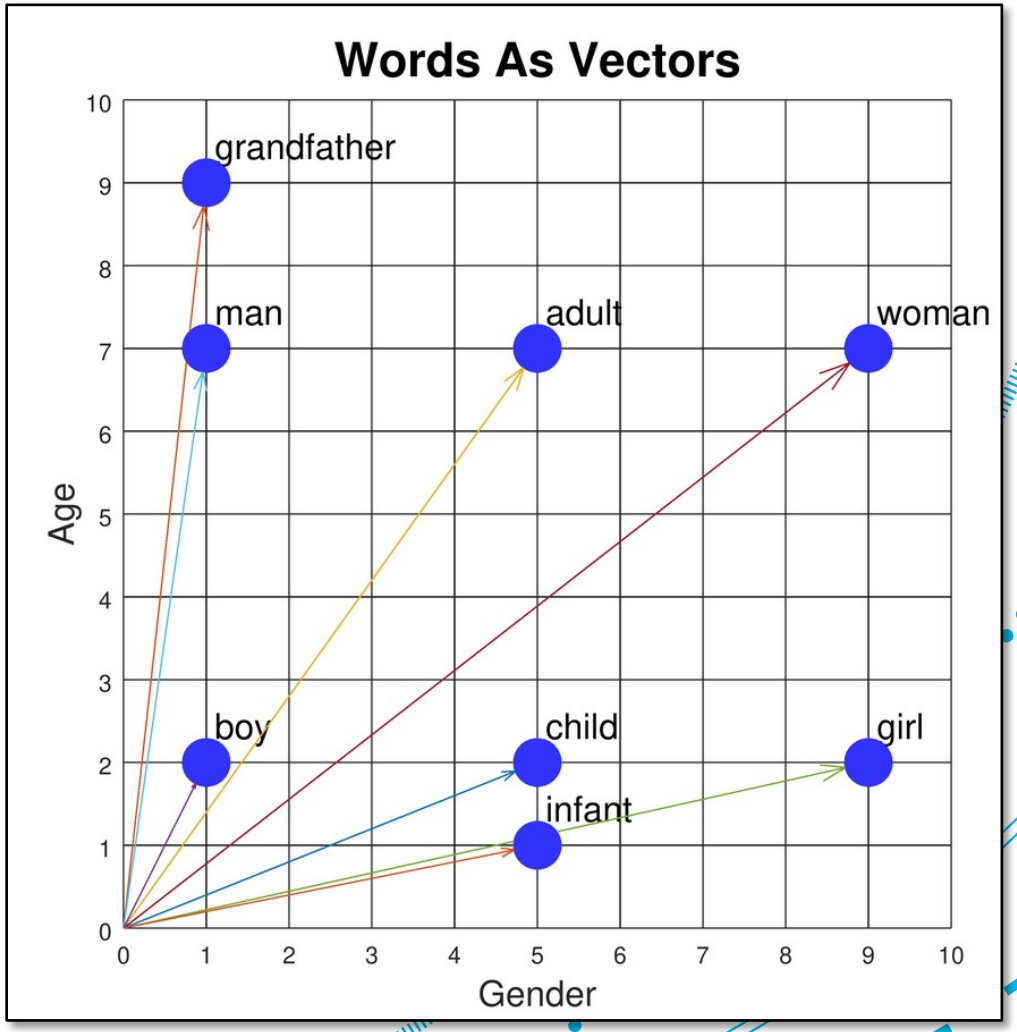


Image source: https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

Word Embeddings

Again, in reality, these columns most likely won’t be as interpretable as simply having Age and Gender.

However, they can still be very useful for a wide range of NLP tasks, such as identifying the semantic relationships between words.

Word2Vec

One of the more popular implementations for word embeddings is Word2Vec.

Word2Vec uses two-layer neural networks to generate word embeddings.

It provides some additional features for speed, efficiency, and convenience. But broadly speaking, the idea is the same.

Summary

Summary

Text data is messy, complex, and can be difficult to work with…

However, mining and analysing text data can transform the way you analyse a dataset, undertake EDA, and understand or uncover broad patterns and trends.

As a result, it can be well worth it!

Summary

“Welcome home to an apartment community located in the heart of it all. This lush community offers one and two beds apartment homes as well as sheltered parking.

Located just off Campbell and Coit, this residential setting offers the finest shops, countless restaurants and Addison nightlife fingertips. So, come home to and discover the lifestyle you deserve.

DeluxeUnit, Fireplace, Select K/B upgrades, STORAGE, Stacked Washer-dryer, Pool, Cloths washer and drier Connections, Bar with sink. Ok for pets

Comments: Pet Charges and Deposits Vary by Property. Assistance animals are always welcome without deposit or fee. Restrictions: Pet Types Allowed: Dogs, Cats, Fish, Caged Birds. Price range: $898 - $1, 388. More units available: two Bd / 2 Bedrooms 1,000 sq. feet for $1,503/mo | two Bd / 1 Bedrooms 890 square ft for $1,563/mo | one Bd / 1 Bedrooms 630 square feet for $948/mo | two Bd / 2 Bedrooms 1,000 sq. feet for $1,308/mo | two Bd / 2 Bedrooms 1,000 sq-ft for $1,708/mo | two Bd / 1 Bedrooms 890 sq. ”

Got any good ideas for the assignment?